ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Rapport Finale du projet

5e annee – Ingenierie Informatique et Reseaux

Sous le thème

Réalisation d'une Application E-commerce Flutter spécialisée en électronique

Réalisée par :

- MAJGHIROU Mohamed Riyad
- AZZAM Mohamed
- IBRAHIMI Aya
- BOUZAIDI TIALI Aya

Encadrer par :

- DEROUSSI Anass

ANNEE UNIVERSITAIRE: 2025-2026

Table des matières

Chapitre 1

Introduction

1.1 Contexte général

Le commerce électronique est devenu un canal majeur pour l'achat de produits high-tech (smartphones, ordinateurs, tablettes, accessoires, etc.). Les utilisateurs attendent des applications mobiles qu'elles soient rapides, ergonomiques, sécurisées et disponibles sur plusieurs plateformes (Android, iOS, Web).

Dans ce contexte, le projet **Electronics Store** consiste à développer une **application mobile e-commerce spécialisée en électronique** basée sur le framework **Flutter** et le backend **Firebase**. L'objectif est de proposer une solution complète allant de la navigation dans le catalogue jusqu'à la gestion des commandes, en passant par l'authentification, le panier et un panel d'administration.

1.2 Problématique

Les plateformes e-commerce existantes sont souvent générales et ne répondent pas toujours aux besoins spécifiques du segment électronique :

- Complexité des fiches produits (caractéristiques techniques détaillées).
- Besoin de filtres avancés (marque, prix, catégorie, popularité).
- Nécessité d'une architecture robuste et maintenable pour faire évoluer l'application.
- Gestion fine des rôles (client, administrateur) et des données (produits, utilisateurs, commandes).

La problématique centrale peut être formulée comme suit :

> « Comment concevoir et développer une application e-commerce mobile spécialisée en électronique, reposant sur une architecture propre, évolutive et sécurisée, tout en offrant une expérience utilisateur moderne et fluide ? »

1.3 Objectifs du projet

1.3.1 Objectif général

Concevoir et implémenter une application e-commerce mobile spécialisée dans les produits électroniques, utilisant **Flutter** pour l'interface utilisateur et **Firebase** comme backend (Auth, Firestore, Storage), en suivant les principes de la **Clean Architecture**.

1.3.2 Objectifs spécifiques

- Créer une application e-commerce fonctionnelle et complète.
- Mettre en place une **architecture propre et maintenable** (Clean Architecture).
- Gérer l'**authentification** via Firebase Authentication (inscription, connexion, gestion de session).
- Utiliser **Cloud Firestore** pour persister les données (produits, catégories, commandes, favoris).
- Stocker les images produits et photos de profil dans **Firebase Storage**.
- Intégrer un **panel d'administration** permettant de gérer les produits, catégories, utilisateurs et commandes.
- Optimiser l'**expérience utilisateur** (animations, cache d'images, navigation fluide).

1.4 Méthodologie

La démarche suivie est structurée en plusieurs étapes :

1. **Analyse des besoins** et définition des exigences fonctionnelles et non fonctionnelles.
2. **Conception** de l'architecture (Clean Architecture), de la navigation et du modèle de données.
3. **Implémentation** de l'application avec Flutter, Dart et les services Firebase.
4. **Tests** unitaires et fonctionnels, validation des scénarios clés.
5. **Déploiement** et préparation à la mise en production (configuration Firebase, build release).

Chapitre 2

Analyse des besoins

2.1 Description générale du système

Electronics Store est une application mobile permettant :

- À un **utilisateur (client)** :
  - de créer un compte, se connecter et gérer son profil ;
  - de parcourir le **catalogue de produits électroniques** organisé en catégories ;
  - de rechercher et filtrer les produits selon différents critères ;
  - d'ajouter des produits au **panier**, de passer des commandes ;
  - de gérer ses **produits favoris** ;
  - de consulter l'historique et le statut de ses commandes.
- À un **administrateur** :
  - de gérer les **produits** (CRUD) ;
  - de gérer les **catégories** ;
  - de consulter et gérer les **commandes** ;
  - de gérer les **utilisateurs** et leurs rôles.

2.2 Acteurs

- **Client** : utilisateur final de l'application, achète des produits.
- **Administrateur** : utilisateur avec des privilèges avancés de gestion.
- **Système Firebase** : Auth, Firestore, Storage, assurant l'authentification et la persistance des données.

2.3 Besoins fonctionnels

2.3.1 Fonctionnalités côté client

1. **Authentification**

   - Inscription avec validation des champs.
   - Connexion et déconnexion sécurisée.
   - Persistance de la session.
   - Gestion du profil (nom, email, photo).
2. **Catalogue produits**
   - Affichage par catégories (smartphones, ordinateurs, tablettes, audio, photo/vidéo, gaming, accessoires, montres connectées).
   - Recherche par mot-clé.
   - Filtrage par prix, popularité, note, marque.
   - Détails complets (description, caractéristiques techniques, images).
3. **Panier et commandes**
   - Ajout / suppression de produits dans le panier.
   - Modification des quantités.
   - Calcul automatique du total.
   - Validation de la commande.
   - Suivi des commandes (En cours, Livrée, Annulée).
4. **Favoris**
   - Ajout / retrait de produits favoris.
   - Liste synchronisée côté serveur.

2.3.2 Fonctionnalités côté administrateur

1. Authentification administrateur.
2. **Dashboard** de synthèse (nombre d'utilisateurs, produits, commandes).
3. **Gestion des produits** :
   - Gestion des stocks, prix, images.
4. **Gestion des catégories**.
5. **Gestion des utilisateurs**.
6. **Gestion des commandes**.

2.4 Besoins non fonctionnels

- **Performance** : temps de chargement réduit, utilisation de cache d'images.
- **Sécurité** : gestion des rôles, accès restreint aux données, règles de sécurité Firestore/Storage.
- **Portabilité** : application multi-plateforme (Android, iOS, Web).
- **Évolutivité** : architecture modulable (Clean Architecture).
- **Maintenabilité** : code structuré, séparation des responsabilités, utilisation de packages standards.

Chapitre 3

Conception du système

3.1 Architecture globale

Le projet suit les principes de la **Clean Architecture** avec une séparation claire des couches :

```
lib/
 core/                    # Code partagé (constantes, erreurs, utils, usecases)
 data/                    # Couche de données
    datasources/          # Firebase / API distantes
    models/               # Modèles de données (DTO)
    repositories/         # Implémentations des repositories
 domain/                  # Couche métier
    entities/             # Entités métier
    repositories/         # Interfaces des repositories
    usecases/             # Cas d'utilisation
 presentation/            # Couche présentation
     pages/               # Écrans Flutter
     providers/           # Gestion d'état (Provider)
     themes/              # Thèmes et styles
     widgets/             # Widgets réutilisables
```

Les dépendances suivent le principe d'inversion : *presentation* et *data* dépendent de *domain*, mais *domain* ne dépend pas de l'interface utilisateur ni des frameworks.

3.2 Modèle de données conceptuel

Les entités principales :

- **Utilisateur**
  - id, nom, email, rôle (client/admin), photoProfil, dateInscription.
- **Catégorie**
  - id, nom, description, icône.
- **Produit**
  - id, nom, description, prix, catégorieId, marque, images, stock, noteMoyenne.
- **Commande**
  - id, utilisateurId, dateCommande, montantTotal, statut, liste d'articles.
- **Article de commande**
  - produitId, quantité, prixUnitaire.
- **Favori**
  - utilisateurId, produitId.

3.3 Organisation des écrans Flutter

Quelques écrans principaux :

- **Écrans d'authentification** : inscription, connexion, réinitialisation de mot de passe.
- **Écran d'accueil** : catégories, produits recommandés, offres.
- **Écran liste produits** : filtrage, tri, recherche.
- **Écran détail produit** : images, caractéristiques, avis, bouton ajouter au panier / favoris.
- **Écran panier** : liste des articles, total, validation.
- **Écran commandes** : historique, détails.
- **Profil utilisateur** : informations, photo de profil, paramètre.
- **Dashboard admin** : gestion produits, catégories, utilisateurs, commandes.

3.4 Navigation et gestion d'état

- **GoRouter** est utilisé pour une navigation déclarative, avec gestion des routes nommées et des redirections selon l'état d'authentification.
- **Provider** est utilisé pour la gestion d'état (authentification, panier, favoris, produits).
- Les use cases du domaine sont injectés via **GetIt** (Service Locator) pour respecter l'inversion de dépendance.

Chapitre 4

Technologies et outils

4.1 Technologies principales

- **Flutter 3.x** : framework UI multiplateforme.
- **Dart 3.x** : langage de programmation orienté objet et fonctionnel.
- **Firebase** :
  - Firebase Auth pour l'authentification.
  - Cloud Firestore pour la base de données NoSQL.
  - Firebase Storage pour le stockage des images.
- **SQLite** : base locale pour la persistance hors-ligne (optionnel selon modules).

4.2 Packages Flutter utilisés

Exemples de packages utilisés dans le projet :

- **firebase_core**, **firebase_auth**, **cloud_firestore**, **firebase_storage**.
- **provider** : gestion d'état.
- **go_router** : navigation.
- **dio**, **pretty_dio_logger** : appels HTTP et logs.
- **cached_network_image** : cache d'images.
- **shimmer**, **flutter_staggered_animations** : effets visuels.
- **get_it**, **dartz**, **equatable** : DI et programmation fonctionnelle.

4.3 Outils de développement

- **Android Studio** / **VS Code** pour le développement Flutter.
- **Git & GitHub** pour le versioning du code source.
- **Firebase Console** pour la gestion du backend.

Chapitre 5

Persistance et sécurité

5.1 Modélisation Firestore

Exemple de collections :

— `users` : documents représentant les utilisateurs.
— `categories` : catégories de produits.
— `products` : produits avec références aux catégories.
— `orders` : commandes passées par les utilisateurs.
— `favorites` : liens utilisateur-produit pour les favoris.

5.2 Règles de sécurité

Les règles Firestore et Storage sont configurées pour :

— Autoriser la lecture / écriture d'un utilisateur uniquement sur ses propres données sensibles.
— Restreindre les opérations d'administration aux comptes avec un rôle `admin`.
— Protéger l'accès aux images stockées dans Firebase Storage.

5.3 Gestion des rôles

Les rôles sont stockés côté Firestore dans le document utilisateur. L'interface lit ce rôle pour :

— Afficher ou non le menu **Administration**.
— Restreindre les routes d'accès via GoRouter.

Chapitre 6

Tests et validation

6.1 Types de tests

— **Tests unitaires** sur les use cases et repositories.
— **Tests d'intégration** sur la communication avec Firebase.
— **Tests fonctionnels** sur les scénarios utilisateurs clés (inscription, ajout au panier, passage de commande).

6.2 Scénarios de test principaux

— Inscription puis connexion d'un nouvel utilisateur.
— Ajout de plusieurs produits au panier, modification des quantités, validation de commande.
— Ajout / suppression de favoris et vérification de la persistance.
— Création / modification / suppression d'un produit par un administrateur.
— Changement du statut d'une commande (En cours →Livrée).

6.3 Résultats observés

— Les flux principaux fonctionnent correctement.
— Les performances sont jugées satisfaisantes, notamment grâce au cache d'images.
— L'architecture modulaire facilite l'ajout de nouvelles fonctionnalités.

Chapitre 7

Déploiement et utilisation

7.1 Prérequis

— Flutter SDK 3.x.
— Dart SDK 3.x.
— Android Studio ou VS Code.
— Compte Firebase configuré.

7.2 Installation du projet

1. Cloner le dépôt :

   ```
   git clone https://github.com/riyad4589/Flutter_Ecommerce_Electronics_Store.git
   cd Flutter_Ecommerce_Electronics_Store
   ```

2. Installer les dépendances :

   ```
   flutter pub get
   ```

3. Configurer Firebase (via `flutterfire configure`).
4. Lancer l'application en mode debug ou release :

   ```
   flutter run
   flutter run --release
   ```

Chapitre 8

Conclusion et perspectives

8.1 Bilan du projet

Le projet **Electronics Store** a permis de :

- Concevoir une application e-commerce complète spécialisée en électronique.
- Mettre en œuvre une architecture **Clean Architecture** claire et maintenable.
- Intégrer efficacement les services Firebase (Auth, Firestore, Storage).
- Proposer une expérience utilisateur moderne, avec des animations et un cache d'images optimisé.

Ce projet constitue une base solide pour des évolutions futures et démontre la pertinence de Flutter pour le développement d'applications mobiles multi-plateformes professionnelles.

8.2 Perspectives d'évolution

Plusieurs améliorations peuvent être envisagées :

- Intégration d'un **module de paiement en ligne** (Stripe, PayPal, etc.).
- Ajout de **notifications push** pour informer l'utilisateur des statuts de commande et des promotions.
- Mise en place d'un **back-office Web** dédié aux administrateurs.
- Ajout de fonctionnalités de **recommandation intelligente** (produits similaires, cross-selling).
- Internationalisation complète (multi-langues, multi-devises).