DOCUMENTACIÓN DEL PROYECTO

1. INTRODUCCIÓN

Este PDF, ha sido realizado con la intención de explicar todo el desarrollo realizado para la prueba técnica realizada para Solardron Tech, S.L. En este, se explicará como se realizó cada requisito pedido y además se adjuntará un corto vídeo ejecutando la aplicación web para ver su funcionalidad.

2. CONFIGURACIÓN DE FIREBASE

Lo primero que se realizó, fue configurar nuestro Backend. Para ello, accedimos a la página de Firebase Console, donde nos creamos nuestro proyecto, como se visualiza en la siguiente imagen.

Ilustración 1. Creación del proyecto en Firebase

En este punto, simplemente se inició sesión con nuestra cuenta y le dimos un nombre a nuestro proyecto, en nuestro caso, Prueba-técnica-Firebase.

2.1 Añadir Web App

Ya que nuestro objetivo es realizar una aplicación web, asignamos a nuestro proyecto de Firebase la opción de añadir una aplicación web, así como se observa en la ilustración 2.

Ilustración 2. Añadir Web App

En este punto, simplemente le ponemos un nombre a nuestra aplicación web y si queremos de forma opcional añadimos el SDK de Firebase, en nuestro caso no lo hicimos, ya que posteriormente se añade automáticamente usando el CLI de Firebase.

2.2 Añadir Autenticación

El siguiente paso que hicimos fue añadir la autenticación a nuestro proyecto, así podemos observar que usuarios acceden a nuestra aplicación y de qué forma. Además, este punto, nos permite implementar el registro e inicio de sesión utilizando Firebase Authentication. Así permitir autenticación con correo electrónico y redes sociales como Google.

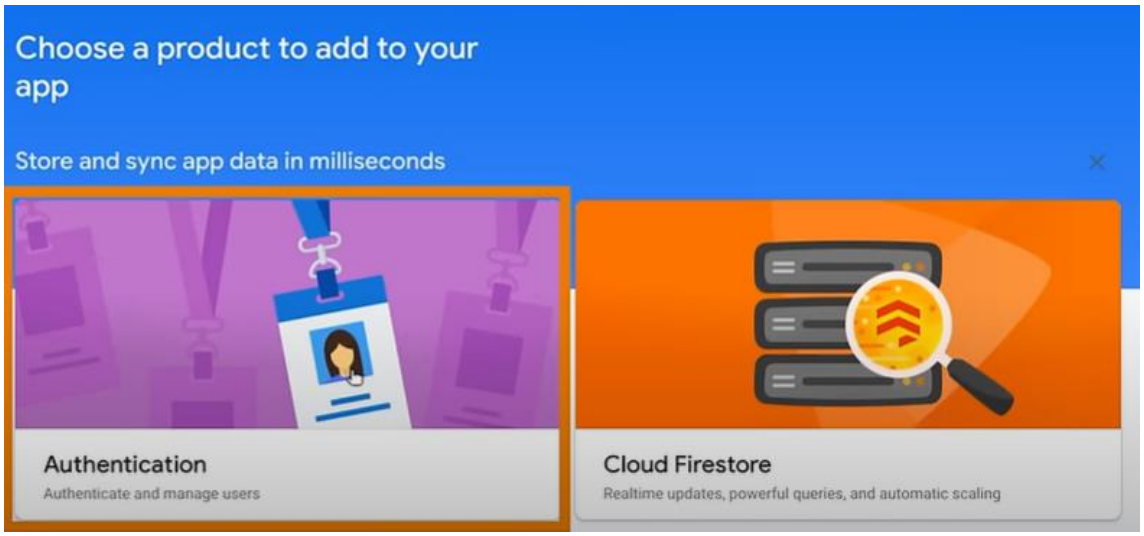


Ilustración 3. Añadir autenticación

2.3 Añadir Cloud Firestore

A continuación, necesitamos una base de datos para el almacenamiento de datos. Firestore, nos permite añadir una base de datos NoSQL en la nube que nos permite almacenar y guardar nuestros datos. Una de las mejores características de esta BBDD es que nos actualiza en tiempo real nuestros datos, tanto inserciones, modificaciones o borrados.

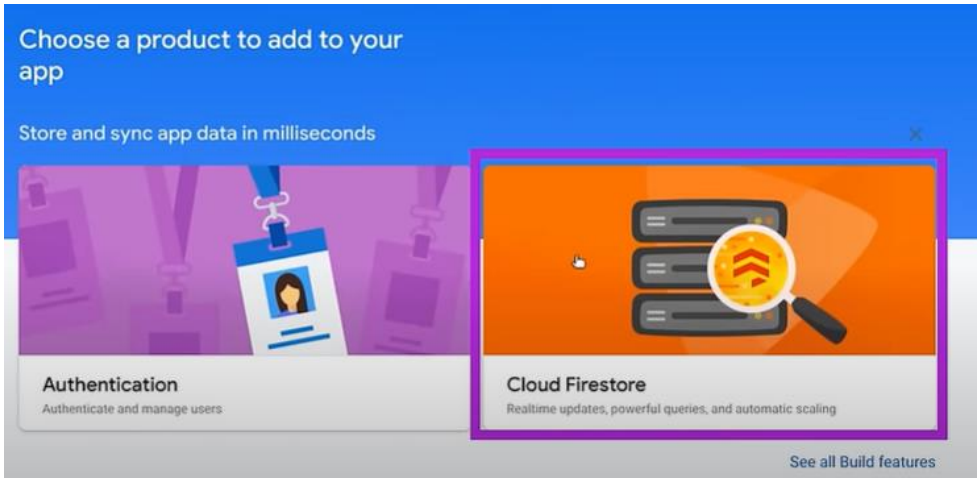


Ilustración 4. Añadir Cloud Firestore

En este punto creamos nuestra base de datos con un nombre y seleccionamos sus reglas. En este caso, al ser un proyecto de prueba se eligió la opción de “Start in test mode”, una opción que nos permite crear nuestra base de datos de forma más rápida y que se eliminarán los datos en un mes aproximadamente. **OJO**, cuidado con esta opción ya que todo el mundo podría tener acceso a estos datos. Finalmente se selecciona la ubicación de donde vamos a trabajar y ya tendríamos nuestra base de datos lista. Así se observa en la ilustración 5.

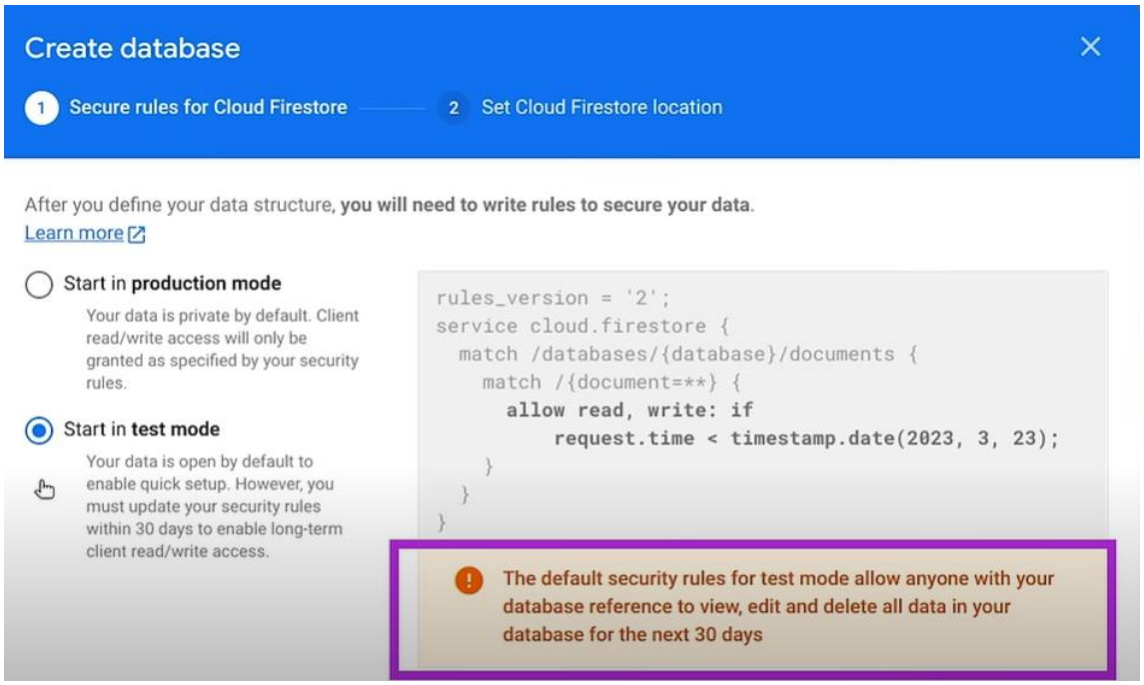


Ilustración 5. Configuración de la BBDD

2.4 Añadir Cloud Storage

Además. Firestore nos proporciona una base de datos para almacenar imágenes que suban nuestros usuarios en la aplicación web y así las podamos analizar, visualizar o incluso descargar desde la base de datos. El funcionamiento para crearnos una base de datos de este tipo es igual que la base de datos anterior sólo que seleccionando en el panel lateral de Firestore la opción de Storage.

2.5 Instalación del CLI de Firebase

El siguiente paso es instalar el CLI de Firebase, el cual nos permite gestionar de forma más sencilla nuestros proyectos en Firebase. Este se instala mediante el comando “*npm i -g firebase-tools”*. Una vez se ha instalado, iniciamos sesión con la cuenta que nos hemos creado nuestro proyecto en Firebase, mediante el comando “*firebase login*”

3. CREACIÓN DEL PROYECTO EN ANGULAR

Una vez hemos configurado totalmente nuestro Backend, vamos a empezar el desarrollo Frontend mediante Angular. Para ello, lo primero es crear nuestro proyecto Angular, pero, antes de nada, necesitamos tener el CLI de Angular instalado, el cual se instala con el comando "*npm i -g @angular/CLI*". Si ya lo tenemos instalado, procedemos a crear nuestro proyecto con el comando "*ng new prueba-tecnica-firebase*", siendo el nombre a gusto del usuario. A partir de ahí, te preguntará una serie de opciones de configuración del proyecto y ya tenemos listo nuestro proyecto para empezar a trabajar.

OJO, cuidado con que versión se instala de Angular, ya que yo tuve varios problemas de compatibilidad de versiones, incluso con boostrap, ya que me instalé la nueva versión (18) y esta omite varios archivos de configuración de Angular, como al app.module o el environments. Así que para ello, cuando se crea el nuevo proyecto se pone en false la librería de standalone, así quedaría el comando "*ng new prueba-tecnica-firebase --standalone=false*". Para generar los environments, se crean mediante el comando "*ng generate environments*".

Finalmente ya podemos añadir el schematics de angular mediante el comando "*ng add @angular/fire*". Este punto es importante, ya que es donde vamos a elegir que características de Android vamos a utilizar, en nuestro caso, Authentication, Firestore y Cloud Storage. Así se observa en la siguiente imagen las diferentes características.

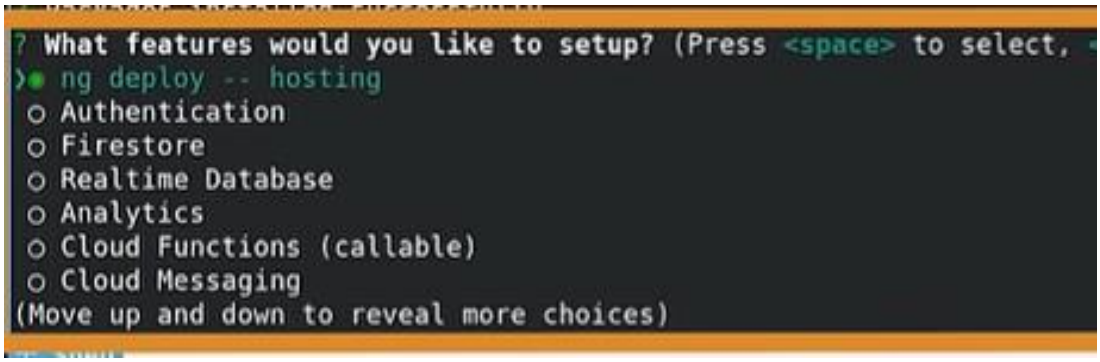
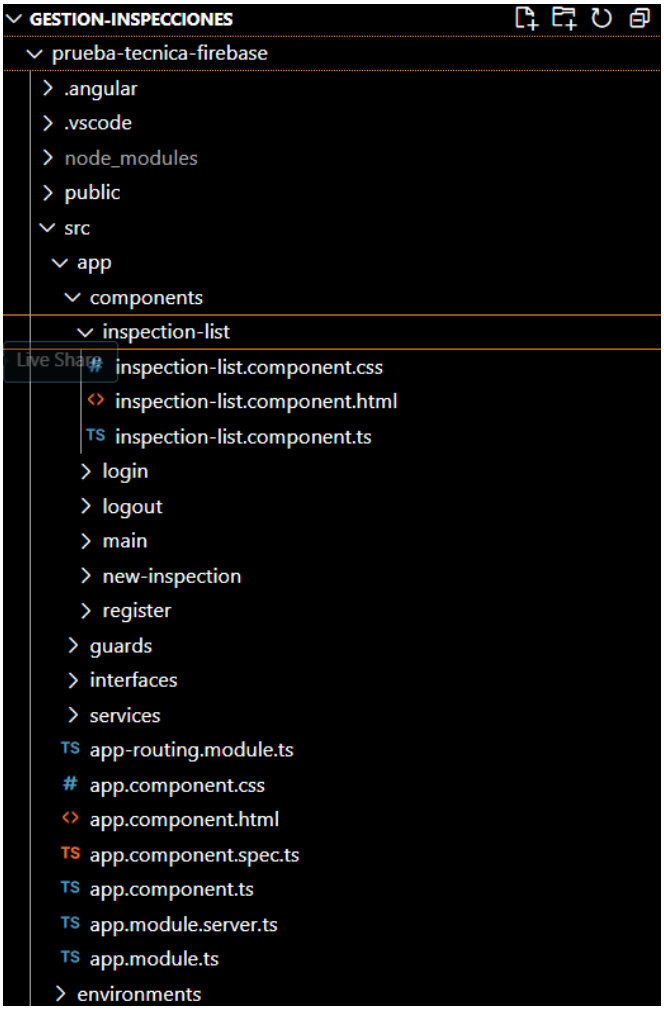


Ilustración 6. Configurar características de Firestore

Una vez elegidas las características, seleccionamos nuestro correo con el que hemos creado nuestro proyecto, nuestro proyecto en Firebase y nuestra aplicación web. Para comprobar que todo ha ido bien, se nos ha tenido que rellenar nuestro fichero environments con las características de nuestra app y se nos habrá completado nuestro archivo app.module con la inicialización de características de Firestore.

4. IMPLEMENTACIONES DE REQUISITOS PEDIDOS

A continuación, vamos a explicar de forma general como hemos implementado los requisitos pedidos. Los resultados se observarán en el vídeo proporcionado. Antes de nada, veamos la estructura de nuestro proyecto en la siguiente ilustración.



Observamos que hemos organizado el código por carpetas, donde dentro de nuestra carpeta app, se encuentran los archivos proporcionados por angular. A partir de ahí, la carpeta components, tendrá las diferentes funciones de la aplicación, a su vez, cada carpeta cuenta con archivo .ts con el código en TypeScript, un .html para incluir elementos a nuestra página y un css para darle buen aspecto visual.

Por otro lado, la carpeta guards incluye un archivo .ts para el redireccionamiento en el inicio de sesión para el caso de que falle el registro, se mande el inicio de sesión.

La carpeta interfaces, incluye un archivo .ts donde definimos una interfaz para definir los elementos de una inspección.

La carpeta services, va a contener las funciones para la carpeta component, es decir, se van a definir las funciones para el inicio de sesión, registro, CRUD de inspecciones, etc.

Por último, existen más archivos que no se observan en la foto, como index.html, main.ts o styles.css que son archivos que se añaden al crear el proyecto.

4.1 Autenticación de usuarios

Este punto consiste en implementar registro e inicio de sesión utilizando Firebase Authentication y la autenticación con correo electrónico y redes sociales (Google).

Para ello, las carpetas involucradas en este punto son login, logout y register, ambas incluidas en las carpetas components. Todas estas carpetas cuentan con los 3 archivos explicados anteriormente. Por tanto, vamos a explicar como se ha realizado este punto, donde sobre todo nos centraremos en el archivo de TypeScript, ya que, si tenemos que explicar los 3 archivos con mucho texto de todas las carpetas, quedaría una documentación un poco extensa, la parte de HTML y CSS se observará mejor en el vídeo.

4.1.1 Login

Este componente de Angular se llama LoginComponent. Su función principal es manejar la autenticación de usuarios a través de un formulario de login, permitiendo iniciar sesión con un email y contraseña o mediante Google. El componente también se encarga de redirigir a los usuarios a diferentes vistas dependiendo del resultado de la autenticación.

El componente utiliza varias importaciones esenciales de Angular, como Component, OnInit, ElementRef, ViewChild, y Router, además de un servicio personalizado de autenticación (AuthService).

El decorador @Component define el selector app-login, que es una etiqueta utilizada para insertar este componente en la plantilla HTML. También especifica los archivos HTML y CSS que definen la estructura y el estilo del componente.

El componente utiliza el decorador @ViewChild para obtener referencias a los elementos del formulario de email y contraseña. Esto le permite acceder y manipular directamente estos elementos desde el código TypeScript del componente.

El constructor del componente recibe dos dependencias: AuthService y Router. AuthService se utiliza para manejar la autenticación de los usuarios, y Router se utiliza para la navegación entre diferentes vistas dentro de la aplicación.

El método ngOnInit forma parte del ciclo de vida del componente y se ejecuta una vez que Angular ha inicializado completamente el componente. En este caso particular, el método está vacío, pero está definido para cumplir con la interfaz OnInit.

El componente tiene dos métodos principales para manejar la autenticación:

Método logIn: Este obtiene los valores de los elementos del formulario de email y contraseña, seguidamente llama al método login del AuthService pasando estos valores. Si la autenticación es exitosa, redirige al usuario a la vista new-inspection.

Método logInGoogle: Similar al método logIn, pero llama a un método diferente (loginWithGoogle) del AuthService, donde si la autenticación es exitosa, redirige al usuario a la vista new-inspection.

Por mencionar un poco en consiste el HTML, se basa en un formulario con dos input para el correo y contraseña y los dos botones que llaman cada uno a los métodos de login explicados anteriormente. El CSS se utiliza para darle forma y color al formulario y utilizando media query para diseño responsive. Se deja a continuación el código del .ts:

```
import { Component, OnInit, ElementRef, ViewChild} from
'@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../../services/auth.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  @ViewChild("Email") Email! : ElementRef;
  @ViewChild("Password") Password! : ElementRef;

  constructor(private auth : AuthService, private _router: Router) {
}

  ngOnInit(): void {
  }

  logIn():void{
    var Email = this.Email.nativeElement.value;
    var Pass = this.Password.nativeElement.value;
    this.auth.login(Email, Pass).then(res=> {
      console.log(res);
      this._router.navigate(['new-inspection']);
    });
  }

  logInGoogle():void{
    var Email = this.Email.nativeElement.value;
    var Pass = this.Password.nativeElement.value;
    this.auth.loginWithGoogle(Email, Pass).then(res=>{
      console.log(res);
      this._router.navigate(['new-inspection']);
    });
  }
```

4.1.2 Register

Siguiendo con la carpeta register, el funcionamiento del archivo .ts, se basa en esto:

Este nuevo componente, RegisterComponent, es similar al LoginComponent que describimos antes, pero está diseñado para manejar el registro de nuevos usuarios en lugar del inicio de sesión.

El componente se llama RegisterComponent, donde su selector para este componente es app-registro. Utiliza archivos diferentes para su plantilla HTML y su hoja de estilos CSS (register.component.html y register.component.css).

En lugar de tener métodos logIn y logInGoogle, RegisterComponent tiene un método llamado registro, donde se obtiene los valores de los elementos del formulario de email y contraseña, y llama al método register del AuthService para registrar al nuevo usuario.

Si el registro es exitoso, redirige al usuario a la vista new-inspection, al igual que en logIn del LoginComponent. Por último, RegisterComponent utiliza un método de registro (register). Este es el código del archivo .ts:

```
import { Component, OnInit, ElementRef, ViewChild } from
'@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../../services/auth.service';

@Component({
  selector: 'app-registro',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {

  @ViewChild("Email") Email! : ElementRef;
  @ViewChild("Password") Password! : ElementRef;

  constructor(private auth : AuthService, private _router : Router)
{ }

  ngOnInit(): void {
  }

  registro():void{
    var mail = this.Email.nativeElement.value;
    var contra = this.Password.nativeElement.value;
    this.auth.register(mail, contra).then(res=>{
      console.log(res);
      this._router.navigate(['new-inspection']);
    });
  }

}
```

4.1.3 Logout

Este nuevo componente, LogoutComponent, está diseñado para manejar el cierre de sesión de los usuarios. Se basa en este sencillo funcionamiento:

El componente se llama LogoutComponent., donde su selector es app-logout. El constructor recibe AuthService para manejar la autenticación y Router para manejar la navegación, igual que en los otros componentes. Este método llama al método logout del AuthService para cerrar la sesión del usuario. Finalmente redirige al usuario a la ruta raíz ("/") utilizando el Router. Este es su código:

```
import { Component, OnInit} from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../../services/auth.service';

@Component({
  selector: 'app-logout',
  templateUrl: './logout.component.html',
  styleUrls: ['./logout.component.css']
})
export class LogoutComponent implements OnInit {

  constructor(private auth : AuthService, private _router : Router)
{ }

  ngOnInit(): void {
  }

  logOut(){
    this.auth.logout();
    console.log("Saliendo...");
    this._router.navigate(["/"]);
  }
}
```

Todas las funciones llamadas en estos métodos se realizan en la carpeta service, en un archivo .ts que funciona como un servicio, la cual contiene este código:

```
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { Injectable } from '@angular/core';
import firebase from "@firebase/app-compat";


@Injectable()

export class AuthService {
  constructor(private Auth: AngularFireAuth ) {}

  // Sign up with email/password
  async register(email: string, password: string){
    try{
```

```
      return await this.Auth.createUserWithEmailAndPassword(email,
password);
    }
    catch(error){
      alert("No se ha podido hacer el registro correctamente. Error:
" + error)
      console.log("No se ha podido hacer el registro correctamente.
Error: " + error);
      return null;
    }
  }

  // Sign in with email/password
  async login(email: string, password: string){
    try{
      return await this.Auth.signInWithEmailAndPassword(email,
password);
    }
    catch(error){
      alert("No se ha podido hacer el log-in correctamente. Error: "
+ error)
      console.log("No se ha podido hacer el log-in correctamente.
Error: " + error);
      return null;
    }
  }

  // Sign out
  async logout() {
    return this.Auth.signOut();
  }

  async loginWithGoogle(email: string, password: string) {
    try{
    return await this.Auth.signInWithPopup( new
firebase.auth.GoogleAuthProvider());
  }
  catch(error){
    alert("No se ha podido hacer el log-in correctamente. Error: " +
error)
    console.log("No se ha podido hacer el log-in correctamente.
Error: " + error);
    return null;  }
}

  getUserLogin(){
    return this.Auth.authState;
}


}
```

Dondeeste código se basa en esto:

Este código define un servicio de autenticación en Angular llamado AuthService, el cual utiliza Firebase Authentication a través de AngularFire para manejar la autenticación de usuarios. A continuación, se describe cada parte del código y su funcionalidad:

AngularFireAuth: Proporciona las funcionalidades de autenticación de Firebase.

Injectable: Permite que esta clase sea inyectada como un servicio en otros componentes.

firebase: Importa Firebase para usar su proveedor de autenticación de Google.

El decorador @Injectable indica que esta clase puede ser inyectada como un servicio. Además, el constructor inyecta AngularFireAuth para interactuar con Firebase Authentication.

El método register permite a un usuario registrarse con un email y una contraseña, usando createUserWithEmailAndPassword de Firebase Authentication. Si hay un error durante el registro, muestra un mensaje de alerta y lo registra en la consola.

El método login permite a un usuario iniciar sesión con un email y una contraseña, usando signInWithEmailAndPassword de Firebase Authentication. Si hay un error durante el inicio de sesión, muestra un mensaje de alerta y lo registra en la consola.

El método logout, cierra la sesión del usuario actual, usando signOut de Firebase Authentication.

El método loginWithGoogle, permite a un usuario iniciar sesión usando Google, usando signInWithPopup con el proveedor de autenticación de Google de Firebase. Si hay un error durante el inicio de sesión con Google, muestra un mensaje de alerta y lo registra en la consola.

El método getUserLogin, retorna el estado de autenticación del usuario actual, usando authState de AngularFireAuth para obtener un observable que emite el estado de autenticación del usuario.

4.2 Gestión de Inspecciones

La idea de este requisito es poder crear, editar y eliminar informes de inspección. Los informes deben contener información como tipo de inspección, fecha, ubicación y resultados. Además, estos van acompañados de una foto opcional que será subida y almacenada en Firebase Storage.

Esto se realiza en la carpeta new-inspection, localizada dentro de components, pero esta se limitará solo a la creación de los informes, para editarlos y eliminarlos se realizará en otro componente. La funcionalidad de este componente se basa en esto:

Este componente de Angular, NewInspectionComponent, está diseñado para manejar la creación de nuevas inspecciones, incluyendo el almacenamiento de imagenes en Firebase Storage y el envío de los datos a un servicio de inspecciones. A continuación, se explican los puntos importantes:

La importaciones que tenemos son las siguientes:

Component, OnInit: Importados de @angular/core, son esenciales para definir el componente y su ciclo de vida.

FormBuilder, FormGroup, Validators: Importados de @angular/forms, se utilizan para crear y gestionar formularios reactivos.

InspectionsService: Un servicio personalizado que se encarga de interactuar con la base de datos.

AngularFireStorage: Importado de @angular/fire/compat/storage, se utiliza para manejar el almacenamiento de archivos en Firebase Storage.

El decorador @Component define el componente con su selector, plantilla HTML y hoja de estilos CSS correspondientes.

La clase NewInspectionComponent tiene Un FormGroup que representa el formulario de la inspección, una cadena para mostrar mensajes al usuario y una variable para almacenar el archivo de imagen seleccionado.

El constructor inyecta InspectionsService, FormBuilder y AngularFireStorage, donde utiliza FormBuilder para crear el formulario con varios campos, cada uno con su validador requerido.

El método onSubmit verifica si el formulario es válido y si se ha seleccionado una imagen, donde genera un nombre único para la imagen usando la marca de tiempo actual. Define la ruta de almacenamiento de la imagen en Firebase Storage. Seguidamente, sube la imagen a Firebase Storage y, una vez completada la carga, obtiene la URL de descarga de la imagen. Agrega esta URL al objeto formData y envía los datos de la inspección al servicio InspectionsService.

Por último, el método onFileSelected, almacena el archivo de imagen seleccionado en la variable imageFile. El archivo HTML, simplemente es un formulario con los distintos campos y botón para enviar los datos. El código del archivo .ts se encuentra a continuación.

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { InspectionsService } from
'../../services/inspections.service';
import { AngularFireStorage } from '@angular/fire/compat/storage';
// Importa AngularFireStorage
```

```
@Component({
  selector: 'app-new-inspection',
  templateUrl: './new-inspection.component.html',
  styleUrls: ['./new-inspection.component.css']
})
export class NewInspectionComponent implements OnInit {

  formulario: FormGroup;
  mensaje: string = '';
  imageFile: any; // Variable para almacenar el archivo de imagen
seleccionado

  constructor(
    private inspectionsService: InspectionsService,
    private formBuilder: FormBuilder,
    private storage: AngularFireStorage // Inyecta
AngularFireStorage
  ) {
    this.formulario = this.formBuilder.group({
      Type_of_inspection: ['', Validators.required],
      date: ['', Validators.required],
      latitude: ['', Validators.required],
      longitude: ['', Validators.required],
      result: ['', Validators.required],
      image: ['']
    });
  }

  ngOnInit(): void {
  }

  async onSubmit() {
    if (this.formulario.valid && this.imageFile) { // Asegúrate de
que se haya seleccionado una imagen
      try {
        const formData = this.formulario.value;
        const imageName = new Date().getTime().toString(); // Genera
un nombre único para la imagen
        const filePath = 'carpeta_principal/' + imageName; // Define
la ruta de almacenamiento de la imagen
        const imageRef = this.storage.ref(filePath); // Referencia
al archivo en Firebase Storage
        const uploadTask = imageRef.put(this.imageFile); // Sube la
imagen a Firebase Storage

        uploadTask.then(async () => {
          // Obtiene la URL de descarga de la imagen
          const imageUrl = await
imageRef.getDownloadURL().toPromise();
          // Agrega la URL de la imagen al objeto formData
          formData.image = imageUrl;
          // Agrega el formulario actualizado a Firebase Firestore
          const response = await
this.inspectionsService.addInspection(formData);
          this.mensaje = '¡La inspección se ha enviado con éxito!';
          console.log("Respuesta del servicio:", response);
```

```
        }).catch(error => {
          // Maneja los errores durante la carga de la imagen
          console.error("Error al cargar la imagen:", error);
          this.mensaje = 'Error al enviar la inspección. Inténtalo
de nuevo más tarde.';
        });
      } catch (error) {
        // Maneja los errores durante el proceso de envío del
formulario
        console.error("Error al enviar la inspección:", error);
        this.mensaje = 'Error al enviar la inspección. Inténtalo de
nuevo más tarde.';
      }
    } else {
      this.mensaje = 'Por favor, completa todos los campos
obligatorios';
    }
  }

  onFileSelected(event: any) {
    // Almacena el archivo seleccionado en la variable imageFile
    this.imageFile = event.target.files[0];
  }
}
```

4.3 Visualización de Informes

El requisito de este punto es listar todos los informes de inspección en una tabla con paginación, aunque por falta de tiempo se listo en forma de cascada. Además de permitir filtrar informes por tipo de inspección y fecha, donde por límites de tiempo sólo se hizo por tiempo de inspección. Así mismo se detalla cada informe con visualización de las imágenes y resultados de la inspección y las restantes características de un informe. Por último, se puede editar un informe y eliminarlo.

Este componente de Angular, InspectionListComponent, está diseñado para mostrar una lista de inspecciones, filtrar la lista, manejar la edición y eliminación de inspecciones, y gestionar la carga de imágenes en Firebase Storage. A continuación, se describe cada parte del código y su funcionalidad:

En las importaciones tenemos las siguientes:

Component, OnInit: Importados de @angular/core para definir el componente y su ciclo de vida.

FormBuilder, FormGroup, Validators: Importados de @angular/forms para manejar formularios reactivos.

Inspection: Una interfaz personalizada para definir el tipo de inspección.

InspectionsService: Un servicio personalizado para interactuar con los datos de las inspecciones.

AngularFireStorage: Importado de @angular/fire/compat/storage para manejar el almacenamiento de archivos en Firebase.

finalize: Importado de rxjs/operators para completar las tareas de carga de archivos.

Por otro lado, el decorador @Component fefine el componente con su selector, plantilla HTML y hoja de estilos CSS correspondientes.

La clase InspectionListComponent tiene las siguientes propiedades:

inspections: Un array para almacenar todas las inspecciones.

filteredInspections: Un array para almacenar las inspecciones filtradas.

formulario: Un FormGroup para gestionar el formulario de edición.

editedInspection: Una inspección seleccionada para editar.

filter: Una cadena para almacenar el filtro seleccionado.

El constructor inyecta InspectionsService, FormBuilder y AngularFireStorage y contiene los siguientes métodos.

El método ngOnInit inicializa el formulario llamando a createForm. Seguidamente, suscribe a las inspecciones del servicio y aplica el filtro.

El método createForm, crea un formulario reactivo con campos obligatorios y un campo opcional para la imagen.

El método onClickDelete, llama al servicio para eliminar una inspección y registra la respuesta.

El método onClickUpdate, establece la inspección seleccionada para editar. Posteriormente, rellena el formulario con los valores de la inspección seleccionada.

El método onSubmitUpdate, verifica si hay una inspección seleccionada para editar. Después, llama al servicio para actualizar la inspección con los datos del formulario. Finalmente, maneja los errores durante el proceso de actualización.

El método updateFilter, actualiza el filtro seleccionado y aplica el filtro.

El método applyFilter, filtra las inspecciones según el filtro seleccionado.

Por último, el método onFileSelected, maneja la selección de archivos para cargar imágenes en Firebase Storage. Seguidamente, sube el archivo seleccionado y actualiza el campo de imagen en el formulario con la URL de descarga de la imagen. A continuación, se deja el código del TypeScript.

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import Inspection from '../../interfaces/inspection.interface';
import { InspectionsService } from
'../../services/inspections.service';
import { AngularFireStorage } from '@angular/fire/compat/storage';
import { finalize } from 'rxjs/operators';
```

```
@Component({
  selector: 'app-inspection-list',
  templateUrl: './inspection-list.component.html',
  styleUrls: ['./inspection-list.component.css']
})
export class InspectionListComponent implements OnInit {

  inspections: Inspection[] = [];
  filteredInspections: Inspection[] = [];
  formulario!: FormGroup;
  editedInspection: Inspection | null = null;
  filter: string = 'Todos';

  constructor(
    private inspectionsService: InspectionsService,
    private formBuilder: FormBuilder,
    private storage: AngularFireStorage
  ) {}

  ngOnInit(): void {
    this.createForm();
    this.inspectionsService.getInspections().subscribe(inspections
=> {
      this.inspections = inspections;
      this.applyFilter();
    });
  }

  createForm() {
    this.formulario = this.formBuilder.group({
      Type_of_inspection: ['', Validators.required],
      date: ['', Validators.required],
      latitude: ['', Validators.required],
      longitude: ['', Validators.required],
      result: ['', Validators.required],
      image: ['']
    });
  }

  async onClickDelete(inspection: Inspection) {
    const response = await
this.inspectionsService.deleteInspection(inspection);
    console.log(response);
  }

  onClickUpdate(inspection: Inspection) {
    this.editedInspection = inspection;
    this.formulario.patchValue({
      Type_of_inspection: inspection.Type_of_inspection,
      date: inspection.date,
      latitude: inspection.latitude,
      longitude: inspection.longitude,
      result: inspection.result,
      image: inspection.image
    });
```

```
  }

  async onSubmitUpdate() {
    if (this.editedInspection) {
      try {
        const updatedData = this.formulario.value;
        const response = await
this.inspectionsService.updateInspection(this.editedInspection,
updatedData);
        console.log("Respuesta del servicio:", response);
        this.editedInspection = null;
      } catch (error) {
        console.error("Error al actualizar la inspección:", error);
      }
    } else {
      console.error("La inspección editada es nula. No se puede
actualizar la inspección.");
    }
  }

  updateFilter(event: Event) {
    const selectElement = event.target as HTMLSelectElement;
    this.filter = selectElement.value;
    this.applyFilter();
  }

  applyFilter() {
    if (this.filter === 'Todos') {
      this.filteredInspections = this.inspections;
    } else {
      this.filteredInspections = this.inspections.filter(inspection
=> inspection.Type_of_inspection === this.filter);
    }
  }

  // Método para manejar la selección de archivos
onFileSelected(event: any) {
  const file = event.target.files[0]; // Obtiene el archivo
seleccionado

  // Si el usuario ha seleccionado un archivo
  if (file) {
      const filePath = 'carpeta_principal/' + file.name; // Define
la ruta de almacenamiento de la imagen
      const fileRef = this.storage.ref(filePath); // Crea una
referencia al archivo en Firebase Storage
      const uploadTask = this.storage.upload(filePath, file); //
Sube el archivo a Firebase Storage

      // Observa el estado de la carga del archivo y maneja los
eventos
      uploadTask.snapshotChanges().pipe(
          finalize(() => {
              fileRef.getDownloadURL().subscribe(downloadURL => {
                  // Actualiza el valor del campo de imagen en tu
formulario con la URL de descarga de la imagen
```

```
                    this.formulario.patchValue({ image: downloadURL
});
                });
            })
      ).subscribe();
  }
}
}
```

Cabe destacar que para que funcionen las operaciones CRUD de las inspecciones, es necesario la interfaz de las inspecciones que se deja su código a continuación. Este se localiza en la carpeta de interfaces.

```
export default interface Inspection {
    id?: string;
    Type_of_inspection: string;
    date: String;
    latitude: number;
    longitude: number;
    result: string;
    image: string;
}
```

Por último, además de la interfaz, es necesario el servicio que realiza las operaciones CRUD de las inspecciones, que se deja su código a continuación:

```
import { Injectable } from '@angular/core';
import { AngularFirestore } from '@angular/fire/compat/firestore';
import { Observable } from 'rxjs';
import Inspection from '../interfaces/inspection.interface';

@Injectable({
  providedIn: 'root'
})
export class InspectionsService {

  constructor(private firestore: AngularFirestore) { }

  addInspection(inspection: Inspection) {
    return this.firestore.collection('inspections').add(inspection);
  }

  getInspections(): Observable<Inspection[]> {
    return this.firestore.collection('inspections').valueChanges({
idField: 'id' }) as Observable<Inspection[]>;
  }

  deleteInspection(inspection: Inspection) {
    return
this.firestore.doc(`inspections/${inspection.id}`).delete();
  }
```

```
  updateInspection(inspectionId: Inspection, newData:
Partial<Inspection>) {
    const { id } = inspectionId;
    const inspectionRef = this.firestore.doc(`inspections/${id}`);
    return inspectionRef.update(newData);
}
```

Este servicio de Angular, InspectionsService, está diseñado para interactuar con Firebase Firestore y realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre una colección de inspecciones. A continuación, se describe cada parte del código y su funcionalidad:

En las importaciones tenemos las siguientes:

Injectable: Importado de @angular/core, permite que la clase sea inyectada como un servicio.

AngularFirestore: Importado de @angular/fire/compat/firestore, se utiliza para interactuar con Firebase Firestore.

Observable: Importado de rxjs, representa una secuencia de datos que se puede observar.

Inspection: Una interfaz personalizada que define la estructura de una inspección.

El decorador @Injectable contiene providedIn: 'root', donde este servicio se proporciona en el nivel raíz de la aplicación, lo que significa que estará disponible en toda la aplicación sin necesidad de importarlo explícitamente en cada módulo.

El constructor inyecta AngularFirestore para interactuar con Firestore y los métodos realizados son los siguientes.

Método addInspection: Añade una nueva inspección a la colección inspections en Firestore, donde su parámetro es un objeto que cumple con la interfaz Inspection. Este devuelve un Promise que se resuelve cuando la inspección se ha añadido correctamente a Firestore.

Método getInspections: Recupera todas las inspecciones de la colección inspections desde Firestore. Este devuelve un Observable que emite un array de objetos Inspection. La opción { idField: 'id' } asegura que cada documento incluya su ID de Firestore como un campo id.

Método deleteInspection: Elimina una inspección específica de la colección inspections. El parámetro es un objeto Inspection que debe contener un campo id para identificar el documento en Firestore. Este devuelve un Promise que se resuelve cuando la inspección ha sido eliminada correctamente de Firestore.

Método updateInspection: Actualiza una inspección específica en la colección inspections. El parámetro es un objeto Inspection que contiene el campo id del documento a actualizar. Se devuelve un Promise que se resuelve cuando la inspección ha sido actualizada correctamente en Firestore.

4.4 Interfaz de usuario

Para completar este punto se ha empleado un diseño limpio, moderno y atractivo visualmente. Todo ello acompañado de un css bueno y limpio y uso de responsive con media query para que sea apropiado para dispositivos móviles. Además de ello, se incluye un navbar, para que el usuario se pueda ir moviendo por la aplicación web de forma rápida y sencilla. Esto se incluye en la carpeta main, localizada dentro de components. No obstante, no se explica ya que es mejor verlo visualmente en el vídeo.

5. ARCHIVOS ADICIONALES MODIFICADOS

Además de los archivos indicados, es obvio que también hay que modificar los archivos que te proporciona angular. Estos son los modificados.

5.1 App.routing.module

Este archivo de configuración de enrutamiento establece cómo se navega entre diferentes vistas en la aplicación, especificando qué componentes se deben cargar para cada ruta. Además, protege ciertas rutas con AuthGuard para garantizar que solo los usuarios autenticados puedan acceder a ellas. Este archivo se integra en la aplicación principal para permitir la navegación y la protección de rutas según sea necesario. Este es su código con el de AuthGuard.

```
import { ModuleWithProviders } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './components/login/login.component';
import { RegisterComponent } from
'./components/register/register.component';
import { LogoutComponent } from
'./components/logout/logout.component';
import { InspectionListComponent } from './components/inspection-
list/inspection-list.component';
import { NewInspectionComponent } from './components/new-
inspection/new-inspection.component';
import { AuthGuard } from './guards/auth.guard';

const Routes : Routes = [
    { path: "", component: LoginComponent },
    { path: "register", component: RegisterComponent },
    { path: "logout", component: LogoutComponent, canActivate:
[AuthGuard] },
    { path: "inspection-list", component: InspectionListComponent,
canActivate: [AuthGuard] },
    { path: "new-inspection", component: NewInspectionComponent,
canActivate: [AuthGuard] }
]

export const appRoutingProviders: any[] = [];
export const routing: ModuleWithProviders<any> =
RouterModule.forRoot(Routes);
```

```
// src/app/guards/auth.guard.ts
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot,
Router } from '@angular/router';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { Observable } from 'rxjs';
import { map, take, tap } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private afAuth: AngularFireAuth, private router:
Router) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean> |
Promise<boolean> | boolean {
    return this.afAuth.authState.pipe(
      take(1),
      map(user => !!user),
      tap(loggedIn => {
        if (!loggedIn) {
          this.router.navigate(['/']);
        }
      })
    );
  }
}
```

5.2 App.component.ts

Este archivo configura el componente raíz de la aplicación Angular. Este componente actúa como el contenedor principal de la aplicación, y generalmente incluye otros componentes hijos dentro de su plantilla HTML. Este es su código:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'prueba-tecnica-firebase';
}
```

5.3 App.module.ts

Este archivo define AppModule, el módulo raíz de la aplicación Angular:

Componentes: Declara los componentes principales de la aplicación.

Módulos: Importa módulos necesarios para la funcionalidad de la aplicación, incluyendo enrutamiento, manejo de formularios, peticiones HTTP, y Firebase.

Servicios: Provee servicios globales como AuthService y AuthGuard.

Firebase: Configura e inicializa Firebase para autenticación y almacenamiento.

Bootstrap: Establece AppComponent como el componente de arranque.

Este módulo actúa como el contenedor central de la aplicación, coordinando la integración de componentes, servicios y módulos necesarios para su funcionamiento.

```
import { NgModule } from '@angular/core';
import { BrowserModule} from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routing.module';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { ReactiveFormsModule } from '@angular/forms';


import { AngularFireAuthModule } from '@angular/fire/compat/auth';
import { AngularFireModule } from '@angular/fire/compat';
import { environment } from '../environments/environment';

import { AuthService } from './services/auth.service';
import { AppComponent } from './app.component';
import { MainComponent } from './components/main/main.component';
import { RegisterComponent } from
'./components/register/register.component';
import { LoginComponent } from './components/login/login.component';
import { LogoutComponent } from
'./components/logout/logout.component';
import { NewInspectionComponent } from './components/new-
inspection/new-inspection.component';
import { InspectionListComponent } from './components/inspection-
list/inspection-list.component';

import { AuthGuard } from './guards/auth.guard';
import { initializeApp, provideFirebaseApp } from
'@angular/fire/app';
import { getStorage, provideStorage } from '@angular/fire/storage';



@NgModule({
  declarations: [
    AppComponent,
    MainComponent,
    RegisterComponent,
```

```
    LoginComponent,
    LogoutComponent,
    NewInspectionComponent,
    InspectionListComponent
  ],
  imports: [
    BrowserModule,
    routing,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule,
    AngularFireModule.initializeApp(environment.firebase),
    AngularFireAuthModule
  ],
  providers: [appRoutingProviders, AuthService, AuthGuard],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5.4 Environment.ts

Este archivo environment.ts proporciona la configuración necesaria para conectar una aplicación Angular con los servicios de Firebase en un entorno de desarrollo. Incluye información crucial como la apiKey, authDomain, projectId, y otros identificadores necesarios para que Firebase funcione correctamente con la aplicación. Esto permite que la aplicación utilice funcionalidades como autenticación, almacenamiento, y mensajería en la nube.

```
// Import the functions you need from the SDKs you need
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
export const environment = {
  production: false,
  firebase: {
    apiKey: "AIzaSyAi8N31YGSEK1arD_5lqbEiwqRqJwCiUAI",
    authDomain: "prueba-tecnica-firebase.firebaseapp.com",
    projectId: "prueba-tecnica-firebase",
    storageBucket: "prueba-tecnica-firebase.appspot.com",
    messagingSenderId: "871432872140",
    appId: "1:871432872140:web:9c791d008977ba1d28a269",
    measurementId: "G-LFR2TB2633"
}

};
```

5.5 Index.html

Este archivo index.html configura la estructura básica de la página principal de una aplicación Angular. Este Incluye:

- Metadatos importantes y configuraciones de viewport para la compatibilidad móvil.
- Enlaces a recursos externos como Bootstrap y jQuery para estilos y funcionalidades adicionales.
- La etiqueta <app-root> que es donde Angular inyecta la aplicación, reemplazándola con el contenido del componente raíz (AppComponent).
- Scripts para jQuery y Bootstrap que proporcionan funcionalidades adicionales en la interfaz de usuario.

Este archivo es esencial para arrancar la aplicación Angular y proporcionar la base de la interfaz de usuario.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>PruebaTecnicaFirebase</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-
scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstra
p.min.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">

</head>
<body>
  <app-root></app-root>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.j
s"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.
bundle.min.js" integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

</body>
</html>
```

5.6 Main.ts

Este archivo configura y arranca la aplicación Angular:

- Importa las herramientas y configuraciones necesarias.
- Comprueba si la aplicación está en modo de producción y habilita las optimizaciones de producción si es necesario.
- Arranca el módulo principal de la aplicación (AppModule) usando la plataforma específica para navegadores (platformBrowserDynamic).
- Proporciona un manejador de errores para capturar y registrar cualquier error que ocurra durante el arranque.

En resumen, este archivo es crucial para iniciar la aplicación Angular y configurar el entorno de ejecución adecuado (producción o desarrollo).

```
import { platformBrowserDynamic } from '@angular/platform-browser-
dynamic';

import { AppModule } from './app/app.module';

import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';



if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

6. CONCLUSIONES

Como conclusiones, a pesar de que todo lo realizado no tenía conocimiento de ello, en poco tiempo he sido capaz de integrarme a ello de forma rápida. Me ha gustado trabajar con Firebase ya que hace muy cómo el tratamiento de datos e imágenes. No obstante, hay cosas que se pueden mejorar, pero con el poco tiempo que he tenido y a pesar de que la mayoría de tiempo he estado solucionando errores por problemas de versiones, espero que podáis observar que me he desenvuelto de muy buena forma. Por último, decir, que hay un par de cosas que no he podido realizar tal cual me lo pedias, pero que se podría realizar sin ningún problema con mayor tiempo. Espero que os sirva de ayuda esta documentación, el código de github y el vídeo realizado para entender lo que he realizado.