Compte rendu - Agora Mobile Sprint 1

Projet réalisé : Création d'une application mobile en React

Objectifs de la mission

L'association Agora souhaitait disposer d'une application mobile permettant de gérer les informations relatives aux jeux vidéo : jeux, genres, marques, classifications pegi et plateformes.

L'objectif de ce sprint était de mettre en œuvre les techniques de développement en JavaScript + React Native (framework Metro) afin de produire une application mobile fonctionnelle sous Android, en couvrant les points suivants :

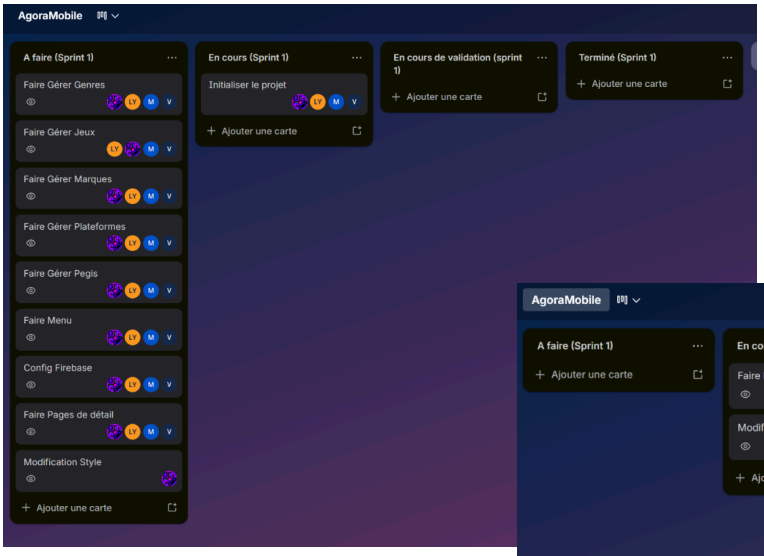
- Création d'une application React Native avec Expo
- Navigation multi-écrans via React Navigation (native-stack)
- CRUD simple sur les collections genres, PEGI, marques et plateformes
- Architecture adaptée à la base de données NoSQL Firebase Firestore
- CRUD plus complexe sur la collection jeux (documents imbriqués de type map)

Bien que l'objectif final sera de proposer un CRUD (Create, Read, Update et Delete) concernant la gestion des jeux, genres, marques, pegis et plateformes, l'application ne permettra pour l'instant que de consulter les différentes tables.

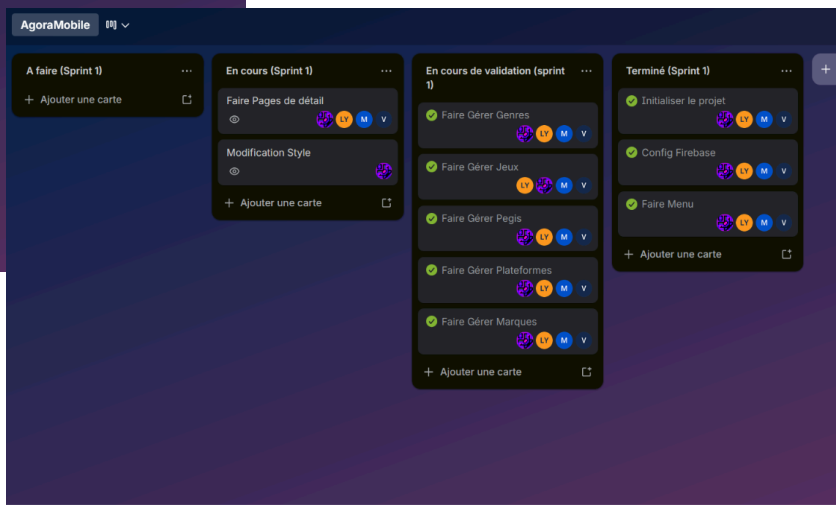
Organisation

Nous avons chacun réalisés intégralement le sprint, puis après une mise en commun, nous avons choisis le projet le mieux réalisé. Nous avons utilisés Trello pour représenter les tâches.

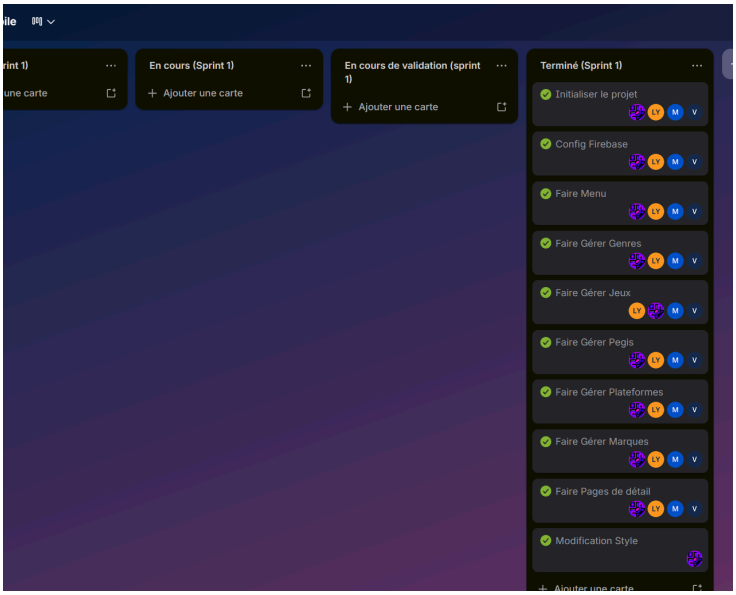
Au début nous avons ajouté les différentes tâches



Avancement au milieu du projet



A la fin du projet



Conception - diagrammes

Diagramme de cas d'utilisation

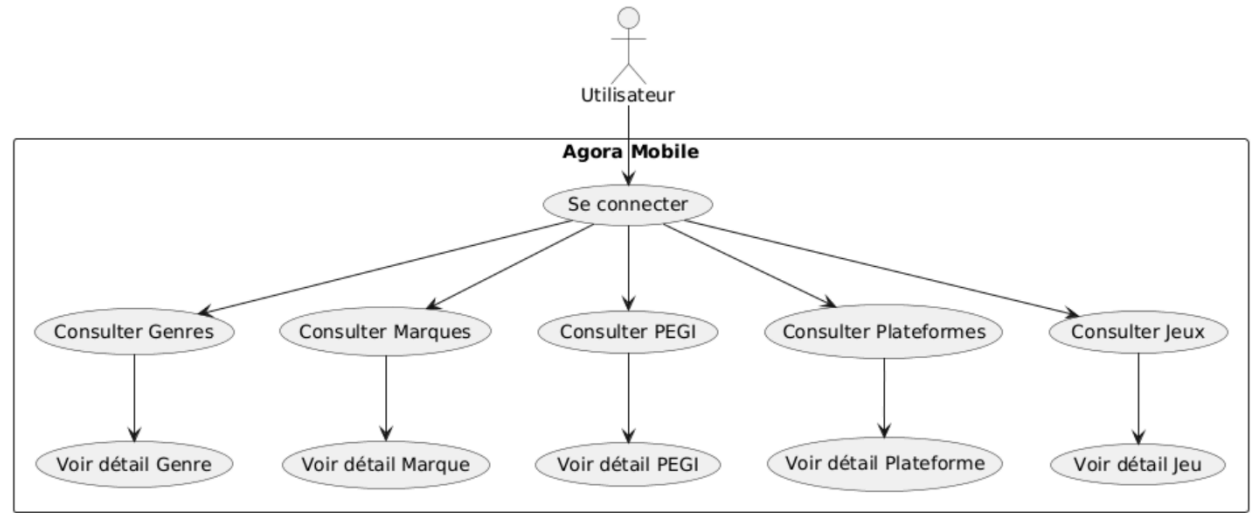
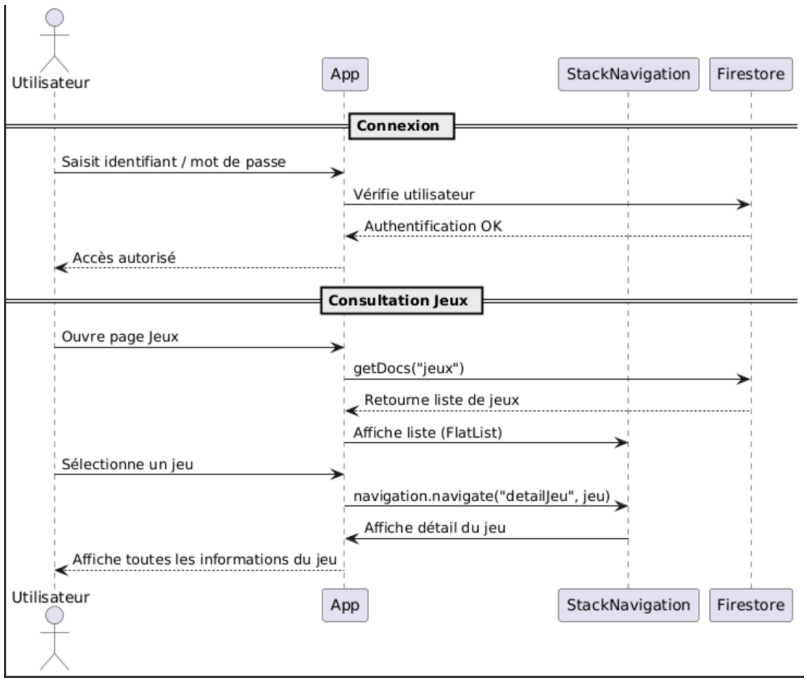


Diagramme de séquence



Technologies utilisées

Le projet a été développé avec les outils et technologies suivants :

- Node.js : environnement d'exécution JavaScript

```
D:\00_Prog\SIO2\Agora_Mobile\agramobile-sprint1>node -v
v22.14.0
```

- React Native : framework mobile cross-platform

```
"@react-native-firebase/app": "^23.8.6",
"@react-native-firebase/firestore": "^23.8.6",
"@react-navigation/native": "^7.1.33",
"@react-navigation/native-stack": "^7.14.4",
```

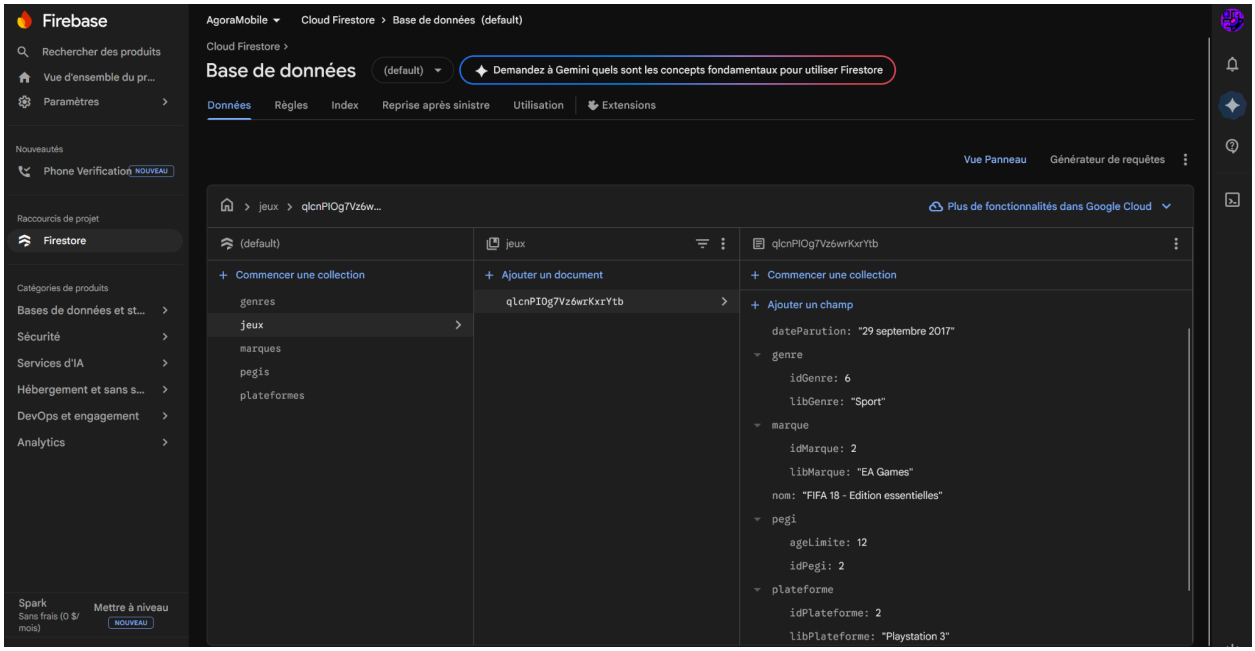
- Expo : chaîne de build et serveur Metro bundler

```
"expo": "~54.0.33",
"expo-status-bar": "~3.0.9",
```

- TypeScript (fichiers .tsx) : typage statique des composantspages

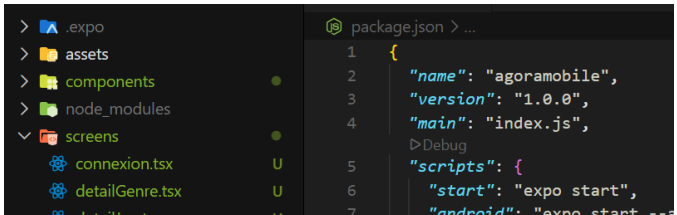
```
"@types/react": "~19.1.10",
"typescript": "~5.9.2"
```

- Firebase Firestore : base de données NoSQL cloud



Pour faire des tests :

- Android Studio + AVD (Android Virtual Device)
- Téléphone android (Exemple : Redmi note 13 Pro 5g)



L'application est structurée autour d'un fichier App.tsx central qui initialise le NavigationContainer et le stack de navigation (MainStack), et d'un dossier /screens/ regroupant l'ensemble des composants-écrans (.tsx). La configuration Firebase est isolée dans un dossier /services/firebaseConfig.js.

Etapes du projet

Création du projet et installation du framework Expo

Le projet a été initialisé avec la commande `npx create-expo-app --template` (template Blank). Après déplacement dans le répertoire et ouverture dans VS Code, le framework Expo a été installé globalement via `npm i -g expo-cli`.

Le serveur de développement se lance ensuite avec `npx expo start --clear`. Metro bundler publie l'application via une URL locale, et en scannant le QR code avec l'application Expo Go, on peut directement avoir l'application sur son téléphone.

Mise en place de la navigation multi-écrans

La navigation entre les pages repose sur React Navigation. Quatre pages ont d'abord été créées dans /screens/ : connexion.tsx, menu.tsx, gererLesGenres.tsx et detailGenre.tsx.

Le fichier App.tsx déclare le MainStack en référençant chaque écran par son nom de route (pageConnexion, pageMenu, pageGererLesGenres, pageDetailGenre). Chaque composant reçoit la prop navigation en paramètre, ce qui lui permet d'appeler navigation.navigate("nomDeLaPage") sur le onPress des boutons.

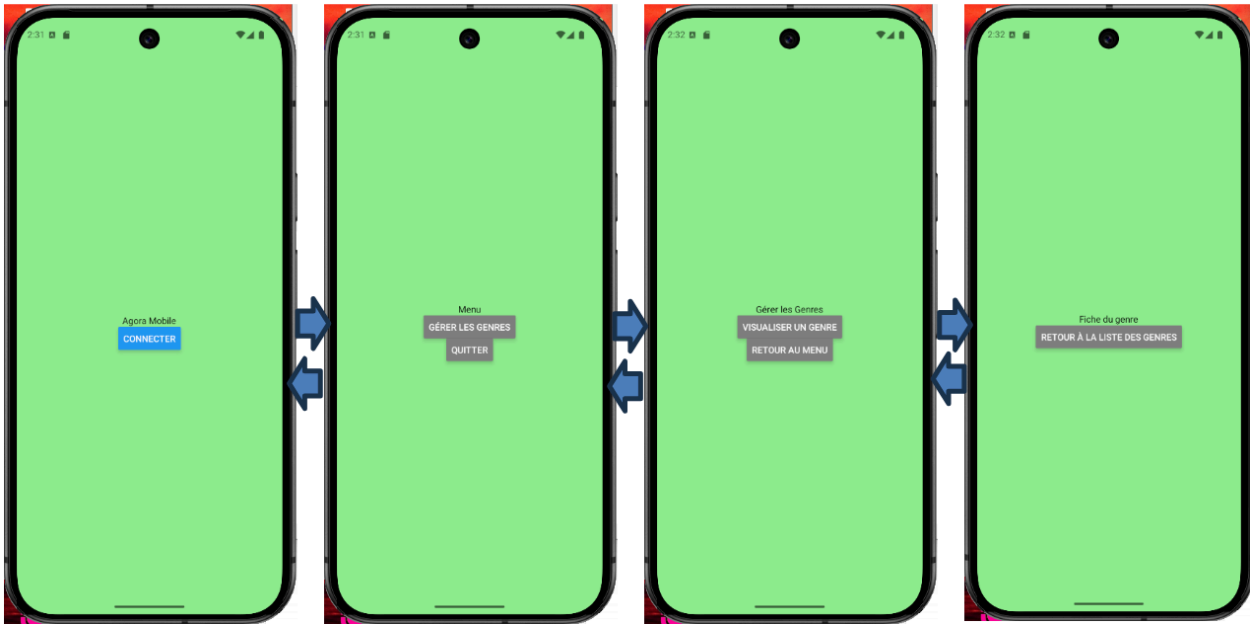
App.tsx

App.tsx > App

```tsx
import React from 'react';
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import { NavigationContainer } from "@react-navigation/native";
import Connexion from "./screens/connexion";
import Menu from "./screens/menu";
import GererLesGenres from "./screens/gererLesGenres";
import GererLesJeux from "./screens/gererLesJeux";
import GererLesMarques from "./screens/gererLesMarques";
import GererLesPegis from "./screens/gererLesPegis";
import GererLesPlateformes from "./screens/gererLesPlateformes";
import DetailGenre from "./screens/detailGenre";
import DetailJeu from "./screens/detailJeu";

const Stack = createNativeStackNavigator();
function MainStack() {
    return (
        <Stack.Navigator id="main-stack" initialRouteName="pageConnexion" screenOptions={{ headerShown: false }}>
            <Stack.Screen name="pageConnexion" component={Connexion} />
            <Stack.Screen name="pageMenu" component={Menu} />
            <Stack.Screen name="pageGererLesGenres" component={GererLesGenres} />
            <Stack.Screen name="pageGererLesJeux" component={GererLesJeux} />
            <Stack.Screen name="pageGererLesMarques" component={GererLesMarques} />
            <Stack.Screen name="pageGererLesPegis" component={GererLesPegis} />
            <Stack.Screen name="pageGererLesPlateformes" component={GererLesPlateformes} />
            <Stack.Screen name="pageDetailGenre" component={DetailGenre} />
            <Stack.Screen name="pageDetailJeu" component={DetailJeu} />
        </Stack.Navigator>
    );
}
export default function App() {
    return (
        <NavigationContainer>
            <MainStack />
        </NavigationContainer>
    );
}
```

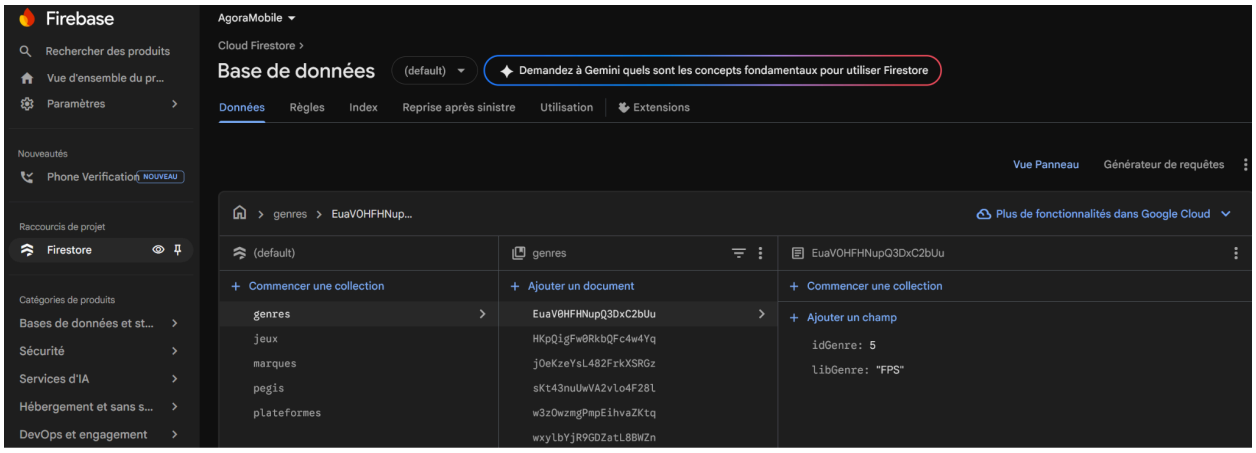
Navigation entre les pages :



Mise en place de la base de données Firebase Firestore

La base de données NoSQL utilisée est Cloud Firestore (Firebase). Le projet Firebase a été créé sur console.firebase.google.com avec l'emplacement europe-west9 (Paris), en mode test. Le SDK Firebase a été installé via npm install firebase.

Aperçu de la base de données :



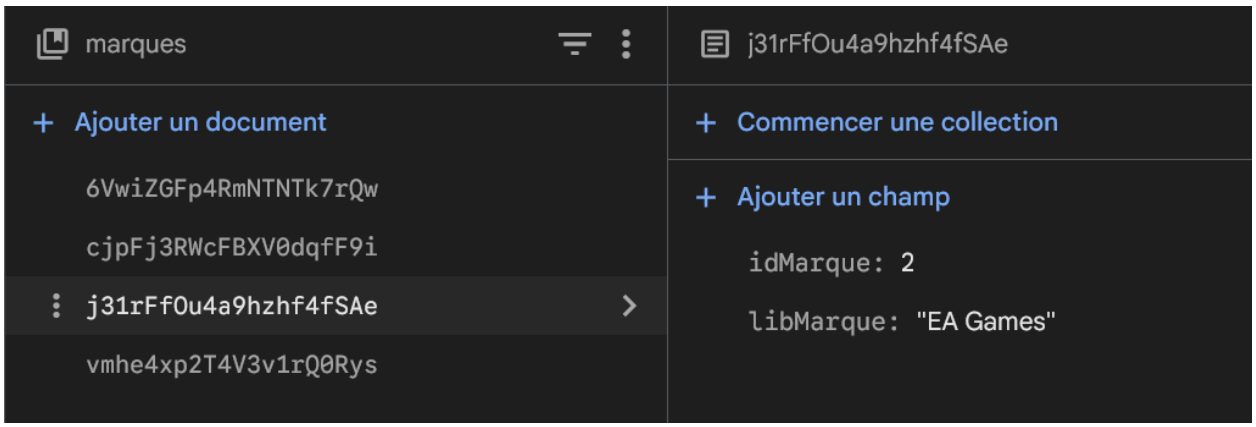
La connexion est configurée dans le fichier /services/firebaseConfig.js qui initialise l'application Firebase (initializeApp) et exporte l'instance Firestore (getFirestore). Les paramètres de connexion (apiKey, projectId, appId…) sont récupérés depuis la console Firebase dans la section "Installation et configuration du SDK".

services > firebaseConfig.js > ...

```
1  import { initializeApp } from "firebase/app";
2  import { getFirestore } from "firebase/firestore";
3
4  // Your web app's Firebase configuration
5  // For Firebase JS SDK v7.20.0 and later, measurementId is optional
6  const firebaseConfig = {
7      apiKey: "AIzaSyByWfZaKdgc3ATdFKu5TzBdmlopUNfH_eQ",
8      authDomain: "agoramobile-eb941.firebaseapp.com",
9      projectId: "agoramobile-eb941",
10     storageBucket: "agoramobile-eb941.firebasestorage.app",
11     messagingSenderId: "526674142113",
12     appId: "1:526674142113:web:b4edc676612d583451a1e0",
13     measurementId: "G-CS43CD8TE3"
14 };
15
16 // Initialize Firebase
17 const app = initializeApp(firebaseConfig);
18 const db = getFirestore(app);
19 export { db };
```

Une collection genres a été créée manuellement dans Firestore, avec des documents contenant deux champs : idGenre (number) et libGenre (string). Plusieurs documents ont été ajoutés pour les tests (Combat, Course, Plateforme, etc.).

Exemple de collection : les marques



Pages de gestion : genres, marques, PEGI et plateformes

La page gererLesGenres.tsx a servi de modèle pour l'ensemble des pages de consultation simples. Elle utilise le hook useEffect pour déclencher une requête asynchrone via getDocs(collection(db, "genres")) au montage du composant, et useState pour stocker le tableau de résultats.

L'affichage se fait avec le composant FlatList de React Native, en itérant sur les données. Chaque élément est rendu sous forme de card (View avec StyleSheet) puis rendu cliquable grâce au composant TouchableOpacity, qui déclenche une navigation vers la page de détail en passant l'objet sélectionné via route.params.

```
<FlatList
    style={styles.list}
    contentContainerStyle={styles.listContent}
    data={genres}
    keyExtractor={(item, index) => index.toString()}
    renderItem={({ item }) => (
        <TouchableOpacity style={styles.card} onPress={() => navigation.navigate("pageDetailGenre", { genre: item })}
            <Text style={styles.cardTitle}>{item.libGenre}</Text>
            <Text style={styles.cardSubtitle}>ID : {item.idGenre}</Text>
        </TouchableOpacity>
    )}
/>
```

La page detailGenre.tsx récupère le paramètre { genre } depuis route.params et affiche libGenre et idGenre. Sur ce même modèle, les pages gererLesMarques.tsx, gererLesPegi.tsx et gererLesPlateformes.tsx ont été créées et ajoutées au stack dans App.tsx, avec leurs boutons respectifs dans menu.tsx.

```
<View style={styles.accentStrip} />
<View style={[styles.containerCard, styles.centerCard]}>
    <Text style={detailStyles.title}>{genre.libGenre}</Text>
    <Text style={detailStyles.text}>ID : {genre.idGenre}</Text>
    <View style={styles.buttonGroup}>
        <TouchableOpacity style={styles.button} onPress={() => navigation.navigate("pageGererLesGenres")}>
            <Text style={styles.buttonText}>Retour à la liste des genres</Text>
        </TouchableOpacity>
    </View>
</View>
```

Collection jeux et pages de gestion des jeux

Architecture de la collection Firestore

Contrairement à une base SQL, Firestore ne gère pas de clés étrangères. La collection jeux a donc été conçue pour contenir toutes les informations nécessaires à l'affichage en une seule requête. Chaque document embarque des sous-objets de type map pour les données liées :

- genre : { idGenre, libGenre }
- pegi : { idPegi, ageLimite }
- plateforme : { idPlateforme, libPlateforme }
- marque : { idMarque, nomMarque }

Les champs scalaires sont refJeu (string), nom (string), prix (number) et dateParution (string). Cette dénormalisation est intentionnelle et alignée avec les bonnes pratiques Firestore pour minimiser le nombre de requêtes.

Pages gererLesJeux.tsx et detailJeu.tsx

Chaque jeu est affiché sous forme de card TouchableOpacity affichant le nom, le genre et la plateforme. Un appui navigue vers pageDetailJeu en passant l'objet jeu complet via route.params. La page detailJeu.tsx affiche alors l'ensemble des informations : référence, prix, date de parution, genre, pegi, plateforme et marque.
Les deux pages ont été enregistrées dans le stack de App.tsx et le bouton d'accès ajouté dans menu.tsx.

screens > detailJeu.tsx > ...

```tsx
import React from "react";
import { TouchableOpacity, View, Text, StyleSheet } from "react-native";
import { styles } from "../ui/commonStyles";

export default function DetailJeu({ route, navigation }) {
    const { jeu } = route.params;

    return (
        <View style={styles.screen}>
            <View style={styles.accentStrip} />
            <View style={[styles.containerCard, styles.centerCard]}>
                <Text style={detailStyles.title}>{jeu.nom}</Text>
                <Text style={detailStyles.text}>Ref : {jeu.refJeu}</Text>
                <Text style={detailStyles.text}>Prix : {jeu.prix} EUR</Text>
                <Text style={detailStyles.text}>Date de parution : {jeu.dateParution}</Text>
                <Text style={detailStyles.text}>Genre : {jeu.genre?.libGenre}</Text>
                <Text style={detailStyles.text}>Marque : {jeu.marque?.libMarque}</Text>
                <Text style={detailStyles.text}>Plateforme : {jeu.plateforme?.libPlateforme}</Text>
                <Text style={detailStyles.text}>PEGI : {jeu.pegi?.ageLimite}</Text>
                <View style={styles.buttonGroup}>
                    <TouchableOpacity style={styles.button} onPress={() => navigation.navigate("pageGererLesJeux")}>
                        <Text style={styles.buttonText}>Retour a la liste des jeux</Text>
                    </TouchableOpacity>
                </View>
            </View>
        </View>
    );
}
```

Conclusion et résultat final

L'ensemble des objectifs du sprint 1 ont été atteints. L'application Agora Mobile est fonctionnelle sur Android et couvre la consultation en lecture seule des cinq rubriques définies dans le cahier des charges. La navigation entre les écrans est fluide grâce au stack natif de React Navigation, et l'interface est homogène grâce à l'utilisation systématique des composants StyleSheet, FlatList et TouchableOpacity de React Native.

Le choix d'une architecture NoSQL dénormalisée pour la collection jeux est cohérent avec les contraintes de Firestore et permet d'afficher toutes les informations d'un jeu en une seule requête, sans jointure.

Ce sprint a permis de maîtriser les bases du développement mobile React Native: initialisation d'un projet Expo, navigation par stack, récupération de données Firestore avec useEffect/useState, affichage avec FlatList, et passage de paramètres entre écrans via route.params.