

SEG3102 – Lab 5

Angular

Persistence and Authentication with Google Firebase

The goal of this lab is to examine persistence and authentication in an Angular application using a cloud-based service. We will be extending the Lab 3 *Address Book* application, by having address entries saved to and read from a database, and by having private address book accounts.

The source code for the lab is available in the Github repository

<https://github.com/stephanesome/angFirebase.git>

Persisting of Address Entries

The Address Book application only stores address entries in memory. This is not very useful. We will extend the application with the capability to make address entries persistent. We could save the data locally on the client, but this would not support multiple users or allow the use of alternative clients such as mobile clients.

Google Firestore

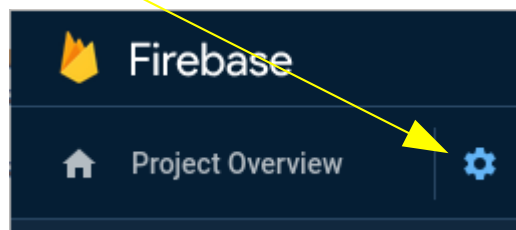
We will use the *Google Firebase Firestore* database (<https://firebase.google.com/products/firestore>). A real-time NoSQL database that can keep applications synchronized with the data store. Data is stored in the form of **Collections** made up of **Documents**.

The AngularFire module (<https://github.com/angular/angularfire>) must be added to the project to be able to interact with Firestore. In the project root folder, execute command `ng add @angular/fire` to add AngularFire to the project.

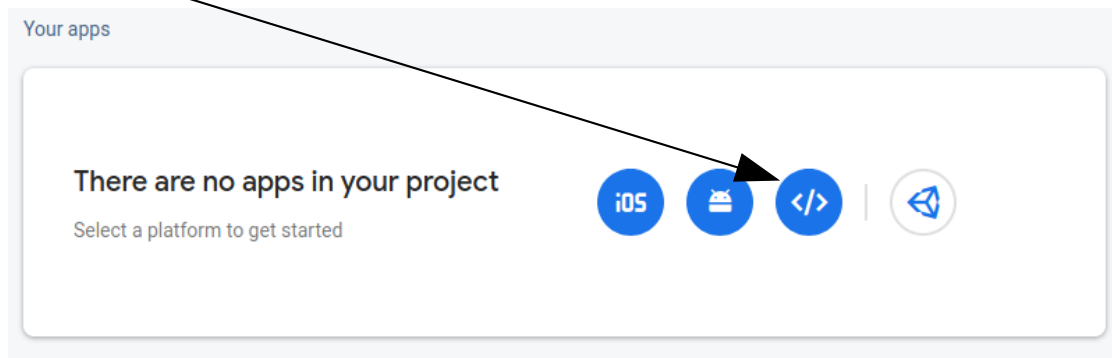
Firebase Setup

The instructions for setting up a Firestore Cloud database are described at <https://firebase.google.com/docs/firestore/quickstart>. We will summarize here.

- Connect to Firebase Console <https://console.firebase.google.com/u/0/> and Add a project (you need a Google account).
- Click on **Firestore Database** and select **Create database**. Select to **Start in test mode**.
- Select a location and **Enable**.
- Select the project **Settings**



- Add a Web App to the project.



- Open the file `src/environments/environment.ts` and copy the `firebase` object from the Firestore Web App setting to the your application environment. The edited value should be similar to the follow with the values corresponding to your settings.

```

1. export const environment = {
2.   production: false,
3.   apiKey: '<your-key>',
4.   authDomain: '<your-project-authdomain>',
5.   databaseURL: '<your-database-URL>',
6.   projectId: '<your-project-id>',
7.   storageBucket: '<your-storage-bucket>',
8.   messagingSenderId: '<your-messaging-sender-id>',
9.   appId: '<your-app-id>',
10.  measurementId: '<your-measurement-id>'
11. }
12. };

```

- Import `AngularFireModule` and `AngularFirestoreModule` to the App module (`src/app/app.module.ts`). The added elements are shown here.

```

1. ...
2. import {AngularFirestoreModule} from '@angular/fire/firestore';
3. import {AngularFireModule} from '@angular/fire';
4. import {environment} from '../environments/environment';
5. ...
6.
7. @NgModule({
8.   ...
9.   imports: [
10.    ...
11.    AngularFireModule.initializeApp(environment.firebase),
12.    AngularFirestoreModule
13.  ],
14.   ...
15. })

```

```
16. export class AppModule { }
```

The call of the `initializeApp()` method of `AngularFireModule` is required to pass the configuration object added earlier to the `src/environments/environment.ts` file.

Address Entry

We will add a *id* field to the model class for `AddressEntry`. The field will serve as a unique identifier to the stored address entries.

Edit class `Address` (`src/app/address-list/address-entry.ts`) as follow

```
1. export class AddressEntry {
2.   public id?: string | null;
3.   public firstName: string;
4.   public lastName: string;
5.   public phone?: string;
6.   public email?: string;
7.   public notes?: string;
8.
9.
10.  constructor(firstName: string, lastName: string, phone?: string, email?: string, notes?: string) {
11.    this.id = null;
12.    this.firstName = firstName;
13.    this.lastName = lastName;
14.    this.phone = phone;
15.    this.email = email;
16.    this.notes = notes;
17.  }
18. }
```

We added a field *id* to the class (line 2) and initialized it to `null` in the constructor (line 11).

Save Option

We add a button to the Address View that a User can use to save an address entry.

Address View HTML Template

Edit the Address View Component HTML Template (`src/app/address-view/address-view.component.html`)

```
1. <form class="ui large form segment" #addressForm="ngForm">
2.   <h3 class="ui header">Address Details</h3>
3.   <fieldset [disabled]="!edit ? 'disabled' : null">
4.     <div class="form-group">
5.       <label for="firstName">First Name</label>
6.       <input class="form-control" name="firstName" id="firstName" [(ngModel)]="address.firstName">
7.     </div>
8.     <div class="form-group">
9.       <label for="lastName">Last Name</label>
10.      <input class="form-control" name="lastName" id="lastName" [(ngModel)]="address.lastName">
```

```

11. </div>
12. <div class="form-group">
13.   <label for="phone">Phone Number</label>
14.   <input class="form-control" name="phone" id="phone" [(ngModel)]="address.phone">
15. </div>
16. <div class="form-group">
17.   <label for="email">Email</label>
18.   <input class="form-control" name="email" id="email" [(ngModel)]="address.email">
19. </div>
20. <div class="form-group">
21.   <label for="notes">Notes</label>
22.   <textarea class="form-control" rows="4" cols="60" name="notes" id="notes"
23.     [(ngModel)]="address.notes"></textarea>
24. </div>
25. </fieldset>
26. <div class="btn-toolbar" role="toolbar">
27.   <button class="btn btn-danger" name="delete" (click)="delete()">Delete</button>
28.   <button (click)="toggleEdit()"
29.     class="btn btn-secondary m-auto" *ngIf="edit">
30.     Protect
31.   </button>
32.   <button (click)="toggleEdit()"
33.     class="btn btn-secondary m-auto" *ngIf="!edit">
34.     Edit
35.   </button>
36.   <button class="btn btn-dark"
37.     [hidden]="addressForm.pristine"
38.     name="save" (click)="save()">Save</button>
39. </div>
40. </form>

```

We added a **Save** button (lines 36-38). Property `hidden` is set so that the button will not be shown unless the form has been changed. A reference to the `ngForm` (added in line 1) allows access to the status of the form.

Address View Class

The click event handler of button **Save** is implemented in the component Class as follow.

```

1. import {Component, EventEmitter, Input, OnInit, Output} from '@angular/core';
2. import {AddressEntry} from '../address-entry';
3.
4. @Component({
5.   selector: 'app-address-view',
6.   templateUrl: '../address-view.component.html',
7.   styleUrls: ['../address-view.component.css']
8. })
9. export class AddressViewComponent implements OnInit {
10.   @Input() address: AddressEntry;
11.   @Output() fireDelete: EventEmitter<AddressEntry> = new EventEmitter();
12.   @Output() fireSave: EventEmitter<AddressEntry> = new EventEmitter();

```

```

13. public edit: boolean;
14.
15. constructor() { }
16.
17. ngOnInit(): void {
18.   this.edit = true;
19. }
20.
21. toggleEdit(): void {
22.   this.edit = !this.edit;
23. }
24.
25. delete(): void {
26.   this.fireDelete.emit(this.address);
27. }
28.
29. save(): void {
30.   this.fireSave.emit(this.address);
31. }
32. }

```

The function `save` (lines 29 to 31) issues an output event (line 12) notifying the parent component of the deletion request.

Firestore Service

We create a service class to encapsulate the interaction with the Firestore database. The service is intended to be injected into other parts of the application that need such access.

Generate a service `ng generate service address-list/firestore/address-db`. Edit `src/app/address-list/firestore/address-db.service.ts` as follow.

```

1. import { Injectable } from '@angular/core';
2. import { AngularFireStore, DocumentChangeAction, DocumentReference } from
   "@angular/fire/compat/firestore";
3. import { Observable } from "rxjs";
4. import { AddressEntry } from "../address-entry";
5.
6. @Injectable({
7.   providedIn: 'root'
8. })
9. export class AddressDbService {
10.
11.   constructor(private firestore: AngularFireStore) { }
12.
13.   getAddresses(): Observable<DocumentChangeAction<unknown>[]> {
14.     return this.firestore
15.       .collection('abooks')
16.       .doc('user')
17.       .collection('addresses')

```

```

18.     .snapshotChanges();
19. }
20.
21. createAddress(address: AddressEntry): Promise<DocumentReference> {
22.     delete address.id;
23.     return this.firestore
24.         .collection('abooks')
25.         .doc('user')
26.         .collection('addresses')
27.         .add({...address});
28. }
29.
30. updateAddress(address: AddressEntry): Promise<void> {
31.     const addressId = address.id;
32.     delete address.id;
33.     return this.firestore
34.         .collection('abooks')
35.         .doc('user')
36.         .collection('addresses')
37.         .doc(addressId!)
38.         .update(address);
39. }
40.
41. deleteAddress(addressId: string): Promise<void> {
42.     return this.firestore
43.         .collection('abooks')
44.         .doc('user')
45.         .collection('addresses')
46.         .doc(addressId)
47.         .delete();
48. }
49. }

```

AngularFirestoreModule provides an API to interact with the Firestore database. The API uses asynchronous communication implemented with **observables** and **promises**. Service **AngularFirestore** injected on line 11 provides functions to interact with the database including function **collection** that returns a reference to a collection of documents. Documents are retrieved or added to a collection using that reference. For instance, function **snapshotChanges** creates a stream of changes synchronized with the database. These changes include references to the affected documents.

Function **getAddresses** (lines 13-19) returns an Observable of document changes. It is used to obtain the list of documents and keep that list synchronized with the Datastore. Function **createAddress** adds a document to the collection. Note that Firestore automatically creates a collection when it does not exist. **updateAddress** changes an existing document and **deleteAddress** removes a document.

Address List Component

We are now modifying the Address List component to get address entries from the AddressDbService and to handle saving as well as deleting entries.

Address List Component HTML Template

Edit the HTML Template (`src/app/address-list/address-list.component.html`) as follow.

```
1. <div class="row">
2.   <div class="col-md-4">
3.     <button class="btn btn-success" (click)="addAddress()">New Address</button>
4.   </div>
5.   <div class="col-md-8" [ngStyle]="msgStyle" [hidden]="hideMsg">
6.     {{message}}
7.   </div>
8. </div>
9. <hr>
10. <div class="row">
11.   <div class="address-list col-md-3" *ngIf="addresses.length > 0">
12.     <app-address-list-element
13.       *ngFor="let address of addresses"
14.       [address]="address"
15.       (click)="select(address)"
16.     >
17.   </app-address-list-element>
18. </div>
19.   <div class="col-md-9">
20.     <app-address-view *ngIf="currentAddress !== null"
21.       [address]="currentAddress"
22.       (fireDelete)="deleteCurrent()"
23.       (fireSave)="saveCurrent()"></app-address-view>
24.   </div>
25. </div>
```

We added an handler for the `fireSave` event that is emitted by the Address View Component when a user clicks to save an entry (line 23).

A message is displayed (lines 5-7) to provide feedback to the user. The style is controlled by a `ngStyle` directive so that the message is hidden or displayed based on property binding to a variable (`hideMsg`) in the component class.

Address List Component Class

The Address List Component class is modified to gets address entries from the AddressDb service.

```
1. import {Component, OnInit} from '@angular/core';
2. import {AddressEntry} from './address-entry';
3. import {NotificationService} from './notification.service';
4. import {AddressDbService} from './firebase/address-db.service';
```

```

5.
6. @Component({
7.   selector: 'app-address-list',
8.   templateUrl: './address-list.component.html',
9.   styleUrls: ['./address-list.component.css'],
10.  providers: [NotificationService]
11. })
12. export class AddressListComponent implements OnInit {
13.   addresses: AddressEntry[] = [];
14.   currentAddress: AddressEntry | null = null;
15.   message: string = "";
16.   hideMsg = true;
17.   msgStyle = {
18.     color: "",
19.     'background-color': 'white',
20.     'font-size': '150%',
21.   };
22.
23.   constructor(private notificationService: NotificationService,
24.               private store: AddressDbService) { }
25.
26.   ngOnInit(): void {
27.     this.message = "";
28.     this.store.getAddresses().subscribe(data => {
29.       this.addresses = data.map(e => {
30.         return {
31.           id: e.payload.doc.id,
32.           ...(e.payload.doc.data() as object)
33.         } as AddressEntry;
34.       });
35.     });
36.   }
37.
38.   select(address: AddressEntry): void {
39.     this.currentAddress = address;
40.     this.notificationService.selectionChanged(address);
41.   }
42.
43.   showMessage(type: string, msg: string): void {
44.     this.msgStyle.color = type === 'error' ? 'red' : 'blue';
45.     this.message = msg;
46.     this.hideMsg = false;
47.     setTimeout(
48.       () => {
49.         this.hideMsg = true;
50.       }, 2500
51.     );
52.   }
53.

```



```

54. addAddress(): void {
55.   const newAddress = new AddressEntry('New', 'Entry', "", "", "");
56.   this.addresses = [newAddress, ...this.addresses];
57.   this.select(newAddress);
58. }
59.
60. deleteCurrent(): void {
61.   if (this.currentAddress && this.currentAddress.id !== null) {
62.     this.addresses =
63.       this.addresses.filter((address: AddressEntry) => address !== this.currentAddress);
64.     // **** permanently delete
65.     this.store.deleteAddress(this.currentAddress.id!)
66.       .then(_ =>
67.         this.showMessage('info', 'The address entry was successfully deleted')
68.       )
69.       .catch(_ =>
70.         this.showMessage('error', 'Error unable to delete the address entry')
71.       );
72.     this.currentAddress = null;
73.   }
74. }
75.
76. saveCurrent(): void {
77.   if (this.currentAddress && this.currentAddress.id === null) {
78.     this.store.createAddress(this.currentAddress)
79.       .then(
80.         docRef => {
81.           this.currentAddress!.id = docRef.id;
82.           this.showMessage('info', 'The address entry was successfully saved');
83.         }
84.       )
85.       .catch(_ =>
86.         this.showMessage('error', 'Error unable to save the address entry')
87.       );
88.   } else {
89.     this.store.updateAddress(this.currentAddress!)
90.       .then(_ =>
91.         this.showMessage('info', 'The address entry was successfully updated')
92.       )
93.       .catch(_ =>
94.         this.showMessage('error', 'Error unable to update the address entry')
95.       );
96.   }
97. }
98. }

```

We use the AddressDb Service injected in the constructor (line 24). The `ngOnInit` lifecycle hook (lines 26-36) gets all the documents and set up the `addresses` array with.

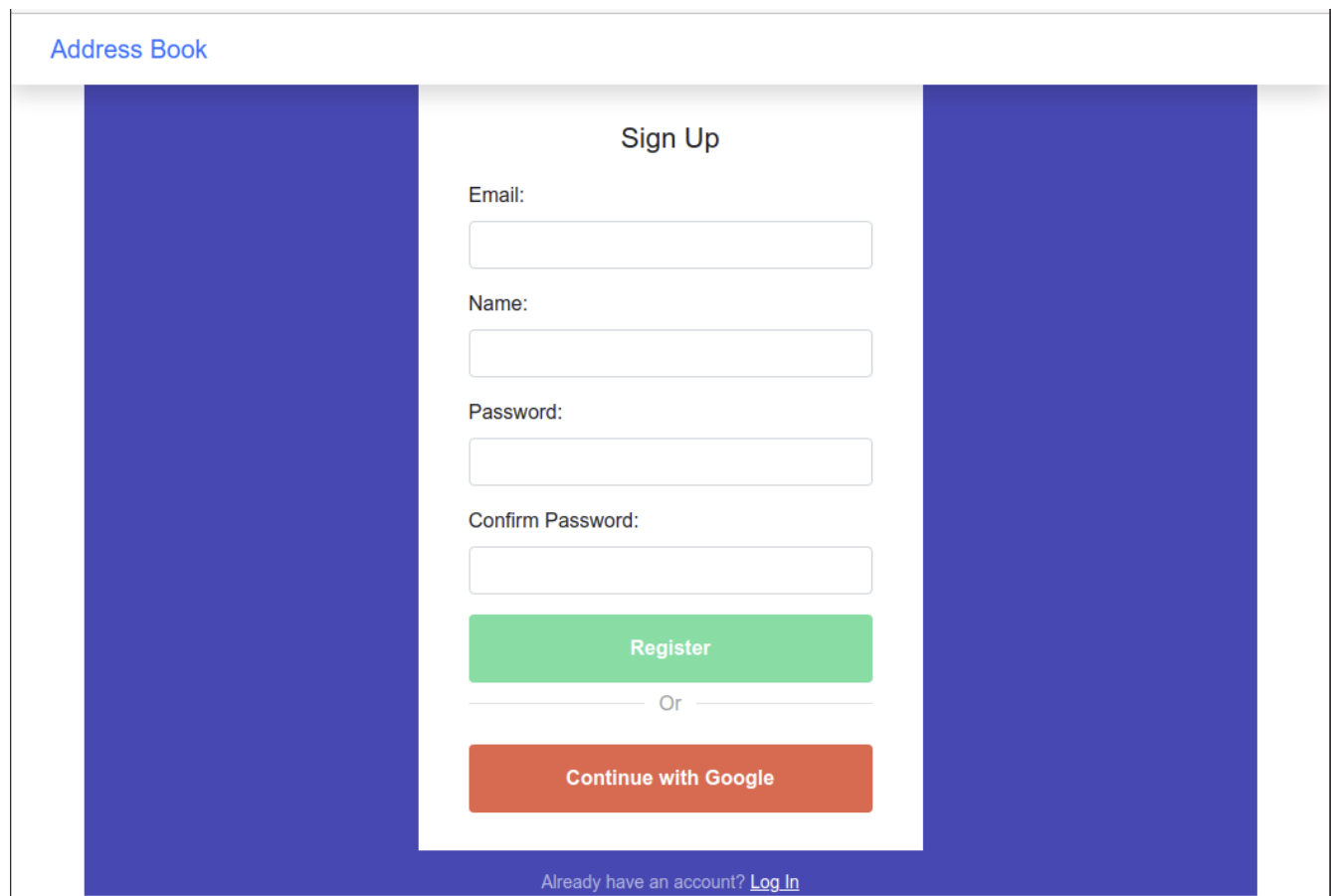
Function `deleteCurrent` (lines 60-74) has been modified to invoke the `AddressDb` service for the actual deletion. The call to `deleteAddress` (line 65) returns a `Promise`. Function then provides a handler for a notification of success while `catch` considers error situations. We set messages according to the status returned (line 67 and line 70).

A similar approach is used for function `saveCurrent` with the addition that it determines whether a new entry is being created or an update is being made by checking the value of the `id` of the address entry. When a new entry is saved, we recuperate the document `id` generated by Firestore and set it as `id` for the address entry (lines 81-82).

Authentication with Firebase

We will now update the application with Firebase Authentication (<https://firebase.google.com/docs/auth>). This will allow our application to be used by multiple users. Each user will have an account and will be able to only access their stored addresses.

We will add a Sign-Up page



The screenshot shows a web application titled "Address Book" in the top left corner. The main content area is a "Sign Up" form centered on a white background, flanked by two large blue vertical bars. The form contains the following elements:

- Sign Up** (Section Header)
- Email:** (Label) followed by a text input field.
- Name:** (Label) followed by a text input field.
- Password:** (Label) followed by a text input field.
- Confirm Password:** (Label) followed by a text input field.
- Register** (Green button)
- Or** (Text separator)
- Continue with Google** (Red button)
- Already have an account? [Log In](#)** (Link at the bottom)

A Sign-In page

Address Book

Sign In

Email

Password

Log in

Or

Log in with Google

[Forgot Password?](#)

Don't have an account? [Sign Up](#)

A Password-Forgot Page

Address Book

Reset Password

Please enter your email address to request a password reset.

Email Address

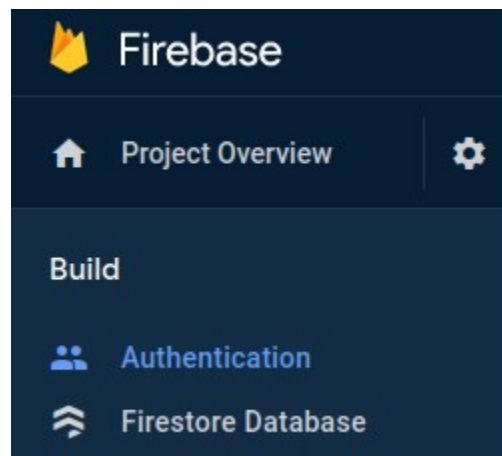
Reset Password

Go back to ? [Log In](#)

A User will be initially presented the Sign-In page and only upon successful authentication, the application will navigate to their Address Book.

Enabling Firebase Authentication

Go to Google Firebase and select **Authentication** from the project console menu then click on **Get started**.



Firebase supports authentication with several authentication providers. We will use a Firestore database stored *Email/Password* and *Google* as providers for this application.

Click to enable sign-in providers *Email/Password* and *Google*. For Google, you need to provide an email address for project support. This email address will be presented to users when they are authenticating with Google for contact.

Routing Module

Add a routing module with command `ng generate module app-routing --flat`. Edit `/src/app/app-routing.module.ts` as follow.

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { RouterModule, Routes } from '@angular/router';
4. import { SignInComponent } from './sign-in/sign-in.component';
5. import { SignUpComponent } from './sign-up/sign-up.component';
6. import { AddressListComponent } from './address-list/address-list.component';
7. import { AuthGuard } from './authentication/auth.guard';
8. import { PasswordForgotComponent } from './password-forgot/password-forgot.component';
9. import { AngularFireAuthGuard } from "@angular/fire/compat/auth-guard";
10.
11. const routes: Routes = [
12.   { path: "", redirectTo: '/sign-in', pathMatch: 'full' },
13.   { path: 'sign-in', component: SignInComponent },
14.   { path: 'sign-up', component: SignUpComponent },
15.   { path: 'address-list', component: AddressListComponent, canActivate: [AngularFireAuthGuard] },
16.   { path: 'password-forgot', component: PasswordForgotComponent }
17. ];
18.
19. @NgModule({
20.   declarations: [],
21.   imports: [
22.     CommonModule,
23.     RouterModule.forRoot(routes)
```

```
24. ]  
25. })  
26. export class AppRoutingModule { }
```

We defined routes to a Sign Up component, a Sign In component, a Password Forgot component and the main application, the Address List component. We use `AngularFireAuthGuard`, a guard provided by `AngularFire` to ensure only authenticated users can access the Address List component.

Register the routing module with the `App` module (`src/app/app.module.ts`) and import the `RouterModule`, by updating the imports array as follow.

```
1. ...  
2. import {FormsModule, ReactiveFormsModule} from "@angular/forms";  
3. import {AppRoutingModule} from "./app-routing.module";  
4. import {RouterModule} from "@angular/router";  
5. ...  
6.   imports: [  
7.     ...  
8.     AppRoutingModule,  
9.     RouterModule,  
10.    ReactiveFormsModule  
11.  ]
```

We also imported the `ReactiveFormsModule` that we will use for the sign-up form.

Generate the Sign-In, Sign-Up and Password-Forgot components: `ng generate component sign-in`, `ng generate component sign-up`, `ng generate component password-forgot`.

Update the HTML Template of the `APP` component (`src/app/app.component.html`) with the router outlet.

```
1. <app-header></app-header>  
2. <div class="container">  
3.   <router-outlet></router-outlet>  
4. </div>
```

CSS Styling

Edit the application CSS file `src/styles.css` according to the source code on GitHub.

Authentication Service

Run `ng generate service authentication/auth` to create a service for the interaction with Firebase authentication services.

Edit `src/app/authentication/auth.service.ts` as follow.

```
1. import { Injectable } from '@angular/core';  
2. import { Observable } from 'rxjs';  
3. import { AngularFireAuth } from "@angular/fire/compat/auth";
```

```

4. import firebase from "firebase/compat/app";
5.
6. @Injectable({
7.   providedIn: 'root'
8. })
9. export class AuthService {
10.   private loggedInUser: firebase.User | null = null;
11.
12.   constructor(
13.     private afAuth: AngularFireAuth
14.   ) {}
15.
16.   get user(): Observable<firebase.User | null> {
17.     return this.afAuth.user;
18.   }
19.
20.   get userid(): string {
21.     if (this.loggedInUser) return this.loggedInUser.uid;
22.     return "";
23.   }
24.
25.   signIn(email: string, password: string): Promise<void> {
26.     return this.afAuth.signInWithEmailAndPassword(email, password)
27.       .then((credential) => {
28.         this.loggedInUser = credential.user;
29.       });
30.   }
31.
32.   signUp(email: string, password: string, name: string): Promise<void> {
33.     return this.afAuth.createUserWithEmailAndPassword(email, password)
34.       .then((credential) => {
35.         const user = credential.user;
36.         this.loggedInUser = credential.user; // shouldn't be null...
37.         if (user) {
38.           user.updateProfile({displayName: name}).then(_ => {});
39.         }
40.       });
41.   }
42.
43.   passwordReset(passwordResetEmail: string): Promise<void> {
44.     return this.afAuth.sendPasswordResetEmail(passwordResetEmail);
45.   }
46.
47.   googleAuth(): Promise<void | firebase.auth.UserCredential> {
48.     return this.afAuth.signInWithPopup(firebase.auth.GoogleAuthProvider())
49.       .then((credential) => {
50.         this.loggedInUser = credential.user;
51.       });
52.   }

```

```

53.
54.   signInOut(): Promise<void> {
55.     return this.afAuth.signInOut()
56.       .then(() => {
57.         this.loggedUser = null;
58.       });
59.   }
60. }

```

Firebase provides an API for the authentication services, accessed through Angular Firebase, using an instance of `AngularFireAuth`. We inject this instance in the constructor (line 13). It is used on lines 26-29 for sign-in with email/password, on lines 33-40 for sign-up, on line 44 for password reset, on lines 48-51 for sign-in with Google and on lines 55-58 to sign-out. The Firebase API functions return Promises that are subscribed to for the authentication operation results. We set the property *loggedUser* to the signed-in user and returns the logged user id on lines 20-23.

Sign-Up Component

Edit the HTML template of the sign-in component `/src/app/sign-up/sign-up.component.html` as follow.

```

1.  <div class="displayTable">
2.    <div class="displayTableCell">
3.      <div class="authBlock">
4.        <h3>Sign Up</h3>
5.        <form [formGroup]="signupForm" (ngSubmit)="register()">
6.          <div class="form-group">
7.            <label for="email">Email:</label>
8.            <input type="text" class="form-control" id="email" formControlName="email">
9.            <div [hidden]="email.pristine || email.valid"
10.               class="alert alert-danger">
11.              Well formatted email is required.
12.            </div>
13.          </div>
14.          <div class="form-group">
15.            <label for="name">Name:</label>
16.            <input type="text" class="form-control" id="name" formControlName="name">
17.            <div [hidden]="name.pristine || name.valid"
18.               class="alert alert-danger">
19.              A name must be provided.
20.            </div>
21.          </div>
22.          <div formGroupName="pwGroup">
23.            <div class="form-group">
24.              <label for="password">Password:</label>
25.              <input type="password" class="form-control" id="password" required
26.                formControlName="password">
27.              <div [hidden]="password.pristine || password.valid"

```

```

28.         class="alert alert-danger">
29.         A Password is required.
30.     </div>
31. </div>
32. <div class="form-group">
33.     <label for="pwconfirm">Confirm Password:</label>
34.     <input type="password" class="form-control" id="pwconfirm" required
35.         formControlName="confirmPassword">
36.     <div [hidden]="confirmPassword.pristine || confirmPassword.valid"
37.         class="alert alert-danger">
38.         A Confirmation Password is required.
39.     </div>
40. </div>
41. <div [hidden]="(password.pristine || confirmPassword.pristine) || pwGroup.valid"
42.     class="alert alert-danger">
43.     The passwords do not match.
44. </div>
45. </div>
46. <button type="submit" class="btn btn-success" [disabled]="signupForm.invalid">Register</button>
47. </form>
48. <div class="formGroup">
49.     <span class="or"><span class="orInner">Or</span></span></div>
50. </div>
51. <!-- Continue with Google -->
52. <div class="formGroup">
53.     <button type="button" class="btn googleBtn" (click)="googleSignIn()">
54.         Continue with Google
55.     </button>
56. </div>
57. </div>
58. <div class="redirectToLogin">
59.     <span>Already have an account? <span class="redirect" routerLink="/sign-in">Log
    In</span></span>
60. </div>
61. </div>
62. </div>

```

We use a reactive form to obtain the user registration data and we perform the usual validations.

Edit the component class as follow.

```

1. import { Component } from '@angular/core';
2. import {AbstractControl, FormBuilder, ValidationErrors, Validators} from "@angular/forms";
3. import {AuthService} from "../authentication/auth.service";
4. import {Router} from "@angular/router";
5.
6. function passwordMatcher(pwGrp: AbstractControl): ValidationErrors | null {
7.     const passwd = pwGrp.get('password');
8.     const confpasswd = pwGrp.get('confirmPassword');
9.     return passwd!.value === confpasswd!.value ? null : {mismatch: true};

```



```

10. }
11.
12. @Component({
13.   selector: 'app-sign-up',
14.   templateUrl: './sign-up.component.html',
15.   styleUrls: ['./sign-up.component.css']
16. })
17. export class SignUpComponent {
18.   signupForm = this.builder.group({
19.     email: ['', [Validators.required, Validators.email]],
20.     name: ['', Validators.required],
21.     pwGroup: this.builder.group({
22.       password: ['', Validators.required],
23.       confirmPassword: ['', Validators.required]
24.     }, { validators: [passwordMatcher] })
25.   });
26.
27.   get email(): AbstractControl {return <AbstractControl>this.signupForm.get('email'); }
28.   get name(): AbstractControl {return <AbstractControl>this.signupForm.get('name'); }
29.   get password(): AbstractControl {return
   <AbstractControl>this.signupForm.get('pwGroup')!.get('password'); }
30.   get confirmPassword(): AbstractControl {return
   <AbstractControl>this.signupForm.get('pwGroup')!.get('confirmPassword'); }
31.   get pwGroup(): AbstractControl {return <AbstractControl>this.signupForm.get('pwGroup'); }
32.
33.   constructor(
34.     private builder: FormBuilder,
35.     private authService: AuthService,
36.     private router: Router
37.   ) {}
38.
39.   register(): void {
40.     this.authService.signUp(this.email.value, this.password.value, this.name.value)
41.       .then(() => {
42.         this.router.navigate(['address-list']).then(() => {});
43.       })
44.       .catch((error) => {
45.         window.alert(error.message);
46.       });
47.   }
48.
49.   googleSignIn(): void {
50.     this.authService.googleAuth()
51.       .then(() => {
52.         this.router.navigate(['address-list']).then(() => {});
53.       })
54.       .catch((error) => {
55.         window.alert(error.message);
56.       });

```

```
57. }  
58. }
```

The Authentication service is injected (line 35) and used in the handlers for the sign-up (lines 39-47). When a user select to continue with Google, we call a function to sign-in with Google (lines 49-56). Upon a successful sign-in, the application navigates to the Address List page.

Sign-In Component

Edit the HTML template of the sign-in component `/src/app/sign-in/sign-in.component.html` as follow.

```
1. <div class="displayTable">  
2.   <div class="displayTableCell">  
3.     <div class="authBlock pt-5">  
4.       <h3>Sign In</h3>  
5.       <div class="formGroup">  
6.         <input type="text" class="formControl" placeholder="Email" #email required>  
7.       </div>  
8.       <div class="formGroup">  
9.         <input type="password" class="formControl" placeholder="Password" #userPassword required>  
10.      </div>  
11.      <div class="formGroup">  
12.        <input type="button" class="btn btn-outline-primary" value="Log in"  
13.          (click)="signIn(email.value, userPassword.value)">  
14.      </div>  
15.      <div class="formGroup">  
16.        <span class="or"><span class="orInner">Or</span></span>  
17.      </div>  
18.      <div class="formGroup">  
19.        <button type="button" class="btn btn-outline-primary" (click)="googleSignIn()">  
20.          Log in with Google  
21.        </button>  
22.      </div>  
23.      <div class="forgotPassword">  
24.        <span routerLink="/password-forgot">Forgot Password?</span>  
25.      </div>  
26.    </div>  
27.    <div class="redirectToLogin">  
28.      <span>  
29.        Don't have an account?  
30.        <span class="redirect" routerLink="/sign-up"> Sign Up</span>  
31.      </span>  
32.    </div>  
33.  </div>  
34. </div>
```

Edit the component class as follow.

```

1. import { Component } from '@angular/core';
2. import { AuthService } from '../authentication/auth.service';
3. import { Router } from '@angular/router';
4.
5. @Component({
6.   selector: 'app-sign-in',
7.   templateUrl: './sign-in.component.html',
8.   styleUrls: ['./sign-in.component.css']
9. })
10. export class SignInComponent {
11.
12.   constructor(
13.     private authService: AuthService,
14.     private router: Router) { }
15.
16.   signIn(email: string, password: string): void {
17.     this.authService.signIn(email, password)
18.       .then(() => {
19.         this.router.navigate(['address-list']).then(() => {});
20.       })
21.       .catch((error) => {
22.         window.alert(error.message);
23.       });
24.   }
25.
26.   googleSignIn(): void {
27.     this.authService.googleAuth()
28.       .then(() => {
29.         this.router.navigate(['address-list']).then(() => {});
30.       })
31.       .catch((error) => {
32.         window.alert(error.message);
33.       });
34.   }
35. }

```

We inject the Authentication service on line 13, and used it for the handlers of sign-in (lines 17-23) and sign-in with Google (lines 27-33).

Password-Forgot Component

Edit the HTML template of the password-forgot component `/src/app/password-forgot/password-forgot.component.html` as follow.

```

1. <div class="displayTable">
2.   <div class="displayTableCell">
3.     <div class="authBlock">
4.       <h3>Reset Password</h3>
5.       <p class="text-center">Please enter your email address to request a password reset.</p>
6.       <div class="formGroup">

```

```

7.     <input type="email" class="formControl"
8.         placeholder="Email Address" #passwordResetEmail required>
9.     </div>
10.    <div class="formGroup">
11.        <input type="submit" class="btn btnPrimary" value="Reset Password"
12.            (click)="passwordReset(passwordResetEmail.value)">
13.    </div>
14. </div>
15. <div class="redirectToLogin">
16.     <span>Go back to ? <span class="redirect" routerLink="/sign-in">Log In</span></span>
17. </div>
18. </div>
19. </div>

```

Edit the component class as follow

```

1. import { Component } from '@angular/core';
2. import { AuthService } from '../authentication/auth.service';
3.
4. @Component({
5.   selector: 'app-password-forgot',
6.   templateUrl: './password-forgot.component.html',
7.   styleUrls: ['./password-forgot.component.css']
8. })
9. export class PasswordForgotComponent {
10.   constructor(private authService: AuthService) {
11.   }
12.
13.   passwordReset(email: string): void {
14.     this.authService.passwordReset(email)
15.       .then(() => {
16.         window.alert('Password reset email sent, check your inbox.');

```

Header Component

We update the Header Component to show the logged-in user name and to present a button for sign out. Edit `src/app/header/header.component.html` as follow.

```

1. <nav class="navbar navbar-expand-lg navbar-fixed-top shadow">
2.   <div class="container-fluid">
3.     <div class="navbar-header">
4.       <a href="#" class="navbar-brand">Address Book</a>
5.     </div>

```

```

6.   <div *ngIf="user | async as user">
7.     <ul class="navbar-nav mr-auto">
8.       <li class="nav-item active">
9.         <span class="navbar-text">{{user.displayName}}</span>
10.      </li>
11.      <li class="nav-item active">
12.        <a class="nav-link" (click)="signOut()">Sign Out </a>
13.      </li>
14.    </ul>
15.  </div>
16. </div>
17. </nav>

```

Edit `src/app/header/header.component.ts` as follow.

```

1. import { Component } from '@angular/core';
2. import { AuthService } from '../authentication/auth.service';
3. import { Router } from '@angular/router';
4. import { Observable } from 'rxjs';
5. import firebase from "firebase/compat/app";
6.
7. @Component({
8.   selector: 'app-header',
9.   templateUrl: './header.component.html',
10.  styleUrls: ['./header.component.css']
11. })
12. export class HeaderComponent {
13.   constructor(
14.     private authService: AuthService,
15.     private router: Router
16.   ) {}
17.
18.   get user(): Observable<firebase.User | null> {
19.     return this.authService.user;
20.   }
21.
22.   signOut(): void {
23.     this.authService.signOut().then(_) => {
24.       this.router.navigate(['sign-in']).then(() => {});
25.     });
26.   }
27. }

```

Firestore Service

In order to associate each of the application users to their own list of addresses, we will subdivide the address books (**abooks**) collection into documents each dedicated to a user, and identified with that user's id. Edit `src/app/address-list/firestore/address-db.service.ts` as follow.

```
1. import { Injectable } from '@angular/core';
2. import { AngularFireStore, DocumentChangeAction, DocumentReference } from
   "@angular/fire/compat/firestore";
3. import { Observable } from "rxjs";
4. import { AddressEntry } from "../address-entry";
5. import { AuthService } from "../../authentication/auth.service";
6.
7. @Injectable({
8.   providedIn: 'root'
9. })
10. export class AddressDbService {
11.
12.   constructor(
13.     private firestore: AngularFireStore,
14.     private authService: AuthService
15.   ) {}
16.
17.   getAddresses(): Observable<DocumentChangeAction<unknown>[]> {
18.     return this.firestore
19.       .collection('abooks')
20.       .doc(this.authService.userid)
21.       .collection('addresses')
22.       .snapshotChanges();
23.   }
24.
25.   createAddress(address: AddressEntry): Promise<DocumentReference> {
26.     delete address.id;
27.     return this.firestore
28.       .collection('abooks')
29.       .doc(this.authService.userid)
30.       .collection('addresses')
31.       .add({...address});
32.   }
33.
34.   updateAddress(address: AddressEntry): Promise<void> {
35.     const addressId = address.id;
36.     delete address.id;
37.     return this.firestore
38.       .collection('abooks')
39.       .doc(this.authService.userid)
40.       .collection('addresses')
41.       .doc(addressId!)
42.       .update(address);
```

```

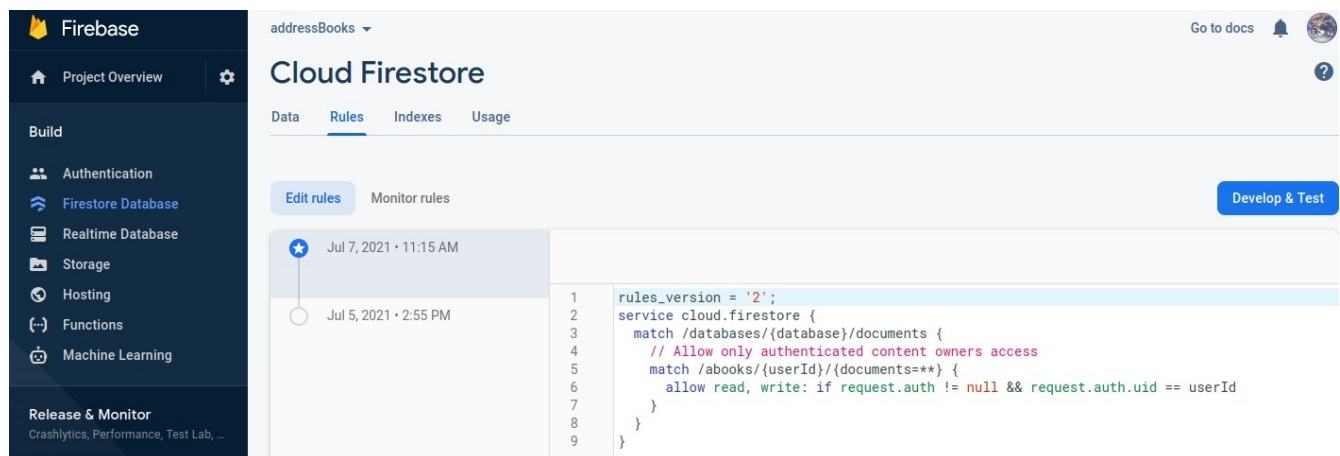
43. }
44.
45. deleteAddress(addressId: string): Promise<void> {
46.   return this.firestore
47.     .collection('abooks')
48.     .doc(this.authService.userid)
49.     .collection('addresses')
50.     .doc(addressId)
51.     .delete();
52. }
53. }

```

We inject the Authentication Service on line 14 and use the currently authenticated user's id to identify a document on lines 14, 20, 29, 39, and 48. This provides a specific document to each user within the data store.

Secure Firestore

Google Firebase allows us to add access rules to a Cloud Firestore database so that to avoid unauthorized access to our application data.



In the example above, we added a rule by which a request to read or write a document is only accepted from an authenticated user whose user id matches the identifier of the accessed document.

You can find more about Firestore Cloud security rules at <https://firebase.google.com/docs/firestore/security/get-started>.

Exercise

The following exercise is the deliverable for the lab. Complete and check-in the code to Github Classroom before the deadline. Only this exercise will be evaluated.

Extend the application developed as part of Lab4 exercise, to:

1. persist user information to a Firestore Cloud Datastore
2. allow for the retrieval and display of all the recorded user information in a table.