



# MVC

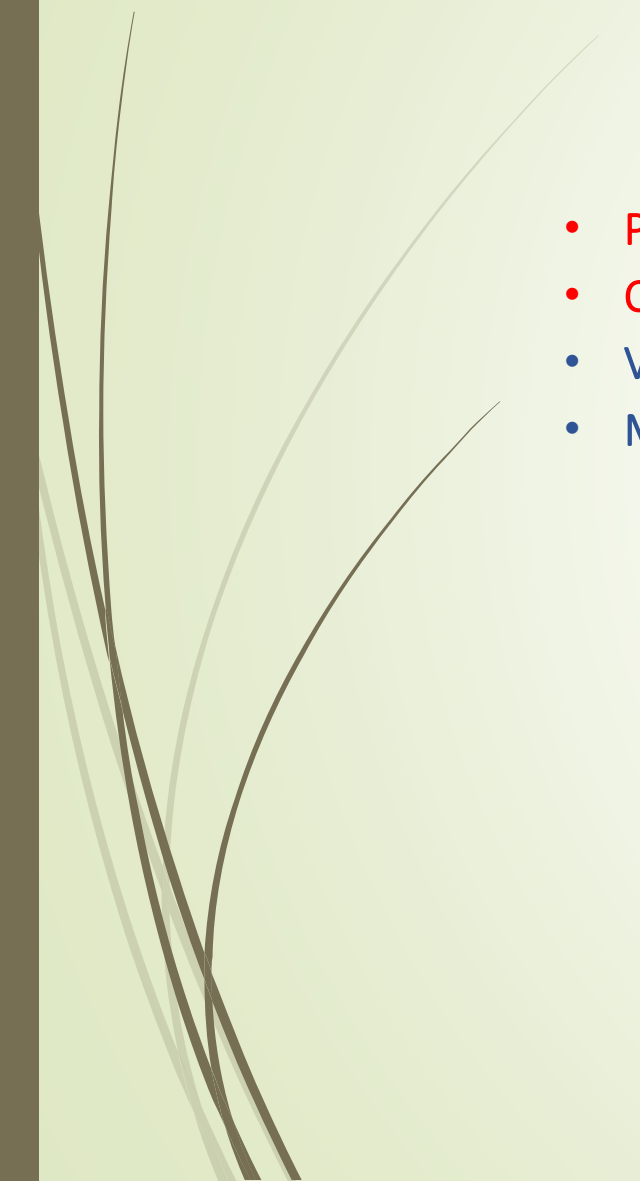
Modelo

Vista

Controlador



# Vocabulario

- Patrón de arquitectura
  - Controlador -> Controller
  - Vista -> View
  - Modelo -> Model
- 

# Definición

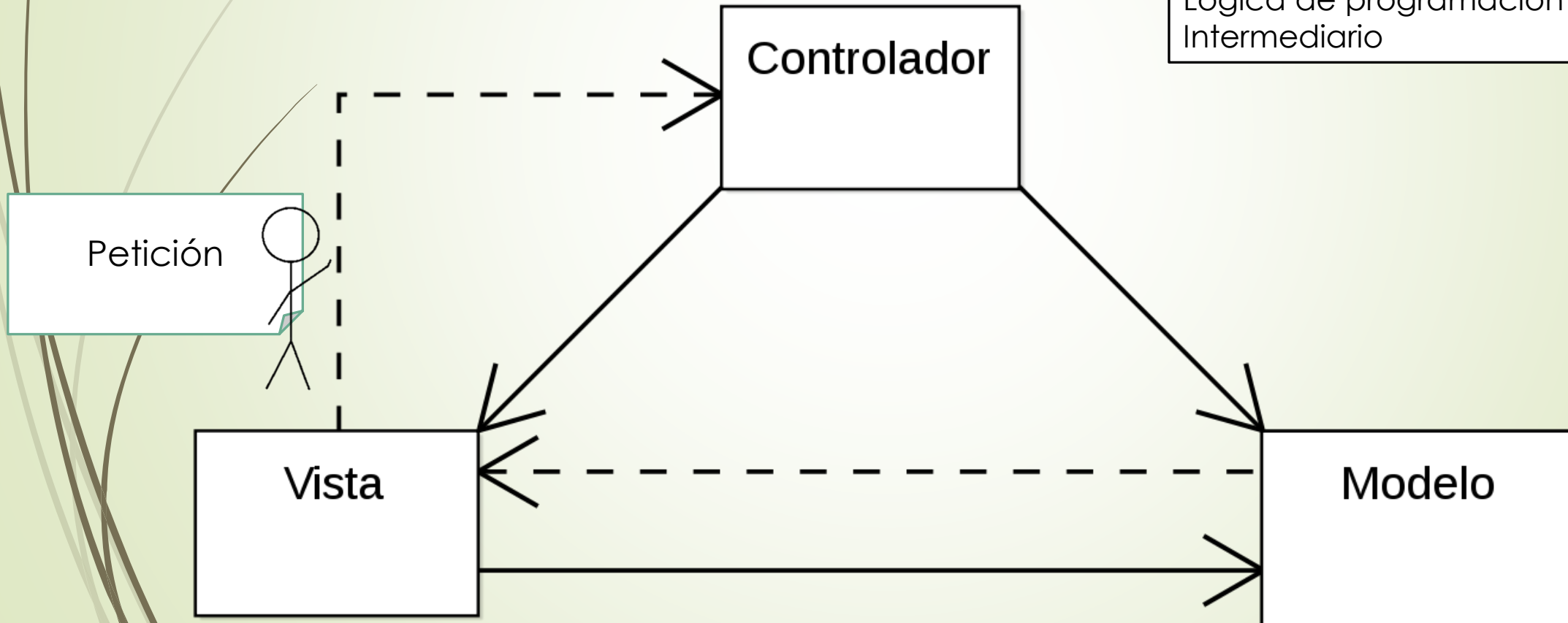
**Modelo-vista-controlador (MVC)** es un **patrón** de arquitectura de software

Presentación de la información

Separa  
el  
código  
en capas

Datos almacenados

Lógica de programación  
Intermediario



# Contolador → Controller

Hace de intermediario entre la 'vista' y el 'modelo'

Responde a eventos


(usualmente  
acciones del  
usuario)

También puede enviar  
comandos a su 'vista' asociada  
si se solicita un cambio en la  
forma en que se presenta el  
'modelo'

(por ejemplo, desplazamiento o scroll por  
un documento o por los diferentes  
registros de una base de datos)

Invoca peticiones al 'modelo'  
cuando se hace alguna solicitud  
sobre la información

(por ejemplo, editar un documento o  
un registro en una base de datos).



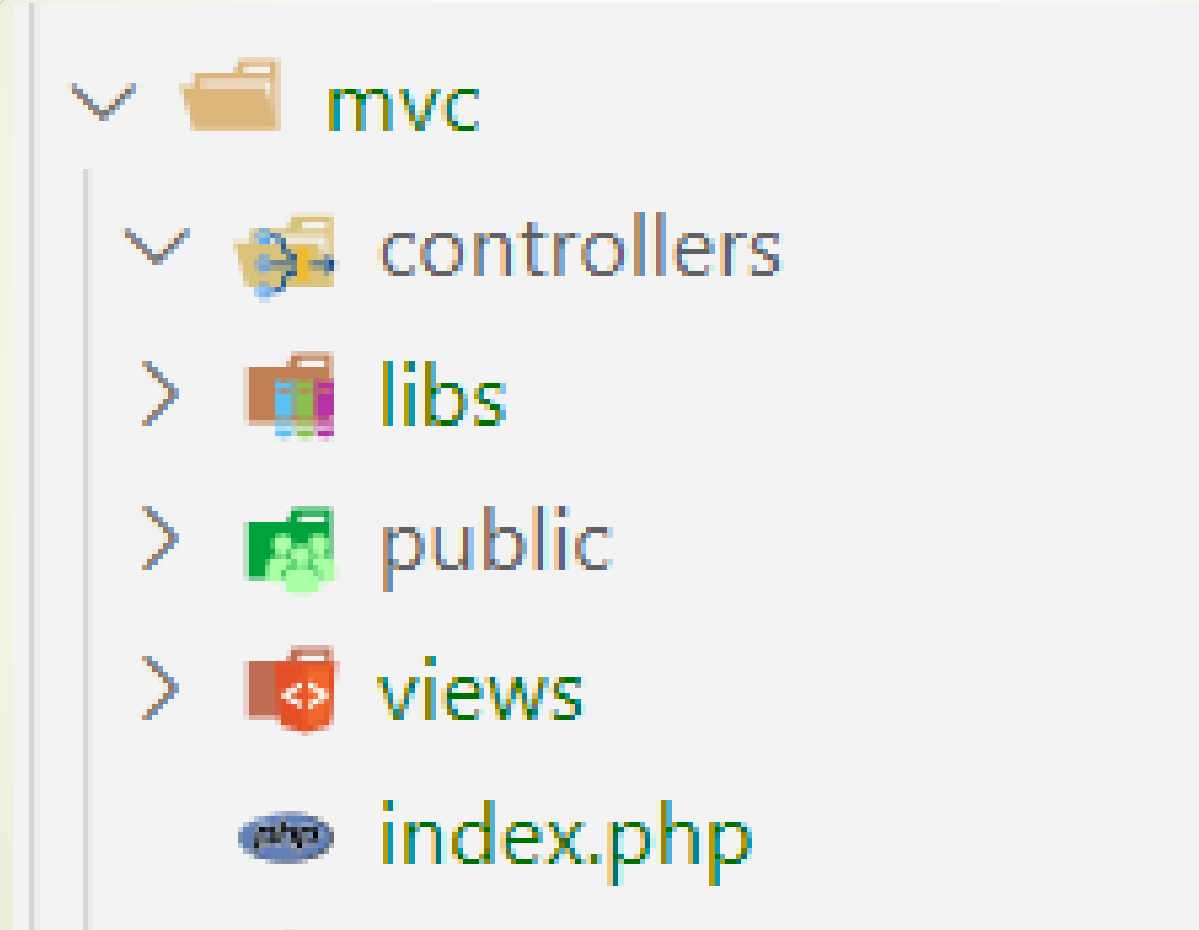
# Contolador → Controller

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad es una capa que sirve de **enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación.**

Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

# Estructura



# Un poco de código

Antes de crear nuestros elementos (objetos) para la gestión de nuestra aplicación habrá que realizar una serie de comprobaciones del sistema. Esto lo realizaremos en el fichero index.php

```
declare(strict_types=1); //declaración "obliga" a utilizar los tipos de datos que indiquemos en funciones y clases

//error_reporting(0); //Si descomentamos esta línea NO se mostrarán errores. En modo producción

$minPHPVersion = phpversion(); //Función que retorna un string que indica la versión de php del servidor

if($minPHPVersion<'7.3.1')
    die('Versión mínima de PHP: 7.3.1');//Función que detiene la aplicación y muestra el mensaje que hay a continuación
unset($minPHPVersion); //Función que elimina una variable. Puesto que no la vamos a utilizar otra vez
```

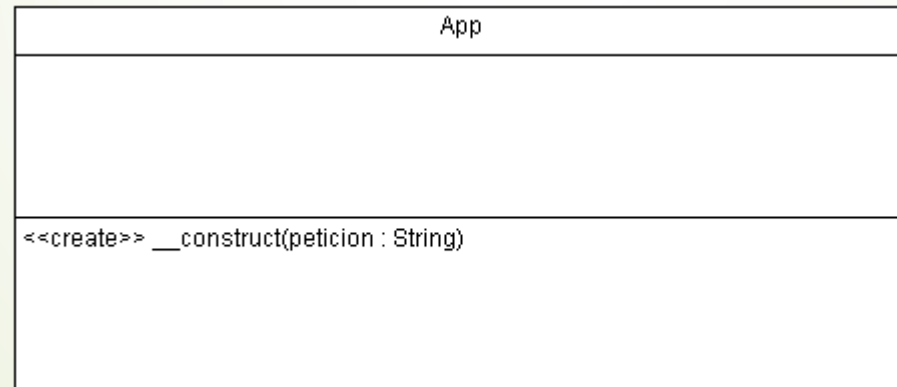


Index.php

# Un poco de código

Nuestra página va a recibir peticiones bien por URI (URL) o por formulario que serán gestionados por nuestro Controlador.

Crearemos una clase que llamará a susodicho Controlador en función de la petición que reciba. Esta clase “inicializadora” la podemos llamar como queramos, en nuestro caso la llamaré **App**  
Por organización la almacenaré en mi carpeta de librería **lib** y posteriormente la instanciaré desde el index



```
require_once('libs/app.php'); //incrustamos la clase App
$peticion = isset($_REQUEST['page'])?$_REQUEST['page']:'home'; //Escuchamos la petición, en cas
o de que no exista será home
new App($peticion); //lanzamos la petición
```



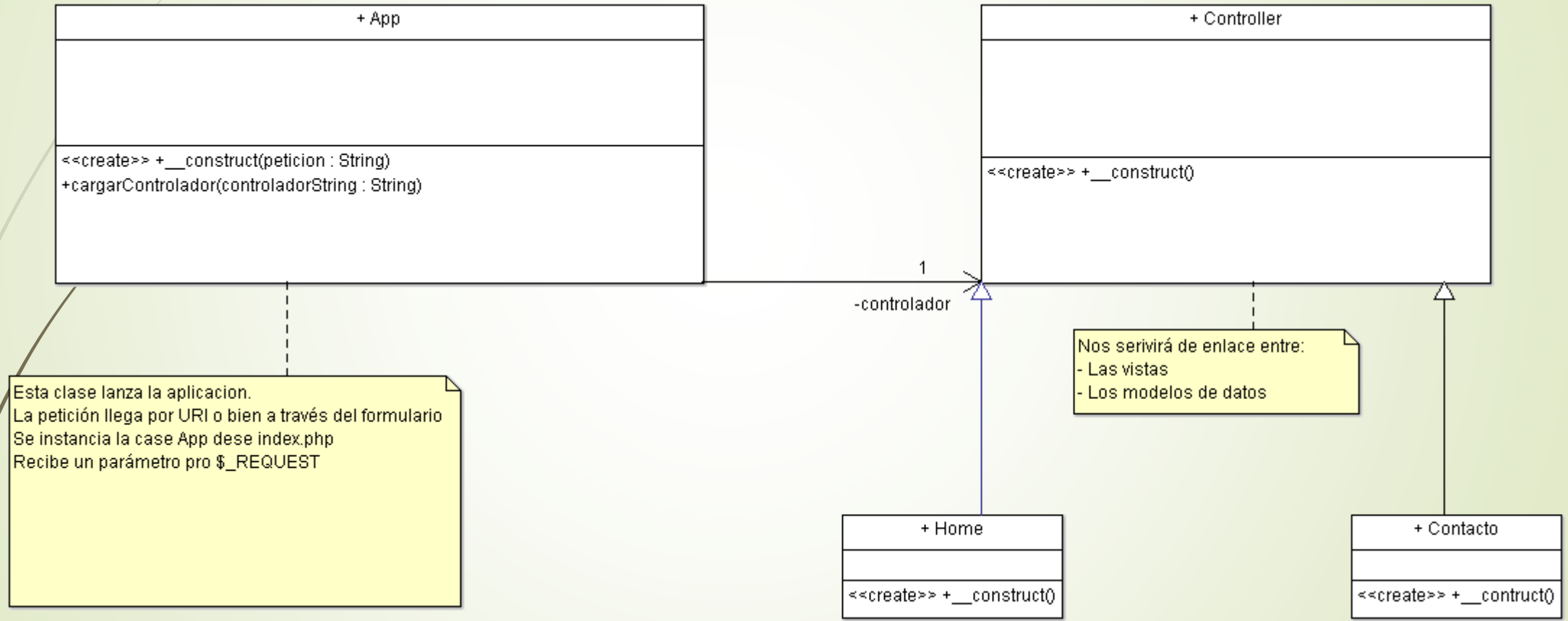
# Un poco de código

Para cargar el controlador (clase base) y como el objetivo es utilizar el lenguaje orientado a objetos crearemos un clase llamada Controller que almacenaremos en nuestra librería de carga **lib**

En la carpeta controllers crearemos tantas clases derivadas como peticiones recibamos. Si una petición no es reconocida cargaremos la clase padre (Controller)

A continuación mostramos el diagrama.

# Un poco de código



# Funciones, constantes interesantes

## `DIRECTORY_SEPARATOR`

Constante que nos devuelve la barra slash “/” o “\” según nos encontremos en un sistema Unix o Windows

## `__DIR__`

Constante que termina y empieza con doble guión bajo. Nos muestra la ruta (no URI) del directorio actual de trabajo. Útil cuando queremos cargar fichero (p.e con `require_once`) y este directorio puede cambiar de nombre

## `substr()`

Función que nos permite extraer una cadena de texto

## `ucfirst()`

Función que nos convierte la primera letra de un string en mayúscula. La utilizaremos para crear nombres de clase.

# Funciones, constantes interesantes

`file_exists()`

Función que comprueba si existe un fichero en una ruta dada

`class_exists()`

Función que comprueba si existe una clase

`require_once()`

Función que nos permite añadir un fichero a nuestro script