**Université de Montpellier**
Faculté des science
Département Informatique


**Master 2 : Intelligence artificielle et science de données**

**HAI912I - Développement mobile avancé, Iot et embarqué**

---

# TP2 Flutter

---

**Réalisé par :**

- Goussem Ayoub

Année universitaire : 2025/2026

# 1 Introduction

## 1.1 Contexte du projet

Ce rapport présente mon travail sur le TP3 portant sur l'intégration de Firebase avec Flutter, réalisé dans le cadre du module HAI914I – Développement Mobile. L'objectif était de concevoir une application de quiz multiplateforme en mettant en pratique plusieurs services de l'écosystème Firebase.

L'application développée est un système de quiz interactif permettant aux utilisateurs de tester leurs connaissances sur différentes thématiques.

## 1.2 Objectifs du TP

Le TP3 s'articule autour de quatre objectifs principaux :

1. Maîtriser le stockage de données structurées sur Cloud Firestore
2. Implémenter un système d'authentification complet
3. Utiliser Firebase Storage pour les médias
4. Intégrer Firebase Analytics (optionnel)

# 2 Architecture du projet

## 2.1 Organisation en packages

Pour structurer l'application, j'ai adopté une architecture en couches qui sépare clairement la présentation, la logique métier et l'accès aux données. Cette organisation facilite la maintenance, les tests et l'évolution du projet :

```
lib/
├── business_logic/         # La logique métier
│   └── providers/          # Gestion d'état avec Provider
│       ├── auth_provider.dart
│       └── quiz_provider.dart
│
├── data/                   # Tout ce qui concerne les données
│   ├── models/             # Mes modèles de données
│   │   ├── question_model.dart
│   │   ├── score_model.dart
│   │   └── user_model.dart
│   ├── repositories/       # Accès aux données
│   │   ├── quiz_repository.dart
│   │   └── score_repository.dart
│   └── services/           # Services Firebase
│       ├── admin_service.dart
│       ├── analytics_service.dart
│       ├── audio_service.dart
│       ├── auth_service.dart
│       └── storage_service.dart
│
├── presentation/           # L'interface utilisateur
│   ├── screens/            # Les 11 écrans de l'app
│   └── widgets/            # Composants réutilisables
│
└── config/                 # Configuration
    └── theme.dart
```

Cette structure m'a permis de garder un code plus lisible, de réutiliser facilement les composants et de séparer proprement l'UI de la logique métier, ce qui est particulièrement important pour un projet amené à évoluer.

### 2.2 Structure des données

La base Firestore est organisée en quatre collections principales :
— `"questions"` : stockage des questions du quiz.
— `"users"` : profils utilisateurs.
— `"scores"` : historique des résultats.
— `"themes"` : catégories et métadonnées des thématiques.
Cette structure offre une bonne scalabilité tout en permettant des requêtes efficaces grâce aux capacités d'indexation de Firestore.

## 3 Question 1 : Extension et Firestore

Le schéma de la collection `"questions"` me permet de décrire chaque question de quiz avec son énoncé, ses options, la bonne réponse, le thème et quelques métadonnées comme

le créateur et la date de création.

Pour la collection `"users"`, j'ai choisi de regrouper les informations essentielles au profil et à la personnalisation : identifiant Firebase Auth, email, nom d'affichage, rôle administrateur, avatar et thématique préférée pour le mode SHOOT.

La collection `"scores"` stocke l'historique des résultats en liant chaque score à un utilisateur, un thème et un horodatage précis, ce qui me permet ensuite de faire des requêtes pour analyser la progression ou la performance par thématique. L'utilisation d'un champ `Timestamp` facilite aussi les tris chronologiques et les futures analyses statistiques sur les données de quiz.

## 3.1 Peuplement de la base de données

Pour tester l'application dans des conditions réalistes, j'ai peuplé Firestore avec plusieurs thématiques (Fruits, Pays, Animaux, etc.).

Chaque thématique contient cinq questions avec image avec une difficulté variable. Le peuplement s'est fait via une interface d'administration (`add_question_screen.dart`) permettant l'upload manuel des questions, la validation des données avant insertion et un enrichissement progressif avec des retours recueillis (tests informels, notamment via Discord).

## 3.2 Écrans d'ajout et de gestion

Les écrans d'administration que j'ai développés me permettent de gérer entièrement le contenu du quiz directement depuis l'application. `add_question_screen.dart` offre un formulaire complet avec validation, upload d'images vers Firebase Storage et sélection de la thématique pour créer de nouvelles questions avec leurs options et métadonnées. `manage_questions_screen.dart` sert à parcourir, filtrer, modifier et supprimer les questions, tout en affichant quelques statistiques simples par thème, tandis que `manage_themes_screen.dart` me donne la main sur la création, l'édition et la suppression des thématiques, ainsi que sur leurs images et descriptions.

# 4 Question 2 : Authentification

## 4.1 Création d'utilisateurs et écrans

Deux écrans principaux gèrent l'authentification :

`sign_up_screen.dart` – **Inscription**

- Champs email, nom d'affichage, mot de passe, confirmation.
- Validation du format d'email (regex).
- Vérification de la longueur minimale et de la confirmation du mot de passe.
- Gestion des erreurs retournées par Firebase (email déjà utilisé, etc.).

`sign_in_screen.dart` – **Connexion**

— Champs email et mot de passe.
— Option « Se souvenir de moi ».
— Lien « Mot de passe oublié ».
— Redirection vers l'inscription si besoin.

### 4.2 Gestion de l'authentification Firebase

Pour l'authentification, j'ai centralisé toute la logique dans `auth_service.dart`, qui encapsule les opérations principales de Firebase Auth comme l'inscription, la connexion, la déconnexion, la réinitialisation de mot de passe et la récupération de l'utilisateur courant.

La gestion d'état associée est déléguée à `auth_provider.dart`, qui écoute en continu le `Stream<User?>`, charge le profil Firestore correspondant et garde le statut administrateur synchronisé tout en assurant la persistance de session. Enfin, j'ai complété ce dispositif côté serveur avec des Firestore Security Rules qui restreignent l'accès aux documents `users` et `questions` en fonction de l'identité de l'utilisateur connecté et de son rôle, ce qui permet de sécuriser les opérations sensibles comme la modification du contenu du quiz.

## 5 Question 3 : Firebase Storage

### 5.1 Avatars utilisateurs

La gestion des avatars utilisateurs est regroupée dans `storage_service.dart`, où j'ai implémenté une méthode dédiée à l'upload de l'image vers Firebase Storage et à la récupération de son URL de téléchargement.

Côté flux, l'utilisateur choisit d'abord un fichier via `file_picker`, l'image est ensuite compressée avec le package `image`, puis envoyée sur un chemin du type `users/$userId/avatar.jpg` avant que l'URL publique ne soit enregistrée dans le document Firestore de l'utilisateur et affichée dans l'interface. Pour garder une bonne expérience utilisateur, j'ai ajouté quelques optimisations comme la limitation de la taille des fichiers, l'usage du cache local et une gestion explicite des erreurs réseau afin d'éviter que l'app ne se bloque en cas de problème de connexion.

### 5.2 Sons de victoire et défaite

Les sons sont gérés dans `audio_service.dart`. Deux fichiers sont utilisés :
— `sounds/victory.mp3` : joué si le score est supérieur ou égal à 50 %.
— `sounds/defeat.mp3` : joué sinon.

Les sons sont pré-chargés au démarrage, puis joués automatiquement dans `summary_screen.dart`. Le service gère aussi le contrôle du volume, l'arrêt automatique et les permissions nécessaires.

# 6 Question 4 : Analytics

## 6.1 Traçage des scores

L'événement personnalisé `quiz_completed` enregistre, pour chaque partie, le score, le nombre de questions, le thème et le pourcentage de bonnes réponses, ce qui enrichit considérablement les données *Analytics*. À partir de ces paramètres, il est ensuite possible d'analyser la performance moyenne par thématique, le taux de réussite global ainsi que la progression des utilisateurs au fil du temps.

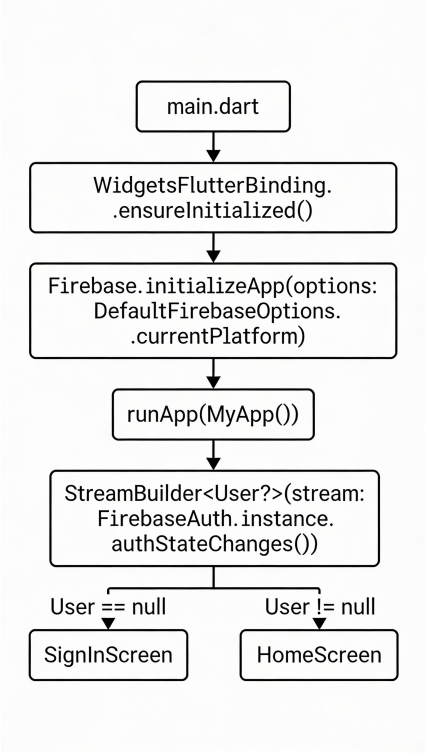
## 6.2 Mode « SHOOT » et thématique préférée

Cette propriété utilisateur `preferred_theme` me permet d'enregistrer, côté Analytics, la thématique favorite de chaque utilisateur dans son profil. Elle est ensuite exploitée par le mode SHOOT : à partir du bouton « Mode SHOOT » sur l'écran d'accueil, l'application récupère automatiquement ce thème et lance directement un quiz de 5 questions dessus.

# 7 Workflow de l'application

## 7.1 Démarrage et authentification

Le flux d'initialisation est le suivant :




La session est gérée automatiquement par Firebase Auth (persistance, reconnexion au redémarrage), avec possibilité de déconnexion manuelle.

## 7.2 Navigation principale

`HomeScreen` propose quatre parcours principaux :

1. Jouer au Quiz : `HomeScreen → ThemeSelectionScreen → QuizScreen → SummaryScreen`.
2. Mode Rapide (SHOOT) : `HomeScreen → QuizScreen (thème préféré) → SummaryScreen`.
3. Profil Utilisateur : `HomeScreen → UserProfileScreen` (modification avatar, nom, thème préféré).
4. Panneau Admin (si `isAdmin == true`) : `HomeScreen → AdminDashboardScreen` avec :
   — Statistiques générales.
   — `ManageQuestionsScreen`.
   — `AddQuestionScreen`.
   — `ManageThemesScreen`.

## 7.3 Gestion des données

L'architecture peut se résumer ainsi :


# 8 Problèmes rencontrés et solutions

## 8.1 Chargement lent des images

Au début, les images hébergées sur Firebase Storage mettaient plusieurs secondes à s'afficher pour chaque question, ce qui nuisait clairement à l'expérience utilisateur. Pour limiter cet effet, j'ai mis en place un widget de pré-chargement (`image_precaching_indicator.dart`) qui tente de charger les images en amont. Malgré ces optimisations (compression, cache, etc.), le chargement reste encore perceptible et prend en pratique autour d'une seconde pour certaines images.

## 8.2 Défis techniques divers

**Fonctionnalités secondaires instables** Quelques fonctionnalités mineures ont été impactées par des refactorings tardifs et n'ont pas pu être entièrement corrigées par manque de temps. Elles n'affectent pas le cœur des exigences du TP3 (authentification, gestion Firestore, Storage, Analytics, interface admin et quiz restent pleinement fonctionnels).

# 9 Conclusion

Ce TP m'a permis, de développer une application de quiz complète en intégrant les principaux services Firebase (Authentication, Firestore, Storage, Analytics) dans une architecture en couches propre et sécurisée. L'app offre une expérience utilisateur solide grâce à des fonctionnalités avancées comme le panneau d'administration et le mode « SHOOT » basé sur la thématique préférée, malgré quelques bugs mineurs sur des aspects secondaires.

# Annexes

## Liens du projet

**Répértoire GitHub** : `https://github.com/ayoubgoussem/tp3-flutter`
Code source complet et documentation `README` y sont disponibles.

**Serveur Discord (Tests et enrichissement)** : `https://discord.com/channels/1215070204611133510/1452297696332218519`
Serveur utilisé pour organiser des tests utilisateurs, recueillir du feedback et enrichir de manière collaborative la base de données Firebase.

**Application hébergée (Firebase Hosting)** : `https://tp3-flutter-840e2.web.app/`
Version déployée de l'application, accessible en ligne pour les démonstrations et les tests, pour que les testeurs puissent y accéder très facilement via une simple URL, sans avoir à installer quoi que ce soit. **Images et droits d'utilisation**

Pour les illustrations (fruits, drapeaux, animaux, etc.), j'ai utilisé des icônes provenant de Flaticon (`https://www.flaticon.com/`) sous la Flaticon Free License, adaptée à un usage non commercial et éducatif. L'attribution « Icons made by Freepik from `https://www.flaticon.com` » est indiquée dans le code source et la documentation.