TP SFFS : Firebase

Calvin EONO, Julio SANTILARIO BERTHILIER, Guillaume SONNET

1 Introduction : Firebase

Vous venez d'avoir une présentation sur le service de déploiement mis en place par Google, Firebase. Dans ce TP, vous allez pouvoir compléter une application web de référencement de livres utilisant le framework Angular. Les modules de Firebase qui seront explorés sont l'authentification, la base de données, et le stockage de fichiers (Storage).

Comme il vous a été communiqué avant cette séance, vous devez avoir au moins un compte Google par groupe ainsi qu'avoir récupéré le projet disponible sur le repository publique suivant : GitHub repository.

Une fois le git cloné, les fichiers d'intérêts se situent selon l'arbre représenté à la figure 1.

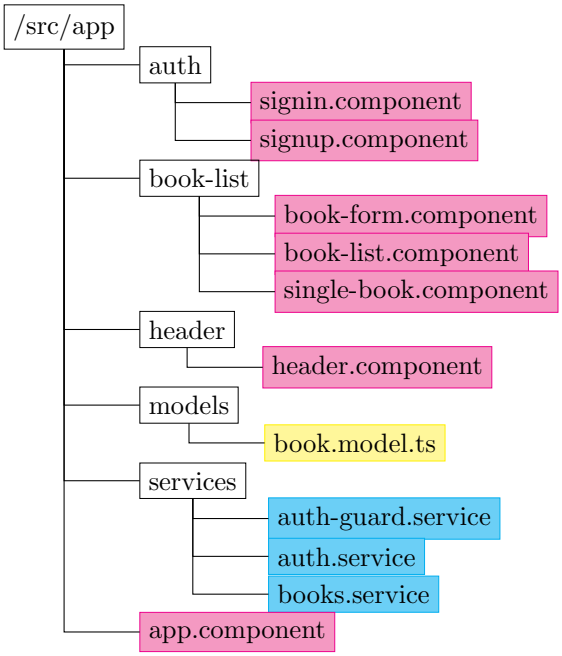

Figure 1: Arbre des fichiers avec les fichiers importants pour ce TP

Sur un terminal à la racine du projet, faites un **npm install** afin d'installer les dépendances, puis **npm install -g firebase-tools** pour pouvoir utiliser le firebase CLI pour la suite du TP. Dans la suite du TP, vous allez pouvoir déployer votre application en ligne, mais si vous voulez tester en local vous pouvez utiliser **ng serve** qui va lancer l'application sur http://localhost:4200/.

2 Création du projet et configuration

Q1- Pour cette application, vous allez créer un nouveau projet sur Firebase (figure 2). Vous devez le nommer de manière unique (avec l'assistance de l'interface), puis vous pouvez désactiver les Analytics pour ce projet. Vous accédez ensuite à la console (figure 3).

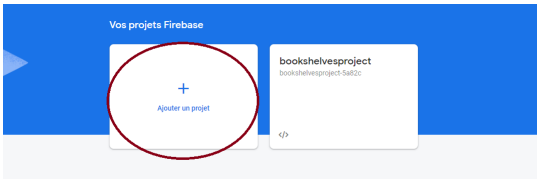

Figure 2: Nouveau projet Firebase

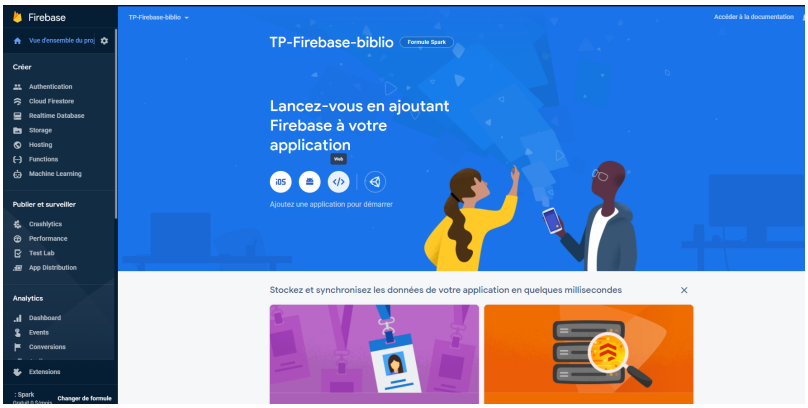

Figure 3: Console Firebase

Une fois l'application créée, la console Firebase vous propose le choix suivant : Android, IOS, Web. Sélectionnez application Web (figure 4).

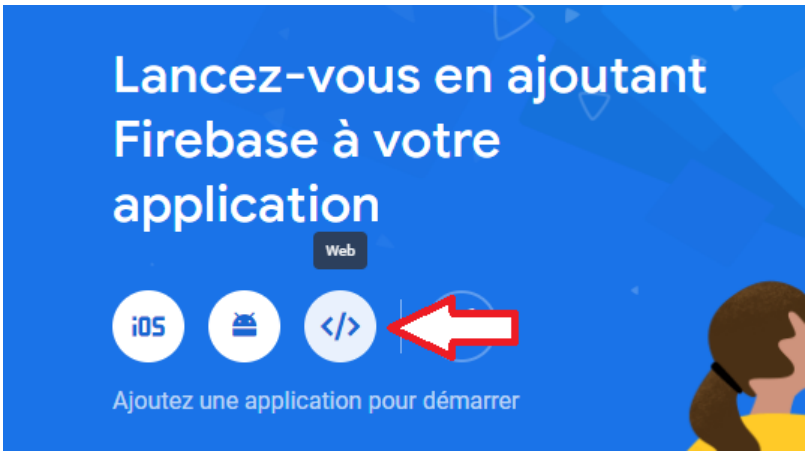

Figure 4: Sélection d'application web

Donnez un nom à votre application puis validez. Copiez-collez la configuration (figure 5) dans le constructeur de votre AppComponent.

Figure 5: Configuration Firebase à copier

Comme il vous a été expliqué, vous allez pouvoir déployer votre application en ligne. Ainsi elle sera accessible par tout le monde ayant accès à l'URL. Vous n'êtes pas obligés de faire le déploiement à ce moment du TP. Vous pouvez revenir à cette partie à tout moment (à condition que l'application build bien sûr !)

Pour déployer votre application, ouvrez un terminal à la racine du projet puis installez la CLI de Firebase soit via npm avec cette commande : **npm install -g firebase-tools**, soit via l'auto installer : **curl -sL https://firebase.tools/ — bash**. Lancez **firebase login** afin de vous connecter avec votre compte Google. Après cela, vous pouvez lancer **firebase init** qui va permettre de configurer le projet pour le déploiement. Plusieurs options vont vous être proposés. Suivez bien les instructions suivantes :

- Sélectionner 'Hosting' pour le choix de Firebase CLI feature.
- Sélectionner 'Use an existing project' et choisisser le projet que vous venez de créer.
- Le 'public directory' correspond au chemin qui contient le projet buildé. Dans notre cas, il s'agit de *dist/MyFirstFirebaseProject*. Renseigner donc ce chemin. Choisisser 'y' pour 'Configure as a single-page app' et 'n' 'Set up authomatic builds and deploys with GitHub'.

L'initialisation est faite. Si vous voulez déployer le projet, vous pouvez faire un **ng build**, suivi de **firebase deploy**. Votre application est maintenant disponible via l'URL de votre projet.

3 Authentification

Votre application utilisera l'authentification par adresse mail et mot de passe proposée par Firebase. Le coeur de cette fonctionnalité va se situer dans les fichiers **auth.service.ts**, **auth-guard.service.ts**, et les différents *components* gérant l'authentification y feront appel.

3.1 Configuration

Pour utiliser l'authentification, il faut d'abord l'activer dans la console Firebase. Dans l'onglet Authentification, appuyez sur "Commencer", puis, dans le menu Signin méthode, sélectionnez "Adresse e-mail/Mot

de passe" dans les méthodes de connexions (figure 6) pour l'activer, puis laissez désactivée la deuxième option "Lien envoyé par e-mail (connexion sans mot de passe)" puis enregistrez (figure 7).

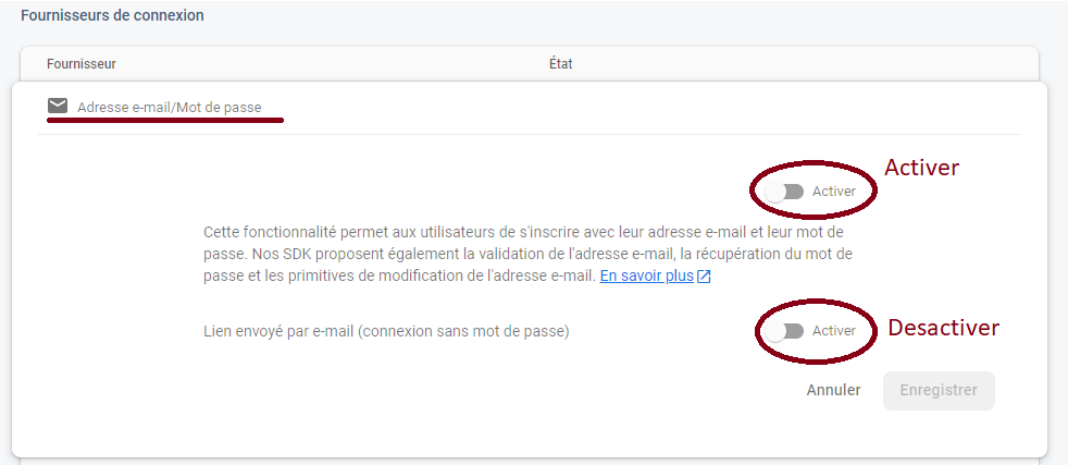

Figure 6: Activation Adresse e-mail/Mot de passe

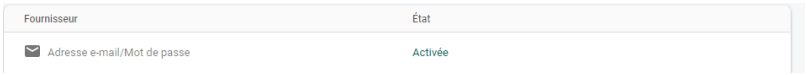

Figure 7: authentification Adresse e-mail/Mot de passe activé

3.2 Service d'authentification

Dans AuthService, nous avons trois méthodes :

- une méthode permettant de créer un nouvel utilisateur ;
- une méthode permettant de connecter un utilisateur existant ;
- une méthode permettant la déconnexion de l'utilisateur.

Puisque les opérations de création, de connexion et de déconnexion sont asynchrones, c'est-à-dire qu'elles n'ont pas un résultat instantané, les méthodes retourneront des Promise, ce qui permettra également de gérer les situations d'erreur.

(Un Promise est un objet qui lance une méthode avec un temps de réponse potentiellement long, qui lorsque elle n'échoue pas retourne un objet via sa méthode resolve, et lorsqu'elle échoue, lance des traitements avec la méthode reject. On peut traiter ces objets avec la méthode then(), qui précise les comportements à adopter en cas de réussite (resolve) ou d'échec (reject))

Q2- Dans le fichier **service/auth.service.ts**,

- Complétez la méthode signInUser, très similaire à createNewUser, qui s'occupera de connecter un utilisateur déjà existant.
- Complétez la méthode signOutUser() en une ligne à partir de firebase.auth().

Tips : Toutes les méthodes liées à l'authentification Firebase (createUserWithEmailAndPassword(), signInWithEmailAndPassword(), signOut(), etc) se trouvent dans firebase.auth().

3.3 Intégration de l'authentification dans votre application

Dans les *components* Signin et SignUp se trouvant dans auth/signin et auth/signout:

- vous générez le formulaire : les deux champs, email et password , sont requis — le champ email utilise Validators.email pour obliger un string sous format d'adresse email ; le champ password emploie Validators.pattern pour obliger au moins 6 caractères alphanumériques, ce qui correspond au minimum requis par Firebase ;
- vous gérez la soumission du formulaire, envoyant les valeurs rentrées par l'utilisateur à la méthode createNewUser() :
  - si la création fonctionne, vous redirigez l'utilisateur vers sa bibliothèque ;
  - si elle ne fonctionne pas, vous affichez le message d'erreur renvoyé par Firebase.

Q3- Complétez la méthode onSubmit() du SigninComponent lors de la soumission du formulaire en vous basant sur la méthode onSubmit() du SignUpComponent.

Q4- Créez un compte et connectez-vous sur votre application puis observez la présence du compte dans l'onglet "Users" sur la console Firebase.

Cependant vous remarquerez que même si vous êtes connecté, le header affiche toujours "Créer un compte" et "Connexion" comme si vous n'étiez pas connecté. Nous allons donc maintenant intégrer l'authentification dans le HeaderComponent afin de montrer les bons liens.

Q5- Pour cela, **décommentez le code des fonctions ngOnInit() et signOut()**, qui permet d'observer l'état de l'authentification de l'utilisateur : à chaque changement d'état, la fonction que vous passez en argument est exécutée.

Si l'utilisateur est bien authentifié, onAuthStateChanged() reçoit l'objet de type firebase.User correspondant à l'utilisateur, en ayant basée la valeur de la variable locale isAuth selon l'état d'authentification de l'utilisateur, et afficher les liens correspondant à cet état. Dans notre cas, les boutons de connexions disparaissent et laissent apparaître le bouton de Deconnexion.

Maintenant, déconnectez-vous sur l'application et essayez d'accéder à la route /books ou à l'une de ces sous-routes alors que vous êtes déconnecté. Vous serez automatiquement redigiré vers la page d'authentification.

En effet, l'authGuard (auth-guard.service.ts) permet de protéger la route /books et toutes ses sous-routes en fonction de la bonne connexion de l'utilisateur et de rediriger vers la page d'authentification si l'utilisateur n'est pas connecté. Puisque la vérification de l'authentification est asynchrone, le service retourne une Promise.

Ainsi, l'application comporte un système d'authentification complet, permettant l'inscription et la connexion/déconnexion des utilisateurs, et qui protège les routes concernées.

4 Base de données

Maintenant que l'authentification est activée, on va pouvoir débloquer un autre aspect de l'application : la base de données. Cette dernière nous permettra d'apporter des fonctions basiques pour gérer notre bibliothèque : visualiser des références, en ajouter de nouvelles ou tout simplement en supprimer. Le coeur de cette fonctionnalité va se situer dans le fichier **books.service.ts**, et les différents *components* gérant des livres y feront appel.

4.1 Configuration

A ce stade, notre application n'a pas de liens avec une base de données. Comme Firebase nous offre cette fonctionnalité, nous allons l'activer et l'ajouter à notre configuration de l'application.

Pour activer la fonctionnalité base de données pour notre application, il faut se rendre sur la console Firebase. Dans l'onglet **Realtime Database**, appuyez sur *"Créer une base de donnée"* et sélectionnez un emplacement (figure 8) pour votre base de donnés puis sur la page suivante sélectionnez *"Démarrer en mode test"* (figure 9). Enfin appuyez sur Activer. A cet instant, vous avez une base de données NoSQL vide de données.

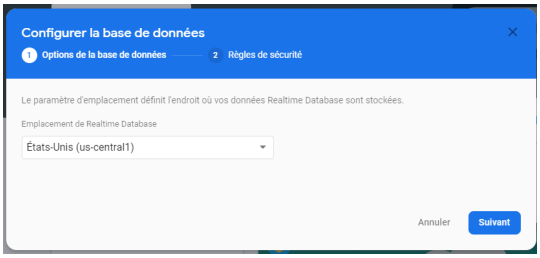

Figure 8: Configuration de l'emplacement de la base de donnée

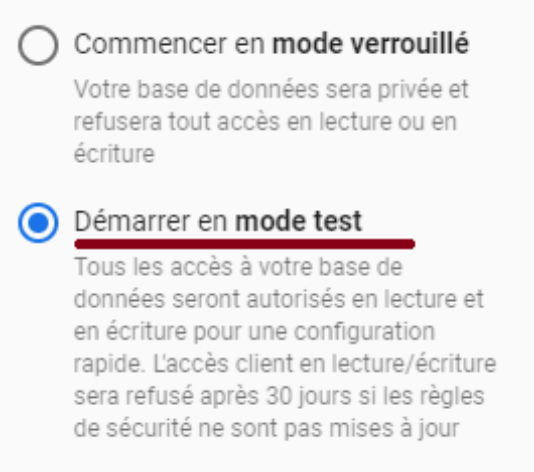

Figure 9: Configuration des règles de sécurité de la base de donnée

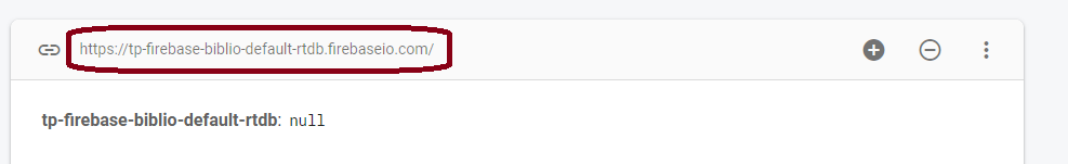

Figure 10: URL de la base de donnée

Pour connecter cette base à notre application, il faut retourner à notre application. Sur la console Firebase, vous avez un lien vers votre base de données qui vous est donné (figure 10), il faut ajouter ce lien dans un nouvel item de la variable firebaseConfig dans le fichier **app.component.ts**.

```
//app.componet.ts
const firebaseConfig = {
//autres entrées de configuration
databaseURL:"votre_url__de_base_de_donnees.firebaseio.com/"
};
```

4.2 Méthodes pour manipuler notre base de données

Pour dire quelques mots sur le modèle de données que nous allons utiliser, il s'agit d'un objet Book qui est composé de quatre champs string, un champ titre, un champ auteur (ces deux derniers sont obligatoires pour créer l'objet), un champ synopsis et un champ photo (représentant l'URL pour retrouver la photo du livre dans l'espace de stockage Firebase). Voici ci-dessous le modèle en TypeScript, tel que présent dans le fichier **book.model.ts**.

```
export class Book {
    photo: string;
    synopsis: string;
    constructor(public title: string, public author: string) {
    }
  }
```

Maintenant que notre application est liée à notre base de données, on va pouvoir s'intéresser aux opérations pour récupérer, insérer et supprimer des données. Pour ce faire, rendez-vous dans le fichier **books.service.ts**. On peut voir que le service possède :

- un attribut *books*, qui est un tableau de tous les Book de notre base. Les changements à notre base de données seront d'abord effectuées sur cet attribut, puis répercutés sur la base et les autres components.
- un attribut *booksSubject*, qui est un Subject renvoyant un tableau de Book. Pour faire court, un objet Subject prend les notifications d'une seule source observable et les transmets à un ou plusieurs observateurs, qui peuvent appliquer différents traitements. L'idée est que les *components* ayant besoin de la base de données souscrivent à cet attribut pour avoir une liste de Book à jour à chaque fois que l'on met à jour la base.
- une méthode emitBooks(), qui met à jour booksSubject avec l'attribut books et une méthode *saveBooks* pour sauvegarder nos changements vers la base.
- des méthodes pour insérer (createNewBook), obtenir (getBooks et getSingleBook), et supprimer (removeBook) des données de la base, et une méthode pour ajouter des images (uploadFile).

Q6- Comme vous pouvez le voir, la fonction *saveBooks* est vide. Complétez la en utilisant l'objet

```
firebase.database()
```

Cette ligne vous permet de faire appel à l'API Firebase concernant les base de données. A cette ligne vous pouvez enchaîner des appels de méthodes. Vous aurez besoin dans l'ordre de la méthode

```
.ref("path-to-db")
```

donnant le chemin de votre base de données vers vos livres (par exemple le chemin '/books') ainsi que la méthode

```
.set(updatedVar)
```

, qui met à jour votre base avec la variable updatedVar.

Q7- Dans la foulée, on va aussi compléter la fonction getBooks. Pour ce faire, utiliser aussi l'objet

```
firebase.database()
```

Vous aurez besoin dans l'ordre de la méthode

```
.ref("path-to-db")
```

avec le même chemin que vous avez utilisé dans la question 6, puis la fonction

```
.on()
```

Cette fonction va nous permettre de récupérer les données de notre base. Compléter les arguments de cette fonction avec ce qui suit :

```
.on('value', (data: DataSnapshot) => {
          //Si on a pas de données, on retourne un tableau vide
          this.books = data.val() ? data.val() : [];
          //On émet les potentiels nouveaux changements
          this.emitBooks();
        }
      );
```

L'argument 'value' précise que l'on souhaite rapatrier les résultats de la base à chaque mise à jour au niveau précisé précédemment par la méthode ref. La méthode val() permet de récupérer les données de l'objet DataSnapshot, ici notre base de livres.

Q8- Pour finir, dé-commenter la fonction *getSingleBook*. Cette fonction renvoie un objet Promise. Ici, on récupère un Book dans notre base de données avec son identifiant (ici son index dans la base de données), on ne fait qu'une seule requête de lecture (méthode once), et on ne lit que s'il y a une valeur (argument value). Si il y a une donnée, on renvoit sa valeur avec la méthode resolve, et si il y a une erreur, on la propage avec la méthode reject. Notez que la méthode once() renvoie aussi une Promise, d'où l'utilisation de la méthode then() pour la traiter.
En annexe, dé-commenter la ligne 44 du fichier book-form.component.ts. Ainsi vous pourrez rajouter un livre via le form de ce component.

Maintenant que l'on peut interagir avec notre base, voyons comment les *components* utilisent notre Service. A titre d'exemple, prenons le fichier book-list.component.ts. Ce *component* utilise notre BooksService lors de son initialisation (méthode *ngOnInit()*): il souscrit à l'attribut booksSubject pour qu'à chaque fois que cet objet est mis à jour, le *component* mette à jour son propre attribut books pour montrer la dernière mise à jour de la base. Il appelle ensuite la méthode emitBooks du service pour mettre à jour ses valeurs. Remarquer que cette souscription est détruite lorsque le *component* est détruit lui aussi (méthode *ngOnDestroy()*). N'hésitez pas (si vous avez le temps) à regarder comment d'autres *components* utilisent ce service.

Envie de tester le travail accompli ? N'hésitez pas à déployer votre application en local (un petit **ng serve** suffira), à vous connecter et à ajouter quelques livres à votre bibliothèque. N'hésitez pas à jeter un oeil à la console Firebase à l'onglet *Realtime Database* pour voir que vos données sont effectivement ajoutées, et à vous déconnecter puis vous reconnecter sur votre application pour constater que les changements sont persistants.

5 Storage

Nous allons pouvoir passer à la partie stockage de fichiers. Dans le contexte du TP on veut pouvoir afficher une image correspondante au livre lorsqu'on clique dessus. Les étapes pour cela vont être de upload une image lors de la création du livre et de récupérer la référence à cette image lorsqu'on clique sur ce livre dans la liste.

5.1 Configuration

Avant toute chose, il va falloir activer la solution Storage dans la console Firebase. Pour cela allez dans l'onglet Storage présent à gauche de la console de votre projet. Vous arrivez sur la page du Cloud Storage. Cliquez alors sur le bouton "Commencer". Une pop-up doit apparaître, cliquez sur "suivant". Vous obtenez la fenêtre de sélection d'emplacement du Cloud Stockage (figure 11). Choisissez un emplacement, cliquez sur "OK" et attendez que le bucket se créé.

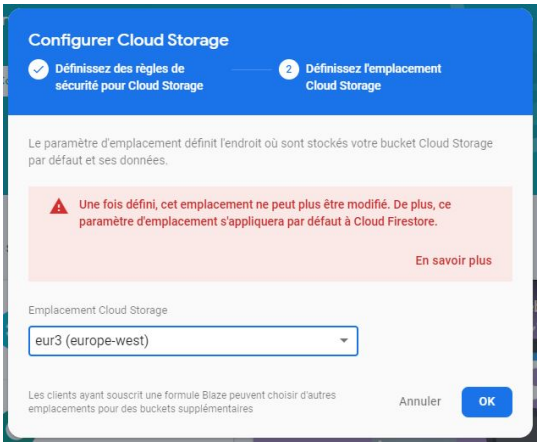

Figure 11: Sélection de l'emplacement du bucket

Lorsque cela est fait vous obtenez une page proche de la figure 12. Il s'agit de la page où tous vos fichiers vont être stockés. L'organisation des fichiers utilise une arborescence de fichiers classique. C'est-à-dire, vous pouvez créer des dossiers et stocker vos fichiers sous ces dossiers.

Figure 12: Storage après création du bucket

Normalement, vous n'avez pas à renseigner l'adresse du bucket dans la configuration. La configuration faite dans la partie 2 prend déjà en compte l'adresse de stockage.

5.2 Upload d'images

Le *component* BookFormComponent possède une méthode onUploadFile() qui utilise le service BookService afin d'upload l'image sur le serveur de stockage. Nous allons voir plus en détails ce processus dans cette partie.

Q10- Dans **book-form.component.ts**, décommenter l'appel de uploadFile() par l'attribut bookService. Cette méthode envoie un objet File qui va être utilisé par le service. Sachant que uploadFile() renvoie un objet Promise, nous pouvons donc utiliser la méthode then() pour rajouter un traitement lors de la réception. Dans notre cas, l'objet Promise retourne l'URL (le chemin du fichier sur le serveur) permettant de renseigner cette valeur pour l'attribut fileUrl.

Q11- Nous allons maintenant pouvoir compléter la méthode uploadFile().
Pour cela, ouvrez **books.service.ts**. Nous nous intéressons à la méthode uploadFile().

Complétez l'appel de child() pour la constante upload. Il s'agit ici de préciser l'emplacement où va être stocker le fichier. On souhaite stocker les images sous un dossier 'images/'. Pour le nom du fichier on peut utiliser la const almostUniqueFileName (permettant de tracer l'heure d'upload dans le nom du fichier) et l'attribut name de l'objet file.

Complétez maintenant le resolve() afin de retourner l'URL. Pour information, il est possible de récupérer une référence de upload de la manière suivante : **upload.snapshot.ref**. De plus, il est possible d'obtenir l'URL à partir d'une référence via la méthode getDownloadURL().

5.3 Suppression de l'image lors de la suppression d'un livre

Maintenant que nous avons vu l'ajout d'images, nous allons pouvoir voir la suppression d'images lorsqu'on supprime un livre. Toujours dans **books.service.ts**, nous allons regarder la méthode removeBook(). Cette méthode prend un Book en paramètre et supprime la référence de ce Book dans le storage, puis dans la base de données.

Q12- Compléter la constante storageRef afin d'obtenir la référence à partir de l'URL **book.photo**. Pour cela vous pouvez utiliser **firebase.storage()** permettant d'utiliser les méthodes propres au Storage. Pour rappel, on veut une référence (**ref**) **à partir d'une URL**.

Q13- Si vous avez le temps, vous pouvez vous rendre sur la console du projet dans la partie Storage (cf figure 12) afin d'observer la présence des images.

6 Pour aller plus loin...

Par soucis de simplification, l'authentification proposée est très basique. Il serait plus sûr d'ajouter une confirmation via l'envoi d'un mail, ou d'utiliser d'autres moyens d'authentifications offert par l'API. N'hésitez pas à consulter la documentation de Firebase pour offrir les moyens d'identification qui vous conviennent.
Doc : https://firebase.google.com/docs/auth/web/manage-users

Vous aimeriez rajouter des fonctions back-end pour votre application, par exemple un algorithme pour compresser les images de votre utilisateur pour ne pas surcharger votre base, ou de création de vignettes pour votre liste de livres ? Vous pouvez utiliser l'API Functions, qui fera tourner ces fonctions back-end comme des micro-services pour votre application.
Doc : https://firebase.google.com/docs/functions/

Envie de surveiller la performance de votre application ? Intégrez l'API Performance de Firebase à votre application pour instrumenter les requêtes réseau, surveiller la latence...
Doc : https://firebase.google.com/docs/perf-mon/

De manière générale, n'hésitez pas à explorez les différents aspects de votre application Firebase via les onglets à gauche de votre console !

Nous espérons que ce TP vous aura plu ;)

7 Si vous souhaitez supprimer votre projet

Si jamais vous voulez supprimer votre projet en ligne après ce TP : aller dans les paramètres dans la console du projet et descendre. Vous pouvez alors clicker sur "Supprimer le projet".

8 Crédits

Ce TP est issu du cours "Développez des applications Web avec Angular", proposez par Will Alexander sur la plateforme OpenClassRooms