UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACIÓN II

Unidad Temática N°4:
Fundamentos del Desarrollo Web

Material Teórico
1° Año – 2° Cuatrimestre

Índice

FUNDAMENTOS DEL DESARROLLO WEB

Introducción

En esta unidad, se explora los pilares esenciales del desarrollo web moderno, ofreciendo una inmersión práctica y teórica en HTML, CSS, Bootstrap y JavaScript. El objetivo se centra en que los estudiantes aprenderán a construir y diseñar páginas web funcionales y estéticamente atractivas, comenzando con la estructura básica en HTML y progresando hacia la creación de interfaces responsivas con Bootstrap. También se abordará el estilizado avanzado mediante CSS y la implementación de interactividad con JavaScript, permitiendo a los estudiantes integrar y aplicar sus conocimientos en proyectos reales.

El contenido de esta unidad no solo se enfocará en el diseño y la construcción de páginas web, sino que también servirá como complemento crucial para los conocimientos adquiridos en las dos unidades anteriores, centradas en el desarrollo de back-end y la creación de WebAPIs. Los conceptos y habilidades adquiridos aquí permitirán a los estudiantes integrar eficazmente las interfaces diseñadas en HTML, CSS y Bootstrap con las APIs desarrolladas, facilitando la comunicación entre el front-end y el back-end. Esta integración permitirá desarrollar aplicaciones web completas y funcionales.

INTRODUCCIÓN AL DESARROLLO WEB

¿Qué es el desarrollo web?

En unidades anteriores, hemos discutido la infraestructura subyacente a las conexiones entre computadoras, hemos visto lo que implica Internet y los protocolos que se usan para las comunicaciones. También hemos mencionado que existe un modelo en el que se basan las comunicaciones, donde hablamos de emisores y receptores, que concretamente los llamamos clientes y servidores.

En este modelo, principalmente identificamos como cliente a un navegador web, sin importar en que dispositivo se encuentre; puede ser un dispositivo móvil o una computadora. Los navegadores web reconocen tres lenguajes y nada más que tres. Éstos son HTML, CSS y JavaScript.

A menudo escuchamos hablar de sitios web desarrollados con otros lenguajes, incluso suelen nombrarse PHP, Java o Python, entre otros. Sin embargo, lo que hacen estos lenguajes es responder a las peticiones enviadas desde un navegador web, de forma tal que pueda ser interpretado, procesado y presentado haciendo uso de HTML, CSS y JavaScript. Es imperativo entender que un navegador web no puede procesar código PHP, Java o Python.

En el enfoque tradicional del desarrollo web, podemos decir que el servidor le entrega al cliente, el navegador, lo que se denomina una página web. Una página web está compuesta de diversos tipos de contenido: títulos, textos, imágenes, tipografías, videos, botones, entre otros.

El primer lenguaje de navegador que mencionamos es HTML. HTML es un lenguaje de maquetado, el cual veremos con más detalle en múltiples secciones de esta unidad. Este lenguaje se encarga del contenido de la página web, define la estructura y los componentes de toda la interfaz visible del sitio web.

Los sitios web serían poco atractivos sin distintos colores, formas, márgenes y tipografías. En este momento, entra en juego CSS. Este lenguaje cumple el objetivo de determinar todos los aspectos visuales del sitio web a través de una jerarquía de reglas de estilo.

Por último, JavaScript es un lenguaje de programación que permite modificar los elementos de una página web de forma dinámica. El hecho de que JavaScript sea un lenguaje de programación, y que HTML y CSS simplemente sean lenguajes, no es un dato menor. JavaScript permite agregar lógica de programación destinada a ejecutarse en el navegador web.

Es importante destacar la separación lógica entre contenido y estilo. En los principios del desarrollo web, estos dos aspectos estaban unificados, ya que el estilo se especificaba directamente sobre el documento HTML. La introducción de las hojas de estilo supuso una separación de las responsabilidades y una mejor organización al momento de desarrollar sitios web.

HTML tiene sus orígenes en XML, el cual pretende identificar entidades semánticas en el texto. Esto es, conocer cuál es el título del documento, quién es el autor, etc. Dado que, en los inicios de la web, la Internet consistía en el intercambio de documentos sin muchos agregados, HTML se creó bajo el mismo concepto.

Herramientas y entornos de desarrollo

En función de lo que conocemos sobre HTML hasta ahora, sabemos que son páginas de texto plano con marcas o etiquetas las cuales les dan valor semántico. Es por este motivo, que se puede utilizar cualquier editor de texto para crear archivos HTML, además, el navegador recibe a la página web tal cual está escrita, sin la necesidad de ser compilada o traducida.

En particular, los entornos de desarrollo y algunos editores de texto avanzados reconocen este lenguaje y distinguen el contenido común en contraposición con las marcas ayudando al desarrollador en su tarea de escritura. Es por esto, que la utilización de estas herramientas resuelta muy provechoso. Algunas herramientas muy utilizadas son Atom, Brackets, Sublime Text, Notepad++, entre otros editores de texto avanzados. También podemos nombrar algunos editores que poseen herramientas aún más avanzadas, por lo que son denominados Entornos de Desarrollo Integrados (IDEs). Algunos ejemplos de IDEs son Visual Studio Code, Visual Studio, Netbeans, entre otros.

FUNDAMENTOS DE HTML

Estructura básica de un documento HTML

HTML son las siglas de HyperText Markup Language, es decir, Lenguaje de Marcado de Hipertexto. Se denomina de esta forma porque en su concepción se basaba en la vinculación entre documentos de texto, en dónde la estructura se define con marcas que no son visibles en el navegador, pero si dividen el texto semánticamente. Hypertext viene de la palabra Hyperlink, que hace que se puedan vincular documentos distintos y “navegar” de uno al otro. Esto fue revolucionario en su época.

En resumen, HTML describe la estructura de página web usando un lenguaje de maquetado a base de marcas de texto. Las veremos a continuación.

Todos los elementos de HTML constituyen una página web, algunos son visibles y otros no. Estos elementos son representados mediante etiquetas en el código fuente de la página y no lo que muestra el navegador en la pantalla. Las etiquetas HTML representan partes del contenido como lo son los títulos, párrafos, tablas, etcétera. Cómo se dijo anteriormente, los navegadores no muestran las etiquetas HTML, sino que las utilizan para renderizar el contenido.

Ejemplo simple de una página HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
  <title>Mi primera
 página web</title>
</head>
<body>

  <h1>¡Hola, mundo!</h1>
  <p>Este es mi primer párrafo.</p>
</body>
</html>
```

A lo largo de la unidad, conoceremos el significado de cada línea de código.

Etiquetas HTML

Las etiquetas HTML son como las palabras clave que utilizamos para darle estructura y significado al contenido de una página web. Son como señales de tráfico que indican al navegador cómo debe interpretar y mostrar la información.

Imagina que estás escribiendo un libro. Utilizas diferentes tipos de letras, párrafos, títulos y otros elementos para organizar tu texto y hacerlo más legible. En HTML, las etiquetas cumplen una función similar, pero para el contenido de una página web.

Cada etiqueta tiene un nombre específico encerrado entre signos menor que (<) y mayor que (>). Por ejemplo, <p> se utiliza para definir un párrafo, <h1> para un encabezado de nivel 1, y así sucesivamente.

¿Cómo funcionan las etiquetas?

Las etiquetas suelen venir en pares: una etiqueta de apertura y una etiqueta de cierre. La etiqueta de apertura tiene el nombre de la etiqueta seguido de un signo mayor que (<), y la etiqueta de cierre tiene el nombre de la etiqueta precedido de una barra diagonal (/) y seguido de un signo mayor que (</).

Ejemplo:

```
<p>Este es un párrafo.</p>
```

En este ejemplo, es la etiqueta de apertura y es la etiqueta de cierre. Todo el texto entre estas dos etiquetas será mostrado como un párrafo en el navegador.

¿Para qué sirven las etiquetas?

Las etiquetas HTML sirven para:

- Definir la estructura: Indican al navegador cómo organizar el contenido de la página (encabezados, párrafos, listas, tablas, etc.).
- Dar formato: Permiten aplicar estilos al contenido, como cambiar el tamaño de la fuente, el color, la alineación, etc.
- Agregar interactividad: Se utilizan para crear elementos interactivos como botones, enlaces, formularios, etc.
- Incluir multimedia: Permiten insertar imágenes, videos y otros tipos de archivos multimedia en una página web.

Ejemplos de etiquetas HTML comunes:

- <html>: Define el inicio y el final de un documento HTML.
- <head>: Contiene metadatos sobre la página, como el título y enlaces a hojas de estilo.
- <body>: Contiene el contenido visible de la página.
- <h1> a <h6>: Definen encabezados de diferentes niveles.
- : Define un párrafo.
- <a>: Crea un enlace a otra página.
- <img>: Inserta una imagen.

- y <ol>: Crean listas no ordenadas y ordenadas, respectivamente.
- <table>: Crea una tabla.

Atributos de etiquetas

Los atributos de HTML son palabras especiales utilizadas dentro de la etiqueta de apertura para controlar el comportamiento del elemento. Los atributos de HTML son un modificador de un tipo de elemento de HTML, proporcionando funcionalidad adicional. En sintaxis HTML, un atributo se añade a una etiqueta de inicio de HTML.

Se reconocen varios tipos de atributos básicos, que incluyen: atributos requeridos, necesarios para que un tipo de elemento funcione correctamente; atributos opcionales, que modifican la funcionalidad por defecto de un tipo de elemento; atributos estándares, soportados por muchos tipos de elementos; y atributos de evento, utilizados para especificar guiones o script que se ejecutan bajo circunstancias concretas para cierto tipo de elementos.

Algunos tipos de atributo funcionan de manera diferente al ser utilizados para modificar diferentes tipos de elemento. Por ejemplo, el atributo name (nombre) se emplea en varios tipos de elementos, pero tiene funciones ligeramente diferentes en cada uno.

Los atributos de HTML generalmente se muestran como una pareja nombre-valor, separados por =, y están escritos dentro de la etiqueta de inicio de un elemento, después del nombre del elemento:

```
<element attribute="value">element content</element>
```

Dónde element nombra el tipo element del HTML, y atributte es el nombre del atributo puesto al value (valor) proporcionado. El valor puede estar encerrado en comillas simples o dobles, aunque hay valores compuestos de ciertos caracteres y que pueden quedar descomillados en HTML (pero no en XHTML).2 3 Dejar valores de atributo descomillados se considera inseguro.

A pesar de que la mayoría de los atributos se proporcionan como nombres y valores parejados, algunos afectan a la simplicidad del elemento con su presencia en la etiqueta de inicio del elemento (como el atributo ismap para el elemento img).

La mayoría de elementos pueden tomar cualquiera de varios de los atributos comunes:

- El atributo id (identidad) proporciona un identificador único en el ámbito de todo el documento para un elemento. Este puede ser utilizado como selector CSS para proporcionar propiedades presentationales, por los navegadores para centrar la atención en el elemento concreto, o por scripts (guiones) para alterar los contenidos o la presentación de un elemento. Anexado a la URL de la página, la URL directamente apunta el elemento concreto dentro del documento, típicamente una subsección de la página. Por ejemplo, los #ID "Atributtes" en http://en.wikipedia.org/wiki/html#Atributtes
- El atributo class proporciona una manera de clasificar elementos similares. Esto puede ser utilizado para propósitos semánticos, o para propósitos de presentación. Semantically, por ejemplo, las clases se utilizan en microformatos. Presentacionalmente, por ejemplo, un documento HTML podría utilizar la designaciónclass="notation", para indicar que todos los elementos con esta clase de valor están subordinados al texto principal del documento. Tales elementos podrían ser reunidos juntos y presentados como notas al pie en una página, en vez de aparecer en el sitio donde se encuentran en la fuente de HTML. Otro uso sería como selector CSS.
- Un autor puede utilizar propiedades no presentacionales de código atribucional de style para un elemento particular.El atributo de estilo se puede utilizar en cualquier elemento de HTML (se validará en cualquier elemento HTML; aun así, no es necesariamente útil). Se consiera mejor práctica usar los atributos iddo classde un elemento para seleccionar el elemento con una stylesheet (hoja de estilo), aunque a veces esto puede ser demasiado pesado para una aplicación de propiedades de estilo simple y específico o ad hoc.
- El atributo de title (título) se suele usar para unir explicación subtextual a un elemento. En la mayoría de los navegadores, este atributo se muestra como lo que a menudo se llama un tooltip o globo.

Los atributos HTML se clasifican generalmente en requeridos, opcionales, estándares (también denominados globales) y de evento. Normalmente los atributos requeridos y opcionales modifican elementos de HTML concretos, mientras que los atributos estándares pueden aplicarse a la mayoría de los elementos de HTML. Los atributos de evento, añadidos en versión HTML 4, permiten que un elemento especifique guiones o script que se ejecutaran en ciertas circunstancias.

Listas y tablas

Las listas HTML son una herramienta muy útil para organizar y presentar información de manera clara y concisa en una página web. Permiten agrupar elementos relacionados en una secuencia ordenada o no ordenada.

Tipos de listas HTML:

- Listas ordenadas (<ol>): Utilizan números para enumerar los elementos. Cada elemento de la lista se define con la etiqueta <li>.
- Listas no ordenadas (<ul>): Utilizan viñetas para enumerar los elementos. También se define cada elemento con la etiqueta <li>.
- Listas de definición (<dl>, <dt>, <dd>): Se utilizan para definir términos y sus descripciones. La etiqueta <dt> define el término y la etiqueta <dd> su descripción.

Ejemplo de lista ordenada:

```
<ol>
  <li>Manzana</li>
  <li>Banana</li>
  <li>Pera</li>
</ol>
```

Ejemplo de lista no ordenada:

```
<ul>
  <li>Rojo</li>
  <li>Verde</li>
  <li>Azul</li>
</ul>
```

Ejemplo de lista de definición:

```
<dl>
  <dt>HTML</dt>
  <dd>Lenguaje de marcado para crear páginas web.</dd>
  <dt>CSS</dt>
  <dd>Lenguaje de estilos para diseñar páginas web.</dd>
</dl>
```

Por otra parte, una tabla HTML es un elemento fundamental en el lenguaje HTML que nos permite organizar información en una estructura de filas y columnas.

Es como una hoja de cálculo, pero dentro de una página web. Se utiliza comúnmente para presentar datos de forma clara y concisa, como listas de productos, horarios, resultados de encuestas, etc.

La estructura básica de una tabla HTML se compone de las siguientes etiquetas:

- <table>: Define el inicio y final de la tabla.
- <tr>: Define una fila dentro de la tabla.
- <th>: Define una celda de encabezado (header).
- <td>: Define una celda de datos (data).

```
<table>
  <tr>
    <th>Nombre</th>
    <th>Apellido</th>
    <th>Edad</th>
  </tr>
  <tr>
    <td>Juan</td>
    <td>Pérez</td>
    <td>30</td>
  </tr>
  <tr>
    <td>María</td>
    <td>García</td>
    <td>25</td>

  </tr>
</table>
```

Explicando cada elemento

- <table>: Esta etiqueta envuelve toda la tabla, desde el encabezado hasta el último dato.
- <tr>: Cada fila de la tabla se define con la etiqueta <tr>. Dentro de cada fila, se colocan las celdas.
- <th>: Las celdas de encabezado se utilizan para identificar las columnas. Generalmente se muestran en negrita y centradas.
- <td>: Las celdas de datos contienen el contenido principal de la tabla.

Atributos útiles para personalizar tablas

- border: Define el grosor del borde de la tabla.
- cellpadding: Establece el espacio entre el contenido de una celda y su borde.
- cellspacing: Define el espacio entre las celdas.
- width: Establece el ancho de la tabla.
- height: Establece la altura de la tabla.
- align: Alinea el contenido de una celda (left, center, right).
- valign: Alinea verticalmente el contenido de una celda (top, middle, bottom).

Otras etiquetas importantes

- <caption>: Añade un título a la tabla.
- <thead>: Agrupa las filas de encabezado.
- <tbody>: Agrupa las filas de datos.
- <tfoot>: Agrupa las filas de pie de página.
- <col>: Define propiedades para una columna específica.
- <colgroup>: Agrupa un conjunto de columnas.

¿Por qué usar tablas HTML?

- Organización de datos: Presenta información de forma clara y estructurada.
- Accesibilidad: Facilita la lectura por parte de usuarios con discapacidad visual.
- Diseño: Permite crear diseños sencillos y efectivos.

Consideraciones importantes

- Evita usar tablas para diseño: Las tablas no están diseñadas para crear layouts complejos. Utiliza CSS para esto.
- Semantica: Utiliza las etiquetas correctas para cada elemento de la tabla.
- Accesibilidad: Asegúrate de que las tablas sean accesibles para todos los usuarios.

Formularios HTML

Un formulario HTML es un elemento fundamental en la creación de páginas web interactivas. Permite recopilar información del usuario a través de diversos campos de entrada, como cajas de texto, botones de radio, casillas de verificación, listas desplegables, entre otros. Esta información recopilada puede ser enviada a un servidor para su procesamiento, como en el caso de un formulario de contacto o un formulario de registro.

La estructura básica de un formulario HTML se define utilizando la etiqueta <form>. Dentro de esta etiqueta, se colocan los diferentes elementos de entrada que componen el formulario.

```
<form action="procesar_formulario.php" method="post">
  </form>
```

- <form>: Define el inicio y final del formulario.
- action: Especifica la URL a la que se enviarán los datos del formulario cuando se envíe.
- method: Indica el método HTTP utilizado para enviar los datos (generalmente get o post).

Elementos de entrada comunes

- <input>: El elemento más versátil. Se utiliza para crear diversos tipos de campos de entrada, como:
  - text: Caja de texto para ingresar texto.
  - password: Campo para ingresar contraseñas.
  - email: Campo para ingresar direcciones de correo electrónico.
  - number: Campo para ingresar números.
  - radio: Botón de radio para seleccionar una opción entre varias.
  - checkbox: Casilla de verificación para seleccionar múltiples opciones.
  - submit: Botón para enviar el formulario.
  - reset: Botón para restablecer los valores de los campos.
- <textarea>: Crea un área de texto multilineal.
- <select>: Crea una lista desplegable.
- <option>: Define cada opción dentro de una lista desplegable.

- <label>: Asocia un texto descriptivo a un elemento de entrada para mejorar la accesibilidad.

Ejemplo de un formulario sencillo

```
<form action="procesar_formulario.php" method="post">
  <label for="nombre">Nombre:</label>
  <input type="text" id="nombre" name="nombre">
  <br>
  <label
 for="email">Correo electrónico:</label>
  <input type="email" id="email" name="email">
  <br>
  <input type="submit" value="Enviar">
</form>
```

INTRODUCCION A CSS

Qué es CSS y cómo se aplica

CSS es una herramienta fundamental en el desarrollo web que te permite darle estilo y diseño a tus páginas HTML.

CSS son las siglas de "Cascading Style Sheets" o "Hojas de Estilo en Cascada" en español. Es un lenguaje de diseño que se utiliza para describir la presentación de un documento escrito en un lenguaje de marcado, como HTML. En otras palabras, CSS te permite definir cómo se verán los elementos de tu página web, como colores, fuentes, tamaños, espaciados, etc.

¿Para qué sirve CSS?

- Separación de contenido y presentación: CSS permite separar la estructura de una página (HTML) de su diseño (CSS), lo que hace que el código sea más organizado y fácil de mantener.
- Personalización: Puedes crear diseños únicos y personalizados para tus páginas web, adaptándolos a tus necesidades y preferencias.
- Responsividad: CSS te permite crear diseños que se adapten a diferentes tamaños de pantalla, como computadoras, tablets y teléfonos móviles.
- Accesibilidad: Puedes mejorar la accesibilidad de tus páginas web utilizando CSS para definir estilos que faciliten la lectura y la navegación para personas con discapacidades.

¿Cómo se usa CSS?

Hay varias formas de incluir CSS en un documento HTML:

- Hojas de estilo externas: Creas un archivo separado con extensión .css y lo enlazas a tu archivo HTML utilizando la etiqueta <link>. Esta es la forma más recomendada para organizar tu código.
- Estilos internos: Incluyes el código CSS dentro de la etiqueta <style> en el encabezado de tu documento HTML.
- Estilos en línea: Aplicas estilos directamente a elementos individuales utilizando el atributo style dentro de la etiqueta HTML.

Ejemplo de código CSS:

```css
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0;
}

h1 {
  color: blue;
  text-align: center;
}

p {
  font-size: 16px;
  line-height: 1.5;

}
```

En este ejemplo:

- body: Se selecciona todo el contenido del cuerpo de la página.
- h1: Se seleccionan todos los encabezados de nivel 1.
- p: Se seleccionan todos los párrafos.

¿Cómo funciona CSS?

CSS funciona seleccionando elementos HTML y aplicándoles reglas de estilo. Estas reglas se componen de un selector (que indica qué elemento se va a seleccionar) y una declaración (que define el estilo a aplicar).

```html
<h1 style="color: red;">Este es un encabezado rojo</h1>
```

En este ejemplo, el selector es h1 y la declaración es color: red;. Esto hará que el texto del encabezado sea de color rojo.

Selectores y propiedades básicas

Los selectores son la parte fundamental de CSS. Sirven para identificar los elementos HTML a los que se les aplicarán los estilos. Imaginemos los selectores como punteros que nos permiten señalar elementos específicos de nuestra página y decirles cómo deben verse.

Tipos de selectores:

- Selectores simples:
    - Por tipo de elemento: p (todos los párrafos), h1 (todos los encabezados de nivel 1).
    - Por clase: .miClase (todos los elementos con la clase "miClase").
    - Por ID: #miId (el elemento con el ID "miId").
- Selectores combinados:
    - Descendiente: div p (todos los elementos <p> dentro de un).
    - Hijo directo: div > p (solo los elementos <p> hijos directos de un).
    - Adyacente: h1 + p (el primer elemento <p> que sigue inmediatamente después de un <h1>).
    - Hermano: h1 ~ p (todos los elementos <p> que siguen a un <h1>, sin importar si hay otros elementos entre ellos).

Las Propiedades: Definiendo los Estilos

Una vez que hemos seleccionado los elementos, utilizamos las propiedades para definir cómo queremos que se vean. Las propiedades establecen características como el color, el tamaño de fuente, el margen, el padding, etc.

Ejemplos de propiedades comunes:

- color: Establece el color del texto.
- font-size: Define el tamaño de la fuente.
- background-color: Establece el color de fondo.
- width: Define el ancho de un elemento.
- height: Define la altura de un elemento.
- margin: Define los márgenes alrededor de un elemento.
- padding: Define el espacio interno de un elemento.
- border: Define el borde de un elemento.

Estructura básica de una regla CSS:

```css
selector {
  propiedad: valor;
  propiedad: valor;
  /* ... */
}
```

Ejemplo:

```css
p {
  color: blue;
  font-size: 16px;
}
```

Este código selecciona todos los elementos <p> y les asigna un color azul y un tamaño de fuente de 16 píxeles.

La Hoja de Estilos

Las reglas CSS se escriben en una hoja de estilos, que puede estar dentro del documento HTML (en la etiqueta <style>), en un archivo externo (con la extensión .css) o en línea (dentro de un atributo style de un elemento HTML).

Ejemplo de una hoja de estilos externa:

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <p>Este es un párrafo.</p>
</body>
</html>
```

Archivo styles.css:

```css
p {
  color: red;
}
```

Más allá de los Básicos

CSS ofrece una amplia gama de posibilidades para diseñar páginas web. Además de los selectores y propiedades básicos, existen:

- Unidades de medida: píxeles, ems, rems, porcentajes, etc.
- Funciones: para realizar cálculos y manipular valores.
- Propiedades abreviadas: para agrupar varias propiedades relacionadas.
- Pseudo-clases y pseudo-elementos: para seleccionar elementos en estados específicos o partes de ellos.
- Media queries: para crear diseños responsivos que se adapten a diferentes dispositivos.

Concepto de Layout: Box model

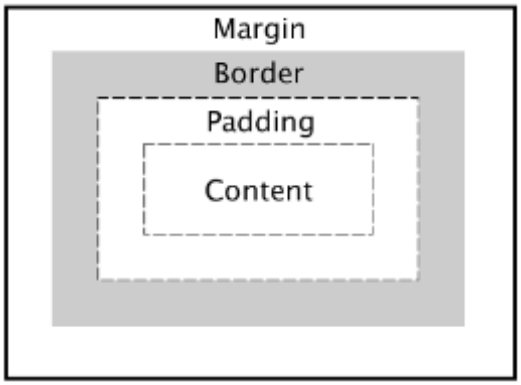
El modelo de caja (box model) es un concepto fundamental en CSS que te permite entender cómo se estructuran y posicionan los elementos en una página web. Imaginemos cada elemento HTML como una caja, y el modelo de caja define cómo se componen estas cajas y cómo interactúan entre sí.

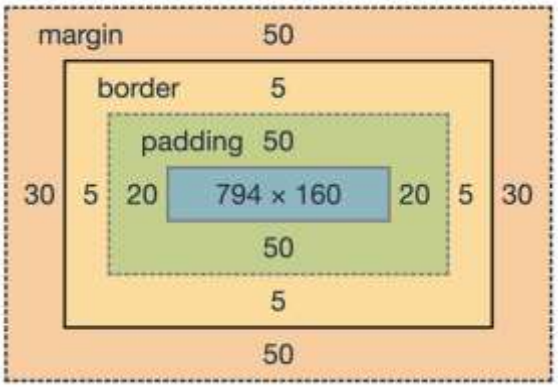
¿Qué compone una caja?

Cada caja en CSS está compuesta por cuatro partes principales:

- Contenido (content): Es el área donde se coloca el texto, imágenes u otros elementos.
- Relleno (padding): Es el espacio transparente que rodea al contenido dentro de la caja.
- Borde (border): Es la línea que delimita la caja.

- Margen (margin): Es el espacio transparente que rodea a la caja y separa a esta de otros elementos.





¿Cómo funciona el modelo de caja?

- Ancho y alto: Cuando establecemos el ancho y alto de una caja, por defecto solo estamos definiendo el tamaño del contenido. El relleno, borde y margen se añaden a este tamaño.
- box-sizing: Esta propiedad nos permite controlar cómo se calcula el ancho total de una caja. Los valores más comunes son:
    - content-box: (por defecto) El ancho y alto especificados se aplican solo al contenido.
    - border-box: El ancho y alto especificados incluyen el contenido, el relleno y el borde.

Otras propiedades:

- padding: Se utiliza para establecer el espacio interno de una caja.
- border: Define el estilo, ancho y color del borde.

- margin: Establece el espacio externo de una caja.

¿Por qué es importante el modelo de caja?

- Diseño de layouts: El modelo de caja es la base para crear diseños complejos, desde diseños simples de una columna hasta diseños más elaborados con múltiples columnas y elementos flotantes.
- Control sobre el espacio: Te permite controlar el espacio entre elementos, el tamaño de los elementos y su posición en la página.
- Solución de problemas: Comprender el modelo de caja te ayudará a solucionar problemas de diseño comunes, como elementos que se superponen o que no tienen el tamaño esperado.

Ejemplo práctico

```CSS
.caja {
  width: 200px;
  height: 150px;
  border: 2px solid black;
  padding: 10px;
  margin: 20px;
}
```

En este ejemplo:

- La caja tendrá un ancho total de 240px (200px de contenido + 20px de padding a cada lado + 4px de borde a cada lado).
- La caja tendrá una altura total de 190px.
- La caja estará separada de otros elementos por un margen de 20px en cada lado.

¿Qué más necesitas saber?

- Modelo de caja en diferentes navegadores: Aunque el modelo de caja es un estándar, puede haber pequeñas diferencias en su implementación entre los diferentes navegadores.

- Propiedades avanzadas: Existen otras propiedades relacionadas con el modelo de caja, como box-shadow para agregar sombras, y overflow para controlar cómo se comporta el contenido que excede el tamaño de la caja.
- Diseño responsivo: El modelo de caja es fundamental para crear diseños que se adapten a diferentes tamaños de pantalla.

Introducción a Bootstrap

Bootstrap es un framework de desarrollo web, gratuito y de código abierto, diseñado para crear sitios web y aplicaciones web responsivos y visualmente atractivos. En términos más simples, es una colección de herramientas prediseñadas (como botones, formularios, tablas, etc.) y estilos CSS que puedes utilizar para acelerar el proceso de desarrollo de tu sitio web.

¿Por qué es tan popular?

- Responsividad: Bootstrap está diseñado con la responsividad en mente, lo que significa que tus sitios web se adaptarán automáticamente a diferentes tamaños de pantalla (desde smartphones hasta pantallas de escritorio).
- Rápido y fácil de usar: Al proporcionar componentes prediseñados y una estructura de cuadrícula flexible, Bootstrap te permite crear diseños complejos de manera rápida y sencilla.
- Amplia comunidad: Cuenta con una gran comunidad de desarrolladores que contribuyen con nuevos componentes, temas y documentación, lo que lo convierte en una herramienta muy completa y actualizada.
- Consistente: Al utilizar Bootstrap, aseguras que todos tus proyectos tengan un aspecto y una sensación similares, lo que facilita la creación de interfaces de usuario coherentes.

¿Cómo funciona Bootstrap?

- Componentes prediseñados: Bootstrap incluye una amplia variedad de componentes HTML, CSS y JavaScript, como botones, formularios, navegaciones, carruseles, etc. Estos componentes están diseñados para funcionar de forma consistente y son altamente personalizables.
- Sistema de cuadrícula: Bootstrap utiliza un sistema de cuadrícula flexible para organizar el contenido en una página. Este sistema te permite crear diseños responsivos de forma fácil y eficiente.

- Clases CSS: Los componentes y estilos de Bootstrap se aplican a través de clases CSS. Al agregar estas clases a tus elementos HTML, puedes personalizar rápidamente su apariencia y comportamiento.

Ventajas de utilizar Bootstrap:

- Ahorra tiempo: Al utilizar componentes prediseñados, puedes acelerar significativamente el desarrollo de tu proyecto.
- Diseño consistente: Bootstrap te ayuda a crear diseños visualmente atractivos y coherentes.
- Responsividad: Tus sitios web se adaptarán automáticamente a diferentes dispositivos.
- Gran comunidad: Existe una gran comunidad de desarrolladores que pueden ayudarte si tienes alguna pregunta.

En resumen, Bootstrap es una herramienta invaluable para cualquier desarrollador web que quiera crear sitios web modernos y responsivos de manera eficiente. Al proporcionar una base sólida y una amplia gama de componentes, Bootstrap te permite concentrarte en la lógica de tu aplicación en lugar de reinventar la rueda.

Diseño responsivo con Bootstrap

El diseño responsivo con Bootstrap es una de sus características más poderosas. Permite que tus sitios web se adapten automáticamente a diferentes tamaños de pantalla, desde teléfonos móviles hasta monitores de escritorio, brindando una experiencia de usuario óptima en cualquier dispositivo.

¿Qué es el diseño responsivo?

El diseño responsivo es una técnica que hace que un sitio web se ajuste y se reajuste para adaptarse a cualquier tamaño de pantalla. Esto se logra utilizando media queries (consultas de medios) en CSS, que permiten aplicar estilos diferentes según el ancho de la pantalla.

¿Cómo funciona el diseño responsivo con Bootstrap?

Bootstrap facilita enormemente el diseño responsivo gracias a su sistema de cuadrícula y a las clases CSS que se adaptan automáticamente a diferentes tamaños de pantalla.

- Sistema de cuadrícula: Bootstrap utiliza un sistema de 12 columnas que se ajusta de forma fluida a diferentes anchos de pantalla. Puedes dividir tu contenido en columnas y controlar cómo se distribuyen en diferentes dispositivos.
- Puntos de quiebre: Bootstrap define puntos de quiebre (breakpoints) que corresponden a anchos de pantalla específicos. En estos puntos, el diseño se reajusta para adaptarse a la nueva resolución. Los puntos de quiebre más comunes en Bootstrap son:
    - xs: Extra small (muy pequeñas pantallas, como teléfonos móviles)
    - sm: Small (pequeñas pantallas, como tablets)
    - md: Medium (pantallas medianas, como laptops)
    - lg: Large (pantallas grandes, como monitores de escritorio)
    - xl: Extra large (pantallas extra grandes)
- Clases CSS responsivas: Bootstrap proporciona una amplia variedad de clases CSS que te permiten controlar la apariencia de los elementos en diferentes puntos de quiebre. Por ejemplo, puedes ocultar un elemento en pantallas pequeñas, o cambiar el tamaño de una columna en pantallas grandes.

Ejemplo práctico

Imagina que quieres crear una página con un menú de navegación en la parte superior y un contenido principal en la parte inferior. En pantallas grandes, el menú y el contenido se muestran uno al lado del otro. En pantallas pequeñas, el menú se colapsa y se muestra en un menú desplegable.

HTML

```html
<div class="container">
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    </nav>
  <div class="row">
    </div>
</div>
```

CSS

```css
/* En pantallas pequeñas, el contenido se apila uno encima del otro */
@media (max-width: 767px) {
  .navbar-expand-lg {
    flex-direction: column;
  }
}
```

En este ejemplo:

- La clase navbar-expand-lg hace que el menú se expanda en pantallas grandes.
- La media query @media (max-width: 767px) se aplica a pantallas con un ancho máximo de 767 píxeles (punto de quiebre sm).
- Dentro de la media query, cambiamos la dirección de flexbox a columna para que el menú se muestre debajo del contenido en pantallas pequeñas

Beneficios del diseño responsivo con Bootstrap

- Experiencia de usuario mejorada: Los usuarios pueden acceder a tu sitio web desde cualquier dispositivo y encontrar una interfaz fácil de usar y visualmente atractiva.
- Mayor alcance: Al ser accesible desde una amplia gama de dispositivos, tu sitio web puede llegar a una audiencia más amplia.
- SEO: Los motores de búsqueda valoran los sitios web responsivos, lo que puede mejorar tu posicionamiento en los resultados de búsqueda.
- Desarrollo más rápido: Bootstrap te proporciona las herramientas y componentes necesarios para crear diseños responsivos de forma rápida y eficiente.

INTRODUCCION A JAVASCRIPT

En las secciones anteriores hemos visto los conceptos de redes de computadoras necesarios para entender cómo funciona la web. También vimos que los navegadores sólo entienden tres lenguajes que son HTML, CSS y JavaScript. Además, habiéndonos introducido en HTML y CSS, ya somos capaces de realizar el maquetado de una página web, es decir, definir su estructura y sus estilos, todos los aspectos a visualizar en el navegador.

Las herramientas vistas sólo nos permiten hacer páginas web estáticas, es decir, sin funcionalidad adicional. Es decir, la página se renderiza (se muestra) con estructura y estilo propio, pero no presenta ningún tipo de funcionalidad, no poseen lógica de programación ni respuestas a eventos.

Los eventos en una página web son bloques de código que se ejecutan como respuesta a acciones que realizamos sobre controles visuales, por ejemplo, la respuesta al click de un botón. También nos referimos a cambios en la visualización, como, por ejemplo, al pasar con el cursor sobre encima de algún elemento podríamos modificar su estilo cambiando las clases asociadas a los elementos. Todas estas acciones, y más, las realizaremos a través de JavaScript.

La importancia de ver este lenguaje deriva de que, virtualmente, cada computadora personal (PC) en el mundo tiene al menos un intérprete de JavaScript instalado y es de uso activo. La popularidad de JavaScript se debe enteramente a su rol como lenguaje de scripting (secuencia de comandos) de la WWW (web).

JavaScript es un lenguaje **diferente** a Java.

Si bien JavaScript tiene una sintaxis similar a Java no significa que tenga algo que ver con Java. Es más, hasta Java tiene una sintaxis similar a C# y esto no quiere decir que Java sea C#.

JavaScript fue creado por Brendan Eich en 1995 para la empresa (y navegador popular es su época) Netscape. Como Java era un lenguaje nuevo en esa era, con mucha aceptación y popularidad, se optó por utilizar parte del nombre para ganar adeptos.

Por último y para cerrar este apartado, vamos a decir que JavaScript está estandarizado en ECMA International (Asociación Europea para la Creación de Estándares para la Comunicación y la Información) con el fin de ofrecer un lenguaje de programación estandarizado e internacional basado en JavaScript y llamado ECMAScript para que todos los navegadores puedan realizar un intérprete que acepte la misma sintaxis de JavaScript.

El núcleo de JavaScript puede extenderse para varios propósitos, complementándolo con objetos adicionales, por ejemplo:

- Client-side JavaScript (JS del lado del cliente) extiende el núcleo del lenguaje proporcionando objetos para controlar un navegador y su modelo de objetos (o DOM, por las iniciales de Document Object Model). Por ejemplo, las extensiones del lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, tales como clicks del mouse, ingreso de datos al formulario y navegación de páginas.
- Server-side JavaScript (JS del lado del servidor) extiende el núcleo del lenguaje proporcionando objetos relevantes a la ejecución de JavaScript en un servidor. Por ejemplo, las extensiones del lado del servidor permiten que una aplicación se comunique con una base de datos, proporcionar continuidad de la información de una invocación de la aplicación a otra, o efectuar manipulación de archivos en un servidor.

Sintaxis del lenguaje

A continuación, vamos a ver la sintaxis del lenguaje JS (JavaScript) para familiarizarnos con la forma en que se escribe. Seguiremos la documentación de Mozilla sobre el lenguaje,

JavaScript es case-sensitive, o sea que distingue entre mayúsculas y minúsculas. Es decir, se puede declarar una variable como Pajaro que será distinta a pajaro.

Respecto a las instrucciones, en JavaScript son llamadas sentencias y son separadas por punto y coma (;).

Observación: Un punto y coma no es necesario al final si está escrita en una sóla línea. Es más, el intérprete según el estándar ECMAScript tiene reglas para insertar automáticamente los punto y coma al ser ejecutadas las sentencias, pero hay casos en donde no queda claro en donde se insertarán esos puntos y comas. Por lo tanto, se recomienda añadir siempre los punto y coma.

Comentarios

La sintaxis de comentarios es la misma que en otros lenguajes.

```
// comentario en una sola línea
/* este es un comentario
   multilínea
*/
/* no puedes, sin embargo, /* anidar comentarios */ SyntaxError */
```

Declaraciones

Hay tres tipos de declaraciones en JavaScript.

var declara una variable, inicializándola opcionalmente a un valor. Si bien ésta ha sido exclusivamente la forma de declarar variables durante mucho tiempo sufre de efectos no deseados en el alcance (scope) de la declaración. En otras palabras, se aconseja no utilizarla ya que el estándar nuevo de JavaScript permite otras formas de declaración que corrige este problema.

El alcance o bloque de ámbito es el lugar en donde es visible esa variable. Por ejemplo, si se declara una variable dentro de una función, esta variable mantiene su valor y es visible sólo dentro de una función.

let declara una variable local en un bloque de ámbito (scope), inicializándola opcionalmente a un valor.

const declara una constante de sólo lectura en un bloque de ámbito.

Variables

Las variables se usan como nombres simbólicos para valores en tu aplicación. Los nombres de las variables, llamados identificadores, se rigen por ciertas reglas.

Un identificador en JavaScript tiene que empezar con una letra, un guión bajo (_) o un símbolo de dólar ($); los valores subsiguientes pueden ser números. Debido a que JavaScript diferencia entre mayúsculas y minúsculas, las letras incluyen tanto desde la "A" hasta la "Z" (mayúsculas) como de la "a" hasta la "z".

Algunos ejemplos de nombre permitidos son Numero_Visitas, temp99, _nombre, $nombre

Ámbito de variable

Cuando declaras una variable fuera de una función, se le denomina variable global, porque está disponible para cualquier otro código en el documento actual. Cuando declaras una variable dentro de una función, se le denomina variable local, porque está disponible sólo dentro de esa función donde fue creada.

Antes de ECMAScript 6, JavaScript no tiene ámbito de sentencias de bloque; más bien, una variable declarada dentro de un bloque es local para la función (o ámbito global) en la que reside el bloque. Por ejemplo, el siguiente código registrará 5, porque el ámbito de x es la función (o contexto global) dentro del cual se declara x, no el bloque, que en este caso es la sentencia if.

```
if (true) {
  var x = 5;
}
console.log(x); // x vale 5
```

Constantes

Una constante funciona como una variable, pero no cambia su valor, es decir, representa un lugar de almacenamiento de tipos de datos en la memoria, pero una vez asignado el valor inicial este no puede ser modificado.

Es de sólo lectura y se define con la palabra clave const.

La sintaxis de la definición del identificador es el mismo que el de las variables.

```
const PI = 3.14;
```

Una constante no puede cambiar su valor mediante la asignación o volver a declararse mientras se ejecute el script.

Las reglas de ámbito para las constantes son las mismas que las de las variables let en un ámbito de bloque. Si la palabra clave const es omitida, el identificador se asume que representa una variable.

No puedes declarar una constante con el mismo nombre que una función o una variable en el mismo ámbito. Por ejemplo:

```
// ESTO CAUSARÁ UN ERROR
function f() {};
const f = 5;
// ESTO TAMBIÉN CAUSARÁ UN ERROR
function f() {
  const g = 5;
  var g;
  //sentencias
}
```

Estructura de datos y tipos

El último estándar ECMAScript define ocho tipos de datos:

Siete tipos de datos que son primitivos:

- Boolean. true y false.
- null. Una palabra clave especial que denota un valor nulo. Como JavaScript es case-sensitive, null no es lo mismo que Null, NULL, o cualquier otra variante.
- undefined. Una propiedad de alto nivel cuyo valor no es definido.
- Number. Un número entero o un número con coma flotante. Por ejemplo: 42 o 3.14159.
- BigInt. Un número entero con precisión arbitraria. Por ejemplo: 9007199254740992n
- String. Una secuencia de caracteres que representan un valor "Hola"
- Symbol (nuevo en ECMAScript 6). Un tipo de dato cuyos casos son únicos e inmutables y Object.

Aunque estos tipos de datos son una cantidad relativamente pequeña, permiten realizar funciones útiles con tus aplicaciones. Los otros elementos fundamentales en el lenguaje son los Objects y las funciones. Puedes pensar en objetos como contenedores con nombre para los valores, y las funciones como procedimientos que puede realizar tu aplicación.

Conversión de tipos de datos

JavaScript es un lenguaje de tipo dinámico. Esto significa que no tienes que especificar el tipo de dato de una variable cuando la declaras, y los tipos de datos son convertidos automáticamente de acuerdo con lo que se necesite en la ejecución del script. Así, por ejemplo, puedes definir una variable de la siguiente manera:

```
var respuesta = 42;
```

Y luego, puedes asignarle una cadena a esa misma variable, por ejemplo:

```
respuesta = "Gracias por el regalo...";
```

Debido a que es un lenguaje de tipos dinámicos, esta asignación no causa un mensaje de error.

Convertir string a números

En el caso que un valor representando un número está en memoria como string, hay métodos para la conversión.

parseInt() y parseFloat()

parseInt sólo retornará números enteros, por lo que su uso es disminuido por los decimales.

Literales

Los literales se utilizan para representar valores en JavaScript. Estos son valores fijos, no variables, que literalmente proporciona en su script.

Se usan para inicializar variables más convenientemente. O sea, en otros lenguajes de programación si queremos asignar una lista a una variable y llenarla, primero debemos instanciar el objeto lista y luego agregarle elementos. En Javascript se puede hacer directamente con una sola sentencia.

Literales Array

Un literal array es un lista de cero o más expresiones, cada uno representa un elemento array, entre corchetes ([ ]). Cuando crea un array usando un literal array, se inicializa con los valores especificados como sus elementos, y su longitud se establece por el número de argumentos especificados.

El siguiente ejemplo crea el array cafes con tres elementos y una longitud de tres:

```
var cafes = ["Tostado Frances", "Colombiano", "Kona"];
```

Comas adicionales en literales array

No tienes que especificar todos los elementos en un literal array. Si pones dos comas en una fila, se crea el array con undefined para los elementos no especificados. Undefined es un tipo de datos que indica que la variable no ha sido inicializada. El siguiente ejemplo crea el array peces:

```
var peces = ["Leon", , "Angel"];
```

Este array tiene dos elementos con valores y un elemento vacío (peces[0] es "Leon", peces[1] es undefined, y peces[2] es "Angel").

Si incluyes una coma al final de la lista de los elementos, la coma es ignorada. En el siguiente ejemplo, la longitud del array es tres. No hay miLista[3]. Todas las demás comas en la lista indican un nuevo elemento. (Nota: Las comas finales pueden crear errores en las versiones anteriores del navegador y es una buena práctica eliminarlos.)

```
var miLista = ['casa', , 'escuela', ];
```

En el siguiente ejemplo, la longitud del array es cuatro, y miLista[0] y miLista[2] faltan.

```
var miLista = [ , 'casa', , 'escuela'];
```

En el siguiente ejemplo, la longitud del array es cuatro, y miLista[1] y miLista[3] faltan. Solo la última coma es ignorada.

```
var miLista = ['casa', , 'escuela', , ];
```

Comprender el comportamiento de las comas adicionales es importante para comprender JavaScript como un lenguaje, sin embargo, cuando escribimos nuestro propio código: declaramos explícitamente los elementos que faltan como undefined esto aumentará la claridad y el mantenimiento de su código.

Literales Booleanos

Los literales de tipo Booleanos tienen 2 valores posibles: true y false.

NO confundir los valores primitivos Booleanos true y false con los valores true y false del Objeto Booleano. El objeto Booleano es un contenedor alrededor del tipo de dato Primitivo Booleano.

Tener en cuenta que existen otros tipos de literales, como de punto flotante, de tipo objeto, enteros, string, etc. Será responsabilidad del alumno investigar sobre dichos temas.

Funciones y eventos

¿Qué es una función?

Una función es un bloque de código reutilizable que realiza una tarea específica. Es como una receta: le proporcionas ciertos ingredientes (parámetros) y ella te devuelve un resultado (valor de retorno).

¿Para qué sirven las funciones?

- Organizar el código: Dividir el código en funciones más pequeñas hace que sea más fácil de leer, entender y mantener.
- Reutilizar código: Una vez que has creado una función, puedes llamarla desde diferentes partes de tu código, evitando repetir el mismo código varias veces.
- Abstracción: Las funciones te permiten ocultar la complejidad de ciertas tareas, permitiendo que otros desarrolladores utilicen tu código sin necesidad de entender todos los detalles de su implementación.

Sintaxis básica de una función:

```javascript
function nombreDeLaFuncion(parametro1, parametro2, ...) {
  // Código que realiza la tarea de la función
  return valorDeRetorno; // Opcional
}
```

Ejemplo:

```javascript
function saludar(nombre) {
  console.log("Hola, " + nombre + "!");
}

saludar("Juan"); // Imprime "Hola, Juan!" en la consola
```

¿Qué es un evento?

Un evento es una acción que ocurre en una página web, como hacer clic en un botón, mover el ratón, cargar una página, etc. JavaScript te permite detectar estos eventos y ejecutar código en respuesta a ellos.

¿Para qué sirven los eventos?

Los eventos hacen que tus páginas web sean interactivas. Por ejemplo, puedes:

- Mostrar un mensaje cuando un usuario hace clic en un botón.
- Validar un formulario cuando el usuario envía los datos.
- Crear animaciones cuando el usuario mueve el ratón sobre un elemento.

Cómo manejar eventos:

1. Seleccionar el elemento: Utiliza métodos como getElementById, querySelector o querySelectorAll para seleccionar el elemento al que deseas agregar un manejador de eventos.
2. Asignar el manejador de eventos: Utiliza métodos como addEventListener para asociar una función (manejador de eventos) a un evento específico del elemento.

Ejemplo:

```html
<button id="miBoton">Haz clic aquí</button>
```

```javascript
const boton = document.getElementById("miBoton");

boton.addEventListener("click", function() {
  alert("¡Has hecho clic en el botón!");
});
```

Relación entre funciones y eventos

Los manejadores de eventos son, en esencia, funciones que se ejecutan cuando ocurre un evento. Al hacer clic en el botón, se invoca la función asociada al evento "click", y el código dentro de esa función se ejecuta.

En resumen:

- Funciones: Bloques de código reutilizables que realizan tareas específicas.
- Eventos: Acciones que ocurren en una página web.
- Manejadores de eventos: Funciones que se ejecutan en respuesta a un evento.

Manejo de DOM

El DOM es una interfaz de programación (API) que trata a un documento HTML como un árbol de objetos. Cada elemento HTML (como <html>, <head>, <body>, , etc.) es un nodo en este árbol. JavaScript nos proporciona métodos y propiedades para acceder y modificar estos nodos.

¿Cómo interactuar con el DOM desde JavaScript?

1. Acceder a elementos:

- getElementById: Obtiene un elemento por su ID.
- querySelector: Obtiene el primer elemento que coincida con un selector CSS.
- querySelectorAll: Obtiene una lista de todos los elementos que coincidan con un selector CSS.

Ejemplo:

```javascript
const miParrafo = document.getElementById("miParrafo");
const todosLosBotones = document.querySelectorAll("button");
```

2. Modificar el contenido:

- textContent: Cambia el texto dentro de un elemento.
- innerHTML: Cambia todo el contenido HTML dentro de un elemento.

Ejemplo:

```javascript
miParrafo.textContent = "Nuevo contenido";
```

4. Crear nuevos elementos:

- createElement: Crea un nuevo elemento HTML.
- appendChild: Agrega un elemento hijo a un elemento padre.

Ejemplo:

```javascript
const nuevoParrafo = document.createElement("p");
nuevoParrafo.textContent = "Este es un párrafo nuevo";
document.body.appendChild(nuevoParrafo);
```

5. Manejar eventos:

- addEventListener: Asocia una función a un evento de un elemento.

Ejemplo:

```javascript
const boton = document.getElementById("miBoton");
boton.addEventListener("click", function() {
  alert("¡Has hecho clic en el botón!");
});
```

Ejemplos más complejos

- Crear una lista dinámica:

```javascript
const lista = document.getElementById("miLista");
const nuevoItem = document.createElement("li");
nuevoItem.textContent = "Nuevo elemento";
lista.appendChild(nuevoItem);
```

- Cambiar el estilo de un elemento al pasar el mouse:

```javascript
const elemento = document.getElementById("miElemento");
elemento.addEventListener("mouseover", function() {
  this.style.backgroundColor = "yellow";
});
elemento.addEventListener("mouseout", function() {
  this.style.backgroundColor = "white";
});
```

Consideraciones importantes

- Rendimiento: Manipular el DOM puede ser costoso en términos de rendimiento. Evita realizar demasiadas modificaciones al DOM en un bucle o en respuesta a un evento.
- Cross-browser compatibility: Asegúrate de que tu código funcione en diferentes navegadores, ya que pueden existir pequeñas diferencias en la implementación del DOM.
- Bibliotecas: Existen bibliotecas como jQuery que facilitan la manipulación del DOM, proporcionando una sintaxis más concisa y funciones adicionales.

En resumen, el DOM es la puerta de entrada para crear páginas web interactivas con JavaScript. Al comprender cómo acceder y modificar los elementos del DOM, podrás crear aplicaciones web dinámicas y personalizadas.

Integración con back-end

La integración con el back-end es fundamental para crear aplicaciones web dinámicas que interactúen con datos externos. Fetch es una API moderna de JavaScript que nos permite realizar solicitudes HTTP de forma asíncrona, ideal para comunicarse con un back-end.

¿Qué es la integración con el back-end?

La integración con el back-end se refiere a la conexión entre la interfaz de usuario (frontend) de una aplicación web y el servidor (back-end) que almacena y procesa los datos. El front-end envía solicitudes al back-end para obtener o modificar datos, y el back-end responde con los resultados.

¿Por qué es importante la integración con el backend?

- Datos dinámicos: Permite que el contenido de una página web se actualice sin necesidad de recargar toda la página.
- Interactividad: Facilita la creación de aplicaciones interactivas que responden a las acciones del usuario.
- Escalabilidad: Separa la lógica de presentación (front-end) de la lógica de negocio (back-end), lo que permite escalar cada parte de forma independiente.
- Seguridad: Permite proteger los datos sensibles del usuario almacenándolos en el servidor.

¿Cómo funciona la integración con el back-end usando Fetch?

1. Hacer una solicitud:

- Se crea una nueva instancia de fetch y se le pasa la URL del recurso que se desea obtener.
- Se puede especificar un método HTTP (GET, POST, PUT, DELETE, etc.) y otros encabezados adicionales.

2. Procesar la respuesta:

- La promesa devuelta por fetch se resuelve cuando se recibe la respuesta del servidor.
- Se utiliza el método json() para convertir la respuesta a formato JSON (si el servidor devuelve JSON).

Ejemplo: Obtener datos de un API y mostrarlos en una página

```javascript
fetch('https://api.example.com/usuarios')
  .then(response => response.json())
  .then(data => {
    // Procesar los datos y mostrarlos en la página
    const listaUsuarios = document.getElementById('lista-usuarios');
    data.forEach(usuario => {
      const li = document.createElement('li');
      li.textContent = usuario.nombre;
      listaUsuarios.appendChild(li);
    });
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

En este ejemplo:

- Hacemos una solicitud GET a la URL https://api.example.com/usuarios para obtener una lista de usuarios.
- Convertimos la respuesta a formato JSON.
- Iteramos sobre los datos y creamos elementos de lista para mostrar los nombres de los usuarios.
- Si ocurre un error, lo mostramos en la consola.

Otros aspectos para considerar

- Métodos HTTP: Además de GET, puedes utilizar otros métodos como POST para enviar datos al servidor, PUT para actualizar datos y DELETE para eliminar datos.
- Encabezados: Puedes incluir encabezados adicionales en la solicitud, como el tipo de contenido (Content-Type) o un token de autenticación.

- Cuerpo de la solicitud: Para enviar datos en el cuerpo de una solicitud POST, puedes utilizar el método body de la opción de fetch.
- Asincronía: Las operaciones de red son asíncronas, por lo que es importante manejar las promesas correctamente para evitar bloquear la ejecución del código.
- Manejo de errores: Siempre debes incluir un bloque catch para manejar posibles errores que puedan ocurrir durante la solicitud o el procesamiento de los datos.

Integración con frameworks

Frameworks como React, Angular y Vue.js ofrecen herramientas y abstracciones para simplificar la integración con el back-end. Por ejemplo, en React, puedes utilizar hooks como useEffect para realizar solicitudes HTTP y actualizar el estado de los componentes.

Seguridad en APIs: Autenticación y Autorización con JWT

Tal como se ha desarrollado en unidades anteriores, sabemos que una API es un conjunto de servicios que pueden ser accedidos a través de la web. Los Endpoints de una API pueden ser ejecutados desde un navegador a través de su URL, desde herramientas web como Swagger y desde herramientas externas como Postman.

Es necesario tomar conciencia sobre lo expuestos que se encuentran estos servicios y lo sencillo que sería acceder a ellos por parte de un tercero no autorizado. Esto le brindaría libre acceso a información confidencial y a todas las funcionalidades que nuestra API contenga.

Lógicamente, hay mecanismos para controlar esta situación. Estos mecanismos concretamente buscan cumplir con dos funciones básicas de la seguridad informática: la autenticación y la autorización.

Autenticación

La autenticación pretende verificar la identidad del usuario que realiza la llamada a nuestra API. Este paso busca conocer quién es el usuario, y validar ciertamente que sea quien dice ser. Este concepto lo podemos ver aplicado en cualquier sitio web, donde iniciemos sesión. Para este proceso utilizamos un nombre de usuario que nos identifica unívocamente y una contraseña, que certifica que realmente somos ese usuario.

Autorización

La autorización se encarga de verificar que el usuario, ya autenticado, esté autorizado para acceder a este recurso. Este paso es de extrema importancia en sistemas con fuertes sistemas de permisos, los cuales brindan acceso a determinados formularios, funcionalidades, información, según el usuario posea los permisos para ello o no.

JWT

JSON Web Token, normalmente abreviado como JWT, es un estándar para la creación de tokens de acceso basados en JSON. JWT define campos comunes y les brinda un significado estandarizado, esto facilita la integración entre diferentes sistemas, ya que todos manejan el mismo conjunto de campos. Estos campos son denominados Claims. Algunos de los Claims más utilizados son:

- JTI: identificador único autogenerado que le brinda unicidad a cada token.
- IAT: fecha en que el token fue emitido.
- EXP: fecha en la que el token pierde validez.

Adicionalmente, podemos crear nuestros propios Claims personalizados según diferentes aspectos del negocio. Por ejemplo, supongamos que tenemos un sistema para llevar el stock en un depósito que es utilizado por diferentes empresas, en tal caso podríamos querer que cada empresa utilice tokens fácilmente diferenciables. En este caso y similares, tiene sentido agregar Claims personalizados.

Implementando JWT en nuestra API

Para lograr una integración completa de JWT en nuestra API, es necesario llevar a cabo tres pasos:

- Desarrollar un nuevo Endpoint que nos permita generar un token de acceso.
- Modificar nuestro Program.cs para configurar los mecanismos de autenticación.
- Modificar los Controllers a los que se desea proteger el acceso.

Para empezar, necesitamos uno o más Endpoints que permitan al usuario autenticarse y así obtener su token. En nombre de la simplicidad, vamos a plantear un Endpoint el cual reciba un objeto UserLogin, el cual posee únicamente dos atributos: Username y Password.

A partir de estas credenciales recibidas, nuestro Endpoint simplemente validará que coincidan con las credenciales hardcodeadas y generará un token acorde. Caso contrario emitirá un código de error 401, sin autorización.

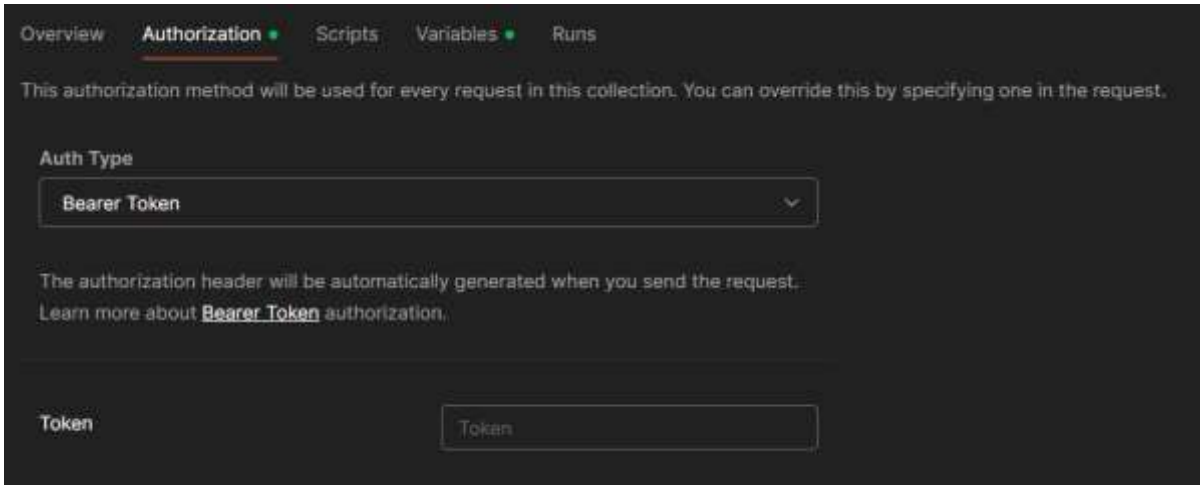
```
[Route("api/login")]
[ApiController]
public class JwtController : ControllerBase
{
    [HttpPost]
    public IActionResult Login([FromBody] UserLogin objUserLogin)
    {
        if (objUserLogin.Username == "testUsername"
            && objUserLogin.Password == "testPassword")
        {
            var NewToken = GenerateJwtToken(userLogin.Username);
            return Ok(new { Token = NewToken });
        }
    }

    private string GenerateJwtToken(string Username)
    {
        var SecurityKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes("MiClaveSuperSecreta")
        );

        var Credentials = new SigningCredentials(
            SecurityKey, SecurityAlgorithms.HmacSha256
        );

        var IdentificadorUnico = Guid.NewGuid().ToString();

        var Claims = new [] {
            new Claim(JwtRegisteresClaimNames.Sub, Username),
            new Claim(JwtRegisteredClaimNames.Jti, IdentificadorUnico)
        };

        var Token = new JwtSecurityToken(
            issuer: "MiAPI",
            audience: "MiAPI",
            claims: Claims,
            expires: DateTime.Now.AddMinutes(60),
            signingCredentials, Credentials
        );

        return new JwtSecurityTokenHandler().WriteToken(Token);
    }
}
```

A continuación, es necesario modificar el Program.cs para agregar los servicios correspondientes de autorización y autenticación. De esta forma, estamos parametrizando los servicios de JWT, indicándoles las claves y los parámetros Issuer y Audience aceptables.

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = "MiAPI",
            ValidAudience = "MiAPI",
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes("MiClaveSuperSecreta")
            )
        };
    });

builder.Services.AddAuthorization();

var app = builder.Build();

app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Finalmente, debemos indicar qué Controllers y Endpoints requieren autenticación, a esto lo logramos con el decorador [Authorize]. Este decorador puede ser agregado sobre una clase Controller, provocando así la autorización de todos sus Endpoints, o puede agregarse sobre un Endpoint específico, para forzar la autorización de ese Endpoint.

Ahora podemos ver que al ejecutar los Endpoints securizados desde Postman, nos devuelve un error 401 Unauthorized. Es necesario, llamar manualmente al Endpoint que nos entrega el token, copiarlo e insertarlo en la configuración de Postman, como se indica en la imagen.



Conclusión

La integración con el back-end es una parte fundamental del desarrollo web moderno. El método fetch() es una herramienta poderosa y fácil de usar para realizar solicitudes HTTP y obtener datos de un servidor. Además, JWT nos permite añadir una capa de seguridad robusta y flexible. Al comprender los conceptos básicos de la integración con el back-end y cómo utilizar fetch(), podrás crear aplicaciones web dinámicas, interactivas y seguras.

CASO DE ESTUDIO INTEGRADOR

Dominio

Este caso de estudio propuesto está basado en el dominio ya visto en clases de “Órdenes de producción” y “Componentes”. Por lo tanto, necesitarán tener en ejecución dicho proyecto, para que la conexión entre el backend y fronted sea exitosa.

Desarrollo e implementación

Empezaremos creando nuestro sitio index.html, el cual, será el punto de entrada a nuestro frontend y desde el cual realizaremos todas las acciones propuestas anteriormente.

Para este desarrollo, recordar que vamos a utilizar Bootstrap para realizar un diseño más agradable visualmente y el cual soporte responsividad.

Index.html y styles.css

A continuación, se detalla el código html el cual define el layout principal:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caso práctico</title>
    <!-- Bootstrap 5 CDN -->
   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css
" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.mi
n.js"></script>
    <!-- Iconos de Bootstrap -->
    <link   href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-
icons.css" rel="stylesheet">
    <link href="./style/style.css" rel="stylesheet">
</head>
<body>
    <!-- Sidebar -->
    <div id="sidebar" class="d-flex flex-column">
```

```
        <button id="toggle-btn" class="btn mb-3 mt-3 mx-auto">
            <i id="toggle-icon" class="bi bi-menu-app"></i>
        </button>
        <ul class="nav flex-column">
   <li class="nav-item">
                <a  class="nav-link"  href="altaComponente.html"><i  class="bi
bi-plus-square"></i> <span>Nuevo componente</span></a>
            </li>
            <li class="nav-item">
                <a     class="nav-link"     href="listadoComponentes.html"><i
class="bi bi-list-ul"></i> <span>Listado componentes</span></a>
            </li>
            <li class="nav-item">
                <a    class="nav-link"    href="alta.html"><i    class="bi    bi-
plus"></i> <span>Nueva orden</span></a>
            </li>
            <li class="nav-item">
              <a   class="nav-link"   href="listado.html"><i   class="bi   bi-
list"></i> <span>Listado órdenes</span></a>
            </li>
        </ul>
    </div>

    <!-- Content -->
    <div id="content">
        <h1 class="mb-4">Bienvenido al Panel Principal</h1>
        <div class="row">
            <div class="col">
                <p>TUP -PROG II - UTN FRC </p>
            </div>
        </div>
    </div>

    <script>
        document.getElementById('toggle-btn').addEventListener('click',
function () {
            const sidebar = document.getElementById('sidebar');
            const icon = document.getElementById('toggle-icon');
            sidebar.classList.toggle('collapsed');
        });
    </script>
</body>
</html>
```

style.css:

```
  body {
    display: flex;
    min-height: 100vh;
    overflow: hidden;
}

#sidebar {
    background-color: #007bff;
    min-width: 250px;
    max-width: 250px;
    color: white;
    transition: all 0.3s;
}

#sidebar.collapsed {
    max-width: 60px;
    min-width: 60px;
}

#sidebar .nav-link {
    color: white;
}

#sidebar .nav-link span {
    margin-left: 10px;
}

#sidebar.collapsed .nav-link span {
    display: none;
}

#content {
    flex-grow: 1;
    padding: 20px;
}

#toggle-btn {
    background-color: transparent;
    color: white;
    border: none;
    font-size: 1.5rem;
}

#toggle-btn:hover {
    color: #ffffffb3;
```

```
}

#toggle-icon {
    transition: transform 0.3s;
}

#sidebar.collapsed #toggle-icon {
    transform: rotate(180deg);
}
```

1. Estructura General del Proyecto

La página utiliza Bootstrap 5.0 para crear un diseño responsivo y atractivo. La estructura se basa en un layout con dos secciones principales:

Un menú lateral (sidebar) a la izquierda que contiene íconos y texto para cada opción de navegación. El menú puede expandirse o colapsarse, mostrando solo los íconos en el estado colapsado.

Un contenido principal (content) en el área restante de la pantalla, donde se muestra el título y otras filas/columnas para el contenido.

La página está diseñada con flexbox para que el layout sea flexible y ocupen todo el alto de la ventana del navegador.

2. Menú Lateral (Sidebar)

El menú lateral está compuesto por:

Un botón de colapso/expansión: El botón ahora es un ícono de flecha (bi-chevron-left), que rota según si la barra está colapsada o expandida.

Una lista de opciones de navegación (ul.nav.flex-column), cada opción tiene:

Un ícono (usando Bootstrap Icons).

Un texto descriptivo (envuelto en un), que se oculta cuando el menú está colapsado.

Cuando la barra lateral está colapsada, solo se muestran los íconos, ocultando los textos asociados a cada opción.

Funcionamiento del Colapso:

El menú lateral usa una clase (#sidebar.collapsed) para controlar su estado colapsado. Cuando el menú se colapsa, el ancho del sidebar se reduce a 60px, y el texto de los ítems (span) se oculta.

El botón de colapso tiene un ícono de flecha que rota 180° cuando el menú está colapsado, indicando el cambio de estado visualmente.

3. Contenido Principal (Content)

El contenido principal es la sección que se ajusta al espacio restante de la pantalla, y contiene:

Un título (<h1>), que se utiliza para dar la bienvenida o presentar la página.

Una fila (.row) que puede contener múltiples columnas (.col) para distribuir el contenido dentro del área principal. En este ejemplo, solo tiene un párrafo, pero puedes agregar más columnas y contenido según sea necesario.

4. Estilo

El estilo personalizado está en un bloque <style> dentro del archivo HTML, y usa principalmente colores y transiciones para hacer que la página se vea atractiva y fluida:

Menú Lateral (Sidebar):

El menú lateral tiene un fondo azul (#007bff) que es el color primario de Bootstrap. Su ancho cambia entre 250px (cuando está expandido) y 60px (cuando está colapsado), con una transición suave.

Los íconos del menú lateral son de color blanco, y el texto también, con un margen izquierdo para separarlos de los íconos cuando el menú está expandido.

El botón de colapso es transparente y tiene un tamaño de ícono de 1.5rem.

Contenido Principal (Content):

El contenido tiene un padding de 20px para asegurar que no se pegue a los bordes de la ventana.

5. Responsividad

La página es completamente responsiva gracias a Bootstrap:

Menú lateral: El ancho del menú se adapta automáticamente. Cuando se colapsa, solo muestra los íconos, lo que lo hace más compacto y adecuado para pantallas pequeñas.

Contenido: El contenido principal se ajusta al espacio restante de la pantalla, expandiéndose y adaptándose al tamaño de la ventana sin romper el layout.

La página está diseñada para ser fluida y funcionar bien en dispositivos móviles, tablets y pantallas grandes sin necesidad de ajustes adicionales.

Listado.html

Teniendo en cuenta el template creado en el punto anterior, crearemos la página encargada de mostrar el listado de todas las órdenes de producción almacenadas en nuestra base de datos.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caso práctico</title>
    <!-- Bootstrap 5 CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css
" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.mi
n.js"></script>
    <!-- Iconos de Bootstrap -->
    <link  href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-
icons.css" rel="stylesheet">
    <link href="./style/style.css" rel="stylesheet">
</head>
<body>
    <!-- Sidebar -->
    <div id="sidebar" class="d-flex flex-column">
        <button id="toggle-btn" class="btn mb-3 mt-3 mx-auto">
            <i id="toggle-icon" class="bi bi-menu-app"></i>
        </button>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a  class="nav-link"  href="altaComponente.html"><i  class="bi
bi-plus-square"></i> <span>Nuevo componente</span></a>
```

```
            </li>
            <li class="nav-item">
                <a     class="nav-link"     href="listadoComponentes.html"><i
class="bi bi-list-ul"></i> <span>Listado componentes</span></a>
            </li>

            <li class="nav-item">
                <a   class="nav-link"   href="alta.html"><i   class="bi   bi-
plus"></i> <span>Nueva orden</span></a>

            </li>
            <li class="nav-item">
               <a   class="nav-link"   href="listado.html"><i   class="bi   bi-
list"></i> <span>Listado órdenes</span></a>
            </li>
        </ul>
    </div>

    <!-- Content -->
    <div id="content">
        <h1 class="mb-4">Listado de órdenes</h1>
        <div class="container mt-4">
            <table class="table table-striped table-hover">
                <thead class="table-primary">
                    <tr>
                        <th>Número</th>
                        <th>Fecha</th>
                        <th>Modelo</th>
                        <th>Cantidad</th>
                        <th>Estado</th>
                        <th>Acciones</th>
                    </tr>
                </thead>
                       <tbody>
                    <tr>
                        <td>1</td>
                        <td>2024-09-22</td>
                        <td>Modelo X123</td>
                        <td>10</td>
                        <td><span class="badge bg-success">Activa</span>
                         </td>
                        <td>
                            <button class="btn btn-info btn-sm">
                                <i class="bi bi-eye"></i> Detalle
                            </button>
                            <button class="btn btn-danger btn-sm">
                                <i class="bi bi-trash3"></i> Eliminar
```

```
                        </button>
                    </td>
                </tr>
                <tr>
                    <td>2</td>
                    <td>2024-09-25</td>
                    <td>Modelo Y456</td>
                    <td>5</td>
                    <td><span class="badge bg-danger">Cancelada</span>
                        </td>
                    <td>
                        <button class="btn btn-info btn-sm">
                            <i class="bi bi-eye"></i> Detalle
                        </button>
                        <button class="btn btn-danger btn-sm">
                            <i class="bi bi-trash3"></i> Eliminar
                        </button>
                    </td>
                </tr>
                <tr>
                    <td>3</td>
                    <td>2024-10-01</td>
                    <td>Modelo Z789</td>
                    <td>20</td>
                    <td><span class="badge bg-success">Activa</span>
                    </td>
                    <td>
                        <button class="btn btn-info btn-sm">
                            <i class="bi bi-eye"></i> Detalle
                        </button>
                        <button class="btn btn-danger btn-sm">
                            <i class="bi bi-trash3"></i> Eliminar
                        </button>
                    </td>
                </tr>
                <tr>
                    <td>4</td>
                    <td>2024-10-03</td>
                    <td>Modelo W321</td>
                    <td>15</td>
                    <td><span  class="badge  bg-warning">Finalizada</span>
</td>
                    <td>
                        <button class="btn btn-info btn-sm">
                            <i class="bi bi-eye"></i> Detalle
                        </button>
                        <button class="btn btn-danger btn-sm">
```

```
                                <i class="bi bi-trash3"></i> Eliminar
                            </button>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>

    <script>
        document.getElementById('toggle-btn').addEventListener('click',
function () {
            const sidebar = document.getElementById('sidebar');
            const icon = document.getElementById('toggle-icon');
            sidebar.classList.toggle('collapsed');
        });
    </script>
</body>
</html>
```

Descripción de la Tabla:

1. Estructura de la Tabla:

La tabla usa las clases de Bootstrap para dar un estilo moderno y atractivo.

La clase table-striped alterna los colores de las filas.

La clase table-hover añade un efecto de resaltado al pasar el cursor sobre las filas.

La cabecera está resaltada con table-primary para darle un fondo azul claro.

2. Columnas:

Número: Un número único para cada fila.

Fecha: Fecha aleatoria que podría representar la fecha de creación o actualización del producto.

Modelo: Un código de modelo para identificar el producto.

Cantidad: Cantidad disponible del producto.

Estado: Indicador del estado del stock. He usado badges de Bootstrap para darle un estilo más visual. Los estados incluyen:

"En Stock" (verde).

"Agotado" (rojo).

"Bajo Stock" (amarillo).

3. Columna "Acciones":

En la columna "Acciones", se incluye un botón con el texto "Detalle" y un ícono de Bootstrap (bi-eye) que simboliza ver detalles. El botón tiene la clase btn-info para darle un color azul claro, y btn-sm para que el tamaño del botón sea pequeño y compacto.

Estilos Visuales:

Badges: Los badges de Bootstrap (ej. bg-success, bg-danger, bg-warning) se usan para indicar el estado del producto de manera visual y clara.

Botones de Acción: El botón de "Detalle" tiene un ícono y un diseño limpio para que la interfaz sea agradable y fácil de usar.

Responsividad:

Gracias a Bootstrap, la tabla es totalmente responsiva. En dispositivos más pequeños, la tabla se ajustará automáticamente al ancho de la pantalla, y se verá adecuadamente en cualquier dispositivo sin necesidad de ajustes adicionales.

Alta.html

A continuación, detallaremos la página, encargada del alta de una orden de producción con su respectivo detalle.

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caso práctico</title>
    <!-- Bootstrap 5 CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css
" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.mi
n.js"></script>
```

```
    <!-- Iconos de Bootstrap -->
    <link  href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css" rel="stylesheet">
    <link href="./style/style.css" rel="stylesheet">
</head>

<body>
    <!-- Sidebar -->
    <div id="sidebar" class="d-flex flex-column">
        <button id="toggle-btn" class="btn mb-3 mt-3 mx-auto">
            <i id="toggle-icon" class="bi bi-menu-app"></i>
        </button>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a  class="nav-link"  href="altaComponente.html"><i  class="bi bi-plus-square"></i> <span>Nuevo componente</span></a>
            </li>
            <li class="nav-item">
                <a     class="nav-link"     href="listadoComponentes.html"><i class="bi bi-list-ul"></i> <span>Listado componentes</span></a>
            </li>

            <li class="nav-item">
                <a   class="nav-link"   href="alta.html"><i   class="bi   bi-plus"></i> <span>Nueva orden</span></a>
            </li>
            <li class="nav-item">
                <a  class="nav-link"  href="listado.html"><i  class="bi  bi-list"></i> <span>Listado</span></a>
            </li>
        </ul>
    </div>

    <!-- Content -->
    <div id="content">
        <div class="container mt-4">
            <form id="form-orden">
                <!-- Primera fila: Formulario básico -->
                <div class="row mb-3">
                    <div class="col-12">
                        <h3>Alta de Orden de Producción</h3>

                        <div class="mb-3">
                            <label        for="orderDate"        class="form-label">Fecha</label>
                            <input      type="text"      class="form-control" id="input-fecha" readonly>
```

```
                        </div>
                        <div class="mb-3">
                            <label          for="model"          class="form-
label">Modelo</label>
                            <input      type="text"      class="form-control"
id="input-modelo" placeholder="Ingrese el modelo">
                        </div>
                        <div class="mb-3">
                            <label        for="quantity"        class="form-
label">Cantidad</label>
                            <input     type="number"     class="form-control"
id="input-cantidad"
                                placeholder="Ingrese la cantidad">
                        </div>
                    </div>
                </div>

                <hr>

                <!-- Segunda fila: Formulario de componentes -->
                <div class="row mb-3">
                    <div class="col-12">
                        <h4>Detalle de la Orden</h4>
                        <div class="row g-3">
                            <div class="col-md-6">
                                <label      for="component"      class="form-
label">Componente</label>
                                <select class="form-select" id="component">
                                    <option value="1">Intel i5</option>
                                    <option value="2">Intel i7</option>
                                    <option value="3">Monitor 24"</option>
                                    <option value="4">Monitor 27"</option>
                                </select>
                            </div>
                            <div class="col-md-4">
                                <label     for="componentQty"     class="form-
label">Cantidad</label>
                                <input    type="number"    class="form-control"
id="componentQty" min="1"
                                    placeholder="Cantidad">
                            </div>
                            <div class="col-md-2 d-flex align-items-end">
                                <button type="button" class="btn btn-primary"
id="addDetailBtn">
                                    <i  class="fas  fa-plus"></i>  Agregar  al
detalle
                                </button>
```

```
                        </div>
                        <!-- Tabla de detalles -->
                        <div class="col-12">
                                <table    class="table    table-bordered"
id="detailsTable">
                                    <thead>
                                        <tr>
                                            <th>ID</th>
                                            <th>Componente</th>
                                            <th>Cantidad</th>
                                        </tr>
                                    </thead>
                                    <tbody>
                                        <!--        Filas        generadas
dinámicamente -->
                                    </tbody>
                                </table>

                        </div>

                        <!-- Botón de Guardar -->
                        <div class="row">
                            <div class="col-12 text-end">
                                <button   type="submit"   class="btn   btn-
success">
                                    <i class="fas fa-save"></i> Guardar
                                </button>
                            </div>
                        </div>
                    </div>
                </div>

            </form>
        </div>
    </div>
    <script>
        // Establecer la fecha actual en el campo de fecha
        document.getElementById('input-fecha').value          =          new
Date().toLocaleDateString('es-ES');

        // Agregar funcionalidad para "Agregar al detalle"
        document.getElementById('addDetailBtn').addEventListener('click',
function () {
            const componentSelect = document.getElementById('component');
            const componentId = componentSelect.value;
```

```
            const componentName = componentSelect.options[componentSelect.selectedIndex].text;
            const componentQty = document.getElementById('componentQty').value;

            if (componentQty > 0) {
                const tableBody = document.querySelector('#detailsTable tbody');
                const newRow = document.createElement('tr');

                newRow.innerHTML = `
                    <td>${componentId}</td>
                    <td>${componentName}</td>
                    <td>${componentQty}</td>
                `;

                tableBody.appendChild(newRow);

                // Limpiar inputs después de agregar
                document.getElementById('componentQty').value = '';
                document.getElementById('component').selectedIndex = 0;
            } else {
                alert("Por favor, ingresa una cantidad válida.");
            }
        });
    </script>
    <script>
        document.getElementById('toggle-btn').addEventListener('click', function () {
            const sidebar = document.getElementById('sidebar');
            const icon = document.getElementById('toggle-icon');
            sidebar.classList.toggle('collapsed');
        });
    </script>
</body>

</html>
```

ListadoComponentes.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Caso práctico</title>
    <!-- Bootstrap 5 CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css
" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.mi
n.js"></script>
    <!-- Iconos de Bootstrap -->
    <link  href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-
icons.css" rel="stylesheet">
    <link href="./style/style.css" rel="stylesheet">
</head>
<body>
    <!-- Sidebar -->
    <div id="sidebar" class="d-flex flex-column">
        <button id="toggle-btn" class="btn mb-3 mt-3 mx-auto">
            <i id="toggle-icon" class="bi bi-menu-app"></i>
        </button>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a  class="nav-link"  href="altaComponente.html"><i  class="bi
bi-plus-square"></i> <span>Nuevo componente</span></a>
            </li>
            <li class="nav-item">
                <a     class="nav-link"     href="listadoComponentes.html"><i
class="bi bi-list-ul"></i> <span>Listado componentes</span></a>
            </li>
            <li class="nav-item">
                <a   class="nav-link"   href="alta.html"><i   class="bi   bi-
plus"></i> <span>Nueva orden</span></a>
            </li>
            <li class="nav-item">
                <a  class="nav-link"  href="listado.html"><i  class="bi  bi-
list"></i> <span>Listado órdenes</span></a>
            </li>
        </ul>
    </div>

    <!-- Content -->
    <div id="content">
        <h1 class="mb-4">Listado de componentes</h1>
        <div class="container mt-4">
            <table class="table table-striped table-hover">
                <thead class="table-primary">
                    <tr>
                        <th>Código</th>
```

```
                        <th>Nombre</th>
                        <th>Fecha de baja</th>
                        <th>Motivo</th>
                        <th>Acciones</th>
                    </tr>
                </thead>
                <tbody id="componentes-body">
                    <!-- Aquí se llenarán dinámicamente los componentes -->
                </tbody>
            </table>
        </div>
    </div>

    <script>
        document.getElementById('toggle-btn').addEventListener('click',
function () {
            const sidebar = document.getElementById('sidebar');
            const icon = document.getElementById('toggle-icon');
            sidebar.classList.toggle('collapsed');
        });
    </script>
</body>
</html>
```

DetalleComponente.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caso práctico</title>
    <!-- Bootstrap 5 CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css
" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.mi
n.js"></script>
    <!-- Iconos de Bootstrap -->
    <link  href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-
icons.css" rel="stylesheet">
    <link href="./style/style.css" rel="stylesheet">
</head>
<body>
    <!-- Sidebar -->
    <div id="sidebar" class="d-flex flex-column">
```

```
        <button id="toggle-btn" class="btn mb-3 mt-3 mx-auto">
            <i id="toggle-icon" class="bi bi-menu-app"></i>
        </button>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a  class="nav-link"  href="altaComponente.html"><i  class="bi
bi-plus-square"></i> <span>Nuevo componente</span></a>
            </li>
            <li class="nav-item">
                <a     class="nav-link"     href="listadoComponentes.html"><i
class="bi bi-list-ul"></i> <span>Listado componentes</span></a>
            </li>
            <li class="nav-item">
                <a   class="nav-link"   href="alta.html"><i   class="bi   bi-
plus"></i> <span>Nueva orden</span></a>
            </li>
            <li class="nav-item">
                <a  class="nav-link"  href="listado.html"><i  class="bi  bi-
list"></i> <span>Listado órdenes</span></a>
            </li>
        </ul>
    </div>

    <div class="container mt-4">
        <h2 class="mb-4">Detalle de componente</h2>

        <!-- Primera Row: Formulario -->
        <div class="row mb-4">
            <div class="col-md-6">
                <form id="componenteForm">
                    <!-- Campo Código (readonly) -->
                    <div class="mb-3">
                        <label for="codigo" class="form-label">Código</label>
                        <input type="number" class="form-control" id="codigo"
readonly>
                    </div>

                    <!-- Campo Nombre (type text) -->
                    <div class="mb-3">
                        <label for="nombre" class="form-label">Nombre</label>
                        <input  type="text"  class="form-control"  id="nombre"
required>
                    </div>

                    <!-- Campo Fecha de Baja (datetime picker) -->
                    <div class="mb-3">
```

```
                        <label for="fechaBaja" class="form-label">Fecha de
Baja</label>
                        <input type="datetime-local" class="form-control"
id="fechaBaja" readonly>
                    </div>

                    <!-- Campo Motivo de Baja (type text) -->
                    <div class="mb-3">
                        <label for="motivoBaja" class="form-label">Motivo de
Baja</label>
                        <input type="text" class="form-control"
id="motivoBaja" readonly>
                    </div>

                    <!-- Botón de Actualizar -->
                    <button type="submit" class="btn btn-
primary">Actualizar</button>
                </form>
            </div>
        </div>
    </div>
    <script>
        document.getElementById('toggle-btn').addEventListener('click',
function () {
            const sidebar = document.getElementById('sidebar');
            const icon = document.getElementById('toggle-icon');
            sidebar.classList.toggle('collapsed');
        });
    </script>
</body>
</html>
```

AltaComponente.html

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caso práctico</title>
    <!-- Bootstrap 5 CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css
" rel="stylesheet">
```

```
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.mi
n.js"></script>
    <!-- Iconos de Bootstrap -->
    <link   href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-
icons.css" rel="stylesheet">
    <link href="./style/style.css" rel="stylesheet">
</head>

<body>
    <!-- Sidebar -->
    <div id="sidebar" class="d-flex flex-column">
        <button id="toggle-btn" class="btn mb-3 mt-3 mx-auto">
            <i id="toggle-icon" class="bi bi-menu-app"></i>
        </button>
        <ul class="nav flex-column">
            <li class="nav-item">
                <a  class="nav-link"  href="altaComponente.html"><i  class="bi
bi-plus-square"></i> <span>Nuevo
                        componente</span></a>
            </li>
            <li class="nav-item">
                <a     class="nav-link"     href="listadoComponentes.html"><i
class="bi bi-list-ul"></i> <span>Listado
                        componentes</span></a>
            </li>
            <li class="nav-item">
                <a    class="nav-link"    href="alta.html"><i    class="bi    bi-
plus"></i> <span>Nueva orden</span></a>
            </li>
            <li class="nav-item">
                <a   class="nav-link"   href="listado.html"><i   class="bi   bi-
list"></i> <span>Listado órdenes</span></a>
            </li>
        </ul>
    </div>

    <!-- Content -->
    <div id="content">
        <div class="container mt-4">
            <form id="form-componente">
                <!-- Primera fila: Formulario básico -->
                <div class="row mb-3">
                    <div class="col-6">
                        <h3>Alta de componente</h3>
                        <div class="mb-3">
```

```
                            <label          for="nombre"          class="form-
label">Nombre</label>
                            <input      type="text"      class="form-control"
id="input-nombre" placeholder="Ingrese el nombre">
                        </div>
                        <!-- Botón de Guardar -->
                        <div class="mb-3">
                            <button type="submit" class="btn btn-success">
                                <i class="fas fa-save"></i> Guardar
                            </button>
                        </div>
                    </div>
                </div>
            </form>
        </div>
    </div>
    <script>
        document.getElementById('toggle-btn').addEventListener('click',
function () {
            const sidebar = document.getElementById('sidebar');
            const icon = document.getElementById('toggle-icon');
            sidebar.classList.toggle('collapsed');
        });
    </script>
</body>

</html>
```

Como hemos visto hasta esta última página html. Todos nuestros sitios, son estaticos con información harcodeada. En la siguiente sección, mediante la utilización de javascript, consumiremos nuestro backend y cargaremos de forma dinámica todos los componentes. ¡Manos a la obra!

Conectar frontend con backend

Comenzaremos creando una nueva carpeta, llamada “js” y dentro de esa carpeta crearemos los archivos que daran soporte a los sitios webs creados anteriormente. los cuales deberán ser referenciados luego, en todos nuestros sitios html.

Como todos nuestros sitios html harán uso de alguna lógica en js crearemos para cada sitio un archivo js particular y luego lo referenciaremos en el <head> de nuetra página, por ejemplo, para listado.html (listado de órdenes de producción):

```
<script src="./js/servicioListado.js"></script>
```

Ahora empezaremos a agregar la lógica a “listado.html”.

Lógica de listado.html

El paso siguiente es escribir el código js necesario para poder cargar la tabla, ver el detalle de las órdenes y por eliminar/cancelar una orden. El código de servicioListado.js se detalla y describe a continuación:

```
document.addEventListener('DOMContentLoaded', () => {
    const API_URL = 'https://localhost:7125/api/OrdenProduccion/ordenes';

    // Función para obtener las órdenes de producción
    async function fetchOrdenes() {
        try {
            const response = await fetch(API_URL);
            const ordenes = await response.json();
            cargarOrdenes(ordenes);
        } catch (error) {
            console.error('Error al obtener las órdenes:', error);
        }
    }

    // Función para formatear la fecha (de ISO a dd/MM/yyyy)
    function formatearFecha(fechaISO) {
        const fecha = new Date(fechaISO);
        const dia = String(fecha.getDate()).padStart(2, '0');
        const mes = String(fecha.getMonth() + 1).padStart(2, '0');
        const anio = fecha.getFullYear();
        return `${dia}/${mes}/${anio}`;
    }

    // Función para crear las filas de la tabla
    function cargarOrdenes(ordenes) {
        const tbody = document.getElementById('ordenes-body');
        tbody.innerHTML = ''; // Limpiar la tabla antes de agregar nuevas
filas

        ordenes.forEach(orden => {
            const row = document.createElement('tr');

            // Columna Número
            const nroTd = document.createElement('td');
            nroTd.textContent = orden.nro;
            row.appendChild(nroTd);

            // Columna Fecha
```

```
            const fechaTd = document.createElement('td');
            fechaTd.textContent = formatearFecha(orden.fecha);
            row.appendChild(fechaTd);

            // Columna Cantidad
            const cantidadTd = document.createElement('td');
            cantidadTd.textContent = orden.cantidad;
            row.appendChild(cantidadTd);

            // Columna Modelo
            const modeloTd = document.createElement('td');
            modeloTd.textContent = orden.modelo;
            row.appendChild(modeloTd);

            // Columna Estado con el estilo adecuado
            const estadoTd = document.createElement('td');
            const estadoBadge = document.createElement('span');
            estadoBadge.classList.add('badge');

            switch (orden.estado) {
                case 'Creada':
                    estadoBadge.classList.add('bg-success');
                    estadoBadge.textContent = 'Creada';
                    break;
                case 'Cancelada':
                    estadoBadge.classList.add('bg-danger');
                    estadoBadge.textContent = 'Cancelada';
                    break;
                case 'Finalizada':
                    estadoBadge.classList.add('bg-warning');
                    estadoBadge.textContent = 'Finalizada';
                    break;
                default:
                    estadoBadge.textContent = orden.estado;
            }
            estadoTd.appendChild(estadoBadge);
            row.appendChild(estadoTd);

            // Columna Acciones (Detalle y Eliminar)
            const accionesTd = document.createElement('td');

            // Botón Detalle
            const detalleBtn = document.createElement('button');
            detalleBtn.classList.add('btn', 'btn-info', 'btn-sm');
            detalleBtn.textContent = 'Detalle';
            detalleBtn.addEventListener('click', () => {
                window.location.href = `detalle.html?id=${orden.nro}`;
```

```
            });

            // Botón Eliminar
            const eliminarBtn = document.createElement('button');
            eliminarBtn.classList.add('btn', 'btn-danger', 'btn-sm');
            eliminarBtn.textContent = 'Eliminar';
            eliminarBtn.addEventListener('click', () => {
                if (confirm('¿Estás seguro que deseas eliminar esta orden?'))
{
                    eliminarOrden(orden.nro);
                }
            });

            accionesTd.appendChild(detalleBtn);
            accionesTd.appendChild(eliminarBtn);
            row.appendChild(accionesTd);

            // Agregar la fila a la tabla
            tbody.appendChild(row);
        });
    }

    // Función para eliminar una orden
    async function eliminarOrden(id) {
        try {
            const response = await fetch(`${API_URL}/${id}`, {
                method: 'DELETE',
            });

            if (response.ok) {
                alert('Orden eliminada con éxito');
                fetchOrdenes(); // Recargar las órdenes después de eliminar
            } else {
                alert('Error al eliminar la orden');
            }
        } catch (error) {
            console.error('Error al eliminar la orden:', error);
            alert('Ocurrió un error al intentar eliminar la orden');
        }
    }

    // Llamar a la función para cargar las órdenes cuando la página cargue
    fetchOrdenes();
});
```

Lógica de alta.html

Al igual que antes, nuestra página html se verá reducida en líneas, ya que el select con los componentes se cargará dinámicamente. Código js para dar soporte a la página:

```
document.addEventListener('DOMContentLoaded', () => {
    const API_URL = 'https://localhost:7125/api';

    // Obtener los elementos del formulario
    const form = document.getElementById('form-orden');
    const inputFecha = document.getElementById('input-fecha');
    const inputModelo = document.getElementById('input-modelo');
    const inputCantidad = document.getElementById('input-cantidad');
    const selectComponentes = document.getElementById('component');

    // Agregar un listener al formulario
    form.addEventListener('submit', async (event) => {
        event.preventDefault();
        console.log('Formulario enviado'); // Esto te ayudará a saber si el
evento se activa

        // Obtener los valores del formulario
        const fechaInput = new Date(inputFecha.value);   // Crea un objeto
Date
        const fecha = fechaInput.toISOString(); // Convierte a formato ISO
8601

        const modelo = inputModelo.value;
        const cantidad = parseInt(inputCantidad.value);

        // Recorrer la tabla de componentes
        const detalles = obtenerDetallesTabla();

        // Construir el cuerpo del POST sin el 'nro'
        const body = {
            fecha: fecha,
            listaDetalles: detalles,
            modelo: modelo,
            estado: 'Creada',
            cantidad: cantidad
        };

        try {
            const response = await fetch(`${API_URL}/OrdenProduccion`, {
                method: 'POST',
                headers: {
```

```
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(body)
            });

            if (response.ok) {
                alert('Orden de producción agregada con éxito');
                // Opcionalmente, redirigir o resetear el formulario
                form.reset();
            } else {
                alert('Error al agregar la orden de producción');
            }
        } catch (error) {
            console.error('Error:', error);
            alert('Ocurrió un error al intentar agregar la orden de producción');
        }
    });

    // Cargar componentes en el select
    cargarComponentes();

    // Cargar componentes en el select
    async function cargarComponentes() {
        try {
            const response = await fetch(`${API_URL}/Componente`);
            if (!response.ok) throw new Error('Error al cargar componentes');

            const componentes = await response.json();
            componentes.forEach(componente => {
                const option = document.createElement('option');
                option.value = componente.codigo; // Código como valor
                option.textContent = componente.nombre; // Nombre como texto
                selectComponentes.appendChild(option);
            });
        } catch (error) {
            console.error('Error al cargar componentes:', error);
            alert('Ocurrió un error al cargar los componentes');
        }
    }


    // Función para recorrer la tabla y obtener los detalles de la orden
    function obtenerDetallesTabla() {
        const tabla = document.getElementById('detailsTable');
        const filas = tabla.querySelectorAll('tbody tr');
        const detalles = [];
```

```
        filas.forEach(fila => {
            const id = parseInt(fila.children[0].textContent);    // Primer
columna: ID
            const componente = fila.children[1].textContent;      // Segunda
columna: Componente
            const cantidad = parseInt(fila.children[2].textContent);    //
Tercera columna: Cantidad

            detalles.push({
                id: id,
                componente: {
                    codigo: id, // Aquí podrías ajustar si el componente
tiene un código distinto
                    nombre: componente,
                    fechaBaja: null, // O algún valor si lo tienes
                    motivoBaja: null // O algún valor si lo tienes
                },
                cantidad: cantidad
            });
        });

        return detalles;
    }
});
```

Lógica de listadoComponentes.html

```
document.addEventListener('DOMContentLoaded', () => {
    const API_URL = ' https://localhost:7125/api/Componente';

    // Función para obtener los componentes
    async function fetchComponentes() {
        try {
            const response = await fetch(API_URL);
            const ordenes = await response.json();
            cargarComponentes(ordenes);
        } catch (error) {
            console.error('Error al obtener los componentes:', error);
        }
    }

    // Función para formatear la fecha (de ISO a dd/MM/yyyy)
    function formatearFecha(fechaISO) {
        const fecha = new Date(fechaISO);
        const dia = String(fecha.getDate()).padStart(2, '0');
```

```
        const mes = String(fecha.getMonth() + 1).padStart(2, '0');
        const anio = fecha.getFullYear();
        return `${dia}/${mes}/${anio}`;
    }

    // Función para crear las filas de la tabla
    function cargarComponentes(componentes) {
        const tbody = document.getElementById('componentes-body');
        tbody.innerHTML = ''; // Limpiar la tabla antes de agregar nuevas
filas

        componentes.forEach(componente => {
            const row = document.createElement('tr');

            // Columna Código
            const codTd = document.createElement('td');
            codTd.textContent = componente.codigo;
            row.appendChild(codTd);

            // Columna Nombre
            const nombreTd = document.createElement('td');
            nombreTd.textContent = componente.nombre;
            row.appendChild(nombreTd);

            // Columna Fecha de baja
            const fechaBajaTd = document.createElement('td');
            fechaBajaTd.textContent = componente.fechaBaja != null ?
formatearFecha(componente.fechaBaja) : '';
            row.appendChild(fechaBajaTd);

            // Columna Motivo de baja
            const motivoBajaTd = document.createElement('td');
            motivoBajaTd.textContent = componente.motivoBaja;
            row.appendChild(motivoBajaTd);



            // Columna Acciones (Detalle y Eliminar)
            const accionesTd = document.createElement('td');

            // Botón Detalle
            const detalleBtn = document.createElement('button');
            detalleBtn.classList.add('btn', 'btn-info', 'btn-sm');
            detalleBtn.textContent = 'Detalle';
            detalleBtn.addEventListener('click', () => {
                window.location.href =
`detalleComponente.html?id=${componente.codigo}`;
```

```
            });

            // Botón Eliminar
            const eliminarBtn = document.createElement('button');
            eliminarBtn.classList.add('btn', 'btn-danger', 'btn-sm');
            eliminarBtn.textContent = 'Eliminar';
            eliminarBtn.addEventListener('click', () => {
                if (confirm('¿Estás seguro que deseas dar de baja este componente?')) {
                    // Preguntar el motivo de baja
                    const motivoBaja = prompt('Por favor, ingresa el motivo de baja:');

                    // Si el motivo de baja no es vacío o nulo
                    if (motivoBaja && motivoBaja.trim() !== '') {
                        darDeBajaComponente(componente.codigo, motivoBaja);
                    } else {
                        alert('Debes ingresar un motivo de baja válido.');
                    }
                }
            });

            accionesTd.appendChild(detalleBtn);
            if (componente.fechaBaja === null) {
                accionesTd.appendChild(eliminarBtn);
            }

            row.appendChild(accionesTd);

            // Agregar la fila a la tabla
            tbody.appendChild(row);
        });
    }

    // Función para dar de baja un componente
    async function darDeBajaComponente(id, motivoBaja) {
        try {
            const response = await fetch(`${API_URL}/${id}?motivo=${motivoBaja}`, {
                method: 'DELETE',
            });

            if (response.ok) {
                alert('Componente dado de baja con éxito');
                fetchComponentes(); // Recargar los componentes despues de dar de baja
            } else {
```

```
                alert('Error al dar de baja el componente');
            }
        } catch (error) {
            console.error('Error al dar de baja componente:', error);
            alert('Ocurrió un error al intentar dar de baja el componente');
        }
    }

    // Llamar a la función para cargar los componentes cuando la página cargue
    fetchComponentes();
});
```

Lógica de detalleComponente.html

```
// Función para formatear fecha en formato dd/MM/yyyy
function formatearFecha(fechaISO) {
    const fecha = new Date(fechaISO);
    const dia = String(fecha.getDate()).padStart(2, '0');
    const mes = String(fecha.getMonth() + 1).padStart(2, '0'); // Los meses comienzan desde 0
    const anio = fecha.getFullYear();
    return `${dia}/${mes}/${anio}`;
}

// Función para obtener el parámetro "id" del query string
function obtenerIdDeQueryString() {
    const params = new URLSearchParams(window.location.search);
    return params.get('id'); // Retorna el valor de "id"
}

// Función para obtener el componente por ID desde la base de datos (fetch)
async function obtenerComponentePorId(id) {
    const response = await fetch(`https://localhost:7125/api/Componente/${id}`);
    const data = await response.json();
    return data;
}

// Función para cargar los valores del JSON en los inputs
function cargarDatosEnFormulario(componente) {
    document.getElementById('codigo').value = componente.codigo;
    document.getElementById('nombre').value = componente.nombre;

    // Convertir fecha ISO a formato "yyyy-MM-ddTHH:mm" para datetime-local
    if (componente.fechaBaja === null) {
```

```
        document.getElementById('fechaBaja').value = '';
    } else {
        // Convertir fecha ISO a formato "yyyy-MM-ddTHH:mm" para datetime-
local si no es null
        const fechaBajaISO = new
Date(componente.fechaBaja).toISOString().slice(0, 16);
        document.getElementById('fechaBaja').value = fechaBajaISO;
    }

    document.getElementById('motivoBaja').value = componente.motivoBaja;
}
// Cargar datos al cargar la página
document.addEventListener('DOMContentLoaded', function () {
    // Aquí va todo el código, incluyendo el addEventListener para el form
    document.getElementById('componenteForm').addEventListener('submit',
async function (event) {
        event.preventDefault();
        // Obtener los valores del formulario
        const codigo = document.getElementById('codigo').value;
        const nombre = document.getElementById('nombre').value;

        // Crear el objeto JSON con los datos
        const componenteActualizado = {
            codigo: parseInt(codigo),
            nombre: nombre
        };

        // Enviar los datos actualizados con un PUT request
        const response = await fetch(`
https://localhost:7125/api/Componente/${codigo}`, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(componenteActualizado)
        });

        if (response.ok) {
            alert('Componente actualizado con éxito');
        } else {
            alert('Hubo un error al actualizar el componente');
        }
    });

    // Resto del código de inicialización
    window.onload = async function () {
        const id = obtenerIdDeQueryString();
```

```
        if (id) {
            const componente = await obtenerComponentePorId(id);
            cargarDatosEnFormulario(componente);
        } else {
            alert('No se ha proporcionado un ID válido en la URL.');
        }
    };
});
```

Lógica de altaComponente.html

```
document.addEventListener('DOMContentLoaded', () => {
    const API_URL = 'http://fpiemontesi-001-site10.atempurl.com/api'; // 
Reemplaza con tu URL real

    // Obtener los elementos del formulario
    const form = document.getElementById('form-componente');
    const inputNombre = document.getElementById('input-nombre');

    // Agregar un listener al formulario
    form.addEventListener('submit', async (event) => {
        event.preventDefault();
        console.log('Formulario enviado'); // Esto te ayudará a saber si el 
evento se activa

        const nombre = inputNombre.value;

        const body = {
            nombre: nombre
        };

        try {
            const response = await fetch(`${API_URL}/Componente`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(body)
            });

            if (response.ok) {
                alert('Componente agregado con éxito');
                // Opcionalmente, redirigir o resetear el formulario
                form.reset();
            } else {
                alert('Error al agregar el componente');
```

```
            }
        } catch (error) {
            console.error('Error:', error);
            alert('Ocurrió un error al intentar agregar el componente');
        }
    });
});
```

BIBLIOGRAFÍA

- Flanagan, D. (2019). *JavaScript: The Definitive Guide* (7th ed.). O'Reilly Media. https://www.oreilly.com/library/view/javascript-the-definitive/9781491952010/
- Mozilla Developer Network (MDN). (n.d.). HTML: HyperText Markup Language. Mozilla. Recuperado de https://developer.mozilla.org/es/docs/Web/HTML
- Keith, J. (2020). *CSS Secrets: Better Solutions to Everyday Web Design Problems*. O'Reilly Media.
- Bootstrap. (n.d.). *Bootstrap Documentation*. Recuperado de https://getbootstrap.com/docs/5.3/getting-started/introduction/
- Lopez, E. (2021). *JavaScript: Guía de referencia*. Ediciones ENI. https://www.ediciones-eni.com/libro/javascript-guia-de-referencia-9782409020506
- Una introducción a Javascript (22 de agosto de 2022). Recuperado de: https://es.javascript.info/intro
- Sitio Oficial de Json (31 de octubre 2022). Recuperado de: https://www.json.org/json-en.html
- Creación de una API (31 de octubre 2022). Recuperado de: https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html