Université de Corse - Master 2 DFS

Cours : Flutter & Firebase - Projet final

Enseignant : Edouard Chevenslove

Introduction

Ce document présente une description détaillée du projet d'application **Chat App**. Il constitue un support de cours magistral permettant aux étudiants de comprendre la structure, les fonctionnalités et les principes architecturaux du développement Flutter moderne avec **Firebase** et **Provider**.

1 Projet Flutter : *Chat App* - Explication

1.1 Objectif du projet

Le projet **ChatApp** consiste à développer une application de messagerie instantanée en Flutter. L'objectif est de comprendre comment concevoir une application mobile complète intégrant :

- une **authentification sécurisée** avec Firebase Authentication,
- une **gestion de base de données en temps réel** via Cloud Firestore,
- un **stockage distant** d'images via Firebase Storage,
- une **architecture logicielle claire (MVVM)** séparant la logique métier, la vue et les modèles.

Les étudiants doivent acquérir la capacité de :

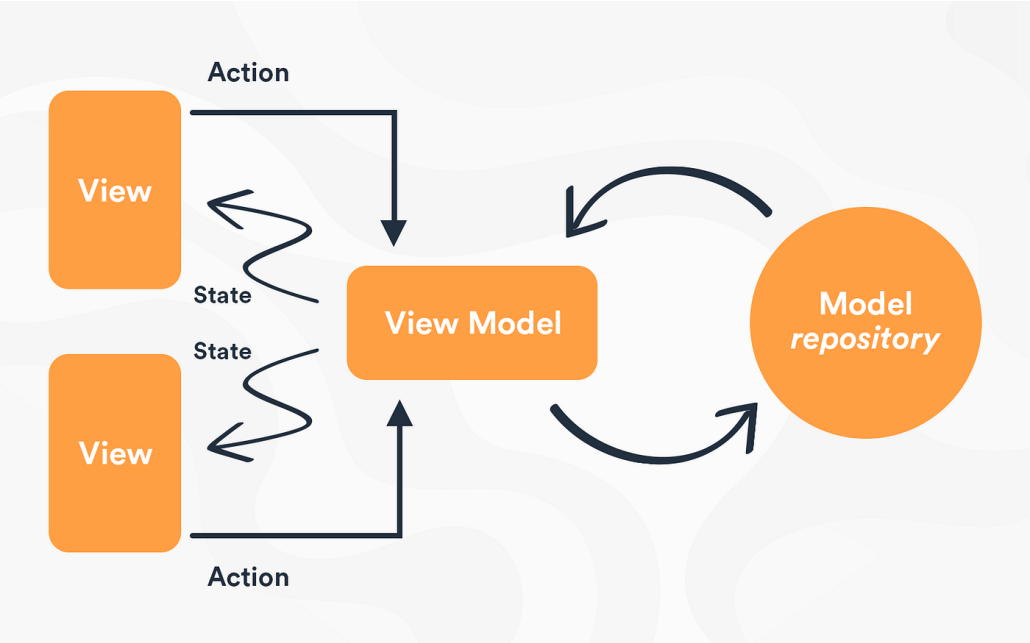
1. Comprendre le cycle de vie d'une application Flutter.
2. Intégrer et configurer un projet Firebase.
3. Structurer leur code selon une architecture maintenable.
4. Manipuler les composants visuels de Flutter.
5. Échanger des données en temps réel.

1.2 Architecture générale du projet

L'architecture adoptée est celle du modèle **MVVM (Model-View-ViewModel)** :

- **Model :** représente les entités (utilisateur, chat, message).

— **View :** regroupe les interfaces graphiques (pages, widgets).
— **ViewModel :** contient la logique métier, gère les interactions avec Firebase et notifie la vue via le package `Provider`.



(Le schéma ci-dessus illustre le flux d'information entre les couches MVVM.)

1.3 Structure du projet

L'application est organisée comme suit :

— `/model` : contient les classes **Chat**, **ChatUser**, **Message**.
— `/viewmodel` : contient la logique d'authentification et de chat (`AuthViewModel`, `ChatUserViewModel`, `ChatViewModel`).
— `/pages` : regroupe les écrans de l'application (connexion, inscription, page d'accueil, chat, profil, splash).
— `/widgets` : contient les composants réutilisables (`ChatListItem`, `MessageItem`, `LoadingScreen`, etc.).
— `constants.dart` : centralise les couleurs, tailles et textes.
— `main.dart` : point d'entrée de l'application.

1.4 Étapes du développement

1.4.1 Configuration de Firebase

1. Création d'un projet Firebase sur console.firebase.google.com.
2. Installation des outils :

```
flutter pub add firebase_core firebase_auth
  ↪ cloud_firestore firebase_storage
flutterfire configure
```

3. Génération automatique du fichier `firebase_options.dart`.

1.4.2 Structure de la base de données

La base de données comporte deux collections principales :

— `users` : contient les informations sur chaque utilisateur.

```
users/{userId} {
  id: "uid",
  displayName: "Nom complet",
  email: "utilisateur@email.com",
  bio: "Description personnelle",
  avatarUrl: "url de l'image"
}
```

— `chats` : contient les conversations entre utilisateurs.

```
chats/{chatId}/messages/{messageId} {
  from: "uid_expéditeur",
  to: "uid_destinataire",
  content: "Message texte",
  timestamp: "1687263000000"
}
```

1.4.3 Splash Screen

Objectif : afficher un écran d'accueil contenant le logo de l'application et un message de bienvenue à `ChatApp`.

Outils utilisés :

— `Image.asset` pour le logo.
— `Scaffold` et `Center` pour la mise en page.
— `Future.delayed` pour rediriger automatiquement vers la page suivante.

Fonctionnement :

1. Au démarrage, le Splash Screen s'affiche pendant 2 secondes.
2. Si l'utilisateur est connecté, il est redirigé vers la `HomePage`.
3. Sinon, il est redirigé vers la `LoginPage`.

1.4.4 Authentification (Login/Signup)

Objectif : permettre à un utilisateur de créer un compte ou de se connecter avec son e-mail et mot de passe.

Composants utilisés :

— `FirebaseAuth` pour la création et la connexion d'un utilisateur.
— `TextFormField` pour les champs e-mail et mot de passe.
— `SnackBar` pour afficher les erreurs.
— `Provider` pour stocker l'état d'authentification.

Fonctionnement : Lorsqu'un utilisateur s'inscrit, son profil est créé dans la collection Firestore :

```
users/{uid} {
  displayName: "...",
  bio: "",
  avatarUrl: "",
  id: uid
}
```

1.4.5 Page d'accueil (HomePage)

Objectif : afficher la liste des utilisateurs connectés.

Outils utilisés :

- `StreamBuilder<QuerySnapshot>` : écoute les changements temps réel dans Firestore.
- `ListView.builder` : affiche la liste des utilisateurs.
- `ChatListItem` : widget personnalisé représentant chaque utilisateur.

Fonctionnement :

1. L'application récupère tous les utilisateurs sauf celui connecté.
2. Un clic sur un utilisateur ouvre la page de chat correspondante.

1.4.6 Page de chat (ChatPage)

Objectif : afficher et envoyer des messages entre deux utilisateurs.

Composants utilisés :

- `StreamBuilder<List<Message»` pour afficher les messages en direct.
- `NewMessageSection` pour la zone de saisie et d'envoi.
- `GroupedListView` pour regrouper les messages par date.

Structure Firestore :

```
chats/{chatId}/messages/{messageId} {
  from: "uid_sender",
  to: "uid_receiver",
  content: "Salut !",
  timestamp: "1687263000000"
}
```

Principe de fonctionnement :

1. Si la conversation n'existe pas, elle est créée.
2. Chaque message est ajouté dans une sous-collection.
3. L'affichage est mis à jour automatiquement grâce aux streams Firebase.

1.4.7 Page de profil

Objectif : permettre à l'utilisateur de modifier son nom, sa bio et sa photo.

Composants utilisés :

- `ImagePicker` pour choisir une image dans la galerie.
- `FirebaseStorage` pour envoyer l'image sur le cloud.
- `TextField` pour la bio et le nom.

1.4.8 Déconnexion

- Appel de la méthode `FirebaseAuth.instance.signOut()`.
- Suppression de la session dans `SharedPreferences`.
- Redirection vers la page de connexion.

1.5 Pour aller plus loin - Fonctionnalités additionnelles

Les fonctionnalités suivantes ne sont pas obligatoires mais sont fortement conseillées pour enrichir le projet final :

- **Envoi de multimédia :** Possibilité d'envoyer des images, fichiers audio ou documents, avec affichage plein écran.
- **Suppression de messages :** Ajout de la possibilité de supprimer un message (localement ou globalement).
- **Visualisation du profil d'autrui :** Consultation du profil d'un autre utilisateur depuis la page de conversation.
- **Notifications push :** Utilisation de `Firebase Cloud Messaging (FCM)` pour notifier les nouveaux messages.
- **Lazy loading :** Chargement paresseux (progressif) des conversations et messages pour de meilleures performances.
- **Émojis et réactions :** Intégration d'un clavier d'émojis ou de réactions rapides.
- **Authentification étendue :** Connexion avec Google, Facebook ou Apple ID.
- **Appels audio/vidéo et live :** Intégration d'un système de visioconférence via WebRTC ou Agora SDK.

1.6 Aspects pédagogiques du projet

Le projet est découpé en **sections progressives** :

1. **Section 1 - Authentification** : configuration Firebase, login/signup.
2. **Section 2 - Home & Modèles** : affichage de la liste des utilisateurs.
3. **Section 3 - Chat** : envoi et réception de messages.
4. **Section 4 - Profil** : modification du profil et upload de photo.

Le projet comprend des exercices de :

- Validation de champs et gestion des erreurs.
- Filtrage des utilisateurs.
- Tri des messages par date.
- Persistance des préférences locales.

1.7 Livrables attendus

— Code source fonctionnel sur GitHub.
— Documentation synthétique (PDF et README).

Critères d'évaluation :
— Architecture correcte (30%)
— Fonctionnalité Firebase (30%)
— Interface et ergonomie (25%)
— Clarté du code et commentaires (15%)

2 Composants Flutter fondamentaux

Widgets de base

— **StatelessWidget** - Widget immuable. Utilisé pour des interfaces statiques, comme `LoadingScreen`.
— **StatefulWidget** - Widget dynamique contenant un `State` modifiable pour réagir aux changements, ex : `ChatPage`.
— **BuildContext** - Référence à l'arbre des widgets. Sert à accéder aux thèmes, navigateurs ou Providers.

Structure et mise en page

— **Scaffold** - Structure de base d'un écran (`AppBar`, `Body`, `FloatingActionButton`).
— **AppBar** - Barre supérieure avec titre, actions, icônes.
— **Column / Row** - Organisation verticale ou horizontale de widgets enfants.
— **Container** - Boîte flexible permettant marges, couleurs, décorations.
— **CircleAvatar** - Affiche un avatar ou une image circulaire (photo de profil).

Affichage de données

— **ListView / ListTile** - Liste scrollable d'éléments. Utilisé dans la `HomePage`.
— **StreamBuilder** - Construit l'interface à partir d'un flux de données en temps réel (`Firestore`).
— **FutureBuilder** - Construit l'interface après une opération asynchrone (`Future`).

Saisie et interaction

— **TextFormField** - Champ de saisie avec validation intégrée.
— **Navigator** - Navigation entre pages (`push`, `pop`, `pushReplacement`).
— **SnackBar** - Message temporaire de notification en bas de l'écran.

Gestion d'état et persistance

— **Provider / ChangeNotifier** - Gestion d'état réactive et globale via le pattern MVVM.
— **SharedPreferences** - Stockage local clé-valeur pour conserver les informations utilisateur.

Services Firebase

— **FirebaseAuth** - Authentification via email/mot de passe.
— **FirebaseFirestore** - Base de données NoSQL pour stocker utilisateurs, chats et messages.
— **FirebaseStorage** - Stockage d'images (avatars).

Autres Packages et Utilitaires

— `grouped_list` : affichage des messages groupés par date.
— `flutter_launcher_icons` : personnalisation de l'icône de l'application.
— `intl` : formatage des dates.
— `image_picker` : sélection d'images depuis la galerie.

3 Synthèse des composants et de leur fonctionnement

Composant	Type	Utilisation
MaterialApp	Structure	Racine de l'application.
Scaffold	Layout	Structure d'un écran complet.
AppBar	Layout	Barre de titre avec actions.
StreamBuilder	Data	Affichage temps réel depuis Firestore.
FutureBuilder	Data	Affichage après chargement d'un Future.
ListView	Layout	Affichage dynamique de listes.
Provider	Architecture	Injection de dépendances (ViewModels).
FirebaseAuth	Service	Authentification utilisateur.
Firestore	Service	Gestion de la base de données cloud.
Storage	Service	Sauvegarde d'images utilisateur.

Conclusion

Le projet **Chat App** constitue un excellent exercice pour maîtriser les principes de Flutter moderne : gestion d'état avec Provider, intégration de Firebase, UI réactive et architecture

MVVM. La compréhension de ces composants est essentielle pour tout développement Flutter professionnel. Alors bon succès les champions(nes).

Université de Corse - Faculté des sciences et techniques - Master 2 DFS - Année 2025/2026