Graphical user interface, text, application, chat or text message

Description automatically generated

firebase projects:list

Table

Description automatically generated

Instala las dependencias, lo que actualizada tu repositorio con las librerías que están en archivo package.json

npm i

Text

Description automatically generated

Seguridad entre paginas/ tabs.

Usaremos GUARDS

Importamos las librerías

import { canActivate, redirectLoggedInTo, redirectUnauthorizedTo } from '@angular/fire/auth-guard';

Creamos 2 constantes para recibir el estado del usuario y poder redirigirle.

const redirectToHomeAutorizado = () =>redirectLoggedInTo(['home']);

const redirectToLoginNoAutorizado= () =>redirectUnauthorizedTo(['']);

Seteamos el servicio Auth.

Importamos Auth

import { Auth } from '@angular/fire/auth';

Creando 3 funciones : register, login, logout

export class AuthService {

constructor(private auth: Auth) { }

async register({mail, passw}){}

async login({mail, passw}){}

logout(){}

Al ser una promesa, uso await y async, controlo que no genere error con un try

try {

const user = await createUserWithEmailAndPassword(this.auth,email, password)

return user

} catch (error) {

return null

}

En la pagina del Login seteamos el constructor y los métodos, como se puede ver usamos un formulario fb, para ello importamos Reacti…..

constructor(

private fb: FormBuilder,

private loadingController: LoadingController,

private alertController : AlertController,

private authService: AuthService,

private router:Router

) { }

Graphical user interface, text

Description automatically generated

En la pagina de Login creamos los métodos y validamos el formulario

Graphical user interface, text, application, chat or text message

Description automatically generated

[AlertController](https://ionicframework.com/docs/api/alert)

Graphical user interface, text, application, chat or text message

Description automatically generated

[LoadingController](https://ionicframework.com/docs/api/loading)

Graphical user interface, text, application

Description automatically generated

Definimos los métodos register, login showAlert en el login.page.ts .

async register({})

async login({})

async showAlert(){}

Usamos los componentes arriba vistos para mostrar la info, para ello debemos :

async register(){

const loading = await this.loadingController.create();

await loading.present();

const user = await this.authService.register(this.credentials.value);

await loading.dismiss();

if(user){this.router.navigateByUrl('/home', {replaceUrl:true});

}else{

this.showAlert('Registration failed', 'Try again');

}

}

Ahora modificamos la vista del Login

<form [formGroup]=" credentials" (ngSubmit)="login()">

<ion-item fill = "solid" class = "ion-margin-bottom">

<ion-input type ="email" placeholder ="Email" formControlName="email"></ion-input>

<ion-note slot="error" \*ngIf="(email.dirty || email.touched) && email.errors">"Email is invalid"</ion-note>

</ion-item>

<ion-button type="submit">Login</ion-button>

Toda la gestión del e-mail, errores nos lo hará automáticamente

Graphical user interface, application

Description automatically generated

Probamos las funcionalidades y vemos que funciona correctamente

Graphical user interface, text, application

Description automatically generated

Y hacemos los mismo con la pagina Home

mport { Component } from '@angular/core';

import { Router } from '@angular/router';

import { AlertController, LoadingController } from '@ionic/angular';

import { AuthService } from '../servicios/auth.service';

import { PerfilService } from '../servicios/perfil.service'

@Component({

selector: 'app-home',

templateUrl: 'home.page.html',

styleUrls: ['home.page.scss'],

})

export class HomePage {

profile = null;

constructor(

private loadingController: LoadingController,

private alertController : AlertController,

private perfilService:PerfilService,

private authService:AuthService,

private router:Router,

) {}

async logout(){

await this.authService.logout();

this.router.navigateByUrl('/', {replaceUrl:true});

}

async changeImage(){}

}

Ahora vamos a implementar el servicio de perfil del usuario.

Vamos a nuestro servicio

24.