



PROYECTO 2º EVALUACIÓN

WEB-DINÁMICA

Tomas Cano Nieto

ÍNDICE

- 1. ¿Cómo funciona esta app web?**
- 2. Implementación del MVC**
- 3. Ficheros de la aplicación:**
 - a. Controlador**
 - i. main.js**
 - b. Modelo**
 - i. data.xml**
 - ii. index.php**
 - c. Vista**
 - i. index.php**
- 4. Fichero CSS**
- 5. Conclusión y Justificación**
- 6. Biografía**

1- ¿Cómo funciona esta app web?

1. Inicio de la Aplicación:

Cuando un usuario accede a la aplicación web, se carga la página de inicio (`index.php`). En esta página, se presenta el título de la librería online y se muestra un formulario de búsqueda donde el usuario puede ingresar los criterios de búsqueda, como el ID del libro, el autor, el género y la página.

2. Búsqueda de Libros:

El usuario completa el formulario de búsqueda con los criterios deseados y hace clic en el botón "Buscar". Esto activa la función `getData()` en el archivo JavaScript `main.js`.

3. Envío de Solicitud al Servidor:

La función `getData()` recopila los valores de los campos de entrada del formulario y realiza una solicitud HTTP utilizando el método `fetch()` a la URL del servicio web proporcionada por el archivo PHP `modelo/index.php`. Los parámetros de búsqueda se incluyen como parte de la URL de la solicitud.

4. Procesamiento en el Servidor:

En el servidor, el archivo PHP `modelo/index.php` recibe la solicitud y carga el documento XML `data.xml`, que contiene la información de los libros disponibles en la librería. Luego, filtra los libros según los parámetros de búsqueda proporcionados por el usuario utilizando la función `filterBooks()`.

5. Generación de Respuesta JSON:

Después de filtrar los libros, el archivo PHP devuelve los resultados en formato JSON como respuesta a la solicitud HTTP.

6. Actualización de la Interfaz de Usuario:

Una vez que se reciben los resultados de la búsqueda, la función `displayResults()` en el archivo JavaScript `main.js` se encarga de actualizar la interfaz de usuario con los resultados obtenidos. Si se encuentran libros que coinciden con los criterios de búsqueda, se muestran en la sección de resultados de la página web. Si no se encontraron resultados, se muestra un mensaje indicando que no se encontraron libros.

7. Visualización de Resultados:

Los resultados de la búsqueda se presentan de manera dinámica en la página web. Para cada libro encontrado, se muestra su `título`, `autor`, `género`, `precio`, `fecha de publicación` y `descripción`. Los resultados se muestran de manera clara y organizada para que el usuario pueda revisarlos fácilmente.

8. Interacción del Usuario:

El usuario puede interactuar con los resultados de la búsqueda, revisando la información de cada libro y realizando nuevas búsquedas si es necesario. El proceso de búsqueda se puede repetir cuantas veces sea necesario para encontrar los libros deseados.

2- Implementación del MVC

El patrón Modelo-Vista-Controlador (MVC) es una arquitectura de software que separa las preocupaciones de una aplicación en tres componentes principales: el Modelo, la Vista y el Controlador. Esta separación de responsabilidades promueve un desarrollo de software más organizado, modular y fácil de mantener.

2.1. Modelo (Model)

En el contexto de nuestra aplicación de librería online, el Modelo representa los datos subyacentes de la aplicación, en este caso, la información de los libros disponibles en la librería. Este modelo de datos se encuentra en el archivo XML `'data.xml'`, donde cada libro está representado como un elemento XML con atributos como autor, título, género, precio, fecha de publicación y descripción.

Además, la lógica de manipulación y filtrado de estos datos se encuentra en el archivo PHP `'modelo/index.php'`. Este archivo actúa como el controlador del modelo, ya que se encarga de cargar el documento XML, filtrar los libros según los parámetros de búsqueda proporcionados por el usuario y devolver los resultados en formato JSON.

La implementación del modelo en nuestro proyecto garantiza que la lógica de negocio y el acceso a los datos estén separados de la interfaz de usuario, lo que facilita la reutilización del código y la escalabilidad de la aplicación.

2.2. Vista (View)

La Vista en nuestra aplicación corresponde a la interfaz de usuario que interactúa con el usuario final. En este caso, la vista está representada por el archivo HTML `'vista/index.php'`, que define la estructura de la página web y proporciona los elementos de entrada (inputs) y la sección de resultados donde se mostrarán los libros encontrados.

La vista también incluye estilos CSS para el diseño y la presentación visual de la página web. Estos estilos están definidos en el archivo CSS `'estilos/styles.css'`, que proporciona una apariencia atractiva y coherente a la interfaz de usuario.

La separación de la lógica de presentación y el contenido de la página web en la vista garantiza una mayor modularidad y mantenibilidad del código, ya que los cambios en el diseño o la presentación no afectan la lógica subyacente de la aplicación.

2.3. Controlador (Controller)

El Controlador en MVC actúa como intermediario entre el Modelo y la Vista, gestionando las interacciones del usuario y actualizando la vista según sea necesario. En nuestra aplicación, el controlador está representado por el archivo JavaScript `controlador/main.js`.

Este archivo contiene la lógica del controlador que maneja las interacciones del usuario, como enviar solicitudes al servidor utilizando la API Fetch y actualizar dinámicamente la interfaz de usuario con los resultados de la búsqueda. Además, el controlador también se encarga de limpiar los resultados anteriores antes de mostrar los nuevos resultados de la búsqueda.

La implementación del controlador en nuestro proyecto garantiza una interacción fluida y receptiva con el usuario, permitiendo búsquedas de libros sin tener que recargar la página web.

2.4. Ventajas de MVC en nuestra Aplicación

La implementación del patrón MVC en nuestra aplicación de librería online ofrece varias ventajas significativas:

- **Separación de Responsabilidades:** MVC divide la aplicación en componentes separados, lo que facilita la gestión del código y su mantenimiento a largo plazo.
- **Reutilización del Código:** Al separar la lógica de negocio, la presentación y el control de las interacciones del usuario, se facilita la reutilización de componentes en diferentes partes de la aplicación.
- **Facilidad de Mantenimiento:** La modularidad y la separación de preocupaciones facilitan la identificación y corrección de errores, así como la introducción de nuevas funcionalidades sin afectar otras partes del sistema.
- **Escalabilidad:** MVC permite escalar la aplicación de manera eficiente, ya que cada componente puede ser desarrollado, probado y desplegado de forma independiente.

3- Ficheros de la aplicación

CONTROLADOR/CÓDIGO DE MAIN.JS

Este código en `'main.js'` es parte del frontend de la aplicación web y se encarga de manejar la interacción del usuario con la página de búsqueda de libros. Aquí tienes una explicación detallada de lo que hace:

La función `'getData()'`:

1. Obtiene los valores de los campos de búsqueda (**ID, autor, género y página**) de la página HTML utilizando `'document.getElementById('id').value'` y asignándoles a variables correspondientes (`'id'`, `'author'`, `'genre'` y `'page'`).
2. Utiliza la función `'fetch()'` para realizar una solicitud HTTP al servicio web proporcionado por el archivo PHP `'modelo/index.php'`. En la URL de la solicitud se incluyen los parámetros de búsqueda obtenidos en el paso anterior.
3. Una vez que se recibe la respuesta de la solicitud HTTP, se convierte a formato JSON utilizando el método `'json()'`.
4. Luego, llama a la función `'displayResults(data)'` pasando los datos obtenidos como argumento.

La función `'displayResults(data)'`:

1. Recibe los datos en formato JSON que representan los resultados de la búsqueda de libros.
2. Obtiene la sección de resultados de la página HTML utilizando `'document.getElementById('result-section)'`.
3. Si hay libros encontrados en los datos recibidos y la longitud de la lista de libros es mayor que cero, itera sobre cada libro y genera dinámicamente elementos HTML que representan la información de cada libro (**título, autor, género, precio, fecha de publicación y descripción**). Estos elementos se crean utilizando la función `'document.createElement('div)'` y se agregan a la sección de resultados utilizando `'resultSection.appendChild(bookElement)'`.
4. Si no se encontraron resultados de búsqueda, se muestra un mensaje indicando que no se encontraron resultados utilizando `'resultSection.innerHTML = '<p>No se encontraron resultados.</p>'`.

CÓDIGO DE MAIN.JS:

```
js main.js x
Proyecto-WebDinamica > controlador > js main.js > ⚙️ getData

1 function getData() {
2   const id = document.getElementById('id').value;
3   const author = document.getElementById('author').value;
4   const genre = document.getElementById('genre').value;
5   const page = document.getElementById('page').value;
6
7   // Realizar llamada a la API con los parámetros proporcionados
8   fetch('https://localhost/DWES/Proyecto-WebDinamica/modelo/index.php?id=${id}&author=${author}&genre=${genre}&page=${page}')
9     .then(response => response.json())
10    .then(data => displayResults(data));
11 }
12
13 function displayResults(data) {
14   const resultSection = document.getElementById('result-section');
15   resultSection.innerHTML = ''; // Limpiar resultados anteriores
16
17   if (data.books && data.books.length > 0) {
18     data.books.forEach(book => {
19       const bookElement = document.createElement('div');
20       bookElement.innerHTML = `
21         <h3>${book.title}</h3>
22         <p><strong>Autor:</strong> ${book.author}</p>
23         <p><strong>Género:</strong> ${book.genre}</p>
24         <p><strong>Precio:</strong> ${book.price}</p>
25         <p><strong>Fecha de Publicación:</strong> ${book.publish_date}</p>
26         <p><strong>Descripción:</strong> ${book.description}</p>
27       `;
28       resultSection.appendChild(bookElement);
29     });
30   } else {
31     resultSection.innerHTML = '<p>No se encontraron resultados.</p>';
32   }
33 }
34 }
```

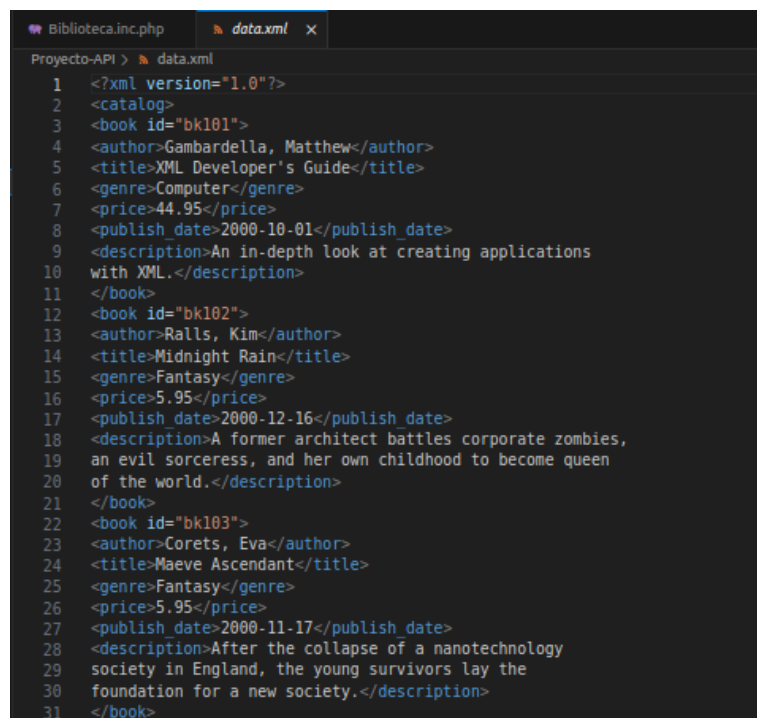

MODELO/CÓDIGO DE DATA.XML:

El archivo XML llamado data.xml contiene un catálogo de libros con la siguiente información:

- **ID:** Identificador único del libro.
- **Autor:** Nombre del autor del libro.
- **Título:** Título del libro.
- **Género:** Género literario al que pertenece el libro.
- **Precio:** Precio del libro.
- **Fecha de publicación:** Fecha en que se publicó el libro.
- **Descripción:** Breve descripción o sinopsis del contenido del libro.

Cada libro está representado por un elemento ``<book>`` en el archivo XML, con atributos y subelementos que contienen los detalles mencionados anteriormente. Este archivo XML sirve como la fuente de datos para la API web, desde donde se extrae la información de los libros para su posterior filtrado y presentación a través de solicitudes HTTP.

CÓDIGO DE DATA.XML:



```
Proyecto-API > data.xml
1  <?xml version="1.0"?>
2  <catalog>
3    <book id="bk101">
4      <author>Gambardella, Matthew</author>
5      <title>XML Developer's Guide</title>
6      <genre>Computer</genre>
7      <price>44.95</price>
8      <publish_date>2000-10-01</publish_date>
9      <description>An in-depth look at creating applications
10     with XML.</description>
11    </book>
12    <book id="bk102">
13      <author>Ralls, Kim</author>
14      <title>Midnight Rain</title>
15      <genre>Fantasy</genre>
16      <price>5.95</price>
17      <publish_date>2000-12-16</publish_date>
18      <description>A former architect battles corporate zombies,
19      an evil sorceress, and her own childhood to become queen
20      of the world.</description>
21    </book>
22    <book id="bk103">
23      <author>Corets, Eva</author>
24      <title>Maeve Ascendant</title>
25      <genre>Fantasy</genre>
26      <price>5.95</price>
27      <publish_date>2000-11-17</publish_date>
28      <description>After the collapse of a nanotechnology
29      society in England, the young survivors lay the
30      foundation for a new society.</description>
31    </book>
```

MODELO/CÓDIGO DE INDEX.PHP:

1. Cabecera de respuesta: Se establece el encabezado de respuesta para indicar que el contenido devuelto será en formato JSON. Esto se hace utilizando la función `header("Content-Type: application/json");`.

2. Cargar el documento XML: Se carga el archivo XML `'data.xml'` que contiene la información sobre los libros. Esto se hace utilizando la función `'simplexml_load_file('data.xml')'`, que carga el contenido del archivo XML y lo convierte en un objeto SimpleXMLElement.

3. Obtención de parámetros de la solicitud: Se obtienen los parámetros de la solicitud HTTP GET, como el ID del libro (`'id'`), el autor (`'author'`), el género (`'genre'`) y el número de página (`'page'`). Estos parámetros se utilizan para filtrar los libros según los criterios especificados en la solicitud.

4. Función para filtrar libros: Se define una función llamada `'filterBooks'` que recibe como argumentos el objeto XML cargado y los parámetros de filtrado. Esta función itera sobre cada elemento `'<book>'` del XML y compara los valores de los atributos y elementos con los parámetros proporcionados. Si un libro coincide con los criterios de búsqueda, se agrega a un arreglo de libros filtrados.

5. Paginación de resultados: Si se proporciona el parámetro `'page'`, se realiza la paginación de los resultados. Se calcula el inicio y el fin de la página actual y se devuelve sólo un subconjunto de libros que corresponden a esa página.

6. Procesamiento de la solicitud y generación de la respuesta: Se define una variable `'$response'` para almacenar la respuesta que se enviará al cliente. Si no se proporcionan parámetros de filtrado, se devuelve todo el contenido del documento XML. En caso contrario, se llamará a la función `'filterBooks'` para obtener los libros filtrados según los parámetros especificados. La respuesta final se estructura como un arreglo asociativo con una clave `"books"` que contiene los libros filtrados.

7. Codificación de la respuesta en JSON: Finalmente, la respuesta se codifica en formato JSON utilizando la función `'json_encode'`, con la opción `'JSON_PRETTY_PRINT'` para que el JSON generado esté formateado de manera legible.

8. Salida de la respuesta: La respuesta JSON generada se envía al cliente como salida de la ejecución del script, y será procesada por el cliente que realizó la solicitud HTTP.

CÓDIGO DE INDEX.PHP:

```
index.php X
Proyecto-API > index.php
1 <?php
2 header("Content-Type: application/json");
3
4 // Cargar el documento XML
5 $xml = simplexml_load_file('data.xml');
6
7 // Obtener parámetros de la solicitud
8 $id = isset($_GET['id']) ? $_GET['id'] : null;
9 $author = isset($_GET['author']) ? $_GET['author'] : null;
10 $genre = isset($_GET['genre']) ? $_GET['genre'] : null;
11 $page = isset($_GET['page']) ? $_GET['page'] : null;
12
13 // Función para filtrar libros según parámetros
14 function filterBooks($xml, $id, $author, $genre, $page) {
15     $filteredBooks = [];
16
17     foreach ($xml->book as $book) {
18         if (
19             (!$id || $book['id'] == $id) &&
20             (!$author || strpos($book->author, $author) !== false) &&
21             (!$genre || strpos($book->genre, $genre) !== false)
22         ) {
23             $filteredBooks[] = [
24                 'id' => (string)$book['id'],
25                 'author' => (string)$book->author,
26                 'title' => (string)$book->title,
27                 'genre' => (string)$book->genre,
28                 'price' => (float)$book->price,
29                 'publish_date' => (string)$book->publish_date,
30                 'description' => (string)$book->description,
31             ];
32         }
33     }
34
35     // Pagar si se proporciona el parámetro 'page'
36     if ($page && is_numeric($page)) {
37         $perPage = 5; // Número de libros por página
38         $start = ($page - 1) * $perPage;
39         $filteredBooks = array_slice($filteredBooks, $start, $perPage);
40     }
41
42     return $filteredBooks;
43 }
44
45 // Procesar la solicitud y devolver la respuesta JSON
46 $response = [];
47
48 if (!$id && !$author && !$genre && !$page) {
49     // Si no se proporcionan parámetros, devolver todo el documento
50     $response = $xml;
51 } else {
52     // Filtrar libros según parámetros
53     $response['books'] = filterBooks($xml, $id, $author, $genre, $page);
54 }
55
56 echo json_encode($response, JSON_PRETTY_PRINT);
57 ?>
```

VISTA/CÓDIGO DE INDEX.PHP:

- **Encabezado (`<header>`):** Presenta el título principal de la página, "[Librería Online](#)", proporcionando una identificación clara del propósito de la aplicación desde el principio.
- **Sección de Filtros (`<section id="filter-section">`):** Contiene un formulario que permite a los usuarios buscar libros por diferentes criterios, como [ID](#), [autor](#), [género](#) y [página](#). Esta sección ofrece una interfaz amigable e intuitiva para realizar búsquedas específicas según las preferencias del usuario.
- **Formulario de Búsqueda (`<form id="filter-form">`):** Este formulario incluye etiquetas y campos de entrada para cada criterio de búsqueda, lo que facilita al usuario comprender qué información puede ingresar en cada campo y cómo usarlos correctamente.
- **Botón de Búsqueda (`<button type="button" onclick="getData()">Buscar</button>`):** Al hacer clic en este botón, se activa una función JavaScript llamada `getData()`, que recopila los datos del formulario y realiza una solicitud al servidor para buscar libros que coincidan con los criterios especificados.
- **Sección de Resultados (`<section id="result-section">`):** En esta sección, se muestran dinámicamente los resultados de la búsqueda de libros. Cuando se obtienen los datos del servidor, una función JavaScript llamada `displayResults()` se encarga de presentar los libros encontrados en esta área de la página, proporcionando información relevante como título, autor, género, precio y descripción de cada libro.
- **Script JavaScript (`<script src="../controlador/main.js"></script>`):** Enlaza el archivo JavaScript `main.js`, que contiene las funciones necesarias para manejar la interacción del usuario y la presentación de los resultados de la búsqueda. Esta parte del código es crucial para la funcionalidad de la página, ya que permite la comunicación con el servidor y la actualización dinámica de la interfaz de usuario en respuesta a las acciones del usuario.

CÓDIGO DE INDEX.PHP:

```
index.php x
Proyecto-WebDinamica > vista > index.php
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" href="../estilos/styles.css">
7      <title>Librería Online</title>
8  </head>
9  <body>
10
11      <header>
12          <h1>Librería Online</h1>
13      </header>
14
15      <section id="filter-section">
16          <!-- Formulario para enviar parámetros al servicio web -->
17          <form id="filter-form">
18              <label for="id">ID:</label>
19              <input type="text" id="id" name="id">
20
21              <label for="author">Autor:</label>
22              <input type="text" id="author" name="author">
23
24              <label for="genre">Género:</label>
25              <input type="text" id="genre" name="genre">
26
27              <label for="page">Página:</label>
28              <input type="text" id="page" name="page">
29
30              <button type="button" onclick="getData()">Buscar</button>
31          </form>
32      </section>
33
34      <section id="result-section">
35          <!-- Aquí se mostrarán los resultados de la búsqueda -->
36      </section>
37
38      <script src="../controlador/main.js"></script>
39  </body>
40  </html>
```

4- Fichero CSS

```
Ejercicio02-formulario.php  Ejercicio01-formulario.php  # styles.css X
Proyecto-WebDinamica > estilos > # styles.css > body
1  body {
2      font-family: Arial, sans-serif;
3      margin: 0;
4      padding: 0;
5  }
6
7  header {
8      background-color: #333;
9      color: white;
10     padding: 1rem;
11     text-align: center;
12 }
13
14 #filter-section {
15     margin: 1rem;
16 }
17
18 #filter-form {
19     display: flex;
20     gap: 1rem;
21 }
22
23 #result-section {
24     margin: 1rem;
25 }
26
```

5- Conclusión y Justificación

En el proceso de desarrollo de esta aplicación web para una librería online, se han identificado varios aspectos clave que destacan la importancia de utilizar el patrón Modelo-Vista-Controlador (MVC) junto con las tecnologías del lado del cliente y del lado del servidor. A continuación, se presentan algunas conclusiones y justificaciones detalladas:

Separación de Responsabilidades con MVC

La adopción del patrón MVC ha demostrado ser una elección acertada para este proyecto. La clara separación de responsabilidades entre el modelo, la vista y el controlador ha facilitado la organización del código y ha permitido un desarrollo más estructurado y modular. Esto ha resultado en una aplicación más fácil de mantener y escalar a medida que se agregan nuevas funcionalidades o se realizan cambios en el código.

Mantenibilidad y Escalabilidad

La estructura MVC ha contribuido significativamente a la mantenibilidad y escalabilidad de la aplicación. Al tener modelos independientes que representan los datos, vistas separadas que definen la interfaz de usuario y controladores que manejan la lógica de negocio, se ha logrado una arquitectura flexible que facilita la introducción de cambios en el código sin afectar otras partes de la aplicación. Esto ha sido especialmente beneficioso en un proyecto de este tipo, donde se espera una evolución continua en función de las necesidades del usuario y del negocio.

Mejora de la Experiencia del Usuario

La aplicación web desarrollada ofrece una experiencia de usuario mejorada gracias a la interactividad y dinamismo proporcionados por JavaScript. Los usuarios pueden realizar búsquedas de libros de forma rápida y sencilla, y los resultados se presentan de manera dinámica sin necesidad de recargar la página. Esto mejora la usabilidad y la eficiencia de la aplicación, lo que a su vez contribuye a una experiencia más satisfactoria para el usuario final.

Flexibilidad y Adaptabilidad

La combinación de tecnologías del lado del cliente y del lado del servidor ha proporcionado una gran flexibilidad y adaptabilidad a la aplicación. Por un lado, HTML, CSS y JavaScript permiten crear interfaces de usuario atractivas y funcionales, mientras que PHP facilita la manipulación de datos y la generación dinámica de contenido en el servidor. Esta combinación de tecnologías ha permitido desarrollar una aplicación versátil que puede adaptarse fácilmente a los requisitos cambiantes del negocio y las necesidades del usuario.

Justificación de las Tecnologías Utilizadas

La elección de HTML, CSS, JavaScript y PHP como tecnologías principales para el desarrollo de la aplicación se ha basado en varios factores. HTML proporciona la estructura básica de la página web, CSS permite estilizar y diseñar la apariencia de la interfaz de usuario, JavaScript ofrece interactividad y dinamismo, y PHP facilita la creación de aplicaciones web dinámicas en el lado del servidor. Estas tecnologías son ampliamente utilizadas, tienen una amplia base de usuarios y cuentan con una gran cantidad de recursos y documentación disponible, lo que ha facilitado el desarrollo del proyecto y la resolución de problemas.

6- Biografía

He estado usando todas estas webs la cual gracias a ellas me ha sido de apoyo para realizar este proyecto:

-<https://www.youtube.com/> (Funcionalidad de VMC para poder realizar el VMC)

-<https://chat.openai.com/> (Errores que me salían y consultas SQL de base de datos, estilos CSS de la página)

-<https://www.php.net/> (Propiedades, funciones...)