



PROYECTO 2º EVALUACIÓN

MVC

Tomas Cano Nieto

ÍNDICE

- 1. ¿Cómo funciona esta app web?**
- 2. Ficheros de la aplicación:**
 - a. Controller**
 - i. borraZapatilla.php
 - ii. grabaZapatilla.php
 - iii. listado.php
 - iv. zapatilla.php
 - b. Model**
 - i. Zapatilla.php
 - ii. ZapatilleriaDB.php
 - c. View**
 - i. Images()
 - ii. formularioZapatilla.php
 - iii. listado.php
 - d. Htaccess**
- 3. Fichero BBDD**
- 4. Ventajas de usar el MVC y comparación con el 1º Proyecto**
- 5. Conclusión y Justificación**
- 6. Biografía**

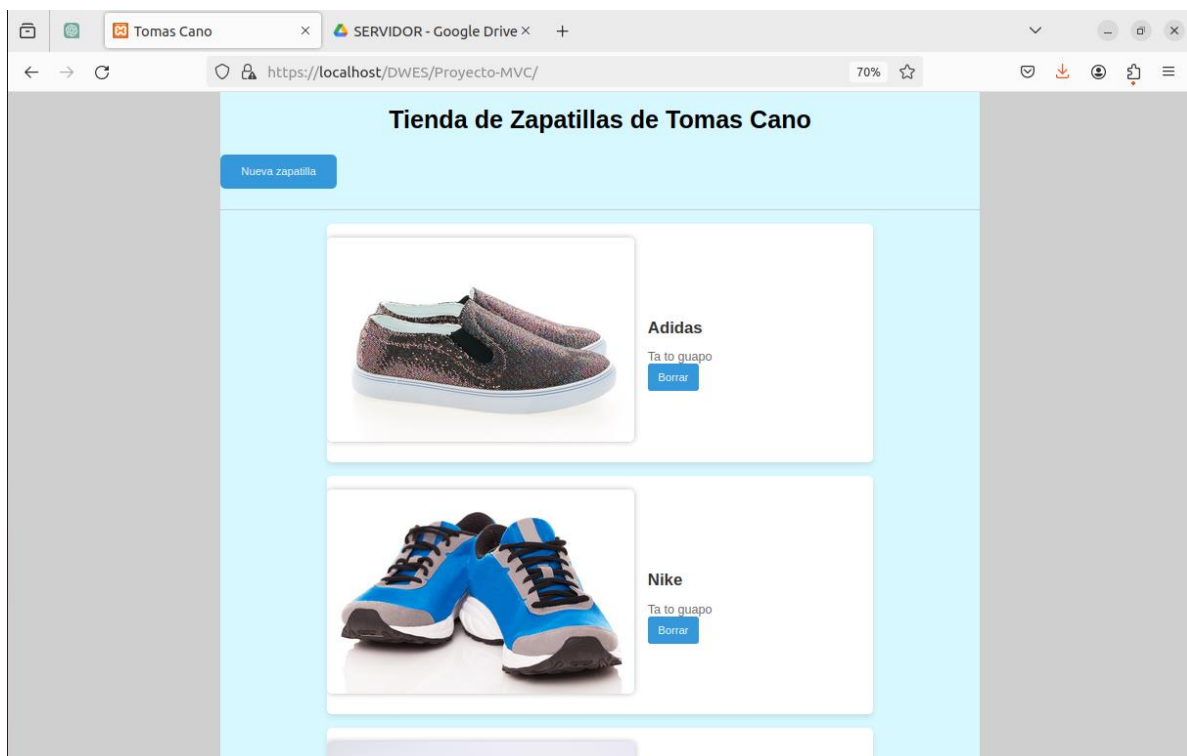
1- ¿Cómo funciona esta app web?

Bienvenido a la documentación de la aplicación web MVC con rutas amigables para la gestión de un catálogo de zapatillas. Esta aplicación proporciona una plataforma fácil de usar para que los usuarios naveguen por una variedad de zapatillas, agreguen nuevas al catálogo y eliminen las existentes según sea necesario.

La aplicación sigue el patrón de diseño Modelo-Vista-Controlador (MVC), lo que garantiza una separación clara de las preocupaciones y una estructura organizada del código. Con una arquitectura sólida y rutas amigables, los usuarios pueden acceder a las funcionalidades de la aplicación de manera intuitiva y eficiente.

A lo largo de esta documentación, explicaremos en detalle las diferentes características, tecnologías utilizadas y recomendaciones de seguridad para garantizar una experiencia óptima tanto para los usuarios como para los desarrolladores.

A continuación, se presenta una imagen que representa la interfaz de usuario de nuestra aplicación:



2- Ficheros de la aplicación

CONTROLLER/CÓDIGO DE BORRAZAPATILLA.PHP:

1. Inclusión del Archivo Zapatilla.php: Se incluye el archivo `'Zapatilla.php'`, que contiene la definición de la clase `'Zapatilla'` y sus métodos asociados para interactuar con la base de datos y gestionar las zapatillas.

2. Extracción de la Ruta de la Aplicación: Se obtiene la URL de la solicitud actual y se divide en partes usando el delimitador `"/"`. Luego, se reconstruye la ruta base de la aplicación utilizando los elementos relevantes del array `'$arrBasename'`.

3. Obtención del ID de la Zapatilla a Eliminar: Se obtiene el parámetro `'id'` de la solicitud GET, que se utiliza para identificar la zapatilla que se va a eliminar.

4. Creación de una Instancia de Zapatilla: Se crea una nueva instancia de la clase `'Zapatilla'`, pasando el ID de la zapatilla que se va a eliminar como argumento al constructor. Esto permite acceder a los métodos de la clase para eliminar la zapatilla correspondiente.

5. Eliminación de la Zapatilla de la Base de Datos: Se llama al método `'delete()'` de la instancia de `'Zapatilla'` para eliminar la zapatilla de la base de datos.

6. Redirección al Listado de Zapatillas: Después de eliminar la zapatilla, el usuario es redirigido de vuelta al listado de zapatillas utilizando la ruta base de la aplicación.

CÓDIGO DE BORRAZAPATILLA.PHP:

```
borraZapatilla.php x
Proyecto-MVC > Controller > borraZapatilla.php
1  <?php
2  require_once "../Model/Zapatilla.php";
3
4  $arrBasename = explode("/", $_SERVER["REQUEST_URI"]);
5  $ruta = "/" . $arrBasename[1] . "/" . $arrBasename[2];
6  $id = $_GET["id"];
7  $zapatillaAux = new Zapatilla($id);
8  $zapatillaAux->delete();
9  header("Location: " . $ruta);
10 ?>
```

CONTROLLER/CÓDIGO DE GRABAZAPATILLA.PHP:

1. Inclusión del Archivo Zapatilla.php: Se incluye el archivo `'Zapatilla.php'`, que contiene la definición de la clase `'Zapatilla'` y sus métodos asociados para interactuar con la base de datos y gestionar las zapatillas.

2. Extracción de la Ruta de la Aplicación: Se obtiene la URL de la solicitud actual y se divide en partes usando el delimitador `"/"`. Luego, se reconstruye la ruta base de la aplicación utilizando los elementos relevantes del array `'$arrBasename'`.

3. Manejo del Archivo de Imagen: Se mueve el archivo de imagen subido (`'$_FILES["imagen"]["tmp_name"]'`) al directorio de imágenes de la vista, lo que permite almacenar la imagen en el servidor.

4. Creación de una Instancia de Zapatilla: Se crea una nueva instancia de la clase `'Zapatilla'` con los datos proporcionados por el formulario de la vista. Se utiliza un constructor especial que permite crear una nueva zapatilla sin proporcionar un ID (que se generará automáticamente en la base de datos).

5. Inserción de la Zapatilla en la Base de Datos: Se llama al método `'insert()'` de la instancia de `'Zapatilla'`, que inserta los datos de la nueva zapatilla en la base de datos.

6. Redirección al Listado de Zapatillas: Después de realizar la inserción, el usuario es redirigido de vuelta al listado de zapatillas utilizando la ruta base de la aplicación.

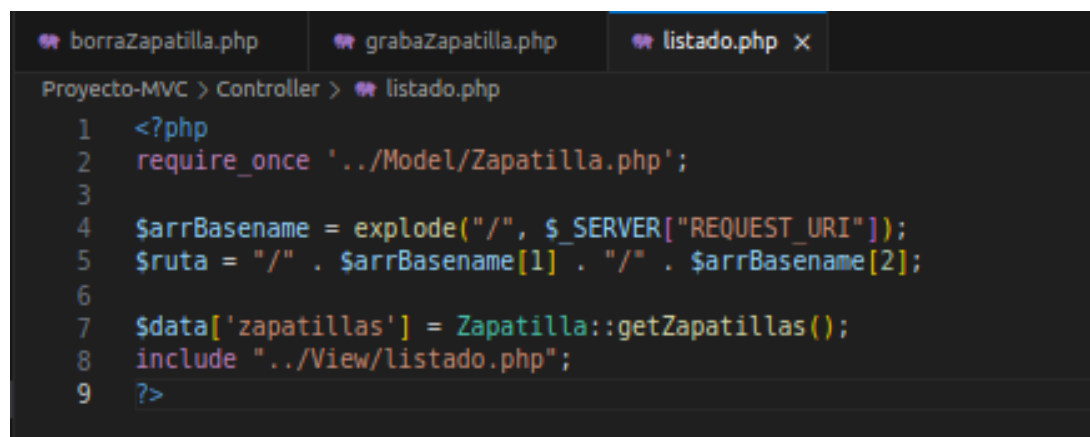
CÓDIGO DE GRABAZAPATILLA.PHP:

```
borraZapatilla.php  grabaZapatilla.php x
Proyecto-MVC > Controller > grabaZapatilla.php
1  <?php
2  require_once '../Model/Zapatilla.php';
3
4  $arrBasename = explode("/", $_SERVER["REQUEST_URI"]);
5  $ruta = "/" . $arrBasename[1] . "/" . $arrBasename[2];
6
7  move_uploaded_file($_FILES["imagen"]["tmp_name"], "../View/images/" . $_FILES["imagen"]["name"]);
8
9  $zapatillaAux = new Zapatilla("", $_POST['modelo'], $_FILES["imagen"]["name"], $_POST['descripcion']);
10 $zapatillaAux->insert();
11 header("Location: " . $ruta);
12 ?>
```

CONTROLLER/CÓDIGO DE LISTADO.PHP:

- 1. Inclusión del Archivo Zapatilla.php:** Se incluye el archivo `'Zapatilla.php'`, que contiene la definición de la clase `'Zapatilla'` y sus métodos asociados para interactuar con la base de datos y gestionar las zapatillas.
- 2. Extracción de la Ruta de la Aplicación:** Se obtiene la URL de la solicitud actual y se divide en partes usando el delimitador `"/"`. Luego, se reconstruye la ruta base de la aplicación utilizando los elementos relevantes del array `'$arrBasename'`.
- 3. Obtención de las Zapatillas desde la Base de Datos:** Se llama al método estático `'getZapatillas()'` de la clase `'Zapatilla'`, que obtiene todas las zapatillas almacenadas en la base de datos. Estas zapatillas se asignan al arreglo `'$data['zapatillas']'` para ser utilizadas en la vista.
- 4. Inclusión de la Vista Listado:** Se incluye la vista `'listado.php'`, que se encarga de mostrar el listado de zapatillas. La variable `'$data['zapatillas']'` se pasa a esta vista para que pueda acceder a la información de las zapatillas y mostrarla en la página.

CÓDIGO DE LISTADO.PHP:



```
borraZapatilla.php  grabaZapatilla.php  listado.php x
Proyecto-MVC > Controller > listado.php
1  <?php
2  require_once '../Model/Zapatilla.php';
3
4  $arrBasename = explode("/", $_SERVER["REQUEST_URI"]);
5  $ruta = "/" . $arrBasename[1] . "/" . $arrBasename[2];
6
7  $data['zapatillas'] = Zapatilla::getZapatillas();
8  include "../View/listado.php";
9  ?>
```

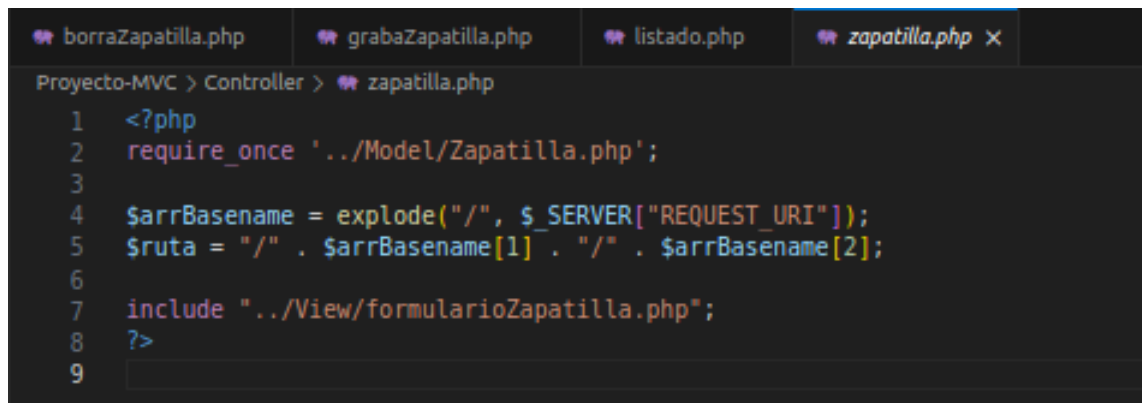
CONTROLLER/CÓDIGO DE ZAPATILLA.PHP:

1. Inclusión del Archivo Zapatilla.php: Se incluye el archivo `'Zapatilla.php'`, que contiene la definición de la clase `'Zapatilla'` y sus métodos asociados para interactuar con la base de datos y gestionar las zapatillas.

2. Extracción de la Ruta de la Aplicación: Se obtiene la URL de la solicitud actual y se divide en partes usando el delimitador `"/"`. Luego, se reconstruye la ruta base de la aplicación utilizando los elementos relevantes del array `'$arrBasename'`.

3. Inclusión de la Vista Formulario de Zapatilla: Se incluye la vista `'formularioZapatilla.php'`, que contiene el formulario para agregar una nueva zapatilla al catálogo. Al incluir esta vista, se presenta al usuario un formulario interactivo donde puede ingresar los detalles de la nueva zapatilla.

CÓDIGO DE ZAPATILLA.PHP:



```
borraZapatilla.php  grabaZapatilla.php  listado.php  zapatilla.php X
Proyecto-MVC > Controller > zapatilla.php
1  <?php
2  require_once '../Model/Zapatilla.php';
3
4  $arrBasename = explode("/", $_SERVER["REQUEST_URI"]);
5  $ruta = "/" . $arrBasename[1] . "/" . $arrBasename[2];
6
7  include "../View/formularioZapatilla.php";
8  ?>
9
```

MODEL/CÓDIGO DE ZAPATILLA.PHP

1. Definición de la Clase Zapatilla: Se define la clase `'Zapatilla'`, que representa una zapatilla en el sistema. La clase contiene propiedades para almacenar el ID, del modelo, la imagen y la descripción de la zapatilla.

2. Constructor: Se define un constructor que permite crear nuevas instancias de la clase `'Zapatilla'` con los valores proporcionados para el ID, el modelo, la imagen y la descripción.

3. Métodos Getter: Se definen métodos `'getId()'`, `'getModelo()'`, `'getImagen()'` y `'getDescripcion()'` para acceder a las propiedades privadas de la clase desde fuera de la misma.

4. Método Insert: Se define el método `'insert()'` que inserta una nueva zapatilla en la base de datos. Se utiliza una consulta preparada para evitar posibles ataques de inyección SQL.

5. Método Delete: Se define el método `'delete()'` que elimina una zapatilla de la base de datos utilizando su ID. También se utiliza una consulta preparada para la eliminación segura de la zapatilla.

6. Método Estático getZapatillas(): Se define el método estático `'getZapatillas()'` que obtiene todas las zapatillas almacenadas en la base de datos y las devuelve como un arreglo de objetos `'Zapatilla'`.

7. Método Estático getZapatillaById(): Se define el método estático `'getZapatillaById($id)'` que obtiene una zapatilla específica de la base de datos utilizando su ID y la devuelve como un objeto `'Zapatilla'`.

CÓDIGO DE ZAPATILLA.PHP:

```
borraZapatilla.php  grabaZapatilla.php  listado.php  Zapatilla.php X
Proyecto-MVC > Model > Zapatilla.php
1  <?php
2
3  require_once 'ZapattilleriaDB.php';
4
5  class Zapatilla {
6      private $id;
7      private $modelo;
8      private $imagen;
9      private $descripcion;
10
11     public function __construct($id, $modelo = "", $imagen = "", $descripcion = "") {
12         $this->id = $id;
13         $this->modelo = $modelo;
14         $this->imagen = $imagen;
15         $this->descripcion = $descripcion;
16     }
17
18     public function getId() {
19         return $this->id;
20     }
21
22     public function getModelo() {
23         return $this->modelo;
24     }
25
26     public function getImagen() {
27         return $this->imagen;
28     }
29
30     public function getDescripcion() {
31         return $this->descripcion;
32     }
33
34     public function insert() {
35         $conexion = ZapattilleriaDB::connectDB();
36         $insercion = "INSERT INTO zapatilla (modelo, imagen, descripcion) VALUES (7, ?, ?)";
37         $stmt = $conexion->prepare($insercion);
38         $stmt->execute([$this->modelo, $this->imagen, $this->descripcion]);
39     }
40
41     public function delete() {
42         $conexion = ZapattilleriaDB::connectDB();
43         $borrado = "DELETE FROM zapatilla WHERE id=?";
44         $stmt = $conexion->prepare($borrado);
45         $stmt->execute([$this->id]);
46     }
47
48     public static function getZapatillas() {
49         $conexion = ZapattilleriaDB::connectDB();
50         $seleccion = "SELECT id, modelo, imagen, descripcion FROM zapatilla";
51         $consulta = $conexion->query($seleccion);
52
53         $zapatillas = [];
54
55         while ($registro = $consulta->fetchObject()) {
56             $zapatillas[] = new Zapatilla($registro->id, $registro->modelo, $registro->imagen, $registro->descripcion);
57         }
58
59         return $zapatillas;
60     }
61
62     public static function getZapatillaById($id) {
63         $conexion = ZapattilleriaDB::connectDB();
64         $seleccion = "SELECT id, modelo, imagen, descripcion FROM zapatilla WHERE id=?";
65         $stmt = $conexion->prepare($seleccion);
66         $stmt->execute([$id]);
67         $registro = $stmt->fetchObject();
68         $zapatilla = new Zapatilla($registro->id, $registro->modelo, $registro->imagen, $registro->descripcion);
69
70         return $zapatilla;
71     }
72 }
```

MODEL/CÓDIGO DE ZAPATILLADB.PHP:

1. Definición de la Clase Abstracta ZapatteriaDB: Se define la clase abstracta `'ZapatteriaDB'`, que proporciona métodos para conectarse a la base de datos de la aplicación de zapatillas.

2. Propiedades Estáticas Privadas: La clase define propiedades estáticas privadas para almacenar la información de conexión a la base de datos, incluyendo el servidor, la base de datos, el usuario y la contraseña.

3. Método Estático connectDB(): Se define el método estático `'connectDB()'` que se encarga de establecer una conexión con la base de datos utilizando los parámetros de conexión definidos en las propiedades estáticas privadas.

4. Bloque Try-Catch: El método `'connectDB()'` utiliza un bloque try-catch para manejar cualquier excepción que pueda ocurrir durante el proceso de conexión. Si se produce un error, se imprime un mensaje de error y se finaliza la ejecución del script.

5. Creación del Objeto PDO: Dentro del bloque try, se crea un objeto PDO que representa la conexión a la base de datos MySQL. Se utiliza el constructor de PDO para especificar el servidor, la base de datos, el conjunto de caracteres y las credenciales de autenticación.

6. Establecimiento del Modo de Error: Se utiliza el método `'setAttribute()'` del objeto PDO para establecer el modo de error en `'PDO::ERRMODE_EXCEPTION'`, lo que significa que PDO lanzará excepciones para errores de conexión y consulta en lugar de simplemente emitir advertencias.

7. Retorno de la Conexión: Finalmente, se retorna el objeto PDO que representa la conexión exitosa a la base de datos.

CÓDIGO DE ZAPATILLERADB.PHP:



```
borraZapattilla.php  grabaZapattilla.php  listado.php  ZapatteriaDB.php x
Proyecto-MVC > Model > ZapatteriaDB.php
1  <?php
2
3  abstract class ZapatteriaDB {
4      private static $server = 'localhost';
5      private static $db = 'zapatteria';
6      private static $user = 'alumno';
7      private static $password = '1234';
8
9      public static function connectDB() {
10         try {
11             $connection = new PDO("mysql:host=" . self::$server . ";dbname=" . self::$db . ";charset=utf8", self::$user, self::$password);
12             $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
13         } catch (PDOException $e) {
14             echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
15             die("Error: " . $e->getMessage());
16         }
17
18         return $connection;
19     }
20 }
21
22 ?>
```

VIEW/CÓDIGO DE FORMULARIOZAPATILLA.PHP:

1. Estructura HTML: El código comienza con la etiqueta `<!DOCTYPE html>`, que define el tipo de documento como HTML5. Luego, se abre la etiqueta `<html lang="es">`, que especifica el idioma del documento como español.

2. Cabecera: En la sección `<head>`, se definen metadatos importantes como la codificación de caracteres, la escala inicial para dispositivos móviles y el título de la página. También se incluye un bloque de estilo CSS interno para personalizar el aspecto del formulario.

3. Estilos CSS: Se definen estilos CSS para dar formato al formulario y sus elementos. Se establece un fondo de color claro, márgenes y rellenos, así como una sombra para resaltar el formulario. Los campos de entrada y la etiqueta `<h3>` se estilizan para que se vean más atractivos y legibles.

4. Cuerpo del Formulario: En la sección `<body>`, se encuentra el formulario en sí. Se utiliza la etiqueta `<form>` para definir el formulario y se especifica la acción (`action`) a la que se enviarán los datos cuando se envíe el formulario. Además, se establece el método de envío como POST y se habilita la carga de archivos (`enctype="multipart/form-data"`) para poder subir imágenes.

5. Campos del Formulario: Dentro del formulario, se definen tres campos: "Modelo", "Imagen" y "Descripción". Para cada campo, se utiliza la etiqueta `<h3>` para el título y se crea un campo de entrada utilizando las etiquetas `<input>` y `<textarea>`. El campo de "Imagen" utiliza `type="file"` para permitir la selección de archivos y `accept="image/*"` para limitar la selección a imágenes.

6. Botón de Envío: Se agrega un botón de envío `<input type="submit">` al final del formulario. Este botón permite enviar los datos ingresados en el formulario al servidor cuando se hace clic en él.

CÓDIGO DE FORMULARIOZAPATILLA.PHP:

```
Proyecto-MVC > View > FormularioZapatilla.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 <title>Formulario de Zapatillas</title>
7 <style>
8     body {
9         font-family: 'Arial', sans-serif;
10        background-color: #f7f7f7;
11        margin: 0;
12        padding: 0;
13        display: flex;
14        align-items: center;
15        justify-content: center;
16        height: 100vh;
17    }
18
19    form {
20        background-color: #fff;
21        padding: 20px;
22        border-radius: 8px;
23        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
24        width: 400px;
25    }
26
27    h3 {
28        color: #333;
29        margin-bottom: 10px;
30    }
31
32    input[type="text"],
33    input[type="file"],
34    textarea {
35        width: 100%;
36        padding: 10px;
37        margin-bottom: 15px;
38        box-sizing: border-box;
39        border: 1px solid #ccc;
40        border-radius: 4px;
41        font-size: 16px;
42    }
43
44    textarea {
45        resize: vertical;
46    }
47
48    input[type="submit"] {
49        background-color: #3498db;
50        color: #fff;
51        padding: 10px 15px;
52        border: none;
53        border-radius: 4px;
54        cursor: pointer;
55        font-size: 16px;
56        transition: background-color 0.3s ease;
57    }
58
59    input[type="submit"]:hover {
60        background-color: #2980b9;
61    }
62 </style>
63 </head>
64 <body>
65     <form action="<?=$ruta; ?>/GrabaZapatilla" enctype="multipart/form-data" method="POST">
66         <h3>Modelo</h3>
67         <input type="text" name="modelo" required>
68         <h3>Imagen</h3>
69         <input type="file" id="imagen" name="imagen" accept="image/*" required>
70         <br><h3>Descripción</h3>
71         <textarea name="descripcion" cols="30" rows="6" required></textarea>
72         <br>
73         <input type="submit" value="Aceptar">
74     </form>
75 </body>
76 </html>
77
```

VIEW/CÓDIGO DE LISTADO.PHP:

1. Estructura HTML: El código comienza con la etiqueta `<!DOCTYPE html>`, que define el tipo de documento como HTML5. Luego, se abre la etiqueta `<html lang="es">`, que especifica el idioma del documento como español.

2. Cabecera: En la sección `<head>`, se definen metadatos importantes como la codificación de caracteres, la escala inicial para dispositivos móviles y el título de la página. Además, se incluye un bloque de estilo CSS interno para personalizar el aspecto de la página.

3. Estilos CSS: Se definen estilos CSS para dar formato a la página y sus elementos. Se establece un fondo de color claro y se utiliza la propiedad `flexbox` para alinear y distribuir los elementos en el centro vertical y horizontal de la página. Se aplican estilos a los títulos, botones, imágenes y tarjetas de zapatillas para mejorar su apariencia y legibilidad.

4. Cuerpo de la Página: En la sección `<body>`, se encuentra el contenido principal de la página. Se define un contenedor principal con la clase `contenedor` para agrupar todo el contenido y aplicarle un fondo de color claro.

6. Botón de Crear Zapatilla: Se agrega un botón `<a>` con la clase `boton-crear` que permite a los usuarios agregar nuevas zapatillas al catálogo. Este botón redirige al usuario a la página de creación de zapatillas.

7. Catálogo de Zapatillas: Se utiliza un bucle `foreach` para recorrer la lista de zapatillas obtenida del controlador. Para cada zapatilla, se crea una tarjeta `<div>` con la clase `zapatilla-card`. Cada tarjeta contiene una imagen de la zapatilla, su modelo y descripción, así como un enlace para borrar la zapatilla.

8. Imagen de Zapatilla: Se muestra la imagen de cada zapatilla utilizando la etiqueta ``. Se utiliza el método `getImagen()` del objeto `Zapatilla` para obtener la ruta de la imagen desde la carpeta `View/images`.

11. Botón de Borrar: Se agrega un enlace `<a>` con la clase `borrar` para permitir a los usuarios eliminar zapatillas del catálogo. Este enlace redirige al usuario al controlador correspondiente para eliminar la zapatilla.

CÓDIGO DE LISTADO.PHP:

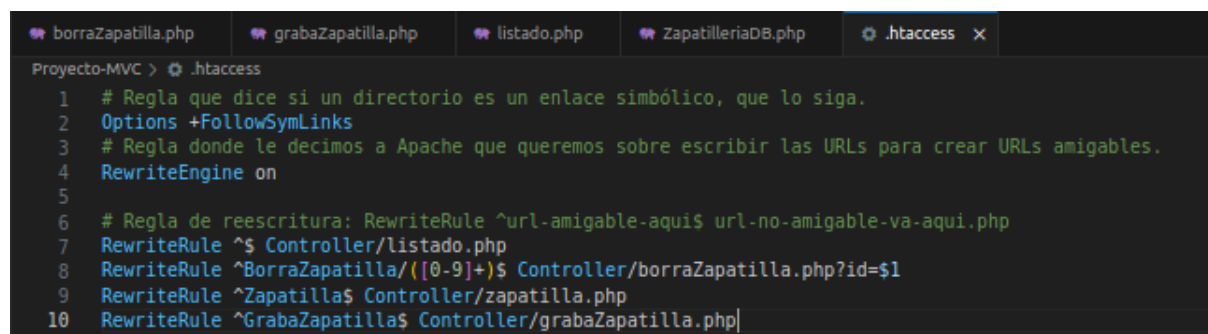
```

Zapatilla.sql  FormularioZapatilla.php  listado.php
Proyecto-MVC > View > listado.php
6  <title>Tomas Cano</title>
7  <style>
8      body {
9          font-family: 'Arial', sans-serif;
10         background-color: #cfcfcf;
11         margin: 0;
12         padding: 0;
13         display: flex;
14         align-items: center;
15         justify-content: center;
16         flex-direction: column;
17     }
18
19     h1 {
20         margin: 20px 0; /* Añadido un margen top y bottom */
21         font-size: 36px;
22         text-align: center; /* Centrar el texto */
23     }
24
25
26
27 </style>
28 </head>
29 <body>
30     <div class="contenedor">
31         <h1>Tienda de Zapatillas de Tomas Cano</h1>
32         <a class="boton-crear" href="<?= $ruta; ?>/Zapatilla">Nueva zapatilla</a>
33         <hr>
34         <?php
35         foreach ($data['zapatillas'] as $zapatilla) {
36             ?>
37             <div class="zapatilla-card">
38                 <div class="zapatilla-container">
39                     getImagen(); ?>" alt="<?= $zapatilla->getModelo(); ?>">
40                     <div class="zapatilla-info">
41                         <h3><?= $zapatilla->getModelo(); ?></h3>
42                         <p><?= $zapatilla->getDescripcion(); ?></p>
43                         <a class="borrar" href="<?= $ruta . "/BorraZapatilla/" . $zapatilla->getId(); ?>">Borrar</a>
44                     </div>
45                 </div>
46             </div>
47         <?php
48         }
49     ?>
50 </div>
51 </body>
52 </html>
53
```

HTACCESS:

1. **`Options +FollowSymLinks`**: Esta directiva indica al servidor que siga los enlaces simbólicos presentes en el sistema de archivos. Esto puede ser útil si se utilizan enlaces simbólicos en el sitio web.
2. **`RewriteEngine on`**: Esta línea habilita el motor de reescritura de URL de Apache, lo que permite la modificación dinámica de las URLs que se solicitan en el servidor.
3. **`RewriteRule ^\$ Controller/listado.php`**: Esta regla de reescritura especifica que cuando se solicita la URL raíz del sitio (es decir, `'http://tudominio.com/'`), se redirige internamente a `'Controller/listado.php'`. Esto suele utilizarse para mostrar la página de inicio del sitio.
4. **`RewriteRule ^BorraZapatilla/([0-9]+)\$ Controller/borraZapatilla.php?id=\$1`**: Esta regla de reescritura está diseñada para manejar URLs que siguen el patrón `'BorraZapatilla/{id}'`, donde `'{id}'` es un número entero. Cuando se recibe una solicitud con este patrón, se redirige internamente a `'Controller/borraZapatilla.php'`, pasando el valor de `'{id}'` como parámetro GET en la URL.
5. **`RewriteRule ^Zapatilla\$ Controller/zapatilla.php`**: Esta regla maneja las solicitudes para la URL `'/Zapatilla'`. Cuando se solicita esta URL, se redirige internamente a `'Controller/zapatilla.php'`, que generalmente se usa para mostrar un formulario de creación de zapatillas.
6. **`RewriteRule ^GrabaZapatilla\$ Controller/grabaZapatilla.php`**: Esta regla maneja las solicitudes para la URL `'/GrabaZapatilla'`. Cuando se recibe esta solicitud, se dirige internamente a `'Controller/grabaZapatilla.php'`, que generalmente se utiliza para procesar los datos enviados desde el formulario de creación de zapatillas y guardarlos en la base de datos.

CÓDIGO DE .HTACCESS:



```
borraZapatilla.php  grabaZapatilla.php  listado.php  ZapatilleriaDB.php  .htaccess x
Proyecto-MVC > .htaccess
1  # Regla que dice si un directorio es un enlace simbólico, que lo siga.
2  Options +FollowSymLinks
3  # Regla donde le decimos a Apache que queremos sobre escribir las URLs para crear URLs amigables.
4  RewriteEngine on
5
6  # Regla de reescritura: RewriteRule ^url-amigable-aqui$ url-no-amigable-va-aqui.php
7  RewriteRule ^$ Controller/listado.php
8  RewriteRule ^BorraZapatilla/([0-9]+)$ Controller/borraZapatilla.php?id=$1
9  RewriteRule ^Zapatilla$ Controller/zapatilla.php
10 RewriteRule ^GrabaZapatilla$ Controller/grabaZapatilla.php|
```

3- Fichero BBDD

CÓDIGO DE BBDD.SQL:

```
borraZapatilla.php  grabaZapatilla.php  listado.php  oferta.sql x
Proyecto-MVC > Model > oferta.sql
23
24  -- -----
25
26  --
27  -- Estructura de tabla para la tabla `zapatilla`
28  --
29
30  CREATE TABLE `zapatilla` (
31    `id` int(11) NOT NULL,
32    `modelo` varchar(200) NOT NULL,
33    `imagen` varchar(100) NOT NULL,
34    `descripcion` varchar(500) NOT NULL
35  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
36
37  --
38  -- Volcado de datos para la tabla `zapatilla`
39  --
40
41  INSERT INTO `zapatilla` (`id`, `modelo`, `imagen`, `descripcion`) VALUES
42  (19, 'Adidas', 'img1.jpg', 'Ta to guapo'),
43  (20, 'Nike', 'img2.jpg', 'Ta to guapo'),
44  (21, 'Puma', 'img3.jpg', 'Ta to guapo'),
45  (22, 'Jordan', 'img4.jpg', 'Ta to guapo'),
46  (23, 'SKECHERS', 'img5.jpg', 'Ta to guapo');
47
48  --
49  -- Índices para tablas volcadas
50  --
51
52  --
53  -- Índices de la tabla `zapatilla`
54  --
55  ALTER TABLE `zapatilla`
56  | ADD PRIMARY KEY (`id`);
57
58  --
59  -- AUTO_INCREMENT de las tablas volcadas
60  --
61
62  --
63  -- AUTO_INCREMENT de la tabla `zapatilla`
64  --
65  ALTER TABLE `zapatilla`
66  | MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=24;
67  COMMIT;
68
69  /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
70  /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
71  /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```


4- Ventajas de usar MVC y comparación con el 1º Proyecto

Usar el patrón MVC (Modelo-Vista-Controlador) ofrece varias ventajas en comparación con un enfoque más desorganizado como el utilizado en el primer proyecto de la biblioteca.

Ventajas del MVC:

1. Separación de preocupaciones: MVC divide la aplicación en tres componentes principales: el Modelo, la Vista y el Controlador. Esto permite una mejor organización del código y una separación clara de las responsabilidades.

2. Reutilización del código: Al dividir la aplicación en componentes independientes, es más fácil reutilizar el código en diferentes partes de la aplicación. Por ejemplo, el mismo modelo puede ser utilizado por varios controladores para diferentes funcionalidades.

3. Facilidad de mantenimiento: La división clara de responsabilidades hace que sea más fácil realizar cambios y mejoras en la aplicación sin afectar otras partes del sistema. Esto facilita el mantenimiento a largo plazo.

4. Escalabilidad: El patrón MVC facilita la escalabilidad de la aplicación, ya que cada componente puede ser escalado de forma independiente según sea necesario. Por ejemplo, si se necesita mejorar el rendimiento de la vista, se puede optimizar sin afectar al modelo o al controlador.

5. Testabilidad: Debido a la separación de preocupaciones, es más fácil escribir pruebas unitarias para cada componente del MVC. Esto ayuda a garantizar que cada parte de la aplicación funcione correctamente de forma independiente.

Comparación con el enfoque del primer proyecto:

- 1. Organización del código:** En el primer proyecto, el código puede estar más desorganizado debido a la falta de una estructura clara como la proporcionada por el patrón MVC. Esto puede dificultar la comprensión y el mantenimiento del código a medida que la aplicación crece.
- 2. Dificultad para realizar cambios:** En un enfoque menos estructurado, realizar cambios o agregar nuevas características puede ser más complicado, ya que las diferentes partes del código pueden estar más entrelazadas y los cambios en un lugar pueden afectar inesperadamente a otras áreas.
- 3. Escalabilidad limitada:** Con un enfoque menos estructurado, la escalabilidad puede ser más difícil de lograr, ya que las diferentes partes del sistema pueden estar más acopladas entre sí. Esto puede dificultar la expansión de la aplicación a medida que aumentan los requisitos y la carga de trabajo.

En resumen, el uso del patrón MVC proporciona una estructura clara y modular para el desarrollo de aplicaciones web, lo que facilita la organización del código, la reutilización, el mantenimiento y la escalabilidad. Comparado con un enfoque menos estructurado, MVC ofrece ventajas significativas en términos de claridad, mantenibilidad y escalabilidad del código.

5- Conclusión y Justificación

Conclusión:

El uso del patrón Modelo-Vista-Controlador (MVC) en el desarrollo de aplicaciones web proporciona una serie de ventajas significativas en comparación con enfoques menos estructurados. A lo largo de nuestra discusión sobre la aplicación web MVC que hemos estado comentando, hemos podido observar cómo MVC promueve la separación clara de responsabilidades entre los componentes del software, lo que facilita la organización del código, la reutilización, el mantenimiento y la escalabilidad.

Justificación:

1. Organización del código: MVC divide la aplicación en tres componentes principales: el Modelo, la Vista y el Controlador. Esto permite una mejor organización del código y una separación clara de las responsabilidades. Cada componente se encarga de una parte específica del flujo de trabajo de la aplicación, lo que facilita la comprensión y el mantenimiento del código a medida que la aplicación crece en tamaño y complejidad.

2. Reutilización del código: Al dividir la aplicación en componentes independientes, es más fácil reutilizar el código en diferentes partes de la aplicación. Por ejemplo, el mismo modelo puede ser utilizado por varios controladores para diferentes funcionalidades. Esto reduce la duplicación de código y promueve una base de código más limpia y mantenible.

3. Facilidad de mantenimiento: La división clara de responsabilidades en MVC hace que sea más fácil realizar cambios y mejoras en la aplicación sin afectar otras partes del sistema. Cada componente puede ser modificado de forma independiente, lo que facilita el mantenimiento a largo plazo y la evolución de la aplicación según las necesidades del usuario.

4. Escalabilidad: MVC facilita la escalabilidad de la aplicación, ya que cada componente puede ser escalado de forma independiente según sea necesario. Por ejemplo, si se necesita mejorar el rendimiento de la vista, se puede optimizar sin afectar al modelo o al controlador. Esto permite que la aplicación crezca y se adapte a medida que aumentan los requisitos y la carga de trabajo.

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 salian y consultas SQL de base de datos, estilos CSS de la página)

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