



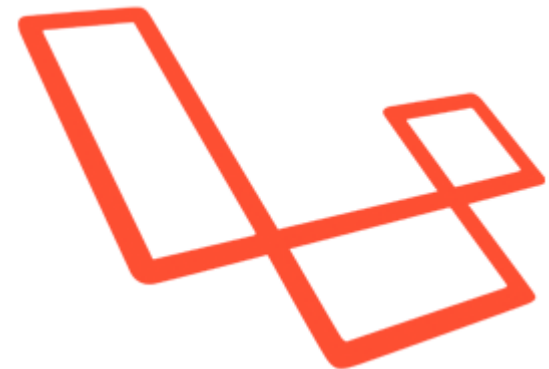
UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE
PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Introducción a Laravel

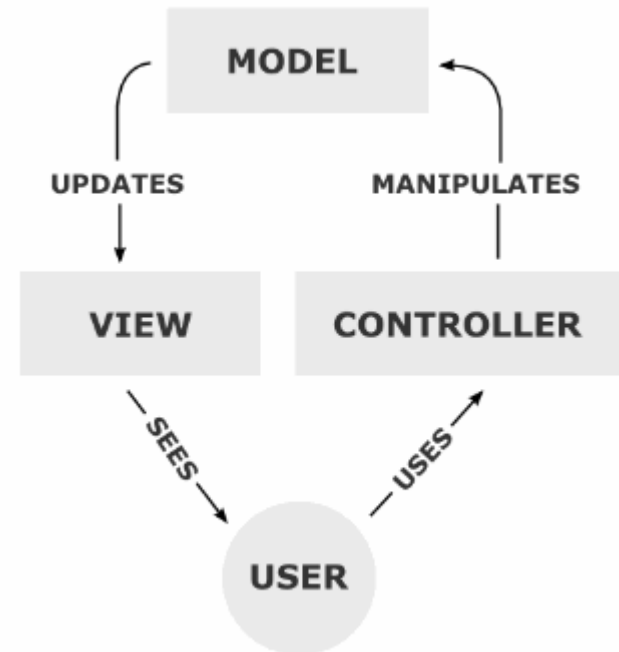
1. Introducción
2. Instalación
3. Estructura y funcionamiento
4. Rutas
5. *Artisan*
6. Vistas
7. Plantillas con *Blade*



- Laravel es un *framework* de código abierto para el desarrollo de aplicaciones web en PHP 5 que posee una sintaxis simple y elegante.
- Características principales:
 - Creado en 2011 por Taylor Otwell.
 - Inspirado en *Ruby on rails* y *Symfony*, de quien posee dependencias.
 - Esta diseñado para utilizar el patrón MVC.
 - Integra un sistema de mapeado de datos relacional llamado *Eloquent* ORM.
 - Utiliza un sistema de plantillas llamado *Blade*, el cual hace uso de la cache para darle mayor velocidad.

MVC es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario.

- **Modelo:** Es la representación de la información, gestiona el acceso a la información.
- **Controlador:** Contiene la lógica de negocio de la aplicación. Responde a eventos y realiza peticiones al modelo cuando es necesario.
- **Vista:** Separa la capa de representación visual de la capa de datos y del controlador.



- Requisitos:
 - PHP \geq 7.2.0
 - MySQL
 - PHP Extensions: BCMath, CType, JSON, Mbstring, OpenSSL, PDO, Tokenizer, XML
- Proceso para la instalación:
 1. Instalar servidor Web XAMPP
 2. Instalación de Composer
 3. Descargar Laravel y generar un proyecto
 4. Instalación de extensiones
 5. Configuración y prueba de Laravel.

- Instalación:
 - Abrir la página <https://www.apachefriends.org> y bajar la última versión para Mac de XAMPP desde la sección de descargas.
 - Hacemos doble clic sobre el fichero DMG descargado para iniciar la instalación.
 - Esto nos instalará el servidor en la ruta “/Applications/XAMPP”.
- Uso:
 - Para iniciar o parar servicios utilizamos la aplicación “manager-osx”.
 - La carpeta pública para nuestros proyectos Web está situada en “/Applications/XAMPP/htdocs”.
 - Para acceder a una web local en el navegador escribimos: <http://localhost>
- *Nota:* en Mac tendremos que parar el servidor por defecto con:

```
$ sudo /usr/sbin/apachectl stop
```

- Desde la versión 4 de Laravel, la creación de un proyecto nuevo se realiza con Composer.
- Composer es un gestor de dependencias para PHP.
- Instalación:

```
$ curl -sS https://getcomposer.org/installer | php  
$ sudo mv composer.phar /usr/local/bin/composer
```

- Comprobar la instalación:

```
$ composer
```

- En la carpeta raíz de nuestro servidor (/Applications/XAMPP/htdocs) ejecutamos:

```
$ composer create-project --prefer-dist laravel/laravel miweb
```

- Esto nos creará la carpeta “miweb” con todo el contenido de la librería Laravel preparado.
- Para comprobar que todo funcione correctamente entramos en la carpeta y ejecutamos:

```
$ php artisan
```

- Es posible que en Mac nos aparezca el siguiente error:
“Mcrypt PHP extension required.”

- La configuración del proyecto se encuentra en la carpeta “`config`”.
- Para crear distintos **entornos** de configuración se usa el sistema “**DotEnv**” de variables de entorno guardadas en el fichero “`.env`”.
- Si no estuviera asignada la clave de encriptación la generamos:

```
$ php artisan key:generate
```

- Damos permisos de escritura a la carpetas de cache:

```
$ sudo chmod -R 777 storage  
$ sudo chmod -R 777 bootstrap/cache
```

- Y accedemos a la siguiente dirección web para comprobar que todo funcione bien:

<http://localhost/miweb/public/>

- **app/** → Carpeta principal de la aplicación (controladores, ...).
- **config/** → Configuración de toda la aplicación.
- **database/** → Configuración /inicialización de la BD.
- **public/** → Carpeta pública del sitio (con los assets).
- **resources/** → Recursos de la aplicación...
 - **lang/** → Traducciones.
 - **views/** → Vistas.
 - **js/ sass/** → Recursos JavaScript y CSS.
- **routes/** → Rutas de la aplicación.
- **storate/** → Caché y temporales
- **vendor/** → librerías y dependencias del framework
- **.env** → Fichero con la configuración de entorno.
- **artisan** → CLI de Laravel
- **composer.json** → Configuración de Composer

- La carpeta “**app**” contiene el código principal del proyecto: controladores, filtros y modelos de datos:

app/

Http/Controllers/

→ Controladores.

Http/Middleware/

→ Filtros para validar las peticiones.

Http/Kernel.php

→ Configuración de *middleware*.

- Los modelos de datos se guardan directamente dentro de “app”. Por defecto se incluye el modelo “User.php”.
- Las rutas de la aplicación se indican en los ficheros:

routes/

api.php

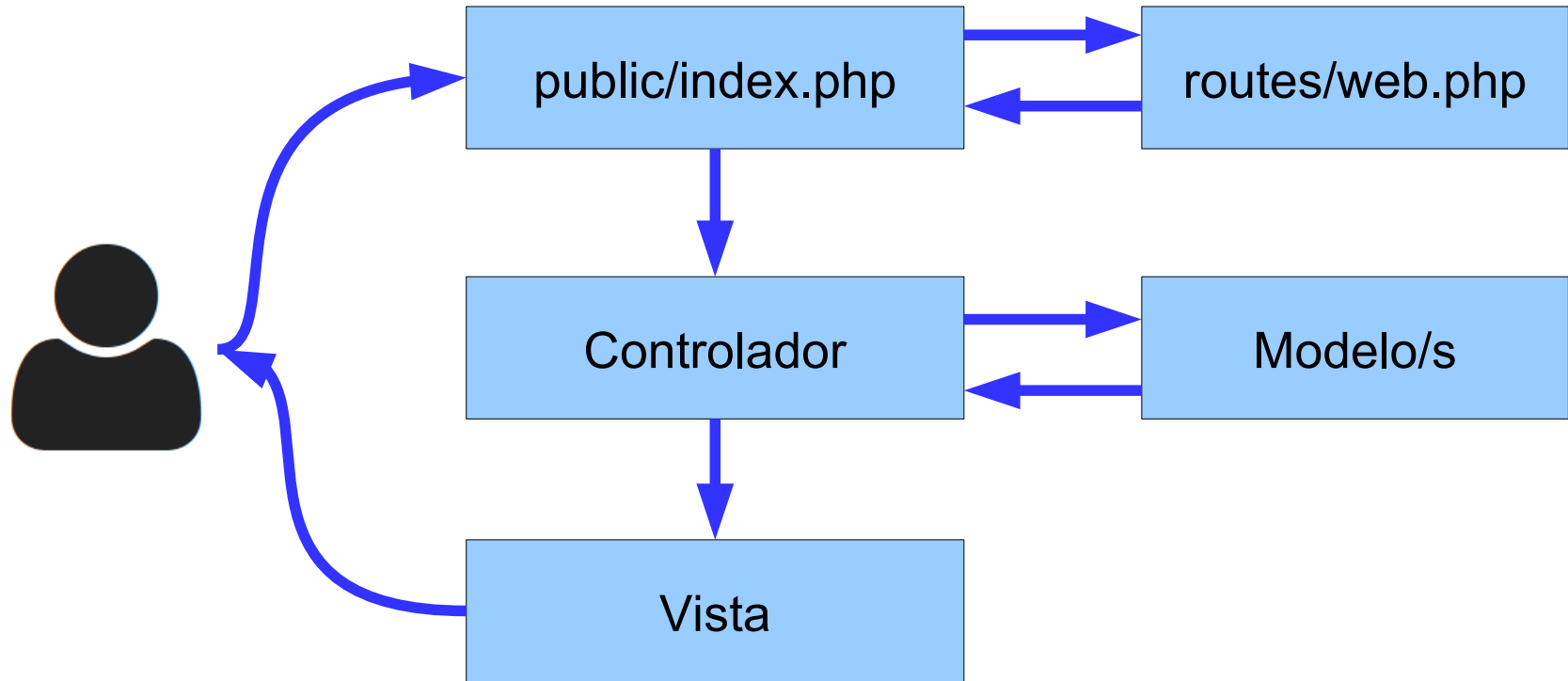
→ Rutas para API.

console.php

→ Rutas para consola.

web.php

→ Rutas para acceso Web.



- Todas las rutas están centralizadas en la carpeta “`routes`”.
- Cualquier ruta no definida → Error 404.
- Para cada ruta se define:
 1. Método HTTP de la petición: GET, POST, PUT, DELETE.
 2. Ruta URL (sin el dominio, el cual está ya definido en la configuración: `config/app.php`).
 3. Respuesta de una petición a esa ruta.
- Podremos devolver tres tipos de respuestas:
 1. Un valor (por ejemplo un número o una cadena).
 2. Una vista.
 3. Enlazar con un método de un controlador.

- Ejemplo de ruta para peticiones tipo GET a la URL raíz de nuestra aplicación web:

```
Route::get('/', function()           // URL raíz de la app
{
    return '¡Hola mundo!';
});
```

- Ejemplo de respuesta a peticiones tipo POST para la ruta “foo/bar”:

```
Route::post('foo/bar', function()
{
    return '¡Hola mundo!';
});
```

- De la misma forma podemos definir rutas para PUT o DELETE.

- También podemos definir una ruta que responda a varios tipos de peticiones:

```
Route::match(array('GET', 'POST'), '/', function()  
{  
    return '¡Hola mundo!';  
});
```

- O que responda a todas:

```
Route::any('foo', function()  
{  
    return '¡Hola mundo!';  
});
```

- Para añadir parámetros a las rutas se indican entre llaves “{ }”:

```
Route::get('user/{id}', function($id) {  
    return 'User ' . $id;  
});
```

- En este caso el parámetro sería **obligatorio**.
- Si queremos indicar que un parámetro es **opcional** tenemos que añadir el símbolo “?”:

```
Route::get('user/{name?}', function($name = null) {  
    return $name;  
});
```

- Para generar un enlace a una ruta usamos el método:

```
$url = url('foo');
```


- Es el interfaz de línea de comandos (CLI) de Laravel.
- Permitir realizar múltiples tareas necesarias durante el proceso de desarrollo o despliegue de una aplicación.
- Para ver una lista de todas las opciones que podemos utilizar con Artisan ejecutamos el siguiente comando:

```
$ php artisan  
# O también:  
$ php artisan list
```

- Para ver más ayuda de una opción ejecutamos:

```
$ php artisan help migrate
```

- Para ver un listado con todas las rutas definidas en nuestra aplicación ejecutamos:

```
$ php artisan route:list
```

- El cual nos mostrará el resultado en una tabla de la forma:

Domain	URI	Name	Action	Before Filters	After Filters
	GET HEAD /		UserController@getHome		
	GET HEAD login		UserController@login		
	GET HEAD logout		UserController@logout		
	GET HEAD catalog		CatalogController@index		
	GET HEAD catalog/show/{id}		CatalogController@show		
	GET HEAD catalog/create		CatalogController@create		
	GET HEAD catalog/edit/{id}		CatalogController@edit		

- En la nueva versión de Laravel se ha incorporado la generación de código desde Artisan.
- A través de la opción “make” podremos generar controladores, modelos, *middleware*, etc.
- Por ejemplo:

```
$ php artisan make:controller TaskController
```

- Presentan el resultado de forma visual al usuario, el cual podrá interactuar con él y volver a realizar una petición.
- Permiten separar toda la parte de presentación de resultados de la lógica (controladores) y de la base de datos (modelos).
- No realizan ningún tipo de consulta ni procesamiento de datos, simplemente reciben datos y los prepararán para mostrarlos.
- Se almacenan en la carpeta ``resources/views`` como ficheros PHP.
- Podrán contener:
 - Código HTML
 - Assets (CSS, imágenes, Javascript, etc. que estarán en la carpeta ``public``)
 - Algo de código PHP (o plantillas con *Blade*) para presentar los datos de entrada como un resultado HTML.

- Ejemplo de vista almacenada en “resources/views/home.php”:

```
<html>
  <head>
    <title>Mi Web</title>
  </head>
  <body>
    <h1>¡Hola <?php echo $name; ?>!</h1>
  </body>
</html>
```

- Para asociarla con una ruta, en el fichero “routes/web.php” añadimos:

```
Route::get('/', function()
{
    return view('home', array('name' => 'Javi'));
});
```

- Al construir una vista podemos pasarle parámetros de varias formas:

```
view('home', array('name' => 'Pedro', 'age' => 18));  
  
view('home')->with('name', 'Javi')  
->with('age', 18);
```

- Para hacer referencia a una vista que está en una subcarpeta:

```
"resources/views/user/profile" → "user.profile"
```

- Por ejemplo:

```
Route::get('user/profile/{id}', function($id) {  
    $user = // Cargar usuario a partir del $id  
    return view('user.profile', array('user' => $user));  
});
```

- Laravel utiliza *Blade* para la definición de plantillas en las vistas.
- Permite realizar todo tipo de operaciones con los datos: sustitución de variables o de secciones, herencia entre plantillas, definición de layouts, etc.
- Los ficheros de Blade tienen que tener la extensión “.blade.php”.
- Para hacer referencia a una vista que utiliza Blade **no** tenemos que indicar la extensión, por ejemplo:

`“home.blade.php” → “view('home');”`

- La notación de Blade empezará siempre por “@” o por “{{” mezclado con el código HTML de la vista.

- Comentarios con “`{{-- comentario --}}`”:

```
{{-- Este comentario no se mostrará en HTML --}}
```

- Para mostrar datos usamos “`{{ var / func }}`”:

```
Hola {{ $name }}.  
La hora actual es {{ time() }}.
```

- Si no queremos escapar los datos (CUIDADO) podemos poner:

```
Hola {!! $name !!}.
```


- Estructuras de control “if”:

```
@if( count($users) === 1 )  
    Solo hay un usuario!  
@elseif (count($users) > 1)  
    ¡Hay muchos usuarios!  
@else  
    No hay ningún usuario :(  
@endif
```

- Bucles:

```
@for ($i = 0; $i < 10; $i++)  
    El valor actual es {{ $i }}  
@endfor
```

```
@while (true)  
    <p>Soy un bucle while infinito!</p>  
@endwhile
```

```
@foreach ($users as $user)  
    <p>Usuario {{ $user->name }}  
        con identificador: {{ $user->id }}</p>  
@endforeach
```

- Incluir una plantilla dentro de otra plantilla:

```
@include('view_name')
```

```
{{-- También podemos pasarle un array de datos  
    como segundo parámetro: --}}
```

```
@include('view_name', array('some'=>'data'))
```

- Definición de un Layout (`resources/views/layouts/master.blade.php`)

```
<html>
  <head>
    <title>Mi Web</title>
  </head>
  <body>
    @section('menu')
      Contenido del menu
    @show

    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

- Posteriormente, en otra plantilla, podemos decir que extienda el layout que hemos creado:

```
@extends('layouts.master')

@section('menu')
    @parent
    <p>Este contenido es añadido al menú principal.</p>
@endsection

@section('content')
    <p>Este apartado aparecerá en
    la sección "content".</p>
@endsection
```

¿PREGUNTAS?