Documentación Técnica

Índice

1.Objetivo y aspectos principales del programa.

1.1 Objetivo:

El objetivo de esta aplicación web era poder visualizar los valores de la bolsa de una lista de empresas predefinida anteriormente. Para ello teníamos que utilizar **drag & drop** para elegir la empresas que quisiéramos seleccionar, utilizar el **fetch** para hacer llamadas a una API, **crear una API** en laravel para hacer llamadas a la base de datos, las llamadas de la api necesitan un **bearer token** para que se acepte la request, generar datos falsos para insertarlos en bases de datos que luego utilizará la API, utilizar **highchartjs o chartjs** para crear gráficos con los valores de la bolsa y por último teníamos que contenerizar toda para poder desplegarlo con docker con el **comando docker compose up**

1.2 Aspectos principales

1.2.1 Base de datos:

Tenemos que tener una base de datos con 2 tablas una para insertar los datos creados y sacar los datos de ella y la otra para meter las empresas , el mínimo histórico era de un año.

1.2.2 API

Para crear la api tuvimos que utilizar el framework laravel, junto con ella tuvimos que instalar distintas dependencias para cumplir con los objetivos. La API tenía que ser creada de tal forma de que cuando un consumidor quisiese utilizar la api para iniciar sesión o para registrarse no se necesitase el token pero cuando quisiese acceder a los datos de la base de datos si.

1.2.3 Generación de datos

En mi caso utilice el framework de laravel para crear los datos falsos. Cuando llamo a la función crearDatos le paso 2 parámetros cada cuánto quiere que se creen los datos y hasta cuando. También está hecho de tal forma que empezara a generar datos desde la última inserción.

1.2.4 Gráficos

Para crear los gráficos utilice una librería llamada highchartjs. Cuando hiciese una request a la base de datos a través de la API y recibiese los datos tenía que procesarlos para que la dependencia pudiera entenderlos.

1.2.5 Fetch y drag & drop

En el frontend tuve que utilizar la librería de jquery para utilizar los drags & drops para seleccionar las empresas que se mostrasen, también tuve que utilizar para arrastrar la empresa que quisiese eliminar al icono de la basura.
Todo las llamadas a la api fueron hechas con la función de fetch que viene incluido por defecto en javascript. Hay distintos tipos de fetch ya que algunos tuve que utilizar el método post, en otros el get y también en ciertas llamadas el token

1.2.6 Docker

Cuando todo el código estuviese terminado teníamos que contenerizar toda la aplicación para poder desplegarlo rápidamente, para ello tuvimos que crear 5 contenedores distintos

1.2.6.I Contenedor httpd

Utilizamos un contenedor con una imagen httpd para desplegar el frontend de la aplicación. Utilizamos un volumen para copiar todo de la maquina host en el contenedor.

1.2.6.II Contenedor ubuntu

Utilizamos un imagen de ubuntu para poder desplegar la aplicación de label en un apache sin tener que estar ejecutando constantemente el comando php artisan serve. Utilizamos un dockerfile para instalar todas las dependencias y servicios necesarios para desplegar la aplicación. Dentro del dockerfile utilizamos el copy para copiar los archivos en la ruta deseada en docker. En este contenedor están simultáneamente la api y la generación de datos.

1.2.6.III Base de datos

Utilizamos una imagen mysql para crear la base de datos, la base de datos es importada al contenedor utilizando un archivo .sql

1.2.6.IV PHPMyAdmin

Utilizamos una imagen de phpmyadmin para poder acceder rápidamente a la base de datos sin tener que meternos en el container.

1.2.6.V Nginx

Utilizamos un imagen de nginx para balancear la carga de la api y que no se saturase. También hicimos que no se pudiese acceder al frontend ni al backend. Solo se puede acceder a ellos a través del nginx

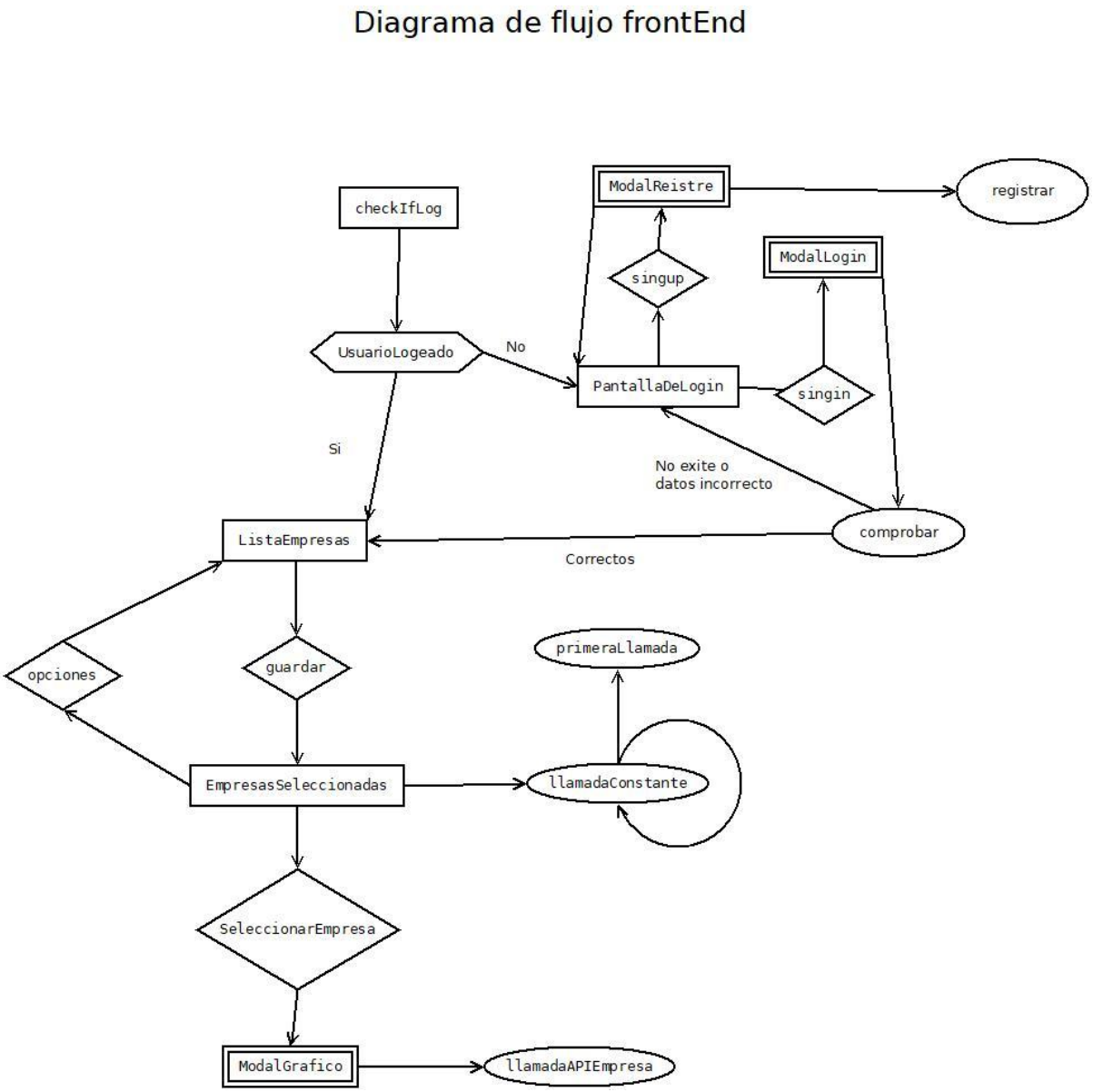
2. Requerimientos del sistema

Esta aplicación ha sido probada con:
Sistema operativo: Windows
Version: 19045.2486
Espacio mínimo: 2.20 gb

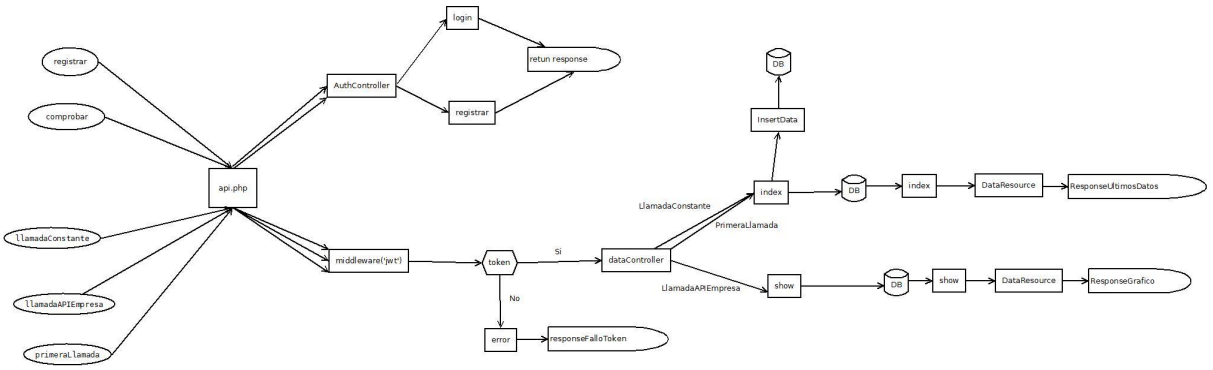
Procesador: i7-8850H
RAM: 16 gb
Tipo de sistema: 64 bytes

3. Diagrama de flujo

3.1 Frontend

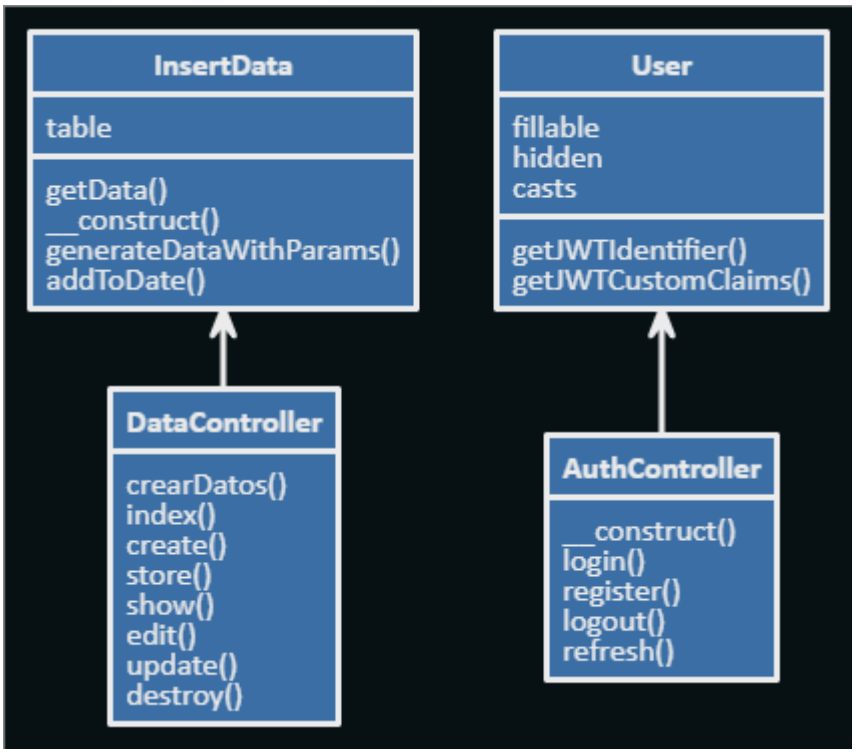


3.2 Backend

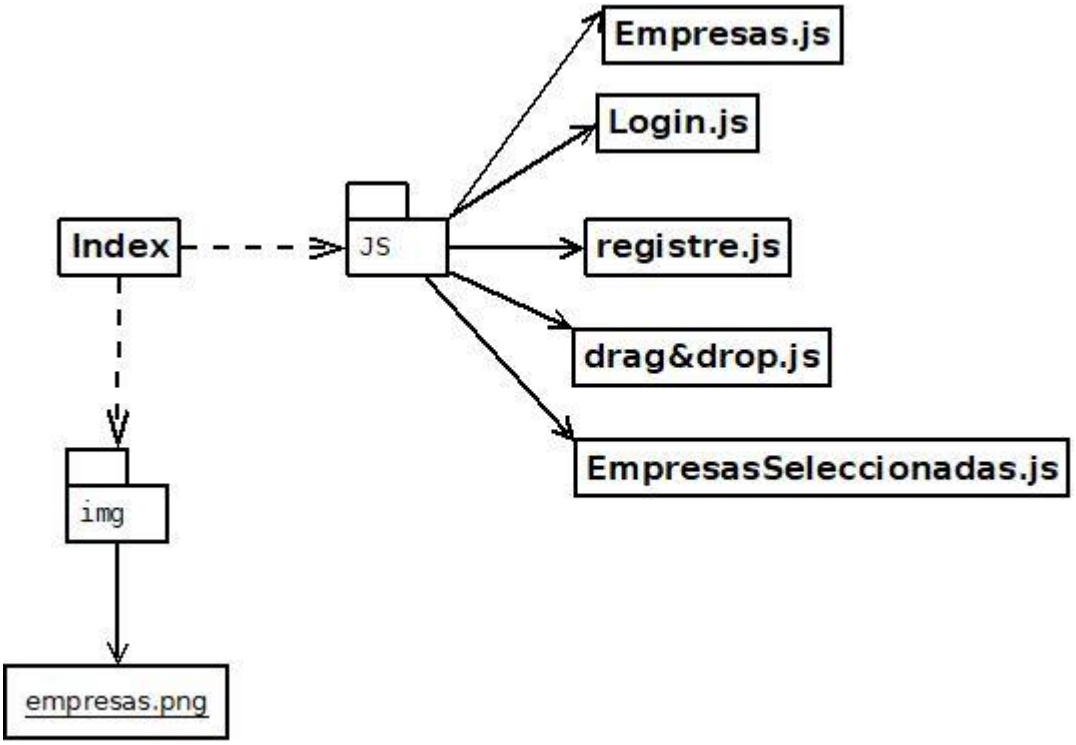


4. Diagrama de clases

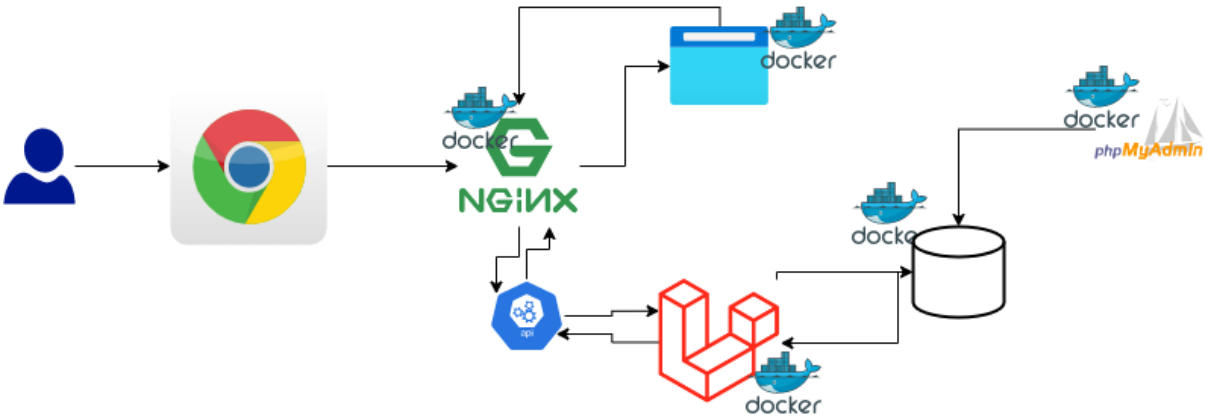
4.1 Backend



4.2 Frontend



5.Diseño o arquitectura del sistema



6.Código más representativo del programa y explicación de variables, datos, archivos.

6.1 Frontend

6.1.I Gráficos

```
data['data'].forEach(function (item, index) {
    if (index < 1000) {

        let date = new Date(item.date)
        let fecha = date.getTime();
        datos[index] = [fecha, item.euros]
    }


});
```

Cuando llama a la función del gráfico después de haber recibido un response en el fetch tuve que hacer varias modificaciones al response que me llegaba. Tuve que modificarlo y ponerle un límite de 1000 elementos al array porque si no mi gráfico crasheaba

6.1.II Drag & Drop

```
function changeClass(id) {
    if(id != "opciones"){
        document.getElementById(id).className = 'img-thumbnail p-2 empresaSeleccionada ui-draggable ui-draggable-handle'
    }

    var imgs = $('#empresaSeleccionadas img');
    let imagenes = [];
    for (let i = 0; i < imgs.length; i++) {
        imagenes[i] = imgs[i].id
    }

    sSeleccionadas = "";
    imagenes.forEach(element => {

        empresas = empresas.filter(e => e !== element);
        sSeleccionadas += `<img src="../img/${element}.png" id="${element}"class="img-thumbnail p-2 empresaSeleccionada" alt="${element}">`
    });
    sSeleccionadas += `<p><button type="button" class="btn btn-primary position-relative align-content-end"
       onclick="guardar()">Save</button></p>`


    $('#empresaSeleccionadas').html(sSeleccionadas);
    $('.empresaSeleccionada').draggable({
        helper: 'clone',
        drag: function () {
            deleteImg = "";
            deleteImg = this;

        }
    });
}
```

Cuando hago que la imagen se mueva desde las empresas a seleccionar a las empresas seleccionadas llamó a la función de changeClass donde le paso el id de la imagen. Después cojo todas las imágenes que estén dentro de la id empresasSeleccionadas para poder

volver a generar ese div pero con la nueva empresas dentro y también tengo que volver a añadirle el evento draggable al div porque al generar el nuevo código pierde el evento.

6.1.III Llamadas API

```
ip = location.host;
```

```
let controller = new AbortController();
controller.abort();
const newController = new AbortController();

fetch(`http://${ip}:1235/api/empresas`, {
    method: 'GET',
    signal: newController.signal,

    headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`,
        'Accept': 'application/x-www-form-urlencoded, application/json',
        'Content-Type': 'application/x-www-form-urlencoded'
    },
}).then(response => response.json()).then(response => { comprobarEmprsas(response, obj) })
    .catch(error => {
        if (error.name === 'AbortError') {
            console.log('Petición cancelada');
        } else {
            console.error(error);
        }
    });

controller = newController;
```

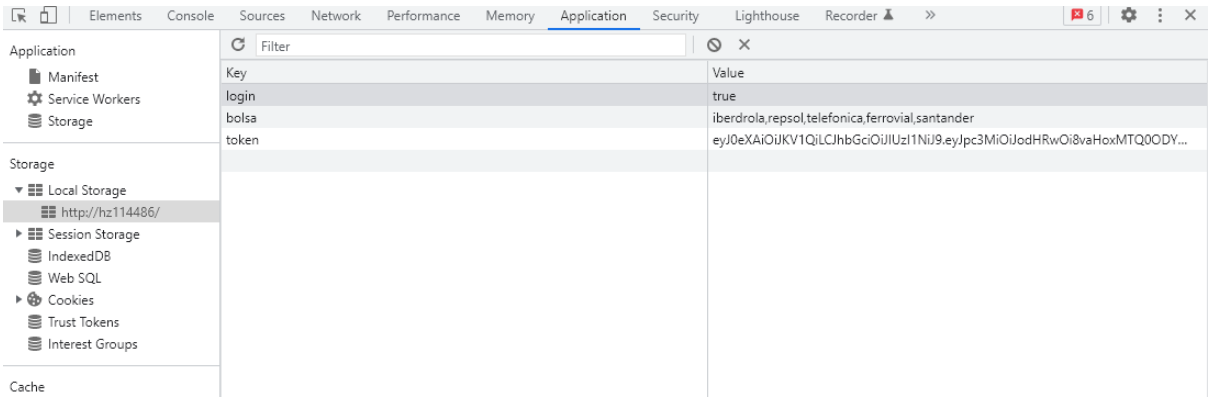
Cuando hacemos una llamada api con un token necesitamos en el header mandarle el token como hago yo, también tenemos que asegurarnos de que si la llamada tarda mucho en responder cancelamos la llamada para no saturar el programar

6.1.IV Papelera

```
$('.empresaSeleccionada').draggable({
    helper: 'clone',
    drag: function () {
        deleteImg = "";
        deleteImg = this;


    }

});




$('.basura').droppable({
    accept: '.empresaSeleccionada',
    hoverClass: 'hovering',
    drop: function () {
        deleteImg.style ="background-color: #e2eaf3;"
        document.getElementById('listaEmpresas').appendChild(deleteImg);

    }
});
```

En el div de empresas seleccionadas cojo la imagen con el elemento this para saber qué imagen estoy moviendo y únicamente la puedo soltar en la papelera que es el que accepta el las empresas Seleccionadas .

6.1.V Localstorage



Utilizamos el localstorage para guardar las empresas seleccionadas,si el usuario estaba logueado y el token de la API

6.2 Backend

6.2.I Generar Datos

```php
public function generateDataWithParams($until, $timeunit)
{
    set_time_limit(60000);
    $companiaIDs = DB::table('companias')->pluck('id');

    foreach ($companiaIDs as $element) {

        $maxCreatedAt = DB::table('bolsa')->where('compañia_id', $element)->max('created_at');

        $datos =  DB::table('bolsa')->where([['compañia_id', $element], ['created_at', $maxCreatedAt]])->first();

        $maxCreatedAt = Carbon::parse($maxCreatedAt);

        $euro = $datos->Euros;

       $dt = $this->addToDate($maxCreatedAt, $timeunit);
        while ($until > $dt) {

            if ($dt->isWeekday() && ($dt->hour  < '17' && $dt->hour >= '9')) {
                $rand = (rand(-10, 10));
                $randomVariation = (rand(-500, 500) / 10000) * 5;
                $variacion = round($randomVariation, 2);
                $variacion = $variacion + $rand;
                $euro = round($euro + ($euro * $variacion)/100, 2);
                if($euro<0.5){
                    $euro = rand(5, 10);
                }
                $variacion = round($variacion,2);
                $euro = round($euro,2);
                DB::table('bolsa')->insert([
                    'id' => Str::uuid()->toString(),
                    'created_at' =>  $dt,
                    'variacion' => $variacion,
                    'Euros' => $euro,
                    'compañia_id' => $element,
                ]);
            }
            $dt = $this->addToDate($dt, $timeunit);
        }
    }
}
```

```php
public function addToDate($dt, $timeunit)
{

    switch ($timeunit) {
        case "day":
            return $dt->addDay();
            break;
        case "hour":
            return $dt->addHour();
            break;
    }

    return $dt->addMinute();
}
```

Estas son las funciones más importantes en mi archivo de generar datos. La primera función lo que hace es coger todas las id de las compañías y se hace un foreach para poder recorrer todas las compañías. Por cada compañía coge el último dato insertado en la base

de datos que tenga la id de la compañía y antes de llamar al bucle while llamamos a la función addToData donde los datos se generan cada minuto,hora o dia según lo que el usuario haya elegido y devuelve la hora actualiza para entrar en el bucle while hasta que las inserciones lleguen hasta la fecha deseada o si está sin especificar hasta la fecha actual. También comprobamos que no sea finde de semana o que no sean las horas que la bolsa esté cerrada para insertar datos. Una vez dentro del if sacamos los datos de euros para hacer las variación que haya salido con el random y se lo aplicamos y por último se lo insertamos a la base de datos.

6.2.II Bearer token

```php
Route::middleware('jwt')->group((function(){
    Route::get('/empresas', [DataController::class, 'index']);
    Route::get('/datos', [DataController::class, 'show']);
}));
```

Cuando pasamos por estas rutas laravel automáticamente pasa por el middleware para comprobar si el token que hemos enviado está bien o tiene algún problema.

```php
public function handle(Request $request, Closure $next)
{
    try{
     JWTAuth::parseToken()->authenticate();
    }catch(Exception $e){
        if($e instanceof TokenInvalidException){

            return response()->json(['status'=>'invalid token']);
        }

        if($e instanceof TokenExpiredException){
            return response()->json(['status'=>'token expired']);
        }

        return response()->json(['status'=>'token not found']);
    }
    return $next($request);
}
```

6.2.III Devolución de datos

```php
Route::middleware('jwt')->group((function(){
    Route::get('/empresas', [DataController::class, 'index']);
    Route::get('/datos', [DataController::class, 'show']);
}));
```

6.2.II.a Devolución minuto a minuto

```php
public function index()
{

    $model = new InsertData();
    $model->getData(null, null);


    $companiaIDs = DB::table('companias')->get();
    $companiaIDs = json_decode(json_encode($companiaIDs), true);
    $datos = array();
    foreach ($companiaIDs as $element) {


        $datos[$element['name']] = DB::table('bolsa')->where('compañia_id', $element['id'])->latest()->first();
    }

    return DataResource::collection($datos);
}
```

A la función de index se llama a través de la ruta api/empresas pero como esta el middleware('jwt') antes de poder acceder a la función comprueba que la request tenga el token. Si se ve lo primero que haces al llamar a la función de index se instancia un objeto del modelo InsertData para poner en marcha la generación de datos minuto a minuto y hasta la fecha actual. Una vez en marcha eso sacamos las id de las compañías y hacemos un foreach para sacar las últimas inserciones de las empresas en la tabla de bolsa y devolvemos en vez de lo que hemos sacado directamente de la base de datos devolvemos un array utilizando el DataResource::collection que llama hasta funcion que lo que hace es convertir lo que hayamos sacado de la base de datos en un array

```php
public function toArray($request)
{
    return [
        'date' => $this->created_at,
        'variacion' => $this->variacion,
        'euros' => $this->Euros,
    ];
}
```

Y esto es lo que devolvemos al frontend

6.2.II.b Devolución entre fechas específicas

```php
public function show(Request $request)
{

    $name = strtoupper($request->name);
    $company = json_decode(json_encode(DB::table('companias')->where('name', '=', $name)->get()));
    if (isset($request->from)) {
        $from = $request->from;
    } else {
        $from = Carbon::now();
    }

    if (isset($request->to)) {
        $to = $request->to;
    } else {
        $to = Carbon::now();
    }

    $from = Carbon::parse($from);
    $to = Carbon::parse($to);

    if($to->eq($from)){
        $bolsa = InsertData::select('*')->where('compañia_id', $company[0]->id,)->whereDate('created_at', $to)->get();

    } else{
        $to->addDay();
        error_log($to);
        $bolsa = InsertData::select('*')->where('compañia_id', $company[0]->id)->whereBetween('created_at', [$from, $to])->get();
    }

    return DataResource::collection($bolsa);
}
```

En esta función en la requst nos llega el nombre de la empresa y las fechas deseadas por lo que lo primero que hacemos es sacar la id de la compañía utilizando where. Después comprobamos que las fechas no hayan llegado vacías si es así las inicializamos con la hora actual. Tenemos que parsear las fechas a carbon ya que las fechas que nos llegan se consideran string. Lo primero que hacemos ahora que tenemos las fechas en carbon es comprobar si las fechas que nos han llegado son el mismo dia si es así utilizamos un where que trae laravel para sacar solo el dia de hoy ya que al parsearlas a carbon empiezan desde las 00:00 del dia y utilizando un where normal daría problemas ya que a esas horas no hay ningún dato. Si las fechas son distintas utilizamos el wherebetween para sacar los datos entre las dos fechas y por último volvemos ha hacer lo mismo llamamos a DataResource::collection para convertir los datos de la base de datos en un array

```php
public function toArray($request)
{
    return [
        'date' => $this->created_at,
        'variacion' => $this->variacion,
        'euros' => $this->Euros,
    ];
}
```

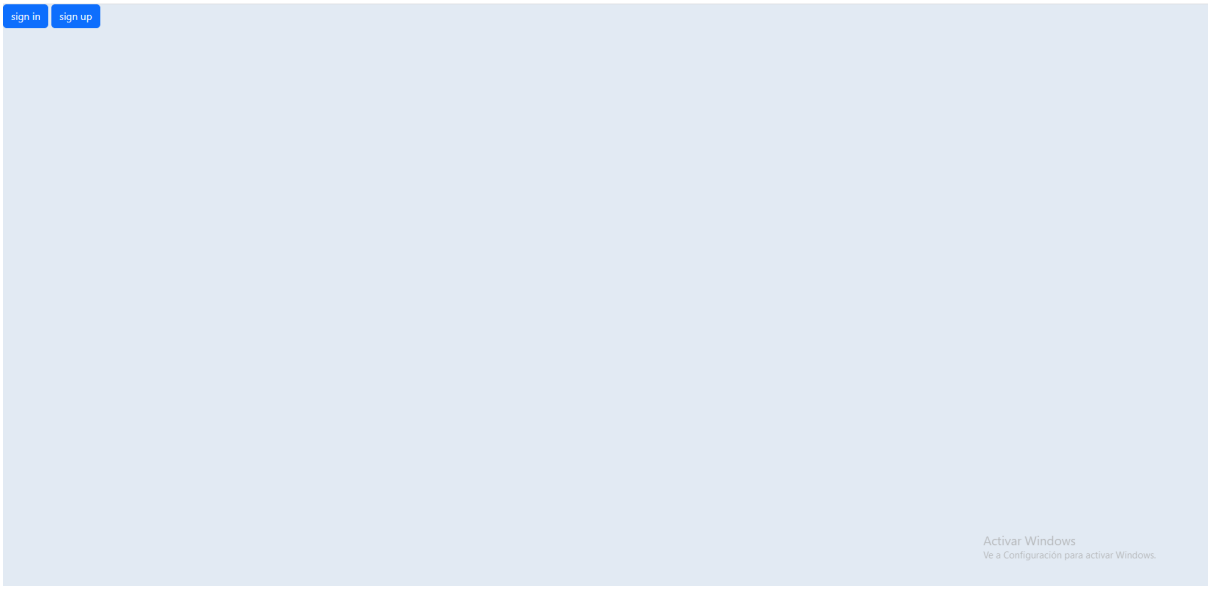
Y esto es lo que devolvemos al frontendç

7.Estilo

7.1 Botones

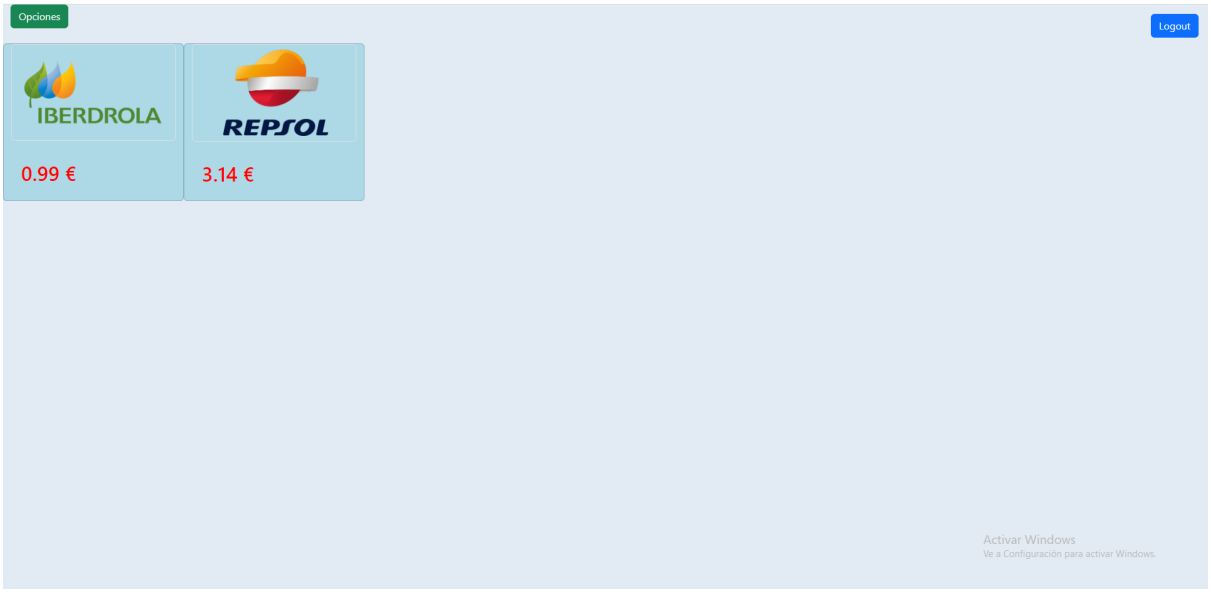
7.1.I Logout

Cuando clickes en el botón de logout que está arriba a la izquierda cerraras session y ocultará el contenido



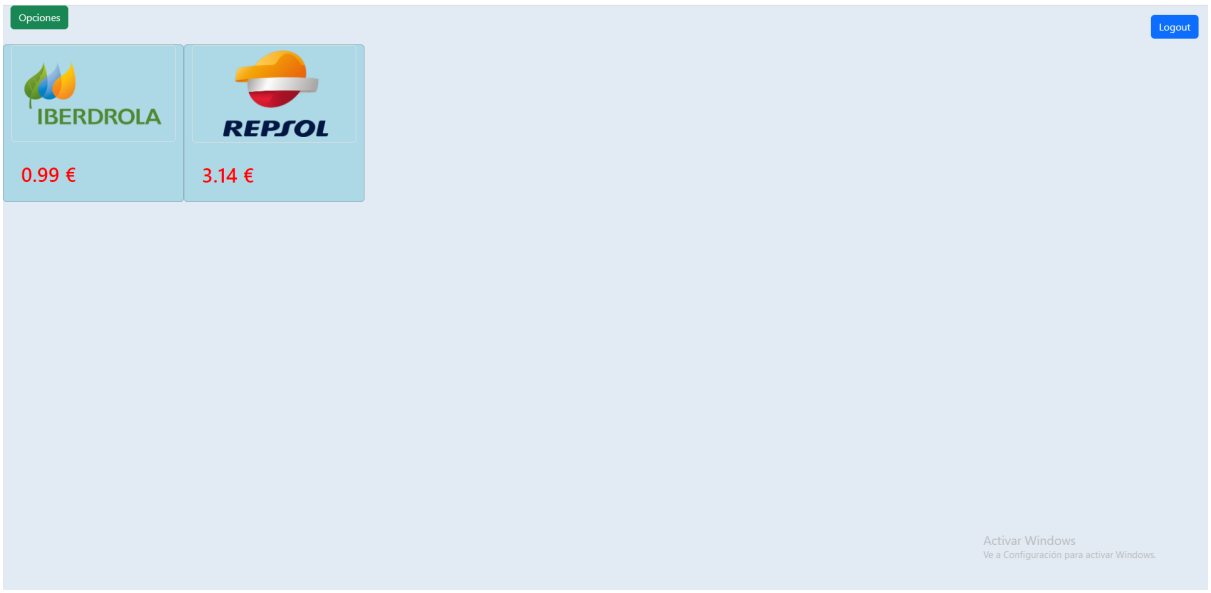
7.1.II Save

Cuando clickeamos en el botón de save nos mostrar la opciones seleccionadas



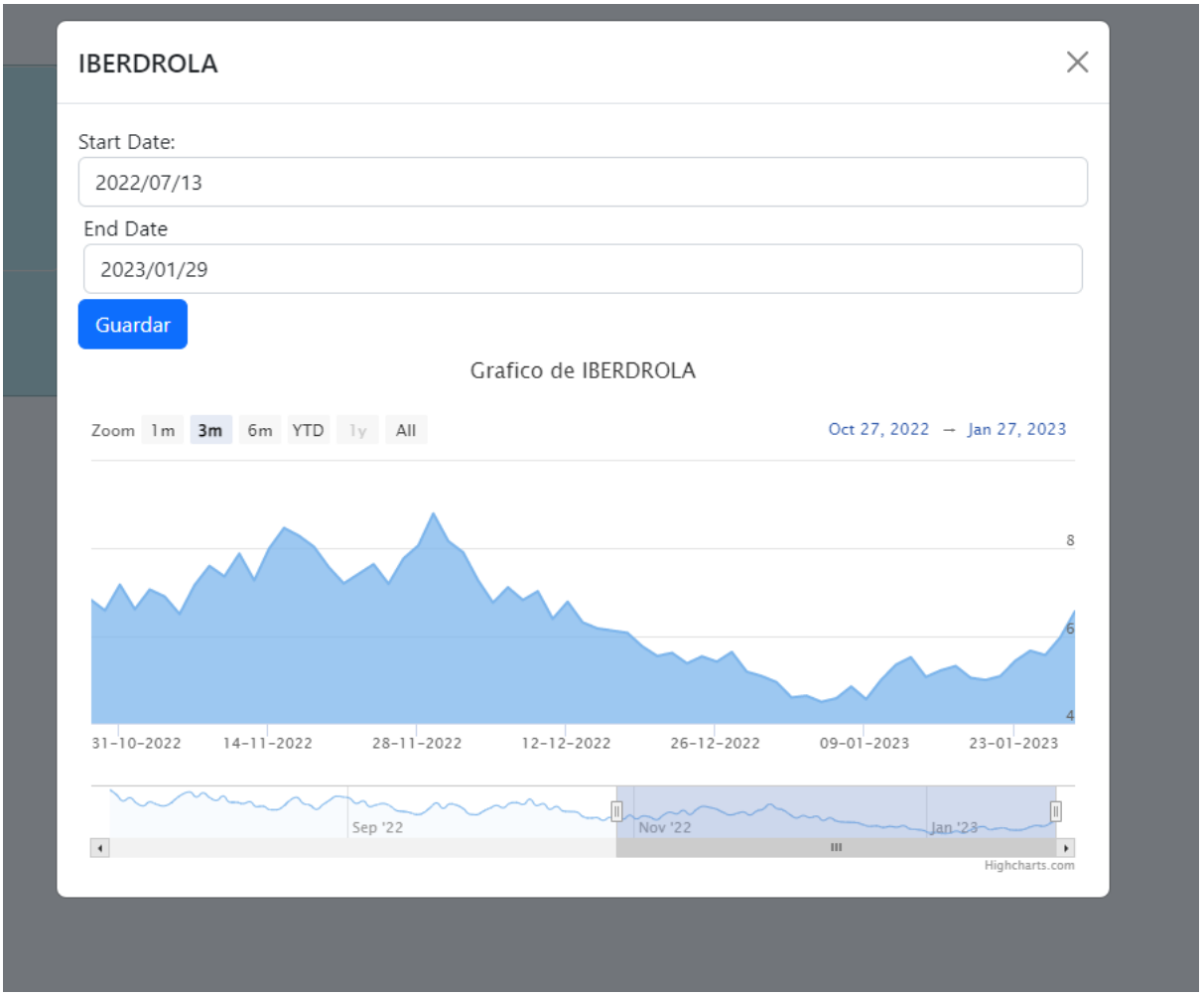
7.1.III Opciones

Cuando clickes en el boton de opciones nos hará volver a la pagina de atras



7.1.IV Gráfico

Cuando pulse el botón guardar mostrar el grafico con los datos de los inputs

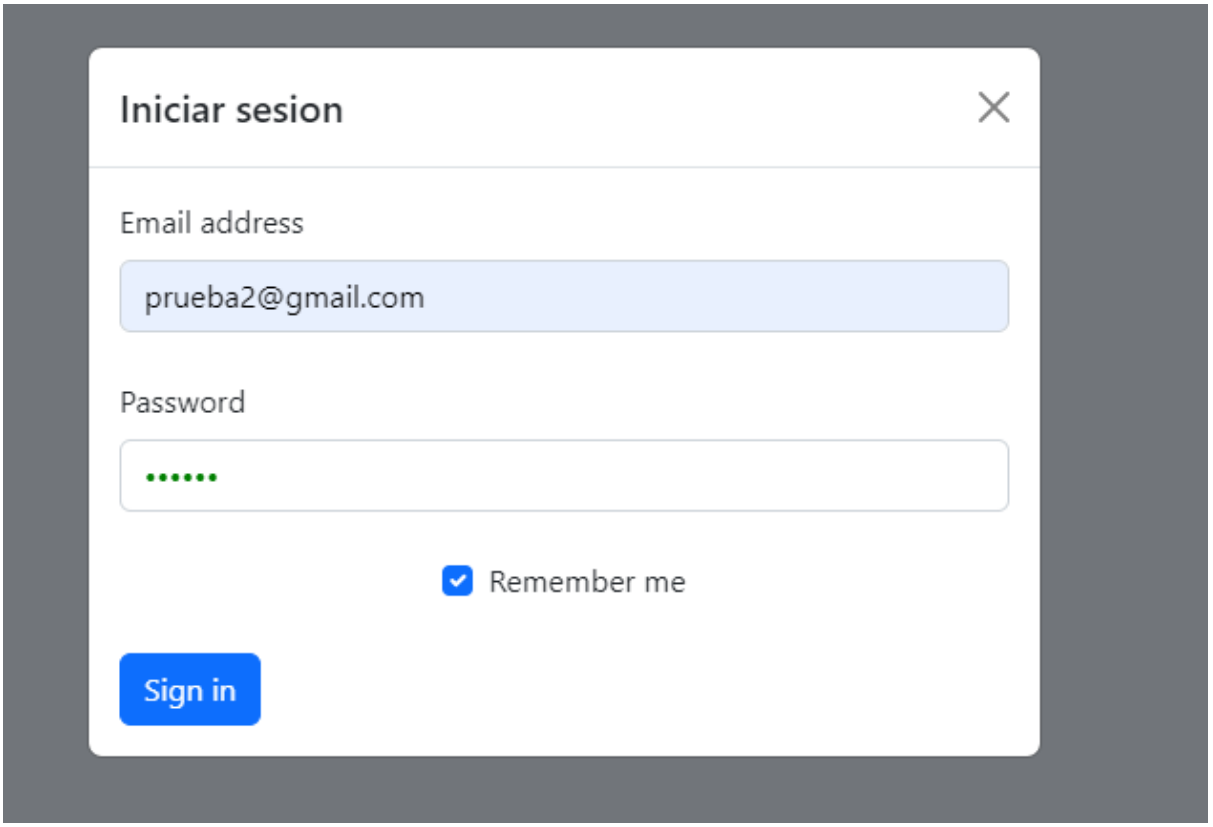


7.2 Modals

Todos los modales están creando con la libreria de boostrap

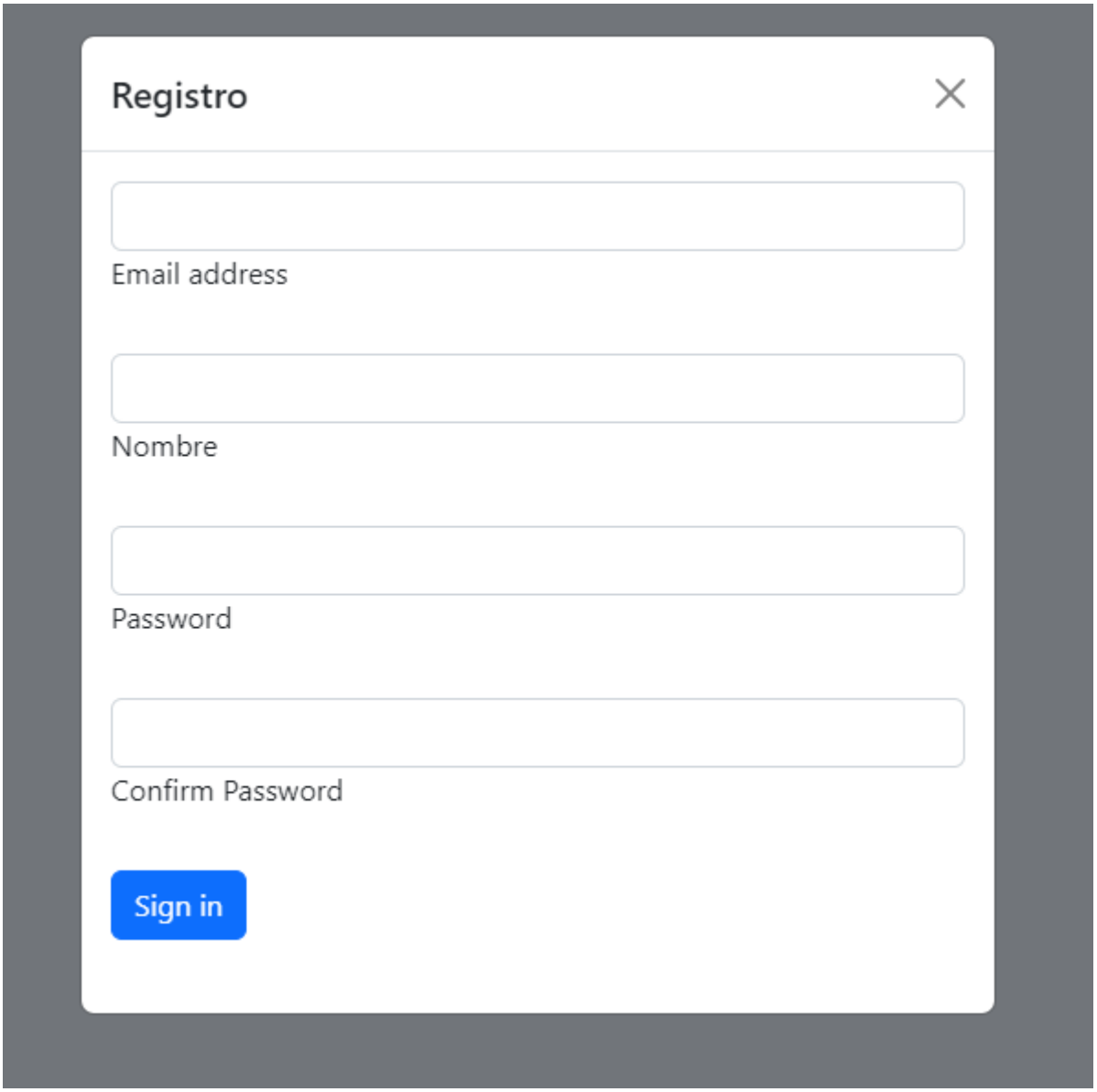
7.2.I Log in

Tengo un modal para iniciar sesión



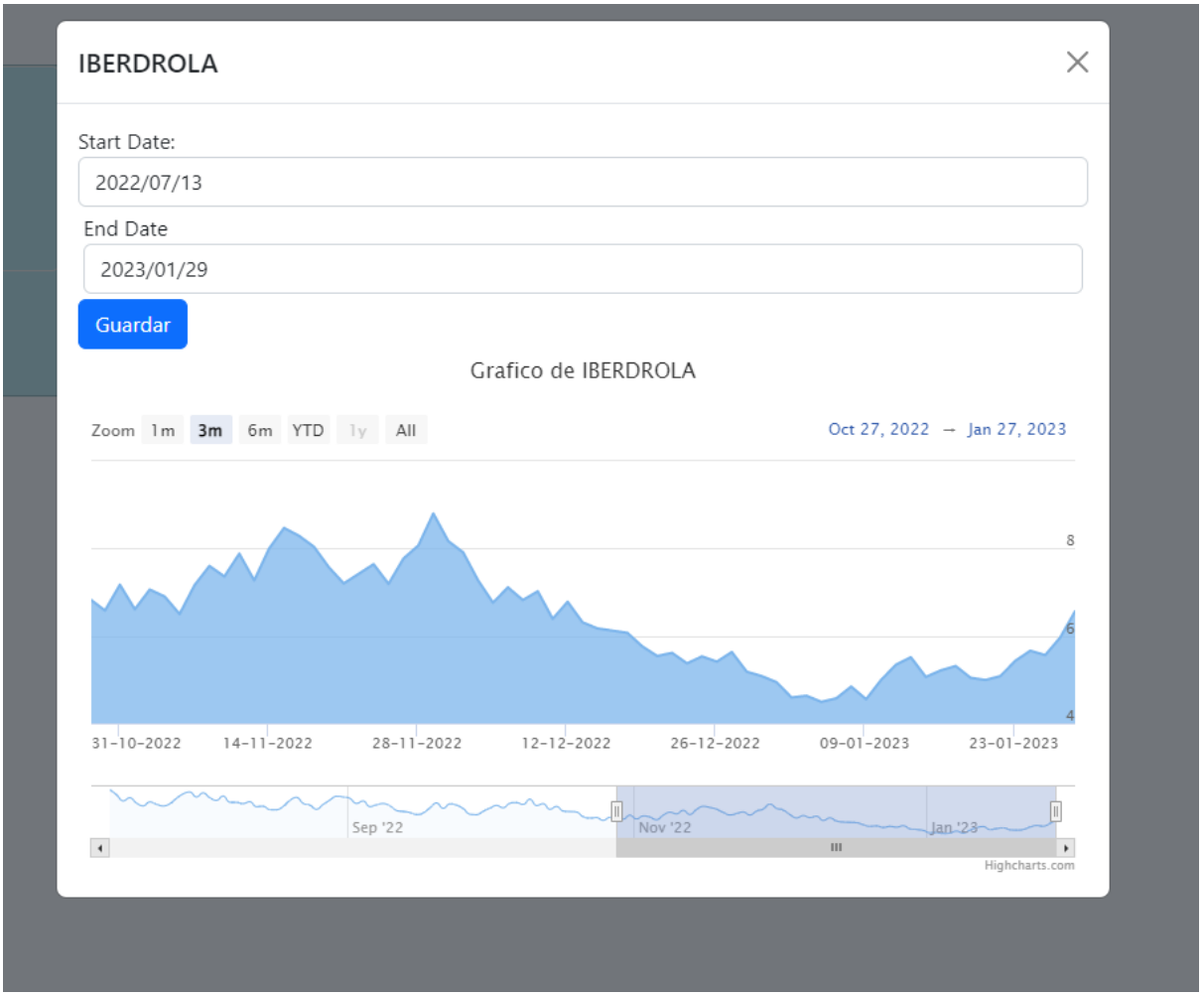
7.2.II Registre

Tengo un modal para registrarse



7.2.III Gráfico

Tengo un modal para mostrar el gráfico de la empresa



8.Explicación de la gestión de errores del programa.

8.1 Laravel en docker

Después de haber probado el código en live server me puse a meter el código en docker después de muchos problemas pude hacer que funcionase gracias a la ayuda de martin. Las request de login y de generar datos funcionaban pero cuando tenía que sacar los datos minuto a minuto o en el gráfico me decía que no existía la clase DataController. Al final mi solución fue sacar la clase dataController fuera de la carpeta api donde está metida.

8.2 Los datos en el gráfico

Cuando sacaba datos entre fechas el gráfico me funcionaba a la perfección pero cuando empecé a intentar sacar los datos del dia de hoy me empezo a fallar por como tenia hecho la respuesta de la api, una vez solucionado eso me di cuenta de otro problema cuando pedimos los datos de días que tenían cada minuto me devolvió una ingente cantidad de

datos que la liberia no podía procesarlos por lo que tuve que sacar solo un dato por hora. Por último tuve que procesar los datos ya que los datos que me venían desde la api no me los acepta la liberia por que tenia que cambiarlos.

8.3 Docker

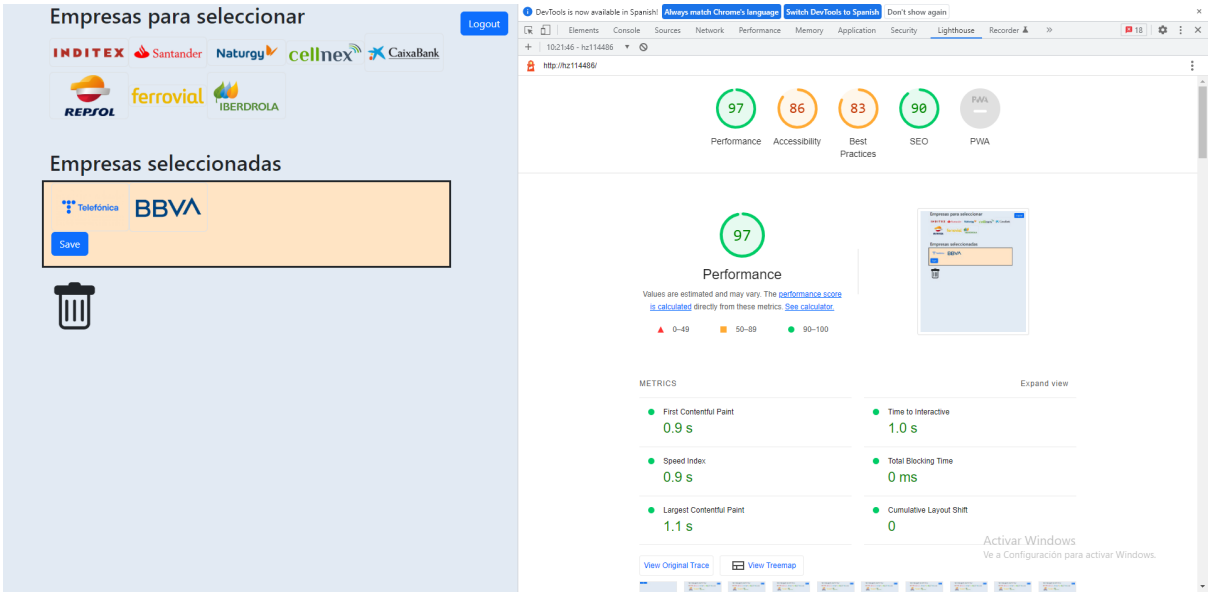
Cuando empecé a contenerizar la aplicación me dio muchos problemas laravel, estuve mirando en internet cómo hacerlo y mucha gente decía hacerlo con esta imagen php8.1-apache pero no conseguía que funcionase por lo que al final hice con lo que nos dijo santi crear una contenedor de ubuntu y instalar todo lo necesario para desplegar la aplicación.

8.4 Tiempo de la API

Cuando la aplicación está en marcha en docker la primera llamada de la API me tarda un 1 minuto entero responder pero después el resto de las llamadas tardan ms, no he sido capaz de saber porque tarda tanto en responder la API en la primera llamada.

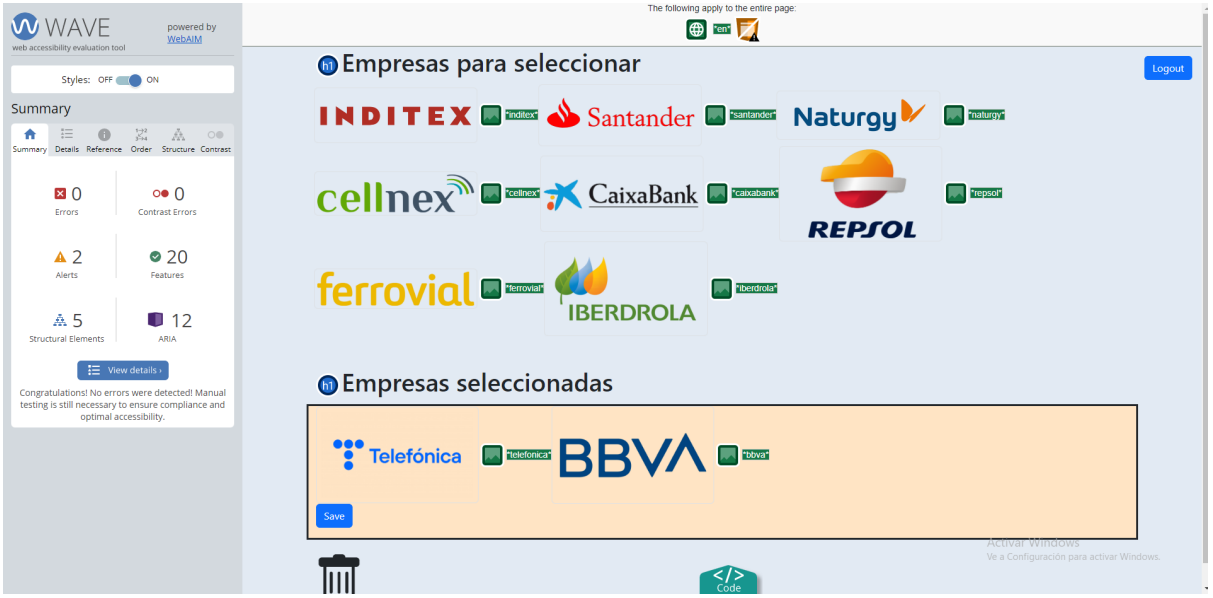
9.Validación y pruebas

9.1 Lighthouse



He utilizado la herramienta de lighthouse para comprobar el rendimiento de la pagina web

9.2 Wave



Utilizando la herramienta wave hemos comprobado que es accesible para gente con discapacidades.

10.Recomendaciones para la mejora y mantenimiento futuro.

10.1 Mejorar el diseño de la página

No he tenido tiempo suficiente para hacer un diseño bonito para el frontend por lo que este debería ser algo a mejorar en un futuro

10.2 Llamadas a la API

Tengo que mejorar el temas de las llamadas de la api por que al ponerlos en set intervals la primera llamada tarda el tiempo que hayas puesto, pero poner otra llamada igual esta vez sin el set intervals para llamarla la primera vez es código espagueti

10.3 Localstorage

Hay que mejorar donde guardar ciertas cosas ya que ahora mismo el token se guarda en el localstorage donde cualquiera puede hacer cualquier cosa con él.

11.Cualquier información que se considere relevante para un programador que haya de trabajar con el programa

11.1 Frontend

Las llamadas a la api en el frontend están separadas en distintos archivos más o menos para tener cierto orden. por ejemplo la llamada para comprobar los datos está en el archivo login.js

11.2 Backend

Laravel por defecto ya viene muy organizado lo único que necesitas saber es dónde está cada carpeta para poder acceder a los controladores o a los modelos

11.3 Docker

Hay un archivo docker-compose.yml en la raíz del proyecto que es lo que permite que se levante utilizando el docker compose up, para poder poner en marcha laravel también hay un dockerfile en la carpeta del backend.

12.Conclusiones

Después de haber estado desarrollando un mes y medio este proyecto he sacado unas cuantas conclusiones de él, separaré lo positivo de lo negativo.

12.1 Positivo

12.1.I

Una vez tienes toda la aplicación monta y el docker configurado es muy fácil levantarla

12.1.II

JavaScript es un lenguaje muy flexible que te permite hacer muchas cosas que en otros lenguajes no te dejaría

12.1.III

Una vez aprender a utilizar laravel es bastante fácil desarrollar en el framework encima con todas las dependencias que tiene y los comando que tiene para auto generar archivos

12.1.IV

Bootstrap es una librería muy útil cuando quieres ponerle estilo fácilmente a una página web y a la vez te da muchas utilidades como poder crear modals super fáciles

12.2 Negativo

12.2.I

Javascript al ser un lenguaje tan flexible que es lo positivo hace que el programador sea más descuidado con el código ya que es poco probable que de algún error y terminas con un código chapucero.

13 Sources

13.1 Github

Repositorio github: git