

DESARROLLO DE SOLUCIONES WEB

- JavaScript
- Node.js
- Express
- MySQL



Guillermo Augusto Bocangel Weydert
Elmer Santiago Chuquiyaury Saldivar
Guillermo Augusto Bocangel Marín
Jhonny Henry Piñán García
Guadalupe Ramírez Reyes
Hernan Wilmer García Bonilla
Lincol Jarly Gomez Meza

Editora jefe	2024 por Atena Editora
Profª Drª Antonella Carvalho de Oliveira	Derechos de autor © Atena Editora
Editora ejecutiva	Derechos de autor del texto © 2024 Los autores
Natalia Oliveira	Derechos de autor de la edición © 2024 Atena Editora
Asistente editorial	Editora
Flávia Roberta Barão	Derechos de esta edición concedidos a Atena Editora por los autores.
Bibliotecario	Publicación de acceso abierto por Atena Editora
Janaina Ramos	



Todo el contenido de este libro tiene una licencia de Creative Commons Attribution License. Reconocimiento-No Comercial-No Derivados 4.0 Internacional (CC BY-NC-ND 4.0).

El contenido del texto y sus datos en su forma, corrección y confiabilidad son de exclusiva responsabilidad de los autores, y no representan necesariamente la posición oficial de Atena Editora. Se permite descargar la obra y compartirla siempre que se den los créditos a los autores, pero sin posibilidad de alterarla de ninguna forma ni utilizarla con fines comerciales.

Todos los manuscritos fueron previamente sometidos a evaluación ciega por pares, miembros del Consejo Editorial de esta editorial, habiendo sido aprobados para su publicación con base en criterios de neutralidad e imparcialidad académica.

Atena Editora se compromete a garantizar la integridad editorial en todas las etapas del proceso de publicación, evitando plagios, datos o entonces, resultados fraudulentos y evitando que los intereses económicos comprometan los estándares éticos de la publicación. Las situaciones de sospecha de mala conducta científica se investigarán con el más alto nivel de rigor académico y ético.

Consejo Editorial

Multidisciplinar

Prof. Dr. Adélio Alcino Sampaio Castro Machado – Universidade do Porto

Profª Drª Alana Maria Cerqueira de Oliveira – Instituto Federal do Acre

Profª Drª Ana Grasielle Dionísio Corrêa – Universidade Presbiteriana Mackenzie

Profª Drª Ana Paula Florêncio Aires – Universidade de Trás-os-Montes e Alto Douro

Prof. Dr. Carlos Eduardo Sanches de Andrade – Universidade Federal de Goiás

Profª Drª Carmen Lúcia Voigt – Universidade Norte do Paraná

Prof. Dr. Cleiseano Emanuel da Silva Paniagua – Colégio Militar Dr. José Aluisio da Silva Luz / Colégio Santa Cruz de Araguaína/TO

Profª Drª Cristina Aledi Felseburgh – Universidade Federal do Oeste do Pará

Prof. Dr. Diogo Peixoto Cordova – Universidade Federal do Pampa, Campus Caçapava do Sul

Prof. Dr. Douglas Gonçalves da Silva – Universidade Estadual do Sudoeste da Bahia

Prof. Dr. Eloi Rufato Junior – Universidade Tecnológica Federal do Paraná

Profª Drª Érica de Melo Azevedo – Instituto Federal do Rio de Janeiro

Prof. Dr. Fabrício Menezes Ramos – Instituto Federal do Pará

Prof. Dr. Fabrício Moraes de Almeida – Universidade Federal de Rondônia
Profª Drª Glécilla Colombelli de Souza Nunes – Universidade Estadual de Maringá
Prof. Dr. Hauster Maximiler Campos de Paula – Universidade Federal de Viçosa
Profª Drª Iara Margolis Ribeiro – Universidade Federal de Pernambuco
Profª Drª Jéssica Barbosa da Silva do Nascimento – Universidade Estadual de Santa Cruz
Profª Drª Jéssica Verger Nardeli – Universidade Estadual Paulista Júlio de Mesquita Filho
Prof. Dr. Juliano Bitencourt Campos – Universidade do Extremo Sul Catarinense
Prof. Dr. Juliano Carlo Rufino de Freitas – Universidade Federal de Campina Grande
Prof. Dr. Leonardo França da Silva – Universidade Federal de Viçosa
Profª Drª Luciana do Nascimento Mendes – Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte
Prof. Dr. Marcelo Marques – Universidade Estadual de Maringá
Prof. Dr. Marco Aurélio Kistemann Junior – Universidade Federal de Juiz de Fora
Prof. Dr. Marcos Vinicius Winckler Caldeira – Universidade Federal do Espírito Santo
Profª Drª Maria Iaponeide Fernandes Macêdo – Universidade do Estado do Rio de Janeiro
Profª Drª Maria José de Holanda Leite – Universidade Federal de Alagoas
Profª Drª Mariana Natale Fiorelli Fabiche – Universidade Estadual de Maringá
Prof. Dr. Miguel Adriano Inácio – Instituto Nacional de Pesquisas Espaciais
Prof. Dr. Milson dos Santos Barbosa – Universidade Tiradentes
Profª Drª Natiéli Piovesan – Instituto Federal do Rio Grande do Norte
Profª Drª Neiva Maria de Almeida – Universidade Federal da Paraíba
Prof. Dr. Nilzo Ivo Ladwig – Universidade do Extremo Sul Catarinense
Profª Drª Priscila Natasha Kinas – Universidade do Estado de Santa Catarina
Profª Drª Priscila Tessmer Scaglioni – Universidade Federal de Pelotas
Prof. Dr. Rafael Pacheco dos Santos – Universidade do Estado de Santa Catarina
Prof. Dr. Ramiro Picoli Nippes – Universidade Estadual de Maringá
Profª Drª Regina Célia da Silva Barros Allil – Universidade Federal do Rio de Janeiro
Prof. Dr. Sidney Gonçalo de Lima – Universidade Federal do Piauí
Prof. Dr. Takeshy Tachizawa – Faculdade de Campo Limpo Paulista

Desarrollo de soluciones Web: JavaScript, Node.js, Express, MySQL

Indexación: Amanda Kelly da Costa Veiga

Revisión: Los autores

Autores: Guillermo Augusto Bocangel Weydert
Elmer Santiago Chuquiyaui Saldivar
Guillermo Augusto Bocangel Marín
Jhonny Henry Piñán García
Guadalupe Ramírez Reyes
Hernan Wilmer García Bonilla
Lincol Jarly Gomez Meza

Datos de catalogación en publicación internacional (CIP)

D441 Desarrollo de soluciones Web: JavaScript, Node.js, Express, MySQL / Guillermo Augusto Bocangel Weydert, Elmer Santiago Chuquiyaui Saldivar, Guillermo Augusto Bocangel Marín, et al. – Ponta Grossa - PR: Atena, 2024.

Otros autores

Jhonny Henry Piñán García
Guadalupe Ramírez Reyes
Hernan Wilmer García Bonilla
Lincol Jarly Gomez Meza

Formato: PDF

Requisitos de sistema: Adobe Acrobat Reader

Modo de acceso: World Wide Web

Inclui bibliografía

ISBN 978-65-258-2807-7

DOI: <https://doi.org/10.22533/at.ed.077242009>

1. Tecnologías de la información. I. Weydert, Guillermo Augusto Bocangel. II. Saldivar, Elmer Santiago Chuquiyaui. III. Marín, Guillermo Augusto Bocangel. IV. Título.

CDD 005.7

Preparado por Bibliotecario Janaina Ramos – CRB-8/9166

Atena Editora
Ponta Grossa – Paraná – Brasil
Telefone: +55 (42) 3323-5493
www.atenaeditora.com.br
contato@atenaeditora.com.br

DECLARACIÓN DE LOS AUTORES

Los autores de este trabajo: 1. Certifican que no tienen ningún interés comercial que constituya un conflicto de interés en relación con el artículo científico publicado; 2. Declaran haber participado activamente en la construcción de los respectivos manuscritos, preferentemente en: a) Concepción del estudio, y/o adquisición de datos, y/o análisis e interpretación de datos; b) Elaboración del artículo o revisión para que el material sea intelectualmente relevante; c) Aprobación final del manuscrito para envío; 3. Acreditan que los artículos científicos publicados están completamente libres de datos y/o resultados fraudulentos; 4. Confirmar la cita y la referencia que sean correctas de todos los datos e interpretaciones de datos de otras investigaciones; 5. Reconocen haber informado todas las fuentes de financiamiento recibidas para la realización de la investigación; 6. Autorizar la publicación de la obra, que incluye las fichas del catálogo, ISBN (Número de serie estándar internacional), D.O.I. (Identificador de Objeto Digital) y demás índices, diseño visual y creación de portada, maquetación interior, así como su lanzamiento y difusión según criterio de Atena Editora.

DECLARACIÓN DEL EDITOR

Atena Editora declara, para todos los efectos legales, que: 1. Esta publicación constituye únicamente una cesión temporal del derecho de autor, derecho de publicación, y no constituye responsabilidad solidaria en la creación de manuscritos publicados, en los términos previstos en la Ley. sobre Derechos de autor (Ley 9610/98), en el artículo 184 del Código Penal y en el art. 927 del Código Civil; 2. Autoriza y estimula a los autores a suscribir contratos con los repositorios institucionales, con el objeto exclusivo de difundir la obra, siempre que cuente con el debido reconocimiento de autoría y edición y sin fines comerciales; 3. Todos los libros electrónicos son de acceso abierto, por lo que no los vende en su sitio web, sitios asociados, plataformas de comercio electrónico o cualquier otro medio virtual o físico, por lo tanto, está exento de transferencias de derechos de autor a los autores; 4. Todos los miembros del consejo editorial son doctores y vinculados a instituciones públicas de educación superior, según recomendación de la CAPES para la obtención del libro Qualis; 5. No transfiere, comercializa ni autoriza el uso de los nombres y correos electrónicos de los autores, así como cualquier otro dato de los mismos, para fines distintos al ámbito de difusión de esta obra.

DEDICATORIA

A todas las personas que, de una manera u otra, han contribuido en el camino hacia la realización de este libro.

A nuestros padres, por su amor incondicional y su apoyo incansable, quienes nos enseñaron el valor del esfuerzo y la perseverancia.

A nuestros lectores, por su interés y dedicación, pues sin ustedes este libro no tendría razón de ser. Sus comentarios, críticas y apoyo son el motor que impulsa nuestra creatividad y nuestro deseo de seguir escribiendo.

Este libro es para ustedes, con todo nuestro cariño y gratitud.

INDICE

CAPITULO I: Diseño de páginas web frontend con HTML y CSS.....	5
CAPITULO II: Fundamentos de JavaScript.....	60
CAPITULO III: Desarrollo de soluciones frontend con JavaScript.....	135
CAPITULO IV:SQL y Motores de Base de Datos: SQLServer, MySQL, MariaDB SQLite, MongoDB.....	157
CAPITULO V: Desarrollo de soluciones backend con Node.JS y Express.....	210
CAPITULO VI: Caso Práctico – Empresa de Turismo PATA AMARILLA	286

PRÓLOGO

En un mundo donde la tecnología avanza a pasos agigantados, el desarrollo web se ha convertido en una habilidad indispensable. Si estás aquí, es porque compartimos una pasión: crear aplicaciones web que no solo funcionen, sino que también ofrezcan experiencias significativas a los usuarios.

Este libro nace de nuestras experiencias en el fascinante ecosistema de JavaScript, Node.js, Express, y MySQL. A lo largo de nuestra carrera profesional, hemos descubierto que dominar estas tecnologías puede abrir un abanico infinito de posibilidades, permitiéndonos construir desde soluciones simples hasta complejas plataformas que impactan en la vida de muchas personas.

Comenzaremos explorando JavaScript, el lenguaje que, aunque empezó siendo sencillo, se ha convertido en una herramienta poderosa tanto en el frontend como en el backend. Les guiaremos a través de sus características más modernas y cómo pueden aprovecharlas para escribir código limpio, eficiente y fácil de mantener.

Luego, nos adentraremos en Node.js, un entorno que ha revolucionado la manera en que desarrollamos para el servidor. Aprenderán a usar su arquitectura no bloqueante para crear aplicaciones rápidas y escalables, capaces de manejar múltiples usuarios de manera simultánea.

En la sección dedicada a Express, les mostraremos cómo este framework minimalista puede simplificar enormemente el desarrollo de aplicaciones web. Juntos, veremos cómo gestionar rutas, manejar solicitudes y respuestas, y construir APIs que sean tanto robustas como seguras.

Finalmente, llegaremos a MySQL, el sistema de gestión de bases de datos que, a pesar de los años, sigue siendo el pilar de innumerables aplicaciones. Aprenderán a diseñar y gestionar bases de datos relacionales que se integren perfectamente con tus aplicaciones Node.js, asegurando un rendimiento óptimo.

Escribimos este libro pensando en ti, en lo que me hubiera gustado saber cuándo comenzamos nuestro camino en el desarrollo web. No importa si estás dando tus primeros pasos o si ya tienes experiencia, nuestro objetivo es que encuentres en estas páginas un recurso valioso que te ayude a avanzar y superar los desafíos que se presenten.

INTRODUCCIÓN

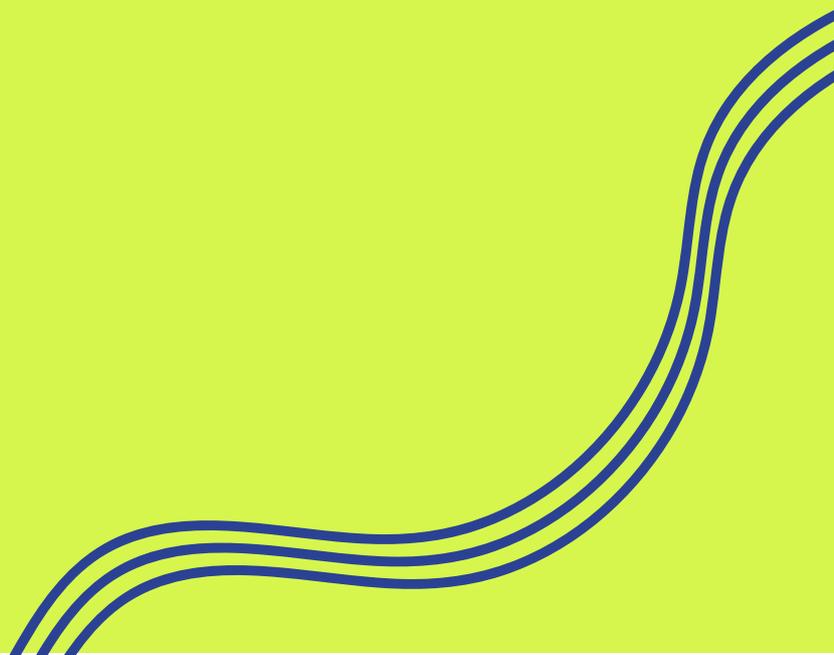
Desde los inicios de la aparición de las tecnologías de información dos componentes avanzaron en forma exponencial, el hardware y software, dependientes el uno del otro, por el lado del software se desarrollan herramientas que facilitan el procesamiento, sistematización, virtualización de muchos quehaceres de la vida cotidiana a nivel de escritorio o vía web. Por la necesidad de masificar dichas herramientas surge la necesidad de nuevas tecnologías, estándares, que permitan la comunicación entre hardware de diferentes partes del mundo. En este contexto aparece el HTML(HyperText Markup Language) por los años 1980 con el afán de compartir documentos, desde sus inicios hasta hoy pasaron por diferentes versiones, el HTML5 ya es un estándar, de la misma manera otras herramientas como lenguajes de programación(JavaScript), CSS con su última versión CSS3 complementarios surgieron para dar un salto gigante en el desarrollo de soluciones web.

En el proceso de desarrollo de soluciones web en este libro estudiaremos a:

- HTML
- CSS
- Bootstrap
- JavaScript
- Formato JSON
- API REST
- PHP
- MySQL
- Node.js
- Framework
- Express
- Motor de plantillas EJS

CAPÍTULO I

Diseño de páginas web frontend con HTML y CSS



1.1. HTML

1.1.1. HTML

HTML es un lenguaje de maquetación que permite definir múltiples etiquetas para agregar los elementos necesarios para presentar o adicionar información a una página web. Estas etiquetas son palabras claves y atributos limitado por los signos mayor (<) y menor (>) Los navegadores de internet leen estos archivos de texto e interpretan las etiquetas para determinar cómo se puede desplegar la página.

```
<html lang="es">
```

En este caso html es la palabra clave y lang es el atributo con valor es. Casi todas las etiquetas HTML se utilizan en pares, de inicio y final con una barra invertida que antecede a la palabra clave:

```
<html lang="es">  
  Contenido  
</html>
```

Los documentos HTML se encuentran estrictamente organizados. Cada parte del documento está diferenciada, declarada y determinada por etiquetas específica.

1.1.2. Versiones HTML

- ✓ **HTML 1:** Primera versión que creó Tim Berners-Lee en 1991.
- ✓ **HTML 2:** Segunda versión del HTML apareció en 1994 y finalizó en 1996 con la publicación del HTML 3.0. Las reglas y la operatividad de esta versión las otorgó el W3C¹.
- ✓ **HTML 3:** En 1996 aparece esta versión del HTML, que añadió muchas posibilidades características, como tablas, applets, scripts, posicionamiento de texto alrededor de las imágenes, etc.
- ✓ **HTML 4:** Versión más común del HTML (HTML 4.01). Apareció por primera vez en 1998 y propuso el uso de tramas, tablas más complejas,

¹ El World Wide Web Consortium (W3C) es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la web a largo plazo.

<html>

Luego de declarar el tipo de documento, debemos comenzar a construir la estructura HTML. Como siempre, la estructura tipo árbol de este lenguaje tiene su raíz en el elemento <html>. Este elemento es el inicio y final de la página:

```
<!DOCTYPE html>
<html lang="es">
  ...
</html>
```

El **atributo lang** en la etiqueta de apertura <html>. Este atributo define el idioma del contenido del documento que estamos creando, en este caso es por español.

<head>

El documento dentro de <html> se divide en dos secciones principales, la primera es la cabecera y el segundo el cuerpo, para lo cual utilizaremos las etiquetas: **<head>** y **<body>** respectivamente.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

Dentro de esta etiqueta **<head>** definiremos el título de la página web con la etiqueta **<title>**; con la etiqueta **<meta>** se declara: el set de caracteres correspondiente con el atributo **charset="utf-8"**, que permita la visualización correcta de tildes o acentos que se encuentran en el contenido de la página; información general a cerca del documento; archivos externos con estilo, código JavaScript, entre otros con la etiqueta **<link>**, que tiene dos atributos **rel** que significa relación y es a cerca de la relación entre el documento y que se está incorporando mediante **href**.

La mayoría de la información contenida en el documento entre esta etiqueta es invisible para el usuario a excepción del título y algunos iconos, pero si es importante para los motores de búsqueda y dispositivos que necesitan hacer vista previa de la página web.

```

<!DOCTYPE html>

<html lang="es">

  <head>
    <meta charset="utf-8" />
    <meta name="description" content="Ejemplo de HTML5" />
    <meta name="keywords" content="HTML5, CSS3, Javascript" />
    <title>EJEMPLO 00001</title>
    <link rel="stylesheet" href="misestilos.css">
  </head>

</html>

```

<body>

Esta etiqueta delimita el cuerpo del documento HTML, permite visualizar todo el contenido de la página web como texto, imágenes, videos, etc. Esta etiqueta tiene atributos, algunas de ellas:

- ✓ **bgcolor:** define el color de fondo de la página.
- ✓ **text:** especifica el color del texto de la página.
- ✓ **link:** color de los vínculos en la página.
- ✓ **alink:** color del vínculo actual o activado en la página.
- ✓ **vlink:** color del vínculo ya visitado.
- ✓ **background:** establece una imagen de fondo en la página web.
- ✓ **style:** permite definir estilos de diseño que afectaran toda la página.

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <meta name="description" content="Ejemplo de HTML5" />
    <meta name="keywords" content="HTML5, CSS3, Javascript" />
    <title>EJEMPLO 00001</title>
  </head>
  <body>
    BIENVENIDO AL CURSO DE PROGRAMACION II
  </body>
</html>

```

1.1.4. El cuerpo de la página, la etiqueta `<body>`

Esta etiqueta delimita el cuerpo del documento HTML, permite visualizar todo el contenido de la página web como texto, imágenes, videos, etc. Esta etiqueta tiene atributos, algunas de ellas:

- ✓ **bgcolor:** define el color de fondo de la página.
- ✓ **text:** especifica el color del texto de la página.
- ✓ **link:** color de los vínculos en la página.
- ✓ **alink:** color del vínculo actual o activado en la página.
- ✓ **vlink:** color del vínculo ya visitado.
- ✓ **background:** establece una imagen de fondo en la página web.
- ✓ **style:** permite definir estilos de diseño que afectaran toda la página.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <meta name="description" content="Ejemplo de HTML5" />
    <meta name="keywords" content="HTML5, CSS3, Javascript" />
    <title>EJEMPLO 00001</title>
  </head>
  <body>
    BIENVENIDO AL CURSO DE PROGRAMACION II
  <body>
</html>
```

Nota. -

Para el color en una página web se especifica el color deseado con el nombre del color en inglés (blue, black, etc.) o mediante números hexadecimales con la siguiente estructura: #RRVVAA (R=rojo, V=verde, A=azul). Por ejemplo, para obtener el color negro, la estructura sería #000000 y para el blanco #FFFFFF, técnicamente.

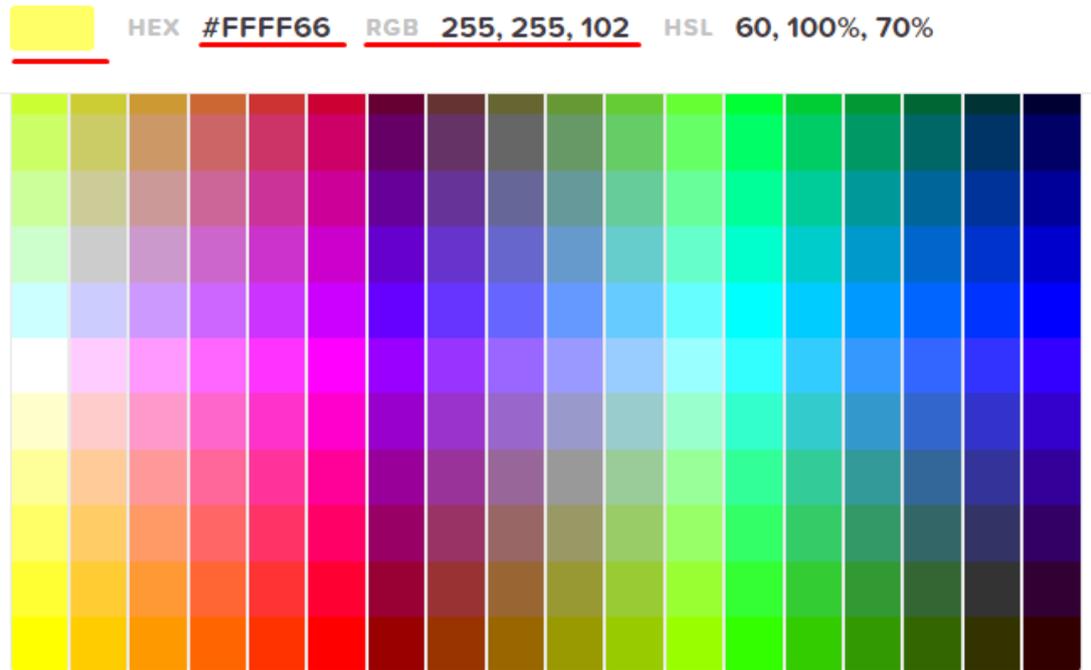


FIGURA N° 01-00002:

Tabla de colores <https://htmlcolorcodes.com/es/tabla-de-colores/>

Estructura más detallada del cuerpo <body></body>

En versiones anteriores a HTML5 la parte visible se estructuraba o diseñaba con las etiquetas <table> y <div>, pero con la nueva versión incorporo nuevos elementos que ayudan a identificar cada sección de la página y del cuerpo.

Una página o aplicación web se divide entre varias áreas visuales para mejorar la interactividad del usuario. Las etiquetas que representan cada nuevo elemento de HTML5 están íntimamente relacionadas con estas áreas.

Una página puede presentar la estructura de la FIGURA N° 01-003

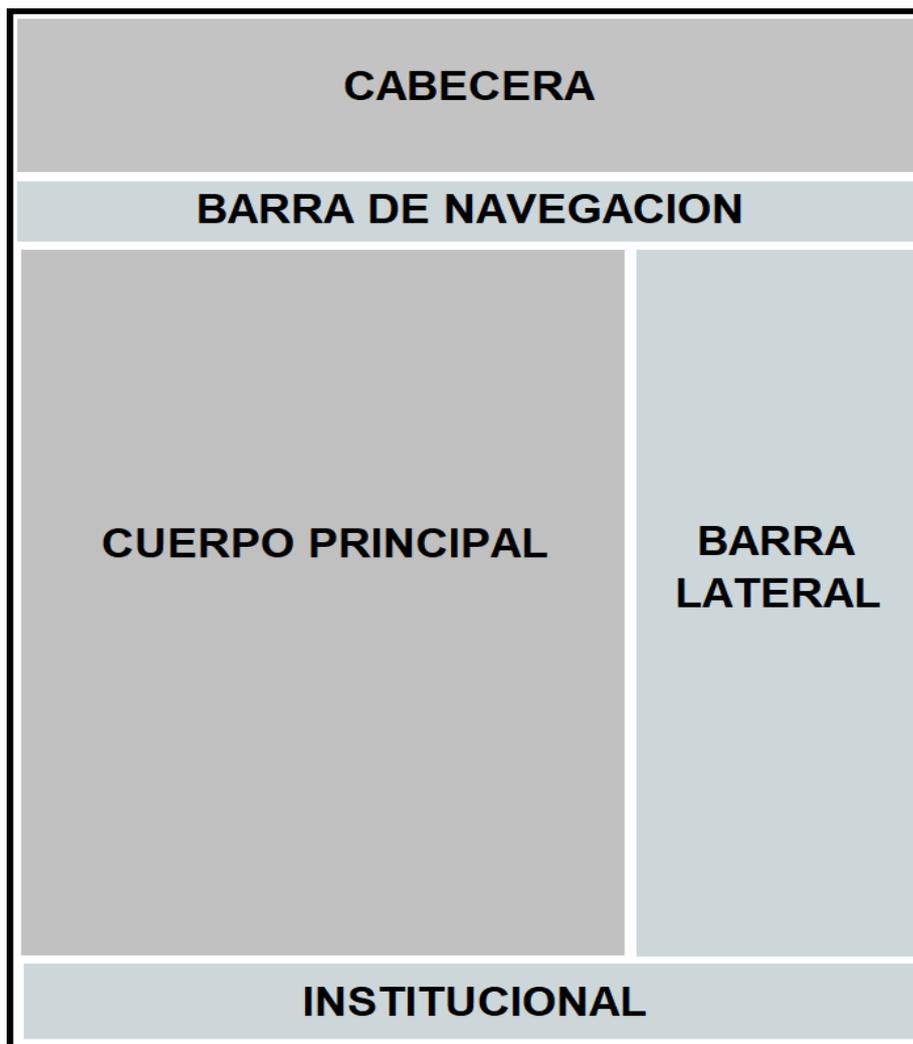


FIGURA N° 01-003: Ejemplo de estructura de una página web

Algunos diseños:



FIGURA N° 01-004. Ejemplo de una página web con una estructura descrito



FIGURA N° 01-005. Ejemplo de una página web con una estructura descrito

La FIGURA N° 01-003, se podría plasmar con las siguientes etiquetas en HTML5

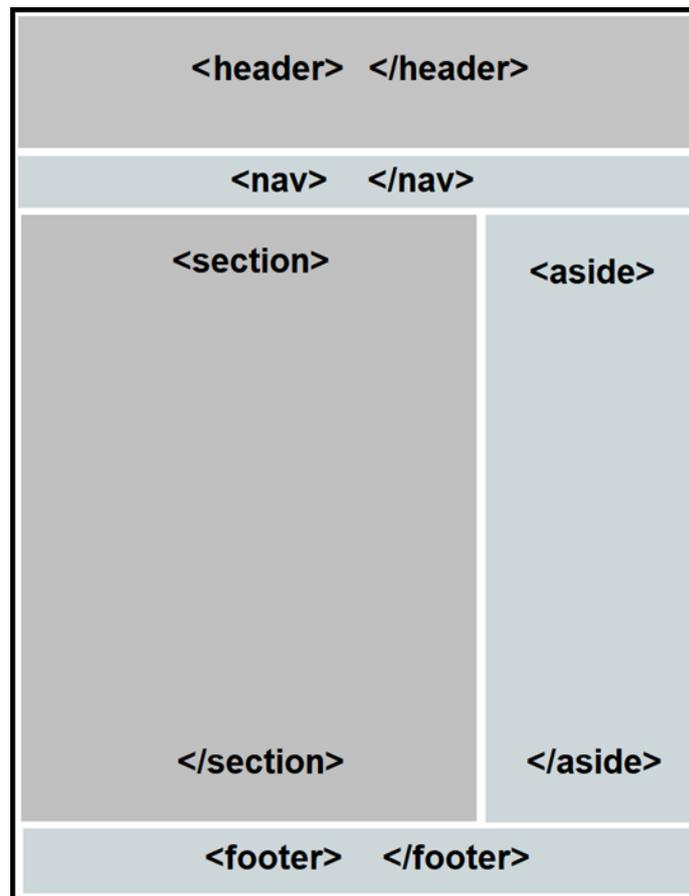


FIGURA N° 01-006: Etiquetas en HTML5 de cada sección de la página web

<header>

En HTML5 se incorporó esta etiqueta <header>, que sirve para poner títulos, subtítulos, logos de la página, no debe confundirse con <head> que sirve para construir la cabecera del documento, en código sería así:

```

<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<meta name="description" content="Ejemplo de HTML5" />
<meta name="keywords" content="HTML5, CSS3, Javascript" />
<title>Titulo de la Pagina Web</title>
</head>
<body>
  <header>
    <h1>TITULO DE LA PAGIAN WEB</h1>
  </header>

</body>
</html>

```

<nav>

Esta etiqueta sirve para agregar una barra de navegación en la página web, en código sería así:

```

.
.
.
<body>
  <header>
<h1>TITULO DE LA PAGIAN WEB</h1>
  </header>
  <nav>
    <ul>
      <li>INICIO</li>
      <li>DOCUMENTOS</li>
      <li>DIRECTORIO</li>
    </ul>
  </nav>

</body>
</html>

```

<section>

Con esta etiqueta según la sección del diseño de la FIGURA N° 01-003, se delimita información más relevante de la página web puede ser diseñada en diferentes formas.

Un ejemplo se ve en el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<meta name="description" content="Ejemplo de HTML5" />
<meta name="keywords" content="HTML5, CSS3, Javascript" />
<title>Titulo de la Pagina Web</title>
</head>
<body>
  <header>
<h1>TITULO DE LA PAGIAN WEB</h1>
</header>
  <nav>
    <ul>
      <li>INICIO</li>
      <li>DOCUMENTOS</li>
      <li>DIRECTORIO</li>
    </ul>
  </nav>
  <section>
  En esto se llena contenido
  </section>
</body>
</html>
```

<aside>

En el diseño planteado en la FIGURA N° 01-003, la columna **Barra Lateral**, que está al lado derecho de la sección **Información Principal**, generalmente se podría poner notas no tan relevantes con la información principal o lo que se considere conveniente.

Esta etiqueta también se puede utilizar para poner información u otras cosas en la parte derecha de la **Información Principal** por lo tanto <aside> solo describe la información que contiene y no el lugar dentro de la estructura, este elemento se puede utilizar también dentro <section>. Ejemplo en el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<meta name="description" content="Ejemplo de HTML5" />
<meta name="keywords" content="HTML5, CSS3, Javascript" />
<title>Titulo de la Pagina Web</title>
</head>
<body>
  <header>
<h1>TITULO DE LA PAGIAN WEB</h1>
</header>
  <nav>
    <ul>
      <li>INICIO</li>
      <li>DOCUMENTOS</li>
      <li>DIRECTORIO</li>
    </ul>
  </nav>
  <section>
</section>
  <aside>
    <blockquote>Nota uno</blockquote>
    <blockquote>Nota dos</blockquote>
  </aside>
</body>
</html>
```

<footer>

Esta etiqueta nos permite poner información en la parte final de la página web según el diseño planteado en la FIGURA N° 01-003, tal como se muestra en el código siguiente:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<meta name="description" content="Ejemplo de HTML5" />
<meta name="keywords" content="HTML5, CSS3, Javascript" />
<title>Titulo de la Pagina Web</title>
</head>
<body>
  <header>
    <h1>Portada</h1>
  </header>
  <nav>
    <ul>
      <li>INICIO</li>
      <li>DOCUMENTOS</li>
      <li>DIRECTORIO</li>
    </ul>
  </nav>
  <section>
  </section>
  <aside>
    <blockquote>Nota uno</blockquote>
    <blockquote>Nota dos</blockquote>
  </aside>
  <footer>
    Correo: amarilis2020@gmail.com
    Telefono: (062)515330
  </footer>
</body>
</html>
```

1.1.5. Formularios HTML <form>

Cuando una página web requiera datos para diferentes propósitos (por ejemplo, páginas de búsqueda, recojo de datos, etc.), contienen elementos o controles que hace posible el ingreso de estos para luego procesarlas, en HTML existe la etiqueta **<form>** que hace posible y contiene diferentes controles de: texto, opciones, listas, listas desplegables, botones, etc.

FORMULARIO DE INGRESO DE DATOS

DNI	22751077
APELLIDOS Y NOMBRES	JUAN DE DIOS GARCIA, JOSE
CORREO	garcia2020@hotmail.com
REGION	HUANUCO ▾
PROVINCIA	HUANUCO ▾
DISTRITO	AMARILIS ▾
DIRECCION	dos de mayo N°45287
SEXO	<input checked="" type="radio"/> MASCULINO <input type="radio"/> FEMENINO
	GRABAR CANCELAR

FIGURA N° 01-007: Formulario de Ingreso de Datos de una persona

En la FIGURA N° 01-007 se puede apreciar un formulario que contiene varios controles que permite interactuar con el usuario para el recojo de datos, para ello a nivel de código la etiqueta **<form>** tiene la siguiente estructura básica:

```
<form id="form1" name="form1" method="post" action="graba.php">
```

El **id**, **name**: son identificadores del formulario; **method**: es el método HTTP usado para envío del conjunto de datos del formulario, los valores podrían ser: **post**, **get**; **action** Sin dudas, este es el principal atributo para el funcionamiento de un formulario. Su valor debe ser la URL de un archivo programado en algún lenguaje de servidor, como PHP, ASP, JSP, etc. para que ese archivo reciba las variables enviadas desde el formulario (una por cada control de formulario), y pueda hacer algo con esos datos (típicamente, insertarlos en una base de datos en el servidor, o enviarlos por email al administrador del sitio).

ALGUNOS CONTROLES

Dentro de la etiqueta <form> se puede insertar controles que permitan ingresar y seleccionar datos, interactuar, etc., para insertar un control generalmente se utiliza **input type="control"**

text

Control que permiten ingresar textos, contraseñas en un formulario:

```
<input type="text" name="txtdni" autofocus />
<input type="password" name="txtpassword">
<textarea name="txtdescripcion"></textarea>
```

checkbox

Este control es un interruptor de encendido/apagado que pueden ser conmutados por el usuario. Una casilla de verificación está "marcada" cuando se establece el atributo **checked** del elemento de control. Cuando se envía un formulario, solamente pueden tener éxito los controles de casillas de verificación que estén marcadas.

```
<input type="checkbox" name="chkDeporte" value="checkbox">
```

radiobutton

Los radios botones son como las casillas de verificación, excepto en que cuando varios comparten el mismo nombre de control, son mutuamente exclusivos: cuando uno está "encendido", todos los demás con el mismo nombre se "apagan".

```
<input type="radio" name="radiobutton" value="radiobutton"
checked="checked" /> MASCULINO
<input type="radio" name="radiobutton" value="radiobutton" />FEMENINO
```

Select

Con este control se crea una lista desplegable para que el usuario pueda seleccionar, para seleccionar por defecto un elemento de la lista se utiliza

selected="selected".

```
<select name="select3">
  <option>HUANUCO</option>
  <option selected="selected">AMARILIS</option>
  <option>PILLCOMARCA</option>
</select>
```

botones

botones de envío (submit): Cuando se activa, un botón de envío envía un formulario. Un formulario puede contener más de un botón de envío.

```
<input type="submit" name="Submit" value="GRABAR" />
```

botones de reinicialización (reset): Cuando se activa, un botón de reinicialización reinicializa todos los controles a sus valores iniciales.

```
<input type="reset" name="Submit2" value="CANCELAR" />
```

Botones genéricos son aquellos que no tiene ninguna función definida, sino la que nosotros queramos darle. Para insertar un botón genérico usaremos la etiqueta INPUT con TYPE="button"

```
<INPUT TYPE="button" VALUE="Pulseme" ONCLICK="código script">
```

```

1 <!DOCTYPE html >
2 <html>
3 <head>
4 <meta charset="utf8" />
5 <title>FORMULARIOS</title>
6 <style type="text/css">
7 <!--
8 .Estilo1 {
9     font-size: 24px;
10    font-weight: bold;
11    color: #000099;
12 }
13 -->
14 </style>
15 </head>
16
17 <body>
18 <form id="form1" name="form1" method="post" action="">
19 <table align="center">
20 <tr>
21 <td width="722">
22 <div align="center" class="Estilo1">FORMULARIO DE INGRESO DE DATOS </div></td>
23 </tr>
24
25 <tr>
26 <td>
27
28 <table width="702" border="1" align="center">
29 <tr>
30 <td width="267">DNI</td>
31 <td width="419"><label>
32 <input type="text" name="textfield" />
33 </label></td>
34 </tr>
35 <tr>
36 <td>APELLIDOS Y NOMBRES </td>
37 <td><label>
38 <input name="textfield2" type="text" size="60" />
39 </label></td>
40 </tr>
41 <tr>
42 <td>CORREO</td>
43 <td><label>
44 <input name="textfield3" type="text" size="60" />
45 </label></td>
46 </tr>
47 <tr>
48 <td>REGION</td>
49 <td><label>
50 <select name="select">
51 <option>LIMA</option>
52 <option>AREQUIPA</option>
53 <option>JUNIN</option>
54 <option selected="selected">HUANUCO</option>
55 </select>
56 </label></td>
57 </tr>
58 <tr>
59 <td>PROVINCIA</td>
60 <td><label>
61 <select name="select2">
62 <option selected="selected">HUANUCO</option>
63 <option>LEONCIO PRADO</option>
64 <option>AMBO</option>
65 <option>FACHITEA</option>
66 </select>
67 </label></td>
68 </tr>
69 <tr>
70 <td>DISTRITO</td>
71 <td><label>
72 <select name="select3">
73 <option>HUANUCO</option>
74 <option selected="selected">AMARILIS</option>
75 <option>PILCOMARCA</option>
76 </select>
77 </label></td>
78 </tr>
79 <tr>
80 <td><label>DIRECCION</label></td>
81 <td><label>
82 <input name="textfield4" type="text" size="60" />
83 </label></td>
84 </tr>
85 <tr>
86 <td>SEXO</td>
87 <td><label>
88 <input name="radiobutton" type="radio" value="radiobutton" checked="checked" />
89 MASCULINO
90 <input name="radiobutton" type="radio" value="radiobutton" />
91 FEMENINO</label></td>
92 </tr>
93 <tr>
94 <td>&nbsp;</td>
95 <td><input type="submit" name="Submit" value="GRABAR" />
96 <input type="submit" name="Submit2" value="CANCELAR" /></td>
97 </tr>
98 </table>
99
100 </td>
101 </tr>
102 </table>
103 </form>
104 </body>
105 </html>
106

```

FIGURA N° 01-008: Código HTML del diseño de la página de la FIGURA N° 01-007

1.2. CSS

1.2.1. CSS (Cascading Style Sheets)

“El CSS es un lenguaje de estilos empleado para definir la presentación, el formato y la apariencia de un documento de marcaje, sea html, xml, o cualquier otro. Comúnmente se emplea para dar formato visual a documentos html o xhtml que funcionan como espacios web. También puede ser empleado en formatos xml, u otros tipos de documentos de marcaje para la posterior generación de documentos.

Las hojas de estilos nacen de la necesidad de diseñar la información de tal manera que podemos separar el contenido de la presentación y, así, por una misma fuente de información, generalmente definida mediante un lenguaje de marcaje, ofrecer diferentes presentaciones en función de dispositivos, servicios, contextos o aplicativos. Por lo que un mismo documento html, mediante diferentes hojas de estilo, puede ser presentado por pantalla, por impresora, por lectores de voz o por tabletas braille. Separamos el contenido de la forma, composición, colores y fuentes...” (CSS3 y Javascript avanzado, Jordi Collell Puig)

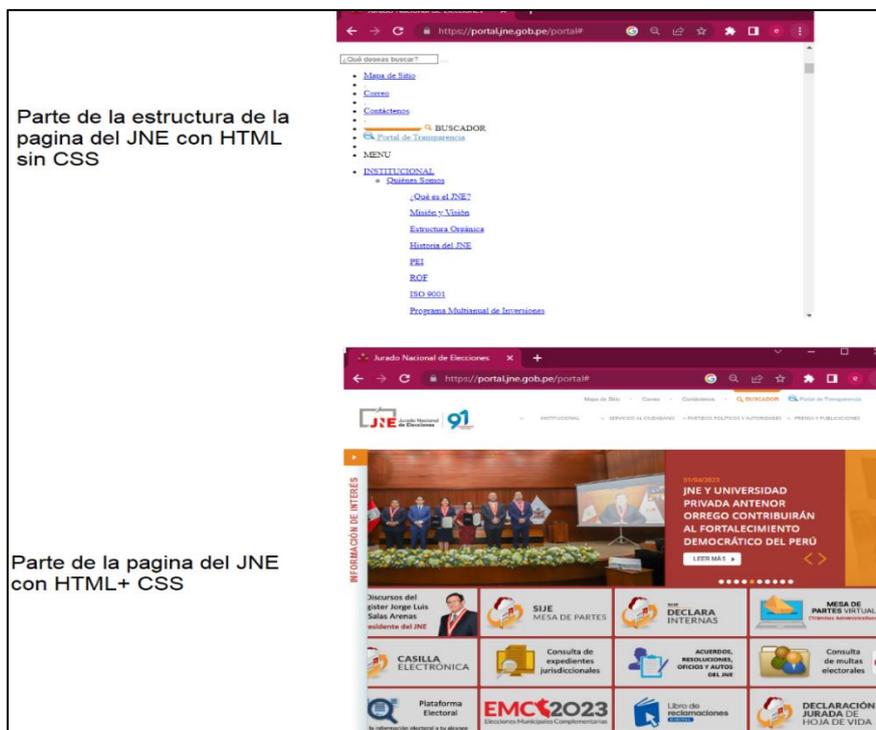


FIGURA N° 01-009: Pagina web HTML sin CSS y con CSS

1.2.2. Versiones del CSS

- ✓ **CSS 1:** la primera versión del CSS se usó a partir de 1996. Se sentó las bases de este lenguaje que permite mostrar las páginas web con colores, márgenes, tipos de letra, etc.
- ✓ **CSS 2:** apareció en 1999 y luego se completó con la versión CSS 2.1. Esta versión nueva del CSS añadió numerosas opciones. Ahora podemos usar con bastante exactitud las técnicas de colocación para mostrar objetos en el lugar que queremos que estén en la página.
- ✓ **CSS 3:** esta es la última versión, esta incluye funciones muy esperadas, como bordes redondeados, degradado, sombras, etc.

✓

El diseño básico de una página descrito en la FIGURA N° 01-003, cuya estructura es el siguiente:

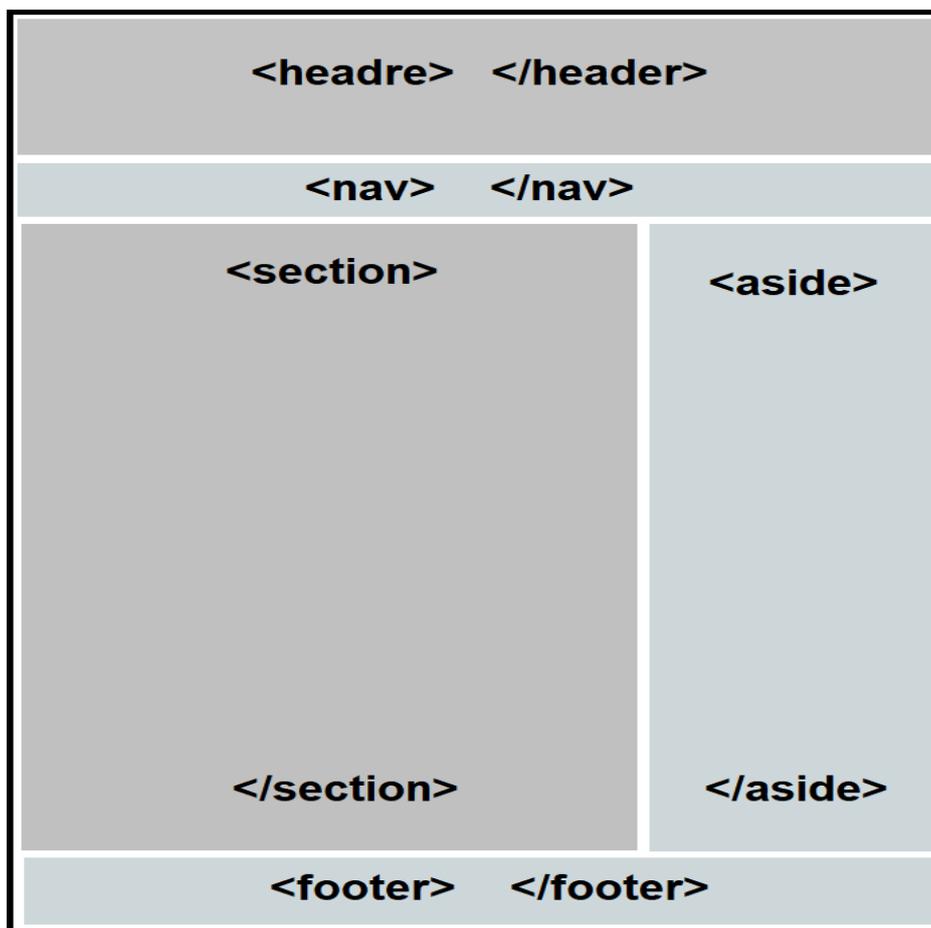


FIGURA N° 01-010: Estructura básica de una página web

Cuyo código en HTML5 es el siguiente:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<meta name="description" content="Ejemplo de HTML5" />
<meta name="keywords" content="HTML5, CSS3, Javascript" />
<title>Titulo de la Pagina Web</title>
</head>
<body>
  <header>
    <h1>TITULO DE LA PAGIAN WEB</h1>
  </header>
  <nav>
    <ul>
      <li>INICIO</li>
      <li>DOCUMENTOS</li>
      <li>DIRECTORIO</li>
    </ul>
  </nav>
  <section>
  </section>
  <aside>
    <blockquote>Nota uno</blockquote>
    <blockquote>Nota dos</blockquote>
  </aside>
  <footer>
    Correo: amarilis2020@gmail.com
    Telefono: (062)515330
  </footer>
</body>
</html>
```

Si se visualiza este código en el navegador Google Chrome sería algo así:



FIGURA N° 01-011: Pagina sin aplicar CSS

Como se puede apreciar, no está como realmente quisiéramos de acuerdo a la estructura diseñada. Esto es básicamente porque cada navegador ordena las etiquetas por defecto de acuerdo a su tipo: **block**(bloque) o **inline** (en línea). Esto se refleja en la forma en que las etiquetas son mostradas en pantalla.

- ✓ Elementos block son posicionados uno sobre otro hacia abajo en la página.

```

<headre> </header>
<nav> </nav>
<section> </section>
<aside> </aside>
<footer> </footer>

```

- ✓ Elementos inline son posicionados lado a lado, uno al lado del otro en la misma línea, sin ningún salto de línea a menos que ya no haya más espacio horizontal para ubicarlos.

```

<headre> </header> <nav> </nav> <section> </section>
<aside> </aside> <footer> </footer>

```

Modelo de Caja

Cada navegador considera cada elemento HTML como una caja. Una página web es un conjunto de cajas ordenadas siguiendo ciertas reglas, estas reglas son establecidas por estilos definidos por defecto en los navegadores o por los diseñadores utilizando **CSS** que tiene un set predeterminado de propiedades que permiten sobrescribir los estilos predefinidos. Combinando estas propiedades se forman reglas que se pueden utilizar para agrupar cajas y lograr la correcta disposición en pantalla. Todas estas reglas aplicadas juntas constituyen lo que se llama un **modelo de caja**.

1.2.3. Aplicando estilos CSS a una página web

Estilos en línea

Se puede aplicar estilos CSS a una página web, directamente asignando estilos dentro de las etiquetas con el atributo `style`. En el siguiente código en la etiqueta `<p>` se modifica el atributo `style` con el valor `font-size: 40px`. Este estilo cambia el tamaño por defecto del texto dentro de la etiqueta `<p>` a un tamaño de 40 píxeles.

```

<! DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>PROBANDO ESTILOS CSS EN LINEA</title>
</head>
<body>
  <p style="font-size: 40px">PROGRAMACION II</p>
</body>
</html>

```

Estilos embebidos

Otra de las formas para aplicar estilos a una página web es insertar estos en la cabecera del documento y luego utilizar referencias para aplicar a las etiquetas HTML correspondiente:

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>PROBANDO ESTILOS CSS EMBEBIDOS</title>
  <!-- ----- -->
  <!--Estilos CSS embebidos -->
  <style>
    p { font-size: 40px }
  </style>
  <!-- ----- -->
</head>
<body>
  <p >PROGRAMACION II</p>
</body>
</html>

```

Estilos con archivos externos

En las dos formas anteriores para aplicar estilos a los elementos de una página hay desventajas cuando se tiene varios documentos en donde queremos que las mismas características o se uniformice ciertos diseños. Lo que se podría hacer es copiar en cada página estos estilos, lo que conlleva a una dificultad de mantenimiento. La solución sería tener todos los estilos a archivos externos y luego referenciarlo utilizando la etiqueta <link> para insertar el archivo dentro de los documentos que se requiera.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>PROBANDO ESTILOS CSS EMBEBIDOS</title>

  <!-- ----- -->
  <!--Estilos CSS con archivo externo -->
  <link rel="stylesheet" href="estilo.css">
  <!-- ----- -->
</head>
<body>
  <p >PROGRAMACION II</p>
</body>
</html>

```

El archivo de estilo externo de nombre: **estilo.css**

```

p {
  font-size: 60px
}

/*estilos para el selector body*/
body {
  text-align: center;
  background: #BBFFFF;
}

```

Lo cual, la página en Google Chrome se vería así:



FIGURA N° 01-012: Aplicando Estilos CSS

1.2.4. Reglas CSS

Las reglas **CSS**, se componen de selector y declaración, estando la declaración compuesta por una serie de pares **propiedad: valor**:

A continuación, se presenta a notación de algunas expresiones que define los valores de las propiedades:

- ✓ Si encontramos que algo se debe escribir cómo **valorA valorB**, separados por espacios sin ningún carácter especial, esto significa que se debe escribir tal cuál a la hora de escribir los valores para la propiedad.
- ✓ Si tenemos elementos separados por barras verticales, |, como en el caso de **<porcentaje> | <medida> | inherit**, significa que el valor de la propiedad puede ser uno de los que se indica (los que están entre < y > serían un % válido o una medida válida, mientras que sin < y > significa que hay que ponerlo literal).
- ✓ Si tenemos elemento separados por ||, se puede indicar uno o más de los valores indicados y en cualquier orden, como en el caso de: **<color> || <estilo> || <medida>**.
- ✓ Si tenemos algo del tipo **expresión***, implica que podemos poner la expresión cero o más veces.
- ✓ Si tenemos **expresión+**, implica que podemos ponerlo 1 o más veces.
- ✓ ¿Si tenemos **expresión?**, implica que se puede tener o no la expresión.
- ✓ Si tenemos expresión **{valor_min, valor_max}**, implica que la expresión puede repetirse un número de veces dentro del rango dado por los extremos indicados.

Ejemplos:

- ✓ [**<family-name>**,]* → Pueden aparecer cero o más veces el par compuesto de un nombre de familia de fuente, seguido de una coma.
- ✓ **<url>? <color>** → Puede ponerse una url o no, e irá seguida de un color, en este caso obligatorio.
- ✓ [**<medida> | thick | thin**] {1,4} → Puede ponerse de 1 a 4 veces un elemento de los indicados: una medida, la palabra thick o la palabra thin.

Selectores

Los selectores son los elementos a los que se aplica estilos o una o más reglas y estos a su vez podría afectar a varios elementos.

Selectores básicos

Selector universal: el asterisco (*). El estilo en cuestión se aplica a todo elemento de la página.

```
*{
  margin: 0;
  padding: 0;
}
```

Selector de tipo o etiqueta: viene dado por el nombre de la etiqueta, y afecta a todos los elementos que tienen dicha etiqueta, como en estos ejemplos:

```
p {
  font-size: 60px;
  color: black;
}

h1 {
  color: red;
}

h2 {
  color: blue;
}
```

Un conjunto de reglas o estilo se puede aplicar a varios selectores, cada uno de ellos se separan con comas. Además, aspectos adicionales de cada elemento se puedan poner en reglas adicionales por separado.

```
h1,h2,h3 {
  font-weight: normal;
  font-family: Arial, Helvetica, sans - serif;
  color: #BBFFFF;
}

h1 { font-size: 2px; }
h2 { font-size: 1.5px; }
h3 { font-size: 1.2px; }
```

Selector descendente: indica que el estilo se aplicará a una etiqueta de un determinado tipo, siempre que ésta esté contenida en algún nivel dentro de otra de otro determinado tipo indicado.

```
a span { color: red; }
```

Esta regla se aplica a todos los selectores `` que se encuentran dentro de `<a>`, por ejemplo, en el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>PROBANDO ESTILOS CSS CON ARCHIVO EXTERNO</title>
  <!-- ----- -->
  <!--Estilos CSS con archivo externo -->
  <link rel="stylesheet" href="estilo001.css">
  <!-- ----- -->
</head>
<body>
  <p>
    <span>
      PROGRAMACION II
    </span>
    <a href="" > <span>SUMILLA</span> </a>
  </p>
</body>
</html>
```

Se vería si:



FIGURA N° 01-013: Aplicando Estilos, selectores descendentes CSS

Se puede anidar de la siguiente forma:

```
p a span em { text-decoration: underline; }
```

En este caso se aplicará la regla a los elementos de tipo **em** (siempre se aplica al último elemento) que este en algún nivel dentro de la etiqueta ****, que además este esté dentro de **<a>** y que este esté dentro de **<p>**

Selector individual: El valor del atributo **id** identifica de manera única a un elemento en todo el documento, esto significa que no puede ser duplicado. Para referenciar un elemento en particular usando el atributo **id** desde nuestro archivo CSS la regla debe ser declarada con el símbolo **#** al frente del valor que usamos para identificar el elemento:

```
#texto1{ font-size: 20px; }
```

En código HTML se referenciaría así:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>PROBANDO ESTILOS CSS CON EL ATRIBUTO ID</title>
  <link rel="stylesheet" href="estimo001.css">
</head>
<body>
  <p id="texto1">PROGRAMACION II</p>
</body>
</html>
```

Selector de clase: afecta a todos los elementos calificados dentro de una clase, con class dentro del código HTML. Se debe declarar la regla CSS con un punto antes del nombre:

```
.texto1 { font-size: 20px }
```

El código HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>PROBANDO ESTILOS CSS CON EL ATRIBUTO ID</title>
  <link rel="stylesheet" href="estimo001.css">
</head>
<body>
  <p class="texto1">Mi texto 1</p>
  <p class="texto1">Mi texto 2</p>
  <p>Mi texto 3</p>
</body>
</html>
```

Se puede crear una regla que referencia la clase llamada **texto1** pero solo para los elementos de tipo **<p>**. Si cualquier otro elemento tiene el mismo valor en su atributo **class** no será modificado por esta regla en particular:

```
p.texto1 { font-size: 20px }
```

Selectores utilizando cualquier atributo: La última versión de CSS ha incorporado nuevas formas de referenciar elementos HTML. Uno de ellas es el Selector de Atributo. Ahora podemos referenciar un elemento no solo por los atributos id y class sino también a través de cualquier otro atributo, la siguiente regla se aplica al elemento **<p>** que tienen un atributo llamado **name**.

```
p[name] { font-size: 20px }
```

1.2.5. CSS: Fuentes web

*“El **Web Open Font Format (WOFF)** es un formato de tipo de letra para usarse en páginas web. Fue desarrollado durante el año 2009,2 y está en el proceso de normalización como una recomendación por el Grupo de Trabajo de Tipos de Letra Web del World Wide Web Consortium (W3C).3 En su mayoría, WOFF contiene tipografías OpenType o TrueType con compresión, además de metadatos. Su objetivo es permitir la distribución de tipografías desde un servidor a un equipo cliente en una red, en favor del ancho de banda”* (https://es.wikipedia.org/wiki/Web_Open_Font_Format).

Para crear fuentes WOFF se puede utilizar webs como **Font Squirrel**(<https://www.fontsquirrel.com/tools/webfont-generator>) que permite subir una fuente TTF u OTF y se descarga la misma fuente en los formatos WOFF2, WOFF, EOT y SVG.

La regla arroba @font-face

Esto permite utilizar fuentes web mediante el uso de la regla arroba @font-face.

```

@font-face{
  font-family: 'BallparkWeiner';
  src: url('fonts/ballpark.eot');
  src: url('fonts/ballpark.eot?#iefix') format('embedded-opentype'),
       url('fonts/ballpark.woff') format('woff'),
       url('fonts/ballpark.ttf') format('truetype'),
       url('fonts/ballpark.svg#BallparkWeiner') format('svg');
  font-weight: normal;
  font-style: normal;
}

header h1{
  font-family: 'BallparkWeiner', serif;
  font-size: 2.5em;
  font-weight: normal;
}

```

En La regla **@font-face**:

- ✓ La propiedad **font-family** establece el nombre con el que vas a referirte a la fuente, se puede poner cualquier nombre. Si el nombre contiene espacios en blanco, es necesario escribir el nombre entre comillas.
- ✓ La propiedad **src** especifican las rutas del fichero de la fuente, la **url()** apunta a un archivo de tipo de letra que tenemos que importar a nuestro CSS y el formato de cada archivo de tipo de letra (**format**).

Una vez establecido el nombre de la fuente, se puede hacer referencia a ella en las propiedades **font-family**.

Google Fonts / Google Fonts API

Google tiene el servicio de alojamiento de fuentes libres, **Google Fonts**(<https://fonts.google.com/>) y permite descargar las fuentes en formato TTF.



1.3. Entornos de desarrollo web

Para el diseño de una página web bastaría con el bloc de notas o cualquier otro editor de texto básico de la computadora, pero requeriría de más esfuerzo y tiempo, por lo que en necesariamente recurrir a editores de código que permita integrar HTML, CSS, JavaScript, PHP entre otros lenguajes de programación para desarrollar soluciones web de manera más intuitiva.

Algunos editores de código para el entorno de desarrollo web:

- ✓ Visual Studio Code
- ✓ Sublime Text
- ✓ Atom
- ✓ Notepad++
- ✓ CoffeCup HTML Editor
- ✓ TextMate
- ✓ NetBeans

En Este libro para el desarrollo de los ejemplos y soluciones web utilizaremos el **Visual Studio Code**

1.3.1. Visual Studio Code

Este editor de código gratuito desarrollado por Microsoft, es una herramienta multiplataforma, intuitiva, completo para el diseño de soluciones web, permite integrar extensiones, librería, entre otros que facilita la codificación y diseño.

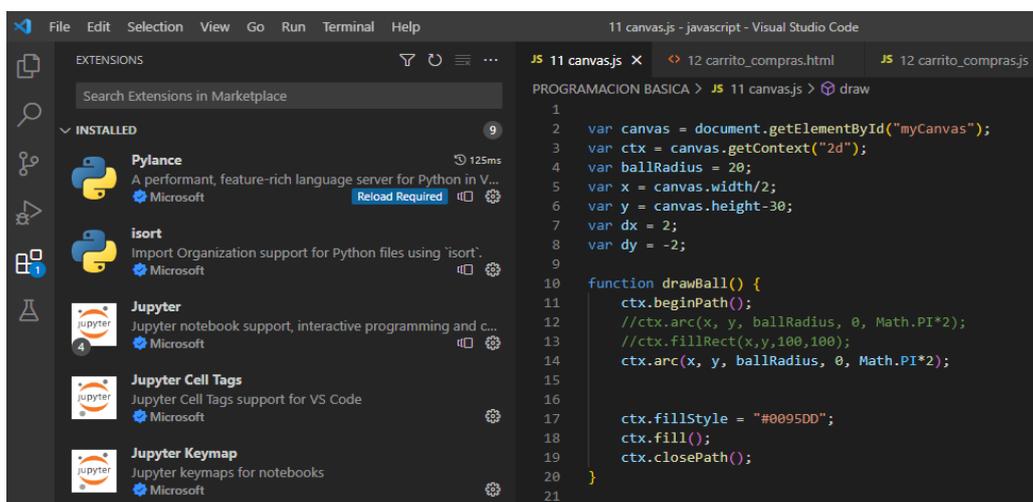


FIGURA N° 01-014: Entorno de desarrollo Visual Studio Code

Para utilizar descargar de su portal:

<https://code.visualstudio.com/>

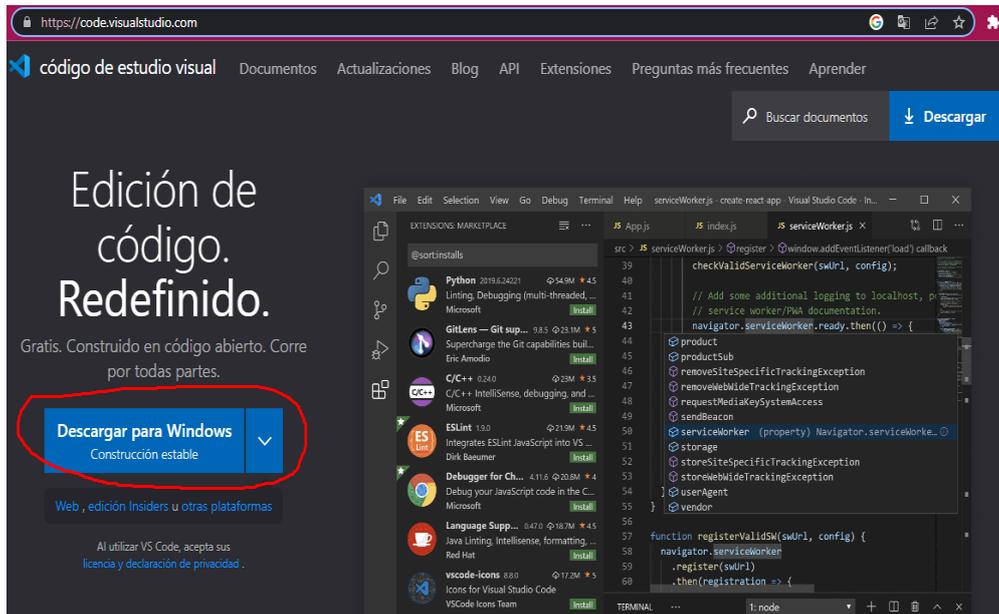


FIGURA N° 01-015: Pagina para descargar Visual Studio Code

Una vez descargado su instalación es sencillo e intuitivo:

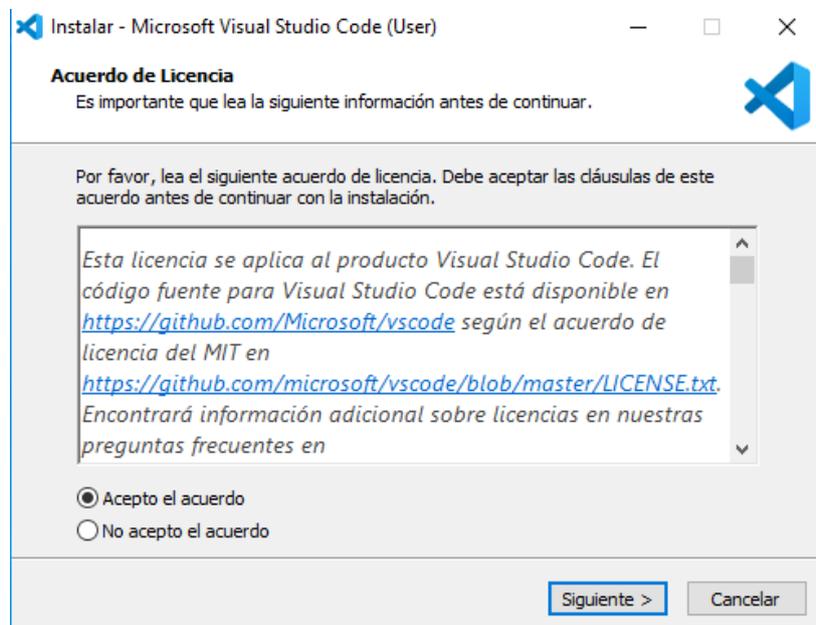


FIGURA N° 01-015: Para instalar Visual Studio Code, aceptar el acuerdo y siguiente, siguiente...

Entorno de trabajo del Visual Studio Code

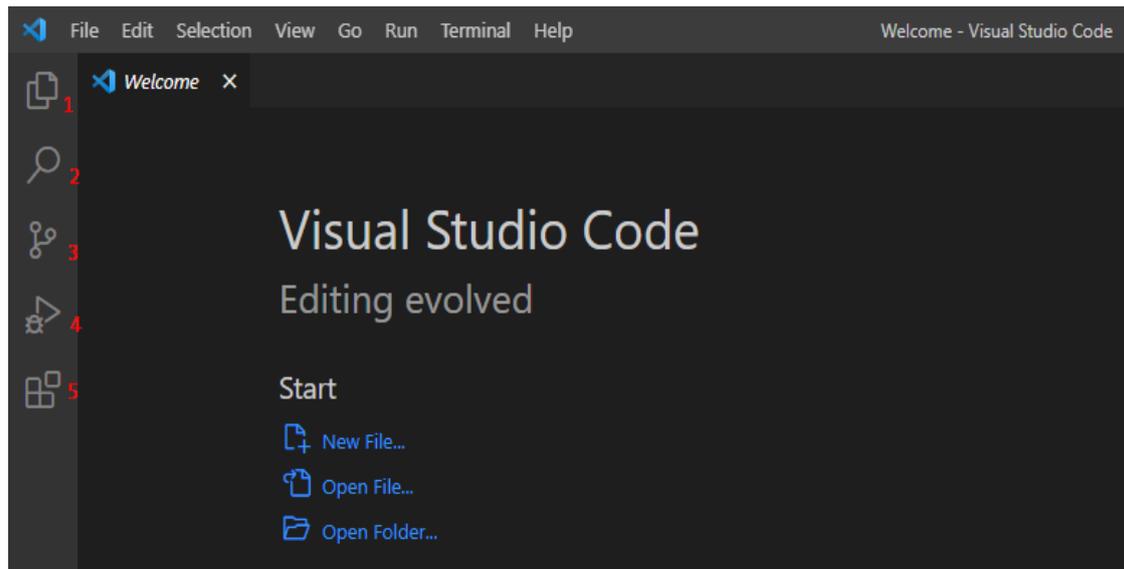


FIGURA N° 01-016: Ventana principal del Visual Studio Code con su menú e iconos de acceso rápido (1,2,3,4,5)

Los iconos de acceso rápido: el 1 permite ver los archivos de trabajo en el directorio actual, el 2 permite hacer búsquedas, el 4 permite ejecutar código, el 5 permite instalar extensiones

Permite trabajar con dos tipos de áreas de trabajo:

Carpetas

El área de trabajo más utilizado de Visual Studio Code es una carpeta con todo su contenido incluido las subcarpetas.

Para abrir una carpeta elegir la opción del menú **Archivo > Abrir Carpeta...** y seleccionar la carpeta que se requiera como carpeta raíz del área de trabajo o sino arrastrar a la ventana

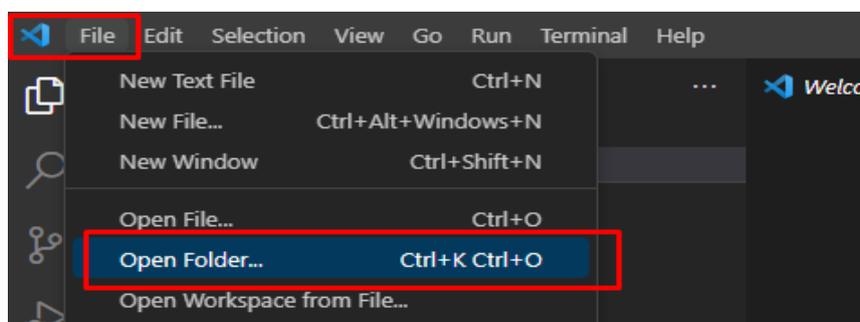


FIGURA N° 01-017: Menú **Archivo > Abrir Carpeta**

Espacios de trabajo

Se puede crear áreas de trabajo con varias carpetas distintas, para ello ir a la opción del menú Archivo > Agregar carpeta al área de trabajo ...

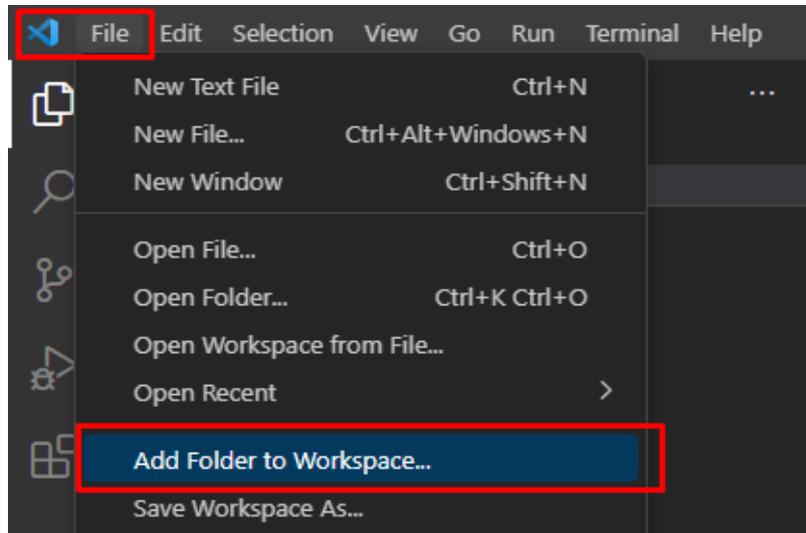


FIGURA N° 01-018: Menú Archivo > Agregar carpeta al área de trabajo...

Para guardar el área de trabajo seleccionar la opción de menú Archivo > **Guardar área de trabajo como**

1.3.2. Extensiones en Visual Studio Code

Las extensiones permiten ampliar las características del Visual Studio Code. Para agregar una nueva extensión hacer click en el icono 1 señalado en la FIGURA N° 01-016

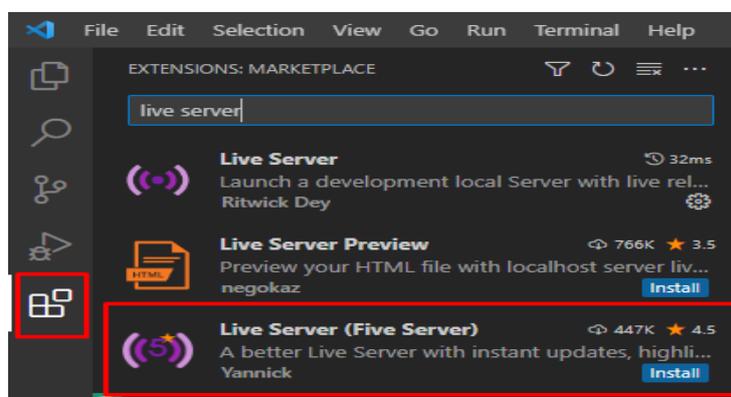


FIGURA N° 01-019: Icono para agregar nuevas extensiones

Algunas extensiones de utilidad

Live Server

Esta extensión crea un pequeño servidor local para que puedas ver lo que estás haciendo y se actualice automáticamente con cada cambio.

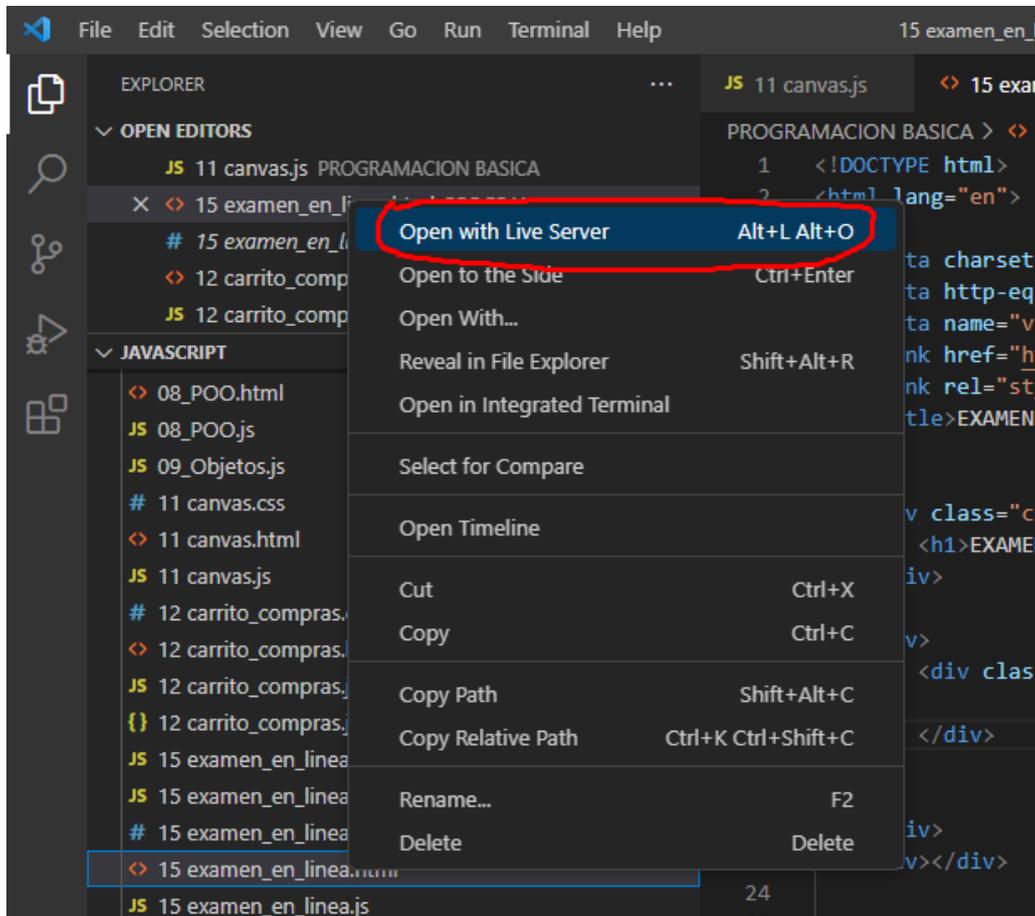


FIGURA N° 01-020: El Live Server permite crear un servidor local que se actualiza automáticamente según realizas cambios en el código y grabas

Prettier

Esta extensión permite formatear el código según las reglas de cada lenguaje. Soporta la sintaxis de una gran cantidad de lenguajes, como JavaScript, JSX, Flow, TypeScript, JSON, HTML, Vue, Angular, CSS, Less, SCSS, GraphQL, Markdown, CommonMark, MDX y YAML. Prettier

Indent rainbow

Esta extensión permite indicarte a través de barras de colores en qué nivel de indentación está el código y esto ayuda a que sea más fácil de leer.

Eslint

Esta extensión para JavaScript ayuda a encontrar errores de sintaxis, lógica o guías de estilo en el código y sugiere las posibles soluciones.

SVG

Esta extensión ayuda con la sintaxis de SVG, resalta las coordenadas y el autocompletado de sintaxis, permite pre visualizar SVG.

La documentación detallada del Visual Studio Code en:

<https://code.visualstudio.com/docs>

1.4. Ejemplos

1.4.1. Creación de Login con HTML y CSS

Se creará este diseño:



FIGURA N° 01-021: Diseño de un Login con HTML y CSS

Para empezar, creamos una carpeta “Login” en cualquiera parte de la computadora, luego arrastramos a la ventana de Visual Studio Code

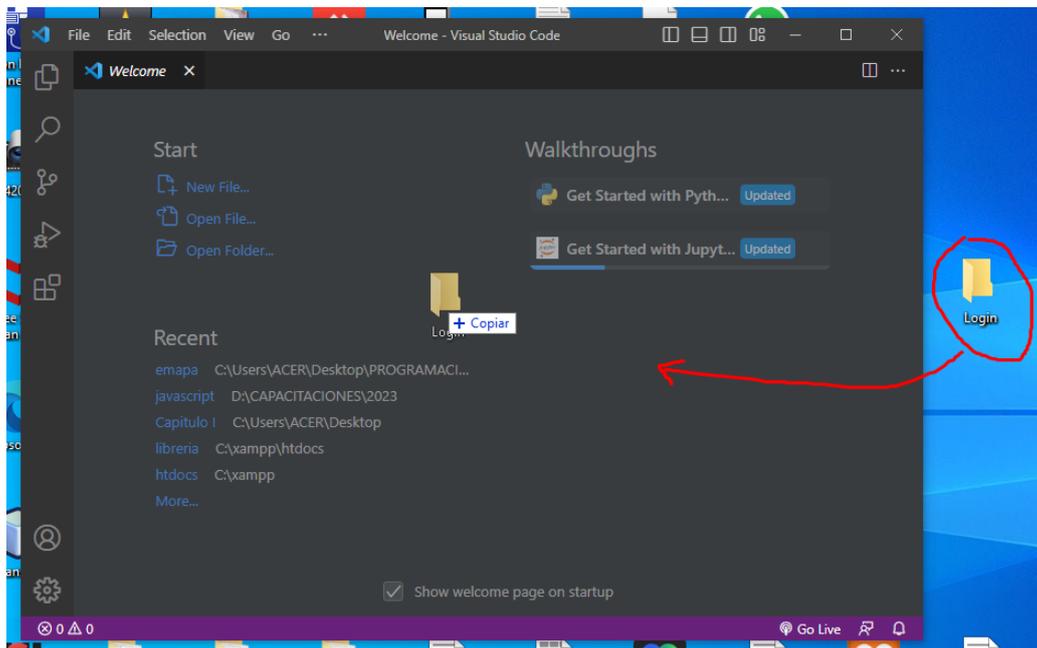


FIGURA N° 01-022: Se crea la carpeta en el escritorio y arrastrar a la ventana del Visual Studio Code

En la parte izquierda aparece la carpeta Login, dentro de ello crear los archivos: **login.html** y **style.css**, para ello hacer click en el icono de **New file...**, tal como se muestra en la FIGURA N° 01-023:

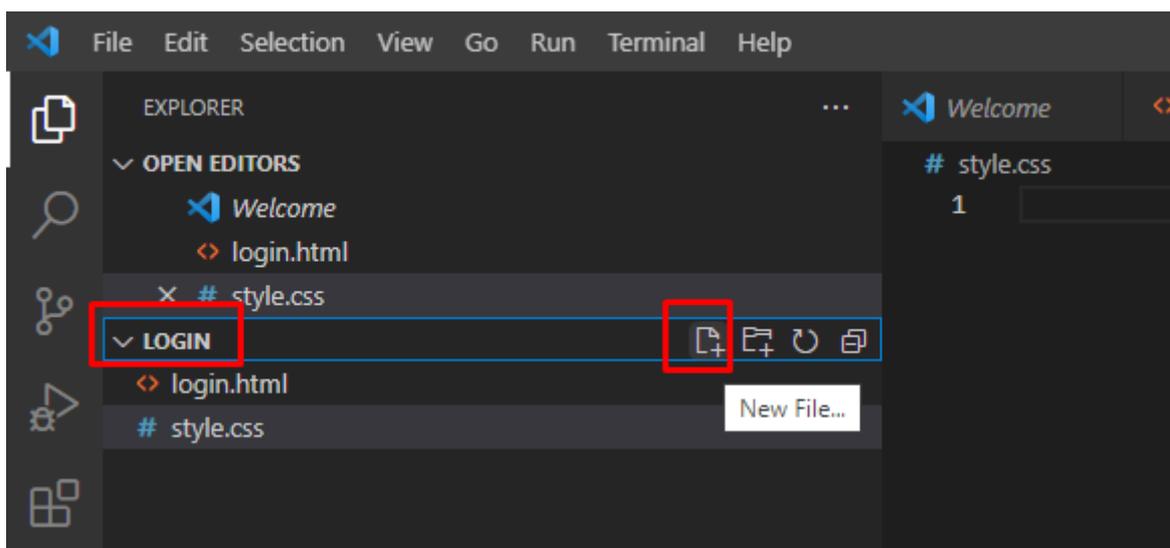


FIGURA N° 01-023: Dentro de la carpeta se crea los archivos login.html y style.css

Creamos la estructura básica de una página en **login.html**, para ello el Visual Studio Code nos ayuda escribiendo en el archivo “html” y autocompleta desplegando un menú tal como se muestra en la FIGURA N° 01-024

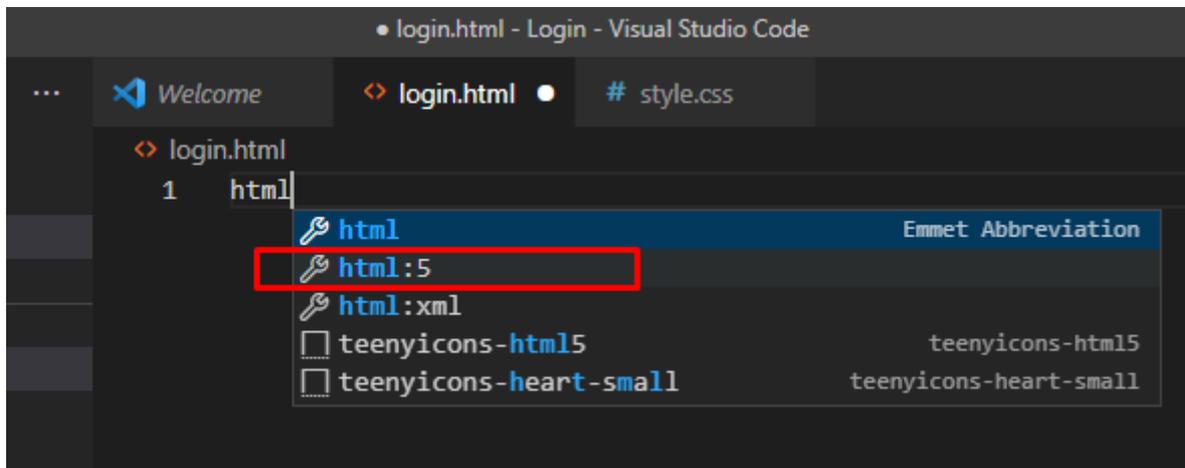


FIGURA N° 01-024: Al escribir html dentro del archivo, Visual Studio Code muestra un menú desplegable lo cual se selecciona html:5

Una vez seleccionado html:5, automáticamente se crea un diseño básico de una página HTML, tal como se ve en la FIGURA N° 01-025:

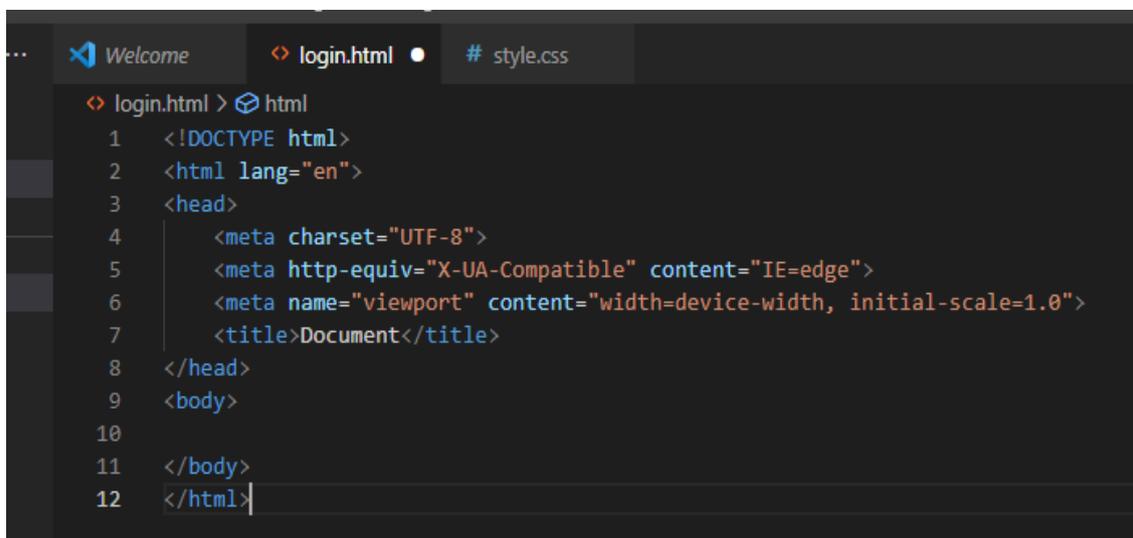


FIGURA N° 01-025: Código HTML básico de una página creado con automáticamente con Visual Studio Code

Para crear el diseño de la página mostrado en la FIGURA N° 01-021, se agregará las etiquetas HTML que se requieran dentro del cuerpo <body>, en este caso:

- ✓ Un formulario <form></form>
- ✓ Una etiqueta para poner el título <h1></h1>
- ✓ Una etiqueta de tipo texto <input type="text" name="" id="">
- ✓ Una etiqueta de tipo password <input type="password" name="" id="">
- ✓ Un button <button type="submit"></button>

A cada uno de ellos cambiaremos y agregaremos propiedades, atributos, como id, name entre otros, así mismo agregamos el link del archivo que contiene al CSS.

```
<link rel="stylesheet" href="style.css">
```

Quedando tal como se muestra en la FIGURA N° 01-026.

```
login.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="style.css">
8   <title>Login</title>
9 </head>
10 <body>
11   <form id="form-login" action="#">
12     <h1 class="form-titulo">Acceso al Sistema</h1>
13     <input type="text" name="txtUsuario" id="txtUsuario" placeholder="Usuario">
14     <input type="password" name="txtPassword" id="txtPassword" placeholder="Password">
15     <button type="submit">LOGIN</button>
16   </form>
17 </body>
18 </html>
```

FIGURA N° 01-026: Código HTML de la estructura de la página

Para visualizar el diseño lo abrimos en un explorador web, en este caso en Google Chrome y se visualiza, así como se muestra en la FIGURA N° 01-027.



FIGURA N° 01-027: Diseño HTML visualizado en un explorador web

Ya se tiene los controles del formulario, pero no se visualiza como el diseño de la FIGURA N° 01-021, para ello en el archivo style.css daremos estilos a las etiquetas y controles del formulario de la página.

Maquetación de la página con CSS, en el archivo **style.css**.

```
/*-----*/
/*Estilos a la pagina*/
/*-----*/

/*estilo por etiqueta HTML*/
html, body{
    height: 100%; /*el body y el html ocupan el alto al 100% */
}

/*estilo por etiqueta*/
body{
    margin: 0; /*los espacios fuera del contenedor desaparecen*/
    padding:0; /*los espacios dentro del contenedor desaparecen*/
    background-color: blueviolet; /*color de fondo del body*/

    /*-----*/
    /*centrar vertical y horizontalmente*/
    display:flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: center;
    align-items: center;
    /*-----*/
}

/*estilo por id*/
#form-login{

    width: 300px; /*el ancho del formulario a 300px*/
    height: 400px; /*el alto del formulario a 400px*/
    background-color: blue; /*color del formulario a azul*/
}

/*estilo por clase*/
.form-titulo{
    color:#000; /*color del titulo de formulario negro*/
    text-transform: uppercase; /*El texto transforma a mayuscula*/
    font-weight: 500; /*el peso de la fuente(o que tan negrita) */
    font-size: 30px; /*tamaño de la fuente*/
}
```

```

/*estilos por tipo de cada control dentro del form-login*/
#form-login input[type="text"],
#form-login input[type="password"],
#form-login button[type="submit"]{

    border:0; /*sin borde los controles*/
    background: none; /*sin color de fondo*/
    display: block; /*un control encima del otro*/
    margin: 20px auto; /*separados cada uno por 20 pixeles y */
                /* automatico asi a los lados*/
    padding: 14px 10px; /*separado 14 px y 10px a si a los lados */
                /* entre controles*/
    text-align: center; /*texto del control alienado al centro*/
    border: 2px solid #fff; /*borde de grosor 2px estilo solido*/
                /*de color blanco*/
    width: 200px; /*ancho fijo de 200px*/
    outline: none; /*no se establece ningún perfil */
    color:#fff; /*color del texto*/
    border-radius: 24px; /*borde redondeado*/
    transition: 0.25s; /*se pone una transición de 0.25 segundos */

}

/*estilo a una propiedad determinada, en este caso al focus*/
#form-login input[type="text"]:focus,
#form-login input[type="password"]:focus{
    width: 270px; /*cambia de tamaño cuando recibe el enfoque*/
    border-color: black; /*color el borde cuando recibe el enfoque */
}

/*estilo al control button del formulario*/
#form-login button[type="submit"] {
    border:0; /*sin borde*/
    background: black; /*color de fondo negro*/
    cursor: pointer; /*icono del cursor una manito*/
}

/*estilo a un evento en este caso: hover (cuando se mueve el mouse)*/
#form-login button[type="submit"]: hover{
    background:#fff; /*se cambia de color de fondo a blanco*/
    color: #000; /*se cambia el color del texto a negro*/
}

```

Para visualizar el resultado final del diseño, click derecho en el archivo dentro del Visual Studio Code y seleccionar Open with Live Server tal como se muestra en la FIGURA N° 01-028.

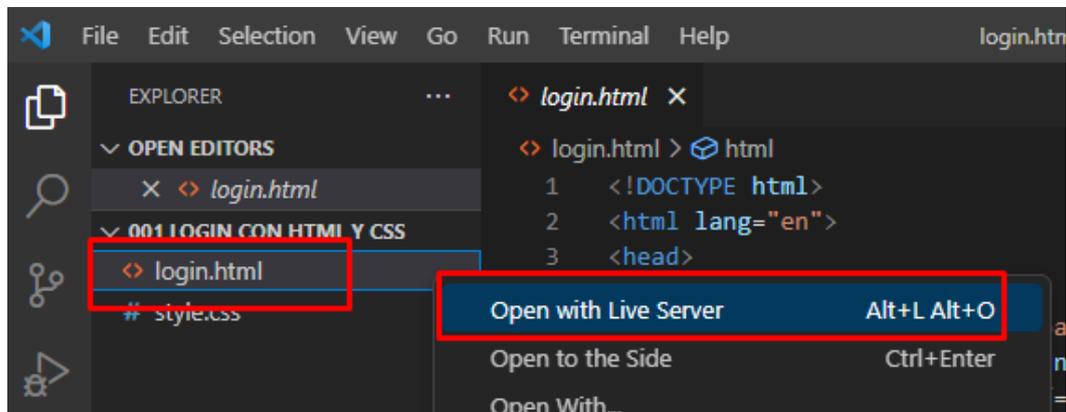


FIGURA N° 01-028: Click derecho en el archivo para visualizar el resultado final

El resultado final se mostrará como en las FIGURAS N° 01-029, 01-030, 01-031, 01-032. con sus diferentes efectos



FIGURA N° 01-029: Diseño final



FIGURA N° 01-030: Al recibir el enfoque el primer text aumenta su tamaño y cambia de color de contorno



FIGURA N° 01-031: Al recibir el enfoque el segundo text aumenta su tamaño y cambia de color de contorno



FIGURA N° 01-032: Al pasar el mouse por encima del botón cambia de color a blanco y de puntero del mouse

1.5. Diseño de una página

Realizaremos el diseño de una página tal como se muestra en la FIGURA N° 01-033, utilizando los conceptos de HTML, CSS



FIGURA N° 01-033: Diseño de página web estática

DESARROLLO

1. Crear el bosquejo de la página con HTML, con todas las etiquetas necesarias, dando nombre de clase e id:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined
:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200" />

    <title>EMAPA SAN LUIS S.A</title>

  </head>

  <body>
    <div id="container">
      <header >
        <div id="titulo_cabecera">
          
          <h1>Emapa San Luis S.A.</h1>
          <h2>El Agua que tomas es el fruto de tu
esfuerzo</h2>
        </div>

        <nav>
          <ul>
            <li><a href="#">Inicio</a></li>
            <li><a href="#">Quienes Somos</a></li>
            <li><a href="#">Institucional</a></li>
            <li><a href="#">Directorio</a></li>
          </ul>
        </nav>
      </header>

      <div id="portada">
        <div id="portada_texto">
          El agua que consume los 5 sectores de San Luis viene
desde la Laguna Verde Cocha
          <a href="#" class="galeria_fotos">Galeria Fotos
&nbsp;
            
          </a>
        </div>
      </div>
    </div>
  </body>
</html>

```

```

</div>

<section>
  <article>

    <h1>
      <span class="material-symbols-
outlined">event_note</span>
      Historia de Emapa San Luis S. A.
    </h1>

    <p>EMAPA SAN LUIS S.A, fue creada el 14 de Enero de
1994 como empresa prestadora de servicio de agua potable y
alcantarillado, con personería jurídica de derecho Privado inscrito en
el Registro de personas Jurídicas. Se crea por la indiferencia, la
inoperancia y falta de capacidad de gestión de las instituciones
responsables de la prestación de los servicios de agua potable y
alcantarillado, al naciente Pueblo de San Luis.</p>

    <p>Ante la inminente carencia de la prestación del
servicio de agua potable y alcantarillado, los pobladores de San Luis
se organizan en un comité central de agua y desagüe, bajo la presidencia
del ciudadano. Cesar Modesto Cornejo, en busca de la prestación del
servicio no ven mejor alternativa de autofinanciar sus obras, donde
aparece la mano milagrosa de un Sacerdote, más conocido como: PADRE
PACO, párroco de la iglesia Santa María de Fátima de Paucarbamba,
nuestros esfuerzos y sacrificios hoy se plasma en una Empresa pujante
con visión de ser una Empresa competitiva con las demás que existen en
nuestro departamento brindándoles agua de alta calidad con tarifas
sociales, sin fines de lucro, lo manifiesta su Gerente General : Luis
E. Lavado Mallqui</p>

    <p> La Empresa Administradora de Servicios de
Agua Potable y Alcantarillado de San Luis, cuya sigla es (EMAPA SAN
LUIS S.A.), cuenta con persona jurídica de derecho privado, constituida
por ante los registros públicos de Huánuco, inscrita en la Partida
Electrónica 11002086 del Registro de Personas Jurídicas – LIBRO ULTIMO
DE SOCIEDADES.</p>

  </article>
  <aside>
    <h1>PERSONAL OPERATIVO</h1>
    

```

```

        <p id="personal_galeria"></p>

        <p>Representantes Gestion 2015-2016</p>

        <p>PRESIDENTE      : Ing.  Elmer Santiago
Chuquiyauri Saldívar.</p>
        <p>DIRECTORES.</p>
        <p>•   Sr. Antonio Orozco Martínez (sector 1 San
Luis)</p>
        <p>•   Sr. Genaro Carbajal Leandro (sector 2 San
Luis)</p>
        <p>•   Prof. Milton Cesar Pacheco Tolentino (
sector 3 San Luis)</p>
        <p>•   Sr. Juan López Espinoza (sector 4 San
Luis)</p>
        <p>•   Prof. Mesías Ureta Chávez (sector 5 San
Luis).</p>

        <p>
        
        
        
        
        

        </p>
    </aside>
</section>

<footer>
    <div id="direccion">
        <h1>DIRECCION Y TELEFONO</h1>
        <p align="center">Av. Jose Carlos Mariategui Mz. V-
Lt 04</p>

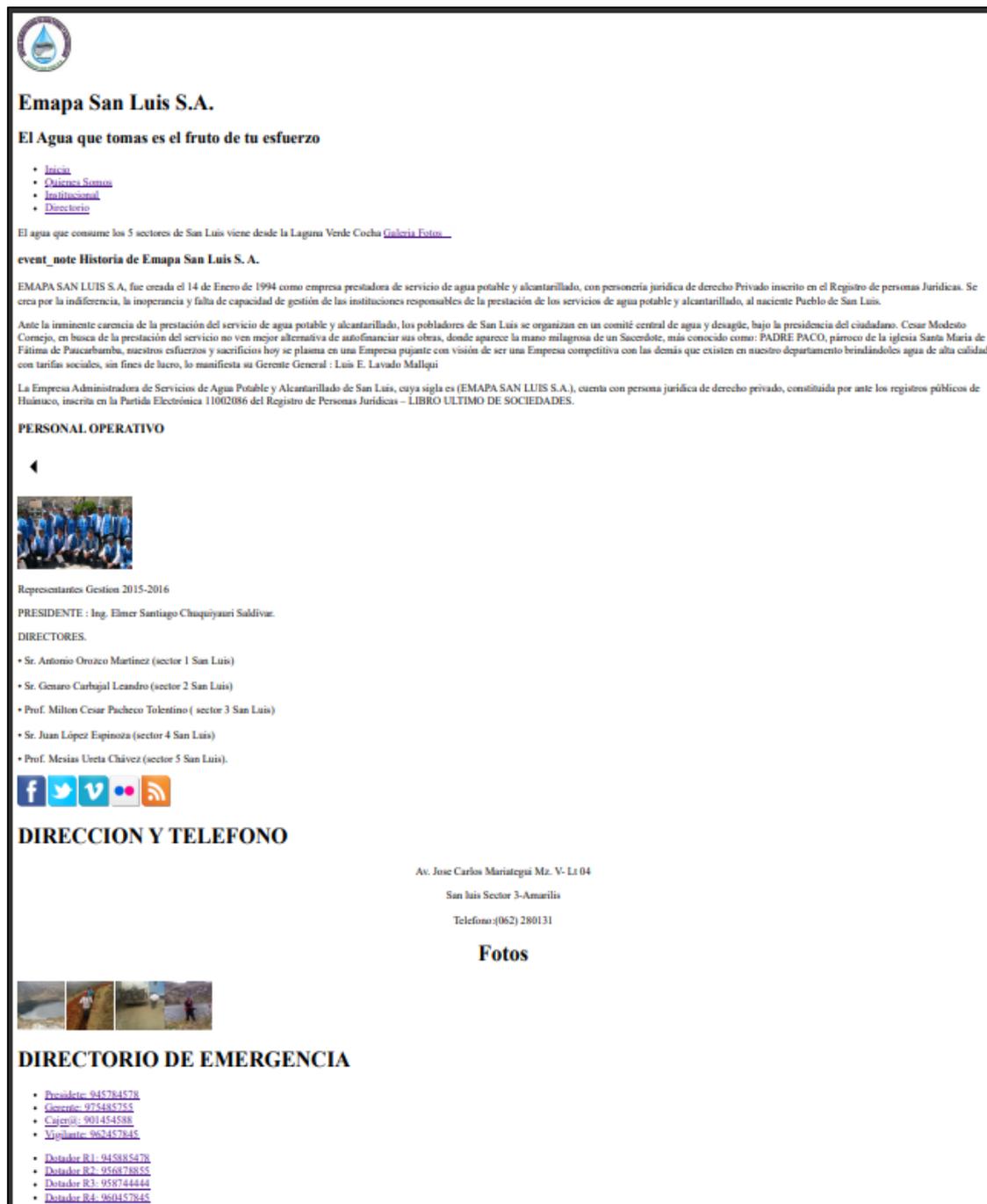
        <p align="center">San luis Sector 3-Amarilis</p>
        <p align="center">Telefono:(062) 280131</p>

    </div>
    <div id="galeria_footer">
        <h1 align="center">Fotos </h1>
        <p>
            
            
            
            
        </p>
    </div>

```

```
</div>
<div id="directorio_emergencia">
  <h1>DIRECTORIO DE EMERGENCIA</h1>
  <ul>
    <li><a href="#">Presidete: 945784578</a></li>
    <li><a href="#">Gerente: 975485755</a></li>
    <li><a href="#">Cajer@: 901454588</a></li>
    <li><a href="#">Vigilante: 962457845</a></li>
  </ul>
  <ul>
    <li><a href="#">Dotador R1: 945885478</a></li>
    <li><a href="#">Dotador R2: 956878855</a></li>
    <li><a href="#">Dotador R3: 958744444</a></li>
    <li><a href="#">Dotador R4: 960457845</a></li>
  </ul>
</div>
</footer>
</div>
</body>
</html>
```

Visualizando en el Google Chrome se vería, así como en la FIGURA N° 01-034



Emapa San Luis S.A.

El Agua que tomas es el fruto de tu esfuerzo

- [Inicio](#)
- [Quiénes Somos](#)
- [Institucional](#)
- [Directorio](#)

El agua que consume los 5 sectores de San Luis viene desde la Laguna Verde Cocha [Galeria Fotos](#)

event_note Historia de Emapa San Luis S. A.

EMAPA SAN LUIS S.A. fue creada el 14 de Enero de 1994 como empresa prestadora de servicio de agua potable y alcantarillado, con personería jurídica de derecho Privado inscrito en el Registro de personas Jurídicas. Se crea por la indiferencia, la inoperancia y falta de capacidad de gestión de las instituciones responsables de la prestación de los servicios de agua potable y alcantarillado, al naciente Pueblo de San Luis.

Ante la inminente carencia de la prestación del servicio de agua potable y alcantarillado, los pobladores de San Luis se organizan en un comité central de agua y desagüe, bajo la presidencia del ciudadano. Cesar Modesto Cornejo, en busca de la prestación del servicio no ven mejor alternativa de autofinanciar sus obras, donde aparece la mano milagrosa de un Sacerdote, más conocido como: PADRE PACO, párroco de la iglesia Santa Maria de Fátima de Patatebamba, nuestros esfuerzos y sacrificios hoy se plasma en una Empresa pujante con visión de ser una Empresa competitiva con las demás que existen en nuestro departamento brindándonos agua de alta calidad con tarifas sociales, sin fines de lucro, lo manifiesta su Gerente General : Luis E. Lavado Mallqui

La Empresa Administradora de Servicios de Agua Potable y Alcantarillado de San Luis, cuya sigla es (EMAPA SAN LUIS S.A.), cuenta con persona jurídica de derecho privado, constituida por ante los registros públicos de Hainuco, inscrita en la Partida Electrónica 11002086 del Registro de Personas Jurídicas – LIBRO ULTIMO DE SOCIEDADES.

PERSONAL OPERATIVO



Representantes Gestion 2015-2016

PRESIDENTE : Ing. Elmer Santiago Chuquiayari Saldívar.

DIRECTORES.

- Sr. Antonio Orozco Martínez (sector 1 San Luis)
- Sr. Genaro Carbojal Leandro (sector 2 San Luis)
- Prof. Milton Cesar Pacheco Tolentino (sector 3 San Luis)
- Sr. Juan López Espinoza (sector 4 San Luis)
- Prof. Mesías Ureta Chávez (sector 5 San Luis).



DIRECCION Y TELEFONO

Av. Jose Carlos Mariategui Mz. V- Lt 04
San Luis Sector 3-Amarillos
Telefono:(062) 280131

Fotos



DIRECTORIO DE EMERGENCIA

- [Presidente: 945784578](#)
- [Gerente: 975485755](#)
- [Cajero: 901454588](#)
- [Vigilante: 962457845](#)
- [Detector R1: 945885478](#)
- [Detector R2: 956878855](#)
- [Detector R3: 958744444](#)
- [Detector R4: 960457845](#)

FIGURA N° 01-034: Página web estática sin CSS

2. Formatearlo y hacer el diseño de la página mediante CSS.

Se crea un archivo style.css en la cual realizaremos lo siguiente:

1. Fuentes externo.
2. Definición de los estilos principales de la página (ancho de la página web, color de fondo, color de texto predeterminado).

3. Encabezado y enlaces de navegación.
4. Banner de la laguna Verde Chocha
5. Sección principal del cuerpo de la página, en el centro.
6. Pie de página.

Fuentes externas

FontSquirrel(<https://www.fontsquirrel.com/tools/webfont-generator>) permite generar fuentes para utilizarlos en nuestra página web, en este caso tenemos dos fuentes externas:

- ✓ BallparkWeiner;
- ✓ Dayrom.

```
1  @font-face
2  {
3      font-family: 'BallparkWeiner';
4      src: url('fonts/ballpark.eot');
5      src: url('fonts/ballpark.eot?#iefix') format('embedded-opentype'),
6           url('fonts/ballpark.woff') format('woff'),
7           url('fonts/ballpark.ttf') format('truetype'),
8           url('fonts/ballpark.svg#BallparkWeiner') format('svg');
9      font-weight: normal;
10     font-style: normal;
11 }
12
13 @font-face
14 {
15     font-family: 'Dayrom';
16     src: url('fonts/dayrom.eot');
17     src: url('fonts/dayrom.eot?#iefix') format('embedded-opentype'),
18         url('fonts/dayrom.woff') format('woff'),
19         url('fonts/dayrom.ttf') format('truetype'),
20         url('fonts/dayrom.svg#Dayrom') format('svg');
21     font-weight: normal;
22     font-style: normal;
23 }
24
```

FIGURA N° 01-035: Definición de las Fuentes externas

Definir los estilos principales

Se define los estilos generales para todo el diseño de la página, tendrá una imagen de fondo, la fuente y el color del texto, establecer el tamaño de la página centrado en la pantalla, color de fondo degradado, etc. tal como se muestra en las FIGURAS del N° 01-036 AL N° 01-041

```

27  body{
28      background: url('images/fondo_body.png');
29      background-color: #1619aa;
30      font-family: 'Trebuchet MS', Arial, sans-serif;
31      color: hwb(120 1% 95%);
32      background: linear-gradient(135deg, #9b9ba7 23%, hwb(156 59% 7%) 100%);
33  }
34  }
35
36  #container{
37      width: 73%;
38      margin: auto;
39  }
40
41  section h1, footer h1, nav a{
42      font-family: Dayrom, serif;
43      font-weight: normal;
44      text-transform: uppercase;
45  }
46
47
48  header{
49      background: url('SWG/remove_FILL0_wght400_GRAD0_opsz48.svg') repeat-x bottom;
50      width: 88%;
51      padding: 10px;
52  }
53
54  #titulo_cabecera{
55      display: inline-block;
56      width: 100%;
57  }
58
59  header h1{
60      font-family: 'BallparkWeiner', serif;
61      font-size: 2.5em;
62      font-weight: normal;
63  }
64
65  #logo, header h1{
66      display: inline-block;
67      margin-bottom: 0px;
68  }
69
70  #logo{
71      border-radius: 50%;
72  }
73
74  header h2{
75      font-family: Dayrom, serif;
76      font-size: 1.1em;
77      margin-top: 0px;
78      font-weight: normal;
79  }

```

FIGURA N° 01-036: Estilos

```
83  nav{
84      display: inline-block;
85      width: 100%;
86      text-align:lefts;
87  }
88
89  nav ul{
90      list-style-type: none;
91  }
92
93  nav li{
94      display: inline-block;
95      margin-right: 15px;
96  }
97
98  nav a{
99      font-size: 1.3em;
100     color: #181818;
101     padding-bottom: 3px;
102     text-decoration: none;
103 }
104
105 nav a:hover{
106     color: #760001;
107     border-bottom: 3px solid #760001;
108 }
109
110
111 #portada{
112     margin-top: 15px;
113     height: 200px;
114     border-radius: 5px;
115     background: url('images/verdecocha.jpg') no-repeat;
116     position: relative;
117     box-shadow: 0px 4px 4px hwb(212 6% 15% / 0.37);
118     margin-bottom: 25px;
119     width:90%;
120 }
121
122
123
124 #portada_texto{
125     position: absolute;
126     bottom: 0;
127     border-radius: 0px 0px 5px 5px;
128     width: 99.5%;
129     height: 33px;
130     padding-top: 15px;
131     padding-left: 4px;
132     background-color: hwb(212 6% 15% / 0.37);
133     color: white;
134     font-size: 0.8em;
135 }
136 }
```

FIGURA Nº 01-037: Estilos

```
138 .galeria_fotos{
139     display: inline-block;
140     height: 25px;
141     position: absolute;
142     right: 5px;
143     bottom: 5px;
144     background: url('images/galeria_fotos.png') repeat-x;
145     border: 1px solid #760001;
146     border-radius: 5px;
147     font-size: 1.2em;
148     text-align: center;
149     padding: 3px 8px 0px 8px;
150     color: white;
151     text-decoration: none;
152 }
153
154 .galeria_fotos img{
155     border: 0;
156 }
157
158 article, aside{
159     display: inline-block;
160     vertical-align: top;
161     text-align: justify;
162 }
163
164 article{
165     width: 61%;
166     margin-right: 15px;
167 }
168
169 .icono_titulo{
170     vertical-align: middle;
171     margin-right: 8px;
172 }
173
174 article p{
175     font-size: 1em;
176 }
177
178
179 article h1{
180     font-size: 0.9em;
181     color: blue;
182     font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
183 }
184
185
```

FIGURA N° 01-038: Estilos

```

186 aside{
187     position: relative;
188     width: 25%;
189     background-color: hwb(212 6% 15% / 0.37);
190     box-shadow: 0px 2px 5px #1c1a19;
191     border-radius: 5px;
192     padding: 10px;
193     color: white;
194     font-size: 0.9em;
195 }
196
197 #flecha_isquierda{
198     position: absolute;
199     top: 100px;
200     left: -28px;
201 }
202
203 #personal_galeria{
204     text-align: center;
205 }
206 }
207
208 #personal_galeria img{
209     border: 1px solid hsl(157, 73%, 43%);
210     border-radius: 50%;
211 }
212
213 aside img{
214     margin-right: 5px;
215 }
216
217 footer{
218     background: url('SWG/remove_FILL0_wght400_GRAD0_opsz48.svg') repeat-x top,
219                 url('SWG/keyboard_double_arrow_up_FILL0_wght400_GRAD0_opsz48.svg') no-repeat top center,
220                 url('SWG/remove_FILL0_wght400_GRAD0_opsz48.svg') repeat-x top;
221     padding-top: 25px;
222     width: 90%;
223 }
224
225
226
227
228 footer p, footer ul{
229     font-size: 0.8em;
230 }
231
232 footer h1{
233     font-size: 1.1em;
234 }
235
236 #direccion, #galeria_footer, #directorio_emergencia{
237     display: inline-block;
238     vertical-align: top;
239 }
240
241 #direccion{
242     width: 30%;
243 }
244
245 #galeria_footer{
246     width: 35%;
247 }

```

FIGURA N° 01-039: Estilos

```

249 #directorio_emergencia{
250     width: 32%;
251 }
252
253 #galeria_footer img{
254     border: 1px solid hwb(251 6% 14%);
255     margin-right: 2px;
256     border-radius: 30%;
257 }
258
259 #directorio_emergencia ul{
260     display: inline-block;
261     vertical-align: top;
262     margin-top: 0;
263     width: 48%;
264     /* list-style-image: url('SWG/arrow_right_FILL0_wght400_GRAD0_opsz48.svg');*/
265
266     padding-left: 2px;
267 }
268
269 #directorio_emergencia a{
270     text-decoration: none;
271     color: #760001;
272 }
273
274 .container #titulo_cabecera,
275 .container #logo,
276 .container header h1,
277 .container nav,
278 .container nav li,
279 .container .galeria_fotos,
280 .container article,
281 .container aside,
282 .container #flecha_isquierda,
283 .container #galeria_footer,
284 .container #directorio_emergencia,
285 .container #directorio_emergencia ul
286 {
287     display: inline;
288     zoom: 1;
289 }
290
291
292
293 #container section h1{
294     font-size: 1.1em;
295 }
296
297 #container footer div{
298     margin-top: 5px;
299     padding: -4px;
300 }
301

```

FIGURA N° 01-040: Estilos

```
303
304 /*-----*/
305 /*iconos de google*/
306 /*-----*/
307
308 .material-symbols-outlined {
309     color: ■ chartreuse;
310     font-variation-settings:
311     'FILL' 0,
312     'wght' 400,
313     'GRAD' 0,
314     'opsz' 48
315 }
316
317
```

FIGURA N° 01-041: Estilos

CAPÍTULO II

Fundamentos de JavaScript



2.1. JavaScript

JavaScript es un lenguaje de programación de scripts de propósito general, fue creado por la empresa Netscape.2.1

2.1.1. Variables, constantes, operadores, parseo

Variables

En JavaScript las variables se declaran con var, let y dependiendo del tipo de valor que se asigne asumirá como: numérico, string, boolean, etc, luego de cada sentencia no es necesario el punto y coma “;” pero es recomendable, para consignar algún comentario en nuestro código se utiliza “//” en comentarios de una sola línea y de varias líneas con “/*comentario*/”

```
/* Se declaran variables de acuerdo a lo que
se requiere en nuestro código */

//variable a que se le asigna un valor entero
var a=12;

//variable nombre que se le asigna el valor de Jose
var nombre="Jose";

// variable peso que se le asigna un valor real
var peso=58.4;

// declaracion de variables utilizando let
let x=14;
let y=20;
```

La diferencia de declarar variables con **var** y **let** está en el alcance, variables declarado con **let** solo tiene alcance en el bloque en donde se declara, las variables declarado con **var** tienen mayor alcance local o global, en el siguiente ejemplo la variable *i* se declaró con **let** y tendrá alcance solo dentro del **if**.

```
//se declara la variable i con let
//que solo tiene alcance dentro del if
if(x1<y1){
  let i=1;
}
```

Constantes

Las variables o expresiones que asumirán un solo valor se declaran con *const*

```
//valores que seran representados como constante
const pi=3.1416;
const gravedad=9.8;
```

Algunos operadores:

Operadores de asignación

El operador “=” permite asignar valores, se podría tener variantes se asignación: Asignacion de adicon(x+=y) asignación de resta(x-=y), asignación de multiplicacio(x*=y) asignación de división(x/=y), asignación de residuo(x%=y), entre otras variantes.

Operadores de comparación

Los operadores de comparación son aquellos que permite generar una proposición lógica de verdadero o falso: Igual(==), no es igual(!=), estrictamente igual(===) hasta el tipo de dato de los operandos, desigualdad estricta (!==) hasta el mismo tipo de datos de los operandos, mayor que(>), mayor o igual que(>=), menor que(<), menor o igual que(<=)

Operadores aritméticos

Los operadores aritméticos más utilizados: suma(+), resta(-), multiplicación(*), división(/), residuo(%), incremento(++), decremento(--).

Operadores lógicos

Los operadores lógicos permiten generar **proposiciones** combinadas a partir de proposiciones simples: and(&&), or(||), NOT(!)

Operador condicional ternario

El operador condicional ternario permite evaluar una proposición, dependiendo del valor de verdad realizara acciones, sintaxis:

Proposición Accion1: accion2

Si la proposición es verdadera realizara la accion1 y si es falso realizara la accion2

```
var x=30;
var y=15;

x%y==0?console.log(" x es divisible por y"):console.log("x
no es no divisible por y");
```

De acuerdo a los valores de x e y , imprime “ x es divisible por y ”

Parseo

Cuando se tiene variables que están asignado a un determinado tipo de datos (entero, string, float, etc.) y se requiera realizar alguna operación, para ello los operadores tienen que ser del mismo tipo, por lo que necesito hacer un cambio de tipo de dato(parseo), para ello se utiliza funciones predeterminado del JavaScript: ***.parseInt(string)***, ***.parseFloat(string)***, ***.toString()***

2.2. Arreglos y objetos

Arreglos

Los arreglos en JavaScript son estructuras que permite almacenar datos u otras estructuras y administrarlo atreves de su índice (posición en el arreglo que inicia desde o), para declararlos se utiliza los corchetes []. A diferencia de otros lenguajes los arreglos en JavaScript permiten almacenar datos de diferentes tipos, algunos ejemplos que se muestra a continuación:

```
const numeros=[];//arreglo vacio
const vocales=["a","e","i","o","u"];
const dato=["a",14,true];//diferentes tipos de datos
```

Para acceder a cualquier elemento de un array se utiliza su índice, por ejemplo del array vocales si se quiere acceder a la vocal “i” seria: **vocales[2]**, así mismo para obtener el tamaño del arreglo se utiliza su propiedad **.length**.

Más adelante veremos formas de recorrer todos los elementos del arreglo con bucles y otras formas. Algunas operaciones con los elementos de un arreglo:

- ✓ Cuando se quiere agregar elementos utilizamos **array.push(e1,e2,...)**
- ✓ Cuando se quiere eliminar el último elemento utilizamos **array.pop()**
- ✓ Para añadir una o varios elementos al inicio del **array.unshift(e1,e2,...)**
- ✓ Para eliminar el primer elemento del **array.shift()**

Prueba de estos métodos se puede ver a continuación, donde se crea un arreglo que guarda nombre de países:

```
//se declara arreglo de paices
const paises=["Argentina","Bolivia","Brasil","Colombia"];
paises.push("Cuba","Chile","Ecuador");//se agrega dos paises
al arreglo
console.log(paises);//se imprime en consola para ver
paises.pop();//se quita el ultimo pais del arreglo
console.log(paises);//se imprime en consola para ver
paises.unshift("México","Perú");//se agrega dos paises al
inicio
console.log(paises);//se imprime en consola para ver
paises.shift();//se quita el primer pais del arreglo
console.log(paises);//se imprime en consola para ver
```

En la FIGURA N° 02-001 se puede visualizar el resultado de la impresión en consola de los arreglos resultantes después de realizar cada operación.



FIGURA N° 02-001: Resultado de la impresión en consola

Para ordenar los elementos de un arreglo se utiliza el método **array.sort()**,

Por ejemplo se declara un arreglo de números y al aplicar el método **sort()** se ordena tal como se muestra en la FIGURA N° 02-002.

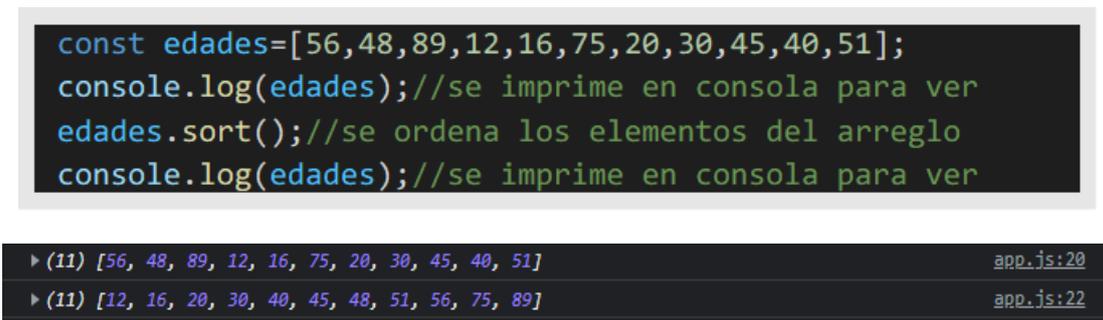


FIGURA N° 02-002: Resultado de aplicar el método **sort()** a un arreglo

Hay otros métodos para hacer operaciones con el arreglo por ejemplo **concat()** que permite unir dos arreglos, métodos de búsqueda:

- ✓ *filter* (filtro)
- ✓ *find* (filtro)
- ✓ *findIndex()*
- ✓ *includes* (valor)
- ✓ *indexOf* (índice de)

Array.filter(filtro)

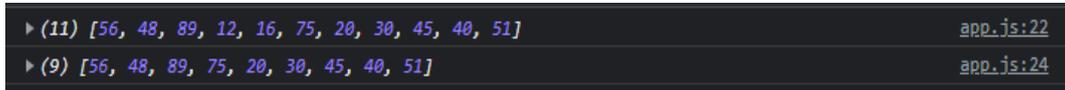
Este método `Array.filter(filtro)` permite encontrar elementos dentro de un arreglo que cumplan con cierta condición. Por ejemplo, si se tiene un arreglo donde se guardan edades y que quiere obtener los mayores o iguales a 18 años, hacemos lo siguiente:

```
const edades=[56,48,89,12,16,75,20,30,45,40,51];
console.log(edades);//se imprime en consola para ver

// se filtra todos los elementos ayores o iguales a 18
const mayorEdad=edades.filter(element=>element>=18);

console.log(mayorEdad);//se imprime en consola para ver
```

El resultado se vería como en la FIGURA N° 02-003.



```
▶ (11) [56, 48, 89, 12, 16, 75, 20, 30, 45, 40, 51] app.js:22
▶ (9) [56, 48, 89, 75, 20, 30, 45, 40, 51] app.js:24
```

FIGURA N° 02-003: Resultado de aplicar el método **filter()** a un arreglo

Array.find(filtro)

Este método `Array.find(filtro)` permite encontrar el primer elemento que cumple cierta condición en el filtro, en el siguiente ejemplo se tiene un arreglo de países, se realiza la búsqueda del país “*Brasil*” y “*Peru*”, el primero cuando encuentra imprimirá “*Brasil*”, el segundo como no hay en el arreglo imprimirá “*undefined*”

```
const paises=["Argentina","Bolivia","Brasil","Colombia"];

const pais1=paises.find(element=>element=="Brasil");
const pais2=paises.find(element=>element=="Peru");
console.log(pais1); //imprime brasil
console.log(pais2); //imprime undefined
```

Array.findIndex(filtro)

Este método `Array.findIndex(filtro)` permite encontrar el **índice** del primer elemento que cumple cierta condición en el filtro, en el siguiente ejemplo se tiene un arreglo de países, se realiza la búsqueda del país “*Brasil*” y “*Peru*”, el primero cuando encuentra imprimirá “*2*”, el segundo como no hay en el arreglo imprimirá “*-1*”

```
const paises=["Argentina","Bolivia","Brasil","Colombia"];

const pais1=paises.findIndex(element=>element=="Brasil");
const
pais2=paises.findIndex(element=>element=="Peru");
console.log(pais1); //imprime 2
console.log(pais2); //imprime -1 por queno esta "Peru"
```

Array.includes (valor)

Este método busca si en el arreglo incluye un elemento, devuelve **true** si está incluido el elemento de lo contrario devuelve **false**. En el siguiente ejemplo en el arreglo se tiene países y se busca "Colombia" con **.includes("Colombia")**, el cual nos retornara **true** y de acuerdo al código se imprime en consola **true**

```
const paises=["Argentina","Brasil","Colombia","Costa Rica"];
const hayPais=paises.includes("Colombia");
console.log(hayPais);//imprime true, por que Colombia estan en
el Array
```

Array.indexOf (valor)

Este método es parecido al **includes()**, pero retorna el índice del primer elemento que coincide con el valor, si no encuentra retorna -1. El método **.lastIndexOf(valor)** es similar, pero retorna el índice contando desde la última posición. En el siguiente ejemplo de arreglo de números, **numeros.indexOf(12)** retorna "3" que pertenece al índice del 12 en el arreglo.

```
const numeros = [16,27,3,12,86,12,47,16,95,4,32];
const indexElemento=numeros.indexOf(12);
console.log(indexElemento);
```

Modificar o crear sub arrays

Para poder modificar el contenido o crear nuevos sub arreglos se pueden utilizar los siguientes métodos:

- ✓ **.slice(idexInicio, indexFinal)**
- ✓ **.splice(idexInicio, size)**
- ✓ **.splice(idexInicio, size, e1, e2...)**
- ✓ **.copyWithin(pos, idexInicio, indexFinal)**
- ✓ **.fill(element, idexInicio, indexFinal)**

Array.slice(indexInicio, indexFinal)

Este método permite crear un arreglo con elementos desde la posición **indexInicio** hasta **indexFinal-1**. Si no se proporciona el parámetro **indexFinal** asumirá el último índice del arreglo. **Este método no genera ningún cambio al arreglo original**. En el siguiente arreglo de números al aplicar el método **numeros.slice(1,6)**, genera un sub arreglo desde el índice 1 hasta 6: [27,3,12,86,12,47].

```
const numeros = [16,27,3,12,86,12,47,16,95,4,32];
console.log(numeros); // se visualiza array completo
const num1=numeros.slice(1,6); // desde la posición 1
console.log(num1); // se visualiza [27,3,12,86,12]
console.log(numeros); // se visualiza array completo
```

Array.splice(indexInicio, size)

Este método permite crear un arreglo con elementos desde la posición **indexInicio** hasta la posición **indexInicio+size-1**. El Arreglo original ya no tendrá los elementos del nuevo. En el arreglo anterior al aplicar el método **numeros.splice(3,5)**, tal como se muestra a continuación se verá los resultado como en la FIGURA N° 02-004.

```
const numeros = [16,27,46,12,86,12,47,16,95,4,32];
console.log(numeros);
const num1=numeros.splice(3,5);// desde la posicion 3
console.log(num1);// se visualiza [12,86,12,47,16]
console.log(numeros);// se visualiza [16,27,46,95,4,32]
```

▶ (11) [16, 27, 46, 12, 86, 12, 47, 16, 95, 4, 32]	app.js:22
▶ (5) [12, 86, 12, 47, 16]	app.js:24
▶ (6) [16, 27, 46, 95, 4, 32]	app.js:25

FIGURA N° 02-004: Resultado de aplicar el método splice() a un arreglo

A partir de tres parámetros a más indica que son nuevos elementos y se agregara en la posición index Inicio, al realizar la operación con este método splice (indexInicio, size, **e1, e2...**)

```
const numeros = [16,27,46,12,86,12,47,16,95,4,32];
console.log(numeros);
const num1=numeros.splice(3,5,200,600,500,800);
console.log(num1);// se visualiza [12,86,12,47,16]
console.log(numeros);//[16,27,46,,200,600,500,800,95,4,32]
```

Array.copyWithIn(pos, indexInicial, indexFinal)

Este método varío el arreglo original en contenido, pero con el mismo tamaño, copia a la posición **pos** para adelante, desde el **indexInicial** al **indexFinal-1** y completa desde el último elemento copiado hasta el final del arreglo. En el siguiente ejemplo se tiene un arreglo de números, al utilizar el método `numero.copyWithIn(2,3,7)`; estamos indicando que copie a la posición 2 para adelante los números desde la posición 3 hasta la posición 6, completando el arreglo desde el último elemento copiado hasta el final.

[16,12,32,18,49,65,65,100,78,41]

```
const numero=[16,12,45,32,18,49,65,100,78,41]
console.log(numero);//[16,12,45,32,18,49,65,100,78,41]
numero.copyWithIn(2,3,7);//se modifica el arreglo
console.log(numero);//[16,12,32,18,49,65,65,100,78,41]
```

Array.fill(element, idexInicio, indexFinal)

Este método permite rellenar **element** desde a posición **indexInicio** a **indexFinal-1**, modificando así el contenido del arreglo original. En el siguiente ejemplo se tiene un arreglo de números, al aplicar `numero.fill(0,2,7)`, se está indicando que ponga el valor 0 desde la posición 2 hasta la posición 6.

```
const numero=[16,12,45,32,18,49,65,78,41]
console.log(numero);//[16,12,45,32,18,49,65,100,78,41]
numero.fill(0,2,7);//se modifica el arreglo
console.log(numero);//[16, 12, 0, 0, 0, 0,0,78, 41]
```

Se podría instanciar un arreglo con valores por defecto, para ello aplicando `fill()`. En el siguiente ejemplo se instancia un Array con 10 posiciones y con valor por defecto `false` y se le asigna a la constante **prendido**.

```
const prendido=new Array(10).fill(false,0,10);
console.log(prendido);
//[false, false, false, false, false, false, false, false, false, false]
```

Array.map()

Este método permite generar otro arreglo con la misma cantidad de elementos a partir del contenido del arreglo sin que se altere este último.

En el siguiente ejemplo tenemos un arreglo de promedios, con el **.map()** se recorre cada elemento y se genera un nuevo arreglo con “A” o “D” dependiendo si es >10 o <=10

```
//se declara un arreglo con 10 promedios ponderado
const promPonderado=[14,15,12,10,18,8,9,11,10,15];
console.log(promPonderado); // imprime el arreglo
//dependiendo si tiene promedio >10 pondra "A"
// o "D" en el nuevo arreglo
const condicion=
promPonderado.map(element=>element>10?"A":"D");
console.log(condicion); //imprime el nuevo arreglo
console.log(promPonderado); //imprime el arreglo anterior
```

Objetos

Los objetos son estructura de datos que permite almacenar diferentes atributos de un elemento o representación del mundo real, por ejemplo: televisor (LG 15', negro, Full HD,...), todo el contenido se encierra entre llaves({}), cada atributo se representa por: **nombre:valor** y separado por “,”. Para acceder a sus atributos se utiliza los corchetes Objeto[“**nombre_atributo**”] o el punto(.) seguido del **nombre_atributo(Objeto.nombre_atributo)**.

En el siguiente ejemplo se declara el objeto alumno con sus atributos:

```
const alumno={
  codigo:"02023000210",
  nombre:"JUAN",
  apellido:"MIRANDA SANTIAGO",
  edad:20,
  correo:"juan@gmail.com",
  EP:"INGENIERIA DE SISTEMAS"
};

console.log(alumno);
console.log(alumno["nombre"]);
console.log(alumno.nombre);

//se cambia el valor del apellido
alumno.apellido="SANTIAGO MIRANDA";
console.log(alumno);
```

Los objetos aparte de tener atributos también tienen métodos que no son otra cosa que funciones que hacen algún tipo de operación. En el ejemplo anterior agregaremos el objeto Escuela Profesional **ep1**, el **alumno001** pertenece a la **ep1** y en este último

objeto agregamos un método ***verEscuelaNombre()***, que permite visualizar el nombre de la escuela, tal como se muestra en la

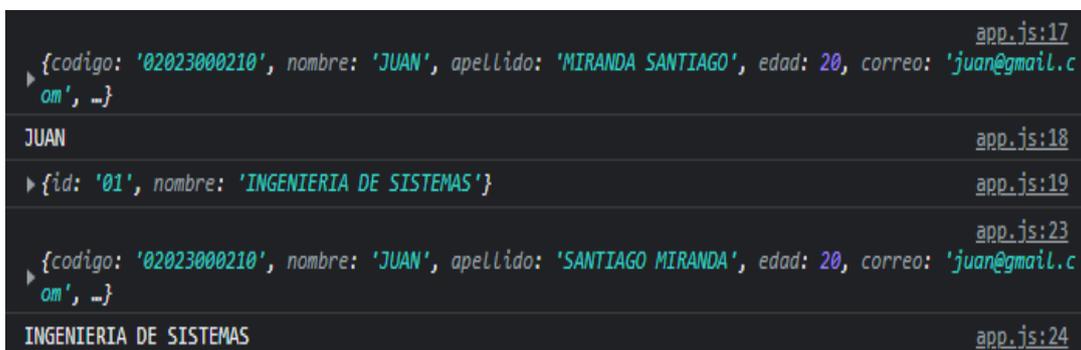
```
const ep1={
  id:"01",
  nombre:"INGENIERIA DE SISTEMAS"
}
```

```
const alumno01={
  codigo:"02023000210",
  nombre:"JUAN",
  apellido:"MIRANDA SANTIAGO",
  edad:20,
  correo:"juan@gmail.com",
  ep:ep1,
  verEscuelaNombre: ()=>{return ep1.nombre}
};

console.log(alumno01);
console.log(alumno01["nombre"]);
console.log(alumno01.ep);

//se cambia el valor del apellido
alumno01.apellido="SANTIAGO MIRANDA";
console.log(alumno01);
console.log(alumno01.verEscuelaNombre());
```

En el siguiente grafico se puede ver el resultado de acceder a los atributos del objeto y al método ***verEscuelaNombre()***.



```
app.js:17
{codigo: '02023000210', nombre: 'JUAN', apellido: 'MIRANDA SANTIAGO', edad: 20, correo: 'juan@gmail.com', ...}
JUAN
app.js:18
{id: '01', nombre: 'INGENIERIA DE SISTEMAS'}
app.js:19
{codigo: '02023000210', nombre: 'JUAN', apellido: 'SANTIAGO MIRANDA', edad: 20, correo: 'juan@gmail.com', ...}
INGENIERIA DE SISTEMAS
app.js:24
```

FIGURA N° 02-005: Resultado en consola del ejemplo anterior

2.3. Sentencias simples

sentencias de control

Sentencias simples

Las sentencias simples son aquellas líneas de código que no cambian el flujo de ejecución del programa. Por ejemplo, para hacer un programa que calcule la suma de n números. Se solicita los n números a través del método `prompt()` del Windows y aplicamos la fórmula $n(n+1)/2$. Estas líneas de código en JavaScript representarían así:

```
//se pide el numero
const n1=prompt("ingresar un numero");
//se calcula la suma
const n2=parseInt(n1);//se realiza el cambio de tipo

//se aplica la formula
const suma=n2*(n2+1)/2;

//se imprime por pantalla
console.log(suma);
```

Sentencias de control simple

Las sentencias de control simple, es la que direccionan el flujo del programa:

```
if(proposicion)
{
//si proposición es verdadero
}else{
//si proposición es falso
}
```

Ejemplo 02-001: Realizar un programa que imprima el máximo valor de 5 números ingresados.

DESARROLLO

Para este programa solicitamos los 5 números con **prompt()**, asignamos a una variable **mayor** el primer número, luego se compara con los demás:

*Si **mayor** < **n2** entonces **mayor**=**n2***

*Si **mayor** < **n3** entonces **mayor**=**n3***

*Si **mayor** < **n4** entonces **mayor**=**n4***

*Si **mayor** < **n5** entonces **mayor**=**n5***

*Se imprime **mayor***

Código en JavaScript:

```
let n1=parseInt(prompt("INGRESAR ENUMERO 1:"));
let n2=parseInt(prompt("INGRESAR ENUMERO 2:"));
let n3=parseInt(prompt("INGRESAR ENUMERO 3:"));
let n4=parseInt(prompt("INGRESAR ENUMERO 4:"));
let n5=parseInt(prompt("INGRESAR ENUMERO 5:"));
let mayor=n1;
if(mayor<n2)
    mayor=n2;
if(mayor<n3)
    mayor=n3;
if(mayor<n4)
    mayor=n4;
if(mayor<n5)
    mayor=n5;
console.log("EL MAYOR DE:",n1,n2,n3,n4,n5,"ES:",mayor);
```

Sentencias de control múltiple

Las sentencias de control múltiple direccionan el flujo del programa, permiten tener múltiples alternativas sobre el valor de comparación de la expresión dada como argumento al **switch()**:

```
switch(expresión)
{
case valor1:
    //sentencia
    Break;
case valor2:
    //sentencia
    break;
default:
    //sentencia
    break;
}
```

Ejemplo 02-002: Realizar un programa que permita capturar la fecha del sistema y a partir de ello formatear de la siguiente manera:

AMARILIS: Domingo 16 DE ABRIL DEL 2023

DESARROLLO

Para capturar la fecha del sistema utilizamos el ***new Date()***, ***.getFullYear()*** para obtener el año, ***.getMonth()*** para obtener el mes, como empieza en 0 se le suma más 1, ***.getDay()*** obtiene día de la semana, ***.getDate()*** obtiene el día del mes, luego de ello utilizamos el ***switch()*** para convertir el mes correspondiente en letras: *1="ENERO"*, *2="FEBRERO"*,...

Luego utilizamos otro ***switch()*** para pasar el número de la semana a letras: *1="LUNES"*, *2="MARTES"*,... Luego se da formato a la fecha con la salida solicitada tal como se muestra en la FIGURA N° 02-006.

Código en JavaScript

```
let fecha=new Date();//se captura la fecha
const anio=fecha.getFullYear();//el año
const mes=fecha.getMonth()+1;//el mes
let mesL="";
const diaSemana=fecha.getDay();//dia de la semana
const diaMes=fecha.getDate();//dia mes
```

```
switch(mes) {  
  case 1:  
    mesL="ENERO";  
    break;  
  case 2:  
    mesL="FEBRERO";  
    break;  
  case 3:  
    mesL="MARZO";  
    break;  
  case 4:  
    mesL="ABRIL";  
    break;  
  
  case 5:  
    mesL="MAYO";  
    break;  
  case 6:  
    mesL="JUNIO";  
    break;  
  case 7:  
    mesL="JULIO";  
    break;  
  case 8:  
    mesL="AGOSTO";  
    break;  
  case 9:  
    mesL="SETIEMBRE";  
    break;  
  case 10:  
    mesL="OCTUBRE";  
    break;  
  case 11:  
    mesL="NOVIEMBRE";  
    break;  
  case 12:  
    mesL="DICIEMBRE";  
    break;  
  default:  
    mesL="";  
    break;  
}
```

```

//-----
//de numero a letra el dia de la semana
//-----
let diaSemanaL="";
switch(diaSemana){
  case 1:
    diaSemanaL="Lunes";
    break;
  case 2:
    diaSemanaL="Martes";
    break;
  case 3:
    diaSemanaL="Miercoles";
    break;
  case 4:
    diaSemanaL="Jueves";
    break;
  case 5:
    diaSemanaL="Viernes";
    break;
  case 6:
    diaSemanaL="Sabado";
    break;
  case 0:
    diaSemanaL="Domingo";
    break;
  default:
    diaSemanaL=" ";
    break;
}
console.log("AMARILIS:",diaSemanaL,diaMes," DE ",mesL," DEL
",anio);

```

La salida del ejemplo en consola se muestra con la FIGURA N° 02-006

AMARILIS: Domingo 16 DE ABRIL DEL 2023

app.js:94

FIGURA N° 02-006: Resultado en consola del ejemplo

2.4. Sentencias repetitivas

Las sentencias que hacen repetir la ejecución de código son: ***for()***, ***while()***, ***do{}while()***, ***foreach***, etc.

Las sintaxis de cada uno de ellos se detallan a continuación

for(condición inicial, condición de salida, variación)

```
{
//sentencia
}
```

while(proposicion)

```
{
//sentencia
}
```

```
do{
//sentencia
}while(proposicion);
```

Array.forEach(function callback(element,index,Array){})

Ejemplo 02-003: Realizar un programa que permita generar aleatoriamente **n** números (**n** se puede ingresar con ***prompt()***) y los almacena en un arreglo con sus respectivos divisores, a partir de ello utilizar el ***.map()*** para generar otro arreglo con la cantidad de divisores de cada número.

DESARROLLO

Para ingresar la cantidad de números utilizamos el ***prompt()***, luego almacenamos en una variable para luego hacer un casting para pasarlo a un tipo numérico, si el ingreso es válido con un bucle ***for()*** se genera los números aleatorios con la función de ***Math.random()*** y se asigna al atributo ***valor*** de un objeto ***numero***, luego de ello se saca los divisores de dicho número, para ello se utiliza el bucle ***do{ }while()*** y se almacena en el arreglo ***divisores*** del objeto ***número***.

Código en JavaScript

```
const cantNum=prompt("INGRESE UN NUMERO:");

//para asegurar que el ingreso sea nun numero
//hacemos un casting
const cN=parseInt(cantNum);

//validamos si es un numero
if(typeof cN =='number')
{
const numeros=[];

for(let i=0;i<cN;i++)
{

//se declara un objeto con dos atributos
//se asigna el numero generado
let numero={
    valor:Math.round(Math.random()*100),
    divisores:[]
}

//-----
//sacamos la cantidad de divisores
//del numero generado aleatoriamente
let n=numero.valor;
let j=1;
do{
    if(n%j===0)
        numero.divisores.push(j);
    j++;

}while(j<=n);

//guardamos al arreglo numeros el numero
numeros.push(numero);
}
```

```

console.log( numeros ); //visualizamos los numeros

//con .map() sacamos la cantidad de divisores
//para ello recorremos cada elemento y contamos
//el tamaño que tiene el atributo divisores
//de cada numero
const divisores =
numeros.map( element => element.divisores.length )

console.log( divisores ); //visualizamos el # de divisores
}

```

EJERCICIOS N°02-001.- Realizar programas en JavaScript para los siguientes casos:

1. Determinar el menor y mayor valor de n números.
2. Realizar un programa para construir la ecuación de una circunferencia a partir de 3 puntos.
3. Hacer una calculadora en letras de tal manera que te realiza las 4 operaciones básicas (+, -, *, /). El Ingreso de los números y los operadores se debe realizar en letras, Por ejemplo.

```

INGRESE EL PRIMER NUMERO.....>>DOS
INGRESE EL SEGUNDO NUMERO..>>DOCE
INGRESE EL OPERADOR.....>>SUMAR
LA SUMA ES.....>>CATORCE

```

4. Realizar un programa para que factorice un número entero, mayor que 99 ingresado por el teclado, Por ejemplo.

```

INGRESE EL NUMERO .....>>100
EL NUMERO EN SUS FACTORES PRIMOS ES>>22x52

```

5. Realizar un programa para que evalúe una función ingresada por el teclado, Por ejemplo.

```

INGRESE LA FUNCION>>sen(x)+x
INGRESE UN VALOR PARA "x">>0
EL VALOR DE:f(0)=0
DESEA SEGUIR EVALUANDO LA FUNCION SI(1)/NO(0)>>1
INGRESE UN VALOR PARA "x">>3.1416

```

EL VALOR DE: $f(3.1416)=3.1416$

DESEA SEGUIR EVALUANDO LA FUNCION SI(1)/NO(0)>>0

DESEA INGRESAR OTRA FUNCION SI(1)/NO(0)>>0

“HASTA PRONTO”

6. Dado un número entero, determinar si es o no es capicúa. Recuerde que un número es capicúa cuando se lee igual de izquierda a derecha que de derecha a izquierda. Por ejm. 22, 656, 2332, etc.
7. Genere números aleatorios enteros de 4 cifras hasta que se obtenga uno que tenga sólo cuatro divisores.
8. El Centro Médico “**Pillco Mozo**” requiere sistematizar su administración con la ayuda de un software, por lo que se necesita gestionar los datos de los pacientes (DNI, apellidos y nombres, teléfono, fecha de nacimiento, historial clínico, diagnóstico de enfermedades, etc.), personal de salud (DNI, apellidos y nombres, título, especialidad, tipo de contrato), personal de administración (DNI, apellidos y nombres, contrato, área, cargo, etc.). Se necesita tener datos de las enfermedades como: nombre, aparato o sistema corporal al que afecta, descripción de la enfermedad. Los pacientes pasan por triaje para recabar datos como: peso, talla, signos vitales, etc. Luego pasan a medicina general en donde el Médico realiza un diagnóstico luego de ello es derivado a las especialidades correspondientes para su atención. Se necesita tener el control de citas. Realizar el programa en JavaScript que permita sistematizar los requerimientos del Centro Médico “**Pillco Mozo**”
9. En el proceso de “**Bienes y Suministros**” de la empresa de turismo “**Pata Amarilla**” que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de “**Bienes y Suministros**” detalla el procedimiento y actividades de cada área:
 - ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.
 - ✓ El área de Logística funciona de la siguiente forma:
 - Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
 - Cada solicitud tiene un responsable que está adscrito a un área.

- Cada solicitud es autorizada por el jefe del área y posteriormente por el área de Administración.
 - De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
 - Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
 - La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
 - Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.
 - Para comprar bienes primero se realiza la cotización dos o tres proveedores para determinar la mejor oferta en calidad y precio que un proveedor ofrece, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: número de la **orden de compra**, nombre del proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega. Cada orden puede tener asociado uno o varios ítems, cada ítem debe tener los siguientes datos: código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.
 - La orden de compra es aprobada por el área de Administración luego el área de Logística envía o hace entrega al proveedor elegido.
- ✓ El área de Almacén funciona de la siguiente forma:
- Recepciona los bienes que llegan de los proveedores y distribuye con una pecosa a las áreas que realizaron las solicitudes.
 - El proveedor hace entrega a Almacén los ítems detallado en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: número de entrada,

fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.

- El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes a través de una peca detallando cada ítem entregado y otros datos más como el número de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega.
- ✓ El área Patrimonio es responsable de:
 - Administrar y controlar la ubicación de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera su estiquete que va pegado al bien.
 - Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

Realizar el programa en JavaScript que permita sistematizar todo lo detallado en el proceso de “**Bienes y Suministros**” de la empresa de turismo “**Pata Amarilla**”.

2.5. JSON

JSON (JavaScript Object Notation) es una estructura de datos estándar que permite el intercambio de los mismos, es soportado por diferentes lenguajes de programación y motores de bases de datos en su almacenamiento. Su amplio uso para el almacenamiento e intercambio de datos radica en la sencillez de sus estructuras: {"Clave": "Valor"}, los tipos de datos en **valor** podrían ser:

- ✓ Cadena
- ✓ Numéricos
- ✓ Booleanos
- ✓ Array
- ✓ Objetos
- ✓ null

En el siguiente ejemplo tenemos una representación de datos de un alumno en JSON, notar que los atributos van entre doble comilla, el atributo **dirección** es un objeto en formato JSON, el atributo **deportes** es un arreglo.

```
{
  "nombre": "JUAN",
  "apellidos": "Lopez Espinoza",
  "direccion": {
    "departamento": "HUANCUO",
    "provincia": "HUANUCO",
    "distrito": "AMARILIS",
    "ubicacion": "jr ricardo palma #1234"
  },
  "telefono": "978458745",
  "correo": "juan64@gmail.com",
  "escuela": "Ingenieria de Sistemas",
  "deportes": ["Futbol", "Natacion", "basketball"]
}
```

Algunos métodos que permite hacer operaciones con el Objeto JSON:

JSON.parse()

Se le pasa como parámetro una cadena JSON y transforma en un objeto de JavaScript. Este método puede tener una función como segundo argumento que podrían transformar los valores del primer parámetro antes de retornar. En el siguiente ejemplo se declara una constante **alumno** como cadena y con **JSON.parse()** lo pasamos a un objeto JavaScript

```
const alumno=` {
  "nombre":"JUAN",
  "apellidos":"Lopez Espinoza",
  "direccion":{
    "departamento":"HUANCUO",
    "provincia":"HUANUCO",
    "distrito":"AMARILIS",
    "ubicacion":"jr ricardo palma #1234"
  },
  "telefono":"978458745",
  "correo":"juan64@gmail.com",
  "escuela":"Ingenieria de Sistemas",
  "deportes":["Futbol","Natacion","basketball"]
}`
console.log(alumno); //se imprime en consola la cadena
console.log(typeof alumno); //se imprime el tipo de dato
//se pasa a un objeto JavaScript con JSON.parse()
const dato=JSON.parse(alumno)
//se imprime en consola el dato como objeto
console.log(dato);
//se imprime en consola para ver el tipo de dato
console.log(typeof dato);
```

En la FIGURA N° 02-007 se puede observar la salida de aplicar el método **JSON.parse()**: En la parte (1) se ver la cadena, en la parte (2) se puede observar que el tipo de dato de “**alumno**” es **string**, en la parte (3) luego de aplicar el método se visualiza el objeto JavaScript, en la parte (4) se observa el tipo de dato de “**dato**” que es **object**

```

{
  "nombre": "JUAN",
  "apellidos": "Lopez Espinoza",
  "direccion": {
    "departamento": "HUANCUO",
    "provincia": "HUANUKO",
    "distrito": "AMARILIS",
    "ubicacion": "jr ricardo palma #1234"
  },
  "telefono": "978458745",
  "correo": "juan64@gmail.com",
  "escuela": "Ingenieria de Sistemas",
  "deportes": ["Futbol", "Natacion", "basketball"]
}
string
{nombre: 'JUAN', apellidos: 'Lopez Espinoza', direccion: {-}, telefono: '978458745', correo: 'juan64@gmail.com', -}
  apellidos: "Lopez Espinoza"
  correo: "juan64@gmail.com"
  deportes: (3) ["Futbol", "Natacion", "basketball"]
  direccion: {departamento: 'HUANCUO', provincia: 'HUANUKO', distrito: 'AMARILIS', ubicacion: 'jr r escuela: "Ingenieria de Sistemas" nombre: "JUAN" telefono: "978458745" [[Prototype]]: Object
object

```

FIGURA N° 02-007: Resultado de aplicar el método *JSON.parse()*

En el ejemplo anterior con el mismo método *JSON.parse()*, como segundo parámetro lo pasamos una función que convierta con el método *toUpperCase()* a mayúscula los atributos que son de tipo *string*.

```

const dato1=JSON.parse(alumno,(key, value) => {
  if (typeof value === 'string') {
    return value.toUpperCase();
  }
  return value;
})
console.log(dato1);

```

Como resultado se puede observar en la FIGURA N° 02-008, en donde todos los atributos de tipo *string* se pasó a mayúscula.

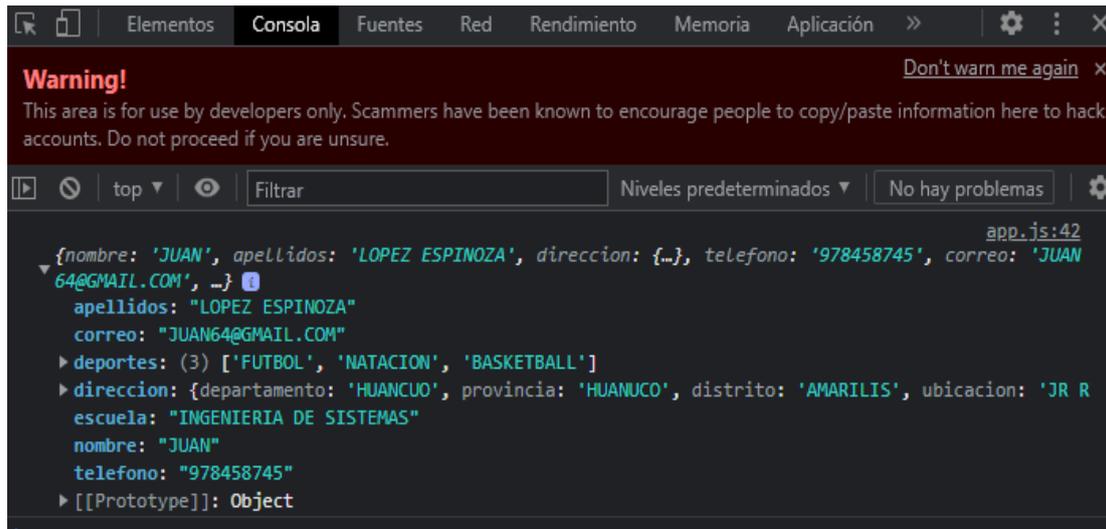


FIGURA N^o 02-008: Al aplicar el método **JSON.parse()** con una función como segundo argumento que pasa los atributos de tipo **string** a mayúscula

JSON. stringify(objeto[, replacer[, space]])

A este método se le pasa como parámetro un objeto JavaScript y lo transforma en una cadena JSON, ósea todas las claves encerrado entre comilla doble, también se le puede pasar como argumento un *array*. Además, puede tomar dos argumentos adicionales: el primero una función *replacer* y el segundo un valor *String* o *Number* que se utiliza para insertar espacio(space) en blanco dentro de la cadena de salida JSON para mejorar su legibilidad. Si es de tipo *Number*, indica el número de espacios a usar como espacios en blanco el máximo valor es 10. Los valores menores a 0 no se toma en cuenta. Si es de tipo *String*, la cadena (o sus 10 primeros caracteres) se utiliza como espacios en blanco

En el siguiente ejemplo se tiene una colección de objetos con datos de dos alumnos, cada uno de ellos tiene un *id* que los identifica de manera única en la colección, esto nos permite hacer modificaciones, eliminar, hacer consultas, entre otros que se requiere.

```

const alumnos={"0001": {
  nombre:"JUAN",
  apellidos:"Lopez Espinoza",
  direccion:{
    "departamento":"HUANCUO",
    "provincia":"HUANUCO",
    "distrito":"AMARILIS",
    "ubicacion":"jr ricardo palma #1234"
  },
  telefono:"978458745",
  correo:"juan64@gmail.com",
  escuela:"Ingenieria de Sistemas",
  deportes:["Futbol","Natacion","basketball"]
},
"0002": {
  nombre:"MARIA",
  apellidos:"Santiago Rivera",
  direccion:{
    "departamento":"HUANCUO",
    "provincia":"HUANUCO",
    "distrito":"HUANUCO",
    "ubicacion":"dos de mayo #129"
  },
  telefono:"954865458",
  correo:"mari_123@gmail.com",
  escuela:"Ingenieria de Sistemas",
  deportes:["agedrez","natacion","basketball"]
}
};

//se visualiza los objetos en consola
console.log(alumnos);
//se visualiza el tipo de dato
console.log(typeof alumnos);
//pasamos de objeto a formato JSON
const dato=JSON.stringify(alumnos);
//se visualiza en consola
console.log(dato);
//se visualiza el tipo de dato
console.log(typeof dato);

```

En la FIGURA N° 02-009 se puede observar la salida de aplicar el método **JSON.stringify()**: En la parte (1) se ve la cadena, en la parte (2) se puede observar que el tipo de dato de la colección **"alumnos"** es **object**, en la parte (3) luego de aplicar

el método se visualiza la salida en formato **JSON**, en la parte (4) se observa el tipo de dato de “**dato**” que es **string**

```

Warning!
This area is for use by developers only. Scammers have been known to encourage people to copy/paste information here to hack accounts. Do not proceed if you are unsure.

{0001: {...}, 0002: {...}}
  0001:
    apellidos: "Lopez Espinoza"
    correo: "juan64@gmail.com"
    deportes: (3) ['Futbol', 'Natacion', 'basketball']
    direccion: {departamento: 'HUANCUO', provincia: 'HUANCUO', distrito: 'AMARILIS', ubicacion: 'jr ricardo palma #1234', telefono: '978458745', correo: 'juan64@gmail.com', escuela: 'Ingenieria de Sistemas', deportes: ['Futbol', 'Natacion', 'basketball']}
    nombre: "JUAN"
    telefono: "978458745"
    [[Prototype]]: Object
  0002:
    apellidos: "Santiago Rivera"
    correo: "mari_123@gmail.com"
    deportes: (3) ['agedrez', 'natacion', 'basketball']
    direccion: {departamento: 'HUANCUO', provincia: 'HUANCUO', distrito: 'HUANCUO', ubicacion: 'dos de mayo #129', telefono: '954865458', correo: 'mari_123@gmail.com', escuela: 'Ingenieria de Sistemas', deportes: ['agedrez', 'natacion', 'basketball']}
    nombre: "MARIA"
    telefono: "954865458"
    [[Prototype]]: Object
    [[Prototype]]: Object
object
{"0001":{"nombre":"JUAN","apellidos":"Lopez Espinoza","direccion":{"departamento":"HUANCUO","provincia":"HUANCUO","distrito":"AMARILIS","ubicacion":"jr ricardo palma #1234"},"telefono":"978458745","correo":"juan64@gmail.com","escuela":"Ingenieria de Sistemas","deportes":["Futbol","Natacion","basketball"]},"0002":{"nombre":"MARIA","apellidos":"Santiago Rivera","direccion":{"departamento":"HUANCUO","provincia":"HUANCUO","distrito":"HUANCUO","ubicacion":"dos de mayo #129"},"telefono":"954865458","correo":"mari_123@gmail.com","escuela":"Ingenieria de Sistemas","deportes":["agedrez","natacion","basketball"]}}
string
  
```

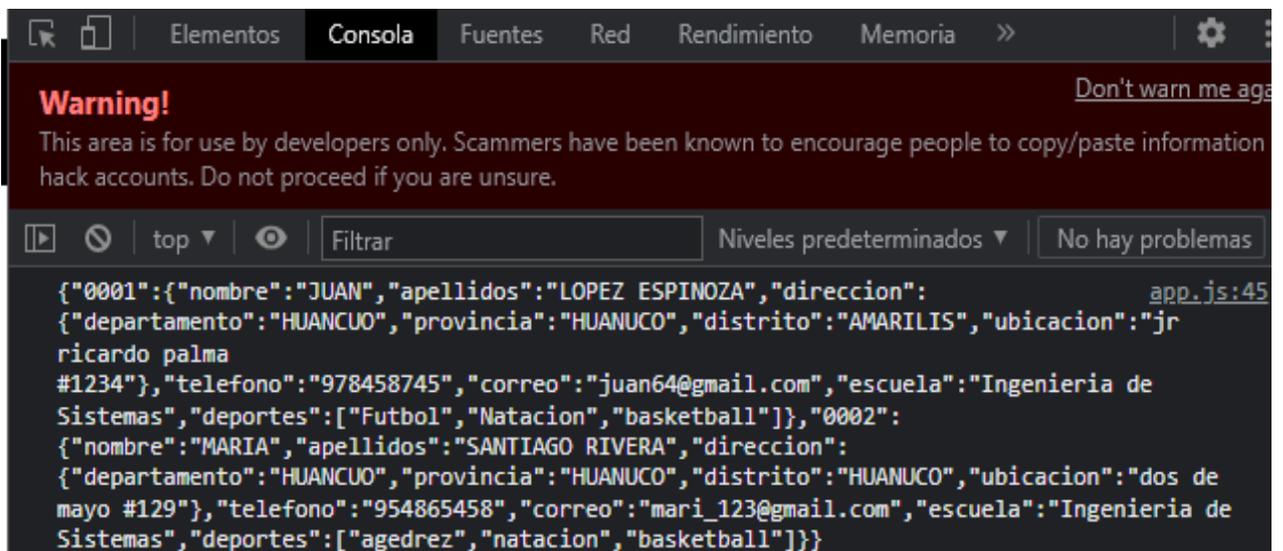
FIGURA N° 02-009: Resultado de aplicar el método *JSON.stringify()*

En el ejemplo anterior con el mismo método **JSON.stringify()**, como segundo parámetro se pasa una función que convierta con el método **.toUpperCase()** a mayúscula los atributos que son de tipo **string**.

```
function replacer(key, value) {
  if (key === 'apellidos') {
    return value.toUpperCase();
  }
  return value;
}

//El metodo stringify() con dos parametros
const alumnos1 = JSON.stringify(alumnos, replacer);
console.log(alumnos1);
```

En la FIGURA N° 02-010 se muestra en consola el objeto convertido en formato JSON, todos los atributos están encerrados entre comillas dobles.



```
Warning! Don't warn me aga
This area is for use by developers only. Scammers have been known to encourage people to copy/paste information
hack accounts. Do not proceed if you are unsure.

Filtrar Niveles predeterminados No hay problemas

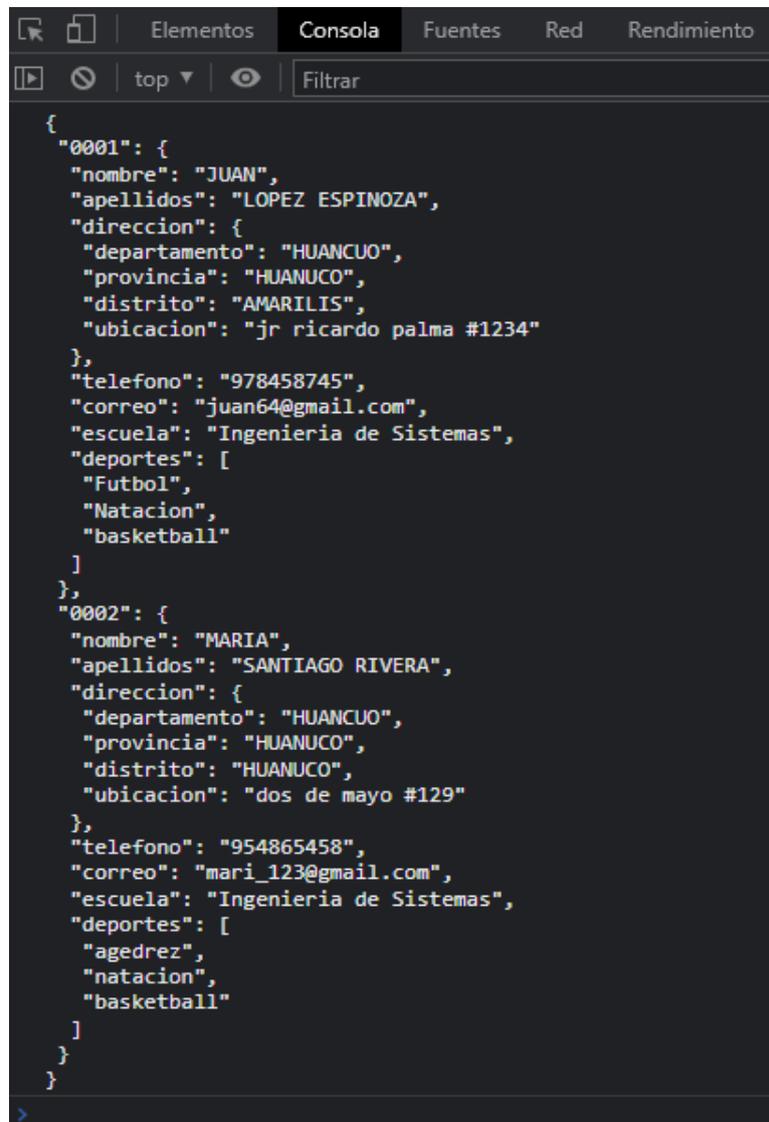
{"0001":{"nombre":"JUAN","apellidos":"LOPEZ ESPINOZA","direccion":
{"departamento":"HUANCUO","provincia":"HUANUCO","distrito":"AMARILIS","ubicacion":"jr
ricardo palma
#1234"},"telefono":"978458745","correo":"juan64@gmail.com","escuela":"Ingenieria de
Sistemas","deportes":["Futbol","Natacion","basketball"]},"0002":
{"nombre":"MARIA","apellidos":"SANTIAGO RIVERA","direccion":
{"departamento":"HUANCUO","provincia":"HUANUCO","distrito":"HUANUCO","ubicacion":"dos de
mayo #129"},"telefono":"954865458","correo":"mari_123@gmail.com","escuela":"Ingenieria de
Sistemas","deportes":["agedrez","natacion","basketball"]}}
```

FIGURA N° 02-010: Resultado de aplicar el método `JSON.stringify()`, con un segundo argumento como función

Al mismo ejemplo con el método `JSON.stringify()`, como segundo parámetro una función que convierta con el método `.toUpperCase()` a mayúscula los atributos que son de tipo **string**, además un tercer parámetro con valor 1, para dar visibilidad a la salida.

```
//El metodo stringify() con tres parametros
const alumnos2 = JSON.stringify(alumnos, replacer, 1);
console.log(alumnos2);
```

En la FIGURA N° 02-011 se muestra en consola el objeto convertido en formato JSON, todos los atributos están encerrados entre comillas dobles, además con una visibilidad mejor.



```

{
  "0001": {
    "nombre": "JUAN",
    "apellidos": "LOPEZ ESPINOZA",
    "direccion": {
      "departamento": "HUANCUO",
      "provincia": "HUANUCO",
      "distrito": "AMARILIS",
      "ubicacion": "jr ricardo palma #1234"
    },
    "telefono": "978458745",
    "correo": "juan64@gmail.com",
    "escuela": "Ingenieria de Sistemas",
    "deportes": [
      "Futbol",
      "Natacion",
      "basketball"
    ]
  },
  "0002": {
    "nombre": "MARIA",
    "apellidos": "SANTIAGO RIVERA",
    "direccion": {
      "departamento": "HUANCUO",
      "provincia": "HUANUCO",
      "distrito": "HUANUCO",
      "ubicacion": "dos de mayo #129"
    },
    "telefono": "954865458",
    "correo": "mari_123@gmail.com",
    "escuela": "Ingenieria de Sistemas",
    "deportes": [
      "agedrez",
      "natacion",
      "basketball"
    ]
  }
}

```

FIGURA N° 02-011: Resultado de aplicar el método `JSON.stringify()`, con un segundo argumento como función y un tercer argumento con el valor 1 para una mejor visibilidad en la salida

Nota. - Los archivos que contienen datos en formato JOSN se guardan con la extensión *.json

2.6. Funciones

Las funciones en JavaScript son un bloque de código que realiza algo específico y que pueden o no retornar un valor. Se tiene la siguiente sintaxis

```
function nombreFuncion(parametro1,parametro2, parametro3,...)
{
    //bloque de código
    //si retorna un valor
    return valor;
}
```

Para invocarlas o utilizarlas las veces que se requiera a las funciones solo hay que escribir su nombre con paréntesis y los parámetros establecidos:

nombreFuncion(parametro1,parametro2, parametro3,...)

Si los parámetros son primitivos (numero, string, booleam, etc.), estos son pasadas a la función por valor. Si se cambia el valor dentro de la función esto no se refleja fuera. En el siguiente ejemplo se declara dos variables: `n1=0`, `n2=3`; se le pasa como parámetro a la función `suma(n1,n2)`, dentro de la funcion se cambian los valores, estos no es reflejado fuera de la función.

```
var n1=0,n2=3;

function sumar(x,y)
{
    x=4; //se cambia de valor a n1
    y=8; //se cambia de valor al n2
    return x+y;//retorna la suma
}

console.log(n1,n2);//se visualiza en consola 0 y 3
console.log(sumar(n1,n2));//se visualiza en consola 12
console.log(n1,n2);//se visualiza en consola 0 y 3
```

Si los parámetros son objetos, array y se cambian sus atributos o valores dentro de la función, estos son reflejados fuera de la función.

En el siguiente ejemplo se declara el objeto `alumno1` con sus atributos **nombre: "JOSE"**, **edad:25**. La función `saludar()` recibe como parámetro el objeto `alumno1`.

Dentro de la función se cambia de valor al parámetro nombre="PEDRO", este cambio se refleja fuera del ámbito de la función.

```
var alumno1={
  nombre:"JOSE",
  edad:25
}

function saludar(objetoAlumno)
{
  objetoAlumno.nombre="PEDRO";
  return "Hola "+objetoAlumno.nombre;
}

console.log(alumno1);//se visualiza el objeto en consola
console.log(saludar(alumno1));//se cambio el atributo nombre
console.log(alumno1);//el atributo nombre tiene otro valor
```

En JavaScript hay tres formas de crear funciones:

- ✓ La primera forma es a través de funciones declaradas
- ✓ La segunda forma es a través de funciones expresadas
- ✓ La tercera forma es a través de funciones flecha

Funciones declaradas

La sintaxis de este tipo de funciones se muestra a continuación:

```
// Function declaration
function nombreFuncion(parametro1, parametro2,...){
  // bloque de codigo
  return valor_retorno
}
```

Estas funciones pueden ser invocadas antes de su definición

Funciones expresadas

La sintaxis de este tipo de funciones se muestra a continuación:

```
// Function expresada
const nombreConstante=function(parametro1, parametro2,...){
  //bloque de codigo
}
```

Estas funciones no pueden ser invocadas antes de su definición.

Otra forma de función expresada son las autoejecutables:

```
// Function autoejecutable
( function() {
// bloque de codigo
}) ();
```

Funciones de tipo flecha (arrow function)

Estas funciones son expresiones más compactas de una función tradicional, se les llama función flecha por que tiene la siguiente sintaxis:

```
const nombreConstante = (param1, param2,...) => línea de codigo;
```

Se crea la función *nombreConstante* que acepta parámetros y ejecuta el código que esta después de la flecha y retorna un resultado. En el caso que se requiera más líneas de código a la derecha de la flecha, se encierra entre **{ }** y se utiliza **return** para devolver un valor.

```
const nombreConstante = (param1, param2,...) => {
//bloque de codigo
return valor }
```

Funciones recursivas

Son aquellas funciones declaradas que se llaman a sí mismo dentro de su definición, podría tener la siguiente sintaxis:

```
// Function recursiva
function nombreFuncion(parametro) {

    if(parámetro==1)
        return 1;
    else
        return nombreFuncion(parámetro-1);
}
```

Parametros rest en una función

Cuando no se tiene establecido la cantidad de parámetros de una función, JavaScript permite majear ello de manera dinámica estableciendo con parámetros rest(**...parametro**) de la siguiente manera:

```
// Function con parámetros rest
function nombreFuncion(...parametro) {

    // bloque de codigo

    return valor;

}
```

En este caso “**parametro**” es un **Array**, para su gestión utilizamos todos los métodos relacionados a ello.

En el siguiente ejemplo se tiene una función que calcula el promedio de **n** notas, al momento de llamar pasamos diferente cantidad de parámetros

```
//-----
//funcion con parametros rest, que calula el promedio
//de n notas
function promedio(...nota){
    //se suman todas las notas y se divide entre
    //la cantidad que vendria a ser el tamaño del array nota
    const pf=nota.reduce((a,p)=>p+a)/nota.length;
    return pf;
}
//-----

//llamamos a la funcion con diferentes cantidades de
//parametros
console.log(promedio(16,20));//imprime 18
console.log(promedio(16,20,18,10));//imprime 16
console.log(promedio(12,13,14,15,16));//imprime 14
```

Para este ejemplo se utilizó el método **Array.reduce()** dentro de la función para sumar los promedios, este método tiene la siguiente sintaxis:

```
Array.reduce(callback(acumulador, valorActual[,
índice[, array]])[, valorInicial])
```

El **callback** es la función a ejecutar sobre cada elemento del **Array** a excepción para el primero en caso que no se proporciona **valorInicial**.

Ejemplo 02-004: Realizar una función en JavaScript que calcule el factorial de un número **n** de manera recursiva.

DESARROLLO

Según definición; el factorial de un numero está representado por $n!$ y se calcula de la siguiente manera: $n! = 1 \times 2 \times 3 \times 4 \dots \times n$ donde $n \in \mathbb{N}$.

Por ejemplo:

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

$$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$

El factorial de un numero n se puede definir de esta manera:

$$f_1 = 1$$

$$f_2 = f_1 \times 2$$

$$f_3 = f_2 \times 3$$

. = .

. = .

. = .

$$f_n = f_{(n-1)} \times n$$

Esto en matemáticas discretas se plantearía con una definición recursiva de la siguiente manera:

$$\begin{cases} f_1 = 1 \\ f_n = n f_{(n-1)}, \forall n \in \mathbb{N} \end{cases}$$

Toda definición recursiva siempre tiene condiciones iniciales o de salida, en este caso: $f_1 = 1$. No establecer esto conllevaría a generarnos un bucle indeterminado.

La definición recursiva de la suma de los n números enteros sería de la siguiente manera:

$$\begin{cases} S_1 = 1 \\ S_n = S_{(n-1)} + n, \forall n \in \mathbb{N} \end{cases}$$

Con esta definición es fácil implementar una función recursiva que calcule el factorial de un numero n tal como se muestra a continuación:

```

//-----
/**
 * Esta función recibe como parametro un numero entero n
 * y calcula el factorial
 * por ejm. factorial(6) -> 720
 * @author Elmer Chuquiyaauri
 * @version v1.0
 * @param {Number} n - numero que se calculara el factorial.
 * @return{Number} el factorial de n
 */
//-----
function factorial(n){
  let fact=1;
  if(n==1)
    fact=1; //condición de salida
  else
    fact=n*factorial(n-1); //se llama a la misma funcion

  return fact;
}
const n=prompt("ingrese un numero:");
console.log(factorial(n));

```

La suma de n números con una función recursiva estaría implementado de la siguiente manera:

```

function sumar(n){
  let suma=0;
  if(n==1)
    suma=1; //condición de salida
  else
    suma=sumar(n-1)+n; //se llama a la misma funcion
  return suma;
}

console.log("la suma es: ",sumar(10))

```

Ejemplo 02-005: Realizar una función en JavaScript que permita pasar un numero entero de hasta 15 cifras a su correspondiente número en letras.

DESARROLLO

Para desarrollar este programa utilizaremos funciones recursivas

```
//-----
/**
 * Esta funcion recibe como parametro un numero entero
 * y lo convierte en su equivalente en letras
 * por ejm. numeroLetra(14) -> CATORCE
 * @author Elmer Chuquiyaury
 * @version v1.0
 * @param {Number} n -El numero que se quiere pasar a letra.
 * @return{String} numero_letra - El numero e letra
 */
//-----

function numeroLetra(n)
{
  let numeroString=n.toString();//EL NUMERO A STRING
  let decena=""
  let numCifras=numeroString.length;
  let numero_letra="";
  //-----
  //numeros unicos
  const numerosUnicos={}

  numerosUnicos[0]={numero:"0",descripcion:"CERO"}
  numerosUnicos[1]={numero:"1",descripcion:"UNO"}
  numerosUnicos[2]={numero:"2",descripcion:"DOS"}
  numerosUnicos[3]={numero:"3",descripcion:"TRES"}
  numerosUnicos[4]={numero:"4",descripcion:"CUATRO"}
  numerosUnicos[5]={numero:"5",descripcion:"CINCO"}
  numerosUnicos[6]={numero:"6",descripcion:"SEIS"}
  numerosUnicos[7]={numero:"7",descripcion:"SIETE"}
  numerosUnicos[8]={numero:"8",descripcion:"OCHO"}
  numerosUnicos[9]={numero:"9",descripcion:"NUEVE"}
  numerosUnicos[10]={numero:"10",descripcion:"DIEZ"}
  numerosUnicos[11]={numero:"11",descripcion:"ONCE"}
  numerosUnicos[12]={numero:"12",descripcion:"DOCE"}
  numerosUnicos[13]={numero:"13",descripcion:"TRECE"}
  numerosUnicos[14]={numero:"14",descripcion:"CATORCE"}
  numerosUnicos[15]={numero:"15",descripcion:"QUINCE"}
}
```

```

numerosUnicos[20]={numero:"20",descripcion:"VEINTE"}
numerosUnicos[30]={numero:"30",descripcion:"TREINTA"}
numerosUnicos[40]={numero:"40",descripcion:"CUARENTA"}
numerosUnicos[50]={numero:"50",descripcion:"CINCUENTA"}
numerosUnicos[60]={numero:"60",descripcion:"SESENTA"}
numerosUnicos[70]={numero:"70",descripcion:"SETENTA"}
numerosUnicos[80]={numero:"80",descripcion:"OCHENTA"}
numerosUnicos[90]={numero:"90",descripcion:"NOVENTA"}
numerosUnicos[100]={numero:"100",descripcion:"CIEN"}
numerosUnicos[500]={numero:"500",descripcion:"QUINIENTOS"}
numerosUnicos[700]={numero:"700",descripcion:"SETECIENTOS"}
numerosUnicos[900]={numero:"900",descripcion:"NOVECIENTOS"}
numerosUnicos[1000]={numero:"1000",descripcion:"MIL"}
numerosUnicos[1000000]={numero:"12",descripcion:"UN MILLON"}
//-----

if(numerosUnicos[n])//si el numero es unico
    numero_letra=numerosUnicos[n].descripcion
else
{
    //si el numero de cifras es mayor que 2
    switch(numCifras)
    {

        case 2://numero con dos cifras
            //desglosamos el numero de dos cifras
            numero_letra=numeroLetra(numeroString[0]*10)+" y
"+numeroLetra(numeroString[1]);
            break;//2
        case 3://numero con tres cifras(100-999)

            //se analiza casos concretos
            switch(numeroString[0])
            {
                case "1"://cuando el primer digito es 1 se nombran con
CIENTO
                    //por ejem. CIETO CINCUENTA Y DOS
                    let decenaL1=numeroString[1]+""+numeroString[2];

```

```

        let decenaN1=parseInt(decena1);//lo
pasamos a entero
                                                    //para
eliminar el cero de la izquierda

        //se da formato al numero en letras
        //y se hace una llamada recursiva a al
funcion numeroLetra()
        numero_letra="CIENTO
"+numeroLetra(decenaN1);
        break;
    case "5":
    case "7":
    case "9":
        let
cent=numerosUnicos[numeroString[0]*100].descripcion;

        let
decenaL5=numeroString[1]+""+numeroString[2];
        let decenaN5=parseInt(decenaL5);

        //se da formato al numero en letras
        //y se hace una llamada recursiva a al
funcion numeroLetra()
        numero_letra=cent+"
"+numeroLetra(decenaN5);

        break;
    default:
        let
decenaLX=numeroString[1]+""+numeroString[2];
        let decenaNX=parseInt(decenaLX);

        //se da formato al numero en letras
        //y se hacen llamadas recursivas a la
funcion numeroLetra()

numero_letra=numeroLetra(numeroString[0])+" CIENTOS
"+numeroLetra(decenaNX);
        break;

```

```

    }

    break; //3

    case 4://NUMERO CON CUATRO CIFRAS(1000-9999)
    case 5://NUMERO CON CINCO CIFRAS(10000-99999)
    case 6://NUMERO CON SEIS CIFRAS(100000-999999)

        //se da formato para completar a 6
cifras(000000)
        let numeroString4="000"+numeroString;
        let
numeroString5=numeroString4.substring(numeroString4.length-
6,numeroString4.length);

        //el numero formateado se parte en dos
bloques,
        // las tres primeras cifras(1er bloque)
y
        // las tres cifras restantes(2da parte)
        let
numeroMenor=numeroString5.substring(numeroString5.length-
3,numeroString5.length);
        let
numeroMayor=numeroString5.substring(numeroString5.length-
6,numeroString5.length-3);

        //se da formato al numero en letras
        //y se hacen llamadas recursivas a la
funcion numeroLetra()
        if(parseInt(numeroMayor)==1)
            numero_letra=" MIL
"+numeroLetra(parseInt(numeroMenor));
        else
            numero_letra=numeroLetra(parseInt(n
umeroMayor))+ " MIL "+numeroLetra(parseInt(numeroMenor));

        break;//4

```

```

case 7://NUMERO CON SIETE CIFRAS(1000000-9999999)
case 8://NUMERO CON OCHO CIFRAS(10000000-99999999)
case 9://NUMERO CON NUEVE CIFRAS(100000000-999999999)

```

```

//se da formato para completar a 9

```

```

//el numero formateado se parte en dos
bloques,
// las tres primeras cifras(1er bloque) y
// los nueve cifras restantes(2da parte)
let
numeroMenor10=numeroString11.substring(numeroString11.length
-9,numeroString11.length);
let
numeroMayor10=numeroString11.substring(0,3);

//se da formato al numero en letras
//y se hacen llamadas recursivas a la
funcion numeroLetra()
numero_letra=numeroLetra(parseInt(numeroMayo
r10))+ " MIL "+numeroLetra(parseInt(numeroMenor10));
break;//10,11,12
case 13://NUMERO CON TRECE CIFRAS(10000000000000-
99999999999999)
case 14://NUMERO CON CATORCE CIFRAS(100000000000000-
999999999999999)
case 15://NUMERO CON QUINCE CIFRAS(1000000000000000-
9999999999999999)

//se da formato para completar a 15
cifras(000000000000000)
let numeroString13="000"+numeroString;
let
numeroString14=numeroString13.substring(numeroString13.lengt
h-15,numeroString13.length);

//el numero formateado se parte en dos bloques,
// las tres primeras cifras(1er bloque) y
// los doce cifras restantes(2da parte)
let
numeroMenor13=numeroString14.substring(numeroString14.length
-12,numeroString14.length);

let numeroMayor13=numeroString14.substring(0,3);

```

```
                                //se da formato al
numero en letras
                                //y se hacen llamadas recursivas a al
funcion numeroLetra()
                                numero_letra=numeroLetra(parseInt(numero
Mayor13))+ " BILLON "+numeroLetra(parseInt(numeroMenor13));
                                break;//13,14,15
                                default:
                                numero_letra="NO IMPLEMENTADO";
                                break;//default
                                }
                                }
                                return numero_letra;//se retorna el numero en letra
                                }

const n=prompt("ingrese un numero:");
console.log(numeroLetra(n));
```

2.7. Clases y P00

2.7.1. Programación Orientado a Objetos

Es un paradigma de programación, donde la característica principal es utilizar **objetos** para representar cosas del mundo real, un conjunto de objetos compartirá características y atributos comunes para ello definimos una estructura genérica o formato que lo llamaremos “**clase**”.

2.7.2. Clases

Un conjunto de objetos que tienen las mismas características y que se comunican con los mismos mensajes podemos definirlos creando un formato genérico para su representación, a esta representación se le llama **clase**, Entonces podemos decir que una clase es la plantilla que representa a un conjunto de objetos con las mismas características.

Si hacemos una analogía con teoría de conjuntos, a un conjunto se define por comprensión o extensión, por ejemplo:

$$A = \{x \in \mathbb{N} / 3 \leq x \leq 6\} \dots \dots \dots \text{Comprensión}$$

$$A = \{3, 4, 5, 6\} \dots \dots \dots \text{Extensión}$$

En el primer caso solo se indica la característica que tendrá x, es algo genérico, ósea es la **clase** lo que definimos, pero en el segundo caso el: 3, 4, 5, 6 son objetos que tienen existencia propia, son valores que tomo x.

Las clases expresan una generalización, una abstracción, mayor que los objetos.

En la FIGURA N° 02-012 se puede observar a un esqueleto humano con sus partes, es una plantilla que toda persona tiene en su estructura interna; esto representaría una clase esqueleto humano; la actriz estadounidense: Katie Colmes, pertenece a esa estructura con sus propios valores de los atributos de la clase por lo que es la instancia de la misma(objeto).

En la FIGURA N° 02-013 se puede observar la plantilla o estructura de todo auto; cuando ya se fabrica en sí, llegan a tener existencia propia(objetos o instancia de la clase auto).

Por lo tanto, cuando a los atributos o propiedades de la estructura(**clase**) se les asigna valores, estamos instanciando(**objeto**) la clase.

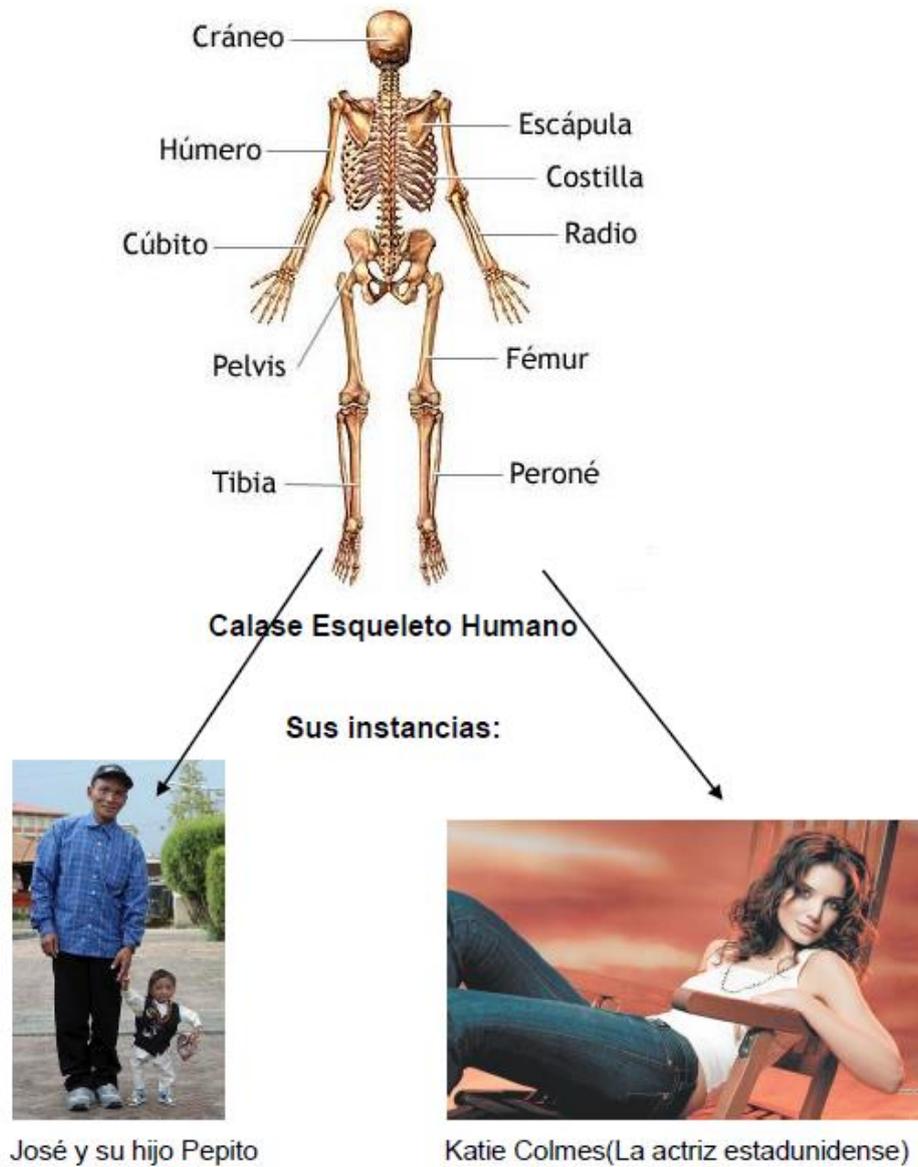


FIGURA N° 02-012: Se tiene el esqueleto humano, y tres instancias de personas con existencia propia

Clase: Auto

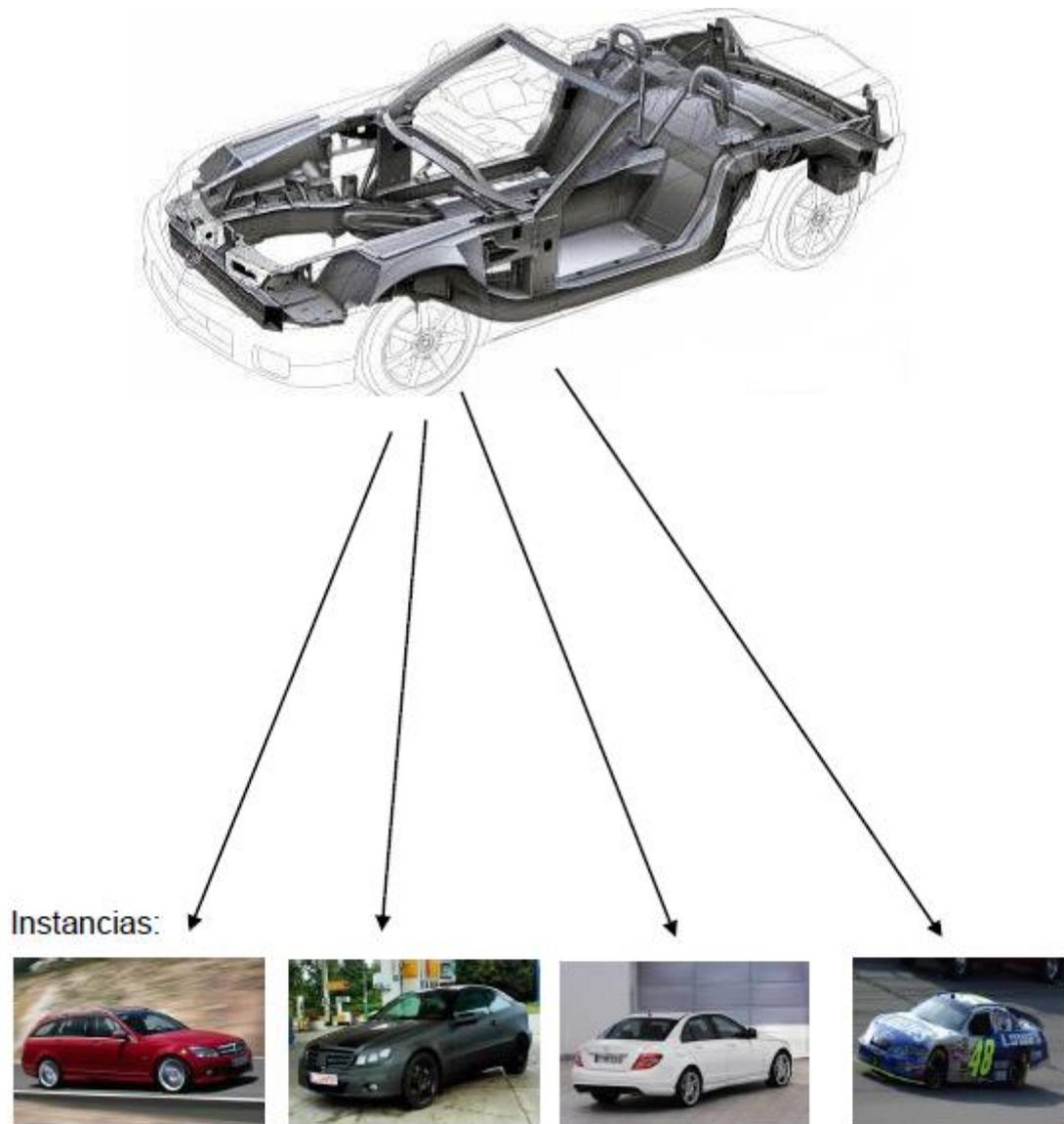


FIGURA N° 02-013: La estructura general de un auto(clase) y sus instancias(objetos)

2.7.3. Representación gráfica de una clase: notación UML

Una clase en UML se representa con un rectángulo de 4 compartimientos, así como se muestra en la FIGURA N° 02-014:



FIGURA N° 02-014: Representación de una clase en UML

Primer compartimiento:

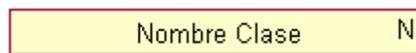


FIGURA N° 02-015: Primer compartimiento de la representación UML

En el primer compartimiento se pone el nombre de la clase y opcionalmente la multiplicidad N de la clase, en la parte superior derecho mostrado en la FIGURA N° 02-015

El nombre

El nombre de la clase debe ser algo que represente la clase, su denominación debe estar en singular, la primera letra en mayúscula y los espacios en blanco con un sub guion (“_”)

En el siguiente ejemplo de la FIGURA N° 02-016 se muestra 6 clases:

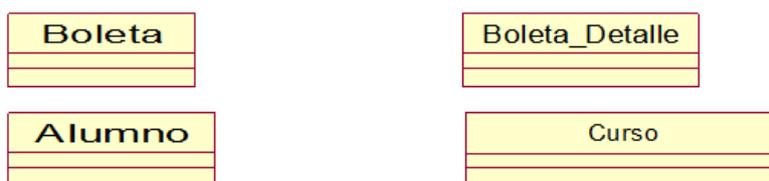


FIGURA N° 02-016: Ejemplo de clases

La multiplicidad(N) de la clase representa la cantidad de objetos que puede tener una clase

Segundo compartimiento

En el segundo compartimiento se lista los atributos que posee una clase

Atributo:

Un atributo es una propiedad o característica común a un conjunto de objetos, por ejemplo, el conjunto de objetos de la FIGURA N° 02-017 los atributos o características común a todos ellos sería: nombre, colorPluma, colorPico, colorPata, formaPico, peso.



FIGURA N° 02-017: Instancias de la clase Ave

La representación de la clase Ave en UML con sus atributos será tal como se muestra en la FIGURA N° 02-018:

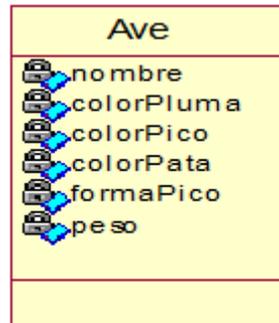


FIGURA N° 02-018: Clase Ave con su lista de atributos en el segundo compartimiento

¿Cuáles sería los atributos del conjunto de objetos de la FIGURA N° 02-013?

Solución

- ✓ modelo
- ✓ anioFabricacion
- ✓ peso
- ✓ caballoFuerza
- ✓ color
- ✓ placa
- ✓ etc.

En UML sería así como en la figura 6.9.

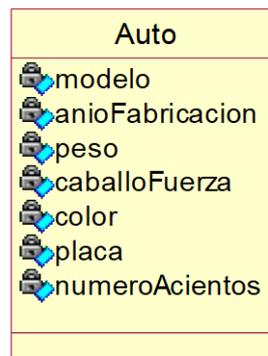


FIGURA N° 02-018: Clase Auto con su lista de atributos en el segundo compartimiento

Especificación de atributos

Un atributo se puede especificar utilizando la siguiente sintaxis:

`[Visibilidad]nombre[multiplicidad]:[tipo][= valor inicial][{cadena – de – propiedades}]`

`[Visibilidad]`: Los atributos de una clase pueden ser accesibles por:

- ✓ Todas las clases
- ✓ Las clases en las que son definidas
- ✓ Las clases en las que son definidas y por sus descendientes

De acuerdo a esto un atributo posee una característica llamado **visibilidad**.

Existe tres tipos de visibilidad:

Public(+): Si un atributo tiene visibilidad pública, indica que se puede acceder desde dentro o fuera de la clase a la que pertenece y para indicar esto se pone el “+” delante del nombre del atributo.

Private(-): Si un atributo tiene visibilidad privada, indica que se puede acceder solo desde dentro de la clase a la que pertenece, en otras palabras esto indica que solo los métodos de la clase tienen permiso para manipular al atributo.

Para indicar que un atributo es privado se pone el “-” delante del nombre del atributo.

Protected(#): Si un atributo tiene visibilidad protegida, indica que se puede acceder desde la clase a la que pertenece y además se puede acceder desde las clases descendientes y para indicar esto se pone el “#” delante del nombre del atributo

Nota. - Cuando no se indica la visibilidad del atributo, se asume que es pública

En el ejemplo que se muestra en la FIGURA N° 02-019 se tiene la representación de la visibilidad de cada atributo.

NombreClase
+atributo1
+atributo2
-atributo3
+atributo4
#atributo5

FIGURA N° 02-019: Representación de la visibilidad de los atributos de la clase

De la clase (FIGURA N° 02-019) se observa que:

- ✓ El atributo1, atributo2 y atributo4 tiene visibilidad publica
- ✓ El atributo3 tiene visibilidad privada y
- ✓ El atributo5 tiene visibilidad protegida

En la herramienta **CASE Rational Rose**, la representación de la visibilidad de cada atributo se muestra en la FIGURA N° 02-020.



FIGURA N° 02-020: Representación de los atributos de una clase en **CASE Rational Rose**

[*multiplicidad*]: Muestra la cantidad de veces que el atributo se repite; se coloca después del nombre del atributo y dentro de corchetes

En la FIGURA N° 02-020. El atributo3 es privada y tiene multiplicidad de: 0, 1, 2 ó 3.

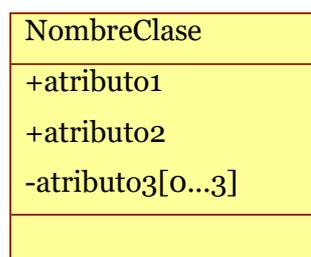


FIGURA N° 02-020: Representación de la multiplicidad de los atributos

Tipo de dato

[*tipo*]: Es el tipo de dato que tiene el atributo y puede ser cualquiera de los tipos de uso común tales como: *int*, *string*, *char*, *float*, *double*, *date*.

En la FIGURA N° 02-021 se presenta la clase persona con sus atributos: *nombre*, *DNI*, *telefono*, *direccion*: de tipo *string*, *fechaNacimiento*: de tipo *Date*

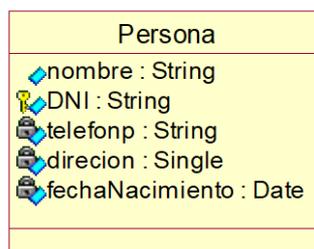


FIGURA N° 02-021: Estableciendo los tipos de datos de cada atributo

Valor inicial

[= *valor – inicial*]: Es el valor inicial por defecto que se le puede asignar a un atributo; esto es útil para no tener inconsistencia de datos, en la FIGURA N° 02-022 se muestra una clase Usuario, con sus tres compartimientos: Nombre Clase, Parámetros y operaciones.

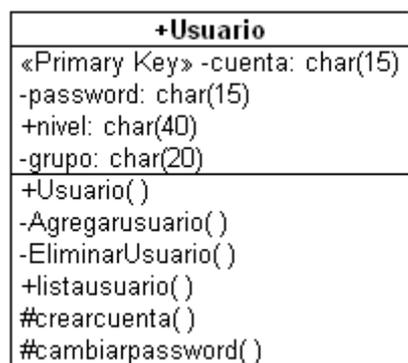


FIGURA N° 02-022: Clase Usuarios con sus atributos y operaciones

Cadena de propiedades

[{*cadena – de – propiedades*}]: es un conjunto de propiedades que indican el grado de cambio que puede sufrir los atributos, que pueden ser:

- ✓ **Changeable:** Se utiliza para indicar que no existen restricciones para modificar el atributo.
- ✓ **AddOnly:** Se usa cuando los atributos tienen una multiplicidad mayor que 1 y significa que se puede añadir más valores, pero una vez creado los valores, no puede ser removido ni alterado.

- ✓ **Frozen:** esto indica que los valores de los atributos no pueden ser cambiados una vez que el objeto se inicializo.

Tercer compartimiento

Contiene las operaciones que los objetos de una clase pueden realizar.

Operaciones

Las operaciones describen el comportamiento de un objeto de una clase, es decir, como un objeto interactúa con su entorno a estos se le denomina también métodos.

En la FIGURA N° 02-023 se puede observar la clase usuario con tres operaciones en el tercer compartimiento: *Usuario ()*, que es el constructor de la clase, *AgregarUsuario()*, *EliminarUsuario()*, entre otros.

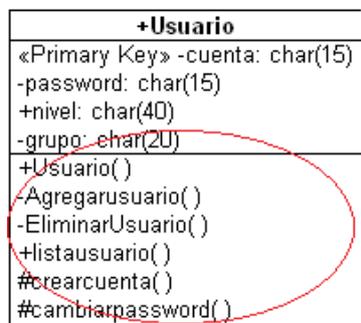


FIGURA N° 02-023: El círculo encierra los métodos de la clase Usuario

Cesar Liza Ávila nos menciona lo siguiente:

“Según UML. Una operación representa el servicio que puede ser requerido por una instancia de la clase y que afecta su comportamiento. Un método es la implementación de una operación, esto es la forma específica de cómo lleva a cabo la operación”

La sintaxis de las operaciones es similar al de los atributos:

$[Visibilidad] nombre [(lista - parametros)]: [tipo - retorno] [\{cadena - de - propiedades\}]$

Visibilidad de las operaciones

[*Visibilidad*]: Visibilidad de las operaciones

Las operaciones de una clase tienen tres tipos de visibilidad que son:

Pública (+): La operación puede ser invocada por cualquier objeto de otras clases; para indicar esto se le antepone al nombre el símbolo “+”.

Privada (-): La operación puede ser invocada solo por la clase en la que fue creada; para indicar esto se le antepone al nombre el símbolo “-”.

Protegida (#): Indica que la operación no puede ser invocada desde otra clase, pero sí desde la clase en la que fue creada y de las clases descendientes; para indicar esto se le antepone al nombre el símbolo “#”.

En la FIGURA N° 02-023 se muestra un diagrama con tres clases, las operaciones Usuario(), AgregarUsuario(), EliminarUsuario() de la clase Usuario son de visibilidad privada; la operación listarUsuario() es de visibilidad pública; las operaciones crearCuenta(), cambiarPassword() tiene visibilidad protegida, esto significa que se puede acceder desde las clases Alumno y Administrativo.

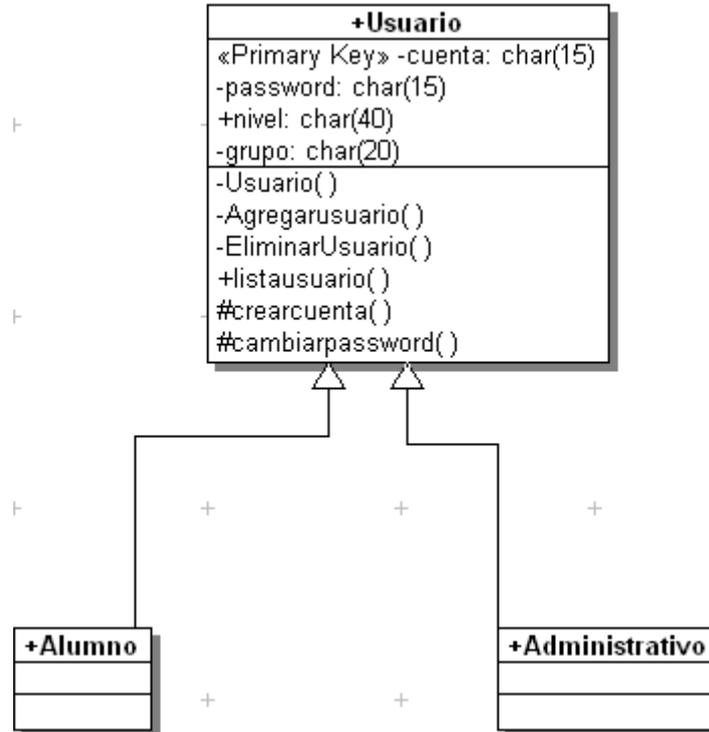


FIGURA N° 02-023: Diagrama de tres clases

Lista de parámetros

[lista – parametros]: Lista de parámetros

En la lista de parámetros se menciona las variables de entrada de la operación.

Tipo de retorno

[tipo – retorno]: Tipo de retorno

El tipo de retorno es el valor que la operación retorna

Constructores y destructores: dos operaciones especiales

Un constructor es una operación de una clase encargada de crear un objeto de esa clase. Opcionalmente, también inicializa el estado del objeto. En algunos lenguajes de programación, el constructor de una clase debe tener el mismo nombre de la clase a la que pertenece. Por ejemplo, en un programa para pintar círculos:

- ✓ el constructor de la clase *Ventana* es *Ventana()*
- ✓ el constructor de la clase *Circulo* es *Circulo()*
- ✓ y el constructor de la clase *FormularioCirculo* es *FormularioCirculo()*

Para instanciar una clase se utiliza la palabra reservada **new** seguido del *constructor()*:

```
circulo1 = new Circulo ();
```

```
circulo2 = new Circulo ();
```

Un destructor es un método que libera la memoria destruyendo el objeto. En algunos lenguajes de programación no es necesario definirlo ni invocarlos de manera explícita.

Responsabilidad de la clase

Cesar Liza Avila menciona lo siguiente:

“Todas las clases deben cumplir una labor y esto es la responsabilidad de la clase.

Una responsabilidad es un contrato u obligación que la clase debe cumplir, pues viene a ser el fin para la cual fue creada. Los atributos y comportamientos son las características de la clase que le permite cumplir con esas responsabilidades”.

Las responsabilidades de la clase se mencionan en un **cuarto compartimiento**, así como se muestra en la FIGURA N° 02-024.

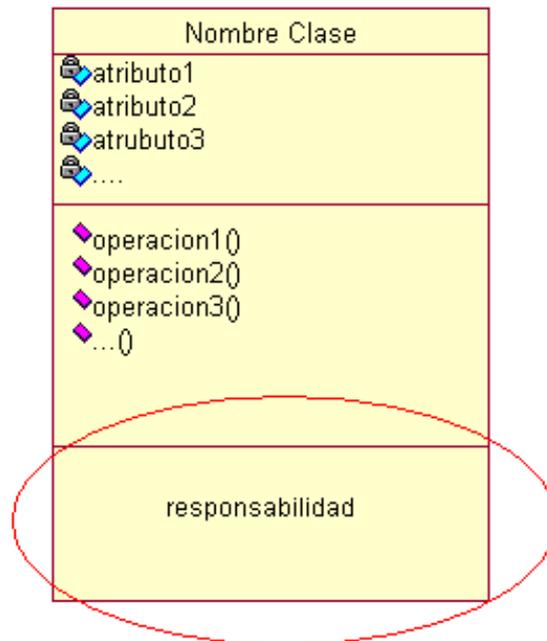


FIGURA N° 02-024: La responsabilidad de la clase se describe en el cuarto compartimiento de la representación gráfica UML

2.8. Clases en JavaScript

2.8.1. Prototipos

Cuando utilizamos objetos para representar cosas del mundo real para estructurar datos, por ejemplo, objetos **alumno**, definimos así:

```
const alumno1={
  codigo:"0000000001",
  nombre:"MARIA",
  apellidos:" SANTILLAN HUAMAN",
  correo:"maria@gmail.com"
}
const alumno2={
  codigo:"0000000002",
  nombre:"JESUS",
  apellidos:"GONZALES CONTRERAS",
  correo:"jesus@gmail.com"
}
const alumno3={
  codigo:"0000000003",
  nombre:"MARIO",
  apellidos:"DAYER COTRINA",
  correo:"mario@gmail.com"
}
```

Se puede seguir definiendo más alumnos, pero se hace inmanejable, este inconveniente se solucionaría utilizando Arrays, otra forma de solucionar sería creando un **prototipo** que represente al conjunto de objetos de alumnos con sus atributos.

Los prototipos son estructuras mediante el cual los objetos heredan atributos o características comunes entre sí. Su implementación en JavaScript se realiza con funciones constructoras.

En el ejemplo anterior definimos un prototipo con la función constructora **alumno()**, que recibe como parámetros atributos del objeto alumno, así mismo las propiedades *saludar()* y *getDatos()* dentro de la función. Para hacer referencia a los atributos, propiedades se utiliza **this**.

```
function Alumno(codigo,nombre,apellido,correo){
  this.codigo=codigo;
  this.nombre=nombre;
  this.apellido=apellido;
  this.correo=correo

  this.saludar=function(){
    console.log("Hola: ");
  }

  this.getDatos=function(){
    return this.codigo+" "+this.nombre+" "+this.apellido;
  }
}

//se instancia el prototipo
const a1=new Alumno("0000000001",
                    "JOSE",
                    "SANTAMARIA ESPIRITU",
                    "jose@gmail.com");

console.log(a1);
console.log(a1.getDatos());
```

Para instanciar un prototipo utilizamos la palabra reservada **new** seguido del La función en este caso Alumno () con sus parametros, Las propiedades **saludar()** y **getDatos()** que están dentro del prototipo, se pueden definir fuera de ello y posteriormente asignarlo al prototipo **Alumno**.

Alumno.prototype.saludar()

Alumno.prototype.getDatos()

```

//-----
//prototipo Alumno
function Alumno(codigo,nombre,apellido,correo){
  this.codigo=codigo;
  this.nombre=nombre;
  this.apellido=apellido;
  this.correo=correo
}

//La propiedad saludar lo asignamos al prototipo Alumno
Alumno.prototype.saludar=function(){
  console.log("Hola: ");
}

//la propiedad getDate lo asignamos al prototipo Alumno
Alumno.prototype.getDatos=function(){
  return this.codigo+" "+this.nombre+" "+this.apellido;
}

//se instancia el prototipo
const a1=new Alumno("0000000001",
                    "JOSE",
                    "SANTAMARIA ESPIRITU",
                    "jose@gmail.com");

a1.saludar();
console.log(a1.getDatos());

```

2.8.2. Clases en JavaScript

Las clases en JavaScript son una mejora en la sintaxis de representar objetos con prototipos, fue introducida en ECMAScript 2015.

Clases Declaradas

Para definir una clase declarada se utiliza la palabra reservada **class** seguida de un nombre de la siguiente manera:

```

class nombreClase{
  constructor(p1,p2,...) {
    this.p1=p1;
    this.p2=p2;
    // otros atributos
  }

  // otros métodos
}

```

Clases Expresadas

Es otra forma de definir una clase. Las expresiones de la clase pueden ser nombradas o anónimas:

Clase anónima:

```

const nombreConstante=class{

  constructor(p1,p2,...) {
    this.p1=p1;
    this.p2Op2;
    // otros atributos
  }

  // otros métodos
}

```

Clase nombrada:

```

const nombreConst=class nombreClase{
  constructor(p1,p2,...) {
    this.p1=p1;
    this.p2Op2;
    // otros atributos
  }

  // otros métodos
}

```

Herencia

Para crear clases que heredan atributos y métodos de otra clase, utilizamos la palabra reservada “**extends**” seguido del nombre la clase padre, el método **super()** en el constructor permite tener acceso a las propiedades heredadas, en el siguiente ejemplo se tiene una clase “**Animal**” con atributos: nombre, especie, numeroPatas, habitat. Otra clase “**Ave**” con sus atributos: colorPluma, colorPico, que además heredara los atributos de la clase Animal.

```
//clase Animal
class Animal{
  constructor(nombre, especie, numeroPatas,habitat)
  {
    this.nombre=nombre;
    this.especie=especie;
    this.numeroPatas=numeroPatas;
    this.habitat=habitat;
  }
  //gette
  get dato(){
    return "Nombre: "+this.nombre+", especie: "+this.especie
  }
  //metodo
  getNombre()
  {
    return this.nombre;
  }
}
```

```

//-----
console.log("CLASE ANIMAL");
//-----
//se instancia la clase Animal
const a=new Animal("Perro","Mamifero","4","vivienda");
console.log(a.dato);//se llama a dato sin parentesis
console.log(a.getNombre());//se llama a getNombre con
                          // parentesis
//-----
//clase Ave que hereda de la clase animal
class Ave extends Animal{
  constructor(colorPluma,colorPico){
    super();
    this.colorPluma=colorPluma;
    this.colorPico=colorPico;

  }

  get colorPicoAve(){
    return this.colorPico;
  }

  get colorPlumaAve(){
    return this.colorPluma;
  }
}
//-----
console.log("clase Ave que hereda de la clase Animal");

//se instancia la clase Ave
const gallina=new Ave("negro","amarilo");
//-----
//se asigna valores a los atributos heredados
//de la clase Animal
gallina.nombre="gallina";
gallina.especie="Aves";
gallina.numeroPatas=2;
gallina.habitat="corral"

//se imprime en consola el objeto gallina
console.log(gallina);

```

2.9. Manejo de DOM y eventos

BOM (Browser Object Model)

Es el modelo de objetos del navegador, que permite interactuar JavaScript con el navegador, para su acceso utilizamos la palabra reservada “*window*.”

Algunas propiedades y método:

`window.innerHeight`- la altura interior de la ventana del navegador (en píxeles)

`window.innerWidth`- el ancho interior de la ventana del navegador (en píxeles)

`window.open()`- abrir una nueva ventana

`window.close()`- cerrar la ventana actual

`window.moveTo()`- mover la ventana actual

`window.resizeTo()`- cambiar el tamaño de la ventana actual

DOM (Document Object Model)

Es la interfaz que permite interactuar documento HTML, CSS desde JavaScript en tiempo de diseño y ejecución. Se puede acceder a cualquier elemento a través de la palabra reservada “*document*.”

Los elementos de un documento HTML están estructurados como un árbol donde cada uno de ellos es un *nodo* que representa una rama o hoja

Nodo

Es cada elemento del DOM, desde el document hasta un elemento. Es un objeto en sí.

Para acceder a un nodo padre que contiene al *nodo* utilizamos el *parentNode*, para acceder a todos los nodos hijos se utiliza *children*

Selectores

- **Selectores por identificador o id**

document.getElementById(id)

- **Selectores por clases**

document.getElementsByClassName(nombreClase)

- **Selectores por tags**

Selector que permite seleccionar por etiqueta HTML

document.getElementsByTagName(nombreEtiqueta)

- **Selectores por nombre**

document.getElementsByName(nombreElemento)

- **Selector general**

Selecciona una o varios elementos que coincida con elemento, puede ser: **id**, **name**, **class**, etiqueta, atributo, entre otros.

- ✓ ***document.querySelector(elemento)***

Selecciona un solo elemento que coincida con elemento

- ✓ ***document.querySelectorAll(elemento)***

Selecciona todos los elementos que coincida con elemento

Creando y removiendo elementos al document

Para crear elementos utilizamos el: ***document.createElement(elemento)***, para crear un texto se utiliza ***document.createTextNode(texto)***, para remover un elemento utilizamos ***elementoEliminar.remove()***;

Se puede clonar a cualquier elemento del ***document*** con ***.cloneNode(true)*** y hacer copias

Siempre es bueno ver si el elemento o nodo existe en nuestro ***document*** para ello el ***elemento.isConnected*** permite verificar, retornando verdadero o falso dependiendo si es que se encuentra en el ***document***

Insertando elementos al document

El método ***elemento.appendChild(nuevoElemento)*** permite insertar en el nodo padre ***elemento***.

El método ***elemento.insertAdjacentElement(posicion ,elementoNuevo)*** inserta un elemento nodo dado en una posición dada con respecto al elemento sobre el que se invoca.

Sintaxis:

elemento.insertAdjacentElement(posicion ,elementoNuevo)

Parametro: posición

Es una cadena que indica el lugar en donde se insertara el ***elementoNuevo*** al ***elemento*** padre: siendo uno de los siguientes valores:

'beforebegin' : Antes del *elemento*.

'afterbegin' : Dentro del *elemento*, antes de su primer hijo.

'beforeend' : Dentro del *elemento*, después de su último hijo.

'afterend' : Después del *elemento*.

Posición en donde se insertará el *nuevoElemento*:

```
<!-- beforebegin -->
```

```
<elemento>
```

```
<!-- afterbegin -->
```

```
<otrosElementos>
```

```
<!-- beforeend -->
```

```
</elemento>
```

```
<!-- afterend -->
```

El HTML del siguiente ejemplo en *index.html*, contiene un formulario con la etiqueta `<form>`:

```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7  |   <title>Document</title>
8  </head>
9  <body>
10 |   <form action="" style="background-color: #gold;">
11 |     <button>HOLA</button>
12 |   </form>
13 |   <script src="app.js"></script>
14 </body>
15 </html>
```

Desde JavaScript se crea y se inserta los elementos, antes de `<form>`, antes del `<button>`, después de `</button>`, después de `</form>`.

Para ello desde JavaScript se accede a la etiqueta `<form>`, luego se crean los elementos con la etiqueta `<h1>` con diferentes textos, luego se agrega según la posición requerida.

Código JavaScript del archivo *app.js*:

```

appjs > ...
1  const form=document.querySelector("form");
2
3  //se crea elemento
4  const elemnetoNuevo1=document.createElement("h1");
5  elemnetoNuevo1.textContent="Antes del form" ;
6
7  //se crea elemento
8  const elemnetoNuevo2=document.createElement("h1");
9  elemnetoNuevo2.textContent="Antes del 1er hijo de form" ;
10
11 //se crea elemento
12 const elemnetoNuevo3=document.createElement("h1");
13 elemnetoNuevo3.textContent="Despues del ultimo hijo de form" ;
14
15 //se crea elemento
16 const elemnetoNuevo4=document.createElement("h1");
17 elemnetoNuevo4.textContent="Despues de form" ;
18
19 //se inserta los elementos
20 form.insertAdjacentElement('beforebegin',elemnetoNuevo1);
21 form.insertAdjacentElement('afterbegin',elemnetoNuevo2);
22 form.insertAdjacentElement('beforeend',elemnetoNuevo3);
23 form.insertAdjacentElement('afterend',elemnetoNuevo4);

```

Como resultado se puede observar en la FIGURA N° 02-025 el código HTML modificado y su visualización en el navegador, tal como se muestra en la FIGURA N° 02-026

```

<!DOCTYPE html> == $0
<html lang="en">
  <head> ... </head>
  <body>
    <h1>Antes del form</h1>
    <form action style="background-color: gold;">
      <h1>Antes del 1er hijo de form</h1>
      <button>HOLA</button>
      <h1>Despues del ultimo hijo de form</h1>
    </form>
    <h1>Despues de form</h1>
    <script src="app.js"></script>
    <!-- Code injected by live-server -->
    <script> ... </script>
  </body>
</html>

```

FIGURA N° 02-025: HTML modificado desde JavaScript con los nuevos elementos insertados

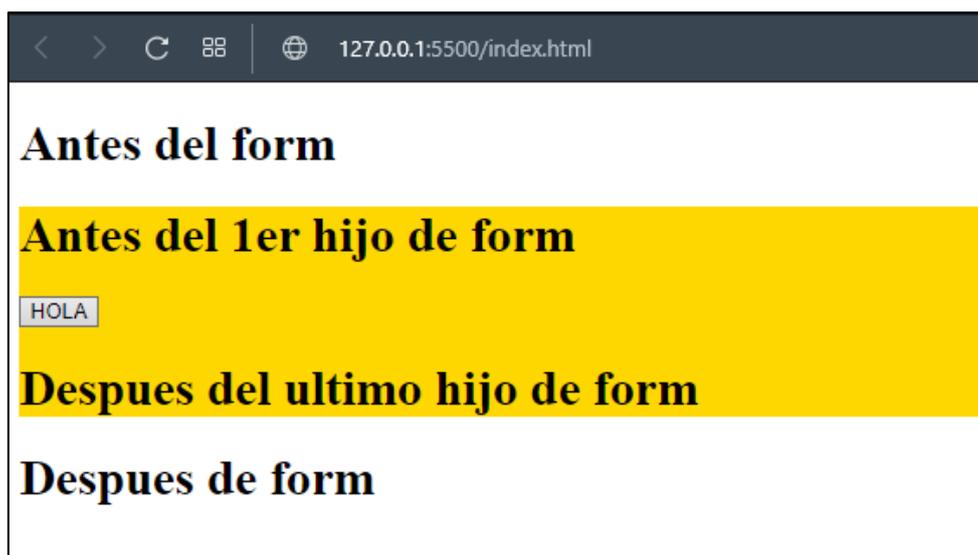


FIGURA N° 02-026: Resultado de insertar nuevos elementos al HTML

elemento.innerHTML=cadenaHTML

Permite reemplazar todo el contenido de **elemento** con el HTML que asignamos en ***cadenaHTML***

Eventos

Los eventos son acciones sobre los objetos del DOM, cada elemento puede tener asociado varios eventos que pueden ocurrir sobre ellos, se utiliza el metodo *.addEventListener(evento,función())* para asociar a un elemento

elemento.addEventListener(evento,función())

Este método permite añadir una escucha del evento indicado (primer parámetro), y en el caso que ocurra, se ejecutará la función asociada indicada (segundo parámetro). De forma opcional, se le puede pasar un tercer parámetro con ciertas opciones:

```
elemento.addEventListener("DOMContentLoaded",function(){
    //bloque de código que se ejecutara cuando sucede el evento
});
```

A continuación, se describe algunos eventos:

DOMContentLoaded

Este evento se desata cuando la página ha terminado de cargarse.

click (onclick)

Se produce cuando se da una pulsación o clic al botón del ratón sobre un elemento de la página, generalmente un botón o un enlace.

DblClick (ondblclick)

Este evento es lanzado cuando el usuario hace doble click en un elemento de formulario o un link.

change (onchange)

Se desata este evento cuando cambia el estado de un elemento de formulario, en ocasiones no se produce hasta que el usuario retira el foco de la aplicación del elemento.

focus (onfocus)

El evento onfocus es lo contrario de onblur. Se produce cuando un elemento de la página o la ventana ganan el foco de la aplicación.

keydown (onkeydown)

Este evento se produce en el instante que un usuario presiona una tecla, independientemente que la suelte o no. Se produce en el momento de la pulsación.

keypress (onkeypress)

Ocurre un evento onkeypress cuando el usuario deja pulsada una tecla un tiempo determinado. Antes de este evento se produce un onkeydown en el momento que se pulsa la tecla.

keyup (onkeyup)

Se produce cuando el usuario deja de apretar una tecla. Se produce en el momento que se libera la tecla.

mousedown (onmousedown)

Se produce el evento onmousedown cuando el usuario pulsa sobre un elemento de la página. onmousedown se produce en el momento de pulsar el botón, se suelte o no.

mousemove (onmousemove)

Se produce cuando el ratón se mueve por la página.

mouseout (onmouseout)

Se desata un evento onmouseout cuando el puntero del ratón sale del área ocupada por un elemento de la página.

mouseover (onmouseover)

Este evento se desata cuando el puntero del ratón entra en el área ocupada por un elemento de la página.

mouseup (onmouseup)

Este evento se produce en el momento que el usuario suelta el botón del ratón, que previamente había pulsado.

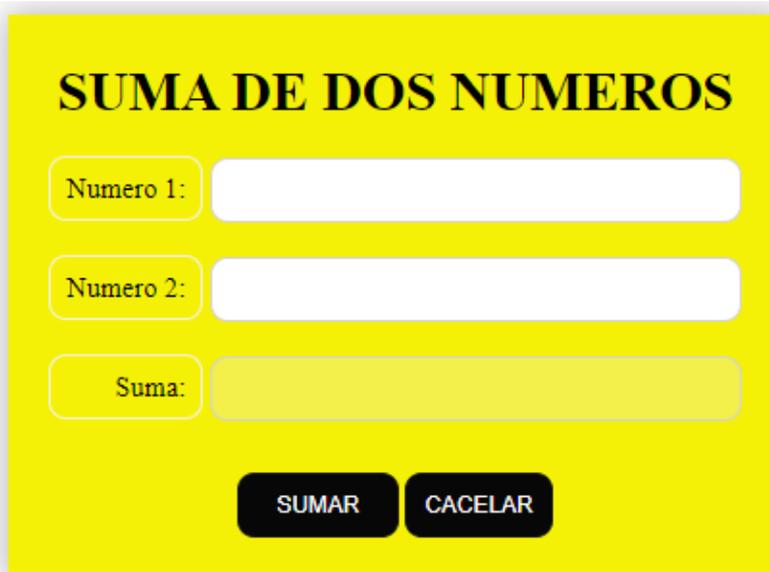
submit (onsubmit)

Ocurre cuando el visitante aprieta sobre el botón de enviar el formulario.

scroll (onscroll)

Este evento se produce cuando se realiza scroll o desplazamiento. Este desplazamiento se puede realizar en la página completa, pero también en elementos de la página que tengan sus propias zonas de scroll.

Ejemplo 02-006: Realizar un formulario para calcular la suma de dos números con el siguiente diseño:



El formulario tiene un fondo amarillo brillante. En la parte superior, el título "SUMA DE DOS NUMEROS" está escrito en letras negras, grandes y en negrita. Debajo del título, hay tres filas de entrada de texto. La primera fila tiene el label "Numero 1:" a la izquierda de un campo de entrada blanco. La segunda fila tiene el label "Numero 2:" a la izquierda de otro campo de entrada blanco. La tercera fila tiene el label "Suma:" a la izquierda de un campo de entrada blanco. En la parte inferior del formulario, hay dos botones rectangulares con esquinas redondeadas y fondo negro. El botón de la izquierda tiene el texto "SUMAR" en blanco, y el botón de la derecha tiene el texto "CACELAR" en blanco.

Ejemplo 02-007: Realizar un formulario para que calcule las 4 operaciones de: suma, resta, multiplicación y división con el siguiente diseño:



CUATRO OPERACIONES

Numero 1:

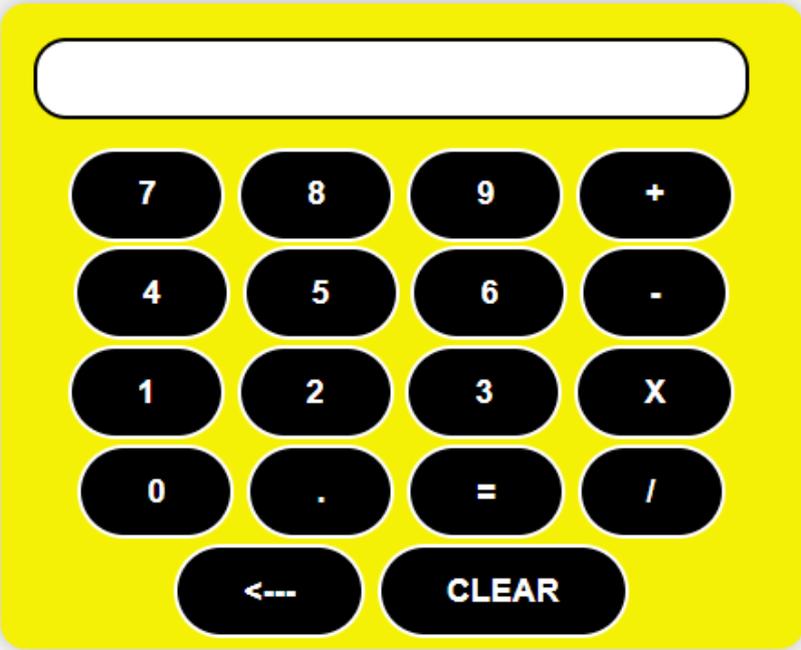
Numero 2:

Operacion:

Resultado:

EJECUTAR LIMPIAR

Ejemplo 02-008: Realizar una calculadora con las cuatro operaciones de: Suma, Resta, Multiplicación y División con el siguiente diseño:



Calculadora con botones de operación y números:

7 8 9 +

4 5 6 -

1 2 3 X

0 . = /

<--- CLEAR

Ejemplo 02-009: Realizar un programa que permita realizar el juego de tres en raya entre una persona y la computadora con el siguiente diseño:



2.10. API fetch

2.10.1. Interfaces de programación de Aplicaciones-API

Son módulos o un conjunto de librería ya desarrolladas en algún lenguaje de programación para hacer algo específico y facilitar a los desarrolladores crear funcionalidades complejas de manera sencilla.

Algunas APIs más comunes:

- ✓ APIs para manipular documentos. - API DOM (Document Object Model), que permite interactuar y manipular elementos del HTML y CSS de una página web.
- ✓ Las API para obtener datos de servidores de base de datos en formato JSON.
- ✓ Las API para dibujar y manipular gráficos
- ✓ Las API de dispositivos
- ✓ APIS de audio y vídeo
- ✓ Las APIs que permiten manipular datos de servidores de base de datos.

Algunas APIs que proporcionan datos en formato JSON:

- API que retorna lista de usuario de manera aleatoria
<https://randomuser.me/api/?results=10>
- API de Mercado Libre que permite listar ítems pasándole a “q” la descripción:
<https://api.mercadolibre.com/sites/MLA/search?q=computadora>
- API REST de GitHub
<https://docs.github.com/es/rest?apiVersion=2022-11-28>
- PokéAPI.- Información de los Pokémon como sus movimientos, habilidades, tipos, poderes, habitad, etc.
<https://pokeapi.co/>
- COVID Tracking.- Datos del COVID-19 de todo el mundo como número de contagios, pruebas, pacientes hospitalizados, fallecidos, etc.
<https://covidtracking.com/data>
- OpenWeather APIs.- Información del clima, como los datos meteorológicos del día y de los años anteriores, previsión climática, radiación solar, e incluso, colección de mapas.
<https://openweathermap.org/api>
- JSON placeholder.- API que proporciona datos ficticios como fotos, publicaciones, comentarios, datos de usuarios falsos, rutas entre otros.
<https://jsonplaceholder.typicode.com/>

2.11. Manejo de APIs en JavaScript

Para consumir datos desde APIs con JavaScript utilizaremos el *API fetch* que permite de manera sencilla acceder a datos en formato JSON.

Este API es una interfaz que permite a JavaScript acceder y manipular peticiones HTTP asincronas, es un estándar para realizar solicitudes de servidor con promesas y que permite obtener datos de la url especificado como parámetro. El método global que proporciona es el `fetch(url)`.

Sintaxis:

```
fetch(url, options)
  .then(function(response) {
    //codigo
  })
  .catch(function(error) {

    //codigo console.log(error.message);
  });
```

El `fetch()` devuelve una promesa que es aceptada si recibe una respuesta “ok”. El manejo de las promesas se realiza con el método `.then(function(response))`, tal como se muestra en la sintaxis del `fetch()`. La función como argumento es un *callback* que tiene como parámetro el objeto de respuesta “**response**”, de la petición realizado.

El segundo método de `fetch()` es un objeto y es opcional, podría tener la siguiente estructura:

```

const options={
  method: 'POST', // *GET, POST, PUT, DELETE, etc.,
  mode: 'cors', // no-cors, *cors, same-origin
  cache: 'no-cache', // *default, no-cache, reload,
                    //force-cache, only-if-cached
  credentials: 'same-origin', // include, *same-origin, omit
  headers: {
    'Content-Type': 'application/json'
    // 'Content-Type': 'application/x-www-form-urlencoded',
  },
  redirect: 'follow', // manual, *follow, error
  refererPolicy: 'no-referrer', // no-referrer,
                    // *no-referrer-when-downgrade,
                    // origin, origin-when-cross-origin,
                    //same-origin, strict-origin,
                    //strict-origin-when-cross-origin,
                    // unsafe-url

  body: JSON.stringify(data) // body data type must
                             // match "Content-Type" header
}

```

Del objeto *options*, el primer atributo es el método HTTP a utilizar en la petición, por defecto es “**GET**”, que podría ser también: **POST**, **PUT**, **HEAD**, entre otros. En el *body* se puede indicar un *objeto* para enviar, en le *headers* se puede indicar para modificar las cabeceras, en el atributo *credentials* se indica el modo en que se realizara la petición por defecto tiene el valor de “*omit*” y esto hace que no se incluya credenciales en la petición.

En el siguiente ejemplo accederemos a los datos del API que proporciona Mercado Libre descrito anteriormente:

<https://api.mercadolibre.com/sites/MLA/search?q=computadora>

```

//url para consumir datos
const url="https://api.mercadolibre.com/sites/MLA/search?q=computadora"
fetch(url)
  .then(response =>response.json())
  .then(data=>console.log(data))
  .catch(function(error) {
    console.log(error.message);
  });

```

El resultado impreso en consola de los datos retornados se puede apreciar en la FIGURA N° 02-027.

```

app.js:6
{site_id: 'MLA', country_default_time_zone: 'GMT-03:00', query: 'computadora', paging: {...}, results:
Array(50), ...}
  ▶ available_filters: (20) [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
  ▶ available_sorts: (2) [{}], [{}]
  country_default_time_zone: "GMT-03:00"
  ▶ filters: [{}]
  ▶ paging: {total: 187335, primary_results: 1000, offset: 0, limit: 50}
  query: "computadora"
  ▶ results: Array(50)
    ▶ 0: {id: 'MLA1399231360', title: 'Silla De Escritorio Vonne Sv-g0 Gamer Ergonómica Negra Y Blanca'
    ▶ 1: {id: 'MLA1408690360', title: 'Computadoras Notebook Laptop Baratas Ssd 256gb 6gb Windows', conc
    ▶ 2: {id: 'MLA901119287', title: 'Pc Gamer Ryzen 3 Pro + 4 Gb Ddr4 +ssd 120 + Gabinete Kit', conditi
    ▶ 3: {id: 'MLA1357167365', title: 'Laptop Intel Core I3 2 En 1 Tablet Pc 13.3 = Mac Air', conditior
    ▶ 4: {id: 'MLA1355799799', title: 'Cable Displayport A Hdmi 1.8 Metros Display Port 4k', condition:
    ▶ 5: {id: 'MLA932293921', title: 'Pc Armada Gamer Amd Ryzen 5 5600g 12 Nucleo Ram 16gb Ssd 480', cor
    ▶ 6: {id: 'MLA1327636549', title: 'Disco Sólido Interno Kingston Sa400s37/480g 480gb Negro', conditi
    ▶ 7: {id: 'MLA874813959', title: 'Pc Armada Intel C I3 + 8 Gb Ssd 240 Wi Fi Nueva W10 Office', conc
    ▶ 8: {id: 'MLA1106754666', title: 'Pc Cpu Computadora Intel I3 Hd 1tb 8gb Oferta Gtia Cuotas ', cor
    ▶ 9: {id: 'MLA1213934809', title: 'Cable Displayport A Hdmi 4k Adaptador Pc Ultra V2.0 3 Metros', cc
    ▶ 10: {id: 'MLA806660789', title: 'Pc Computadora Amd 3.5ghz Completa + Monitor 19 Nueva Cuotas', cc
    ▶ 11: {id: 'MLA881334859', title: 'Teclado Silicona Macbook Pro 13 A1706 A1708 A2159', condition: 'r
    ▶ 12: {id: 'MLA1403734136', title: 'Cpu Dell Optiplex 780 Core 2 Duo - Ram 4 Gb - Disco 250 Gb', cor
    ▶ 13: {id: 'MLA806868836', title: 'Pc Computadora Completa Intel I3 C/monitor Led 19 Cuotas S/i', cc
    ▶ 14: {id: 'MLA870679115', title: 'Woox - Atril Base De Diseño Apoya Notebook Computadora', conditi
    ▶ 15: {id: 'MLA1212783041', title: 'Cable Displayport A Hdmi 4k Adaptador Pc Ultra V2.0 1.8 Mts', cc
    ▶ 16: {id: 'MLA874819289', title: 'Pc Armada Intel Core I5 8gb Ssd240 W10 Office', condition: 'new',

```

FIGURA N° 02-027: Datos obtenidos con *fetch* desde el API de Mercado Libre

CAPÍTULO III

Desarrollo de Soluciones frontend con JavaScript



JavaScript

JavaScript es un lenguaje de programación interpretado y de alto nivel, ampliamente utilizado para crear páginas web dinámicas e interactivas. Es un lenguaje del lado del cliente, lo que significa que se ejecuta en el navegador web del usuario, aunque también se puede utilizar del lado del servidor con plataformas como Node.js.

Características principales de JavaScript:

1. **Interactividad:** Permite a los desarrolladores agregar comportamientos interactivos a las páginas web, como formularios interactivos, juegos, animaciones y actualizaciones en tiempo real.
2. **Lenguaje de Script:** Es un lenguaje de script, lo que significa que no necesita ser compilado antes de ejecutarse. Los navegadores web interpretan el código JavaScript directamente.
3. **Orientado a Objetos:** Aunque JavaScript no es un lenguaje de programación orientado a objetos en el sentido tradicional, permite la creación y manipulación de objetos, lo que facilita la reutilización del código y la organización del mismo.
4. **Lado del Cliente y Servidor:** Originalmente diseñado para ser ejecutado en el navegador del usuario (lado del cliente), JavaScript también se puede utilizar en el lado del servidor con tecnologías como Node.js, permitiendo construir aplicaciones web completas con un solo lenguaje de programación.
5. **Versatilidad:** Se puede utilizar para una amplia variedad de tareas, desde la validación de formularios hasta la creación de aplicaciones web completas y móviles.
6. **Integración con HTML y CSS:** JavaScript se integra estrechamente con HTML y CSS, los lenguajes de marcado y estilo utilizados para crear la estructura y apariencia de las páginas web, respectivamente.

3.1. Desarrollo de Soluciones

Frontend con JavaScript

Se refiere a la creación y mejora de la interfaz de usuario de una aplicación web utilizando JavaScript. Esto incluye la interacción del usuario con la página web, la manipulación del DOM (Document Object Model), la actualización de contenidos de manera dinámica, y la comunicación con servidores para obtener y enviar datos.

Elementos Clave del Desarrollo Frontend con JavaScript:

- ✓ **Manipulación del DOM:** JavaScript permite acceder y manipular los elementos HTML y CSS de una página web, lo que es fundamental para crear una interfaz de usuario interactiva.
- ✓ **Eventos del Usuario:** Permite capturar y responder a eventos del usuario, como clics, desplazamientos, entrada de datos, entre otros.
- ✓ **AJAX (Asynchronous JavaScript and XML):** Facilita la comunicación asíncrona con el servidor para actualizar partes de una página web sin necesidad de recargarla por completo.
- ✓ **Frameworks y Librerías:** Existen numerosas herramientas como React, Angular y Vue.js que simplifican y estructuran el desarrollo frontend.

Ejemplos de Desarrollo Frontend con JavaScript:

1. Manipulación del DOM:

En este ejemplo, al hacer clic en el botón, el texto del div con id "mensaje" se cambia dinámicamente.



```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Manipulación del DOM</title>
</head>
<body>
  <div id="mensaje">Hola, mundo!</div>
  <button id="cambiarTexto">Cambiar Texto</button>

  <script>
    document.addEventListener('DOMContentLoaded', (event) => {
      document.getElementById('cambiarTexto').addEventListener('click', function() {
        document.getElementById('mensaje').innerText = '¡Texto cambiado!';
      });
    });
  </script>
</body>
</html>

```

2. Eventos del Usuario:

Este ejemplo muestra cómo se puede capturar la entrada del usuario en tiempo real y mostrarla en un párrafo.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Eventos del Usuario</title>
</head>
<body>
  <input type="text" id="entrada" placeholder="Escribe algo">
  <p id="salida"></p>

  <script>
    document.getElementById('entrada').addEventListener('input', function(event) {
      document.getElementById('salida').innerText = event.target.value;
    });
  </script>
</body>
</html>

```

Primer programa

Primer programa

3. AJAX para Comunicación Asíncronica:

Aquí, al hacer clic en el botón, se envía una solicitud a un servidor y se muestra el contenido recibido en el div.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AJAX</title>
</head>
<body>
  <button id="cargarDatos">Cargar Datos</button>
  <div id="datos"></div>

  <script>
    document.getElementById('cargarDatos').addEventListener('click', function() {
      const xhr = new XMLHttpRequest();
      xhr.open('GET', 'https://jsonplaceholder.typicode.com/posts/1', true);
      xhr.onload = function() {
        if (this.status === 200) {
          const post = JSON.parse(this.responseText);
          document.getElementById('datos').innerText = `Título: ${post.title}\nContenido: ${post.body}`;
        }
      };
      xhr.send();
    });
  </script>
</body>
</html>

```

Cargar Datos

Título: sunt aut facere repellat provident occaecati excepturi optio reprehenderit
 Contenido: quia et suscipit
 suscipit recusandae consequuntur expedita et cum
 reprehenderit molestiae ut ut quas totam
 nostrum rerum est autem sunt rem eveniet architecto

3.2. Casos de Soluciones

Frontend con JavaScript

Construcción de una Página Web de Venta de Automóviles

El Proyecto es desarrollar una página web para la venta de automóviles de diferentes marcas. La página debe ser atractiva, fácil de navegar y permitir a los usuarios visualizar información sobre diversas marcas y modelos de automóviles. Además, debe incluir funcionalidades interactivas que mejoren la experiencia del usuario.

Requisitos No Funcionales

a. Diseño Responsivo

La página debe ser accesible y usable en dispositivos móviles, tabletas y computadoras de escritorio.

b. Estilo y Apariencia

Utilizar CSS para crear un diseño atractivo y coherente.

c. Rendimiento

Optimizar las imágenes y el código para asegurar tiempos de carga rápidos.

Tecnologías Utilizadas

HTML: Para estructurar el contenido de la página web.

CSS: Para el diseño y la apariencia visual de la página.

JavaScript: Para añadir interactividad y manipulación dinámica del contenido.

Pasos para la Implementación

a. Planificación

Definir el alcance del proyecto y los requisitos específicos.

b. Desarrollo

Estructura HTML: Crear la estructura básica de las páginas con HTML.

Estilo CSS: Aplicar estilos y diseños usando CSS.

Interactividad JavaScript: Añadir funcionalidad interactiva con JavaScript.

c. Pruebas

Probar la página web en diferentes navegadores y dispositivos para asegurar la compatibilidad.

Realizar pruebas de usabilidad para asegurar que la página sea fácil de navegar.

d. Lanzamiento

Subir la página web a un servidor y asegurar que esté disponible públicamente

Codificación en CSS

```

VENTA DE AUTOS > css > # estilo1.css > .product-stripe
1  body{
2  |   margin:0cm
3  | }
4  |
5  | .cover {
6  | |   height: 200px;
7  | |   background-image:url(https://www.rch.com.ar/graficos/banner.jpg);
8  | |   color: white;
9  | |   background-size: cover;
10 | |   background-position: center;
11 | }
12 |
13 | .cover-small {
14 | |   height: 200px;
15 | }
16 |
17 | .card {
18 | |   border: 0;
19 | }
20 |
21 | .card-title {
22 | |   min-height: 3rem;
23 | }
24 |
25 | .card-text {
26 | |   min-height: 5rem;
27 | }
28 |
29 | .product-stripe {
30 | |   overflow-x: scroll;
31 | |   padding-top: 1rem;
32 | }

```

```

34  .stripe-container {
35      display: flex;
36      flex-direction: row;
37      padding-left: 1rem;
38  }
39  }
40
41  .stripe-container .card {
42      width: 300px;
43      flex-shrink: 0;
44      margin-right: 1rem;
45  }
46
47  .responsive-iframe {
48      position: relative;
49      padding-top: 56.25%;
50  }
51
52  .responsive-iframe iframe{
53      position: absolute;
54      top: 1rem;
55      left: 0;
56      width: 100%;
57      height: 100%;
58  }
59  h2{ color: rgba(0, 0, 0);
60      font-size: 25px;
61      font-family:Georgia, 'Times New Roman', Times, serif
62      }
63  h1 {color: white;
64      font-size: 30px;
65  }
66  }
67  h3{color: rgba(9, 122, 98);
68      font-size: 30px;
69      font-family:'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif;
70  }
71  }
72  h5{color: rgba(27, 25, 177);
73      font-size: 30px;
74      font-family:Georgia, 'Times New Roman', Times, serif;
75  }
76  }
77  h6{
78      color: black;
79      font-size: 30px;
80      font-family:Georgia, 'Times New Roman', Times, serif
81  }
82  }
83  .btn{
84      border-radius:5px ;
85      padding: 15px 10px;
86  }
87  }
88  .btn-green{
89      color: rgba(253, 237, 237);
90      background-color: #ff2e2e;
91  }
92  }

```

Codificación JS

```

VENTA DE AUTOS > javascript > JS trabajos > mostrar
1  function marcas(){
2
3      document.form1.text2.value=document.form1.select1.options[document.form1.select1.selectedIndex].value;
4  }
5
6  function modelosToyota(){
7
8      document.form2.text2.value=document.form2.select1.options[document.form2.select1.selectedIndex].value;
9  }
10 function modelosHonda(){
11
12     document.form3.text2.value=document.form3.select1.options[document.form3.select1.selectedIndex].value;
13 }
14 function modelosSuzuki(){
15
16     document.form4.text2.value=document.form4.select1.options[document.form4.select1.selectedIndex].value;
17 }
18
19 //Función de las selecciones//
20 function mostrar() {
21
22     var num1=document.form1.select1.value;
23     var num2= document.form2.select1.value;
24     var num3= document.form3.select1.value;
25     var num4= document.form4.select1.value;
26
27     //Modelos de Toyota//
28     if(num1=="TOYOTA" && num2=="AURIS"){
29         document.write("<img src='https://acs2.blob.core.windows.net/imgcatalogo/x1/VA_0695f12bb99c469a844dc03028319b25.jpg' border =5px width=550 height=450px />");
30         document.write("<h2>"+"USTED ESCOGIO EL MODELO AURIS DE LA MARCA TOYOTA"+"<h2>");
31         document.write("PRECIO AL CONTADO: 5/125,000.00"+<br>");
32         document.write("<br>");
33         document.write("<h2>"+"CARACTERISTICAS"+"<h2>");
34         document.write("<h2>"+"EFICIENTE MOTOR DE: 1.2L"+"<h2>");
35         document.write("<h2>"+"TECNOLOGIA: Pantall; Mylink 7in "+"<h2>");
36         document.write("<h2>"+"DISEÑO: Amplia maletera de 390L "+"<h2>");
37         document.write("<h2>"+"SEGURIDAD: Doble Airbag + ABS "+"<h2>");
38
39     }
40
41     else if(num1=="TOYOTA" && num2=="AYGO"){
42         document.write("<img src='https://www.diarimotor.com/imagenes/picscache/750x/toyota-aygo-oferta-2019_750x.jpg' border =5px width=700 height=450px/>");
43         document.write("<h2>"+"USTED ESCOGIO EL MODELO AYGO DE LA MARCA TOYOTA"+"<h2>");
44         document.write("PRECIO AL CONTADO: 5/57,900.00"+<br>");
45         document.write("<br>");
46         document.write("<h2>"+"CARACTERISTICAS"+"<h2>");
47         document.write("<h2>"+"EFICIENTE MOTOR DE: 1.4L"+"<h2>");
48         document.write("<h2>"+"CONFORD : Lunas Electricas "+"<h2>");
49         document.write("<h2>"+"DISEÑO: Black Edition "+"<h2>");
50         document.write("<h2>"+"SEGURIDAD: Doble Airbag + ABS "+"<h2>");
51
52     }
53
54     else if(num1=="TOYOTA" && num2=="COROLLA"){
55         document.write("<img src='https://acs2.blob.core.windows.net/imgcatalogo/x1/VA_efb2f80f9a7a4a878852f629cf870871.jpg' border =5px width=550 height=450px />");
56         document.write("<h2>"+"USTED ESCOGIO EL MODELO COROLLA DE LA MARCA TOYOTA"+"<h2>");
57         document.write("PRECIO AL CONTADO: 5/106,000.00"+<br>");
58         document.write("<br>");
59         document.write("<h2>"+"CARACTERISTICAS"+"<h2>");
60         document.write("<h2>"+"EFICIENTE MOTOR DE: 1.4L"+"<h2>");
61         document.write("<h2>"+"CONFORD : Maletera de 300L "+"<h2>");
62         document.write("<h2>"+"CONFORD: Aire Acondicionado "+"<h2>");
63         document.write("<h2>"+"SEGURIDAD: Frenos ABS con EBD "+"<h2>");
64
65     }
66
67     else if(num1=="TOYOTA" && num2=="CARRY"){
68         document.write("<img src='https://img.autocosmos.com/noticias/fotos/preview/WAZ_231d75746a0b422f94123eee06fa607e.jpg' border =5px width=550 height=450px />");
69         document.write("<h2>"+"USTED ESCOGIO EL MODELO CARRY DE LA MARCA TOYOTA"+"<h2>");
70         document.write("PRECIO AL CONTADO: 5/183,000.00"+<br>");
71         document.write("<br>");
72         document.write("<h2>"+"CARACTERISTICAS"+"<h2>");
73         document.write("<h2>"+"MOTOR: Turbo Torque 160 Nm"+"<h2>");
74         document.write("<h2>"+"CONECTIVIDAD : WIFI 4G"+"<h2>");
75         document.write("<h2>"+"SEIS: Airbags "+"<h2>");
76         document.write("<h2>"+"ASISTENTE AUTOMÁTICO: Estacionamiento"+"<h2>");
77
78     }
79
80 }

```

```

77     else if(num1=="TOYOTA" && num2=="C-HR"){
78         document.write("<img src='https://www.buyatoyota.com/assets/img/vehicle-info/C-HR/2021/hero_image_c-hr.png' border =5px width=550 height=450px/>");
79         document.write("<h2>+USTED ESCOGIO EL MODELO C-HR DE LA MARCA TOYOTA+</h2>");
80         document.write("PRECIO AL CONTADO: S/123,000.00+<br>");
81         document.write("<br>");
82         document.write("<h2>+CARACTERISTICAS+</h2>");
83         document.write("<h2>+MOTOR: HP 115 Torque: 160 Nm+</h2>");
84         document.write("<h2>+CONNECTIVIDAD : WIFI 4G+</h2>");
85         document.write("<h2>+CONFORD: Easy Park "+</h2>");
86         document.write("<h2>+SEGURIDAD: 6 Airbags+</h2>");
87     }
88     else if(num1=="TOYOTA" && num2=="ETIOS"){
89         document.write("<img src= 'https://noticias.coches.com/wp-content/uploads/2014/07/toyota_etios-sedan-concept-2010_r4.jpg'border =5px width=550 height=450px/>");
90         document.write("<h2>+USTED ESCOGIO EL MODELO ETIOS DE LA MARCA TOYOTA+</h2>");
91         document.write("PRECIO AL CONTADO: S/55,000.00+<br>");
92         document.write("<br>");
93         document.write("<h2>+CARACTERISTICAS+</h2>");
94         document.write("<h2>+DISEÑO: 3 Filas de Asiento+</h2>");
95         document.write("<h2>+TECNOLOGIA : Toyota MyLink+</h2>");
96         document.write("<h2>+CONFORD: Asientos Ecocuero "+</h2>");
97         document.write("<h2>+SEGURIDAD:Camara de retroceso con censor"+</h2>");
98     }
99     else if(num1=="TOYOTA" && num2=="FJ CRUISER"){
100         document.write("<img src='https://www.toyotaperu.com.pe/sites/default/files/camioneta-fj-cruiser-toyota_1_0.png' border =5px width=700 height=450px/>");
101         document.write("<h2>+USTED ESCOGIO EL MODELO FJ CRUISER DE LA MARCA TOYOTA+</h2>");
102         document.write("PRECIO AL CONTADO: S/206,000.00+<br>");
103         document.write("<br>");
104         document.write("<h2>+CARACTERISTICAS+</h2>");
105         document.write("<h2>+MOTOR: 6.2L V8 455 HP+</h2>");
106         document.write("<h2>+TECNOLOGIA :Caja Automática de 10 Velocidades+</h2>");
107         document.write("<h2>+CONFORD: Hed UP Diplay "+</h2>");
108         document.write("<h2>+SEGURIDAD: 6 Airbags para toda las versiones+</h2>");
109     }

```

```

110     else if(num1=="TOYOTA" && num2=="FORTUNER"){
111         document.write("<img src='https://expo.baccredomatic.com/wp-content/uploads/2021/03/BAC-TOYOTA-FORTUNER-EW-21.jpg' border =5px width=550 height=450px />");
112         document.write("<h2>+USTED ESCOGIO EL MODELO FORTUNER DE LA MARCA TOYOTA+</h2>");
113         document.write("PRECIO AL CONTADO: S/150,000.00+<br>");
114         document.write("<br>");
115         document.write("<h2>+CARACTERISTICAS+</h2>");
116         document.write("<h2>+DISEÑO: Faros Neblineros LED+</h2>");
117         document.write("<h2>+TECNOLOGIA: Smart Entry+</h2>");
118         document.write("<h2>+CONFORD: Maletera de 700L+</h2>");
119         document.write("<h2>+SEGURIDAD: Doble Airbags + ABS+</h2>");
120     }
121 }
122 }
123 //Modelos de honda//
124 else if(num1=="HONDA" && num3=="JAZZ" ){
125     document.write("<img src='https://i.blogs.es/6a4d7c/honda-jazz-2020-011/1366_2000.jpg' border =5px width=550 height=450px/>");
126     document.write("<h2>+USTED ESCOGIO EL MODELO JAZZ DE LA MARCA HONDA+</h2>");
127     document.write("PRECIO AL CONTADO: S/117,000.00+<br>");
128     document.write("<br>");
129     document.write("<h2>+CARACTERISTICAS+</h2>");
130     document.write("<h2>+DISEÑO: Faros Neblineros LED+</h2>");
131     document.write("<h2>+TECNOLOGIA: Smart Entry+</h2>");
132     document.write("<h2>+CONFORD: Maletera de 700L+</h2>");
133     document.write("<h2>+SEGURIDAD: Doble Airbags + ABS+</h2>");
134 }

```

```

135 else if(num1=="HONDA" && num3=="NSX" ){
136     document.write("<img src='https://cdn.autobild.es/sites/navi.axelspringer.es/public/styles/1200/public/media/image/2015/04/382539-honda-nsx-2015-produccion.jpg?h=382539' border =5px width=550 height=450px/>");
137     document.write("<h2>+USTED ESCOGIO EL MODELO NSX DE LA MARCA HONDA+</h2>");
138     document.write("PRECIO AL CONTADO: S/317,000.00+<br>");
139     document.write("<br>");
140     document.write("<h2>+CARACTERISTICAS+</h2>");
141     document.write("<h2>+MOTOR: 6.2L V8 455 HP+</h2>");
142     document.write("<h2>+TECNOLOGIA :Caja Automática de 10 Velocidades+</h2>");
143     document.write("<h2>+CONFORD: Hed UP Diplay "+</h2>");
144     document.write("<h2>+SEGURIDAD: 6 Airbags para toda las versiones+</h2>");
145 }
146 else if(num1=="HONDA" && num3=="ODYSSEY" ){
147     document.write("<img src='https://www.pereuratacar.com/wp-content/uploads/CarRentalGallery/1599614734_odyssey.jpg' border =5px width=550 height=450px/>");
148     document.write("<h2>+USTED ESCOGIO EL MODELO ODYSSEY DE LA MARCA HONDA+</h2>");
149     document.write("PRECIO AL CONTADO: S/227,000.00+<br>");
150     document.write("<br>");
151     document.write("<h2>+CARACTERISTICAS+</h2>");
152     document.write("<h2>+DISEÑO: 3 Filas de Asiento+</h2>");
153     document.write("<h2>+TECNOLOGIA : Toyota MyLink+</h2>");
154     document.write("<h2>+CONFORD: Asientos Ecocuero "+</h2>");
155     document.write("<h2>+SEGURIDAD:Camara de retroceso con censor"+</h2>");
156 }

```

```

157 else if(num1=="HONDA" && num3=="CR-V" ){
158     document.write("<img src='https://maquinarias.pe/wp-content/uploads/2020/10/CR-V-calado.png' border =5px width=700 height=450px/>");
159     document.write("<h2>*USTED ESCOGIO EL MODELO CR-V DE LA MARCA HONDA*</h2>*");
160     document.write("PRECIO AL CONTADO: S/135,000.00*<br>*");
161     document.write("<br>");
162     document.write("<h2>*CARACTERISTICAS*</h2>*");
163     document.write("<h2>*MOTOR: HP 115 Torque: 160 Nm*</h2>*");
164     document.write("<h2>*CONECTIVIDAD : WiFi 4G*</h2>*");
165     document.write("<h2>*CONFORD: Easy Park *</h2>*");
166     document.write("<h2>*SEGURIDAD: 6 Airbags*</h2>*");
167 }
168 else if(num1=="HONDA" && num3=="ACCORD" ){
169     document.write("<img src='https://acroadtrip.blob.core.windows.net/catalogo-imagenes/x1/RT_V_F12bc286a064ebda8a5dee523839140.jpg' border =5px width=550 height=450px />");
170     document.write("<h2>*USTED ESCOGIO EL MODELO ACCORD DE LA MARCA HONDA*</h2>*");
171     document.write("PRECIO AL CONTADO: S/139,000.00*<br>*");
172     document.write("<br>");
173     document.write("<h2>*CARACTERISTICAS*</h2>*");
174     document.write("<h2>*MOTOR: Turbo Torque 160 Nm*</h2>*");
175     document.write("<h2>*CONECTIVIDAD : WiFi 4G*</h2>*");
176     document.write("<h2>*SEIS: Airbags *</h2>*");
177     document.write("<h2>*ASISTENTE AUTOMÁTICO: Estacionamiento*</h2>*");
178 }
179 else if(num1=="HONDA" && num3=="PILOT" ){
180     document.write("<img src='https://maquinarias.pe/wp-content/uploads/2020/10/PILO1-calado.png' border =5px width=700 height=450px />");
181     document.write("<h2>*USTED ESCOGIO EL MODELO PILOT DE LA MARCA HONDA*</h2>*");
182     document.write("PRECIO AL CONTADO: S/160,000.00*<br>*");
183     document.write("<br>");
184     document.write("<h2>*CARACTERISTICAS*</h2>*");
185     document.write("<h2>*EFICIENTE MOTOR DE: 1.4L*</h2>*");
186     document.write("<h2>*CONFORD : Maletera de 500L *</h2>*");
187     document.write("<h2>*CONFORD: Aire Acondicionado *</h2>*");
188     document.write("<h2>*SEGURIDAD: Frenos ABS con EBD *</h2>*");
189 }
190 else if(num1=="HONDA" && num3=="WR-V" ){
191     document.write("<img src='https://acs2.blob.core.windows.net/img/catalogo/x1/VA_4f776947285a4e5fbd9999660a37519.jpg' border =5px width=650 height=450px />");
192     document.write("<h2>*USTED ESCOGIO EL MODELO WR-V DE LA MARCA HONDA*</h2>*");
193     document.write("PRECIO AL CONTADO: S/69,054.19*<br>*");
194     document.write("<br>");

```

```

195     document.write("<h2>*CARACTERISTICAS*</h2>*");
196     document.write("<h2>*EFICIENTE MOTOR DE: 1.4L*</h2>*");
197     document.write("<h2>*CONFORD : Lunas Electricas *</h2>*");
198     document.write("<h2>*DISEÑO: Black Edition *</h2>*");
199     document.write("<h2>*SEGURIDAD: Doble Airbag + ABS *</h2>*");
200 }
201 else if(num1=="HONDA" && num3=="CIVIC" ){
202     document.write("<img src='https://maquinarias.pe/wp-content/uploads/2020/10/CIVIC-calado.png' border =5px width=700 height=450px />");
203     document.write("<h2>*USTED ESCOGIO EL MODELO CIVIC DE LA MARCA HONDA*</h2>*");
204     document.write("PRECIO AL CONTADO: S/574,000.00*<br>*");
205     document.write("<br>");
206     document.write("<h2>*CARACTERISTICAS*</h2>*");
207     document.write("<h2>*EFICIENTE MOTOR DE: 1.2L*</h2>*");
208     document.write("<h2>*TECNOLOGIA: Pantalla Mylink 7in *</h2>*");
209     document.write("<h2>*DISEÑO: Amplia maletera de 390L *</h2>*");
210     document.write("<h2>*SEGURIDAD: Doble Airbag + ABS *</h2>*");
211 }
212
213 //Modelos de suzuki//
214 else if(num1=="SUZUKI" && num4=="CELERIO"){
215     document.write("<img src='https://alcalasuzuki.com/wp-content/uploads/2020/02/suzuki-celerio.jpg' border =5px width=700 height=450px/>");
216     document.write("<h2>*USTED ESCOGIO EL MODELO CELERIO DE LA MARCA SUZUKI*</h2>*");
217     document.write("PRECIO AL CONTADO: S/44,000.00*<br>*");
218     document.write("<br>");
219     document.write("<h2>*CARACTERISTICAS*</h2>*");
220     document.write("<h2>*DISEÑO: Faros Neblineros LED*</h2>*");
221     document.write("<h2>*TECNOLOGIA: Smart Entry*</h2>*");
222     document.write("<h2>*CONFORD: Maletera de 700L*</h2>*");
223     document.write("<h2>*SEGURIDAD: Doble Airbags + ABS *</h2>*");
224 }
225 else if(num1=="SUZUKI" && num4=="SWIFT"){
226     document.write("<img src='https://derco-pe-prod.s3.amazonaws.com/images/models/2021-06-24-Gris%20480x320.jpg' border =5px width=650 height=450px />");
227     document.write("<h2>*USTED ESCOGIO EL MODELO SWIFT DE LA MARCA SUZUKI*</h2>*");
228     document.write("PRECIO AL CONTADO: S/51,381.00*<br>*");
229     document.write("<br>");
230     document.write("<h2>*CARACTERISTICAS*</h2>*");
231     document.write("<h2>*MOTOR: 6.2L V8 455 HP*</h2>*");

```

```

232     document.write("<h2>"+"TECNOLOGIA :Caja Automática de 10 Velocidades"+"</h2>");
233     document.write("<h2>"+"CONFORD: Hed UP Display "+"</h2>");
234     document.write("<h2>"+"SEGURIDAD: 6 Airbags para toda las versiones"+"</h2>");
235 }
236 else if(num1=="SUZUKI" && num4=="JIMNY"){
237     document.write("<img src='https://p4.wallpaperbetter.com/wallpaper/615/749/160/jimmy-suzuki-wallpaper-preview.jpg' border =5px width=650 height=450px />");
238     document.write("<h2>"+"USTED ESCOGIO EL MODELO JIMNY DE LA MARCA SUZUKI"+"</h2>");
239     document.write("PRECIO AL CONTADO: 5/168,622,392"+"<br>");
240     document.write("<br>");
241     document.write("<h2>"+"CARACTERISTICAS"+"</h2>");
242     document.write("<h2>"+"DISEÑO: 3 Filas de Asiento"+"</h2>");
243     document.write("<h2>"+"TECNOLOGIA : Toyota MyLink"+"</h2>");
244     document.write("<h2>"+"CONFORD: Asientos Ecocuero "+"</h2>");
245     document.write("<h2>"+"SEGURIDAD:Camara de retroceso con sensor"+"</h2>");
246 }
247 else if(num1=="SUZUKI" && num4=="VITARA"){
248     document.write("<img src='https://p4.wallpaperbetter.com/wallpaper/68/801/242/cars-suzuki-wallpaper-preview.jpg' border =5px width=700 height=450px />");
249     document.write("<h2>"+"USTED ESCOGIO EL MODELO VITARA DE LA MARCA SUZUKI"+"</h2>");
250     document.write("PRECIO AL CONTADO: 5/121,957.00"+"<br>");
251     document.write("<br>");
252     document.write("<h2>"+"CARACTERISTICAS"+"</h2>");
253     document.write("<h2>"+"MOTOR: HP 115 Torque: 160 Nm"+"</h2>");
254     document.write("<h2>"+"CONECTIVIDAD : WIFI 4G"+"</h2>");
255     document.write("<h2>"+"CONFORD: Easy Park "+"</h2>");
256     document.write("<h2>"+"SEGURIDAD: 6 Airbags"+"</h2>");
257 }
258 else if(num1=="SUZUKI" && num4=="ERTIGA"){
259     document.write("<img src='https://s1.lzooom.me/65050/242/Suzuki_2018-19_Maruti_Ertiga_White_Background_576511_1600x1200.jpg' border =5px width=650 height=450px />");
260     document.write("<h2>"+"USTED ESCOGIO EL MODELO ERTIGA DE LA MARCA SUZUKI"+"</h2>");
261     document.write("PRECIO AL CONTADO: 5/77,703.00"+"<br>");
262     document.write("<br>");
263     document.write("<h2>"+"CARACTERISTICAS"+"</h2>");
264     document.write("<h2>"+"MOTOR: Turbo Torque 160 Nm"+"</h2>");
265     document.write("<h2>"+"CONECTIVIDAD : WIFI 4G"+"</h2>");
266     document.write("<h2>"+"SEIS: Airbags "+"</h2>");
267     document.write("<h2>"+"ASISTENTE AUTOMÁTICO: Estacionamiento"+"</h2>");
268 }

```

```

269 else if(num1=="SUZUKI" && num4=="APV"){
270     document.write("<img src='https://perco-pe-prod.s3.amazonaws.com/medias/suzuki/migration/front-image/apv-furgon-destacado.jpg' border =5px width=650 height=450px />");
271     document.write("<h2>"+"USTED ESCOGIO EL MODELO APV DE LA MARCA SUZUKI"+"</h2>");
272     document.write("PRECIO AL CONTADO: 5/69,477.00"+"<br>");
273     document.write("<br>");
274     document.write("<h2>"+"CARACTERISTICAS"+"</h2>");
275     document.write("<h2>"+"EFICIENTE MOTOR DE: 1.4L"+"</h2>");
276     document.write("<h2>"+"CONFORD : Maletera de 500L "+"</h2>");
277     document.write("<h2>"+"CONFORD: Aire Acondicionado "+"</h2>");
278     document.write("<h2>"+"SEGURIDAD: Frenos ABS con EBD "+"</h2>");
279 }
280 else if(num1=="SUZUKI" && num4=="BALENO"){
281     document.write("<img src='https://www.diarionmotor.com/imagenes/picacache/375x250c/suzuki-baleno-ficha-1118-002_375x250c.jpg' border =5px width=650 height=450px />");
282     document.write("<h2>"+"USTED ESCOGIO EL MODELO BALENO DE LA MARCA SUZUKI"+"</h2>");
283     document.write("PRECIO AL CONTADO: 5/35,440.00"+"<br>");
284     document.write("<br>");
285     document.write("<h2>"+"CARACTERISTICAS"+"</h2>");
286     document.write("<h2>"+"EFICIENTE MOTOR DE: 1.4L"+"</h2>");
287     document.write("<h2>"+"CONFORD : Lunas Electricas "+"</h2>");
288     document.write("<h2>"+"DISEÑO: Black Edition "+"</h2>");
289     document.write("<h2>"+"SEGURIDAD: Doble Airbag + ABS "+"</h2>");
290 }
291 else if(num1=="SUZUKI" && num4=="CIAZ"){
292     document.write("<img src='https://www.portalautomotriz.com/sites/portalautomotriz.com/files/media/photos/suzuki-ciaz-1.jpg' border =5px width=650 height=450px />");
293     document.write("<h2>"+"USTED ESCOGIO EL MODELO CIAZ DE LA MARCA SUZUKI"+"</h2>");
294     document.write("PRECIO AL CONTADO: 5/54,400.00"+"<br>");
295     document.write("<br>");
296     document.write("<h2>"+"CARACTERISTICAS"+"</h2>");
297     document.write("<h2>"+"EFICIENTE MOTOR DE: 1.2L"+"</h2>");
298     document.write("<h2>"+"TECNOLOGIA: Pantalla Mylink 7in "+"</h2>");
299     document.write("<h2>"+"DISEÑO: Amplia maletero de 390L "+"</h2>");
300     document.write("<h2>"+"SEGURIDAD: Doble Airbag + ABS "+"</h2>");
301 }
302 }

```

Codificación HTML secundario

```

VENTA DE AUTOS > 1.1.html > html > body > div
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <title>Vicasakedi</title>
5 <meta charset="UTF-8">
6 <meta http-equiv="X-UA-Compatible" content="IE=edge">
7 <meta name="viewport" content="width=device-width, initial-scale=1">
8 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-KjZfEg30hdM668r+8F4XLRK2vvoC2f389z706"
9 <link rel="stylesheet" href="css/estilo1.css">
10 </head>
11 <body>
12 </body>
13 </html>
14 <nav class="navbar navbar-expand-lg navbar-light bg-light">
15 <div class="nav">
16 <li class="nav-item active">
17 <a class="nav-link" href="javascript:history.go(-1)">Inicio</a>
18 </li>
19 </div>
20 </nav>
21 </header>
22 <div class="row">
23 <div class="col-12 bg-success p-2 text-dark bg-opacity-25">
24 <script src="java script/trabajo.js"></script>
25 <center>
26 <br>
27 <h2><b>SELECCIONE EL AUTOMÓVIL DE SU PREFERENCIA </b></h2>
28 <form name="form1">
29 <br>
30 <h2>MARCAS DE AUTOMÓVILES:
31 <br>
32 <select size="1" style="border:double;background:■rgb(43, 223, 255);color:□rgb(0, 0, 0)" name="select1" onchange="marcas()">
33 <option value="TOYOTA"> TOYOTA</option>
34 <option value="HONDA"> HONDA</option>
35 <option value="SUZUKI"> SUZUKI</option>
36 </select>
37 <br><br>
38 </div>
39 </div>
40 </div>

```

```

41 <h2>
42 Usted Escogio la Marca: <input type="text" name="text2">
43 <br>
44 </form>
45 <br>
46 <center><h2><b>MODELOS: </b></h2></center>
47 <br>
48 <center><h2>* Seleccione el modelo de su automóvil, según la marca escogida anteriormente.</h2></center>
49 <br>
50 <form name="form2">
51 <h2>MODELOS DE TOYOTA:
52 <select size="1" style="border:double;background:■rgb(43, 223, 255);color:□rgb(0, 0, 0)" name="select1" onchange="modelosToyota()">
53 <option value="AURIS"> AURIS</option>
54 <option value="AYGO">AYGO</option>
55 <option value="COROLLA">COROLLA</option>
56 <option value="CAMRY"> CAMRY</option>
57 <option value="C-HR">C-HR</option>
58 <option value="ETIOS"> ETIOS</option>
59 <option value="FJ CRUISER">FJ CRUISER</option>
60 <option value="FORTUNER"> FORTUNER</option>
61 </select>
62 <br>
63 <h2>
64 El modelo que escogio es: <input type="text" name="text2">
65 </form>
66 <form name="form3">
67 <br>
68 <h2> MODELOS DE HONDA:
69 <select size="1" style="border:double;background:■rgb(43, 223, 255);color:□rgb(0, 0, 0)" name="select1" onchange="modelosHonda()">
70 <option value="JAZZ">JAZZ </option>
71 <option value="NSX">NSX</option>
72 <option value="ODYSSEY">ODYSSEY</option>
73 <option value="CR-V">CR-V</option>
74 <option value="ACCORD">ACCORD</option>
75 <option value="PILOT">PILOT</option>
76 </select>

```

```

76     <option value="PILOT">PILOT</option>
77     <option value="MR-V">MR-V</option>
78     <option value="CIVIC">CIVIC</option>
79
80     </select>
81     <h2>
82     El modelo que escogio es: <input type="text" name="text2">
83     </form>
84     <form name="form4">
85     <br>
86     <h2> MODELOS DE SUZUKI:
87     <select size="1" style="border:double;background:■rgb(43, 223, 255);color:□rgb(0, 0, 0)" name="select1" onchange="modelosSuzuki()">
88     <option value="CELERIO">CELERIO</option>
89     <option value="SMIFT">SMIFT</option>
90     <option value="JIMNY">JIMNY</option>
91     <option value="VITARA">VITARA</option>
92     <option value="ERTIGA">ERTIGA</option>
93     <option value="APV">APV</option>
94     <option value="BALENO">BALENO</option>
95     <option value="CIAZ">CIAZ</option>
96
97     </select>
98     <h2>
99     El modelo que escogio es: <input type="text" name="text2">
100    </form>
101    <br>
102    <br>
103    <p class="center-content"><input type="submit" class="btn btn-green" value="ENVIAR" onclick="mostrar()"></p>
104    </form>
105    </center>
106
107 </div>
108 </div>
109
110 </div>
111 </body>
112 </html>

```

Codificación HTML Principal

```

VENTA DE AUTOS > index.html > html > body > div:row > div:col-12bg-sucess.p-2text-darkbg-opacity.35 > section
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <title>VICASHAKEDI</title>
5     <a name="inicio"></a>
6     <meta charset="UTF-8">
7     <meta http-equiv="x-UA-Compatible" content="IE=edge">
8     <meta name="viewport" content="width=device-width,initial-scale=1">
9     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Ky7XEAg3Qh1Mq68r+8Fh4XlRk2vwoC2F3809+Wn8CA5Q1V7C03I"
10    <link rel="stylesheet" href="css/estilo1.css">
11 </head>
12 <body>
13 <header>
14     <nav class="navbar navbar-expand-lg navbar-light bg-light">
15     
17     <li class="nav-item active">
18     <a class="nav-link" href="#servicios">Autos</a>
19     </li>
20     <li class="nav-item">
21     <a class="nav-link" href="#mapa">Ubicación</a>
22     </li>
23     <li class="nav-item">
24     <a class="nav-link" href="1.1.html">Costear Vehículo</a>
25     </li>
26     </ul>
27 </nav>
28
29 <div class="cover d-flex justify-content-end align-items-center flex-column">
30 <h1>
31     "VICASHAKEDI SAC."
32 </h1>
33 <p>
34     VENTA DE AUTOMÓVILES
35 </p>
36 </div>
37 </header>
38 <section>
39 <div class="container mt-5 mb-5">
40 <div class="row justify-content-center">

```

```

41 <br><h3>NUESTRAS MARCAS</h3><br/>
42 <br><br/>
43 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
44   <div class="card">
45     <div title="..." class="cover cover-small" style="background-image: url(https://pixy.org/src/147/1478452.jpg)">
46     </div>
47     <div class="card-body">
48       <h5 class="card-title">TOYOTA</h5>
49       <p class="card-text">Toyota es una marca llena de historia y una de las empresas fabricantes de vehículos más grandes de Japón, que ha
50       marcado el desarrollo del mercado automovilístico. Sin dudas, goza de la confianza, aceptación, liderazgo y fiabilidad de millones de
51       usuarios en todo el mundo.</p>
52     </div>
53   </div>
54 </div>
55 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
56   <div class="card">
57     <div title="..." class="cover cover-small" style="background-image: url(https://1000marcas.net/wp-content/uploads/2019/12/Logo-Honda.jpg)">
58     </div>
59     <div class="card-body">
60       <h5 class="card-title">HONDA</h5>
61       <p class="card-text">Honda es una marca japonesa dedicada a la fabricación de motocicletas, automóviles, motores acuáticos, robots y hasta
62       aeronaves. Fundada en 1959, es considerada como la compañía de motores de combustión interna más grande del mundo.</p>
63     </div>
64   </div>
65 </div>
66 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
67   <div class="card">
68     <div title="..." class="cover cover-small" style="background-image: url(https://www.v1png.com/png/detail/110-1100017_suzuki-motorcycle-logo-png-car-suzuki-logo.png)">
69     </div>
70     <div class="card-body">
71       <h5 class="card-title">SUZUKI</h5>
72       <p class="card-text">Suzuki Motor Corporation es una empresa japonesa dedicada a la fabricación de automóviles, motocicletas, motores fuera borda, y
73       variedad de productos equipados con motores de combustión. Fue fundada en 1909 en la localidad de Hamamatsu, Japón.</p>
74     </div>
75   </div>
76 </div>
77 </div>
78 </div>
79 </section>

```

```

80 <!-- TIRA DE IMAGENES DE SERVICIOS QUE OFRECE -->
81 <section>
82   <!-- MODELOS TOYOTA -->
83   <div class="container mt-5 mb-5" id="servicios">
84     <h3 class="title">MODELOS DE LA MARCA TOYOTA</h3>
85     <a href="#inicio">Inicio</a>
86     <br>
87     <div class="product-stripe">
88       <div class="stripe-container">
89         <!-- IMAGEN 1 -->
90         <div class="card">
91           <div title="..." class="cover cover-small" style="background-image: url(https://acs2.blob.core.windows.net/ingcatalogo/x1/VA_0696f12bb99c469a8d4dc03028319e26.jpg)">
92           </div>
93           <div class="card-body">
94             <h5 class="card-title">TOYOTA AURIS</h5>
95             <span class="badge bg-danger">Precio Antes: $/125,476.00</span>
96             <span class="badge bg-info text-dark">Precio Despues: $/125,000.00</span>
97             <p class="card-text">El toyota Auris es un automovil de turismo, es un hatchback de cinco puertas que sustituye en Sudamérica y Europa al Toyota Corolla
98             Sport o Corolla Hatchback.</p>
99           </div>
100         </div>
101         <!-- IMAGEN 2 -->
102         <div class="card">
103           <div title="..." class="cover cover-small" style="background-image: url(https://www.diariomotor.com/imagenes/picscache/750x/toyota-aygo-oferta-2019_750x.jpg)">
104           </div>
105           <div class="card-body">
106             <h5 class="card-title">TOYOTA AYGO</h5>
107             <span class="badge bg-danger">Precio: $/57,350.00</span>
108             <span class="badge bg-info text-dark">Precio Despues: $/57,000.00</span>
109             <p class="card-text">El Toyota Aygo es la actualización del modelo más pequeño de la marca japonesa, el Toyota Aygo contará con tres niveles
110             de equipamiento en nuestro país.</p>
111           </div>
112         </div>
113         <!-- IMAGEN 3 -->
114         <div class="card">
115           <div title="..." class="cover cover-small" style="background-image: url(https://acs2.blob.core.windows.net/ingcatalogo/x1/VA_eFb2f80f9a764a878852f620cf870871.jpg)">
116           </div>
117           <div class="card-body">
118             <h5 class="card-title">TOYOTA COROLLA</h5>
119             <span class="badge bg-danger">Precio: $/166,730.00</span>

```

```

120 <span class="badge bg-info text-dark">Precio Despues: S/106,000.00</span>
121 <p class="card-text">El Toyota Corolla ha sido el caballito de batalla de millones de familias por más de 50 años, recomendada por la empresa.</p>
122 </div>
123 </div>
124 <!-- IMAGEN 4 -->
125 <div class="card">
126 <div title="..." class="cover cover-small" style="background-image: url(https://img.autocosmos.com/noticias/fotos/preview/IAZ_231d75746a0b422f94123eee06fa6b7e.jpg)">
127 </div>
128 <div class="card-body">
129 <h5 class="card-title">TOYOTA CAMRY</h5>
130 <span class="badge bg-danger">Precio: S/183,040.00</span>
131 <span class="badge bg-info text-dark">Precio Despues: S/183,000</span>
132 <p class="card-text">Lllega a España el Toyota Camry Hybrid, el modelo híbrido que completa la gana Toyota Camry de 2018, este modelo es
133 uno de los mas exitosos y vendidos de nuestra empresa.</p>
134 </div>
135 </div>
136 <!-- IMAGEN 5 -->
137 <div class="card">
138 <div title="..." class="cover cover-small" style="background-image: url(https://www.buytoyota.com/assets/img/vehicle-info/C-HR/2021/hero_image_c-hr.png)">
139 </div>
140 <div class="card-body">
141 <h5 class="card-title">TOYOTA C-HR</h5>
142 <span class="badge bg-danger">Precio Antes: S/123,390.00</span>
143 <span class="badge bg-info text-dark">Precio Despues: S/123,000.00</span>
144 <p class="card-text">Su diseño futurista, prestaciones únicas y tecnología híbrida auto-recargable, harán de la c-hr, la mejor compañera para ti y
145 el medio ambiente.</p>
146 </div>
147 </div>
148 <!-- IMAGEN 6 -->
149 <div class="card">
150 <div title="..." class="cover cover-small" style="background-image: url(https://noticias.coches.com/wp-content/uploads/2014/07/toyota_etios-sedan-concept-2010_r4.jpg)">
151 </div>
152 <div class="card-body">
153 <h5 class="card-title">TOYOTA ETIOS</h5>
154 <span class="badge bg-danger">Precio Antes: S/55,965.00</span>
155 <span class="badge bg-info text-dark">Precio Despues: S/55,000.00</span>
156 <p class="card-text">El nuevo etios trae consigo una renovada máscara frontal que sorprenderá a muchos gracias a su diseño con un estilo más deportivo.</p>
157 </div>
158 </div>

```

```

159 <!-- IMAGEN 7 -->
160 <div class="card">
161 <div title="..." class="cover cover-small" style="background-image: url(https://www.toyotaperu.com.pe/sites/default/files/comonete-fj-cruiser-toyota_1_0.png)">
162 </div>
163 <div class="card-body">
164 <h5 class="card-title">TOYOTA FJ CRUISER</h5>
165 <span class="badge bg-danger">Precio Antes: S/206,700.00</span>
166 <span class="badge bg-info text-dark">Precio Despues: S/205,000.00</span>
167 <p class="card-text">Probada en los terrenos más abruptos, la fj cruiser hace honor a su legado y no se deja intimidar por ningún desafío.</p>
168 </div>
169 </div>
170 <!-- IMAGEN 8 -->
171 <div class="card">
172 <div title="..." class="cover cover-small" style="background-image: url(https://expo.baccredonatic.com/wp-content/uploads/2021/03/BAC-TOYOTA-FORTUNER-EW-22.jpg)">
173 </div>
174 <div class="card-body">
175 <h5 class="card-title">TOYOTA FORTUNER</h5>
176 <span class="badge bg-danger">Precio Antes: S/150,111.00</span>
177 <span class="badge bg-info text-dark">Precio Despues: S/150,000.00</span>
178 <p class="card-text">La nueva fortunier cuenta con un renovado diseño exterior que muestra un nivel más alto de distinción y fuerza que no conoce imposibles
179 para llegar más allá.</p>
180 </div>
181 </div>
182 </div>
183 </div>
184 </div>
185 <!-- MODELOS HONDA -->
186 <div class="container mt-5 mb-5" id="servicios">
187 <h3 class="title">MODELOS DE LA MARCA HONDA</h3>
188 <a href="#inicio">Inicio</a>
189 <br>
190 <div class="product-stripe">
191 <div class="stripe-container">
192 <!-- IMAGEN 1 -->
193 <div class="card">
194 <div title="..." class="cover cover-small" style="background-image: url(https://i.blogs.es/6a467c/honda-jazz-2020-011/1366_2000.jpg)">
195 </div>
196 <div class="card-body">
197 <h5 class="card-title">HONDA JAZZ</h5>
198 <span class="badge bg-danger">Precio antes: S/118,430.00</span>

```

```

199 <span class="badge bg-info text-dark">Precio despues: $/117,000.00</span>
200 <p class="card-text">El Jazz es el modelo más compacto de la gama Honda, pero ni es pequeño ni carece del encanto de la marca japonesa en el aspecto dinámico.</p>
201 </div>
202 </div>
203 <!-- IMAGEN 2 -->
204 <div class="card">
205 <div title="..." class="cover cover-small" style="background-image: url(https://con.autobild.es/sites/navi.axelspringer.es/public/styles/1280/public/media/image/2015/04/3
206 </div>
207 <div class="card-body">
208 <h5 class="card-title">HONDA NSX</h5>
209 <span class="badge bg-danger">Precio antes: $/318,830.00</span>
210 <span class="badge bg-info text-dark">Precio despues: $/317,000.00</span>
211 <p class="card-text">El Honda NSX siempre ha sido el modelo más radical y deportivo de la firma japonesa. Un canto de cisne que con los años se ha convertido en leyenda. </p>
212 </div>
213 </div>
214 <!-- IMAGEN 3 -->
215 <div class="card">
216 <div title="..." class="cover cover-small" style="background-image: url(https://www.perurentacar.com/wp-content/uploads/CarRentalGallery/1599614734_odyseysej.jpg)">
217 </div>
218 <div class="card-body">
219 <h5 class="card-title">HONDA ODYSSEY</h5>
220 <span class="badge bg-danger">Precio antes: $/227,509.00</span>
221 <span class="badge bg-info text-dark">Precio despues: $/227,000.00</span>
222 <p class="card-text">La Honda Odyssey 2021 es un vehiculo de grandes dimensiones. Como beneficio, se adjudica un excelente espacio interior, y visualmente podrá
223 comprobarlo en los estacionamientos.</p>
224 </div>
225 </div>
226 <!-- IMAGEN 4 -->
227 <div class="card">
228 <div title="..." class="cover cover-small" style="background-image: url(https://maquinarias.pe/wp-content/uploads/2020/10/CR-V-calado.png)">
229 </div>
230 <div class="card-body">
231 <h5 class="card-title">HONDA CR.V</h5>
232 <span class="badge bg-danger">Precio antes: $/135,259.00</span>
233 <span class="badge bg-info text-dark">Precio despues: $/135,000.00</span>
234 <p class="card-text">El Honda CR-V es un automóvil todoterreno del segmento C producido por el fabricante japonés de automóviles Honda.</p>
235 </div>

```

```

236 </div>
237 <!-- IMAGEN 5 -->
238 <div class="card">
239 <div title="..." class="cover cover-small" style="background-image: url(https://acrcadtrip.blob.core.windows.net/catalogo-imagenes/x1/RT_V_f12bc28b6a064ebda8a6dee52383924
240 </div>
241 <div class="card-body">
242 <h5 class="card-title">HONDA ACCORD</h5>
243 <span class="badge bg-danger">Precio antes: $/139,359</span>
244 <span class="badge bg-info text-dark">Precio despues: $/139,000</span>
245 <p class="card-text">La marca Honda Accord 2021 presenta un modelo deportivo con una manejabilidad óptima para la ciudad.</p>
246 </div>
247 </div>
248 <!-- IMAGEN 6 -->
249 <div class="card">
250 <div title="..." class="cover cover-small" style="background-image: url(https://maquinarias.pe/wp-content/uploads/2020/10/PILOT-calado.png)">
251 </div>
252 <div class="card-body">
253 <h5 class="card-title">HONDA PILOT</h5>
254 <span class="badge bg-danger">Precio antes: $/163,959</span>
255 <span class="badge bg-info text-dark">Precio despues: $/160,000</span>
256 <p class="card-text">El Honda Pilot 2021 acaba de aterrizar en Perú junto a su última actualización, acompañado de nuevas distinciones que le permiten continuar
257 peleando en el segmento más competitivo del mercado automotriz.</p>
258 </div>
259 </div>
260 <!-- IMAGEN 7 -->
261 <div class="card">
262 <div title="..." class="cover cover-small" style="background-image: url(https://ecs2.blob.core.windows.net/ingcatalogo/x1/VA_4f776947265a4e5fbd9989660a17519.jpg)">
263 </div>
264 <div class="card-body">
265 <h5 class="card-title">HONDA WR-V</h5>
266 <span class="badge bg-danger">Precio antes: $/69,055.19</span>
267 <span class="badge bg-info text-dark">Precio despues: $/69,054.19</span>
268 <p class="card-text">Dentro de la gama Honda WR-V 2021 en Brasil, la nueva versión de entrada LX conservó el sistema de iluminación del modelo previo con
269 sistema halógeno, incluyendo además rines de 16 pulgadas.</p>
270 </div>
271 </div>
272 <!-- IMAGEN 8 -->
273 <div class="card">
274 <div title="..." class="cover cover-small" style="background-image: url(https://maquinarias.pe/wp-content/uploads/2020/10/CIVIC-calado.png)">

```

```

275 </div>
276 <div class="card-body">
277 <h5 class="card-title">HONDA CIVIC</h5>
278 <span class="badge bg-danger">Precio antes: S/574,170.00</span>
279 <span class="badge bg-info text-dark">Precio despues: S/574,000.00</span>
280 <p class="card-text">El Honda Civic es un automóvil del segmento C fabricado por la empresa japonesa Honda. Tras haber pasado por varias modificaciones
281 de generación, el Civic ha crecido en tamaño, colocándose entre el Honda Jazz y el Honda Accord.</p>
282 </div>
283 </div>
284 </div>
285 </div>
286 </div>
287 <!-- MODELOS SUZUKI -->
288 <div class="container mt-5 mb-5" id="servicios">
289 <h3 class="title">MODELOS DE LA MARCA SUZUKI</h3>
290 <a href="#inicio">Inicio</a>
291 <br>
292 <div class="product-stripe">
293 <div class="stripe-container">
294 <!-- IMAGEN 1 -->
295 <div class="card">
296 <div title="..." class="cover cover-small" style="background-image: url(https://alcalasuzuki.com/wp-content/uploads/2020/02/suzuki-celerio.jpg)">
297 </div>
298 <div class="card-body">
299 <h5 class="card-title">SUZUKI CELERIO</h5>
300 <span class="badge bg-danger">Precio antes: S/44,200.20</span>
301 <span class="badge bg-info text-dark">Precio ahora: S/44,000.00</span>
302 <p class="card-text">Presentamos el nuevo Celerio, un auto pequeño pero cargado con grandes ideas y con mayor espacio</p>
303 </div>
304 </div>
305 <!-- IMAGEN 2 -->
306 <div class="card">
307 <div title="..." class="cover cover-small" style="background-image: url(https://derco-pe-prod.s3.amazonaws.com/images/models/2021-06-24-Gris%20480x320.jpg)">
308 </div>
309 <div class="card-body">
310 <h5 class="card-title">SUZUKI SWIFT</h5>
311 <span class="badge bg-danger">Precio antes: S/51,781.11</span>
312 <span class="badge bg-info text-dark">Precio ahora: S/51,381.00</span>
313 <p class="card-text">Presentamos el Swift, un auto con un precio atractivo y recomendable </p>
314 </div>

```

```

315 </div>
316 <!-- IMAGEN 3 -->
317 <div class="card">
318 <div title="..." class="cover cover-small" style="background-image: url(https://p4.wallpaperbetter.com/wallpaper/615/749/160/jimmy-suzuki-wallpaper-preview.jpg)">
319 </div>
320 <div class="card-body">
321 <h5 class="card-title">SUZUKI JIMNY</h5>
322 <span class="badge bg-danger">Precio antes: S/168,623.39</span>
323 <span class="badge bg-info text-dark">Precio ahora: S/168,622.39</span>
324 <p class="card-text"> Presentamos el Jimmy, construida para enfrentar el clima y le terreno más hostil, el Jimmy va donde otros vehiculos temen.</p>
325 </div>
326 </div>
327 <!-- IMAGEN 4 -->
328 <div class="card">
329 <div title="..." class="cover cover-small" style="background-image: url(https://p4.wallpaperbetter.com/wallpaper/68/801/242/cars-suzuki-wallpaper-preview.jpg)">
330 </div>
331 <div class="card-body">
332 <h5 class="card-title">SUZUKI VITARA</h5>
333 <span class="badge bg-danger">Precio antes: S/122,257.41</span>
334 <span class="badge bg-info text-dark">Precio ahora: S/121,957.00</span>
335 <p class="card-text">Presentamos el Vitara que es un vehiculo a todoterreno, este vehiculo es recomendado por B-SUV.</p>
336 </div>
337 </div>
338 <!-- IMAGEN 5 -->
339 <div class="card">
340 <div title="..." class="cover cover-small" style="background-image: url(https://s1.1zoom.me/o5050/242/Suzuki_2018-19_Maruti_Ertiga_White_background_576511_1600x1200.jpg)">
341 </div>
342 <div class="card-body">
343 <h5 class="card-title">SUZUKI ERTIGA</h5>
344 <span class="badge bg-danger">Precio antes: S/ 78,103.53</span>
345 <span class="badge bg-info text-dark">Precio ahora: S/ 77,703.00</span>
346 <p class="card-text"> Presentamos el nuevo Ertiga - un auto que te permitirá ir más allá, recomendada por muchos</p>
347 </div>
348 </div>
349 <!-- IMAGEN 6 -->
350 <div class="card">
351 <div title="..." class="cover cover-small" style="background-image: url(https://derco-pe-prod.s3.amazonaws.com/medias/suzuki/migration/front-image/apv-furgon/APV-FURGON-de)">
352 </div>
353 <div class="card-body">

```

```

354 <h5 class="card-title">SUZUKI APV</h5>
355 <span class="badge bg-danger">Precio antes: $/ 69,877.76</span>
356 <span class="badge bg-info text-dark">Precio ahora: $/ 69,477.00</span>
357 <p class="card-text">APV Minivan es la mejor alternativa para tu familia.</p>
358 </div>
359 </div>
360 <!-- IMAGEN 7 -->
361 <div class="card">
362 <div title="..." class="cover cover-small" style="background-image: url('https://www.diaromotor.com/imagenes/picscache/375x250c/suzuki-baleno-ficha-1118-902_375x250c.jpg')">
363 </div>
364 <div class="card-body">
365 <h5 class="card-title">SUZUKI BALENO</h5>
366 <span class="badge bg-danger">Precio antes: $/35,740.89</span>
367 <span class="badge bg-info text-dark">Precio ahora: $/35,440.00</span>
368 <p class="card-text">Con el nuevo Suzuki Baleno, cualquier momento es un buen momento lleno de aventura y diversión.</p>
369 </div>
370 </div>
371 <!-- IMAGEN 8 -->
372 <div class="card">
373 <div title="..." class="cover cover-small" style="background-image: url('https://www.portalautomotriz.com/sites/portalautomotriz.com/files/media/photos/suzuki-ciaz-1.jpg')">
374 </div>
375 <div class="card-body">
376 <h5 class="card-title">SUZUKI CIAZ</h5>
377 <span class="badge bg-danger">Precio antes: $/54,600.12</span>
378 <span class="badge bg-info text-dark">Precio ahora: $/54,400.00</span>
379 <p class="card-text">Presentamos el Ciaz, no vas a creer cuánto te va a gustar.</p>
380 </div>
381 </div>
382 </div>
383 </div>
384 </div>
385 </section>
386 <section>
387 <div class="container mt-5 mb-5" id="mapa">
388 <h3>ENCUENTRANOS:</h3>
389 <p>PILLCOMARCA</p>
390 <a href="#inicio">Inicio</a>
391 <br>
392 <div class="responsive-iframe">

```

```

393 <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1n3!1e868.9698169074301!2d-76.2506958164307!3d-9.949350457539182!2m3!1f0!2f0!3m2!1!1624!2!768!4!13!1!3m!1m2
394 </div>
395 </div>
396 </section>
397 <!-- MISIÓN - VISIÓN - CONTACTANOS - SIGUIENOS -->
398 <div class="row">
399 <div class="col-12 bg-success p-2 text-dark bg-opacity-25">
400 <section>
401 <div class="container mt-5 mb-5">
402 <div class="row justify-content-center">
403 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
404 <div class="card">
405 <div class="card-body">
406 <center>
407 <h2 class="card-title">MISIÓN</h2></center>
408 <p class="card-text">Ser la empresa líder en la comercialización de vehículos y prestación de servicios post ventas, brindando soluciones a la medida de
409 <br>nuestros clientes. Ser reconocida por la calidad humana y profesional de nuestra gente, tanto por clientes como por proveedores.</p>
410 </div>
411 </div>
412 </div>
413 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
414 <div class="card">
415 <div class="card-body">
416 <center>
417 <h2 class="card-title">VISIÓN</h2></center>
418 <p class="card-text">Brindarte el mejor servicio y a la vez ser la mejor opción del mercado en venta de vehículos y servicios posteriores, y no solo por
419 <br>nuestros bajos precios sino por las facilidades que ofrecemos al cliente al momento de realizar una compra o cotización de un vehículo.</p>
420 </div>
421 </div>
422 </div>
423 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
424 <div class="card">
425 <div class="card-body">
426 <center>
427 <h2 class="card-title">CONTACTANOS</h2></center>
428 <div title="..." class="cover cover-small" style="background-image: url('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQvKUCtC8WJ-3Fdc9FEuE8EccTU679Q0d7RkoRdechnKv6
429 <br class="card-text">Teléfono: 924497701
430 <br class="card-text">Teléfono: 948164548
431 <br class="card-text">Teléfono: 949964308
432 <br class="card-text">Teléfono: 942951809

```

```

433 <br class="card-text">Teléfono: 935612046
434 </div>
435 </div>
436 </div>
437 <div class="col-12 col-sm-6 col-md-4 col-lg-3 mt-0">
438 <div class="card">
439 <div class="card-body">
440 <center>
441 <h2 class="card-title">SÍGUEMOS</h2></center>
442 <div title="..." class="cover cover-small" style="background-image: url(https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQ5s0-vQ0T-LjBoOdvN9gDf70xNZF2mkF_SVjSrVc8E9
443
444 <br><a id="facebookLink" href="https://www.facebook.com/profile.php?id=100072183277342" target="_blank" rel="noopener">
445 <span class="icon-fb">FACEBOOK</span>
446 </a><br>
447 <br><a id="instagramLink" href="https://www.instagram.com/vicashakedi/" target="_blank" rel="noopener">
448 <span class="icon-fb">INSTAGRAM</span>
449 </a><br>
450 <br><a id="facebookLink" href="https://twitter.com/vicashakedi" target="_blank" rel="noopener">
451 <span class="icon-fb">TWITTER</span>
452 </a><br>
453 </div>
454 </div>
455 </div>
456 </div>
457 </div>
458 </section>
459 </div>
460 </div>
461 </body>
462 </html>

```

Ejecución de la Página

venta-de-autos-vicashakedi-unheval-2021.netlify.app

Autos Ubicación Costear Vehículo

"VICASHAKEDI SAC."
VENTA DE AUTOMÓVILES

NUESTRAS MARCAS

Toyota Honda Suzuki

NUESTRAS MARCAS



TOYOTA

Toyota es una marca llena de historia y una de las empresas fabricantes de vehículos más grandes de Japón, que ha marcado el desarrollo del mercado automovilístico. Sin dudas, goza de la confianza, aceptación, liderazgo y fiabilidad de millones de usuarios en todo el mundo.



HONDA

Honda es una marca japonesa dedicada a la fabricación de motocicletas, automóviles, motores acuáticos, robots y hasta aeronaves. Fundada en 1959, es considerada como la compañía de motores de combustión interna más grande del mundo.



SUZUKI

Suzuki Motor Corporation es una empresa japonesa dedicada a la fabricación de automóviles, motocicletas, motores fuera borda, y variedad de productos equipados con motores de combustión. Fue fundada en 1909 en la localidad de Hamamatsu, Japón.

MODELOS DE LA MARCA TOYOTA

[Inicio](#)



TOYOTA COROLLA

Precio: S/106,736.00

Precio Después: S/106,000.00

ción del
marca
ntará con
o en

El Toyota Corolla ha sido el caballito de batalla de millones de familias por más de 50 años, recomendada por la empresa.



TOYOTA CAMRY

Precio: S/183,040.00

Precio Después: S/183,000

Llega a España el Toyota Camry Hybrid, el modelo híbrido que completa la gama Toyota Camry de 2018, este modelo es uno de los más exitosos y vendidos de nuestra empresa.



TOYOTA C-HR

Precio Antes: S/123,390.00

Precio Después: S/123,000.00

Su diseño futurista, prestaciones únicas y tecnología híbrida auto-recargable, harán de la c-hr, la mejor compañera para ti y el medio ambiente.



TOYOTA ETIOS

Precio Antes: S/55,965.00

Precio Después: S/55,000.00

El nuevo etios trae consigo una renovada máscara frontal que sorprenderá a muchos gracias a su diseño con un estilo más deportivo.

MODELOS DE LA MARCA HONDA

[Inicio](#)



HONDA ODYSSEY

Precio antes: S/227,509.00

Precio después: S/227,000.00

o el
vo de la
cine
rtido en

La Honda Odyssey 2021 es un vehículo de grandes dimensiones. Como beneficio, se adjudica un excelente espacio interior, y visualmente podrá comprobarlo en los estacionamientos.



HONDA CR.V

Precio antes: S/135,259.00

Precio después: S/135,000.00

El Honda CR-V es un automóvil todoterreno del segmento C producido por el fabricante japonés de automóviles Honda.



HONDA ACCORD

Precio antes: S/139,359

Precio después: S/139,000

La marca Honda Accord 2021 presenta un modelo deportivo con una manejabilidad óptima para la ciudad



HONDA PILOT

Precio antes: S/163,959

Precio después: S/160,000

El Honda Pilot 2021 acaba de aterrizar en Perú junto a su última actualización, acompañado de nuevas distinciones que le permiten continuar peleando en el segmento más competitivo del mercado automotriz.

MODELOS DE LA MARCA SUZUKI

[Inicio](#)



SUZUKI JIMNY

Precio antes: S/168.623.39
Precio ahora: S/168.622.39

Presentamos el Jimny, construida para enfrentar el clima y le terreno más hostil, el Jimny va donde otros vehículos temen.



SUZUKI VITARA

Precio antes: S/122.257.44
Precio ahora: S/121.957.00

Presentamos el Vitara que es un vehículo a todoterreno, este vehículo es recomendado por B-SUV.



SUZUKI ERTIGA

Precio antes: S/ 78.103.53
Precio ahora: S/ 77.783.00

Presentamos el nuevo Ertiga – un auto que te permitirá ir más allá, recomendada por muchos



SUZUKI APV

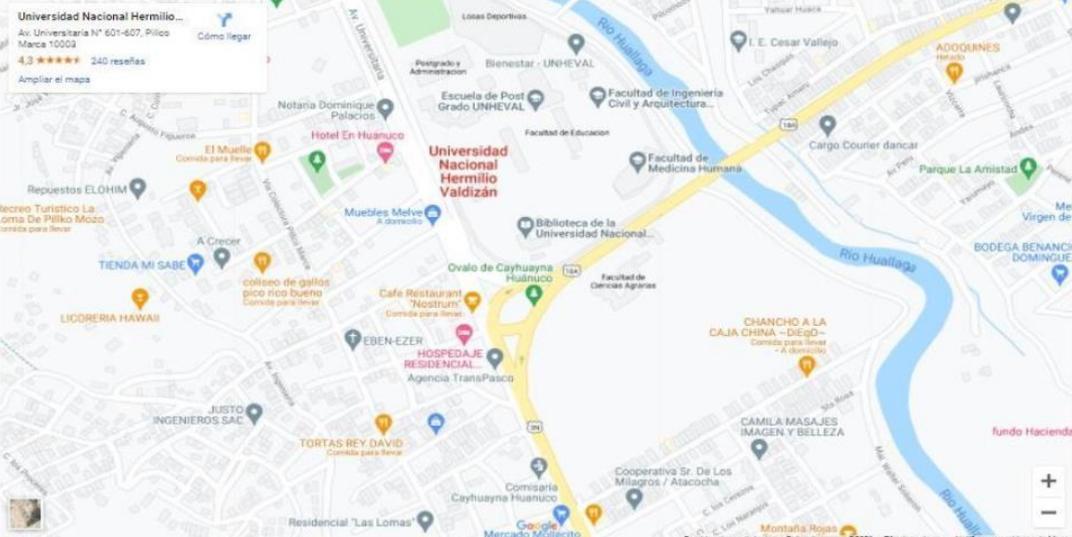
Precio antes: S/ 69.877.76
Precio ahora: S/ 69.477.00

APV Minivan es la mejor alternativa para tu familia.

ENCUENTRANOS:

PILLCOMARCA

[Inicio](#)



The map displays the Píllcomarca area, featuring the Universidad Nacional Hermilio Valdizán and the Río Huallaga. Numerous points of interest are marked, including 'El Muelle', 'Tienda Mi Sabe', 'Llibrería Hawaii', 'Café Restaurant Nostrom', 'Hospedaje Residencial', 'Tortas Rey David', 'Cajita China', and 'Camila Masajes Imagen y Belleza'. A search box in the top left corner shows 'Universidad Nacional Hermilio...' with a 4.3 star rating and 240 reviews.

MISIÓN

Ser la empresa líder en la comercialización de vehículos y prestación de servicios post ventas, brindando soluciones a la medida de nuestros clientes. Ser reconocida por la calidad humana y profesional de nuestra gente, tanto por clientes como por proveedores.

VISIÓN

Brindarte el mejor servicio y a la vez ser la mejor opción del mercado en venta de vehículos y servicios posteriores, y no solo por nuestros bajos precios sino por las facilidades que ofrecemos al cliente al momento de realizar una compra o cotización de un vehículo.

CONTÁCTANOS



Teléfono: 924497701
 Teléfono: 948164548
 Teléfono: 940964308
 Teléfono: 942951809
 Teléfono: 935622046

SÍGUENOS





[FACEBOOK](#)
[INSTAGRAM](#)
[TWITTER](#)

SELECCIONE EL AUTOMÓVIL DE SU PREFERENCIA

MARCAS DE AUTOMÓVILES: **TOYOTA** ▾

Usted Escogio la Marca: **TOYOTA**
HONDA
SUZUKI

MODELOS:

* Seleccione el modelo de su automóvil, según la marca escogida anteriormente.

MODELOS DE TOYOTA: **AURIS** ▾

El modelo que escogio es:

MODELOS DE HONDA: **JAZZ** ▾

El modelo que escogio es:

MODELOS DE SUZUKI: **CELERIO** ▾

El modelo que escogio es:

ENVIAR



USTED ESCOGIO EL MODELO CR-V DE LA MARCA HONDA

PRECIO AL CONTADO: S/135,000.00

CARACTERISTICAS

MOTOR: HP 115 Torque: 160 Nm

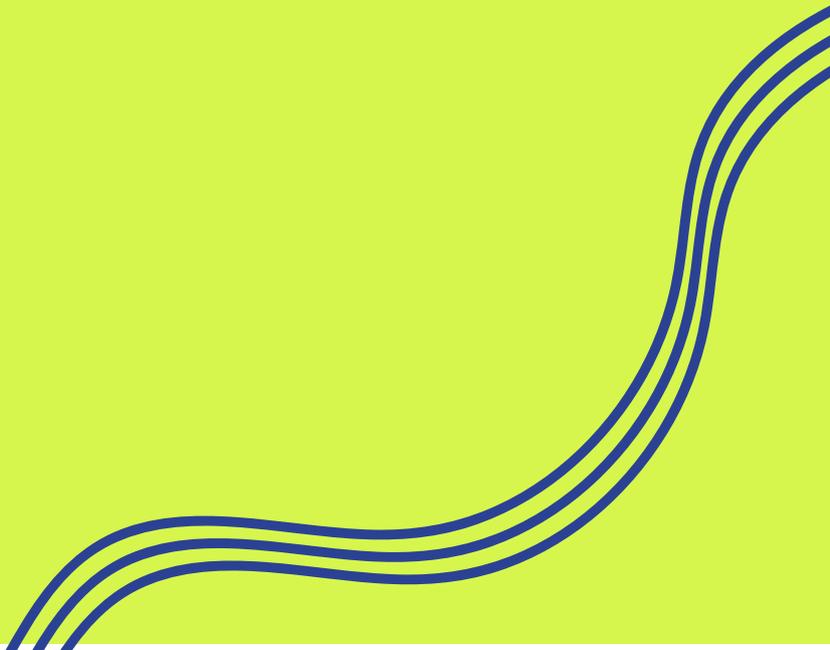
CONECTIVIDAD : WIFI 4G

CONFORD: Easy Park

SEGURIDAD: 6 Airbags

CAPÍTULO IV

SQL y Sistemas
Gestores de Base de
Datos: SQL Server,
MySQL, MariaDB,
SQLite, MongoDB



SQL (Structured Query Language)

Lenguaje de consulta estructurado, estándar que permite interactuar con bases de datos relacional en: Definición (Lenguaje de definición de datos-LDD), consulta y Manipulación de datos (Lenguaje interactivo de manipulación de datos-LMD) y control de datos y transacciones.

4.1. DDL (Lenguaje de Definición de Datos)

Es el encargado de la gestión de la estructura de los objetos de la base de datos con cuatro operaciones: **CREATE**, **ALTER**, **DROP** y **TRUNCATE**.

CREATE

Esta sentencia permite crear objetos de base de datos: base de datos, tablas, vistas, procedimientos almacenados, funciones, entre otros.

Por ejemplo, para crear una base de datos, una tabla:

```
- CREATE DATABASE sis_academico;  
- CREATE TABLE table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

Sintaxis en MySQL

```
CREATE TABLE alumno (  
    codigo varchar(10),  
    apellidos varchar(50),  
    nombre varchar(50),  
    direccion varchar(50),  
    fecha_nacimiento date  
)
```

```
CREATE TABLE curso(  
    codigo char(10),  
    nombre varchar(50),  
    credito int  
)
```

ALTER

Esta sentencia permite modifica la estructura de un objeto de base de datos. Se puede modificar, quitar campos de una tabla, vistas, entre otros.

Por ejemplo, se agrega una columna a la tabla alumno:

```
- ALTER TABLE alumno ADD correo varchar(50);
```

DROP

Esta sentencia permite eliminar un objeto de la base de datos, puede ser base de datos, tabla, vista, procedimiento almacenado, entre otros.

Por ejemplo, para eliminar una tabla “curso” la sintaxis seria así:

```
- DROP TABLE curso;
```

TRUNCATE

Esta sentencia permite borrar el contenido de una tabla, por ejemplo, para borrar todo el contenido de la tabla alumnos escribiríamos así:

```
- TRUNCATE TABLE alumno;
```

4.2. DML (Lenguaje de Manipulación de Datos)

Permite alterar el contenido de los registros de una tabla, realizar consultas, las cuatro operaciones fundamentales de la gestión de toda base de datos: **INSERT**, **UPDATE**, **DELETE**, **SELECT**.

INSERT

Esta sentencia permite agregar una o más registros a una tabla relacional, la cantidad de columnas y valores tienen que ser iguales y del mismo tipo asignado en su definición.

Sintaxis:

```
INSERT INTO
    nombre_tabla (column1, column2, ...)
VALUES
    ('valor1', 'valor2', ...)
```

Insertando múltiples registros o filas:

```
INSERT INTO
    nombre_tabla (column1, column2, ...)
VALUES
    ('valor11', 'valor21', ...),
    ('valor12', 'valor22', ...),
    ('valor13', 'valor23', ...),
```

Insertando todas las columnas:

```
INSERT INTO nombre_tabla VALUES ('valor1', 'valor2', ...)
```

Insertando desde una consulta:

Por ejemplo, insertando un registro a la tabla alumno:

```
INSERT INTO nombre_tabla SELECT 'valor1','valor2',... FROM
nombre_tabla_consulta
```

```
UPDATE alumno
SET direccion='JR. PUNO #4587',correo='ana@gmail.com'
WHERE codigo='2023012890'
```

```
INSERT INTO
    alumno(codigo,apellido,nombre,dirección,
    fecha_nacimiento,correo)
VALUES
    ('2023012354', 'CHUQUIYAURI SALDIVAR','ELMER', 'JR.
```

Insertando varios registros a la tabla alumno:

```
INSERT INTO
    alumno(codigo,apellidos,nombre,direccion,
    fecha_nacimiento,correo)
VALUES
    ('2023012888', 'GARCIA CHAVEZ','MIRANDA', 'JR. 28 DE
    JULIO #125', '01-06-1990', 'miranda@gmail.com'),
    ('2023012889', 'MARTINEZ SANDOVAL','JUAN', 'JR. DOS DE
    MAYO #658', '06-30-1998', 'juan@gmail.com'),
```

UPDATE

Esta sentencia permite modificar valores de los campos que se requiera.

Sintaxis:

```
UPDATE nombre_tabla
SET campoa='valor1',campo2='valor2',...
WHERE proposicion
```

Por ejemplo, para modificar la dirección del alumno “COTRINA CONDOR” cuyo código es “2023012890”:

DELETE

Esta sentencia borra una o más registros de una tabla.

Sintaxis:

```
DELETE FROM nombre_tabla WHERE proposicion
```

Para borrar al alumno “COTRINA CONDOR” cuyo código es “2023012890”.

```
DELETE FROM alumno WHERE codigo='2023012890'
```

SELECT

Esta sentencia permite realizar consultas a los datos de las tablas de una base de datos.

Sintaxis básica:

```
SELECT [{ALL|DISTINCT}]
    column1, column2,...

FROM {<nombre_tabla1>|<nombre_vista1>}[,
    {<nombre_tabla2>|<nombre_vista2>}...]

[WHERE <proposicion1> [{AND|OR} <proposicion2>...]]

[GROUP BY <column1>[, <column2>...]]
```

SELECT	Con esta sentencia se seleccionan registros con columnas de uno o más tablas dependiendo de la condición del WHERE. el ALL anteponiendo a una columna indica que se seleccionaran todos los registros. Anteponer el DISTINCT a una columna seleccionara solo los valores distintos.
FROM	Aquí se indica la tabla o tablas de donde se obtendrán los datos
WHERE	Aquí se establece la condición o proposición según el cual se escogerán los registros, para generar proposiciones compuestas se utiliza los conectores lógicos AND o OR

GROUP BY	Con esto se agrupan registros de una tabla dependiendo de lo que se requiera con la columna establecida.
HAVING	Esto es similar al de WHERE , pero aplicado a los registros devueltos por la consulta. Se aplica junto a GROUP BY , la condición debe estar referida a las columnas contenidos en ella.
ORDER BY	Esto permite ordenar la consulta de forma ascendente(ASC) o descendente(DESC) por la columna que se indique.

EJEMPLO N° 04-001.

En la FIGURA N° 04-001 se muestra parte de un modelo lógico de una base de datos de facturación con algunas tablas y sus relaciones entre ellos:

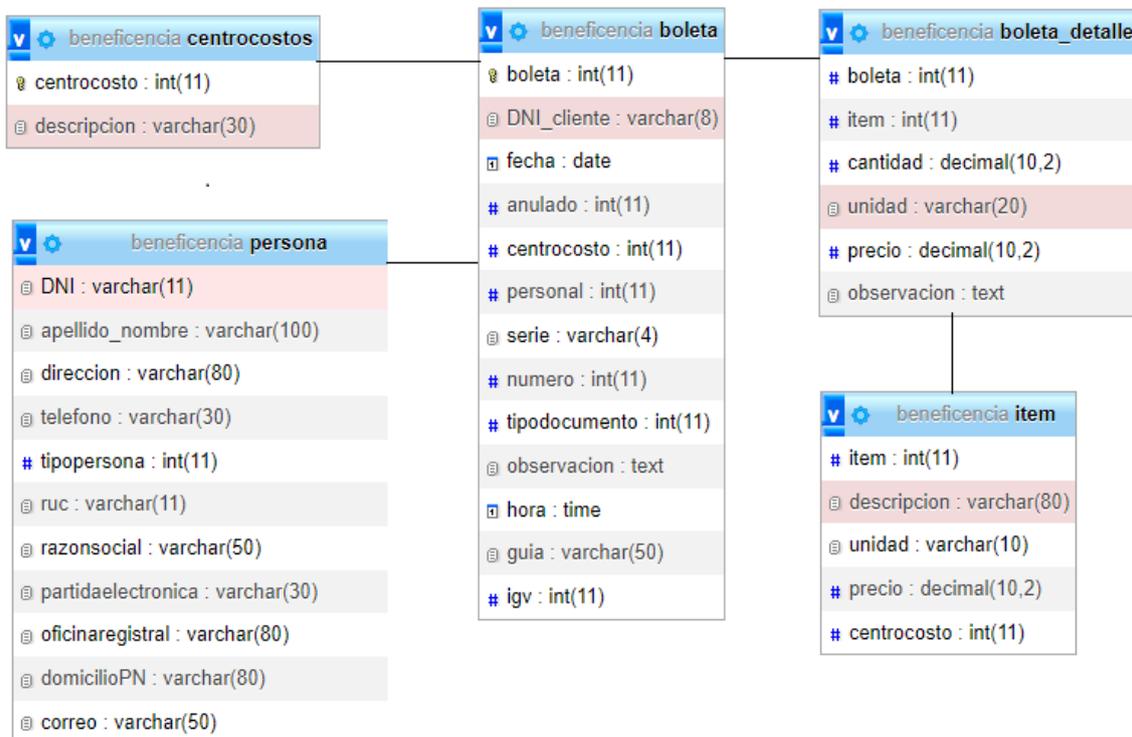


FIGURA N° 04-001: Diagrama que muestra algunas tablas de una base de datos de facturación

- Del diagrama mostrado en la FIGURA N° 04-001, realizar una consulta para que me muestre los apellidos y nombres de los clientes con sus respectivas cantidades de boletas y ordenado por apellido.
- Del diagrama mostrado en la FIGURA N° 04-001, realizar una consulta para que me muestre los apellidos y nombres de los clientes con sus respectivas cantidades de ítems por boleta.

DESARROLLO

- a) Para realizar la consulta requerida se consultará a la tabla **bolete** y a la tabla **persona** que están relacionado por la columna DNI, así mismo se agrupara por la columna DNI para contar la cantidad de boletas por persona.

Consulta:

```
SELECT p.apellido_nombre, COUNT(b.boleta) AS cantidad
FROM boleta AS b, persona AS p
WHERE b.DNI_cliente=p.DNI
```

Resultado consulta:

Mostrando filas 0 - 24 (total de 101, La consulta tardó 0,0816 segundos.)

```
SELECT p.apellido_nombre, COUNT(b.boleta) as cantidad FROM
boleta AS b, persona AS p WHERE b.DNI_cliente=p.DNI GROUP BY
b.DNI_cliente ORDER BY cantidad DESC;
```

Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

1 > >> | Mostrar todo | Número de filas: 25

Opciones extra

apellido_nombre	cantidad
DATOS DE PRUEBA	5
EULOGIO CESPEDES, FLAVIO	3
QUISPE ARCE, KATERINE LIZ	3
OSORIO PILLCO, JOSEFINA	3
GONZALES ACUÑA, YRENE BEATRIZ	3
MALLQUI NIETO, MARCOS BRUCE	2
OBREGON TARAZONA, PATRICIA ALEJANDRA	2
BAZAN BECERRA, JAIME	2

FIGURA N° 04-002: Resultado de la consulta realizada

- b) Para realizar la consulta requerida se consultará a la tabla **bolete**, **boleta_detalle** y la tabla **persona**, la tabla **boleta** está relacionado con la tabla **persona** por la columna **DNI**, la tabla **boleta** está relacionado con la tabla **boleta_detalle** por la columna **boleta**, así mismo se agrupa por la

columna **boleta** de la tabla **boleta_detalle** de la tabla **boleta_detalle** para contar la cantidad de ítems por boleta y se ordena por apellido

Consulta:

```

SELECT p.apellido_nombre,b.boleta, COUNT(bd.item) AS cantidad
FROM boleta AS b, boleta_detalle AS bd, persona AS p
WHERE b.boleta=bd.boleta AND b.DNI_cliente=p.DNI
GROUP BY bd.boleta ORDER BY p.apellido_nombre
    
```

Resultado consulta:

The screenshot shows a database query tool interface. At the top, the SQL query is displayed in a text area. Below the query, there are several control buttons: 'Perfilando' (checked), 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. Below these buttons, there are options for 'Mostrar todo' (checked), 'Número de filas' (set to 25), and 'Filtrar filas' (with a search box containing 'Buscar en esta tabla'). At the bottom, there is a table with the following data:

apellido_nombre	boleta	cantidad
ALETOIS PEREZ	22	10
ANDRES FABIAN	1	7
CALERO Y ACOSTA, FORTUNATO	34	5
CORDOVA ROQUE ,Maria Jasus	53	1
DATOS DE PRUEBA	122	5
DAYRON YARA	21	13
DURAND LEANDRO, YERSY	39	8
FELICIANO PRINCIPE	2	15
FIERRO REQUENA ELIZABETH	9	8
GOMEZ MEGIA	4	46
GOMEZ MEGIA	4	47
GOMEZ MEGIA	4	58
LANDEO PINCHES, JAHZEEL ISABEL	19	1
LANDEO PINCHES, JAHZEEL ISABEL	19	8
MALLQUI NIETO, MARCOS BRUCE	28	7
QUISPE ARCE, KATERINE LIZ	27	6

FIGURA N° 04-003: Resultado de la consulta realizada

4.3. DCL (Lenguaje de Control de Datos)

Estas sentencias permiten controlar el acceso a los objetos de una base de datos, otorgando o denegando permisos a uno o más usuarios para realizar determinadas tareas.

Los comandos que se utilizan para ello son: **GRANT** y **REVOKE**

GRANT

Con este comando se otorga permisos

REVOKE

Con este comando se quita permisos que previamente se han concedido con el comando GRANT.

4.4. TCL (Transaction Control Language)

Lenguaje de control de transacción en una base de datos, esto es; agrupar operaciones en un solo concepto, para indicar a la base de datos que haga todas las operaciones o ninguno. Para esto se utiliza estos comandos: COMMIT, ROLLBACK y SAVEPOINT.

COMMIT

Con este comando se graba todos los cambios en la base de datos, completando así la transacción. Es importante indicar el inicio de cada transacción.

ROLLBACK

Si hay algún inconveniente en la transacción se puede deshacer las operaciones o cambios realizados en la base de datos, para ello se utiliza este comando ROLLBACK.

SAVEPOINT

Este comando permite definir un punto de salvaguarda dentro de una transacción.

4.5. Gestor de Base de Datos

SQL Server

SQL Server es un sistema de gestión de base de datos relacional cuya función principal es la de almacenar y recuperar datos según lo requiera otras aplicaciones. Fue desarrollado por **Microsoft** por lo que su uso depende de una licencia, pero también se cuenta con versión gratuita SQLServer Express y SQLServer Developer, este último tiene todas las características que se puede usar como servidor de base de datos de desarrollo y pruebas en los entornos de no productivos.

Algunas características:

- ✓ Soporte de transacciones.
- ✓ Escalabilidad, estabilidad y seguridad.
- ✓ Soporte de procedimientos almacenados.
- ✓ Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- ✓ Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red solo acceden a la información.
- ✓ Permite administrar información de otros servidores de datos.
- ✓ Versiones de SQL Server.

Últimas versiones del SQLServer:

- ✓ Microsoft SQL Server 2022
- ✓ Microsoft SQL Server 2019 (64 bits)
- ✓ Microsoft SQL Server 2019 en Linux (64 bits)
- ✓ Microsoft SQL Server 2017 (64 bits)
- ✓ Microsoft SQL Server 2017 en Linux (64 bits)
- ✓ Microsoft SQL Server 2016 (64 bits)
- ✓ Microsoft SQL Server 2014 SP3 (64 bits)

Instalación y configuración de SQL Server 2022 Express

Para su instalación se descarga desde la página oficial de Microsoft:

<https://www.microsoft.com/es-mx/sql-server/sql-server-downloads>

Seleccionar la versión requerida:

O descarga una edición especializada gratis.



Desarrollador

SQL Server 2022 Developer es una edición gratuita con todas las características, que se puede usar como base de datos de desarrollo y pruebas en los entornos no productivos.

Descargar ahora



Express

SQL Server 2022 Express es una edición gratuita de SQL Server, que es ideal para el desarrollo y la producción, para aplicaciones de escritorio, Internet y pequeños servidores.

Descargar ahora

FIGURA N° 04-004: Descarga del instalador del SQLServer2022 Express

Una vez que se descargue se ejecuta el archivo de instalación.

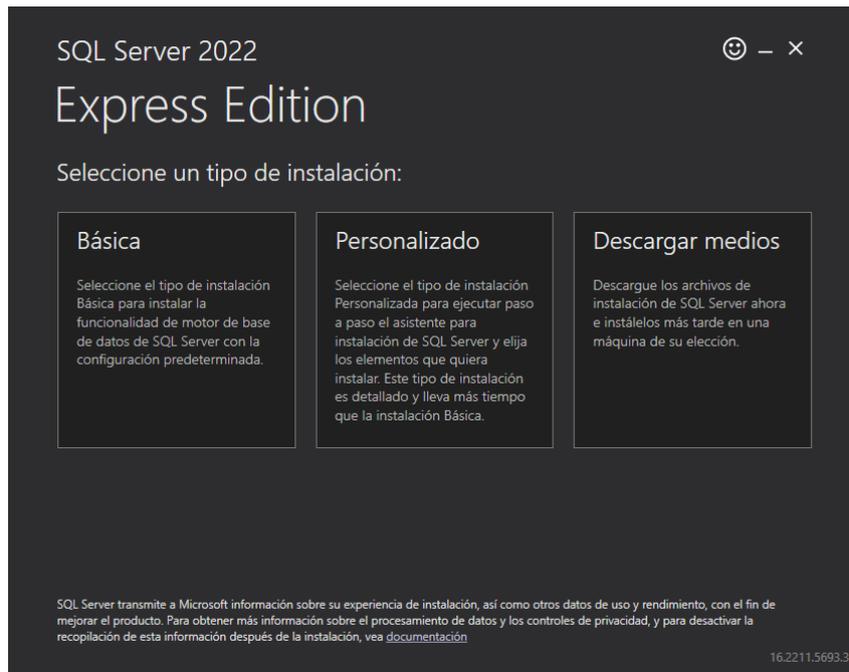


FIGURA N° 04-005: Instalación de SQLServer2022 Express

Se elige la **Personalizado**. La instalación predeterminada está en inglés. Para la instalación en español, el **Windows del sistema operativo se tiene que configurar a español**.



FIGURA N° 04-006: Instalación Personalizado de SQLServer2022 Express

Click en si para continuar:

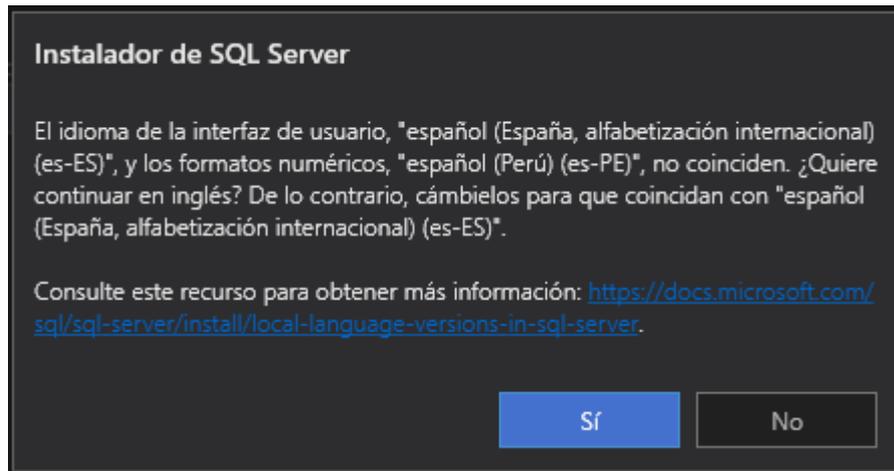


FIGURA N° 04-007: Click en Si para continuar

Click en instalar:

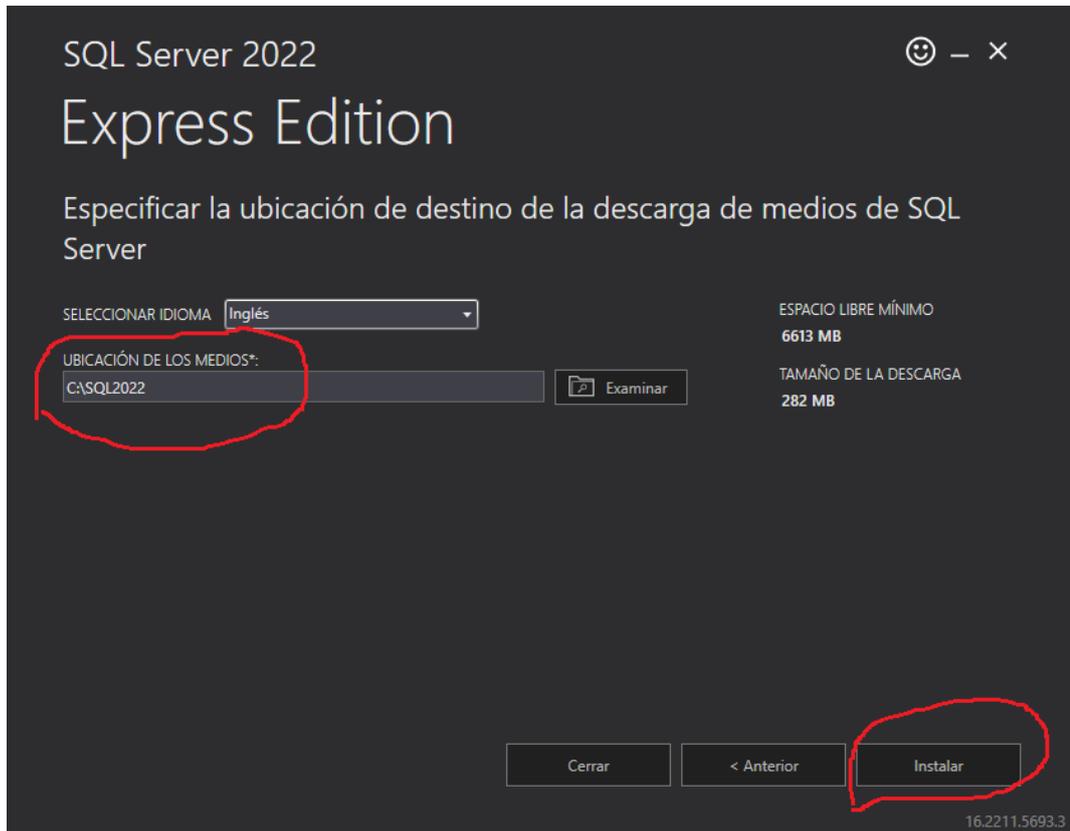


FIGURA N° 04-008: Click en instalar para continuar

El proceso de descarga de paquetes de instalación:

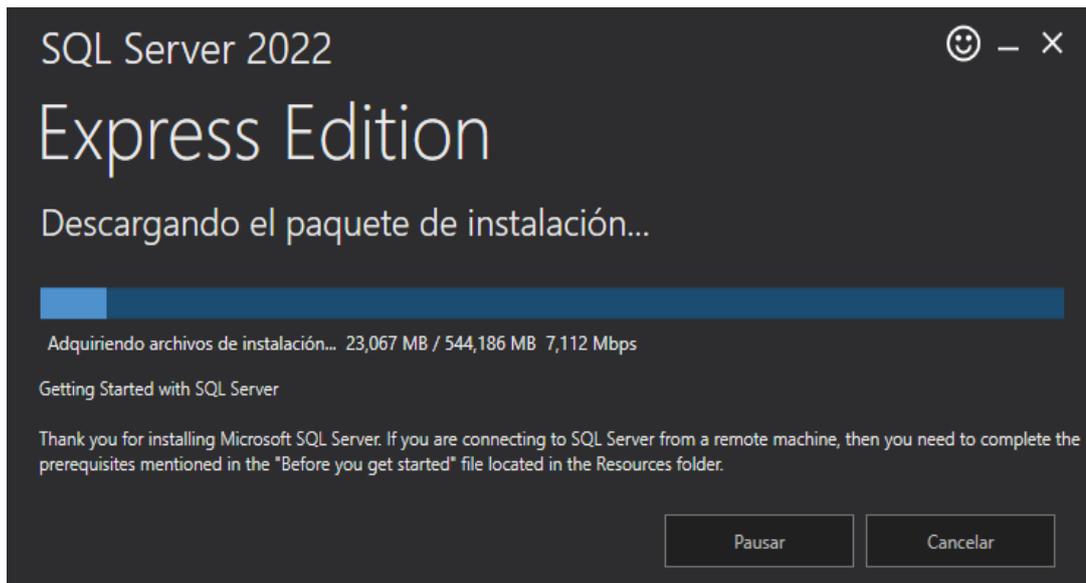


FIGURA N° 04-009: En proceso de descarga paquetes de instalación

Una vez terminado la descarga de paquetes, saldrá la ventana mostrada en la FIGURA N° 04-010, para poder instalar SQLServer2022, del cual seleccionamos New SQL Server, para instalar una nueva instancia.



FIGURA N° 04-010: instalación de una nueva instancia

Check en aceptar los términos de la licencia y Next:

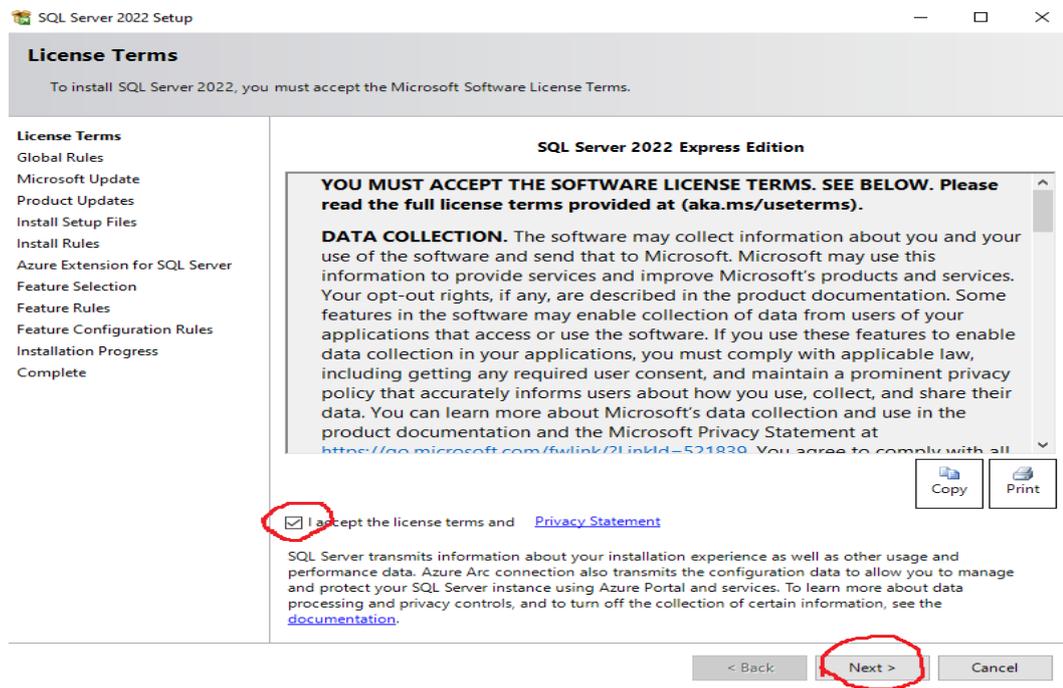


FIGURA N° 04-011: Check en aceptar licencia y Next

Deseleccionar Azure Extension for SQL Server:

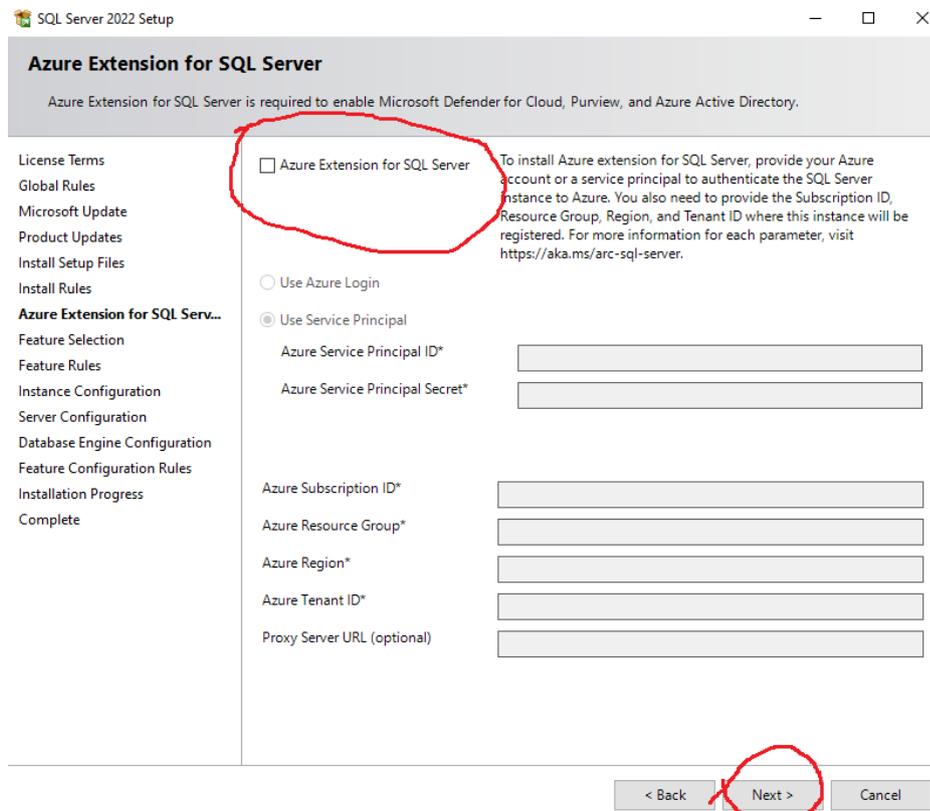


FIGURA N° 04-012: Deseleccionar Azure Extension for SQL Server y Next

Click en Next para proseguir con la instalación:

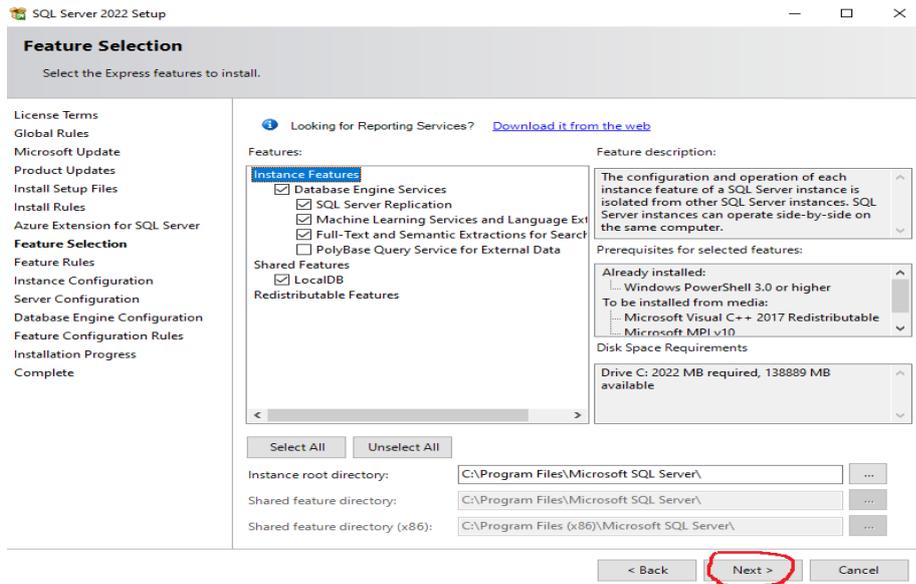


FIGURA N° 04-013: Next para proseguir la instalación

Si se desea crear una instancia nueva seleccionar **Named instance** e ingresar el nombre, tal como se muestra en la FIGURA N° 04-014:

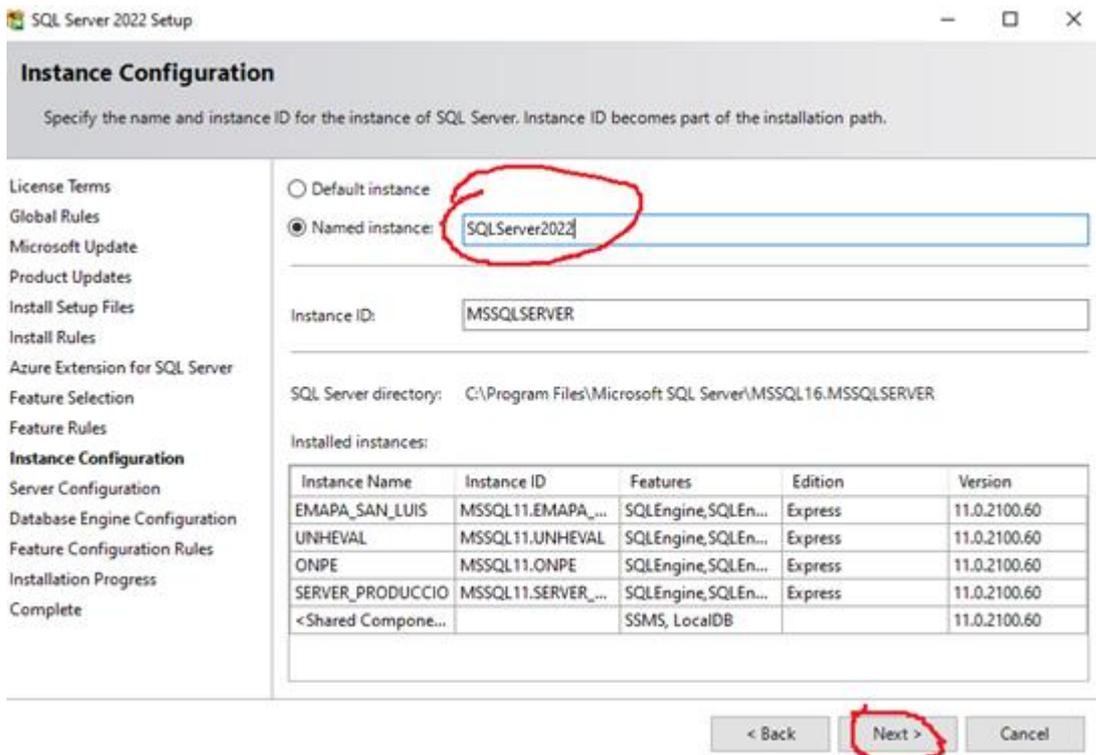


FIGURA N° 04-014: Nombre de la nueva instancia

En la FIGURA N° 04-015 se muestra la forma de Authentication, en este caso Mixed y se ingresa el password del súper usuario "sa":

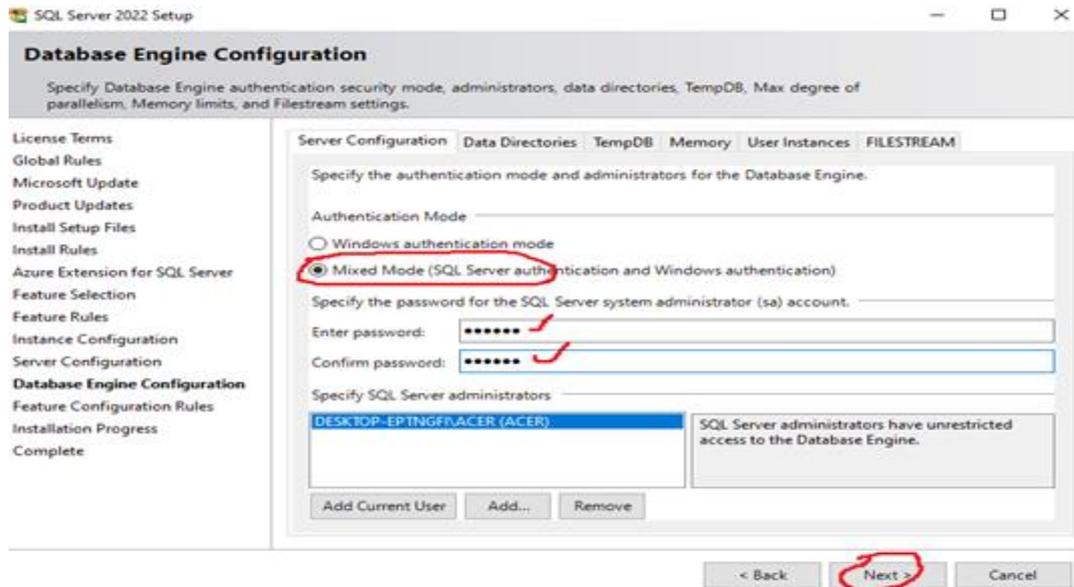


FIGURA N° 04-015: Authentication Mixed y se ingresa clave del 'sa'

Una vez terminado la instalación si todo está ok saldrá la ventana sugiriendo el reinicio de la computadora, así como se muestra en la FIGURA N° 04-016:

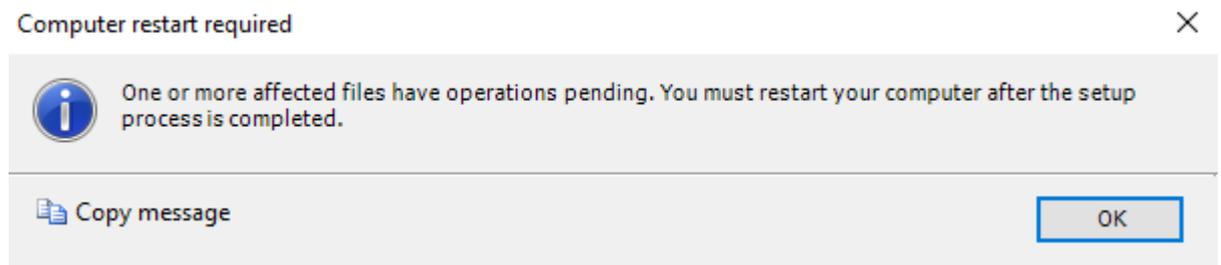


FIGURA N° 04-016: Authentication Mixed y se ingresa clave del 'sa'

Para ver la instancia de SQL Server 2022, en el inicio de Windows seleccionamos:

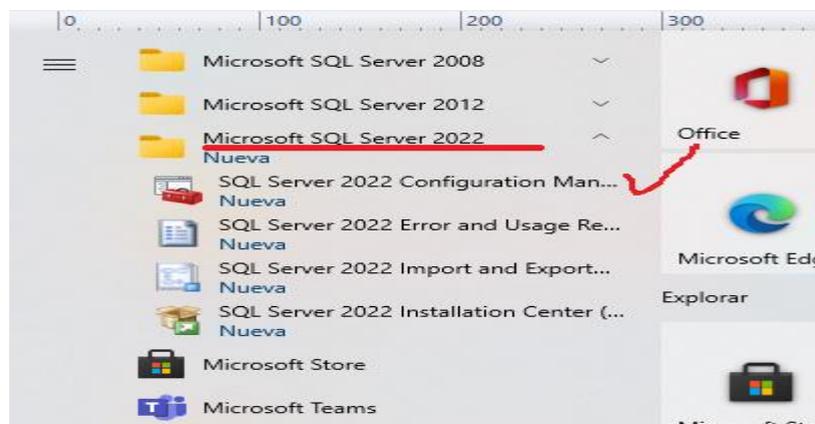


FIGURA N° 04-017: En el menú de inicio de Window ya figura el SQL Server

En la figura FIGURA N° 04-018 se puede apreciar que la nueva instancia instalada está corriendo: SQL Server (SQLSERVER2022)

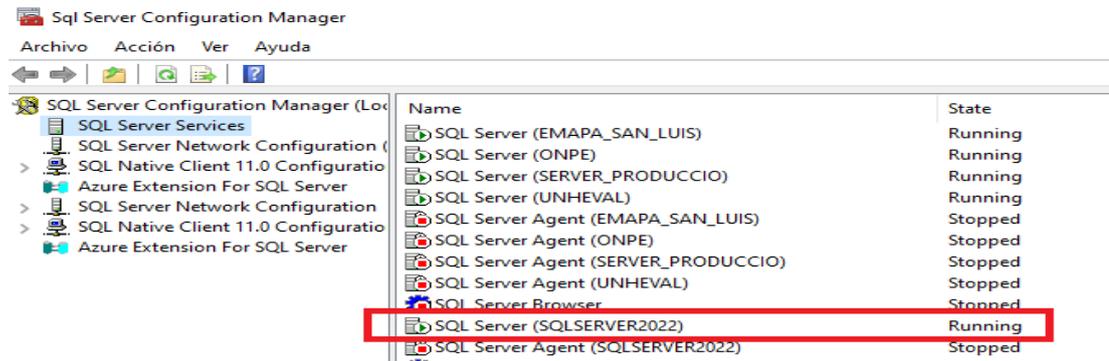


FIGURA N° 04-018: La instancia SQLSERVER2022 está corriendo

Para poder tener un entorno de trabajo amigable y poder gestionar nuestra base de datos se necesita instalar el SQL Server Management Studio(SSMS)

“SQL Server Management Studio (SSMS) es un entorno integrado para administrar cualquier infraestructura de SQL, desde SQL Server hasta Azure SQL Database. SSMS proporciona herramientas para configurar, monitorear y administrar instancias de SQL Server y bases de datos. Use SSMS para implementar, monitorear y actualizar los componentes del nivel de datos que usan sus aplicaciones y crear consultas y scripts. Use SSMS para consultar, diseñar y administrar sus bases de datos y almacenes de datos, donde sea que estén, en su computadora local o en la nube”.

<https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>

Instalación de SQL Server Management Studio (SSMS)

Para su instalación se descarga desde la página de Microsoft:

<https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>

En la FIGURA N° 04-019, se muestra el entorno de trabajo del SQL Server Management Studio, con las bases de datos creado por defecto:

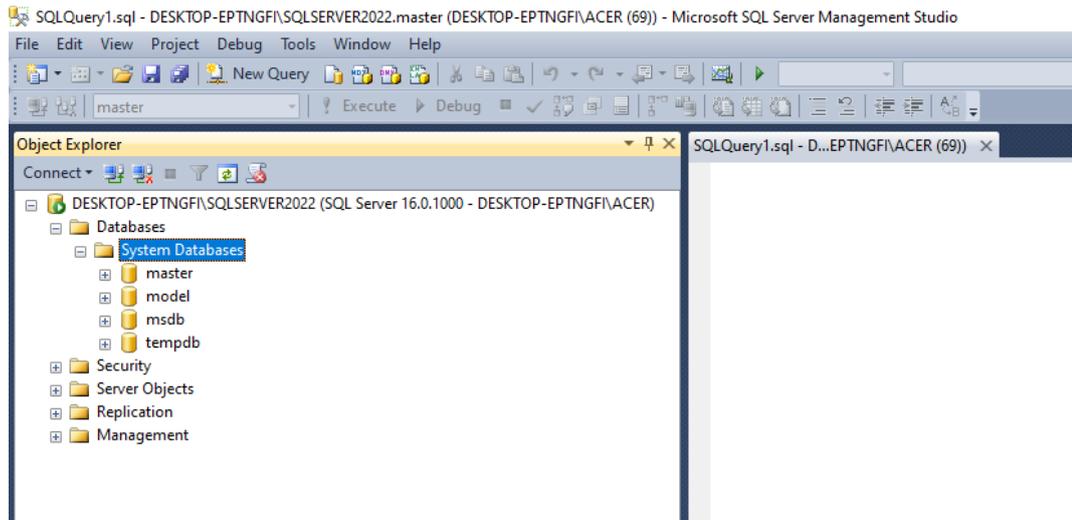


FIGURA N° 04-019: Entorno de trabajo del SQL Server Management Studio

Para crear una base de datos, click derecho sobre Databases, tal como se muestra en la FIGURA N° 04-020:

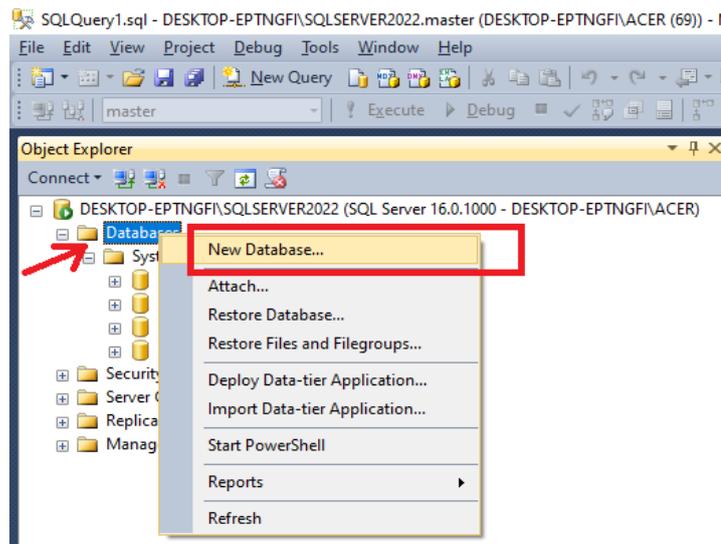


FIGURA N° 04-020: Creación de una base de datos

En la ventana que aparece en la FIGURA N° 04-021, se ingresa el nombre de la base de datos, en este caso: ***sis_logistica***

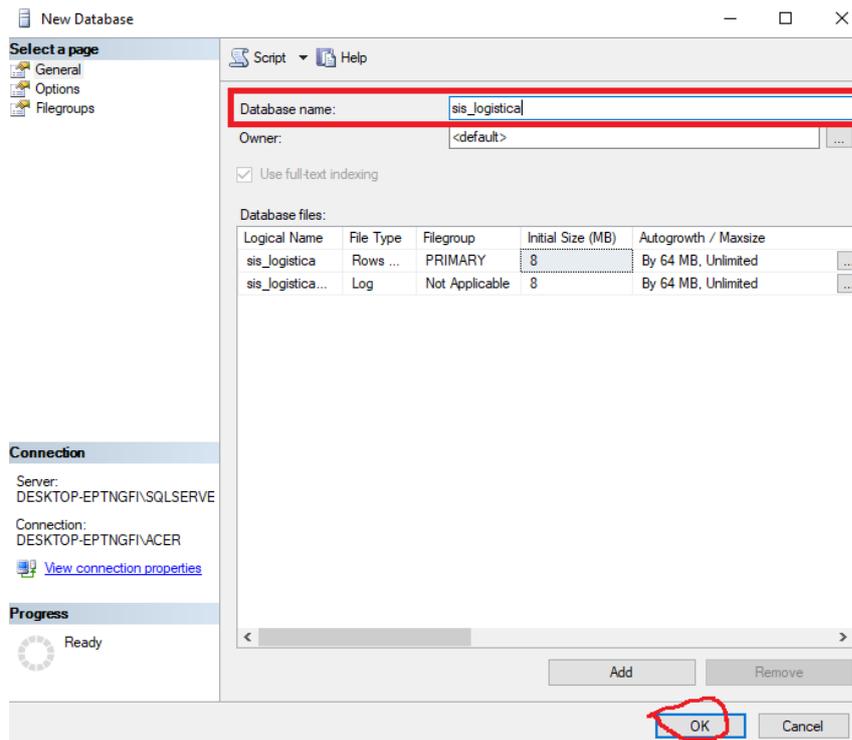


FIGURA N° 04-021: Creación de la base de datos “sis_logistica

También se puede crear la base de datos utilizando sentencia SQL:

CREATE DATABASE sis_logistica, así como se muestra en la FIGURA N° 04-022

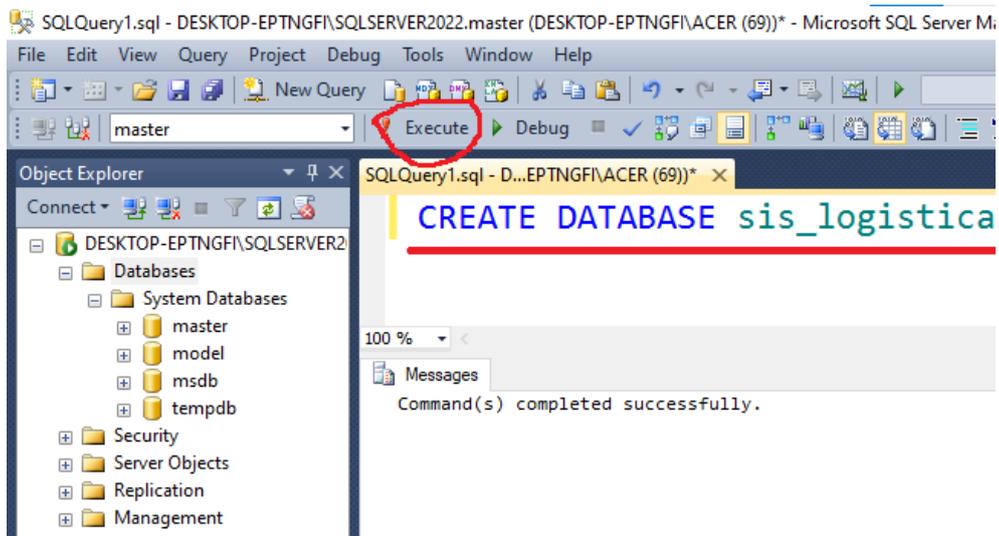


FIGURA N° 04-021: Creación de base de datos, con sentencia SQL

EJEMPLO N°04-002

Se tiene el siguiente sistema para desarrollar:

En el proceso de “**Bienes y Suministros**” de la empresa de turismo “**Pata Amarilla**” que pertenece al área de Administración, se realizan las actividades de manera manual, por lo que se requiere sistematizar todo ello con la ayuda de un software, el dueño del proceso de “**Bienes y Suministros**” detalla el procedimiento y actividades de cada área:

- ✓ El proceso involucra cinco (5) áreas: Administración, Logística, Almacén, Patrimonio, Áreas usuarias.
- ✓ El área de Logística funciona de la siguiente forma:
 - Recibe las solicitudes de requerimientos de bienes o suministros de las diferentes áreas de la empresa.
 - Cada solicitud tiene un responsable que está adscrito a un área.
 - Cada solicitud es autorizada por el jefe del área y posteriormente por el área de Administración.
 - De la solicitud se requiere la siguiente información: Número de la solicitud (consecutivo), fecha, responsable, área, meta presupuestal. En cada solicitud se pueden contener uno o muchos ítems con la siguiente información: código, nombre del ítem, cantidad solicitada, unidad de medida, valor unitario.
 - Cada ítem es identificado por un código único y es de carácter devolutivo (suministro) (útiles de escritorio, útiles de limpieza, etc.) o un bien (computadora, escritorio, etc.).
 - La solicitud es enviada al área de Logística para atender si es que hay en stock o realizar la compra a uno o varios proveedores. Para realizar la compra se juntan todas las solicitudes para tener la cantidad total de ítems de cada rubro a comprar.
 - Las cotizaciones son realizadas con uno o varios proveedores de los bienes solicitados.
 - Para de un producto, primero se realiza la cotización a dos o tres proveedores para determinar la mejor oferta en calidad y precio que ofrecen, una vez elegido el proveedor se genera la orden de compra, con los siguientes datos: número de **orden de compra**, nombre del

proveedor al cual se le va a realizar la compra, fecha de la orden, monto total de la orden, fecha de entrega. Cada orden puede tener asociado uno o varios ítems, cada ítem debe tener los siguientes datos: código del ítem, nombre del ítem, cantidad de compra, unidad de medida del ítem, valor unitario y sub total.

- La orden de compra es aprobada por el área de Administración luego el área de Logística envía o hace entrega al proveedor elegido.
- ✓ El área de Almacén funciona de la siguiente forma:
 - Recepciona los bienes que llegan de los proveedores y distribuye con una pecosa a las áreas que realizaron las solicitudes.
 - El proveedor hace entrega a Almacén los ítems detallados en la Orden de Compra, para ello adjunta la guía de remisión y factura para su pago correspondiente si todo está conforme, se registra una entrada de almacén por cada factura relacionada, con los siguientes datos: número de entrada, fecha, número de factura, Proveedor, total bienes, valor total, los ítems recibidos con la cantidad respectiva.
 - El área de Almacén hace entrega de los bienes a las diferentes áreas solicitantes atreves de una pecosa detallando cada ítem entregado y otros datos más como el número de salida, empleado solicitante del ítem o los ítems, cantidad, fecha de salida, fecha de entrega.
- ✓ El área Patrimonio es responsable de:
 - Administrar y controlar la ubicación de los bienes dentro de la empresa, por esto antes de que el bien salga del Almacén es codificado y se genera su estiker que se pega al bien.
 - Cada trabajador tiene a su cargo una o varios bienes, esto permite controlar su ubicación.

Diseñar las tablas en SQL Server, para poder almacenar los datos de que responda al sistema de “**Bienes y Suministros**” de la empresa de turismo “**Pata Amarilla**”.

DESARROLLO

Para el sistema de acuerdo a los requerimientos y descripción de datos se puede considerar las siguientes tablas:

- Área
- Trabajador
- cargo
- Contrato
- Ítem
- Categoría
- Unidad_medida
- Solicitud
- Metas_presupuestal
- Orden_compra
- proveedor

Relacionando las tablas se diseña el modelo lógico mostrado en la FIGURA N° 04-022:

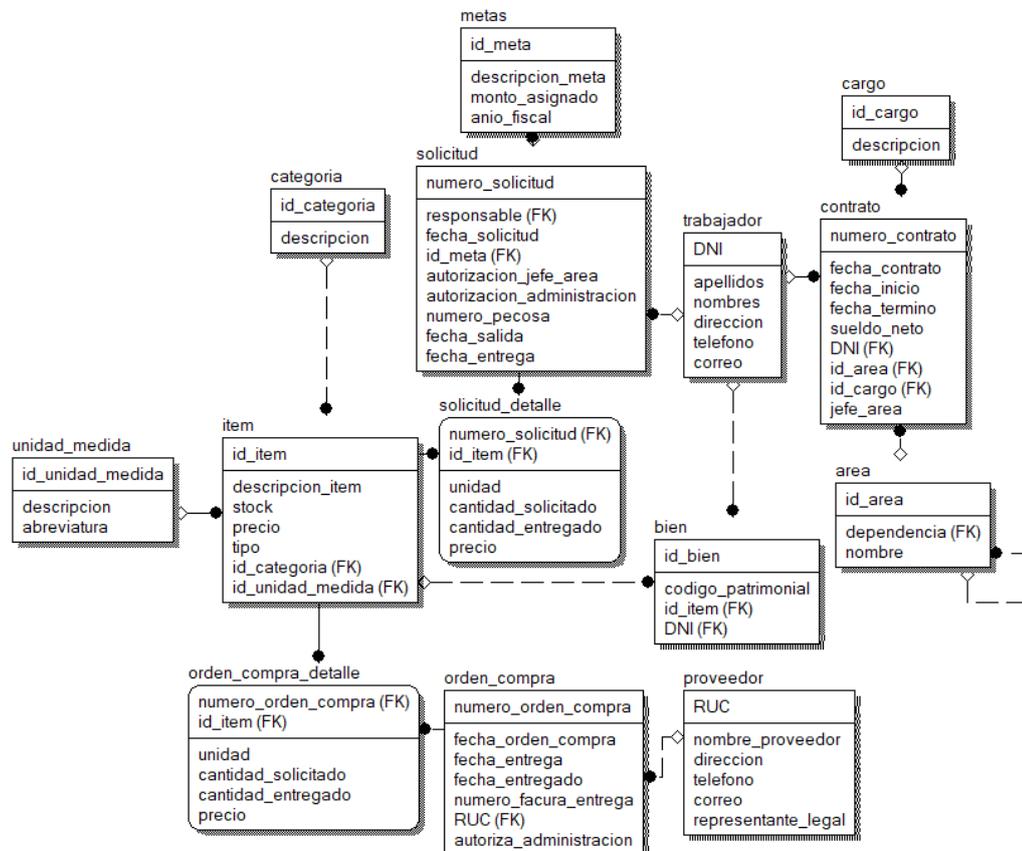


FIGURA N° 04-022: Modelo lógico de la base de datos logística

Script de creación de las tablas en SQL Server:

```
CREATE TABLE area
```

```
(  
    id_area          int NOT NULL ,  
    dependencia      int  NULL ,  
    nombre           varchar(80) NULL  
)  
go
```

```
ALTER TABLE area
```

```
    ADD CONSTRAINT XPKarea PRIMARY KEY NONCLUSTERED (id_area  
ASC)  
go
```

```
CREATE TABLE bien
```

```
(  
    id_bien          char(18) NOT NULL ,  
    codigo_patrimonial char(18) NULL ,  
    id_item          char(10) NULL ,  
    DNI              char(8) NULL  
)  
go
```

```
ALTER TABLE bien
```

```
    ADD CONSTRAINT XPKbien PRIMARY KEY NONCLUSTERED (id_bien  
ASC)  
go
```

```
CREATE TABLE cargo
```

```
(  
    id_cargo         int NOT NULL ,  
    descripcion      varchar(80) NULL  
)  
go
```

```
ALTER TABLE cargo
    ADD CONSTRAINT XPKcargo PRIMARY KEY NONCLUSTERED
(id_cargo ASC)
go
```

```
CREATE TABLE categoria
(
    id_categoria          int NOT NULL ,
    descripcion           varchar(100) NULL
)
go
```

```
ALTER TABLE categoria
    ADD CONSTRAINT XPKcategoria PRIMARY KEY NONCLUSTERED
(id_categoria ASC)
go
```

```
CREATE TABLE contrato
(
    numero_contrato      char(10) NOT NULL ,
    fecha_contrato       datetime NULL ,
    fecha_inicio         datetime NULL ,
    fecha_termino        datetime NULL ,
    sueldo_netto         FLOAT NULL ,
    DNI                  char(8) NULL ,
    id_area              int NULL ,
    id_cargo             int NULL ,
    jefe_area            int NULL
)
go
```

```
ALTER TABLE contrato
    ADD CONSTRAINT XPKcontrato PRIMARY KEY NONCLUSTERED
(numero_contrato ASC)
go
```

```
CREATE TABLE item
```

```
(  
    id_item          char(10) NOT NULL ,  
    descripcion_item varchar(80) NULL ,  
    stock            FLOAT NULL ,  
    precio           FLOAT NULL ,  
    tipo             CHAR(1) NULL ,  
    id_categoria     int NULL ,  
    id_unidad_medida int NULL  
)  
go
```

```
ALTER TABLE item
```

```
    ADD CONSTRAINT XPKitem PRIMARY KEY NONCLUSTERED (id_item  
ASC)  
go
```

```
CREATE TABLE metas
```

```
(  
    id_meta          int NOT NULL ,  
    descripcion_meta varchar(80) NULL ,  
    monto_asignado   FLOAT NULL ,  
    anio_fiscal      int NULL  
)  
go
```

```
ALTER TABLE metas
```

```
    ADD CONSTRAINT XPKpetas PRIMARY KEY NONCLUSTERED (id_meta  
ASC)  
go
```

```
CREATE TABLE orden_compra
```

```
(  
    numero_orden_compra char(10) NOT NULL ,  
    fecha_orden_compra  datetime NULL ,  
    fecha_entrega       datetime NULL ,
```

```
RUC                char(11)  NULL ,
autoriza_administracion int  NULL ,
fecha_entregado    char(18)  NULL ,
numero_facura_entrega char(18) NULL
)
go
```

```
ALTER TABLE orden_compra
    ADD CONSTRAINT XPKorden_compra PRIMARY KEY NONCLUSTERED
    (numero_orden_compra ASC)
go
```

```
CREATE TABLE orden_compra_detalle
(
    unidad                char(18)  NULL ,
    cantidad_solicitado  char(18)  NULL ,
    precio                char(18)  NULL ,
    numero_orden_compra  char(10)  NOT NULL ,
    id_item               char(10)  NOT NULL ,
    cantidad_entregado   char(18)  NULL
)
go
```

```
ALTER TABLE orden_compra_detalle
    ADD CONSTRAINT XPKorden_compra_detalle PRIMARY KEY
NONCLUSTERED (numero_orden_compra ASC, id_item ASC)
go
```

```
CREATE TABLE proveedor
(
    RUC                char(11)  NOT NULL ,
    nombre_proveedor  varchar(80)  NULL ,
    direccion          varchar(80)  NULL ,
    telefono           varchar(30)  NULL ,
```

```
        correo          varchar(30) NULL ,
        representante_legal varchar(100) NULL
    )
go
```

```
ALTER TABLE proveedor
    ADD CONSTRAINT XPKproveedor PRIMARY KEY NONCLUSTERED (RUC
ASC)
go
```

```
CREATE TABLE solicitud
(
    numero_solicitud char(18) NOT NULL ,
    responsable      char(8)  NULL ,
    fecha_solicitud  char(18) NULL ,
    id_meta          int      NULL ,
    autorizacion_jefe_area int  NULL ,
    autorizacion_administracion int NULL ,
    numero_pecosa    char(18) NULL ,
    fecha_salida     char(18) NULL ,
    fecha_entrega    char(18) NULL
)
go
```

```
ALTER TABLE solicitud
    ADD CONSTRAINT XPKsolicitud PRIMARY KEY NONCLUSTERED
(numero_solicitud ASC)
go
```

```
CREATE TABLE solicitud_detalle
(
    unidad          char(18) NULL ,
    cantidad_solicitado char(18) NULL ,
    numero_solicitud char(18) NOT NULL ,

```

```
id_item          char(10) NOT NULL ,
precio          char(18) NULL ,
cantidad_entregado char(18) NULL
)
go
```

```
ALTER TABLE solicitud_detalle
    ADD CONSTRAINT XPKsolicitud_detalle PRIMARY KEY
NONCLUSTERED (numero_solicitud ASC, id_item ASC)
go
```

```
CREATE TABLE trabajador
(
    DNI          char(8) NOT NULL ,
    apellidos   varchar(50) NULL ,
    nombres     varchar(50) NULL ,
    direccion   varchar(50) NULL ,
    telefono    varchar(30) NULL ,
    correo      varchar(30) NULL
)
go
```

```
ALTER TABLE trabajador
    ADD CONSTRAINT XPKtrabajador PRIMARY KEY NONCLUSTERED
(DNI ASC)
go
```

```
CREATE TABLE unidad_medida
(
    id_unidad_medida int NOT NULL ,
    descripcion      VARCHAR(80) NULL ,
    abreviatura      VARCHAR(50) NULL
)
go
```

```
ALTER TABLE unidad_medida
    ADD CONSTRAINT XPKunidad_medida PRIMARY KEY NONCLUSTERED
(id_unidad_medida ASC)
go
```

```
ALTER TABLE area
    ADD CONSTRAINT id_area_dependencia FOREIGN KEY
(dependencia) REFERENCES area(id_area)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
go
```

```
ALTER TABLE bien
    ADD CONSTRAINT R_16 FOREIGN KEY (id_item) REFERENCES
item(id_item)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
go
```

```
ALTER TABLE bien
    ADD CONSTRAINT R_17 FOREIGN KEY (DNI) REFERENCES
trabajador(DNI)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
go
```

```
ALTER TABLE contrato
    ADD CONSTRAINT R_3 FOREIGN KEY (DNI) REFERENCES
trabajador(DNI)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
go
```

```
ALTER TABLE contrato
```

```
ADD CONSTRAINT R_4 FOREIGN KEY (id_area) REFERENCES
area(id_area)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION
```

```
go
```

```
ALTER TABLE contrato
```

```
ADD CONSTRAINT R_5 FOREIGN KEY (id_cargo) REFERENCES
cargo(id_cargo)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION
```

```
go
```

```
ALTER TABLE item
```

```
ADD CONSTRAINT R_6 FOREIGN KEY (id_categoria) REFERENCES
categoria(id_categoria)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION
```

```
go
```

```
ALTER TABLE item
```

```
ADD CONSTRAINT R_18 FOREIGN KEY (id_unidad_medida)
REFERENCES unidad_medida(id_unidad_medida)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION
```

```
go
```

```
ALTER TABLE orden_compra
```

```
ADD CONSTRAINT R_15 FOREIGN KEY (RUC) REFERENCES
proveedor(RUC)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION
```

```
go
```

```
ALTER TABLE orden_compra_detalle
```

```
ADD CONSTRAINT R_13 FOREIGN KEY (numero_orden_compra)
REFERENCES orden_compra(numero_orden_compra)
ON DELETE NO ACTION
ON UPDATE NO ACTION
```

go

```
ALTER TABLE orden_compra_detalle
ADD CONSTRAINT R_14 FOREIGN KEY (id_item) REFERENCES
item(id_item)
ON DELETE NO ACTION
ON UPDATE NO ACTION
```

go

```
ALTER TABLE solicitud
ADD CONSTRAINT R_7 FOREIGN KEY (responsable) REFERENCES
trabajador(DNI)
ON DELETE NO ACTION
ON UPDATE NO ACTION
```

go

```
ALTER TABLE solicitud
ADD CONSTRAINT R_8 FOREIGN KEY (id_meta) REFERENCES
metas(id_meta)
ON DELETE NO ACTION
ON UPDATE NO ACTION
```

go

```
ALTER TABLE solicitud_detalle
ADD CONSTRAINT R_11 FOREIGN KEY (numero_solicitud)
REFERENCES solicitud(numero_solicitud)
ON DELETE NO ACTION
ON UPDATE NO ACTION
```

go

```
ALTER TABLE solicitud_detalle
```

```
ADD CONSTRAINT R_12 FOREIGN KEY (id_item) REFERENCES
item(id_item)

ON DELETE NO ACTION

ON UPDATE NO ACTION

go
```

Se ejecuta en la ventana de consultas del SQL Server Management Studio, y se tendría todas las tablas creadas en SQL Server, tal como se muestra en la FIGURA N° 04-023:

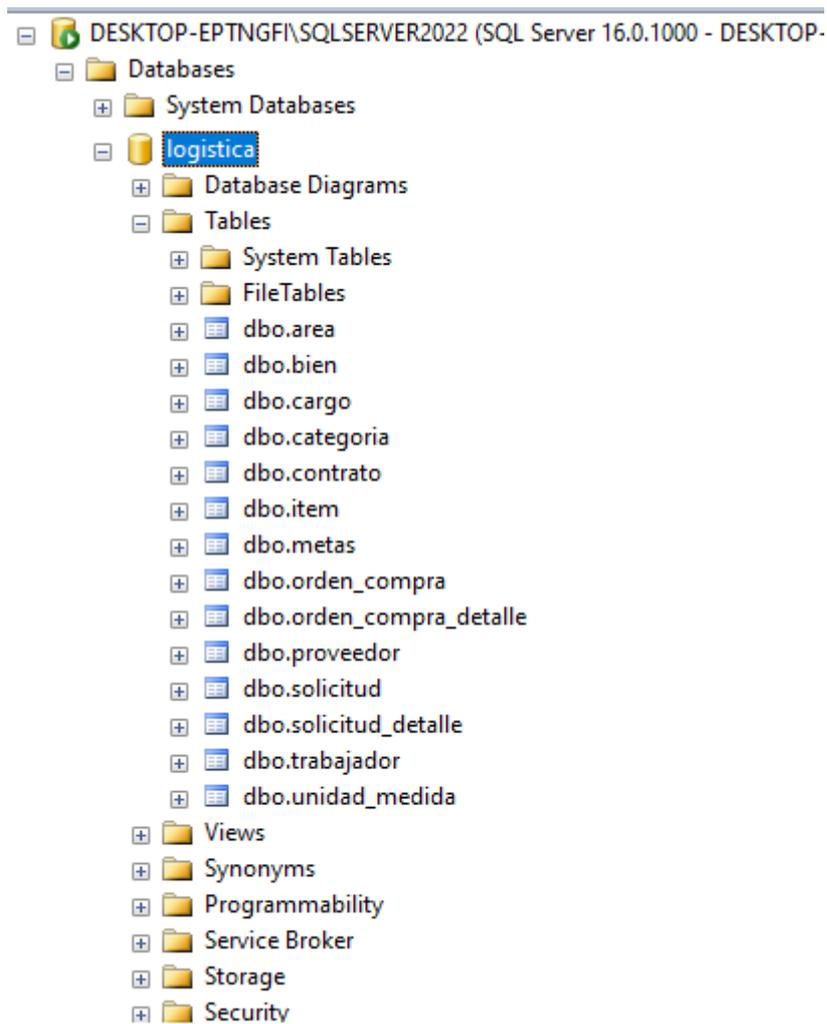


FIGURA N° 04-023: Tablas de la base de datos **logistica** en SQL Server

Gestor de Base de Datos MySQL y MariaDB

MySQL y MariaDB, son dos gestores de base de datos muy parecido de código abierto. Funcionan con un modelo cliente-servidor, son multiplataforma, por lo que se puede instalar en Windows, Linux entre otros sistemas operativos.

Algunas características de MySQL:

- ✓ Sirve para crear y manejar de bases de datos relacionales
- ✓ Implementa varios motores de almacenamiento con características y velocidades distintas.
- ✓ Software libre y gratuito para el uso en su versión de la comunidad
- ✓ Dispone de una arquitectura cliente / servidor y la instalación proporciona tanto el programa de cliente (por línea de comandos) como el del servidor (sistema gestor de la base de datos en sí)
- ✓ Sistema ligero, fácil de usar, fácil de mantener.
- ✓ Es robusto y seguro.

Instalacion de MySQL

Para la instalación de MySQL, se descarga desde su página oficial: <https://www.mysql.com/>



FIGURA N° 04-024: Página oficial de MySQL

Otra alternativa para instalar MariaDB o MySQL con: Xampp, AppServ, WampServer que también a su vez instala Apache + PHP + MySQL o MariaDB.

En la FIGURA N° 04-025 se muestra la página oficial de descarga del instalador del XAMPP:

<https://www.apachefriends.org/es/index.html>



FIGURA N° 04-025: Página oficial para descargar el instalador de XAMPP

Una vez instalado el XAMPP con su panel de control se pone en marcha los servidores Apache y MySQL tal como se muestra en la FIGURA N° 04-026:

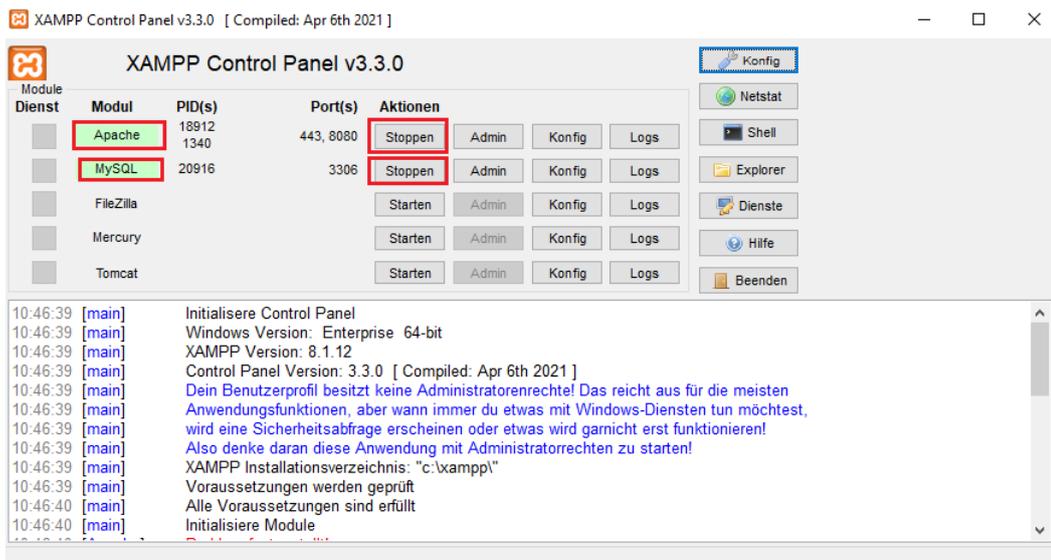


FIGURA N° 04-026: Panel de control del XAMPP

Para tener un entorno de trabajo más amigable en modo gráfico con MySQL o MariaDB, se podría utilizar el **MySQL Workbench** o el **phpMyAdmin** en entorno web. En la FIGURA N° 04-027 tenemos la interfaz del phpMyAdmin, que se accede poniendo en la barra de direcciones de cualquier navegador web: localhost:8080/phpmyadmin (el 8080 depende del puerto del servidor web)



FIGURA N° 04-027: la Interfaz del phpMyAdmin

EJEMPLO N°04-003

Del EJEMPLO N°04-002, crear la base de datos logistica y todas sus tablas en MySQL.

DESARROLLO

El modelo lógico vendría a ser lo mismo mostrado en la FIGURA N° 04-022

Script de creación de las tablas en MySQL:

```
CREATE TABLE area
(
    id_area          INTEGER NOT NULL,
    dependencia      INTEGER NULL,
    nombre           varchar(80) NULL
);
```

```
ALTER TABLE area
ADD PRIMARY KEY (id_area);
```

```
CREATE TABLE bien
(
    id_bien          CHAR(18) NOT NULL,
    codigo_patrimonial char(18) NULL,
    id_item          char(10) NULL,
    DNI              char(8) NULL
);
```

```
ALTER TABLE bien
ADD PRIMARY KEY (id_bien);
```

```
CREATE TABLE cargo
(
    id_cargo        INTEGER NOT NULL,
    descripcion     varchar(80) NULL
);
```

```
ALTER TABLE cargo
ADD PRIMARY KEY (id_cargo);
```

```
CREATE TABLE categoria
(
    id_categoria    INTEGER NOT NULL,
    descripcion     varchar(100) NULL
);
```

```
ALTER TABLE categoria
ADD PRIMARY KEY (id_categoria);
```

```
CREATE TABLE contrato
(
    numero_contrato char(10) NOT NULL,
    fecha_contrato  datetime NULL,
```

```

    fecha_inicio         datetime NULL,
    fecha_termino       datetime NULL,
    sueldo_netto         FLOAT NULL,
    DNI                  char(8) NULL,
    id_area              INTEGER NULL,
    id_cargo             INTEGER NULL,
    jefe_area           INTEGER NULL
);

```

```

ALTER TABLE contrato
ADD PRIMARY KEY (numero_contrato);

```

```

CREATE TABLE item
(
    id_item              char(10) NOT NULL,
    descripcion_item     varchar(80) NULL,
    stock               FLOAT NULL,
    precio              FLOAT NULL,
    tipo                CHAR(1) NULL,
    id_categoria        INTEGER NULL,
    id_unidad_medida    INTEGER NULL
);

```

```

ALTER TABLE item
ADD PRIMARY KEY (id_item);

```

```

CREATE TABLE metas
(
    id_meta             INTEGER NOT NULL,
    descripcion_meta    varchar(80) NULL,
    monto_asignado     FLOAT NULL,
    anio_fiscal        INTEGER NULL
);

```

```
ALTER TABLE metas
ADD PRIMARY KEY (id_meta);
```

```
CREATE TABLE orden_compra
(
    numero_orden_compra char(10) NOT NULL,
    fecha_orden_compra datetime NULL,
    fecha_entrega datetime NULL,
    RUC char(11) NULL,
    autoriza_administracion INTEGER NULL,
    fecha_entregado CHAR(18) NULL,
    numero_facura_entrega CHAR(18) NULL
);
```

```
ALTER TABLE orden_compra
ADD PRIMARY KEY (numero_orden_compra);
```

```
CREATE TABLE orden_compra_detalle
(
    unidad CHAR(18) NULL,
    cantidad_solicitado CHAR(18) NULL,
    precio CHAR(18) NULL,
    numero_orden_compra char(10) NOT NULL,
    id_item char(10) NOT NULL,
    cantidad_entregado CHAR(18) NULL
);
```

```
ALTER TABLE orden_compra_detalle
ADD PRIMARY KEY (numero_orden_compra,id_item);
```

```
CREATE TABLE proveedor
(
```

```
RUC                char(11) NOT NULL,  
nombre_proveedor  varchar(80) NULL,  
direccion          varchar(80) NULL,  
telefono           varchar(30) NULL,  
correo             varchar(30) NULL,  
representante_legal varchar(100) NULL  
);
```

```
ALTER TABLE proveedor  
ADD PRIMARY KEY (RUC);
```

```
CREATE TABLE solicitud  
(  
    numero_solicitud CHAR(18) NOT NULL,  
    responsable      char(8) NULL,  
    fecha_solicitud  CHAR(18) NULL,  
    id_meta          INTEGER NULL,  
    autorizacion_jefe_area INTEGER NULL,  
    autorizacion_administracion INTEGER NULL,  
    numero_pecosa    CHAR(18) NULL,  
    fecha_salida     CHAR(18) NULL,  
    fecha_entrega    CHAR(18) NULL  
);
```

```
ALTER TABLE solicitud  
ADD PRIMARY KEY (numero_solicitud);
```

```
CREATE TABLE solicitud_detalle  
(  
    unidad          CHAR(18) NULL,  
    cantidad_solicitado CHAR(18) NULL,  
    numero_solicitud CHAR(18) NOT NULL,  
    id_item         char(10) NOT NULL,
```

```
precio          CHAR(18) NULL,  
cantidad_entregado CHAR(18) NULL  
);
```

```
ALTER TABLE solicitud_detalle  
ADD PRIMARY KEY (numero_solicitud,id_item);
```

```
CREATE TABLE trabajador  
(  
    DNI          char(8) NOT NULL,  
    apellidos    varchar(50) NULL,  
    nombres      varchar(50) NULL,  
    direccion    varchar(50) NULL,  
    telefono     varchar(30) NULL,  
    correo       varchar(30) NULL  
);
```

```
ALTER TABLE trabajador  
ADD PRIMARY KEY (DNI);
```

```
CREATE TABLE unidad_medida  
(  
    id_unidad_medida    INTEGER NOT NULL,  
    descripcion         VARCHAR(80) NULL,  
    abreviatura         VARCHAR(50) NULL  
);
```

```
ALTER TABLE unidad_medida  
ADD PRIMARY KEY (id_unidad_medida);
```

```
ALTER TABLE area  
ADD FOREIGN KEY id_area_dependencia (dependencia) REFERENCES  
area (id_area);
```

```
ALTER TABLE bien
ADD FOREIGN KEY R_16 (id_item) REFERENCES item (id_item);

ALTER TABLE bien
ADD FOREIGN KEY R_17 (DNI) REFERENCES trabajador (DNI);

ALTER TABLE contrato
ADD FOREIGN KEY R_3 (DNI) REFERENCES trabajador (DNI);

ALTER TABLE contrato
ADD FOREIGN KEY R_4 (id_area) REFERENCES area (id_area);

ALTER TABLE contrato
ADD FOREIGN KEY R_5 (id_cargo) REFERENCES cargo (id_cargo);

ALTER TABLE item
ADD FOREIGN KEY R_6 (id_categoria) REFERENCES categoria
(id_categoria);

ALTER TABLE item
ADD FOREIGN KEY R_18 (id_unidad_medida) REFERENCES unidad_medida
(id_unidad_medida);

ALTER TABLE orden_compra
ADD FOREIGN KEY R_15 (RUC) REFERENCES proveedor (RUC);

ALTER TABLE orden_compra_detalle
ADD FOREIGN KEY R_13 (numero_orden_compra) REFERENCES
orden_compra (numero_orden_compra);

ALTER TABLE orden_compra_detalle
ADD FOREIGN KEY R_14 (id_item) REFERENCES item (id_item);
```

```
ALTER TABLE solicitud
ADD FOREIGN KEY R_7 (responsable) REFERENCES trabajador (DNI);

ALTER TABLE solicitud
ADD FOREIGN KEY R_8 (id_meta) REFERENCES metas (id_meta);

ALTER TABLE solicitud_detalle
ADD FOREIGN KEY R_11 (numero_solicitud) REFERENCES solicitud
(numero_solicitud);

ALTER TABLE solicitud_detalle
ADD FOREIGN KEY R_12 (id_item) REFERENCES item (id_item);
```

Se ejecuta en la ventana de consultas del SQL phpMyAdmin, y se tendría todas las tablas creadas en MySQL, tal como se muestra en la FIGURA N^o 04-028:

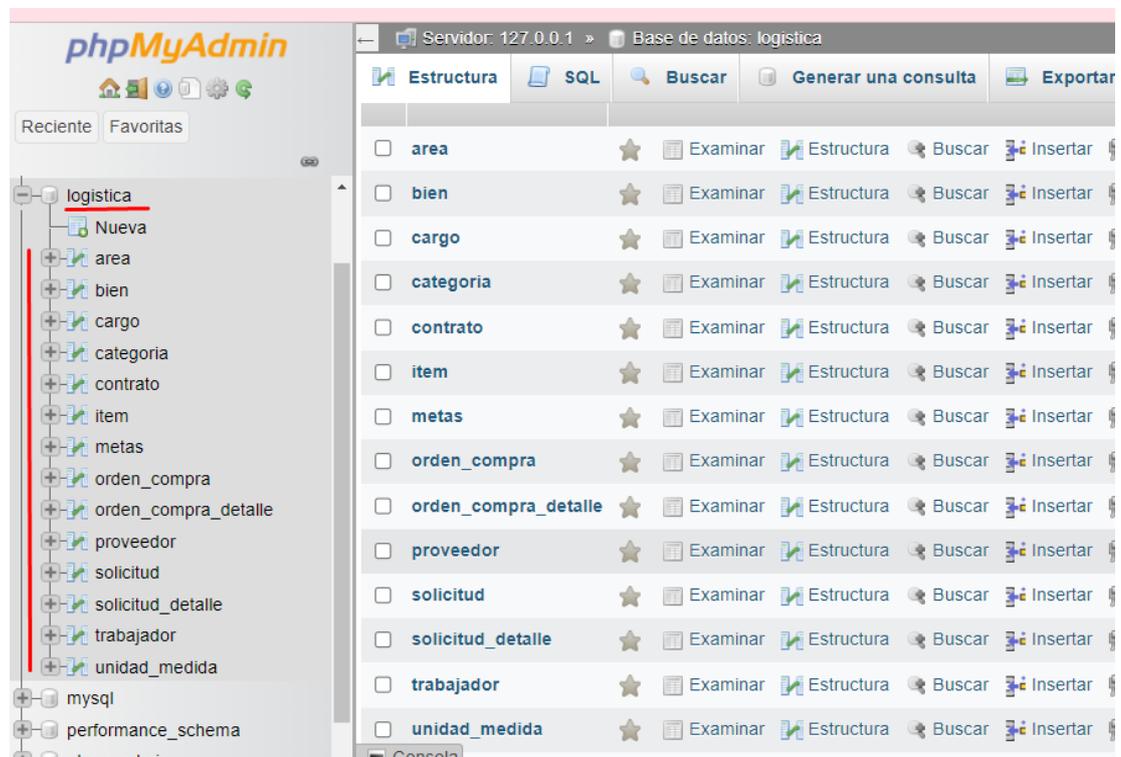


FIGURA N^o 04-028: Tablas de la base de datos logistica creadas en MySQL

Poblado y consultas SQL a la Base de datos

Insertando datos a algunas tablas:

Tabla unidad_medida

```
INSERT INTO unidad_medida (id_unidad_medida, descripcion,
abreviatura) VALUES
```

```
(1, 'ACCION', 'ACCION'),
(2, 'ACERVO', 'ACERVO'),
(52, 'EXPEDIENTE PROCESADO', 'EXPEDIEN'),
(53, 'EXPEDIENTE RESUELTO', 'EXPEDIEN'),
(54, 'EXPEDIENTE TECNICO', 'EXPEDIEN'),
(55, 'EXPEDIENTE TRAMITADO', 'EXPEDIEN'),
(56, 'FAMILIA', 'FAMILIA'),
(57, 'FOLLETO', 'FOLLETO'),
(67, 'KILOMETRO', 'KILOMETR'),
(68, 'LICENCIA', 'LICENCIA'),
(69, 'M2', 'M2'),
(70, 'M3', 'M3'),
(72, 'MAPA', 'MAPA'),
(73, 'METRO', 'METRO'),
(74, 'METRO LUZ', 'METRO LU'),
(126, 'PAQUETE ESCOLAR', 'PAQUETE '),
(127, 'PLANTAS', 'PLANTAS'),
(128, 'KILOGRAMO', 'KLG'),
(129, 'DIQUE', 'DIQUE'),
(162, 'NUEVOS SOLES', 'NUEVOS SOLES'),
(163, 'DOLARES', 'DOLARES'),
(164, 'LT./SEG.', 'LT./SEG.'),
(172, 'GLOBAL', 'GLOBAL'),
(173, 'CASOS RESUELTOS', 'CASOS RESUELTOS'),
(174, 'NORMAS APROBADAS', 'NORMAS APROBADA'),
(175, 'APELACIONES RESUELTAS', 'APELACIONES RES'),
(176, 'PASAJEROS/DIA', 'PASAJEROS/DIA'),
```

```
(177, 'TRANSFERENCIA', 'TRANSFERENCIA'),
(178, 'MEDIDAS CAUTELARES', 'MEDIDAS CAUTELA'),
(181, 'HORAS', 'HORAS'),
(182, 'LOCAL', 'LOCAL'),
(183, 'ACCIONES DE AUDITORIA', 'ACCIONES DE AUD'),
(185, 'CRIADERO', 'CRIADERO'),
(186, 'KW', 'KW'),
(187, 'ASPERSORES', 'ASPERSORES'),
(188, 'AUTORIDADES', 'AUTORIDADES'),
(189, 'TELEFONOS EN ZONAS RURALES', 'TELEFONOS EN Z'),
(190, 'NUMERO', 'NUMERO'),
(191, 'PERSONA PROTEGIDA', 'PERSONA PROTEGI'),
(192, 'CATALOGO', 'CATALOGO'),
(193, 'QUINTAL', 'QUINTAL');
```

Tabla categoría:

```
INSERT INTO categoria (id_categoria, descripcion) VALUES
(1, 'MATERIAL DE GUERRA'),
(2, 'OBRAS RELACIONADAS CON LA AGRICULTURA'),
(3, 'ACONDICIONAMIENTO'),
(4, 'ABRASIVOS'),
(5, 'OBRAS RELACIONADAS CON EL TRANSPORTE'),
(6, 'SERVICIOS RELACIONADOS CON LA ENERGIA NUCLEARES'),
(7, 'OBRAS RELACIONADAS CON LA ENERGÍA Y MINERÍA'),
(8, 'ALAMBRES, BARRAS, PLANCHAS, PERFILES Y SIMILARES DE METAL'),
(9, 'ALIMENTACIÓN'),
(10, 'AGROPECUARIO Y PESQUERO'),
(11, 'OBRAS RELACIONADAS CON EL TURISMO'),
(12, 'OBRAS RELACIONADAS CON LA VIVIENDA Y CONSTRUCCIÓN'),
(13, 'SERVICIOS RELACIONADOS CON ALOJAMIENTOS Y ASISTENCIA PARA VIAJES'),
(14, 'AGROPECUARIA, GANADERÍA Y JARDINERÍA : REPUESTOS, ACCESORIOS Y MATERIALES');
```

```
(15, 'AIRE ACONDICIONADO Y REFRIGERACION : REPUESTOS Y ACCESORIOS'),
(16, 'SERVICIO DE ASEO Y LIMPIEZA'),
(17, 'OBRAS RELACIONADAS CON LAS COMUNICACIONES'),
(18, 'OBRAS RELACIONADAS CON LA ZOOLOGIA'),
(19, 'ASESORIA, CONSULTORIA E INTERVENCIONES ESPECIALIZADAS'),
(20, 'AISLANTES TERMICOS Y ACUSTICOS, REFRACTARIOS, FILTROS Y PURIFICADORES'),
(21, 'CONTRATACION ADMINISTRATIVA DE SERVICIOS - CAS'),
(22, 'SERVICIOS NO PERSONALES'),
(23, 'ALIMENTOS PARA ANIMALES'),
(24, 'SERVICIOS PRESTADOS POR TERCEROS'),
(25, 'ALIMENTOS Y BEBIDAS PARA PERSONAS'),
(26, 'ATENCIONES Y CELEBRACIONES'),
(27, 'ARMAMENTO, MUNIC. Y EXPLOS. : RPTOS, ACCESORIOS Y MATERIALES'),
(28, 'AIRE ACONDICIONADO Y REFRIGERACION'),
(29, 'SERVICIOS RELACIONADOS CON LAS CONSTRUCCIONES.'),
(30, 'SERVICIOS RELACIONADOS CON EL MEDIO AMBIENTE Y SANEAMIENTO'),
(31, 'EQUIPOS PARA ARTES GRAFICAS E IMPRESIONES : REPUESTOS Y ACCESORIOS'),
(32, 'ASEO, LIMPIEZA Y TOCADOR : REPUESTOS, ACCESORIOS, PISELES Y MATERIALES'),
(33, 'SERVICIOS RELACIONADOS CON FINANZAS E INVERSIONES Y SIMILARES'),
(34, 'BIENES DE ACTIVO FIJO NO CATALOGADOS POR SBN');
```

Tabla ítem:

```
INSERT INTO item (id_item, descripcion_item, stock, precio, tipo, id_categoria, id_unidad_medida) VALUES
('1', 'PIEDRA ESMERIL CIRCULAR 2 in X 10 in GRANO GRUESO', 0, 0, 'B', 2, 112),
('10', 'LIJA CIRCULAR PARA FIERRO N? 24', 0, 0, 'B', 2, 112),
('100', 'ACERO REDONDO ST37 ? 5/8 in', 0, 0, 'B', 3, 73),
('1000', 'CAFE INSTANTANEO X 7 G', 0, 0, 'B', 9, 358),
```

('1001', 'AVENA X 200 G X 24 UNI', 0, 0, 'B', 9, 335),
 ('1002', 'LENTEJA X 20 KG', 0, 0, 'B', 9, 378),
 ('1003', 'ARROZ CORRIENTE X 48 Kg', 0, 0, 'B', 9, 378),
 ('1004', 'FRIJOL OJO DE PALOMA', 0, 0, 'B', 9, 128),
 ('1005', 'FREJOL DE SOYA', 0, 0, 'B', 9, 128),
 ('1006', 'KIWICHA ENTERA TOSTADA', 0, 0, 'B', 9, 128),
 ('1007', 'CEBADA TOSTADA', 0, 0, 'B', 9, 128),
 ('1008', 'MAIZENA', 0, 0, 'B', 9, 128),
 ('1009', 'CREMA DE CHOCLO X 5 kg', 0, 0, 'B', 9, 112),
 ('101', 'ACERO REDONDO TREFILADO SAE 1020 ? 1/4', 0, 0, 'B', 3, 73),
 ('1010', 'CEBADA', 0, 0, 'B', 9, 128),
 ('1011', 'CREMA DE HABAS CALIDAD SUPERIOR', 0, 0, 'B', 9, 128),
 ('1012', 'HARINA DE TRIGO X 50 KG', 0, 0, 'B', 9, 378),
 ('1013', 'SEMOLA X 20 UNIDADES DE 250 GRS C/U', 0, 0, 'B', 9, 352),
 ('1014', 'HARINA PREPARADA X KILO X 12 UNIDADES', 0, 0, 'B', 9, 352),
 ('1015', 'HOJUELA DE KIWICHA ', 0, 0, 'B', 9, 128),
 ('1016', 'CREMA DE ALVEJA', 0, 0, 'B', 9, 128),
 ('1017', 'HOJUELA DE TRIGO ', 0, 0, 'B', 9, 128),
 ('1018', 'HOJUELA DE QUINUA', 0, 0, 'B', 9, 128),
 ('1019', 'MAIZENA IMPORTADA DE 25 KG', 0, 0, 'B', 9, 335),
 ('102', 'ACERO REDONDO SAE 4340 (VCN 150) ? 30MM', 0, 0, 'B', 3, 73),
 ('1020', 'MAIZENA DE 500 G', 0, 0, 'B', 9, 337),
 ('1021', 'HOJUELA DE CEBADA', 0, 0, 'B', 9, 128),
 ('1022', 'HARINA DE HABAS 1 KG X 12 UNI', 0, 0, 'B', 9, 352),
 ('1023', 'HOJUELA DE QUINUA 1 KG X 12 UNI', 0, 0, 'B', 9, 352),
 ('1024', 'CREMA DE ESPARRAGOS X 70 g', 0, 0, 'B', 9, 112),
 ('1025', 'CREMA DE CHAMPI?ONES', 0, 0, 'B', 9, 128),
 ('1026', 'CREMA DE ESPARRAGOS', 0, 0, 'B', 9, 128),
 ('1027', 'QUINUA PERLADA x 50 kg', 0, 0, 'B', 9, 112),
 ('1028', 'TRIGO MORON x 25 kg', 0, 0, 'B', 9, 112),
 ('1029', 'HARINA PASTELERA', 0, 0, 'B', 9, 128),

('103', 'ACERO SAE 4140 O CVL 140 REDONDO 81/2 in X 2 MT', 0, 0, 'B', 3, 112),
('1030', 'MAIZENA X 100 G X 96 UNI', 0, 0, 'B', 9, 352),
('1031', 'HOJUELA DE AVENA', 0, 0, 'B', 9, 128),
('1032', 'HOJUELA DE CEREALES', 0, 0, 'B', 9, 128),
('1033', 'HOJUELA DE AVENA X 250 g', 0, 0, 'B', 9, 112),
('1034', 'HARINA DE PLATANO', 0, 0, 'B', 9, 128),
('1035', 'HARINA DE MAIZ', 0, 0, 'B', 9, 128),
('1036', 'CREMA DE HABAS', 0, 0, 'B', 9, 128),
('1037', 'HOJUELA DE QUINUA AVENA', 0, 0, 'B', 9, 128),
('1038', 'HARINA DE TRIGO FORTIFICADO', 0, 0, 'B', 9, 128),
('1039', 'CREMA DE HABAS LACTEADA', 0, 0, 'B', 9, 128),
('104', 'ACERO SAE 4140 O VCL 140 REDONDO 11 in X 2 MTS', 0, 0, 'B', 3, 112),
('1040', 'HOJUELA DE AVENA X 225 g', 0, 0, 'B', 9, 112),
('1041', 'HOJUELA DE AVENA X 450 g', 0, 0, 'B', 9, 112),
('1042', 'HOJUELA DE AVENA ENRIQUECIDA', 0, 0, 'B', 9, 128),
('1043', 'HOJUELA DE AVENA X 10 kg', 0, 0, 'B', 9, 112),
('1044', 'HOJUELA DE AVENA ENRIQUECIDA X 500 g', 0, 0, 'B', 9, 112),
('1045', 'HOJUELA DE CEBADA ENRIQUECIDA', 0, 0, 'B', 9, 128),
('1046', 'HOJUELA DE CEREAL ENRIQUECIDA', 0, 0, 'B', 9, 128),
('1190', 'AGUA MINERAL ENVASE NO RETORNABLE CON GAS X 500 ML', 0, 0, 'B', 9, 112),
('1191', 'AGUA MINERAL DE PLASTICO NO RETORNABLE C/GAS X 2 L', 0, 0, 'B', 9, 336),
('1192', 'AGUA MINERAL DE PLASTICO RETORNABLE X 20 L', 0, 0, 'B', 9, 333),
('1193', 'AGUA MINERAL ENVASE NO RETORNABLE S/GAS X 500 ML', 0, 0, 'B', 9, 112),
('1194', 'AGUA MINERAL VIDRIO RETORNABLE C/GAS X 600 ML', 0, 0, 'B', 9, 336),
('1195', 'AGUA MINERAL VIDRIO RETORNABLE S/GAS X 600 ML', 0, 0, 'B', 9, 336),
('1196', 'AGUA MINERAL DE PLASTICO RETORNABLE X 19 L', 0, 0, 'B', 9, 333),
('1197', 'AGUA MINERAL CAJA X 20 L', 0, 0, 'B', 9, 337),

```
( '1198', 'AGUA MINERAL BOTELLA SIN GAS X 02 L X 06 UNI', 0, 0,
'B', 9, 352),
( '1199', 'AGUA MINERAL S/GAS EN VASO X 500 ML', 0, 0, 'B', 9,
112),
( '12', 'LIJA PARA METAL N? 100 X 50 UNI', 0, 0, 'B', 2, 112),
( '120', 'PLANCHA DE FIERRO DE 90 cm X 30 cm X 1.5 mm', 0, 0,
'B', 3, 112),
( '1200', 'AGUA MINERAL S/GAS X 500 ML', 0, 0, 'B', 9, 112),
( '1201', 'AGUA MINERAL C/GAS X 500 ML', 0, 0, 'B', 9, 112),
( '1202', 'AGUA MINERAL BOTELLA CON GAS X 02 L X 06 UNI', 0, 0,
'B', 9, 352),
( '1203', 'AGUA MINERAL CON GAS EN ENVASE NO RETORNABLE X 625
ML', 0, 0, 'B', 9, 336);
```

EJEMPLO N°04-004

Del base de datos de “logística” diseñando en el EJEMPLO N°04-003 hacer las siguientes consultas:

1. Lista de trabajadores con su respectiva área de trabajo
2. Cantidad de trabajadores por área
3. Lista de trabajadores por área con su respectivo cargo
4. Cantidad de solicitudes por área
5. Cantidad de órdenes de compra por proveedor
6. Lista de categoría con la cantidad de ítem que pertenecen a dicha categoría

DESARROLLO

1 Lista de trabajadores con su respectiva área de trabajo Para esta consulta se:

Para esta consulta, del modelo lógico mostrado en la FIGURA N° 04-022, se tiene que consultar a las tablas “**contrato**”, “**trabajador**”, “**area**”; como la tabla “**contrato**” está relacionado con “**trabajador**” por la columna común **DNI** entonces en la cláusula **WHERE** se tiene que poner esa relación. Así mismo la tabla “**contrato**” está relacionado con “**area**” por la columna común “**id_area**” también en la cláusula **WHERE** se tiene poner esa relación.

```
SELECT t.apellidos,t.nombres, a.nombre AS descripcion_area
FROM contrato AS c,trabajador AS t, area AS a
```

```
WHERE c.DNI=t.DNI and c.id_area=a.id_area
ORDER BY t.apellidos ASC
```

2 Cantidad de trabajadores por área

Para esta consulta, del modelo lógico mostrado en la FIGURA N° 04-022, se tiene que consultar a las tablas “**contrato**” y “**area**”; como la tabla “**contrato**” está relacionado con la tabla “**area**” por la columna común “**id_area**” en la cláusula **WHERE** se tiene poner esa relación.

```
SELECT a.nombre AS descripcion_area,COUNT(c.DNI) AS cantidad
FROM contrato AS c, area AS a
WHERE c.id_area=a.id_area
ORDER BY a.nombre ASC
```

3 Lista de trabajadores por área con su respectivo cargo

Para esta consulta, del modelo lógico mostrado en la FIGURA N° 04-022, se tiene que consultar a las tablas “**contrato**”, “**trabajador**”, “**area**” y “**cargo**”; como la tabla “**contrato**” está relacionado con “**trabajador**” por la columna común **DNI** entonces en la cláusula **WHERE** se tiene que poner esa relación. Así mismo la tabla “**contrato**” está relacionado con “**area**” por la columna común “**id_area**” también en la cláusula **WHERE** se tiene que poner esa relación. También la tabla “**cargo**” está relacionado con “**contrato**” por la columna “**id_cargo**”, por lo que se tiene que poner en la cláusula **WHERE** esa relación.

```
SELECT t.apellidos,t.nombres,
       a.nombre AS descripcion_area,
       ca.descripcion
FROM contrato AS c,trabajador AS t, area AS a,cargo AS ca
WHERE c.DNI=t.DNI
       AND c.id_area=a.id_area
       AND c.id_cargo=ca.id_cargo
ORDER BY t.apellidos ASC
```

4 Cantidad de solicitudes por área

Para esta consulta, del modelo lógico mostrado en la FIGURA N° 04-022, se tiene que consultar a las tablas “**solicitud**”, “**trabajador**”, “**contrato**” y “**area**”; como la tabla “**solicitud**” está relacionado con “**trabajador**” por la columna responsable y **DNI** respectivamente entonces en la cláusula **WHERE** se pone esa relación. Así mismo la tabla “**trabajador**” está relacionado con “**contrato**” por la columna común “**DNI**” en la cláusula **WHERE** se pone esa relación. También la tabla “**contrato**” está relacionado con “**area**” por la columna “**id_area**”, por lo que se pone en la cláusula **WHERE** esa relación.

```
SELECT a.nombre, COUNT(s.numero_solicitud) AS cant
FROM solicitud AS s, trabajador AS t,
     contrato AS c, area AS a
WHERE s.responsable=t.DNI AND t.DNI=c.DNI AND c.id_area=a.id_area
GROUP BY c.id_area
ORDER BY a.nombre
```

5 Cantidad de órdenes de compra por proveedor

```
SELECT p.nombre_proveedor,
     COUNT(oc.numero_orden_compra) AS cant
FROM proveedor AS p, orden_compra AS oc
WHERE p.RUC=oc.RUC
GROUP BY p.RUC
ORDER BY p.nombre_proveedor
```

6 Lista de categoría con la cantidad de ítem que pertenecen a dicha categoría

```
SELECT c.descripcion, COUNT(i.id_categoria)
FROM item AS i, categoria AS c
WHERE i.id_categoria=c.id_categoria
GROUP BY i.id_categoria;
```

El resultado de la consulta se muestra en la figura:

	descripcion	count(i.id_categoria)
<input type="checkbox"/> Editar Copiar Borrar	OBRAS RELACIONADAS CON LA AGRICULTURA	21
<input type="checkbox"/> Editar Copiar Borrar	ACONDICIONAMIENTO	351
<input type="checkbox"/> Editar Copiar Borrar	ABRASIVOS	17
<input type="checkbox"/> Editar Copiar Borrar	OBRAS RELACIONADAS CON EL TRANSPORTE	36
<input type="checkbox"/> Editar Copiar Borrar	SERVICIOS RELACIONADOS CON LA ENERGIA NUCLEARES	32
<input type="checkbox"/> Editar Copiar Borrar	OBRAS RELACIONADAS CON LA ENERGÍA Y MINERÍA	294
<input type="checkbox"/> Editar Copiar Borrar	ALAMBRES, BARRAS, PLANCHAS, PERFILES Y SIMILARES D...	35
<input type="checkbox"/> Editar Copiar Borrar	ALIMENTACIÓN	1625
<input type="checkbox"/> Editar Copiar Borrar	AGRICOLA Y PESQUERO	4
<input type="checkbox"/> Editar Copiar Borrar	OBRAS RELACIONADAS CON EL TURISMO	175
<input type="checkbox"/> Editar Copiar Borrar	OBRAS RELACIONADAS CON LA VIVIENDA Y CONSTRUCCIÓN	8
<input type="checkbox"/> Editar Copiar Borrar	SERVICIOS RELACIONADOS CON ALOJAMIENTOS Y ASISTENC...	1056

CAPÍTULO V

Desarrollo de soluciones backend con Node.JS y Express



Programación del lado del servidor(backend)

Para trabajar con base de datos y hacer persistente los datos recogidos desde algún formulario diseñado en HTML, CSS y JavaScript desde un browser, se necesita programar desde el lado de un servidor web y servidor de base de datos. Hay lenguajes de programación con ese propósito: Java, Python, PHP, Ruby, ASP.NET.

JavaScript permite programar desde el lado del servidor a través de Node.js, un entorno de ejecución multiplataforma que tiene todo lo necesario para ejecutar un programa desarrollado en JavaScript, así mismo para facilitar el trabajo de desarrollo se tiene el framework backend Express.js para Node.js

5.1. Node.js

Es un **entorno** de ejecución de JavaScript orientado a **eventos asíncronos** (evento que se ejecuta independientemente del proceso principal de la aplicación en ejecución, FIGURA N° 05-001), para la capa de servidor(backend).

Para ejecutar código JavaScript se necesitaba un navegador, pero Node.js nos da la **infraestructura(entorno)** en el cual se ejecuta código JavaScript sin la necesidad de un navegador.

Evento asíncrono (que se ejecuta como parte del proceso principal de la aplicación):

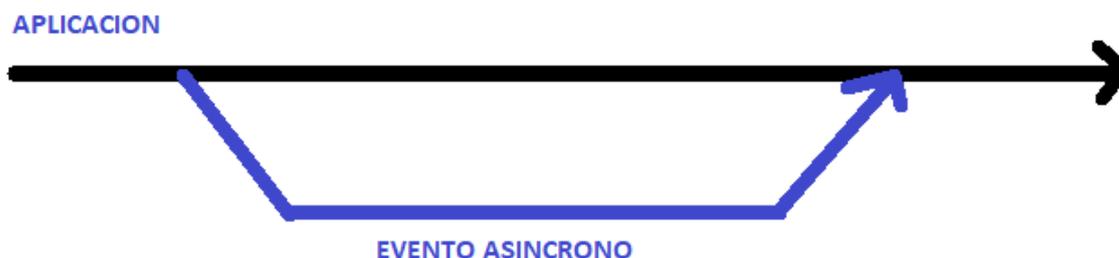


FIGURA N° 05-001: Evento asíncrono

Evento síncrono (que se ejecuta uno de tras de otro bloqueando el proceso principal de la aplicación FIGURA N° 05-002):



FIGURA N° 05-002: Evento síncrono

Características de Node.js

- ✓ Open-source(código abierto)
- ✓ Multiplataforma (se ejecuta en varias plataformas)
- ✓ Basado en el motor V8 de Google (Motor de JavaScript desarrollado por el Chromium Project para Google Chrome)

Node.js no es:

- ✓ Lenguaje de programación
- ✓ Framework
- ✓ Librería

Instalación de Node.js

El logo y página oficial de Node.js es tal como se muestra en la FIGURA N° 05-003 y FIGURA N° 05-004 respectivamente:



FIGURA N° 05-003: Logo del Node.js

<https://nodejs.org/en>



FIGURA N° 05-004: Pagina web para descargar el instalador de Node.js

En la FIGURA N° 05-004 se puede ver la página oficial del Node.js en donde se tiene las dos últimas versiones: LTS (Long-Term- Support) y Actual, del cual hay que descargar la versión recomendada.

Una vez descargado su instalación es sencilla, tal como se muestra en la en la FIGURA N° 05-005.

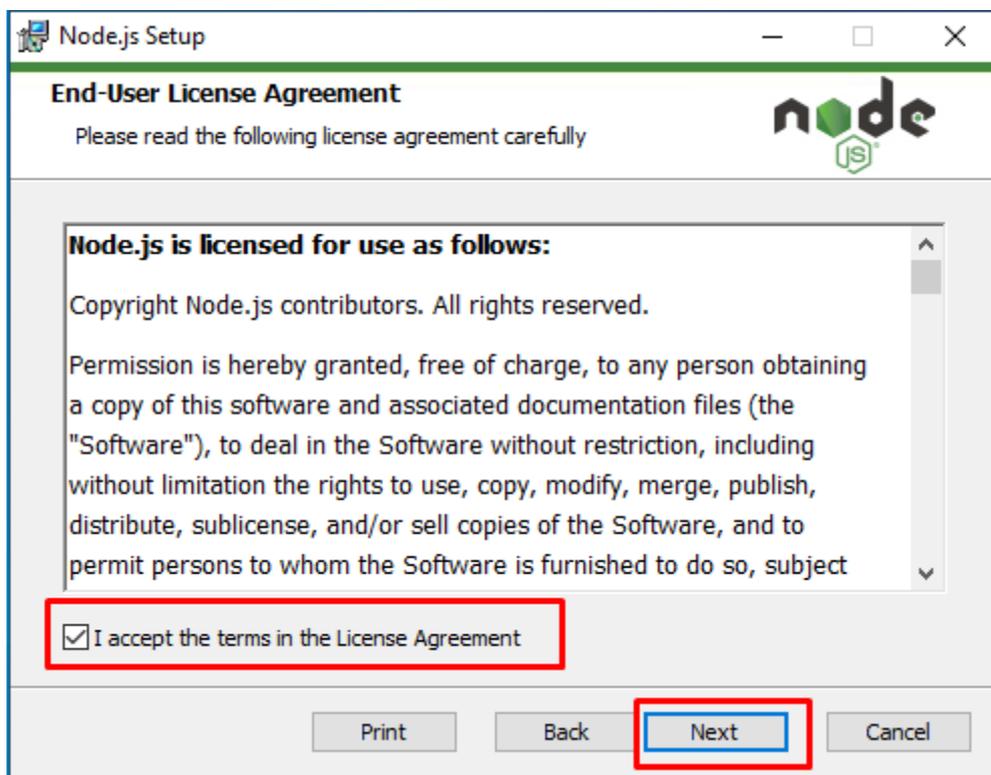


FIGURA N° 05-005: Instalación del Node.js

Una vez terminada la instalación verificamos en la lista de programas FIGURA N° 05-006:

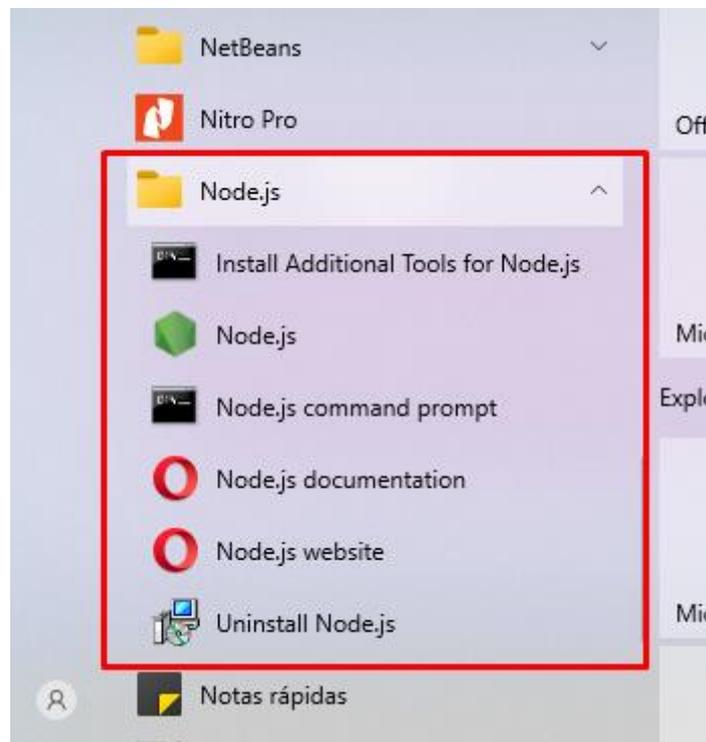


FIGURA N° 05-006: Node.js

Seleccionamos **Node.js command prompt** e interactuar con Node.js, se puede ver la versión digitando **>node --version**, FIGURA N° 05-007.

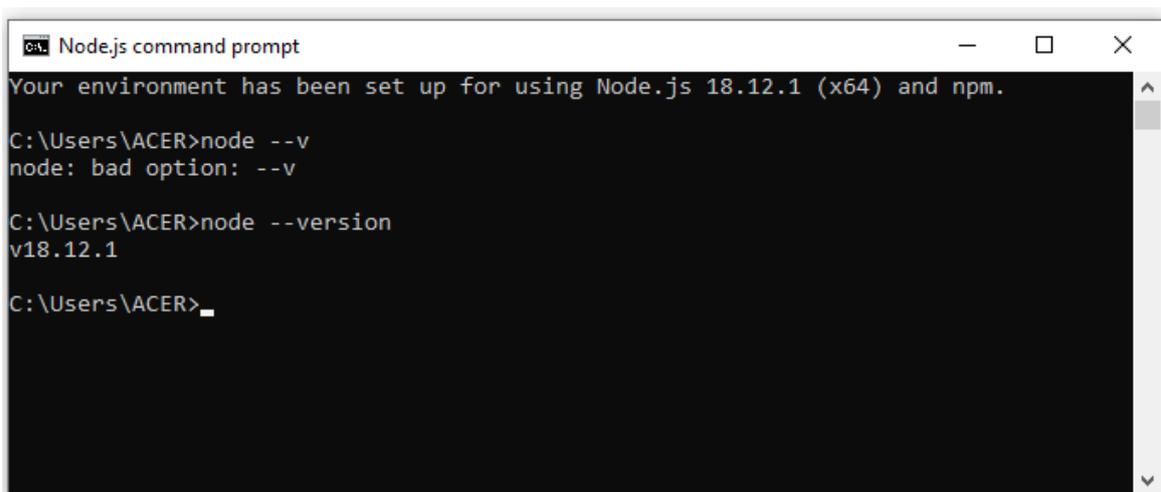
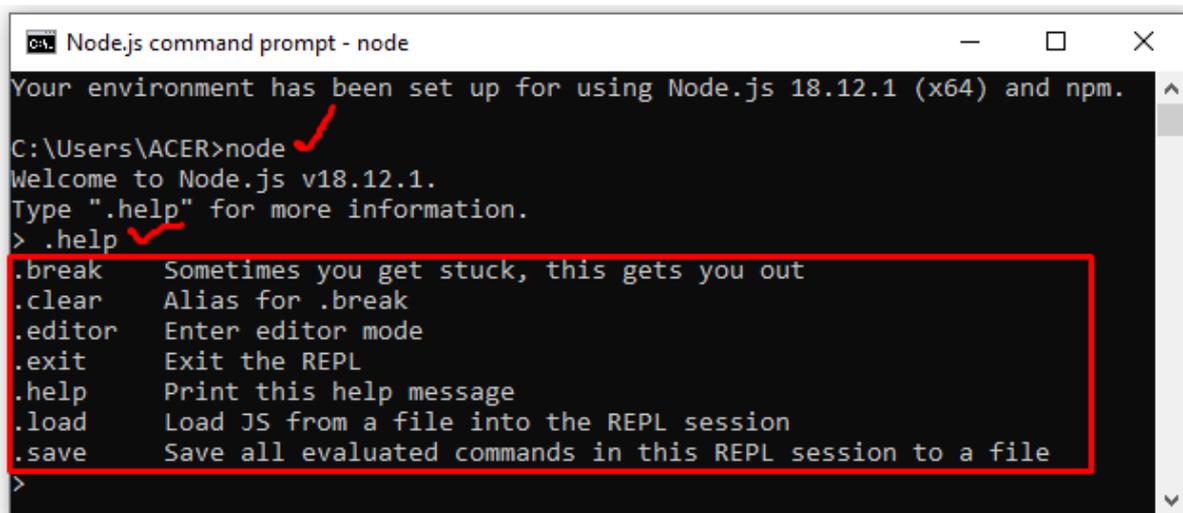
A screenshot of the 'Node.js command prompt' window. The window title is 'Node.js command prompt'. The text inside the window reads: 'Your environment has been set up for using Node.js 18.12.1 (x64) and npm.' Below this, the user has entered the command 'C:\Users\ACER>node --v' and the output is 'node: bad option: --v'. The user then enters 'C:\Users\ACER>node --version' and the output is 'v18.12.1'. The prompt 'C:\Users\ACER>' is visible at the bottom of the window.

FIGURA N° 05-006: Node.js command prompt

REPOL (Read Eval Print Loop-El ciclo de leer evaluar y mostrar) de command prompt-node Node.js

Para ver todo los comandos digitamos >node, luego > .help, FIGURA N° 05-007



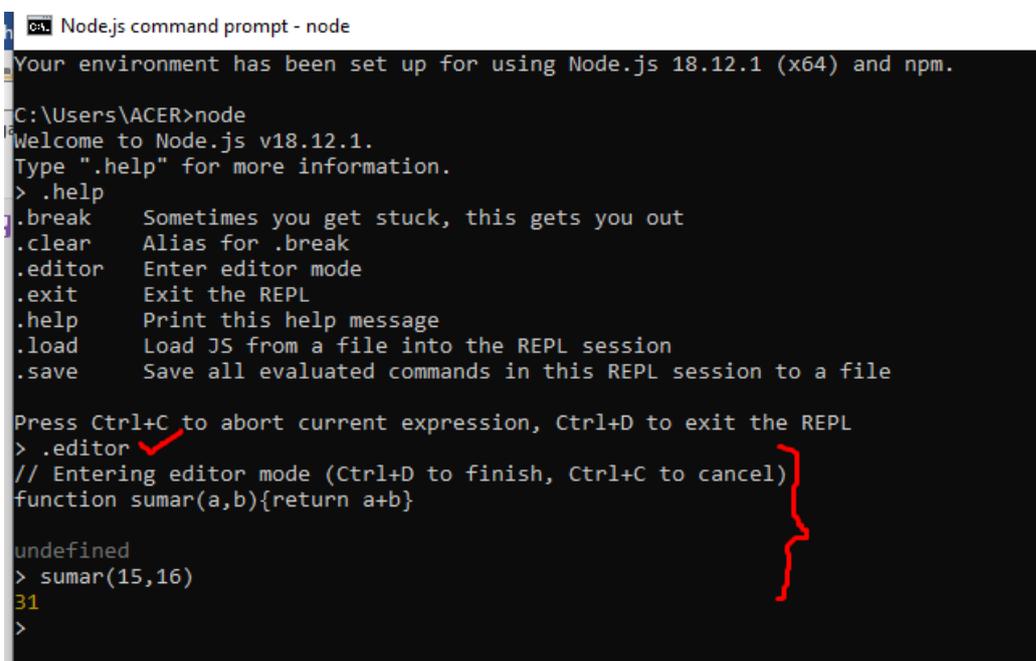
```

Node.js command prompt - node
Your environment has been set up for using Node.js 18.12.1 (x64) and npm.
C:\Users\ACER>node
Welcome to Node.js v18.12.1.
Type ".help" for more information.
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the REPL
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file
>
  
```

FIGURA N° 05-007: Comandos de command prompt-node

Para editar código JavaScript digitamos >. editor, creamos la función sumar (a, b) y guardamos con ctrl+D luego se llama a la función sumar (15,16), FIGURA N° 05-008.

Nota: shift+enter para ingresar siguiente línea en el editor.



```

Node.js command prompt - node
Your environment has been set up for using Node.js 18.12.1 (x64) and npm.
C:\Users\ACER>node
Welcome to Node.js v18.12.1.
Type ".help" for more information.
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the REPL
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file

Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
function sumar(a,b){return a+b}

undefined
> sumar(15,16)
31
>
  
```

FIGURA N° 05-008: Probando código JavaScript con Node.js

Trabajando desde Visual Studio Code

En el siguiente ejemplo creamos una función que genere números aleatorios enteros entre dos valores pasado como argumento a la función, luego se llama dentro de un for para generar varios números aleatorios.

Para ello se crea el archivo **app.js** tal como se muestra en la FIGURA N° 05-009

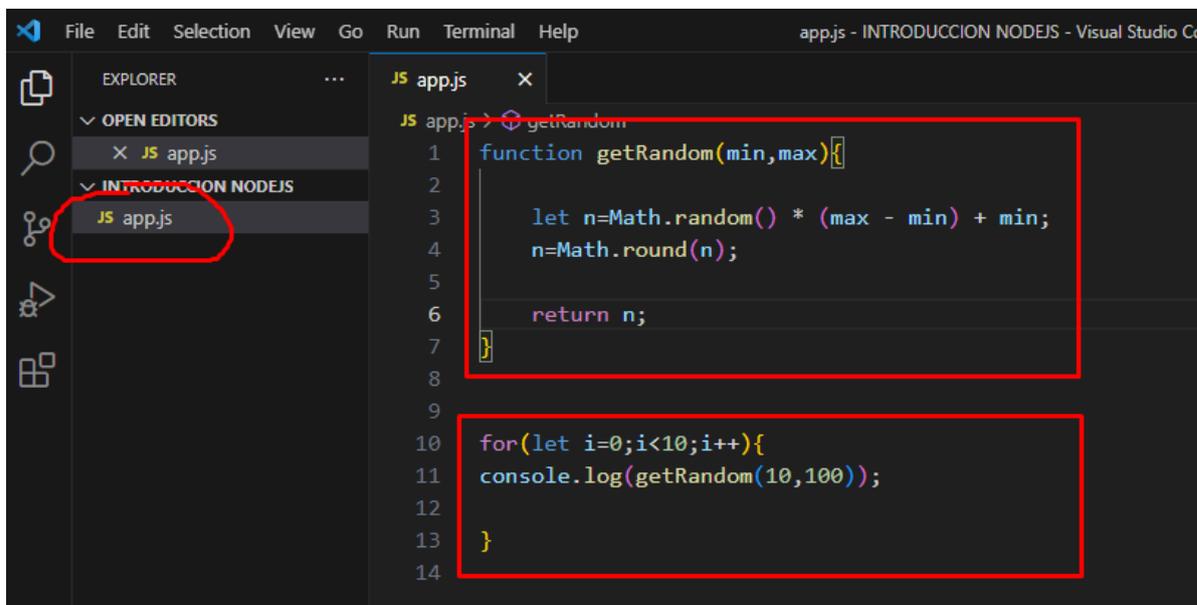


FIGURA N° 05-009: Archivo app.js y función que genera números aleatorios

Para poder visualizar o ejecutar el código sin la necesidad de un navegador, con el terminal de Visual Studio Code (FIGURA N° 05-010) con el comando **>node app.js** El resultado se muestra en la FIGURA N° 05-011.

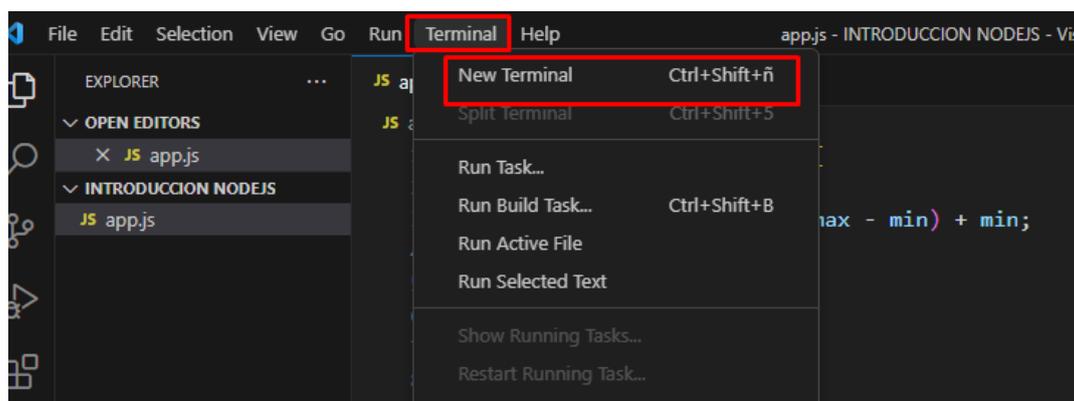


FIGURA N° 05-010: Terminal de Visual Studio Code

```

JS app.js > getRandom
1 function getRandom(min,max){
2
3   let n=Math.random() * (max - min) + min;
4   n=Math.round(n);
5
6   return n;
7 }
8
9 for(let i=0;i<10;i++){
10 console.log(getRandom(10,100));
11 }
12
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS> node app.js
83
73
18
55
34
16
57
72
63
64
PS D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS>

```

FIGURA N° 05-011: Ejecutando JavaScript desde el terminal de Visual Studio Code

Módulos en Node.Js

Los módulos son una o más archivos en paquete, encapsulado con cierta funcionalidad en código JavaScript que puede ser **reutilizada** en una aplicación (FIGURA N° 05-012).

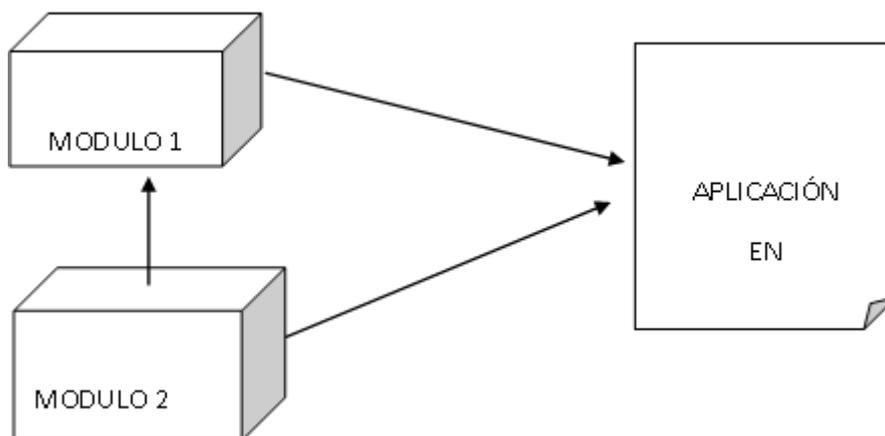


FIGURA N° 05-012: Módulos que se reutilizan en una aplicación JavaScript

Para utilizar un módulo en una aplicación JavaScript se importa con: ***require(nombre_archivo)***.

Así mismo se exporta un módulo que se puede crear en JavaScript con:

module.exports.funcion_exportar= funcion_exportar

Ejemplo de Modulo

En el siguiente ejemplo se crea un archivo ***utilidades.js*** que contendrá varias funciones de utilidad y luego se exporta como modulo y se puede utilizar en otras aplicaciones JavaScript, en este caso se importa a ***app.js***.

Se crea tres funciones para exportar:

utilidades.js

```
function getRandom(min,max){
    let n=Math.random() * (max - min) + min;
    n=Math.round(n);

    return n;
}

function sumar(a,b){
    return a+b;
}

function factorial(n){
    let f=1;
    for(let i=1;i<=n;i++){
        f=f*i;
    }

    return f;
}
```

Se crea el archivo `app.js` en donde se importará el modulo y se utilizara las funciones que se requiera, en este caso `getRandom()` dentro de un `for` `app.js`

```
//se importa el modulo utilidades
//para ello a require se le pasa el nombre del
//archivo en donde esta el modulo
const utilidades=require("../utilidades.js");

//dentro de este for se utilizara
//la funcion getRandom del modulo
//para ello se llama con utilidades.getRandom()

for(let i=0;i<10;i++){
console.log(utilidades.getRandom(10,100));
}
```

Los resultados de ejecutar el `app.js` con `node.js` desde el terminal de Visual Studio Code se muestra en la FIGURA N° 05-013.

```
File Edit Selection View Go Run Terminal Help
app.js - INTRODUCCION NODEJS - Visual Studio Code

JS app.js x JS utilidades.js
JS app.js > ...
3 //se importa el modulo utilidades
4 //para ello a require se le pasa el nombre del
5 //archivo en donde esta el modulo
6 const utilidades=require("../utilidades.js");
7
8 //dentro de este for se utilizara
9 //la funcion getRandom del modulo
10 //para ello se llama con utilidades.getRandom()
11
12 for(let i=0;i<10;i++){
13 console.log(utilidades.getRandom(10,100));
14 }

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS> node app.js
50
27
62
38
25
55
29
95
57
35
```

FIGURA N° 05-013: Importando modulo con `require` a la aplicación `app.js`

Se podría importar solo algunas funciones del módulo para ello se haría con una sintaxis de desestructuración de la siguiente manera:

```
const {sumar,getRandom}=require("./utilidades.js");
```

En ese caso solo se está importando de “./*utilidades.js*” las funciones **sumar** y **getRandom**.

Para llamar a las funciones importados solo se utiliza las constantes definidas relacionados a cada uno de ellos.

```
//se importa el modulo solo sumar y getRandom
const {sumar1,getRandom}=require("./utilidades.js");

//dentro de este for se utilizara
//la funcion getRandom del modulo
//para ello se llama con de manera directa a getRandom

for(let i=0;i<10;i++){
  console.log(getRandom1(10,100));
}
```

Algunos otros ejemplos para importar módulos:

- ✓ Importar módulos del **core** **const filesystems = require('fs')**.
- ✓ Importar módulos de **npm** **const express = require('express')**.
- ✓ Importar un archivo en un proyecto **const server = require('./archivo.js')**.
- ✓ Importar un archivo JSON **const configuracion = require('./config/db.json')**.
- ✓ Importa un index.js que se encuentra en un directorio sin tener que especificarlo **const rutas = require('./rutas')**

Exportando una clase

```
class Alumno {
  constructor (codigo,nombre, apellido,) {
    this.codigo = ciodigo;
    this.nombre = nombre;
    this.apellido = apellido;
  }
  log (message) {
    console.log(`[${this.nombre}] ${message}`)
  }
  info (message) {
    this.log(`info: ${message}`)
  }
}
```

Exportando una instancia:

Módulo fs(File System)

```
class Alumno {
  constructor (codigo,nombre, apellido,) {
    this.codigo = ciodigo;
    this.nombre = nombre;
    this.apellido = apellido;
  }
  log (message) {
    console.log(`[${this.nombre}] ${message}`)
  }
  info (message) {
    this.log(`info: ${message}`)
  }
  verbose (message) {
    this.log(`verbose: ${message}`)
  }
}
```

```
module exports = new Alumno("2023","JUAN","GARCIA SANTILLAN");
```

Módulo de Systema de Archivos, que contiene funcionalidad para trabajar con el sistema de archivos y carpetas. Algunas operaciones:

- ✓ Leer

- ✓ Modificar
- ✓ Copiar
- ✓ Eliminar
- ✓ Cambiar nombres

Ejemplo de manejo de archivo

En el siguiente ejemplo se tiene un archivo **ítem.json**, sobre el realizaremos algunas operaciones con el modulo fs.

Leyendo el archivo **ítem.json**:

```
//se importa el modulo de manejo de archivos fs
const fs=require('fs');

//se leera el archivo item.json

fs.readFile('item.json','utf-8',(err,archivo)=>{
  if(err){
    console.log(err);
  }else{
    console.log(archivo)
  }
})
```

Se importa el módulo con **require('fs')**, luego con el método **fs.readFile** se pasa el nombre del archivo y otros argumentos más.

En el tercer argumento tenemos la función que tiene dos parámetros: **err, archivo**,

Si se produce algún error lo tendremos en '**err**' el cual se imprime en consola, si todo esta correcto el contenido del archivo **ítem.json** estará en **archivo** el cual es impreso en consola tal como se muestra en la FIGURA N° 05-014.

```

File Edit Selection View Go Run Terminal Help
app.js - INTRODUCCION NODEJS - Visual Studio Code

EXPLORER
OPEN EDITORS
  X JS app.js
INTRODUCCION NODEJS
  JS app.js
  {} item.json
  JS utilidades.js

JS app.js
1 fs.readFile('item.json', 'utf-8') callback
2 //se importa el modulo de manejo de archivos fs
3 const fs=require('fs');
4 //se leera el archivo ite.json
5 fs.readFile('item.json', 'utf-8', (err, archivo)=>{
6   if(err){
7     //console.log(err);
8     //utilizamos throw en remplazo de console
9     //para paralizar la ejecucion de codigo
10    //luego que se detecto un error
11    throw err;
12  }else{
13    console.log(archivo);
14  }
15 }

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS> node app.js

{
  "type": "articles",
  "id": "1",
  "attributes": {
    "title": "Rails is Omakase"
  },
  "relationships": {
    "author": {
      "links": {
        "self": "http://example.com/articles/1/relationships/author",
        "related": "http://example.com/articles/1/author"
      },
      "data": { "type": "people", "id": "9" }
    }
  },
  "links": {
    "self": "http://example.com/articles/1"
  }
}
  
```

FIGURA N° 05-014: Leyendo archivo con el módulo **fs**

El mensaje de error cuando es el mismo se muestra en la FIGURA N° 05-015, para que ocurra ello vasca pasarlo como parámetro un archivo que no existe:

```

JS app.js X
JS app.js > ...
1 //se importa el modulo de manejo de archivos fs
2 const fs=require('fs');
3
4 //se leera el archivo ite.json
5 fs.readFile('item1.json', 'utf-8', (err, archivo)=>{
6   if(err){
7     //console.log(err);
8     //utilizamos throw en remplazo de console
9     //para paralizar la ejecucion de codigo
10    //luego que se detecto un error
11    throw err;
12  }else{
13    console.log(archivo);
14  }
15 }

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE powershell

PS D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS> node app.js
D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS>app.js:11
  throw err;
  ^
[Error: ENOENT: no such file or directory, open 'D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS\item1.json'] {
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'D:\UNHEVAL\POYECTO DE LIBROS PARA PUBLICAR\LIBRO DESARROLLO DE SOLUCIONES WEB\CAPITULO V\INTRODUCCION NODEJS\item1.json'
}

Node.js v18.12.1
  
```

FIGURA N° 05-015: Cuando ocurra un error se imprime el contenido de **err**

Para modificar el nombre del archivo **ítem.json** a **producto.json: fs.rename()**

```
//se importa el modulo de manejo de archivos fs
const fs=require('fs');

//se modifica el nombre del archivo item.json
fs.rename('item.json','producto.json',(err,archivo)=>{
  if(err){
    throw err;
  }else{
    console.log('Se modificó el nombre del archivo')
  }
})
```

Agregando contenido al final del archivo *producto.json*: *fs.appendFile()*

```
//se importa el modulo de manejo de archivos fs
const fs=require('fs');
//agregando texto al final del archivo
fs.appendFile ('producto.json','TEXTO AGREGADO',(err)=>{
  if(err){
    throw err;
  }else{
    console.log('Se agrego texto al archivo')
  }
})
```

Remplazar todo el contenido del archivo ***producto.json: fs.writeFile()***

```
//se importa el modulo de manejo de archivos fs
const fs=require('fs');

//eliminando todo el contenido de un archivo
fs.writeFile('producto.json','CONTENIDO NUEVO',(err)=>{
  if(err){
    throw err;
  }else{
```

Eliminando archivo ***producto.json: fs.unlink()***

```
//se importa el modulo de manejo de archivos fs
const fs=require('fs');

//eliminando archivo
fs.unlink('producto.json',(err)=>{
  if(err){
    throw err;
  }else{
    console.log('Archivo eliminado')
  }
})
```

Nota: Para trabajar de manera síncrono y que se ejecute las sentencias una de tras del otro conforme al orden programado, a cada método agregar “Sync”: fs.writeFile Sync()

npm(Node Package Manager)

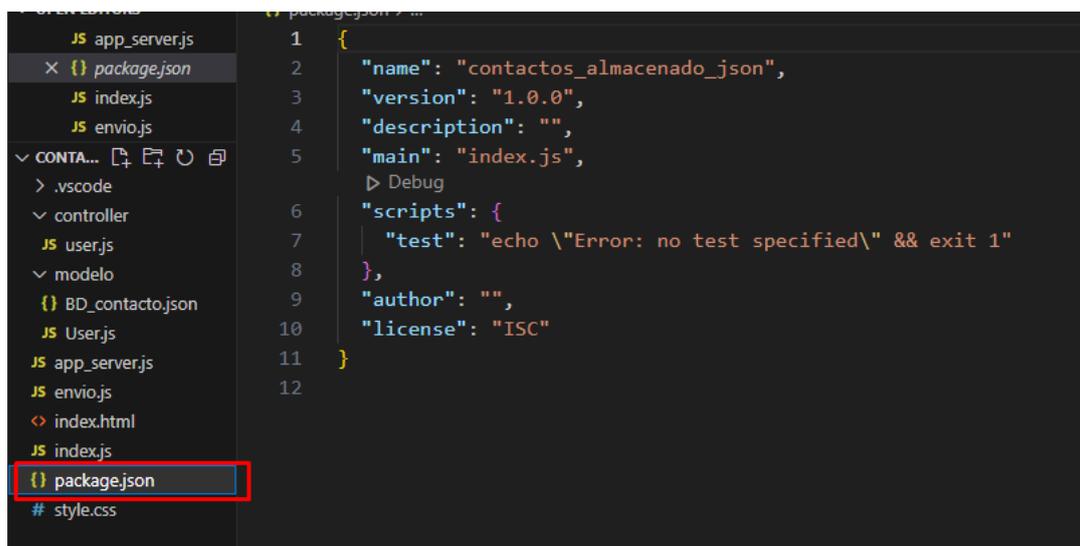
Es el archivo de software más grande, que contiene **paquetes (archivo o directorio descrito por un archivo package.json)** que se puede instalar y usar en Node.js, permitiendo que cientos de desarrolladores lo puedan compartir entre distintos proyectos. Se compone de al menos dos partes principales.

- ✓ Un repositorio online para publicar paquetes de software libre para ser utilizados en proyectos Node.js
- ✓ Una herramienta para la terminal (command line utility) para interactuar con dicho repositorio que te ayuda a la instalación de utilidades, manejo de dependencias y la publicación de paquetes

package.json

Este archivo contiene la información del paquete incluyendo la descripción del mismo, versión, autor y dependencias.

Es generado automáticamente mediante la ejecución de un script de **npm: >npm init** este script es ejecutado para inicializar un proyecto JavaScript. Se puede ver el contenido en la FIGURA N° 05-016.



```

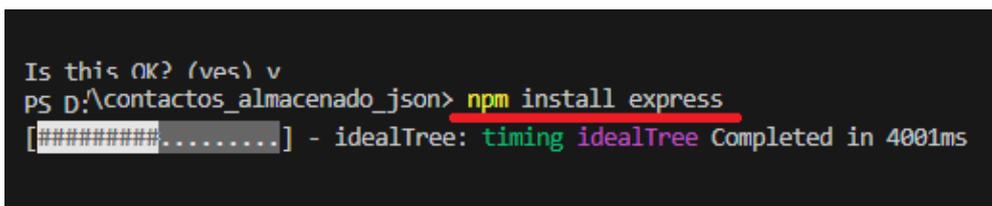
1  {
2    "name": "contactos_almacenado_json",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC"
11  }
12

```

FIGURA N° 05-016: El archivo package.json

package-lock.json

Este archivo es auto generado por **npm install** y es una lista descriptiva y exacta de las versiones instaladas durante el proceso, en proceso de instalación se puede ver la pantalla tal como se muestra en la FIGURA N° 05-017.



```

Is this OK? (yes) y
ps D:\contactos_almacenado_json> npm install express
[#####.....] - idealTree: timing idealTree Completed in 4001ms

```

FIGURA N° 05-017: Progreso de instalación del Express

Parte del contenido de este archivo se puede visualizar en la FIGURA N° 05-018.

```

1  {
2    "name": "contactos_almacenado_json",
3    "version": "1.0.0",
4    "lockfileVersion": 2,
5    "requires": true,
6    "packages": {
7      "": {
8        "name": "contactos_almacenado_json",
9        "version": "1.0.0",
10       "license": "ISC",
11       "dependencies": {
12         "express": "^4.18.2"
13       }
14     },
15     "node_modules/accepts": {
16       "version": "1.3.8",
17       "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.8.tgz",
18       "integrity": "sha512-PYAthTa2m2VKxuvSD3DPC/Gy+U+sOA1LAuT8mkmRuvw+NACSA
19     },
20     "dependencies": {
21       "mime-types": "~2.1.34",
22       "negotiator": "0.6.3"

```

FIGURA N° 05-018: Parte del contenido del archivo package-lock.json

5.2. Express

Es un **framework backend** que proporciona un conjunto de herramientas para aplicaciones web en Node.js, peticiones y respuestas HTTP, enrutamiento y middleware (sistema de software que ofrece funciones y servicios de nube comunes para las aplicaciones) para construir y desplegar aplicaciones a gran escala, diseñado para construir aplicaciones web de una sola página, multipágina e híbridas.

Par utilizar en un proyecto se instala utilizando npm: **> npm install express.**

Creando un servidor que escuche peticiones entrantes con Express

Para ello una vez instalado Express, para una aplicación se importa con **require('express')**, tal como se muestra en el siguiente ejemplo.

Se crea el archivo **app_server.js**, sobre ello ingresamos el siguiente código:

```
const express = require('express')

const app = express()

const port = 4000

//método que permite escuchar por el puerto
app.get('/', (request, response) => {
  response.send('Servidor corriendo!')
})
```

Para poner en marcha el servidor ejecutamos **>node app_server.js**

Se podría visualizar desde un navegador web ingresando en la barra de direcciones: localhost:8888, y tendremos la respuesta del servidor tal como se muestra en la FIGURA N° 05-019.

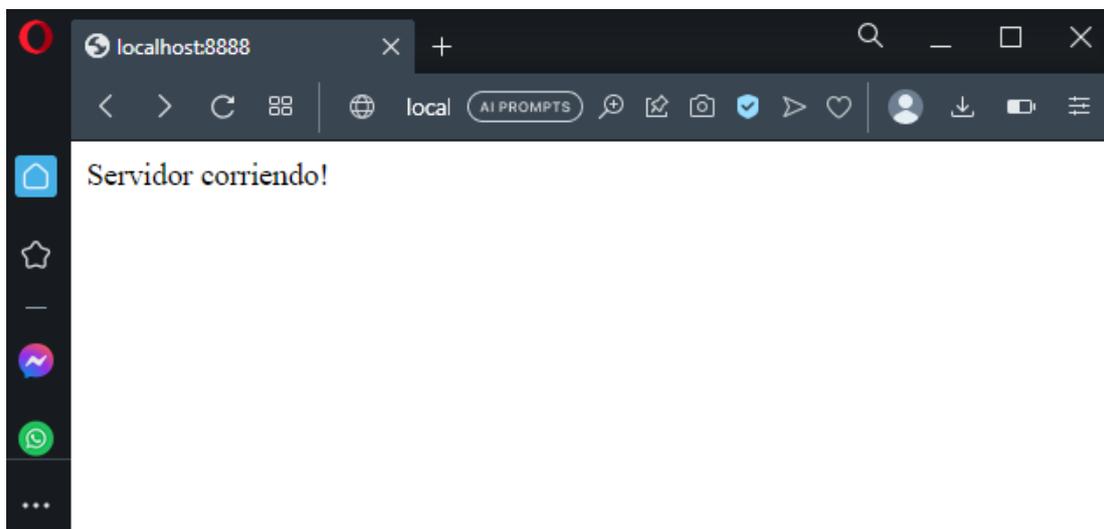


FIGURA N° 05-019: Servidor en marcha con Express

Haciendo que el servidor responda el contenido de un arreglo a la solicitud: localhost:8888/listaAlumno, luego de configurar el servidor podemos ingresar el siguiente código:

```
const alumnos = [  
  {  
    codigo: "2015120128", nombre: "JOSE", apellido: "CHAVEZ CARDENAS",  
  },  
  {  
    codigo: "2015120999", nombre: "MARIA", apellido: "COTRINA CHAVEZ"  
  },  
  {  
    codigo: "2018235488", nombre: "JUAN", apellido: "SANTILLAN CORNEJO"  
  },  
  {  
    codigo: "2023654789", nombre: "ELMER", apellido: "CUBA SALDVAR"  
  }  
];
```

La respuesta del servidor a la petición **localhost:8888/listaAlumnos** se muestra como en la FIGURA N° 05-020.

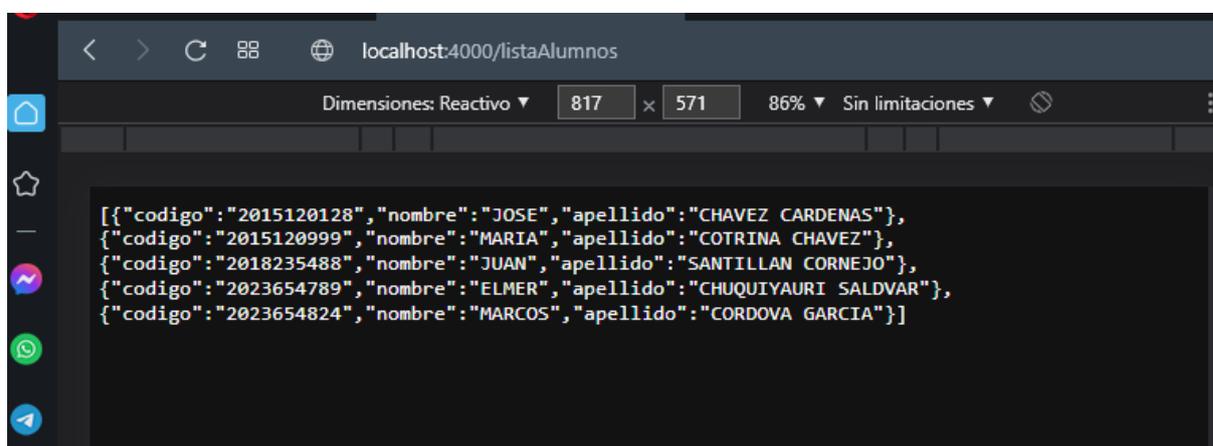


FIGURA N° 05-020: Respuesta del servidor en formato JSON

5.3. Motor de plantilla EJS

Un **motor de plantilla** es texto que es procesado y convertido en html.

El **EJS** es un motor de plantilla que permite generar HTML e insertar código JavaScript del lado del cliente. Una representación de lo que es Node.js, Express.js y EJS se muestra en la FIGURA N° 05-021.

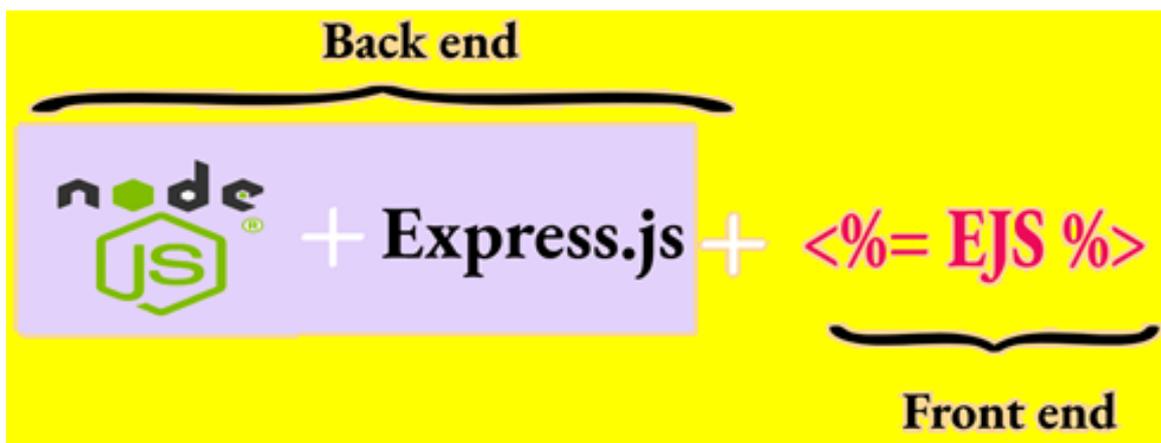


FIGURA N° 05-021: Rol del EJS

Para utilizar se instala la dependencia con **>npm install ejs**

Cuando se requiere hacer uso del EJS, los archivos se crean con extensión **.ejs** a cambio de **.html**.

La documentación del EJS se puede revisar en la página (FIGURA N° 05-022 y FIGURA N° 05-023): <https://ejs.co/>

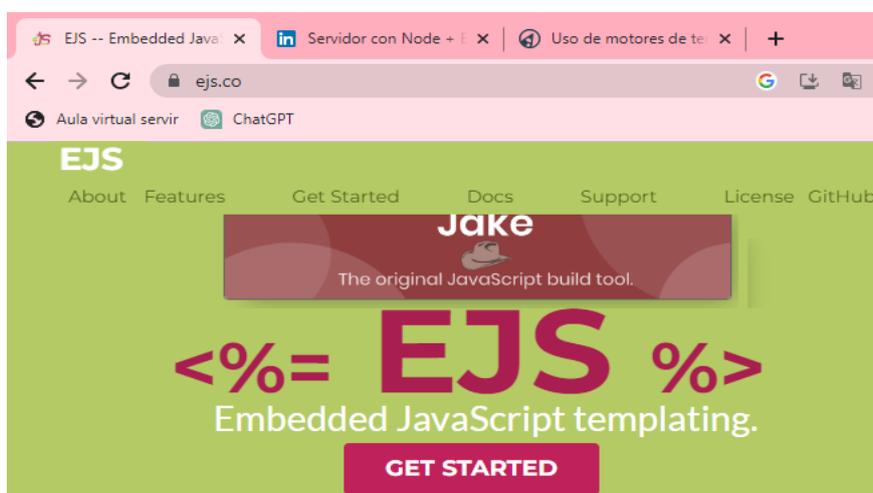


FIGURA N° 05-022: Pagina de la documentación de EJS

About Features **Get Started** Docs Support Li

Get Started

Install

It's easy to install EJS with NPM.

```
$ npm install ejs
```

Use

Pass EJS a template string and some data. BOOM, you've got some HTML.

```
let ejs = require('ejs');
let people = ['geddy', 'neil', 'alex'];
let html = ejs.render('<%= people.join(", "); %>', {people: people});
```

CLI

Feed it a template file and a data file, and specify an output file.

```
ejs ./template_file.ejs -f data_file.json -o ./output.html
```

FIGURA N° 05-023: Pagina de la documentación de EJS

En el siguiente ejemplo se creará una carpeta **views**, luego se crea el archivo **index.ejs**

index.ejs

```
<html>
  <head>
    <link rel="stylesheet" href="./css/main.css" />
  </head>
  <body>
    <h1>Iniciando con EJS</h1>
  </body>
</html>
```

Luego se llama desde el archivo del servidor:

```
// ruta para el index
app.get("/", function (request, response) {
  response.render("index.ejs");
});
```

Enviando parámetros a un template(.ejs) en Express

Desde la ruta en `.render()` como segundo parámetro enviamos datos:

```
// ruta para matricula con un DNI y nombre
app.get("/matricula/:DNI/:nombre/", function (request, response) {
  response.render("indexAlumno.ejs", {
    DNI: request.params.DNI,
    nombre: request.params.nombre,
  });
});
```

Se crea el archivo `indexAlumno.ejs` para recepcionar los parámetros de: **DNI**, **nombre** para llamarlo desde la url:

<http://localhost:4000/matricula/22451077/ELMER%20CHUQUIYAURI>

Recibiendo parámetros en el archivo `indexAlumno.ejs` y pasando como variable en el HTML con `<%= variable %>` para que se visualice: `indexAlumno.ejs`

```
<html>
  <head>
    <link rel="stylesheet" href="/css/main.css" />
  </head>
  <body>
    <h1>El DNI del alumno es:<%= DNI %></h1>
    <p>El NOMBRE del alumno es:<%= nombre %></p>
  </body>
</html>
```

Para visualizar desde la barra de direcciones de un navegador:

<http://localhost:4000/matricula/22451077/ELMER%20CHUQUIYAURI>

Se puede visualizar el resultado como en la FIGURA N° 05-024

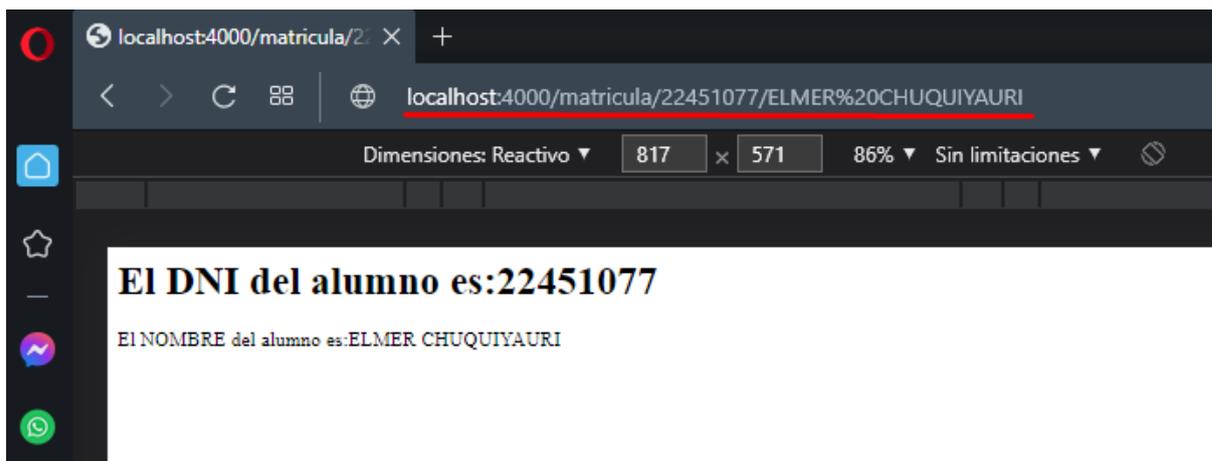


FIGURA N° 05-024: Pasando parámetros a una plantilla ejs

EJEMPLO DE APLICACIÓN N°001

Desarrollar un aplicativo para gestionar usuarios (crear nuevos, modificar, eliminar, listar). El almacenamiento de los datos se hacerlo en un archivo ***data.json***

Para el aplicativo crear el diseño mostrado en la FIGURA N° 05-025

 The figure shows a user management interface on a yellow background. On the left is a form titled 'USUARIOS' with input fields for 'Apellidos y Nombres' (with a sub-field for 'Apellido'), 'Correo', 'Telefono', 'Cuenta', and 'Password'. Below the form are two buttons: 'NUEVO' and 'CANCELAR'. On the right is a table with the following data:

N°	Apellidos y Nombres	Correo	Telefono	Cuenta	Password
	CHUQUIYAURI SALDIVAR, ELMER	ELMER@HOTMAIL.COM	970988938	ELMER	123456
	SANTILLAN GARCIA, MARIA	MARIA@GMAIL.COM	9854785	MARIA	23547
	GARCIA MATOS, JUAN	JUAN@GMAIL.COM	95684658	JUAN@GMAIL.COM	125478

FIGURA N° 05-025: Diseño de formulario para gestionar usuarios

DESARROLLO

Para el desarrollo del siguiente aplicativo se utiliza todo lo descrito anteriormente:

- ✓ Node.js
- ✓ Express
- ✓ EJS

Se crea el archivo ***gestion_usuario*** y se abre en el ***Visual Studio Code*** para instalar las dependencias que se requiere y se crea las carpetas y archivos necesarios:

1. Iniciando la creación del Proyecto >*node init*
2. Instalando Express >*npm i express*
3. Instalando EJS >*npm i ejs*

Se crea las carpetas y los archivos necesarios: **css**, **src**, **views** y **server** (para programar del lado del servidor), la carpeta **node_modules** y **.vscode** se crean automáticamente cuando se instalan las dependencias, tal como se muestra en la FIGURA N° 05-026.

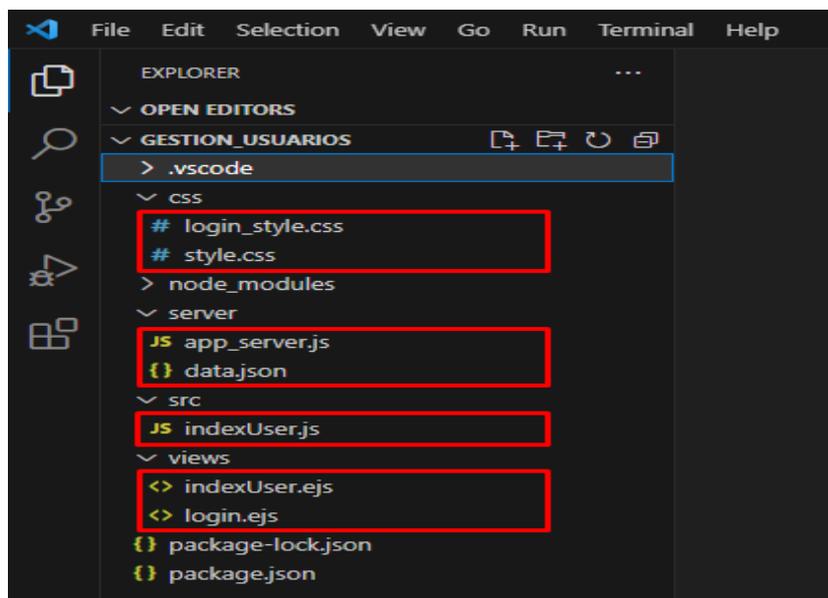


FIGURA N° 05-026: Carpetas para el aplicativo

Diseño de la interfaz (archivo indexUser.ejs):

Para el diseño de la interfaz en la carpeta **views** se crea el archivo **indexUser.ejs** y se agrega las etiquetas **HTLM** necesarias; dentro de <body> se agrega <main> y dentro de <main> se agregan dos etiquetas <section>, uno como contenedor de la parte 1 que contendrá también un formulario <form> con todas las etiquetas que se requiere según el prototipo y otro para la parte 2 que contendrá una tabla <table>, como se ve en la FIGURA N° 05-027.



FIGURA N° 05-027: Prototipo del aplicativo

Etiquetas del <head>:

Etiqueta	href
<link>	href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200"
	Google Fonts: fuentes e iconos de google
<link>	href="style.css"
	Estilos de la aplicación

Etiquetas del <body>:

Etiqueta	type	id	clas	name	texto
<main>					
<section>		formContainer			
<h1>					USUARIOS
<form>		formulario			
<Input>	hidden	txtId		id	
<label>					Apellidos y Nombres
<input>	text	txtApellido		apellido_nombre	Apellido
<label>					Correo
<input>	text	txtCorreo		correo	Correo
<label>					Telefono
<input>	text	txtTelefono		telefono	Telefono
<label>					Cuenta
<input>	text	txtCuenta		cuenta	cuenta
<label>					Password
<input>	text	txtPasswor		password	password
<button>		btnOperacion			NUEVO
		data-operacion="C"			

<button>		btnCancelar			CANCELAR
<form>		formularioHidden			
<input>	hidden	id_txt_formularioHidden		id	
<section>		tableContainer			
<table>					
<thead>					
<tr>					
<th>					E
<th>					M
<th>					Apellidos y Nombres
<th>					Correo
<th>					Telefono
<th>					Cuenta
<th>					Password
<tbody>	CONTENDRA LAS FILAS QUE SE GENERARA DINAMICAMENTE				
<script>	src="indexUser.js"				
	Script que tiene la lógica para interactuar con aplicativo				
<script>	src="https://cdn.jsdelivr.net/npm/sweetalert2@11"				
	<!--PARA LAS VENTANAS DE MENSAJE-->				

Contenido del archivo indexUser.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined:opsz,wght,FILL,GRAD@20..48,100..700,0..1,-50..200" />
  <link rel="stylesheet" href="style.css">
  <title>Document</title>
</head>
<body>
  <main>
    <section id="formContainer">
      <h1>USUARIOS</h1>
      <form id="formulario">
        <input type="hidden" id="txtId" name="id">
        <label for="">Apellidos y Nombres</label>
        <input type="text" id="txtApellido" name="apellido_nombre" placeholder="Apellido" disabled>
        <label for="">Correo</label>
        <input type="text" id="txtCorreo" name="correo" placeholder="Correo" disabled>
        <label for="">Telefono</label>
        <input type="text" id="txtTelefono" name="telefono" placeholder="Telefono" disabled>
        <label for="">Cuenta</label>
        <input type="text" id="txtCuenta" name="cuenta" placeholder="cuenta" disabled>
      </form>
    </section>
  </main>
</body>
</html>
```

```
<input type="text" id="txtPasswor" name="password" disabled>
  <button id="btnOperacion" data-operacion="C">NUEVO</button>
  <button id="btnCancelar" >CANCELAR</button>
</form>

<form action="" id="formularioHidden">
  <input type="hidden" id="id_txt_formularioHidden" name="id">
</form>

</section>
<section id="tableContainer">
  <table>
    <thead>
      <tr>

        <th>E</th>
        <th>M</th>
        <th>Apellidos y Nombres</th>
        <th>Correo</th>
        <th>Telefono</th>
        <th>Cuenta</th>
        <th>Password</th>
      </tr>
    </thead>
```

```

<tbody>
  <!-- CON EJS SE CARGARA LAS FILAS CON LOS DATOS PASADO DESDE EL SERVIDOR -->
  <% data.forEach(function(user){%>
    <!--SE RECORRE TODO EL ARREGLO DE DATOS Y SE VA CREANDO LAS FILAS DE LA TABLA-->
    <tr>
      <td><span class="material-symbols-outlined" data-id="<%=user.id %>">delete</span></td>
      <td><span class="material-symbols-outlined" data-id="<%=user.id %>">edit</span></td>
      <td><%=user.apellido_nombre %> </td>
      <td><%=user.correo %> </td>
      <td><%=user.telefono %> </td>
      <td><%=user.cuenta %> </td>
      <td><%=user.password %> </td>
    </tr>
  <% }) %>
</tbody>
</table>
</section>
</main>
<script src="indexUser.js" ></script>
<!--PARA LAS VENTANAS DE MENSAJE-->
<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
</body>
</html>

```

Contenido del archivo style.css: para dar estilos al HTML

```
*{
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}

/*Estilos al body*/
body{
  background-color: rgb(255, 238, 0);
}

/**-----/
/*Estilos al contenedor main*/
main{
  display: grid;
  grid-template-columns: 1fr 2fr;
  gap: 10px;
}

/**-----/
```

```
/**-----*/  
/*Estilos al formulario*/  
#formContainer{  
  padding: 20px;  
  /*-----*/  
  /*estilo de fondo*/  
  width: 95%;  
  max-width: 100%;  
  margin: auto;  
  margin-top: 5px;  
  padding: 5px;  
  box-shadow: 0 0 20px 1px rgba(0, 0, 0, 0.3);  
  position: relative;  
  /*-----*/  
}  
  
#formContainer h1{  
  text-align:center;  
  margin-bottom: 20px;  
}  
  
#formContainer form{  
  display: flex;  
  flex-direction: column;  
  gap: 5px;  
}
```

```
#formContainer form input{
  padding: 10px;
  border-radius: 10px;
  border: 2px solid rgb(212,212,212);
}

#formContainer form button{
  margin-top: 10px;
  padding: 10px;
  border-radius: 10px;
  border: none;
  border:2px solid rgb(11, 15, 240);
  background-color: black;
  color:white;
}

#formContainer form button:hover{
  cursor: pointer;
  transform: scale(0.95);
  background-color: rgb(7, 23, 161);
}

/*-----*/
```

```
/*-----*/  
/*Estilos de la tabla*/  
#tableContainer{  
  background-color:white;  
  
  /*-----*/  
  /*estilo de fondo*/  
  width: 95%;  
  max-width: 100%;  
  margin: auto;  
  margin-top: 5px;  
  padding: 5px;  
  box-shadow: 0 0 20px 1px rgba(0, 0, 0, 0.3);  
  position: relative;  
  /*-----*/  
  overflow:auto;  
  height:500px;  
}  
table,td,th,tr{  
  border:1px solid black;  
  border-collapse: collapse;  
}  
table{  
  width: 100%;  
}
```

```
th{
  background-color: black;
  color: white;
  padding: 10px;
}

td{
  padding: 10px;
}

table span:hover{
  cursor: pointer;
}

table tr:hover{
  cursor: pointer;
  background-color: grey;
}
```

```
/**-----*/  
/**PARA QUE LA PANTALLA SE ADECUÉ SEGUN EL TAMAÑO*/  
@media only screen and (max-width: 600px) and (min-width: 400px) {  
  
  main{  
    grid-template-columns: 1fr;  
  }  
  
}  
  
@media only screen and (max-width: 1200px) and (min-width: 600px) {  
  
  main{  
    grid-template-columns: 1fr 2fr;  
  }  
  
}  
/**-----*/
```

Programación del lado del cliente (Frontend)

Para interactuar con el aplicativo desde el lado del cliente se crea dentro de la carpeta **scr** el archivo **indexUser.js**, que contendrá la logia de la interacción.

En el formulario inicialmente se tiene todos los **inputs** deshabilitado, en el botón **“btnOperacion”** inicialmente con etiqueta **“NUEVO”** al hacer click habilita todos los inputs(**.disabled=false**) y cambia su etiqueta a **“GRABAR”**(**e.target.textContent="GRABAR"**), siempre en cuando el valor del **.dataset.operacion** sea **“C”** que indica que es un nuevo usuario, este mismo botón permitirá modificar datos, para lo cual se cambia el valor de **.dataset.operacion** a **“M”** y su etiqueta a **“MODIFICAR”** (**btnOperacion.textContent="MODIFICAR"**) desde el icono de **edit** de un registro de la tabla(que a su vez captura todo los valores del registro y se asigna a los inputs del formulario), luego cuando se hace click en el botón **“btnOperacion”** con etiqueta **“MODIFICAR”** se cambia a **“GRABAR”**(**e.target.textContent="GRABAR"**).

Para ambos casos (creando un nuevo usuario o modificando datos de un usuario) cuando el botón **“btnOperacion”** tiene como etiqueta **“GRABAR”**, al hacer click se envía al servidor los datos del formulario, para ello se asigna el método de envío a **“post”**(**formulario.method="post"**), la ruta de envío a **“newUser”**(**formulario.action="newUser"**), y se llama a **submit()** (**formulario.submit()**), así mismo forzamos un click al botón **“btnCancelar”** (**btnCancelar.onclick(e)**) para limpiar los inputs del formulario así como se muestra en la FIGURA N° 05-028.

```

else{//PARA MODIFICAR
  if(e.target.textContent==="MODIFICAR")
  {
    //se habilita los controles
    txtApellido.disabled=false;
    txtCorreo.disabled=false;
    txtTelefono.disabled=false;
    txtCuenta.disabled=false;
    txtPasswor.disabled=false;

    e.target.textContent="GRABAR";
  }
  else{
    //se envia datos al servidor
    formulario.method="post"
    formulario.action="newUser"
    formulario.submit();
    btnCancelar.onclick(e);
  }
}

```

FIGURA N° 05-028: Parte de código de envío de datos al servidor

Contenido completo del archivo indexUser.js:

```
//-----  
//capturando las etiquetas HTML  
const body=document.querySelector("body");  
  
//etiquetas del formulario  
const formulario=document.getElementById("formulario");  
const txtId=document.querySelector("#txtId");  
const txtApellido=document.querySelector("#txtApellido");  
const txtCorreo=document.querySelector("#txtCorreo");  
const txtTelefono=document.querySelector("#txtTelefono");  
const txtCuenta=document.querySelector("#txtCuenta");  
const txtPasswor=document.querySelector("#txtPasswor");  
  
//capturando el contenedor de la tabla  
const tableContainer=document.querySelector("#tableContainer");  
  
//capturando los botones  
const btnOperacion=document.querySelector("#btnOperacion");  
const btnCancelar=document.querySelector("#btnCancelar");  
  
//capturando el formulario y su input que esta como oculto, para el envio para eliminar  
const formularioHidden=document.getElementById("formularioHidden");  
const id_txt_formularioHidden=document.getElementById("id_txt_formularioHidden");  
//-----
```

```
//-----  
//se programa en el evento click del boton btnOperacion  
btnOperacion.onclick=(e)=>{  
    e.preventDefault();  
  
    if(e.target.dataset.operacion==="C")  
    {//PARA INSERTAR UN NUEVO USUARIO  
  
        if(e.target.textContent==="NUEVO")  
        {  
            //se limpia el contenido de controles  
            txtApellido.value="";  
            txtApellido.disabled=false;  
            txtCorreo.value="";  
            txtCorreo.disabled=false;  
            txtTelefono.value="";  
            txtTelefono.disabled=false;  
            txtCuenta.value="";  
            txtCuenta.disabled=false;  
            txtPasswor.value="";  
            txtPasswor.disabled=false;  
  
            e.target.textContent="GRABAR";  
        }  
    }  
}
```

```
    else{
        formulario.method="post"
        formulario.action="newUser"
        formulario.submit();
        btnCancelar.onclick(e);
    }
}
else{//PARA MODIFICAR DATOS DE USUARIO
    if(e.target.textContent==="MODIFICAR"){
        //se habilita los controles
        txtApellido.disabled=false;
        txtCorreo.disabled=false;
        txtTelefono.disabled=false;
        txtCuenta.disabled=false;
        txtPasswor.disabled=false;
        e.target.textContent="GRABAR";
    }
    else{
        //se envia datos al servidor
        formulario.method="post"
        formulario.action="newUser"
        formulario.submit();
        btnCancelar.onclick(e);
    }
}
}
```

```
//-----  
//evento click del btnCancelar  
//se asigna cadena vacia a todos los inputs  
btnCancelar.onclick=(e)=>{  
  
    e.preventDefault();  
  
    txtId.value="";  
    txtApellido.value="";  
    txtApellido.disabled=true;  
    txtCorreo.value="";  
    txtCorreo.disabled=true;  
    txtTelefono.value="";  
    txtTelefono.disabled=true;  
    txtCuenta.value="";  
    txtCuenta.disabled=true;  
    txtPasswor.value="";  
    txtPasswor.disabled=true;  
  
    btnOperacion.textContent="NUEVO";  
    btnOperacion.dataset.operacion="C";  
}  
//-----
```

```
//-----  
// se programa el evento click de los iconos de delete y edit de  
// cada registro de la tabla  
tableContainer.onclick=(e)=>{  
  
  //si se hace click en el icono delete  
  if(e.target.textContent==="delete")  
  {  
  
    //-----  
    //SACAMOS UNA VENTANA DE DIALOGO  
    //PARA PEDIR CONFIRMACION DE BORRADO  
  
    Swal.fire({  
      title: 'Eliminar registro de todas maneras?',  
      text: "Se borrarán los datos definitivamente de la BD",  
      icon: 'warning',  
      showCancelButton: true,  
      confirmButtonColor: '#3085d6',  
      cancelButtonColor: '#d33',  
      confirmButtonText: 'Si, Borrar!' })  
    .then((result) => {  
      if (result.isConfirmed) {
```

```
//-----  
//se borra de la coleccion de objetos  
//delete users[e.target.dataset.id];  
Swal.fire(  
  'Se borro',  
  'los datos correctamente.',  
  'success'  
)  
  
id_txt_formularioHidden.value=e.target.dataset.id;  
formularioHidden.method="post"  
formularioHidden.action="deleteUser"  
formularioHidden.submit();  
  
}  
})  
//-----  
  
}
```

```
//se captura el id del dataset del icono edit
txtId.value=e.target.dataset.id;

//se captura el nodo(columna) siguiente donde esta el icono edit
//que en este caso seria el apellido y nombre
nodo=nodo.nextElementSibling;
txtApellido.value=nodo.textContent;

//se captura la siguiente columna donde esta el correo
nodo=nodo.nextElementSibling;
txtCorreo.value=nodo.textContent;

//se captura la siguiente columna donde esta el telefono
nodo=nodo.nextElementSibling;
txtTelefono.value=nodo.textContent;

//se captura la siguiente columna donde esta la cuenta
nodo=nodo.nextElementSibling;
txtCuenta.value=nodo.textContent;

//se captura la siguiente columna donde esta el password
nodo=nodo.nextElementSibling;
txtPasswor.value=nodo.textContent;
```

```
//se cambia la etiqueta del boton btnOperacion
btnOperacion.textContent="MODIFICAR";

//se cambia el data-operacion del btnOperacion
//para indicar que es una modificacion de datos
btnOperacion.dataset.operacion="M"

}
```

Programación desde el lado del servidor (Backend)

Con Node.js y Express instalado, se programa el servidor para atender todos los pedidos del cliente así mismo gestionar el almacenamiento de los usuarios (almacenar nuevos usuarios, modificar, eliminar, generar reportes, entre otros); por lo que se crea el archivo **app_server.js** dentro de la carpeta **server**, en lo que se programara la configuración del servidor, así mismo la gestión de usuarios (crear, modificar, eliminar y listar).

Para hacer persistente los datos se crea el archivo dentro de la carpeta **server: data.json** que permitirá gestionar usuarios en formato JSON.

Se crea la **const express** y se requiere el módulo o dependencia express (**const express = require('express')**), de igual forma se crea la constante **app** y se llama al módulo express(), se crea la constante **PORT** el cual será la variable de entorno **process.env.PORT** la que lo asigne, si no encuentra la variable de entorno entonces asigne el puerto 4000. Es recomendable dejar que sea la variable de entorno la que asigne el puerto.

Se levanta el servidor con express, para ello se utiliza el comando.

app.listen(PORT, (req, res) => { console.log(`Url de prueba: http://localhost:\${PORT} `)}); Este comando le dirá al servidor que escuche el puerto y reciba una respuesta. Si es correcta indicará en la terminal el mensaje “**Url de prueba: http://localhost:4000**” y con los template literal (``) se llama a la variable **`\${PORT}`**.

Para el motor de plantillas se crea la **const ejs** y se requiere el módulo o dependencia “ejs” (**const ejs = require('ejs')**).

Para la gestión del archivo utilizamos el modulo “fs”.

Contenido completo del archivo server/app_server.js:

```
//-----  
//Creamos la const express y la requerimos del módulo o dependencia express  
const express = require('express')  
  
//Para analizar y procesar los datos de solicitudes HTTP,  
//como JSON o datos de formulario  
const bodyParser = require('body-parser')  
  
//motor de plantilla EJS  
const ejs = require('ejs')  
  
//se crea la constante app (puede ser cualquier nombre,  
//pero por convención se llama app de aplicación) y se llama del módulo express.  
const app = express()  
  
//se crea la constante PORT y se indica que será la variable de entorno  
// process.env.PORT la que lo asigne, si no encuentra la variable de entorno  
// entonces se asigna el puerto 4000  
const PORT = process.env.PORT || 4000
```

```
//-----  
//se configura el motor de plantilla ejs  
app.set('views', './views')  
app.set('view engine', 'ejs')  
//-----  
  
//extended: false significa que parsea solo string  
// (no archivos de imagenes por ejemplo)  
app.use(bodyParser.urlencoded({ extended: false })))  
  
//se especifica el subdirectorio donde  
// se encuentran las páginas estáticas  
//app.use(express.static(__dirname + '/public'))  
app.use(express.static( './views'))  
app.use(express.static( './src'))  
app.use(express.static( './css'))  
  
app.get('/', (request, response) => {  
  
  // response.send('Servidor corriendo!')  
  response.render("login");  
  //response.sendFile("./views/index");  
  
})
```

```
//-----  
//se lee el archivo y se carga al index.ejs  
app.get("/listaUsuarios", (request, response) => {  
  
  //se importa el modulo de manejo de archivos fs  
  const fs=require('fs');  
  //se lee el archivo data.json  
  fs.readFile('./server/data.json','utf-8',(err,archivo)=>{  
    if(err){  
      console.log(err);  
    }  
    else{  
      var data=[]  
      if(archivo.length>0)  
        data=Object.values(JSON.parse(archivo));  
  
      //se responde con la pagina de usuarios pasando  
      //toda la data del archivo  
      response.render("indexUser",{data});  
    }  
  })  
});  
//-----
```

```
//-----  
//clase User  
class User{  
  constructor(id,apellido_nombre,correo,telefono,cuenta,password){  
    this.id=id;  
    this.apellido_nombre=apellido_nombre;  
    this.correo=correo;  
    this.telefono=telefono;  
    this.cuenta=cuenta;  
    this.password=password;  
  }  
}  
//-----  
  
//-----  
//para agregar nuevo contacto  
app.post('/newUser', (req, res) => {  
  
  let id="";  
  console.log(req.body.apellido_nombre)  
  if(req.body.id.length>0)//si id no es blanco se modifica  
    id=req.body.id;  
  else{ //si id es blanco se crea un nuevo contacto  
    //genera codigo unico para el contacto  
    id=generarIdUnico();  
  }  
}
```

```
//se capturan los datos del formulario y se
//pasa a unobjeto user
const user=new User( id,req.body.apellido_nombre,
                    req.body.correo, req.body.telefono,
                    req.body.cuenta, req.body.password)
//se importa el modulo de manejo de archivos fs
const fs=require('fs');

//se lee el archivo en donde esta los contactos
fs.readFile('./server/data.json','utf-8',(err,archivo)=>{
  if(err){
    console.log(err);
  }else{
    var usuarios={}
    if(archivo.length>0)
    {
      //el archivo leído se pasa a coleccion de objetos
      usuarios=JSON.parse(archivo)
    }
    usuarios[id]=user;//se adiciona el nuevo contacto
                      //a la coleccion

    //la coleccion de usuarios se pasa a una cadena
    //para poder almacenar en el archivo data
    const usuarios_s=JSON.stringify(usuarios);
```

```
//se reemplaza el contenido del archivo con
//usuarios_s adicionamos el ultimo
fs.writeFile('./server/data.json',usuarios_s,(err)=>{
  if(err){
    throw err;
  }else{
    console.log('SE ADICIONO....')

    //console.log(usuarios);
    //de coleccion a arreglo
    var data=[]
      data=Object.values(usuarios);
    //se carga la pagina de inicio
    //res.redirect("index.ejs",{data}) ;
    res.render("indexUser",{data});
  }
})

}
})

})
//-----
//para eliminar contacto
app.post('/deleteUser', (req, res) => {
  let id=req.body.id;
```

```
const fs=require('fs');//se importa el modulo de manejo de archivos fs

//se lee el archivo en donde esta los contactos
fs.readFile('./server/data.json','utf-8',(err,archivo)=>{
  if(err){
    console.log(err);
  }else{
    var usuarios={}
    if(archivo.length>0){
      usuarios=JSON.parse(archivo)//el archivo leido se pasa a coleccion de objetos
    }

    delete usuarios[id];//se elimina el usuario de la coleccion

    //la coleccion de usuarios se pasa a una cadena
    //para poder almacenar en el archivo data
    const usuarios_s=JSON.stringify(usuarios);

    //se remplaza el cotenido del archivo con usuarios_s adicionamos el ultimo
    fs.writeFile('./server/data.json',usuarios_s,(err)=>{
      if(err){
        throw err;
      }else{
        console.log('SE ELIMINO....')
      }
    })
  }
})
```

```

        var data=[]
        data=Object.values(usuarios);//la coleccion se pasa a arreglo
        res.render("indexUser",{data});//se renderiza la pagina indexUser pasandolo la data

    }
    })

}
})

})

//levanta el servidor con express con el metodo app.listen()
app.listen(PORT, () => {
    console.log(`Url de prueba: http://localhost:${PORT}`)
})

//-----
//funcion que permite generar codigo unico
generarIdUnico = () => {
    return Math.random().toString(30).substring(2);
}
//-----

```

Para poner en marcha el aplicativo, desde el terminal de Visual Studio Code ingresar ...>**node server/app_server.js**

Luego se prueba desde un navegador web: **localhost:4000/**

Tendremos la pantalla mostrada en la FIGURA N° 05-029

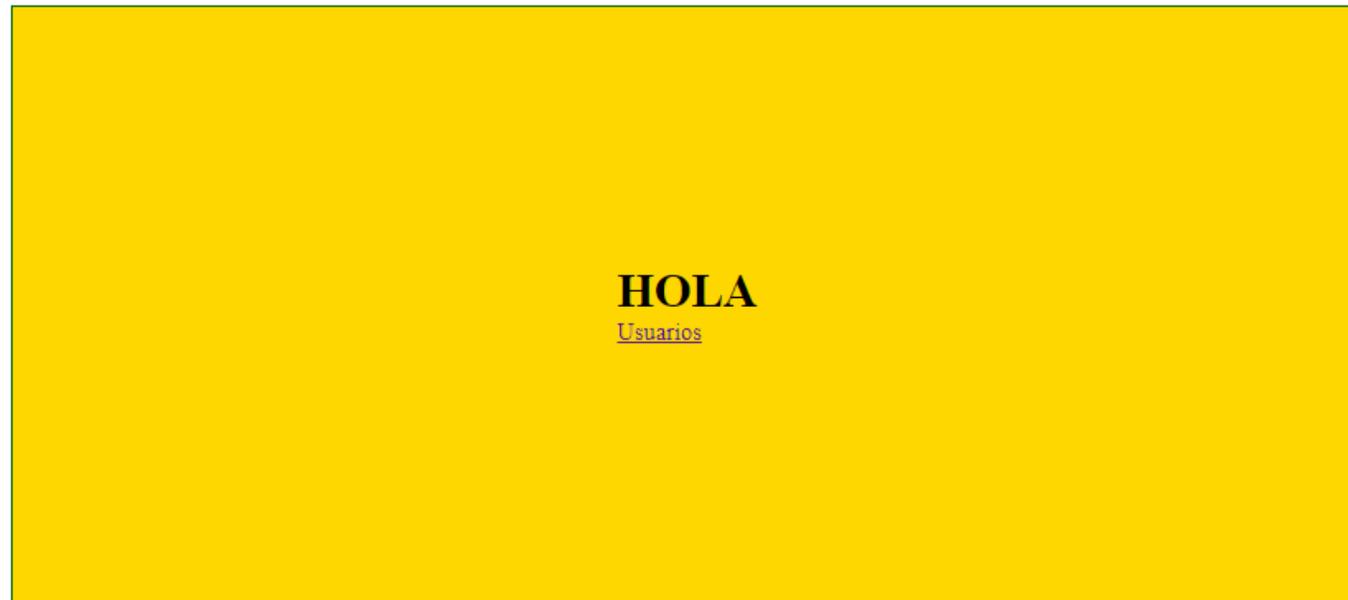
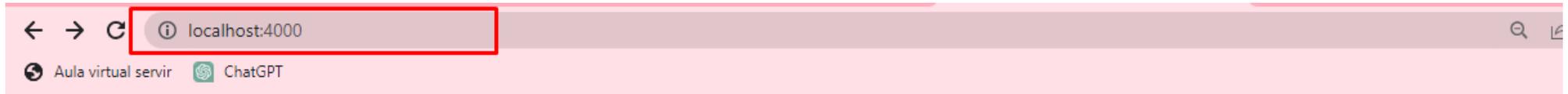


FIGURA N° 05-029: Pantalla de inicio del aplicativo de gestión de usuarios

Click en el link y tendremos la pantalla mostrada en la FIGURA N° 05-030

The screenshot shows a web browser window with the address bar displaying 'localhost:4000/listaUsuarios'. The page content is divided into two main sections:

Formulario de Usuario (Left Panel):

- USUARIOS** (Title)
- Apellidos y Nombres:** Input field with placeholder 'Apellido'.
- Correo:** Input field with placeholder 'Correo'.
- Telefono:** Input field with placeholder 'Telefono'.
- Cuenta:** Input field with placeholder 'cuenta'.
- Passwor:** Input field (partially visible).
- NUEVO** (Button)
- CANCELAR** (Button)

Tabla de Usuarios (Right Panel):

E	M	Apellidos y Nombres	Correo	Telefono	Cuente	Password
		CHUQUIYAURI SALDVAR, ELMER SANTIAGO	elmer@gmail.com	970988938	ELMER	123456
		SANTILLAN CORNEJO, JOSE	jose@gmail.com	65478888	JOSE	6985
		HUAMAN SOBRADO, DIANA	diana@gmil.com	9754851	ELMER	123456
		HUAMAN SOBRADO, NILVER	diana@gmil.com	970988938	ELMER	123546

FIGURA N° 05-030: Pantalla principal del aplicativo de gestión de usuarios

5.4. Conexión a MySQL

Para la conexión con MySQL se instala el modulo “mysql” `>npm install mysql`. Luego del lado del servidor se configura los parámetros de conexión a la base de datos y se realiza la conexión, así como se muestra en el siguiente ejemplo:

```
//se importa el paquete mysql
const mysql=require('mysql');

//se configure parametros de conexion
const conector=mysql.createConnection(
  {
    host:'localhost',
    user:'root',
    password:'',
    database:'bd_usuario',
    // port:3308
  }
)
//se realiza la conexión a la base de datos
const conexion={()=>{
  conector.connect(err=>{
    if(err) throw err
    console.log('conectado')
  })
}}
```

Luego de realizar la conexión se puede realizar operaciones de: INSERT, UPDATE, DELETE O SELECT a una o más tablas de la base de datos. Para ello se utiliza el método: `.query(“cadSQL”, function(err,result,field){ })`, pasando como el primer parámetro cualquier sentencias de SQL visto en el capítulo IV(INSERT, UPDATE, DELETE O SELECT).

Nota. En los parámetros de conexión cuando no se asigna explícitamente un **puerto** para MySQL lo toma por defecto 3306, pero si eso es diferente hay que asignar explícitamente dicho valor al parámetro “**port:valor**”, por ejemplo: **port:3308**

EJEMPLO DE APLICACIÓN N°002

Desarrollar un aplicativo con el mismo diseño del **EJEMPLO DE APLICACIÓN N°001** para gestionar usuarios (crear nuevos, modificar, eliminar, listar). En este caso el almacenamiento de los datos se realiza en una base de datos **bd usuario**.

Para el aplicativo, la primera pantalla deberá ser el logeo de usuario tal como se muestra el diseño en la FIGURA N° 05-031. Cuando un usuario y password es correcto se muestra la pantalla como en la FIGURA N° 05-032.

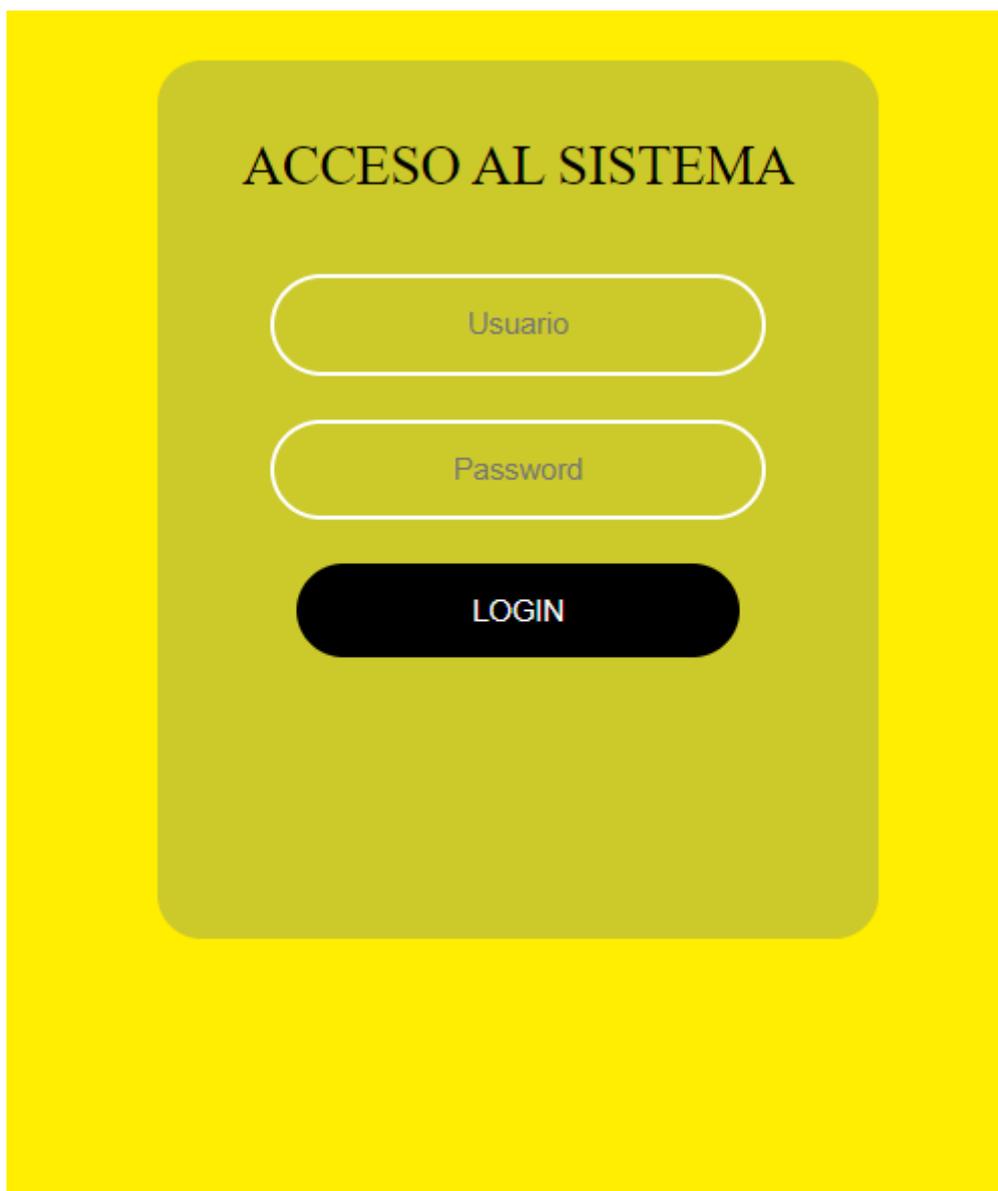


FIGURA N° 05-031: Pantalla de logeo al aplicativo

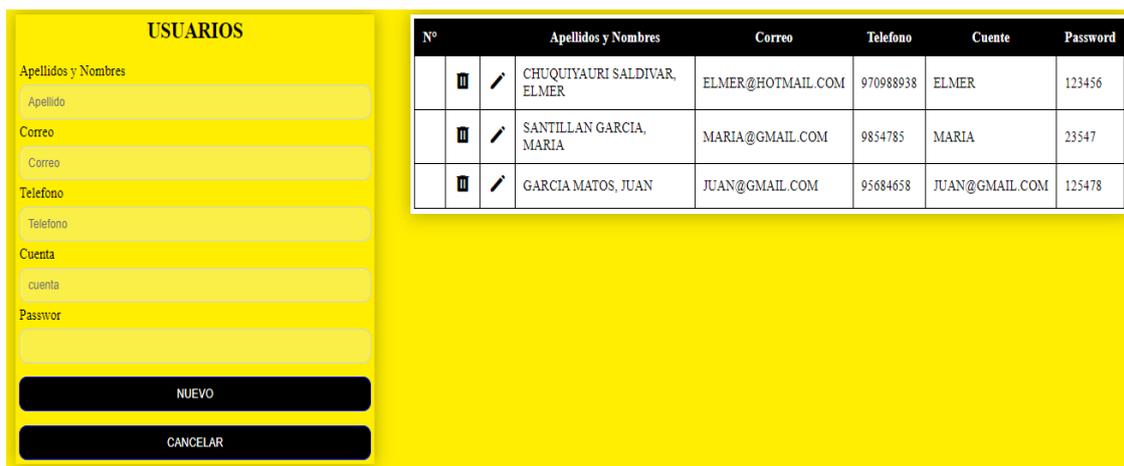


FIGURA N° 05-032: Diseño de formulario para gestionar usuarios con una base de datos

DESARROLLO

Programación del lado del cliente(Frontend)

Para el desarrollo se utiliza los módulos, dependencias, componentes, carpetas y archivos del aplicativo desarrollado en el **EJEMPLO DE APLICACIÓN N°001**. Adicional a ello instalamos el módulo mysql > **npm install mysql**.

La programación de la lógica de interacción del lado del cliente(Frontend) de la FIGURA N° 05-032 es lo mismo que del **EJEMPLO DE APLICACIÓN N°001**.

El diseño y la programación de la lógica de interacción para el logeo tal como se muestra en la FIGURA N° 05-031, se describe a continuación:

Etiquetas del <head>:

Etiqueta	href
<link>	href="style_login.css"
	Estilos de la ventana de logeo

Etiquetas del <body>:

Etiqueta	type	id	class	name	texto
<form>		form-login			
		action="login"		method="post"	
<h1>			form-titulo		Acceso al Sistema
<input>	text	txtUsuario		txtUsuario	Usuario
<input>	password	txtPassword		txtPassword	Password
<button>	submit				LOGIN

Se crea el archivo **login.ejs** dentro de la carpeta **views**, el contenido completo del dicho archivo se muestra a continuación:

login.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <link rel="stylesheet" href="style_login.css">
  <title>Login</title>
</head>
<body>
  <form id="form-login" action="login" method="post">
    <h1 class="form-titulo ">Acceso al Sistema</h1>
    <input type="text" name="txtUsuario" id="txtUsuario" placeholder="Usuario">
    <input type="password" name="txtPassword" id="txtPassword" placeholder="Password">
    <button type="submit">LOGIN</button>
  </form>
</body>
</html>
```

Como se tiene el formulario y el botón **submit** que al hacer click se enviará dos datos del formulario a la ruta “**login**”, que será creado y programado del lado del servidor.

Para aplicar estilo al HTML del **login.ejs** se crea el archivo “**style_login.css**” dentro de la carpeta “**css**”, tal como se muestra a continuación:

```
/*-----*/
/*Estilos a la pagina*/
/*-----*/

/*estilo por etiqueta HTML*/
html, body{
    height: 100%; /*el body y el html ocupan el alto al 100% */
}

/*estilo por etiqueta*/
body{
    margin: 0; /*los espacios fuera del contenedor desaparecen*/
    padding:0; /*los espacios dentro del contenedor desaparecen*/
    background-color: rgb(255, 238, 0); /*color de fondo del body*/

    /*-----*/
    /*centrar vertical y horizontalmente*/
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: center;
    align-items: center;
    /*-----*/
}

/*estilo por id*/
#form-login{
    width: 325px; /*el ancho del formulario a 300px*/
    height: 400px; /*el alto del formulario a 400px*/
    background-color: rgb(204, 201, 43);
    border-radius: 20px;
    margin-top: 30px;
}
}
```

```

/*estilo por clase*/
.form-titulo{
  color:#000;/*color del titulo de formulario negro*/
  text-transform: uppercase;/*El texto transforma a mayuscula*/
  font-weight: 500;/*el peso de la fuente(o que tan negrita) */
  font-size: 25px;/*tamaño de la fuente*/
  background: none; /*sin color de fondo*/
  display: block; /*un control encima del otro*/
  margin-top: 50px;
  margin: 20px auto; /*separados cada uno por 20 pixeles y */
                    /* automatico asi a los lados*/
  padding: 14px 10px;/*separado 14 px y 10px a si a los lados */
                    /* entre controles*/
  text-align: center; /*texto del control alienado al centro*/
}

/*estilos por tipo de cada control dentro del form-login*/
#form-login input[type="text"],
#form-login input[type="password"],
#form-login button[type="submit"]{
  border:0; /*sin borde los controles*/
  background: none; /*sin color de fondo*/
  display: block; /*un control encima del otro*/
  margin: 20px auto; /*separados cada uno por 20 pixeles y */
                    /* automatico asi a los lados*/
  padding: 14px 10px;/*separado 14 px y 10px a si a los lados */
                    /* entre controles*/
  text-align: center; /*texto del control alienado al centro*/
  border: 2px solid #fff; /*borde de grosor 2px estilo solido*/
                    /*de color blanco*/
  width: 200px; /*ancho fijo de 200px*/
  outline: none;/*no se establece ningun perfil */
  color:#fff;/*color del texto*/
  border-radius: 24px; /*borde redondeado*/
  transition: 0.25s;/*se pone una transicion de 0.25 segundos */
}

```

```

/*estilo a una propiedad determinada, en este caso al focus*/
#form-login input[type="text"]:focus,
#form-login input[type="password"]:focus{
    width: 270px; /*cambia de tamaño cuando recibe el enfoque*/
    border-color: black; /*color el borde cuando recibe el enfoque */
}

/*estilo al control button del formulario*/
#form-login button[type="submit"] {
    border:0; /*sin borde*/
    background: black; /*color de fondo negro*/
    cursor: pointer; /*icono del cursor una manito*/
}

```

Programación del lado del servidor(Backend)

Desde el lado del servidor se utiliza todo lo programado en el **EJEMPLO DE APLICACIÓN N°001**, solo que, los datos ya no se almacenarán en un archivo sino en una base de datos, en este caso “**bd_usuario**”, para lo cual instalamos el Mysql a través del XAMPP tal como se detalló en el capítulo IV, luego con el phpMyAdmin se crea la base de datos y la tabla usuario, así como se muestra en la FIGURA N° 05-033.

Script de creación de base de datos:

```
CREATE DATABASE bd_usuario
```

Script de creación de la tabla usuario

```
CREATE TABLE `usuario` (
  `id` varchar(50) NOT NULL,
  `apellido_nombre` varchar(80) NOT NULL,
  `correo` varchar(50) NOT NULL,
  `telefono` varchar(30) NOT NULL,
```

```

`cuenta` varchar(30) NOT NULL,
`password` varchar(30) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;

```

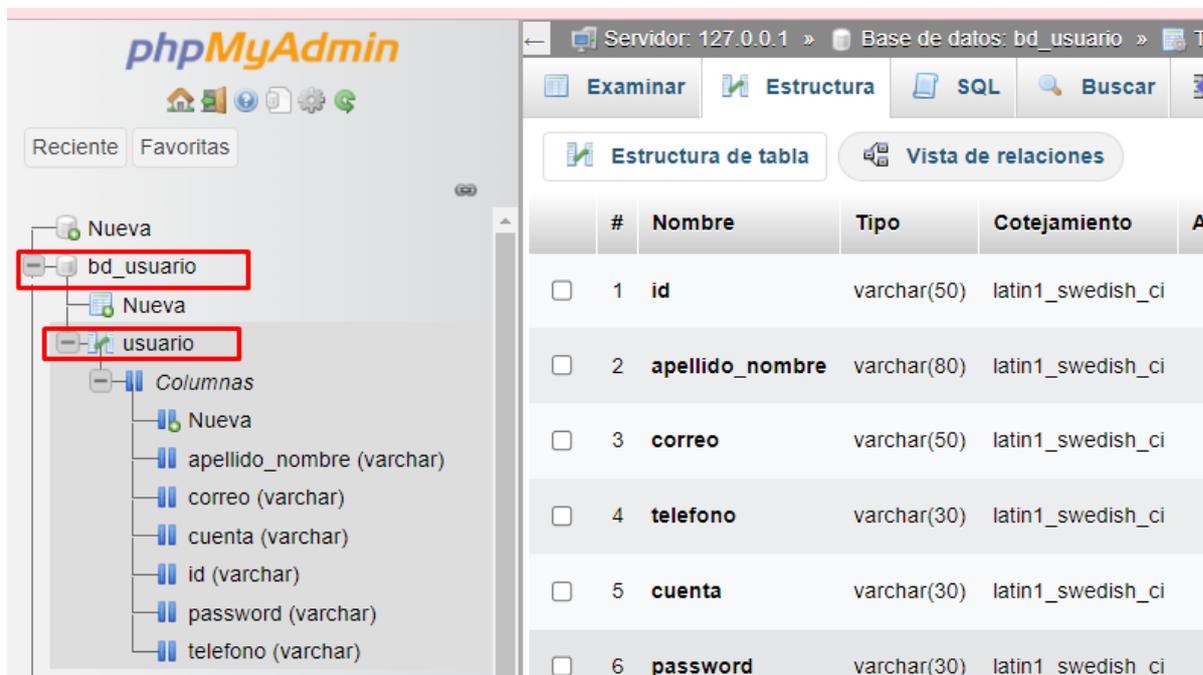


FIGURA N° 05-033: Estructura de la base de datos “*bd_usuario*” y la tabla “*usuario*”

En el archivo **app_server.js** del **EJEMPLO DE APLICACIÓN N°001**, se agrega código y se hace algunos cambios en las mismas rutas para realizar operaciones con la base de datos (INSERT, UPDATE, DELETE O SELECT), así mismo se crea el archivo **db_mysql.js** dentro de la carpeta server, donde se tendrá la programación de la lógica de conexión y operaciones con la base de datos, los cuales se exporta como modulo para utilizarlos desde el archivo **app_serve.js**

Contenido del archivo db_mysql.js**db_mysql.js**

```
//se importa el paquete mysql
const mysql=require('mysql');

//se configure parametros de conexion
const conector=mysql.createConnection(
  {
    host:'localhost',
    user:'root',
    password:'',
    database:'bd_usuario'
  }
)

//se realiza la conexion a la base de datos
const conexion={()=>{
  conector.connect(err=>{
    if(err) throw err
    console.log('conectado')
  })
}}
```

```
const addUsuario=(data)=>{

  const cadSQL=`INSERT INTO usuario(id,apellido_nombre,correo,telefono,cuenta,password)`
    + `VALUES('${data.id}','${data.apellido_nombre}','${data.correo}','`
    + `${data.telefono}','${data.cuenta}','${data.password}')`

  conector.query(cadSQL,function(err,result,filed){
    if(err) throw err;
    console.log(result);
  });
}

const updateUsuario=(data)=>{
  console.log(data)
  const cadSQL=`UPDATE usuario set apellido_nombre='${data.apellido_nombre}',correo=`
    + `${data.correo}',telefono='${data.telefono}',cuenta='${data.cuenta}'`
    + `,password='${data.password}' WHERE id='${data.id}'`

  conector.query(cadSQL,function(err,result,filed){
    if(err) throw err;
    console.log(result);
  });
}
```

```
const deleteUsuario=(id)=>{
  const cadSQL=`DELETE FROM usuario WHERE id=${id}`

  conector.query(cadSQL,function(err,result,filed){
    if(err) throw err;
    console.log(result);
    console.log("se elimino")
  });
};

const getUsuarios=(req,res)=>{
  const cadSQL=`SELECT *FROM usuario`
  conector.query(cadSQL,(err,result,filed)=>{
    if(err){
      res.json(err);
    }
    data=result;
    res.render('indexUser',{data});

  });
};
```

```
//-----  
  
//se exporta las funciones  
module.exports.conexion=conexion;  
module.exports.addUsuario=addUsuario;  
module.exports.updateUsuario=updateUsuario;  
module.exports.getUsuarios=getUsuarios;  
module.exports.deleteUsuario=deleteUsuario;
```

Contenido del archivo `app_server.js`

```
//-----  
//Creamos la const express y la requerimos del módulo o dependencia express  
const express = require('express')  
  
//se crea la constante mysql y se importa el modulo db_mysql.js  
const mysql=require('./db_mysql.js');  
  
//Para analizar y procesar los datos de solicitudes HTTP,  
//como JSON o datos de formulario  
const bodyParser = require('body-parser')  
  
//motor de plantilla EJS  
const ejs = require('ejs')  
  
//se crea la constante app (puede ser cualquier nombre,  
//pero por convención se llama app de aplicación) y se llama del módulo express.  
const app = express()  
  
//se crea la constante PORT y se indica que será la variable de entorno  
// process.env.PORT la que lo asigne, si no encuentra la variable de entorno  
// entonces se asigna el puerto 4000  
const PORT = process.env.PORT||4000
```

```
//-----  
//se configura el motor de plantilla ejs  
app.set('views', './views')  
app.set('view engine', 'ejs')  
//-----  
  
//extended: false significa que parsea solo string  
// (no archivos de imagenes por ejemplo)  
app.use(bodyParser.urlencoded({ extended: false })))  
  
//se especifica el subdirectorio donde  
// se encuentran las páginas estáticas  
app.use(express.static( './views'))  
app.use(express.static( './src'))  
app.use(express.static( './css'))  
  
//creamos la ruta con app.get para localhost:4000/  
app.get('/', (req, res) => {  
  res.render("login");//se renderiza a la pagina login.ejs que esta en views  
})
```

```
//ruta cuando se logea
app.post('/login',(req,res)=>{

  if((req.body.txtUsuario==="admin")&&(req.body.txtPassword==="123"))
  {
    mysql.getUsuarios(req,res);
  }
  else{
    console.log("clave o password incorrecto");
    res.render("login");//se renderiza a la pagina login.ejs que esta en views
  }

})

//-----
//se crea la ruta /listaUsuarios y se renderiza indexUser
app.get("/listaUsuarios", (req, res) => {

  mysql.getUsuarios(req,res);

});
```

```
//-----  
  
//para agregar nuevo contacto  
app.post('/newUser', (req, res) => {  
  
  let id="";  
  const data=req.body;  
  if(data.id.length>0)//si id no es blanco se modifica  
  {  
    mysql.updateUsuario(data);  
    mysql.getUsuarios(req,res);  
  }  
  else{ //si id es blanco se crea un nuevo contacto  
    //genera codigo unico para el contacto  
    id=generarIdUnico();  
    data.id=id;  
    mysql.addUsuario(data);  
    mysql.getUsuarios(req,res);  
  }  
}
```

```
//para eliminar contacto
app.post('/deleteUser', (req, res) => {
  let id=req.body.id;
  mysql.deleteUsuario(id);
  mysql.getUsuarios(req,res);

  })

//levanta el servidor con express con el metodo app.listen()
app.listen(PORT, () => {
  console.log(`Url de prueba: http://localhost:${PORT}`)
})

//-----
//funcion que permite generar codigo unico
generarIdUnico = () => {
  return Math.random().toString(30).substring(2);
}

//-----
```

En la ruta “**login**” se recepciona los datos del **usuario** y su **password** para comparar con “**admin**” y “**123**”, el cual en otra versión del aplicativo se verificará con las cuentas de los usuarios registrado en la base de datos, FIGURA N° 05-034

```

server > JS app_server.js > ...
42 //creamos la ruta con app.get para localhost:4000/
43 app.get('/', (req, res) => {
44     res.render("login");//se renderiza a la pagina login.ejs que esta en views
45 }
46 })
47
48 //ruta cuando se logea
49 app.post('/login', (req, res)=>{
50
51     if((req.body.txtUsuario==="admin")&&(req.body.txtPassword==="123"))
52     {
53         mysql.getUsuarios(req, res);
54     }
55     else{
56         console.log("clave o password incorrecto");
57         res.render("login");//se renderiza a la pagina login.ejs que esta en views
58     }
59 }
60 })
  
```

FIGURA N° 05-034: Archivos adicionados y creación de la ruta de login desde el lado del servidor

Cuando ponemos en marcha el servidor del aplicativo > **node server/app_server.js** y desde la barra de direcciones de un navegador ingresamos: localhost:4000, tendremos la pantalla de la FIGURA N° 05-035



FIGURA N° 05-035: Pantalla de logeo

Se ingresa en usuario: admin y en password: 123, se tendrá la pantalla tal como se muestra en la FIGURA N° 05-036

USUARIOS

Apellidos y Nombres

Correo

Telefono

Cuenta

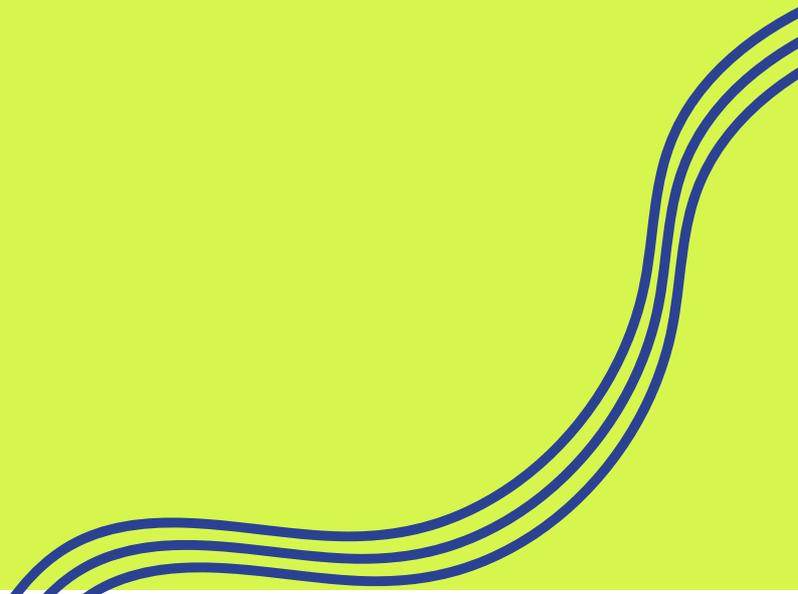
Passwor

E	M	Apellidos y Nombres	Correo	Telefono	Cuenta	Password
🗑	✎	SANTILLAN CORNEJO, JOSE	jose@hotmail.com	970988938	jose	7896547
🗑	✎	HUAMAN SOBRADO, DIANA MARIA	diana@gmil.com	97065485	ELMER	123546
🗑	✎	CHUQUIYAURI SADLVIAR, ELMER	elmer23@gmail.com	98547554	diana	123456

FIGURA N° 05-036: Pantalla principal de gestión de usuario

CAPÍTULO VI

Caso Práctico (Empresa de Turismo PATA AMARILLA)



Empresa de Turismo "Pata Amarilla"

1. Introducción

La creación del sistema digital para una empresa que lo llamamos, la empresa de turismo "Pata Amarilla", para la región de Huánuco, consiste para automatizar a varias áreas y evitar problemas de aglomeración y las colas donde las personas esperan por muchas horas para ser atendidos y muchas veces no reciben una atención adecuada como ellos desean lo que ha generado la necesidad de mejorar sus procesos internos para mantenerse competitiva en el mercado. En respuesta a este desafío, para esta empresa se ha implementado un sistema innovador para la gestión de bienes y suministros. Este sistema está diseñado para abordar las ineficiencias operativas y mejorar la transparencia en la administración de recursos, áreas que son cruciales para el éxito de cualquier empresa moderna.

El objetivo principal de este sistema es aumentar la eficiencia operativa al automatizar procesos porque muchas veces las personas de manera manual no tienen una atención adecuada y las empresas por esta razón no tienen muchos usuarios. La gestión manual de bienes y suministros puede ser propensa a errores, demoras y una falta de control riguroso sobre los inventarios. Al implementar un sistema automatizado, "Pata Amarilla" puede reducir estos problemas, asegurando que los recursos se gestionen de manera más precisa y efectiva. Además, la transparencia en la administración de recursos permite a la empresa tomar decisiones informadas basadas en datos actualizados y precisos.

Por otra parte, también se detallarán las funcionalidades del sistema implementado, proporcionando una visión clara de cómo cada área de la empresa interactúa con el sistema para asegurar un flujo de trabajo óptimo. Se describirán las áreas involucradas, tales como Usuario, Administración, Proveedor, Logística y Almacén, y cómo estas áreas están interconectadas para garantizar que los procesos se realicen de manera coordinada y eficiente. Esta interacción entre las áreas es esencial para el funcionamiento integral del sistema y para la consecución de los objetivos empresariales.

La implementación de este sistema busca no solo optimizar los procesos internos sino también garantizar una experiencia satisfactoria para todos los usuarios involucrados. Los usuarios del sistema incluyen tanto a los empleados de la empresa que gestionan los bienes y suministros como a los proveedores que interactúan con el sistema para cumplir con las órdenes de compra.

2. Descripción General del Sistema

El sistema "Pata Amarilla" está diseñado para optimizar la gestión de bienes y suministros dentro de la empresa, abordando varias áreas clave que son esenciales para el funcionamiento eficiente y coordinado de la organización. Este sistema es una solución integral que no solo facilita la administración de inventarios, sino que también mejora la comunicación y la colaboración entre los diferentes departamentos o áreas. A continuación, se describen en detalle las áreas clave y las funcionalidades específicas del sistema.

El sistema integra seis áreas principales: Usuario, jefe, Administración, Proveedor, Logística y Almacén. Cada una de estas áreas tiene un conjunto de funcionalidades diseñadas para manejar sus responsabilidades específicas de manera eficiente. La integración de estas áreas en un único sistema centralizado permite una coordinación y un flujo de trabajo sin interrupciones, lo cual es fundamental para la gestión efectiva de bienes y suministros.

Área de Usuario

El área de Usuario está diseñada para que los empleados de la empresa puedan interactuar con el sistema de manera sencilla y efectiva. Los usuarios pueden realizar acciones como ver ítems disponibles, generar solicitudes de suministros, rastrear el estado de sus solicitudes para que los usuarios puedan ver en qué área se quedó su solicitud y realizar sus reclamos o quejas directamente a esa área, consultar el historial de solicitudes y actualizar su perfil personal. Estas funcionalidades aseguran que los usuarios tengan acceso a la información que necesitan y puedan gestionar sus necesidades de suministros de manera autónoma y eficiente.

Área del jefe

En esta parte lo que el jefe hace es ver todas las listas de las solicitudes que los usuarios están enviando y solamente lo que el jefe hace es aceptar todos los ítems que los usuarios solicitan y cuando acepta la lista de ítems solicitados por los usuarios automáticamente se envía al área de administración. También tiene un buscador si desea buscar por persona para que acepte solamente la solicitud de esa persona buscada.

Área de Administración

El área de Administración es crucial para la supervisión y la gestión de todas las actividades relacionadas con bienes y suministros. Los administradores pueden aprobar usuarios, monitorear solicitudes, gestionar inventarios, generar reportes y ajustar configuraciones del sistema, además de eso ellos son los encargados de aceptar la inscripción de los trabajadores para que formen parte de la empresa y solamente los que se inscriben por usuario podrán ingresar directamente al programa y los demás tienen que ser aceptados por administración. Esta área garantiza que las políticas y procedimientos de la empresa se cumplan, y que todas las operaciones se realicen de manera transparente y conforme a las normas establecidas.

Área de Proveedor

El área de Proveedor facilita la interacción entre la empresa y sus proveedores, pero más está relacionado con el área de logística porque tienen una interacción mutua estas dos áreas ya que logística realiza la orden de compra de todos los bienes y suministros que desea y el proveedor los acepta y los bienes y suministros son enviados al área de almacén según la lista solicitada. Los proveedores pueden gestionar ofertas, aceptar órdenes de compra, registrar envíos, manejar facturación, consultar el historial de transacciones y actualizar su perfil. Esta funcionalidad asegura una comunicación clara y efectiva con los proveedores, permitiendo una gestión eficiente de las órdenes de compra y el cumplimiento oportuno de los pedidos.

Los proveedores son los encargados de registrar o subir todos sus productos a la página para que sean vistos por el área de logística y también ellos se encargaran de administrar sus propios productos, cambiar el stock y los precios.

Área de Logística

El área de Logística se encarga de la gestión y coordinación de todas las actividades relacionadas con el suministro de bienes. Los responsables de logística pueden ver solicitudes, generar órdenes de compra, hacer seguimiento de las órdenes, gestionar proveedores y controlar inventarios. Esta área es esencial para asegurar que los bienes se adquieran y distribuyan de manera eficiente, minimizando retrasos y asegurando que los recursos estén disponibles cuando se necesiten.

Área de Almacén

El área de Almacén maneja la recepción, almacenamiento y distribución de bienes. El personal de almacén puede registrar la recepción de bienes, gestionar inventarios, distribuir bienes, consultar el historial de inventario, generar reportes y mantenimiento de los bienes. Estas funcionalidades aseguran que los bienes se manejen de manera organizada y que se mantenga un control riguroso sobre el inventario, reduciendo pérdidas y asegurando la disponibilidad de bienes y suministros. En esta parte también a través de una peca se entregará los ítems solicitados por las diferentes áreas de la empresa.

☰

AREA ALMACEN

🔔

ALEXANDER EYNOR
HERRERA ESTEBAN

Perfil

Recepción de Bienes

Gestión de Inventario

Distribución de Bienes

Reportes de Almacén

Perfil

Nombre: ALEXANDER EYNOR **Apellido:** HERRERA ESTEBAN

Email: alexander@gmail.com

Área: Almacén **Cargo:** supervisor

Actualizar Perfil

Nombre Apellido

Email

Área Cargo

Nueva Contraseña

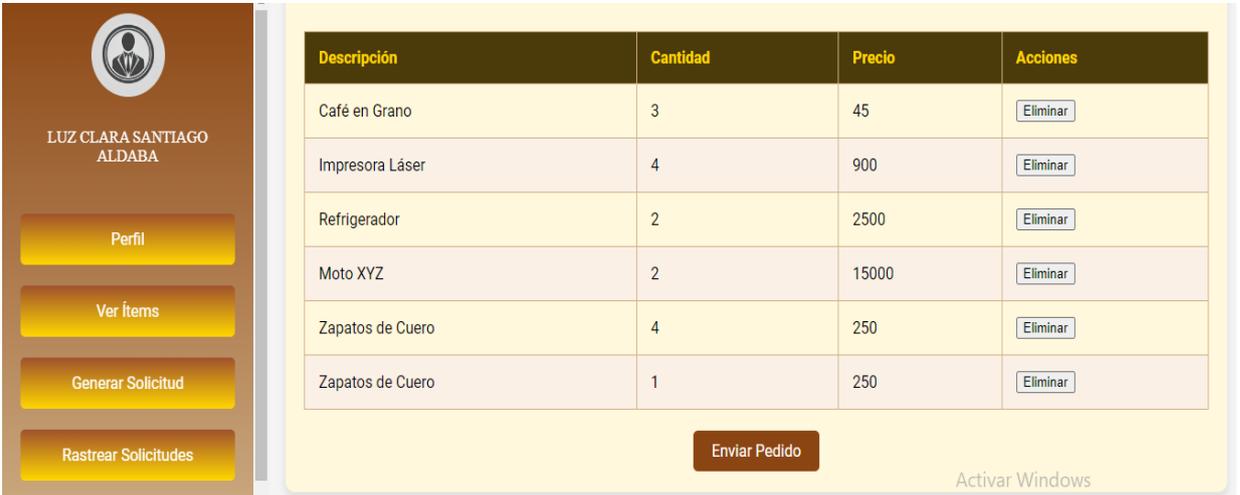
Confirmar Nueva Contraseña

Activar Windows
Ve a Configuración para activar Windows.

3. Relación de envío de solicitud y Flujo del Proceso

Solicitud de Ítems por el Usuario

Generación de Solicitud: Los usuarios acceden al sistema y visualizan los ítems disponibles. Generan una solicitud especificando los ítems necesarios y cantidades. La solicitud se envía al jefe del área correspondiente para su aprobación.



Descripción	Cantidad	Precio	Acciones
Café en Grano	3	45	Eliminar
Impresora Láser	4	900	Eliminar
Refrigerador	2	2500	Eliminar
Moto XYZ	2	15000	Eliminar
Zapatos de Cuero	4	250	Eliminar
Zapatos de Cuero	1	250	Eliminar

Aprobación de Solicitudes por el jefe: El jefe de área revisa y aprueba las solicitudes de los usuarios. Las solicitudes aprobadas se envían al área de Administración para una segunda aprobación.



ID	Descripción	Cantidad	Precio	DNI Usuario	Numero de Boleto
184	Café en Grano	3	45	72137461	12
185	Impresora Láser	4	900	72137461	12
186	Refrigerador	2	2500	72137461	12
187	Moto XYZ	2	15000	72137461	12
188	Zapatos de Cuero	4	250	72137461	12
189	Zapatos de Cuero	1	250	72137461	12

Aprobación de por Administración: Las solicitudes luego de ser aprobados por el jefe se envían al área de Administración para una segunda aprobación. Administración aprueba la solicitud y la envía al área de Logística.

Aprobar Usuarios

▼

▼

▼

▼

Descripción Item	Cantidad	Precio	DNI Registrador	Número Pedido	DNI Jefe
Café en Grano	3	45.00	72137461	12	73268144
Impresora Láser	4	900.00	72137461	12	73268144
Refrigerador	2	2500.00	72137461	12	73268144
Moto XYZ	2	15000.00	72137461	12	73268144
Zapatos de Cuero	4	250.00	72137461	12	73268144
Zapatos de Cuero	1	250.00	72137461	12	73268144

Gestión de Solicitudes en el Área de Logística: Logística recibe las solicitudes aprobadas y verifica el stock disponible. Si el stock es insuficiente, Logística consolida las solicitudes y genera órdenes de compra. Realiza cotizaciones con varios proveedores y selecciona la mejor oferta. Genera las órdenes de compra y las envía para los proveedores y los proveedores recibirán la lista de bienes y suministros y si cuentan con la necesario ellos aceptan esa orden de compra y llevan sus productos al almacén.

📄 Solicitudes

▼

▼

ID	DNI Registrador	Número Pedido	Descripción Item	Cantidad	Precio	DNI Administrador	Fecha Creación
235	72137461	12	Café en Grano	3	45.00	22660066	21/7/2024, 7:13:16 a. m.
236	72137461	12	Impresora Láser	4	900.00	22660066	21/7/2024, 7:13:16 a. m.
237	72137461	12	Refrigerador	2	2500.00	22660066	21/7/2024, 7:13:16 a. m.
238	72137461	12	Moto XYZ	2	15000.00	22660066	21/7/2024, 7:13:16 a. m.
239	72137461	12	Zapatos de Cuero	4	250.00	22660066	21/7/2024, 7:13:16 a. m.
240	72137461	12	Zapatos de Cuero	1	250.00	22660066	21/7/2024, 7:13:16 a. m.

El área de logística también genera las ordenes de comprar para que almacén contenga todos los productos necesarios y para ello hace sus pedidos de los productos que necesita y luego son enviados al proveedor para su aprobación

la siguiente tabla muestra todos los productos que son cargados por los proveedores:

AREA LOGISTICA							
ID	Nombre	Categoría	Precio	Cantidad	Unidad de Medida	DNI Proveedor	Acción
1	Moto XYZ	Vehículos	15000.00	95	Unidades	71885493	Seleccionar
2	Laptop ABC	Electrónicos	3500.00	35	Unidades	71885493	Seleccionar
3	Camisa Formal	Ropa	120.00	187	Unidades	71885493	Seleccionar
4	Manzana	Alimentos	3.50	478	Kilogramos	71885493	Seleccionar
5	Escritorio	Muebles	450.00	0	Unidades	71885493	Seleccionar
6	Refrigerador	Electrodomésticos	2500.00	3	Unidades	71885493	Seleccionar
7	TV LED 50	Electrónicos	2200.00	20	Unidades	71885493	Seleccionar
8	Silla de Oficina	Muebles	320.00	52	Unidades	71885493	Seleccionar

Luego se selecciona los productos y se hace una lista para pedir a los proveedores los productos que están faltando en almacén. La lista de productos que esta seleccionando para la orden de compras está en la siguiente tabla:

Lista de Bienes y Suministros Para Solicitar

ID	Nombre	Precio	Cantidad	DNI Proveedor	Acción
3	Camisa Formal	120	7	71885493	Eliminar
8	Silla de Oficina	320	7	71885493	Eliminar
9	Zapatos de Cuero	250	7	71885493	Eliminar
13	Juguete Educativo	45	5	71885493	Eliminar

Cerrar

Si estás de acuerdo con la lista de los bienes y suministros que estas solicitando entonces lo que sigue es solamente completar tu ubicación y presionar un botón para que se envíe a los proveedores tus bienes y suministros seleccionados. En la siguiente parte te muestra ese proceso:

Órdenes de Compra

REGIÓN:

PROVINCIA:

DISTRITO:

DIRECCIÓN:

Activar Windows
 Ve a Configuración para activar Windows.

Aceptación del orden de compra por parte del proveedor: los proveedores pueden visualizar toda la lista de los productos que son solicitados por parte de logística y luego ellos aceptan todos los pedidos, pero antes el programa verifica si tienen todos los bienes y suministros cargados en el sistema y si no lo tienen no se va aceptar y para ello ellos tienen que cargar sus productos contantemente para que en el programa su stock no este vacío.

AREA DEL PROVEEDOR

orden de compra pendiente

ID	Nombre	Precio	Cantidad	Acciones
3	Camisa Formal	120	7	<input type="button" value="Aceptar"/>
8	Silla de Oficina	320	7	<input type="button" value="Aceptar"/>
9	Zapatos de Cuero	250	7	<input type="button" value="Aceptar"/>
13	Juguete Educativo	45	5	<input type="button" value="Aceptar"/>

Activar Windows

Recepción de Bienes en el Área de Almacén: Almacén recibe los bienes entregados por los proveedores. Verifica los bienes contra las órdenes de compra y registra la entrada en el sistema. Emite una guía de remisión y distribuye los bienes solicitados mediante una pecosa.

AREA ALMACEN



LISTA DE ITEMS RECIBIDOS DEL PROVEEDOR

Seleccione una Orden: 19 ▾ Confirmar los Productos

ID	Nombre	Precio	Cantidad
3	Camisa Formal	120	7
8	Silla de Oficina	320	7
9	Zapatos de Cuero	250	7
13	Juguete Educativo	45	5

Gestión de Bienes en el Área de Patrimonio: Antes de distribuir bienes, Patrimonio asigna códigos patrimoniales y genera etiquetas para los bienes. Administra y controla la ubicación de los bienes dentro de la empresa.

Distribución Final y Notificaciones: Almacén distribuye los bienes a los usuarios solicitantes y actualiza el sistema con la entrega. Los usuarios reciben notificaciones sobre el estado y la entrega de sus solicitudes.

Flujo del Proceso de la empresa

- Solicitud de Ítems: Usuario → jefe de Área → Administración → Logística
- Aprobación de Solicitudes: jefe de Área → Administración → Logística
- Generación de Órdenes de Compra: Logística → Proveedores
- Recepción y Distribución de Bienes: Proveedores → Almacén → Patrimonio o pecosa → Usuarios
- Gestión de Bienes: Patrimonio → Almacén → Usuarios
- Seguimiento de las solicitudes: → jefe de Área → Administración → Logística
- Apartado de envío de mensaje: todas las áreas se pueden enviar mensajes sobre las inconveniencias que tiene para que se solucionen

4. Descripción del Flujo

El usuario genera una solicitud de ítems a través del sistema

El proceso comienza cuando un usuario, generalmente un empleado de la empresa, identifica la necesidad de ciertos ítems o suministros para realizar sus tareas diarias. Utilizando el sistema, el usuario genera una solicitud donde detalla los ítems necesarios, la cantidad y las fechas en que se requieren. Esta solicitud incluye información crucial como la descripción de los ítems, la cantidad solicitada, el área de destino y la prioridad de la solicitud. Este paso es fundamental porque establece la base para todo el flujo de trabajo posterior, asegurando que las necesidades de los usuarios se registren de manera precisa y sistemática. La interfaz amigable del sistema facilita la creación de solicitudes, permitiendo a los usuarios especificar sus requerimientos con claridad y sin errores.

El jefe de área revisa y aprueba la solicitud

Una vez que la solicitud ha sido creada, se envía automáticamente al jefe de área correspondiente para su revisión y aprobación. El jefe de área tiene la responsabilidad de verificar que la solicitud sea válida y necesaria para el funcionamiento del departamento. Esta revisión incluye la evaluación de la justificación de la solicitud, la cantidad de ítems solicitados y la disponibilidad de presupuesto para cubrir la compra. Si la solicitud cumple con todos los criterios, el jefe de área la aprueba y la envía a la siguiente etapa. Esta capa de aprobación inicial es crucial para asegurar que solo se procesen solicitudes legítimas y necesarias, evitando el uso indebido de recursos y garantizando que los suministros solicitados sean realmente necesarios para las operaciones del área.

El área de Administración verifica y aprueba la solicitud

Después de la aprobación del jefe de área, la solicitud se envía al área de Administración para una segunda revisión. La administración verifica la precisión y la necesidad de los recursos solicitados, asegurándose de que la solicitud cumpla con las políticas y presupuestos de la empresa. Esta capa adicional de aprobación garantiza un control riguroso sobre los recursos, evitando duplicidades y asegurando una distribución eficiente de los mismos. La administración también verifica que la solicitud esté alineada con las metas y prioridades estratégicas de la empresa. Si la solicitud pasa esta revisión, se aprueba y se envía al área de Logística para su procesamiento. Esta segunda capa de

aprobación es fundamental para mantener la integridad y la eficiencia del proceso de gestión de suministros.

El área de Logística consolida las solicitudes, verifica el stock y realiza órdenes de compra

Con la solicitud aprobada por la administración, el área de Logística asume la responsabilidad de gestionar la adquisición de los ítems solicitados. Logística revisa el inventario actual para determinar si los ítems están disponibles en stock. Si los ítems no están disponibles, Logística consolida todas las solicitudes similares y procede a generar órdenes de compra para adquirir los ítems necesarios. Este proceso incluye la solicitud de cotizaciones a varios proveedores, la evaluación de las ofertas recibidas y la selección del proveedor que ofrezca la mejor combinación de precio y calidad. Una vez seleccionado el proveedor, se genera una orden de compra detallada y se envía al proveedor para su cumplimiento. Este paso es crucial para asegurar que los suministros se adquieran de manera eficiente y económica, manteniendo un control riguroso sobre el inventario y los costos.

El proveedor entrega los bienes al área de Almacén

Una vez que el proveedor ha recibido y procesado la orden de compra, prepara los bienes solicitados y los envía al área de Almacén de la empresa. Al llegar los bienes, el personal de almacén realiza una verificación detallada para asegurarse de que los ítems entregados coincidan con las especificaciones y cantidades indicadas en la orden de compra. Esta verificación incluye la inspección de la calidad de los bienes y la revisión de la documentación adjunta, como guías de remisión y facturas. Este paso es fundamental para asegurar que los bienes recibidos cumplen con los estándares de la empresa y están en condiciones óptimas para su uso. Cualquier discrepancia o problema identificado durante esta verificación se comunica inmediatamente al proveedor para su resolución.

El área de Almacén recibe y verifica los bienes, los registra y los distribuye a los usuarios

Después de verificar que los bienes recibidos cumplen con las especificaciones, el área de Almacén procede a registrar la entrada de los ítems en el sistema de inventario. Este registro incluye detalles como la cantidad recibida, la fecha de recepción y el estado de los ítems. Una vez registrados, los bienes se almacenan de manera organizada, asegurando que estén fácilmente accesibles para su distribución futura. El personal de almacén luego distribuye los bienes a las áreas y usuarios que generaron las solicitudes, siguiendo un proceso organizado que asegura que cada ítem llegue a su destino

correspondiente de manera eficiente. Este paso asegura que los recursos necesarios estén disponibles para los usuarios a tiempo, minimizando interrupciones en las operaciones diarias de la empresa.

El área de Patrimonio asigna códigos patrimoniales a los bienes y los etiqueta antes de la distribución

Antes de que los bienes se distribuyan a los usuarios finales, el área de Patrimonio asigna códigos patrimoniales únicos a cada ítem y los etiqueta en consecuencia. Esta codificación es esencial para mantener un seguimiento preciso y detallado de todos los bienes de la empresa. Los códigos patrimoniales permiten rastrear la ubicación y el estado de cada ítem, facilitando la gestión de inventarios y la planificación de futuras adquisiciones. La etiqueta de cada ítem incluye información relevante como el código patrimonial, la descripción del bien y su ubicación dentro de la empresa. Este proceso de etiquetado y codificación asegura que todos los bienes se gestionen de manera organizada y que cualquier movimiento o uso de los ítems pueda ser fácilmente monitoreado.

Los usuarios reciben los bienes y notificaciones sobre el estado de sus solicitudes

Finalmente, una vez que los bienes han sido etiquetados y registrados, se distribuyen a los usuarios que realizaron las solicitudes originales. El sistema actualiza el estado de las solicitudes para reflejar la entrega de los ítems, y los usuarios reciben notificaciones sobre la disponibilidad de los bienes solicitados. Estas notificaciones pueden incluir detalles como la fecha de entrega, la ubicación de recogida y cualquier instrucción adicional relevante. La transparencia y la comunicación continua son clave en este paso para asegurar que los usuarios estén informados sobre el estado de sus solicitudes y puedan planificar en consecuencia. Esta comunicación efectiva ayuda a mantener la satisfacción del usuario y asegura que los recursos estén disponibles cuando y donde se necesiten.

5. Estructura del Código y la Función del Programa

Estructura del código

```

├── config
├── controllers
│   └── RECE_funciones.js
├── models
├── public
│   ├── css
│   │   ├── administracion.css
│   │   ├── almacen.css
│   │   ├── AREA_jefe.css
│   │   ├── logistica.css
│   │   ├── logueo.css
│   │   ├── proveedor.css
│   │   ├── registro.css
│   │   └── usuario_pantalla.css
│   ├── imagenes
│   └── js
│       ├── ACE_funciones.js
│       ├── administracion_perfil.js
│       ├── administracion.js
│       ├── com_orden_frontend.js
│       ├── entrada.js
│       ├── frontend.js
│       ├── jefe.js
│       ├── logi_logistica.js
│       ├── orden_compra.js
│       ├── registro.js
│       └── usuario.js
├── routes
│   ├── ACE_funciones.js
│   ├── administracion_perfil.js
│   ├── aprobacion_solicitud.js
│   ├── com_orden_rutas.js
│   └── consultaDni.js

```

```

|   |— items.js
|   |— jefe_area.js
|   |— logistica.js
|   |— pedidos.js
|   |— RECE_ordenes.js
|   |— RUTA_JEFE_jefe_manejo.js
|   |— usuarios.js
|   └─ views
|       |— administracion.ejs
|       |— almacen.ejs
|       |— jefe.ejs
|       |— logistica.ejs
|       |— logueo.ejs
|       |— proveedor.ejs
|       |— registro.ejs
|       └─ usuario.ejs
|   └─ app.js
|   └─ package-lock.json
└─ package.json

```

Explicación de la Estructura del Código

Config

La carpeta config es esencial para centralizar todas las configuraciones de tu aplicación. Aquí se almacenan archivos con configuraciones globales, como la configuración de la base de datos. Esto permite que ajustes estos valores sin modificar directamente el código fuente, facilitando la gestión y el despliegue de la aplicación en diferentes entornos.

Controllers

En la carpeta controllers se encuentra la lógica de negocio de la aplicación. Los controladores actúan como intermediarios entre las peticiones del cliente (a través de las rutas) y los modelos de datos. Se encargan de procesar las solicitudes entrantes, aplicar la lógica necesaria y devolver las respuestas adecuadas. Por ejemplo, un controlador puede manejar la lógica para crear, leer, actualizar o eliminar datos en la base de datos.

Models

La carpeta models contiene los modelos de datos que representan las estructuras de tu base de datos. Los modelos definen cómo se estructura la información y cómo se interactúa con la base de datos. Aquí se establecen los esquemas de los datos y los métodos necesarios para acceder y manipular dichos datos, asegurando que todas las operaciones con la base de datos sigan las mismas reglas.

Public

La carpeta public es donde se almacenan todos los recursos estáticos accesibles públicamente desde el cliente, como archivos CSS, JavaScript e imágenes.

CSS: La subcarpeta css contiene las hojas de estilo que definen la apariencia visual de la aplicación. Cada archivo CSS puede estar dedicado a una sección específica de la aplicación, permitiendo una organización clara y modular. En esta parte se encuentra el diseño que va tener cada apartado de las pantallas que se van a mostrar.

Imágenes: La subcarpeta imagenes almacena todas las imágenes utilizadas en el sitio web, como logos, iconos y gráficos.

JS: La subcarpeta js incluye archivos JavaScript que contienen la lógica o las funciones que va hacer el programa, encargándose de la interactividad y el comportamiento dinámico de la aplicación web.

Routes

En la carpeta routes se definen todas las rutas de la aplicación. Cada archivo de esta carpeta corresponde a un conjunto de rutas relacionadas con diferentes funcionalidades de la aplicación. Las rutas gestionan las solicitudes HTTP, determinan qué controlador se debe llamar y qué acción realizar en función de la URL y el método de solicitud.

Views

La carpeta views almacena las vistas de la aplicación. Estas vistas son plantillas HTML. Utilizan motores de plantillas como EJS para generar contenido HTML dinámico basado en los datos proporcionados por los controladores. Cada archivo en esta carpeta representa una página o una sección de la aplicación, proporcionando una estructura clara y organizada para el contenido visual.

app.js

El archivo app.js es el punto de entrada principal de la aplicación Node.js. Aquí se configura y se inicia la aplicación, incluyendo la configuración del servidor, la conexión

a la base de datos y la definición de rutas principales. Este archivo actúa como el núcleo de la aplicación, inicializando todos los componentes necesarios para que la aplicación funcione.

package-lock.json y package.json

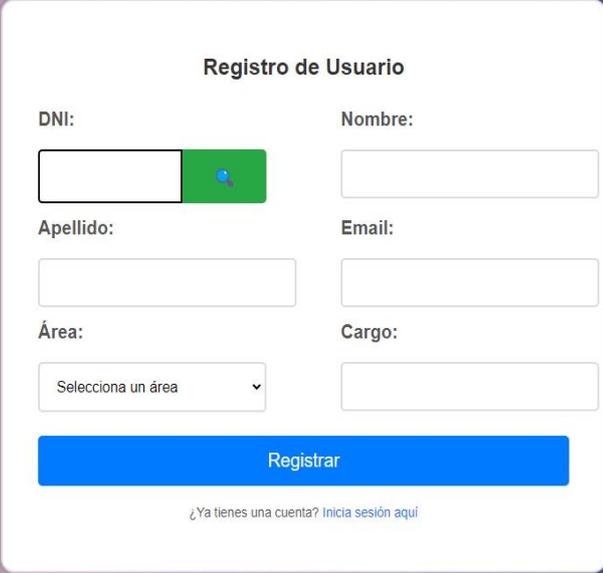
Estos archivos son cruciales para la gestión de dependencias del proyecto.

package.json: Define las dependencias que necesita tu aplicación para funcionar, especificando las versiones y los scripts de ejecución.

package-lock.json: Asegura que se instalen las mismas versiones de los paquetes en todos los entornos donde se despliegue la aplicación, garantizando consistencia y estabilidad en las dependencias.

6. Funcionalidad de toda la página y el código

➤ **Primero para usar el programa tienes que registrarte**



Registro de Usuario

DNI:	<input type="text"/>	Nombre:	<input type="text"/>
Apellido:	<input type="text"/>	Email:	<input type="text"/>
Área:	<input type="text" value="Selecciona un área"/>	Cargo:	<input type="text"/>

[¿Ya tienes una cuenta? Inicia sesión aquí](#)

Activar Windows
Ve a Configuración para activar Windows.

El bloque `<head>` del documento HTML contiene meta-información sobre la página, incluyendo el título y enlaces a hojas de estilo CSS.

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registro</title>
  <link rel="stylesheet" href="/public/css/registro.css">
</head>
```

Cuerpo de la Página (`<body>`)

El bloque `<body>` contiene el contenido visible de la página web. Dentro de este, el `<div class="container">` actúa como el contenedor principal que encapsula todo el contenido de la página. Este contenedor incluye un encabezado principal `<h1>Registro de Usuario</h1>` que identifica la página como el formulario de registro de usuarios para que todos los que desean ingresar a la página puedan rellenar sus datos y poder entrar a la página.

Formulario de Registro (`<form id="registerForm">`)

El formulario permite a los usuarios ingresar sus datos personales para registrarse. Cada campo de entrada se agrupa en `<div class="form-row">` para facilitar el diseño y la estructura. Las filas del formulario ayudan a alinear los campos de entrada uno al lado del otro, mejorando la organización visual. Dentro de cada fila, los campos de entrada están agrupados en `<div class="form-group">`, que contiene una etiqueta (`<label>`) y el campo de entrada correspondiente (`<input>`).

Por ejemplo, el primer grupo de formulario contiene una etiqueta `<label for="dni">DNI:</label>` para el campo de entrada del DNI, junto con un `<div class="input-group">` que agrupa el campo de entrada del DNI y el botón de búsqueda. El campo de entrada del DNI se define como `<input type="text" id="dni" name="dni" required style="flex: 2;">` y el botón de búsqueda se define como `<button type="button" id="fetchDniData" class="small-button" style="flex: 1;">buscar</button>`, ambos diseñados para ajustarse proporcionalmente dentro del contenedor padre.

Otros campos de entrada incluyen el nombre (`<input type="text" id="nombre" name="nombre" readonly required>`), apellido (`<input type="text" id="apellido" name="apellido" readonly required>`), y correo electrónico (`<input type="email" id="email" name="email" required>`). Además, se incluye un menú desplegable para seleccionar el área (`<select id="area" name="area" required>`) con varias opciones como "Usuarios", "Administración",

"Almacén", "Proveedores", y "Logística". También hay un campo de entrada para el cargo (`<input type="text" id="cargo" name="cargo" required>`).

El botón de envío del formulario se define como `<button type="submit">Registrar</button>`, permitiendo a los usuarios enviar el formulario. Además, hay un `<div id="successMessage" class="success"></div>` que se utiliza para mostrar mensajes de éxito después de enviar el formulario, proporcionando retroalimentación visual al usuario. Finalmente, un enlace de inicio de sesión dentro de `<div class="login-link">` permite a los usuarios existentes redirigirse a la página de inicio de sesión mediante `<p>¿Ya tienes una cuenta? Inicia sesión aquí</p>`.

La función que permite capturar el DNI para consultar en la reniec

```
document.getElementById('fetchDniData').addEventListener('click', async () => {
  const dni = document.getElementById('dni').value;
  if (dni) {
    try {
      const response = await fetch('/api/consulta-dni', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ dni })
      });
      const data = await response.json();
      if (data.error) {
        alert(data.error);
      } else {
        document.getElementById('nombre').value = data.nombres;
        document.getElementById('apellido').value = `${data.apellidoPaterno} ${data.apellidoMaterno}`;
      }
    } catch (error) {
      alert('Error al consultar el DNI');
    }
  } else {
    alert('Por favor, ingresa un DNI');
  }
});
```

Primero, se añade un **listener** al botón con el **ID fetchDniData** para manejar el evento de clic. Cuando el usuario hace clic en este botón, se ejecuta una función asíncronica. Esta función obtiene el valor del campo de entrada con el **ID dni**. Si el campo **dni** no está vacío, la función procede a realizar una solicitud a la **API**. Esta solicitud se realiza mediante un método **POST** a la **ruta /api/consulta-dni**, incluyendo el **DNI** en el cuerpo de la solicitud. La función espera la respuesta de la **API**, y luego intenta convertir esta respuesta a formato **JSON**. Si la API devuelve un error, se muestra una alerta con el mensaje de error. En caso de que la consulta sea exitosa, los campos de entrada para nombre y apellido se completan automáticamente con los datos recibidos de la **API**. Si ocurre algún error durante la solicitud, como problemas de red, se muestra una alerta informando al usuario del error ocurrido.

Función para capturar todos los valores y registrar en la base de datos

```
document.getElementById('registerForm').addEventListener('submit', async (event) => {
  event.preventDefault();

  const dni = document.getElementById('dni').value;
  const nombre = document.getElementById('nombre').value;
  const apellido = document.getElementById('apellido').value;
  const email = document.getElementById('email').value;
  const area = document.getElementById('area').value;
  const cargo = document.getElementById('cargo').value;

  const response = await fetch('/api/registros/registrar', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ dni, nombre, apellido, email, area, cargo })
  });
  const data = await response.json();

  if (response.ok) {
    document.getElementById('successMessage').innerText = data.success;
  } else {
    document.getElementById('successMessage').innerText = 'Error al registrar usuario';
  }
});
```

El código se encarga del registro de un nuevo usuario. Se añade un **listener** al formulario con el **ID registerForm** para manejar el evento de envío. Cuando se envía el formulario, se ejecuta una función asincrónica que previene el comportamiento por defecto del formulario utilizando **event.preventDefault()**. Esto permite manejar el envío del formulario mediante JavaScript en lugar de la forma tradicional. La función obtiene los valores de los campos de entrada del formulario, incluyendo DNI, nombre, apellido, email, área y cargo. Luego, se envía una solicitud **POST** a la ruta **/api/registros/registrar**, incluyendo estos datos en el cuerpo de la solicitud. La función espera la respuesta de la API y la convierte a formato JSON. Si el registro es exitoso, se muestra un mensaje de éxito en el elemento con el **ID successMessage**. En caso de que ocurra un error, se muestra un mensaje de error en el mismo elemento, proporcionando retroalimentación al usuario sobre el estado del registro.

Función de la ruta para conectarse con la reniec y sacar los datos

```
// Ruta para consultar datos de un usuario por DNI
router.post('/', async (req, res) => {
  const { dni } = req.body;
  if (!dni) {
    return res.status(400).json({ error: 'DNI es requerido' });
  }

  const token = 'apis-token-1.aTS1IU7KEuT-6bbbCguH-4Y8TI6KS73N';

  try {
    const response = await axios.get(`https://api.apis.net.pe/v1/dni?numero=${dni}`, {
      headers: {
        'Referer': 'https://apis.net.pe/consulta-dni-api',
        'Authorization': `Bearer ${token}`
      }
    });

    const persona = response.data;

    if (persona.numeroDocumento) {
      res.json(persona);
    } else {
      res.status(404).json({ error: 'DNI no encontrado.' });
    }
  } catch (error) {
    console.error('Error al consultar el DNI:', error);
    res.status(500).json({ error: 'Error al conectar con la API.' });
  }
});
```

El código presentado define una ruta en un servidor para consultar datos de un usuario mediante su DNI. La ruta está configurada para manejar solicitudes **POST** en el **endpoint** raíz (**/**). Dentro de la función asíncrona que maneja la solicitud, primero se extrae el DNI del cuerpo de la solicitud utilizando **const { dni } = req.body;**. Si el DNI no está presente, la función devuelve una respuesta con un código de estado **400 (Bad Request)** y un mensaje de error indicando que el DNI es requerido.

Una vez que se ha validado la presencia del DNI, se define un token de autenticación (**const token = 'apis-token-1.aTS1IU7KEuT-6bbbCguH-4Y8TI6K7S3N';**). Este token es necesario para autenticar las solicitudes a la API externa que se utiliza para obtener los datos del usuario. La función entonces intenta realizar una solicitud GET a la API externa (**https://api.apis.net.pe/v1/dni?numero=\${dni}**) utilizando **axios**. En la solicitud se incluyen encabezados de referencia y autorización para autenticar la solicitud.

Si la solicitud a la API es exitosa, la respuesta de la API se almacena en la variable **persona**. La función verifica si la API ha devuelto un número de documento (**persona.numeroDocumento**). Si es así, se envían los datos de la persona en la

respuesta al cliente. Si no se encuentra el DNI, la función devuelve una respuesta con un código de estado **404 (Not Found)** y un mensaje de error indicando que el DNI no fue encontrado.

En caso de que ocurra un error durante la solicitud a la API, este es capturado en un bloque catch. El error se registra en la consola (**console.error('Error al consultar el DNI:', error);**) y se devuelve una respuesta con un código de estado **500 (Internal Server Error)** y un mensaje de error indicando que hubo un problema al conectar con la API.

La función principal de registrarse

```
const express = require('express');
const router = express.Router();
const Usuario = require('../models/Mentrada'); // Ajuste de ruta al nuevo modelo

/**
 * POST /registrar
 * Ruta para registrar un nuevo usuario.
 */
router.post('/registrar', async (req, res) => {
  const { dni, nombre, apellido, email, area, cargo } = req.body;

  // Validar que todos los campos estén completos
  if (!dni || !nombre || !apellido || !email || !area || !cargo) {
    return res.status(400).send('Todos los campos son obligatorios');
  }

  try {
    // Verificar si el usuario ya existe
    const usuarioExistente = await Usuario.findOne({ where: { dni } });
    if (usuarioExistente) {
      return res.status(400).send('El usuario con este DNI ya existe');
    }

    const nuevoUsuario = await Usuario.create({
      dni,
      nombre,
      apellido,
      email,
      contraseña: dni, // La contraseña es el mismo DNI
      area,
      cargo,
    });

    res.json({ success: 'Usuario registrado exitosamente' });
  } catch (error) {
    console.error('Error al registrar usuario:', error);
    res.status(500).send('Error al registrar usuario');
  }
});

module.exports = router;
```

Activar W
Ve a Configu

El código define una ruta en un servidor Express para registrar un nuevo usuario. La ruta está configurada para manejar solicitudes **POST** en el **endpoint /registrar**. Dentro de la función asincrónica que maneja la solicitud, se extraen los datos del cuerpo de la solicitud (**req.body**), incluyendo dni, nombre, apellido, email, área y cargo. Es importante verificar que todos estos campos estén completos, y si falta alguno, la función devuelve una respuesta con un código de estado **400 (Bad Request)** y un mensaje indicando que todos los campos son obligatorios.

Una vez que se ha validado la presencia de todos los campos, la función intenta verificar si ya existe un usuario con el mismo DNI en la base de datos. Esto se hace mediante la función **Usuario.findOne({ where: { dni } })**, que busca en la base de datos un registro que coincida con el DNI proporcionado. Si se encuentra un usuario existente, la función devuelve una respuesta con un código de estado 400 y un mensaje indicando que el usuario ya existe.

Si no se encuentra un usuario con el mismo DNI, se procede a crear un nuevo registro en la base de datos. La función **Usuario.create** se utiliza para crear un nuevo usuario con los datos proporcionados. La contraseña del usuario se establece como el mismo DNI por simplicidad. Si la creación del usuario es exitosa, se devuelve una respuesta **JSON** con un mensaje de éxito indicando que el usuario ha sido registrado exitosamente.

En caso de que ocurra un error durante el proceso de verificación o creación del usuario, este es capturado en un bloque catch. El error se registra en la consola (**console.error('Error al registrar usuario:', error);**) y se devuelve una respuesta con un código de estado **500 (Internal Server Error)** y un mensaje indicando que hubo un problema al registrar el usuario.

El **Router de Express** se utiliza para definir rutas de forma modular y manejarlas en archivos separados, mejorando la organización del código. Esto se hace mediante la **línea const router = express.Router();** La ruta está configurada para manejar solicitudes **POST** en el **endpoint /registrar, definida por router.post('/registrar', async (req, res) => { ... });**. Esto significa que el servidor espera recibir datos en el cuerpo de la solicitud para procesar el registro de un nuevo usuario.

Es crucial verificar que todos los campos necesarios estén presentes en la solicitud. Esto se realiza con la condición **if (!dni || !nombre || !apellido || !email || !area || !cargo) { ... }**, que devuelve un error 400 si falta algún campo obligatorio, asegurando que todos los datos necesarios para registrar un usuario están completos. Antes de crear un nuevo usuario, el código verifica si ya existe un usuario con el mismo DNI en la base

de datos usando `const usuarioExistente = await Usuario.findOne({ where: { dni } })`; Esto previene duplicados y asegura la unicidad del DNI en el sistema.

Si no se encuentra un usuario con el mismo DNI, el código procede a crear un nuevo registro en la base de datos con `const nuevoUsuario = await Usuario.create({ ... })`; Esto añade un nuevo usuario a la base de datos con la información proporcionada en la solicitud. El bloque catch captura cualquier error que ocurra durante el proceso de verificación o creación del usuario. Los errores se registran en la consola y se devuelve una respuesta con un error 500 al cliente, indicando que hubo un problema interno al registrar el usuario.

La base de datos de los usuarios registrados

```
--
-- Estructura de tabla para la tabla `usuarios`
--
CREATE TABLE `usuarios` (
  `dni` varchar(8) NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `apellido` varchar(100) NOT NULL,
  `email` varchar(100) NOT NULL,
  `contraseña` varchar(100) NOT NULL,
  `area` varchar(100) NOT NULL,
  `cargo` varchar(100) NOT NULL,
  `createdAt` datetime NOT NULL DEFAULT current_timestamp(),
  `updatedAt` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(),
  `activo` varchar(255) DEFAULT 'inactivo'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

La tabla usuarios se utiliza para almacenar información básica sobre los usuarios registrados en el sistema. El campo dni es de tipo `varchar(8)` y sirve como clave primaria, asegurando la unicidad de cada registro y no permitiendo valores nulos (`NOT NULL`). Este campo almacena el dni del usuario.

Los campos nombre, apellido, email, contraseña, área y cargo son de tipo `varchar(100)` y también tienen la restricción `NOT NULL`, garantizando que estos campos siempre contengan información válida y esencial sobre cada usuario. nombre y apellido almacenan el nombre y apellido del usuario respectivamente, mientras que email almacena la dirección de correo electrónico y contraseña guarda la contraseña del usuario. área y cargo especifican el departamento y el puesto del usuario dentro de la organización.

Los campos createdAt y updatedAt son de tipo `datetime` y se utilizan para registrar las marcas de tiempo de creación y última actualización de cada registro. createdAt se

establece automáticamente en la fecha y hora actuales al crear un nuevo registro, y `updatedAt` se actualiza automáticamente cada vez que se modifica el registro.

Finalmente, el campo activo es de tipo **`varchar(255)`** con un valor predeterminado de 'inactivo'. Este campo indica si el usuario está activo en el sistema, proporcionando una manera sencilla de gestionar el estado de los usuarios.

➤ Continuamos con la parte de Logueo para ingresar a las áreas



La página de inicio de sesión se estructura mediante el archivo **`logueo.ejs`**, que define la estructura HTML de la interfaz, y el archivo `logueo.css`, que define los estilos visuales. Estos dos archivos trabajan juntos para crear una experiencia de usuario atractiva y funcional.

En el archivo **`logueo.ejs`**, el encabezado (**`<head>`**) contiene meta-información sobre la página, incluyendo el juego de caracteres, la configuración de la vista para asegurar la responsividad en dispositivos móviles y el título de la página. Además, incluye un enlace a la hoja de estilos **`logueo.css`**. En el cuerpo (**`<body>`**), la página se estructura en dos secciones principales: el encabezado y el contenido principal. El encabezado (**`<div class="header">`**) incluye una imagen de logo y un título de bienvenida. La imagen del logo se carga desde la **`carpeta public/images`** y se muestra al lado del título "Bienvenido". El contenido principal (**`<div class="main-content">`**) contiene un formulario de inicio de sesión, con campos para el correo electrónico y la contraseña, y un botón para enviar el formulario. También hay una sección para mostrar mensajes de error y un enlace para registrar nuevos usuarios si aún no tienen una cuenta.

El archivo **logueo.css** proporciona los estilos para asegurar que la página de inicio de sesión sea atractiva y fácil de usar. El cuerpo de la página (**body**) utiliza la fuente Roboto y tiene un fondo con un degradado lineal, asegurando que el contenido esté centrado tanto vertical como horizontalmente. Además, se añade una capa oscura sobre el fondo para mejorar el contraste y la legibilidad del contenido. El encabezado (**header**) está diseñado para centrar el logo y el título, y el contenido principal (**.main-content**) asegura que el formulario y otros elementos estén bien distribuidos. El contenedor del formulario (**.container**) tiene un fondo blanco semitransparente, bordes redondeados y una sombra para darle relieve, creando un efecto visual agradable.

Los campos de entrada y los botones del formulario están diseñados para ser fáciles de leer y usar. Los campos de entrada tienen un fondo blanco semitransparente y bordes redondeados, mientras que los botones tienen un color de fondo llamativo y cambian de color al pasar el cursor sobre ellos, proporcionando una clara indicación de interactividad. Los mensajes de error se muestran en rojo y están centrados, asegurando que sean visibles para el usuario. Además, los enlaces de registro están diseñados para ser visibles y atractivos, con un efecto de subrayado al pasar el cursor sobre ellos.

Función de envío de datos

```
document.getElementById('loginForm').addEventListener('submit', async (event) => {
  event.preventDefault();

  const email = document.getElementById('email').value;
  const contraseña = document.getElementById('contraseña').value;

  const response = await fetch('/logueo', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ email, contraseña })
  });

  const data = await response.json();

  if (data.success) {
    if (localStorage.getItem('soledad')) {
      localStorage.removeItem('soledad');
    }
  }
});
```

Este código maneja el evento de envío del formulario de inicio de sesión. Al detectar que el formulario ha sido enviado (**submit**), se previene el comportamiento predeterminado del navegador (**event.preventDefault()**), lo que permite manejar el envío del formulario utilizando JavaScript en lugar de una solicitud estándar.

Los valores de los campos de entrada email y contraseña se obtienen mediante `document.getElementById('email').value` y `document.getElementById('contraseña').value`, respectivamente. Estos valores se utilizan para construir un objeto JSON que se envía al servidor mediante una solicitud `fetch` a la ruta `/logueo`. La solicitud `fetch` se configura para usar el método `POST`, se establecen los encabezados para indicar que el contenido es JSON, y se envían los datos de inicio de sesión en el cuerpo de la solicitud.

El uso de `JSON.stringify` convierte el objeto `JavaScript { email, contraseña }` en una cadena JSON, que es el formato adecuado para enviar datos en el cuerpo de una solicitud HTTP. `fetch` es una API moderna de JavaScript para realizar solicitudes HTTP. La palabra clave `await` se utiliza para esperar la respuesta de la solicitud `fetch` antes de continuar con la ejecución del código, lo que permite manejar las solicitudes asíncronas de manera más sencilla.

Una vez que el servidor responde, la respuesta se convierte a un objeto `JSON (const data = await response.json())`. Si la respuesta indica que el inicio de sesión fue exitoso (`data.success`), se procede a gestionar la información del usuario. Primero, si ya existe un elemento soledad en el `localStorage`, se elimina (`localStorage.removeItem('soledad')`). Luego, se almacena la información del usuario en `localStorage` bajo la clave soledad. `localStorage` es una API de almacenamiento web que permite almacenar datos en el navegador del usuario de manera persistente.

Si el inicio de sesión no fue exitoso, el mensaje de error devuelto por el servidor se muestra en el elemento con el ID `errorMessage` (`document.getElementById('errorMessage').innerText = data.message`). Esto proporciona una retroalimentación inmediata al usuario sobre el motivo del fallo en el inicio de sesión.

Función de dirigir a las diferentes áreas

```

const usuario = JSON.parse(localStorage.getItem('soledad'));
const area = usuario.area;

switch (area) {
  case 'Usuarios':
    localStorage.removeItem('productos');
    window.location.href = '/usuarios';
    break;
  case 'Administración':
    window.location.href = '/administracion';
    break;
  case 'Logística':
    window.location.href = '/logistica';
    break;
  case 'Proveedores':
    window.location.href = '/proveedor';
    break;
  case 'Almacén':
    window.location.href = '/almacen';
    break;
  case 'jefe':
    window.location.href = '/jefe';
    break;
  default:
    window.location.href = '/logueo';
}

```

Activa

Este código se enfoca en redirigir al usuario a la página correspondiente según su área una vez que el inicio de sesión ha sido validado exitosamente. Después de almacenar la información del usuario en **localStorage**, se recupera esta información y se convierte nuevamente en un objeto JavaScript mediante **JSON.parse(localStorage.getItem('soledad'))**.

La propiedad área del usuario se utiliza para determinar a qué página redirigir al usuario. Un bloque switch se encarga de redirigir al usuario a la página correspondiente según su área. Por ejemplo, si el área es Usuarios, se elimina cualquier elemento productos del **localStorage** y se redirige al usuario a la página **/usuarios**. Otros casos incluyen redirecciones a páginas de administración, logística, proveedores, almacén o jefe, según el área del usuario. Si el área no coincide con ninguno de los casos, se redirige al usuario nuevamente a la página de inicio de sesión **(/logueo)**.

El uso de **localStorage** permite almacenar y recuperar datos de manera persistente en el navegador del usuario. **window.location.href** se utiliza para redirigir al usuario a una nueva URL. Este mecanismo de redirección asegura que el usuario sea llevado a la página correcta según su rol o área dentro del sistema.

La función principal de Logueo que se conecta con la base de datos

```

    if (usuario) {
      // Establecer sesión del usuario
      req.session.usuario = usuario;

      // Enviar los datos del usuario al cliente
      res.json({
        success: true,
        usuario: {
          dni: usuario.dni,
          nombre: usuario.nombre,
          apellido: usuario.apellido,
          email: usuario.email,
          area: usuario.area,
          cargo: usuario.cargo,
          createdAt: usuario.createdAt,
          updatedAt: usuario.updatedAt
        }
      });
    } else {
      res.json({ success: false, message: 'Usuario o contraseña incorrectos o usuario no activo' });
    }
  } catch (error) {
    console.error('Error al iniciar sesión:', error);
    res.status(500).json({ success: false, message: 'Error al iniciar sesión' });
  }
});

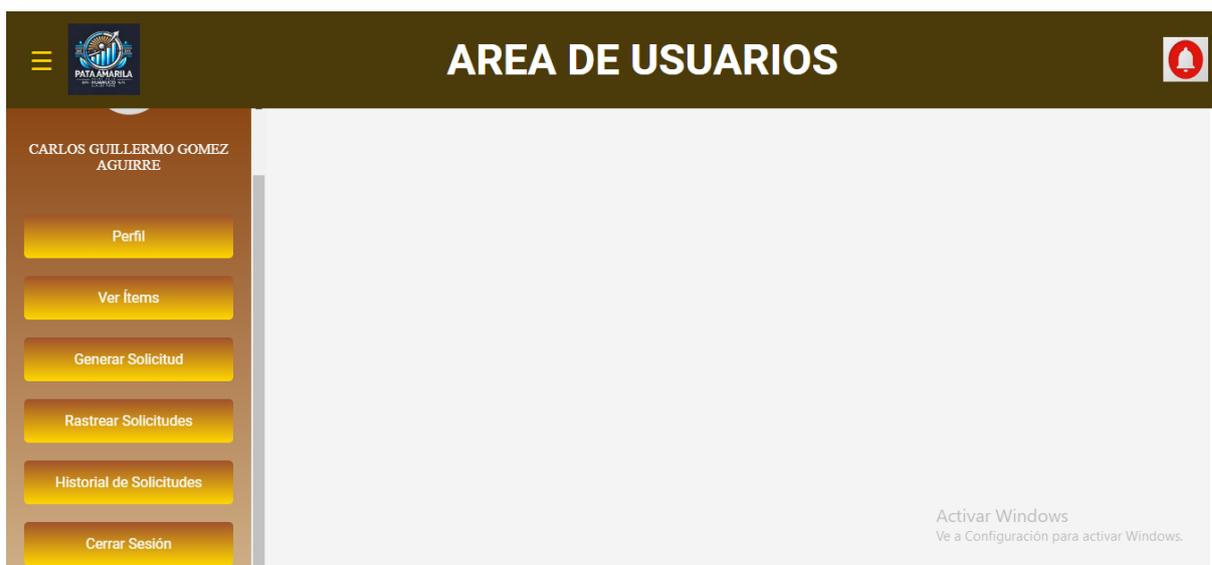
```

El código define la ruta **POST /logueo** para manejar el inicio de sesión de los usuarios. Utiliza el paquete Express para crear un enrutador y el modelo Usuario para interactuar con la base de datos. Cuando se envía una solicitud POST a **/logueo**, el servidor recibe el email y la contraseña del cuerpo de la solicitud. El código intenta encontrar un usuario en la base de datos con el correo electrónico y la contraseña proporcionados, y que esté marcado como activo (**activo: 'activo'**). Si se encuentra un usuario que coincide, se establece una sesión para el **usuario (req.session.usuario = usuario)**, y se envían los datos del usuario de vuelta al cliente en formato JSON, indicando un inicio de sesión exitoso (**success: true**). Si no se encuentra un usuario que coincida, se devuelve una respuesta JSON indicando que el inicio de sesión falló, junto con un mensaje explicativo. En caso de que ocurra algún error durante el proceso, se captura y se registra en la consola, y se devuelve una respuesta JSON con un código de estado 500 y un mensaje de error.

El segundo código define la ruta **GET /logout** para manejar el cierre de sesión de los usuarios. Cuando se accede a esta ruta, se destruye la sesión del usuario actual utilizando **req.session.destroy()**, lo que elimina todos los datos de sesión almacenados en el servidor. Después de destruir la sesión, el servidor redirige al usuario a la página de inicio de sesión (**/logueo**) utilizando **res.redirect('/logueo')**. Esta redirección asegura que, después de cerrar la sesión, el usuario sea llevado de vuelta a la página de inicio de sesión para que pueda iniciar sesión nuevamente si lo desea.

El método **JSON.stringify** se utiliza para convertir un objeto JavaScript en una cadena JSON. En el contexto de enviar datos al servidor, **JSON.stringify** se utiliza para convertir el objeto que contiene email y contraseña en una cadena antes de enviarlo en el cuerpo de una solicitud HTTP. **localStorage** es una API de almacenamiento web que permite almacenar datos en el navegador del usuario de manera persistente. Los datos almacenados en **localStorage** no se eliminan cuando se cierra el navegador y se pueden acceder en futuras visitas a la página. **fetch** es una API moderna para realizar solicitudes HTTP. **fetch** devuelve una promesa que se resuelve con la respuesta de la solicitud. Es utilizada para realizar solicitudes asincrónicas al servidor, como enviar los datos de inicio de sesión y recibir una respuesta. **await** es una palabra clave utilizada para esperar una promesa. Cuando se usa **await** en una función asincrónica, la ejecución de la función se detiene hasta que la promesa se resuelve, permitiendo escribir código asincrónico de manera más clara y lineal.

➤ Área de usuarios



La estructura HTML comienza con la inclusión de los metadatos y la hoja de estilos **usuario_pantalla.css**. En el cuerpo (**<body>**), se define un encabezado fijo que contiene el logo y un botón de cierre de sesión. La clase **header** se encarga de mantener el encabezado fijo en la parte superior de la página. La sección principal se divide en una barra lateral (**sidebar**) y el contenido principal (**main-content**). La barra lateral contiene la foto de perfil del usuario y botones de navegación para mostrar diferentes secciones (**perfil, actividades, ítems**). La sección de perfil muestra los datos del usuario y un formulario para actualizar su información. La sección de actividades

contiene una tabla para listar actividades y un botón para enviar pedidos. La sección de items incluye un campo de búsqueda y una tabla para listar items.

El archivo **usuario_pantalla.css** define estilos detallados para diferentes elementos de la página de administración de usuarios. El encabezado (**header**) está fijado en la parte superior de la página, con un fondo marrón oscuro y texto amarillo dorado. La barra lateral (**sidebar**) utiliza un gradiente marrón y tiene estilos para mostrar y ocultar elementos cuando está colapsada. El contenido principal (**main-content**) está alineado a la derecha de la barra lateral y tiene un fondo claro. La sección de perfil y la tabla de actividades tienen estilos específicos para la disposición de los datos y formularios, asegurando una apariencia limpia y profesional.

Función para cargar datos en la tabla

```
function cargarActividades() {
  const productos = JSON.parse(localStorage.getItem('productos')) || [];
  const tbody = document.querySelector('#actividadesTable tbody');
  tbody.innerHTML = '';

  productos.forEach((producto, index) => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${producto.descripcion}</td>
      <td>${producto.cantidad}</td>
      <td>${producto.precio}</td>
      <td><button onclick="eliminarProducto(${index})">Eliminar</button></td>
    `;
    tbody.appendChild(row);
  });
}
```

La función **cargarActividades** se encarga de cargar y mostrar los productos almacenados en el carrito en una tabla de actividades. Primero, obtiene los datos del carrito almacenados en el **localStorage** bajo la clave **productos**, utilizando **JSON.parse** para convertir la cadena JSON en un objeto JavaScript. Si no hay productos almacenados, se asigna un array vacío por defecto. Luego, selecciona el elemento **<tbody>** de la tabla con el ID **actividadesTable** y limpia su contenido estableciendo **tbody.innerHTML** en vacío. Esto asegura que la tabla se limpie antes de añadir nuevos productos, evitando duplicados.

A continuación, la función recorre el array de productos utilizando **forEach**. Para cada producto, se crea una nueva fila de tabla (**<tr>**) y se define su contenido utilizando plantillas literales para insertar los valores de descripción, cantidad y precio del producto. También se añade un botón para eliminar el producto, que llama a la función **eliminarProducto** pasando el índice del producto como argumento. Finalmente, la fila creada se añade al cuerpo de la tabla (**tbody.appendChild(row)**). Este proceso se

repite para cada producto en el carrito, asegurando que todos los productos se muestren en la tabla de actividades.}

Función para cargar datos en la tabla

```
function eliminarProducto(index) {
  const productos = JSON.parse(localStorage.getItem('productos')) || [];
  productos.splice(index, 1);
  localStorage.setItem('productos', JSON.stringify(productos));
  cargarActividades();
}
```

La función **eliminarProducto** se encarga de eliminar un producto específico del carrito de compras, basado en su índice. Primero, la función obtiene los productos almacenados en el **localStorage** bajo la clave productos. Utiliza **JSON.parse** para convertir la cadena JSON almacenada en un array de objetos JavaScript. Si no hay productos almacenados, se asigna un array vacío por defecto.

Una vez obtenido el array de productos, la función utiliza el método **splice** para eliminar un elemento del array en el índice especificado (**index**). El método **splice** modifica el array original, eliminando el elemento en la posición indicada. Después de eliminar el producto del array, la función actualiza el **localStorage** con el array modificado. Utiliza **localStorage.setItem** para almacenar el array actualizado, convirtiéndolo en una cadena JSON con **JSON.stringify**.

Finalmente, la función llama a **cargarActividades** para actualizar la tabla de actividades en la interfaz de usuario. Esto asegura que la tabla refleje correctamente el estado actual del carrito de compras, mostrando los productos restantes después de la eliminación.

La datos de la tabla se cargan en la tabla del siguiente grafico.donde las personas pueden hacer sus pedidos de los bienes y suministros que desean.

The screenshot shows a web application interface titled "AREA DE USUARIOS". On the left, there is a user profile sidebar for "CARLOS GUILLERMO GOMEZ AGUIRRE" with buttons for "Perfil", "Ver ítems", "Generar Solicitud", "Rastrear Solicitudes", and "Historial de Solicitudes". The main area features a search bar with the letter "c" and a table of products. The table has columns for ID, Descripción, Unidad, Stock, Precio, and Acciones. Each row includes a "Seleccionar" button.

ID	Descripción	Unidad	Stock	Precio	Acciones
3	caramelos	bolsa	854	4.00	Seleccionar
4	celular	celular	843	1300.00	Seleccionar
5	agua cielo	botella 500ml	818	2.00	Seleccionar
7	Camisa Formal	Unidades	4	120.00	Seleccionar
8	Laptop ABC	Unidades	2	3500.00	Seleccionar
9	Zapatos de Cuero	Unidades	2	250.00	Seleccionar
10	Libro de Matemáticas	Unidades	3	80.00	Seleccionar

Función para enviar todos los bienes y suministros elegidos

```

    async function enviarPedido() {
const productos = JSON.parse(localStorage.getItem('productos')) || [];
const usuario = JSON.parse(localStorage.getItem('soledad'));

if (productos.length === 0) {
    alert('No hay productos en el carrito.');
```

```

    return;
}

const pedido = productos.map(producto => ({
    descripcion_item: producto.descripcion,
    cantidad: producto.cantidad,
    precio: producto.precio,
    dni_registrador: usuario.dni
}));

try {
    const response = await fetch('/api/pedidos', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(pedido)
    });

    if (response.ok) {
        alert('Pedido enviado correctamente');
        localStorage.removeItem('productos');
        cargarActividades();
    } else {
        alert('Error al enviar el pedido');
    }
} catch (error) {
    console.error('Error al enviar el pedido:', error);
    alert('Error al enviar el pedido');
}
}

```

Activar Windows

Activar Windows

La función **enviarPedido** se encarga de enviar un pedido al servidor utilizando los datos almacenados en el **localStorage**. Primero, la función obtiene los productos almacenados en el **localStorage** bajo la clave **productos** y los datos del usuario bajo la clave **soledad**. Utiliza **JSON.parse** para convertir las cadenas JSON almacenadas en objetos JavaScript. Si no hay productos en el carrito, se muestra una alerta indicando que no hay productos en el carrito y se finaliza la ejecución de la función utilizando **return**.

Si hay productos en el carrito, se construye un objeto de pedido. Utilizando el método **map**, se transforma el array de productos en un array de objetos que contienen las propiedades **descripcion_item**, **cantidad**, **precio** y **dni_registrador**. Estas propiedades se asignan a partir de los datos de cada producto y el DNI del usuario registrado.

La función **fetch** se utiliza para enviar una solicitud HTTP al servidor en la ruta **/api/pedidos**. La solicitud se configura para utilizar el método POST y enviar los datos del pedido en formato JSON. Para lograr esto, se establece el encabezado **Content-Type**

como `application/json` y se utiliza `JSON.stringify` para convertir el objeto de pedido en una cadena JSON antes de enviarla en el cuerpo de la solicitud.

La palabra clave `await` se utiliza para esperar la respuesta del servidor. Si la respuesta indica que la solicitud fue exitosa (`response.ok`), se muestra una alerta indicando que el pedido se envió correctamente. Luego, se elimina el carrito de productos del `localStorage` utilizando `localStorage.removeItem('productos')` y se actualiza la tabla de actividades llamando a la función `cargarActividades`. Si la respuesta no es exitosa, se muestra una alerta indicando que hubo un error al enviar el pedido.

Si ocurre un error durante la solicitud `fetch`, se captura en el bloque `catch`, se registra en la consola para propósitos de depuración y se muestra una alerta indicando que hubo un error al enviar el pedido. En resumen, la función `enviarPedido` proporciona una manera robusta de manejar el envío de pedidos desde el cliente al servidor, asegurando que los datos se envíen correctamente y manejando cualquier error que pueda ocurrir durante el proceso.

La función para obtener el número de boleta

```
// Obtener el último número de pedido
async function obtenerUltimoNumeroPedido() {
  const pedidoNumero = await PedidoNumero.findByPk(1);
  return pedidoNumero ? pedidoNumero.numero : 0;
}
```

La función `obtenerUltimoNumeroPedido` es una función asíncrona que se encarga de obtener el último número de pedido registrado en la base de datos. Utiliza el método `findByPk` del modelo `PedidoNumero` para buscar un registro con la clave primaria 1. Si encuentra un registro, retorna el valor del campo `numero`. Si no encuentra un registro, retorna 0 como valor por defecto. Esta función asegura que siempre se tenga un número de pedido válido, incluso si no hay registros previos en la tabla.

Función para actualizar el número de pedidos

```
// Actualizar el número de pedido
async function actualizarNumeroPedido(nuevoNumero) {
  let pedidoNumero = await PedidoNumero.findByPk(1);
  if (pedidoNumero) {
    pedidoNumero.numero = nuevoNumero;
    await pedidoNumero.save();
  } else {
    await PedidoNumero.create({ id: 1, numero: nuevoNumero });
  }
}
```

Es una función asíncrona que se encarga de actualizar el número de pedido en la tabla **PedidoNumero**. Primero, intenta encontrar un registro con la clave primaria 1 utilizando **findByPk**. Si encuentra un registro, actualiza el campo número con el **nuevoNumero** proporcionado y guarda los cambios en la base de datos utilizando **save**. Si no encuentra un registro, crea un nuevo registro en la tabla con el id de 1 y el **nuevoNumero** proporcionado. Esta función garantiza que el número de pedido siempre esté actualizado y listo para el próximo pedido.

Conexión con la base de datos para crear un nuevo pedido

```
// Ruta para crear un nuevo pedido
router.post('/', async (req, res) => {
  try {
    const pedidos = req.body;
    const ultimoNumeroPedido = await obtenerUltimoNumeroPedido();
    const nuevoNumeroPedido = ultimoNumeroPedido + 1;

    // Capturar la fecha y hora locales usando moment-timezone
    const fechaHoraActual = moment().tz('America/Lima').format('YYYY-MM-DD HH:mm:ss');

    // Añadir el número de pedido y la fecha y hora a cada pedido
    const pedidosConNumeroYFecha = pedidos.map(pedido => ({
      ...pedido,
      numero_pedido: nuevoNumeroPedido,
      createdAt: fechaHoraActual,
      updatedAt: fechaHoraActual
    }));

    // Crear los pedidos en la base de datos
    await Pedido.bulkCreate(pedidosConNumeroYFecha);

    // Actualizar el número de pedido en la tabla PedidoNumero
    await actualizarNumeroPedido(nuevoNumeroPedido);

    res.status(201).send('Pedido creado correctamente');
  } catch (error) {
    console.error('Error al crear el pedido:', error);
    res.status(500).send('Error al crear el pedido');
  }
});
```

Esta ruta define el manejo de solicitudes POST para crear nuevos pedidos. Cuando se recibe una solicitud POST en esta ruta, la función primero obtiene el cuerpo de la solicitud (**req.body**), que contiene los datos de los pedidos. Luego, llama a **obtenerUltimoNumeroPedido** para obtener el último número de pedido y calcula el nuevo número de pedido incrementándolo en uno.

Usando **moment-timezone**, captura la fecha y hora actuales en la zona horaria de **'America/Lima'** y las formatea en **'YYYY-MM-DD HH:mm'**. A continuación, agrega el número de pedido y la fecha y hora a cada pedido en el cuerpo de la solicitud utilizando **map**.

Luego, utiliza **bulkCreate** del modelo **Pedido** para insertar todos los pedidos en la base de datos en una sola operación. Después de crear los pedidos, llama a **actualizarNumeroPedido** para actualizar el número de pedido en la tabla **PedidoNumero**.

Si todo el proceso es exitoso, la función responde con un código de estado 201 indicando que los pedidos se crearon correctamente. Si ocurre un error, lo captura en el bloque catch, registra el error en la consola y responde con un código de estado 500 y un mensaje de error indicando que hubo un problema al crear el pedido. Esta ruta asegura la creación eficiente y correcta de nuevos pedidos, manejando adecuadamente los errores y actualizando el número de pedido para futuras solicitudes.

En la siguiente grafica se muestra cuando realiza la solicitud de los bienes y suministros.



Descripción	Cantidad	Precio	Acciones
galleta	2	5	Eliminar
caramelos	2	4	Eliminar
celular	2	1300	Eliminar
agua cielo	2	2	Eliminar

Enviar Pedido

➤ Base de datos de bienes solicitados

```
-- Estructura de tabla para la tabla `pedido`
CREATE TABLE `pedido` (
  `id` int(11) NOT NULL,
  `descripcion_item` varchar(255) NOT NULL,
  `cantidad` int(11) NOT NULL,
  `precio` float NOT NULL,
  `dni_registrador` varchar(20) NOT NULL,
  `numero_pedido` int(11) NOT NULL,
  `createdAt` datetime NOT NULL DEFAULT current_timestamp(),
  `updatedAt` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

La tabla de pedido está diseñada para almacenar información detallada de cada pedido realizado. Contiene los campos id (clave primaria), descripcion_item (descripción del ítem pedido), cantidad (número de ítems), precio (costo del ítem), dni_registrador (DNI del usuario que registró el pedido), numero_pedido (número secuencial del pedido), createdAt (fecha y hora de creación), y updatedAt (fecha y hora de la última actualización). Los campos createdAt y updatedAt utilizan marcas de tiempo que se establecen automáticamente cuando se crea o se actualiza un registro. La tabla está

configurada para utilizar el motor de almacenamiento InnoDB y el conjunto de caracteres UTF-8.

➤ Funciones del área del jefe

The screenshot shows a web application interface for a manager. At the top, a dark header displays the name "JEFE, FRANK NILER AMBROSIO SANCHEZ" in yellow. Below the header is a search form with a text input field containing "car", two buttons labeled "CARLOS GUILLERMO" and "GOMEZ AGUIRRE", and a green "Aceptar" button. Below the search form is a table titled "TABLA DE ITEMS SOLICITADOS" with the following data:

ID	Descripción	Cantidad	Precio	DNI Usuario	Numero de Boleto
180	galleta	2	5	71302561	11
181	caramelos	2	4	71302561	11
182	celular	2	1300	71302561	11

The footer of the page includes the text "EMPRESA DE TURISMO PATA AMARILLA" and a watermark for Windows activation.

La estructura HTML comienza con la inclusión de los metadatos y el enlace a la hoja de estilos [AREA_jefe.css](#). El cuerpo de la página está dividido en tres secciones principales: el encabezado, el contenido principal y el pie de página.

El **encabezado** (`<div class="header">`) contiene un título centrado que indica el área de trabajo del jefe.

El **contenido principal** (`<div class="main-content">`) incluye una sección de búsqueda (`<div class="search-section">`) con dos campos de entrada para buscar por apellido y nombre, y un botón para realizar la búsqueda. La sección de búsqueda ayuda a los jefes a filtrar los datos rápidamente. También se incluye un **panel de control** (`<div class="dashboard">`) que contiene una tabla para mostrar los pedidos registrados. La tabla tiene columnas para el **ID** del pedido, descripción, cantidad, precio, registrado por, y fecha.

El **pie de página** (`<div class="footer">`) contiene un mensaje de derechos de autor. Finalmente, se incluye un enlace a un archivo JavaScript ([jefe.js](#)) que manejará la interacción del usuario con la página.

El encabezado tiene un fondo con un gradiente de negro a gris oscuro, con texto en amarillo y una sombra en la parte inferior. El título en el encabezado tiene un tamaño de fuente grande y está centrado. La sección de contenido principal está centrada y tiene un relleno para separación del borde de la página.

La sección de búsqueda está diseñada con un fondo blanco, bordes redondeados y una sombra ligera. Los campos de entrada y el botón de búsqueda están estilizados para ser fáciles de usar y atractivos visualmente, con bordes redondeados y transiciones suaves al enfocarse o al pasar el cursor. Los contenedores de entrada están organizados en una columna en pantallas pequeñas y en una fila en pantallas más grandes para una mejor responsividad.

El panel de control tiene un fondo blanco, bordes redondeados y una sombra ligera para darle relieve. La tabla de pedidos está completamente estilizada para ser clara y fácil de leer, con filas alternas de colores para mejorar la legibilidad. Los encabezados de las columnas tienen un fondo amarillo claro y texto en negrita.

El pie de página está fijado en la parte inferior de la página, con un fondo negro y texto en amarillo, y también tiene una sombra en la parte superior para separarlo visualmente del contenido principal.

Función para cargar la tabla por el dni

```

searchInput.addEventListener('input', async () => {
  const searchTerm = searchInput.value;
  if (searchTerm) {
    const response = await fetch(`/api/usuarios?search=${searchTerm}`);
    const usuarios = await response.json();
    if (usuarios.length > 0) {
      const usuario = usuarios[0];
      nombreInput.value = usuario.nombre;
      apellidoInput.value = usuario.apellido;
      dniSeleccionado = usuario.dni;
      cargarPedidos(usuario.dni);
    } else {
      nombreInput.value = '';
      apellidoInput.value = '';
      dniSeleccionado = '';
      cargarPedidos();
    }
  } else {
    nombreInput.value = '';
    apellidoInput.value = '';
    dniSeleccionado = '';
    cargarPedidos();
  }
});

```

Esta función se ejecuta cada vez que el usuario escribe algo en el campo de búsqueda. Obtiene el término de búsqueda y realiza una solicitud a la API para obtener los usuarios que coinciden con el término. Si se encuentran usuarios, se llenan los campos de nombre y apellido con los datos del primer usuario encontrado, y se almacena su DNI. Luego, se cargan los pedidos asociados a ese DNI. Si no se encuentran usuarios, se limpian los campos de nombre y apellido y se cargan todos los pedidos sin filtrar por DNI.

Función para cargar toda la tabla de solicitudes

ID	Descripción	Cantidad	Precio	DNI Usuario	Numero de Boleto
180	galleta	2	5	71302561	11
181	caramelos	2	4	71302561	11
182	celular	2	1300	71302561	11
183	agua cielo	2	2	71302561	11

```
const cargarPedidos = async (dni = '') => {
  const response = await fetch(`/api/pedidos?dni=${dni}`);
  const pedidos = await response.json();
  pedidosTableBody.innerHTML = '';
  pedidos.forEach(pedido => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${pedido.id}</td>
      <td>${pedido.descripcion_item}</td>
      <td>${pedido.cantidad}</td>
      <td>${pedido.precio}</td>
      <td>${pedido.dni_registrador}</td>
      <td>${pedido.numero_pedido}</td>
    `;
    pedidosTableBody.appendChild(row);
  });
};
```

La función se encarga de obtener y mostrar los pedidos desde el servidor en una tabla HTML. Utiliza el método **fetch** para realizar una solicitud HTTP GET a la API del servidor en la **ruta /api/pedidos**, pasando opcionalmente un parámetro dni para filtrar los pedidos por el DNI del registrador. El uso de **fetch** devuelve una promesa que se resuelve con la respuesta de la solicitud. Una vez que se recibe la respuesta, se utiliza **response.json()** para convertirla en un objeto JavaScript, que en este caso es un array de pedidos. La propiedad **await** se utiliza para esperar a que esta promesa se resuelva antes de continuar con la ejecución del código.

Después de obtener los pedidos, se limpia el contenido actual del cuerpo de la tabla de pedidos (**pedidosTableBody.innerHTML = '';**) para evitar mostrar datos duplicados. Luego, se recorre el array de pedidos utilizando el método **forEach**. Para cada pedido, se crea una nueva fila de tabla (**<tr>**), y se define su contenido HTML

utilizando plantillas literales para insertar dinámicamente los valores del ID del pedido, descripción del ítem, cantidad, precio, DNI del registrador y número del pedido en las celdas de la fila. La propiedad **fetch** es una API moderna para realizar solicitudes HTTP. En este caso, se utiliza para enviar una solicitud GET al servidor para obtener los pedidos. La URL incluye el parámetro dni para filtrar los pedidos si se proporciona un valor. El método **response.json()** se utiliza para parsear la respuesta de la solicitud HTTP y convertirla en un objeto JavaScript. Es una operación asíncronica y se espera con **await**. Finalmente, cada fila creada se añade al cuerpo de la tabla (**pedidosTableBody.appendChild(row);**). Este proceso asegura que todos los pedidos se muestren correctamente en la tabla, reflejando el estado actual de los datos obtenidos del servidor. La línea de código **pedidosTableBody.innerHTML = ''**; limpia el contenido actual del cuerpo de la tabla de pedidos, asegurando que la tabla se actualice correctamente sin duplicar los datos existentes. El método **forEach** itera sobre cada elemento del array de pedidos, ejecutando una función para cada elemento. En este caso, se utiliza para crear y añadir una nueva fila en la tabla para cada pedido. Las plantillas literales son una forma de incluir expresiones dentro de cadenas de texto utilizando la sintaxis **\${expresion}**. En esta función, se utilizan para insertar dinámicamente los valores de las propiedades de cada pedido en las celdas de la fila de la tabla.

Función para obtener todos los datos de la base de datos y filtrar por dni

```
const express = require('express');
const router = express.Router();
const JefeArea = require('../models/JefeArea');

router.get('/', async (req, res) => {
  const { dni, numero_pedido } = req.query;
  let whereClause = {};

  if (dni) {
    whereClause.dni_registrador = dni;
    console.log(`Filtrando por DNI: ${dni}`);
  }

  if (numero_pedido) {
    whereClause.numero_pedido = numero_pedido;
    console.log(`Filtrando por número de pedido: ${numero_pedido}`);
  }

  try {
    const datos = await JefeArea.findAll({ where: whereClause });
    res.json(datos);
  } catch (error) {
    console.error('Error al obtener datos:', error);
    res.status(500).send('Error al obtener datos');
  }
});
```

La ruta `router.get('/')` define un manejador para las solicitudes GET a la ruta raíz. Esta función asíncrona se encarga de obtener datos del modelo `JefeArea` según los parámetros de consulta (`dni y numero_pedido`) que se envían en la solicitud. Primero, se inicializa un objeto `whereClause` vacío que se usará para construir la cláusula WHERE de la consulta. Si se proporciona un `dni`, se añade al objeto `whereClause` y se imprime un mensaje en la consola indicando que se está filtrando por ese DNI. De manera similar, si se proporciona un `numero_pedido`, también se añade al objeto `whereClause` y se imprime un mensaje en la consola.

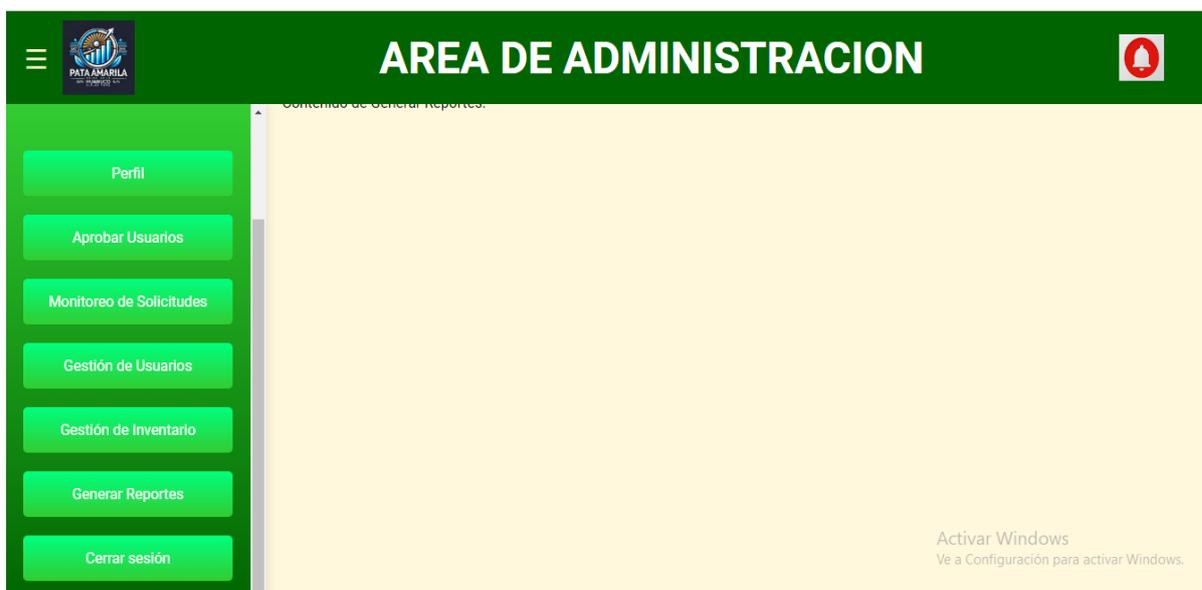
La función utiliza `await` para esperar a que el modelo `JefeArea` busque todos los registros que coincidan con la cláusula WHERE construida (`whereClause`). Si la búsqueda es exitosa, los datos obtenidos se envían como respuesta en formato JSON. Si ocurre un error durante el proceso de obtención de datos, se captura en el bloque `catch`, se imprime un mensaje de error en la consola, y se responde con un código de estado 500 y un mensaje de error indicando que hubo un problema al obtener los datos.

Tabla de la base de datos de los 328tems aceptados por el jefe

```
CREATE TABLE `jefe_area` (
  `id` int(11) NOT NULL,
  `descripcion_item` varchar(255) NOT NULL,
  `cantidad` int(11) NOT NULL,
  `precio` decimal(10,2) NOT NULL,
  `dni_registrador` varchar(20) NOT NULL,
  `numero_pedido` varchar(50) NOT NULL,
  `createdAt` timestamp NOT NULL DEFAULT current_timestamp(),
  `updatedAt` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(),
  `dni_jefe` varchar(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

La tabla `jefe_area` almacena información detallada de los pedidos supervisados por el jefe de área. Incluye un `id` como clave primaria, `descripcion_item` para la descripción del ítem, `cantidad` para la cantidad de ítems, `precio` como decimal, `dni_registrador` para el DNI del registrador, `numero_pedido` para el número del pedido, `createdAt` y `updatedAt` como marcas de tiempo automáticas, y `dni_jefe` para el DNI del jefe. Todos los campos son obligatorios y la tabla utiliza el motor InnoDB con el conjunto de caracteres UTF-8 para asegurar compatibilidad internacional y manejo eficiente de datos.

➤ Funciones del área de administración



El archivo **administracion.ejs** define la estructura HTML de la página de administración, la cual incluye un encabezado fijo, una barra lateral de navegación y una sección de contenido principal. El encabezado (**<header class="header">**) tiene un diseño flexible y fijo en la parte superior de la página, con un título centrado que indica "Panel de Administración". La barra lateral (**<aside class="sidebar">**) contiene el perfil del administrador y botones de navegación para acceder a diferentes secciones, como "Perfil" y "Aprobar Usuarios". El contenido principal (**<main class="main-content">**) se divide en secciones que se muestran u ocultan según la navegación del usuario.

Función de los eventos

```
document.addEventListener('DOMContentLoaded', function() {
  const searchUser = document.getElementById('searchUser');
  const selectDNI = document.getElementById('selectDNI');
  const selectNombre = document.getElementById('selectNombre');
  const selectApellido = document.getElementById('selectApellido');
  const selectNumeroPedido = document.getElementById('selectNumeroPedido');
  const tablaDatos = document.getElementById('tablaDatos').querySelector('tbody');

  searchUser.addEventListener('input', buscarUsuarios);
  selectNombre.addEventListener('change', sincronizarComboboxes);
  selectApellido.addEventListener('change', sincronizarComboboxes);
  selectNumeroPedido.addEventListener('change', cargarDatosPorNumeroPedido);
});
```

El primer bloque de código utiliza **document.addEventListener** con el evento **DOMContentLoaded** para asegurar que la función anónima que contiene el resto del código solo se ejecute una vez que todo el contenido del DOM haya sido completamente cargado y parseado. Esto es crucial para garantizar que todos los elementos del DOM a los que el código hace referencia estén disponibles cuando se intenta acceder a ellos.

Dentro de esta función anónima, se declaran varias constantes que se refieren a elementos específicos del DOM usando `document.getElementById`. Estas constantes son `searchUser`, `selectDNI`, `selectNombre`, `selectApellido`, `selectNumeroPedido` y `tablaDatos`. Cada una de estas constantes se asocia con un elemento HTML específico en la página, como campos de entrada o selectores (**comboboxes**), y la tabla donde se mostrarán los datos. Por ejemplo, `searchUser` se refiere al campo de entrada donde los usuarios pueden escribir para buscar otros usuarios, y `tablaDatos` se refiere al cuerpo de la tabla donde se mostrarán los resultados de las búsquedas y otras operaciones.

El segundo bloque de código asigna manejadores de eventos a estos elementos utilizando el método `addEventListener`. Para `searchUser`, se agrega un evento input que llama a la función `buscarUsuarios` cada vez que el usuario escribe en el campo de búsqueda. Esto permite que la búsqueda de usuarios se realice en tiempo real a medida que el usuario escribe.

Para los selectores `selectNombre` y `selectApellido`, se agrega un evento `change` que llama a la función `sincronizarComboboxes` cada vez que el usuario cambia la selección en estos combobox. Esto asegura que los combobox se mantengan sincronizados, mostrando opciones consistentes y actualizadas.

Finalmente, para `selectNumeroPedido`, se agrega un evento `change` que llama a la función `cargarDatosPorNumeroPedido` cada vez que el usuario selecciona un número de pedido diferente. Esto permite que los datos correspondientes al número de pedido seleccionado se carguen y se muestren en la tabla.

Buscador de usuarios para cargar en el combobox

```

async function buscarUsuarios() {
  const query = searchUser.value;
  if (query === "") {
    cargarDatos();
  } else {
    const response = await fetch(`/api/usuarios?search=${query}`);
    const usuarios = await response.json();
    actualizarComboboxes(usuarios);
  }
}

```

Es una función asíncrona que maneja la búsqueda de usuarios en tiempo real. Primero, obtiene el valor del campo de búsqueda `searchUser` y lo almacena en la constante `query`. Si el campo de búsqueda está vacío (`query === ""`), llama a la función `cargarDatos()` para cargar todos los datos disponibles sin ningún filtro. La función `cargarDatos` se encarga de obtener y mostrar todos los registros en la tabla, lo que es útil cuando no se ha ingresado ningún término de búsqueda específico.

Si el campo de búsqueda no está vacío, la función realiza una solicitud HTTP a la API utilizando **fetch**, pasando el valor de **query** como parámetro de búsqueda en la URL (**/api/usuarios?search=\${query}**). La palabra clave **await** se utiliza para esperar a que la promesa de **fetch** se resuelva, lo que significa que el código se detendrá en esta línea hasta que la solicitud HTTP se complete. Una vez que se obtiene la respuesta, se convierte a formato JSON usando **response.json()**, lo que también es una operación asíncrona que requiere **await**. Los datos obtenidos, que representan los usuarios que coinciden con el término de búsqueda, se pasan a la función para actualizar el combobox con los resultados de la búsqueda.

Actualizar los combobox

```
function actualizarComboboxes(usuarios) {
  limpiarComboboxes();

  usuarios.forEach(usuario => {
    const optionDNI = new Option(usuario.dni, usuario.dni);
    const optionNombre = new Option(`${usuario.nombre} ${usuario.apellido}`, usuario.dni);
    const optionApellido = new Option(usuario.apellido, usuario.dni);

    selectDNI.add(optionDNI);
    selectNombre.add(optionNombre);
    selectApellido.add(optionApellido);
  });

  cargarNumeroPedidos(); // Cargar números de pedido según el primer resultado
  cargarDatos(); // Cargar todos los datos al inicio
}
```

La función se encarga de actualizar los elementos **comboboxes (select)** con los datos de los usuarios obtenidos de la búsqueda. Primero, llama a la función **limpiarComboboxes** para vaciar los combobox actuales y asegurarse de que no haya datos previos. Luego, utiliza el método **forEach** para iterar sobre cada objeto de usuario en el array **usuarios**.

Dentro del **forEach**, se crean nuevas opciones (**optionDNI**, **optionNombre** y **optionApellido**) para cada usuario utilizando el constructor **new Option**. Este constructor toma dos argumentos: el texto que se mostrará en la opción y el valor de la opción. En el caso de **optionDNI**, ambos argumentos son el DNI del usuario. Para **optionNombre**, el texto mostrado es el nombre completo del usuario (nombre y apellido) y el valor es el **DNI**. **optionApellido** usa el apellido del usuario como texto y el DNI como valor.

Después de crear estas opciones, se añaden a los respectivos combobox (**selectDNI**, **selectNombre** y **selectApellido**) utilizando el método **add**.

Finalmente, la función llama a **cargarNumeroPedidos** para cargar los números de pedido únicos asociados al primer usuario en la lista y **cargarDatos** para cargar todos los datos

disponibles en la tabla. Esto asegura que los comboboxes se llenen con los datos más recientes y que la tabla de datos se actualice para reflejar cualquier cambio.

Cargar los números de las solicitudes según el dni

```

async function cargarNumeroPedidos() {
  const dni = selectDNI.value;
  console.log(`DNI seleccionado: ${dni}`); // Verificar el DNI seleccionado

  try {
    const response = await fetch(`/api/jefe_area?dni=${dni}`);
    const datos = await response.json();

    console.log('Datos recibidos:', datos); // Verificar los datos recibidos

    const numerosPedidoUnicos = [...new Set(datos.map(dato => dato.numero_pedido))];
    console.log('Números de pedido únicos:', numerosPedidoUnicos); // Verificar los números de pedido único

    limpiarCombobox(selectNumeroPedido);

    numerosPedidoUnicos.forEach(numero => {
      const optionNumeroPedido = new Option(numero, numero);
      selectNumeroPedido.add(optionNumeroPedido);
    });

    // Verificar que los números de pedido se han añadido al combobox
    console.log('Opciones en el combobox de número de pedido:',
      Array.from(selectNumeroPedido.options).map(option => option.value));

    // Cargar datos en la tabla según el dni seleccionado
    cargarDatos(dni);
  } catch (error) {
    console.error('Error al cargar números de pedido:', error); // Mostrar errores en la consola
  }
}

```

Es una función asíncrona que se encarga de cargar y mostrar los números de pedido únicos asociados a un DNI seleccionado. Primero, obtiene el valor del combobox selectDNI y lo almacena en la constante dni, mostrando este valor en la consola para verificar cuál DNI ha sido seleccionado.

Dentro de un bloque try, la función realiza una solicitud HTTP a la API utilizando **fetch**, pasando el valor del DNI como parámetro en la URL (**/api/jefe_area?dni=\${dni}**). La palabra clave **await** se utiliza para esperar a que la promesa de **fetch** se resuelva y para convertir la respuesta a formato JSON, almacenando los datos recibidos en la constante datos. Estos datos se muestran en la consola para verificar su contenido.

La función luego crea un array de números de pedido únicos usando new Set para eliminar duplicados y **map** para extraer los números de pedido de los datos recibidos. Este array se almacena en la constante numerosPedidoUnicos y se muestra en la consola para verificar que los números de pedido únicos se han extraído correctamente.

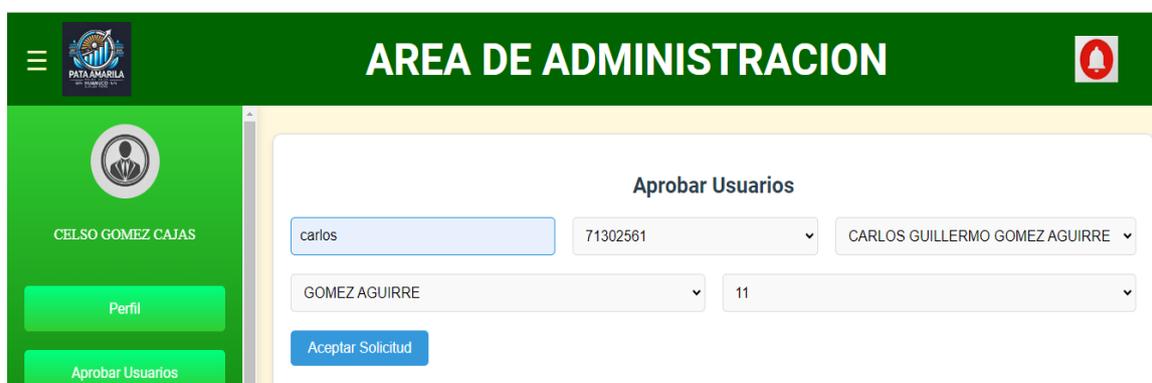
Después de esto, la función llama a limpiarCombobox para vaciar el combobox selectNumeroPedido antes de añadir las nuevas opciones. Luego, utiliza **forEach** para iterar sobre cada número de pedido único y crea una nueva opción

(optionNumeroPedido) para cada número utilizando el constructor **new Option**. Estas opciones se añaden al combobox selectNumeroPedido.

Para verificar que las opciones se han añadido correctamente, la función muestra en la consola los valores de todas las opciones actuales del combobox. Finalmente, llama a cargarDatos pasando el DNI seleccionado para cargar los datos correspondientes en la tabla.

Si ocurre un error en cualquier parte del proceso, se captura en el bloque catch y se muestra un mensaje de error en la consola, asegurando que cualquier problema durante la carga de los números de pedido se registre para su posterior depuración.

En la siguiente imagen se muestra los resultados que se muestran



Función para cargar datos a la tabla

```

async function cargarDatos(dni = null) {
  let query = '/api/jefe_area';

  if (dni) {
    query += `?dni=${dni}`;
  }

  const response = await fetch(query);
  const datos = await response.json();

  console.log('Datos para cargar en la tabla:', datos); // Verificar los datos recibidos para la tabla

  tablaDatos.innerHTML = '';

  datos.forEach(dato => {
    const row = document.createElement('tr');
    row.dataset.id = dato.id; // Asignar el ID del registro a data-id
    row.innerHTML = `
      <td>${dato.descripcion_item}</td>
      <td>${dato.cantidad}</td>
      <td>${dato.precio}</td>
      <td>${dato.dni_registrador}</td>
      <td>${dato.numero_pedido}</td>
      <td>${dato.dni_jefe}</td>
    `;
    tablaDatos.appendChild(row);
  });
}

```

Es una función que se encarga de cargar y mostrar datos en una tabla, opcionalmente filtrados por DNI. La función toma un parámetro opcional dni, que por defecto es **null**.

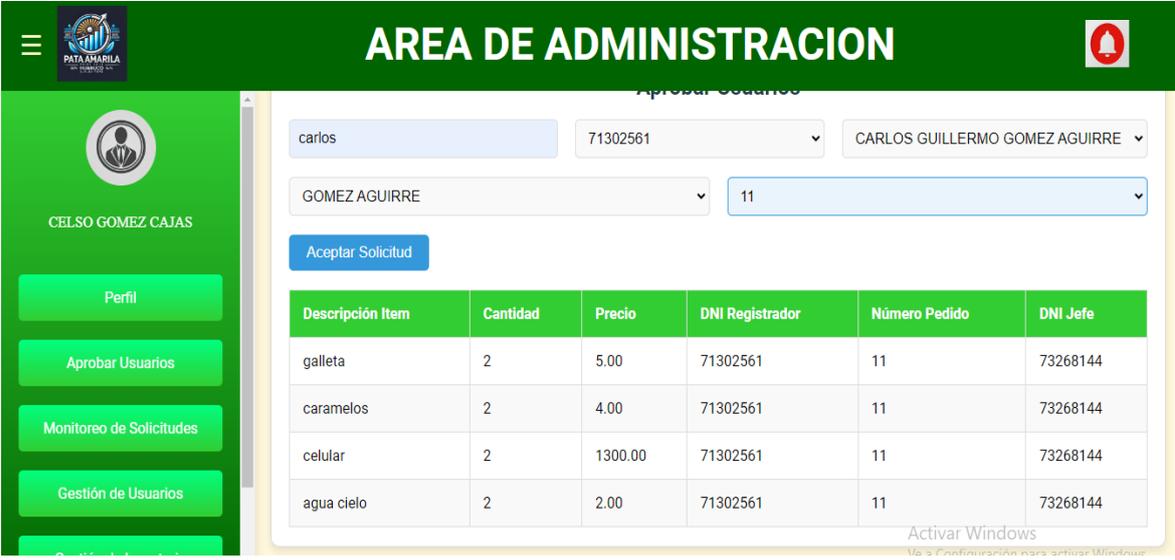
Primero, construye la URL de consulta (**query**) para la solicitud HTTP, que inicialmente es **/api/jefe_area**. Si se proporciona un dni, se agrega como parámetro de consulta a la URL (**query += ?dni=\${dni}**).

Luego, la función realiza una solicitud HTTP a la API utilizando **fetch**, pasando la URL de consulta como argumento. La palabra clave **await** se utiliza para esperar a que la promesa de **fetch** se resuelva y para convertir la respuesta a formato JSON, almacenando los datos recibidos en la constante **datos**. Estos datos se muestran en la consola para verificar su contenido.

La función vacía el contenido actual de la tabla (**tablaDatos.innerHTML = ''**) para asegurarse de que no haya datos previos antes de añadir los nuevos datos. Utiliza el método **forEach** para iterar sobre cada objeto de dato en el array **datos**.

Dentro del **forEach**, se crea una nueva fila de tabla (**row**) utilizando **document.createElement('tr')**. Se asigna el ID del registro al atributo **data-id** de la fila (**row.dataset.id = dato.id**). Luego, se construye el contenido HTML de la fila (**row.innerHTML**) utilizando los valores de cada campo en el objeto **dato** y se añade la fila a la tabla (**tablaDatos.appendChild(row)**).

La tabla para mostrar los datos quedaría de la siguiente manera.



The screenshot shows the 'AREA DE ADMINISTRACION' interface. On the left is a sidebar with a user profile for 'CELSO GOMEZ CAJAS' and navigation buttons: 'Perfil', 'Aprobar Usuarios', 'Monitoreo de Solicitudes', and 'Gestión de Usuarios'. The main area is titled 'Aprobar Solicitudes' and contains a form with the following fields:

- Text input: carlos
- DNI dropdown: 71302561
- Name dropdown: CARLOS GUILLERMO GOMEZ AGUIRRE
- Name dropdown: GOMEZ AGUIRRE
- Quantity dropdown: 11
- Button: Aceptar Solicitud

Below the form is a table with the following data:

Descripción Item	Cantidad	Precio	DNI Registrador	Número Pedido	DNI Jefe
galleta	2	5.00	71302561	11	73268144
caramelos	2	4.00	71302561	11	73268144
celular	2	1300.00	71302561	11	73268144
agua cielo	2	2.00	71302561	11	73268144

Función para aceptar las solicitudes

```

async function aceptarSolicitud() {
  // Verificar si la tabla tiene datos
  if (!tablaDatos.querySelectorAll('tr').length) {
    alert('No hay datos en la tabla para aceptar.');
```

```

    return;
  }

  const usuario = JSON.parse(localStorage.getItem('soledad'));
  const dniAdministrador = usuario ? usuario.dni : null;

  const datos = [...tablaDatos.querySelectorAll('tr')].map(row => {
    const cells = row.querySelectorAll('td');
    return {
      descripcion_item: cells[0].innerText,
      cantidad: cells[1].innerText,
      precio: cells[2].innerText,
      dni_registrador: cells[3].innerText,
      numero_pedido: cells[4].innerText,
      dni_jefe: cells[5].innerText,
      dni_administrador: dniAdministrador // Añadir el DNI del administrador
    };
  });

  try {
    const response = await fetch('/api/aprobacion_solicitud', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(datos)
    });

    if (response.ok) {
      alert('Solicitud aceptada');
      // Eliminar los registros después de aceptar la solicitud
      await eliminarSolicitudesPorNumeroPedido(datos.map(dato => dato.numero_pedido));
      cargarDatos(); // Recargar todos los datos después de eliminar
      // Limpiar comboboxes y campo de búsqueda
      limpiarComboboxes();
      searchUser.value = '';
    } else {
      alert('Error al aceptar la solicitud');
    }
  } catch (error) {
    console.error('Error al enviar la solicitud:', error);
    alert('Error al enviar la solicitud');
  }
}

```

Es una función asíncrona que maneja el proceso de aceptación de solicitudes y actualización de la tabla de datos. Primero, verifica si la tabla tiene datos utilizando **`tablaDatos.querySelectorAll('tr').length`**. Si la tabla está vacía, muestra una alerta y sale de la función (**`return`**).

Luego, obtiene los datos del usuario administrador desde **`localStorage`** y los almacena en el constante usuario. Si usuario existe, extrae el dni y lo guarda en `dniAdministrador`.

La función recopila los datos de todas las filas de la tabla usando **querySelectorAll** para seleccionar todas las filas (**tr**). Utiliza el método **map** para crear un array de objetos con los datos necesarios, extrayendo los valores de cada celda (**td**) de cada fila. Para cada fila, crea un objeto con las propiedades **descripcion_item**, **cantidad**, **precio**, **dni_registrador**, **numero_pedido** y **dni_jefe**. También añade **dni_administrador**, que es el dni del administrador que está aceptando la solicitud.

Dentro del bloque **try**, la función realiza una solicitud HTTP POST a la API en la ruta **/api/aprobacion_solicitud**. La solicitud incluye un encabezado **Content-Type** que especifica que el contenido de la solicitud es JSON, y el cuerpo de la solicitud contiene los datos convertidos a formato JSON usando **JSON.stringify(datos)**. La palabra clave **await** se utiliza para esperar a que la promesa de **fetch** se resuelva.

Si la solicitud se realiza con éxito (**response.ok**), muestra una alerta indicando que la solicitud ha sido aceptada. Luego, llama a **eliminarSolicitudesPorNumeroPedido** para eliminar los registros aceptados, pasando los números de pedido como argumento. La función **cargarDatos** se llama para recargar la tabla con los datos actualizados. También se limpian los comboboxes y el campo de búsqueda.

Si la solicitud falla, muestra una alerta indicando que hubo un error al aceptar la solicitud. Cualquier error que ocurra dentro del bloque **try** se captura en el bloque **catch**, y se registra en la consola, mostrando una alerta adicional para informar al usuario del error.

Función para eliminar solicitudes por número de boleta

```

async function eliminarSolicitudesPorNumeroPedido(numerosPedido) {
  // Verificar si la tabla tiene datos
  if (!tablaDatos.querySelectorAll('tr').length) {
    console.log('La tabla está vacía. No se ejecutará la eliminación.');
```

```

    return;
  }

  try {
    const response = await fetch('/api/jefe_area', {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ numerosPedido })
    });

    if (response.ok) {
      alert('Solicitudes eliminadas');
      cargarDatos(); // Recargar todos los datos después de eliminar
    } else {
      alert('Error al eliminar las solicitudes');
    }
  } catch (error) {
    console.error('Error al eliminar las solicitudes:', error);
    alert('Error al eliminar las solicitudes');
  }
}

```

La función se encarga de eliminar solicitudes específicas basadas en sus números de pedido. Primero, verifica si la tabla tiene datos utilizando `tablaDatos.querySelectorAll('tr').length`. Si la tabla está vacía, registra un mensaje en la consola (`console.log('La tabla está vacía. No se ejecutará la eliminación.')`) y sale de la función con `return`.

Dentro de un bloque `try`, la función realiza una solicitud HTTP DELETE a la API en la ruta `/api/jefe_area`. La solicitud incluye un encabezado `Content-Type` que especifica que el contenido de la solicitud es JSON, y el cuerpo de la solicitud contiene los números de pedido convertidos a formato JSON usando `JSON.stringify({ numerosPedido })`. La palabra clave `await` se utiliza para esperar a que la promesa de `fetch` se resuelva.

Si la solicitud se realiza con éxito (`response.ok`), muestra una alerta indicando que las solicitudes han sido eliminadas y llama a la función para recargar la tabla con los datos actualizados. Esto asegura que la interfaz de usuario se actualice para reflejar los cambios realizados. Si la solicitud falla, muestra una alerta indicando que hubo un error al eliminar las solicitudes.

Cualquier error que ocurra dentro del bloque `try` se captura en el bloque `catch`, y se registra en la consola (`console.error('Error al eliminar las solicitudes:', error)`) y se muestra una alerta adicional para informar al usuario del error.

La base de datos donde se almacenan los datos luego de ser aceptados

```
-- Estructura de tabla para la tabla `aprobacion_solicitus`
CREATE TABLE `aprobacion_solicitus` (
  `id` int(11) NOT NULL,
  `dni_registrador` varchar(20) NOT NULL,
  `numero_pedido` varchar(50) NOT NULL,
  `descripcion_item` varchar(255) NOT NULL,
  `cantidad` int(11) NOT NULL,
  `precio` decimal(10,2) NOT NULL,
  `dni_administrador` varchar(20) NOT NULL,
  `createdAt` timestamp NOT NULL DEFAULT current_timestamp(),
  `updatedAt` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Esta es la base de datos donde se registra todos los bienes y suministros que son aceptados por administracio y para ello es importante todos los datos que se están mostrando en la tabla.

➤ Funciones del área de logística



Con los códigos de CSS y HTML se formó lo que se observa y en el código se incluye varios elementos importantes como el encabezado, la barra lateral de navegación y la sección principal de contenido.

Encabezado: El encabezado (`<header class="header">`) es fijo y se encuentra en la parte superior de la página. Incluye un título centrado y dos secciones para íconos y botones de navegación, ubicadas a la izquierda y a la derecha.

Barra Lateral: La barra lateral (`<aside class="sidebar">`) contiene el perfil del usuario y botones de navegación para acceder a diferentes secciones de la página de logística. Los botones de navegación permiten a los usuarios moverse entre las diferentes secciones, como "Órdenes de Compra" y "Solicitudes".

Contenido Principal: La sección principal (`<main class="main-content">`) está diseñada para mostrar el contenido específico seleccionado por el usuario, como las órdenes de compra o las solicitudes. Las diferentes secciones de contenido se muestran u ocultan según la navegación del usuario.

Funciones para que se ejecuten automáticamente

```
document.addEventListener('DOMContentLoaded', () => {
  const buscador = document.getElementById('buscador');
  const selectDNI = document.getElementById('selectDNI');
  const selectNumeroPedido = document.getElementById('selectNumeroPedido');
  const tablaSolicitudes = document.getElementById('tablaSolicitudes').querySelector('tbody');
  const aceptarCompra = document.getElementById('aceptarCompra');
```

La función encapsulada por `document.addEventListener('DOMContentLoaded', () => { ... })` se asegura de que todo el código dentro de ella se ejecute solo después de que el DOM se haya cargado completamente. Esto es

importante para asegurar que todos los elementos del DOM a los que el código hace referencia estén disponibles.

Primero, se selecciona el elemento del DOM con el ID buscador y se asigna a la constante buscador. Este elemento es probablemente un campo de entrada donde el usuario puede escribir texto para buscar solicitudes. Luego, se selecciona el elemento del DOM con el ID selectDNI y se asigna a la constante selectDNI, que es un combobox (**select**) donde el usuario puede seleccionar un DNI específico.

Asimismo, se selecciona el elemento del DOM con el ID selectNumeroPedido y se asigna a la constante selectNumeroPedido, que es otro combobox donde el usuario puede seleccionar un número de pedido específico. A continuación, se selecciona el elemento del DOM con el ID tablaSolicitudes y luego se selecciona su elemento tbody, asignándolo a la constante tablaSolicitudes. Este tbody es donde se añadirán las filas de datos de las solicitudes. Finalmente, se selecciona el elemento del DOM con el ID aceptarCompra y se asigna a la constante aceptarCompra, que es probablemente un botón que, cuando se hace clic, acepta las compras seleccionadas.

Funciones para cargar toda la interfaz gráfica que se muestra en pantalla

```

async function fetchSolicitudes(query = '') {
  const response = await fetch(`/api/logistica/aprobaciones?query=${query}`);
  const data = await response.json();
  return data;
}

```

Se encarga de obtener las solicitudes de logística desde una API. La función acepta un parámetro opcional **query**, que se utiliza para filtrar las solicitudes basadas en una consulta específica. El valor por defecto de **query** es una cadena **vacía ("**), lo que significa que, si no se proporciona ningún valor, se utilizará una consulta vacía.

Dentro de la función, se realiza una solicitud HTTP a la API utilizando **fetch**, pasando la URL de la API junto con el parámetro de consulta **query**. La palabra clave **await** se utiliza para esperar a que la promesa de fetch se resuelva, es decir, hasta que se obtenga una respuesta del servidor. La respuesta se almacena en la constante response.

Después de recibir la respuesta, se utiliza **response.json()** para convertir la respuesta a formato JSON. La palabra clave **await** se usa nuevamente para esperar a que esta operación se complete. El resultado de esta conversión se almacena en la constante data.

Finalmente, la función devuelve los datos obtenidos (data). Esto permite que cualquier parte del código que llame a **fetchSolicitudes** pueda acceder a las solicitudes de logística obtenidas desde la API.

```

async function updateSelectDNI(query = '') {
  const solicitudes = await fetchSolicitudes(query);
  selectDNI.innerHTML = '<option value="">Seleccionar DNI</option>';
  const dniUnicos = [...new Set(solicitudes.map(solicitud => solicitud.dni_registrador))];
  dniUnicos.forEach(dni => {
    const option = document.createElement('option');
    option.value = dni;
    option.textContent = dni;
    selectDNI.appendChild(option);
  });
}

```

Se encarga de actualizar el combobox (**select**) que permite seleccionar un DNI en la interfaz de usuario. La función acepta un parámetro opcional `query`, que se utiliza para filtrar las solicitudes basadas en una consulta específica. El valor por defecto de `query` es una cadena **vacía** (`''`).

Primero, la función llama a **fetchSolicitudes(query)** para obtener las solicitudes desde la API, utilizando el valor de `query` para filtrar los resultados. La palabra clave `await` se utiliza para esperar a que la promesa de **fetchSolicitudes** se resuelva y devuelva los datos. Estos datos se almacenan en la constante `solicitudes`.

Luego, la función establece el contenido HTML del combobox **selectDNI** con una opción por defecto que dice "Seleccionar DNI". Esto asegura que el combobox se reinicie y contenga una opción predeterminada.

A continuación, la función crea un array de los dni únicos utilizando `new Set` y **map**. **solicitudes.map(solicitud => solicitud.dni_registrador)** crea un array con todos los dni de los registradores en las solicitudes. `new Set` se utiliza para eliminar duplicados, y el resultado se convierte nuevamente en un array utilizando el operador de propagación (`...`).

Después, la función utiliza **forEach** para iterar sobre cada DNI único en el array **dniUnicos**. Para cada DNI, se crea un nuevo elemento opción utilizando **document.createElement('option')**. El valor (**value**) y el contenido de texto (**textContent**) de cada opción se establecen en el DNI actual. Finalmente, cada opción se añade al combobox `selectDNI` utilizando **selectDNI.appendChild(option)**.

```

async function updateSelectNumeroPedido(dni = '') {
  const solicitudes = await fetchSolicitudes(dni);
  selectNumeroPedido.innerHTML = '<option value="">Seleccionar Número de Pedido</option>';
  const numerosPedidoUnicos = [...new Set(solicitudes.map(solicitud => solicitud.numero_pedido))];
  numerosPedidoUnicos.forEach(numero => {
    const option = document.createElement('option');
    option.value = numero;
    option.textContent = numero;
    selectNumeroPedido.appendChild(option);
  });
}

```

La función se encarga de actualizar el combobox de números de pedido en la interfaz de usuario utilizando un DNI específico para filtrar las solicitudes. Cuando se proporciona un DNI, la función obtiene las solicitudes relacionadas mediante una llamada a la API y actualiza las opciones del combobox para reflejar solo los números de pedido relevantes.

Primero, la función llama a **fetchSolicitudes(dni)**, que recupera las solicitudes de la API y devuelve los datos en formato JSON. Usando **await**, la función espera a que esta operación se complete y almacena los datos recibidos en la constante `solicitudes`.

Luego, la función reinicia el combobox de números de pedido estableciendo una opción predeterminada que dice "**Seleccionar Número de Pedido**". Esto limpia cualquier opción previa en el combobox.

Para asegurar que solo se muestren números de pedido únicos, la función utiliza **new Set** junto con **map** para extraer los números de pedido de las solicitudes y eliminar duplicados. El resultado es una lista de números de pedido únicos.

La función itera sobre cada número de pedido único utilizando **forEach**. Para cada número, crea un nuevo elemento opción, establece su valor y su contenido de texto con el número de pedido correspondiente, y luego lo añade al combobox.

```

async function updateTable(query = '', dni = '', numeroPedido = '') {
  const solicitudes = await fetchSolicitudes(query);
  const filteredSolicitudes = solicitudes.filter(solicitud =>
    (dni === '' || solicitud.dni_registrador === dni) &&
    (numeroPedido === '' || solicitud.numero_pedido === numeroPedido)
  );
  tablaSolicitudes.innerHTML = '';
  filteredSolicitudes.forEach((solicitud, index) => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${solicitud.id}</td>
      <td>${solicitud.dni_registrador}</td>
      <td>${solicitud.numero_pedido}</td>
      <td>${solicitud.descripcion_item}</td>
      <td>${solicitud.cantidad}</td>
      <td>${solicitud.precio}</td>
      <td>${solicitud.dni_administrador}</td>
      <td>${solicitud.createdAt ? new Date(solicitud.createdAt).toLocaleString() : ''}</td>
    `;
    tablaSolicitudes.appendChild(row);
  });
}

```

La función **updateTable** es una función asíncrona que se encarga de actualizar la tabla de solicitudes en la interfaz de usuario, filtrando los datos según los parámetros opcionales `query`, `dni`, y **numeroPedido**.

Primero, la función llama a **fetchSolicitudes(query)** para obtener las solicitudes desde la API. Utilizando `await`, se asegura de esperar a que la promesa se resuelva y devuelva los datos, que se almacenan en la constante `solicitudes`.

Luego, la función filtra las solicitudes utilizando el método **filter**. Este método crea un nuevo **array**, **filteredSolicitudes**, que solo incluye las solicitudes que cumplen con los criterios especificados: si dni es una cadena vacía o coincide con el dni_registrador de la solicitud, y si numeroPedido es una cadena vacía o coincide con el numero_pedido de la solicitud.

La función entonces vacía el contenido actual del cuerpo de la tabla de solicitudes estableciendo **tablaSolicitudes.innerHTML** a una cadena vacía. Esto asegura que cualquier contenido anterior en la tabla se elimine antes de añadir nuevas filas.

A continuación, la función itera sobre cada solicitud en **filteredSolicitudes** utilizando **forEach**. Para cada solicitud, crea un nuevo elemento de **fila (tr)** con **document.createElement('tr')** y construye el contenido HTML de la fila utilizando las propiedades de la solicitud. El contenido de la fila incluye varias **celdas (td)** que muestran el ID, el DNI del registrador, el número de pedido, la descripción del ítem, la cantidad, el precio, el DNI del administrador y la fecha de creación formateada. La fecha de creación se formatea utilizando **new Date(solicitud.createdAt).toLocaleString()** si está disponible.

Finalmente, la nueva fila se añade al cuerpo de la tabla (**tablaSolicitudes**) utilizando **appendChild(row)**.

Los datos de toda la tabla se muestran en la siguiente imagen.

Solicitudes

carlos

71302561 10 Aceptar Compra

ID	DNI Registrador	Número Pedido	Descripción Item	Cantidad	Precio	DNI Administrador	Fecha Creación
227	71302561	10	galleta	1	5.00	71305795	19/7/2024, 8:21:29 p. m.
228	71302561	10	caramelos	1	4.00	71305795	19/7/2024, 8:21:29 p. m.
229	71302561	10	celular	1	1300.00	71305795	19/7/2024, 8:21:29 p. m.
230	71302561	10	agua cielo	1	2.00	71305795	19/7/2024, 8:21:29 p. m.

Activar Windows
Ve a Configuración para activar Windows

Esta función es para verificar el stock y aceptar la solicitud

```

aceptarCompra.addEventListener('click', async () => {
  const solicitudes = [...tablaSolicitudes.querySelectorAll('tr')].map(row => {
    const cells = row.querySelectorAll('td');
    return {
      id: cells[0].textContent,
      dni_registrador: cells[1].textContent,
      numero_pedido: cells[2].textContent,
      descripcion_item: cells[3].textContent,
      cantidad: parseInt(cells[4].textContent),
      precio: parseFloat(cells[5].textContent),
      dni_administrador: cells[6].textContent
    };
  });

  if (solicitudes.length === 0) {
    alert('No hay solicitudes para procesar.');
```

```

    return;
  }

  try {
    const response = await fetch('/api/logistica/verificar_items', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(solicitudes)
    });

    if (!response.ok) {
      const errorText = await response.text();
      throw new Error(errorText);
    }

    const result = await response.json();
    if (result.success) {
      if (confirm('Tienes los items necesarios. ¿Aceptar o cancelar?')) {
        const actualizarResponse = await fetch('/api/logistica/actualizar_items', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify(solicitudes)
        });

        if (!actualizarResponse.ok) {
          const errorText = await actualizarResponse.text();
          throw new Error(errorText);
        }
      }
    }
  }
}

```

Activar Window

```

        const actualizarResult = await actualizarResponse.json();
        if (actualizarResult.success) {
            alert('Compra realizada con éxito.');
```

```

            await updateTable();
        } else {
            alert('Error al realizar la compra.');
```

```

        }
    } else {
        alert('No tienes los items necesarios: ' + result.faltantes.join(', '));
    }
} catch (error) {
    console.error('Error:', error);
    alert('Error al verificar los items: ' + error.message);
}
});

```

El código proporcionado maneja el evento de clic en el botón aceptar Compra y realiza una serie de operaciones asíncronas para procesar las solicitudes de compra.

Cuando se hace clic en el botón **aceptarCompra**, se ejecuta una función asíncrona. Primero, esta función recopila todas las filas (**<tr>**) de la tabla de solicitudes (**tablaSolicitudes**) y mapea cada fila para extraer sus celdas (**<td>**). Utilizando **querySelectorAll('tr')**, se seleccionan todas las filas, y con **map**, se itera sobre cada fila para construir un objeto que representa una solicitud. Este objeto contiene los valores de texto (**textContent**) de las celdas correspondientes a los atributos **id**, **dni_registrador**, **numero_pedido**, **descripcion_item**, **cantidad**, **precio** y **dni_administrador**. Las cantidades y precios se convierten a números usando **parseInt** y **parseFloat**, respectivamente.

Si no hay solicitudes en la tabla (es decir, la longitud de solicitudes es 0), se muestra una alerta indicando "No hay solicitudes para procesar" y la función termina con **return**.

Luego, la función intenta enviar las solicitudes al servidor para verificar si los ítems necesarios están disponibles. Esto se hace mediante una solicitud **POST** a la ruta **/api/logistica/verificar_items**, pasando las solicitudes en el cuerpo de la solicitud formateadas como **JSON (JSON.stringify(solicitudes))**. Se utilizan **await** y **fetch** para esperar la respuesta del servidor. Si la respuesta no es exitosa (**!response.ok**), se obtiene el texto del error de la respuesta y se lanza una excepción con ese mensaje.

Si la verificación es exitosa y el servidor indica que los ítems están disponibles (**result.success**), se muestra una confirmación al usuario preguntando si desea aceptar o cancelar. Si el usuario acepta, se envía otra solicitud **POST** a la ruta **/api/logistica/actualizar_items** para actualizar los ítems, pasando las solicitudes nuevamente en el cuerpo de la solicitud. Si la actualización es exitosa

(actualizarResult.success), se muestra una alerta indicando que la compra se realizó con éxito y se llama a **updateTable()** para recargar la tabla. Si hay un error en la actualización, se muestra una alerta indicando **"Error al realizar la compra"**.

Si la verificación inicial del servidor indica que faltan ítems **(result.success es false)**, se muestra una alerta listando los ítems faltantes.

En el caso de que ocurra cualquier error durante estas operaciones, se captura en el bloque `catch` y se muestra una alerta con el mensaje de error, además de registrar el error en la consola para depuración.

Este código maneja de manera completa y eficiente el proceso de verificación y actualización de solicitudes de compra, asegurando que los usuarios sean notificados adecuadamente de cualquier problema o éxito en el proceso. Utiliza varias técnicas importantes como **async**, **await**, **fetch**, **map**, y manejo de eventos, proporcionando una interacción de usuario fluida y robusta.

Funciones para la conexión con la base de datos de logística

```
const express = require('express');
const router = express.Router();
const { sequelize } = require('../config/dbConfig');
const AprobacionSolicitud = require('../models/logi_AprobacionSolicitud');
const Historial = require('../models/logi_Historial');
const Item = require('../models/logi_Item');

// Ruta para obtener aprobaciones de solicitudes
router.get('/aprobaciones', async (req, res) => {
  try {
    const aprobaciones = await AprobacionSolicitud.findAll();
    res.json(aprobaciones);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

El código mostrado define una ruta en un servidor Express para manejar solicitudes GET a **/aprobaciones**. Esta ruta está diseñada para obtener todas las aprobaciones de solicitudes almacenadas en la base de datos y devolverlas en formato **JSON**.

Primero, se importan las dependencias necesarias utilizando **require**. Se importa **Express** y se crea un **router** con **express.Router()**. Luego, se importan **sequelize** para manejar la conexión a la base de datos y los modelos **AprobacionSolicitud**, **Historial**, e **Item** que representan diferentes tablas en la base de datos.

La ruta para obtener aprobaciones de solicitudes se define con **router.get('/aprobaciones', async (req, res) => { ... })**. Aquí se utiliza la palabra clave **async** para indicar que la función es asíncrona y podrá usar **await** dentro de ella.

Dentro de la función, se utiliza un bloque **try-catch** para manejar posibles errores durante la operación. En el bloque **try**, se llama a **AprobacionSolicitud.findAll()**, que es un método de **Sequelize** para obtener todos los registros de la tabla **AprobacionSolicitud**. La palabra clave **await** se utiliza para esperar a que la promesa de **findAll** se resuelva y devuelva los datos. Estos datos se almacenan en la constante **aprobaciones**.

Luego, los datos obtenidos se envían como respuesta en formato **JSON** utilizando **res.json(aprobaciones)**. Si ocurre un error durante la operación, el bloque **catch** captura el error y responde con un código de estado 500 y un mensaje de error en formato **JSON** (**res.status(500).json({ error: error.message })**). Esto asegura que cualquier problema durante la operación sea manejado adecuadamente y comunicado al cliente.

Tabla de la base de datos para que las solicitudes queden archivados

```
-- Estructura de tabla para la tabla `historial`
--
CREATE TABLE `historial` (
  `id` int(11) NOT NULL,
  `dni_registrador` varchar(50) NOT NULL,
  `numero_pedido` varchar(50) NOT NULL,
  `descripcion_item` varchar(255) NOT NULL,
  `cantidad` int(11) NOT NULL,
  `precio` decimal(10,2) NOT NULL,
  `dni_administrador` varchar(50) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Generación de la orden de compra

The screenshot shows a web application interface for 'ÁREA LOGÍSTICA'. The main heading is 'Órdenes de Compra'. Below this, there are four form fields: 'REGIÓN:' with a dropdown menu showing 'Amazonas', 'PROVINCIA:' with an empty dropdown, 'DISTRITO:' with an empty dropdown, and 'DIRECCIÓN:' with a text input field. At the bottom of the form, there are two buttons: 'Ver Carrrito' and 'Generar Orden de Compra'. In the bottom right corner, there is a small notification: 'Activar Windows. Ve a Configuración para activar Windows.'

```

async function fetchRegiones() {
  const response = await fetch('/api/orden/regiones');
  return response.json();
}

async function fetchProvincias(regionId) {
  const response = await fetch(`/api/orden/provincias/${regionId}`);
  return response.json();
}

async function fetchDistritos(provinciaId) {
  const response = await fetch(`/api/orden/distritos/${provinciaId}`);
  return response.json();
}

async function fetchProductos(query = '') {
  const url = `/api/orden/productos${query ? `?query=${query}` : ''}`;
  const response = await fetch(url);
  return response.json();
}

function renderOptions(selectElement, options) {
  selectElement.innerHTML = '';
  options.forEach(option => {
    const opt = document.createElement('option');
    opt.value = option.id;
    opt.textContent = option.nombre;
    selectElement.appendChild(opt);
  });
  selectElement.disabled = options.length === 0;
}

async function updateProvincias() {
  const regionId = regionSelect.value;
  if (regionId) {
    const provincias = await fetchProvincias(regionId);
    renderOptions(provinciaSelect, provincias);
    distritoSelect.innerHTML = '';
    distritoSelect.disabled = true;
  }
}

async function updateDistritos() {
  const provinciaId = provinciaSelect.value;
  if (provinciaId) {
    const distritos = await fetchDistritos(provinciaId);
    renderOptions(distritoSelect, distritos);
  }
}

```

```

function renderProductos(productos) {
  tablaProductos.innerHTML = '';
  productos.forEach(producto => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${producto.id}</td>
      <td>${producto.nombre}</td>
      <td>${producto.categoria}</td>
      <td>${producto.precio}</td>
      <td>${producto.cantidad}</td>
      <td>${producto.unidad_de_medida}</td>
      <td>${producto.dni_proveedor}</td>
      <td>
        <button onclick="seleccionarProducto(${producto.id},
          '${producto.nombre}', ${producto.precio}, '${producto.dni_proveedor}')">Seleccionar</button>
      </td>
    `;
    tablaProductos.appendChild(row);
  });
}

```

Las funciones proporcionadas son responsables de manejar la obtención y renderización de datos relacionados con regiones, provincias, distritos y productos en una interfaz de usuario. Aquí está la explicación detallada de cómo funcionan cada una de estas funciones y los conceptos clave que utilizan.

Primero, las funciones **fetchRegiones**, **fetchProvincias**, **fetchDistritos**, y **fetchProductos** son funciones asíncronas (**async**) que realizan solicitudes a la API utilizando **fetch** para obtener datos en formato JSON. La palabra clave **await** se usa para esperar a que la promesa de **fetch** se resuelva antes de continuar con la ejecución del código. Esto asegura que los datos se obtengan correctamente antes de **proceder**. **fetch** es una API nativa de **JavaScript** que permite hacer solicitudes **HTTP**. **response.json()** convierte la respuesta HTTP en un objeto JavaScript, lo que facilita el manejo de los datos.

La función **renderOptions** se encarga de poblar un elemento **<select>** con opciones. Primero, limpia el contenido previo del elemento select (**selectElement.innerHTML = ''**). Luego, itera sobre las opciones recibidas, crea un nuevo elemento **<option>** para cada una (**document.createElement('option')**), establece sus valores (**opt.value = option.id; opt.textContent = option.nombre;**), y los añade al select (**selectElement.appendChild(opt);**). También deshabilita el **select** si no hay opciones disponibles (**selectElement.disabled = options.length === 0;**).

Las funciones **updateProvincias** y **updateDistritos** se utilizan para actualizar los selectores de provincias y distritos cuando el usuario selecciona una región o provincia, respectivamente. **updateProvincias** obtiene el ID de la región seleccionada (**regionSelect.value**). Si hay un ID válido, llama a **fetchProvincias(regionId)** para

obtener las provincias correspondientes, luego usa **renderOptions** para actualizar el **provinciaSelect** y limpia el **distritoSelect**. De manera similar, **updateDistritos** obtiene el ID de la provincia seleccionada, llama a **fetchDistritos(provinciaId)**, y usa **renderOptions** para actualizar el **distritoSelect**.

Finalmente, la función **renderProductos** se encarga de mostrar los productos en una **tabla (tablaProductos)**. Primero, limpia el contenido de la tabla (**tablaProductos.innerHTML = ""**). Luego, itera sobre los productos, creando **filas (<tr>)** para cada producto. Cada fila contiene **celdas (<td>)** con los detalles del producto y un botón para seleccionar el producto (**<button onclick="seleccionarProducto(...)">**). Estas filas se añaden al cuerpo de la tabla (**tablaProductos.appendChild(row)**).

Realizar la lista de bienes y servicios para generar la orden de compra

```
function renderCarrito() {
  let carritoHTML = `
    <table style="width: 100%; border-collapse: collapse;">
      <thead>
        <tr>
          <th>ID</th>
          <th>Nombre</th>
          <th>Precio</th>
          <th>Cantidad</th>
          <th>DNI Proveedor</th>
          <th>Acción</th>
        </tr>
      </thead>
      <tbody>
        ;

        carrito.forEach(producto => {
          carritoHTML += `
            <tr>
              <td>${producto.id}</td>
              <td>${producto.nombre}</td>
              <td>${producto.precio}</td>
              <td>${producto.cantidad}</td>
              <td>${producto.dni_proveedor}</td>
              <td><button onclick="window.opener.eliminarProducto(${producto.id});
                window.close();">Eliminar</button></td>
            </tr>
          `;
        });
      </tbody>
    </table>
  `;
}
```

```

carritoHTML += `
  </tbody>
</table>
`;
const left = (screen.width - 800) / 2;
const top = (screen.height - 600) / 2;
const carritoWindow = window.open("", "Carrito de Compras", `width=800,height=600,top=${top},left=${left}`)
carritoWindow.document.write(`
  <html>
  <head>
    <title>Carrito de Compras</title>
    <style>
      body { font-family: Arial, sans-serif; text-align: center; }
      table { width: 100%; border-collapse: collapse; margin-top: 20px; }
      th, td { border: 1px solid #000; padding: 8px; text-align: left; }
      th { background-color: #f2f2f2; }
      button { background-color: red; color: white; border: none; padding: 5px 10px; cursor: pointer; }
      button:hover { background-color: darkred; }
    </style>
  </head>
  <body>
    <h1>Carrito de Compras</h1>
    ${carritoHTML}
    <button onclick="window.close()">Cerrar</button>
  </body>
  </html>
`);
carritoWindow.document.close();
}

```

La función **renderCarrito** se encarga de mostrar el contenido del carrito de compras en una nueva ventana emergente. Este proceso se realiza en varias etapas clave. Primero, la función construye el HTML necesario para una tabla que contendrá los productos del carrito. Utiliza una plantilla de cadena (**template string**) para definir la estructura básica de la tabla, incluyendo el encabezado de la tabla con columnas para ID, Nombre, Precio, Cantidad, DNI del Proveedor y una columna para las acciones disponibles.

Dentro de la función, se itera sobre cada producto en el carrito utilizando el método `forEach`. Para cada producto, se agrega una fila a la tabla. Cada fila contiene celdas que muestran el ID, nombre, precio, cantidad y DNI del proveedor del producto. Además, se incluye un botón "Eliminar" en cada fila. Este botón, al ser clicado, llama a la función **eliminarProducto** definida en la ventana principal (**window.opener**) y luego cierra la ventana emergente. Esto permite al usuario eliminar productos del carrito directamente desde la ventana emergente.

Después de iterar sobre todos los productos, la función cierra la etiqueta del cuerpo de la tabla y completa la estructura de la tabla en la cadena de HTML. A continuación, la función calcula las posiciones para centrar la ventana emergente en la pantalla. Esto se logra restando el ancho y alto de la ventana emergente del ancho y alto de la pantalla, respectivamente, y dividiendo los resultados por dos. Estas coordenadas aseguran que la ventana se abra centrada.

La función procede a abrir una nueva ventana utilizando **window.open**, especificando las dimensiones y la posición de la ventana emergente. La nueva ventana se crea con un nombre ("Carrito de Compras") y se configuran sus dimensiones y posición en la pantalla para que esté centrada.

Dentro de la nueva ventana, la función escribe el HTML necesario para mostrar el carrito de compras. Esto incluye una estructura básica de HTML con etiquetas HTML, head y body. En el head, se define el título de la página ("Carrito de Compras") y se incluyen estilos en línea para dar formato a la tabla y los botones. Los estilos aseguran que la tabla tenga un ancho del 100%, bordes colapsados y márgenes superiores, y que los botones tengan un diseño coherente y atractivo. Por ejemplo, los estilos definen que los bordes de las celdas sean colapsados y que el fondo de los encabezados de la tabla sea de color gris claro, mientras que los botones son rojos con texto blanco y cambian de color al pasar el cursor sobre ellos.

En el body, se incluye un encabezado ("Carrito de Compras"), la tabla generada dinámicamente con los productos del carrito, y un botón "Cerrar" que permite al usuario cerrar la ventana emergente. La función utiliza **document.write** para insertar este HTML en la nueva ventana y luego cierra el flujo de escritura con **document.close**.

Esta función crea y muestra una ventana emergente que contiene una tabla detallada de los productos en el carrito de compras. Utiliza técnicas de manipulación del DOM y gestión de ventanas emergentes para proporcionar una experiencia de usuario interactiva y dinámica. La función facilita la visualización y gestión del carrito de compras, permitiendo agregar, eliminar y revisar productos de manera eficiente.

Conceptos importantes utilizados en la función incluyen `forEach`, que es un método utilizado para iterar sobre cada producto en el carrito, y **window.opener**, que es una referencia a la ventana principal que abrió la ventana emergente. **window.open** es el método utilizado para abrir una nueva ventana del navegador, mientras que `document.write` se usa para escribir contenido HTML en la nueva ventana y **document.close** cierra el flujo de escritura del documento. Las plantillas de cadena (**template strings**) permiten la creación de cadenas de texto complejas y **multi-línea** de manera más legible y manejable. Por último, **screen.width** y **screen.height** son propiedades que devuelven el ancho y alto de la pantalla del usuario, utilizadas para centrar la ventana emergente.

Seleccionar bienes y suministros para la lista que se va ordenar

```
function seleccionarProducto(producto) {
  const cantidad = parseInt(prompt('Ingrese la cantidad deseada:'), 10);
  if (!cantidad || isNaN(cantidad)) {
    alert('Cantidad no válida');
    return;
  }

  const productoExistente = carrito.find(item => item.id === producto.id);
  if (productoExistente) {
    productoExistente.cantidad += cantidad;
  } else {
    carrito.push({ ...producto, cantidad });
  }

  localStorage.setItem('carrito_orden', JSON.stringify(carrito));
  console.log('Producto seleccionado:', producto);
}
```

La función se encarga de agregar un producto al carrito de compras. Primero, solicita al usuario que ingrese la cantidad deseada del producto mediante un cuadro de diálogo. Este cuadro de diálogo, conocido como **prompt**, permite al usuario introducir un valor que luego se convierte a un número entero utilizando **parseInt**. Este método asegura que el valor ingresado sea tratado como un número, lo que es esencial para las operaciones matemáticas que siguen.

Después de obtener la cantidad deseada, la función verifica si la cantidad ingresada es válida. Utiliza **isNaN** para comprobar si el valor no es un número. Si la cantidad es inválida, la función muestra una alerta indicando que la cantidad no es válida y luego interrumpe su ejecución mediante **return**. Esta validación es crucial para evitar errores en el procesamiento de datos no numéricos.

A continuación, la función busca en el carrito si el producto ya existe. Utiliza el método **find** del **array**, que recorre el carrito buscando un producto con el mismo ID que el producto actual. Si encuentra un producto existente, simplemente incrementa su cantidad sumando la cantidad ingresada a la cantidad actual del producto en el carrito. Este enfoque optimiza la gestión del carrito al evitar la duplicación de productos con el mismo ID.

Si el producto no existe en el carrito, la función lo agrega como un nuevo objeto. Este nuevo objeto contiene todas las propiedades del producto y la cantidad ingresada. Para ello, utiliza el método **push del array**, que añade elementos al final del **array**. Esta acción asegura que el carrito mantenga una estructura coherente y actualizada con los productos seleccionados por el usuario.

Finalmente, la función actualiza el carrito almacenado en el **localStorage** del navegador. Utiliza el método **setItem** de **localStorage** para guardar el carrito actualizado, convirtiéndolo primero en una cadena JSON mediante **JSON.stringify**. Este paso es importante porque **localStorage** solo puede almacenar cadenas de texto. Al convertir el carrito a una cadena JSON, se asegura que la estructura del objeto se preserve correctamente al guardarlo.

Para fines de depuración, la función también registra en la consola el producto seleccionado. Esto es útil para los desarrolladores, ya que proporciona una forma de verificar que los productos se están agregando correctamente al carrito.

Enviar la orden de compra

```

async function generarOrdenCompra() {
  const direccion = document.getElementById('direccion').value;
  const dni_cliente = '71885493'; // Valor definido del DNI del cliente
  const regionSelect = document.getElementById('region');
  const provinciaSelect = document.getElementById('provincia');
  const distritoSelect = document.getElementById('distrito');
  const dni_logistica = '12345678'; // Valor definido del DNI de logística
  const carrito = JSON.parse(localStorage.getItem('carrito_orden')) || [];

  if (!direccion || regionSelect.value === "" || provinciaSelect.value === "" || distritoSelect.value === "") {
    alert('Complete todos los campos requeridos');
    return;
  }

  if (carrito.length === 0) {
    alert('El carrito está vacío.');
```

```

    return;
  }

  const region = regionSelect.selectedOptions[0].text;
  const provincia = provinciaSelect.selectedOptions[0].text;
  const distrito = distritoSelect.selectedOptions[0].text;

  const productos = carrito.map(item => {
    return {
      id: item.id,
      cantidad: item.cantidad
    };
  });
}

```

```

try {
  const response = await fetch('/api/orden/crear', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ direccion, productos, dni_cliente, region, provincia, distrito, dni_logistica })
  });

  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }

  const nuevaOrden = await response.json();
  console.log('Orden generada:', nuevaOrden);

  localStorage.removeItem('carrito_orden');
  alert('Orden generada exitosamente.');
```

La función está diseñada para crear una nueva orden de compra basada en los datos ingresados por el usuario y los productos almacenados en el carrito de compras. Esta función realiza varias tareas clave, desde la validación de los campos del formulario hasta el envío de una solicitud a la API para crear la orden. Primero, la función obtiene los valores de varios elementos del DOM, como la dirección, la región, la provincia y el distrito seleccionados por el usuario. Además, se definen dos valores fijos: el DNI del cliente y el DNI de logística. Estos valores son necesarios para identificar al cliente y al responsable de logística en la orden de compra. La función también recupera el contenido del carrito de compras desde el **localStorage**, convirtiéndolo de una cadena JSON a un objeto JavaScript utilizando **JSON.parse**.

Luego, la función realiza una serie de validaciones para asegurarse de que todos los campos requeridos del formulario estén completos. Si alguno de los campos (dirección, región, provincia, distrito) está vacío, se muestra una alerta al usuario y la función se interrumpe utilizando **return**. Esta validación es esencial para garantizar que se proporcionen todos los datos necesarios para crear la orden. Además, la función verifica si el carrito de compras está vacío. Si no hay productos en el carrito, se muestra una alerta indicando que el carrito está vacío y la función se interrumpe. Esto asegura que no se intenten generar órdenes de compra sin productos.

Una vez que se han validado todos los campos, la función extrae los nombres de la región, provincia y distrito seleccionados, utilizando las opciones seleccionadas en los elementos **<select>**. Luego, crea una lista de productos a partir del carrito, mapeando cada producto para incluir solo su ID y cantidad. Este mapeo se realiza utilizando el método **map** de los **arrays**. A continuación, la función intenta enviar una solicitud a la API para crear la orden de compra. Utiliza **fetch** para enviar una solicitud POST a la URL

/api/orden/crear, incluyendo en el cuerpo de la solicitud un objeto JSON con la dirección, productos, DNI del cliente, región, provincia, distrito y DNI de logística. La solicitud incluye encabezados que especifican que el contenido es de tipo **JSON (Content-Type: application/json)**.

Si la respuesta de la API no es exitosa (**!response.ok**), se lanza un error con un mensaje que incluye el estado HTTP de la respuesta. Si la respuesta es exitosa, la función convierte la respuesta JSON a un objeto JavaScript utilizando **response.json** y registra en la consola la nueva orden generada. Finalmente, la función elimina el carrito de compras del **localStorage** utilizando **localStorage.removeItem**, y muestra una alerta indicando que la orden se generó exitosamente. Si ocurre algún error durante el proceso, se captura en el bloque catch, se registra en la consola y se muestra una alerta con el mensaje de error.

Los conceptos importantes utilizados en esta función incluyen **async/await**, que permiten escribir código asíncrono de manera más legible y manejable, y **getElementById**, un método del DOM para seleccionar elementos HTML por su ID. La propiedad **selectedOptions** devuelve una colección de las opciones seleccionadas en un elemento **<select>**. **JSON.parse** convierte una cadena JSON en un objeto JavaScript, mientras que **JSON.stringify** convierte un objeto JavaScript en una cadena JSON. El método fetch se utiliza para hacer solicitudes HTTP. El método map crea un nuevo array con los resultados de llamar a una función para cada elemento del **array**. **localStorage** es una interfaz para almacenar datos de manera persistente en el navegador del usuario, y **removeItem** es un método de **localStorage** para eliminar un ítem específico.

➤ Funciones del área del proveedor



La página de proveedor está estructurada para incluir un encabezado fijo, una barra lateral de navegación y una sección principal de contenido. El encabezado, con la clase **header**, está diseñado para ser flexible y permanecer fijo en la parte superior de la página, ofreciendo siempre accesibilidad a la navegación principal. Contiene un título centrado que indica "Panel de Proveedor" y varios íconos de navegación para una fácil interacción del que lo usa. La barra lateral, identificada con la clase **sidebar**, alberga el perfil del proveedor y botones de navegación para acceder a diferentes secciones, como "Perfil" y "Aprobar Usuarios". Esta barra es colapsable, lo que permite ahorrar espacio en la pantalla y mejorar la experiencia de usuario. La sección principal del contenido, con la clase **main-content**, se ajusta dinámicamente según el estado de la barra lateral, manteniendo un diseño responsivo y limpio.

Obtenemos los productos de la base de datos

```
const express = require('express');
const router = express.Router();
const { sequelize } = require('../config/dbConfig');

// Ruta para obtener los productos
router.get('/', async (req, res) => {
  try {
    const [rows] = await sequelize.query('SELECT * FROM ordenes');
    res.json(rows);
  } catch (error) {
    console.error(error);
    res.status(500).send('Error al obtener los datos de productos');
  }
});
```

En esta función, **router.get('/')** define una ruta para manejar solicitudes GET en la URL raíz del **router**. La función es asíncrona, lo que permite el uso de `await` para esperar la resolución de las promesas. Dentro del bloque `try`, se ejecuta una consulta **SQL** para seleccionar todos los registros de la tabla `ordenes` utilizando **sequelize.query**. La respuesta de la consulta se guarda en **rows**, que luego se envía al cliente en formato JSON mediante **res.json(rows)**. Si ocurre un error durante el proceso, este se captura en el bloque `catch`, se registra en la consola con **console.error(error)**, y se envía una respuesta de error al cliente con un código de estado 500 usando **res.status(500).send('Error al obtener los datos de productos')**.

Algunos conceptos clave en esta función incluyen **async/await**, que permiten manejar operaciones asíncronas de manera más legible y eficiente; **sequelize.query**, que es una forma de ejecutar consultas SQL directamente en la base de datos; y **res.json**, que envía una respuesta JSON al cliente. También se utiliza **console.error** para registrar errores

en la consola, y **res.status(500).send** para enviar una respuesta de error al cliente en caso de que ocurra un problema al obtener los datos.

Función para aceptar las órdenes de compra

```
// Ruta para actualizar el dni_proveedor
router.post('/aceptar', async (req, res) => {
  const { productId, nombre, precio, cantidad, dni } = req.body;
  try {
    // Verificar el stock del producto en la tabla 'productos'
    const [productos] = await sequelize.query('SELECT * FROM productos WHERE id = ? AND dni_proveedor = ?', {
      replacements: [productId, dni]
    });

    if (productos.length === 0) {
      return res.status(404).json({ success: false, message: 'Producto no encontrado en el stock' });
    }

    const productoStock = productos[0];

    if (productoStock.cantidad < cantidad || productoStock.cantidad <= 0) {
      return res.status(400).json({ success: false, message: 'Stock insuficiente o agotado' });
    }

    // Actualizar el stock
    const nuevoStock = productoStock.cantidad - cantidad;
    await sequelize.query('UPDATE productos SET cantidad = ? WHERE id = ?', {
      replacements: [nuevoStock, productId]
    });

    // Actualizar el dni_proveedor en la tabla 'ordenes'
    const [ordenes] = await sequelize.query('SELECT * FROM ordenes');
    let updated = false;

    for (let orden of ordenes) {
      let productos = JSON.parse(orden.productos);
      for (let producto of productos) {
        if (producto.id == productId && producto.nombre == nombre && producto.precio == precio && producto.
            producto.dni_proveedor = "9999999";
            updated = true;
          }
        }
      if (updated) {
        await sequelize.query('UPDATE ordenes SET productos = ? WHERE id = ?', {
          replacements: [JSON.stringify(productos), orden.id]
        });
        break;
      }
    }

    if (updated) {
      res.json({ success: true });
    } else {
      res.status(404).json({ success: false, message: 'Producto no encontrado en la orden' });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ success: false, message: 'Error al actualizar productos' });
  }
});

module.exports = router;
```

En esta función se maneja una solicitud POST en la ruta **/aceptar** para actualizar el `dni_proveedor` de un producto específico. La función comienza extrayendo `productId`, `nombre`, `precio`, `cantidad` y `dni` del cuerpo de la solicitud (**req.body**). Luego, dentro de un bloque **try**, se ejecuta una consulta SQL para verificar el stock del producto en la tabla

productos utilizando **sequelize.query**. Esta consulta utiliza los parámetros **productId** y dni con **replacements** para prevenir inyecciones SQL. Si no se encuentra el producto en el stock, se devuelve una respuesta con un estado 404, indicando que el producto no está disponible.

Si el producto está disponible, se verifica si la cantidad solicitada es menor o igual a la cantidad en stock. Si no hay suficiente stock, se devuelve una respuesta con un estado 400, indicando que el stock es insuficiente o agotado. Si hay suficiente stock, se calcula el nuevo stock restando la cantidad solicitada del stock actual y se actualiza en la base de datos mediante otra consulta SQL con **sequelize.query**.

Posteriormente, se seleccionan todas las órdenes de la tabla ordenes con una consulta SQL. La función inicializa una variable **updated** como false. Se itera sobre cada orden, parseando la columna productos (**que está en formato JSON**) a un array de objetos utilizando **JSON.parse**. La función busca en este array el producto específico que coincide con **productId**, nombre, precio, cantidad y dni. Si se encuentra el producto, se actualiza el dni_proveedor a "9999999" y se marca updated como true. Después de actualizar el producto, se guarda la orden actualizada en la base de datos, convirtiendo el array de objetos de nuevo a una cadena JSON con **JSON.stringify**.

Finalmente, si el producto fue actualizado correctamente, se envía una respuesta de éxito con **res.json({ success: true })**. Si no, se envía una respuesta con un estado 404 indicando que el producto no se encontró en la orden. Si ocurre un error durante cualquier parte del proceso, este se captura en el bloque catch, se registra en la consola con **console.error(error)** y se envía una respuesta de error con un estado 500 utilizando **res.status(500).json({ success: false, message: 'Error al actualizar productos' })**.

El método **sequelize.query** de **Sequelize** permite ejecutar consultas SQL directamente en la base de datos, proporcionando una interacción directa y flexible con la base de datos. Esta función ejecuta comandos SQL precisos para obtener y actualizar los datos necesarios para la lógica de la aplicación.

La opción **replacements** se utiliza en **sequelize.query** para sustituir valores en la consulta SQL de manera segura. Esto es crucial para evitar inyecciones SQL, una técnica maliciosa que podría comprometer la seguridad de la base de datos al ejecutar comandos SQL no autorizados. Usar **replacements** asegura que los valores proporcionados sean tratados de manera segura, previniendo posibles ataques.

JSON.parse es un método que convierte una cadena JSON en un objeto JavaScript. En esta función, se utiliza para manipular los datos almacenados como JSON en la base de

datos. Esto es útil para trabajar con datos complejos que se almacenan en formato JSON, permitiendo acceder y modificar propiedades de los objetos de manera más sencilla.

JSON.stringify, por otro lado, convierte un objeto JavaScript en una cadena JSON. Este método se utiliza para almacenar datos complejos en la base de datos en un formato que pueda ser fácilmente serializado y **deserializado**. Al finalizar las actualizaciones, los datos del objeto se convierten nuevamente en una cadena JSON antes de ser almacenados.

El uso de los bloques **try/catch** en esta función es esencial para manejar errores. El bloque try contiene el código que puede generar errores, como las consultas a la base de datos o la manipulación de datos. Si ocurre un error en este bloque, el control se transfiere al bloque catch, donde el error se captura y maneja de manera segura. Esto incluye registrar el error en la consola y enviar una respuesta adecuada al cliente.

El método **res.status** se utiliza para establecer el código de estado HTTP de la respuesta. Esto permite al servidor indicar claramente el resultado de la solicitud, como un éxito, un error del cliente, o un error del servidor. Establecer el código de estado es importante para que el cliente entienda el resultado de su solicitud.

Finalmente, el método **res.json** se emplea para enviar una respuesta en formato JSON. Este formato es común para intercambiar datos entre el servidor y el cliente en aplicaciones web, proporcionando una estructura clara y fácil de interpretar. Enviar respuestas en formato JSON permite una comunicación eficaz y estructurada entre el **backend** y el **frontend** de la aplicación.

Función para cargar a la pantalla la solicitud de orden de compra

```

function cargarProductos() {
  // Hacer una solicitud para obtener los productos
  fetch('/api/productos')
    .then(response => {
      if (!response.ok) {
        throw new Error('Error en la respuesta del servidor');
      }
      return response.json();
    })
    .then(data => {
      console.log('Datos recibidos del servidor:', data);
      const productos = data.flatMap(orden => {
        try {
          const productosParsed = JSON.parse(orden.productos);
          return productosParsed.filter(producto => {
            const comparacion = producto.dni_proveedor === dniProveedor;
            console.log(`Comparando dni_proveedor: ${producto.dni_proveedor} con ${dniProveedor} ->`);
            return comparacion;
          });
        } catch (error) {
          console.error('Error al analizar el JSON de productos:', error);
          return [];
        }
      });

      const tableBody = document.getElementById('productos-table-body');
      tableBody.innerHTML = ''; // Limpiar la tabla antes de volver a renderizar

      // Llenar la tabla con los productos filtrados
      productos.forEach(producto => {
        const row = document.createElement('tr');
        row.innerHTML = `
          <td>${producto.id}</td>
          <td>${producto.nombre}</td>
          <td>${producto.precio}</td>
          <td>${producto.cantidad}</td>
          <td><button class="aceptar-btn" data-id="${producto.id}" data-nombre="${producto.nombre}" c
        `;
        tableBody.appendChild(row);
      });
    });
}

```

```

document.querySelectorAll('.aceptar-btn').forEach(button => {
  button.addEventListener('click', function () {
    const productId = this.getAttribute('data-id');
    const nombre = this.getAttribute('data-nombre');
    const precio = this.getAttribute('data-precio');
    const cantidad = this.getAttribute('data-cantidad');
    const dni = this.getAttribute('data-dni');
    fetch('/api/productos/aceptar', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ productId, nombre, precio, cantidad, dni })
    })
    .then(response => response.json())
    .then(data => {
      if (data.success) {
        alert('Producto aceptado exitosamente');
        cargarProductos(); // Recargar los productos después de una actualización exitosa
      } else {
        alert('Error al aceptar producto: ' + data.message);
      }
    })
    .catch(error => console.error('Error al aceptar producto:', error));
  });
});
.catch(error => console.error('Error al cargar los productos:', error));
}

```

Es una función compleja diseñada para interactuar con el servidor, obtener datos, procesarlos y renderizarlos en una interfaz de usuario, permitiendo además la interacción con los productos mostrados. En primer lugar, la función hace una solicitud HTTP GET a la ruta `/api/productos` utilizando el método `fetch`. Este método devuelve una promesa que se resuelve con la respuesta de la solicitud. Si la respuesta no es exitosa, es decir, si `response.ok` es falso, se lanza un error indicando que hubo un problema en la respuesta del servidor. Si la respuesta es exitosa, se convierte en un objeto JSON utilizando `response.json`.

Una vez que los datos JSON se han recibido, se procesan mediante el método `flatMap`, que permite aplanar el array de órdenes y extraer los productos. Dentro de este método, se intenta parsear el campo `productos` de cada orden utilizando `JSON.parse`. Si ocurre un error durante el `parseo` del JSON, este se captura y se maneja con un bloque `catch`, retornando un array vacío para evitar que la función falle. Los productos `parseados` se filtran para incluir solo aquellos cuyo `dni_proveedor` coincida con una variable `dniProveedor`. Esta comparación se realiza dentro del método `filter`, y se registra en la consola para fines de depuración.

Después de filtrar los productos, la función se encarga de limpiar el contenido actual de la tabla HTML donde se mostrarán los productos. Esto se hace estableciendo `tbody.innerHTML` a una cadena vacía. Luego, para cada producto filtrado, se crea una nueva fila de `tabla` (`<tr>`) utilizando `document.createElement('tr')`, y se establece su contenido HTML con las propiedades del producto, incluyendo un botón

"Aceptar". Este botón contiene varios atributos de datos que representan las propiedades del producto, tales como data-id, data-nombre, data-precio, data-cantidad y data-dni.

Una vez que las filas de productos se han agregado a la tabla, se añade un evento de clic a cada botón "Aceptar" utilizando `document.querySelectorAll('.aceptar-btn')` y el método `forEach` para iterar sobre cada botón. Cuando se hace clic en un botón, se recogen los atributos de datos del producto y se realiza una solicitud POST a la [ruta /api/productos/aceptar](#) con esos datos en el cuerpo de la solicitud, utilizando `JSON.stringify` para convertir el objeto JavaScript en una cadena JSON. Si la respuesta del servidor indica éxito, se muestra una alerta y se recargan los productos llamando nuevamente a `cargarProductos`. Si ocurre un error, se muestra una alerta con el mensaje de error y se registra el error en la consola.

El método `fetch` es crucial para realizar solicitudes HTTP desde JavaScript y manejar respuestas de manera asíncrona. La combinación de `fetch` con `then` y `catch` permite manejar las respuestas y errores de las solicitudes de manera estructurada. Además, `JSON.parse` y `JSON.stringify` son métodos esenciales para convertir entre cadenas JSON y objetos JavaScript, lo cual es fundamental para procesar y almacenar datos de manera efectiva.

El uso de `flatMap` y `filter` permite manipular y filtrar `arrays` de manera eficiente, mientras que `querySelectorAll` y `addEventListener` son métodos del DOM que permiten seleccionar elementos y agregar manejadores de eventos para hacer la interfaz interactiva. El método `innerHTML` se utiliza para establecer o devolver el contenido HTML de un elemento, y `appendChild` permite añadir nuevos nodos al DOM.

Base de datos para que proveedor registre sus bienes y suministros

```
-- Estructura de tabla para la tabla `productos`
--
CREATE TABLE `productos` (
  `id` int(11) NOT NULL,
  `nombre` varchar(255) NOT NULL,
  `categoria` varchar(255) NOT NULL,
  `precio` decimal(10,2) NOT NULL,
  `cantidad` int(11) NOT NULL,
  `unidad_de_medida` varchar(50) NOT NULL,
  `dni_proveedor` varchar(8) NOT NULL DEFAULT '71885493'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

➤ Funciones del área de almacén



El encabezado de la página está diseñado para ser fijo en la parte superior de la ventana, permitiendo siempre la accesibilidad a la navegación principal. Utiliza un fondo con un gradiente de azul marino a azul cielo y contiene un título centrado "Panel de Almacén". El encabezado está estilizado para ser flexible, permitiendo la inclusión de íconos de navegación a la izquierda y derecha, los cuales cambian de color y se agrandan cuando se pasa el cursor sobre ellos. Este encabezado también incluye un botón de alternancia que permite colapsar o expandir la barra lateral, proporcionando una experiencia de usuario dinámica.

La barra lateral contiene el perfil del administrador del almacén y varios botones de navegación para acceder a diferentes secciones. Está diseñada con un gradiente de azul medianoche a azul cielo y puede colapsarse para ahorrar espacio en la pantalla. Cuando la barra lateral está colapsada, solo se muestran los íconos, ocultando los textos y las imágenes de perfil. Esto mejora la usabilidad en pantallas pequeñas y permite a los usuarios concentrarse en el contenido principal.

La sección de contenido principal se ajusta dinámicamente según el estado de la barra lateral. Está diseñada con un fondo gris claro y márgenes adecuadas para mantener una separación visual clara entre los elementos. Esta sección contiene diversas subsecciones que se muestran u ocultan según la navegación del usuario. Las secciones se activan mediante clases CSS y se estilizan para ser coherentes y profesionales, utilizando sombras y bordes suaves para destacar los elementos.

El formulario de búsqueda y filtros permite a los usuarios buscar productos específicos en el almacén. Los usuarios pueden filtrar los productos por diferentes criterios, como nombre, categoría o proveedor. Los elementos del formulario están diseñados para ser

claros y accesibles, con etiquetas en negrita y campos de entrada amplios. Los botones de formulario tienen transiciones suaves de color y sombra, mejorando la experiencia de interacción del usuario.

La tabla de productos es una parte central del diseño de la página. Está diseñada para ser clara y legible, con columnas que muestran el ID, nombre, precio, cantidad y proveedor de cada producto. Cada fila de la tabla incluye un botón "Aceptar" que permite al usuario interactuar con los productos directamente desde la tabla. Los botones están estilizados para ser visibles y accesibles, con cambios de color al pasar el cursor sobre ellos para indicar interactividad.

El archivo CSS proporciona estilos detallados para todos estos elementos, asegurando una apariencia coherente y profesional. Los encabezados tienen un fondo con un gradiente azul y texto blanco, diseñados para mantenerse fijos en la parte superior de la ventana. La barra lateral tiene un gradiente y puede colapsarse, mientras que los botones de navegación cambian de color al pasar el cursor sobre ellos. La sección de contenido principal tiene un fondo gris claro y sombras para separar visualmente los elementos.

Funcionalidad del combobox para cargar la tabla

```
function gestionarOfertas() {
  document.addEventListener('DOMContentLoaded', function() {
    cargarOrdenes();

    document.getElementById('actualizarEstado').onclick = function() {
      const select = document.getElementById('ordenes');
      if (select.selectedIndex === -1) {
        alert('Seleccione una orden para actualizar');
        return;
      }
      const orden = JSON.parse(select.value);
      gestionarArticulosYActualizarEstado(orden.id, orden.productos);
    };

    document.getElementById('ordenes').addEventListener('change', function() {
      actualizarTablaProductos();
    });
  });
}
```

Primero, la función agrega un **event listener** para el evento **DOMContentLoaded** en el documento. Este evento se dispara cuando todo el HTML ha sido completamente cargado y parseado, sin esperar a que se carguen las hojas de estilo, imágenes y subframes. Esto asegura que todos los elementos del DOM estén listos para ser manipulados.

Dentro del **event listener DOMContentLoaded**, se llama a la función `cargarOrdenes`. Esta función es responsable de hacer una solicitud al servidor para obtener las órdenes válidas y llenar el combobox con estas órdenes. Esto prepara la página con los datos iniciales necesarios para la gestión de las órdenes.

A continuación, se configura el evento **onclick** para el botón con el ID actualizarEstado. Cuando se hace clic en este botón, se verifica si hay una orden seleccionada en el combobox (select). Si no hay una orden seleccionada (**selectedIndex === -1**), se muestra una alerta indicando al usuario que debe seleccionar una orden. Si hay una orden seleccionada, se parsea el valor seleccionado (que está en formato JSON) para obtener el objeto de la orden. Luego, se llama a la función gestionarArticulosYActualizarEstado con el ID de la orden y los productos de la orden como argumentos. Esta función maneja la lógica para gestionar los artículos y actualizar el estado de la orden.

Finalmente, se configura un **event listener** para el evento **change** del combobox con el ID órdenes. Este evento se dispara cuando el usuario selecciona una orden diferente en el combobox. Cuando esto ocurre, se llama a la función actualizarTablaProductos. Esta función es responsable de actualizar la tabla de productos en la página para mostrar los detalles de los productos asociados con la orden seleccionada.

Función para cargar la tabla con las ordenes aceptadas por el proveedor

```
function cargarOrdenes() {
  fetch('/api/rece_ordenes/validas')
    .then(response => response.json())
    .then(ordenes => {
      const select = document.getElementById('ordenes');
      const actualizarEstadoButton = document.getElementById('actualizarEstado');
      select.innerHTML = ''; // Limpiar el combobox antes de llenarlo
      const tbody = document.getElementById('productos-table').querySelector('tbody');
      tbody.innerHTML = ''; // Limpiar la tabla antes de llenarla

      if (ordenes.length === 0) {
        alert('No hay órdenes válidas disponibles');
        actualizarEstadoButton.disabled = true;
        return;
      }

      ordenes.forEach(orden => {
        const option = document.createElement('option');
        option.value = JSON.stringify(orden);
        option.text = orden.id;
        select.appendChild(option);
      });
    });
}
```

```

// Habilitar el botón si hay órdenes
actualizarEstadoButton.disabled = false;

// Actualizar la tabla al cargar las órdenes por primera vez
if (select.options.length > 0) {
  select.selectedIndex = 0;
  actualizarTablaProductos();
} else {
  tbody.innerHTML = ''; // Limpiar la tabla si no hay opciones
}
})
.catch(error => {
  console.error('Error al obtener las órdenes:', error);
  alert('Error al obtener las órdenes válidas');
});
}

```

La función tiene como objetivo obtener una lista de órdenes válidas desde el servidor, poblar un combobox con esas órdenes y actualizar una tabla de productos basada en la orden seleccionada. Para lograr esto, la función realiza una solicitud HTTP GET a la ruta [/api/rece_ordenes/validas](#) utilizando el método `fetch`, que permite realizar solicitudes HTTP de manera asíncrona. El método `fetch` devuelve una promesa que, una vez resuelta, contiene la respuesta del servidor.

Cuando la respuesta se recibe, se convierte en un objeto JSON mediante el método `response.json()`. Esta conversión es esencial para poder trabajar con los datos en un formato utilizable dentro de la función. Una vez convertida la respuesta, se procesan las órdenes recibidas. Primero, se seleccionan los elementos HTML correspondientes al combobox y al botón de actualizar estado. Luego, se limpia el contenido actual del combobox y de la tabla de productos para asegurarse de que no haya datos antiguos.

Si no hay órdenes válidas disponibles, se muestra una alerta al usuario y se desactiva el botón de actualizar estado. Esta verificación es importante para informar al usuario sobre la ausencia de órdenes y para evitar acciones innecesarias. Si hay órdenes disponibles, se itera sobre cada una de ellas y se crea un elemento `option` para cada orden. El valor de cada opción es la orden completa en formato JSON, y el texto visible es el ID de la orden. Estos elementos se añaden al combobox, permitiendo al usuario seleccionar una orden específica.

El botón de actualizar estado se habilita si hay órdenes disponibles. Luego, si hay opciones en el combobox, se selecciona la primera por defecto y se llama a la función `actualizarTablaProductos` para mostrar los productos de la primera orden en la tabla. Esta inicialización asegura que los productos de la primera orden se muestren automáticamente al cargar la página.

En caso de que ocurra un error durante la solicitud o el procesamiento de las órdenes, este se captura en un bloque `catch`. El error se registra en la consola y se muestra una

alerta al usuario indicando que hubo un problema al obtener las órdenes válidas. Este manejo de errores es crucial para mantener informados a los usuarios y para la depuración del sistema.

Entre los conceptos y métodos importantes utilizados en esta función se encuentran **fetch**, que permite realizar solicitudes HTTP de manera asíncrona; **then**, que se utiliza para encadenar acciones a una promesa resuelta; y **catch**, que maneja errores en promesas. El método **response.json()** convierte la respuesta HTTP en un objeto JSON. Además, **document.getElementById** se utiliza para seleccionar elementos HTML por su ID, y **innerHTML** se usa para establecer o devolver el contenido HTML de un elemento, permitiendo limpiar el contenido de elementos HTML. Por último, **createElement** crea un nuevo elemento HTML, y **appendChild** añade un nodo al final de la lista de un nodo padre especificado.

Función para la creación de la tabla

```
function actualizarTablaProductos() {
  const select = document.getElementById('ordenes');
  const tbody = document.getElementById('productos-table').querySelector('tbody');
  tbody.innerHTML = ''; // Limpiar la tabla antes de llenarla

  if (select.selectedIndex === -1) {
    return; // No hacer nada si no hay selección
  }

  const orden = JSON.parse(select.value);
  const productos = orden.productos;

  productos.forEach(producto => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${producto.id}</td>
      <td>${producto.nombre}</td>
      <td>${producto.precio}</td>
      <td>${producto.cantidad}</td>
    `;
    tbody.appendChild(row);
  });
}
```

La función `actualizarTablaProductos` se encarga de actualizar el contenido de una tabla HTML para mostrar los productos asociados con una orden seleccionada en un combobox. A continuación, se detalla lo que hace cada parte de esta función:

Primero, la función selecciona el elemento del combobox que contiene las órdenes mediante **document.getElementById('ordenes')** y el cuerpo de la tabla (**tbody**) donde se mostrarán los productos utilizando **document.getElementById('productos-table').querySelector('tbody')**. Para asegurarse de que la tabla esté limpia antes de añadir nuevos datos, se establece

tbody.innerHTML a una cadena vacía, lo que elimina cualquier contenido previo en la tabla.

La función verifica si hay una orden seleccionada en el combobox. Si no hay ninguna selección (**selectedIndex === -1**), la función simplemente retorna sin hacer nada, evitando así cualquier operación innecesaria o errores.

Si hay una orden seleccionada, se obtiene el valor de la selección, que está en formato JSON, y se convierte en un objeto JavaScript utilizando **JSON.parse**. Este objeto orden contiene los detalles de la orden, incluyendo una lista de productos asociados.

Luego, se itera sobre la lista de productos utilizando **forEach**. Para cada producto, se crea una nueva fila de tabla (**<tr>**) y se define su contenido HTML, añadiendo celdas (**<td>**) para mostrar el ID, nombre, precio y cantidad del producto. Cada nueva fila se añade al cuerpo de la tabla (**tbody**) utilizando **tbody.appendChild(row)**.

Función para actualizar el stock luego de recibir la orden de compra

```

async function gestionarArticulosYActualizarEstado(id, productos) {
  try {
    // Obtener la información completa de los productos desde la base de datos
    const productosCompletos = await Promise.all(productos.map(async (producto) => {
      const response = await fetch(`/api/rece_ordenes/productos/${producto.id}`);
      if (!response.ok) {
        throw new Error(`Producto no encontrado: ${producto.id}`);
      }
      const data = await response.json();
      return {
        nombre: data.nombre,
        unidad: data.unidad,
        cantidad: producto.cantidad,
        precio: data.precio
      };
    }));
  }
}

```

En esta parte del código, la función está obteniendo la información completa de los productos desde la base de datos. Esto se realiza utilizando el método **Promise.all** junto con el método **map** para manejar múltiples solicitudes asíncronas de manera concurrente.

Primero, la función **Promise.all** se usa para ejecutar todas las solicitudes en paralelo y esperar hasta que todas se completen. **Promise.all** toma un array de promesas (en este caso, las promesas generadas por **fetch**) y devuelve una sola promesa que se resuelve cuando todas las promesas del **array** se han resuelto o se rechaza si alguna de las promesas se rechaza.

Dentro de **Promise.all**, se utiliza el método **map** para iterar sobre el **array** de productos. Para cada producto, se realiza una solicitud HTTP con **fetch** a un **endpoint**

específico (`/api/rece_ordenes/productos/${producto.id}`). Esta solicitud busca obtener los detalles del producto basado en su id.

Después de realizar la solicitud con `fetch`, se verifica si la respuesta del servidor es exitosa utilizando la propiedad `ok` de la respuesta. Si la respuesta no es exitosa (`!response.ok`), se lanza un error con un mensaje que indica que el producto no se encontró. Esto se hace para asegurarse de que la función maneje correctamente los casos en que la solicitud no pueda recuperar la información del producto.

Si la respuesta es exitosa, se convierte el cuerpo de la respuesta a formato JSON utilizando el método `json()` de la respuesta. Este método también es asíncrono y devuelve una promesa que se resuelve con el resultado de convertir el cuerpo de la respuesta en un objeto JavaScript.

Finalmente, se crea y retorna un nuevo objeto que contiene los detalles del producto, incluyendo nombre, unidad, cantidad y precio. La cantidad proviene del objeto producto original, mientras que nombre, unidad y precio se obtienen de la respuesta JSON. Este nuevo objeto representa la información completa del producto que se necesita para los pasos posteriores en la gestión de la orden.

```
// Gestionar los artículos antes de actualizar el estado de la orden
await Promise.all(productosCompletos.map(producto => {
  return fetch('/api/rece_ordenes/gestionarArticulo', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      nombre: producto.nombre,
      unidad: producto.unidad,
      cantidad: producto.cantidad,
      precio: producto.precio
    })
  });
}));
```

En esta parte del código, se gestionan los artículos antes de actualizar el estado de la orden. Este proceso se realiza usando `Promise.all` junto con `fetch` para manejar múltiples solicitudes HTTP de manera concurrente.

Primero, se utiliza `Promise.all` para ejecutar todas las solicitudes en paralelo y esperar hasta que todas se completen. `Promise.all` toma un array de promesas (en este caso, las promesas generadas por `fetch`) y devuelve una sola promesa que se resuelve cuando todas las promesas del array se han resuelto o se rechaza si alguna de las promesas se rechaza.

Dentro de **Promise.all**, se usa el método `map` para iterar sobre el array `productosCompleto`. Para cada producto, se realiza una solicitud HTTP con `fetch` al **endpoint** `/api/rece_ordenes/gestionarArticulo`. Esta solicitud tiene el propósito de enviar los detalles del producto al servidor para que los gestione adecuadamente.

La solicitud HTTP se configura con el método `POST`, lo que indica que se está enviando información nueva al servidor. Se incluyen los **headers** necesarios para que el servidor interprete correctamente la solicitud, específicamente **'Content-Type': 'application/json'**, lo que indica que el cuerpo de la solicitud está en formato JSON.

El cuerpo de la solicitud (**body**) se construye usando **JSON.stringify**, que convierte el objeto JavaScript con los detalles del producto (nombre, unidad, cantidad, precio) en una cadena JSON. Esta cadena JSON se envía al servidor como parte de la solicitud `POST`.

Al final, **Promise.all** asegura que todas las solicitudes `fetch` para gestionar los artículos se completen antes de que la función continúe con los siguientes pasos. Este enfoque garantiza que todos los productos se gestionen adecuadamente antes de proceder a actualizar el estado de la orden.

```
// Después de gestionar los artículos, actualizar el estado de la orden
const response = await fetch(`/api/rece_ordenes/${id}/estado`, {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ estado: 'almacen' })
});

if (!response.ok) {
  throw new Error('Error al actualizar el estado de la orden');
}

alert('Estado de la orden actualizado correctamente');
// Recargar las órdenes y actualizar la tabla
cargarOrdenes();
} catch (error) {
  console.error('Error al gestionar los artículos o actualizar el estado de la orden:', error);
  alert('Error al gestionar los artículos o actualizar el estado de la orden');
}
```

Después de gestionar los artículos, la función procede a actualizar el estado de la orden. Utiliza `fetch` para enviar una solicitud HTTP `PUT` al **endpoint** `/api/rece_ordenes/${id}/estado`, donde `id` es el identificador de la orden. La solicitud incluye un **header Content-Type** con el valor `application/json` y un cuerpo (**body**) que contiene un objeto JSON con el nuevo estado de la orden (`{ estado: 'almacen' }`). Si la respuesta del servidor no es exitosa (`!response.ok`), se lanza un error indicando que hubo un problema al actualizar el estado de la orden.

Si la actualización del estado es exitosa, se muestra una alerta informando al usuario que el estado de la orden se ha actualizado correctamente. Luego, la función llama a cargarOrdenes para recargar las órdenes y actualizar la tabla de productos. Si ocurre algún error durante la gestión de los artículos o la actualización del estado, se captura en el bloque catch, se registra el error en la consola y se muestra una alerta al usuario indicando que hubo un problema en el proceso.

La conexión con la base de datos que se realiza para obtener datos

```
const express = require('express');
const router = express.Router();
const { obtenerOrdenesValidas, actualizarEstadoOrden, gestionarArticulo, obtenerProductoPorId } = require('../contr

// Ruta para obtener órdenes válidas
router.get('/validas', async (req, res) => {
  try {
    const ordenes = await obtenerOrdenesValidas();
    res.json(ordenes);
  } catch (error) {
    console.error('Error al obtener las órdenes válidas:', error);
    res.status(500).json({ error: 'Error al obtener las órdenes válidas' });
  }
});

// Ruta para actualizar el estado de una orden
router.put('/:id/estado', async (req, res) => {
  const { id } = req.params;
  const { estado } = req.body;
  try {
    await actualizarEstadoOrden(id, estado);
    res.json({ message: 'Estado de la orden actualizado correctamente' });
  } catch (error) {
    console.error('Error al actualizar el estado de la orden:', error);
    res.status(500).json({ error: 'Error al actualizar el estado de la orden' });
  }
});
```

En esta parte del código se definen dos rutas en un **router** de Express para manejar solicitudes HTTP relacionadas con órdenes. La primera ruta, **/validas**, es una ruta GET que se utiliza para obtener todas las órdenes válidas. Cuando se recibe una solicitud GET en esta ruta, se llama a la función obtenerOrdenesValidas para obtener las órdenes válidas desde la base de datos u otro origen de datos. Si la operación es exitosa, se envían las órdenes como una respuesta JSON al cliente. En caso de que ocurra un error durante el proceso, se captura y se envía una respuesta con un estado HTTP 500 y un mensaje de error.

La segunda ruta, **/:id/estado**, es una ruta PUT que se utiliza para actualizar el estado de una orden específica. El id de la orden se pasa como un parámetro en la URL, y el nuevo estado de la orden se proporciona en el cuerpo de la solicitud. Cuando se recibe una solicitud PUT en esta ruta, se extraen el id de la orden y el estado del cuerpo de la solicitud. Luego, se llama a la función actualizarEstadoOrden con estos valores para actualizar el estado de la orden en la base de datos. Si la operación es exitosa, se envía

una respuesta JSON con un mensaje indicando que el estado de la orden se actualizó correctamente. Si ocurre un error, se captura y se envía una respuesta con un estado HTTP 500 y un mensaje de error.

Los métodos utilizados en este contexto son métodos asíncronos (**async**) que permiten manejar operaciones que pueden tardar en completarse, como acceder a una base de datos. La estructura **try-catch** se usa para manejar errores de manera eficiente, garantizando que se capture cualquier problema y se informe adecuadamente al cliente mediante respuestas HTTP adecuadas. El método **res.json** se usa para enviar respuestas en formato JSON, mientras que **res.status(500).json** se usa para enviar respuestas de error con un estado HTTP 500.

La base de datos que se está utilizando para cargar los productos recibidos

```
-- Estructura de tabla para la tabla `item`  
  
CREATE TABLE `item` (  
  `id_item` int(11) NOT NULL,  
  `descripcion_item` varchar(80) NOT NULL,  
  `unidad` varchar(30) NOT NULL,  
  `stock` int(11) NOT NULL,  
  `precio` decimal(10,2) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
  
--
```

CONCLUSIÓN

La implementación del sistema "Pata Amarilla" va permitir a la empresa de turismo optimizar significativamente la gestión de sus bienes y suministros. Este sistema integral abarca y conecta múltiples áreas críticas como Usuario, jefe de área, Administración, Proveedor, Logística y Almacén, asegurando que todos los procesos internos se realicen de manera eficiente y coordinada. A continuación, se detallan las conclusiones clave derivadas del uso y beneficios de este sistema:

La implementación de este sistema va ser una herramienta importante para la empresa, permitiendo una administración más precisa y eficiente de los bienes y suministros. La capacidad de rastrear y gestionar inventarios en tiempo real reduce significativamente los errores y pérdidas asociadas con la gestión manual, mejorando así la eficiencia operativa y el funcionamiento de todo el sistema.

El diseño del sistema permite la integración fluida de diversas áreas funcionales dentro de la empresa. Cada módulo, ya sea de Usuario, Administración, Proveedor, Logística o Almacén, está interconectado, facilitando una comunicación efectiva y una transferencia de datos sin interrupciones entre departamentos o áreas. Esta integración asegura que las actividades de un área informen y apoyen a las de otra, creando un flujo de trabajo sinérgico y cohesivo.

Las funcionalidades detalladas del sistema, explicadas en este documento, aseguran que cada área pueda cumplir con sus responsabilidades de manera efectiva. Por ejemplo, los módulos de Administración y Logística trabajan conjuntamente para asegurar que los pedidos se gestionen y procesen de manera eficiente, desde la solicitud inicial hasta la entrega final. Esta coordinación no solo mejora la eficiencia interna, sino que también reduce los tiempos de respuesta y mejora la satisfacción del cliente.

El sistema "Pata Amarilla" no solo mejora la eficiencia operativa de la empresa, sino que también asegura una mayor transparencia y control sobre los recursos. La capacidad de generar reportes detallados y de acceder a datos en tiempo real permite a los administradores tomar decisiones informadas y estratégicas. Además, la transparencia en los procesos reduce la probabilidad de errores y fraudes, aumentando la confianza en la gestión interna.

Se podría decir también, que la implementación del sistema "Pata Amarilla" va ser un catalizador para el crecimiento y éxito a largo plazo de la empresa. Al mejorar la eficiencia, transparencia y control sobre los recursos, el sistema no solo optimiza las operaciones actuales, sino que también prepara a la empresa para futuros desafíos y oportunidades. La capacidad de adaptarse y evolucionar con las necesidades de la empresa asegura que "Pata Amarilla" seguirá siendo una herramienta invaluable en la gestión de bienes y suministros, contribuyendo al crecimiento sostenido y al éxito de la empresa.

SOBRE LOS AUTORES

GUILLERMO AUGUSTO BOCANGEL WEYDERT: Destacado académico y líder en el ámbito de la Ingeniería Industrial y la Gestión del Conocimiento. Doctor en Ingeniería Industrial por la Universidad Nacional Federico Villareal, Magíster en Gestión del Conocimiento por la Escuela de Organización Industrial de España, Magister en Ingeniería Industrial en la Universidad Mayor de San Marcos, Post Doctor en Ciencias e Ingeniería y Doctor Honoris causa por la Universidad Nacional de Ucayali y la Universidad Intercultural de la Amazonia. Con una sólida trayectoria académica es docente en diversas universidades peruanas y extranjeras. Su expertise abarca la gerencia estratégica y los procesos, y es reconocido por su trabajo como consultor y conferencista internacional en temas de Balanced Scorecard (BSC) y control de procesos. <https://orcid.org/0000-0003-1216-0944>

ELMER S. CHUQUIYURI SALDIVAR: Ingeniero de Sistemas e Informática, Magíster en Gestión Pública Para del Desarrollo Social, docente de la Universidad Nacional Hermilio Valdizán, Consultor en Modernización y Transformación Digital. Especialista en Desarrollo de Soluciones con TIC's. Universidad Nacional Hermilio Valdizán de Huánuco. <https://orcid.org/0009-0009-4268-3034>

GUILLERMO AUGUSTO BOCANGEL MARÍN: Ingeniero industrial con una destacada trayectoria académica y profesional, graduado de la Pontificia Universidad Católica del Perú (PUCP) con Maestría en Gestión y Dirección de Empresas Constructoras e Inmobiliarias y especialización en áreas como Automatización Industrial, Lean Manufacturing, Gestión de Proyectos y Sistemas de Gestión Integrados. Docente en la Facultad de Ingeniería y Arquitectura de la Universidad San Martín de Porres e investigador universitario. La combinación de su formación académica y su vasta experiencia empresarial le permite contribuir de manera significativamente al desarrollo de estrategias innovadoras y eficientes, tanto en el ámbito educativo como en el industrial. <https://orcid.org/0000-0002-5431-9805>

JHONNY HENRY PIÑÁN GARCÍA: Docente investigador. Ingeniero Industrial con especializaciones en Sistemas y Tecnologías de la Información, amplia experiencia en las áreas de desarrollo de sistemas y soporte tecnológico, en empresas líderes del país. con una Maestría en Didáctica y Tecnologías de la Información y egresado del Doctorado en Gestión Empresarial. Docente en la Universidad Nacional Hermilio Valdizán adscrito a la Facultad de Ingeniería Industrial. Autor de publicaciones de artículos científicos y de libros. Universidad Nacional Hermilio Valdizán de Huánuco. <https://orcid.org/0000-0002-0263-7668>

GUADALUPE RAMIREZ REYES: Doctora en Ingeniería por la Universidad Nacional Federico Villarreal y Maestro en Ciencias con mención en Ingeniería de Sistemas por la Universidad Nacional de Ingeniería. Cuenta con artículos publicados en revistas científicas desarrollados en colaboración y en autonomía, con experiencia en temas de programación, simulación medio ambiente, seguridad y salud en el trabajo y

optimización de procesos de producción usando herramientas Lean Manufacturing.
<https://orcid.org/0000-0002-4007-7729>

HERNAN WILMER GARCIA BONILLA: Ingeniero de sistemas, cuya formación y experiencia le ha permitido desarrollar un enfoque integral, especialmente en el análisis de sistemas y la gestión de información. Su pasión por entender la complejidad de las organizaciones complementa con un profundo interés en el análisis de datos y la prospectiva, capacitándolo para perfeccionar su capacidad para diseñar y gestionar sistemas que no solo respondan a las necesidades actuales, sino que también sean flexibles, escalables y adaptables a futuros desafíos.

<https://orcid.org/0009-0001-6542-079X>

LINCOL JARLY GOMEZ MEZA: Ingeniero de sistemas con una sólida formación técnica y una Maestría en Gestión Pública, lo que me permite combinar experticia en desarrollo de software, redes y telecomunicaciones con una comprensión profunda de la administración pública y la gestión organizacional. Mi experiencia abarca una amplia gama de habilidades y conocimientos que me permiten abordar desafíos complejos desde una perspectiva integral. <https://orcid.org/0000-0001-6173-5463>

ISBN 978-65-258-2807-7

**Atena**
Editora

Año 2024