	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERÍA** **ESCUELA DE COMPUTACIÓN**	
CICLO: 02	**GUIA DE LABORATORIO #06**	
	Nombre de la Práctica:	**Auth Angular Firebase**
	MATERIA:	**Diseño y Programación de Software Multiplataforma**

I. OBJETIVOS

Que el estudiante:

- Diseñe aplicaciones web utilizando funciones de Angular.
- Diseñe aplicaciones con navegación.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a crear componente personalizado.
- Hacer uso de bingding en el diseño de interfaz angular.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a utilizar pipes dentro de la interfaz de usuario.
- Hacer uso de los componentes en el diseño de interfaz angular.

II. INTRODUCCION TEORICA

Proteger tus vistas con Angular y sus guards

Imagínense que tienen un sistema web en donde cualquier usuario puede registrarse y loguearse y como la mayoría de los sistemas, poseen una zona de panel de administración, a la cual, solo los usuarios Administradores pueden entrar. ¿Cómo solucionarías esto? Fácil, dependiendo de qué usuario este logueado, me fijo que rol o permiso tiene y lo habilito a ingresar al panel o no, esto sin importar el framework o lenguaje se debe cumplir. Ahora bien, imagínense el caso de que tengan que bloquear el acceso a 20 vistas, ¿No se vuelve muy tedioso tener que hacer el código en cada una de ellas para chequear si tiene permisos o no? Bueno, en el caso de Angular, aquí se podrían usar los guards (además de muchas otras funciones que poseen).

¿Qué es un guard?

Los guards son métodos que nos permiten acceder a ciertas rutas, dependiendo de la condición que nosotros queramos, también se podrían utilizar para evitar cargar módulos, etc. Allí podemos hacer

validaciones, llamar al backend o cualquier cosa que necesitemos. Dentro de todo lo que son los guards, existen 5 tipos:

- CanActivate: Según una condición, que nosotros queramos, ingresamos a la ruta o página.
- CanActivateChild: Funciona de la misma manera que CanActivate, la única diferencia es que se utiliza para rutas hijas (lazy loading).
- CanDeactivate: Es lo opuesto a CanActivate, según una condición, nos permite salir de una ruta o una página. Se puede utilizar por ejemplo, cuando tenemos un formulario y queremos preguntar si desea descartar los cambios del mismo.
- CanLoad: Lo podemos utilizar para evitar al usuario cargar módulos innecesarios.
- Resolve: Podríamos utilizarlo para retrasar la representación de un componente deseado hasta que se hayan obtenido los resultados necesarios.

Nosotros vamos a ver el más común, **CanActivate**, el resto se aplican de forma similar.

¿Cómo se crea un guard?

El guard se genera como si fuese un servicio, debemos hacerlo inyectable. Estos pueden retornar true o false, una promesa o un observable.

Lo que vamos a hacer es crear un guard que nos permita bloquearle la ruta a un usuario o no, dependiendo de su rol y allí pondremos el método canActivate

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRoute, Router, ActivatedRouteSnapshot, RouterStateSnapshot } from
'@angular/router';
import {AuthService} from '../../Auth.service.ts'
@Injectable()
export class AuthGuard implements CanActivate {
 constructor(private authService: AuthService, private router: Router) { }
 public canActivate()
 {
  if (this.authService.getRol() !== 'Admin') //Obtenemos en nuestro servicio el rol y nos fijamos si es igual o
no al de 'Admin
 {
        console.log('Usted no posee permisos para acceder a esta ruta');
        this.router.navigate(['/']); //Lo enviamos a la página que queramos
        return false;
  }
   return true; //Este camino deja continuar con la vista con normalidad
 }
}
```

Lo que hace este código es, chequear si tenemos el rol de admin, en caso de no tenerlo nos redirige a la vista de error y retorna false y en caso contrario muestra la vista con normalidad.

Ahora debemos decirle en que vista iría este guard, para eso vamos al routing y alli lo incluimos en la ruta deseada.

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { PanelComponent} from './panel.component';
import { AuthGuard} from 'src/app/guards/auth.guard';
const routes: Routes = [
 {
  path: '',
  component: PanelComponent,
  canActivate: [AuthGuard] //Acá indicamos cual es el guard y que tipo es
 }
];
@NgModule({
 imports: [RouterModule.forChild(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule { }
```

Por último debemos incluirlo dentro de nuestros providers del modulo correspondiente, de la siguiente manera:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AuthGuard} from './guards/auth.guard';

@NgModule({
 declarations: [
  AppComponent
 ],
 imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule
 ],
 providers: [AuthGuard], //Agregamos a los providers el guard
 bootstrap: [AppComponent]
```

```
})
export class AppModule { }
```

Con esto estaría finalizado nuestro guard.

Si deseamos realizar un guard más genérico para, según la ruta y los permisos, permitirle entrar o no, lo que podríamos hacer, es obtener la ruta a donde quiere ir el usuario y según eso chequear que permiso tiene. Lo único que cambiaría con respecto al que hicimos es qué el método **canActivate** recibirá un route del tipo **ActivatedRouteSnapshot** y un state del tipo **RouterStateSnapshot** quedándonos asi:

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot)

Luego haremos lo que corresponda según el caso.

Conclusión

Hoy vimos cómo proteger tus vistas con Angular y sus guards y los distintos tipos que existen, dependiendo del caso verás si utilizar una o la otra, pero esto nos simplifica mucho el trabajo a la hora de realizar estas validaciones.

IV. PROCEDIMIENTO

En esta Guía de Angular, vamos a construir el sistema completo de autenticación Angular y Firebase desde cero utilizando la base de datos en la **nube NoSQL en tiempo real de Firebase**.

¿Qué crearemos?

- Inicia sesión con Google
- Iniciar sesión con nombre de usuario / contraseña
- Regístrese con nombre de usuario / contraseña
- Recuperar contraseña olvidada
- Enviar verificación por correo electrónico a un usuario recién creado
- Proteja la URL de las páginas internas de la aplicación utilizando el método canActivate de route guard
- Impedir que el usuario acceda al inicio de sesión y a la URL de registro cuando un usuario ya haya iniciado sesión
- Mantener el estado de inicio de sesión del usuario de Firebase en localStorage

Repositorio : https://github.com/AlexanderSiguenza/AuthAngularFirebase

1. Crear un nuevo proyecto Angular. ***(Se deben de activar las rutas --routing)**

```
ng new AuthAngularFirebase --routing
```

2. Instale el paquete AngularFire2 en la aplicación Angular.

```
npm install firebase @angular/fire --save
```

3. Agrega tu configuración Firebase en los archivos enviorment.ts y enviorment.prod.ts

```
export const environment = {
 production: false,
 firebase: {
  apiKey: "xxxxxxxx-xxxxxxxx",
  authDomain: "xxxxxxxxxxxxxxxxxxxxxxxx",
  databaseURL: "xxxxxxxxxxxxxxxxxxxxxxxx",
  projectId: "xxxxxxxx",
  storageBucket: "xxxxxxxx",
  messagingSenderId: "xxxxxx",
  appId: "xxxxx",
  measurementId: "xxxxxxxxxxxxxxxx"
 }
};
```

4. Instalar Bootstrap4 en su proyecto.

```
npm install bootstrap
```

Vaya al angular.json archivo y reemplace el código dado a continuación con "estilos": [] matriz.

```
"styles": [
        "node_modules/bootstrap/dist/css/bootstrap.min.css",
        "src/styles.css"
       ]
```

5. Ejecute el siguiente comando para realizar pruebas.

```
ng serve –open
```

6. Necesitamos generar los componentes para el proyecto.

```
ng g c components/dashboard
ng g c components/sign-in
ng g c components/sign-up
ng g c components/forgot-password
ng g c components/verify-email
```

7. Importe y registre módulos AngularFire2 en **app.module.ts .**

```
// Firebase services + enviorment module
import { AngularFireModule } from "@angular/fire";
import { AngularFireAuthModule } from "@angular/fire/auth";
import { AngularFirestoreModule } from '@angular/fire/firestore';
import { environment } from '../environments/environment';



@NgModule({
 imports: [
  AngularFireModule.initializeApp(environment.firebase),
  AngularFireAuthModule,
  AngularFirestoreModule,
 ]
})
```

8. Crear rutas angulares, Agregue el siguiente código en el archivo **app-routing.module.ts** .

```
import { NgModule } from '@angular/core';
// Required services for navigation
```

```
import { Routes, RouterModule } from '@angular/router';

// Import all the components for which navigation service has to be activated
import { SignInComponent } from '../../components/sign-in/sign-in.component';
import { SignUpComponent } from '../../components/sign-up/sign-up.component';
import { DashboardComponent } from '../../components/dashboard/dashboard.component';
import { ForgotPasswordComponent } from '../../components/forgot-password/forgot-password.component';
import { AuthGuard } from "../../shared/guard/auth.guard";
import { VerifyEmailComponent } from '../../components/verify-email/verify-email.component';

const routes: Routes = [
  { path: '', redirectTo: '/sign-in', pathMatch: 'full' },
  { path: 'sign-in', component: SignInComponent },
  { path: 'register-user', component: SignUpComponent },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'forgot-password', component: ForgotPasswordComponent },
  { path: 'verify-email-address', component: VerifyEmailComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

9. Habilite las rutas a la vista, agregue el siguiente código en el archivo **app.component.html .**

```
<router-outlet></router-outlet>
```

router-outlet es una etiqueta especial en Angular que sirve para mostrar los componentes hijos de un componente. Por defecto todos los componentes son hijos del componente AppComponent, por lo que si incluímos esta etiqueta dentro de la vista de AppComponent, se renderizará cada uno de los componentes del routing dependiendo de la página en la que nos encontremos.

10. Crear clase para almacenar usuario.

```
ng g cl models/user
```

```
export interface User {
  uid: string;
  email: string;
  displayName: string;
  photoURL: string;
  emailVerified: boolean;
}
```

11. Crear servicio de autenticación, Este archivo contiene la lógica central de nuestro sistema de autenticación. Cubriré el inicio de sesión social utilizando el proveedor de autenticación de Google de Firebase. El servicio de autenticación cubrirá el inicio de sesión con nombre de usuario / contraseña, registro con inicio de sesión con correo electrónico / contraseña, restablecimiento de contraseña, verificación de correo electrónico.

```
ng g s services/user
```

12. Después de eso, vaya al archivo **app.module.ts** e importe el servicio de autenticación y pase la clase **AuthService a la providers: [AuthService]matriz.** Al hacer esto, nuestro servicio de autenticación estará disponible en toda la aplicación.

```
// Auth service
import { AuthService } from "./services/auth.service";

@NgModule({
 declarations: [...],
 imports: [...],
 providers: [AuthService],
 bootstrap: [...]
})
```

13. Es hora de usar la clase AuthService, nos ayudará a crear la autenticación de inicio de sesión en Angular con Firebase. Nos centraremos en iniciar sesión con nombre de usuario y contraseña. Iniciar sesión con la autenticación de Gmail o Google Necesitamos importar **AuthServiceen sign-in / sign-in.component.ts , luego inyectar AuthService en el constructor**.

    ***Tomarlo del repositorio GitHub.**

14. Coloque el siguiente código dentro del archivo **sign-in / sign-in.component.html .**
    ***Tomarlo del repositorio GitHub.**

15. **Registro de usuario**, **sign-up / sign-up.component.ts** y **sign-up / sign-up.component.html** agregue el código. ***Tomarlo del repositorio GitHub.**

16. **Olvidé mi contraseña**, Vamos a crear la función de contraseña olvidada usando Firebase en Angular. Vaya a **missing-password.component.ts** y **missing-password.component.html** agregue el código. ***Tomarlo del repositorio GitHub.**

17. **Envía un correo electrónico de verificación**, Firebase nos permite enviar correos electrónicos de verificación fácilmente, agregar código en **verify-email / verify-email.component.ts** y **verify-email.component.html. *Tomarlo del repositorio GitHub.**

18. **Utilice protectores de ruta para proteger rutas angulares,** Routes guarda rutas seguras en Angular. Cómo proteger fácilmente las rutas del acceso no autorizado usando **canActivate()** método de protección de ruta. Dirígete a **auth.service.ts** y busca el método **isLoggedIn()**. Esta función devuelve el resultado booleano a verdadero cuando el usuario inicia sesión. Si no se encuentra el usuario, devolverá falso y no permitirá que los usuarios accedan a las páginas deseadas.

```
// Devuelve verdadero cuando el usuario está conectado y el correo electrónico está verificado
get isLoggedIn(): boolean {
  const user = JSON.parse(localStorage.getItem('user'));
  return (user !== null && user.emailVerified !== false) ? true : false;
}
```

19. Tenemos que asegurar las páginas internas, para obtener esta funcionalidad, tenemos que generar archivos de guardia de ruta. Ejecute el comando para crear guardias de ruta.

```
ng generate guard guard/auth
```

```
C:\Users\Alex-Siguenza\Desktop\VARIOS\AuthAngularFirebase>ng generate guard guard/auth
? Which interfaces would you like to implement? (Press <space> to select, <a> to toggle all, <i> to invert selection)
>(*) CanActivate
 ( ) CanActivateChild
 ( ) CanDeactivate
 ( ) CanLoad
```

20. Vaya al archivo **auth.guard.ts** y coloque el código. ***Tomarlo del repositorio GitHub.** Hemos asegurado con éxito las rutas de la aplicación, ahora el usuario debe estar autenticado antes de acceder a las páginas internas de la aplicación.

21. Vaya al archivo **app-routing.module.ts** e incluya la protección de rutas .

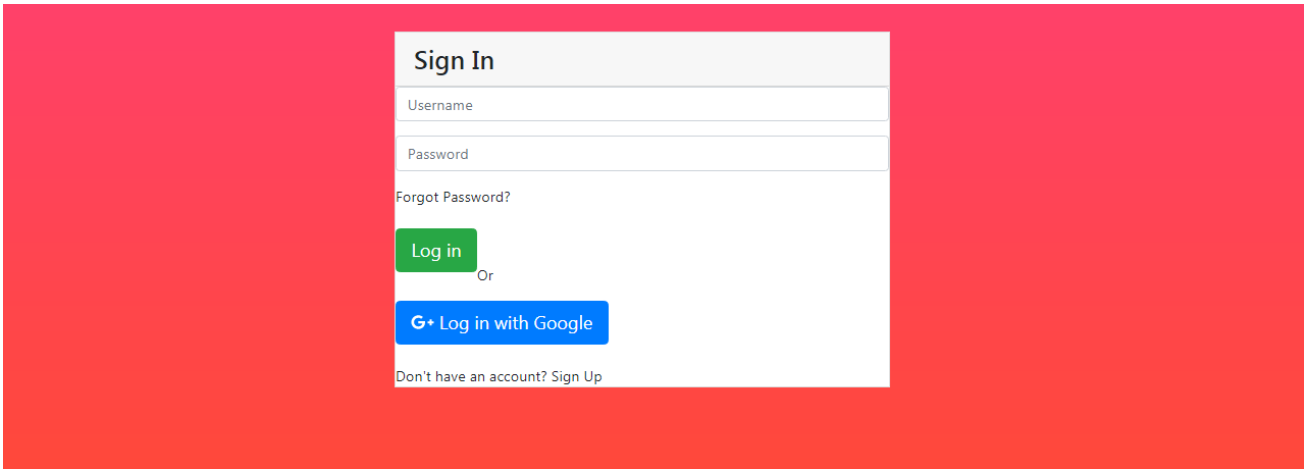
```
import { AuthGuard } from "./guard/auth.guard";
```

22. **Administra el estado de autenticación de usuario de Firebase con LocalStorage**
Este paso le explica cómo administrar el estado de autenticación de usuario de Firebase mediante la **API HTML LocalStorage.**

Administrar los datos de los usuarios registrados en el almacenamiento local es fácil con Angular y Firebase. Guardaremos el estado del usuario en Almacenamiento local cuando el usuario inicie sesión, los detalles del usuario estarán disponibles incluso si actualizamos la página. Además, elimine los datos del usuario del almacenamiento local si cerramos la sesión de la aplicación. Vaya a **auth.service.ts** y agregue el código. ***Tomarlo del repositorio GitHub.**

```
/*
 Guardar datos de usuario en almacenamiento local cuando
  iniciado sesión y configurando nulo al cerrar sesión
*/
  this.afAuth.authState.subscribe(user => {
    if (user) {
      this.userData = user;
      localStorage.setItem('user', JSON.stringify(this.userData));
      JSON.parse(localStorage.getItem('user'));
    } else {
      localStorage.setItem('user', null);
      JSON.parse(localStorage.getItem('user'));
    }
  })
 }
```

Pasamos los datos del usuario en el método **localStorage.setItem()** y obtuvimos los datos del método **localStorage.getItem('user').** Para luego Inyectar Servicio a **dashboard.component.html**, así tomar el beneficio de la misma.
Para finalizar debe de tener un resultado similar a este:

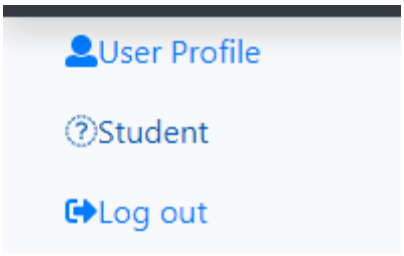


Investigación: Para que la autenticación sea exitosa deben de dar lectura a la documentación oficial

https://firebase.google.com/docs/firestore/security/get-started

V. DISCUSION DE RESULTADOS

1. Se debe de modificar el ejemplo y agregar dos métodos de ingreso más a la aplicación, **Facebook y Twitter** y además en el **dashboard**, agregar el componente alumno de la guía #05



VII. BIBLIOGRAFIA

- Flanagan, David. JavaScript La Guía Definitiva. 1ra Edición. Editorial ANAYA Multimedia. 2007. Madrid, España.
- Tom Negrito / Dori Smith. JavaScrip y AJAX para diseño web. Editorial Pearson Prentice Hall. Madrid, España. 2007.