

INTÉGRATION EN COMPÉTENCES  
UNIVERSITÉ DE MONTPELLIER

DÉPARTEMENT D'INFORMATIQUE  
PROGRAMMATION MOBILE  
RAPPORT DE PROJET 1 MOBILE

---

# Rapport du Projet Développement d'une Application Mobile E-commerce Programmation mobile et objets connectés

---

*Étudiants :*

Jancy Fridelin KOUD BANGA,  
Nada BLOUNDI

*Enseignants*  
Abdelhak-Djamel SERIAI

## Table des matières

<b>1</b>	<b>1. Introduction</b>	<b>2</b>
1.1	1.1 Contexte et objectifs . . . . .	2
1.2	1.2 Technologies utilisées . . . . .	2
<b>2</b>	<b>Architecture du projet</b>	<b>4</b>
2.1	Structure globale . . . . .	4
2.2	Structure détaillée du projet . . . . .	4
2.3	Ressources et layouts . . . . .	4
2.4	Méthodes clés . . . . .	5
2.5	Gestion des avertissements . . . . .	6
2.6	Intégration Firebase . . . . .	6
<b>3</b>	<b>Implémentation technique</b>	<b>7</b>
3.1	Navigation . . . . .	7
3.2	Chargement des données Firebase . . . . .	7
3.3	Gestion des RecyclerView . . . . .	7
<b>4</b>	<b>Résultats et captures d'écran</b>	<b>8</b>
4.1	Interface utilisateur . . . . .	8
4.2	Tests et validation . . . . .	10
<b>5</b>	<b>Perspectives et améliorations</b>	<b>10</b>
<b>6</b>	<b>Conclusion</b>	<b>11</b>

## Table des figures

1	Langage de développement - Java. . . . .	2
2	Environnement de développement - Android Studio. . . . .	2
3	Base de données - Firebase Realtime Database. . . . .	3
4	Bibliothèque tierce - Glide. . . . .	3
5	Bibliothèque tierce - RecyclerView. . . . .	3
6	Bibliothèque tierce - Firebase Authentication. . . . .	3
7	Exemple de structure Firebase. . . . .	6
8	Écran d'accueil (IntroActivity) . . . . .	8
9	Écran principal (MainActivity) . . . . .	9
10	Catalogue de produits . . . . .	9
11	Détail d'un produit . . . . .	10
12	Panier (CartActivity) . . . . .	10
13	Diagramme de la structure Firebase. . . . .	11
14	Diagramme de classes de l'architecture des activités. . . . .	12
15	Diagramme de cas d'utilisation. . . . .	12

# 1 1. Introduction

## 1.1 1.1 Contexte et objectifs

À une époque où le commerce en ligne connaît une croissance fulgurante, il devient impératif de proposer des solutions mobiles performantes et accessibles. Ce projet vise à développer une application e-commerce mobile intuitive que j'ai intitulé **CongoStore**, offrant une expérience utilisateur fluide et immersive. Cette plateforme ambitionne de permettre aux utilisateurs de découvrir des produits variés, d'organiser leurs achats et de vivre une expérience d'achat agréable.

L'objectif principal est de combiner simplicité, modernité et performance pour répondre aux attentes croissantes des consommateurs en ligne.

## 1.2 1.2 Technologies utilisées

Pour atteindre ces objectifs, une combinaison d'outils et de technologies robustes a été mise en œuvre :

- Langage de développement : Java.



FIGURE 1 – Langage de développement - Java.

- Environnement de développement : Android Studio.

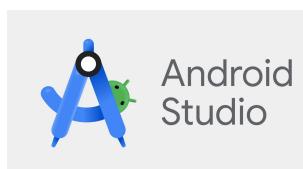


FIGURE 2 – Environnement de développement - Android Studio.

- Base de données : Firebase Realtime Database, pour une synchronisation en temps réel des données.



FIGURE 3 – Base de données - Firebase Realtime Database.

— **Bibliothèques tierces :**

- **Glide** pour le chargement rapide et efficace des images.



FIGURE 4 – Bibliothèque tierce - Glide.

- **RecyclerView** pour une gestion optimisée des listes.



FIGURE 5 – Bibliothèque tierce - RecyclerView.

- **Firebase Authentication** pour la gestion des utilisateurs.



FIGURE 6 – Bibliothèque tierce - Firebase Authentication.

## 2 Architecture du projet

### 2.1 Structure globale

L'application est structurée autour de trois activités principales, chacune jouant un rôle clé dans le parcours utilisateur :

#### 1. **IntroActivity** :

- Une porte d'entrée captivante pour accueillir les utilisateurs et les guider vers l'interface principale.

#### 2. **MainActivity** :

- Un espace riche en fonctionnalités où les utilisateurs peuvent explorer des catégories, découvrir les produits les plus populaires, et accéder à des promotions exclusives.

#### 3. **CartActivity** :

- Une interface dédiée à la gestion du panier, offrant une expérience d'achat pratique et organisée.

### 2.2 Structure détaillée du projet

L'application est organisée en différents packages pour une meilleure modularité et maintenabilité :

- **Activity** : Contient les activités principales comme `IntroActivity`, `MainActivity` et `CartActivity`, chacune responsable d'une étape spécifique dans le parcours utilisateur.
- **Adapter** : Inclut des adaptateurs tels que `CartAdapter` et `CategoryAdapter`, utilisés pour gérer l'affichage dynamique des données dans des listes (`RecyclerView`).
- **Domain** : Regroupe les classes représentant les modèles de données (par exemple, `CategoryDomain` et `ItemsDomain`), facilitant la manipulation des objets au sein de l'application.
- **Helper** : Fournit des classes utilitaires pour des fonctionnalités transversales.

### 2.3 Ressources et layouts

Les ressources de l'application sont organisées dans le dossier `res` :

- **Drawable** : Contient les éléments graphiques (icônes, formes, arrière-plans personnalisés).
- **Layout** : Regroupe les fichiers XML définissant l'interface utilisateur, comme `activity_cart.xml` et `viewholder_cart.xml`.
- **Values** : Centralise les valeurs constantes, telles que les couleurs dans `colors.xml` et les chaînes de caractères dans `strings.xml`, garantissant la cohérence visuelle et textuelle de l'application.
- **Themes** : Gère les thèmes, avec des variantes pour le mode jour et nuit.

## 2.4 Méthodes clés

Un exemple notable est la méthode `calculatorCart()` dans `CartActivity`, qui calcule dynamiquement les totaux du panier :

```

1  public class CartActivity extends BaseActivity {
2      private ActivityCartBinding binding;
3      private double tax;
4      private ManagmentCart managmentCart;
5
6
7      @Override
8      protected void onCreate(Bundle savedInstanceState) {
9          super.onCreate(savedInstanceState);
10         binding = ActivityCartBinding.inflate(getApplicationContext());
11         setContentView(binding.getRoot());
12
13         managmentCart = new ManagmentCart(this);
14
15         calculatorCart();
16         setVariable();
17         initCartList();
18
19     }
20
21     private void initCartList() {
22         if (managmentCart.getListCart().isEmpty()) {
23             binding.emptyTxt.setVisibility(View.VISIBLE);
24             binding.scrollViewCart.setVisibility(View.GONE);
25         } else {
26             binding.emptyTxt.setVisibility(View.GONE);
27             binding.scrollViewCart.setVisibility(View.VISIBLE);
28         }
29
30         binding.cartView.setLayoutManager(new LinearLayoutManager(this,
31             LinearLayoutManager.VERTICAL, false));
32         binding.cartView.setAdapter(new
33             CartAdapter(managmentCart.getListCart(), this, () ->
34                 calculatorCart()));
35
36     }
37
38     private void setVariable() {
39         binding.backBtn.setOnClickListener(v -> finish());
40     }
41
42     private void calculatorCart() {
43         double percentTax = 0.02;
44         double delivery = 10;
45         tax = Math.round((managmentCart.getTotalFee() * percentTax *
46             100.0)) / 100.0;
47
48         double total = Math.round((managmentCart.getTotalFee() + tax +
49             delivery) * 100.0) / 100.0;
50         double itemTotal = Math.round((managmentCart.getTotalFee() *
51             100.0)) / 100.0;
52
53         binding.totalFeeTxt.setText("$" + itemTotal);
54
55     }
56
57 }
```

```

47         binding.taxTxt.setText("$" + tax);
48         binding.deliveryTxt.setText("$" + delivery);
49         binding.totalTxt.setText("$" + total);
50     }
51 }
```

Cette méthode met à jour dynamiquement les montants affichés, garantissant une expérience utilisateur fluide et intuitive.

## 2.5 Gestion des avertissements

Quelques optimisations sont recommandées pour améliorer la qualité du code :

- Utiliser des ressources de chaînes avec des espaces réservés (placeholders) dans `setText` pour une gestion optimale des localisations.
- Convertir certains champs en variables locales lorsque cela est pertinent, pour réduire la consommation de mémoire.

## 2.6 Intégration Firebase

Firebase joue un rôle central dans ce projet, garantissant une gestion des données rapide et fiable. La base de données est organisée en plusieurs noeuds pour gérer les produits, les catégories, et les bannières publicitaires.

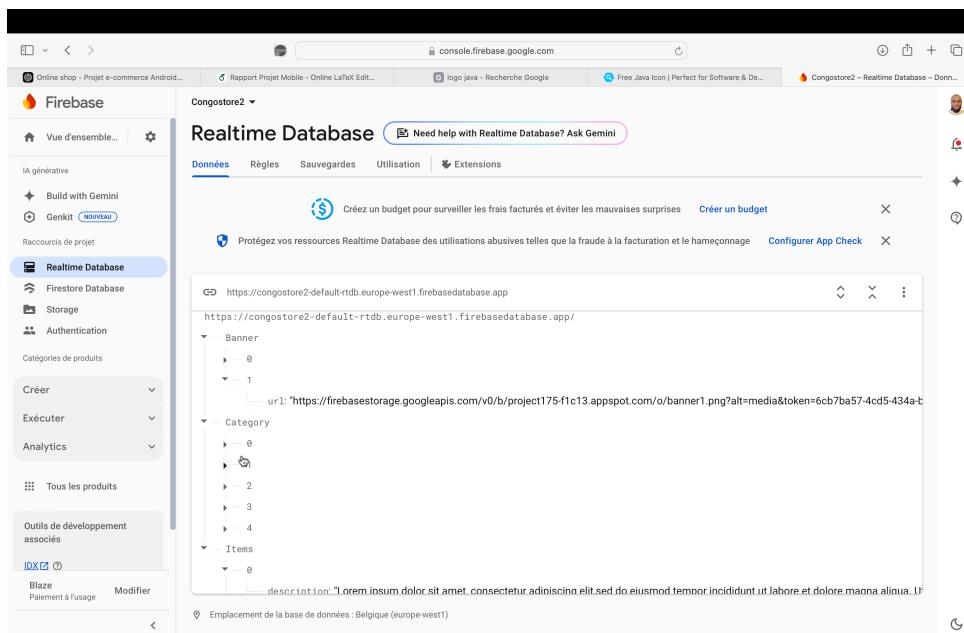


FIGURE 7 – Exemple de structure Firebase.

Cette organisation garantit une synchronisation fluide entre les données affichées dans l'application et la base de données en temps réel.

## 3 Implémentation technique

### 3.1 Navigation

La navigation entre les différentes activités a été soigneusement implémentée pour offrir une transition fluide et intuitive. Par exemple, l’interaction dans `IntroActivity` pour accéder à `MainActivity` se fait via l’écouteur suivant :

```

1 binding.startBtn.setOnClickListener(v -> {
2     startActivity(new Intent(this, MainActivity.class));
3     finish();
4 });

```

Listing 1 – Interaction entre `IntroActivity` et `MainActivity`

Cette simplicité garantit une expérience utilisateur fluide dès les premières interactions.

### 3.2 Chargement des données Firebase

Les données des catégories et des produits sont récupérées dynamiquement depuis Firebase et affichées dans des `RecyclerView`. Chaque élément est traité en temps réel, garantissant une mise à jour instantanée. Par exemple, le chargement des catégories est géré par :

```

1 public CategoryAdapter(ArrayList<CategoryDomain> items) {
2     this.items = items;
3 }
4
5 @NonNull
6 @Override
7 public CategoryAdapter.ViewHolder onCreateViewHolder(@NonNull
8     ViewGroup parent, int viewType) {
9     context = parent.getContext();
10    ViewHolderCategoryBinding binding =
11        ViewholderCategoryBinding.inflate(LayoutInflater.from(context),
12            parent, false);
13    return new ViewHolder(binding);
14 }
15
16 @Override
17 public void onBindViewHolder(@NonNull CategoryAdapter.ViewHolder
18     holder, int position) {
19     holder.binding.title.setText(items.get(position).getTitle());
20
21     Glide.with(context)
22         .load(items.get(position).getPicUrl())
23         .into(holder.binding.pic);
24 }

```

### 3.3 Gestion des `RecyclerView`

L’affichage des listes dynamiques (produits, catégories, bannières) repose sur des `Adapter` personnalisés, garantissant une présentation élégante et fonctionnelle. Exemple d’`Adapter` pour les produits populaires :

```

1
2 public class PopularAdapter extends
3     RecyclerView.Adapter<PopularAdapter.ViewHolder> {
4     ArrayList<ItemsDomain> items;
5     Context context;
6
7     public PopularAdapter(ArrayList<ItemsDomain> items) {
8         this.items = items;
9     }
10
11    @NotNull
12    @Override
13    public PopularAdapter.ViewHolder onCreateViewHolder(@NotNull
14        ViewGroup parent, int viewType) {
15        context = parent.getContext();
16        ViewHolderPopularBinding binding =
17            ViewHolderPopularBinding.inflate(LayoutInflater.from(context), parent, false);
18        return new ViewHolder(binding);
19    }
20
21    @Override
22    public void onBindViewHolder(PopularAdapter.ViewHolder holder, int position) {
23        ItemsDomain item = items.get(position);
24        holder.bind(item);
25    }
26
27    @Override
28    public int getItemCount() {
29        return items.size();
30    }
31
32    class ViewHolder extends RecyclerView.ViewHolder {
33        ...
34    }
35}

```

## 4 Résultats et captures d'écran

### 4.1 Interface utilisateur

Les interfaces utilisateur de l'application ont été soigneusement conçues pour offrir une expérience fluide et agréable. Voici un aperçu des différents écrans de l'application :

- **Écran d'accueil (IntroActivity)**

- **Description :** Cet écran d'accueil introduit les utilisateurs à l'application avec un design épuré et engageant. Le bouton "*Let's Get Started*" guide les utilisateurs vers l'interface principale.

- **Capture d'écran :**

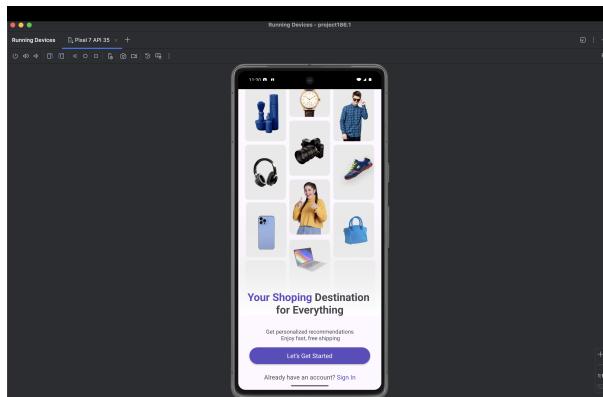


FIGURE 8 – Écran d'accueil (IntroActivity)

- **Écran principal (MainActivity)**

- **Description :** Une interface riche présentant les catégories, les marques officielles, et les produits populaires. Les utilisateurs peuvent naviguer aisément grâce à des sections claires et des visuels attractifs.

— Capture d'écran :

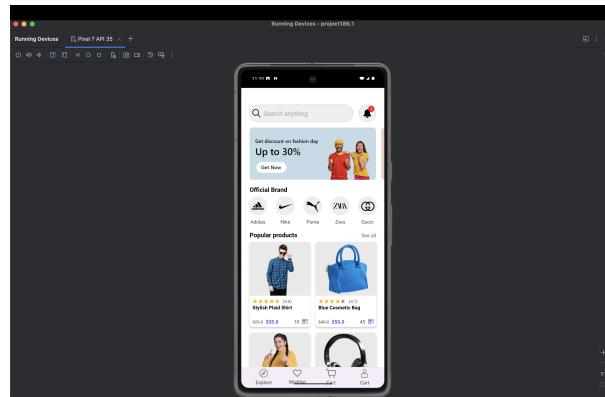


FIGURE 9 – Écran principal (MainActivity)

— Catalogue de produits

- **Description :** Les produits sont affichés sous forme de grille, avec des détails tels que le nom, le prix et la note. Cela permet une navigation intuitive.
- **Capture d'écran :**

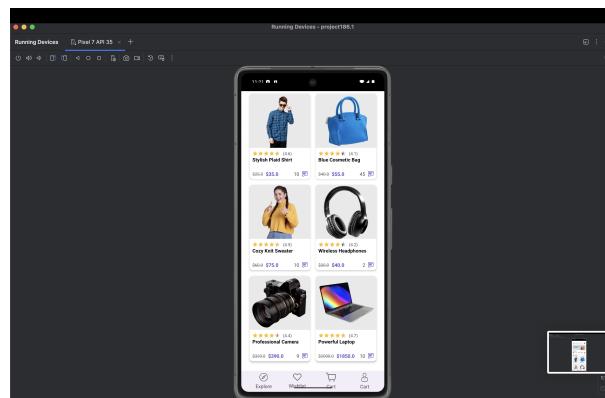


FIGURE 10 – Catalogue de produits

— Détail d'un produit

- **Description :** L'utilisateur peut consulter les détails d'un produit spécifique, y compris sa description, son prix, ses options de taille et de couleur. Un bouton permet d'ajouter le produit au panier.
- **Capture d'écran :**

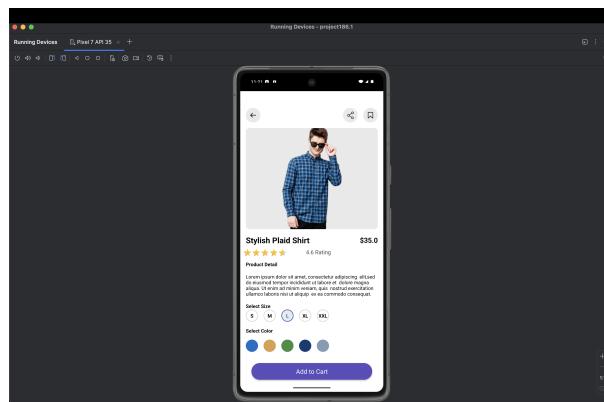


FIGURE 11 – Détail d'un produit

### — Panier (CartActivity)

- **Description :** Une gestion claire et simplifiée du panier, incluant le résumé des articles, les frais de livraison, et le total. L'utilisateur peut ajuster les quantités ou passer à la commande.
- **Capture d'écran :**

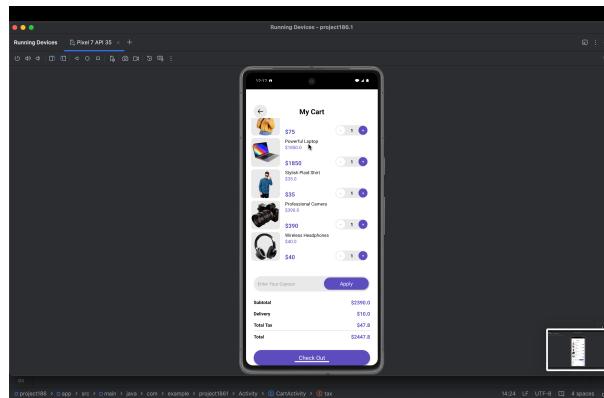


FIGURE 12 – Panier (CartActivity)

## 4.2 Tests et validation

- **Transitions fluides :** Les transitions entre les activités sont fluides et sans latence.
- **Chargement des données :** Toutes les données sont chargées rapidement depuis Firebase et affichées sans erreur.
- **Interface réactive :** L'application s'adapte bien à différentes résolutions, comme celles du Pixel 7 utilisé lors des tests.

## 5 Perspectives et améliorations

Pour améliorer davantage cette application, voici quelques pistes d'évolution :

### 1. Paiements en ligne :

- Intégrer une API sécurisée pour permettre les transactions financières. Cette fonctionnalité offrirait une expérience utilisateur plus complète et augmenterait la conversion des utilisateurs en acheteurs.

## 2. Notifications push :

- Informer les utilisateurs des nouvelles offres ou des promotions en temps réel. Cela permettrait d'augmenter l'engagement des utilisateurs avec l'application et de stimuler les ventes.

## 3. Mode hors ligne avancé :

- Étendre les capacités de persistance Firebase pour permettre un fonctionnement complet sans connexion Internet. Les utilisateurs pourraient consulter les produits et gérer leur panier même en cas de déconnexion, améliorant ainsi l'accessibilité et la satisfaction des utilisateurs.

# 6 Conclusion

Ce projet illustre la combinaison réussie d'un design moderne et d'une architecture performante. L'intégration de Firebase a permis de simplifier la gestion des données en temps réel, tandis que l'utilisation de `RecyclerView` et d'`Adapters` a garanti une présentation fluide et intuitive.

Avec des fonctionnalités bien établies et une base solide, cette application est prête pour des améliorations futures visant à enrichir l'expérience utilisateur. Pour visualiser une démonstration complète de l'application, accédez à la vidéo disponible ici :

**Lien vers la démonstration vidéo :** <https://youtu.be/mH8q9B0G3kE>

# Annexes

## — Fichiers clés :

- `build.gradle` : Pour les dépendances.
- `AndroidManifest.xml` : Pour la déclaration des activités et des permissions.

## — Diagrammes ou schémas :

- Structure Firebase.

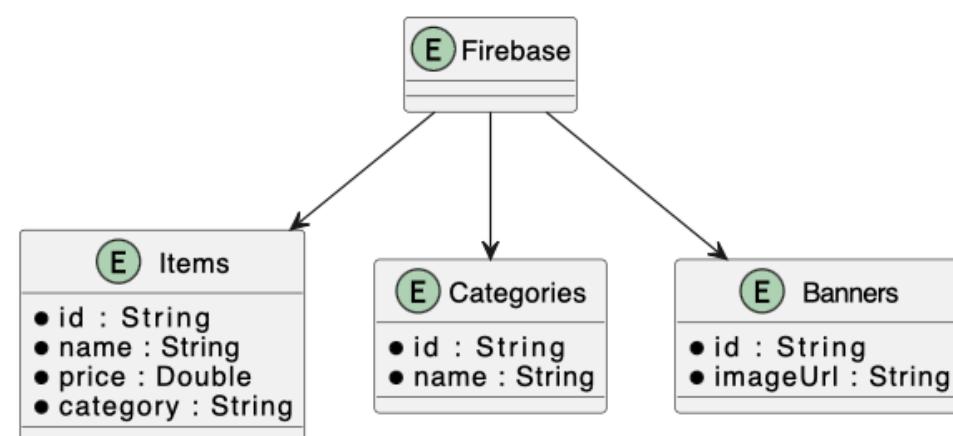


FIGURE 13 – Diagramme de la structure Firebase.

— Architecture des activités.

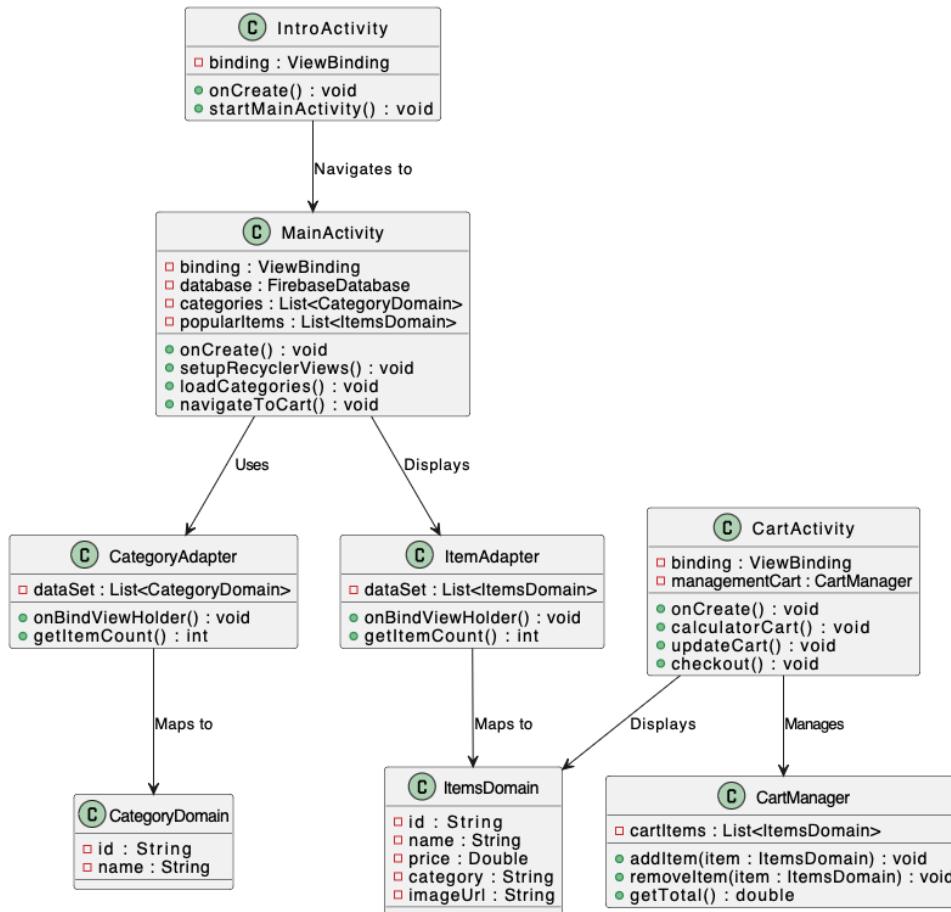


FIGURE 14 – Diagramme de classes de l'architecture des activités.

— Diagramme de cas d'utilisation.



FIGURE 15 – Diagramme de cas d'utilisation.

## Contributions au projet

- **Jancy Fridelin KOUD BANGA** : Responsable principal du développement, ayant réalisé 90% du projet, incluant :
  - Le développement des fonctionnalités principales.
  - L'intégration Firebase et la gestion des données.
  - La documentation technique et utilisateur.
- **Nada BLOUNDI** : Contributions complémentaires, incluant :
  - La relecture et l'amélioration des documents.
  - Le retour utilisateur pour affiner l'interface.