DOCUMENTACIÓN PROYECTO “QUESITO”

CESUR
Tu Centro Oficial de FP

INDICE

Front - end

Back – end

CESUR
Tu Centro Oficial de FP

DOCUMENTACIÓN PROYECTO QUESITO - FRONT END

Este documento tiene la finalidad de explicar y orientar tanto a los compañeros involucrados en éste proyecto, como a los ajenos que participarán en un futuro. En resumen, documentar el progreso del proyecto y el porqué de las diferentes aportaciones que se han realizado y cómo llevarlas a cabo para su comprensión y para la continuación del mismo a vista futura con una mayor claridad.

1. Introducción

El Proyecto Quesito es una iniciativa destinada a proporcionar a los alumnos de un centro educativo una plataforma de visualización intuitiva y accesible para consultar sus notas y recursos de aprendizaje de cada asignatura. Esta plataforma utiliza gráficos interactivos para presentar de manera clara y comprensible la información sobre el desempeño académico de los estudiantes.

1.2. Objetivos del Proyecto

Los objetivos principales del Proyecto Quesito son los siguientes:

Plataforma de Visualización de Notas y Recursos: Desarrollar una plataforma web que permita a los alumnos acceder de manera centralizada a sus notas y recursos de aprendizaje de cada asignatura, utilizando gráficos interactivos para mejorar la comprensión y el análisis de la información.

Transparencia y Claridad en la Evaluación: Proporcionar a los alumnos la capacidad de comprender el origen de sus notas y el proceso de evaluación, promoviendo así la transparencia y la participación activa en su propio aprendizaje.

1.3. Tecnologías Utilizadas

El desarrollo del Proyecto Quesito se fundamenta en las siguientes tecnologías:

Frontend:

Angular: Un framework de desarrollo de aplicaciones web de código abierto basado en TypeScript.

Material y Bootstrap: Bibliotecas de componentes de interfaz de usuario para Angular, que ofrecen un diseño atractivo y responsivo.

Chart.js: Una biblioteca JavaScript para crear gráficos interactivos y personalizables en la web.

Backend:

PHP: Lenguaje de programación utilizado para el desarrollo del backend.

MySQL: Sistema de gestión de bases de datos relacional para almacenar la información de los alumnos y las asignaturas.

JWT (JSON Web Tokens): Un estándar abierto para la creación de tokens de acceso que pueden ser verificados y firmados digitalmente, para garantizar la seguridad de la autenticación en la plataforma.

PhpOffice: Una biblioteca PHP para manejar archivos de office, que puede ser útil para generar informes o documentos relacionados con las notas de los alumnos.

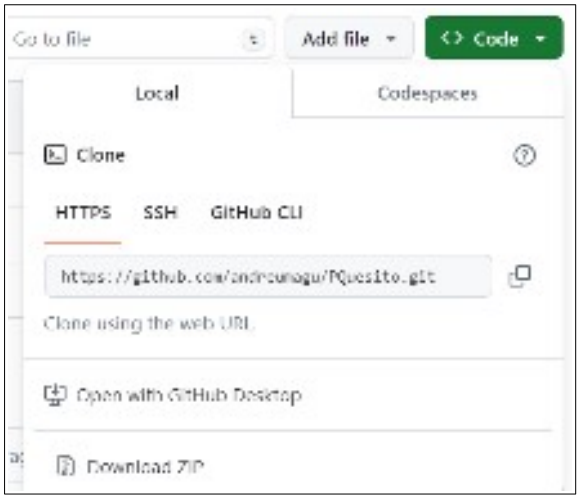
Con estas tecnologías, se pretende ofrecer una solución integral que cumpla con los requerimientos del proyecto y brinde una experiencia óptima a los usuarios finales.

2. Instalación del proyecto en Angular

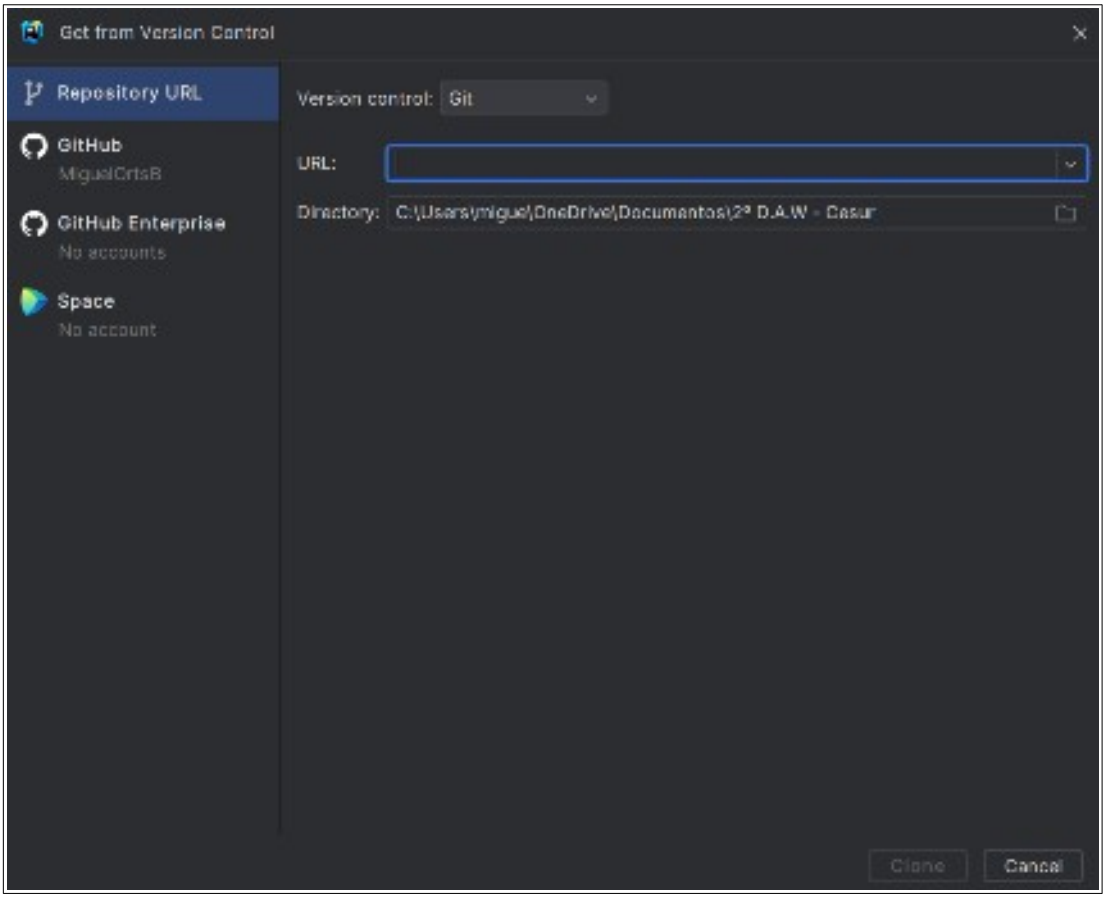
Primero, crearemos el espacio de trabajo donde se situará el proyecto. Para ello, abriremos cualquier IDE (como dato este documento se basa en su totalidad en el IDE WebStorm, y por tanto, recomiendo utilizar el mismo) y haremos click en New Project (Angular CLI) o similar, donde elegiremos la localización de éste en nuestro sistema. Una vez hecho esto, lo crearemos.
A continuación, lanzaremos el comando en la consola (siempre situandonos en la localización del mismo) para crear el proyecto Angular:

```
ng new nombre-proyecto
```

También cabe la posibilidad de que el proyecto ya haya sido creado, por un compañero por ejemplo, y esté publicado en GitHub, que será donde el equipo encargado de desarrollar este apartado cliente realizará sus respectivas Pull Request con sus respectivas ramas individuales según vayan avanzando en el trabajo y luego, el responsable, haciendo Merge en la rama de desarrollo o principal (Master en nuestro caso) que será la rama donde irán los cambios definitivos. Cabe señalar que para ese cometido es muy importante la comunicación y cooperación de todo el equipo para evitar posibles conflictos y que unos pisen en trabajo de otro, evitando posibles errores y trabajo de más innecesario.
Resumiendo, si el proyecto ya existe en GitHub, bastaría con clonarlo. Esto lo realizaremos copiando la URL del Code del repositorio:



A continuación, iremos al IDE y hariamos click en New Project from Version Control, donde se abriría una ventana parecida a la siguiente:



Introduciriamos la URL del repositorio y haríamos click en Clonar para terminar el proceso.

*Nota** Es posible que si el compañero encargado de la creación del proyecto lo ha hecho bajo Angular CLI, cuando nosotros lo clonemos nos saltará un mensaje de advertencia avisándonos de que deberíamos instalarlo. Lo único que deberemos hacer lanzar el siguiente comando para realizar la instalación de Angular CLI y Material, que se utiliza para crear proyectos, generar código de biblioteca y aplicación y realizar una variedad de tareas de desarrollo continuas, como pruebas, agrupación e implementación:

```
npm install -g @angular/cli
ng add @angular/material
```

Una vez terminado el proceso de clonación o creación del proyecto, es importante situarnos dentro del directorio mismo para poder crear los diferentes componentes que explicaré mas abajo. Para ello, simplemente nos introduciremos en él con el siguiente comando en la Consola del IDE:

```
cd nombre-proyecto
```

3. Breve introducción a Angular

A continuación, voy a proceder a explicar los comandos para la creación de los diferentes componentes, servicios, interfaces, modelos etc que serán con los que trabajaremos diariamente para desarrollar nuestra aplicación.

Una buena práctica es organizar todos nuestros componentes del mismo tipo en Carpetas, por ello, el primer paso será crear estas para cada uno de ellos.

Componentes

ng generate component nombre-componente

Para situarlo dentro de su respectiva carpeta:

ng generate component nombre-carpeta/nombre-componente

Servicios

ng generate service nombre-servicio

Interfaces

ng generate interface nombre-interfaz

Clase / Modelo

ng generate class nombre-clase

Servidor de desarrollo

ng serve --open

Este comando inicia el servidor, observa sus archivos y reconstruye la aplicación a medida que realiza cambios en esos archivos.

4. Estructura, configuración y desarrollo (aportaciones personales)

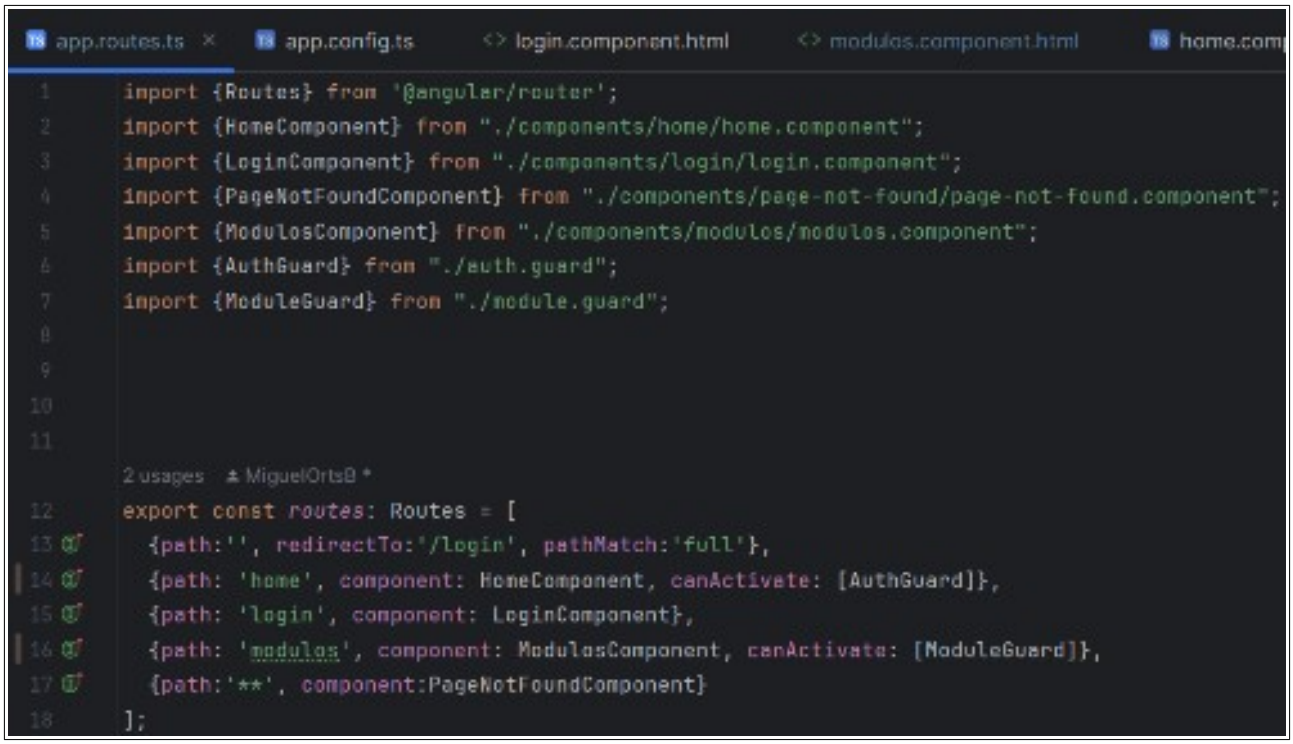
Ahora ya entraremos en la parte más personal y técnica del documento, donde se explicarán las diferentes aportaciones y la lógica de ellas para una fácil comprensión para todo aquel que se vea involucrado en el proyecto futuramente y no tenga por mano Angular o el desarrollo front-end y back-end en general.

Navegación entre componentes

Uno de los problemas iniciales que se producieron fue la navegación entre las diferentes pestañas de la app, esto ocurría porque no se gestionaban bien las rutas y su configuración. Angular se basa en modelos de aplicación SPA (de una sola página), es decir, cambiamos lo que ve el usuario mostrando u ocultando partes de la pantalla que corresponden a componentes particulares, en lugar de ir al servidor para obtener una nueva página. A medida que los usuarios realizan acciones

en la aplicación, deben moverse entre las diferentes vistas que hayamos definido. Para manejar la navegación de una vista a la siguiente se utiliza el **Router**.
En un principio cuando queríamos cambiar de pestaña (componente) al realizar alguna acción (loguearse y acceder al Home, por ejemplo) lo realizaba en la misma página con un scroll-bar. Esto no es lo que se buscaba, y para solucionarlo se realizó lo siguiente:

- Hay que entender que en Angular cada componente visible (denominado plantilla o template) es como un capa que va encima de otra, y la capa principal visible de todas es el archivo 'app.component.html'. Es muy importante que unas no se pisen con las otras y estén claramente indentificadas. Para ello, accedemos al archivo 'app-routes.ts', que será el archivo donde se gestionarán las navegaciones y rutas de nuestra app:



Aquí podemos observar que en la constante de rutas *Routes* hay contenida una matriz en la que se definen las diferentes rutas de la app. La primera propiedad, *path* define la URL de la ruta (lo que se introduce después del dominio). La segunda propiedad, *component* define el componente que Angular debe usar para la ruta correspondiente. Entonces:

1- En la primera ruta el path está vacío (' '), esto es así porque queremos utilizar la URL relativa inicial para que la ruta por defecto al iniciar la app sea el componente de Login. Por ello si introducimos lo siguiente en el navegador (como ejemplo):

http://localhost:4200

Equivaldría y nos llevaría a:

http://localhost:4200/login

Para ello si el path está vacío, se realiza un *redirectTo* a la página de Login automáticamente.

2- Las cinco siguientes rutas sirven para asignar las páginas de Login, Home, Módulos, RA y CE respectivamente. Digamos que es la asignación normal y estándar. La propiedad de *canActivate* es algo particular y la comentaremos en el apartado siguiente de Protectores de Ruta.

http://localhost:4200/login
http://localhost:4200/home
http://localhost:4200/modulos
http://localhost:4200/ra
http://localhost:4200/ce

3- La última ruta es la que se denomina como "comodín". Una aplicación que funcione bien debería manejar con elegancia cuando los usuarios intentan navegar a una parte de su aplicación que no existe (por ejemplo: http://localhost:4200/loginnnnn) . Para agregar esta funcionalidad a su aplicación se configura esta ruta ('**') y se lleva al usuario a una página de error (componente PageNotFound creado anteriormente y que explicaremos posteriormente) con el mismo formato que es resto de la app y que informa al usuario de ello, en lugar de las típicas páginas propias del navegador de error.

Por último cabe señalar que si queremos acceder a alguna de éstas páginas, por ejemplo al hacer click en el botón de Iniciar Sesión del Login para acceder al Home, lo haremos de la siguiente manera:

Crearemos una función dentro de archivo TypeScript del LoginComponent que en su interior contiene el método *navigate* del servicio Router (que es el que hace posible y permite todas estas navegaciones) que nos llevará a dicha página. Acordarse de importar Router en el archivo para poder utilizarlo.
Ya por último, con un evento (click) del HTML asignamos la función anterior para que al hacer click haga posible la navegación:

Protectores de ruta

Los protecores de ruta se utilizan para evitar que el usuario pueda acceder a las diferentes páginas de nuestra app si nosotros no lo deseamos, por lo tanto es una medida de seguridad muy importante. Por ejemplo, no es conveniente que un usuario pueda acceder a la página de Home a través de la URL del navegador sin haber hecho antes un inicio de sesión correcto con usuario y contraseña y se haya autenticado.
Para ello crearemos lo que se llaman 'guardias' usando Angular CLI ejecutando lo siguiente:

ng generate guard auth

Se nos crearán dos archivos, y dentro del TypeScript (.ts) con el mismo nombre crearemos la función c*anActivate* que hemos comentado previamente en el punto 3 de 'Navegación entre componentes'. Dentro de esta función, generaremos una condición de autenticación, por ejemplo comprobar si se encuentra un token en el almacenamiento local, si se se encentra se permite la navegación, si no se encuentra se deniega la navegación a la página que posea esta función (en el ejemplo la página de Home, regresando a la página previa de Login).

```
canActivate(): boolean {
  // Verificar si el token está presente en el almacenamiento local
  const token : string | null  = localStorage.getItem( key: 'token');

  if (token) {
    // El usuario ha iniciado sesión, permitir la navegación
    return true;
  } else {
    // El usuario no ha iniciado sesión, redirigir a la página de inicio de sesión
    this.router.navigate( commands: ['/login']);
    return false;
  }
}
```

Para asignar esta función *canActivate* lo haremos a través del archivo 'app.routes.ts' que hemos comentado previamente, en la matriz de rutas *Routes* a través de la propiedad con el mismo nombre seguido como valor el nombre del import que hemos realizado del archivo donde se encuentra nuestra guardia:

```
import {AuthGuard} from "./auth.guard";
```

```
{path: 'home', component: HomeComponent, canActivate: [AuthGuard]},
```

Página Not-Found

Se decidió diseñar y maquetar esta sencilla página para darle un toque más elegante a la app, siguiendo la misma estética del resto de la misma en caso de que se produjera algún tipo de error:

Para diseñarla se creó un componente llamado ‘page-not-found’ y para asignarla como ruta comodín para que cuando se produjera algún tipo de error se redirigiera a esta página lo explico en el punto 3 del apartado ‘Navegación entre componentes’.

Archivo de constantes de URL

Como en el proyecto se utilizan gran cantidad de servicios que acceden a API externas (archivos PHP de los compañeros de back-end) y por tanto, utilizamos gran cantidad de variables que contienen las URL de esas API, es una mala práctica estar creando una gran cantidad de éstas para cada una de las URL a lo largo de toda la app.
Por ello, creé un archivo TypeScript llamado *constantUrl* que en su interior contiene una constante con la URL del servidor web que nos proporcionaron y que reutilizaré en todo el proyecto:

```
//Archivo donde se almacenarán las constantes de las URL del servidor
4 usages   ± MiguelOrtsB
export const MY_CONSTANT: string = 'http://cicloweb.cesurformacion.com/API';
```

Cabe decir que en un futuro pueden añadirse más constantes dependiendo del sufijo que nos interese. Actualmente estamos utilizando el directorio API, que es donde de momento se encuentran todos los web services, pero puede que en un futuro se tenga que acceder a otro directorio dependiendo de la configuración de carpetas del servidor.
Para ‘reutilizar’ esa constante se exportará e importará respectivamente en el servicio que nos interese:

```
import {MY_CONSTANT} from "../constantUrl";
```

```
public serviceName : string  = '/jwt/signin.php';
public url : string  = MY_CONSTANT + this.serviceName;
```

Por ejemplo, en el servicio de Login el web service al que se accede está ubicado en la carpeta /jwt y por tanto, se crea otra variable de tipo string para concatenar ese trozo de sufijo a la constante creada anteriormente para formar la URL completa en una variable que será la que utilizemos para ese cometido.

Servicios (Frases aleatorias, etc)

Uno de los requerimientos que se nos propuso, entre todos, es que cuando el usuario accediera a la página de Home se generaran una serie de frases motivadoras para fomentar una buena energía y motivación a los alumnos.
Para ello se creó un servicio en Angular que llama a un web service externo mediante método GET, el cuál devuelve una frase aleatoria de un conjunto de ellas ubicadas en una tabla de la base de datos (esto lo realiza el equipo de back-end):

```
export class FrasesService {

  private url : string  = 'http://cicloweb.cesurformacion.com/API/frases.php';

  no usages   ± MiguelOrtsB
  constructor(private http: HttpClient) { }

    1 usage   ± MiguelOrtsB
    obtenerFraseAleatoria(): Observable<any> {
        return this.http.get<any>(this.url);
    }
```

A continuación, en el archivo .ts de la página Home, importaremos este servicio de Angular y lo pasamos al constructor:

```
import {FrasesService} from "../../services/frases.service";
```

Inicializaremos una variable vacía de tipo string que contendrá la frase:

```
fraseAleatoria: string = '';
```

A continuación, en el método *ngOnInit* (el cuál también hay que importar previamente y que es el que permite ejecutar código cuando el componente se renderice por primera vez) crearemos una función la cuál en su interior invoca al servicio y su función que obtiene esa frase aleatoria. Si la encuentra la asigna con un condicional a la variable 'fraseAleatoria' creada anteriormente, si no la encuentra muestra un mensaje de error por consola informando del fallo.

```
import {Component, OnInit} from '@angular/core';
```

```
1 usage   MiguelOrtsB
obtenerFraseAleatoria() : void  {
      this.frasesService.obtenerFraseAleatoria().subscribe(
          next: data => {
              if (data.frase) {
                  this.fraseAleatoria = data.frase;
              } else {
                  console.error('Error al obtener la frase aleatoria.');
              }
          },
          error: error => {
              console.error('Error en la solicitud HTTP:', error);
          }
      );
  }
}
```

Por último, para poder mostrar esa frase obtenida en la página lo realizaremos en el HTML, por ejemplo en una etiqueta <p> de la siguiente manera:

```
<p>{{fraseAleatoria}}</p>
```

Esto anterior es lo que se llama interpolación, que es el mecanismo para sustituir una expresión por un valor de tipo string entre dobles llaves en la plantilla o template (HTML) para así poder visualizarlo.

Finalmente, en la imagen siguiente podremos ver el resultado final de este apartado en la pagina de Home:

Visualización de medias, RA y CE en tabla y gráfico (Páginas Modulos, RA y CE)

En este apartado es donde ya se verán por fin resultados del alumno y sus notas, en este caso, las notas medias de cada asignatura en general y un gráfico representándolas.

Mismo procedimiento, creamos un servicio 'DatosService' con una función 'getMedias' que a través del token que se debería encontrar en el almacenamiento local, y el año y id del alumno como parámetros, enviamos una solicitud POST al web service de nuestros compañeros de back-end, el cuál debería devolver un JSON con la información solicitada (medias, ra y ce).

```
2 usages  ± andreumaqu +1
getMedias(ano: string, id: string):Observable<any>{

  const token :string|null  = localStorage.getItem( key: 'token');

  // Configurar el header con el token de autorización
  const headers :HttpHeaders  = new HttpHeaders( headers: {
    'Authorization': `Bearer ${token}`
  });

  // Definir el cuerpo de la solicitud
  const body :{ano: string, id: string}  = {
    "ano": ano,
    "id": id
  };

  // Enviar la solicitud POST con el header y el cuerpo
  return this.httpClient.post<any>(this.url, body,  options: { headers: headers });
}


}
```

IMPORTANTE* para cada servicio que hagamos es importante importar y pasar por parámetro del constructor el *HttpClient* para poder realizar solicitudes HTTP.

```
import {HttpClient, HttpHeaders} from "@angular/common/http";
```

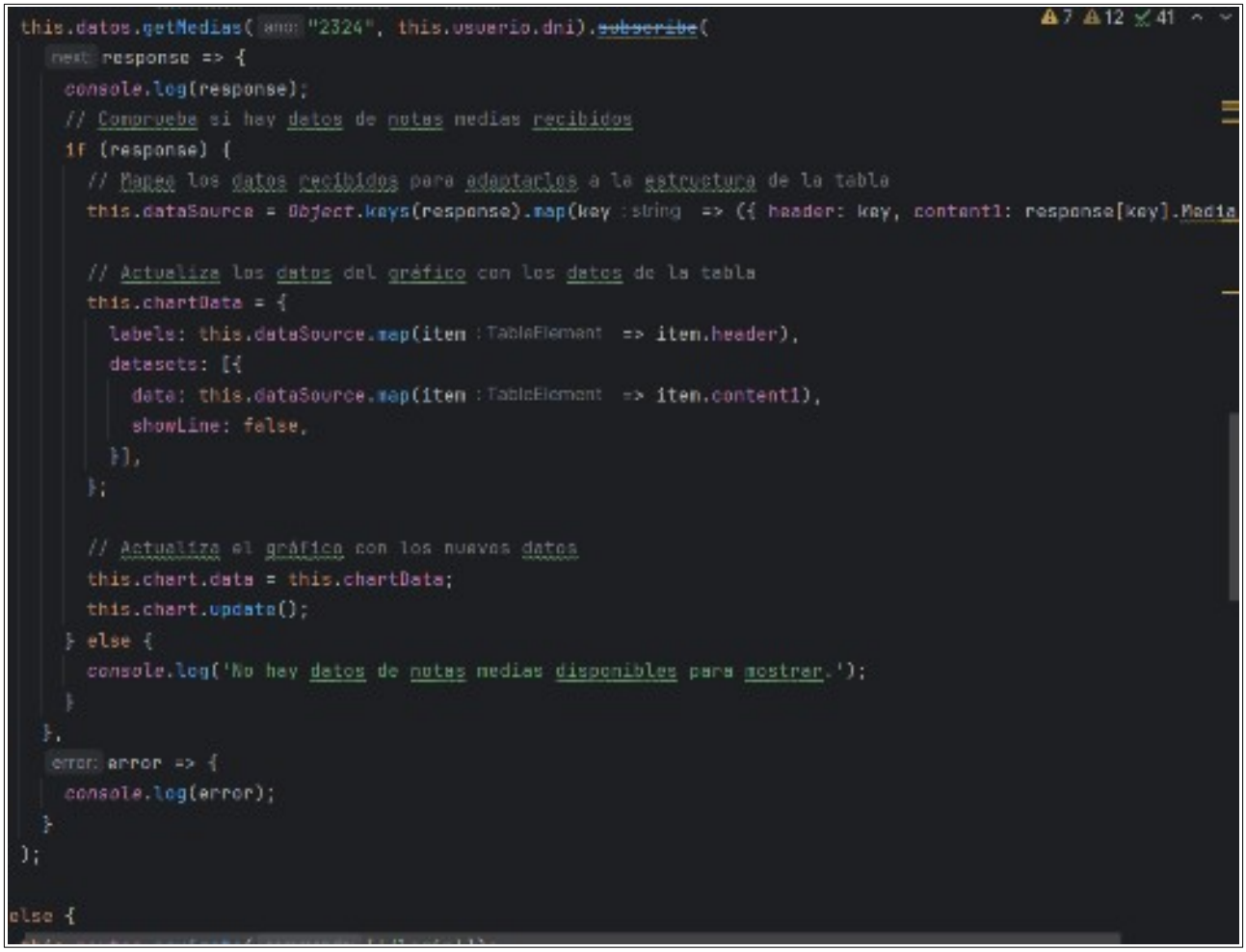
Seguidamente, en el .ts del componente 'Modulos' importamos el servicio y los pasamos por parámetro en el constructor:

```
import { DatosService } from "../../services/datos.service";
```

```
constructor(private router: Router, private datos: DatosService) {}
```

Creamos una función *getMedias* que llama a dicho servicio la cuál pasamos por parámetros DNI y ID del alumno como hemos comenteado previamente. Si recibe una respuesta de la solicitud, asignamos los valores a la variable 'dataSource' que es la que contiene la tabla que e visualizará en la plantilla.

```
displayedColumns: string[] = ['header', 'content1'];
dataSource: TableElement[] = [];
```



Voy a proceder a explicar esa línea en la que se asigna los valores a 'dataSource' porque puede generar confusión:

- Object.key(response): devuelve un array con los nombres de las propiedades de la respuesta.

- map(key => ({ header: key, content1: response[key].Media })): este método 'map' itera sobre cada elemento del array devuelto por 'Object.key(response)'. Para cada elemento (llamado 'key'), crea un objeto con dos propiedades : 'header', que es igual nombre de la propiedad r*esponse* y 'content1' que es igual al valor de *Media* correspondiente a esa propiedad en *response*.

En resumen, esta línea transforma el objeto *response* en un array de objetos, donde cada objeto tiene una propiedadd 'header' con el nombre de la propiedad en *response* y una propiedad 'content1' con el valor de *Media* asociado a esa propiedad. Esto se hace para preparar los datos en un formato adecuado para su visualización en la tabla.

Las siguientes líneas son para actualizar los datos del gráfico con los datos de la tabla y actualizarlos.

5. Despliegue del proyecto en el servidor

Para poner en marcha el proyecto definitivamente tendremos que subirlo al servidor propio.

Para ello, primero tendremos que compilarlo desde nuestro IDE (WebStorm), ubicándonos dentro del mismo con el siguiente comando desde la Terminal:

ng build --base-href /pquesito/

Donde `--base-href` se refiere a la URL base de la aplicación que se está creando (pquesito).

Puede que al compilar nos salte el siguiente error:

swap exceeded maximum budget. Budget 4.00 kB was not met by 25.55 kB with a total of 29.55 kB.

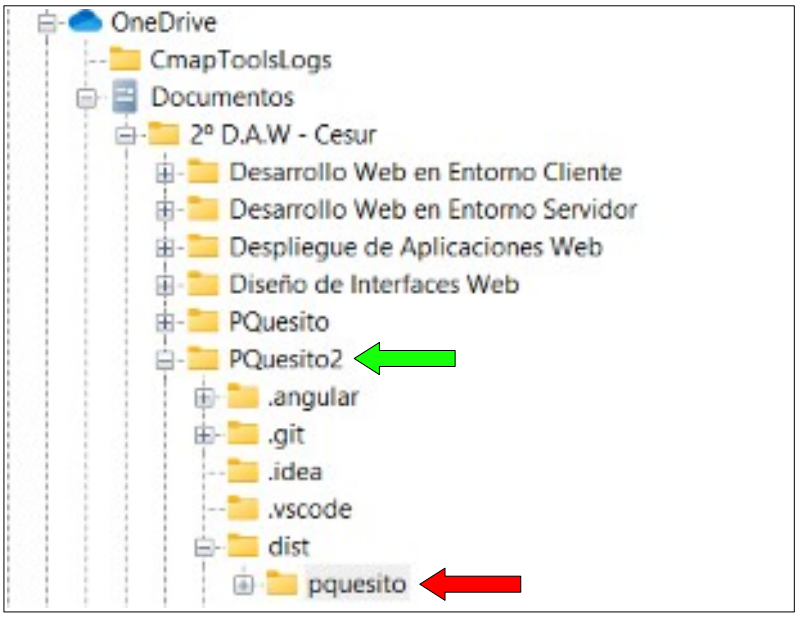
Budget es un conjunto de límites a ciertos valores que afectan el rendimiento del sitio, que no pueden excederse en el diseño y desarrollo de cualquier proyecto web. En nuestro caso, es el límite para el tamaño de los paquetes.

Esto se soluciona accediendo al archivo 'angular.json' y con *Control + F* buscamos 'budget', donde seguidamente cambiaremos los valores de *maximumError* aumentándolos de su valor inicial (5kb).

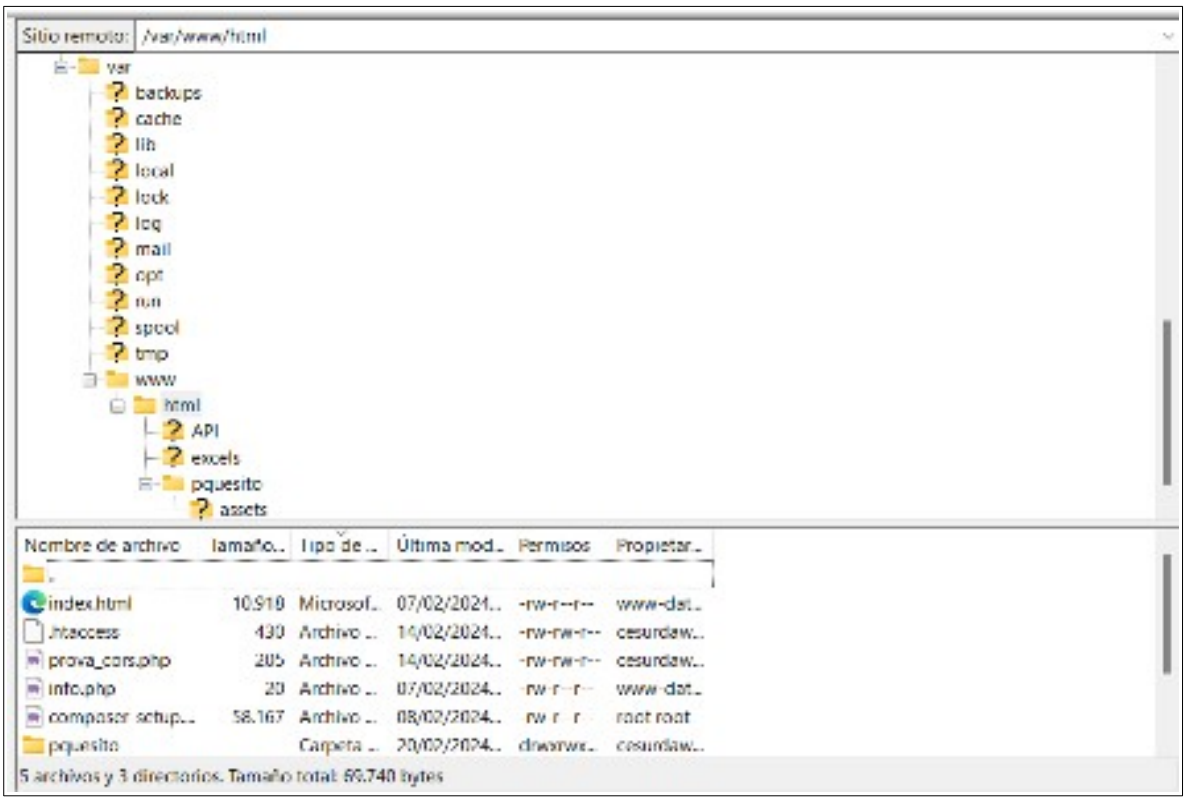
Seguidamente veremos que se nos habrá creado el directorio 'dist' en la estructura de nuestro proyecto, que será el mismo pero ya compilado:

Lo siguiente a realizar será subirlo ya al servidor, podemos utilizar alguna aplicación cliente de transferencia de archivos como FileZilla. Desde la interfaz de ésta, deberemos realizar la conexión a dicho servidor introduciendo los datos del mismo:

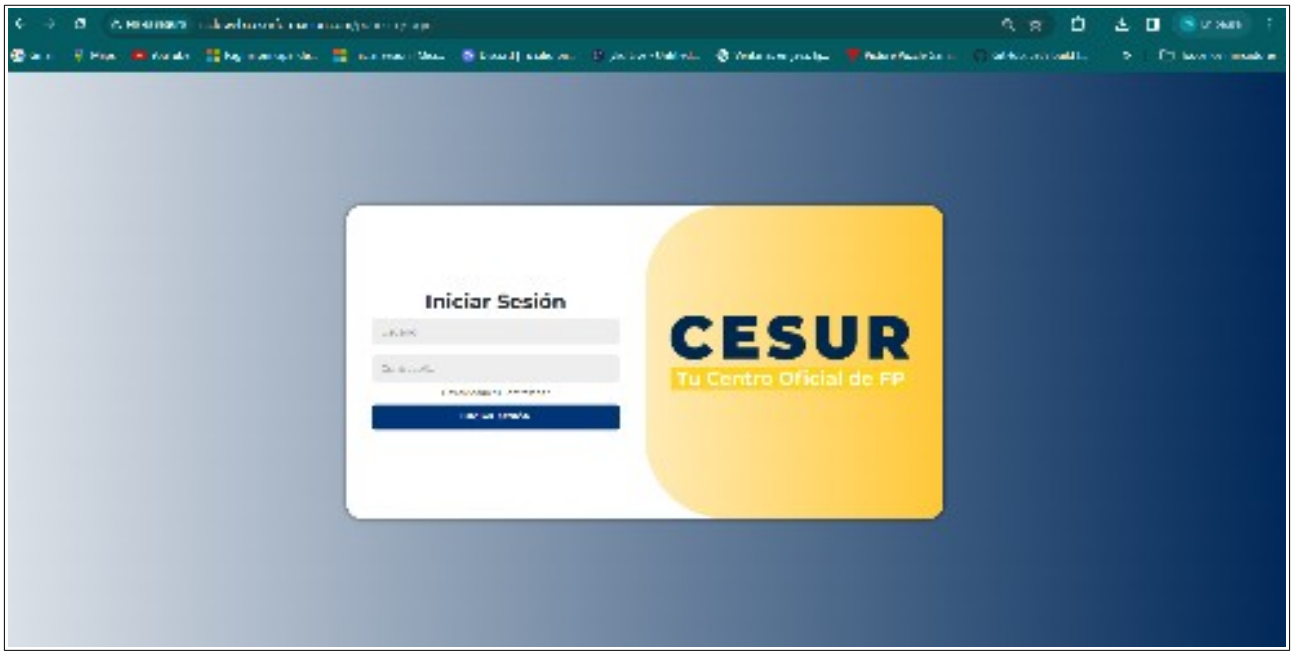
Una vez hecha la conexión, el explorador de archivos de la parte izquierda pertenece al de nuestro sistema, y el de la derecha al servidor. Por lo tanto, tendremos que buscar donde se encuentra ubicado nuestro proyecto (PQuesito2) en nuestro sistema y dentro de éste, accederemos al proyecto compilado (pquesito) de la carpeta *dist*:



Una vez ubicado, lo arrastraremos (pquesito) a la carpeta */var/www/html* de nuestro servidor y será allí donde finalmente se ubicará para el despliegue.
Cabe señalar, que se nos genera un directorio llamado *browser* dentro de nuestra carpeta de proyecto *pquesito*, tenemos que sacar todo lo que haya allí dentro y ubicarlo dentro de esta última que será la principal:



Finalmente, ya podremos acceder a la aplicación desde el navegador poniendo el dominio de nuestro servidor seguido de la URL base de la aplicación que hemos generado con `--base-href`, en nuestro caso /pquesito:



6. Despliegue de la API en el servidor

Creamos una carpeta principal (API, por ejemplo) allí se creará el composer. Cada cosa que use un composer creamos otra carpeta (JWT, EXCELS, etc) si vemos que da problema lo creamos todo en la principal. Dentro de la carpeta jwt tendremos tema de tokens, tanto creación como comprobar y sus respectivos web services. Luego tenemos la otra carpeta para los excels con todos sus web services para leer estos.

Importante para tema CORS los header en todos los archivos, el 'require' con la ruta del autoload.php y luego lo que utilicemos (use + librería que utilicemos).

Finalmente, para desplegarlo se realizará exactamentede la misma manera que el punto 4 a cerca de la app de Angular, con un cliente como Filezilla, transferiremos los archivos al servidor.

DOCUMENTACIÓN PROYECTO "QUESITO" - BACK END

1. Instalación

Para el desarrollo de la parte de servidor de nuestro proyecto, hemos de tener en cuenta que hacen falta una serie de requisitos para poder empezar a desarrollar. Nosotros hemos hecho uso de las siguientes herramientas:

Primeramente necesitamos un servidor web. Para ejecutar la API, en nuestro caso utilizamos Apache. Todo nuestro código en la parte de servidor está realizado con PHP, por tanto necesitamos tener PHP instalado en nuestro sistema. Luego, recomendamos tener instalado un gestor de dependencias como Composer. En nuestro caso hemos utilizado Composer para instalar y administrar las bibliotecas de creación de token jwt (firebase/php-jwt) y para lectura de archivos excel (phpoffice/phpspreadsheet). Y por último también se hace uso de una base de datos por tanto se necesita tener instalado MySQL para su implementación.

2. Guía de instalación de la Base de Datos

Crear la Base de Datos

En la carpeta DB, está el archivo cesur.sql . Este es el archivo que debemos importar en MySQL. Una vez creada la base de datos continuamos con la configuración del conector a la misma.

Configuración del conector

Para la utilización actual de los Web Services debemos configurar una base de datos. Para ello primero configuramos el archivo db.php. Este archivo es el conector a nuestra base de datos. Dentro de él, debemos editar los campos $dsn, $user y $pass que hay en la clase DB_Configuration. Completando con los datos de nuestra Base de Datos.

```
public $dsn = "mysql:host=localhost;dbname=cesur";

public $user = "usuario";

public $pass = "contraseña";
```

Una vez cambiados los campos, los Web services serán funcionales.

3. Guia de uso de los Web Services

Rutas y métodos

Método	Nombre	Ruta
POST	signin.php	API/jwt/signin.php
POST	frases.php	API/frases.php
POST	mediasexcel.php	API/php-excel/mediasexcel.php
POST	raexcel.php	API/php-excel/raexcel.php

signin.php

Este archivo es el encargado de crear el token jwt, para ello necesita leer un archivo JSON con la siguiente estructura:


```
{
    "email":"xxx",
    "password": "xxx"
}
```


Al hacer uso de este web service envía al servidor una respuesta también en JSON como la siguiente:


```
{
    "message": "Successful login",
    "token":
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJsb2NhbGhvc3QiLCJhdWQiOiJUSEVfQVVESUVOQ0UiLCJpYXQiOjE3MDYwODMxODcsIm5iZiI6MTcwNjA4MzE5NywiZXhwIjoxNzA2MDgzMjQ3LCJkYXRhIjp7ImRuaSI6IjExMTIyMjMzMyIsIm5vbWJyZSI6IkV2YSIsImFwZWxsaWRvMSI6IkV2YW5zIiwiYXBlbGxpZG8yIjoiTWlsbGVyIiwiRW1haWwiOiJldmEuZUBleGFtcGxlLmNvbSJ9fQ.TRHoI_gXwLmbaflaUa9iZzGvoR_KmQ9bR34EgwVJQwA",
    "email_address": "eva.e@example.com",
    "expire": 1706083247
}
```


frases.php

Este web service devuelve al front-end una frase aleatoria, de una tabla con frases motivadoras de nuestra base de datos. Codificándola en JSON con el siguiente formato:

```
{
    "frase": "No importa lo lento que vayas, siempre y cuando no te detengas."
}
```


mediasexcel.php

Este WS lee todos los archivos Excel (.xlsx) según el año y una id de usuario. También necesita obtener mediante las cabeceras HTTP un token válido.


```
{
    "ano": "2324",
    "id": 1
}
```


Devolverá un JSON con las asignaturas donde se encuentra matriculado ese usuario y la media de la correspondiente asignatura


```
{
    "DWES": {
        "Media": 3.0833333333333335
},
    "DWEC": {
        "Media": 3.0833333333333335
},
    "DAW": {
        "Media": 3.0833333333333335
},
    "DIW": {
        "Media": 3.0833333333333335
    }
}
```


raexcel.php

Este archivo realiza un web service que envía un JSON con las notas de todos los RA y sus respectivos porcentajes. Para ello necesita obtener el módulo, el año del curso y la id del alumno. Al igual que mediasexcel.php necesita también un token válido obtenido desde las cabeceras HTTP.


```
{
"modul": "DWES",
```

"ano": "2324",

"id": 1

}

El resultado será codificado y enviado en JSON con el siguiente formato:

{

"Notas": {

"RA1": {

"Nota": 4,

"Porcentaje": 0.111

},

"RA2": {

"Nota": 5.29,

"Porcentaje": 0.111

},

"RA3": {

"Nota": 5.92,

"Porcentaje": 0.111

},

"RA4": {

"Nota": 5.26,

"Porcentaje": 0.111

},

"RA5": {

"Nota": 6.29,

"Porcentaje": 0.111

},

"RA6": {

"Nota": 6.54,

"Porcentaje": 0.111

},

```
                "RA7": {
                        "Nota": 0,
                                "Porcentaje": 0.111
},
                "RA8": {
                        "Nota": 0,
                                "Porcentaje": 0.111
},
                "RA9": {
                        "Nota": 0,
                        "Porcentaje": 0.111
        }
}
```