ACTIVIDAD #2

04MASW - Desarrollo de aplicaciones Web I:  
 Lado del Servidor (back-end)

22/02/2022

Alberto Mateo Martínez

Índice de contenido

[1. Introducción 3](#_Toc96444409)

[2. Requisitos Funcionales 4](#_Toc96444410)

[3. Desarrollo conceptual 5](#_Toc96444411)

[4.Implementación 7](#_Toc96444412)

[Instalación del entorno 7](#_Toc96444413)

[Creación de la Base de datos (vacía) 7](#_Toc96444414)

[Verificación / instalación de composer y Laravel 8](#_Toc96444415)

[Creación del Proyecto” series” 10](#_Toc96444416)

[Migraciones – Creación de los datos de las tablas 15](#_Toc96444417)

[Creación de los modelos 22](#_Toc96444418)

[Creación de los Controladores: 24](#_Toc96444419)

[Creación de las rutas 27](#_Toc96444420)

[Creación de vistas utilizando Blade 30](#_Toc96444421)

[Autentificación 31](#_Toc96444422)

[Seeder -” sembradores ” de datos. 35](#_Toc96444423)

[Paginación 40](#_Toc96444424)

[SoftDeletes 41](#_Toc96444425)

[Uso del Layout 43](#_Toc96444426)

[Código empleado en los controladores ( Platfom, Language, Actor, Director, Serie, Cast) 44](#_Toc96444427)

[Método Index 44](#_Toc96444428)

[Método Create 45](#_Toc96444429)

[Método Store 45](#_Toc96444430)

[Método Edit 45](#_Toc96444431)

[Método Update 45](#_Toc96444432)

[Vista de Listar (y buscar) 46](#_Toc96444433)

[Vista de Crear Actualizar 47](#_Toc96444434)

[5. Conclusiones 48](#_Toc96444435)

[5. Bibliografía 49](#_Toc96444436)

[6. Anexos 50](#_Toc96444437)

[Anexo1: Vistas de la Aplicación 50](#_Toc96444438)

[Pantalla inicial 50](#_Toc96444439)

[Creación, Listado, Edición y Borrado de Plataformas 51](#_Toc96444440)

[Creación, Listado, Edición y Borrado de Idiomas 53](#_Toc96444441)

[Creación, Listado, Edición y Borrado de directores 56](#_Toc96444442)

[Creación, Listado, Edición y Borrado de Actores 57](#_Toc96444443)

[Creación, Listado, Edición y Borrado de series 58](#_Toc96444444)

[Creación, Listado, Edición y Borrado de la relación Actores - Series 59](#_Toc96444445)

# 1. Introducción

El objetivo de esta actividad es la programación de un sitio web desarrollado con el Framework LARAVEL 6 de PHP del lado del servidor.

El sitio web solicitado es una biblioteca de Series que contendrá datos de Series, Plataforma, Idioma, Directores, Actores, etc.

Se acompaña dicha aplicación web con el presente documento. De forma breve quedan desarrollados los Requisitos, Diseño conceptual y la Implementación. Más que constituir un documento formal, aprendido en otras asignaturas, se trata de una presentación de la aplicación web, de los fundamentos y bases de Back-End con las que se ha construido.

Por la facilidad de uso, por integración con **Visual Studio Code**, y sobre todo por el conocimiento del conjunto de herramientas se elige realizar la practica con **MySQL** (Mariadb) sobre el paquete o pila de aplicaciones de código abierto **XAMP. Versión 3.3.0** (Compilación 6 de abril de 2021)

# 2. Requisitos Funcionales

La actividad establece unos requisitos generales. (similares a los de la Actividad 1) Se trata de realizar una aplicación Web para almacenar una biblioteca de series. El sistema realizado ha de almacenar las series de manera que sean recuperables. Se fijan otros condicionantes para asegurar la integridad de los datos como control de duplicados, control de claves foráneas, etc.

Las funcionalidades posibles son muy numerosas y todas muy interesantes. Se podrían crear más tablas, vistas, etc.. Debido al tiempo limitado, se ha optado por una simplificación del sistema al objeto de poder experimentar las características con mayor profundidad. Por todo ello se realizan las siguientes simplificaciones:

* Una serie solo puede estar en una plataforma
* Una serie solo puede tener un idioma
* Una serie solo puede tener un director.

En cambio, un actor puede participar “Cast” en varias series y una Serie tiene varios Actores. Por ello, la relación “n a m”, muchos a muchos, se transforma en una tabla “Cast” para dar soporte a las relaciones entre Actores y Series.

Las funcionalidades definitivas son:

1. Creación y listado de **Plataformas**. Desde el listado se pueden borrar y editar las plataformas existentes.
2. Creación y listado de **Idiomas**. Desde el listado se pueden borrar y editar los idiomas existentes.
3. Creación y listado de **Directores**. Desde el listado se pueden borrar y editar los directores existentes.
4. Creación y listado de **Actores**. Desde el listado se pueden borrar y editar los Actores existentes.
5. Creación y listado de **Series**. Desde el listado se pueden borrar y editar los directores existentes.
6. Creación y listado de **Relaciones entre Actores y Series**. Desde el listado se pueden borrar y editar los directores existentes.

Así mismo se establecen como requisitos no funcionales:

* Las **fechas se muestran en formato español.**
* Se ha de evitar que la creación de **registros duplicados**
* Se ha de evitar que se borren registros de entidades con **claves foráneas.**

# 3. Desarrollo conceptual

Igual que en la Actividad 1, Para comprender e ilustrar mejor los requisitos funcionales se realiza un diagrama de casos de uso:

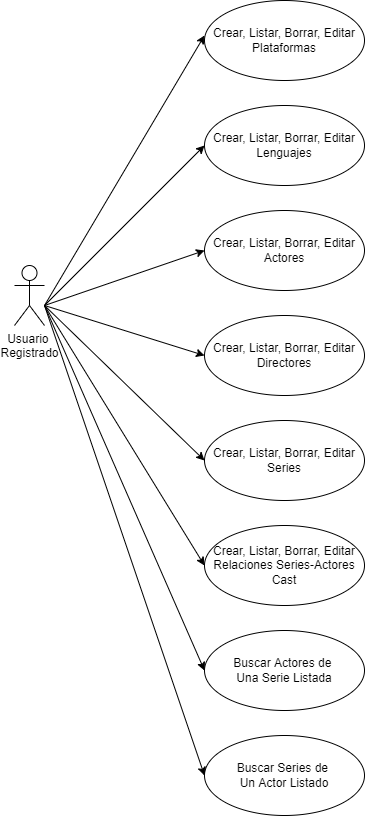


Figura 1. Diagrama de casos de uso.

**CASO DE USO**

1. **Mantenimiento de Plataformas.** El usuario puede crear nuevas plataformas, listar las existentes. Sobre el listado de las existente puede editar o eliminar
2. **Mantenimiento de Lenguajes.** El usuario puede crear nuevos lenguajes, listar los existentes. Sobre el listado de los existente puede editar o eliminar.
3. **Mantenimiento de Actores.** El usuario puede crear nuevos Actores, listar los existentes. Sobre el listado de los existentes puede editar o eliminar.
4. **Mantenimiento de Directores.** El usuario puede crear nuevos Directores, listar los existentes. Sobre el listado de los existentes puede editar o eliminar.
5. **Mantenimiento de Series.** El usuario puede crear nuevas Series, listar las existentes. Sobre el listado de las existente puede editar o eliminar
6. **Mantenimiento de Relaciones Series-Actor.** El usuario puede crear nuevas Relaciones, listar las existentes. Sobre el listado de las existente puede editar o eliminar
7. **Búsqueda de Plataformas, Actores, etc.**

Se elabora también un **diagrama Entidad-Relación** con el fin de organizar y representar los datos de contenido relativos a cada una de las entidades principales y las relaciones existentes entre ellos:

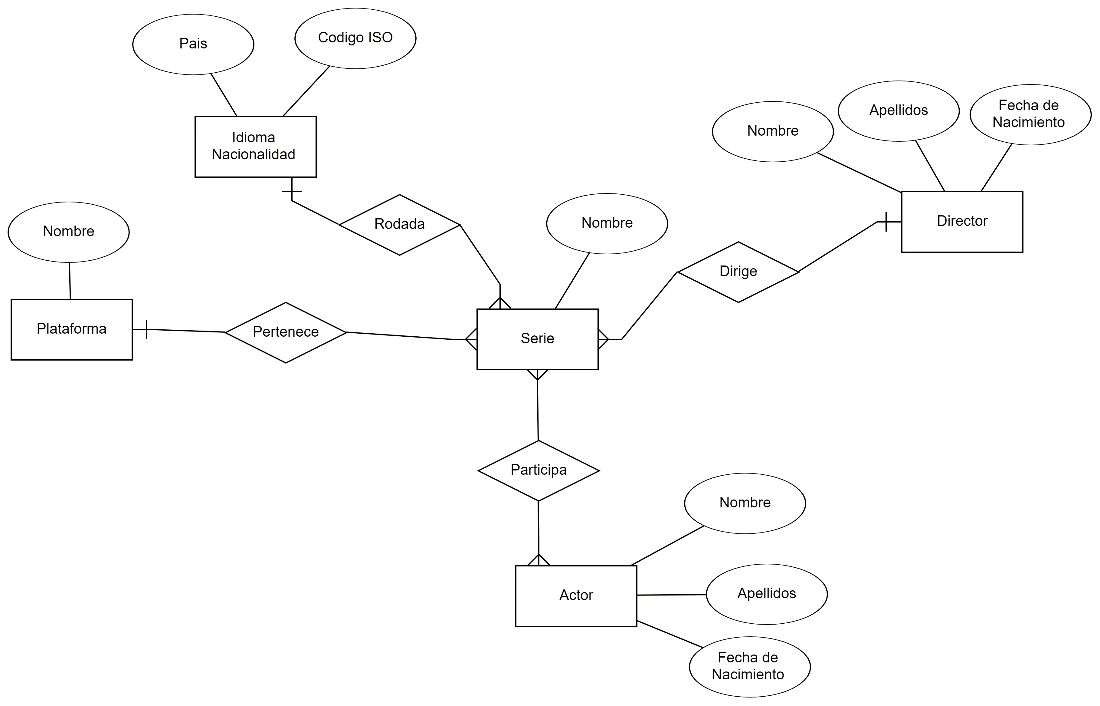
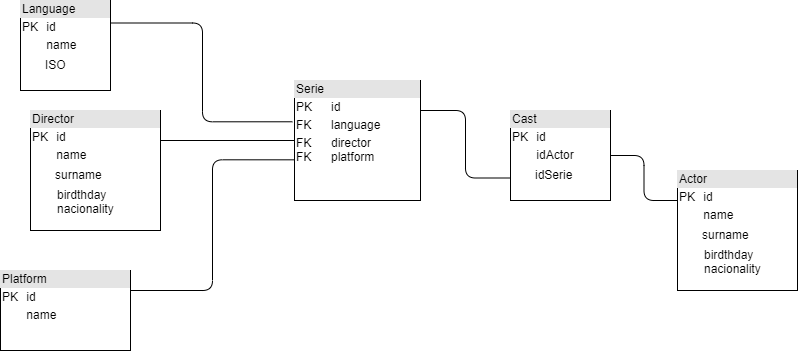


Figura 2. Diagrama Entidad – Relación.

El diagrama de clases de la solución adoptada es:



# 4.Implementación

Se incluye los detalles del proceso de implementación con Laravel 6 considerados más interesantes. Se ha desarrollado en entorno Windows, no obstante, se podría haber desarrollado perfectamente también en un entorno Linux.

El proceso de instalación se explica de manera simplificada en el fichero readme.txt. En este documento se desarrolla más detalladamente con el apoyo de imágenes y con razonamientos más completos de las decisiones realizadas.

Instalación del entorno

### Creación de la Base de datos (vacía)

Se ha creado una base de datos llamada” actividad2”.

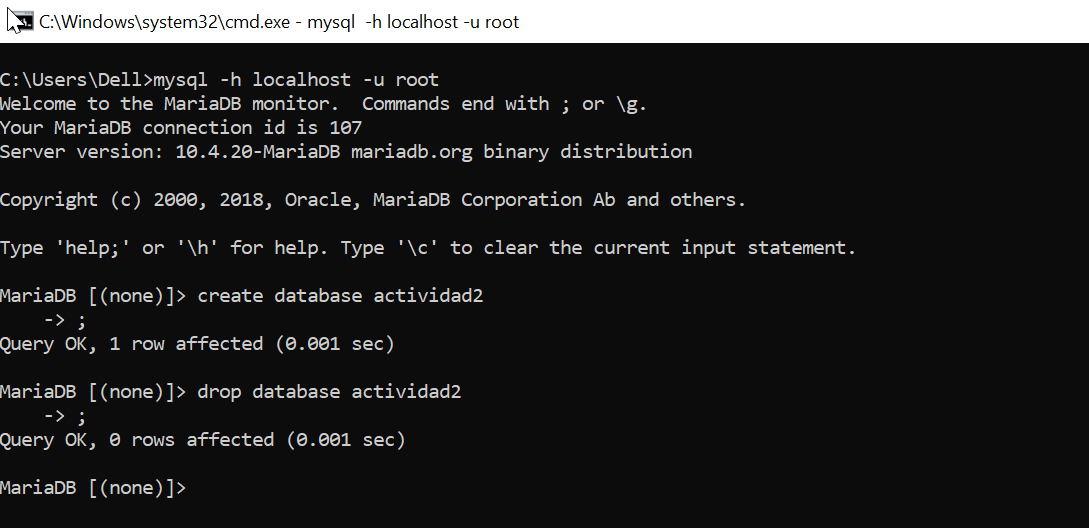
Se ha probado a crearla de dos formas:

1.- Por línea de comandos

mysql –h localhost –u root

Y después

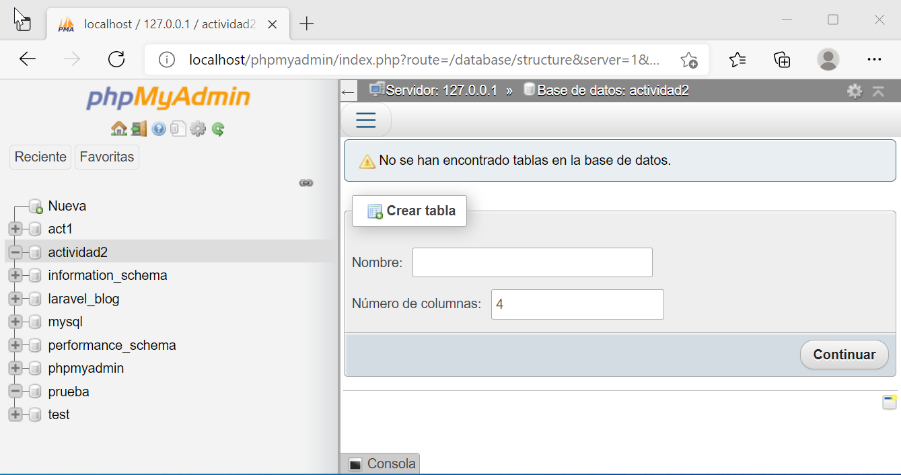
create database actividad2



2.- Desde phpmyadmin



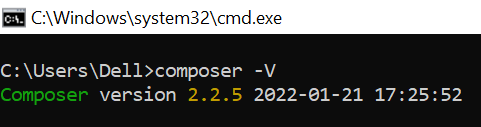
El resultado es una base de datos ”actividad2” vacía



### Verificación / instalación de composer y Laravel

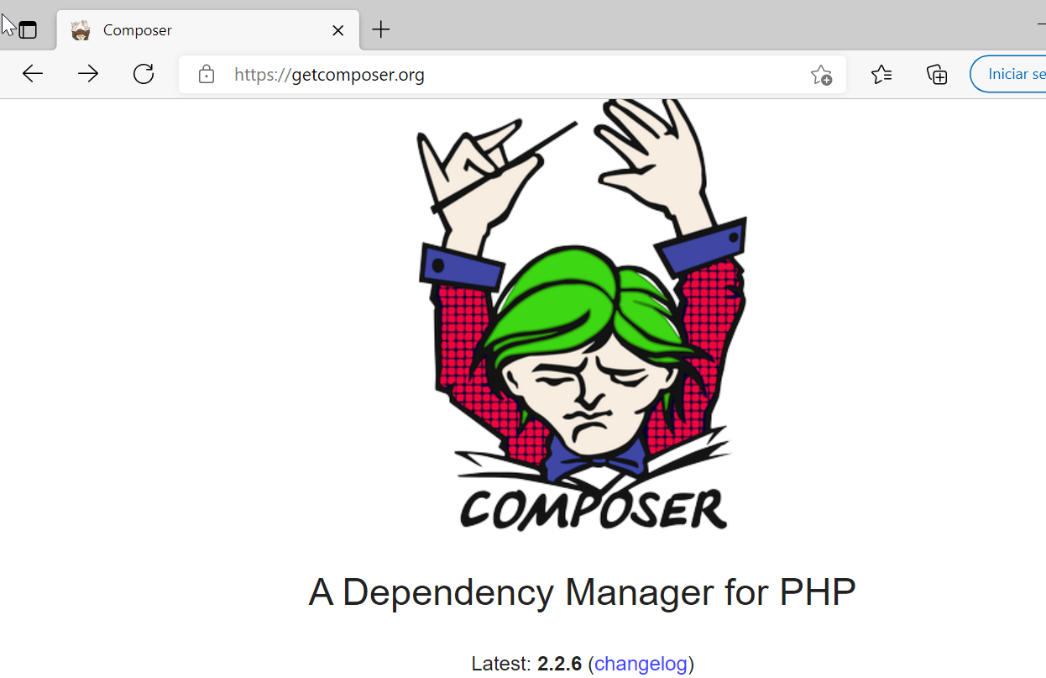
Compruebo que tengo instalado composer en el equipo donde estamos desarrollando la actividad.

Composer -V



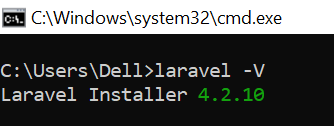
Si no lo tuviésemos, o quisiéramos actualizarlo iríamos a la web de composer:

https://getcomposer.org



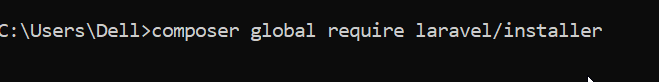
Verifico la versión de Laravel que tengo instalado Laravel en el equipo de desarrollo

Laravel -V



Si no lo tuviésemos, procederíamos a instalarlo vía composer

composer global require laravel/installer



## Creación del Proyecto” series”

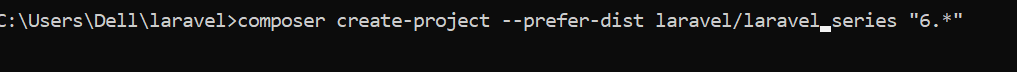
Se puede utilizar de dos formas para crear el proyecto: por medio del comando

laravel new nombredeproyecto

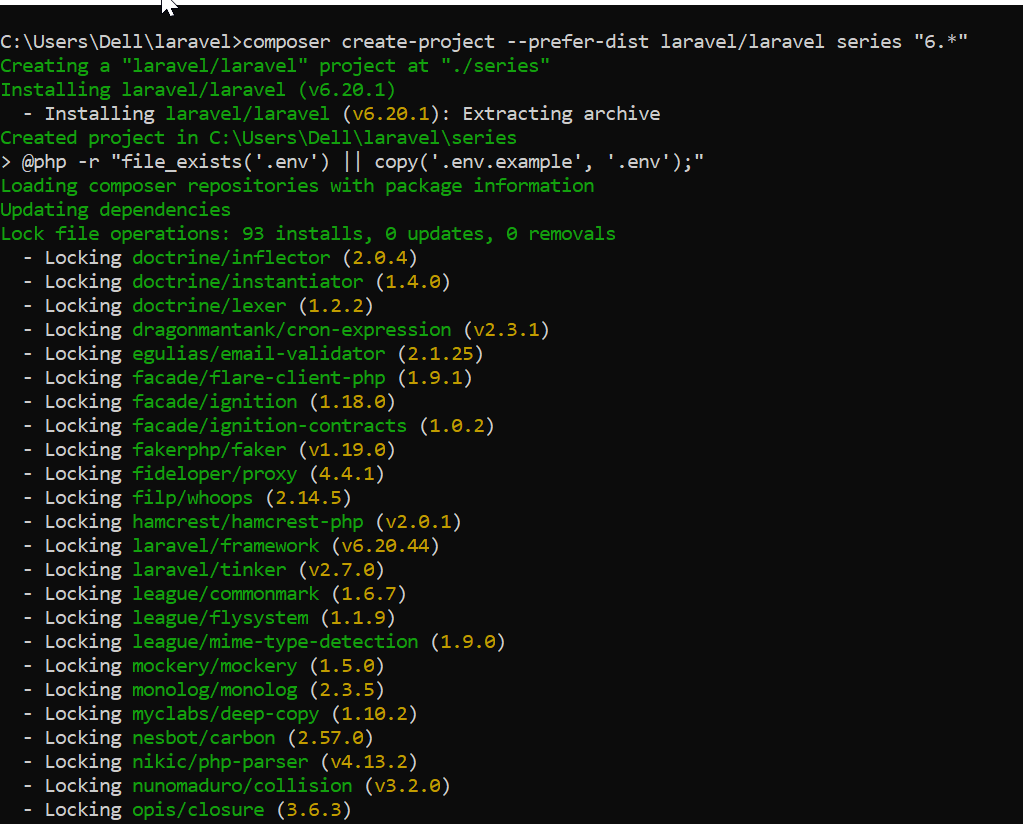


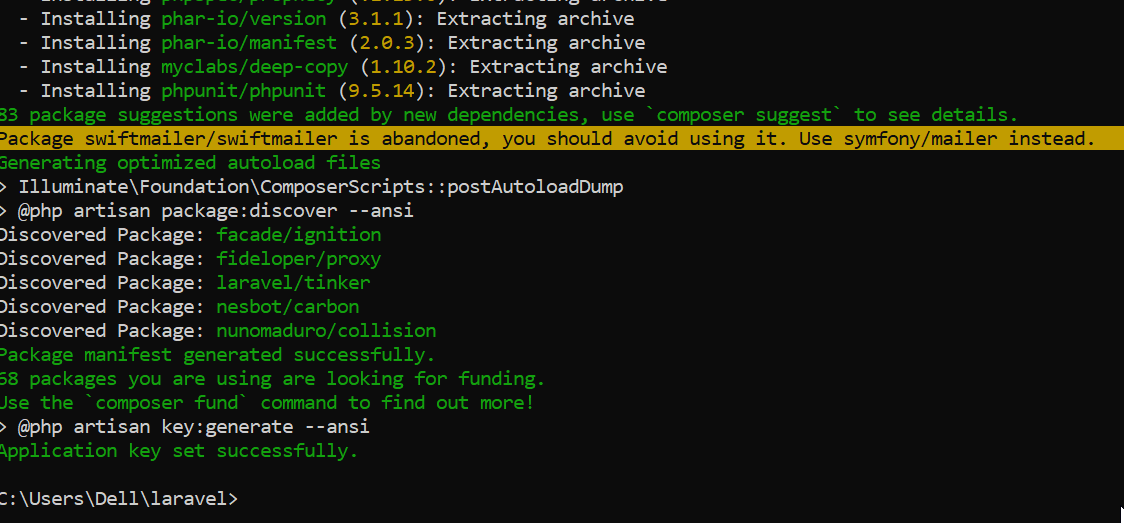
O también a través de composer. Utilizamos el siguiente comando:

composer create-project --prefer-dist laravel/laravel series ”6.\*”



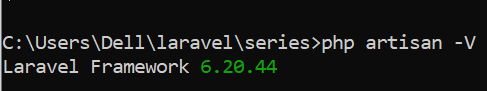
Se ha elegido realizarlo a través de composer tal y como se muestra en las siguientes figuras



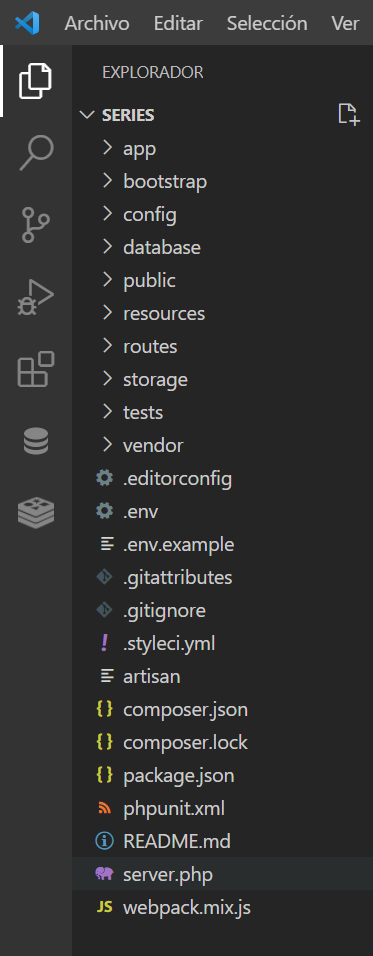
Termina satisfactoriamente

Para saber la versión exacta me mi proyecto Laravel ejecuto en una terminal el siguiente comando

php artisan -V

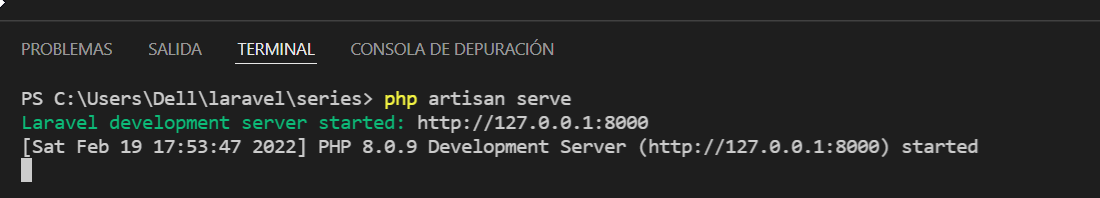


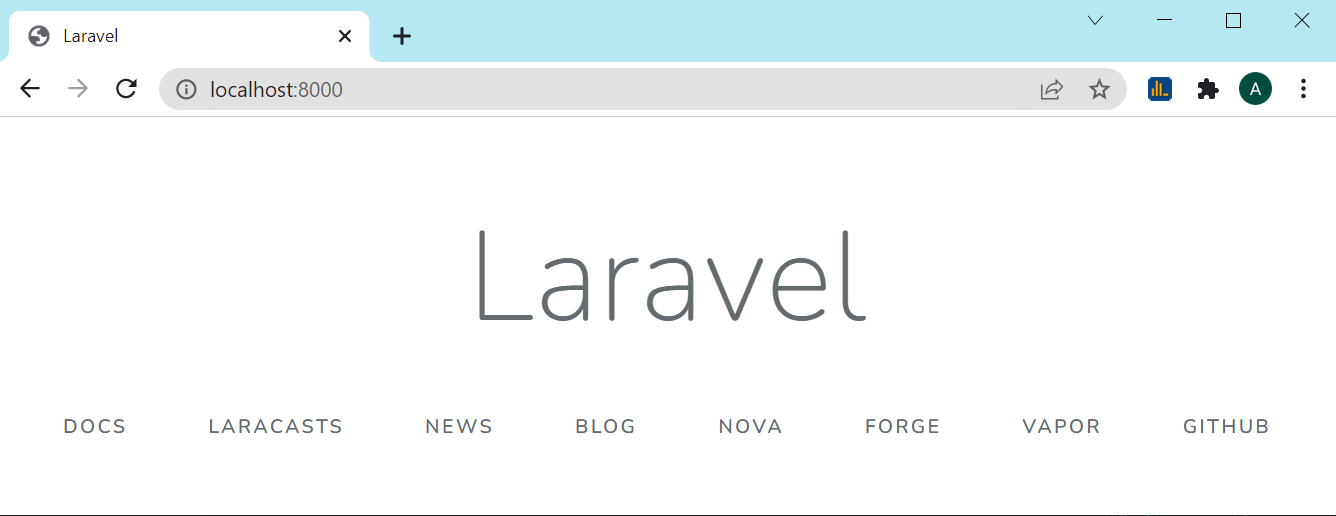
Abrimos con visual studio code el árbol de carpetas del proyecto para ver que todo esta correcto



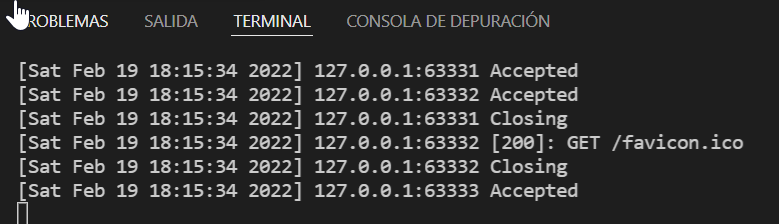
Lanzamos el servidor, y comprobamos que la página de bienvenida se carga correctamente

php artisan serve





En la terminal en la que levanto el servidor puedo observar las llamadas de la página

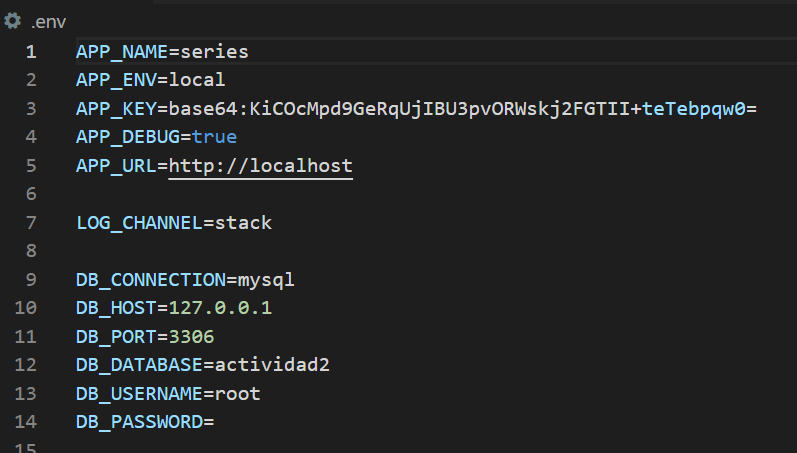


## Configuracion del Proyecto Series

En el fichero .env configuramos el nombre de la aplicación ”series“ y el nombre de la base de datos:”actividad2”

DB\_DATABASE=nombredelabasededatos.

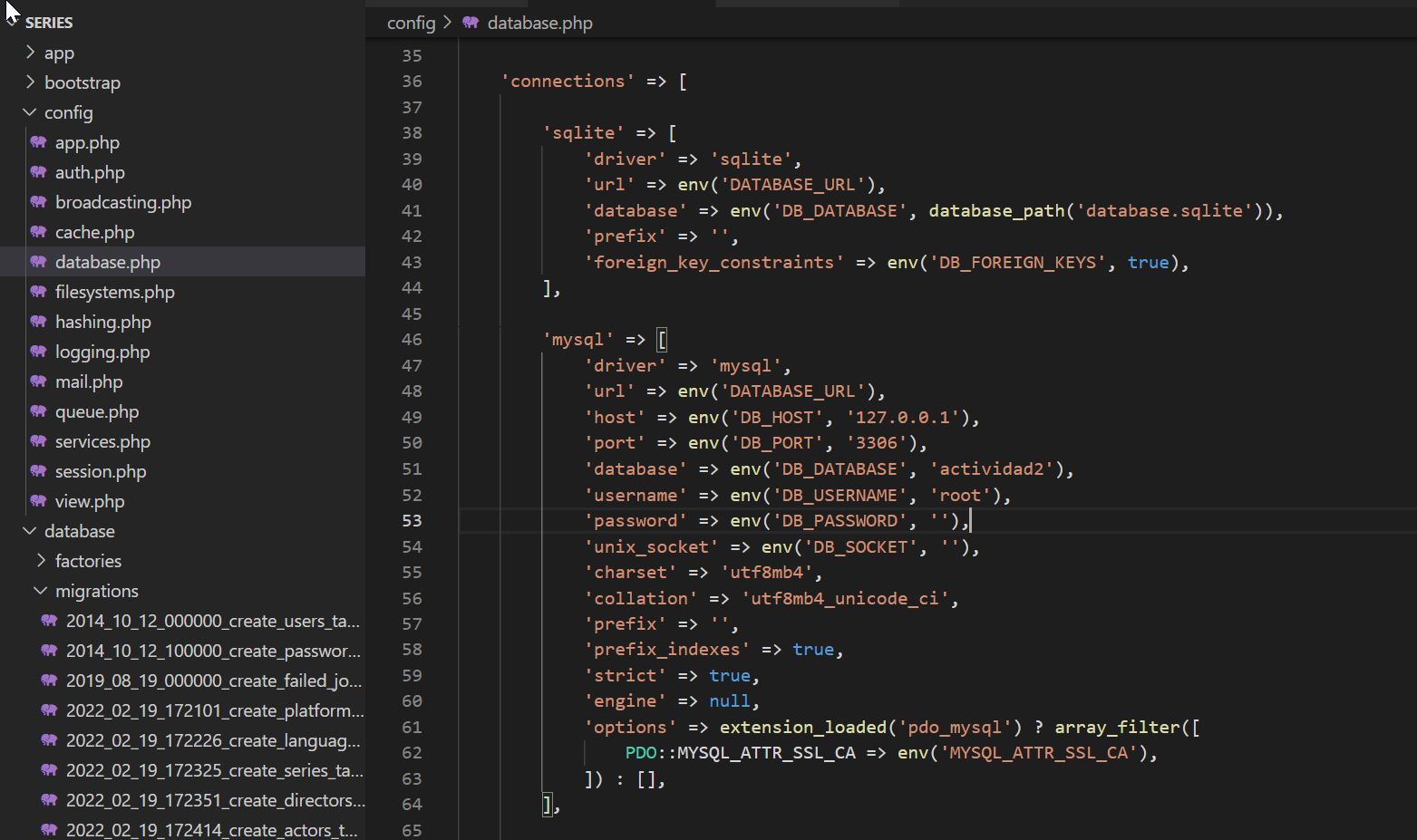
Dejo el entorno como viene por defecto: APP\_ENV=local más adelante se deberá de cambiar a APP\_ENV=production



También debo configurar el fichero database.php dentro del directorio /config

Cambio ’database‘ =>(‘DB\_DATABASE‘, ’actividad2’)

Cambio ’username’ =>(‘DB\_USERNAME, ’root’)



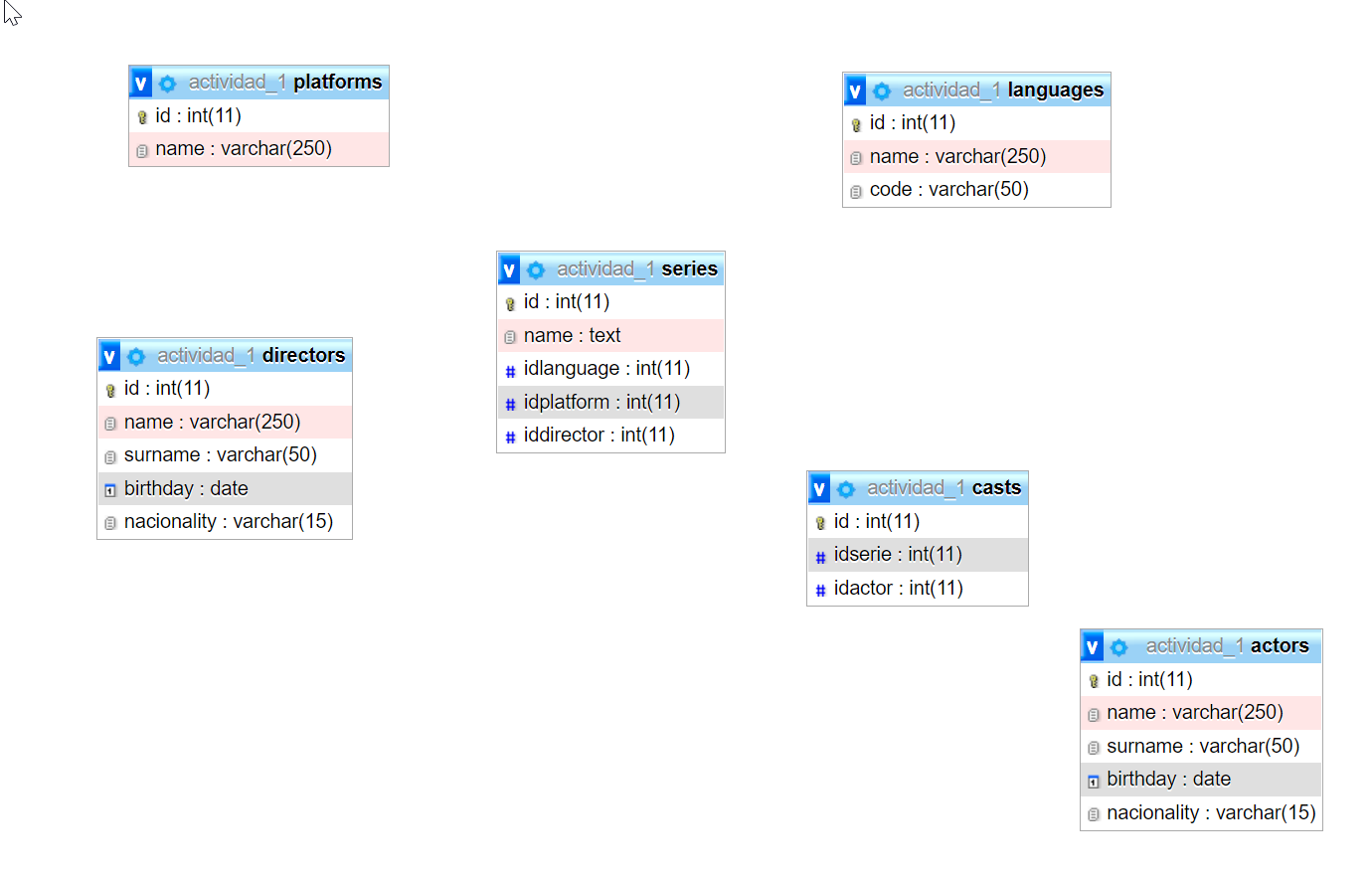
## Migraciones – Creación de los datos de las tablas

Todo el proceso de creación de tablas se ha realizado través de” migraciones” incluso la creación del usuario por defecto.

Se crea un fichero de migración para una de las tablas.

Al lanzar las migraciones puedo comprobar si se han creado bien las conexiones en el apartado anterior. En caso de error, la aplicación nos lo mostrará.

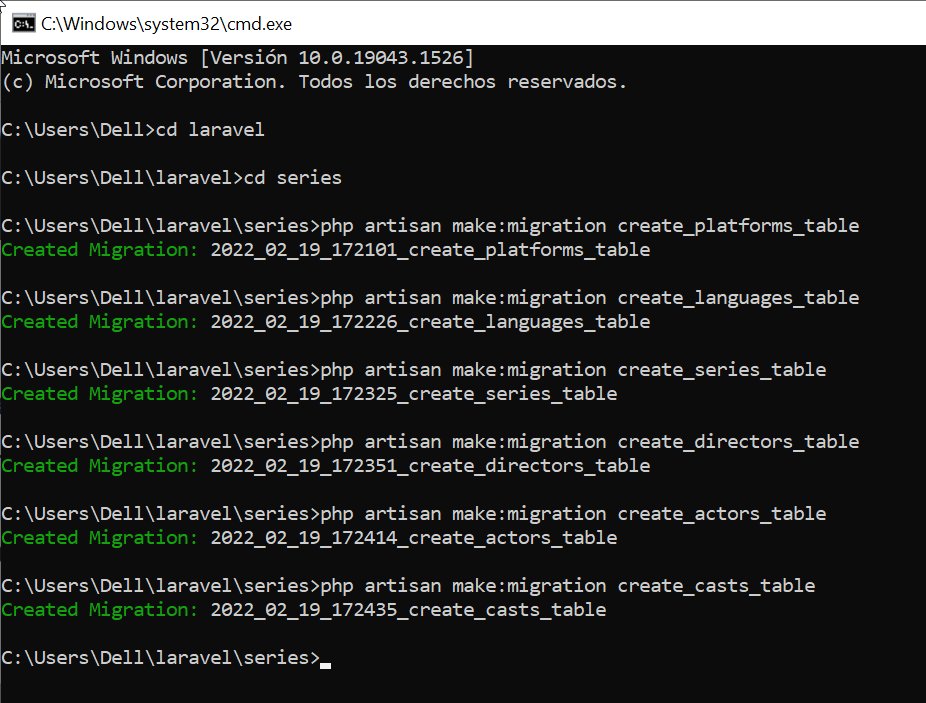
Para la creación de las tablas se ha usado como plantilla la misma estructura utilizada en la actividad 1. Se observa en la figura siguiente.



Se crean los archivos de migración con el siguiente comando de artisan:

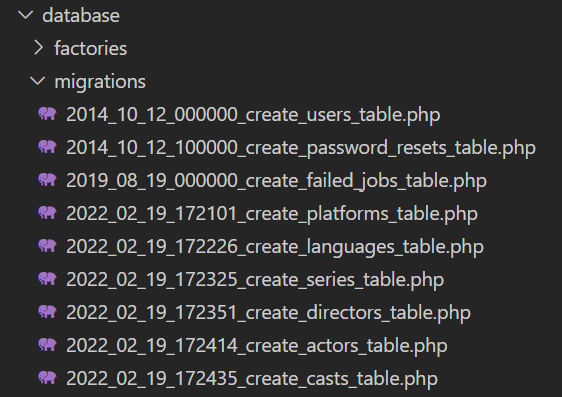
php artisan make:migration Nombredelamigracion

Como son clases deben empezar el nombre debe empezar por letras mayúsculas y el fichero ha de llamarse igual que la clase



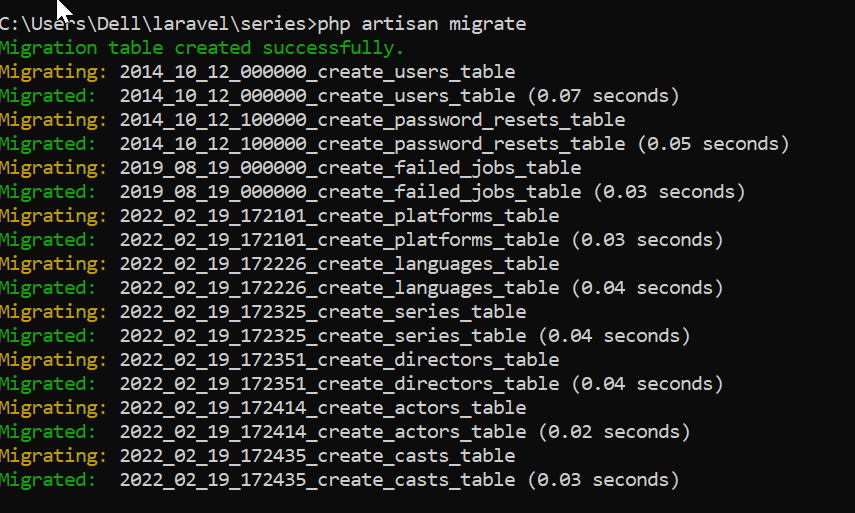
Los ficheros son creados automáticamente en la ruta /database/migrations.

Los tres primeros se habían creado automáticamente junto a la creación del proyecto, los otros siguientes, son los que se acaban de generar con el comando anterior

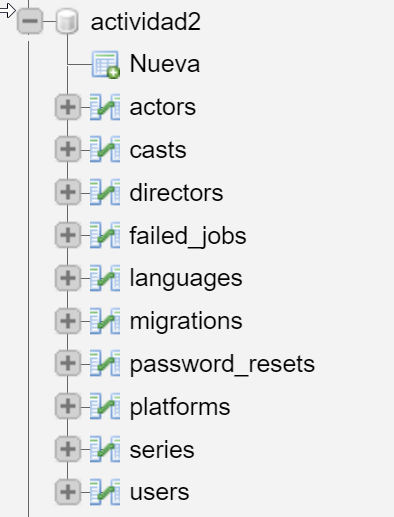


Ejecuto las migraciones (Todas juntas) observando que la conexión con la base de datos es satisfactoria con el siguiente comando de artisan:

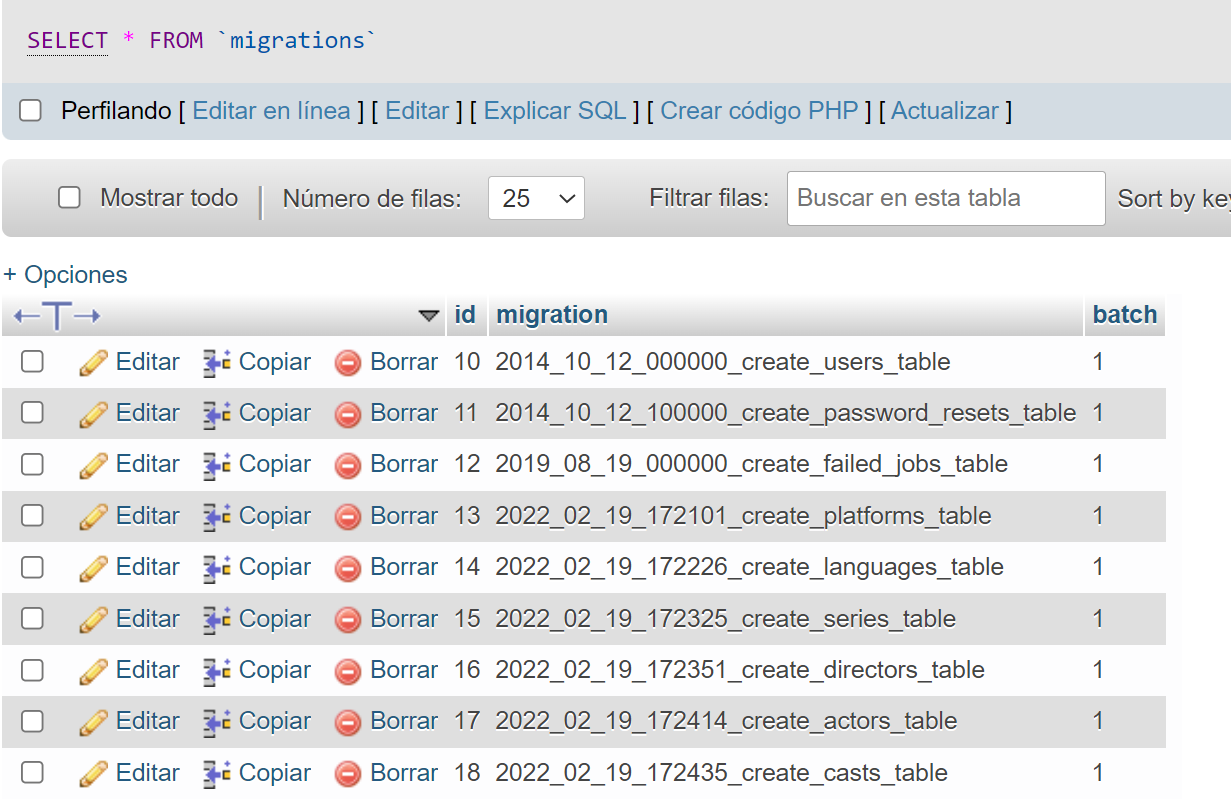
php artisan migrate



Compruebo el resultado viendo las tablas creadas en la base de datos con phpmyadmin



Observo también el contenido de la tabla migrations para asegurarme que todo esta correctamente.

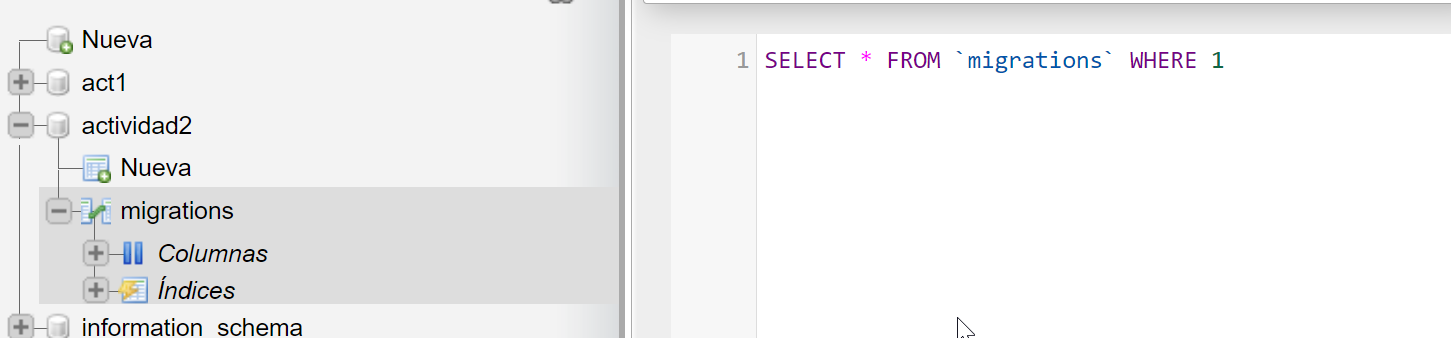


Deshago las migraciones. Para ello, como únicamente se ha lanzado un paso de migrate, batch=1 con un solo paso de rollback elimino la totalidad de las tablas creadas.

php artisan migrate:rollback



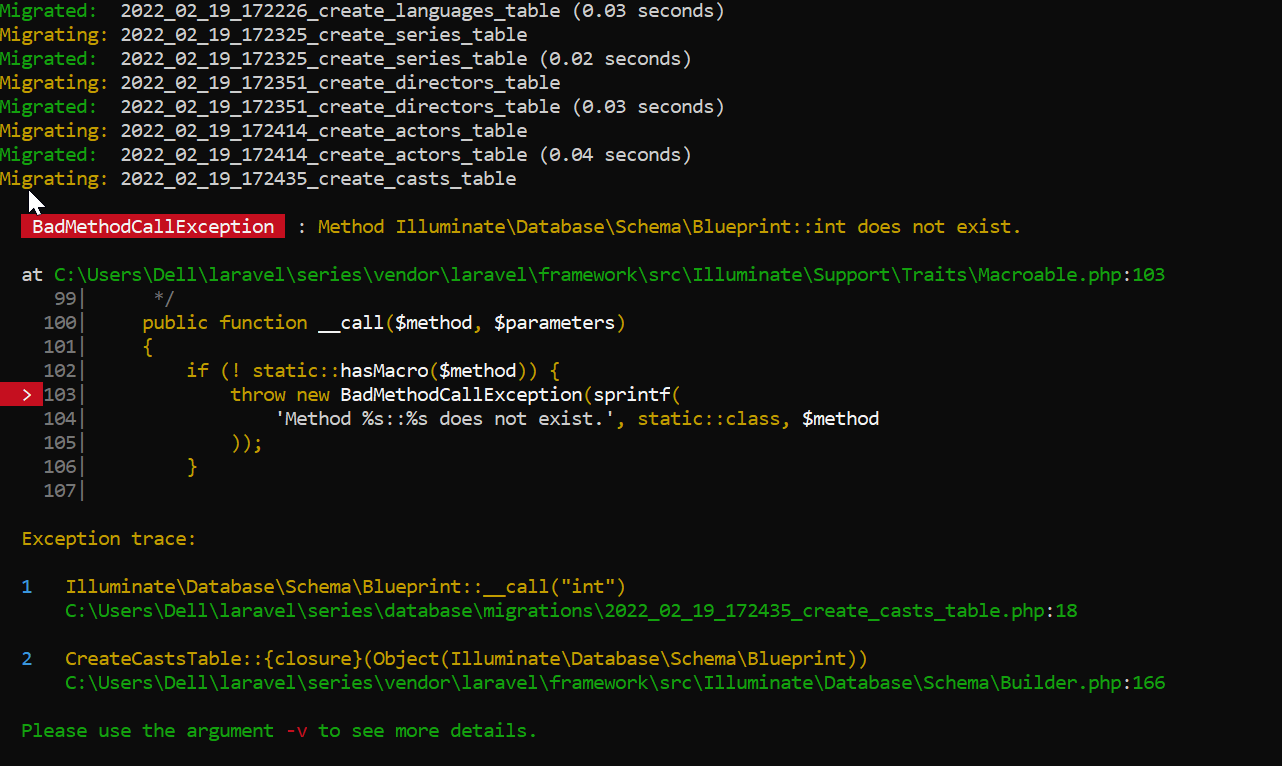
Compruebo el resultado en phpmyadmin. Únicamente ha quedado la tabla migrate pero sin registros.



Modifico desde visual studio los archivos de migración creados en la carpeta database para introducir los campos requeridos de cada tabla.

Este proceso de crear y borrar, lo repetiré varias veces hasta que no haya errores y todas las tablas queden con la estructura que deseada.

Depurando los errores que se cometan. Por ejemplo este:

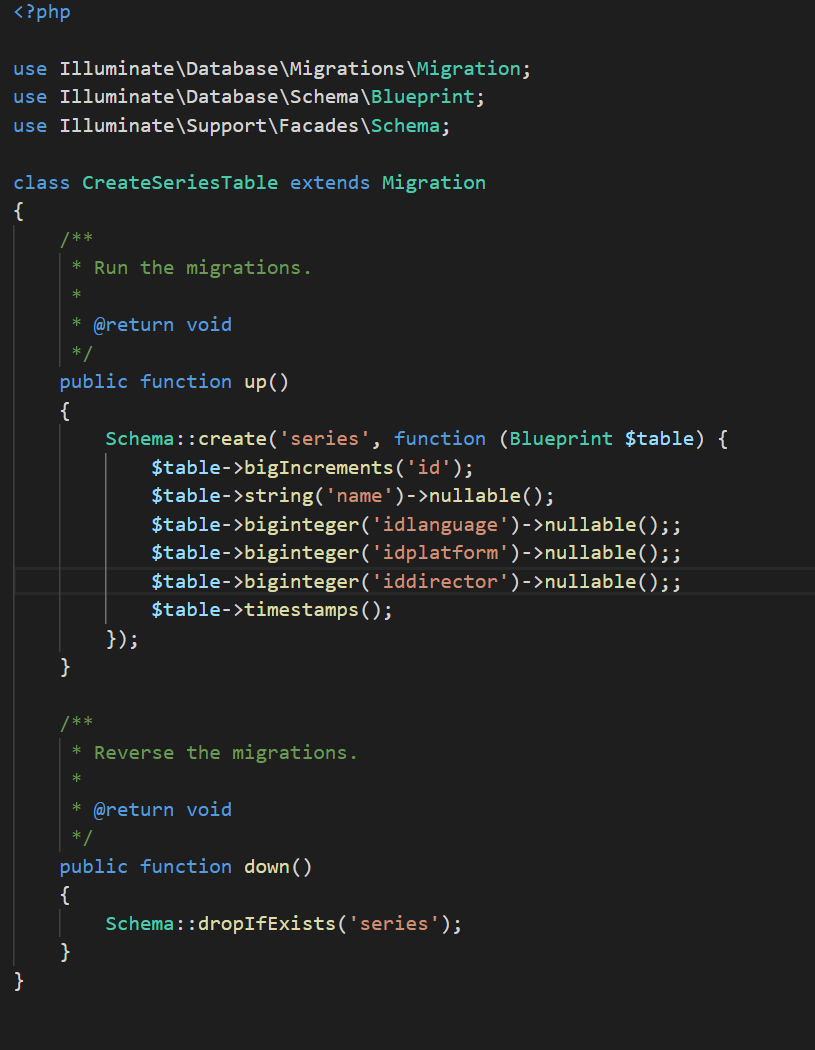


Se crean muchos campos con el modificador ->nullable(); (Permitir valores nulos) para poder realizar un desarrollo más rápido. En producción, muchos campos, no estará permitido ni por el código de la aplicación código ni por las condiciones impuestas a la base de datos.

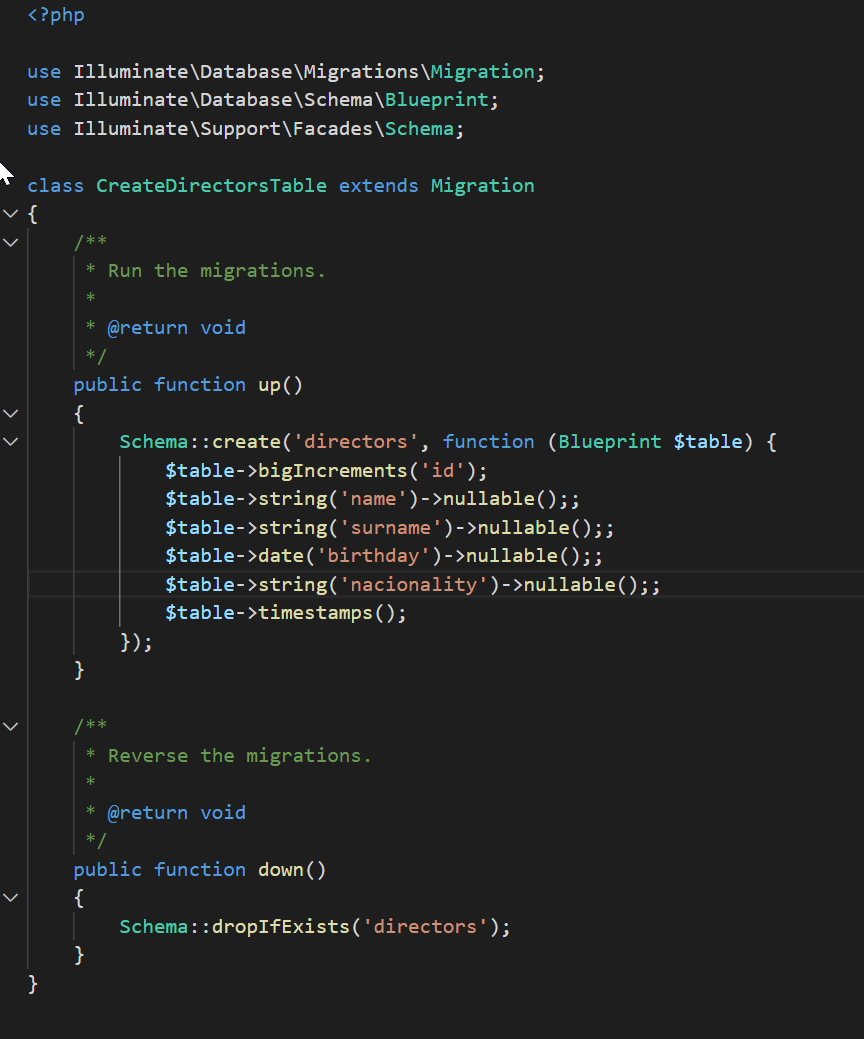
Por igual motivo, tampoco se establecen relaciones entre campos de las tablas. Quedando dicha responsabilidad al principio, en el propio código empleado. Las relaciones quedan establecidas en la Lógica de la propia aplicación. En desarrollos posteriores se pueden definir relaciones en la base de datos y capturar los errores producidos en los borrados y actualizaciones.

Detalle a modo de ejemplo tres de las migraciones creadas

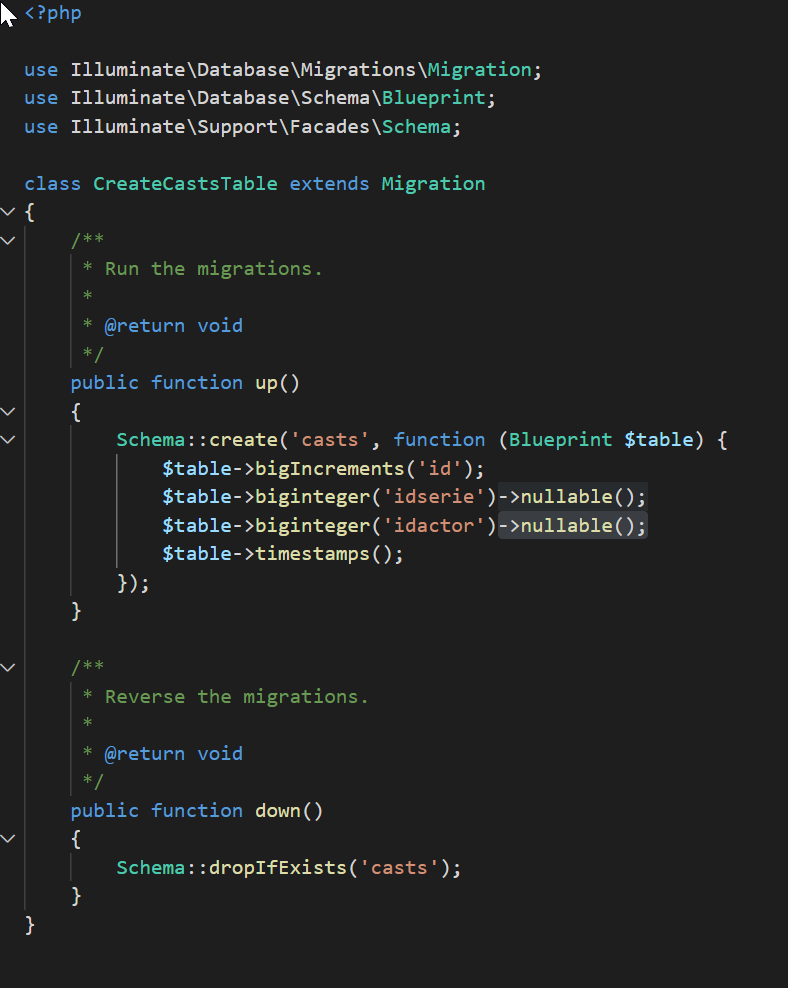
create\_series\_table



create directors\_table



Y create\_casts\_table



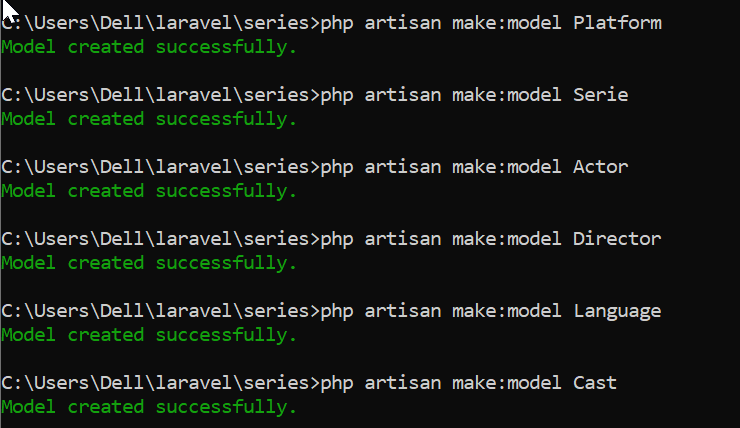
## Creación de los modelos

En la versión 8 se ubican automáticamente en una carpeta llamada Models. Podríamos definir esta ubicación en Laravel 6 y establecerla en el fichero auth.php. No obstante, para evitar problemas inesperados, vamos a dejar que cree los modelos en la carpeta que por defecto establece Laravel 6. Es decir, directamente en la carpeta app

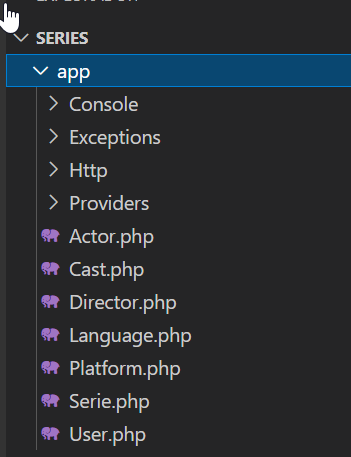


Los modelos, por comodidad, los creamos con el comando de artisan

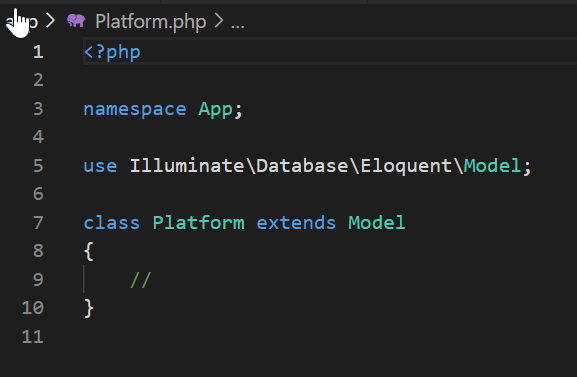
php artisan make:model Nombremodelo



Detalle de los seis modelos creados y su ubicación en el proyecto



Dentro de cada modelo creado, tiene el contenido recién creado es similar a este:



Existe una forma de generar a la vez la migracion y el modelo, pero en nuestro caso no la hemos utilizado. Seria así:

php artisan make:model Nombredelmodelo –-migrate

## Creación de los Controladores:

Su misión fundamental es poder acceder a los modelos y devolverlos a las vistas.

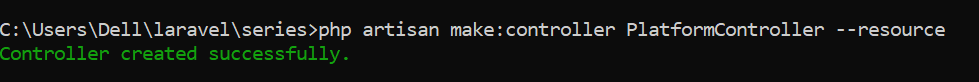
Existen varias formas de crearlos, desde cero con el IDE empleado o con utilizando diversos comandos Artisan de consola.

Se crean dentro de la carpeta app\http\controllers.

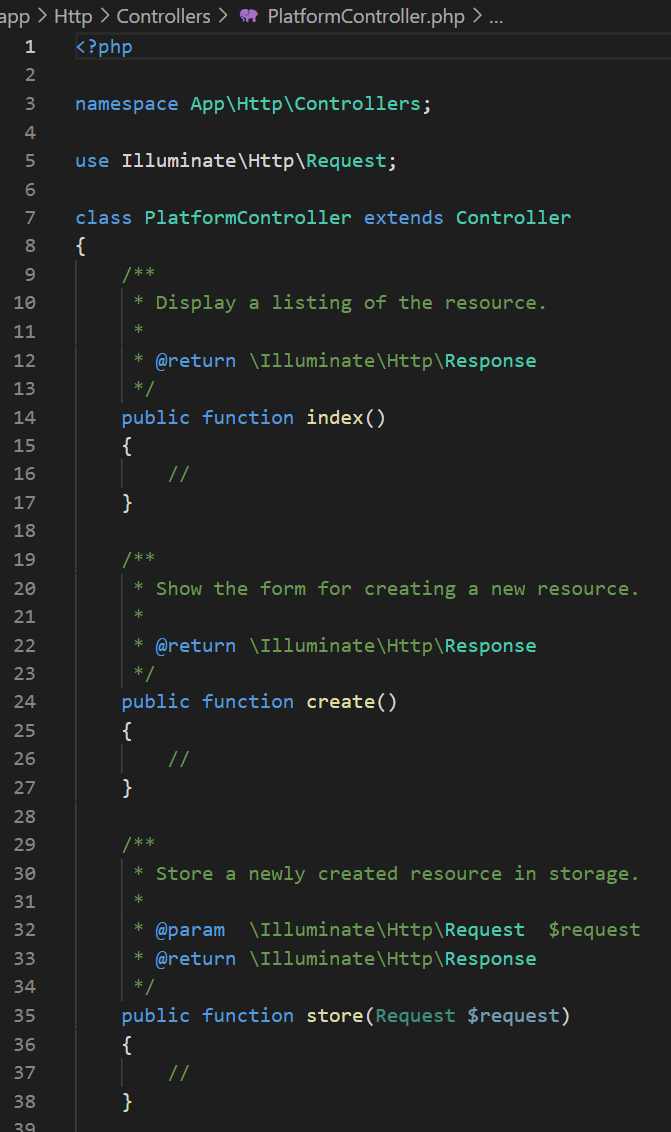
Se pueden organizar de distinta forma agrupando por entidades, por entidades y métodos, etc. En nuestro caso lo he agrupado por entidad. Un controlador para todo lo que se realiza sobre las plataformas, otro para todas las acciones sobre actores, etc.

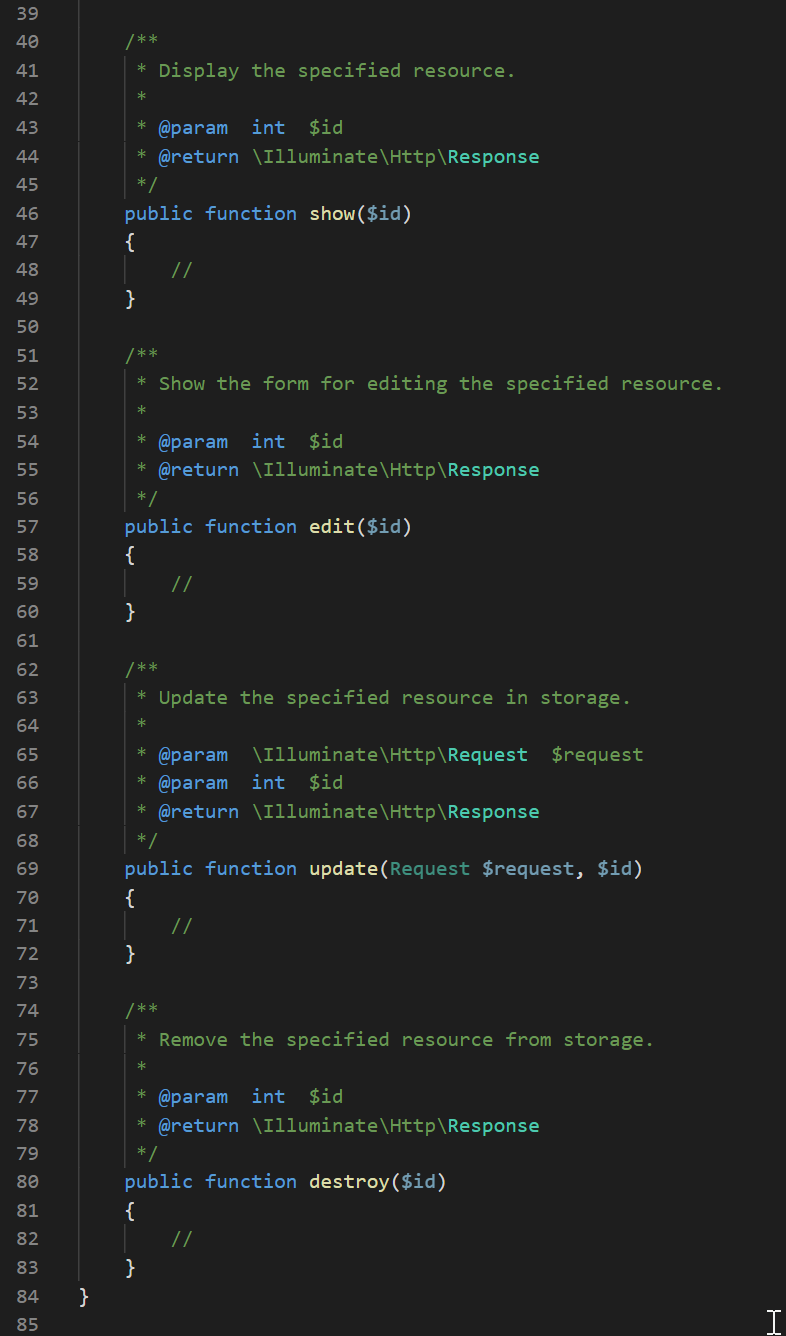
Hemos usado el comando

Php artisan make: controller NombredeControlador –resource ya que implementa todos los métodos básicos CRUD

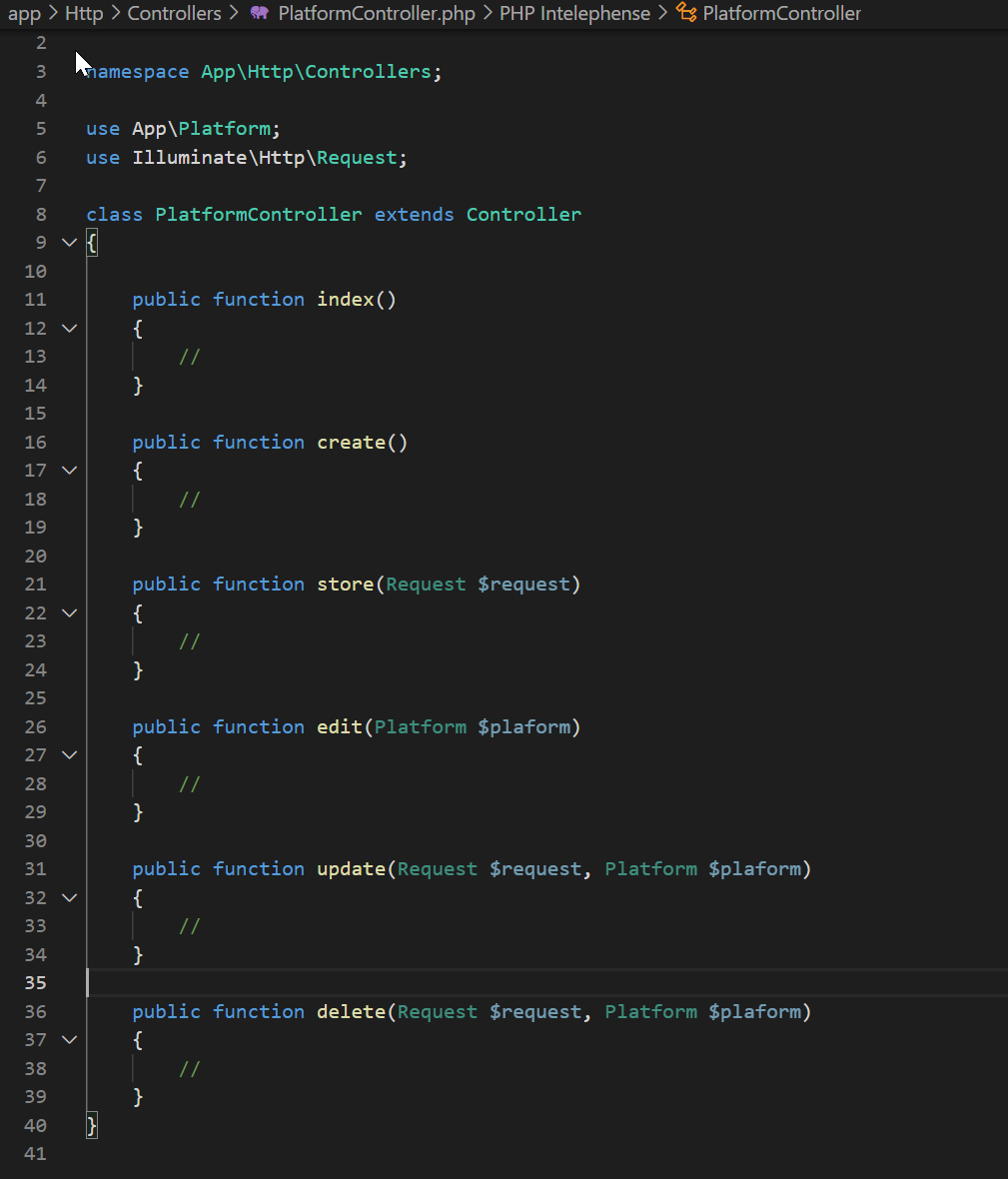


Esto es lo que nos crea:





Eliminando comentarios y estableciendo los métodos del controlador para que actúen de forma similar a la práctica 1, quedaría así.

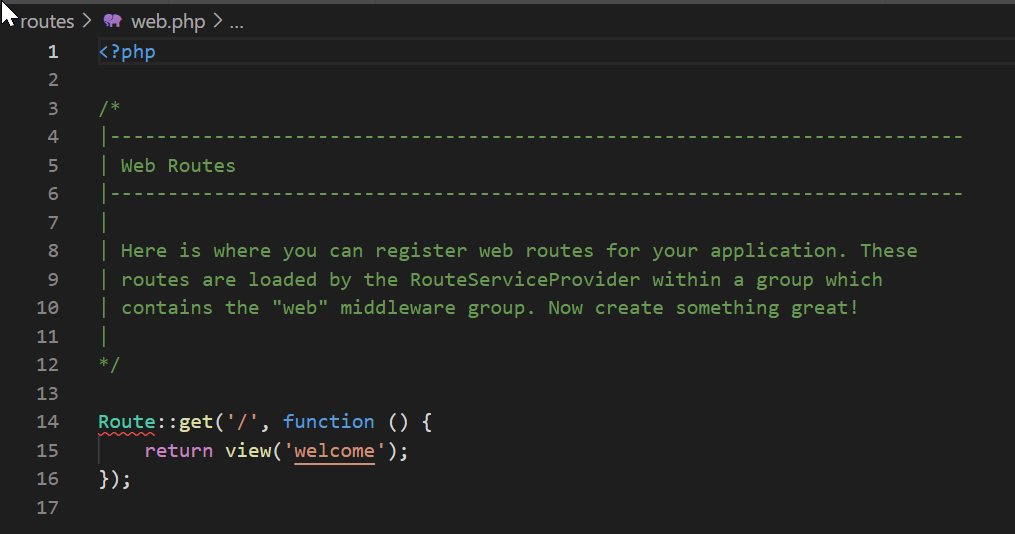


## Creación de las rutas

Las rutas son utilizadas normalmente para devolver una vista o para llamar a un método de un controlador que realizará a su vez acciones con la base de datos y/o con una vista.

El fichero web.php contiene todas las rutas. Al crearse un nuevo proyecto tiene este contenido (Una sola ruta creada):

Por defecto al entrar ”/“ te lleva a /welcome



Añadiremos rutas a lo largo de la actividad para cada uno de los requisitos a implementar

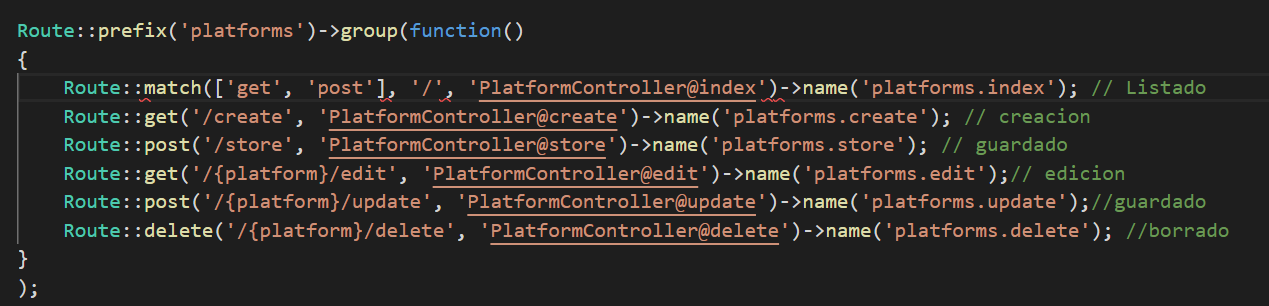
Para cada tabla, agrupamos las distintas rutas (Listar, crear, guardar, editar y borrar) utilizando prefijos.

Ejemplo de las rutas creadas sobre la entidad Platform:

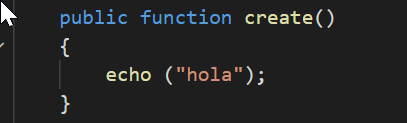
El Método get o post te lleva al controlador PlatformCotroller y dentro de él, al método index. (Si hubiésemos creado controladores autoinvocables no necesitaríamos definir el método puesto que se ejecutaría el definido como \_\_invoke).

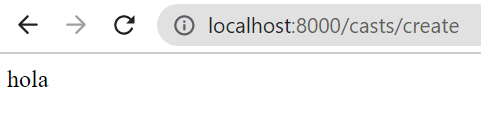
A esta ruta le llamamos platforms.index

La cuarta ruta /{platform}/edit e estoy pasando un parámetro. Igual con update y delete.



Para comprobar si están funcionando bien las rutas realizo una prueba sobre el método create del controlador cast

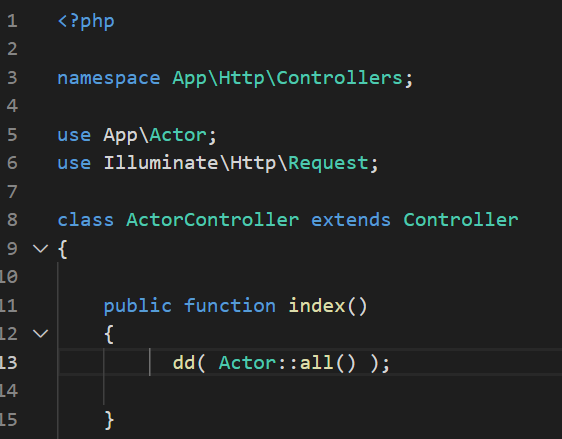




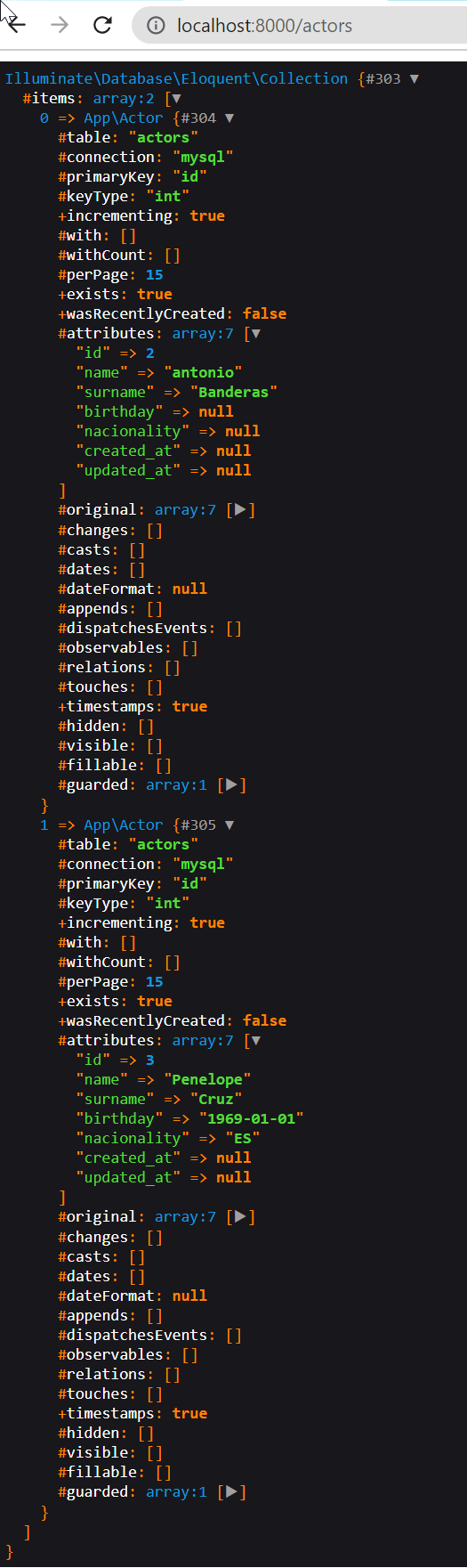
Esto mismo lo realizo con las seis tablas (entidades) del proyecto.

Para comprobar que se accede a los datos de las tablas utilizando el modelo realizo la siguiente prueba : Introduzco dos actores desde phpmyadmin y coloco el siguiente código en el controlador

dd(Actor:all() );



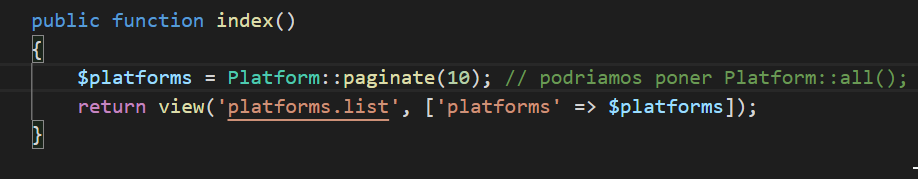
El resultado es satisfactorio porque nos devuelve dos objetos (actores)



## Creación de vistas utilizando Blade

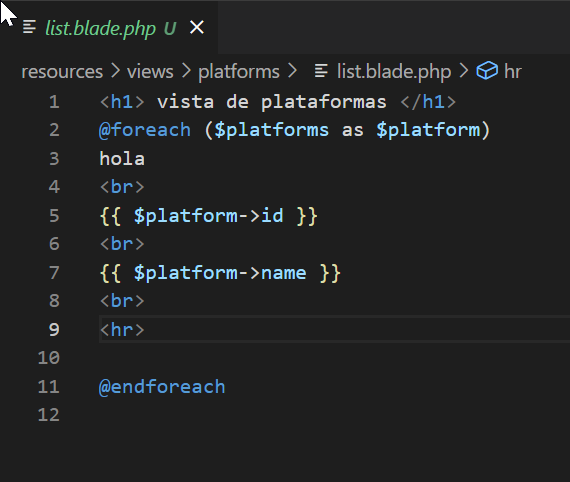
En este caso no se ha utilizado comandos php artisan para que las cree automáticamente

El controlador de plataformas tiene el siguiente método index():



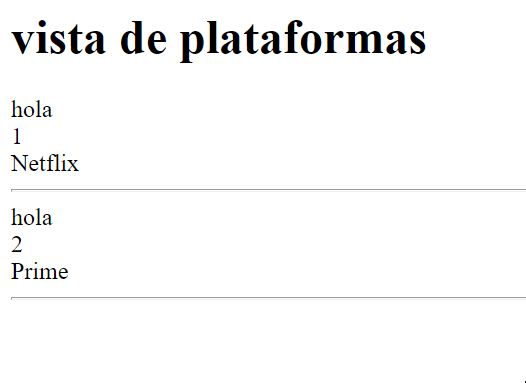
Dentro de resources, vistas, creo una carpeta para las vistas de cada tabla.

En dicha carpeta creo el fichero list.blade.php con el siguiente código:



Tras introducir desde phpmyadmin un par de plataformas pruebo a ver si la vista me devuelve los registros.

El resultado es satisfactorio



Llegados a este punto, vemos que todo funciona correctamente, modelos, controladores, rutas, vistas.

## Autentificación

Hemos usado la autentificación que implementa por defecto Laravel por medio del paquete laravel/ui.

Podríamos haberlo realizado al comienzo, junto con la creación del proyecto desde cero con el comando:

Laravel new nombredeproyecto - - auth

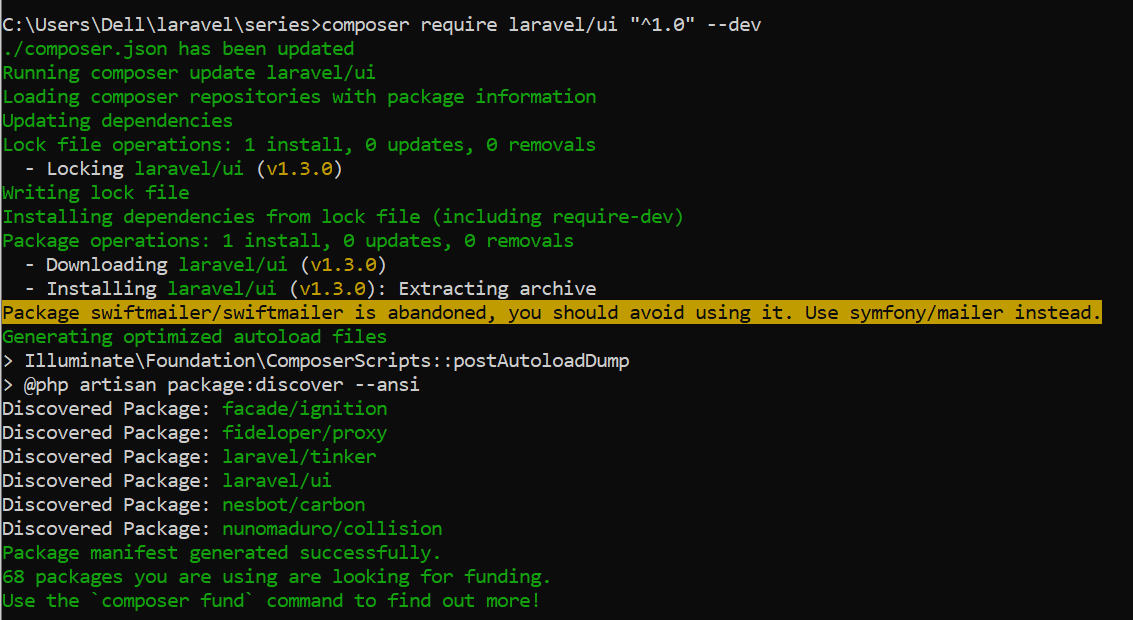
En nuestro caso, con el proyecto ya creado, ejecutamos dos comandos:

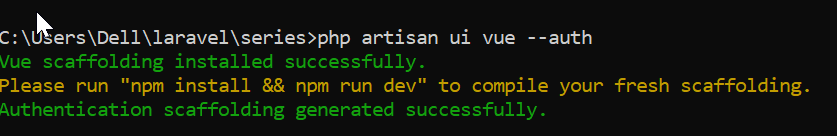
composer require laravel/ui ”^1.0” -- dev

Y a continuación el comando

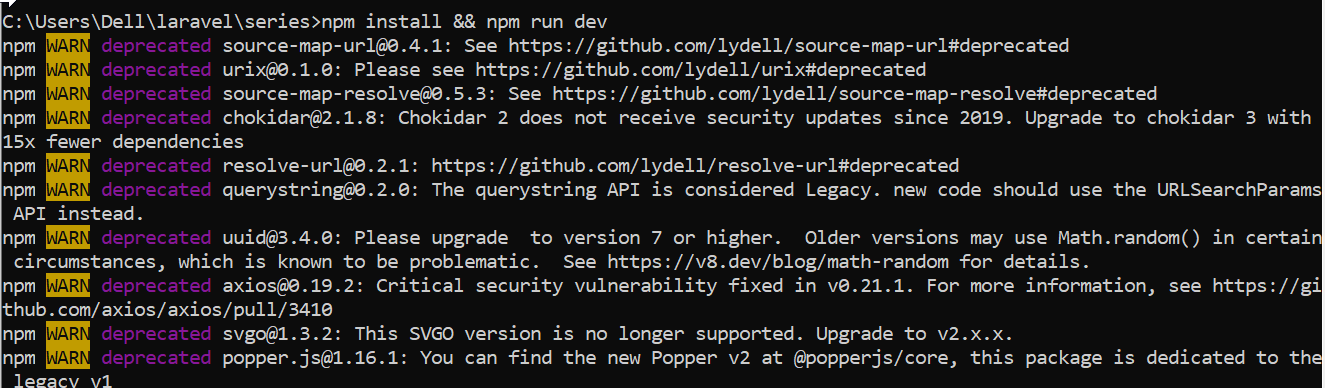
php artisan ui vue – auth.

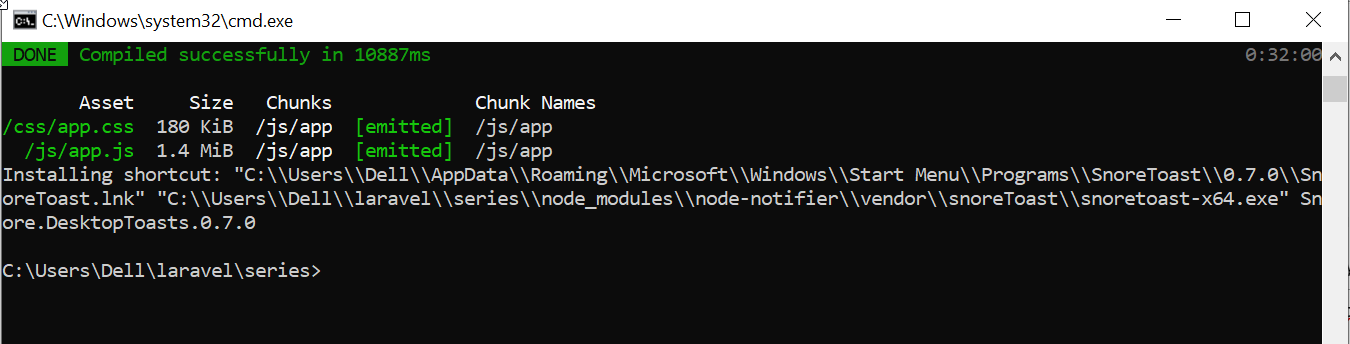
Detenemos primeramente el servidor para evitar posibles problemas



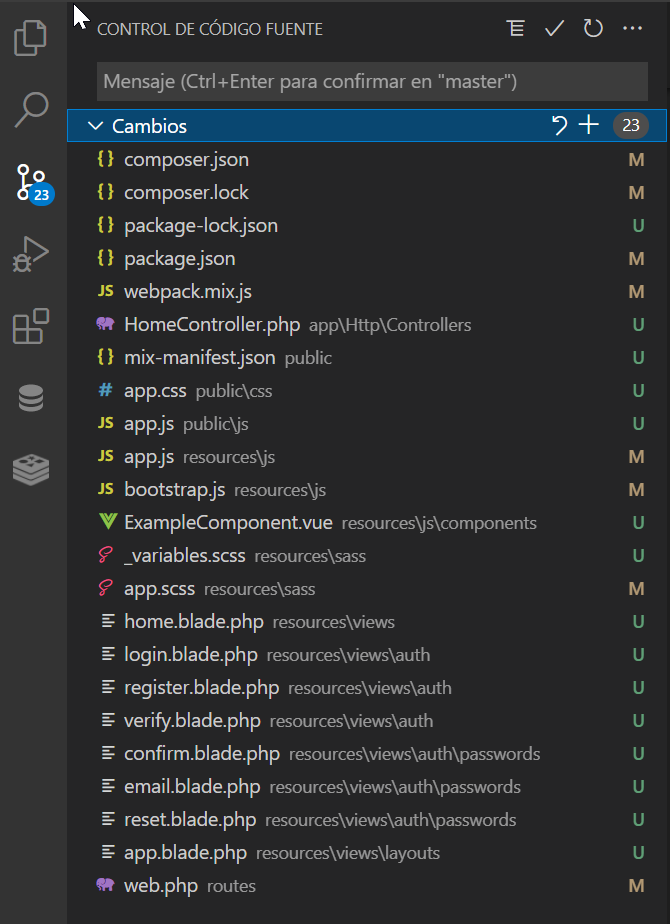


Nos solicita que ejecutemos npm install && npm run dev y así se ha procedido

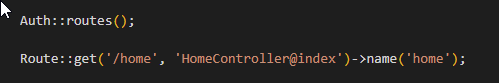




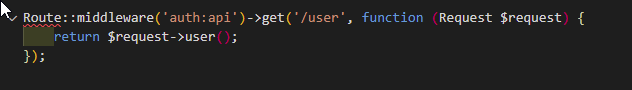
A través de GIT podemos ver todo lo que ha cambiado en el proyecto al añadir la autentificación.



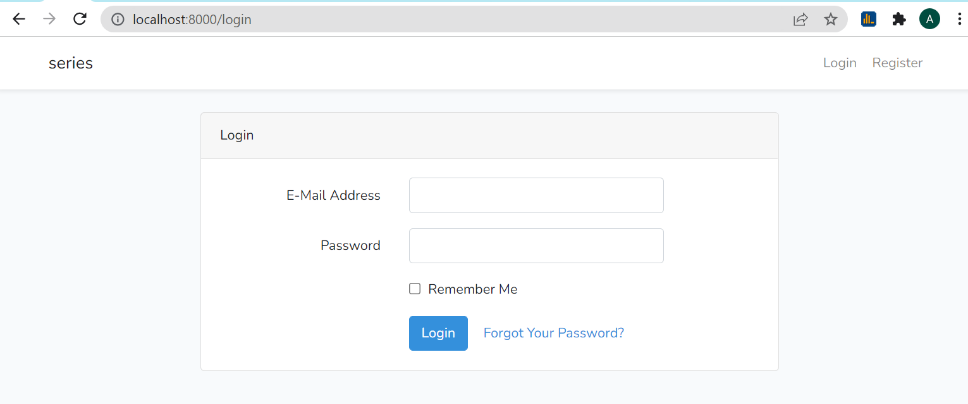
Entre otras modificaciones, ha creado vistas y ha añadido nueva ruta al fichero web.php



Y en el fichero de rutas api.php ya estaba creado



Pruebo la nueva vista de autentificación: localhost:8000/home



Pruebo a registrarme con éxito

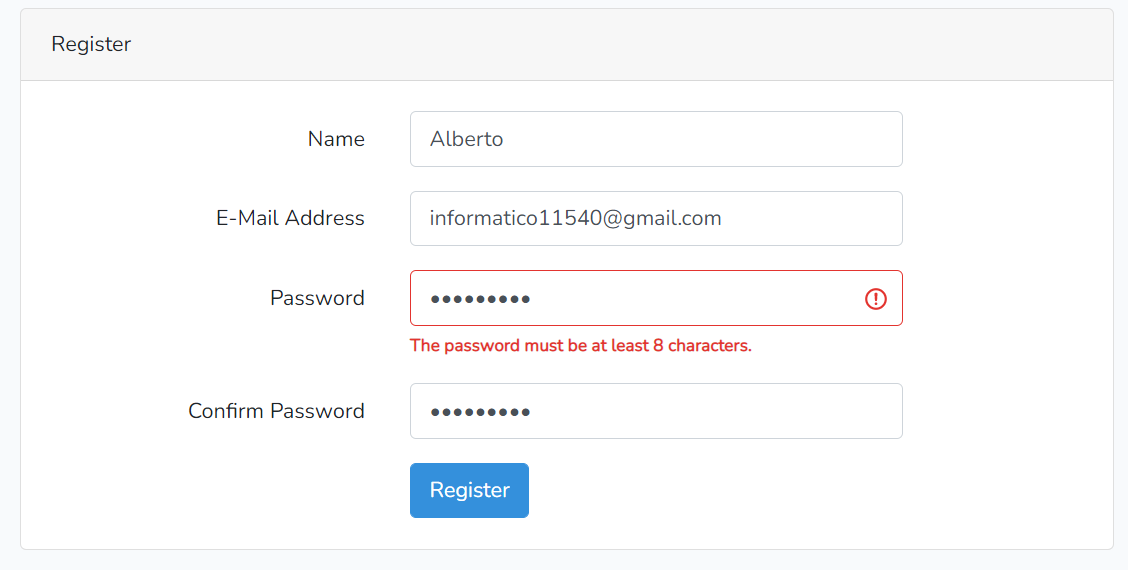
Los datos de prueba con los que me registro son:

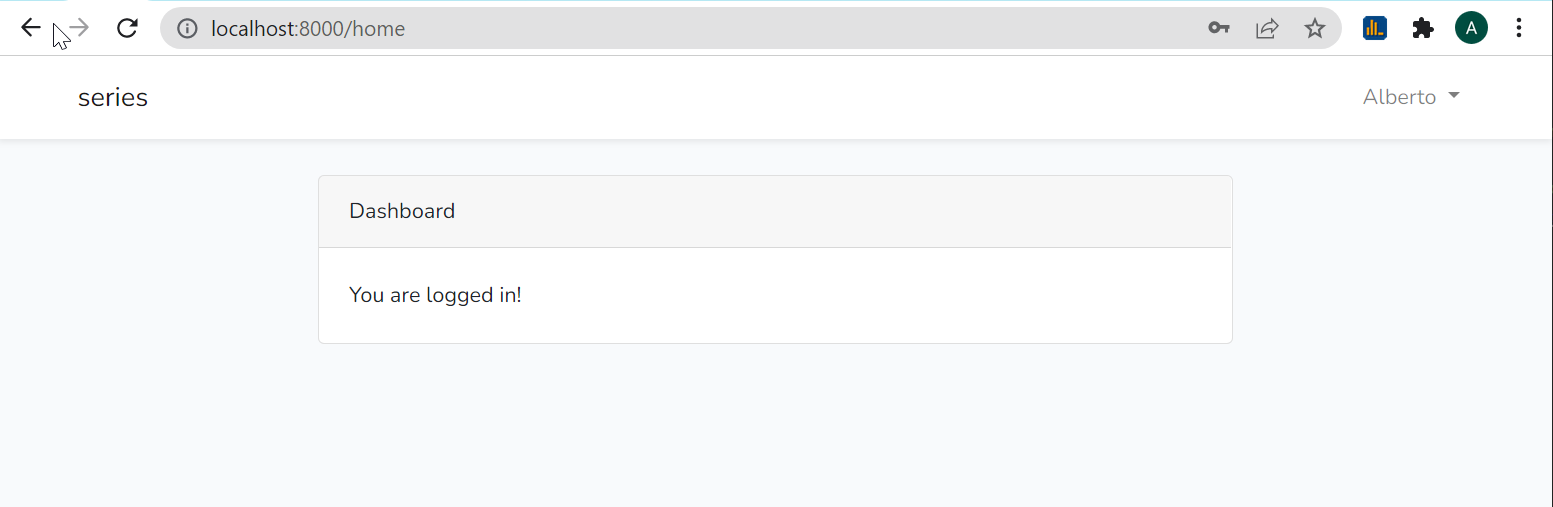
Nombre: no lo usa en la validación

Email: correo informatico11540@gmail.com

Contraseña: abcdefghi

Estos datos se introducen utilizando un seeder



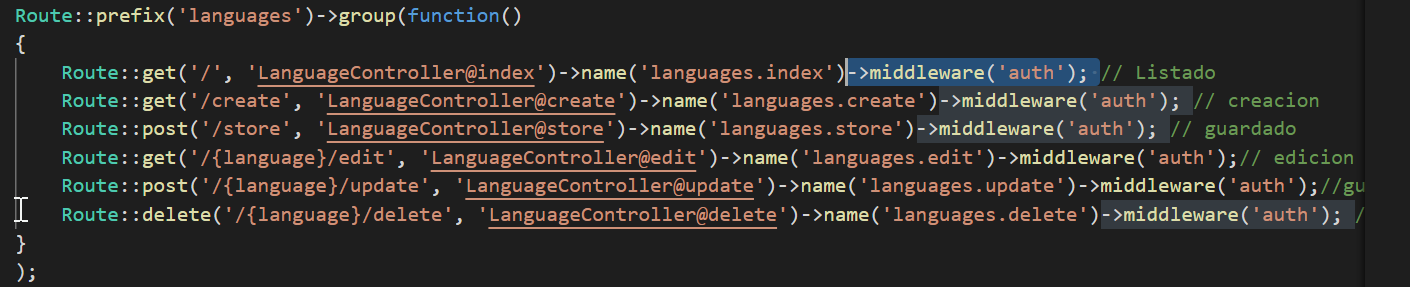


A través de Phpmyadmin comprobamos que no guarda la contraseña sino el hash de la contraseña.



Como requisito se establece bloquear el acceso a todas las entidades menos la de plataformas. Esto lo realizamos a través del enrutado modificando el fichero web.php

Se añade ->middleware('auth'); a las rutas en las que tenemos que estar autentificados

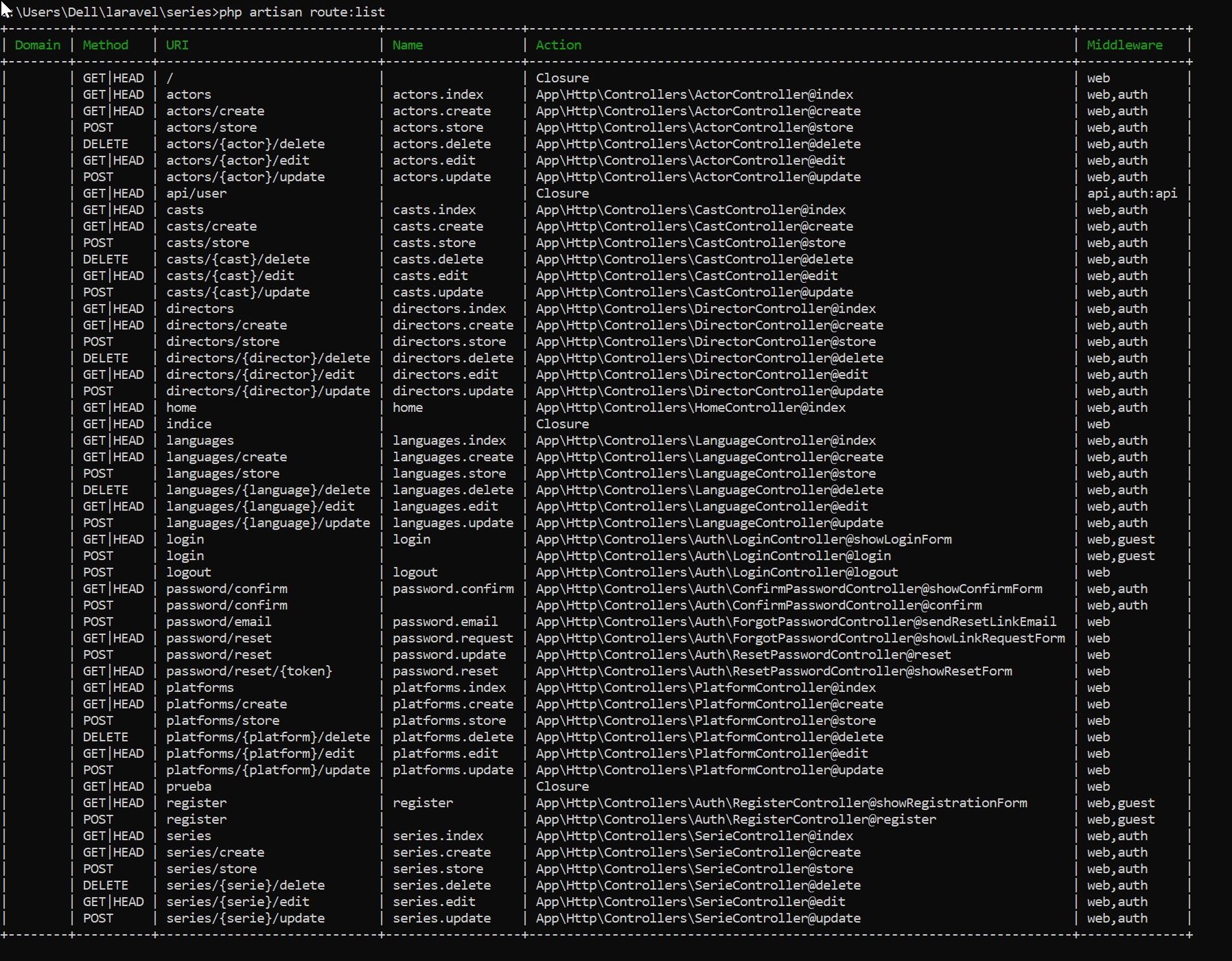


Muchos parámetros relacionados con la autentificación pueden modificarse en el fichero

Auth.php dentro de la carpeta app/config

A través de un comando puedo ver también un resumen de todas las rutas establecidas hasta el

Momento.



## Seeder -” sembradores ” de datos.

Los seeder se utilizan para crear datos de prueba. Se crean distintos seeder

ActorsTableSeeder

DirectorsTableSeeder

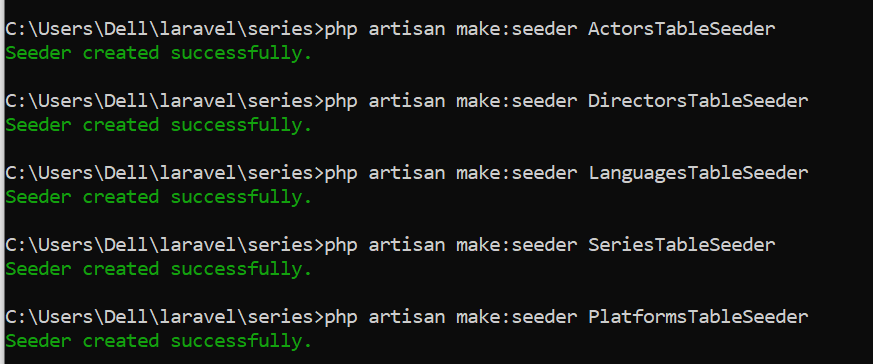
LanguagesTableSeeder

PlatformsTableSeeder

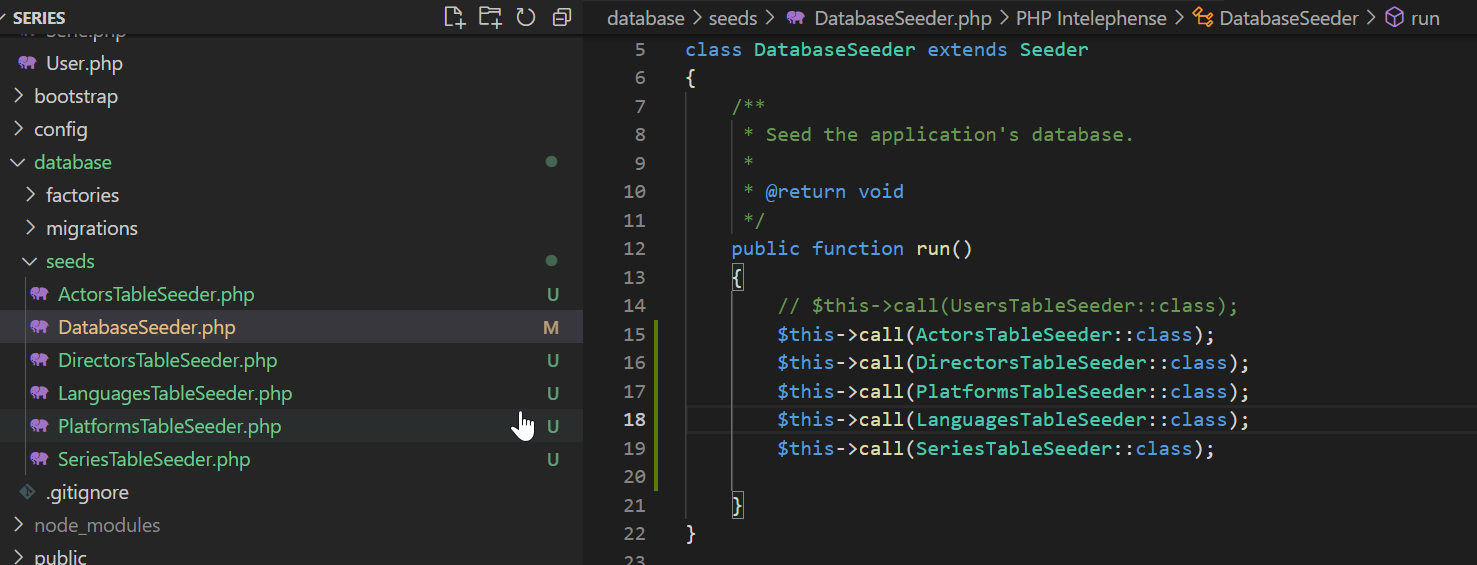
SeriesTableSeeder

UsuariopordefectoSeeder

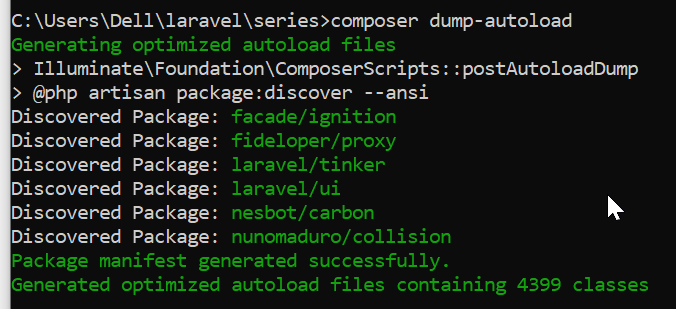
php artisan make:seeder nombredelseeder



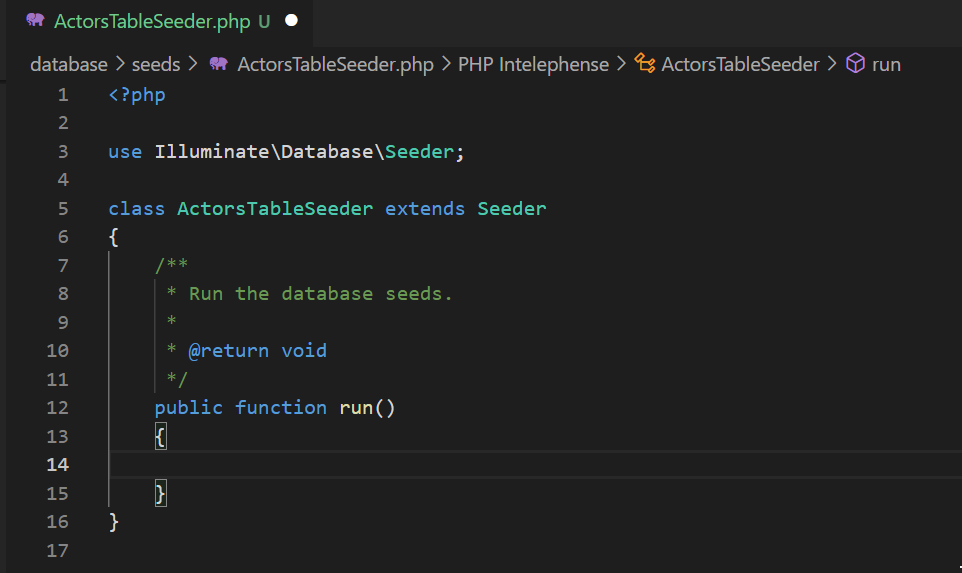
También hay que modificar el fichero DatabaseSeeder.php para añadir los seeder que se ejecutaran



También es necesario ejecutar composer dump-autoload (paro el servidor primero)



Así es como los seeder quedan generados:

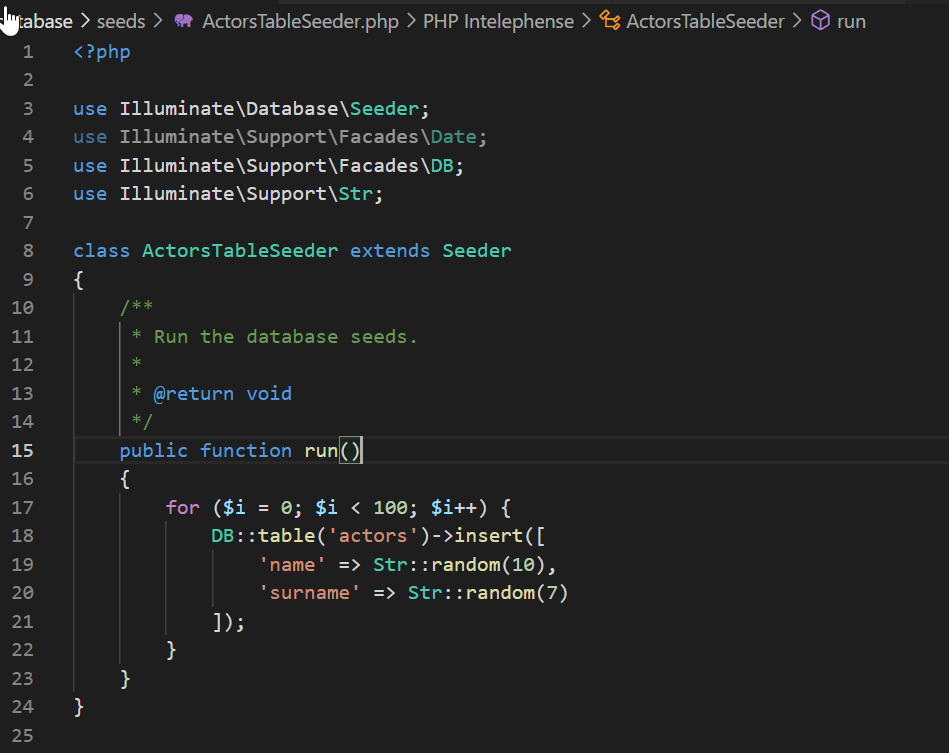


Si queremos que genere 100 actores, este es el código que se utiliza:

(hay que añadir las librerías)

Illuminate\Support\Facades\DB;

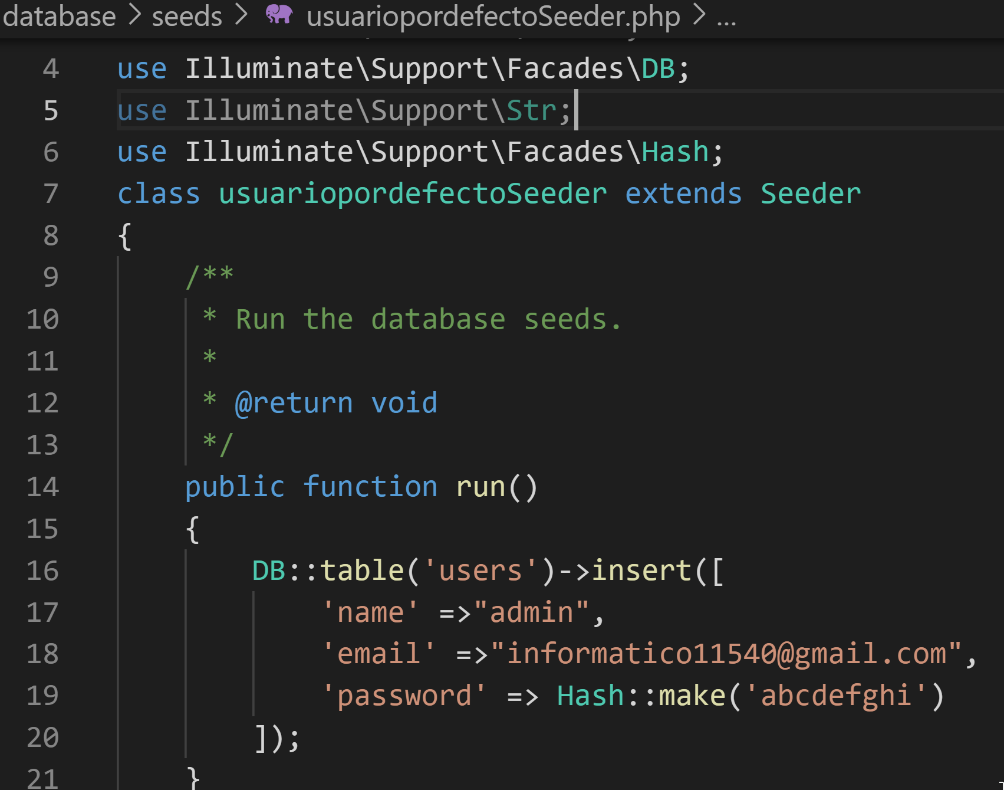
Illuminate\Support\Str



Tambien creamos un Seeder para crear el usuario por defecto para pruebas:

admin, [informatico11540@gmail.com](mailto:informatico11540@gmail.com), contraseña:abcdefghi (guardo su hash)





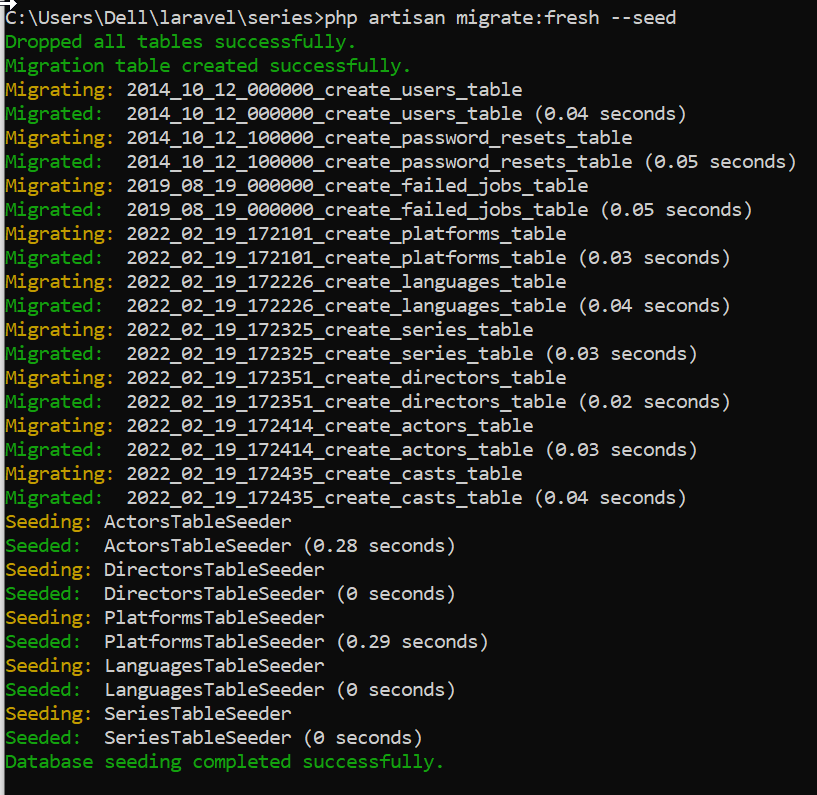
Ahora voy especificando los campos de cada tabla a rellenar

Después ejecuto el comando:

php artisan migrate:fresh - - seed

Este comando elimina los datos y tablas y los vuelve a crear con el seeder.

Es un comando que he usado en muchas ocasiones. Por ejemplo, al realizar cambios en datos o estructura



## Paginación

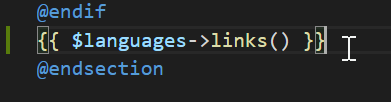
Se realiza en varios pasos:

Primeramente, en el controlador se recuperan los registros en bloques determinados en la constante PAGINATE\_SIZE, por ejemplo, de diez en diez 

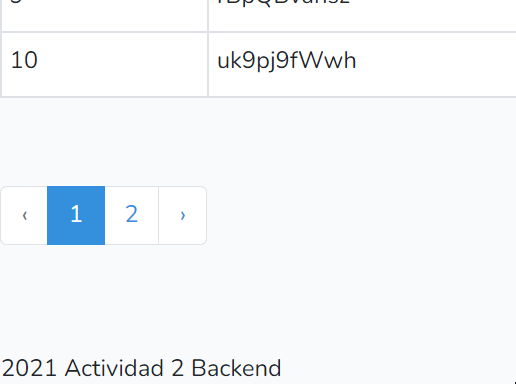
A continuación, el controlador llama a la vista y a la vez que la llama, le pasa el nombre de la variable.



A continuación en la vista hay que poner el siguiente código:



Y ya está funcionando como podemos ver la barra inferior correctamente

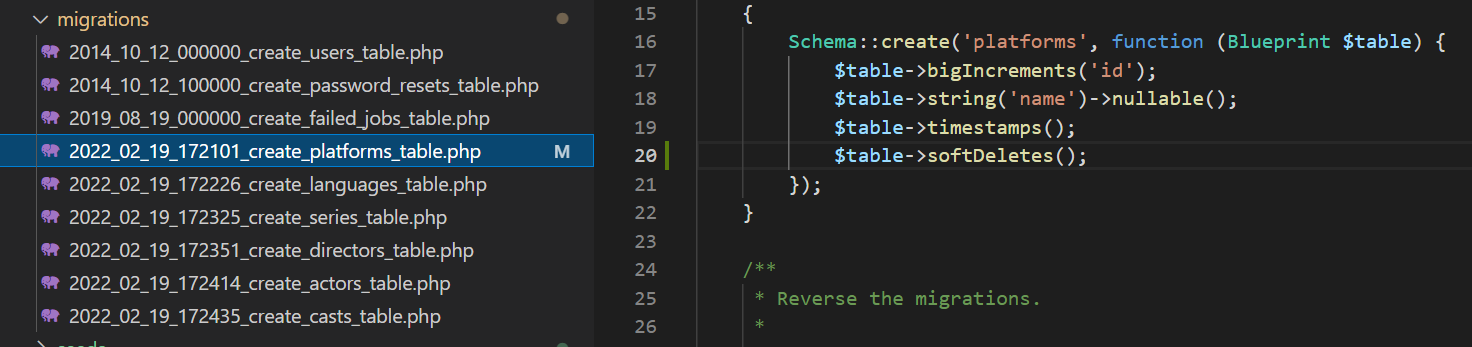


## SoftDeletes

Es de gran utilidad. Permite una mejor gestión de los borrados no permanentes. Se ha implementado en todas las tablas. Se requieren dos pasos:

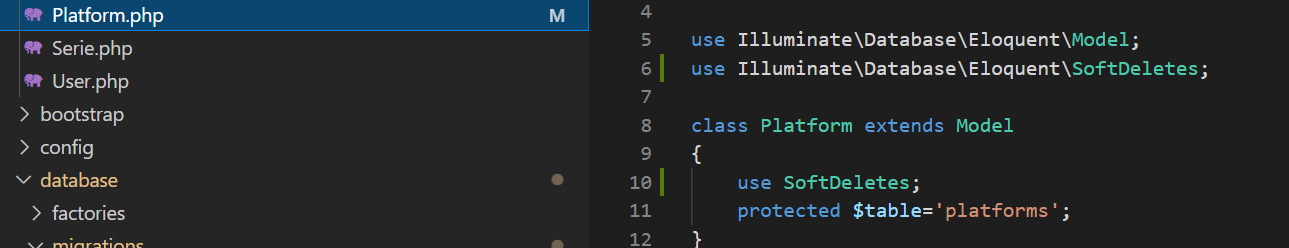
1.- Creación del campo en la migracion

$table->softDeletes()



2.- Modificación del modelo de cada tabla, añadiendo” use Softdeletes“ y la librería:

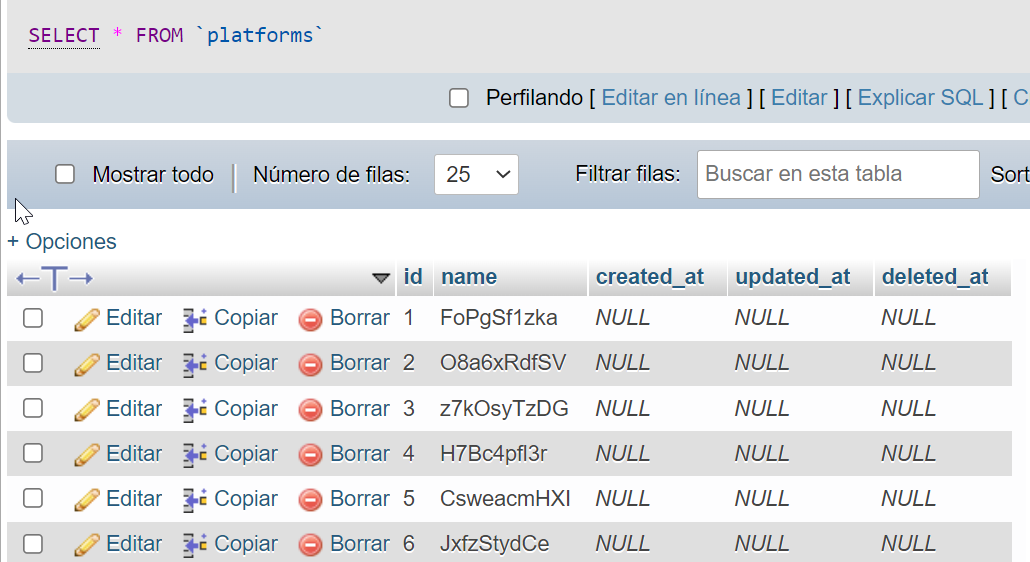
”use Illuminate\Database\Eloquent\SoftDeletes;”



Después vuelvo a ejecutar el comando:

php artisan migrate:fresh - - seed

Y compruebo el resultado satisfactorio en phpmyadmin:



## Uso del Layout

Se ha usado la herencia de plantillas de blade.

Se consigue de forma fácilmente tener un aspecto homogéneo en toda la aplicación.

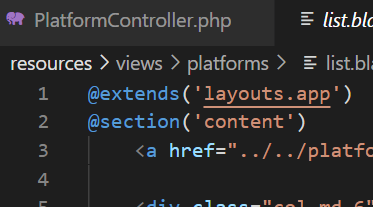
Una cabecera, un footer común y además poder usar Bootstrap fácilmente sin tener que poner enlaces de Bootstrap en cada pagina

@extends(’layouts.app’)

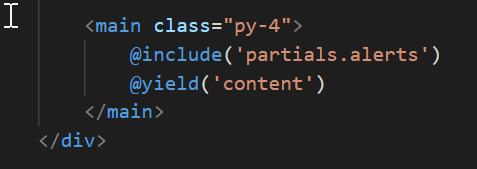
@section(’content’)

…

@endsection



También incluyo un @include (’partials.alerts’) arriba del content para mostrar los distintos mensajes



# Código empleado en los controladores ( Platfom, Language, Actor, Director, Serie, Cast)

Cada entidad tiene su vista, su controlador y su modelo.

La estructura es similar a empleada en la actividad 1.

Las diferencias son:

Se incorpora una función de búsqueda por nombre.

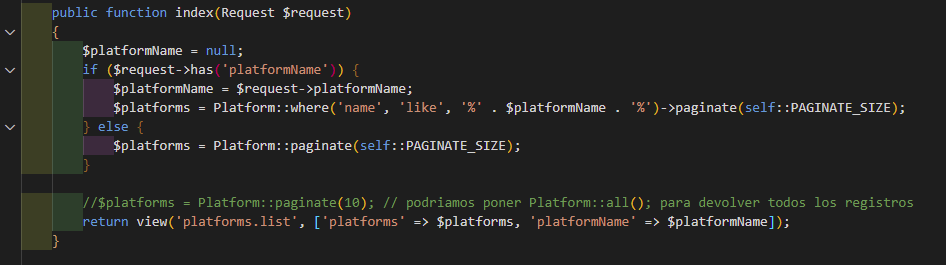
Se junta en una misma vista la función de creación y actualización

El borrado se hace directamente en el controlador sin necesidad de usar una vista.

En cada controlador se definen los siguientes métodos:

* Index() para listar todas las plataformas. Posteriormente se cambia para poder hacer búsquedas
* Create ()
* Store (Request $request)
* Edit(Platform $platform) nos indican la plataforma a modificar
* Update (Request $request, Platform $platform)
* Delete (Platform $platform) Recibe la plataforma a borrar

### Método Index



Explicación:

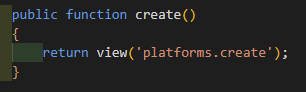
Si el parámetro con el que se llama este relleno con caracteres de búsqueda, entonces utilizando el modelo Platform recupera todas las plataformas cuyo nombre contenga esos caracteres (like% ..%).

Si no recibe nada, entonces recupera todas las plataformas.

En ambos casos recupera paginando con el número de registros establecido en la constante PAGINATE\_SIZE.

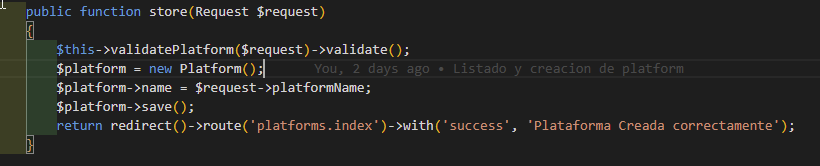
Por último, llama a la vista a través del nombre definido en las rutas del fichero de rutas web.php y le pasa como parámetro el array de plataformas para que puedan ser visualizadas.

### Método Create



Este método solo llama a la vista

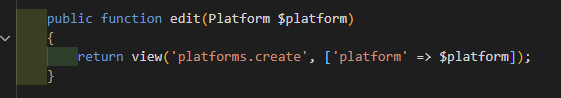
### Método Store



Recibe como parámetro la plataforma a crear. Crea un nuevo objeto a partir del modelo y le asigna uno a uno los atributos. Después lo almacena.

Por último, llama a la vista de listado pasándole un mensaje a la vista en forma de variable.

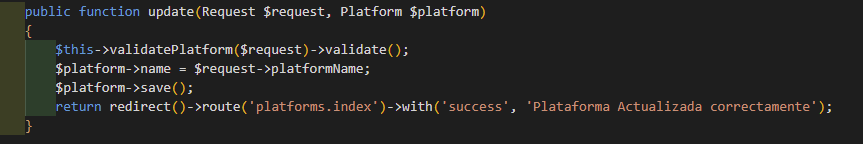
### Método Edit



Llama a la vista definida en el fichero de rutas como plataforma.create.

Le pasa como parámetro la plataforma recibida.

### Método Update



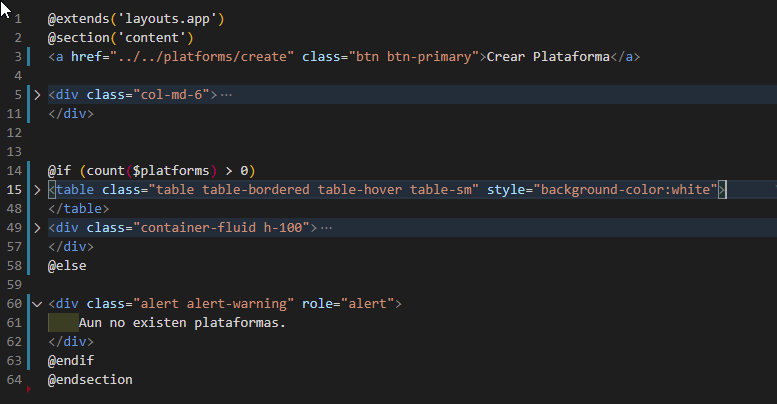
Recibe como parámetros el array request con los nuevos datos y el objeto plataformas.

Actualizo el nombre y guardo.

Por último, llamo al listado de plataformas y le paso la variable con el mensaje de operación realizada con éxito.

### Vista de Listar (y buscar)

Colapsando el código se aprecia mejor las distintas partes de la vista:



Se utiliza la herencia de layouts.app

Se incluye todo el código dentro de la sección content de la plantilla layouts.app.

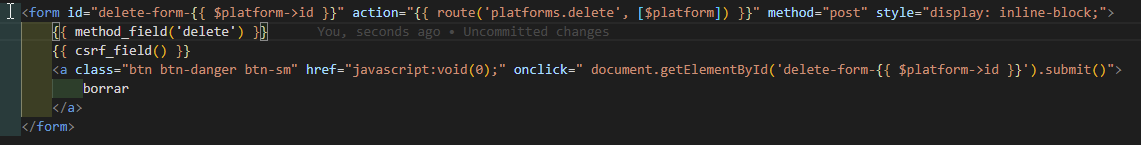
Otra característica es que en función de si el campo de búsqueda este relleno o no,

Se envía le formulario con datos o vacío. búsqueda o listado 

Contiene la llamada a edición.



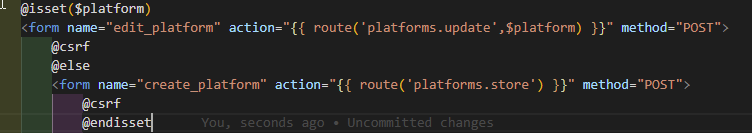
Llamada al borrado



Hay que especificarlo expresamente al no ser ni post ni get {{ method\_field('delete') }}

Hay que poner el token para verificar que el usuario autenticado es quien está haciendo la petición a la aplicación. Si no se pone da error 419. {{ csrf\_field() }}

### Vista de Crear Actualizar



Según se reciban o no datos dentro de la variable $platform, se establece la propiedad action para que llame a la actualización o a la creación de plataformas.

* Tiene datos. -> Actualizacion (update)
* No tiene datos -> Creacion (store)

# 5. Conclusiones

A través de esta Actividad se ha puesto en práctica muchos de los conocimientos teóricos aprendidos en la asignatura sobre Laravel.

Se ha realizado sobre la base de los mismos requisitos funcionales de la Actividad 1 y por ello, se ha podido comparar las diferencias de realizar el proyecto con y sin framework.

El patrón Modelo-Vista-Controlador se emplea de forma distinta. En la Actividad 1, las vistas llamaban al controlador que a su vez llamaban al modelo. En esta Actividad 2, con Laravel, es distinto. Las llamadas, aunque también pueden realizarse a las vistas, normalmente se realizan al controlador, el cual, tras utilizar el modelo, llama a las vistas pasándole a la vez los datos oportunos. Es definitiva, con Laravel, el controlador es la piedra angular de la aplicación.

La utilización del framework en general me ha parecido muy interesante por todo lo que aporta al desarrollo. Ha facilitado muchísimo la creación y conexión a la base de datos a través de los ficheros de configuración .env, database.php, las migraciones y los seeders. El mecanismo de rutas de Laravel me parece fácil y útil a la vez que muy potente. La autentificación que crea a través de un middleware es muy trasparente y completa.

Se ha utilizado mucho los comandos de artisan para experimentar y crear todo lo posible: modelos, controladores, etc. (Las vistas no se puede)

He usado decenas de veces el comando superútil que regenera la información y tablas a partir de migraciones y seeder.

El código con Laravel es mucho más legible que utilizando únicamente PHP. La interpolación que aporta Blade es muy útil ya que permite no tener que estar todo el tiempo saltando de código HTML a PHP y viceversa como ocurría en la Actividad 1.

El uso del Layout también me ha parecido muy útil añadiendo facilidad en el desarrollo mantenimiento y aportando homogeneidad al proyecto.

Me ha parecido también muy útil el orden que aporta el framework al desarrollo.

La integración con el IDE Visual Studio es muy buena y tras instalar extensiones, el autocompletado ha funcionado muy bien.

Se ha realizado labor de investigación de desarrollo haciendo llamadas de métodos a métodos dentro del mismo controlador, paso de información de controladores a vistas, ejecución de php dentro de las vistas, y realizando todo tipo de generación de consultas con mayor o menor éxito debido al tiempo limitado.

Los posibles desarrollos futuros pueden ser: Una mejor presentación de las vistas, elaboración de artefactos para mostrar mensajes y ampliación de los sistemas de búsqueda de datos.

# 5. Bibliografía

Documentación Oficial de Laravel. Recuperado de <https://laravel.com/docs/6.x> con fecha 18 de febrero de 2022

Documentación de Laravel 6 en español. recuperado de: <https://styde.net/documentacion-de-laravel-6/> con fecha 18 de febrero de 2022

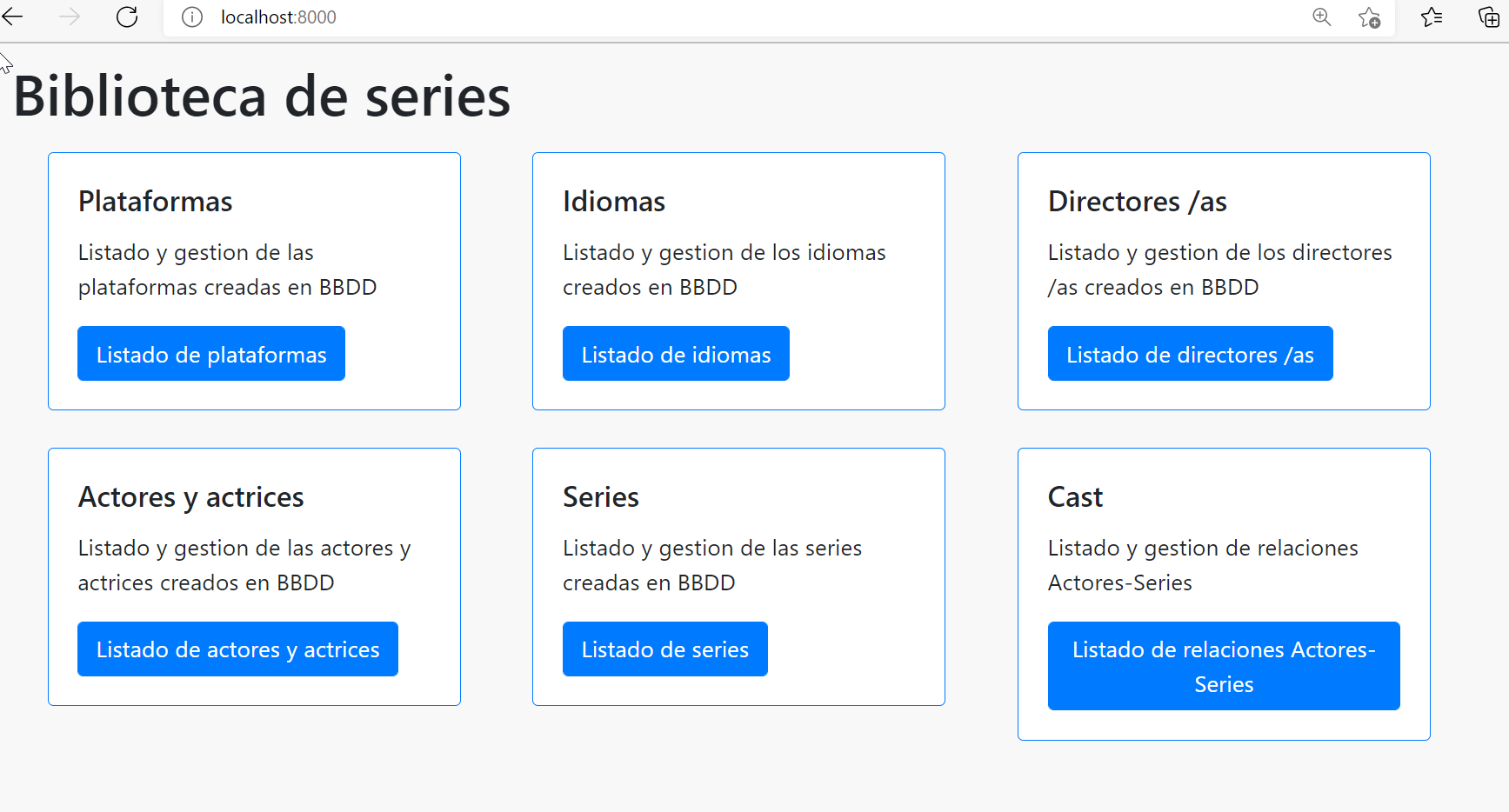
Apuntes de la Asignatura Desarrollo de Aplicaciones WEB I, lado del servidor (Backend) 2022. Universidad Internacional de Valencia.

Cabezas Granado, L.M. et González Lozano, F.J. Cursos de PHP 8 y MySql 8. Anaya Multimedia.

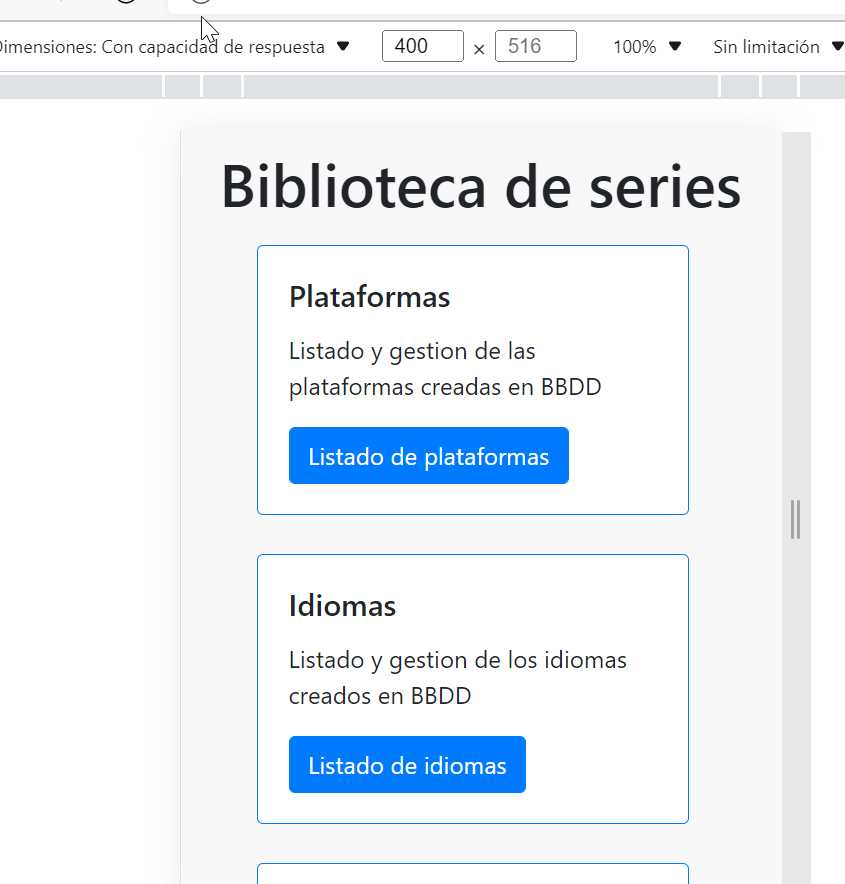
# 6. Anexos

# Anexo1: Vistas de la Aplicación

### Pantalla inicial



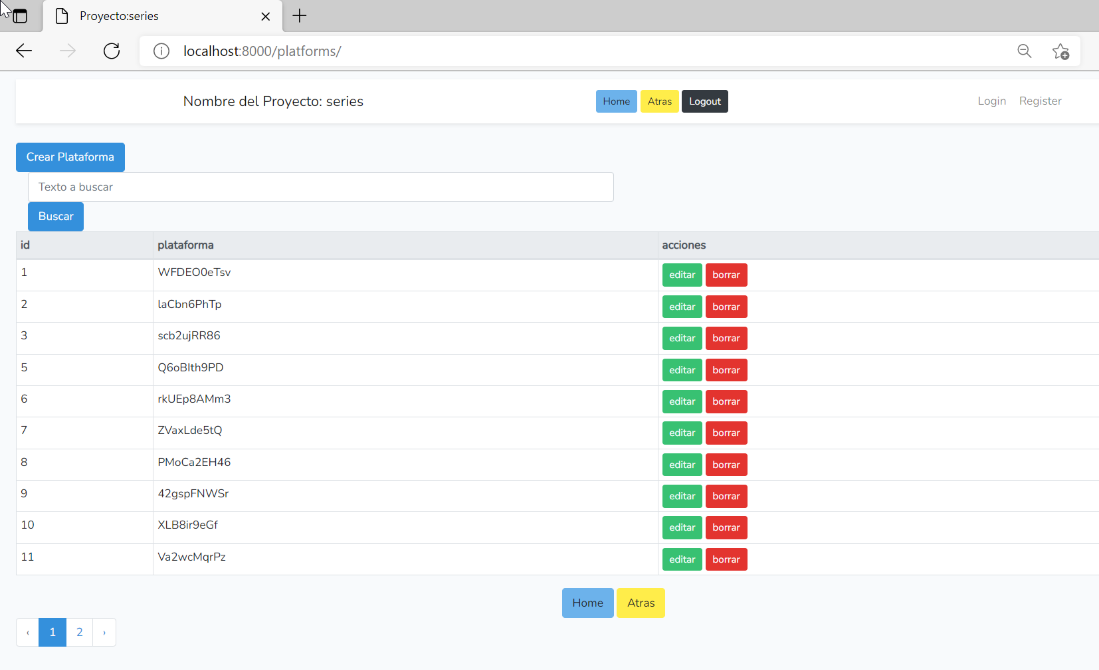
Prueba del diseño Responsive



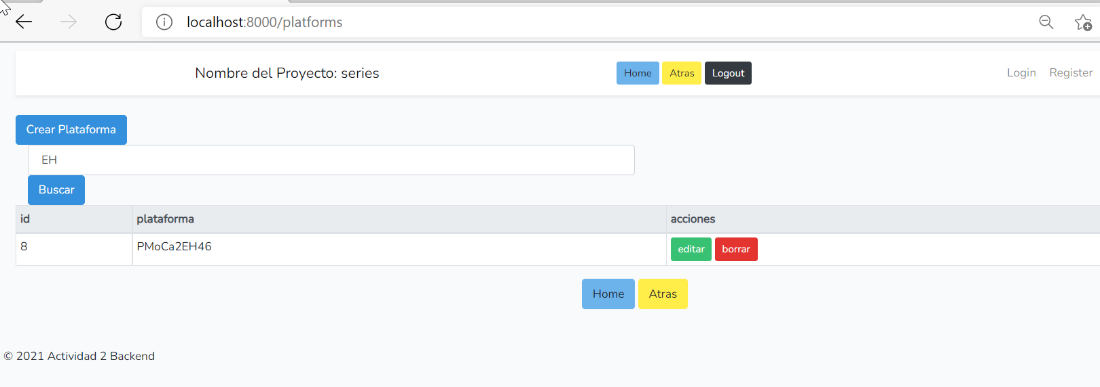
### Creación, Listado, Edición y Borrado de Plataformas

Entidad Plataformas. Se entra sin necesidad de identificación

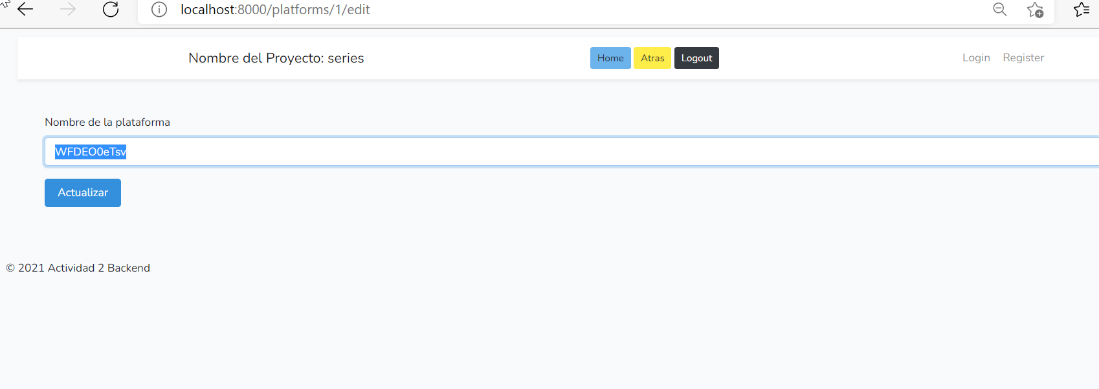
Listado



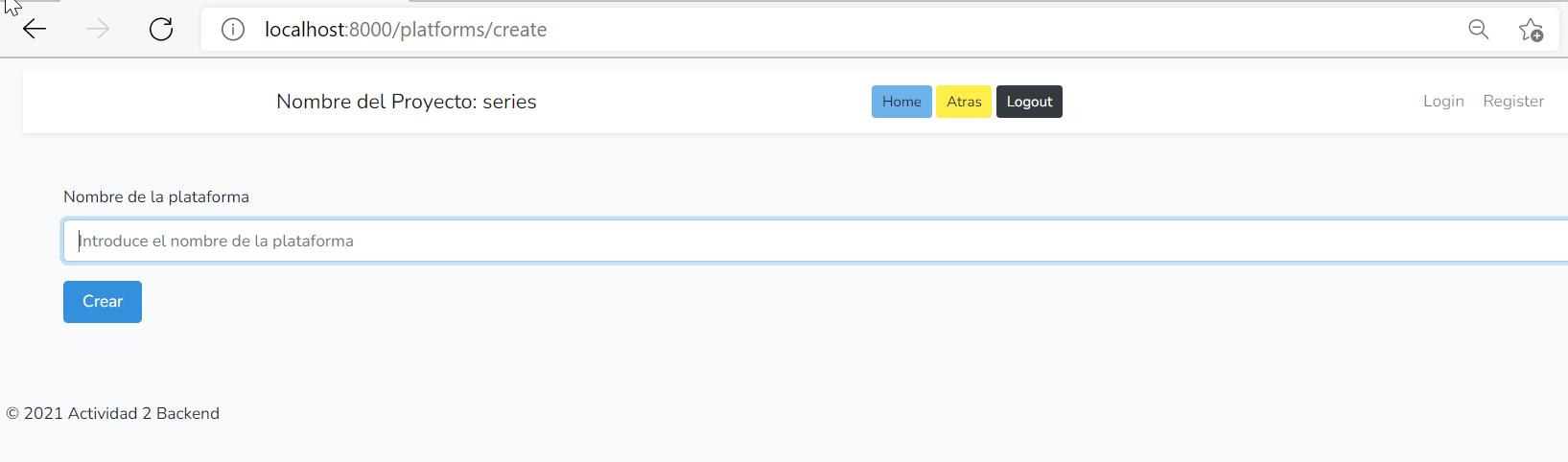
Búsqueda de Plataformas



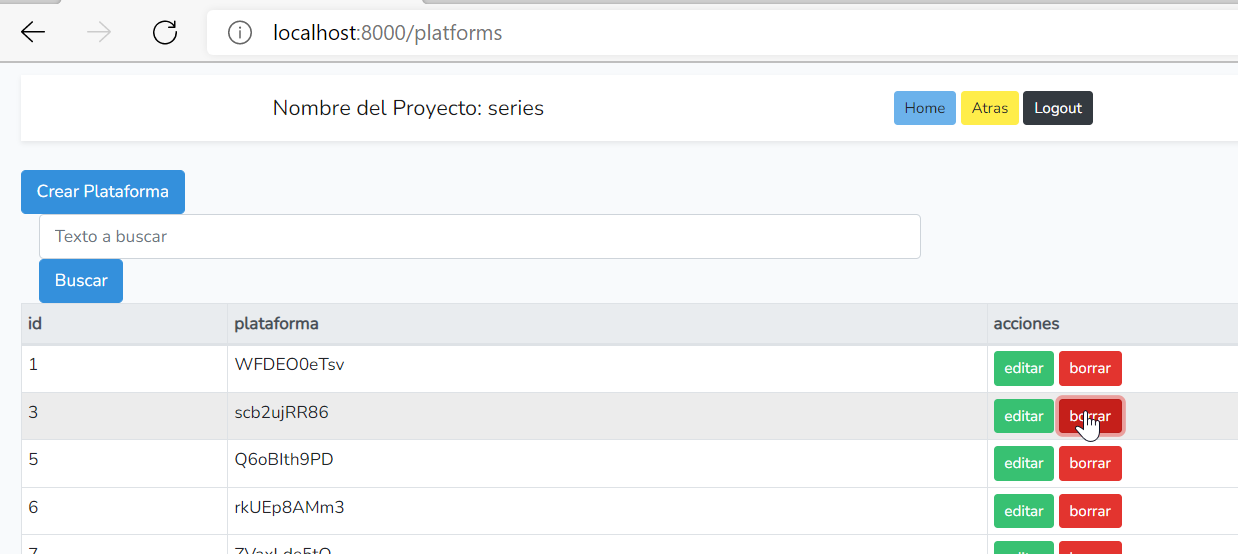
Actualización



Creación

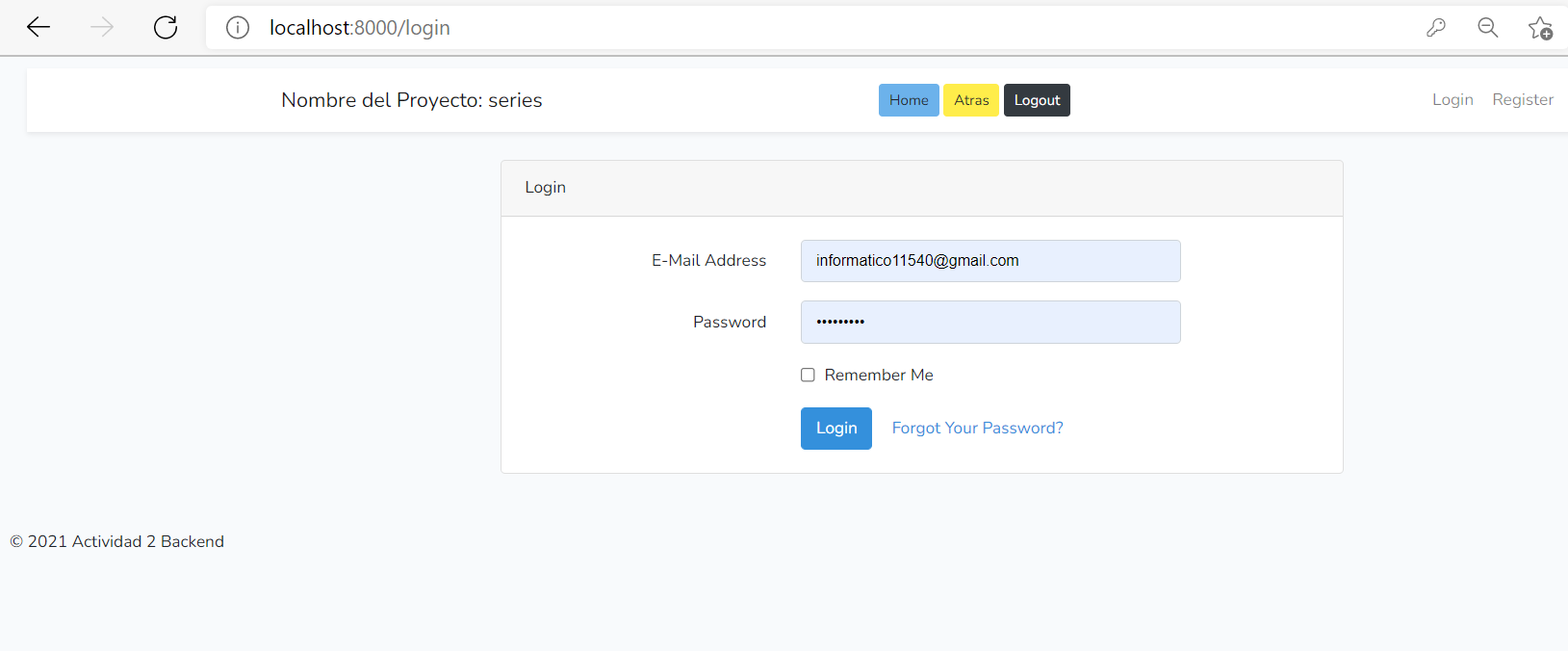


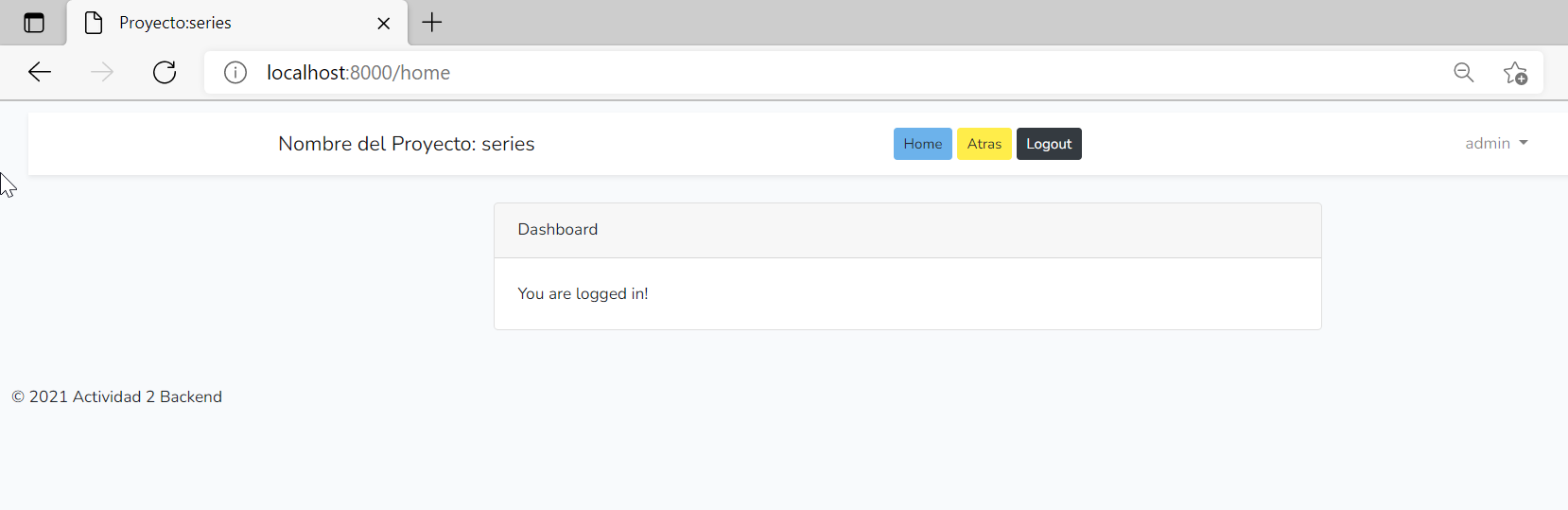
Borrado



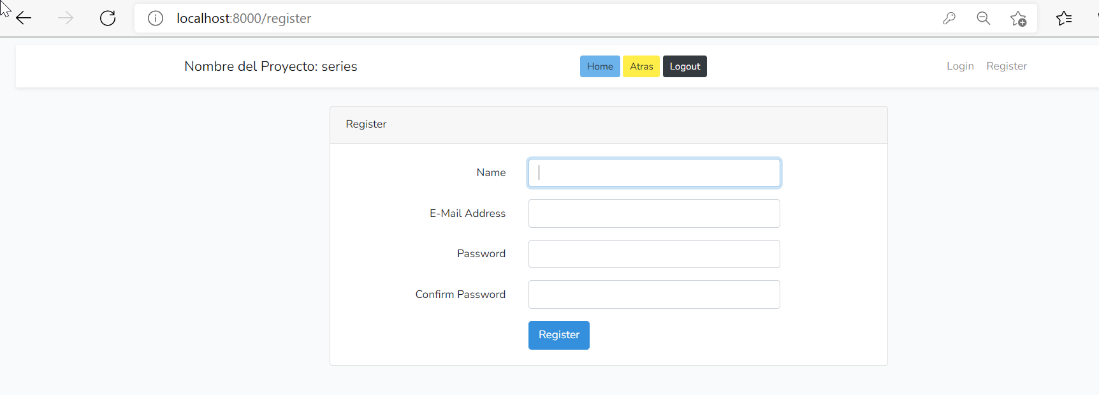
### Creación, Listado, Edición y Borrado de Idiomas

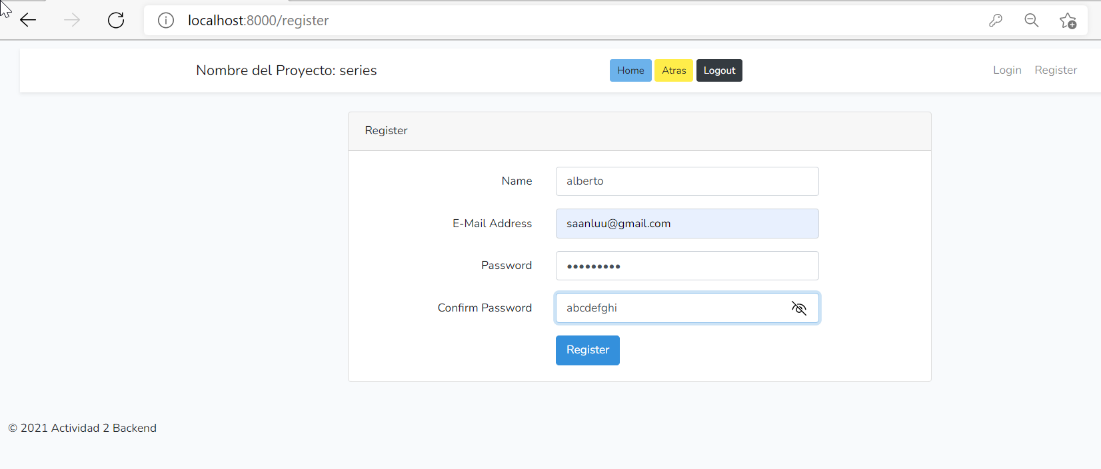
Entidad Idioma. Me lleva automáticamente a Identificarme si no lo estoy



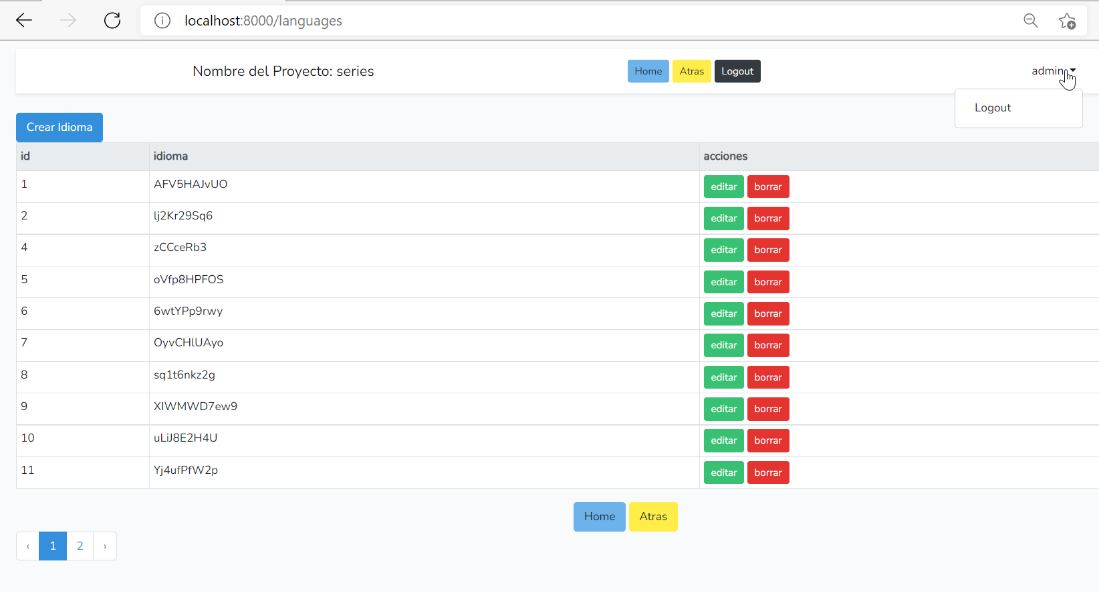


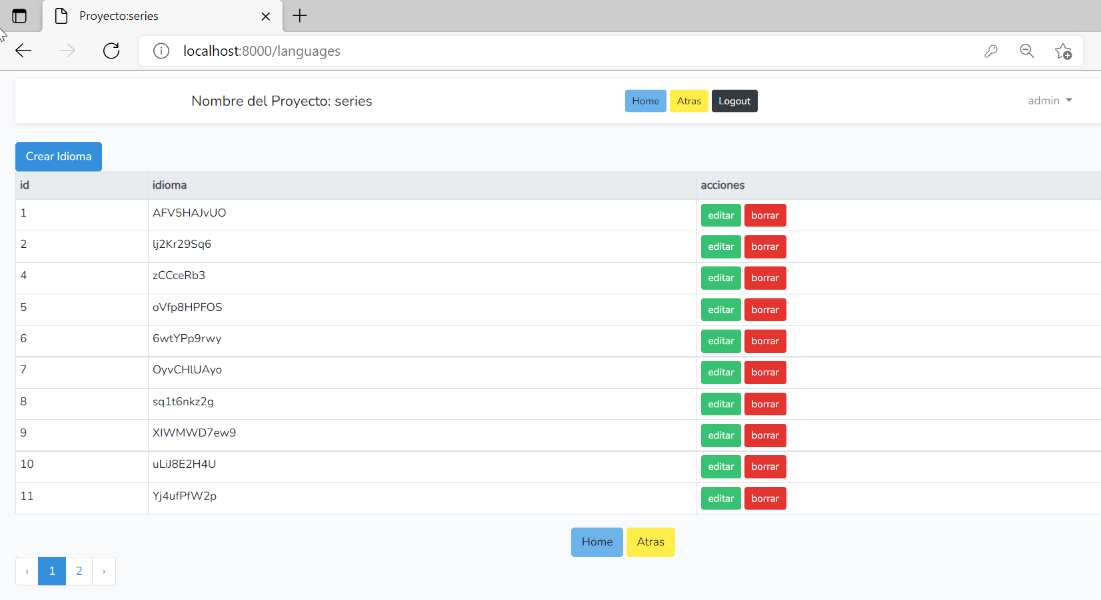
Registro de usuario



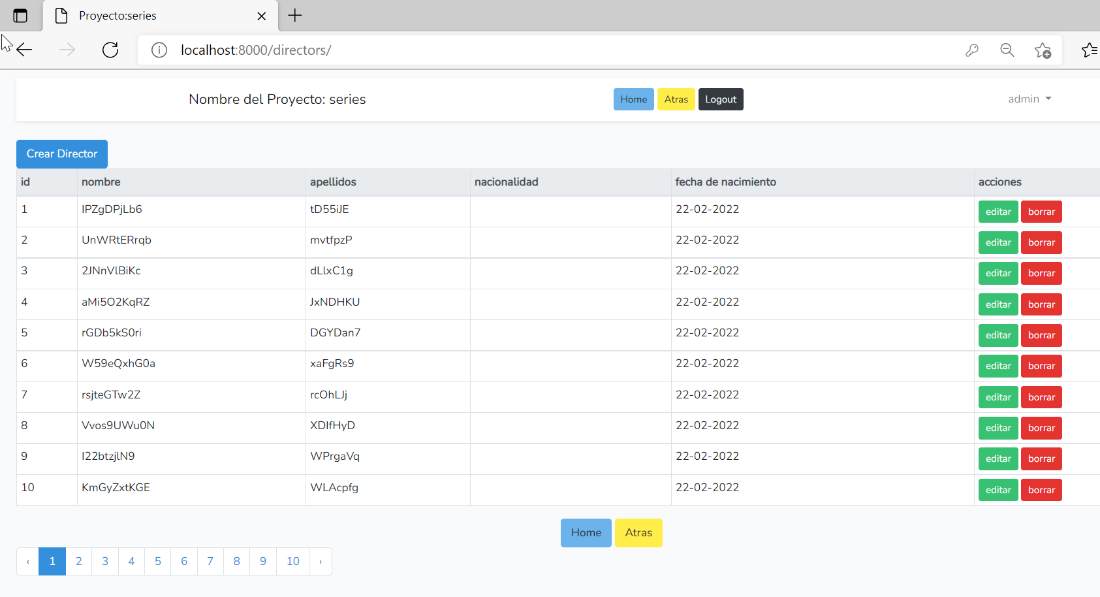


Identificado como Admin me lleva a las distintas pantallas de aplicación

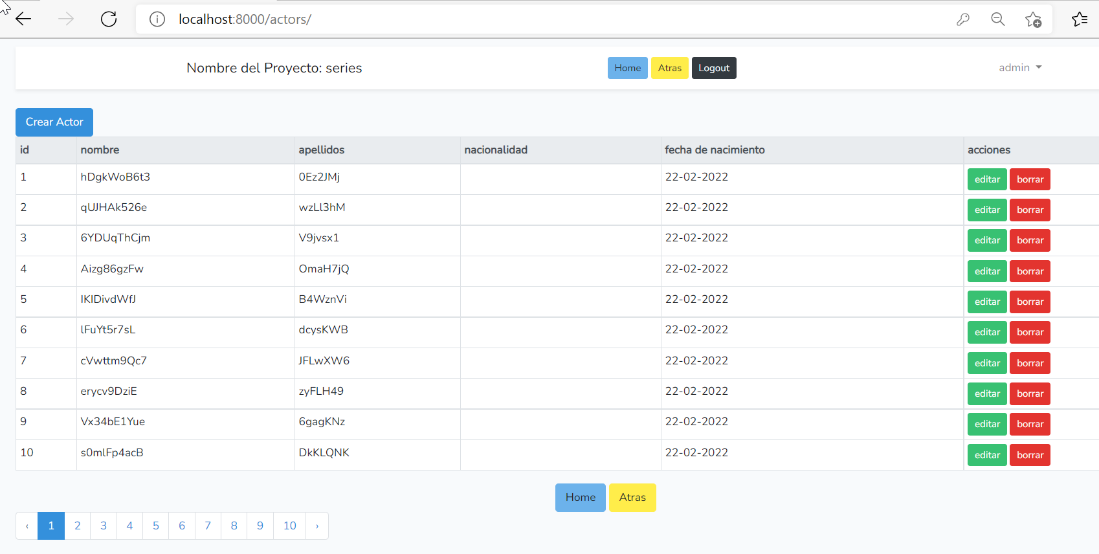




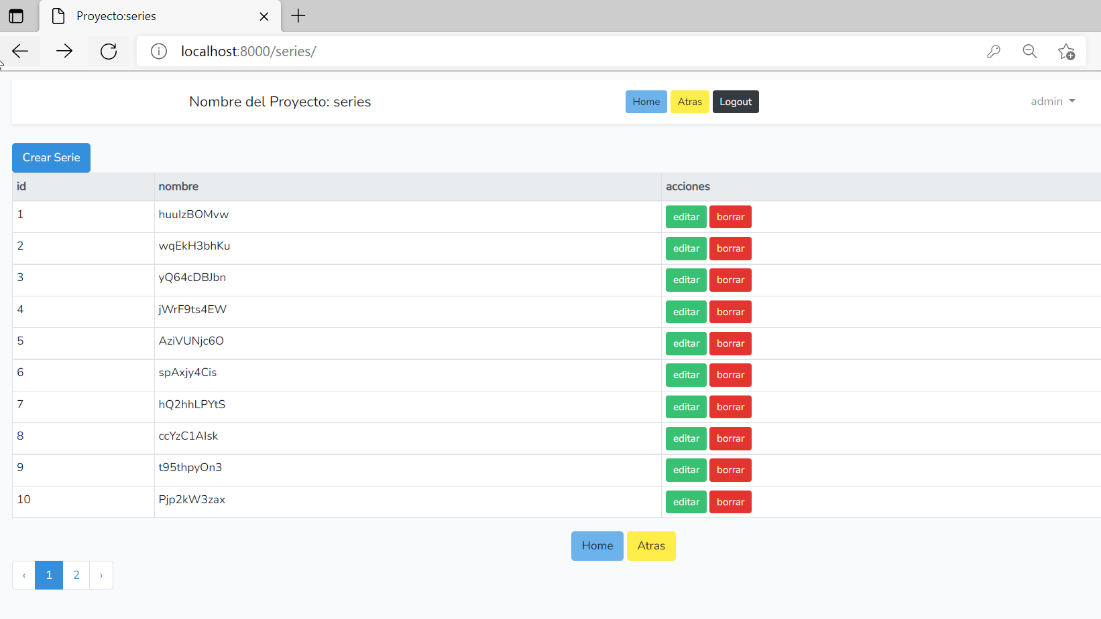
### Creación, Listado, Edición y Borrado de directores



### Creación, Listado, Edición y Borrado de Actores



### Creación, Listado, Edición y Borrado de series



### Creación, Listado, Edición y Borrado de la relación Actores - Series

