# StageConnect

Hugo Siliveri

Lien vers le dépôt GitHub

## Présentation

L'objectif de ce projet était de réaliser une application pour centraliser l'ensemble du processus de création d'un stage, depuis la recherche du stage jusqu'à sa validation académique. C'est dans ce cadre que StageConnect a vu le jour. L'application a été développé en utilisant Jetpack Compose avec Material 3 pour le style. De plus, afin de faciliter le développement, Firebase a été utilisé pour gérer les données, les notifications et l'authentification. Enfin, l'application a été développée suivant le modèle MVVM (Model View ViewModel)

## Fonctionnalités

Dans StageConnect, il existe 3 types d'utilisateurs : les stagiaires, les entreprises et les établissements de formation. Chaque utilisateur a accès à différentes fonctionnalités selon son type.

### Fonctionnalités communes à tous les utilisateurs

| Fonction | Description |
|---|---|
| **Authentification** | Inscription, connexion, déconnexion via FirebaseAuth (email/mot de passe) |
| **Navigation conditionnelle** | Redirection automatique vers l'interface correspondant au rôle (stagiaire, entreprise, établissement) |
| **Messagerie intégrée** | Chat en temps réel entre deux utilisateurs, avec enregistrement des messages dans Firestore |
| **Profils utilisateurs** | Consultation des profils des utilisateurs, modification de son propre profil |
| **Notifications** | Réception de notifications via Firebase Cloud Messaging (ex : nouvelle candidature, acceptation d'une candidature, message reçu) |
| **Téléchargement de documents** | Téléchargement des CV et des conventions de stages grâce a un Intent |
| **Support multilangue** | Application disponible en français et anglais (anglais par défaut, détecte la langue de l'appareil) |
| **Thème personnalisé** | Grâce à Material 3, les couleurs de l'application changent en fonction de l'appareil (thème sombre et couleurs dynamiques) |

### Entreprises

| Fonction | Description |
|---|---|

| Fonction | Description |
|---|---|
| **Création d’offres** | Ajout d’offres de stage avec titre, description, durée, localisation |
| **Suppression** | Suppression d'une offre postée (suppression également des candidatures associées) |
| **Consultation des candidatures** | Liste des candidats avec leur profil |
| **Acceptation / refus** | Statut modifiable depuis l’interface |
| **Visualisation des stagiaires recrutés** | Suivi des stages en cours liés à l’entreprise |

### 🎓 Stagiaires

| Fonction | Description |
|---|---|
| **Parcourir les offres de stage** | Liste dynamique des offres avec filtre (nom d'entreprise, titre de l'offre, description de l'offre et localisation) |
| **Voir les détails d’une offre** | Description complète de l'offre de stage (titre, entreprise, durée, localisation, description du stage) |
| **Candidater à une offre** | Envoi d’une demande pour l'offre à l'entreprise |
| **Supprimer une candidature** | Possibilité d’annuler une candidature tant qu’elle n’est pas acceptée |
| **Suivi des candidatures** | Voir le statut (en attente / acceptée / refusée) et accéder aux offres correspondantes |
| **Suivi du stage** | Si une candidature est acceptée, un objet `Internship` est créé : suivi du processus jusqu’à la validation finale |
| **Gestion de documents** | Envoi de CV et gestion des signatures pour la convention de stage |

### 🏫 Établissements de formation

| Fonction | Description |
|---|---|
| **Liste des étudiants affiliés** | Voir les étudiants inscrits à l’établissement |
| **Accès aux stages associés** | Pour chaque étudiant, possibilité de consulter les stages en cours |
| **Validation des conventions** | Consultation et validation de la convention de stage soumis par l'étudiant |

## 🧱 Architecture et conception

### 💻 Frontend : Jetpack Compose (Material 3)

L'application suit le **modèle MVVM (Model - View - ViewModel)** pour une séparation claire des responsabilités :

- `MainActivity.kt` : Activité principale de l'application et qui gère la navigation
- `model/` : définitions des entités et interaction avec Firebase (`repository/`)
- `ui/` : organisation par rôles (`intern/`, `company/`, `educational/`) et authentification (`auth/`)
- `viewmodel/` : logique métier et gestion des états
- `utils/` : fonctions utilitaires
- `theme/` : thème du projet

## ☁️ Backend : Firebase

### 🔐 Authentification

- Firebase Authentication (email/password)
- Chaque utilisateur est enregistré dans la collection `users`.

### 📦 Firestore Database

- `users` : Collection des utilisateurs

```
{
  "uid": "String",
  "type": "String", // intern, company, educational
  "firstname": "String", // pour les stagiaires
  "lastname": "String", // pour les stagiaires
  "structname": "String", // pour les entreprises et les établissements de
formation
  "phone": "String",
  "adress": "String",
  "email": "String",
  "description": "String",
  "fcmToken": "String",
  "institutionId": "String", // pour les stagiaires
  "cvName": "String" // pour les stagiaires
}
```

- `offers` : Collection des offres de stage

```
{
  "id": "String",
  "companyId": "String",
  "companyName": "String",
  "title": "String",
  "description": "String",
  "duration": "String",
  "location": "String"
}
```

- `applications` : Collection des candidatures

```
{
  "id": "String",
  "userId": "String",
  "offerId": "String",
  "status": "String" // pending, accepted, denied
}
```

- `internships` : Collection des stages

```
{
  "id": "String",
  "offerId": "String",
  "userId": "String",
  "status": "String", // not_started, in_progress, finished
  "step": "Int",
  "agreementName": "String"
}
```

- `conversations` : Collection des conversations

```
{
  "id": "String",
  "lastMessage": "String",
  "timestamp": "Int",
  "userIds" : ["userId1", "userId2"],
  "messages" : [
    {
      "id": "String",
      "senderId": "String",
      "content": "String",
      "timestamp": "Int"
    }
  ]
}
```

Voici les règles que j'utilise dans Firebase pour la base de données :

```
service cloud.firestore {
match /databases/{database}/documents {

  match /users/{userId} {
    allow read, write: if request.auth != null;
  }
```

```
    match /offers/{offerId} {
      allow read, write: if request.auth != null;
    }

    match /applications/{applicationId} {
      allow read, write: if request.auth != null;
    }

    match /internships/{internshipId} {
      allow read, write: if request.auth != null;
    }

    match /conversations/{chatId} {
      allow read, write:  if request.auth != null;

      match /messages/{messageId} {
        allow read, write: if request.auth != null;
      }
    }
  }
  }
```

### ☁️ Firebase Storage

- CVs : `/users/{userId}/cv/`
- Conventions de stage : `/internships/{internshipId}/agreements/`

Voici les règles que j'utilise dans Firebase pour le stockage :

```
rules_version = '2';

service firebase.storage {
  match /b/{bucket}/o {
    match /users/{userId}/cv/{fileName} {
      allow read: if request.auth != null;
      allow write: if request.auth != null && request.auth.uid == userId;
    }

    match /internships/{internshipId}/agreements/{fileName} {
      allow read, write: if request.auth != null;
    }
  }
}
```

### 🔄 Firebase Functions

- `sendNotificationToUser` : envoi de notifications push.
- `cleanDeniedApplications` : suppression automatique des candidatures refusées après 2 jours.

Voici le code de mes fonctions Firebase :

```
import * as admin from "firebase-admin";
import { HttpsError, onCall } from "firebase-functions/https";
import {onSchedule} from "firebase-functions/scheduler";

admin.initializeApp();

interface Notification {
    uid: string;
    title: string;
    body: string;
}

/**
 * Envoie une notification à un utilisateur spécifique via Firebase Cloud
Messaging (FCM).
 *
 * @param request - Requête Firebase Functions avec les données :
 *   - `uid`: UID de l'utilisateur cible
 *   - `title`: Titre de la notification
 *   - `body`: Corps de la notification
 *   - `auth`: Contexte d'authentification Firebase
 *
 * @throws HttpsError - Si des champs sont manquants, l'utilisateur n'est pas
trouvé, ou le token FCM est absent.
 * @returns Un objet de succès ou un message expliquant pourquoi la notification
n'a pas été envoyée.
 */
export const sendNotificationToUser = onCall<Notification>(async (request) => {
    const { uid, title, body } = request.data || {};

    if (!uid || !title || !body) {
        throw new HttpsError(
            "invalid-argument",
            `Missing uid: ${uid}, title: ${title}, body: ${body}`
        );
    }

    try {
        const userDoc = await
admin.firestore().collection("users").doc(uid).get();

        if (!userDoc.exists) {
            throw new HttpsError("not-found", `User with uid ${uid} not found`);
        }

        const fcmToken = userDoc.data()?.fcmToken;

        if (!fcmToken) {
            throw new HttpsError("not-found", `FCM token not found for user
${uid}`);
        }
```

```
        const message = {
            token: fcmToken,
            notification: { title, body },
        };

        await admin.messaging().send(message);
        return { success: true };
    } catch (error: any) {
        throw new HttpsError("internal", error.message || "Unknown error");
    }
});

/**
 * Fonction planifiée (Cloud Scheduler) exécutée une fois par jour.
 *
 * Elle supprime tous les documents de la collection "applications"
 * ayant le statut "denied" **et** une date de refus (`deniedAt`)
 * datant de plus de 2 jours.
 *
 * @param event - Contexte d'exécution du scheduler (non utilisé ici)
 */
export const cleanDeniedApplications = onSchedule(
    {
        schedule: "every 24 hours",
        timeZone: "Europe/Paris",
    },
    async (event) => {
        const twoDaysAgo = admin.firestore.Timestamp.fromDate(
            new Date(Date.now() - 2 * 24 * 60 * 60 * 1000)
        );

        const snapshot = await admin.firestore()
            .collection("applications")
            .where("status", "==", "denied")
            .where("deniedAt", "<=", twoDaysAgo)
            .get();

        const batch = admin.firestore().batch();

        snapshot.docs.forEach(doc => {
            batch.delete(doc.ref);
        });

        await batch.commit();
    }
);
```

---

## ⚙️ Environnement de développement

- Minimum SDK : API 24 (Android 7.0 Nougat)

- Appareil virtuel : Pixel 8, Android 15 (API 35)
- Appareil physique testé : Samsung S20, Android 13 (API 33)
- Langage : Kotlin
- Framework UI : Jetpack Compose
- Backend : Firebase (Auth, Firestore, Storage, Functions)
- Architecture : MVVM