MEMORIA ACTIVIDAD 2

Desarrollo de aplicaciones web I: Lado del Servidor (back-end)

Universidad Internacional de Valencia

Máster U. en Desarrollo de Aplicaciones y Servicios Web

Josu Gutiérrez Nebreda

Jesús López González

12-08-2022

Índice

1. Introducción

En la presente actividad vamos a utilizar el ***framework*** **Laravel** como estructura principal del trabajo el cual trabaja con **PHP**, lenguaje de programación principal del lado servidor junto a **HTML** para diseñar una biblioteca de series. Emplearemos adicionalmente una de las librerías más conocidas que trata sobre cambiar el estilo/diseño de sitios y aplicaciones web.

Concretamente, emplearemos la librería **Bootstrap**. Bootstrap tiene un sin fin de clases que nos permite mejorar las vistas de cada una de las diferentes páginas de las que consta nuestra aplicación, entre ellas, mostrar botones con un aspecto estético mejorado respecto a la representación estándar, uso de columnas para mejorar la distribución de las páginas, formularios, etc..

Por tanto, en esta actividad vamos a utilizar lo estudiado en clases de teoría. Aunque las especificaciones pueden ser incompletas, deberían ser suficientes para conseguir un aspecto bastante similar a las referencias gráficas que aquí aparecen [1].

2. Software utilizado

El objetivo de este apartado es explicar las diferentes herramientas software que hemos utilizado para conseguir realizar con éxito la actividad.

1. **PHP** (Hypertext Preprocessor): "Es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML." [2]
2. **HTML: "**(Lenguaje de Marcas de Hipertexto, del inglés *HyperText Markup Language*) es el componente más básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento (JavaScript)." [3]
3. **Bootstrap:** "Es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios web y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end." [4]
4. **MAMP:** "Es un entorno de servidor local gratuito que se puede instalar en macOS y Windows con solo unos pocos clics. MAMP les proporciona todas las herramientas que necesitan para ejecutar WordPress en su PC de escritorio con fines de prueba o desarrollo, por ejemplo. Incluso puede probar fácilmente sus proyectos en dispositivos móviles. No importa si prefiere el servidor web Apache o Nginx además de MySQL como servidor de base de datos, o si quiere trabajar con PHP, Python, Perl o Ruby." [5]
5. **Laravel:** "Laravel es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5, PHP 7 y PHP 8. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti". Fue creado en 2011 y tiene una gran influencia de frameworks como Ruby on Rails, Sinatra y ASP.NET MVC." [6]

3. Diseño Base de datos

Este apartado trata sobre mostrar un esquema de la base de datos, a partir de la cual se ha realizado esta actividad. Dado que se trata de la misma base de datos de la actividad 1, no es necesario volver a detallar todas las tablas. En la Imagen 1 veremos el esquema mencionado:

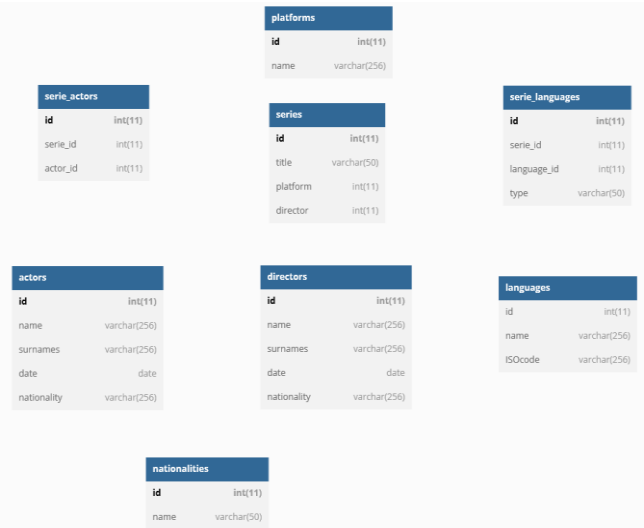


Imagen 1: *base de datos*

4. Desarrollo en ficheros

En este apartado hablaremos de las funciones implementadas que hemos realizado para conseguir el cumplimiento de los objetivos de la actividad. Dado que en la actividad se han modificado muchos ficheros y todos tienen la misma estructura y funcionalidad. Para no ser muy repetitivos con las explicaciones, únicamente nos centraremos en explicar los cambios realizados en la entidad ***actors***.

a) Modelo (Actor.php)

En la Imagen 2 veremos la clase ***Actor*** la cual consta principalmente de una función llamada ***serieActor()*** que nos indica que la tabla *actor* tiene una relación mediante el uso de una ***foreign key*** con otra tabla.

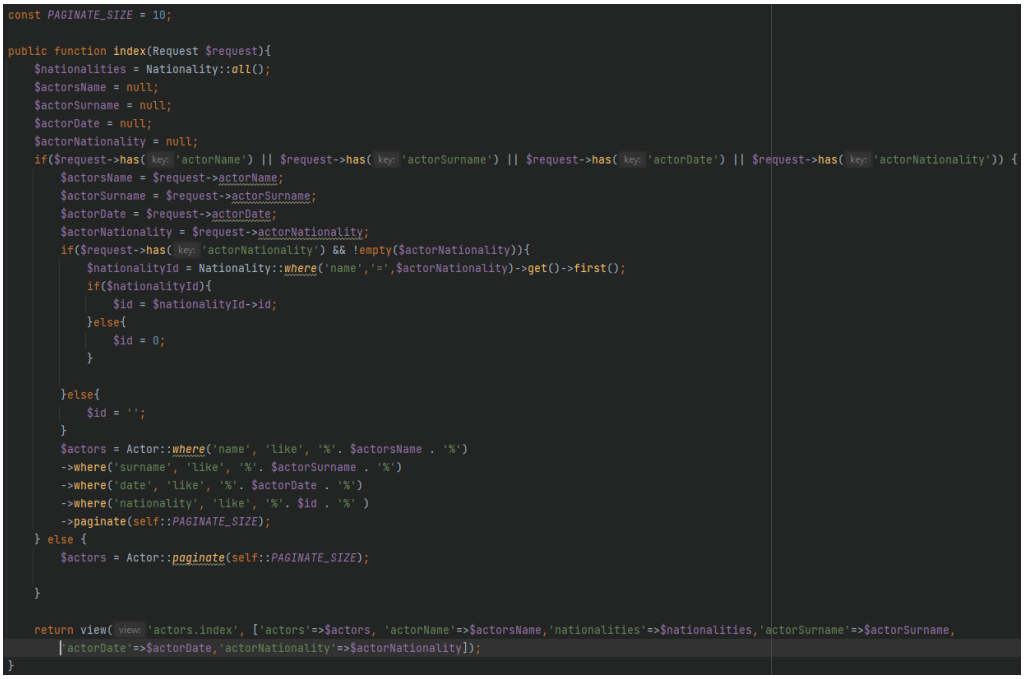
```php
class Actor extends Model
{
    protected $table = 'actors';

    public function serieActor()
    {
        return $this->belongsTo( related: serieActor::class, foreignKey: 'actor_id');
    }
}
```

Imagen 2: *modelo actor*

b) Controlador (PlatformController.php)

En este fichero se almacenan todos los métodos relacionados con las acciones de crear, leer, actualizar y borrar. Comúnmente conocido como las acciones ***CRUD (create, read, update y delete)***. A continuación, hablaremos de cada uno de ellos:

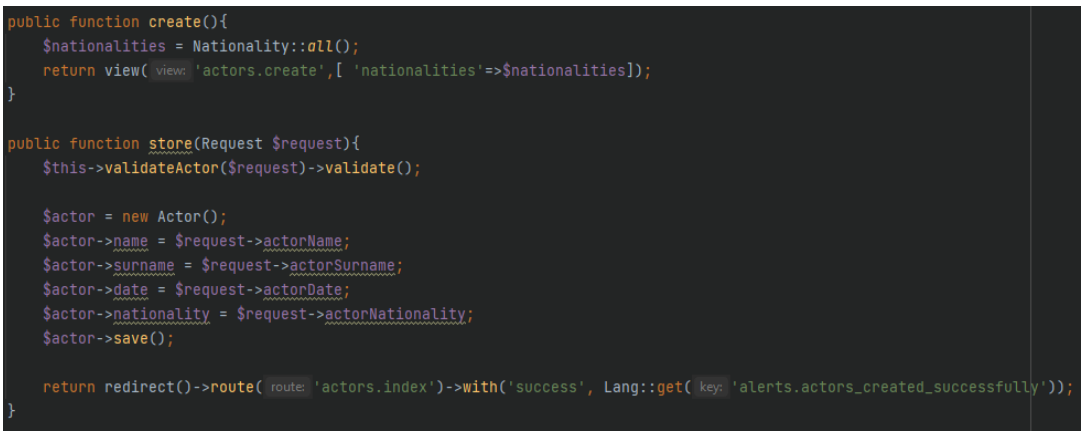
En la Imagen 3, podemos observar la función ***index()*** la cual trata sobre mostrar, en formato lista, todos los actores que tenemos almacenadas en la base de datos. Con relación a las acciones *CRUD* esta tiene que ver con la **R** de ***read***. También destacamos que, en esta función se ha añadido el código necesario para cumplir con la tarea del buscador por campos.



```
const PAGINATE_SIZE = 10;

public function index(Request $request){
    $nationalities = Nationality::all();
    $actorsName = null;
    $actorSurname = null;
    $actorDate = null;
    $actorNationality = null;
    if($request->has( key: 'actorName') || $request->has( key: 'actorSurname') || $request->has( key: 'actorDate') || $request->has( key: 'actorNationality')) {
        $actorsName = $request->actorName;
        $actorSurname = $request->actorSurname;
        $actorDate = $request->actorDate;
        $actorNationality = $request->actorNationality;
        if($request->has( key: 'actorNationality') && !empty($actorNationality)){
            $nationalityId = Nationality::where('name','=',$actorNationality)->get()->first();
            if($nationalityId){
                $id = $nationalityId->id;
            }else{
                $id = 0;
            }

        }else{
            $id = '';
        }
        $actors = Actor::where('name', 'like', '%'. $actorsName . '%')
        ->where('surname', 'like', '%'. $actorSurname . '%')
        ->where('date', 'like', '%'. $actorDate . '%')
        ->where('nationality', 'like', '%'. $id . '%' )
        ->paginate(self::PAGINATE_SIZE);
    } else {
        $actors = Actor::paginate(self::PAGINATE_SIZE);

    }

    return view( view: 'actors.index', ['actors'=>$actors, 'actorName'=>$actorsName,'nationalities'=>$nationalities,'actorSurname'=>$actorSurname,
        'actorDate'=>$actorDate,'actorNationality'=>$actorNationality]);
}
```

Imagen 3: *función index()*

En la Imagen 4, podemos observar las funciones ***create()*** y ***store()*** la cual trata sobre mediante un formulario, poder crear y guardar un nuevo actor con todos sus campos. Con relación a las acciones *CRUD* esta tiene que ver con la **C** de ***create***.



```
public function create(){
    $nationalities = Nationality::all();
    return view( view: 'actors.create',[ 'nationalities'=>$nationalities]);
}

public function store(Request $request){
    $this->validateActor($request)->validate();

    $actor = new Actor();
    $actor->name = $request->actorName;
    $actor->surname = $request->actorSurname;
    $actor->date = $request->actorDate;
    $actor->nationality = $request->actorNationality;
    $actor->save();

    return redirect()->route( route: 'actors.index')->with('success', Lang::get( key: 'alerts.actors_created_successfully'));
}
```

Imagen 4: *función create() y store()*

En la Imagen 5, podemos observar las funciones ***edit()*** y ***update()*** la cual trata sobre mediante un formulario, poder editar y actualizar uno o varios campos de unos de los actores que ya tenemos en la base de datos. Con relación a las acciones *CRUD* esta tiene que ver con la **U** de ***update***.

```php
public function edit(Actor $actor){
    $nationalities = Nationality::all();
    return view( view: 'actors.edit', ['actor'=>$actor,'nationalities'=>$nationalities]);
}

public function update(Request $request, Actor $actor){
    $this->validateActor($request)->validate();
    Log::info( message: 'Fallo',array($actor));
    $actor->name = $request->actorName;
    $actor->surname = $request->actorSurname;
    $actor->date = $request->actorDate;
    $actor->nationality = $request->actorNationality;
    $actor->save();

    return redirect()->route( route: 'actors.index')->with('success', Lang::get( key: 'alerts.actors_update_successfully'));
}
```

Imagen 5: *función edit() y update()*

En la Imagen 6, podemos observar la función ***delete()*** la cual trata sobre eliminar un actor ya existente en la aplicación.

```php
public function delete(Request $request, Actor $actor){
    if($actor != null) {
        $actor->delete();
        return redirect()->route( route: 'actors.index')->with('success', Lang::get( key: 'alerts.actors_delete_successfully'));
    }
    return redirect()->route( route: 'actors.index')->with('error', Lang::get( key: 'alerts.actors_delete_error'));
}
```

Imagen 6: *función delete()*

En la Imagen 7, podemos observar la función ***find()*** la cual trata sobre cumplir con la tarea de implementar un buscador avanzado el cual permite buscar cualquier tipo de información relacionada con actor.

```php
public function find(Request $request){
    $nationalities = Nationality::all();
    $actorsName = null;
    if($request->has( key: 'actorFind')) {
        $actorsName = $request->actorFind;

        if($request->has( key: 'actorFind') && !empty($actorsName)){
            $nationalityId = Nationality::where('name','=',$actorsName)->get()->first();
            if($nationalityId){
                $id = $nationalityId->id;
            }else{
                $id = 0;
            }
        }else{
            $id = '';
        }

        $actors = actor::where('name', 'like', '%'. $actorsName . '%')
        ->orWhere('surname', 'like', '%'. $actorsName . '%')
        ->orWhere('date', 'like', '%'. $actorsName . '%')
        ->orWhere('nationality', 'like', '%'. $id . '%')
        ->paginate(self::PAGINATE_SIZE);
    } else {
        $actors = actor::paginate(self::PAGINATE_SIZE);
    }

    return view( view: 'actors.index', ['actors'=>$actors, 'actorFind'=>$actorsName,'nationalities'=>$nationalities]);
}
```

Imagen 7: *función find()*

Además, en este controlador disponemos de un método que nos sirve para validar que los datos introducidos se introduzcan de forma correcta. En la Imagen 8, podemos observar la función ***validateActor()*** la cual trata sobre lo mencionado anteriormente.

```php
protected function validateActor($request) {
    return Validator::make($request->all(), [
        'actorName' => ['required', 'string', 'max:255', 'min:1'],
        'actorSurname' => ['required', 'string', 'max:255', 'min:1'],
        'actorDate' => ['required', 'date'],
        'actorNationality' => ['required', 'int']
    ]);
}
```

Imagen 8: *función validateActor()*

a) Vistas

i. index.php

Este fichero trata sobre mostrar, en formato lista, las diferentes plataformas disponibles en nuestra aplicación. Esto lo podemos conseguir gracias a la función *index()* que nos creamos en *ActorController*. En la Imagen 9 se muestra el diseño final de esta página.

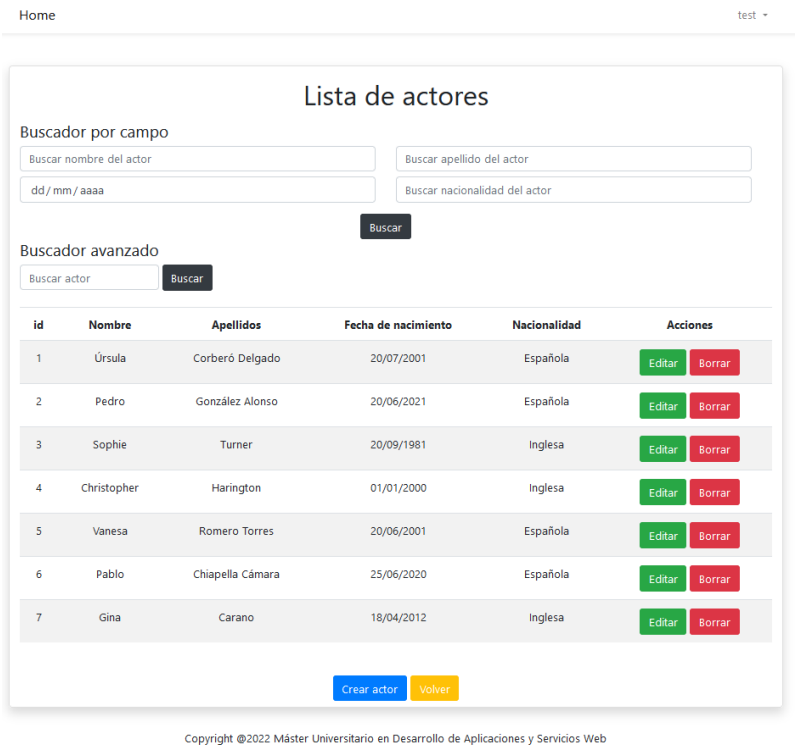


Imagen 9: *lista de actores*

ii. create.php

Este fichero trata sobre crear los actores insertando el *name*, *surname*, *date* y *nationality* mediante el formulario que permite este fin. Esto lo podemos conseguir gracias a la función *store()* que nos creamos en *ActorController*. En la Imagen 10 se muestra el diseño final de esta página.

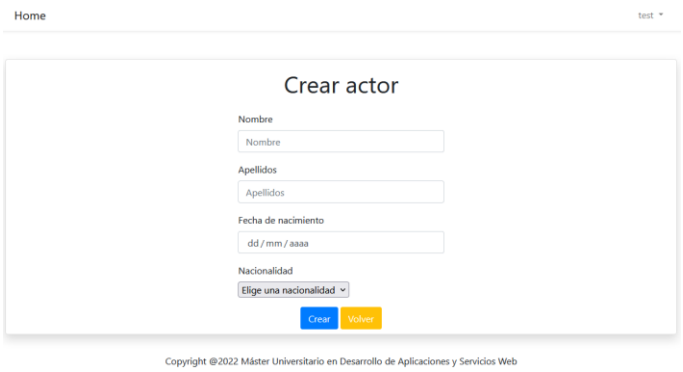


Imagen 10: *formulario crear actor*

iii. edit.php

Este fichero trata sobre editar los actores que ya se encuentran creados en nuestra aplicación. Esto lo podemos conseguir gracias a la función *update()* que nos creamos en *ActorController*. En la Imagen 11 se muestra el diseño final de esta página.

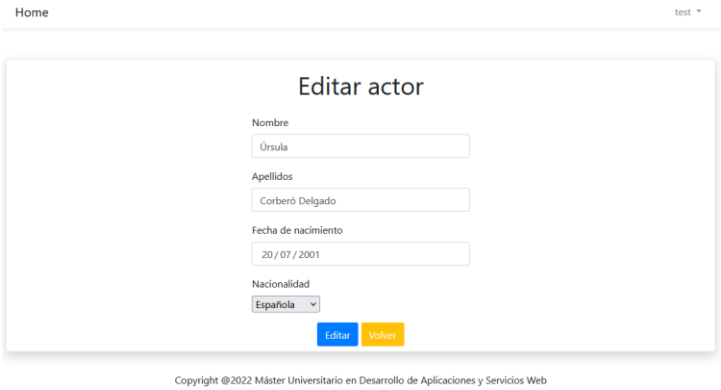


Imagen 11: *formulario editar actor*

iv. delete.php

Este fichero trata sobre eliminar un actor que ya se encuentra creado en nuestra aplicación. Esto lo podemos conseguir gracias a la función *delete()* que nos creamos en *ActorController*. En la Imagen 12 se muestra el diseño final de esta página.

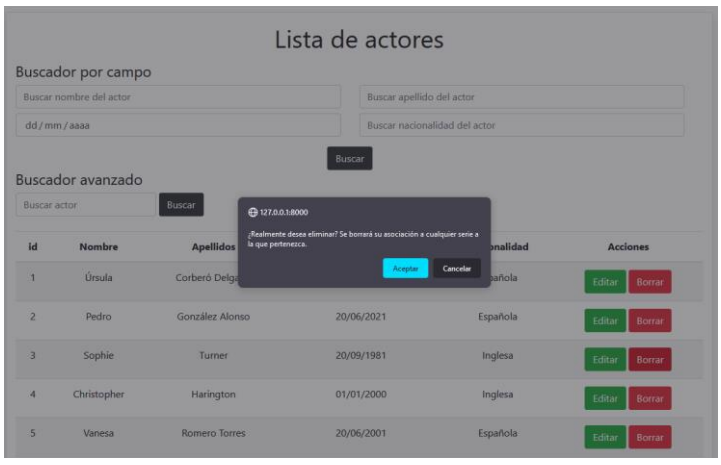


Imagen 12: *mensaje de borrar actor*

Cabe destacar que, para conseguir mostrar un mensaje de advertencia cuando pulsamos en cada uno de los botones de "borrar", es necesario añadir esta función *<button type="submit" class="btn btn-danger"* ***onclick = "return confirm('¿Realmente desea eliminar? Se borrará su asociación a cualquier serie a la que pertenezca.')"****>{{ __('string.delete_btn') }} </button>*

Para finalizar, vamos a mostrar el resultado final que tiene nuestra página de inicio ya que ha sido modificada respecto a la original. En la Imagen 13 veremos nuestro ***welcome.blade.php***:

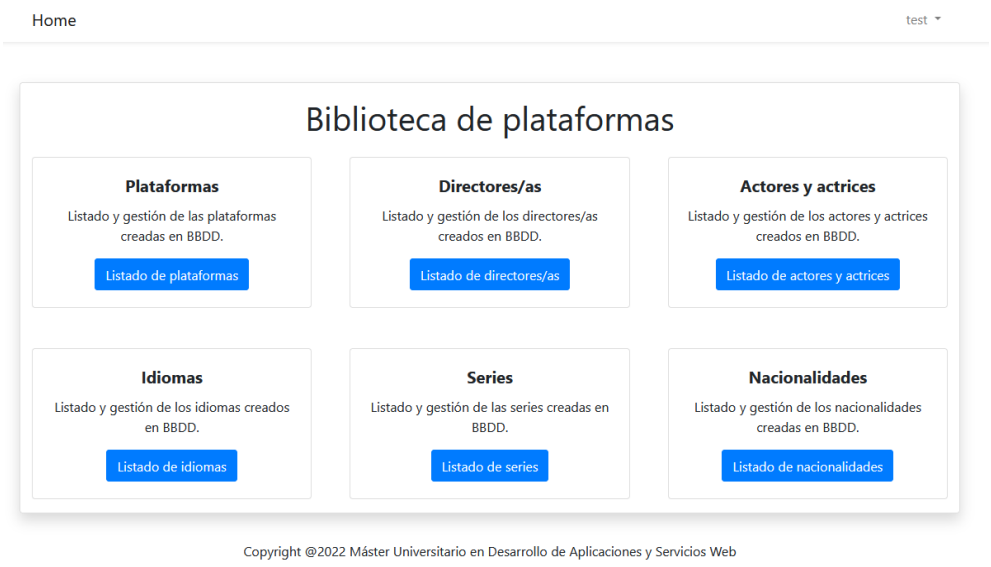


Imagen 13: *pantalla home*

5. Conclusión

Como conclusión, tanto mi compañero como yo, podemos decir que hemos conseguido realizar con éxito el desarrollo de esta actividad. Esta actividad ha sido bastante interesante de implementar porque nunca habíamos trabajado con la herramienta *MAMP* como soporte para nuestro servidor local y además, junto al *framework* Laravel.

Haciendo énfasis al *framework* Laravel, nos ha resultado bastante útil la explicación que recibimos en las diferentes clases de teoría junto a los ejemplos y ejercicios que realizamos, porque sin esa ayuda puede que nos hubiese costado un poco más instalar todo el software necesario y por tanto, arrancar con la actividad.

También cabe mencionar que, gracias a la explicación a modo de ejemplo que realizó el profesor Rubén en clase sobre la entidad "*platforms*"(plataformas), nos ayudó mucho con el resto de las entidades ya que nos facilitó el camino para no tener que realizar algunas funciones desde cero.

Cabe destacar que, gracias al gran conocimiento previo del equipo en cuanto a Laravel y los lenguajes de PHP, y HTML junto con la librería de Bootstrap, se ha podido conseguir realizar todas las funcionales mínimas para superar esta actividad además de añadir vistas más amigables de cara al usuario.

Para finalizar, mi compañero y yo hemos aprendido más sobre nuevas funcionalidades sobre el *framework* Laravel, haciendo que en nuestros próximos proyectos podamos realizar una aplicación haciendo uso de este lenguaje.

6. Bibliografía

[1] Rubén Martínez Albiñana. (2022). Introducción [Consultado en: 11-ago-2022].

[2] Definición de PHP. Software utilizado [Consultado 11-ago-2022].

[3] Definición de HTML. Software utilizado [Consultado 11-ago-2022].

[4] Definición de Bootstrap. Software utilizado [Consultado 11-ago-2022].

[5] Definición de MAMP. Software utilizado [Consultado 11-ago-2022].

[6] Definición de Laravel. Software utilizado [Consultado 11-ago-2022].

[7] Stack overflow.