UD8 – MVC

2º CFGS
Desarrollo de Aplicaciones Web

2022-23

1.- Patrón MVC

Al desarrollar software suelen darse **problemas generales que son similares** en estructura.

Estos problemas se **resuelven de manera parecida** independientemente del software a desarrollar.

Los **patrones de diseño del software** son una gran ayuda en este aspecto.

Uno de los patrones más conocidos y usados es:

Modelo, Vista, Controlador → MVC

1.- Patrón MVC

El patrón MVC promueve la organización de la aplicación **en tres partes bien diferenciadas** y **débilmente acopladas**.

Un **acoplamiento débil** indica que los cambios en una parte del código afectan muy poco al resto de partes.

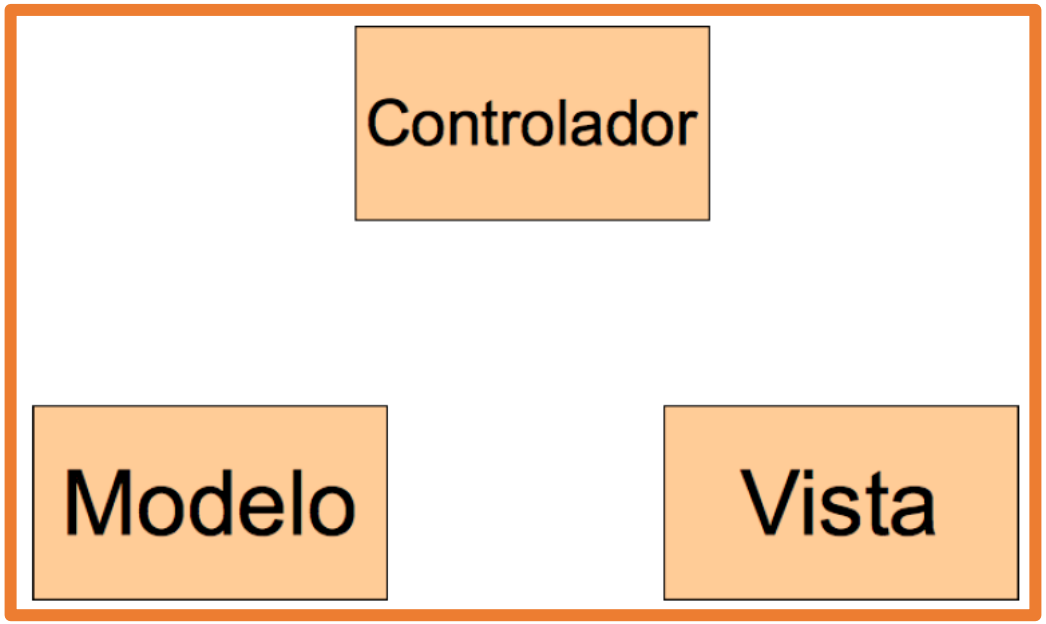
En el mejor caso, al usar el patrón MVC, un cambio no afectará en nada a otras partes.

1.- Patrón MVC

Modelo: en este componente se trabaja con los datos (accesos a la base de datos).

Vista: se encarga de organizar los datos obtenidos por el modelo y con ellos genera la interfaz gráfica que se muestra al usuario.

Controlador: se encarga de gestionar las peticiones a la aplicación.



1.- Patrón MVC

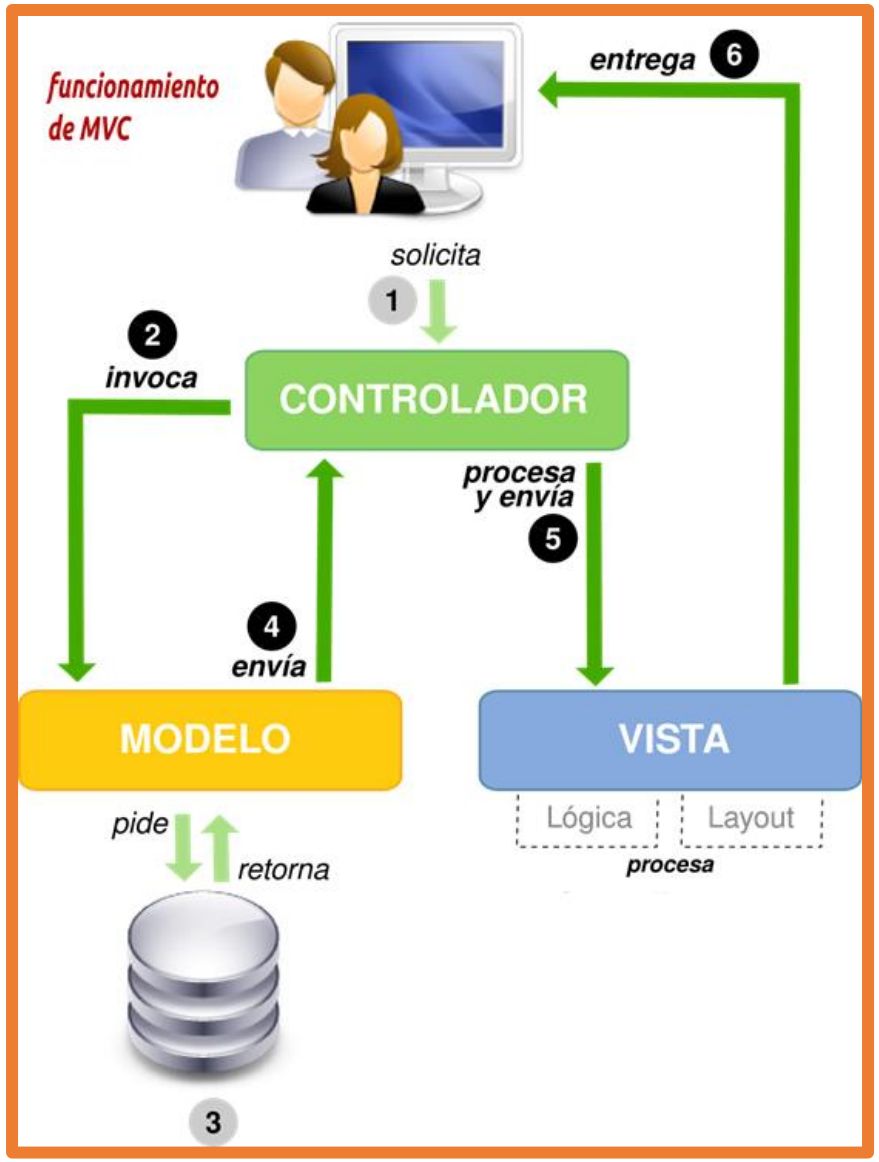
El **controlador recibe la orden de entrada** y se encarga de **gestionar todo el proceso**.

En el caso de necesitarlo, el controlador usará los servicios **del modelo para obtener los datos** necesarios.

Una vez se ha trabajado con los datos el controlador los **enviará los datos en crudo** a la **vista**.

En ocasiones no se envían datos a la vista.

La vista se encarga de mostrar la funcionalidad relacionada con la orden recibida por el procesador.

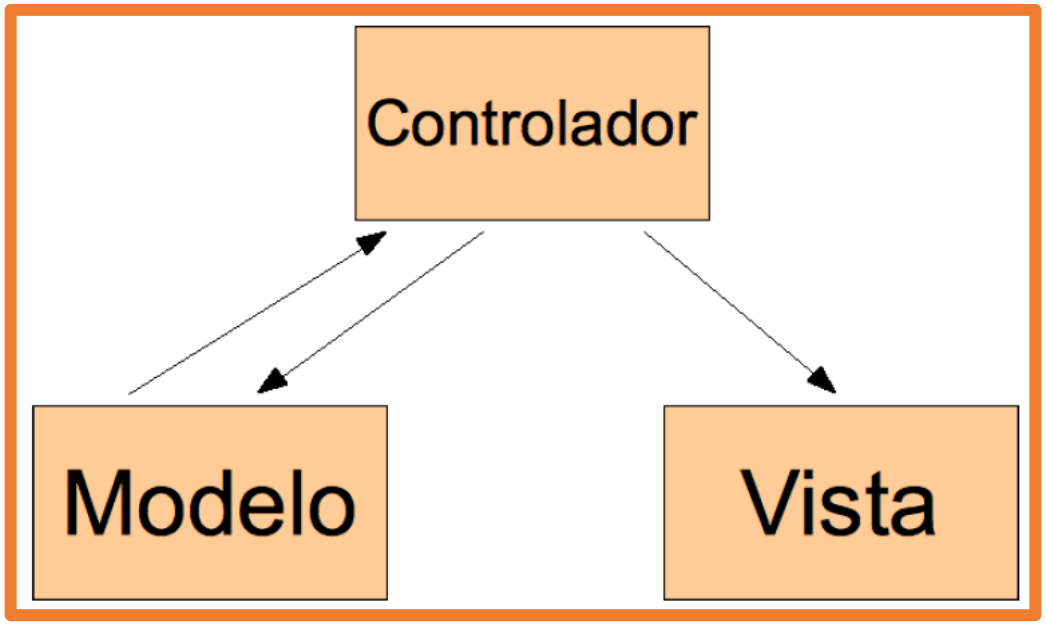


1.- Patrón MVC

El patrón MVC tiene otras soluciones posibles.

Esta solución tiene la característica de que la vista nunca interacciona con el modelo.

En las aplicaciones web se usa esta solución ya que permite centralizar en el controlador todas las peticiones (URL's) a la aplicación web.



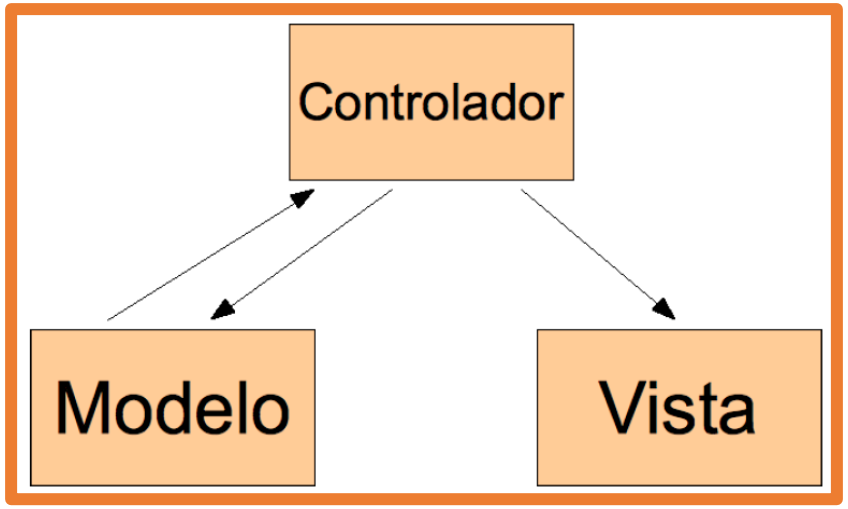
1.- Patrón MVC

Con el patrón MVC:

Habrá **un controlador por cada tabla de la base de datos**. Generalmente un controlador gestiona las acciones sobre una única tabla, aunque hay casos en los que no gestiona ninguna.

Habrá **un modelo por cada tabla de la base de datos**. Un modelo realiza acciones sobre una única tabla.

Habra **una o más vistas por cada tabla de la base de datos** para poder mostrar las diferentes acciones sobre los elementos de esa tabla.



1.- Patrón MVC

Las **aplicaciones web** típicamente se plantean con esta solución del patrón.

- El **controlador recibe** una petición HTTP y la **procesa**.
- El controlador, haciendo uso del **modelo** calcula los datos de salida y los entrega a la vista.
- La **vista** se encarga de construir una respuesta HTTP con las cabeceras y el cuerpo adecuado. El documento generado por la vista será un HTTP, XML o JSON.

2.- Organización de los archivos

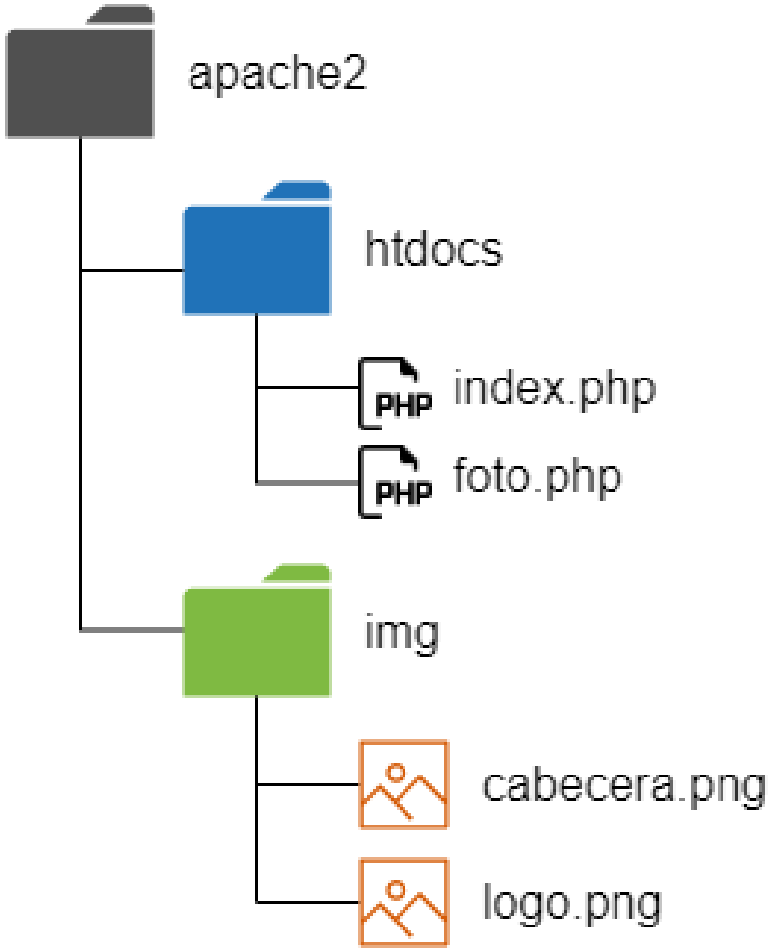
Como se ha visto con anterioridad, el servidor web aloja una aplicación web dentro de una carpeta, el **Document Root** (directorio raíz de la aplicación).

De esta manera los clientes solo podrán pedir directamente recursos que se encuentren dentro del directorio raíz.

Internamente el servidor web puede trabajar con todos los archivos del ordenador aunque no se encuentren en el directorio raíz.

2.- Organización de los archivos

Ejemplo: teniendo esta estructura de directorios se añade una medida de seguridad de cara al robo de imágenes.



Para cargar una foto desde index.php se utiliza el código:

```
<img src="foto.php?img=logo" alt="logo_web">
```

Dentro de de foto.php para cargar la imagen se utiliza un código similar a:

```
$img = imagecreatefrompng('../img/'. $_GET['img'] .'.png';
…
$imagepng($img);
$imagedestroy($img);
```

2.- Organización de los archivos

El cliente nunca podrá acceder a una imagen directamente ya que no se encuentran alojadas dentro del directorio raíz.

Para que el cliente pueda acceder a una imagen siempre se pasará por el archivo foto.php.

De esta manera si se añade una marca de agua a la imagen no habrá forma de eliminar dicha marca de agua.

2.- Organización de los archivos

Gracias a esta característica, explicada anteriormente, se puede organizar el código de la aplicación web para que desde el navegador solo se pueda acceder al código estrictamente imprescindible.

Así, para una aplicación web desarrollada con **MVC** se tendrá una configuración similar a la siguiente:

El directorio **web** será el directorio raíz (htdocs).

El directorio **app** contendrá:

- Librerías PHP
- Ficheros de configuración (XML, JSON…)

3.- URL's amigables

Las URL's amigables son imprescindibles en una aplicación web actual.

Ventajas de las URL amigables:

- Desaparece la query string.
- Eliminar la extensión de los archivos (.php) de la URL.
- **Esconder la estructura de directorios del servidor web.**
- Mejor posicionamiento SEO.
- Facilitar a los usuarios el recordar las URL.
- Permiten sustituir caracteres "extraños" o poco recomendables

El uso de las URL's amigables ayuda en el desarrollo de aplicaciones web con el patrón MVC

4.- Aplicación web MVC

Se pueden desarrollar aplicaciones web con el patrón MVC directamente, como se va a realizar en esta unidad.

Hoy en día la mayoría de aplicaciones web se desarrollan utilizando algún Framwork debido a las ventajas que estos ofrecen como seguridad, organización.

Lo más habitual es que los Framework utilicen alguna de las soluciones del modelo MVC.

Más adelante se usará un Framework para desarrollar aplicaciones web y este Framework también hará uso del patrón MVC.

5.- Aplicación de ejemplo

Para ver la creación y desarrollo de una aplicación bajo el patrón MVC se va a realizar una aplicación de manera guiada.

En las siguientes sesiones de clase se estudiarán los diferentes conceptos y los archivos necesarios.

La solución que se estudiará es una entre las muchas posibles soluciones.

5.- Aplicación de ejemplo

La aplicación que se va a desarrollar permitirá elaborar y consultar un base de datos de manga y anime.

Solo se usará una tabla en la base de datos, que contendrá los siguientes campos:

nombre	creador
genero	demografía
estreno	fin
tomos	capitulos

5.- Aplicación de ejemplo

La aplicación permitirá:

- Insertar nuevos manga/anime
- Realizar consultas por los diferentes campos: nombre, genero…
- Contendrá un menú accesible desde cualquier parte de la aplicación.

5.- Aplicación de ejemplo

Creación de la estructura de directorios

Por razones de comodidad se usará el directorio raíz. Esta práctica no es segura ya que el cliente podrá acceder a todo el directorio, cosa que se tiene que evitar.

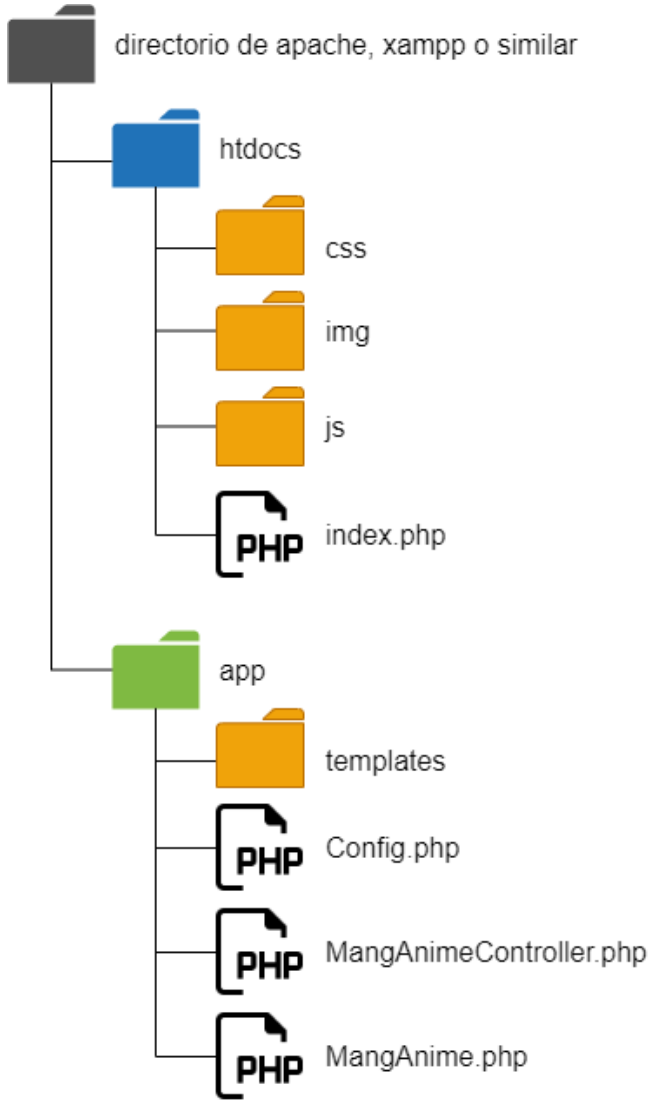
La implementación del patrón MVC será la siguiente:

- index.php: **Controlador frontal** para centralizar todas las peticiones.
- Clase para el controlador: **MangaAnimeController.php**.
- Clase para el modelo: **MangAnime.php**.
- Clase para los parámetros de configuración: **Config**.
- Las vistas serán **plantillas PHP** dentro del directorio **app/templates**.
- Los archivos CSS, JS, imágenes y el **controlador frontal** estarán en el directorio **web**.

5.- Aplicación de ejemplo

Con esta estructura de directorios desde el cliente solo se tiene acceso a la ejecución del archivo index.php.

De esta manera se añade un nivel de seguridad a la aplicación web.



5.- Aplicación de ejemplo

Aclaración sobre archivos que se incluyen:

Anteriormente se ha visto que los nombres de los archivos que se incluyen acaban en **.inc.php .**

Estos archivos se guardan en un directorio para ello llamado **inc** o **includes**.

Por norma general, cuando se usa el MVC esta notación no se utiliza.

Práctica

Actividad 1:

Creación del entorno.

6.- El controlador frontal

Al desarrollar aplicaciones web sin seguir ningún patrón de diseño lo más habitual es crear un archivo por cada página de la aplicación, además de los posibles archivos a incluir (inc.php):

index.php
registro.php
carrito.php
noticia.php
grupo.php
…

Esta práctica presenta algunos problemas cuando la aplicación crece.

6.- El controlador frontal

Problemas más significativos:

- Los scripts suelen realizar **tareas que son comunes**:
  Interpretar y manipular la petición.
  Comprobar las credenciales de seguridad.
  Cargar la configuración.
  Por ello se utiliza la técnica de incluir ficheros: **include/require**.

¿Y si cuando se ha finalizado el desarrollo se requiere añadir una nueva funcionalidad/característica que hace uso de una nueva librería?

Habrá que añadir este código a todos los scripts PHP de la aplicación.
- Degradación en el mantenimiento.
- Aumenta la probabilidad de fallos.

6.- El controlador frontal

Problemas más significativos:

- Si se realiza una petición de un script PHP que no existe en la aplicación, el servidor contesta con un error 404.

Para poder controlar el aspecto de esa página 404 y que sea similar al estilo del resto de la aplicación se requiere de cambiar la configuración del servidor.

6.- El controlador frontal

Como el problema es tener muchos scripts "de entrada" que comparten una gran cantidad de código la solución es tan simple como **utilizar un único script de entrada**.

A este script de entrada se le conoce como **controlador frontal**.

De esta manera el único recurso "ejecutable" accesible para los navegadores será el archivo index.php.

La clave para poder realizar esta acción es el uso de la **query string**:

index.php**?accion=saludo**

6.- El controlador frontal

Mapeo de rutas

Como las query string no son una buena práctica, es preferible configurar **URL amigables** para que las sustituyan.

Por ello antes de iniciar el desarrollo se deben definir las URL asociadas a cada una de las páginas de la aplicación web:

URL + query string	URL amigable	Acción
/index.php?accion=inicio	/inicio	Mostrar la pantalla de inicio
/index.php?accion=listar	/listar	Listar mangas/animes
/index.php?accion=insertar	/insertar	Insertar un manga/anime
/index.php?accion=buscar	/buscar	Buscar manga/anime
/index.php?accion=ver&id=x	/ver/x	Ver el mangaanime con id x

6.- El controlador frontal

Mapeo de rutas

Cada una de las URL's definidas será una funcionalidad de la aplicación y a cada una se le asociará un método público en la clase **Controller**.

A estos métodos se les suele llamar **acciones**.

Cada acción se encarga de calcular los datos necesarios para la página que se debe mostrar, usando la clase **Model** si es necesario.

Cuando tenga los datos se los pasará a una plantilla (**Vista**) que será la que construirá el **documento HTML** que se envía al cliente.

6.- El controlador frontal

Mapeo de rutas

Para implementar el controlador frontal se usará el archivo **index.php** que estará ubicado en el directorio **web**.

Aunque se le llame **controlador** central, este archivo no pertenece al componente **Controller** del patrón **MVC**.

Este controlador central solo es el punto de entrada a la aplicación web.

- Se declara un array asociativo para mapear las rutas hacia las acciones de los controladores.
- Se carga la configuración del modelo y de los controladores.
- Se parsea la URL y se carga la acción correspondiente a la petición.

7.- Configuración global

En este tipo de aplicaciones es habitual agrupar todos los parámetros de configuración de la aplicación en un único archivo.

Así, incluyendo este archivo en el controlador frontal, toda la configuración estará disponible en cualquier parte de la aplicación.

Para esta práctica guiada el archivo de configuración será **Config.php**.

En muchos Framework el archivo de configuración se llama **.env**.

Práctica

Actividad 2:

Controlador frontal, rutas y configuración global.

8.- Clase Controller

Secciones de la aplicación y controladores

Para una aplicación web se pueden crear tantos controladores que se quiera.

Generalmente en un mismo controlador se agrupan las funcionalidades sobre un elemento del modelo (una tabla de la base de datos).

Se pueden crear controladores adicionales para agrupar funciones.

En la aplicación de ejemplo como solo se trabaja con la tabla **manganime** solo se necesitará un controlador.

8.- Clase Controller

Secciones de la aplicación y controladores

Cada controlador contendrá la definición de una clase.

Un controlador para una tabla de la base de datos generalmente se llamará como el nombre de la tabla seguido de la palabra controller.

En la aplicación de ejemplo, **ManganimeController** y el archivo se llamará de manera similar **ManganimeController.php**.

Cada clase Controller implementará tantos métodos como URL's se hayan definido sobre la tabla en cuestión. Estos métodos también son llamados **acciones**.

8.- Clase Controller

Las acciones del controlador MangaAnimeControlador

Si una **acción** tiene que trabajar con la base de datos, utilizará los métodos de la clase que define el **modelo** (MangAnime.php) necesarios.

Si una **acción** tiene que mandar información a la vista, en esa acción se declarará un **array asociativo** habitualmente llamado **params** con los datos a enviar.

En una acción no aparece información de cómo se mostrarán los datos ya que esto es tarea de la vista.

8.- Clase Controller

Todas las acciones tienen las misma estructura:

- Realizar operaciones.
- Realizar acciones sobre la base de datos (si es necesario).
- Llamar a una plantilla.

A continuación, se estudiarán algunas de las acciones necesarias para la aplicación de ejemplo MangAnime.

8.- Clase Controller

listar()

- Se declara un **objeto del modelo** que permitirá el acceso a la base de datos.
- Con el objeto anterior se obtendrán los elementos de la tabla necesarios.
- Los elementos obtenidos se almacenan en el array **params**.
- Se incluye el archivo de la vista: **templates/mostrarMangaAnimes.php**. Esta plantilla se encargará de crear el documento HTML con los datos del array **params**.

8.- Clase Controller

insertar() – la lógica es un poco más compleja porque debe mostrar el formulario y también recibir los datos del mismo, validarlos y si son correctos, insertarlos en la base de datos.

- Se declara un array asociativo vacío (**params**). Los índices del array deben coincidir con los campos de la tabla.
- Se comprueba que sea una petición **POST**, en caso negativo se mostrará el formulario, en caso afirmativo se han recibido datos y se debe almacenar el manga/anime.
- Se incluye la plantilla con el formulario (**formInsertar.php**)**,** que mostrará los campos vacíos o con los campos si se produjo un error.

Práctica

Actividad 3:

Clase Controller.

9.- Vistas

Vista

Las aplicaciones web generalmente generan un documento HTML que es el que envían al navegador (cliente), pero también puede generar otro tipo de documentos como JSON, XML…

En esta unidad se verá la generación de documentos HTML.

En la siguiente unidad se verá la utilidad de generar archivos JCON, XML… y cómo generarlos.

9.- Vistas

Con el fin de reutilizar código (premisa fundamental de la programación), para las vistas se hará uso de plantillas:

Layout - parte que representa todo el marco común de la aplicación:
cabecera, menú, pie de página

Plantilla – parte que representa la acción que se ejecuta en cada momento.

9.- Vistas

Como se ha visto en el controlador, todas las acciones terminan incluyendo una plantilla que se encuentra en **app/templates**.

Por ejemplo: app/templates/**mostrarMangAnime.php**

Cada plantilla se encarga de generar el contenido específico de la acción y de cargarlo en el layout, para ello almacena el contenido en una variable llamada **$contenido**.

El **layout** principal de la aplicación está definido en el archivo **app/templates/layout.php**.

Este archivo contiene la estructura básica de HTML con las partes comunes y además, utiliza la variable **$contenido** para completar el documento HTML.

9.- Vistas

En las plantillas es habitual utilizar la notación alternativa de las instrucciones de control para facilitar la lectura:

```
<?php foreach ($array as $elemento) : ?>
    Aquí el código HTML y PHP (entre <?php ?>)
<?php endforeach; ?>
```

En las plantillas se recomienda solo usar las sentencias **echo**, **if/endif** y **foreach/endforeach**.

También se recomienda usar la versión abreviada de echo **<?= ?>**.

9.- Vistas

ob_start() y **ob_get_clean()**

Estas funciones permiten que la salida estándar se guarde en un buffer interno en vez de añadirla al documento HTML de salida.

De esta manera se puede almacenar una cadena de caracteres en una variable sin ir concatenando con . Como se ha hecho hasta ahora.

Así en el IDE que se utilice ser visualizará mejor el código generado.

Práctica

Actividad 4:

Layout y plantillas.

10.- Modelo

Igual que ocurre con los controladores, en una aplicación web desarrollada con el patrón MVC puede haber tantos modelos como se quiera.

Habitualmente se crea un modelo por cada tabla de la base de datos.

En cada modelo se implementarán todas las acciones para la tabla con la que se relaciona el modelo: insert, update, delete y select.

En un archivo para un modelo se define una clase que se suele llamar de la misma manera que la tabla con la que se relaciona y de la misma manera se nombrará al archivo: MangaAnime.php.

10.- Modelo

Clase MangaAnime

Contiene una serie de funciones que permiten guardar o recuperar datos de la base de datos.

Dependiendo de la cantidad de funcionalidades de la aplicación se tendrán que implementar más o menos funciones.

Cuando el controlador necesita usar el modelo creará un objeto de la clase MangAnime, en el constructor se realizará la conexión a la base de datos.

Práctica

Actividad 5:

El modelo.

Actividad 6:

Completando la funcionalidad.

10.- Modelo

Organización de archivos

Cuando en una aplicación se utilizan varios controladores y modelos estos se deben organizar en carpetas para ello.

Así, igual que se tiene la carpeta **templates** se tendrán las carpetas **controllers** y **models**.

Práctica

Actividad 7:

Añadir nuevo controlador.