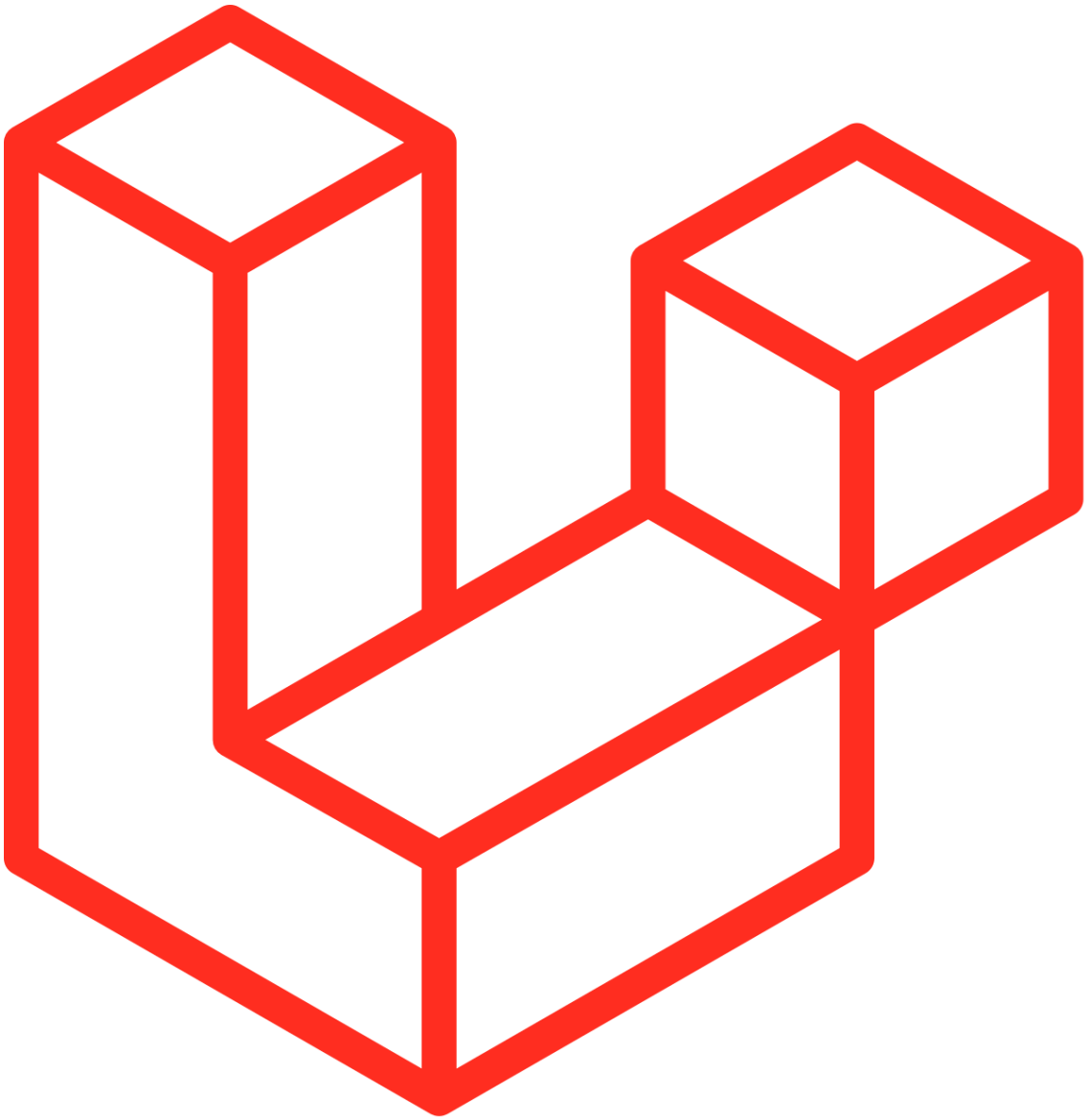
LARAVEL

Miguel A. M. Rasteu

2021



Contenido

1 – Introducción a Laravel

1.1 – Instalar Laravel

En primer lugar nos dirigimos a la documentación de Laravel (ver Documentación) para comprobar si tenemos todos los requerimientos necesarios para la instalación de Laravel.
Podemos comprobar la versión de PHP que tenemos usando el siguiente comando en la consola de comandos:

```
php -v
```

Ahora deberemos instalar composer, que es el programa encargado de manejar las dependencias de Laravel, ya que Laravel es solo el framework.

También deberemos comprobar si están activadas todas las extensiones de PHP que se nos indica en la documentación de Laravel.

Crear proyecto

Para crear un proyecto de Laravel con composer, deberemos dirigirnos a la carpeta donde se creará dicho proyecto, posteriormente introduciremos el siguiente comando en la consola.

```
$ composer create-project laravel/laravel nombre-proyecto
```

Composer se encargará de descargar las dependencias y de crear todas las carpetas del proyecto Laravel.

Crear un host virtual

Un host virtual es una simulación de una url de un dominio real. De esta forma podemos trabajar de una forma mucho más orgánica y simple.
A continuación veremos cómo crear un host virtual usando wampp

Paso 1: Entrar al fichero **C:\wamp\bin\apache\apache2.4.9\conf\httpd.conf** y añadir o descomentar el include del fichero de los hosts virtuales:

```
# Virtual hosts
Include conf/extra/httpd-vhosts.conf
```

Paso 2: Entrar al fichero **C:\wamp64\bin\apache\apache2.4.46\conf\extra\httpd-vhosts.conf** y añadir los virtualhosts.

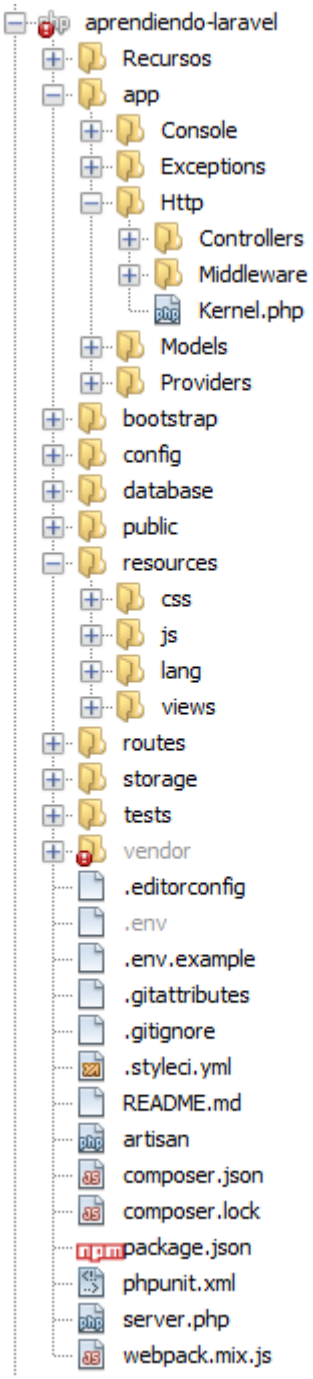
```
# Virtual Hosts
#
<VirtualHost *:80>
  ServerName localhost
  ServerAlias localhost
  DocumentRoot "${INSTALL_DIR}/www"
  <Directory "${INSTALL_DIR}/www/">
    Options +Indexes +Includes +FollowSymLinks +MultiViews
    AllowOverride All
    Require local
  </Directory>
</VirtualHost>

# VHOST CURSO LARAVEL
<VirtualHost *:80>
    DocumentRoot "${INSTALL_DIR}/www/master-php/aprendiendo-
laravel/public"
    ServerName aprendiendo-laravel.com.devel
    ServerAlias www.aprendiendo-laravel.com.devel
    <Directory "${INSTALL_DIR}/www/master-php/aprendiendo-laravel/public">
        Options Indexes FollowSymLinks
        AllowOverride All
        Order Deny,Allow
        Allow from all
    </Directory>
</VirtualHost>
```

Paso 3: Añadir al fichero hosts de nuestro sistema, en el caso de Windows **C:\Windows\System32\drivers\etc\hosts** (si estas en Windows 8 o 10 ejecuta el programa de edición de código como Administrador para poder guardar los cambios), y añadir las IP y las url.

```
127.0.0.1 localhost::1 localhost
127.0.0.1 localhost
127.0.0.1 aprendiendo-laravel.com.devel
```

Con estos tres pasos tendríamos creado y configurado nuestro propio host virtual.

Estructura de un proyecto de Laravel



El directorio **app** es donde está el contenido principal de la aplicación y contiene el código principal tanto de Laravel como de la aplicación que nosotros vamos a construir. Dentro de este directorio tenemos el directorio **Http** donde tenemos los **Controllers** que iremos creando nosotros y los **Models**, entre otros recursos.

También tenemos el directorio **bootstrap** que contiene un archivo que inicia el framework y una caché.

El directorio **config** contiene los archivos de configuración de la aplicación.

El directorio **database** lo que incluye son las migraciones de la base de datos y los seeds, las migraciones nos permiten versionar nuestros cambios en la base de datos y los seeds nos permite rellenar nuestra base de datos de forma programática.

El directorio **public** es donde se contienen los archivos públicos, es decir, assets, el archivo index.php que es por el cual entra la aplicación, se carga de forma automática y es donde se muestra la página web de forma automática y las **views** de nuestra aplicación.

El directorio **routes** contiene todas las rutas de nuestra aplicación que podamos tener.

En el directorio **storage** se usará para almacenar archivos cuando hagamos subidas, etc.

El directorio **test** para hacer test unitarios

EL directorio **vendor** es donde se guardan todos los paquetes, librerías y dependencias que instalamos con composer.

El resto de archivos que se ven son archivos de configuración.

2 – Routing

2.1 – Rutas básicas

Las rutas dentro de un framework son uno de los pilares más importantes porque es uno de los puntos que más nos facilita a la hora de trabajar con un framework es la posibilidad de configurar nuestras rutas y de configurar nuestras url, pasarles parámetros, tener rutas limpias y amigables, etc.

Si abrimos la carpeta de **routes**, en Laravel, es donde podemos configurar nuestras **routes**. Lo más común es tener nuestras **routes** en el archivo **web.php** pero eso puede variar según la funcionalidad que le demos a la ruta.

La **route** que tenemos por defecto en Laravel lo único que hace es devolvernos una **view**, nos devuelve la **view** Welcome. Pero igual que nos muestra una **view** podemos ponerle un echo e imprimir por pantalla un h1.

Es decir que lo que hace una ruta es ejecutar un bloque de instrucciones.

Podemos añadir todas las rutas con los métodos http que queramos. Los principales métodos http (que veremos en mayor profundidad más adelante) son los siguientes:

GET: Conseguir datos

POST: Guardar datos

PUT: Actualizar datos

DELETE: Borrar datos

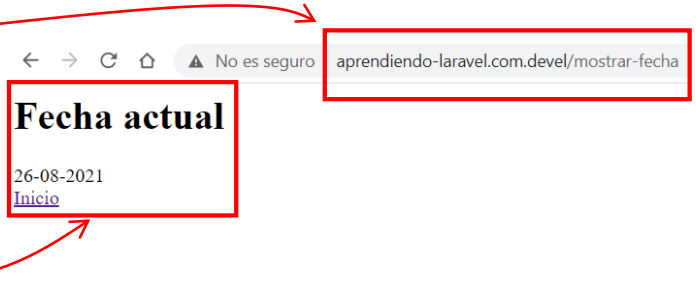
En base a esto crearemos nuestras rutas, además Laravel es un framework RESTFUL y permite crear rutas con el método http que queramos y testarlas con un cliente REST.

La sintaxis para definir una ruta es la siguiente:

```
Route::método-Http('/nombre-de-la-ruta', function(){
      Bloque de código
      a ejecutar
});
```

A continuación, podemos ver un ejemplo práctico y su resultado.

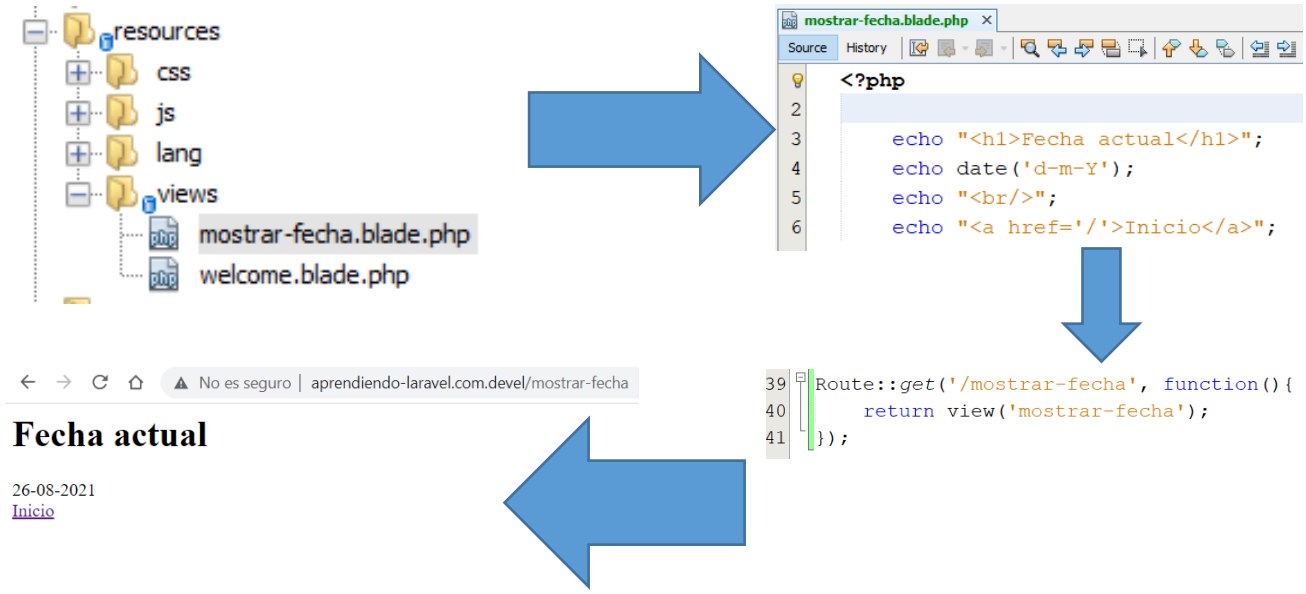
```
31  Route::get('/mostrar-fecha', function(){
32      echo "<h1>Fecha actual</h1>";
33      echo date('d-m-Y');
34      echo "<br/>";
35      echo "<a href='/'>Inicio</a>";
36  });
```



Podemos crearnos una vista en **resources>view** y podemos organizar las vistas en carpeta.
Creamos un archivo que se llame mostrar-fecha.blade.php con el mismo contenido que tenemos en los echo dentro del bloque de código de la **route** que hemos creado anteriormente.
Es importante que las **views** tengan el .blade porque las **views** de Laravel se hacen con ese motor de plantillas que veremos en profundidad más adelante.

Una vez creada la **view** deberemos cargar en la **route** la **view** que hemos creado.
En las **routes** solo debemos tener **views** y lógica, nunca debemos imprimir datos en las **routes**.

El resultado lo podemos ver a continuación:

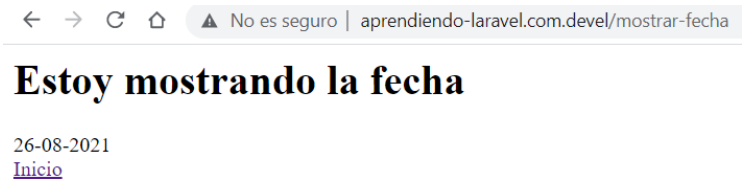


Incluso podríamos crearnos una variable en la **route**, por ejemplo $titulo, y pasarle esta variable como parámetro a la vista.
Para ello debemos pasarle un segundo parámetro en forma de array() y en una índice le pasamos el titulo con la variable $titulo.
Como vemos en la figura.

```php
Route::get('/mostrar-fecha', function(){
    $titulo = "Estoy mostrando la fecha ";
    return view('mostrar-fecha', array(
        'titulo' => $titulo
    ));
});
```

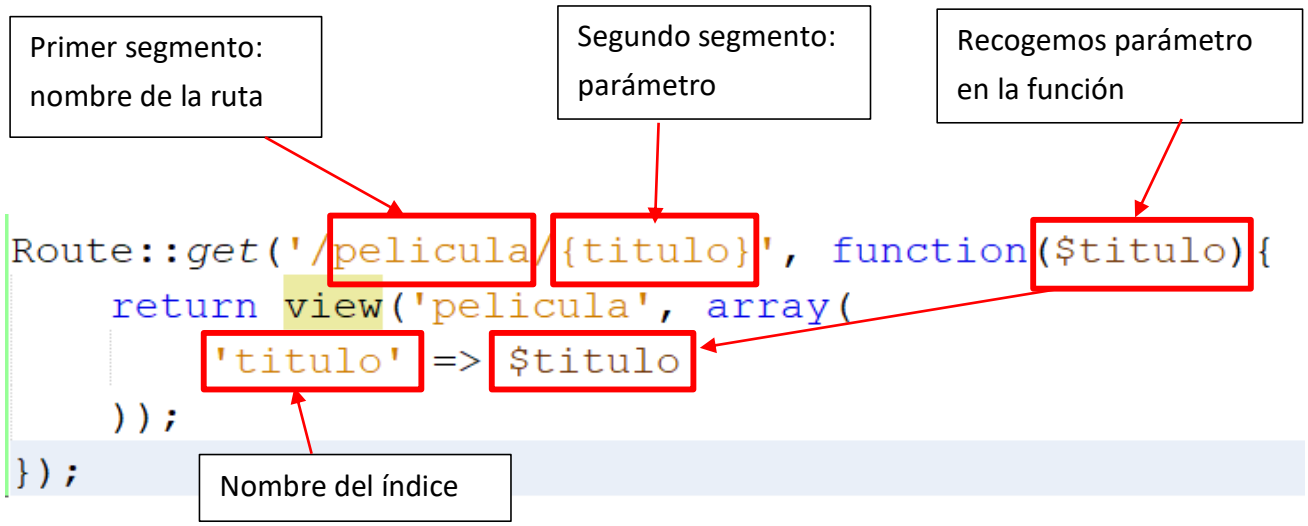
Y a su vez si en la **view** mostrar-fecha.blade.php cambio el contenido del primer h1 por una variable llamada $titulo nos mostrará en dicha **view** el contenido del array() con los parámetros que le hemos indicado en la **route**, como vemos en la figura:

```php
<?php

    echo "<h1>$titulo</h1>";
    echo date('d-m-Y');
    echo "<br/>";
    echo "<a href='/'>Inicio</a>";
```



2.2 – Parámetros en las rutas (routes)

Los parámetros se indican tras el nombre de la **route** y se encierra entre corchetes "**{ }**" . Si no indicamos nada en el parámetro es por defecto un parámetro obligatorio.
Posteriormente deberemos recoger ese parámetro en la función asociada a la **route**.



Si queremos que el parámetro sea opcional solo deberemos añadirle un símbolo "**?**" y en la función a la variable le asignaremos un valor por defecto.

```
Route::get('/pelicula/{titulo?}', function($titulo = "Empty"){
    return view('pelicula', array(
        'titulo' => $titulo
    ));
});
```

Otro punto interesante de las **routes** de Laravel es que me permiten hacer condiciones para permitir o denegar el acceso a la mismas.
Para ello usaremos un where con un array indicándolo en el valor del índice con expresiones regulares, como vemos en la siguiente imagen.

```
Route::get('/pelicula/{titulo}', function($titulo = "Empty"){
    return view('pelicula', array(
        'titulo' => $titulo
    ));
})->where(array(
    'titulo' => '[a-zA-z]+'
));
```

La condición dentro del where indica que el titulo debe contener únicamente texto en minúscula o en mayúsculas.

2.3 – Consola Artisan

Desde la consola de comandos tenemos acceso al siguiente comando:

```
$ php artisan
```

Este nos permite acceder a los comandos de Laravel.

Ahora listaremos todas las **routes** que tenemos en el proyecto donde nos encontramos con el siguiente comando:

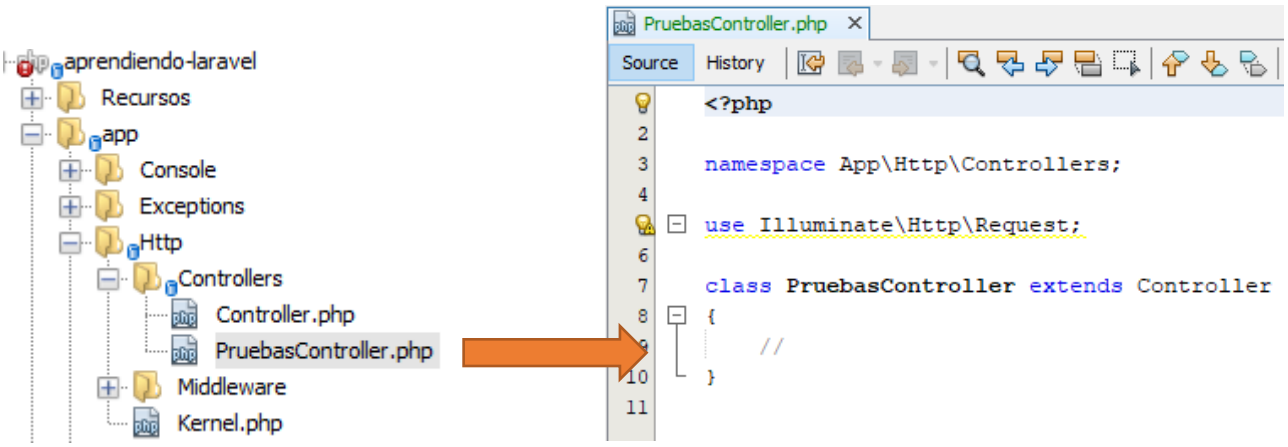
```
$ php artisan route:list
```

También podemos crear controladores o **Controllers**, que veremos en mayor profundidad más adelante, con el siguiente comando:

```
$ php artisan make:controller  NombreController
```

Este comando nos genera un controlador en el proyecto con las clases correctas y los espacios de nombres correctos, como podemos ver en la siguiente imagen.

```
mmras@DESKTOP-UG00QBB /cygdrive/c/wamp64/www/master-php/aprendiendo-laravel
$ php artisan make:controller PruebasController
Controller created successfully.
```



Para ver los comandos más comunes podemos usar

```
$ php artisan help
```

O si queremos listar todos los comandos usaremos

```
$ php artisan list
```

3 – Plantillas y Vistas

3.1 – Vistas en Laravel

Vamos a crear una nueva **route** llamada listado-peliculas donde tendremos una nueva **view** llamada listado. Dentro del directorio **view** vamos a crear una nueva carpeta llamada **películas** y dentro de esta tendremos la **view** de listado.blade.php y todas las **views** relacionadas con películas.

Esta **view** al estar dentro de una carpeta deberemos indicarlo en la **route** de una forma especial, la sintaxis para ello sería la siguiente:

NombreCarpeta.NombreView

Si no queremos pasar los parámetros mediante un array disponemos de un método en la función **view** para indicarlo. El método es **with** que nos permite adjuntar variables nuevas, podemos llamarlo tantas veces como necesitemos.

```
Route::get('/listado-peliculas', function(){

    $titulo = "Listado de peliculas";
    $listado = array('Batman', 'Superman', 'Gran Torino');

    return view('peliculas.listado')
            ->with('titulo', $titulo)
            ->with('listado', $listado);
});
```

3.2 – Sistemas de plantillas Blade

Laravel utiliza Blade para trabajar con las **views** y las plantillas. Entre las ventajas de Blade es que nos permite la herencia de plantillas.

Imprimir por pantalla

Podemos mostrar el contenido de una variable con la **interpolación**. Para hacer esto usaremos las dobles llaves **{{$variable}}**
Como podemos ver a pesar de que Blade puede funcionar con PHP también tiene su propio lenguaje, que sería lo más correcto de usar en las **views.**

La intención de Blade es que las **views** tengan una sintaxis diferente a las **routes** o **controladores** para ayudar a la organización del proyecto.

Dentro de las dobles llaves también podemos poner funciones de php.

Para comentar en Blade usaremos la siguiente sintaxis

```
{{-- ESTO ES UN COMENTARIO --}}
```

En el caso de que queramos mostrar una variable en el caso de que exista, porque pudiera ser una variable opcional en el código podríamos usar la siguiente condición PHP para llevarlo a cabo.

```
<?php
      isset($variable) ? $variable : 'No hay variable'
¿>
```

Esto es una condición ternaria que mostrara la variable si existe o imprimirá el texto 'No hay variable' en el caso de que no exista. Pero también podríamos hacerlo según la sintaxis de Blade de una forma mucho más rápida.

```
{{ $director ?? 'No hay director' }}
```

Obteniendo el mismo resultado.

Condicionales

Blade tiene estructuras de control condicionales como otros lenguajes de programación ya que a fin de cuentas es solo una librería de php con su propia sintaxis para las vistas.
La sintaxis de los condicionales en Blade la podemos ver en el siguiente ejemplo:

```
@if($titulo)
    El titulo existe y es este: {{$titulo}}
@else
    El titulo no existe
@endif
```

Bucles

Ahora veremos algunos ejemplos de bucles en Blade.
Ejemplo bucle for:

```
@for($i = 0; $i <= 20; $i++)
      El numero es: {{$i}}
@endfor
```

Ejemplo bucle while

```
@while($contador < 50)
    @if($contador % 2 == 0)
        NUMERO PAR: {{$contador}} <br/>
    @endif

    <?php $contador++; ?>
@endwhile
```

Ejemplo bucle foreach

```
@foreach($listado as $pelicula)
    <p>{{$pelicula}}</p>
@endforeach
```

4 – Controladores

5 – Formularios

6 – Bases de datos

7 – Proyecto en Laravel

8 – Entidades y modelos

Documentación

Curso Udemy

https://www.udemy.com/course/master-en-php-sql-poo-mvc-laravel-symfony-4-wordpress

Documentación Laravel

https://laravel.com/docs/8.x/installation

Crear un Host Virtual

https://victorroblesweb.es/2016/03/26/crear-varios-hosts-virtuales-en-wampserver/

Documentación Blade

https://styde.net/laravel-6-doc-plantillas-blade/