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/address-book-firebase.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

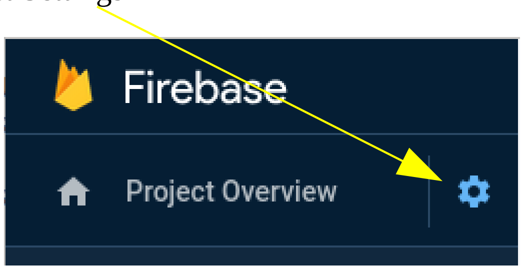
We will use the *Google Firebase Firestore* database (https://firebase.google.com/products/firestore). A real-time NoSQL database that can keep applications synchronized with the data store. Data is stored in the form of Collections made up of Documents.

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. Select to setup the *Firestore* feature.

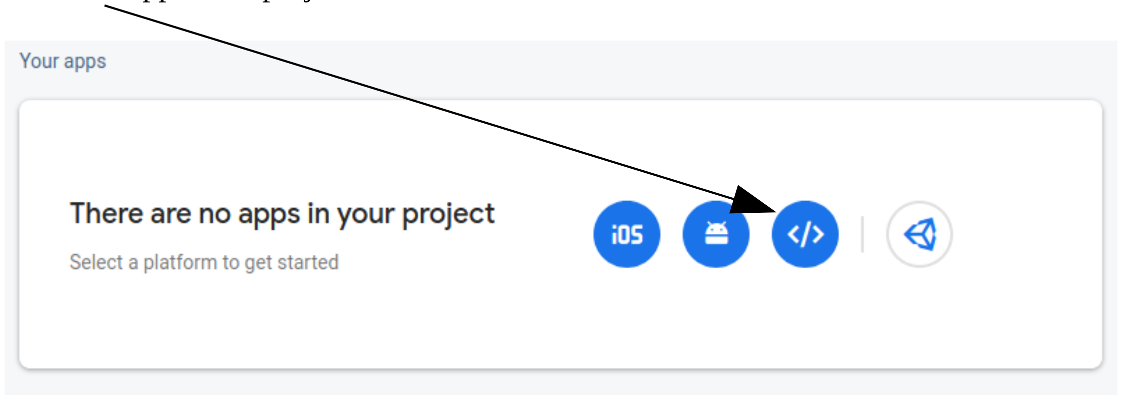
Firebase Setup

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.


Your apps
There are no apps in your project
Select a platform to get started
iOS

- Generate environment configuration files by running the command ng g environments
- Open the file `src/environments/environment.development.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.

```
export const environment = {
  firebase: {
    apiKey: '<your-key>',
    authDomain: '<your-project-authdomain>',
    databaseURL: '<your-database-URL>',
    projectId: '<your-project-id>',
    storageBucket: '<your-storage-bucket>',
    messagingSenderId: '<your-messaging-sender-id>',
    appId: '<your-app-id>',
    measurementId: '<your-measurement-id>'
  }
};
```

- Setup Firestore provider instance in `src/app.config.ts`.

```
import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import {initializeApp, provideFirebaseApp} from "@angular/fire/app";
import {getFirestore, provideFirestore} from "@angular/fire/firestore";
import {environment} from "../environments/environment.development";

export const appConfig: ApplicationConfig = {
  providers: [
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes),
    provideFirebaseApp(() => initializeApp(environment.firebase)),
    provideFirestore(() => getFirestore())
  ]
};
```

The call of function initializeApp() is required to pass the configuration object added earlier to the `src/environments/environment.development.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`)

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

```
import { Injectable, inject} from '@angular/core';
import {
  Firestore,
  collection,
  collectionData,
  addDoc,
  doc,
  setDoc,
  deleteDoc
} from '@angular/fire/firestore';
import {Observable} from "rxjs";
import {AddressEntry} from "../address-entry";

@Injectable({
  providedIn: 'root'
})
export class AddressDbService {
  private firestore:  Firestore = inject(Firestore);
19.
20.   getAddresses(): Observable<AddressEntry[]> {
21.     const addresses = collection(this.firestore, 'abooks', 'user', 'addresses');
22.     return collectionData(addresses, {idField: 'id'}) as Observable<AddressEntry[]>;
23.   }
24.
25.   createAddress(address: AddressEntry) {
26.     const addresses = collection(this.firestore, 'abooks', 'user', 'addresses');
27.     delete address.id;
28.     // @ts-ignore
29.     return addDoc(addresses, address);
30.   }
31.
32.   updateAddress(address: AddressEntry) {
33.     const addressId = address.id;
34.     delete address.id;
35.     const addresses = collection(this.firestore, 'abooks', 'user', 'addresses');
36.     const addressRef = doc(addresses, addressId!);
37.     // @ts-ignore
38.     return setDoc(addressRef, address);
39.   }
40.
41.   deleteAddress(addressId: string): Promise<void> {
42.     const addresses = collection(this.firestore, 'abooks', 'user', 'addresses');
43.     const addressRef = doc(addresses, addressId);
44.     return deleteDoc(addressRef);
45.   }
46. }
```

The Angular Firestore module provides an API to interact with the Firestore database. The API uses asynchronous communication implemented with observables and promises. A reference to the data store service is provided by an instance of Firestore injected on line 18.
Function getAddresses (lines 20-23) returns an Observable of documents (instances of AddressEntry). We start by calling function collection that returns a reference to a collection of documents based on a path. Documents are retrieved or added to a collection using that reference. Function collectionData get and returns the data asynchronously in an observable (line 22). 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, inject, OnInit} from '@angular/core';
2.   import {AddressEntry} from "./address-entry";
3.   import {NotificationService} from "./notification.service";
4.   import {AddressListElementComponent} from "./address-list-element/address-list-element.component";
5.   import {AddressViewComponent} from "./address-view/address-view.component";
6.   import {NgForOf, NgIf, NgStyle} from "@angular/common";
7.   import {AddressDbService} from "./firestore/address-db.service";
8.
9.   @Component({
10.    selector: 'app-address-list',
11.    standalone: true,
12.    imports: [
13.      AddressListElementComponent,
    AddressViewComponent,
    NgForOf,
    NgIf,
    NgStyle
  ],
  templateUrl: './address-list.component.html',
  styleUrl: './address-list.component.css',
  providers: [NotificationService]
})
export class AddressListComponent implements OnInit {
  addresses: AddressEntry[] = [];
  currentAddress: AddressEntry | null = null;
  message: string = '';
  hideMsg = true;
  msgStyle = {
    color: '',
    'background-color': 'white',
    'font-size': '150%',
  };
  private notificationService: NotificationService = inject(NotificationService);
  private store: AddressDbService = inject(AddressDbService);

  ngOnInit(): void {
    this.message = '';
    this.store.getAddresses().subscribe(data => {
      this.addresses = data.map(e => {
        return {
          id: e.id,
          ...e
        } as AddressEntry;
      });
    });
  }

  select(address: AddressEntry): void {
    this.currentAddress = address;
    this.notificationService.selectionChanged(address);
  }

  showMessage(type: string, msg: string): void {
    this.msgStyle.color = type === 'error' ? 'red' : 'blue';
    this.message = msg;
    this.hideMsg = false;
    setTimeout(
      () => {
        this.hideMsg = true;
      }, 2500
    );
  }
```

```
63.
64.   addAddress(): void {
65.     const newAddress = new AddressEntry('New', 'Entry');
66.     this.addresses = [newAddress, ...this.addresses];
67.     this.select(newAddress);
68.   }
69.
70.   deleteCurrent(): void {
71.     if (this.currentAddress && this.currentAddress.id !== null) {
72.       this.addresses =
73.         this.addresses.filter((address: AddressEntry) => address !== this.currentAddress);
74.       // **** permanently delete
75.       this.store.deleteAddress(this.currentAddress.id!)
76.         .then(_ =>
77.           this.showMessage('info', 'The address entry was successfully deleted')
78.         )
79.         .catch(_ =>
80.           this.showMessage('error', 'Error unable to delete the address entry')
81.         );
82.       this.currentAddress = null;
83.     }
84.   }
85.
86.   saveCurrent(): void {
87.     if (this.currentAddress && this.currentAddress.id === null) {
88.       this.store.createAddress(this.currentAddress)
89.         .then(
90.           (docRef: any) => {
91.             this.currentAddress!.id = docRef.id;
92.             this.showMessage('info', 'The address entry was successfully saved');
93.           }
94.         )
95.         .catch((_: any) =>
96.           this.showMessage('error', 'Error unable to save the address entry')
97.         );
98.     } else {
99.       this.store.updateAddress(this.currentAddress!)
100.                  .then((_: any) =>
101.                    this.showMessage('info', 'The address entry was successfully updated')
102.                  )
103.                  .catch((_: any) =>
104.                    this.showMessage('error', 'Error unable to update the address entry')
105.                  );
106.              }
107.            }
108.          }
```
We use the AddressDb Service injected in line 34. The ngOnInit lifecycle hook (lines 36-46) gets all the documents and set up the addresses array with.

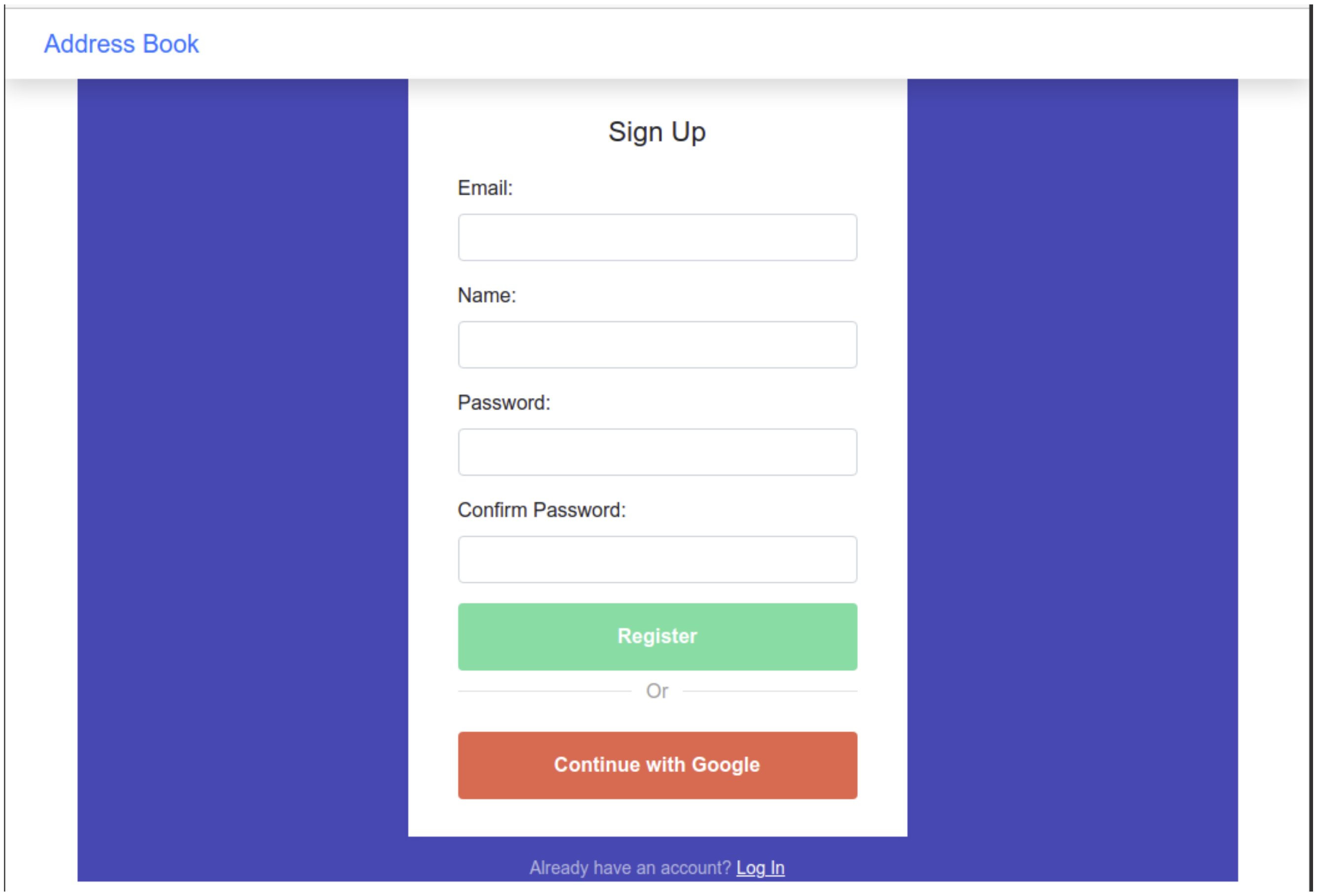
Function deleteCurrent (lines 70-84) has been modified to invoke the AddressDb service for the actual deletion. The call to deleteAddress (line 75) returns a Promise. Function then provides a handler for a notification of success while catch considers error situations. We set messages according to the status returned (line 77 and line 80).

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 91-92).

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


Address Book

Sign Up

Email:

Name:

Password:

Confirm Password:

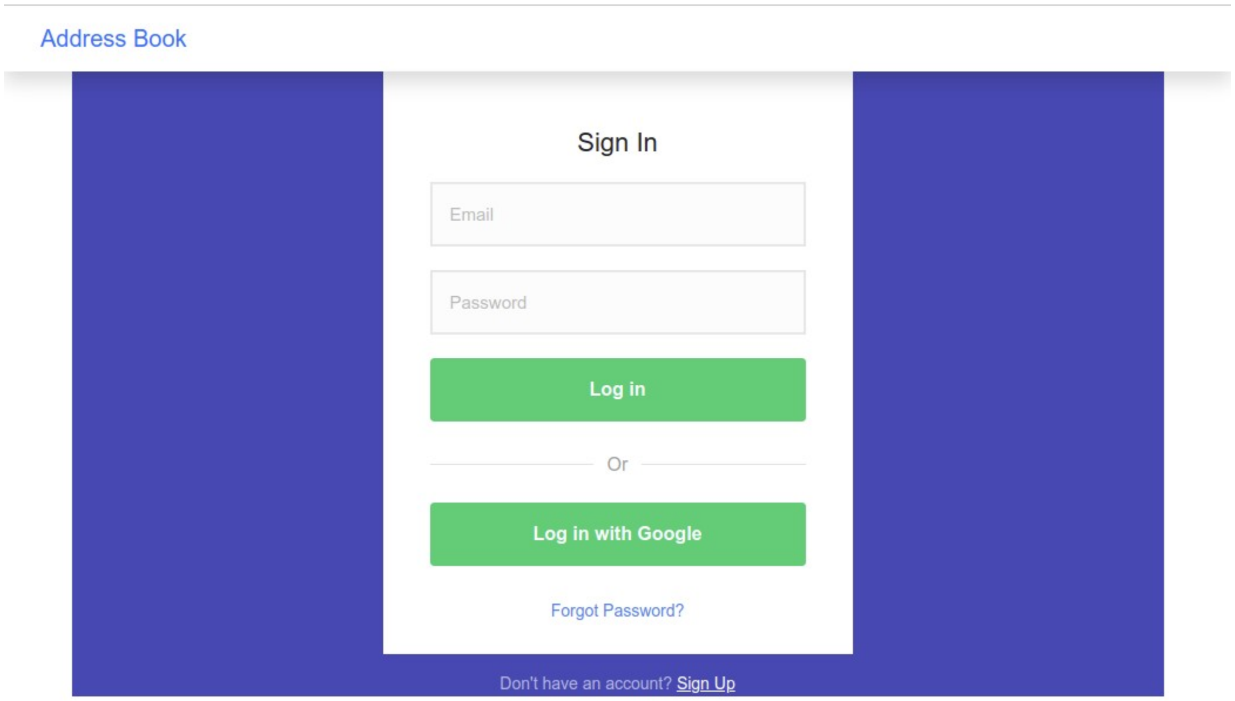
Register

Or

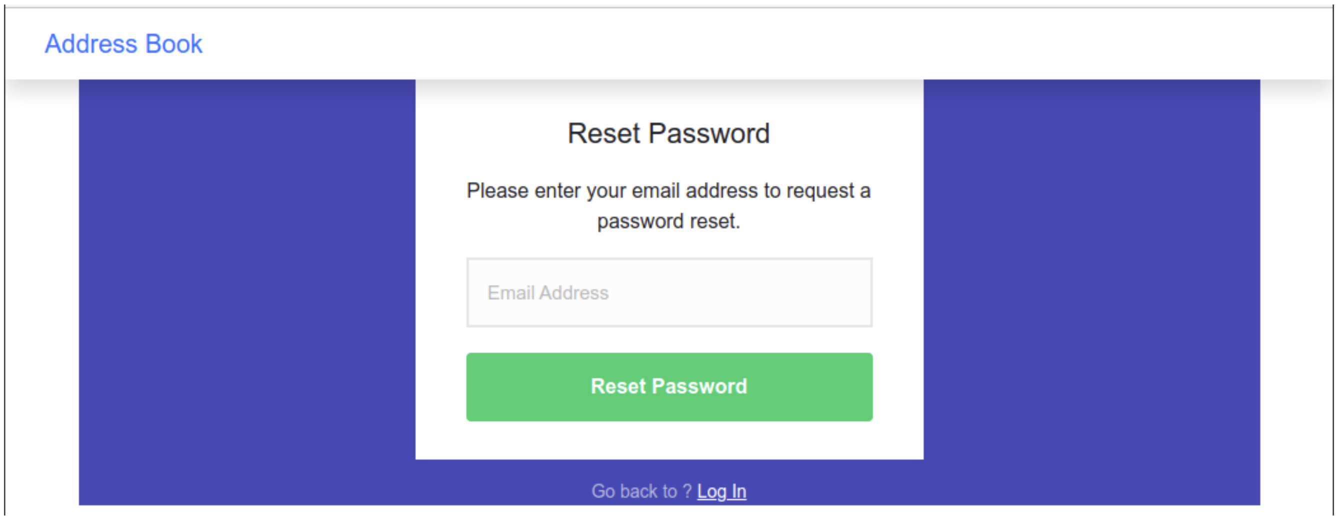
Continue with Google

Already have an account? Log In

A Sign-In page



A Password-Forgot Page

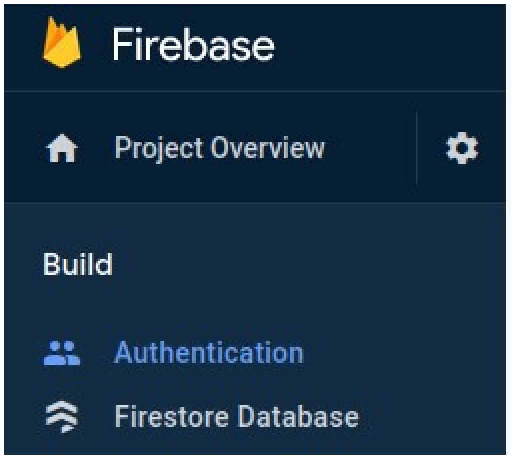


Please enter your email address to request a password reset.

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.

Auth instance

Configure authentication provisioning`src/app/app.config.ts`. The added elements are shown here.

```
...
import {getAuth, provideAuth} from "@angular/fire/auth";

export const appConfig: ApplicationConfig = {
  providers: [
    ...
    provideAuth(() => getAuth())
  ]
};
```

Routing

Edit `/src/app/app-routes.ts` to specify routes as follow.

```
import { Routes } from '@angular/router';
import {AuthGuard} from "@angular/fire/auth-guard";

export const routes: Routes = [
  { path: '', redirectTo: '/sign-in', pathMatch: 'full' },
  { path: 'sign-in', component: SignInComponent },
  { path: 'sign-up', component: SignUpComponent },
  { path: 'address-list', component: AddressListComponent, canActivate: [AuthGuard] },
  { path: 'password-forgot', component: PasswordForgotComponent }
];
```

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.

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`. Add imports of the generated components to `/src/app/app-routes.ts`.

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

```
<app-header></app-header>
<div class="container">
  <router-outlet></router-outlet>
</div>
```

Add RouterOutlet to the imports array of AppComponent.

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.

```
import { Injectable, inject } from '@angular/core';
import { Subscription} from 'rxjs';
import {
  Auth, User, user,
  createUserWithEmailAndPassword,
  sendPasswordResetEmail,
  signInWithEmailAndPassword, signInWithPopup, signOut,
  GoogleAuthProvider, updateProfile
} from "@angular/fire/auth";

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private afAuth: Auth = inject(Auth);
  private loggedUser: User | null = null;
```

```
user$ = user(this.afAuth);
userSubscription: Subscription;

constructor() {
  this.userSubscription = this.user$.subscribe((aUser: User | null) => {
    this.loggedUser = aUser;
  });
}

get userid(): string {
  if (this.loggedUser) return this.loggedUser.uid;
  return '';
}

signIn(email: string, password: string): Promise<void> {
  return signInWithEmailAndPassword(this.afAuth, email, password)
    .then((credential) => {
      this.loggedUser = credential.user;
    });
}

signUp(email: string, password: string, name: string): Promise<void> {
  return createUserWithEmailAndPassword(this.afAuth, email, password)
    .then((credential) => {
      const user = credential.user;
      this.loggedUser = credential.user; // shouldn't be null...
      if (user) {
        updateProfile(user, {displayName: name}).then(_ => {});
      }
    });
}

passwordReset(passwordResetEmail: string): Promise<void> {
  return sendPasswordResetEmail(this.afAuth, passwordResetEmail);
}

googleAuth(): Promise<void> {
  const provider = new GoogleAuthProvider();
  return signInWithPopup(this.afAuth, provider)
    .then((credential) => {
      this.loggedUser = credential.user;
    });
}

signOut(): Promise<void> {
  return signOut(this.afAuth)
    .then((_) => {
      this.loggedUser = null;
    });
66.   }
67. }
```

Firebase provides an API for the authentication services. The Firebase Auth service is represented as an instance of Auth. This object holds various properties including the currently signed-in user if any. We inject the instance on line 15 an pass it as argument to the various functions of the API. It is used on line 32 for sign-in with email/password, on line 39 for sign-up, on line 50 for password reset, on line 55 for sign-in with Google and on line 62 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 26-29.

Sign-Up Component

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

```
<div class="displayTable">
  <div class="displayTableCell">
    <div class="authBlock">
      <h3>Sign Up</h3>
      <form [formGroup]="signupForm" (ngSubmit)="register()">
        <div class="form-group">
          <label for="email">Email:</label>
          <input type="text" class="form-control" id="email"  formControlName="email">
          <div [hidden]="email.pristine || email.valid"
               class="alert alert-danger">
            Well formatted email is required.
          </div>
        </div>
        <div class="form-group">
          <label for="name">Name:</label>
          <input type="text" class="form-control" id="name"  formControlName="name">
          <div [hidden]="name.pristine || name.valid"
               class="alert alert-danger">
            A name must be provided.
          </div>
        </div>
        <div formGroupName="pwGroup">
          <div class="form-group">
            <label for="password">Password:</label>
            <input type="password" class="form-control" id="password" required
                   formControlName="password">
            <div [hidden]="password.pristine || password.valid"
                 class="alert alert-danger">
              A Password is required.
            </div>
          </div>
          <div class="form-group">
```

```
            <label for="pwconfirm">Confirm Password:</label>
            <input type="password" class="form-control" id="pwconfirm" required
                   formControlName="confirmPassword">
            <div [hidden]="confirmPassword.pristine || confirmPassword.valid"
                 class="alert alert-danger">
              A Confirmation Password is required.
            </div>
          </div>
          <div [hidden]="(password.pristine || confirmPassword.pristine) || pwGroup.valid"
               class="alert alert-danger">
            The passwords do not match.
          </div>
        </div>
        <button type="submit" class="btn btn-success" [disabled]="signupForm.invalid">Register</button>
      </form>
      <div class="formGroup">
        <span class="or"><span class="orInner">Or</span></span>
      </div>
      <!-- Continue with Google -->
      <div class="formGroup">
        <button type="button" class="btn googleBtn" (click)="googleSignIn()">
          Continue with Google
        </button>
      </div>
    </div>
    <div class="redirectToLogin">
      <span>Already have an account? <span class="redirect" routerLink="/sign-in">Log In</span></span>
    </div>
  </div>
</div>
```

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

Edit the component class as follow.

```
import {Component, inject} from '@angular/core';
import {AbstractControl, ReactiveFormsModule, UntypedFormBuilder, ValidationErrors, Validators} from "@angular/forms";
import {Router, RouterLink} from "@angular/router";
import {AuthService} from "../authentication/auth.service";

function passwordMatcher(pwGrp: AbstractControl): ValidationErrors | null {
  const passwd = pwGrp.get('password');
  const confpasswd = pwGrp.get('confirmPassword');
  return passwd!.value === confpasswd!.value ? null : {mismatch: true};
}

@Component({
  selector: 'app-sign-up',
  standalone: true,
  imports: [
    ReactiveFormsModule,
    RouterLink
  ],
  templateUrl: './sign-up.component.html',
  styleUrl: './sign-up.component.css'
})
export class SignUpComponent {
  private builder: UntypedFormBuilder = inject(UntypedFormBuilder);
  private authService: AuthService = inject(AuthService);
  private router: Router = inject(Router);
  signupForm = this.builder.group({
    email: ['', [Validators.required, Validators.email]],
    name: ['', Validators.required],
    pwGroup: this.builder.group({
      password: ['', Validators.required],
      confirmPassword: ['', Validators.required]
    }, { validators: [passwordMatcher] })
  });

  get email(): AbstractControl {return <AbstractControl>this.signupForm.get('email'); }
  get name(): AbstractControl {return <AbstractControl>this.signupForm.get('name'); }
  get password(): AbstractControl {return <AbstractControl>this.signupForm.get('pwGroup')!.get('password'); }
  get confirmPassword(): AbstractControl {return <AbstractControl>this.signupForm.get('pwGroup')!.get('confirmPassword'); }
  get pwGroup(): AbstractControl {return <AbstractControl>this.signupForm.get('pwGroup'); }

  register(): void {
    this.authService.signUp(this.email.value, this.password.value, this.name.value)
      .then((_) => {
        this.router.navigate(['address-list']).then(() => {});
      })
      .catch((error) => {
        window.alert(error.message);
      });
  }

  googleSignIn(): void {
    this.authService.googleAuth()
      .then((_) => {
        this.router.navigate(['address-list']).then(() => {});
      })
      .catch((error) => {
        window.alert(error.message);
      });
  }
}
```

The Authentication service is injected (line 24) and used in the handlers for the sign-up (lines 41-49). When a user select to continue with Google, we call a function to sign-in with Google (lines 51-59). 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.

```
<div class="displayTable">
  <div class="displayTableCell">
    <div class="authBlock pt-5">
      <h3>Sign In</h3>
      <div class="formGroup">
        <input type="text" class="formControl" placeholder="Email" #email required>
      </div>
      <div class="formGroup">
        <input type="password" class="formControl" placeholder="Password" #userPassword required>
      </div>
      <div class="formGroup">
        <input type="button" class="btn btn-outline-primary" value="Log in"
               (click)="signIn(email.value, userPassword.value)">
      </div>
      <div class="formGroup">
        <span class="or"><span class="orInner">Or</span></span>
      </div>
      <div class="formGroup">
        <button type="button" class="btn btn-outline-primary" (click)="googleSignIn()">
          Log in with Google
        </button>
      </div>
      <div class="forgotPassword">
        <span routerLink="/password-forgot">Forgot Password?</span>
      </div>
    </div>
    <div class="redirectToLogin">
      <span>
        Don't have an account?
        <span class="redirect" routerLink="/sign-up"> Sign Up</span>
      </span>
    </div>
  </div>
</div>
```

Edit the component class as follow.

```
import {Component, inject} from '@angular/core';
import {Router, RouterLink} from "@angular/router";
```

```
3.  import {AuthService} from "../authentication/auth.service";
4.
5.  @Component({
6.    selector: 'app-sign-in',
7.    standalone: true,
8.    imports: [
9.      RouterLink
10.   ],
11.   templateUrl: './sign-in.component.html',
12.   styleUrl: './sign-in.component.css'
13. })
14. export class SignInComponent {
15.   private authService: AuthService = inject(AuthService);
16.   private router: Router = inject(Router);
17.
18.   signIn(email: string, password: string): void {
19.     this.authService.signIn(email, password)
20.       .then(() => {
21.         this.router.navigate(['address-list']).then(() => {});
22.       })
23.       .catch((error) => {
24.         window.alert(error.message);
25.       });
26.   }
27.
28.   googleSignIn(): void {
29.     this.authService.googleAuth()
30.       .then((_) => {
31.         this.router.navigate(['address-list']).then(() => {});
32.       })
33.       .catch((error) => {
34.         window.alert(error.message);
35.       });
36.   }
37. }
```

We inject the Authentication service on line 15, and use it for the handlers of sign-in (lines 18-26) and sign-in with Google (lines 28-36).

Password-Forgot Component

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

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

```
          <input type="email" class="formControl"
                     placeholder="Email Address" #passwordResetEmail required>
        </div>
        <div class="formGroup">
          <input type="submit" class="btn btnPrimary" value="Reset Password"
                 (click)="passwordReset(passwordResetEmail.value)">
        </div>
      </div>
      <div class="redirectToLogin">
        <span>Go back to ? <span class="redirect" routerLink="/sign-in">Log In</span></span>
      </div>
    </div>
</div>
```

Edit the component class as follow

```
import {Component, inject} from '@angular/core';
import {RouterLink} from "@angular/router";
import {AuthService} from "../authentication/auth.service";

@Component({
  selector: 'app-password-forgot',
  standalone: true,
  imports: [
    RouterLink
  ],
  templateUrl: './password-forgot.component.html',
  styleUrl: './password-forgot.component.css'
})
export class PasswordForgotComponent {
  private authService: AuthService = inject(AuthService);

  passwordReset(email: string): void {
    this.authService.passwordReset(email)
      .then(() => {
        window.alert('Password reset email sent, check your inbox.');
      }).catch((error: any) => {
      window.alert(error);
    });
  }
}
```

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.

```
<nav class="navbar navbar-expand-lg navbar-fixed-top shadow">
```

```
  <div class="container-fluid">
    <div class="navbar-header">
      <a href="#" class="navbar-brand">Address Book</a>
    </div>
    <div *ngIf="user | async as user">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <span class="navbar-text">{{user.displayName}}</span>
        </li>
        <li class="nav-item active">
          <a class="nav-link" (click)="signOut()">Sign Out </a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

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

```
import {Component, inject} from '@angular/core';
import {AsyncPipe, NgIf} from "@angular/common";
import {User} from "@angular/fire/auth";
import {AuthService} from "../authentication/auth.service";
import {Router} from "@angular/router";
import {Observable} from "rxjs";

@Component({
  selector: 'app-header',
  standalone: true,
  imports: [
    NgIf,
    AsyncPipe
  ],
  templateUrl: './header.component.html',
  styleUrl: './header.component.css'
})
export class HeaderComponent {
  private authService: AuthService = inject(AuthService);
  private router: Router = inject(Router);

  get user(): Observable<User | null> {
    return this.authService.user$;
  }

  signOut(): void {
    this.authService.signOut().then((_) => {
      this.router.navigate(['sign-in']).then(() => {});
    });
  }
}
```

Firestore Service

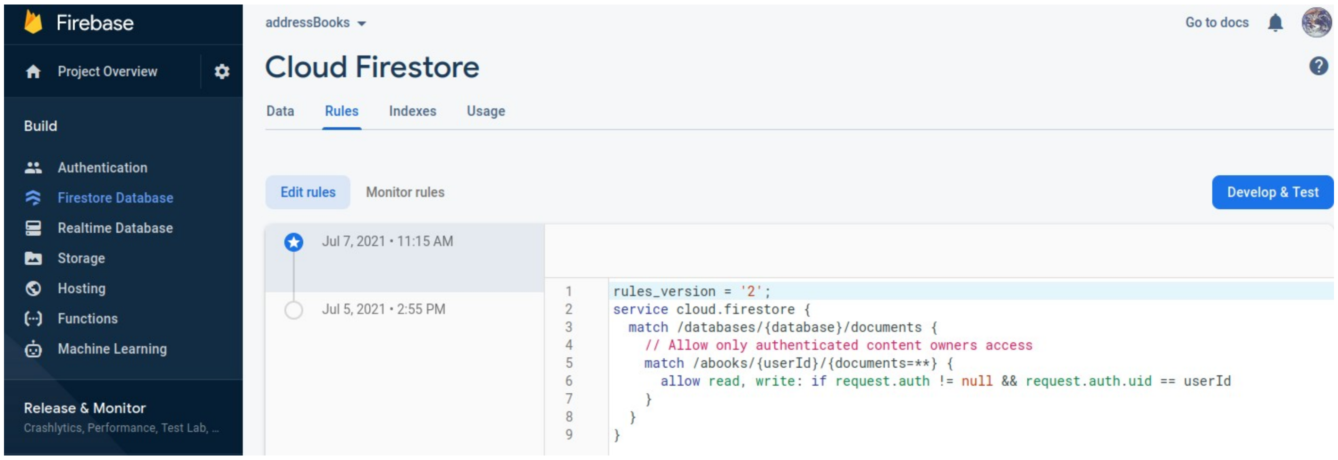
In order to associate each of the application users to their own list of addresses, we will subdivide the address books (abooks) collection into documents each dedicated to a user, and identified with that user's id. Edit `src/app/address-list/firestore/address-db.service.ts` as follow.

```
import { Injectable, inject} from '@angular/core';
import {
  Firestore,
  collection,
  collectionData,
  addDoc,
  doc,
  setDoc,
  deleteDoc
} from '@angular/fire/firestore';
import {Observable} from "rxjs";
import {AddressEntry} from "../address-entry";
import {AuthService} from "../../authentication/auth.service";

@Injectable({
  providedIn: 'root'
})
export class AddressDbService {
  private firestore:  Firestore = inject(Firestore);
  private authService: AuthService = inject(AuthService);

  getAddresses(): Observable<AddressEntry[]> {
    const addresses = collection(this.firestore, 'abooks', this.authService.userid, 'addresses');
    return collectionData(addresses, {idField: 'id'}) as Observable<AddressEntry[]>;
  }

  createAddress(address: AddressEntry) {
    const addresses = collection(this.firestore, 'abooks', this.authService.userid, 'addresses');
    delete address.id;
    // @ts-ignore
    return addDoc(addresses, address);
  }

  updateAddress(address: AddressEntry) {
    const addressId = address.id;
    delete address.id;
    const addresses = collection(this.firestore, 'abooks', this.authService.userid, 'addresses');
    const addressRef = doc(addresses, addressId!);
    // @ts-ignore
    return setDoc(addressRef, address);
  }
42.
43.   deleteAddress(addressId: string): Promise<void> {
44.     const addresses = collection(this.firestore, 'abooks', this.authService.userid, 'addresses');
45.     const addressRef = doc(addresses, addressId);
46.     return deleteDoc(addressRef);
47.   }
48. }
```

We inject the Authentication Service on line 20 and use the currently authenticated user's id to identify a document on lines 23, 28, 37, and 44. 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.



```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Allow only authenticated content owners access
    match /abooks/{userId}/{documents=**} {
      allow read, write: if request.auth != null && request.auth.uid == userId
    }
  }
}
```

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.

When you accept the assignment for this lab on Github, your repository will be initialized with some starter application code. Extend the provided starter application to:

1. persist employee information entered to a Firestore Cloud Datastore
2. allow for the retrieval from the Firestore Cloud Datastore, and display of all the recorded employees information in a table when button "Employee List" is clicked.

You must set the proper environment values so that access to your Firestore Datastore is possible.