Université de Montpellier
Master 2 Informatique Génie Logiciel

Développement mobile avancé, IoT et embarqué

Firebase

Rapport du TP3

Étudiante :
GARCIA Léa

1 Lien vers dépôt GitHub

Lien vers dépôt GitHub https://github.com/Ravenn2203/RenduTP3Flutter.git

2 Utilisation de la base de données "Firebase Database"

2.1 Création et mise en place d'une base de données distante

Afin d'utiliser Firebase dans mon projet j'ai utilisé le tutoriel proposé et créé mon propre projet firebase, ajouté le fichier firebase-options.dart dans le dossier lib de mon projet et configuré les dépendances, etc. Une fois ceci installé, j'ai choisi le plugin 'Cloud Firestore' et j'ai créé ma propre base de données

Ensuite j'ai initialisé dans main.dart, ma base de données firebase avec les lignes de code suivant :

```
WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  FireStoreTP3();
```

Depuis le site FireBase j'ai choisi de représenter mes questions comme ceci : chaque thématique sera représentée par une collection, chaque document sera une question et chaque champ du document sera un élément de la question (lien image, texte ou booléen de réponse).

2.2 Création d'une page d'accueil

Rapidement, j'ai créé une page d'accueil HomePage depuis laquelle on peut choisir via des boutons, de faire un des quizz ou de créer des questions. Ces deux boutons renvoient sur la même page QuizzThemePage() mais avec un argument, 0 pour faire des quizz et 1 pour créer une question. Cette dernière page permet de choisir un des deux thèmes proposés. Une fois le thème choisi et en fonction du tout premier choix, soit le quizz est lancé, soit un formulaire est proposé pour rentrer une question.

FIGURE 2.1 – Accueil de l'application

FIGURE 2.2 – Choix des thèmes

2.3 Affichage des quizz

Si l'utilisateur choisi de faire une des quizz et choisi un thème, alors via le code 'BlocProvider.of<QuizzCubit>(context).retrieveData('disney') ;' on lance un récupération des données (en argument on donne le thème choisi, ici cette ligne vient du code qui dirige vers le quizz disney). La méthode retrieveData() permet de lancer un état Loading() tant que les données n'ont pas été récupérées (qui affiche des cercles d'attentes), comme on l'a vu dans les derniers TPs. Elle permet de récupérer les documents de la collection/-thème concernée, via une instance de 'FirebaseFirestore' puis elle fait appel à la méthode nextQuestion() pour récupérer la prochaine question dans la collection, qui est ici la toute première. Une fois la collection récupérée la fonction émet un QuizzLoaded() qui permet de reconstruire les widget concernés.

```dart
Future<void> retrieveData(String categorie) async {
  //On attend que les données soient récupérées
  emit(QuizLoading());

  QuerySnapshot<Map<String, dynamic>> retrievedData = await db.collection(categorie).get();
  collectionQuestions = retrievedData;

  retrievedData.docs.forEach((doc) {
    print('Document récupéré :'+doc.data().toString());
  });

  nextQuestion();

  //On a récupéré les données
  emit(QuizzLoaded());

}
```

FIGURE 2.3 – Code de la fonction retrieveData()

Lorsque l'utilisateur appuie sur le bouton vrai ou le bouton faux, un nouvel état est emis : QuizzAnswered, et permet de reconstruire les boutons avec la bonne couleur. Pour éviter de reconstuire le reste j'ai utilisé la fonction 'buildWhen' qui permet de définir les états où le blocBuilder doit reconstruire le widget.

Lorsque l'utilisateur passe à la question suivante, on appelle une fonction modifiée par rapport au TP précédent, qui s'appelle : nextQuestion() (voir figure 2.4). Elle récupère dans la collection, la prochaine question si il y en a une, et construit un objet Question qui est stocké dans une variable du cubit et permet de reconstruire tous les widgets de la page de quizz qui n'a pas changé depuis le dernier TP (voir figure 2.5).

```dart

void nextQuestion() {
  if (nextQuestionAvailable()) {
    QueryDocumentSnapshot<Map<String, dynamic>>? nouvelleQuestion =
        collectionQuestions?.docs
            .elementAt(compteurQuestionsCollectionFirebase);
    question = Question(
        questionText: nouvelleQuestion?['questionText'],
        isCorrect: nouvelleQuestion?['isCorrect'],
        imagePath: nouvelleQuestion?['imagePath']);
    compteurQuestionsCollectionFirebase++;
    vrai = Colors.grey;
    faux = Colors.grey;
    aRepondu = false;
    emit(QuizzLoaded());
  }
}
```

FIGURE 2.4 – Code de la fonction nextQuestion()

FIGURE 2.5 – Interface du quizz actuellement

2.4 Formulaire de création de questions

Pour pouvoir créer de nouvelles questions j'ai donc mis en place un formulaire avec deux champs pour du texte et une checkbox pour le booléen qui sert à dire si la réponse est 'vrai' ou 'faux, voir figure 2.6.

Lorsque l'utilisateur a rempli ses champs et qu'il envoie sa question, on lui indique qu'elle a bien été ajoutée avec un snackbar et on le redirige vers l'accueil de l'application. Côté back, on lance la méthode addQuestion qui prend en argument les trois valeurs et s'occupe d'ajouter à la collection hebergé par firebase, cette nouvelle question.

J'aurai voulu pouvoir donner l'opportunité à l'utilisateur de créer de nouvelles collections de questions mais je n'ai pas réussi à récupérer la totalité de mes collections depuis firebase. Sûrement que les choix de définition de mes collections et documents n'étaient pas les bons car aucune méthode ne permet de récupérer toutes les collections. J'ai donc écrit en dur mes deux collections et donné seulement la possibilité de créer des questions dans une des deux collections déjà existantes.

FIGURE 2.6 – Formulaire pour ajouter une question