PORTADA: APUNTES LARAVEL

Contenido

[1. ROUTING. 5](#_Toc53502722)

[1.1 RUTAS BASICAS. 5](#_Toc53502723)

[1.2 PASO DE VARIABLES A LA VISTA. 5](#_Toc53502724)

[1.3 RUTAS CON PARAMETROS. 5](#_Toc53502725)

[1.3.1 PARAMETROS OBLIGATORIOS. 5](#_Toc53502726)

[1.3.2 PARAMETROS OPCIONALES. 5](#_Toc53502727)

[1.4 VALIDACION DE PARAMETROS EN LAS RUTAS. 5](#_Toc53502728)

[1.5 RUTAS CON LLAMADA A CONTROLADOR. 5](#_Toc53502729)

[1.6 GRUPO DE RUTAS. 5](#_Toc53502730)

[1.7 ESTABLACER ALIAS A UNA RUTA. 6](#_Toc53502731)

[2. VISTAS. 6](#_Toc53502732)

[invocar una vista en Laravel 5 7](#_Toc53502733)

[Pasar datos a las vistas en Laravel 9](#_Toc53502734)

[2.1 MOSTRAR INFORMACION EN VISTAS. 9](#_Toc53502735)

[2.2 MOSTRAR CUANDO EXISTE 9](#_Toc53502736)

[2.3 INCLUDES. 9](#_Toc53502737)

[3.PLANTILLAS. 10](#_Toc53502738)

[3.1 PLANTILLA BASE. 10](#_Toc53502739)

[4.CONDICIONALES. 12](#_Toc53502740)

[4.1 IF. 12](#_Toc53502741)

[4.2 IF-ELSE. 12](#_Toc53502742)

[4.3 IF-ELSEIF-ELSE. 12](#_Toc53502743)

[5. BUCLES 13](#_Toc53502744)

[5.1 FOR. 13](#_Toc53502745)

[5.2 WHILE. 13](#_Toc53502746)

[5.3 FOREACH. 13](#_Toc53502747)

[6. CONTROLLERS. 13](#_Toc53502748)

[6.1 CONTROLADORES BASICOS. 13](#_Toc53502749)

[6.1.1 RUTA LLAMANDO CONTROLADOR. 13](#_Toc53502750)

[6.1.2 RUTA CON PARAMETROS LLAMANDO CONTROLADOR. 14](#_Toc53502751)

[6.2 CONTROLADORES RESOURCE 14](#_Toc53502752)

[6.3 GENERAR URL A UNA ACCION DEL CONTROLADOR. 15](#_Toc53502753)

[6.3.1 LLAMANDO A CONTROLADOR 15](#_Toc53502754)

[6.3.2 LLAMANDO A UNA RUTA 15](#_Toc53502755)

[6.4 REDIRECCIONES. 15](#_Toc53502756)

[6.4.1 REDIRECCION A METODO DEL CONTROLADOR. 15](#_Toc53502757)

[6.3.4 REDIRECCION A RUTA. 16](#_Toc53502758)

[6.5 MIDDLEWARES O FILTRO 16](#_Toc53502759)

[6.5.1 CREACION DE MIDDLEWARE. 16](#_Toc53502760)

[7. FORMULARIOS. 19](#_Toc53502761)

[8. RESPONSES. 20](#_Toc53502762)

[Ejemplo de respuesta usando una instancia de Response 20](#_Toc53502763)

[Enviar una vista mediante una instancia Response 22](#_Toc53502764)

[9. CONEXIÓN A BD. 23](#_Toc53502765)

[9.1 CONECTAR BD. 23](#_Toc53502766)

[9.2 MIGRACIONES. 24](#_Toc53502767)

[9.2.1 CON SCHEMA-BUILDER(Blueprint). 24](#_Toc53502768)

[9.2.2 MEDIANTE CONSULTAS SQL. 25](#_Toc53502769)

[9.3 SEEDERS. 27](#_Toc53502770)

[9.4 LISTAR DATOS. 28](#_Toc53502771)

[9.5 MOSTRAR UNA FILA. 29](#_Toc53502772)

[9.7 ORDER BY. 31](#_Toc53502773)

[9.8 INSERTAR REGISTROS 31](#_Toc53502774)

[9.8.1 FORMULARIO DE INSERCION. 31](#_Toc53502775)

[9.2.2 RUTAS . 31](#_Toc53502776)

[9.3.3 METODOS DEL CONTROLADOR . 32](#_Toc53502777)

[9.9 ELIMINAR REGISTROS. 33](#_Toc53502778)

[RUTA 33](#_Toc53502779)

[METODO 33](#_Toc53502780)

[VISTA 34](#_Toc53502781)

[9.10 EDITAR REGISTROS. 35](#_Toc53502782)

[10. SESIONES. 38](#_Toc53502783)

[10.1 Helper global session . 38](#_Toc53502784)

[instancia de Request 39](#_Toc53502785)

[10.3 Crear y recuperar variables de sesión con Laravel 39](#_Toc53502786)

[Almacenar una variable de sesión 39](#_Toc53502787)

[Recuperar una variable de sesión 39](#_Toc53502788)

[10.4 Objeto session. 40](#_Toc53502789)

[10.4.1 BORRAR VARIABLE DE SESION. 40](#_Toc53502790)

[10.4.2 Comprobar la existencia de una variable de sesión . 40](#_Toc53502791)

[10.4.5 Regenerar el identificador de sesión . 40](#_Toc53502792)

[10.4.6 Borrar los datos de sesión al hacer logout . 41](#_Toc53502793)

# 1. ROUTING.

## 1.1 RUTAS BASICAS.

***Route*::get**('/', ***function*** () {

**return** **view**('welcome');

});

## 1.2 PASO DE VARIABLES A LA VISTA.

## 1.3 RUTAS CON PARAMETROS.

### 1.3.1 PARAMETROS OBLIGATORIOS.

### 1.3.2 PARAMETROS OPCIONALES.

## 1.4 VALIDACION DE PARAMETROS EN LAS RUTAS.

## 1.5 RUTAS CON LLAMADA A CONTROLADOR.

***Route*::resource**('usuario', 'UsuarioController');

## 1.6 GRUPO DE RUTAS.

Supongamos que queremos definir una serie de rutas relacionadas con el producto frutas en concreto, es decir que contengan el prefijo frutas en su URL.

***Route*::group**(['prefix' **=>** 'frutas'], ***function*** () {

***Route*::get**('index','FrutasController@index');

***Route*::get**('detalle','FrutasController@ver');

});

## 1.7 ESTABLACER ALIAS A UNA RUTA.

Para asignar un alias o nombre a una ruta le vamos a pasar en un array el campo alias con “as”

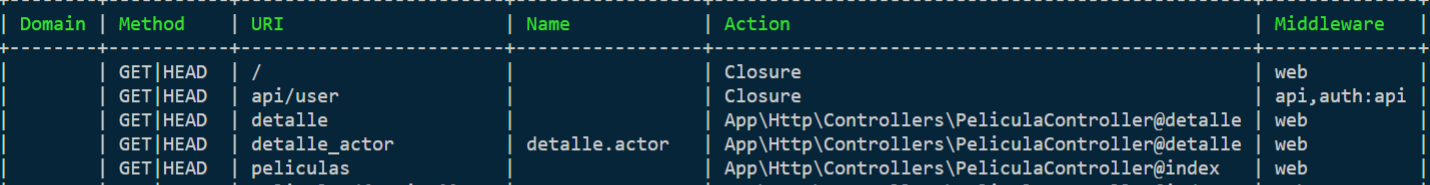
***Route*::get**('detalle\_actor',**array**(

    'uses'**=>**'PeliculaController@detalle',

    'as' **=>** 'detalle.actor'

    )

);



# 2. VISTAS.

Qué son las vistas

Como decíamos, la mayoría seguro entenderá este concepto, no obstante, vamos a exponerlo rápidamente para aquel que no sepa de qué estamos hablando. Las vistas son una de las capas que tiene el sistema MVC, que trata de la separación del código según sus responsabilidades. En este caso, las vistas mantienen el código de lo que sería la capa de presentación.

Como capa de presentación, las vistas se encargan de realizar la salida de la aplicación que generalmente en el caso de PHP será código HTML. Por tanto, una vista será un archivo PHP que contendrá mayoritariamente código HTML, que se enviará al navegador para que éste renderice la salida para el usuario.

**Nota:** En la práctica una vista podrá tener cualquier tipo de salida, no solo HTML. Hay ocasiones que será código PHP para generar una imagen, un archivo de texto o cualquier otra necesidad. Por ejemplo, si ante una solicitud el servidor debe enviar como respuesta datos en notación JSON, esos datos se escribirán mediante una vista.

Dónde almacenar las vistas en Laravel 5

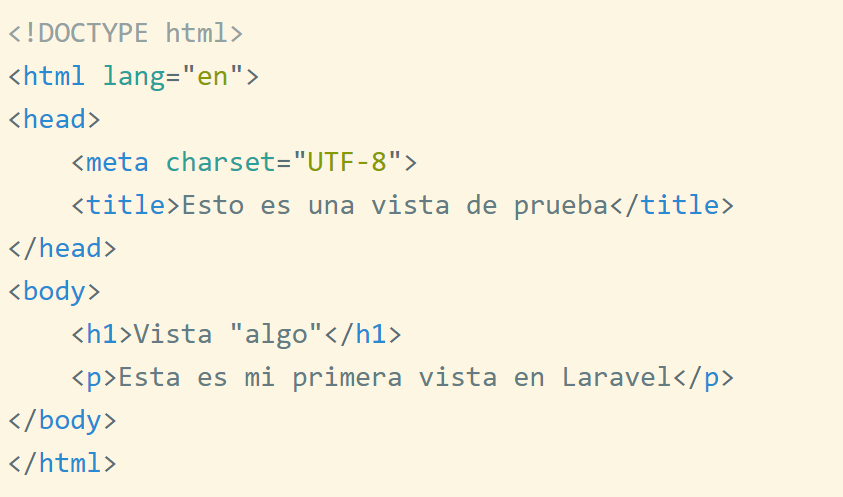
Existe una carpeta en el proyecto que es donde debemos colocar las vistas en Laravel. Está en "resources/views". Si navegas a esa carpeta observarás que dentro ya hay diversos archivos, incluso directorios. Esto es porque las vistas se pueden organizar por carpetas, para mantener agrupadas las de cada una de las secciones de la aplicación. Nosotros podemos hacer lo mismo, o dejarlas sueltas en el directorio view.

Además verás que las vistas tiene una extensión ".blade.php". Esto hace referencia a que es un archivo de salida que usa el motor de plantillas "Blade", el oficial de Laravel.

Ten en cuenta que, a pesar de la extensión ".blade.php", las vistas no dejan de ser archivos PHP donde podríamos colocar HTML plano mezclado con códigos PHP. De hecho, podríamos perfectamente nombrar a nuestras vistas como ".php" y el tratamiento que haremos para invocarlas será exactamente el mismo que si tienen la extensión ".blade.php". Es más, usar Blade es opcional, por lo que para mantener la simplicidad de nuestras primeras vistas no vamos a usar el motor de plantillas.

Así pues, para construir nuestra primera vista lo tenemos fácil. Simplemente creamos un archivo llamado "algo.php" en la carpeta "resources/views/". Dentro le colocamos cualquier documento HTML, con cualquier contenido. Con eso ya tenemos la vista lista para ser utilizada.

Como decíamos, el código de ese archivo "algo.php" de momento es indiferente, pero por si alguien necesita la aclaración sería algo como esto:



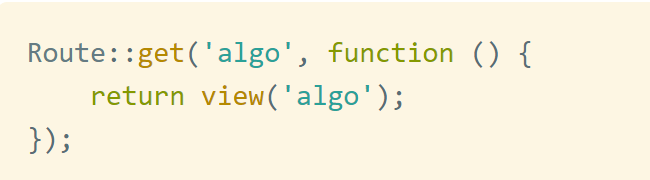
## invocar una vista en Laravel 5

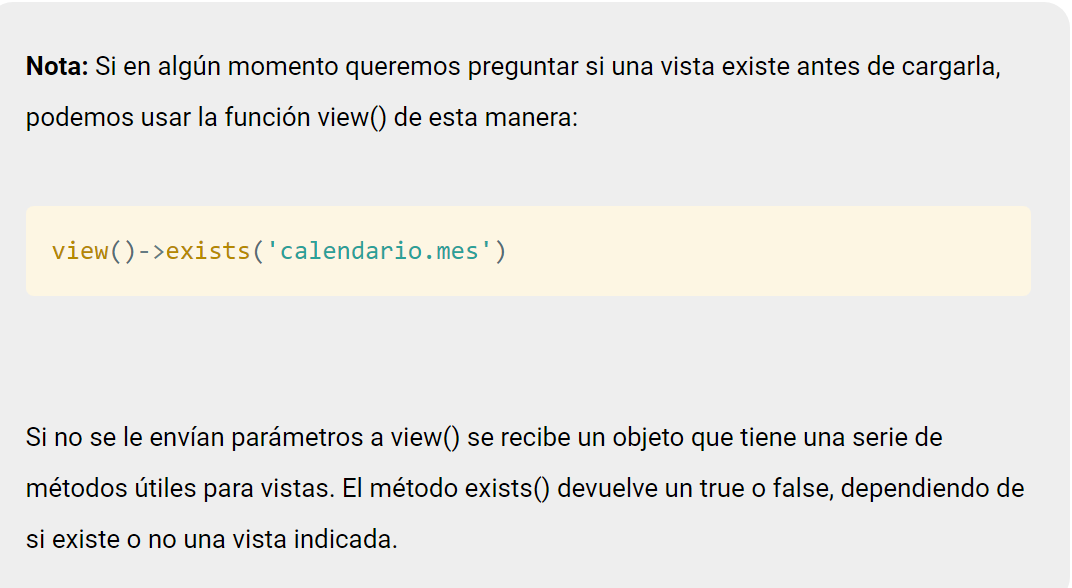
Ahora toca usar la vista creada en el paso anterior. Lo vamos a hacer desde el propio sistema de rutas, por simplificarnos la vida en este primer ejemplo. De hecho es como se invoca a la vista "welcome" en la ruta predeterminada que podremos ver al instalar el framework.

**Nota:** Aunque en una arquitectura MVC muchos desarrolladores prefieren que sean los controladores quienes invoquen a las vistas, nosotros nos vamos a saltar de momento ese paso porque realmente Laravel no lo necesita, pero sobre todo porque todavía no hemos llegado a explicar los controladores. En lugar de cargar la vista desde el controlador la vamos a invocar directamente desde el sistema de routing.

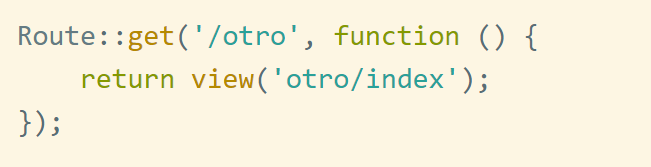
La corrección de esta operación sería discutible. No es lo habitual pero en ocasiones puede ser adecuada, por ejemplo que sea una vista que no necesita datos externos para mostrarse a si misma. Pero esto es otra discusión diferente al objetivo de este artículo.

En el siguiente código realizamos la invocación a una vista. Debes apreciar la función view() dentro del closure, a la que enviamos el nombre de la vista a mostrar.

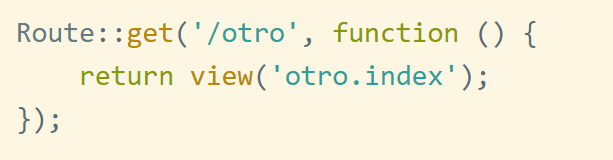




Ahora invocaremos esa vista indicando la ruta donde se encuentra, desde el directorio views. Omitimos de nuevo la extensión, ya sea ".php" o ".blade.php".

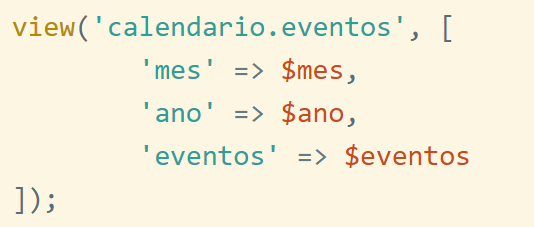


Como alternativa podemos especificar la ruta de la vista con un punto en lugar de una barra.

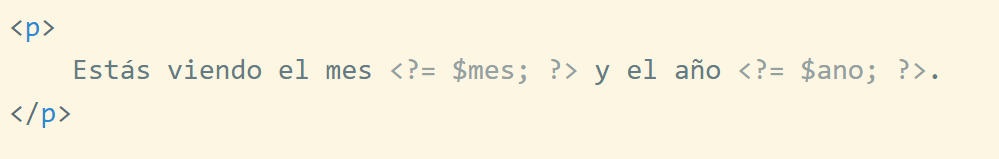


## Pasar datos a las vistas en Laravel

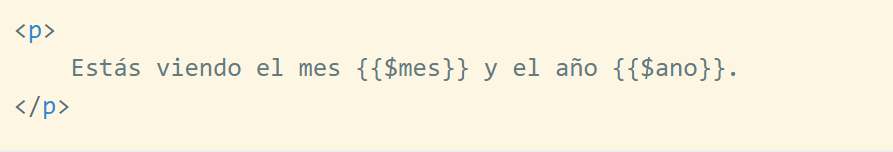
Seguro que lo siguiente que te preguntabas era cómo pasar los datos a las vistas. Si tienes que pasar datos para que las vistas los representen, los enviarás a través de la función global view() como un array asociativo.

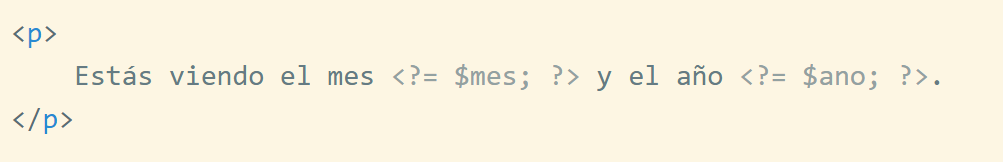


Una vez dentro de la vista los recoges en el ámbito global, a partir de las llaves de los elementos del array de datos. En el ejemplo anterior $mes, $ano o $eventos.



## 2.1 MOSTRAR INFORMACION EN VISTAS.



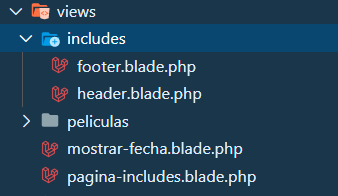


## 2.2 MOSTRAR CUANDO EXISTE

$value="**{{** ***$fruta*->*nombre*** **or** ''**}}**"

## 2.3 INCLUDES.

Directorios



RUTA

***Route*::get**('pagina', ***function*** () {

***$contenido*** **=** "Soy el contenido";

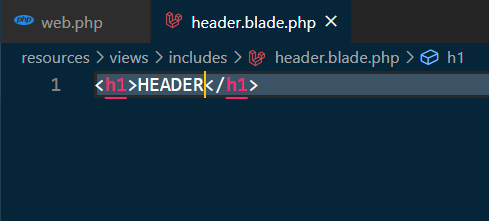
**return** **view**('pagina-includes', **array**(

        'contenido' **=>** ***$contenido***

    ));

});

Resources/views/Includes/header.blade.php



Pagina.blade.php

**@include**('includes.header')

<h1>pagina-includes</h1>

<div>

    <?=***$contenido***?>

</div>

**@include**('includes.footer')

# 3.PLANTILLAS.

## 3.1 PLANTILLA BASE.

Resources/views/layouts/master.blade.php

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="UTF-8">

        <meta name="viewport" content="width=device-width, initial-scale=1.0">

        <title>Document - **@yield**('titulo')</title>

    </head>

    <body>

**@section**('cabecera')

            CABECERA DE LA WEB

**@show**

        <div>

**@yield**('contenido')

        </div>

**@section**('pie')

            PIE DE LA WEB

**@show**

    </body>

</html>

Vista pagina-plantilla.blader.php

<?php

*// Cargamos plantilla maestra*?>

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

<?php

*// sustituimos el @yield que teniamos en el title de master.blade.php*?>

**@section**('titulo', 'Curso Laravel')

**@section**('cabecera')

**@parent**

        <h3>Hola soy informacion añadida a la @-section(cabecera) usando @-parent</h3>

**@stop**

<?php

*// sustituimos el @yield que teniamos en el contenido de master.blade.php*?>

**@section**('contenido')

    <p>

        Consectetur laborum elit culpa duis quis occaecat. Non ullamco velit cillum eu sint id. Exercitation mollit aliqua sunt voluptate ea aute magna velit.

        Fugiat Lorem sit sint officia dolore aliqua culpa velit veniam fugiat in minim. Consequat Lorem est dolor laboris qui amet aliqua aliquip est officia officia. Nisi culpa nisi et occaecat ullamco nostrud officia deserunt mollit. Mollit in ad sint dolore ad aute laborum ipsum.

    </p>

**@stop**

**@section**('pie')

**@parent**

        <h3>Hola soy informacion añadida a la @-section(pie) usando @-parent</h3>

**@stop**

# 4.CONDICIONALES.

## 4.1 IF.

**@if**(**isset**(***$fruta***))

    <h1>Detalle de la fruta con id: **{{*$fruta*->*id*}}**</h1>

**@endif**

## 4.2 IF-ELSE.

**@if**(**isset**(***$fruta***))

    <h1>Detalle de la fruta con id: **{{*$fruta*->*id*}}**</h1>

    <h2>Nombre: **{{*$fruta*->*nombre*}}**</h2>

    <h4>Descripcion: **{{*$fruta*->*descripcion*}}**</h4>

**@else**

    <h2> No existe la fruta seleccionada</h2>

**@endif**

## 4.3 IF-ELSEIF-ELSE.

# 5. BUCLES

## 5.1 FOR.

## 5.2 WHILE.

## 5.3 FOREACH.

<h1>Mi listado de frutas:</h1>

<ul>

**@foreach**(***$frutas*** **as** ***$fruta***)

        <li> Nombre:**{{*$fruta*->*nombre*}}** | **{{*$fruta*->*precio*}}**</li>

**@endforeach**

</ul>

# 6. CONTROLLERS.

## 6.1 CONTROLADORES BASICOS.

Crear controlador mediante cmd:



### 6.1.1 RUTA LLAMANDO CONTROLADOR.

Ruta:

***Route*::get**('/peliculas', 'PeliculaController@index' );

Controller:

<?php

**namespace** **App\Http\Controllers**;

**use** Illuminate\Http\***Request***;

***class*** **PeliculaController** extends ***Controller***

{

*// Implementamos nuestro metodos*

    public ***function*** **index**() {

***$titulo*** **=** "listado de peliculas";

**return** **view**('pelicula.index', **array**(

            'titulo' **=>** ***$titulo***

        ));

    }

}

### 6.1.2 RUTA CON PARAMETROS LLAMANDO CONTROLADOR.

Ruta:

***Route*::get**('/peliculas/{pagina?}', 'PeliculaController@index' );

Controller:

**namespace** **App\Http\Controllers**;

**use** Illuminate\Http\***Request***;

***class*** **PeliculaController** extends ***Controller***

{

*// Implementamos nuestro metodos*

    public ***function*** **index**(***$pagina*** **=** 1) {

***$titulo*** **=** "listado de peliculas";

**return** **view**('pelicula.index', **array**(

            'titulo' **=>** ***$titulo***,

            'pagina' **=>** ***$pagina***

        ));

    }

}

Vista:

<h1>**{{*$titulo*}}**</h1>

**@if**(**isset**(***$pagina***))

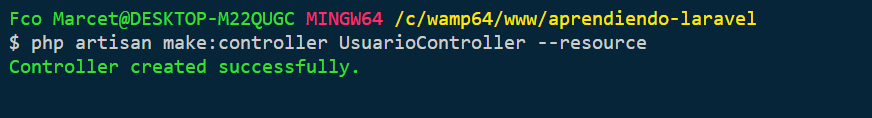
    <h3>Estamos en la página nº: **{{*$pagina*}}**</h3>

**@endif**

## 6.2 CONTROLADORES RESOURCE

Controlador con métodos CRUD ya predefinidos

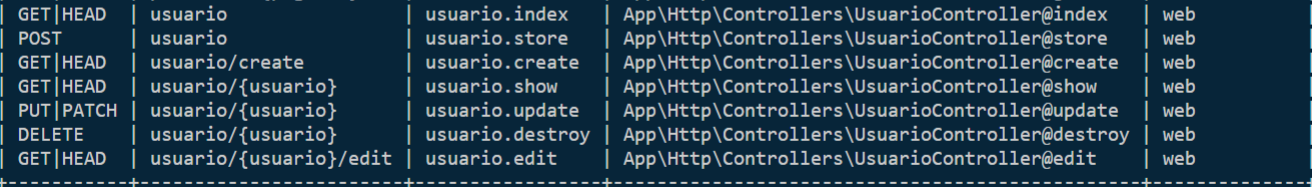
1- Creamos controlador tipo resource



2- Creamos ruta:

***Route*::resource**('usuario', 'UsuarioController');

Si ejecutamos php artisan route:list vemos la rutas que se han generado automáticamente:



## 6.3 GENERAR URL A UNA ACCION DEL CONTROLADOR.

Por ejemplo en una vista vamos a implementar 2 enlaces que nos lleven a inicio y a detalle

### 6.3.1 LLAMANDO A CONTROLADOR

Mediante action:

<a href="**{{** **action**('PeliculaController@index') **}}**" >Ir a inicio</a>

<a href="**{{** **action**('PeliculaController@detalle') **}}**" >Ir a detalle</a>

### 6.3.2 LLAMANDO A UNA RUTA

Para asignarlo a una ruta hemos de usar el **alias** de la ruta definida:

<a href="**{{** **route**('detalle.actor') **}}**" >Ir a detalle</a>

PASAR PARAMETROS

<a href="**{{** **route**('detalle.actor', ['id'**=>** 10]) **}}**" >Ir a detalle actor</a>

## 6.4 REDIRECCIONES.

Aparte de crear enlaces que no redirigan a determinadas acciones del controlador, podemos redireccionar desde una acción del controlador a otra o bien realizar una redirección desde una URL a otra, incluso sin que el usuario tenga que pulsarlo es decir de manera programatica.

### 6.4.1 REDIRECCION A METODO DEL CONTROLADOR.

Asi que vamos a crear un método para realizar esto:

 public ***function*** **redirigir**() {

**return** **redirect**()**->action**('PeliculaController@detalle');

    }

Si ahora creamos la ruta:

***Route*::get**('redirigir','PeliculaController@redirigir' );

Al acceder a la ruta directamente nos llevara al detalle

### 6.3.4 REDIRECCION A RUTA.

De manera análoga podemos redireccionar a una ruta

public ***function*** **redirigir**() {

**return** **redirect**('/peliculas');

    }

public ***function*** **redirigir**() {

**return** **redirect**()**->route**('detalle.actor');

    }

## 6.5 MIDDLEWARES O FILTRO

Nos permite filtrar las peticiones HTTP, es una clase de PHP que nos permite realizar lógica antes de mostrar una vista o recibir los datos en un controlador.

Por ejemplo deseamos validar que nos llegan a la página de detalle sin nos llega un determinado parámetro, validar que llega con cierto valor, para dejar al usuario ver o no la página.

### 6.5.1 CREACION DE MIDDLEWARE.

Partimos de esta ruta por ejemplo, la cual tiene un parámetro opcional year

***Route*::get**('detalle/{year?}',**array**(

    'uses'**=>**'PeliculaController@detalle',

    'as' **=>** 'detalle.actor'

    )

);

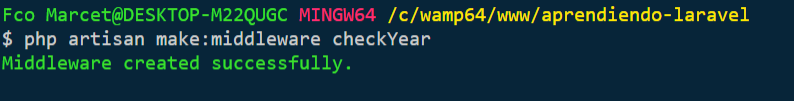
Recibimos este parámetro en el controlador:

public ***function*** **detalle**(***$year*** **=** null) {

**return** **view**('pelicula.detalle');

    }

La idea es mostrar la pagina cuando el año sea 2019 y si esta condición no se cumple que nos rediriga a otra, esto se consigue con un MIDDLEWARE, el cual generamos mediante comandos:



Si vamos al fichero donde se almacena los middlewares app/http/controllers/middleware:

Y este seria el código generado:

<?php

**namespace** **App\Http\Middleware**;

**use** ***Closure***;

***class*** **checkYear**

{

*/\*\**

*\* Handle an incoming request.*

*\****@param***\Illuminate\Http\****Request****$request*

*\****@param***\****Closure****$next*

*\****@return****mixed**

*\*/*

    public ***function*** **handle**(***$request***, ***Closure*** ***$next***)

    {

**return** $***next***(***$request***);

    }

}

En el cual podemos implementar la lógica de comprobar el año que recibimos, vendrá en la variable $request

<?php

**namespace** **App\Http\Middleware**;

**use** ***Closure***;

***class*** **CheckYear**

{

*/\*\**

*\* Handle an incoming request.*

*\****@param***\Illuminate\Http\****Request****$request*

*\****@param***\****Closure****$next*

*\****@return****mixed**

*\*/*

    public ***function*** **handle**(***$request***, ***Closure*** ***$next***)

    {

*//almacenamos en la variable year lo que recibimos que llega en el obj$request y accedemos al parametro que no llega por la ruta*

***$year*** **=** ***$request*->route**('year');

**if** (**is\_null**(***$year***) **||** ***$year*** **!=** 2019) {

*//hacemos redireccion*

**return** **redirect**('/peliculas');

        }

**return** $***next***(***$request***);

    }

}

Ahora necesitamos aplicarlo a nivel de rutas para que se ejecute, ***añadimos en la ruta otro índice en el array, hemos de añadirlo al fichero Kernel.php***

**AÑADIR AL ARRAY:**

***Route*::get**('detalle/{year?}',**array**(

    'middleware' **=>** 'checkyear',

    'uses'**=>**'PeliculaController@detalle',

    'as' **=>** 'detalle.actor'

    )

);

**AÑADIR EN FICHERO kernel.php :**

protected ***$routeMiddleware*** **=** [

 'auth' **=>** \App\Http\Middleware\***Authenticate*::class**,

 'auth.basic' **=>** \Illuminate\Auth\Middleware\***AuthenticateWithBasicAuth*::class**,

 'bindings' **=>** \Illuminate\Routing\Middleware\***SubstituteBindings*::class**,

 'cache.headers' **=>** \Illuminate\Http\Middleware\***SetCacheHeaders*::class**,

 'can' **=>** \Illuminate\Auth\Middleware\***Authorize*::class**,

 'guest' **=>** \App\Http\Middleware\***RedirectIfAuthenticated*::class**,

 'signed' **=>** \Illuminate\Routing\Middleware\***ValidateSignature*::class**,

 'throttle' **=>** \Illuminate\Routing\Middleware\***ThrottleRequests*::class**,

 'verified' **=>** \Illuminate\Auth\Middleware\***EnsureEmailIsVerified*::class**,

 'checkyear' **=>** \App\Http\Middleware\***CheckYear*::class**

    ];

# 7. FORMULARIOS.

Partimos del formulario situado en**: resources/views/película/formulario.blade.php**

**NOTA:** Imprescindible poner {{ csrf\_field() }}

<h1>Formulario en Laravel</h1>

<form action="**{{** **action**('PeliculaController@recibir') **}}**"

method="POST">

*<!-- llamamos a este metodo para proteger contra ataques -->*

**{{** **csrf\_field**() **}}**

    <label for="nombre">Nombre</label>

    <input type="text" name="nombre"></input></br>

    <label for="email">Email</label>

    <input type="text" name="email"></input></br>

    </br>

    <input type="submit" value="Enviar"/>

</form>

**RUTAS**

***Route*::get**('/formulario', 'PeliculaController@formulario');

***Route*::post**('/recibir', 'PeliculaController@recibir');

**METODOS**

public ***function*** **formulario**() {

**return** **view**('pelicula.formulario');

    }

public ***function*** **recibir**(***Request*** ***$request***) {

***$nombre*** **=** ***$request*->input**('nombre');

***$email*** **=** ***$request*->input**('email');

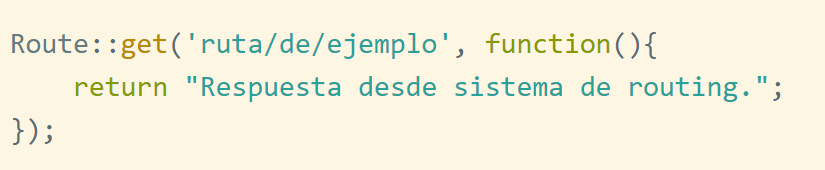
**return** "El nombre es: ***$nombre*** y el email es: ***$email***";

    }

# 8. RESPONSES.

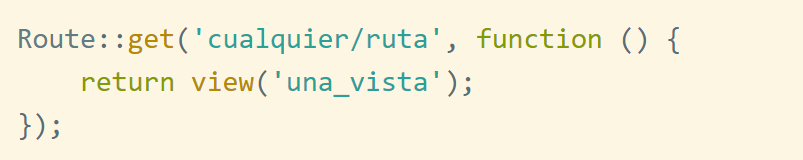
Las responses son las "respuestas" que nos debe devolver Laravel ante cualquier solicitud que se realice al servidor. Según la documentación oficial "Cualquier ruta [del sistema de routing] y controlador debe devolver algún tipo de respuesta al navegador del usuario". Existen diversos modos de devolver respuestas

El sistema de Response depende de la clase Illuminate\Http\Response aunque muchas veces nos saltamos realmente el uso de esa clase y simplemente le dejamos a Laravel que la use internamente. Por ejemplo cuando escribimos salida desde el sistema de routing:



En este caso, como decimos, internamente Laravel recibirá esa cadena de respuesta la convertirá en un HTTP response para enviarla al cliente.

Cuando devolvemos una vista, ya sea desde un controlador o un closure en el sistema de routing también se pone en marcha el sistema de response de Laravel sin que el programador necesite usarlo directamente.

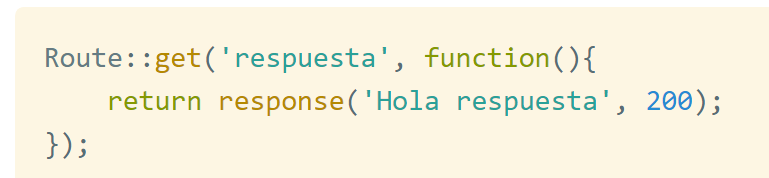


¿Entonces se puede abordar directamente el sistema de Response de Laravel y en ese caso, qué utilidad tiene? Efectivamente, nosotros podemos usar directamente una instancia de Response y ello nos permite personalizar todavía más elementos de la respuesta HTTP, enviando códigos de status particulares o cabeceras de HTTP.

## Ejemplo de respuesta usando una instancia de Response

A través del helper response() podemos crear fácilmente una instancia del objeto Response con el que podemos hacer varios ejemplos de uso de personalización de la respuesta.

En este primer ejemplo tenemos una respuesta exactamente igual a la que conseguiríamos con un simple return 'Hola respuesta'.



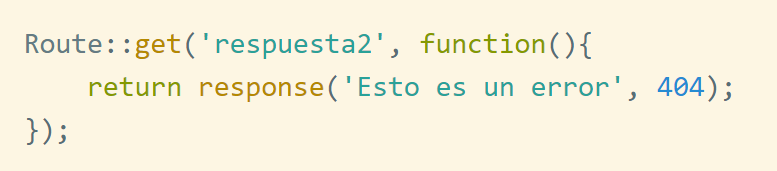
El helper response(), como puedes ver, recibe dos parámetros:

1. Contenido de la respuesta
2. Código de status

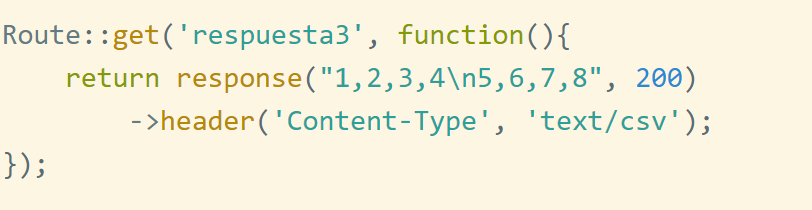
El contenido es el texto que devuelve como respuesta la solicitud HTTP. El código de status son los típicos que debes de conocer del protocolo HTTP. 200 significa "todo correcto".

Después de la invocación al helper response() recibimos como valor de devolución un objeto de la clase Response. Éste objeto es el que devuelve el closure y que Laravel toma para componer la respuesta HTTP.

En este segundo ejemplo devolvemos un error de página no encontrada, status 404, con contenido 'Esto es un error'.



En este ejemplo realizamos algo también simple pero más elaborado, ya que enviamos una respuesta a la que le agregamos una cabecera adicional para especificar que el contenido es una hoja de cálculo, archivo CSV.

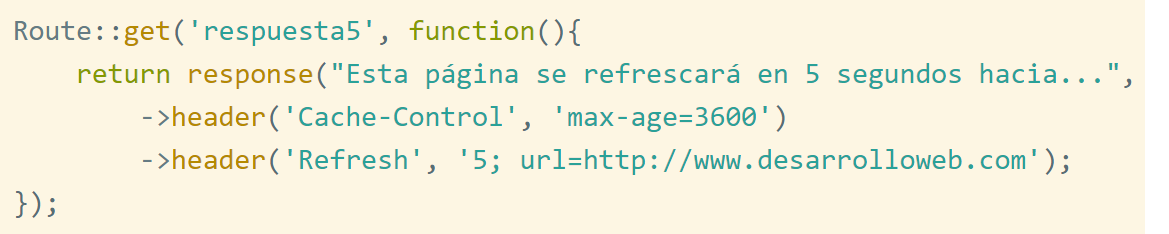


Como decíamos, el helper response() devuelve un objeto de la clase Response y sobre él estamos encadenando la invocación de su método header(), que sirve para agregar nuevas cabeceras a la respuesta HTTP. En este caso el nombre de la cabecera es 'Content-Type' y el valor que estamos aplicando es 'text/csv'.

En el ejemplo siguiente probamos otra cabecera diferente, para realizar una redirección. Tómalo simplemente como un test, puesto que existe otro método más rápido y sencillo para enviar responses de tipo redirect.



Podemos encadenar varias llamadas al método header() si quisiéramos agregar varias cabeceras diferentes a la respuesta. como vemos en el siguiente ejemplo:



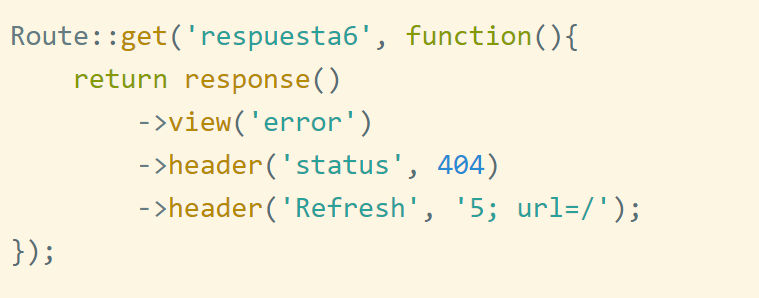
Hemos puesto dos cabeceras de HTTP response, la primera para definir la antigüedad máxima del elemento cacheado y la segunda para decir que en 5 segundos se refresque la página enviando al navegador a una nueva URL.

## Enviar una vista mediante una instancia Response

Habrás advertido que resulta especialmente incómodo enviar como respuesta una cadena en el primer parámetro del helper response(). Si la respuesta es muy larga para colocarla, así tal cual, en una cadena, o simplemente prefieres separar la salida en una vista, tal como has aprendido, tenemos una alternativa muy útil.

Se trata de invocar al helper de response() sin enviar ningún parámetro. La respuesta (objeto Response) estará limpia para configurar todos sus detalles. Uno de los métodos del objeto Response que te devuelve el helper response() es view(), en el que indicamos la vista que queremos procesar.

Luego podemos indicar nuevas cabeceras HTTP en la respuesta, encadenando métodos, tal como hemos visto en ejemplos anteriores



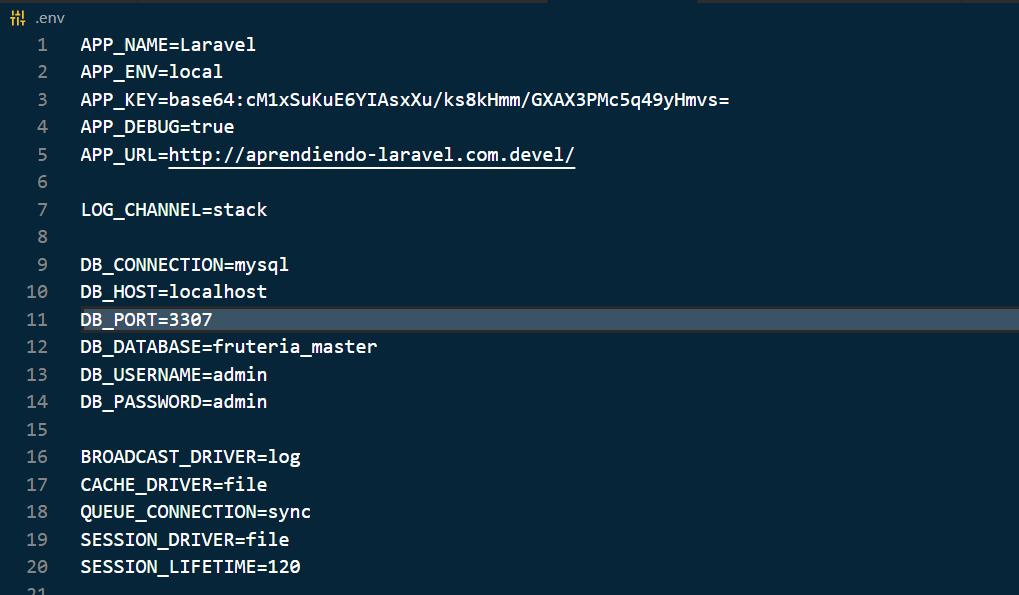
En el código anterior tendrías un ejemplo de response en la que cargamos una vista llamada "error" y luego enviamos dos cabeceras adicionales, una para mandar un código de status (404 de error "página no encontrada") y una redirección pasados 5 segundos a la home del dominio.

# 9. CONEXIÓN A BD.

## 9.1 CONECTAR BD.

Vamos a crear una BD para el testeo -> fruteria\_master

**CONEXIÓN MEDIANTE FICHERO .env**



**NOTA**

PUERTO (MariaDb): 3307

PUERTO (Mysql): 3306

Si no funciona DB\_HOST con 127.0.0.1 usar localhost

Al efectuar cambios en .env es conveniente usar:



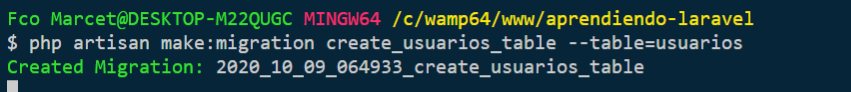
Ahora estas credenciales las utlizara el fichero config/database.php, podríamos directamente cambiar los valores del array de dicho fichero pero veo mas recomendable modificarlos en .env

## 9.2 MIGRACIONES.

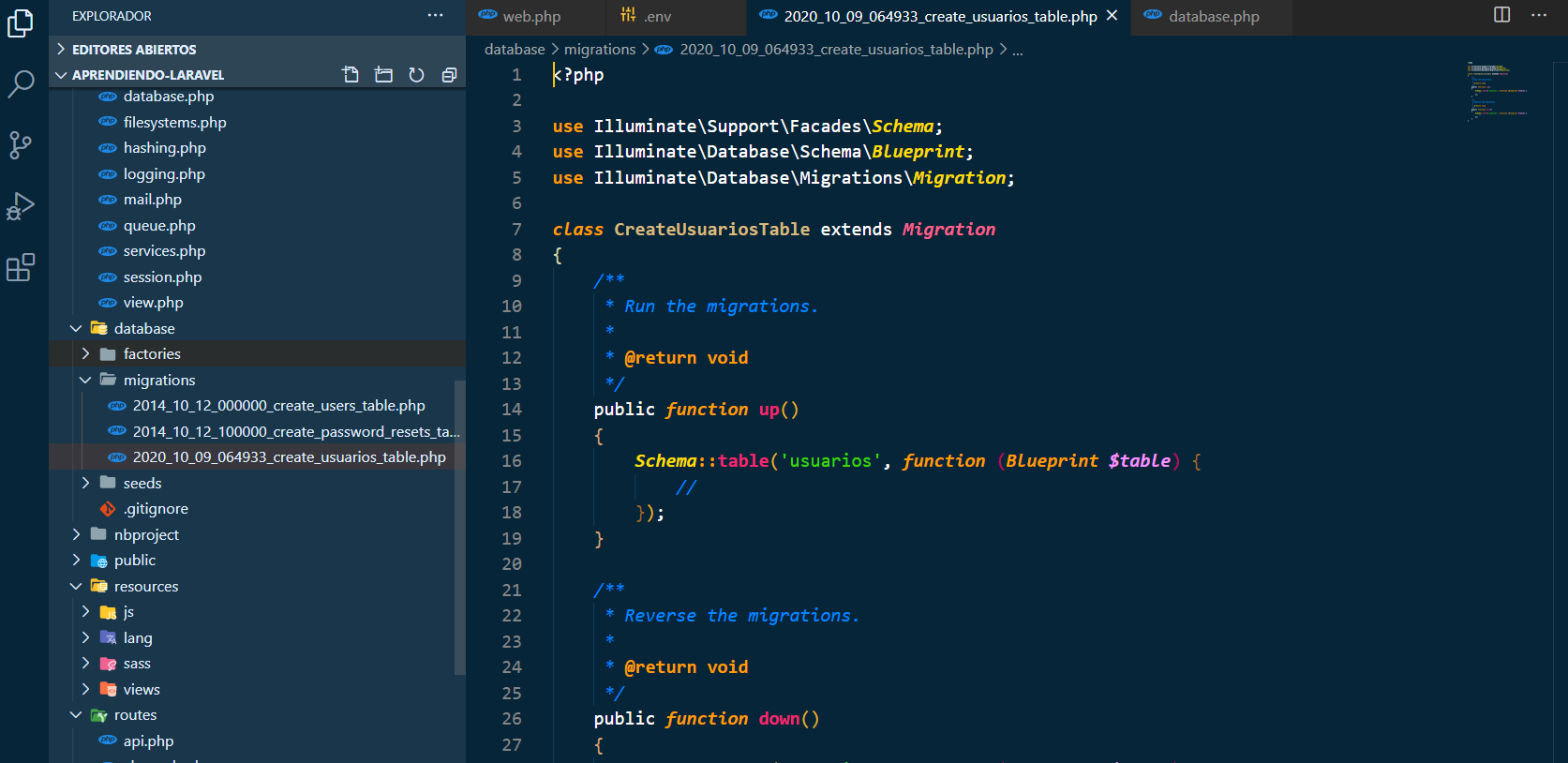
Las migraciones nos permiten versionar nuestra base de datos, no es estrictamente necesario pero si es una buena practica.



Si además queremos que esta migración nos genere la estructura de la tabla con schemaBuilder de laravel con sus métodos, utilizamos --table=NombreTabla



Si nos fijamos en el directorio database/migrations tendremos las migraciones creadas:



### 9.2.1 CON SCHEMA-BUILDER(Blueprint).

Donde podemos observar el método up() utilizado para levantar la migración y down() para echarla atrás, es decir para crear la tabla y para borrar la tabla.

public ***function*** **up**()

    {

***Schema*::create**('usuarios', ***function*** (***Blueprint*** ***$table***) {

*// implementamos la tabla con el schema-Builder*

***$table*->increments**('id'); *//PK auto\_increment*

***$table*->string**('usuario', 255);

***$table*->string**('email', 255);

***$table*->string**('password', 255);

***$table*->integer**('edad');

***$table*->timestamps**(); *//created\_at, updated\_at*

        });

    }

Eliminar tabla:

 public ***function*** **down**()

    {

***Schema*::drop**('usuarios');;

    }

### 9.2.2 MEDIANTE CONSULTAS SQL.

Utilizamos DB::statement(“SENTENCIA\_SQL”)

public ***function*** **up**()

    {

***DB*::statement**("

        CREATE TABLE usuarios

        (

            id INT(255) auto\_increment NOT NULL,

            nombre VARCHAR(255) NOT NULL,

            email VARCHAR(255) NOT NULL,

            password VARCHAR(255) NOT NULL,

            PRIMARY KEY(id)

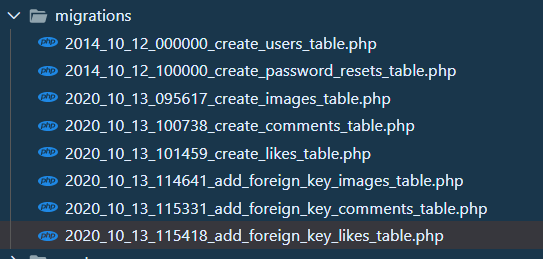
        );

        ");

    }

**NOTA**

Me parece mas sencillo crear un fichero de migración para las tablas y otro para añadir las FK

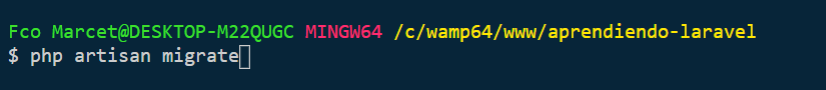


Y ejecutamos el comando:

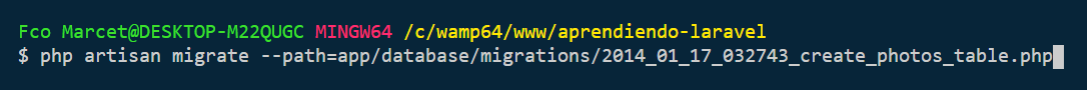


**LANZAR MIGRACIONES(TODAS)**:

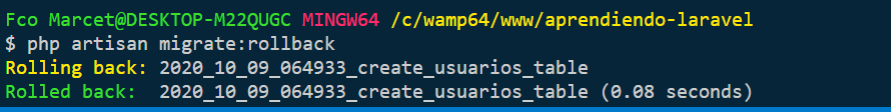
Con este cmd nos lanza TODAS las migraciones que tengamos



**LANZAR MIGRACION FICHERO ESPECIFICO:**



**REVERTIR MIGRACIONES**:



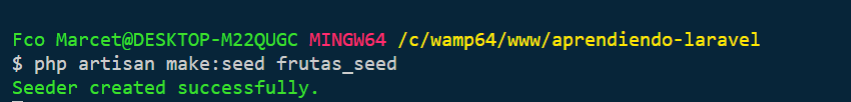
**MODIFICAR REGISTROS Y LANZAR MIGRACION CON NUEVOS CAMBIOS.**

Imaginemos que modificamos una columna de una tabla

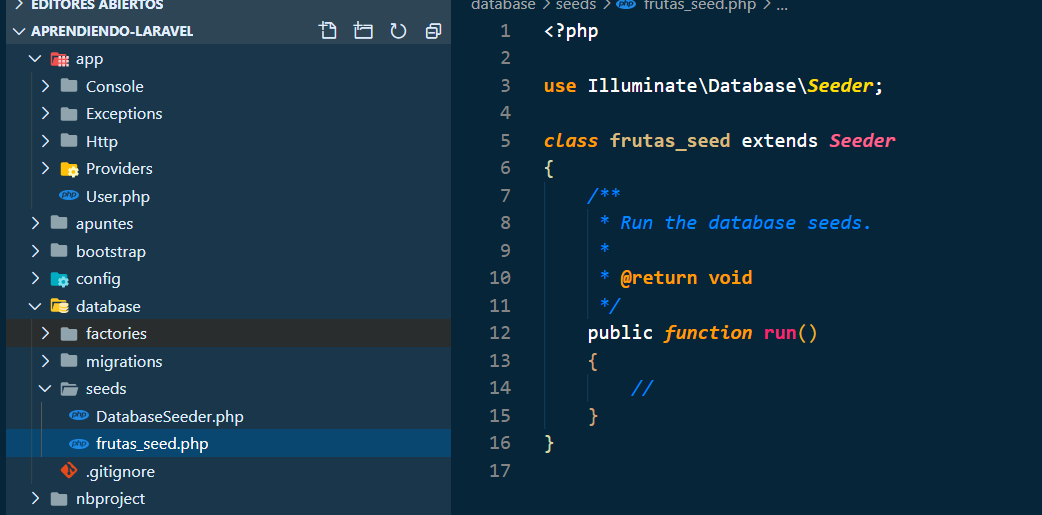


## 9.3 SEEDERS.

Su utilidad es para rellenar inicialmente campos en las tablas para tener unos datos de testeo para la API



Nos crea el fichero:



Vamos a implementar un script para que nos rellene de datos las tablas con 20 frutas por ejemplo

 public ***function*** **run**()

  {

**for** (***$i*=**0; ***$i*** **<** 20; ***$i*++**) {

***DB*::table**('frutas')**->insert**(**array**(

                'nombre' **=>** 'Fruta'**.*$i***,

                'descripcion' **=>** 'Descripcion de la fruta'**.*$i***,

                'precio' **=>** 1 **\*** ***$i***,

                'fecha\_reg' **=>**  **date**("Y-m-d H:i:s")

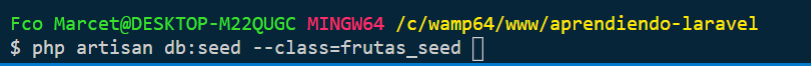
            ));

        }

    $this**->*command*->info**('registros insert en la tabla frutas');

  }

Ejecutamos el método para insertar los registros



## 9.4 LISTAR DATOS.

Vamos a utilizar el método estatico DB::table(), que nos devuleve un conjunto de filas de la tabla indicada:

***$frutas*** **=** ***DB*::table**('NombreTabla')**->get**();

NOTA :

get() es similar a un SELECT \* FROM NombreTabla;

Añadimos en un array la variable devuelta,

**return** **view**('fruta.index' , **array**(

            'frutas' **=>** ***$frutas***

        ));

Vamos a implementar también un grupo de rutas :

***Route*::group**(['prefix' **=>** 'frutas'], ***function*** () {

***Route*::get**('index','FrutasController@index');

});

Y por ultimo ya podemos recorrer el array ($frutas) que nos devuelve el controlador en la vista:

<h1>Mi listado de frutas:</h1>

<ul>

**@foreach**(***$frutas*** **as** ***$fruta***)

        <li>Nombre:**{{*$fruta*->*nombre*}}**|**{{*$fruta*->*precio*}}**</li>

**@endforeach**

</ul>

## 9.5 MOSTRAR UNA FILA.

Para mostrar un solo registro, es muy parecido a obtener todos los registros pero utilizando **->where()**, este recibe 3 parametros

1. Campo de la tabla
2. Operador (=, <, >, <=, >= …)
3. $id, que recibe el método

 public ***function*** **detail**(***$id***)

  {

***$fruit*** **=** ***DB*::table**('frutas')**->where**('id', '=', ***$id***)**->get**();

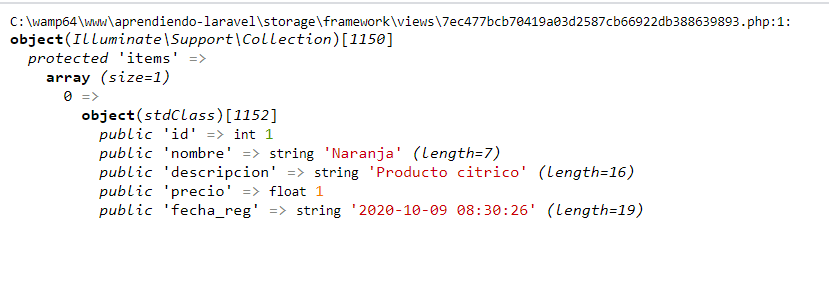
**return** **view**('fruta.detalle', **array**(

                'fruta' **=>** ***$fruit***

        ));

  }

Como vemos si hacemos un var\_dump($fruta), nos devuelve :



Pero es mas útil utilizar ->***first()***, que nos devuelve directamente un objeto en vez de un array cuyo primer índice es un objeto

public ***function*** **detail**(***$id***)

  {

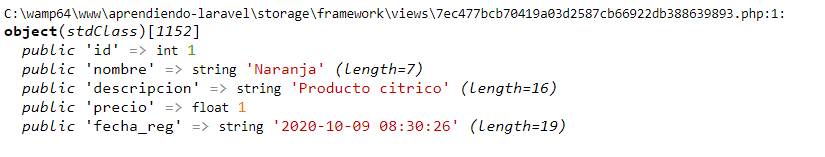
***$fruit*** **=** ***DB*::table**('frutas')**->where**('id', '=', ***$id***)**->first**();

**return** **view**('fruta.detalle', **array**(

                'fruta' **=>** ***$fruit***

        ));

  }



Creamos la ruta en el grupo de rutas definido anteriormente, pasando el parámetro id en la ruta

***Route*::group**(['prefix' **=>** 'frutas'], ***function*** () {

***Route*::get**('index','FrutasController@index');

***Route*::get**('detalle/{id}','FrutasController@detail');

});

Y por ultimo la vista:

CON ENLACE AL DETALLE

<h1>Mi listado de frutas:</h1>

<ul>

**@foreach**(***$frutas*** **as** ***$fruta***)

      <li>

          <a href="**{{action**('FrutasController@detail',['id' **=>**  ***$fruta*->*id***])**}}**">

                Nombre:**{{*$fruta*->*nombre*}}**

            </a>

        </li>

**@endforeach**

</ul>

**DETALLE FRUTA**

**@if**(**isset**(***$fruit***))

    <h1>Detalle de la fruta con id: **{{*$fruit* ->*id*}}**</h1>

    <h2>Nombre: **{{*$fruit*->*nombre*}}**</h2>

    <h4>Descripcion: **{{*$fruit*->*descripcion*}}**</h4>

**@else**

    <h2> No existe la fruta seleccionada</h2>

**@endif**

## 9.7 ORDER BY.

public ***function*** **index**()

  {

***$frutas*** **=** ***DB*::table**('frutas')

**->orderBy**('id', 'desc')

**->get**();

**return** **view**('fruta.index' , **array**(

            'frutas' **=>** ***$frutas***

        ));

    }

## 9.8 INSERTAR REGISTROS

### 9.8.1 FORMULARIO DE INSERCION.

<h1>Añadir nueva Fruta:</h1>

<form action="**{{action**('FrutasController@save')**}}**" method="POST">

**{{** **csrf\_field**() **}}**

    <label for="nombre">Nombre</label>

    <input type="text" name="nombre" /></br>

    <label for="nombre">Descripcion </label>

    <input type="text" name="descripcion" /></br>

    <label for="nombre">Precio</label>

    <input type="text" name="precio" /></br>

    <input type="submit" name="submit" value="Añadir" />

</form>

### 9.2.2 RUTAS .

***Route*::group**(['prefix' **=>** 'frutas'], ***function*** () {

***Route*::get**('index','FrutasController@index');

***Route*::get**('detail/{id}','FrutasController@detail');

***Route*::get**('create', 'FrutasController@create' );

***Route*::post**('save', 'FrutasController@save' );

});

### 9.3.3 METODOS DEL CONTROLADOR .

-> FrutasController

*// metodo para cargar vista con el form*

    public ***function*** **create**()

    {

**return** **view**('fruta.create');

    }

*//\* metodo para insertar un nuevo registro*

    public ***function*** **save**(***Request*** ***$request***)

    {

*// guardar datos*

***$fruta*** **=** ***DB*::table**('frutas')**->insert**(**array**(

                'nombre' **=>** ***$request*->input**('nombre'),

                'descripcion' **=>** ***$request*->input**('descripcion'),

                'precio' **=>** ***$request*->input**('precio'),

                'fecha\_reg' **=>** **date**("Y-m-d H:i:s")

        ));

**return** **redirect**()

**->action**('FrutasController@index')

*//añadimos sesion flash(solo se muestra 1 vez)*

*// llamada estatus*

**->with**('status', 'Fruta creada correctamente');

    }

En la vista vamos a mostrar la variable de sesión creada:

* Mediante **session('status')**

## 9.9 ELIMINAR REGISTROS.

Vamos a añadir un enlace para eliminar cada registro pasando el identificador para asi poder borrarlo.

<hr>

<a href="**{{action**('FrutasController@create')**}}**">Añadir nueva fruta</a>

<hr>

<ul>

**@foreach**(***$frutas*** **as** ***$fruta***)

      <li>

          <a href="**{{action**('FrutasController@detail', ['id' **=>** ***$fruta*->*id***])**}}**">

                Nombre:**{{*$fruta*->*nombre*}}**

           </a>

          <a href="**{{action**('FrutasController@delete', ['id' **=>** ***$fruta*->*id***])**}}**">

                | Eliminar

          </a>

      </li>

**@endforeach**

</ul>

### RUTA

***Route*::get**('delete/{id}', 'FrutasController@delete' );

### METODO

public ***function*** **delete**(***$id***)

    {

***$fruta*** **=** ***DB*::table**('frutas')**->where**('id', ***$id***)**->delete**();

**return** **redirect**()

**->action**('FrutasController@index')

**->with**('status', 'Fruta eliminada correctamente');

    }

Con with hemos creado una variable de sesión flash que solo se muestra 1 vez

### VISTA

Por ultimo la vista quedaría:

<h1>Mi listado de frutas:</h1>

**@if**(**session**('status'))

    <p style=" background:green; color:white;">

**{{session**('status')**}}**

    </p>

**@endif**

<hr>

<a href="**{{action**('FrutasController@create')**}}**">Añadir nueva fruta</a>

<hr>

<ul>

**@foreach**(***$frutas*** **as** ***$fruta***)

       <li>

           <a href="**{{action**('FrutasController@detail', ['id' **=>** ***$fruta*->*id***])**}}**">

               Nombre:**{{*$fruta*->*nombre*}}**

            </a>

           <a href="**{{action**('FrutasController@delete', ['id' **=>** ***$fruta*->*id***])**}}**">

                | Eliminar

           </a>

           <a href="**{{action**('FrutasController@update', ['id' **=>** ***$fruta*->*id***])**}}**">

                | Actualizar

           </a>

       </li>

**@endforeach**

</ul>

## 9.10 EDITAR REGISTROS.

Para aprovechar código v

**@if**(**isset**(***$fruta***) **&&** **is\_object**(***$fruta***))

    <h1>Editar fruta</h1>

**@else**

    <h1>Crear una fruta</h1>

**@endif**

<form

action="**{{** **isset**(***$fruta***) ? **action**('FrutaController@update')  : **action**('FrutaController@save') **}}**" method="POST">

**{{** **csrf\_field**() **}}**

**@if**(**isset**(***$fruta***) **&&** **is\_object**(***$fruta***))

       <input type="hidden" name="id" value="**{{** ***$fruta*->*id*** **}}**"/>

**@endif**

    <label for="nombre">Nombre</label>

    <input type="text" name="nombre"

value="**{{** ***$fruta*->*nombre*** **or** ''**}}**"/><br/>

    <label for="descripcion">Descripción</label>

    <input type="text" name="descripcion"

value="**{{** ***$fruta*->*descripcion*** **or** ''**}}**"/><br/>

    <label for="precio">Precio</label>

    <input type="number" name="precio"

value="**{{** ***$fruta*->*precio*** **or** 0**}}**"/><br/>

    <input type="submit" value="Guardar" />

</form>

*// Carga vista con form add fruta*

    public ***function*** **create**(){

**return** **view**('fruta.create');

    }

*//almacena registro*

    public ***function*** **save**(***Request*** ***$request***){

***$nombre*** **=** ***$request*->input**('nombre');

***$descripcion*** **=** ***$request*->input**('descripcion');

***$precio*** **=** ***$request*->input**('precio');

***$fecha\_reg*** **=** **date**('Y-m-d');

*// guardar el registro*

***$fruta*** **=** ***DB*::table**('frutas')**->insert**(**array**(

            'nombre' **=>** ***$nombre***,

            'descripcion'  **=>** ***$descripcion***,

            'precio' **=>** ***$precio***,

            'fecha\_reg' **=>** ***$fecha\_reg***

        ));

**return** **redirect**()

**->action**('FrutaController@index')

**->with**('status', 'Fruta creada correctamente');

    }

    public ***function*** **edit**(***$id***){

*// sacar el registro de la bd*

***$fruta*** **=** ***DB*::table**('frutas')**->where**('id', ***$id***)**->first**();

*// pasarle a la vista el objeto y rellenar el formulario*

**return** **view**('fruta.create',[

            'fruta' **=>** ***$fruta***

        ]);

    }

    public ***function*** **update**(***Request*** ***$request***){

***$id*** **=** ***$request*->input**('id');

***$nombre*** **=** ***$request*->input**('nombre');

***$descripcion*** **=** ***$request*->input**('descripcion');

***$precio*** **=** ***$request*->input**('precio');

***$fruta*** **=** ***DB*::table**('frutas')

**->where**('id', ***$id***)

**->update**(**array**(

                               'nombre' **=>** ***$nombre***,

                               'descripcion' **=>** ***$descripcion***,

                               'precio' **=>** ***$precio***

                                    ));

**return** **redirect**()

**->action**('FrutaController@index')

**->with**('status', 'Fruta actualizada correctamente');

    }

***Route*::group**(['prefix'**=>**'frutas'], ***function***(){

***Route*::get**('index', 'FrutaController@index');

***Route*::get**('detail/{id}', 'FrutaController@detail');

***Route*::get**('crear', 'FrutaController@create');

***Route*::post**('save', 'FrutaController@save');

***Route*::get**('editar/{id}', 'FrutaController@edit');

***Route*::post**('update', 'FrutaController@update');

***Route*::get**('delete/{id}', 'FrutaController@delete');

});

# 10. SESIONES.

La configuración del sistema HTTP Session se realiza en el archivo config/session.php. Es aquí donde decimos cuál es el soporte donde se van a almacenar las sesiones, por cuánto tiempo, etc.

Como decíamos, la principal novedad del sistema de sesiones de Laravel con respecto a PHP nativo es la facilidad con la que podemos alterar el funcionamiento interno de las sesiones, para definir un soporte de almacenamiento diverso. Las opciones principales son las siguientes:

* **file** - Las variables de sesión se almacenan en ficheros, sobre la ruta: storage/framework/sessions.
* **cookie** - Las variables de sesión son almacenadas en cookies en el navegador, debidamente encriptadas.
* **database** - Las sesiones son implementadas en la base de datos. Atención, pues esta configuración requiere que se ejecuten ciertas migraciones para disponer de las tablas de almacenamiento.
* **memcached / redis** - Este soporte permite usar un sistema de caché en memoria, más rápido que otras alternativas.

El archivo de configuración de las sesiones está bastante bien auto-explicado. Un vistazo rápido te permitirá configurar el sistema de sesiones. De manera predeterminada Laravel usa el sistema de archivos. Es importante que leas la documentación con atención si quieres cambiar este soporte, para conocer los requisitos de aquel que desees configurar.

**Mecanismos para el acceso al sistema de sesión de Laravel**

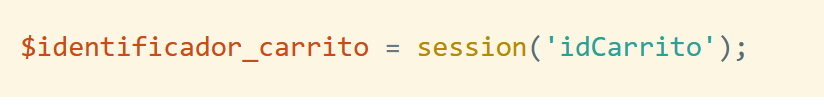
A diferencia de cuando trabajamos directamente con las sesiones PHP, Laravel se ocupa de diversas operativas administrativas, como abrir la sesión al ejecutarse una solicitud. Por ello, podemos usar las variables de sesión en el momento que deseemos.

Para comenzar debes saber que existen dos alternativas para acceder a las funciones necesarias para trabajo con la sesión.

## 10.1 Helper global session .

Este helper te permite acceder al API para el acceso a la sesión desde cualquier parte de tu código, incluso desde las vistas si fuera necesario. Es la manera más cómoda para acceder a los datos de la sesión, modificarlos, etc.

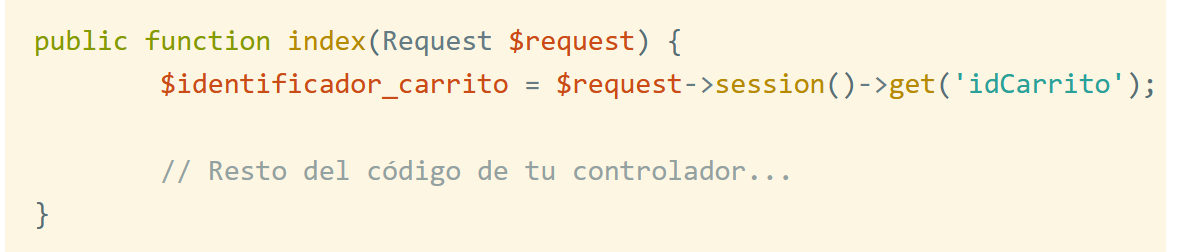
Por ejemplo, así podemos acceder a una variable de sesión llamada "idCarrito".



### instancia de Request

A partir del objeto Request de Laravel podemos también acceder al API de las sesiones. Esta alternativa requiere por tanto disponer de un objeto $request

Este sería el código de un método de un controlador. Puedes observar cómo se declara la inyección del objeto $request como parámetro. Una vez tenemos la request, podemos obtener una variable de sesión "idCarrito" de esta manera.



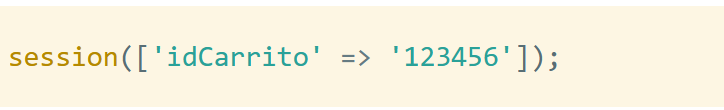
Sólo debes tener en cuenta que los mecanismos para el acceso a los datos de la sesión cambian un poco de uno a otro modelo de trabajo. Nosotros en lo sucesivo vamos a mostrar código accediendo al helper global, ya que es más universal y lo puedes usar sin ceremonia alguna desde cualquier parte de tu aplicación.

## 10.3 Crear y recuperar variables de sesión con Laravel

Ahora veamos distintas operativas que se pueden hacer con las sesiones, con la modalidad de trabajo permitida por el helper session.

### Almacenar una variable de sesión

Simplemente le indicamos en un array asociativo qué variables de sesión deseamos almacenar. Este array contiene pares clave/valor. La clave indica el nombre de la variable y el valor, su valor.



### Recuperar una variable de sesión

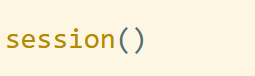
El helper nos permite de una manera muy cómoda recuperar un dato almacenado en la sesión, indicando simplemente qué nombre de variable deseamos acceder.



## 10.4 Objeto session.

Algunas operativas de la sesión se realizan accediendo a un objeto "session", a partir del cual tenemos acceso a varios métodos de utilidad.

El objeto session lo obtenemos ejecutando el helper global sin enviarle ningún parámetro



A menudo ese objeto de sesión se usa sin necesidad de almacenarlo en ninguna variable, enganchando directamente el método que quieras ejecutar.

### 10.4.1 BORRAR VARIABLE DE SESION.

Mediante el objeto de session podemos decirle a Laravel que borre una variable de sesión. Usamos para ello el método forget(), de este modo:



### 10.4.2 Comprobar la existencia de una variable de sesión .

A veces es necesario saber si existe en el sistema de sesiones de Laravel una variable determinada. Para ello usamos el método has().



### 10.4.5 Regenerar el identificador de sesión .

El identificador de la sesión permite a Laravel, y a nosotros mismos, identificar una sesión de un usuario en la página.

Cada sesión tiene su identificador, que se mantiene a lo largo de toda la visita. Sin embargo, cuando un usuario se loguea en el sistema, con sus datos de usuario, este identificador se regenera automáticamente.

Esta es una funcionalidad predeterminada en Laravel, pero si por cualquier motivo necesitas regenerarlo de manera manual, también puedes hacerlo a partir del método regenerate().

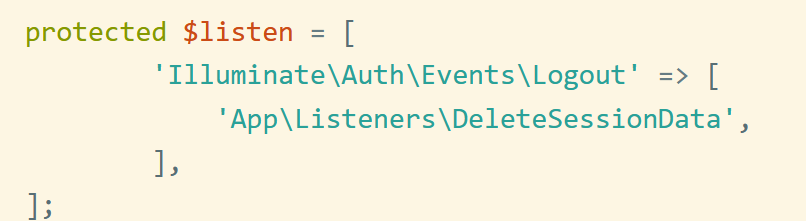


### 10.4.6 Borrar los datos de sesión al hacer logout .

Aunque al hacer un logout podríamos suponer que los datos de la sesión deberían borrarse, este no es un comportamiento predeterminado en Laravel, al menos no en todas las versiones.

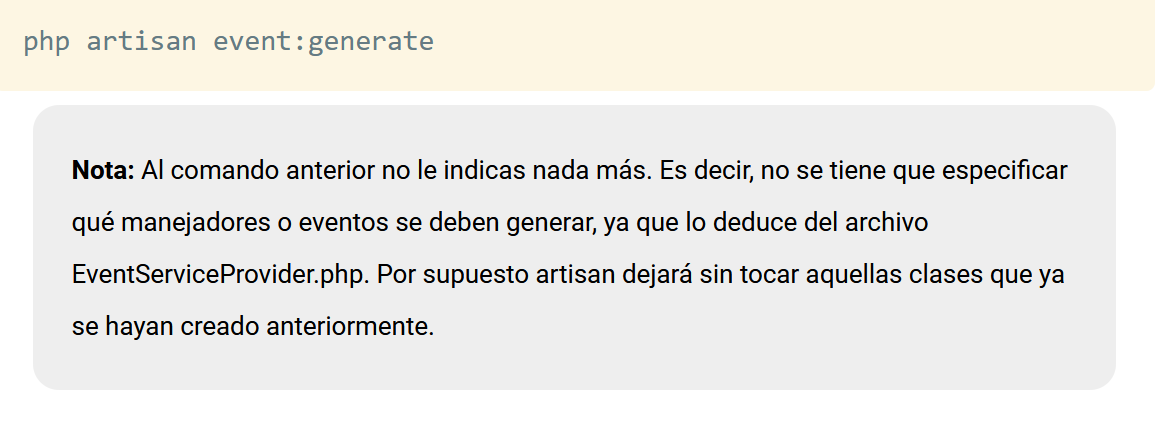
Si no queremos modificar a mano el sistema de login ya implementado por Laravel, podemos realizar una sencilla operativa, que consiste básicamente en generar un manejador de evento para "Logout".

Esto se declara en el archivo app/Providers/EventServiceProvider.php mediante el array $listeners.



Mediante el código anterior estamos indicando que cuando se dispare el evento "Logout", se invoque el manejador DeleteSessionData.

Ese manejador se puede auto-generar gracias a un comando de Artisan. Ese comando de generación lo tienes que ejecutar después de haber editado el EventServiceProvider.php y se encarga de crear el scaffolding de todas las clases necesarias que se estén usando para gestionar eventos.



La clase DeleteSessionData, una vez generada y editada, tendrá un código como este:

