

SEG3502 – Lab 5

Angular

Persistence et Authentification avec Firebase

L'objectif de cet laboratoire est d'examiner la persistance et l'authentification d'applications Angular à l'aide d'un service basé sur le cloud. Nous allons étendre l'application *Address Book* du Lab 3, en enregistrant et en lisant les entrées d'adresses à partir d'une base de données, et ayant des comptes de carnet d'adresses privés.

Le code source du laboratoire est disponible dans le référentiel Github

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

Persistence des entrées d'adresse

L'application *Address Book* ne stocke les entrées d'adresses qu'en mémoire. Ce qui n'est pas très utile. Nous allons étendre l'application avec la capacité de rendre les entrées d'adresses persistantes. Nous pourrions enregistrer les données localement au niveau client, mais cela ne supporterait pas plusieurs usagers ou ne permettrait pas l'utilisation de clients alternatifs tels que les clients mobiles.

Google Firestore

Nous utiliserons la base de données *Google Firebase Firestore*

(<https://firebase.google.com/products/firestore>). Une base de données NoSQL en temps réel qui permet de synchroniser les applications avec le magasin de données. Les données sont stockées sous forme de Collections constituées de Documents.

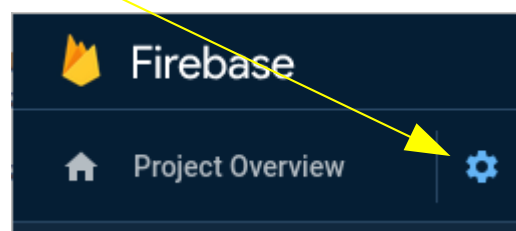
Le module AngularFire (<https://github.com/angular/angularfire>) doit être ajouté au projet pour pouvoir interagir avec Firestore. Dans le dossier racine du projet, exécutez la commande

`ng add @angular/fire` pour ajouter AngularFire au projet.

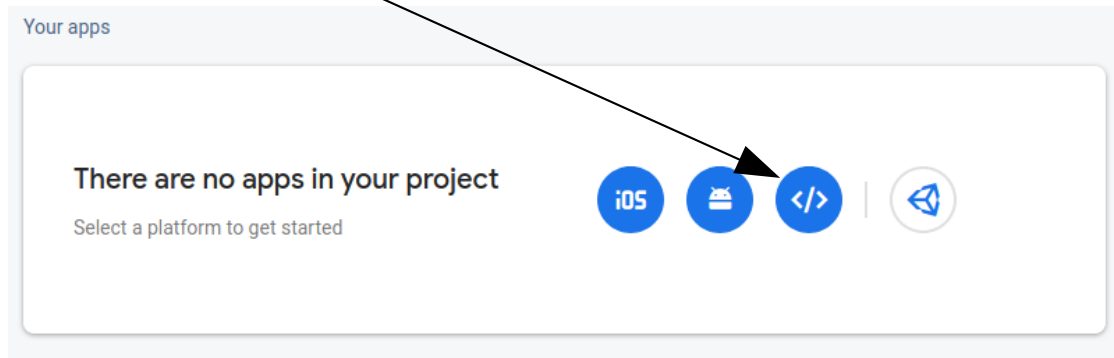
Configuration de Firestore

Les instructions de configuration d'une base de données Firestore Cloud sont décrites à l'adresse <https://firebase.google.com/docs/firestore/quickstart>. Nous allons résumer ici.

- Connectez-vous à la console Firebase <https://console.firebase.google.com/u/0/> puis ajoutez un projet (vous devez avoir un compte Google).
- Cliquez sur Firestore Database puis sélectionnez Create database. Sélectionnez de débiter en mode test (Start in test mode).
- Sélectionnez une location puis Enable.
- Sélectionnez les Settings du projet



- Ajoutez une application Web au projet.



- Ouvrez le fichier `src/environments/environment.ts` et copiez les données de configuration de Firestore Web App vers l'objet `firebase` de l'environnement de l'application. La valeur modifiée doit être similaire à ce qui suit avec les valeurs correspondant à vos paramètres.

```

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. };

```

- Importez les modules `AngularFireModule` et `AngularFirestoreModule` dans le module App (`src/app/app.module.ts`). Les éléments ajoutés sont montrés ici.

```

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.firebaseConfig),
12.         AngularFirestoreModule
13.     ],
14.     ...
15. })
16. export class AppModule { }

```

L'appel de la méthode `initializeApp()` de `AngularFireModule` est nécessaire pour transmettre l'objet de configuration ajouté précédemment dans le fichier `src/environments/environment.ts`.

Entrées d'Adresses

Nous ajouterons un champ `id` à la classe de modèle pour `AddressEntry`. Le champ servira d'identifiant unique aux entrées d'adresse stockées.

Modifiez la classe (`src/app/address-list/address-entry.ts`) comme suit

```

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. }

```

Nous avons ajouté un champ `id` à la classe (ligne 2) et l'initialisons à `null` dans le constructeur (ligne 11).

Option Save

Nous ajoutons un bouton à `Address View` qu'un utilisateur peut utiliser pour sauvegarder une entrée d'adresse.

Template HTML d'Address View

Modifier le template HTML du composant Address View (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>
```

Nous avons ajouté un bouton **Save** (lignes 36-38). La propriété **hidden** est définie de manière à ce que le bouton ne s'affiche que si le formulaire a été modifié. Une référence au **ngForm** (ajoutée à la ligne 1) permet d'accéder au statut du formulaire.

Class Address View

Le gestionnaire de l'événement clic du bouton **Save** est implémenté dans la classe de composant comme suit.

```
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. }
```

La fonction **save** (lignes 29 à 31) émet un événement output (ligne 12) notifiant le composant parent de la demande de suppression.

Service Firestore

Nous créons une classe de service pour encapsuler l'interaction avec la base de données Firestore. Le service serait injecté dans d'autres parties de l'application qui ont besoin d'un tel accès.

Générer un service **ng generate service address-list/firestore/address-db**. Modifiez `src/app/address-list/firestore/address-db.service.ts` comme suit.

```
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. }

```

Le module `AngularFirestoreModule` fournit une API pour interagir avec la base de données Firestore. L'API utilise la communication asynchrone implémentée avec des **observables** et des **promises**. Le service `AngularFirestore` injecté à la ligne 11 fournit des fonctions pour interagir avec la base de données, y compris la fonction `collection` qui retourne une référence à une collection de documents. Les documents sont lus ou ajoutés à une collection à l'aide de cette référence. Par exemple, la fonction `snapshotChanges` crée un flux de modifications synchronisées avec la base de données. Ces modifications incluent des références aux documents concernés.

La fonction `getAddresses` (lignes 13 à 19) retourne un observable des modifications du document. Elle est utilisée pour obtenir la liste des documents et garder cette liste synchronisée avec la banque de données. La fonction `createAddress` ajoute un document à la collection. Notez que Firestore crée automatiquement une collection lorsqu'elle n'existe pas. `updateAddress` modifie un document existant et `deleteAddress` supprime un document.

Composant Address List

Nous modifions à présent le composant `Address List` pour obtenir les entrées d'adresses à partir du service `AddressDbService` et pour gérer la sauvegarde ainsi que la suppression des entrées.

Composant Address List Template HTML

Modifiez le Template HTML (`src/app/address-list/address-list.component.html`) comme suit.

```
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>
```

Nous avons ajouté un gestionnaire pour l'événement `fireSave` qui est émis par le composant `Address View` lorsqu'un utilisateur clique pour enregistrer une entrée (ligne 23).

Un message s'affiche (lignes 5 à 7) pour fournir une rétroaction à l'utilisateur. Le style est contrôlé par une directive `ngStyle` et le message est masqué ou affiché en fonction de la liaison de propriété à une variable dans la classe de composant (`hideMsg`).

Class de composant d'Address List

La classe de composant de `Address List` est modifiée pour obtenir les entrées d'adresse à partir du service `AddressDb`.

```

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;
15.   message: string;
16.   hideMsg = true;
17.   msgStyle = {
18.     color: null,
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.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.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. }

```

Nous utilisons le service **AddressDb** injecté dans le constructeur (ligne 24). Le hook de cycle de vie **ngOnInit** (lignes 26-36) récupère tous les documents et initialise le tableau des adresses avec.

La fonction **deleteCurrent** (lignes 60-74) a été modifiée pour invoquer le service **AddressDb** pour la suppression réelle. L'appel à **deleteAddress** (ligne 65) retourne une **Promise**. La fonction **then** spécifie un gestionnaire pour le cas d'une notification de réussite tandis que **catch** prend en compte les situations d'erreur. Nous avons défini les messages en fonction du statut retourné (ligne 67 et ligne 70).

Une approche similaire est utilisée pour la fonction **saveCurrent** avec l'ajout qu'elle détermine si une nouvelle entrée est en cours de création ou qu'une mise à jour est effectuée en vérifiant la valeur de l'*id* de l'entrée d'adresse. Lorsqu'une nouvelle entrée est enregistrée, nous récupérons l'identifiant du document généré par Firestore et le définissons comme *id* pour la nouvelle entrée d'adresse (lignes 81-82).

Authentification avec Firebase

Nous allons à présent mettre à jour l'application avec l'authentification Firebase (<https://firebase.google.com/docs/auth>). Cela permettra à notre application d'être utilisée par plusieurs usagers. Chaque usager disposera d'un compte et ne pourra accéder qu'aux adresses qu'elle a enregistrées.

Nous allons ajouter une page d'inscription (Sign-Up)

Address Book

Sign Up

Email:

Name:

Password:

Confirm Password:

[Register](#)

Or

[Continue with Google](#)

[Already have an account? Log In](#)

Une page de connexion (Sign-In)

Address Book

Sign In

[Log in](#)

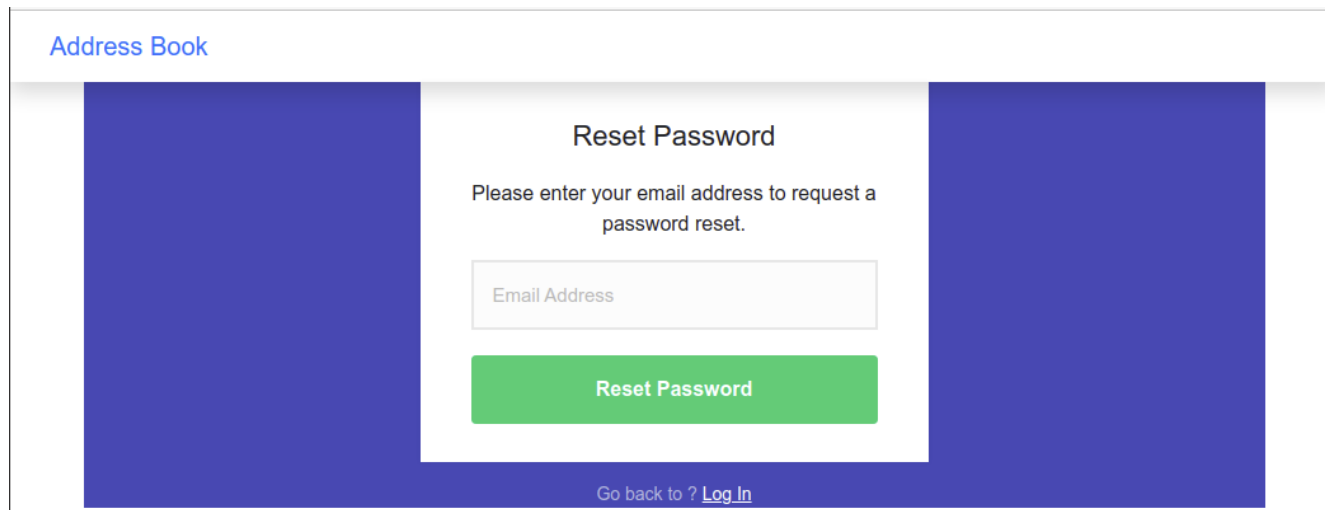
Or

[Log in with Google](#)

[Forgot Password?](#)

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

Une page de réinitialisation de mot-de-passe (Password-Forgot)

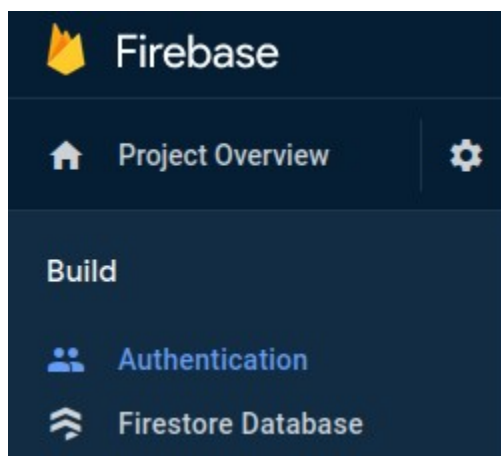


The screenshot shows a web interface for an "Address Book" application. At the top left, the text "Address Book" is displayed in blue. The main content area has a white background with a blue border. It features a "Reset Password" heading, followed by the instruction "Please enter your email address to request a password reset." Below this is a text input field labeled "Email Address". A green button labeled "Reset Password" is positioned below the input field. At the bottom of the white area, there is a link that says "Go back to ? [Log In](#)".

La page de connexion sera initialement présentée à un utilisateur et, une fois l'authentification réussie, l'application accédera à son carnet d'adresses.

Activation de l'authentification Firebase

Accédez à Google Firebase et sélectionnez **Authentication** dans le menu de la console du projet, puis cliquez sur **Get started**.



Firebase permet l'authentification à l'aide de plusieurs fournisseurs de connexion. Nous utiliserons la base de données Firestore stockant l'E-mail/mot de passe (*Email/Password*) et *Google* comme fournisseurs pour cette application.

Cliquez pour activer les fournisseurs de connexion *Email/Password* et *Google*. Pour Google, vous devez fournir une adresse e-mail pour le support du projet. Cette adresse e-mail sera présentée aux utilisateurs lorsqu'ils s'authentifieront auprès de Google comme contact.

Module de Routage

Ajouter un module de routage avec la commande `ng generate module app-routing --flat`. Modifiez `/src/app/app-routing.module.ts` comme suit.

```
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 { }
```

Nous avons défini des routes vers un composant Sign Up, un composant Sign In, un composant Password Forgot et l'application principale, le composant Address List. Nous utilisons AngularFireAuthGuard, une garde fournie par AngularFire pour assurer que seuls les utilisateurs authentifiés puissent accéder au composant Address List.

Enregistrez le module de routage avec le module App (`src/app/app.module.ts`) et importez le RouterModule, en mettant à jour le tableau des imports comme suit.

```
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. ]
```

Nous avons également importé `ReactiveFormsModule` que nous utiliserons pour le formulaire d'inscription.

Générez les composants Sign-In, Sign-Up et Password-Forgot: `ng generate component sign-in`, `ng generate component sign-up`, `ng generate component password-forgot`.

Mettez à jour le modèle HTML du composant APP (`src/app/app.component.html`) avec le *router outlet*.

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

Styling CSS

Modifiez le fichier CSS de l'application `src/styles.css` selon le code source sur GitHub.

Service d'authentification

Exécutez `ng generate service authentication/auth` pour créer un service pour l'interaction avec les services d'authentification de Firebase.

Modifiez `src/app/authentication/auth.service.ts` tel que suit.

```
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.loggedUser = 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.loggedUser = 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(new firebase.auth.GoogleAuthProvider())
49.     .then((credential) => {
50.       this.loggedUser = credential.user;
51.     });
52. }
53.
54. signOut(): Promise<void> {
55.   return this.afAuth.signOut()
56.     .then(_ => {
57.       this.loggedUser = null;
58.     });
59. }
60. }

```

Firebase fournit une API pour les services d'authentification, accessible via **AngularFirestore**, en utilisant une instance d'**AngularFireAuth**. Nous injectons cette instance dans le constructeur (ligne 13). Elle est utilisée aux lignes 26-29 pour la connexion par e-mail/mot de passe, aux lignes 33-40 pour l'inscription, à la ligne 44 pour la réinitialisation du mot de passe, aux lignes 48-51 pour la connexion avec Google et aux lignes 55- 58 pour se déconnecter. Les fonctions de l'API Firebase retournent des **Promises** auxquelles on s'abonne pour obtenir les résultats des opérations. Nous définissons la propriété *loggedUser* sur l'utilisateur connecté et retournons l'id de l'utilisateur connecté aux lignes 20-23.

Composant Sign-Up

Modifiez le modèle HTML du composant Sign Up `/src/app/sign-up/sign-up.component.html` comme suit.

```
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>
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>

```

Nous utilisons un formulaire réactif pour obtenir les données d'inscription et effectuons les validations usuelles.

Modifiez la class du composant comme suit.

```

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. }

```

Le service d'authentification est injecté (ligne 35) et utilisé dans le gestionnaire pour l'inscription (lignes 39-47). Lorsqu'un utilisateur choisit de continuer avec Google, nous appelons une fonction pour effectuer la connexion avec Google (lignes 49-56). Une fois la connexion réussie, l'application navigue à la page de la liste d'adresses.

Composant Sign-In

Modifiez le modèle HTML du composant de connexion `/src/app/sign-in/sign-in.component.html` comme suit.

```

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>

```

Modifiez la classe du composant comme suit.

```

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. }

```

Nous injectons le service d'authentification à la ligne 13 et l'utilisons dans les gestionnaires pour la connexion (lignes 17-23) et la connexion avec Google (lignes 27-33).

Composant Password-Forgot

Modifiez le modèle HTML du composant de réinitialisation de mot de passe

/src/app/password-forgot/password-forgot.component.html comme suit.

```

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>

```

Modifiez la classe du composant comme suit.

```
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.');
```

Composant Header

Nous mettons à jour le composant d'en-tête pour afficher l'utilisateur connecté et présenter un bouton de déconnexion. Modifiez `src/app/header/header.component.html` comme suit.

```
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>
```

Modifiez `src/app/header/header.component.ts` comme suit.

```
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. }

```

Service Firestore

Afin d'associer chacun des utilisateurs de l'application à sa propre liste d'adresses, nous allons subdiviser la collection de carnets d'adresses (**abooks**) en documents dédiés chacun à un utilisateur, et identifiés avec l'identifiant de cet utilisateur. Modifiez

`src/app/address-list/firestore/address-db.service.ts` comme suit.

```

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

```

```

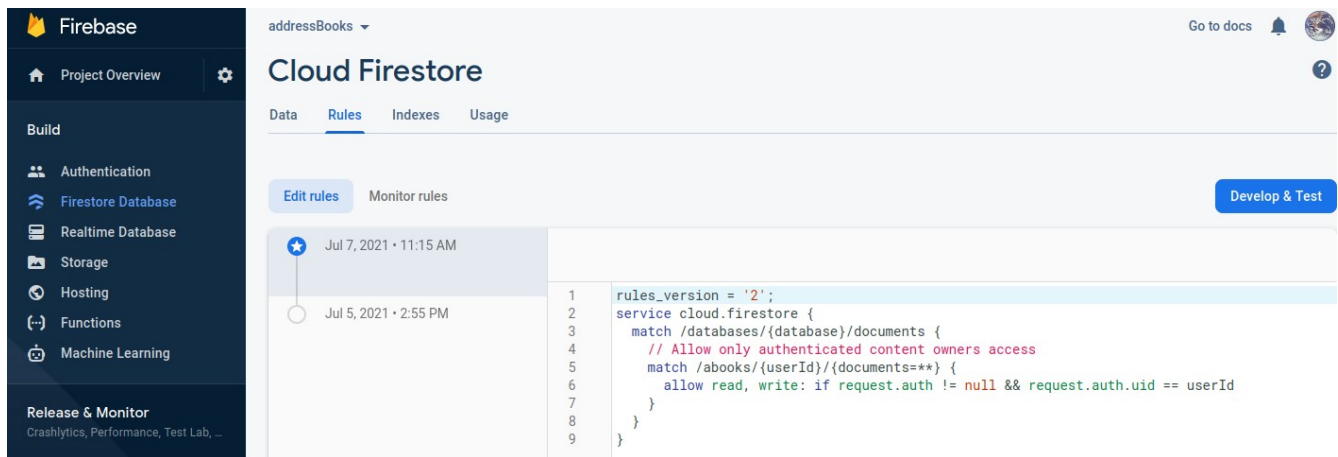
15.   } {}
16.
17.   getAddresses(): Observable<DocumentChangeAction<unknown>[]> {
18.     return this.firestore
19.       .collection('abooks')
20.       .doc(this.authService.userid)
21.       .collection('addresses')
22.       .snapshotChanges();
23.   }
24.
25.   createAddress(address: AddressEntry): 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. }

```

Nous injectons le service d'authentification à la ligne 14 et utilisons l'identifiant de l'utilisateur authentifié pour identifier un document aux lignes 14, 20, 29, 39 et 48. Cela fournit un document spécifique à chaque utilisateur dans le magasin de données.

Sécurisation de Firestore

Google Firebase nous permet d'ajouter des règles d'accès aux bases de données Cloud Firestore afin d'éviter tout accès non autorisé aux données d'applications.



Dans l'exemple ci-dessus, nous avons ajouté une règle par laquelle une requête de lecture ou d'écriture d'un document n'est acceptée que d'un utilisateur authentifié dont l'ID usager correspond à l'identifiant du document.

Vous pouvez en savoir plus sur les règles de sécurité de Firestore Cloud à <https://firebase.google.com/docs/firestore/security/get-started>.

Exercise

L'exercice suivant est le livrable pour le laboratoire. Complétez et enregistrez le code dans Github Classroom avant la date limite. Seul le travail de ce exercice sera évalué.

Étendre l'application développée dans le cadre de l'exercice Lab4, pour :

1. persister les informations utilisateur dans un Cloud Datastore Firestore,
2. permettre la récupération et l'affichage de toutes les données d'utilisateurs enregistrées dans un tableau.