Guía para implementar el MVC con un ejemplo

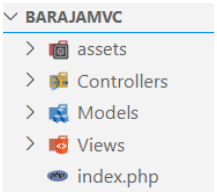
Objetivos:

- Crear una aplicación básica usando el MVC
- Uso de POO

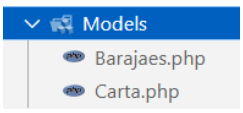
En concreto, se desea crear una aplicación web para un casino que nos permita realizar algunas de las acciones típicas:

Pasos:

1. Crea las carpetas Models, Views y Controllers ,así como el archivo index.php en el directorio raíz. Dicho archivo se encargará de gestionar la aplicación y las diferentes peticiones de rutas.



2. En la carpeta Models cada clase estará aislada en un único. Crea los archivos Barajaes.php y Carta.php y posteriormente le daremos contenido



3. Comenzaremos definiendo dentro de la carpeta Models el modelo de clase Barajaes y el de la clase Carta. Recuerda usar espacio de nombres y tomar decisiones de diseño acerca del constructor, promoción de propiedades, etc.

   Ejemplo:

Models > Carta.php

```php
<?php
namespace Models;
class Carta {
    const PALOS = ["ESPADAS", "OROS", "COPAS", "BASTOS"];
    const CARTAS = array(1=>"as", 2=>"dos", 3=>"tres", 4=>"cuatro", 5=>"cinco",
                         6=>"seis", 7=>"siete", 8=>"ocho", 9=>"nueve",
                         10=>"sota", 11=>"caballo", 12=>"rey");

    function __construct(private int $numero,
                         private string $palo){
```

```php
<?php
namespace Models;

class Barajaes {

    private array $baraja;

    function __construct() {
        $baraja=[];
        $palos = Carta::PALOS;
        for ($i=0;$i<sizeof($palos);$i++) {
            for ($j=1;$j<=12;$j++){
                $carta = new Carta($j,$palos[$i]);
                $baraja[] = $carta;

            }
        }
        $this->setBaraja($baraja);

    }
```

Añade los getters y setters oportunos en cada una de las clases y aquellos métodos que consideres necesarios.

Prueba a crear instancias de ambas clases en el archivo index.php y muestra el valor de alguna de sus propiedades.

index.php

```php

<?php
//Probamos que podemos acceder a las clases definidas en el espacio de nombres Models
require_once 'Models/Carta.php';
require_once 'Models/Barajaes.php';
use Models\Barajaes;
use Models\Carta;

// Probamos que podemos trabajar con la clase Carta
echo "<p><strong>Creando un objeto de la clase Carta</strong></p>";
$numero =5;
$palo = "OROS";
if (Carta::compruebaNumero($numero)& (Carta::compruebaPalo($palo))){
    $micarta = new Carta(5,"OROS");
    echo $micarta . "<br>";
}else {
    echo "Revise el número de la carta y el palo";

}

// Probamos que podemos trabajar con la clase Barajaes
echo "<p><strong>Creando un objeto de la clase Barajaes</strong></p>";
$mibaraja = new Barajaes;
echo " <pre>";
var_dump($mibaraja);
echo " </pre>";
```

4. A medida que nuestro proyecto crece, añadir cada clase mediante el require_once() correspondiente no será productivo. Por ello, lo más lógico es que se carguen automáticamente con un script .
   En clase hemos creado para otros proyectos el archivo “ autoloader.php”. Este archivo contiene una función, que recibe la clase que está buscando en formato namespace. Convierte el formato y busca un archivo con el mismo nombre que el namespace. Si el archivo existe, se importa usando la función require(se puede modificar a requiere_once() o include_once()).

   autoloader.php

   ```php
   <?php
   spl_autoload_register(function ($clase) {
       $directorio_clase = str_replace('\\', '/', $clase);
       if (file_exists($directorio_clase . '.php')) {
           require $directorio_clase . '.php';
       }
   });
   ```

   Las clases, a partir de ahora, se cargarán con el comando *use*.

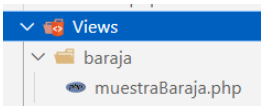
5. Continuando con el patrón MVC trabajaremos de la siguiente forma: el usuario solicitará información a través de una vista, la vista se comunica con el controlador y éste le indica al modelo que se encargue de extraer la información de proporcionarle la información requerida. A su vez, el modelo le pasará la información al controlador y éste enviará al usuario la información a través de una vista.

   Por lo tanto, necesitamos definir en la carpeta Controller un controlador llamado BarajaController. Este controlador será el responsable de gestionar las llamadas al objeto de información y después generará la página HTML que se devuelve al navegador con los datos solicitados por el usuario.

Esta nueva clase tendrá definido el namespace y hará uso , como mínimo del modelo Barajaes. Además, implementará el método apropiado para mostrar las cartas de la baraja.

```php
namespace Controllers;

use Models\Barajaes;
class BarajaController
{
    public function mostrarBaraja():void
    {
        $baraja = new Barajaes();
        require_once './Views/baraja/muestraBaraja.php';
    }
}
```

Como puedes observar, es necesario crear en la carpeta Views la vista correspondiente.
Para organizar mejor el código, creamos una carpeta asociada a cada controlador dentro de la carpeta Views. En este caso, dentro de la carpeta baraja pondremos un archivo por cada acción que realice el controlador y conlleve mostrar información al usuario. En nuestro caso el archivo a crear se llamará muestraBaraja.php



Por ahora, las vistas se cargan en el controlador usando el correspondiente require_once().
El contenido de la vista será el siguiente:

Views > baraja > muestraBaraja.php

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

<?php
    if(!is_dir('assets/img'))
    {
    echo "No tenemos imágenes";
    }else {
            $barajaes = $baraja->getBaraja();
            foreach($barajaes as $carta)
            {
                $image = "./assets/img/".$carta->getPalo()."_".$carta->getNumero().".jpg";
                    if(file_exists($image))
                    {
                       echo "<img src='$image'/>";

                    }else{
                        echo "No tenemos la imagen de esta carta";
                    }

                }
    }

?>
</body>
</html>
```

Por último, solo nos queda usar el controlador desde el archivo index.php. Crea una instancia del controlador BarajaController y llama al método muestraBaraja() de dicho controlador.

```php
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset = "utf-8">
    <meta http-equiv = "X-UA-Compatible" content = "IE=edge">
    <meta name = "viewport" content = "width=device-width, initial-scale=1">
    <title>Juego Cartas</title>
  </head>
  <body>
    <?php
    // Mostrar las cartas de una baraja usando el controlador BarajaController
    require_once 'autoloader.php';

    use Controllers\BarajaController;

    $controlador = new BarajaController();
    $controlador->mostrarBaraja();

    ?>
  </body>
</html>
```

Con esto, ya tenemos prácticamente implementado nuestro MVC.

6. Crear un punto de entrada único a la aplicación. Diseñando nuestro controlador Frontal.

   Comenzaremos escribiendo el código en el archivo index.php y después crearemos la clase oportuna.
   La idea es acceder mediante una URL reconocida por la aplicación y así llegar al objeto de la clase que se encarga de gestionar esta petición. Al ponerlo en la misma

URL los valores nos llegarán por GET, esa es la forma de recoger las variables que necesitamos.
Vamos a avanzar en su creación paso a paso:

- Deseamos no tener que poner cada uno de los métodos de forma manual y para ello recogeremos variables por GET. Modificamos el index.php para que reciba por la URL la acción a realizar (el método del controlador). Usaremos el mismo fichero index.php como vista para solicitar que se realice una acción determinada.

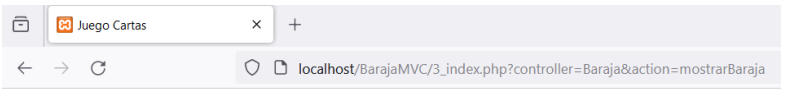
```php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset = "utf-8">
    <meta name = "viewport" content = "width=device-width, initial-scale=1">
    <title>Juego Cartas</title>
</head>
<body>
<h2>Casino </h2>
<!-- Creando un controlador frontal -->
<?php require_once 'autoloader.php';
use Controllers\BarajaController;

if((!empty($_GET))):

    $controlador = new BarajaController;

    if(isset($_GET['action']) && method_exists($controlador, $_GET['action'])){
                $action = $_GET['action'];
                $controlador->$action();
            }else{
                echo "La pagina que buscas no existe";
            }

 ?>
<?php else: ?>

    <a href="http://localhost/BarajaMVC/3_indexA.php?action=mostrarBaraja">Mostrar las cartas de la baraja española</a>

<?php endif; ?>

</body>
</html>
```

Puedes probar este ejemplo pasando el parámetro ?action=mostrarBaraja en la URL.De esta forma, acabamos de hacerlo dinámico.
A continuación, mejoramos el código para que los controladores también se carguen de forma dinámica. Además haremos que nuestra URL sea más corta ya que podemos aprovechar que todos nuestros controladores acaban en “Controller” y están dentro del espacio de nombre Controllers.

index.php

```php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset = "utf-8">
    <meta name = "viewport" content = "width=device-width, initial-scale=1">
    <title>Juego Cartas</title>
</head>
<body>
<h2>Casino </h2>
<!-- Creando un controlador frontal -->
<?php require_once 'autoloader.php'

if((!empty($_GET))):

    if(isset($_GET['controller'])){
        //al llamarse todos igual acortamos la URL añadiendo directamente nosotros Controller
        $nombre_controlador = 'Controllers\\'.$_GET['controller'].'Controller';

        if(class_exists($nombre_controlador)){
            $controlador = new $nombre_controlador();
            if(isset($_GET['action']) && method_exists($controlador, $_GET['action'])){
                $action = $_GET['action'];
                $controlador->$action();
            }else{
                echo "La pagina que buscas no existe";
            }

        }else{
            echo "La pagina que buscas no existe";
        }

    }else{
        echo "La pagina que buscas no existe";
        exit();
    }
    ?>
<?php else: ?>

    <a href="http://localhost/BarajaMVC/index.php?controller=Baraja&action=mostrarBaraja">Mostrar las cartas de la baraja española</a>

<?php endif; ?>

</body>
</html>
```

Como puedes observar, accedemos añadiendo a la URL los parámetros correspondientes al nombre del controlador y de la acción. Ejemplo:



Casino

Las modificaciones que acabamos de hacer en el index.php nos han permitido construir el **controlador frontal** para que dependiendo de lo que llega por la URL cargue diferentes ficheros.

- Una vez que los conceptos están claros, el siguiente paso consiste en poner este código en una clase con un método estático para que podamos usarlo con todos los controladores de nuestro proyecto.
  Añadimos un nuevo controlador a nuestro proyecto con el nombre FrontController.php:

```php
<?php
namespace Controllers;

class FrontController {
        public static function main()
        {
            if(isset($_GET['controller'])){
                    //al llamarse todos igual acortamos la URL añadiendo directamente nosotros Controller
            $nombre_controlador = 'Controllers\\'.$_GET['controller'].'Controller';



            if(class_exists($nombre_controlador)){
                $controlador = new $nombre_controlador();
                if(isset($_GET['action']) && method_exists($controlador, $_GET['action'])){
                    $action = $_GET['action'];
                    $controlador->$action();
                  }else{
                    echo "La pagina que buscas no existe";
                  }

                }else{
                        echo "La pagina que buscas no existe";
                    }

            }else{
                        echo "La pagina que buscas no existe";
                        exit();
                    }

            }
}
```

Cambiamos el fichero index.php para que cargue el controlador y ejecute el método main()

```php
<!-- Creamos una clase para el controlador frontal y la cargamos -->
<?php
require_once 'autoloader.php';
use Controllers\FrontController;
FrontController::main();
?>
<header>
    <h1>Casino </h1>
    <nav>
        <ul>
            <li><a href="http://localhost/BarajaMVC/4_index.php?controller=Baraja&action=mostrarBaraja">Mostrar las cartas de la baraja española</
```

- Mejoramos el código de la clase FrontController para que cuando no exista el controlador nos muestre un mensaje (queremos mejorar el típico 404). Para controlar los errores creamos un nuevo controlador que llamaremos ErrorController.php. El contenido será básico y se irá modificando a medida que crezca la aplicación.

ollers > ErrorController.php

```php
<?php
namespace Controllers;

class ErrorController{

    public static function show_error404():string{
        return "<p>La página que buscas no existe</p>";
    }

}
```

- Una vez creada esta nueva clase, hacemos uso de ella en nuestro controlador frontal.

```
llers > FrontController.php
<?php
namespace Controllers;

class FrontController {

    public static function main(): void
    {

        if(isset($_GET['controller'])){

           $nombre_controlador = 'Controllers\\'.$_GET['controller'].'Controller';

        }else{
                $nombre_controlador = 'Controllers\\'.CONTROLLER_DEFAULT;

            }
            /* si ha ido bien creamos una instancia del controlador y llamamos a la acción*/
        if(class_exists($nombre_controlador)){

            $controlador = new $nombre_controlador();

            if(isset($_GET['action']) && method_exists($controlador, $_GET['action'])){
                    $action = $_GET['action'];
                    $controlador->$action();
            }elseif(!isset($_GET['controller']) && !isset($_GET['action'])){
                    $action_default = ACTION_DEFAULT;
                    $controlador->$action_default();

                }else{
                    echo ErrorController::show_error404();
                }
        }else{
              echo ErrorController::show_error404();
            }
    }


}
```

Observa que otra de las mejoras introducidas en que tenemos un controlador por defecto:

```
$nombre_controlador = 'Controllers\\'.CONTROLLER_DEFAULT; // se establece en un archivo de configuración
```

Como nuestra aplicación aún es pequeña y no tenemos claro cuál será el controlador por defecto pondremos uno de los que ya hemos creado. En nuestro proyecto debemos pensar en el mejor y añadirlo en un archivo junto con otras constantes que iremos necesitando.
Puedes añadir el siguiente código en el archivo config.php . No olvides cargarlo en el index.php.

```
config.php
<?php

// defino el controlador por defecto y el método por defecto que tengo
define('CONTROLLER_DEFAULT','BarajaController');
define('ACTION_DEFAULT', 'mostrarBaraja');
```

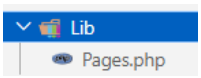
Sería interesante añadir una lista blanca de URLs en tu proyecto.

7. Hacer uso del principio DRY (Don't Repeat Youself) .
   Podemos seguir haciendo mejoras tales como incluir plantillas Bootstrap o bien usar algo que ya conocemos de ejercicios anteriores.
   En la mayoría de las vistas de nuestra aplicación existen partes comunes que podemos aislar . Por ejemplo creando el archivo header.php y footer.php. Puedes crear una nueva carpeta dentro de Views para organizar este contenido.
   Organiza el fichero index.php anterior para que produzca el mismo resultado:

```php
<?php
require_once 'autoloader.php';
require_once 'config/config.php';
require_once 'Views/layout/header.php';
use Controllers\FrontController;
FrontController::main();
require_once 'Views/layout/footer.php';
?>
```

8. Continuaremos mejorando nuestro código creando un motor de plantillas.
   Es decir crearemos una biblioteca o un módulo para generar contenido dinámico y personalizado mediante una plantilla que contiene variables que se reemplazan con datos específicos antes de que la página se entregue al cliente.

   Crearemos una nueva carpeta llamada " Lib" y dentro de ella una clase llamada Pages con un único método que nos permitirá incluir una plantilla y asignarle variables.



   El proceso que seguiremos para crear cualquier página en nuestra web, nuestras vistas, será utilizar un objeto de la clase Pages. Este objeto, será el responsable de llamar al fichero que se le indique pasando los datos correspondientes, que serán tratados como variables PHP dentro de ese archivo, y genera el contenido HTML definitivo.

   El contenido del archivo Pages.php es el siguiente:

Pages.php

```php
<?php
namespace Lib;

class Pages {

    public function render(string $pageName, array $params = null): void{
        /*$pageName es el nombre de nuestra plantilla. El nombre del fichero que se pretende mostrar(sin la extensión).Por ejemplo, muestraBaraja
          $params este array es el contenedor de las variables que deseamos pasar a la vista (la página a mostrar).
          $params es un array con un índice asociativo. Para crear las variables, recorremos la lista y
          usamos el índice como nombre de variable usando la propiedad variable de PHP ($$name = $value)
          */
        if($params != null){
            foreach($params as $name => $value){
                $$name = $value;
            }
        }
        require_once "views/layout/header.php";
        require_once "views/$pageName.php"; // incluimos la página indicada
        require_once "views/layout/footer.php";
    }
```

Cuando el controlador pasa los datos obtenidos a la vista, en lugar de llamar directamente al fichero usando require_once() usaremos un objeto de la clase Pages y pasaremos el nombre del fichero a mostrar y las variables que sean necesarias.

```php
<?php
    namespace Controllers;
    use Models\Barajaes;
    use Lib\Pages;

    class BarajaController
    {
        private Barajaes $baraja;
        private Pages $pages;

        public function __construct()
        {
            $this->baraja = new Barajaes();
            $this->pages= new Pages();
        }
        public function mostrarBaraja(Barajaes $mazo = null):void
        {
            if($mazo == null) {
                $mazo = $this->baraja;

            }
            $this->pages->render('baraja/muestraBaraja', ['mazo' => $mazo->getBaraja()]);
            // require_once './Views/baraja/muestraBaraja.php';
        }
```

Modificamos también la vista muestraBaraja.php

```php
<?php

    if(!is_dir('assets/img'))
    {
    echo "No tenemos imágenes";
    }else {

            foreach($mazo as $carta)
            {

                $image = "./assets/img/".$carta->getPalo()."_".$carta->getNumero().".jpg";
                    if(file_exists($image))
                    {
                       echo "<img src='$image'/>";

                    }else{
                        echo "No tenemos la imagen de esta carta";
                    }

                }
    }
```