ISEL / ADEETC

Master in Communication and Multimedia Network Engineering

Interactive Multimedia Applications

Lab Work 6

Interactive Multimedia Applications

Ionic with Firebase Services App

Rui Jesus

Introduction

This work aims to implement a set of firebase cloud services in an Ionic app. The tutorial begins with configuration of the firebase services. Next, a simple mobile application, ToDo App, is developed with emphasis on the most used Ionic techniques for accessing to cloud services. Namely, the Firebase realtime database and the Firebase authentication services.

The development of the this app covers the fundamentals of Ionic framework in terms of **Web Forms** (user interface), **Routing** and **Services** for accessing cloud information.
In this tutorial students will start a blank mobile app and built an app with three main pages (layouts). First page is the "**login**" page which is built with the **Web Forms controls** to collect and validate (automatically) the user information and submit it to the cloud service. Second page is the "**register**" page that is similar to the first one. Both uses the **Firebase authentication services**. Last page is the "**home**" page, with the goal of list from the **Firebase realtime database service** the **tasks to do** of the logged user. For the user experience the task actions are done using the **ion-item-sliding** component with is based on swipe user interaction.

The following main parts of the Ionic framework are used in the application:

(1) **Web Forms controls** to collect and validate user information;
(2) **ion-item-sliding**, a sliding item that contains an item that can be dragged to reveal buttons.
(3) **ion-item-options,** represent the option buttons for an **ion-item-sliding**;
(4) Angular **Services** to get the data in the cloud;
(5) **Routing** based on Angular **routes** to navigate among different pages;
(6) **Dependency injection system** for components communication;
(7) **Observable data** for asynchronous operations with the cloud realtime database;
(8) **AngularFireAuthModule** and **AngularFireDatabaseModule** modules to use the Firebase cloud services in Ionic.

The final application will look like the following:

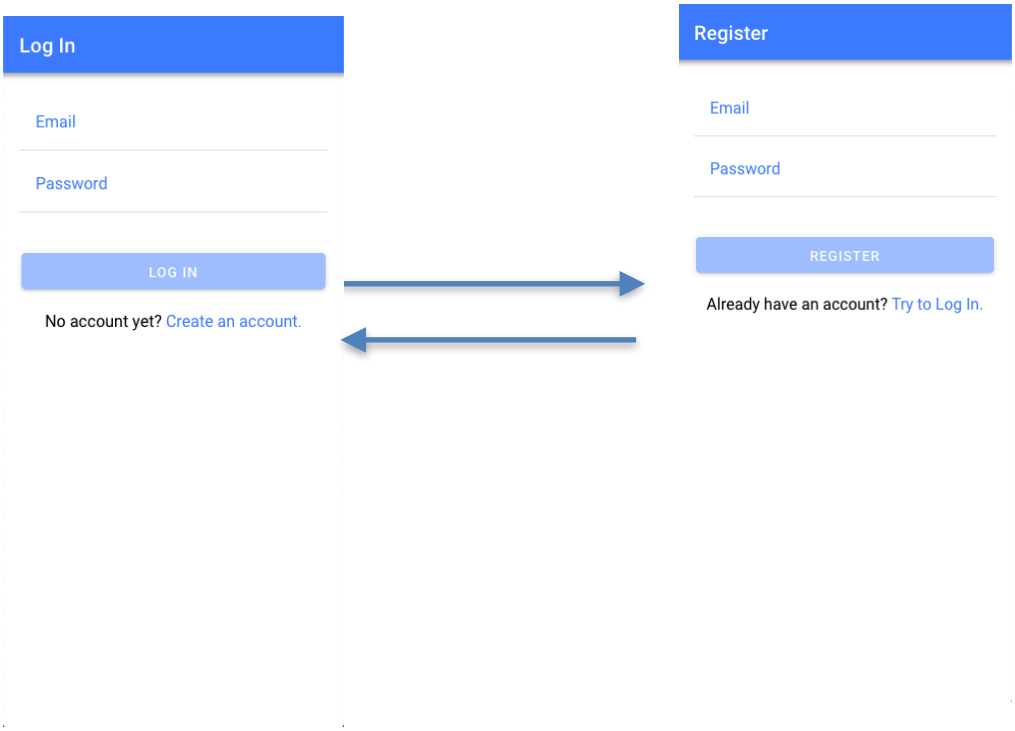

Figure 1. Login and Register Pages composed by Web Forms controls to submit validated user data.

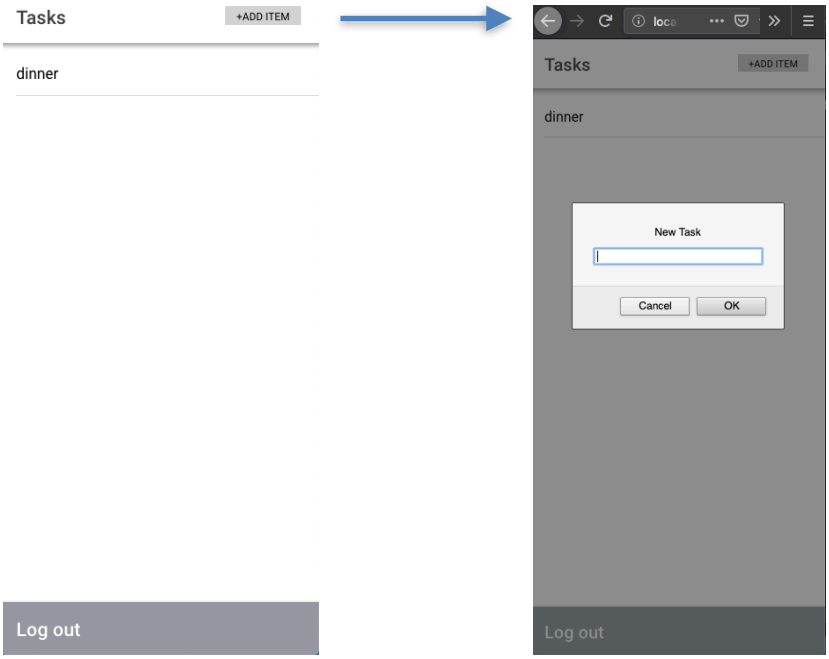

Figure 2. Tasks page with the list of the task to do.

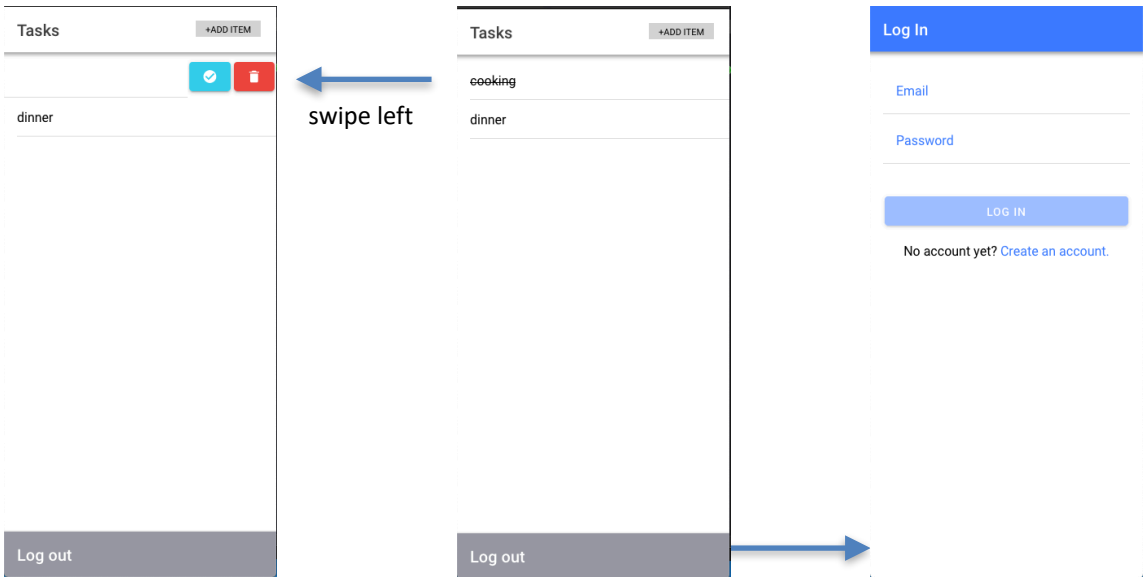

Figure 3. Sliding property of the tasks with button to manage the task list.

Note: this lab work should be done in class and the resulting code must be delivered through the Moodle platform until **June 6th**.

Laboratory Work

Blank app

1. Create an Ionic blank app to start fresh. The three most common starters are the `blank` starter, `tabs` starter, and `sidemenu` starter. Get started with the `ionic start` command:

   ionic start myApp blank

   Note: If a starter template is not defined, the **Ionic CLI** will present a list of starter templates that can be selected. Choose the `blank` options, and press enter. It also may ask if you wish to create an **Ionic.io** account.

2. One of the advantages of building hybrid applications is that much of the development and testing can be done locally in the browser. To run and try the starter app in the browser use the line command:

   ionic -o serve

Home (Tasks) Page

3. Change the title of the home page to "Tasks" and add a button inside the <ion-navbar> element after the title:

```
<ion-buttons end>
      <button (click)="addTask()">+ADD ITEM</button>
</ion-buttons>
```

Run the code, the result should be similar to the left image of Figure 2.

4. To build the list of tasks, include the following code in the home page (html):

```
<ion-content>
      <ion-list>
            <ion-item *ngFor="let task of tasks">
                  {{task.title}}
            </ion-item>
      </ion-list>
</ion-content>
```

5. Include in the file "home.page.ts", as a property of the class an array of tasks:

```
tasks: Array<any> = [];
```

6. Proceed with the initialization of the array in the constructor:

```
this.tasks = [
      {title: 'Milk', status: 'open'}
      {title: 'Eggs', status: 'open'}
      {title: 'Pancake Mix', status: 'open'}
];
```

7. To represent new tasks, create a file "tasks.ts" with:

```
export class Task {
      $key: any;
```

```
      title: string;
      status: string;
}
```

The $key element is the Firebase key. At this stage it can be empty.

8. Add the following method to the class HomePage:

```
addItem() {
      let theNewTask: string = prompt("New Task");
      if (theNewTask !== '') {
            this.tasks.push({title: theNewTask, status: 'open'});
  }
```

Run the code in the browser.

9. At this stage, new tasks can be added but it is missing a away of delete and change the status. This can be done with buttons that are presented by right to left sliding of the list element (html):

```
<ion-list>
  <ion-item-sliding #slidingItem *ngFor="let task of tasks">
      <ion-item >
          {{task.title}}
      </ion-item>
      <ion-item-options side="end">
            <ion-button (click)="markAsDone(slidingItem, task)"
            color="secondary">
                  <ion-icon name="checkmark"></ion-icon>
            </ion-button>
            <ion-button (click)="removeTask(slidingItem, task)"
            color="danger">
                  <ion-icon name="trash"></ion-icon>
            </ion-button>
      </ion-item-options>
  </ion-item-sliding>
</ion-list>
```

Run the code in the browser.

10. In the class “HomePage” create the two methods:

```
markAsDone(slidingItem, task) {
      task.status = “done”;
      slidingItem.close();
}
removeTask(slidingItem, task) {
      task.status = “removed”;

      // Include code to remove the task element of the array
      tasks
      slidingItem.close();
}
```

Run the code in the browser and try the sliding experience.

11. When the user swipe from left to right (completely), this usually means he wants to remove the element. Change the code:

```
      <ion-item-options
side=“end" (ionSwipe)=“removeTask(slidingItem,task)>
      <ion-button (click)="markAsDone(slidingItem, task)"
      color="secondary">
            <ion-icon name="checkmark"></ion-icon>
      </ion-button>
      <ion-button (click)="removeTask(slidingItem, task)"
      color="danger">
            <ion-icon name="trash"></ion-icon>
      </ion-button>
</ion-item-options>
```

12. When the user mark the task as “done” the task name should change in the list:

```
 <ion-item [ngClass]=“{taskDone: task.status == ‘done’}”>
    {{task.title}}
 </ion-item>
```

Also include in the style file the following:

```
.taskDone {
      text-decoration-line: line-through;
}
```

Run the code. Results should be similar to Figure 2 (middle image).

13. When the user click on the sliding button to mark the task as done, the icon of this button should change. Create the code the change the icon. Use the icon with the name " checkmark-circle-outline" when the task is done and the icon "checkmark-circle" when the task is not done.

Firebase Configuration

14. Go to the Firebase website, if you are not logged, login with a gmail account. Click on "Get started" (see Figure 4).

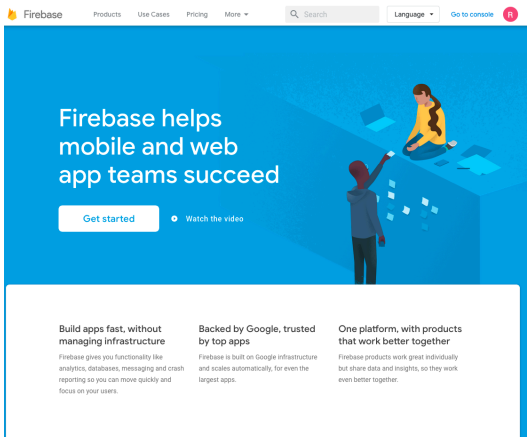

Figure 4. Firebase main layout.

15. In the next screen create a project. Choose a name and the location of the database.

16. Figure 5 shows the following screen. Add your app the Firebase Project. Choose add Web App.

17. To connect this Firebase project with your app, go to the project settings , at the bottom choose "Config" to obtain the configuration object which is similar to the following (for security reason I omit my apiKey):

```
const firebaseConfig = {
    apiKey: "XXXX",
    authDomain: "ionic-database-1.firebaseapp.com",
```

```
    databaseURL: "https://ionic-database-1.firebaseio.com",
     projectId: "ionic-database-1",
    storageBucket: "ionic-database-1.appspot.com",
    messagingSenderId: "79469847549",
    appId: "1:79469847549:web:800f65d0da1ce191"
};
```

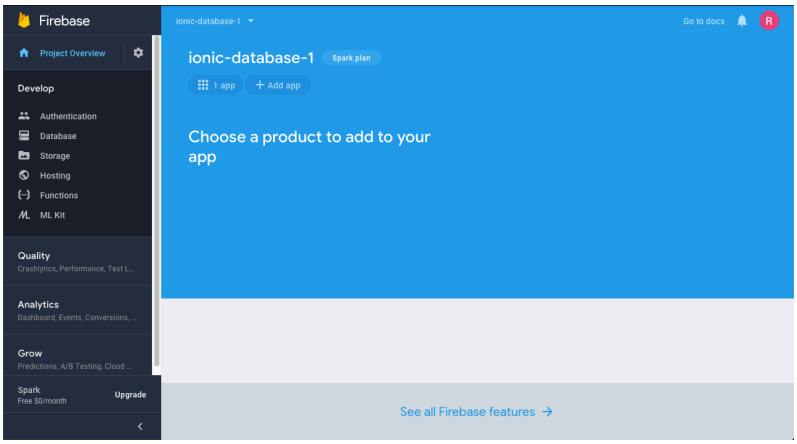

Figure 5 Firebase Project layout.

18. In the screen represented in Figure 5, choose Authentication (left side), select "**Sign-in-method"** tab and enabled the authentication by email/password.

19. To communicate our ionic 4 application to the Firebase project we'll use the **angularfire2** plugin. Run the following code in the terminal/console window:

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

20. Open the file "app.module.ts" and after the "imports" put the configuration information of the Firebase:

```
export const firebaseConfig = {
    apiKey: "XXXX",
    authDomain: "ionic-database-1.firebaseapp.com",
    databaseURL: "https://ionic-database-1.firebaseio.com",
     projectId: "ionic-database-1",
    storageBucket: "ionic-database-1.appspot.com",
    messagingSenderId: "79469847549",
    appId: "1:79469847549:web:800f65d0da1ce191"
```

```
};
```

21. Below inside the @NgModule, in the imports entry include:

```
AngularFireModule.initializeApp(firebaseConfig),
AngularFirestoreModule,
AngularFireDatabaseModule,
AngularFireAuthModule,
AngularFireStorageModule,
```

The module for the Firebase authentication services are also included. Do not forget the imports.

22. In the same file, inside the @NgModule, in the end of the "providers" entry include the following:

```
{ provide: FirestoreSettingsToken, useValue: {} }
```

App/Firebase Communication

23. To provide the data from the database in the Cloud (Firebase service), create the service "fireservice" using the Ionic CLI in the terminal/console window.

24. Update the "fileservice.ts" to the following:

```
import { AngularFirestore } from '@angular/fire/firestore'
import { Task } from '../home/task';
import * as firebase from 'firebase/app';


@Injectable({
      providedIn: 'root'
})
export class FireserviceService {
      private snapshotChangesSubscription: any;

       constructor(
            public af:AngularFirestore,
       )
       {}
```

```
    getTasks () {
          return this.af.collection('tasks').snapshotChanges();
    }

    createTask(t:Task) {
          return this.af.collection('tasks').add(t);
    }

    updateTask(TaskID:any,t:Task){
          this.af.collection('tasks').doc(TaskID).set(t);
    }

    deleteTask(TaskID:any) {
          this.af.collection('tasks').doc(TaskID).delete();
    }

    unsubscribeOnLogOut(){
          this.snapshotChangesSubscription.unsubscribe();
    }
}
```

NOTA: Information about the AngularFire library can be found in the link: https://github.com/angular/angularfire

25. To start subscribing the data from the Firebase, add the following the "HomePage" class:

```
    ngOnInit() {
          this.fser.getTasks().subscribe(data => {
          this.tasks = data.map(e => {
                return {
                      $key: e.payload.doc.id,
                      title: e.payload.doc.data()['title'],
                      status: e.payload.doc.data()['status'],
                };
                });
                console.log(this.tasks);
          });
    }
```

Do not forget to inject the "fireservice". **"ngOnInit()"** is a handler function that runs when this component is lunched. It is one of several handler functions that capture one state of the lifecycle of a component.

26. Change the methods of the "HomePage" class to communicate with the Firebase:

```
    addTask() {
  let ntask:string = prompt("New Task");
  if (ntask !== "") {
    let t:Task = {$key:'', title:ntask, status:'open'};
    console.log(t);
    this.fser.createTask(t).then(resp => {
      console.log("createTask: then - " +resp);
    })
    .catch(error => {
      console.log("createTask: catch - " +error);
    });
    console.log("addTask: " +this.tasks);
  }

}

markAsDone(slidingItem: IonItemSliding, task:any) {
 task.status = (task.status === "done")? "open":"done";
 console.log ("markAsDone " + task);
 this.fser.updateTask (task.$key, task);
  slidingItem.close();
}

removeTask(slidingItem: IonItemSliding, task:any) {
  task.status = "removed";
  this.fser.deleteTask(task.$key);
  slidingItem.close();
}
```

Run the code in the browser.

Authentication

27. Create another service, “fireauthservice.ts”.

28. Update the file to the following:

```
import * as firebase from 'firebase/app';
import { FireserviceService } from '../services/fireservice.service';
import { AngularFireAuth } from '@angular/fire/auth';

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

  constructor(
    private firebaseService: FireserviceService,
    public afAuth: AngularFireAuth
  ){}

  doRegister(value){
   return new Promise<any>((resolve, reject) => {
     firebase.auth().createUserWithEmailAndPassword(value.email,
value.password)
     .then(
       res => resolve(res),
       err => reject(err))
   })
  }

  doLogin(value){
   return new Promise<any>((resolve, reject) => {
     firebase.auth().signInWithEmailAndPassword(value.email,
value.password)
     .then(
       res => resolve(res),
       err => reject(err))
   })
```

```
  }

  doLogout(){
    return new Promise((resolve, reject) => {
      this.afAuth.auth.signOut()
      .then(() => {
        this.firebaseService.unsubscribeOnLogOut();
        resolve();
      }).catch((error) => {
        console.log(error);
        reject();
      });
    })
  }
}
```

29. Create a page, "Login" page using the Ionic CLI.

30. Update the "login.page.html" with the following:

```
      <ion-header>
    <ion-toolbar color="primary">
        <ion-title>Log In</ion-title>
    </ion-toolbar>
</ion-header>

<ion-content padding class="form-content">
    <form
class="form" [formGroup]="validations_form" (ngSubmit)="tryLogin(validatio
ns_form.value)">
        <ion-item>
            <ion-label position="floating" color="primary">Email</ion-
label>
            <ion-input type="text" formControlName="email"></ion-input>
        </ion-item>
        <div class="validation-errors">
            <ng-container *ngFor="let validation of
validation_messages.email">
```

```
                <div class="error-message"
*ngIf="validations_form.get('email').hasError(validation.type) &&
(validations_form.get('email').dirty ||
validations_form.get('email').touched)">
                    {{ validation.message }}
                </div>
            </ng-container>
        </div>
        <ion-item>
            <ion-label position="floating" color="primary">Password</ion-
label>
            <ion-input type="password" formControlName="password"></ion-
input>
        </ion-item>
        <div class="validation-errors">
            <ng-container *ngFor="let validation of
validation_messages.password">
                <div class="error-message"
*ngIf="validations_form.get('password').hasError(validation.type) &&
(validations_form.get('password').dirty ||
validations_form.get('password').touched)">
                    {{ validation.message }}
                </div>
            </ng-container>
        </div>
        <ion-button class="submit-btn" expand="block"
type="submit" [disabled]="!validations_form.valid">Log In</ion-button>
        <label class="error-message">{{errorMessage}}</label>
    </form>

    <p class="go-to-register">
        No account yet? <a (click)="goRegisterPage()">Create an account.</
a>
    </p>
</ion-content>
```

31. Update the “login.page.ts” file with the following:

```
import { Validators, FormBuilder, FormGroup, FormControl } from '@angular/
forms';
import { FireauthService } from '../service/fireauth.service';
import { Router } from '@angular/router';

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

  validations_form: FormGroup;
  errorMessage: string = '';

  validation_messages = {
   'email': [
     { type: 'required', message: 'Email is required.' },
     { type: 'pattern', message: 'Please enter a valid email.' }
   ],
   'password': [
     { type: 'required', message: 'Password is required.' },
     { type: 'minlength', message: 'Password must be at least 5 characters
long.' }
   ]
 };

  constructor(
    private authService: FireauthService,
    private formBuilder: FormBuilder,
    private router: Router
  ) { }

  ngOnInit() {
    this.validations_form = this.formBuilder.group({
      email: new FormControl('', Validators.compose([
        Validators.required,
        Validators.pattern('^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+
$')
```

```
      ])),
      password: new FormControl('', Validators.compose([
        Validators.minLength(5),
        Validators.required
      ])),
    });
  }

  tryLogin(value){
    this.authService.doLogin(value)
    .then(res => {
      this.router.navigate(["/home"]);
    }, err => {
      this.errorMessage = err.message;
      console.log(err)
    })
  }

  goRegisterPage(){
    this.router.navigate(["/register"]);
  }
}
```

32. In the style file include:

```
.error-message {
  color: var(--ion-color-danger);
}

.go-to-register {
  text-align: center;
  margin-top: 20px;
}

.submit-btn {
  margin-top: 40px;
}

a:hover {
```

```
  cursor: pointer;
}
```

Register Page

33. Create the register page (Ionic CLI) and update the html file with the following:

```
    <ion-header>
  <ion-toolbar color="primary">
      <ion-title>Register</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding class="form-content">
                                                                    <form
class="form" [formGroup]="validations_form" (ngSubmit)="tryRegister(valida
tions_form.value)">

      <ion-item>
              <ion-label position="floating" color="primary">Email</ion-
label>
          <ion-input type="text" formControlName="email"></ion-input>
      </ion-item>
      <div class="validation-errors">
                          <ng-container *ngFor="let validation of
validation_messages.email">
                                  <div class="error-message"
*ngIf="validations_form.get('email').hasError(validation.type) &&
(validations_form.get('email').dirty                    ||
validations_form.get('email').touched)">
                  {{ validation.message }}
              </div>
          </ng-container>
      </div>

      <ion-item>
           <ion-label position="floating" color="primary">Password</ion-
label>
            <ion-input type="password" formControlName="password"></ion-
input>
      </ion-item>
      <div class="validation-errors">
```

```
                            <ng-container *ngFor="let validation of
validation_messages.password">
                                    <div class="error-message"
*ngIf="validations_form.get('password').hasError(validation.type) &&
(validations_form.get('password').dirty                    ||
validations_form.get('password').touched)">
                  {{ validation.message }}
               </div>
            </ng-container>
        </div>

                  <ion-button class="submit-btn" expand="block"
type="submit" [disabled]="!validations_form.valid">Register</ion-button>
        <label class="error-message">{{errorMessage}}</label>
        <label class="success-message">{{successMessage}}</label>
    </form>
         <p class="go-to-login">Already have an account? <a
(click)="goLoginPage()">Try to Log In.</a></p>

</ion-content>
```

34. Update the “register.page.ts” file with:

```
      import { Component, OnInit } from '@angular/core';
import { Validators, FormBuilder, FormGroup, FormControl } from '@angular/
forms';
import { FireauthService } from '../service/fireauth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-register',
  templateUrl: './register.page.html',
  styleUrls: ['./register.page.scss'],
})
export class RegisterPage implements OnInit {

  validations_form: FormGroup;
  errorMessage: string = '';
  successMessage: string = '';

  validation_messages = {
```

```
    'email': [
      { type: 'required', message: 'Email is required.' },
      { type: 'pattern', message: 'Enter a valid email.' }
    ],
    'password': [
      { type: 'required', message: 'Password is required.' },
      { type: 'minlength', message: 'Password must be at least 5 characters
long.' }
    ]
  };

  constructor(
    private authService: FireauthService,
    private formBuilder: FormBuilder,
    private router: Router
  ) { }

  ngOnInit() {
    this.validations_form = this.formBuilder.group({
      email: new FormControl('', Validators.compose([
        Validators.required,
         Validators.pattern('^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+
$')
      ])),
      password: new FormControl('', Validators.compose([
        Validators.minLength(5),
        Validators.required
      ])),
    });
  }

  tryRegister(value){
    this.authService.doRegister(value)
     .then(res => {
       console.log(res);
       this.errorMessage = "";
         this.successMessage = "Your account has been created. Please log
in.";
     }, err => {
       console.log(err);
       this.errorMessage = err.message;
       this.successMessage = "";
```

```
    })
  }

  goLoginPage(){
    this.router.navigate(["/login"]);
  }

}
```

Do not forget to include the following to the style file:

```
.error-message {
    color: var(--ion-color-danger);
}

.success-message {
    color: var(--ion-color-success);
}

.go-to-login {
    text-align: center;
    margin-top: 20px;
}

.submit-btn {
    margin-top: 40px;
}

a:hover {
    cursor: pointer;
}
```

35. To run the app, update the following in the "app.component.ts" file (at the bottom):

```
   initializeApp() {
    this.platform.ready().then(() => {
      this.afAuth.user.subscribe(user => {
        if(user){
          this.router.navigate(["/home"]);
        } else {
```

```
          this.router.navigate(["/login"]);
        }
      }, err => {
        this.router.navigate(["/login"]);
      }, () => {
        this.splashScreen.hide();
      })
      this.statusBar.styleDefault();
    });
  }
```

This is to initialise the authentication service and to start the app with the login page. Run the code. Do not forget to include the import:

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

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

36. In the files "register.module.ts" and "login.module.ts" add the ReactiveFormsModule to the imports in the @NgModule.

37. Change the routes table in the "app-routing.module" file to start with the login:

const routes: Routes = [

{ path: '', redirectTo: 'login', pathMatch: 'full' },

{ path: 'home', loadChildren: './home/home.module#HomePageModule' },

{ path: 'login', loadChildren: './login/login.module#LoginPageModule' },

{ path: 'register', loadChildren: './register/register.module#RegisterPageModule' },

];

38. Update the fireservice with the information of the logged user:

```
import { Injectable } from '@angular/core';
import { AngularFirestore } from '@angular/fire/firestore'
import { Task } from '../home/task';
import * as firebase from 'firebase/app';


@Injectable({
```

```
  providedIn: 'root'
})
export class FireserviceService {
  private snapshotChangesSubscription: any;

  constructor(
    public af:AngularFirestore,
  )
  {}

  getTasks () {
    let currentUser = firebase.auth().currentUser;
    return
this.af.collection('people').doc(currentUser.uid).collection('tasks').snap
shotChanges();
  }

  createTask(t:Task) {
    let currentUser = firebase.auth().currentUser;
    return
this.af.collection('people').doc(currentUser.uid).collection('tasks').add(
t);
  }

  updateTask(TaskID:any,t:Task){
    let currentUser = firebase.auth().currentUser;

this.af.collection('people').doc(currentUser.uid).collection('tasks').doc(
TaskID).set(t);
    //this.af.doc('tasks/' + TaskID).update(t);
  }

  deleteTask(TaskID:any) {
    let currentUser = firebase.auth().currentUser;

this.af.collection('people').doc(currentUser.uid).collection('tasks').doc(
TaskID).delete();
    //this.af.doc('tasks/' + TaskID).delete();
  }
```

```
  unsubscribeOnLogOut(){
    //remember to unsubscribe from the snapshotChanges
    this.snapshotChangesSubscription.unsubscribe();
  }
}
```

Run the code in the browser.

39. Update the home page (html) with the "logout" button at the bottom:

```
    <ion-footer>
    <ion-toolbar color="medium">
        <ion-title (click)="logout()">Log out</ion-title>
    </ion-toolbar>
</ion-footer>
```

40. Add the "logout" method to the HomePage class:

```
    logout(){
    this.authService.doLogout()
    .then(res => {
      this.router.navigate(["/login"]);
    }, err => {
      console.log(err);
    })
  }
```

Run the code in the browser.