Modelo-Vista-Controlador

En este punto vamos a desarrollar un sistema Modelo-Vista-Controlador (MVC) básico para entender el funcionamiento de este patrón de diseño. La utilización del MVC nos facilitará la portabilidad de las aplicaciones y un mantenimiento más organizado de las mismas.

El MVC separa la programación entres aspectos fundamentales:

- El Modelo: Es la lógica de negocio de la aplicación y se encarga de la gestión de los datos.
- La Vista: Es la presentación de los datos al usuario.
- El Controlador: Es el encargado de gestionar los eventos y las comunicaciones entre el Modelo y la Vista.

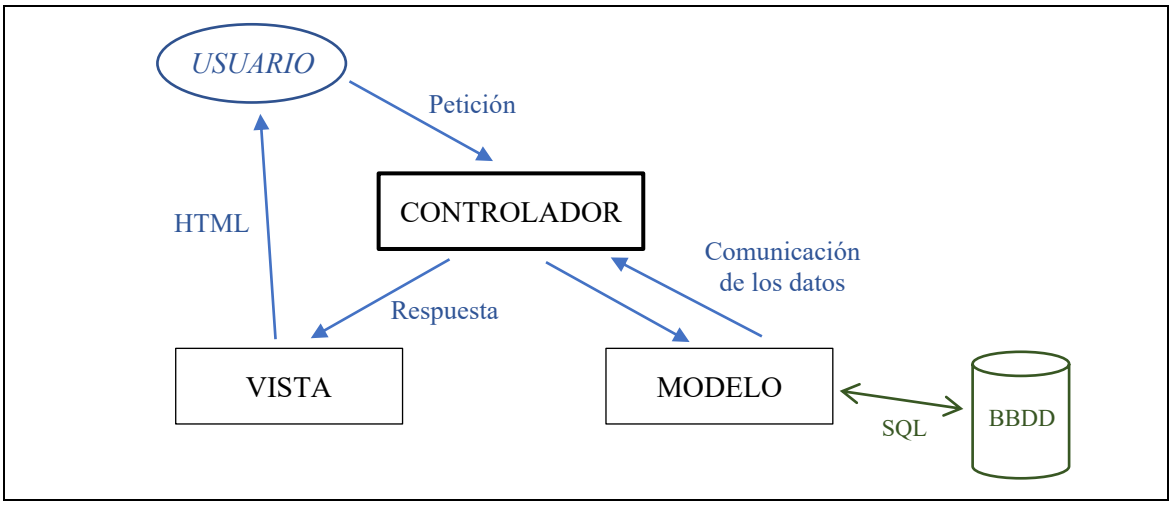

Figura 2. Modelo-Vista-Controlador

Desarrollando un CRUD

Vamos a realizar un CRUD utilizando el patrón MVC. Nos vamos a centrar en la gestión de un cliente, el cual estará almacenado en una base de datos MySQL. La tabla clientes tendrá la siguiente estructura.

Campo	Tipo
id	int(11), PK, autoincremental
nombre	varchar(32)
email	varchar(100)

La estructura de archivos de un proyecto MVC puede estar distribuida de la siguiente forma:

- model
  - bd.php
  - clientes.php
- view
  - css
    - app.css

    - layout
        - header.php
        - menu.php
        - footer.php
    - clientesmantenimiento.php
    - clientes.php
    - app.php
- controller
    - clientes.php
    - app.php
- config.php
- index.php

Archivo de configuración y punto de entrada a la aplicación

El archivo `config.php` contendrá la configuración de nuestra aplicación.

```
define("URLSITE", "http://localhost/crudmvc/");

define("SERVIDOR", "localhost");
define("USUARIO", "root");
define("CONTRASENA", "12345");
define("BASEDATOS", "mvc");
```

Una vez que hemos definido la dirección nuestro sitio web y los valores de acceso a la base de datos, desarrollaremos el punto de entrada de la aplicación `index.php`.

```
<?php
if (session_status() === PHP_SESSION_NONE)
    session_start();

require_once("config.php");
require_once("controller/app.php");
require_once("controller/clientes.php");

$controlador = '';
if(isset($_GET['c'])) :
    $controlador = $_GET['c'];

    $metodo = '';
    if(isset($_GET['m']))
        $metodo = $_GET['m'];

    switch($controlador) :
        case 'clientes' :
            if (method_exists('ClientesControlador', $metodo)) :
                ClientesControlador::{$metodo}();
            else :
                ClientesControlador::index();
            endif;

            break;
        default:
            AppControlador::index();
    endswitch;
else :
    AppControlador::index();
endif;
?>
```

Lo primero que realizamos en el requerimiento de la configuración y de los controladores de la aplicación (`controller/app.php`) y clientes (`controller/clientes.php`). Después verificamos que se ha pasado por GET tanto el controlador, en el argumento `c`, como el método del controlador a ejecutar, en el argumento `m`. Si se ha pasado, verificaremos que existe el método en

el controlador `ClientesControlador` mediante la función `method_exists`[90]. En caso contrario, llamaremos al método `index` del controlador de clientes.

Si no han pasado ningún controlador válido, vamos a la página principal de la aplicación, es decir, al método `index` del controlador `AppControlador`.

Modelo

Vamos a analizar el modelo que se encargará de desarrollar todo lo necesario para acceder a la base de datos. Los archivos se guardarán en la carpeta `model`. El primer archivo que codificaremos será el archivo `bd.php` que será el encargado de hacer todo el trabajo con la base de datos.

```
<?php
require once("config.php");

class BD
{
    private $con   = null; // Conexión a la BBDD.
    private $error = '';   // Mensaje de error.

    function __construct()
    {
        $this->error = '';

        try
        {
            // Creamos la conexión.
            $this->con = new PDO('mysql:host=' . SERVIDOR .
                                 ';dbname=' . BASEDATOS .
                                 ';charset=utf8',
                                 USUARIO,
                                 CONTRASENA);

            // Si se logra crear la conexión.
            if ($this->con)
            {
                // Ponemos los atributos para gestionar los errores con excepciones.
                $this->con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

                // El juego de caracteres será utf-8
                $this->con->exec('SET CHARACTER SET utf8');
            }
        }
        catch (PDOException $e)
        {
            $this->error = $e->getMessage();
        }
    }

    function __destruct()
    {
        // Cerramos la conexión a la BBDD.
        $this->con = null;
    }

    protected function _consultar($query)
    {
        $this->error = '';

        $filas = null;

        try
        {
            // Preparamos la consulta...
            $stmt = $this->con->prepare($query);

            // y la ejecutamos.
            $stmt->execute();

            // Si nos devuelve alguna fila...
```

[90] https://www.php.net/manual/es/function.methos_exists.php

```
            if ($stmt->rowCount() > 0)
            {
                // Creamos el array...
                $filas = array();

                // y lo rellenamos con los datos de la consulta.
                while ($registro = $stmt->fetchObject())
                    $filas[] = $registro;
            }
        }
        catch (PDOException $e)
        {
            $this->error = $e->getMessage();
        }

        // Devolvemos las filas obtenidas de la consulta.
        return $filas;
    }

    protected function _ejecutar($query)
    {
        $this->error = '';

        $filas = 0;

        try
        {
            // Ejecutamos la sentencia y guardamos el número de filas afectadas.
            $filas = $this->con->exec($query);
        }
        catch (PDOException $e)
        {
            $this->error = $e->getMessage();
        }

        // Devolvemos el número de filas afectadas.
        return $filas;
    }

    protected function _ultimoId()
    {
        // Devolvemos el id de la última fila insertada.
        return $this->con->lastInsertId();
    }

    public function GetError()
    {
        // Obtenemos el mensaje del error, si este se produce.
        return $this->error;
    }

    public function Error()
    {
        // Indicamos si ha habido algún error.
        return ($this->error != '');
    }
}
?>
```

Ahora crearemos el modelo para gestionar la tabla clientes de nuestra base de datos.

```
<?php
require_once("bd.php");

class ClientesModelo extends BD
{
    // Campos de la tabla.
    public $id;
    public $nombre;
    public $email;

    public $filas = null;

    public function Insertar()
    {
        $sql = "INSERT INTO clientes VALUES".
```

```
                " (default, '$this->nombre', '$this->email')";

            return $this->_ejecutar($sql);
        }

        public function Modificar()
        {
            $sql = "UPDATE clientes SET" .
                   " nombre='$this->nombre', email='$this->email'" .
                   " WHERE id=$this->id";

            return $this->_ejecutar($sql);
        }

        public function Borrar()
        {
            $sql = "DELETE FROM clientes WHERE id=$this->id";

            return $this->_ejecutar($sql);
        }

        public function Seleccionar()
        {
            $sql = 'SELECT * FROM clientes';

            // Si me han pasado un id, obtenemos solo el registro indicado.
            if ($this->id != 0)
                $sql .= " WHERE id=$this->id";

            $this->filas = $this->_consultar($sql);

            if ($this->filas == null)
                return false;

            if ($this->id != 0)
            {
                // Guardamos los campos en las propiedades.
                $this->nombre = $this->filas[0]->nombre;
                $this->email  = $this->filas[0]->email;
            }

            return true;
        }
    }
    ?>
```

Vista

Vamos a programar las vistas de nuestra aplicación, las cuales las guardaremos en la carpeta `view`. Empezaremos codificando la cabecera `header.php` y el pie `footer.php`, estos dos archivos los guardaremos en la carpeta `view/layout`. Por último, se codifica el archivo `app.css`, el cual guardaremos en la carpeta `view/css`, para darle un poco de estilo junto con Bootstrap a nuestra aplicación.

Primero codificamos la cabecera en el fichero `header.php`.

```
    <!DOCTYPE html>
    <html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <!-- Bootstrap CSS -->
        <link
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
            rel="stylesheet"
            integrity=
                 "sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
            crossorigin="anonymous">

        <link rel="stylesheet" type="text/css"
            href="<?php echo URLSITE; ?>view/css/app.css">
        <title>CRUD MVC</title>
```

```
</head>
<body>
    <?php require("menu.php"); ?>

    <br>

    <div class="panel">
```

Después el fichero `menu.php`.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="<?php echo URLSITE; ?>">Inicio</a>
    <button class="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent"
            aria-expanded="false"
            aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle"
             href="#"
             id="ClientesDropdown"
             role="button"
             data-bs-toggle="dropdown"
             aria-expanded="false">Clientes</a>
          <ul class="dropdown-menu"
              aria-labelledby="clientesDropdown">
            <li><a class="dropdown-item"
                   href="<?php echo URLSITE . '?c=clientes'; ?>">Clientes...</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item"
                   href="<?php echo URLSITE . '?c=clientes&m=facturas'; ?>">
                 Facturas...</a></li>
          </ul>
        </li>
      </ul>
      <span class="navbar-text">
          <a class="nav-item nav-link active"
             href="<?php echo URLSITE . '?c=ayuda'; ?>"> Ayuda</a>
      </span>
    </div>
  </div>
</nav>
```

Y por último, el pie en el fichero `footer.php`.

```
    </div>
</body>
</html>
```

Seguimos con un poco de estilo de nuestra aplicación codificando el fichero `app.css`.

```
.panel
{
   background: white;
   padding: 20px;
   box-shadow: 0px 0px 10px;
}
```

Finalmente codificaremos cada una de las vistas de nuestro CRUD.

Empezamos por la vista de la página principal codificando el fichero `app.php`.

```
<?php require("layout/header.php"); ?>

<h1>CRUD :: PÁGINA PRINCIPAL</h1>
```

```
<br/>

<?php require("layout/footer.php"); ?>
```

Para mostrar los clientes a gestionar codificando el fichero `clientes.php`.

```
<?php require("layout/header.php"); ?>

<h1>CLIENTES</h1>

<br/>

<table class="table table-striped table-hover" id="tabla">
<thead>
<tr class="text-center">
    <th>Id</th>
    <th>Nombre</th>
    <th>Email</th>
    <th></th>
</tr>
</thead>
<tbody>
<?php
if ($clientes->filas) :
    foreach ($clientes->filas as $fila) :
?>
<tr>
    <td style="text-align: right; width: 5%;"><?php echo $fila->id; ?></td>
    <td><?php echo $fila->nombre; ?></td>
    <td><?php echo $fila->email; ?></td>
    <td style="text-align: right; width: 50%;">
        <a href="index.php?c=clientes&m=editar&id=<?php echo $fila->id; ?>">
            <button type="button" class="btn btn-success">Editar</button></a>
        <a href="index.php?c=clientes&m=borrar&id=<?php echo $fila->id; ?>" >
            <button type="button" class="btn btn-danger borrar"
            onclick="return confirm('¿Estás seguro de borrar el registro <?php
                     echo $fila->id; ?>?');">Borrar</button></a>
    </td>
</tr>
<?php
    endforeach;
endif;
?>
</tbody>
<tfoot>
<tr>
    <td colspan="4">
        <a href="index.php?c=clientes&m=nuevo">
        <button type="button" class="btn btn-primary">Nuevo</button>
        </a>
    </td>
</tr>
</tfoot>
</table>

<?php require("layout/footer.php"); ?>
```

A continuación, codificamos la vista para añadir o modificar los clientes en el fichero `clientesmantenimiento.php`. Observar que según el valor de la variable `$opcion` cambiamos el comportamiento del formulario, haciendo que se comporte como un formulario para insertar o bien para modificar.

```
<?php require("layout/header.php"); ?>

<h1>CLIENTES</h1>

<br/>

<h2><?php echo ($opcion == 'EDITAR' ? 'MODIFICAR' : 'NUEVO'); ?></h2>

<form action="<?php echo 'index.php?c=clientes&m=' .
             ($opcion == 'EDITAR' ? 'modificar&id=' . $cliente->id : 'insertar'); ?>"
      method="POST">
  <label for="nombre" class="form-label">Nombre</label>
```

```
    <input type="text"
           class="form-control"
           name="nombre"
           id="nombre"
           value="<?php echo ($opcion == 'EDITAR' ? $cliente->nombre : ''); ?>"
           required/>
    <br/>
    <label for="email" class="form-label">Email</label>
    <input type="text"
           class="form-control"
           name="email"
           id="email"
           value="<?php echo ($opcion == 'EDITAR' ? $cliente->email : ''); ?>"
           required/>
    <br/>
    <button type="submit" class="btn btn-primary">Aceptar</button>
    <a href="<?php echo URLSITE . '?c=clientes'; ?>">
    <button type="button"
            class="btn btn-outline-secondary float-end">Cancelar</button>
    </a>
  </form>

  <?php require("layout/footer.php"); ?>
```

La ultimo vista que vamos a ver es la de mostrar los errores de nuestra aplicación

```
  <?php
  if (session_status() === PHP_SESSION_NONE)
      session_start();

  require_once("../config.php");
  require("layout/header.php");
  ?>

  <br/>
  <div class="alert alert-danger" role="alert">
    <h4 class="alert-heading">¡Error!</h4>
    <p>Ha habido un error al realizar la operación:</p>
    <p style="font-style: italic;"><?php echo $_SESSION["CRUDMVC_ERROR"]; ?></p>
    <hr>
    <p class="mb-0">
      <button type="submit" class="btn btn-primary"
              onclick="window.history.back()">Reintentar</button>
      <a href="<?php echo urlsite; ?>"><button type="button"
         class="btn btn-outline-secondary float-end">Cancelar</button></a>
  </p>
  </div>

  <?php require("layout/footer.php"); ?>
```

Controlador

El controlador de nuestra aplicación gestiona las peticiones solicitadas por el usuario. En la función Nuevo, le asignamos a la variable `$opcion` el valor de `NUEVO` , mientras que en la función `Editar` se le asigna el valor `EDITAR`. Con esto controlamos el comportamiento del formulario como vimos anteriormente.

La codificación del controlador se muestra a continuación.

```
  <?php
  if (session_status() === PHP_SESSION_NONE)
      session_start();

  require_once("model/clientes.php");

  class ClientesControlador
  {
      static function index()
      {
          $clientes = new ClientesModelo();

          $clientes->Seleccionar();
```

```
        require_once("view/clientes.php");
    }

    static function Nuevo()
    {
         $opcion  = 'NUEVO';  // Opción de insertar un cliente.

        require_once("view/clientesmantenimiento.php");
    }

    static function Insertar()
    {
        $cliente = new ClientesModelo();

        $cliente->nombre = $_POST['nombre'];
        $cliente->email  = $ POST['email'];

        if ($cliente->Insertar() == 1)
            header("location:" . URLSITE . '?c=clientes');
        else
        {
            $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

            header("location:" . URLSITE . "view/error.php");
        }
    }

    static function Editar()
    {
        $cliente = new ClientesModelo();

        $cliente->id = $_GET['id'];

        $opcion = 'EDITAR'; // Opción de modificar un cliente.

        if ($cliente->seleccionar())
            require_once("view/clientesmantenimiento.php");
        else
        {
            $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

            header("location:" . URLSITE . "view/error.php");
        }
    }

    static function Modificar()
    {
        $cliente = new ClientesModelo();

        $cliente->id     = $_GET['id'];
        $cliente->nombre = $_POST['nombre'];
        $cliente->email  = $_POST['email'];

        // Aquí hay que tener cuidado, en el caso de que se pulse el botón de aceptar
        // pero no se haya modificado nada, la función modificar devolverá un cero,
        // por eso hay que comprobar que no hay error.
        if (($cliente->Modificar() == 1) || ($cliente->GetError() == ''))
            header("location:" . URLSITE. '?c=clientes');
        else
        {
            $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

            header("location:" . URLSITE . "view/error.php");
        }
    }

    static function Borrar()
    {
        $cliente = new ClientesModelo();

        $cliente->id = $_GET['id'];

        if ($cliente->Borrar() == 1)
            header("location:" . URLSITE. '?c=clientes');
        else
        {
            $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();
```

```
                header("location:" . URLSITE . "view/error.php");
            }
        }
    }
```

Conclusión

Hemos visto una aplicación sencilla, básica y claramente mejorable, de un CRUD utilizando el patrón MVC. El objetivo es entender cómo funciona este patrón de cara al desarrollo de aplicaciones PHP con frameworks[91] como Laravel, Symfony , Zend, etc.

Le hemos añadido algo de vistosidad utilizando Bootstrap y gestionamos el error a mostrar mediante sesiones.

Ejercicios

115. Amplía el CRUD para que el cliente tenga: apellidos, contraseña, dirección, cp, población, provincia y fecha de nacimiento.

116. Amplia el CRUD para gestionar las facturas de un cliente[92]. Las facturas tendrán la siguiente estructura:

Campo	Tipo
id	int(11), PK, autoincremental
cliente_id	int(11)
numero	int(11)
fecha	datetime

117. Amplía el CRUD para gestionar las líneas de una factura. Las líneas tendrán la siguiente estructura:

Campo	Tipo
id	int(11), PK, autoincremental
factura_id	int(11)
referencia	int(11)
descripcion	varchar(32)
cantidad	decimal(10, 3)
precio	decimal(10,2)
iva	decimal(5,2)
importe	decimal(10, 2)

En la tabla anterior el campo "importe" que será el resultado de la siguiente operación:

importe = cantidad * precio * (1 + iva / 100.0)

[91] https://en.wikipedia.org/wiki/Category:PHP_frameworks

[92] Utilizar la función parse_url, https://www.php.net/manua/es/function.parse-url.php, para saber qué controlador hay que seleccionar: