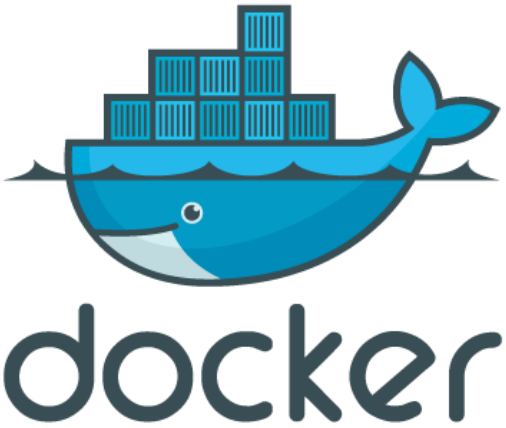
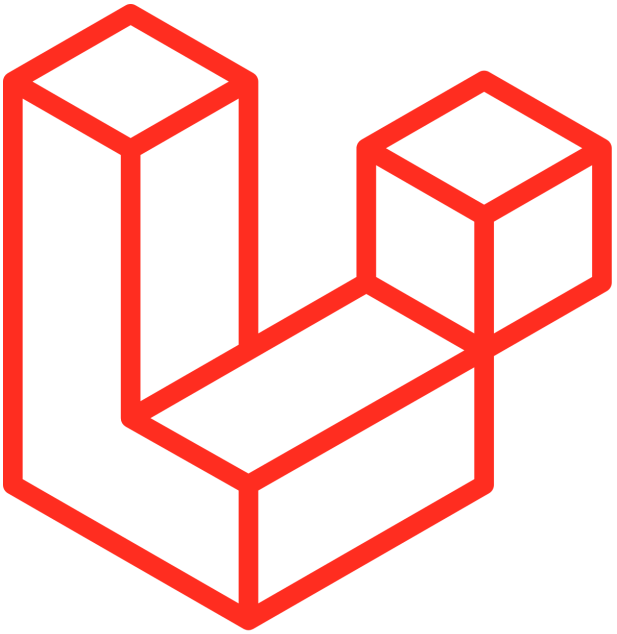
Documentación del Proyecto[1]





[1]Joseba Andoni Herranz Suescun

1- Idea del proyecto

a- Idea

La idea de este proyecto era hacer una aplicación web que permitiera visualizar los valores en bolsa de 10 empresas. Para cumplir este objetivo los requisitos era usar drag & drop para mover las empresas a la zona de selección, hacer uso del fetch para hacer llamadas a la API, la creación de API en laravel para la gestión de usuarios en la página con token de verificación para mantener la sesión activa, la introducción y generación de datos en una base de datos para posteriormente con el fetch recibir esos datos y mostrarlos usando Highchartjs. Para acabar el proyecto debía estar contenerizado en docker para poder desplegarlo con un “docker-compose up”

2-Front-end

a- DWC

a.a- Drag and Drop

En esta parte he creado 4 funciones:

-La primera de ellas permite mover las imágenes del div “inicio” y otorga la clase draggable a la imagen, la cual perderá al ser dropeada.

-La segunda añade la clase dropped para indicar que ha sido dropeada y guarda la id de esta en localStorage

```
$("#inicio img").draggable({
    revert: "invalid",
    refreshPositions: true,
    drag: function (event, ui) {
        ui.helper.addClass("draggable");
    },
    stop: function (event, ui) {
        ui.helper.removeClass("draggable");
        var image = this.src.split("/")[this.src.split("/").length - 1];

    }
});


$("#fin").droppable({
    drop: function (event, ui) {
        if ($("#fin img").length == 0) {
            $("#fin").html("");
        }
        ui.draggable.addClass("dropped");
        $("#fin").append(ui.draggable);

        localStorage.setItem(ui.draggable.attr('id'), ui.draggable.attr('id'));

    }
});
```

-La tercera selecciona las imagenes que esten dentro del div “fin” y les otorga la capacidad de ser drageadas otra vez

-La ultima funcion sirve para la eliminacion de las empresas seleccionadas y su devolucion al div “inicio” con una animacion y la eliminacion de estas empresas del localStorage

```
    $("#fin img").draggable({
        revert: "invalid",
        refreshPositions: true,
        drag: function (event, ui) {
            ui.helper.addClass("draggable");
        },
        stop: function (event, ui) {
            ui.helper.removeClass("draggable");
        }
    });

    $("#papelera").droppable({
        drop: function (event, ui) {
        ui.draggable.removeClass("dropped");
        ui.draggable.animate({
        top: 0,
        left: 0
        }, "slow");
        $("#inicio").append(ui.draggable);
        localStorage.removeItem(ui.draggable.attr('id'));
        }
    });
});
```

a.b- Fetch y llamadas a la API

Cuando hacemos llamadas a la API hay dos formas:

-GET: En esta solicitamos la información a la API y con la información que tenemos de lo que nos devuelve generamos el codigo que se mostrara en pantalla.

```
const options = {method: 'GET'};

fetch('http://hz114496:1912/api/mostrar', options)
.then(response => response.json())
.then(response => { response.data.forEach(element => {
        var van = document.getElementById(`valor${element.empresa_id}`);
        if(van != null){
            if(element.SoB == 1){
                van.innerHTML = `${element.valor}€`;
                document.getElementById(`valor${element.empresa_id}`).style.color = "green";
                setTimeout(function() {
                    document.getElementById(`valor${element.empresa_id}`).style.color = "black";
                }, 3000);
            }else {
                van.innerHTML = `${element.valor}€`;
                document.getElementById(`valor${element.empresa_id}`).style.color = "red";
                setTimeout(function() {
                    document.getElementById(`valor${element.empresa_id}`).style.color = "black";
                }, 3000);
            }
        }
        previousValue = element.valor;
    });
})
.catch(err => console.error(err));
```

-POST: En este tipo de solicitud, enviamos una información a la API para obtener una respuesta y generar un token que necesitaremos para mantener la sesión abierta, en caso de ser información errónea la que enviemos nos saltara un error en forma de alerta.

```
function login(){
    const email = document.getElementById('logE').value;
    const password = document.getElementById('logP').value;

    var formdata = new FormData();
    formdata.append("email", email);
    formdata.append("password", password);

    var requestOptions = {
      method: 'POST',
      body: formdata,
      redirect: 'follow'
    };

    fetch("http://hz114496:1912/api/login", requestOptions)
      .then(response => response.json())
      .then(result => {
        localStorage.setItem("token", result['authorisation']['token']);
        document.getElementById("form").style.display = "none";
        document.getElementById("contenido").style.display = "block";
      })
      .catch(error => {
        alert("Tu correo o contraseña no son correctos");
      });
}
```

a.c- localStorage

En localStorage mantenemos la información de sesión guardada para después poder mostrarla o hacer cambios nada mas inicializarse la página, en estos casos la información es guardada de las empresas seleccionadas anteriormente y el inicio de sesión de cuando nos hemos logueado o registrado.

```
for(var x=0; x<10; x++){
    if(localStorage.getItem(x)!=null){
        document.getElementById(x).style.display = "none";
        document.getElementById('fin').innerHTML += show[x];
    }
}

if(localStorage.getItem("token")!=null){
    document.getElementById("form").style.display = "none";
    document.getElementById("contenido").style.display = "block";
}
```

a.d-Gráfico

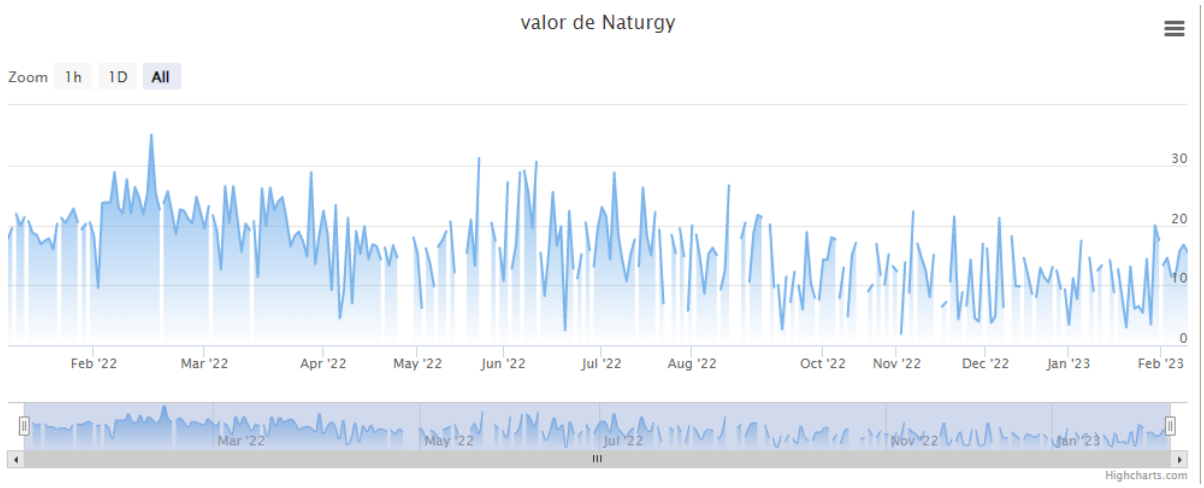
Usamos un Post en el fetch para enviar la id de la empresa, el cual mandará este a la API y nos devolverá los datos de la empresa correspondiente, los cuales modificaremos para adaptarlos al grafico (en este caso la fecha la pasaremos a milisegundos).

```
function grafico(img){
        var id = img.id;
        var nombre = img.alt;
          var dId = { id: id };

          var requestOptions = {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(dId),
            redirect: 'follow'
          };

            fetch("http://hz114496:1912/api/grafico", requestOptions)
                .then(res => res.json())
                .then(result => graf(result,id,nombre))
                .catch(error => console.error(error.message));
    }

function graf(result,id,nombre){
    let data = [];
    result['data'].forEach(element =>{
        if (element['empresa_id'] == id) {
          let fecha = new Date(element['fecha'])
          let date = fecha.getTime();
          data.push([date,element['valor']]);
        }
    });
    mostrarGraf(data, nombre)
}
```

b- DIW

b.a- Muestra del grafico

Al clicar en la imagen de la empresa, se mostrara un grafico con los valores obtenidos, el cual puede ser filtrado con los botones que tiene en la esquina superior izquierda o con la barra inferior y ajustarla a la franja de días u horaria que desee visualizar.



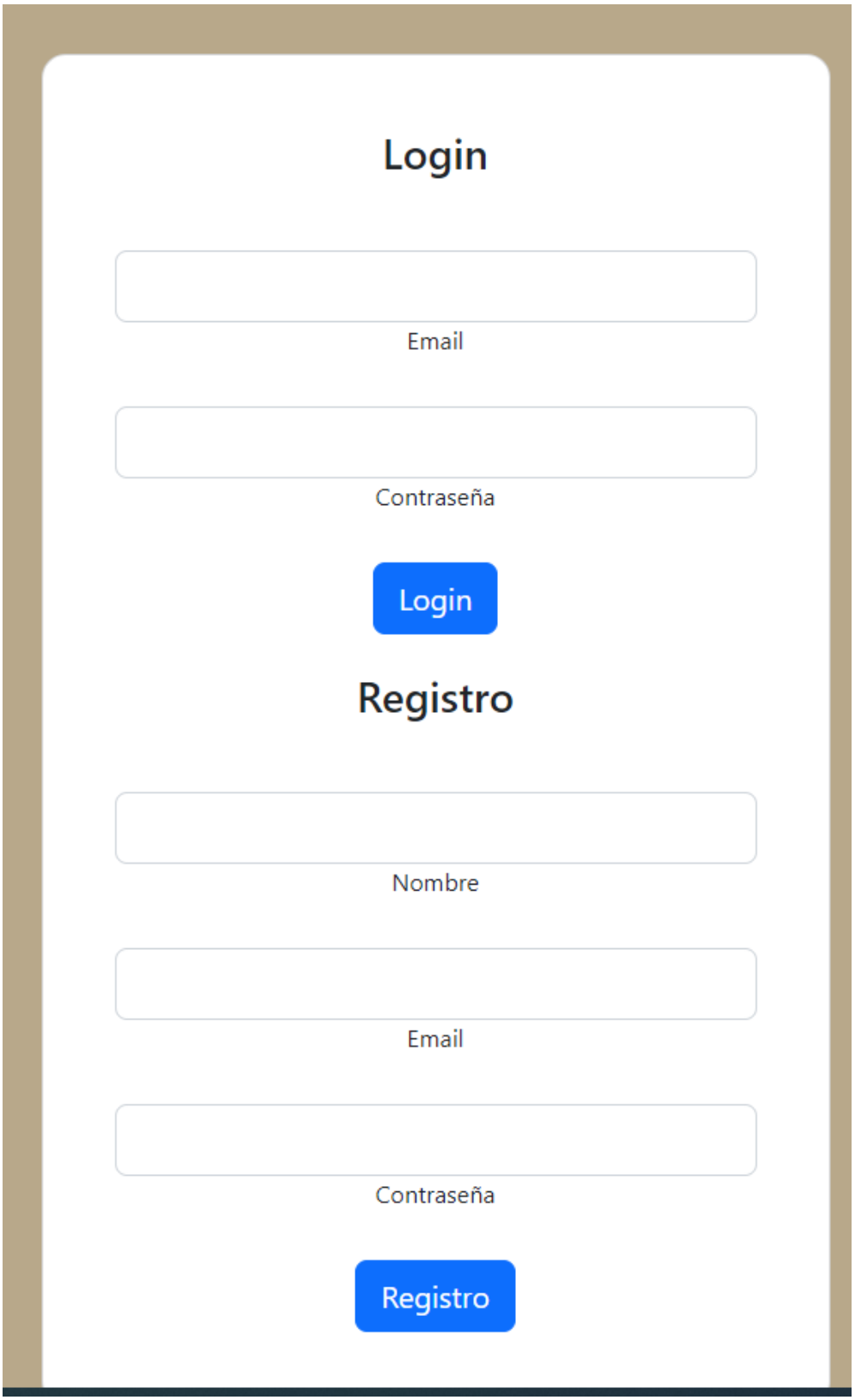
Highchart da un problema mostrando algunos datos en el gráfico, aunque pasando el puntero por encima de esos huecos o zonas en blanco se puede observar que los recibe pero en algunos casos no los dibuja.

b.b- Botones

La página tiene varios botones, todos ellos con bootstrap

b.c- Login/Register:

Nada más entrar en la pagina por primera vez nos aparecerá un login/register en el cual necesitara que introduzcas sus datos para poder darte acceso a la pagina y si ya entraste anteriormente te mostrara la pagina principal



b.d- Pagina principal

En la zona de “Selección de empresas” tendremos a nuestra disposición las 10 empresas de las cuales podremos obtener la información.

Para seleccionarlas bastara solo con arrastrarlas a la zona inferior y luego seleccionando “guardar” se mostrarán los datos de la empresa. En caso de que alguna de las empresas seleccionadas no quiera verla más, arrastrandola hacia la papelera de la zona inferior derecha la devolvera a la seccion superior.

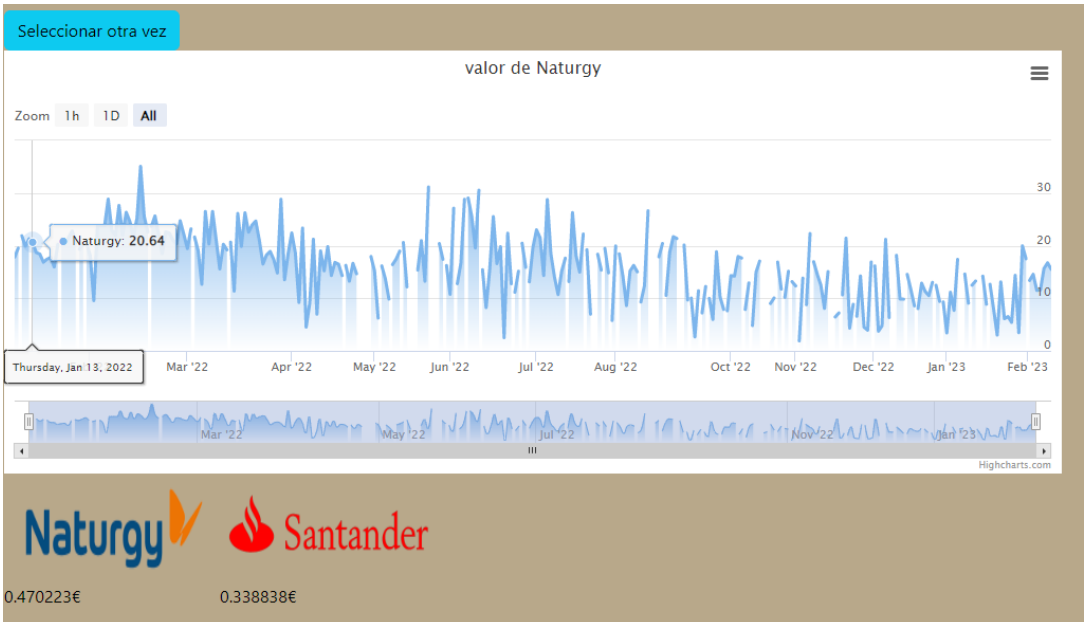
Para desloguearse bastara simplemente con seleccionar el botón de la esquina superior izquierda y volverá al menu de Login/Registro.

b.e- Pagina de muestra de datos

En esta pagina se mostraran las empresas que ha seleccionado, estas empresas varían su valor cada minuto y durante 3s su color cambiará, rojo si es un cambio negativo o verde en caso de ser positivo.

Cuando clique en una de las imágenes se mostrara un gráfico con los valores históricos de la empresa desde enero del año pasado

En caso de querer seleccionar otras empresas bastara con seleccionar “Seleccionar otra vez” y le devolvera a la seccion anterior donde podra elegir otra vez que empresas desea visualizar



3- Back-end

a- Laravel

a.a- Generación de datos

Usaremos la biblioteca de Carbón para poder facilitarnos el tema de obtención y uso de fechas, nada más empezar obtenemos la cantidad de líneas que hay en la tabla “empresas”, después de esto definimos la fecha de inicio y comprobamos si en la base de datos tenemos alguna fecha mayor que esa, al obtener esto obtenemos la fecha actual de ese mismo momento y empezamos a generar hasta que lleguemos a la fecha actual, al después de generarlos y insertarlos si dicha fecha esta en la semana anterior al dia actual los datos se generan cada hora y si es el mismo dia cada minuto

```php
$id = DB::table('empresas')->count();

for($j = 0; $j < $id; $j++) {
    $fechaInicial = Carbon::parse('2022-01-01 12:35:35');
    $valor = 15;
        $fechaDB = DB::table('valores')
            ->where('empresa_id', $j)
            ->orderBy('fecha', 'desc')
            ->first();
            if($fechaDB != null){
                $fechaInicial = $fechaDB->fecha;
                $valor = $fechaDB->valor;
            }

    $fechaActual = Carbon::now();

    while ($fechaInicial <= $fechaActual) {

        $variacion = mt_rand(1, 10) / 100;

        $resto = $valor * $variacion;

        $SoB = mt_rand(1, 2);

        if ($valor < 0.3) {
            $SoB = 1;
        }

        if ($SoB == 1) { //Subida
            $valor = $valor + $resto;
        } else if ($SoB == 2) { //Bajada
            $valor = $valor - $resto;
        }

        DB::table('valores')->insert([
            'empresa_id' => $j,
            'fecha' => $fechaInicial,
            'valor' => $valor,
            'SoB' => $SoB
        ]);



        if ($fechaInicial <= Carbon::now()->subWeek()) {
            $fechaInicial = Carbon::parse($fechaInicial)->addDay();
        } else if($fechaInicial <= Carbon::now()->subDay()){
            $fechaInicial= Carbon::parse($fechaInicial)->addHour();
        } else{
            $fechaInicial= Carbon::parse($fechaInicial)->addMinutes();
```

a.b- Creacion de usuarios

En la creación de usuarios gracias al POST obtenemos los datos necesarios para poder generar el usuario, en este caso obtendremos el nombre, correo y contraseña, haciendo que el correo sea único evitamos la redundancia y los problemas que podría causar esta, ademas de ello por mas seguridad la contraseña sera hasheada, al confirmar el registro se enviará el token de sesión

```php
public function register(Request $request){
    $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:6',
    ]);

    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
    ]);

    $token = Auth::login($user);
    return response()->json([
        'status' => 'success',
        'message' => 'User created successfully',
        'user' => $user,
        'authorisation' => [
            'token' => $token,
            'type' => 'bearer',
        ]
    ]);
}
```

a.c- Login

Gracias al POST que hemos hecho anteriormente recibimos el correo y contraseña para la verificación de usuario y verificamos si los datos son correctos para darle acceso a las página y devolvemos el token de sesión, en caso de que los datos sean erróneos se devolvera una señal de error.

```php
public function login(Request $request)
{
    $request->validate([
        'email' => 'required|string|email',
        'password' => 'required|string',
    ]);
    $credentials = $request->only('email', 'password');

    $token = Auth::attempt($credentials);
    if (!$token) {
        return response()->json([
            'status' => 'error',
            'message' => 'Unauthorized',
        ], 401);
    }

    $user = Auth::user();
    return response()->json([
            'status' => 'success',
            'user' => $user,
            'authorisation' => [
                'token' => $token,
                'type' => 'bearer',
            ]
        ]);

}
```

a.d- Rutas

En el archivo routes/api.php definimos las rutas de las api que llamaremos en los fetch

```php
Route::controller(AuthController::class)->group(function () {
    Route::post('login', 'login');
    Route::post('register', 'register');
    Route::post('logout', 'logout');
    Route::post('refresh', 'refresh');

});


Route::get('mostrar', [obtencionInfo::class, 'mostrarValores']);
Route::post('grafico', [obtencionInfo::class, 'grafico']);
Route::get('creacion', [obtencionInfo::class, 'creacion']);
```

a.c- Muestra de datos:

Los datos que obtendremos para mostrar al seleccionar la opción guardar envian una llamada a este controlador, el cual mostrara el ultimo valor de la empresa.

```php
public function mostrarValores()
{
    $query = "SELECT valores.* FROM valores JOIN ( SELECT empresa_id, MAX(fecha) AS fecha  FROM valores  GROUP BY empresa_id ) max_fecha ON valores.empresa_id = max_fecha.empresa_id AND valores.fecha = max_fecha.fecha";

    $data = DB::select($query);

    return response()->json([
        'data' => $data
    ]);
}
```

Para la parte del gráfico, debido a la cantidad de datos que obtiene está filtrado entre las horas de la apertura de la bolsa y los valores impares de la id de la tabla, así obteniendo una cantidad menor que Highchart pueda manejar

```php
1 reference | 0 overrides
public function grafico(Request $request)
{
    $request->validate([
        'id' => 'required',
    ]);

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

    $query = "SELECT * FROM valores WHERE empresa_id = $id AND HOUR(fecha) > 9 AND HOUR(fecha) < 17 AND (id % 2)";
    $data = DB::select($query);

    return response()->json([
        'data' => $data
    ]);
}
```

4-Docker

a- Docker-compose

```yaml
version: '3'
services:
  front:
    build: ./front
    ports:
      - 1911:80
    networks:
      - default


  api:
    build: ./back
    ports:
      - 1912:80
    networks:
      - default

  api1:
    build: ./back
    ports:
      - 1913:80
    networks:
      - default

  nginx:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    networks:
      - default
    depends_on:
      - api
      - api1
      - front
      - mysql

  database:
    image: mysql:latest
    environment:
      MYSQL_DATABASE: bolsa
      MYSQL_ALLOW_EMPTY_PASSWORD: "yes"

    volumes:
      - ./db:/docker-entrypoint-initdb.d

    networks:
      - default

  mysql:
    image: phpmyadmin/phpmyadmin
    ports:
      - 1914:80
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
    networks:
      - default
    depends_on:
      - database

networks:
  default:
    driver: bridge
```

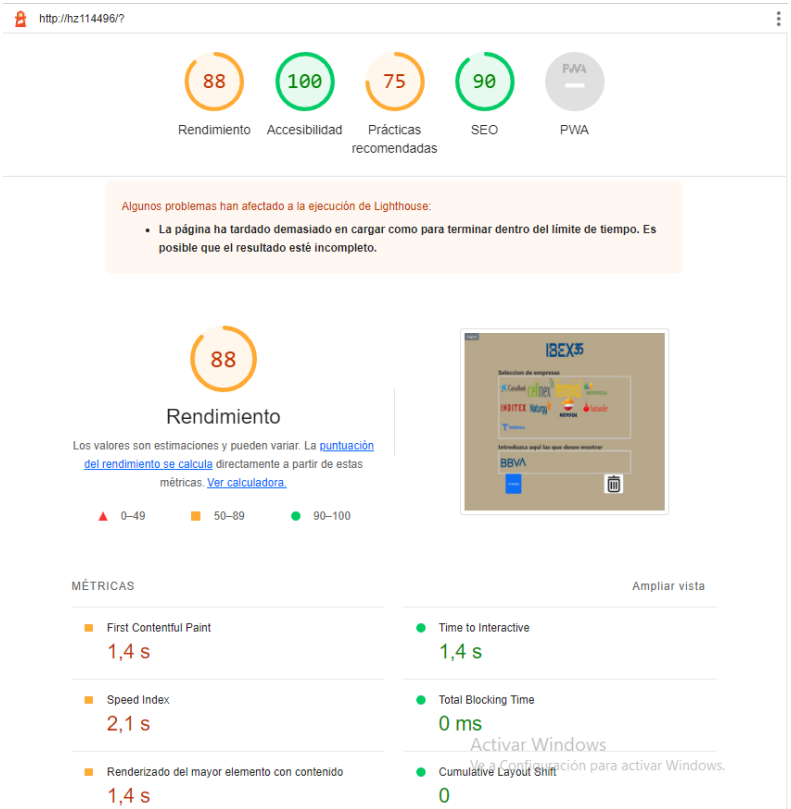
b- Balanceador de carga

```
events{}
http {

    upstream mysql {
        server mysql:80;
    }

    upstream back {
        server api:80;
        server api1:80;

    }

    server {
        listen  80;
        location / {
            proxy_pass http://front:80;
        }
        location /back {
            proxy_pass http://back;
        }
    }
}
```

5- Wave y Lighthouse

a- Wave

b- Lighthouse



5- Consejos de uso

Nada más inicializar el docker desde 0, la información de las empresas será generada desde 0, con lo cual esto podría causar que tardase un poco más en obtener la información de las últimas empresas de la lista.

6-GitHub

Link al repositorio