Rapport Technique du Projet Chaton

Presenté par :
MUNUNGA KAYINDA ELIEZER
BARREA MUKENGE ALEXANDRE
KALALA LUKUSA DANIEL

Lien important **:**

- **Google Drive**
- **Github**

1. Introduction

Le projet **Chaton** est une application mobile de messagerie instantanée développée sous Android. Son objectif principal est de permettre aux utilisateurs de communiquer en temps réel à travers des conversations privées (one-to-one), dans un environnement sécurisé et ergonomique.

L'application repose sur une architecture client–serveur basée sur les services **Firebase**, offrant une infrastructure backend robuste sans nécessiter la mise en place d'un serveur dédié.

2. Description Générale du Projet

2.1 Objectif

L'application vise à :

- Permettre l'inscription et l'authentification des utilisateurs.
- Offrir un système de messagerie instantanée en temps réel.
- Gérer les conversations individuelles.

2.2 Fonctionnement global

Le cycle fonctionnel de l'application est le suivant :

1. L'utilisateur s'inscrit ou se connecte.

2. Il accède à la liste de ses conversations.
3. Il peut démarrer une nouvelle discussion ou continuer une conversation existante.
4. Les messages sont envoyés et reçus en temps réel via Firestore.

3. Technologies et Outils Utilisés

3.1 Plateforme de développement

- **Langage** : Kotlin
- **IDE** : Android Studio
- **Système de build** : Gradle (Kotlin DSL)

3.2 Backend

L'application s'appuie entièrement sur **Firebase**, qui fournit :

- **Firebase Authentication**
  Gestion de l'authentification (email/password, Google).
- **Cloud Firestore**
  Base de données NoSQL temps réel pour stocker :
  - Utilisateurs
  - Conversations
  - Messages
- **Firebase Cloud Messaging**
  Envoi de notifications push.

3.3 Bibliothèques UI

- Material Design

- ConstraintLayout
- Coil (chargement d’images)
- Lottie (animations)
- CircleIndicator

3.4 Tests

- JUnit (tests unitaires)
- AndroidX Test & Espresso (tests instrumentés)

4. Structure du Projet

À la racine du projet :

Élément	Description
`.git`	Gestion de version avec Git
`.idea`	Configuration Android Studio
`app/`	Module principal Android
`gradle/`	Gradle Wrapper
`gradlew / gradlew.bat`	Scripts d’exécution Gradle
`build.gradle.kts`	Configuration globale du projet
`settings.gradle.kts`	Déclaration des modules
`gradle.properties`	Paramètres de build
`README.md`	Documentation
`design.json`	Fichier de conception UI

`.gitignore`	Fichiers ignorés par Git

5. Analyse du build.gradle.kts (Module App)

5.1 Configuration Android

- `namespace` : com.mecatrogenie.chaton
- `compileSdk` : 36
- `minSdk` : 24
- `targetSdk` : 36
- `versionCode` : 1
- `versionName` : 1.0

Cette configuration garantit une compatibilité avec Android 7.0 (API 24) et supérieur.

5.2 Dépendances principales

Firebase

- firebase-auth
- firebase-firestore
- firebase-messaging
- play-services-auth (Google Sign-In)

Interface utilisateur

- material
- constraintlayout

- coil
- lottie
- circleindicator

Tests

- junit
- androidx.test.ext:junit
- androidx.espresso:espresso-core

6. Architecture de l'Application

L'application suit une architecture proche du modèle **Model-View-Controller (MVC)**.

6.1 Modèle (Model)

Représente les données métier :

- User
- Chat
- Message

Ces entités sont stockées dans Firestore sous forme de documents.

6.2 Vue (View)

Composée des :

- Activités (Activities)
- Layouts XML

Responsable uniquement de l'affichage et des interactions utilisateur.

6.3 Contrôleur (Controller)

La logique applicative est directement implémentée dans les activités :

- Gestion des requêtes Firestore
- Traitement des événements utilisateur
- Mise à jour dynamique de l'interface

7. Analyse des Composants Principaux

7.1 MainActivity.kt

Point d'entrée principal pour un utilisateur authentifié.

Responsabilités :

- Vérification de session utilisateur.
- Chargement des conversations depuis Firestore.
- Gestion des messages non lus.
- Recherche de conversations.
- Navigation vers :
    - Nouvelle discussion
    - Paramètres
- Déconnexion sécurisée.

7.2 NewChatActivity.kt

Gestion de la création de nouvelles conversations.

Fonctionnalités :

- Chargement de tous les utilisateurs sauf l'utilisateur courant.
- Recherche dynamique d'utilisateurs.

- Vérification d’existence d’une conversation.
- Création ou ouverture d’une conversation.
- Redirection vers ChatActivity.

7.3 ChatActivity.kt

Cœur fonctionnel de l’application.

Fonctions principales :

- Écoute en temps réel des messages.
- Affichage dynamique via RecyclerView.
- Envoi de messages.
- Mise à jour du statut de lecture.
- Gestion automatique du clavier (ajustement UI).

8. Sécurité et Gestion des Données

- Authentification sécurisée via Firebase.
- Données stockées dans Firestore avec règles de sécurité.
- Isolation des conversations entre utilisateurs.
- Utilisation de tokens FCM pour notifications ciblées.

9. Limites Actuelles

- Logique fortement couplée aux activités.
- Gestion des erreurs réseau encore basique.
- Couverture de tests insuffisante.

- Architecture difficilement scalable à long terme.

10. Perspectives d’Amélioration

10.1 Migration vers MVVM

Adopter une architecture **Model-View-ViewModel** permettrait :

- Une meilleure séparation des responsabilités.
- Une gestion optimisée du cycle de vie.
- Une testabilité accrue.
- Une maintenance facilitée.

10.2 Amélioration UX

- Gestion avancée des erreurs réseau.
- Indicateur de saisie ("typing…").
- Accusés de réception (✓✓).
- Mode sombre.

10.3 Modernisation UI

Migration vers **Jetpack Compose** pour :

- Réduire la complexité XML.
- Améliorer la lisibilité du code.
- Accélérer le développement UI.

10.4 Extension fonctionnelle

- Discussions de groupe.

- Envoi d’images et fichiers.
- Messages vocaux.
- Chiffrement de bout en bout.

11. Conclusion

Le projet **Chaton** constitue une application de messagerie instantanée fonctionnelle, reposant sur une architecture solide basée sur Firebase.

L’application démontre :

- Une maîtrise des services backend cloud.
- Une structuration cohérente du projet Android.
- Une implémentation complète d’un système de chat temps réel.

Bien que l’architecture actuelle soit adaptée à un projet académique ou prototype fonctionnel, une évolution vers une architecture plus modulaire (MVVM), une amélioration de la robustesse et une extension fonctionnelle permettraient d’en faire une application scalable et prête pour un environnement de production.