Projet Application Android

Evisual

Groupe A2S3

HERAUD Marie-Amélie

DE BELIZAL Aude

DE LOYNES Jean-Guilhem

Table des matières

I. Remerciements *3*

II. Introduction *4*

III. Description de l'application *5*

1. **Présentation de l'application** 5
2. **Maquette de l'application** 6
3. **Caractéristique de l'application** 7

IV. Architecture de l'application *8*

1. **Architecture Frontend** 9
2. **Base de données** 11
3. **Les ViewModels** 13

V. Outils de développement *14*

1. **Persistance des données grâce à Room** 14
2. **Layout** 15
3. **GitHub** 16

VI. Problème rencontré *17*

VII. Axe d'amélioration *18*

VIII. Diagramme de GANTT *19*

IX. Conclusion *21*

Bibliographie *22*

Index des figures

Figure 1 Maquette préliminaire au développement de l'application 6
Figure 2 : Maquette de la présentation 7
Figure 3 **Architecture MVVM** 8
Figure 4 : Architecture frontend 11
Figure 5 : Modèle conceptuel des données (MCD) 12
Figure 6 Schéma explicatif de notre adaptation du modèle MVVM 13
Figure 7 : Relation entre les différents composants de Room. 14

I. Remerciements

Nous tenons à exprimer notre profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet de programmation mobile.

Tout d'abord, nous remercions chaleureusement Monsieur Hamza, notre superviseur de projet, pour son soutien, ses conseils avisés et sa disponibilité tout au long de ce semestre. Ses remarques constructives et les réunions d'avancements ont été une aide précieuse pour la réalisation de ce travail.

Nous souhaitons également remercier tout le corps enseignant, pour le temps qu'ils nous ont accordés et leurs encouragements constants, ainsi que pour avoir su éveiller notre intérêt pour cette nouvelle manière de programmer.

Un grand merci à l'administration de l'ESME, et en particulier Monsieur Gemmal, pour leur accueil chaleureux et leur assistance technique. Leur expertise et leurs ressources ont été d'une grande aide dans le développement de ce projet.

Nous sommes également reconnaissant envers nos camarades de la classe de système embarqué pour leur soutien moral, leurs discussions stimulantes et leur collaboration tout au long de ce semestre.

Merci à tous pour votre aide et votre soutien.

II. Introduction

La gestion des finances personnelles a toujours été une priorité pour de nombreuses personnes, et est devenu encore plus préoccupant face à l'augmentation des incertitudes économiques. Nous voulons proposer un outil simple et efficace pour suivre ses revenus et ses dépenses au cours du temps. Actuellement, bien que divers outils existent, beaucoup d'entre eux présentent des interfaces complexes, des abonnements coûteux ou un manque d'adaptabilité aux besoins réels des utilisateurs. Ces contraintes créent un besoin grandissant pour des solutions accessibles, intuitives et respectueuses des données personnelles.

Dans ce cadre, l'idée d'une application de gestion budgétaire simple s'est imposée. Cette application vise à offrir une expérience utilisateur intuitive tout en répondant aux besoins fondamentaux de gestion financière. Grâce à des fonctionnalités claires telles que l'enregistrement des dépenses et des revenus, la visualisation graphique des données et des statistiques personnalisées, elle permettra à chaque utilisateur de reprendre le contrôle de ses finances, quel que soit son niveau de familiarité avec la gestion budgétaire.

Ce projet s'inscrit non seulement dans une démarche de réponse à un besoin réel, mais également comme une opportunité de mettre en pratique des compétences techniques en développement mobile sur Android. La conception de cette application représente ainsi un défi stimulant, conciliant utilité sociale et expertise technologique.

Le développement d'une application de gestion des dépenses constitue une excellente pratique pour apprendre à manipuler le langage kotlin et les nombreuses fonctionnalités qu'il propose. Nous détaillerons dans ce rapport notre méthode pour concevoir et implémenter notre application mobile.

III. Description de l’application

1. Présentation de l’application

EVisual est une application de gestion de budget personnel intuitive, conçue pour offrir une solution complète de suivi des revenus et des dépenses. Elle permet à l’utilisateur de surveiller l’évolution de ses finances au fil du temps grâce à des outils de visualisation détaillés et personnalisables (ajout et suppression de dépenses). L’utilisateur renseigne ses revenus réguliers, tels que les salaires ou les allocations, ainsi que ces dépenses quotidiennes, hebdomadaires ou exceptionnelles. L’application s’occupe de trier et de les présenter soit sous forme de liste ordonnée chronologiquement, soit sous forme de graphe récapitulatif ordonné par catégorie de dépense. Ces graphiques permettent de repérer rapidement les postes de dépenses les plus importants et de comparer les tendances au fil des mois ou des années.

Afin de garantir la sécurité des informations, chaque utilisateur se connecte via une adresse électronique unique et un mot de passe. L’application inclut des options pour renforcer la sécurité, avec l’obligation d’utiliser un mot de passe de minimum 8 charactères.

Toutes les données de l’application, y compris les informations personnelles et financières, sont stockées localement sur l’appareil de l’utilisateur, conformément aux exigences du RGPD, garantissant ainsi la confidentialité et le contrôle total sur les données. Lors de la création de son compte, il fournira des informations personnelles modifiables à tout instant via la section "Compte".

Pour trier efficacement les dépenses, l’application se base sur plusieurs critères renseignés par l’utilisateur : le montant, la date, la catégorie (alimentaire, transport, loisirs, etc.). Ces critères permettent à l’utilisateur de retrouver facilement ses dépenses dans l’historique et de générer des rapports précis. Grâce à des graphiques dynamiques, mis à jour par intervalle de temps régulier, l’utilisateur peut suivre l'évolution de ses dépenses mensuelles et annuelles, identifier des schémas récurrents et ajuster ses habitudes financières pour mieux respecter son budget.

2. Maquette de l'application

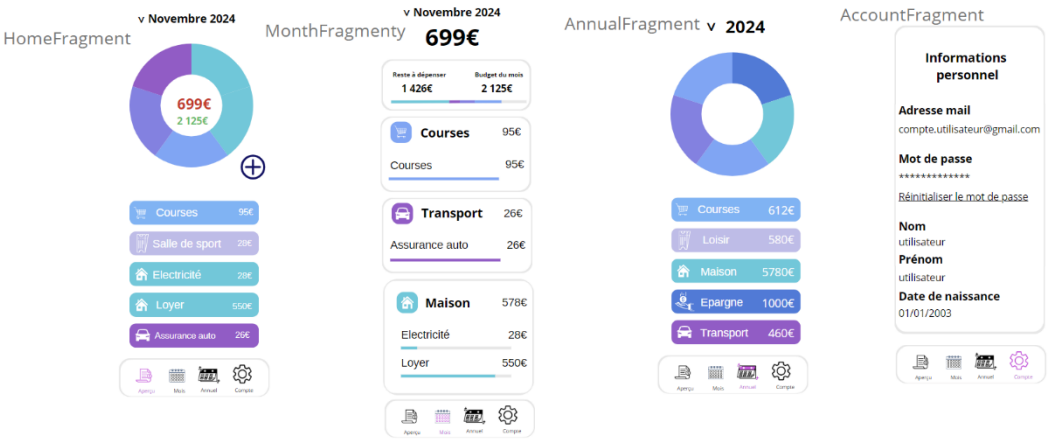

Figure 1 Maquette préliminaire au développement de l'application

Initialement, nous nous sommes basés sur la maquette de la Figure 1, le résultat d'application idéal, pour configurer l'aspect visuel de notre application. Une approche intuitive, qui met en avant la simplicité d'accès aux informations importantes. Chaque page répond à un besoin spécifique, les graphiques permettent une visualisation claire des données utilisées.

Cependant, après avoir commencé à développer l'application, nous avons constaté qu'une fonctionnalité clé n'était pas correctement prise en compte : la gestion des dépenses, notamment la possibilité de modifier ou supprimer une dépense existante. Cette fonctionnalité était cruciale pour l'expérience utilisateur, car elle permettait de gérer facilement les dépenses au fil du temps.

À la suite de cette constatation, nous avons réévalué le design et apporté des modifications importantes à l'interface. Nous avons repensé l'ergonomie et la présentation des écrans pour intégrer ces fonctionnalités de manière fluide. Nous avons repensé les fragments HomeFragment et MonthFragment. Nous avons ajouté la fonctionnalité de modification d'une dépense dans le fragment HomeFragment. L'utilisateur peut grâce à ce nouveau design, cliquer sur une dépense pour l'éditer. Ces ajustements ont permis de rendre l'application plus complète et plus fonctionnelle, tout en assurant une expérience utilisateur plus intuitive et cohérente.

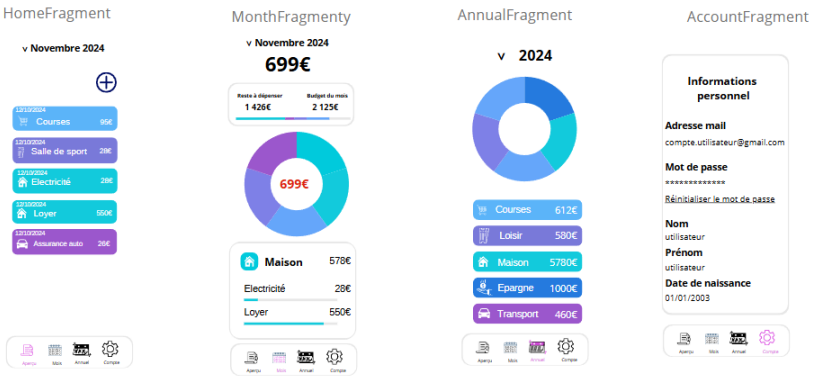

Figure 2 : Maquette de la présentation

3. Caractéristique de l'application

L'application a été développée en utilisant un émulateur Android configuré avec la version Android 8.0 (Oreo), qui repose sur une architecture x86 et utilise l'API 26. Cette configuration nous permet de développer une application compatible avec 96 % des appareil sur le marché tout en profitant des dernières nouveautés proposées par Android 15.0. Le choix d'une version d'Android plus ancienne qu'Oreo n'était pas possible dû à l'utilisation du Framework Room qui aurait pu entraîner des problèmes de compatibilités pour certaines de ses fonctionnalités.

Room est un Framework facilitant la gestion de base de données locale. Cette bibliothèque offre une interface simple et performante pour interagir avec SQLite, permettant de gérer efficacement les données liées aux dépenses et aux comptes des utilisateurs.

IV. Architecture de l'application

Android Studio, l'IDE recommandé par Google pour le développement mobile, propose de travailler en suivant la structure Model-View-ViewModel (MVVM) [1]. Cette manière de travailler plus moderne permet de séparer les fichiers en charges du stockage des donnée de ceux en charges de l'acquisition de donnée ou de signaux. Le modèle représente l'ensemble des entités et des structures de données dont aura besoin la vue pour présenter les informations à l'utilisateur. La vue représente la partie visible de l'iceberg, ce sont les interfaces graphiques (activités, fragments), XML, les widgets, les animations, les textes et tous les composants en interaction directe avec l'utilisateur final. Elle constitue la partie visuelle de l'application, c'est la vitrine qui présente les informations fournies par le Model et traitées par le ViewModel.

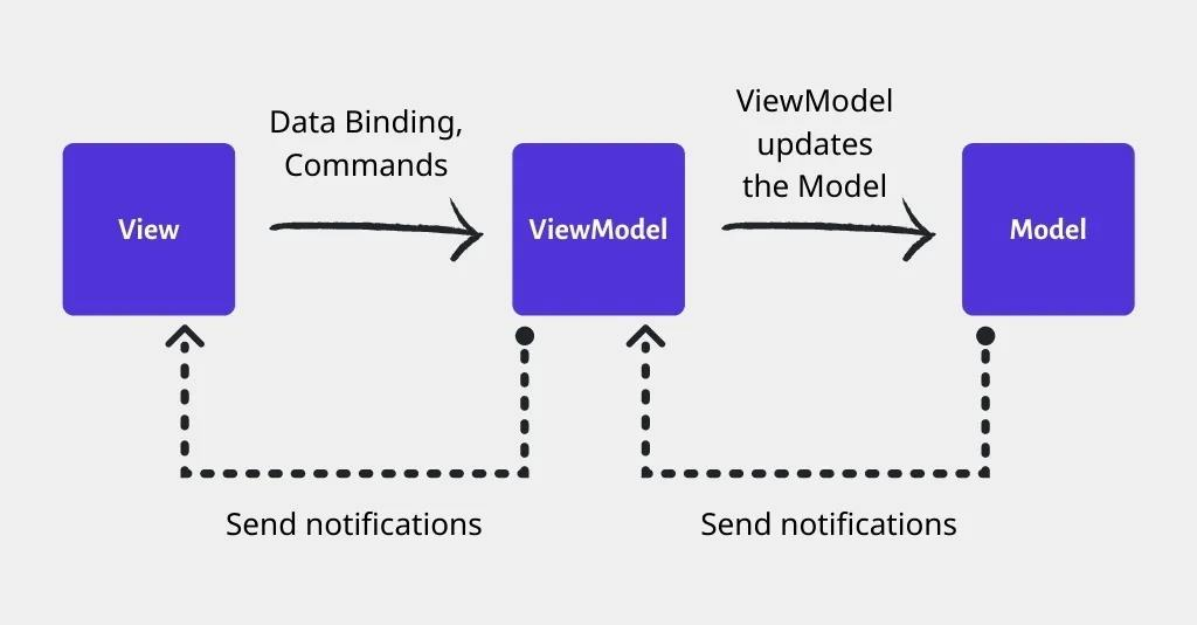

Figure 3 Architecture MVVM

Dans notre projet, les views [2] sont matérialisées par les fichiers XML et Activity, les viewModel sont des classes kotlin contenant les méthodes de traitements de données, et le model est la base de données géré par Room.

Dans le cadre de notre application, chaque Activity et chaque Fragment dispose d'un fichier XML dédié, ce qui permet de définir de manière indépendante la structure et l'agencement des éléments visuels (boutons, champs de texte, images, etc.) pour chaque écran ou section de l'application. Cette organisation modulaire permet de

maintenir une architecture claire et évolutive, et assure une gestion cohérente de l'interface utilisateur.

1. Architecture Frontend

L'application Evisual est structurée autour de quatre activités principales, contenant les quatre fonctionnalités principales, et de plusieurs fragments pour assurer une navigation fluide et intuitive.

La première activité, ConnectionActivity, constitue la page de connexion de l'application. L'utilisateur doit y saisir son adresse mail et son mot de passe pour accéder à l'application s'il dispose déjà d'un compte. Dans le cas contraire, un bouton permet d'être redirigé vers la deuxième activité, CreateComptActivity, où il pourra renseigner ses informations. Une fois cette étape réalisée, l'utilisateur accède à MainActivity.

Le MainActivity gère les différents fragments de navigation de l'application : HomeFragment, MonthFragment, AnnualFragment, et AccountFragment. Ces fragments permettent à l'utilisateur d'accéder aux fonctionnalités clés de l'application.

Le HomeFragment offre une vue d'ensemble des dépenses de l'utilisateur. Ce fragment inclut un sous-fragment, AddDepenseDialogFragment, une modale permettant à l'utilisateur d'ajouter une dépense ou un revenu. L'utilisateur doit associer une catégorie parmi la liste proposée, ajouter une description, un montant, ainsi qu'une date.

Les dépenses sont représentées dans HomeFragment sous forme de cartes colorées, associées à la catégorie sélectionnée lors de la création de la dépense. Chaque carte affiche le montant et la description de la dépense.

À travers le HomeFragment, l'utilisateur est en mesure de modifier ou supprimer une dépense. En cliquant sur une dépense, l'utilisateur ouvre EditExpenseFragment, qui permet de modifier tous les champs de la dépense. Avant la suppression, un message de confirmation est affiché pour que l'utilisateur confirme l'action. Cette interaction avec EditExpenseFragment permet à l'utilisateur de garder le contrôle total sur ses dépenses.

De plus, HomeFragment communique directement avec la base de données. Cela garantit que l'interface utilisateur reste synchronisée avec les changements effectués dans l'application. Par exemple, après la suppression d'une dépense, l'interface de HomeFragment sera automatiquement mise à jour sans avoir besoin de redémarrer l'activité.

Le MonthFragment permet à l'utilisateur de sélectionner un mois et une année spécifique pour visualiser ses dépenses par catégorie. Un message s'affiche lorsqu'aucune dépense n'est trouvé pour la date sélectionné. L'affichage présent permet à l'utilisateur

de visualiser ses revenus, et ce qu'il lui reste à dépenser. Lorsqu'au moins une dépense est trouvée, l'utilisateur a accès à un visuel de ses dépenses, avec un diagramme circulaire qui lui permet de se représenter visuellement la répartition de ses dépenses. Les revenus de l'utilisateur ne sont pas pris en compte dans le diagramme. Au centre de ce diagramme, l'utilisateur peux voir la somme totale de ses dépenses. L'utilisateur dispose également d'une vue plus précise de ses dépense, grâce aux cartes qui répertorie le nom de la catégorie, le montant total dépensé par catégorie, et le détail des dépenses par catégorie, avec la description et le prix associé. Les cartes permettent à l'utilisateur de voir toutes les transactions liées à une catégorie donnée.

Le AnnualFragment fonctionne de manière similaire, mais à l'échelle annuelle. L'utilisateur peut sélectionner une année pour obtenir une répartition budgétaire globale par catégorie. Les informations sont présentées sous forme de graphique circulaire, accompagné d'une liste de carte comportant le nom de la catégorie et le montant total associés. Les revenus de l'utilisateur ne sont pas pris en compte sur le graphique mais sont visible grâce à une carte.

Enfin, le AccountFragment permet à l'utilisateur de gérer ses informations personnelles et son mot de passe. À l'aide du sous-fragment ResetPasswdFragment, il peut réinitialiser son mot de passe en cas d'oubli. Il est également possible de compléter ou de modifier ses informations personnelles, telles que son nom, son prénom, et sa date de naissance. L'utilisateur peut également se déconnecter de l'application en utilisant le bouton déconnection dans ce fragment.

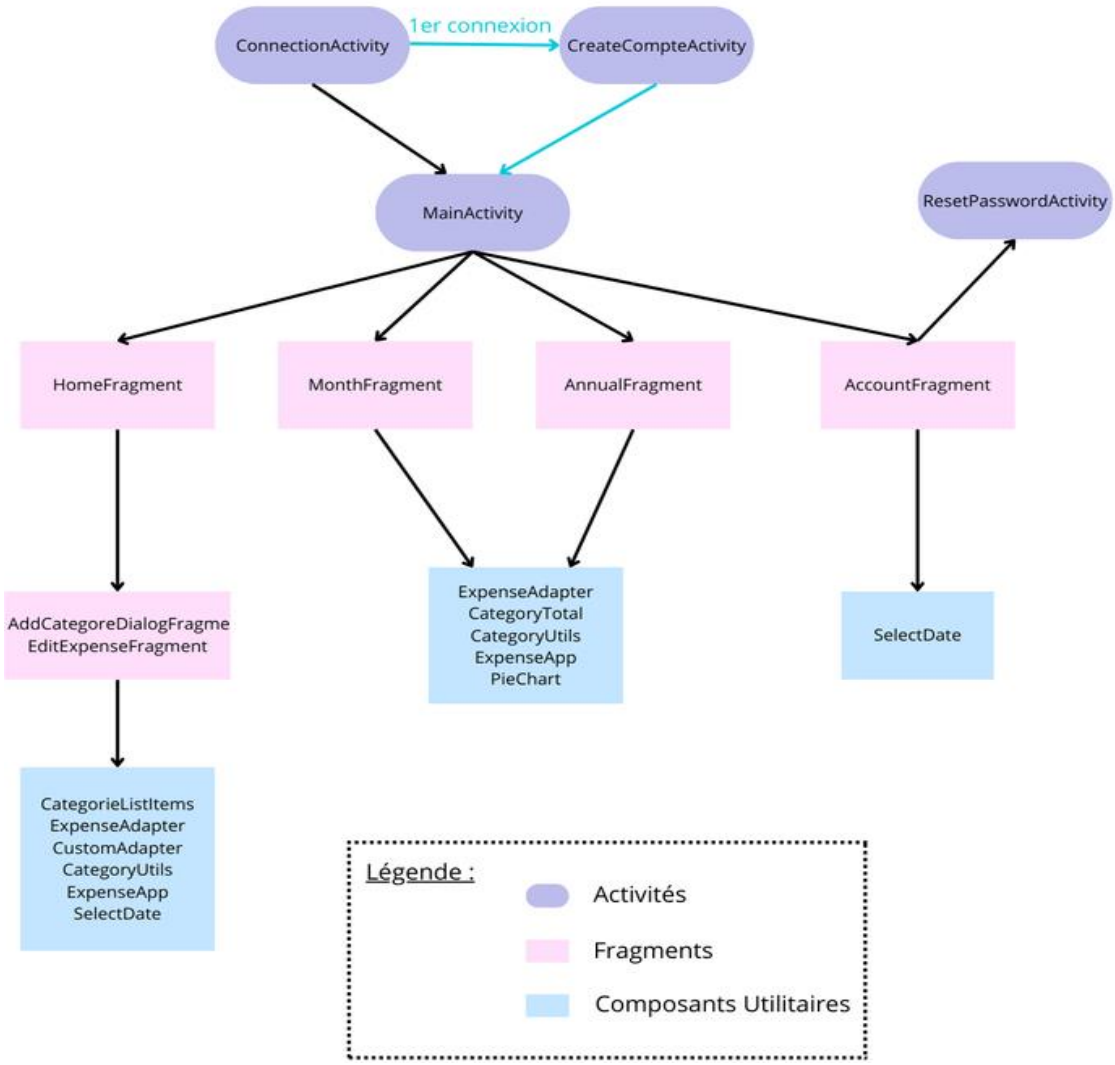

Figure 4 : Architecture frontend

2. Base de données

Dans notre application, la gestion des données repose sur l'utilisation d'une base de données relationnelle, qui permet l'organisation et l'accès rapide aux informations. Cette base permet de traiter efficacement de grands volumes de données tout en garantissant la sécurité, l'intégrité et la cohérence des informations. Pour assurer cette gestion, nous avons conçu un modèle relationnel, qui est représenté dans le diagramme des entités **Figure 5**. Ce modèle structuré se compose de plusieurs tables, chacune représentant une entité du système.

Le modèle logique de données (MLD) que nous avons défini repose sur deux tables : Users et Expense. La table Users contient les informations personnelles des utilisateurs,

tandis que la table Expense stocke les détails des dépenses réalisées par ces utilisateurs. Chaque dépense de la table Expense est associée à un utilisateur spécifique par la clé étrangère Id_User. Plus précisément, la table Users se compose des colonnes suivantes : Id_User (clé primaire), Name, Firstname, Birthday, Email, et Password. Ces informations sont nécessaires pour garantir l'unicité des utilisateurs et la sécurisation de l'accès aux données.

La table Expense, quant à elle, comprend les colonnes Id_Expense (clé primaire), Id_User (clé étrangère faisant référence à Id_User de la table Users), Category, Description, Date, et Amount. La présence de la clé étrangère permet la liaison entre les deux tables : chaque utilisateur peut avoir plusieurs dépenses, mais chaque dépense est associée à un et un seul utilisateur. Cette organisation reflète une relation de type "un à plusieurs" (1, N).

Cette séparation en deux tables présente plusieurs autres avantages. Elle permet d'éviter la redondance des données, les informations étant relatives à un utilisateur ne sont pas répétées pour chaque nouvelle dépense. Cela permet également de maintenir la cohérence des données : toute modification des informations personnelles d'un utilisateur dans la table Users se répercute automatiquement sur les données liées dans la table Expense. Enfin, cette structure respecte la troisième forme normale (3NF) de la normalisation des bases de données, ce qui assure qu'aucune donnée dans une table ne dépend d'une autre que de sa clé primaire, réduisant ainsi les risques d'anomalies ou d'incohérences.

Pour l'intégration de ce modèle dans notre application, nous avons opté pour l'utilisation de la bibliothèque Room. Cette bibliothèque simplifie l'interaction avec la base de données SQLite, en facilitant la gestion des entités et des relations tout en garantissant une gestion fluide des données et des opérations de maintenance.

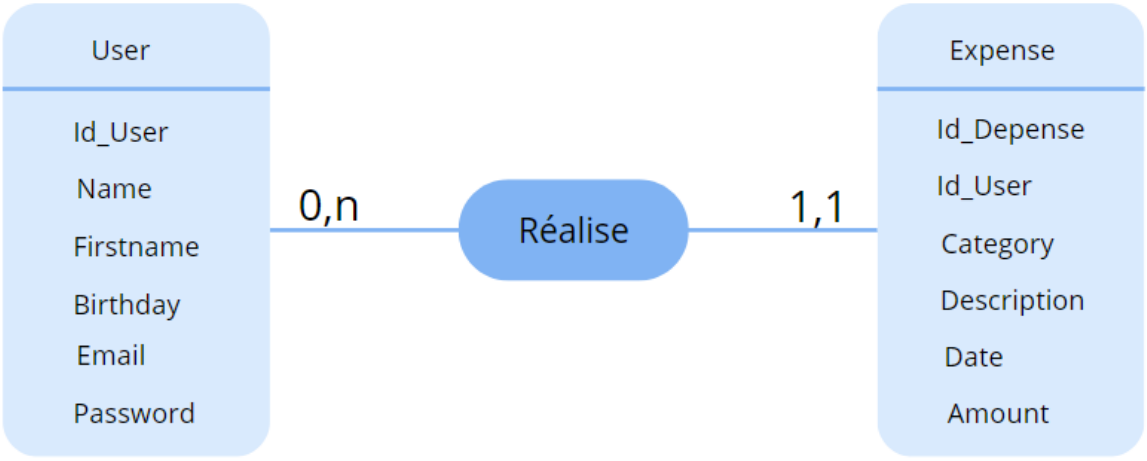

Figure 5 : Modèle conceptuel des données (MCD)

3. Les ViewModels

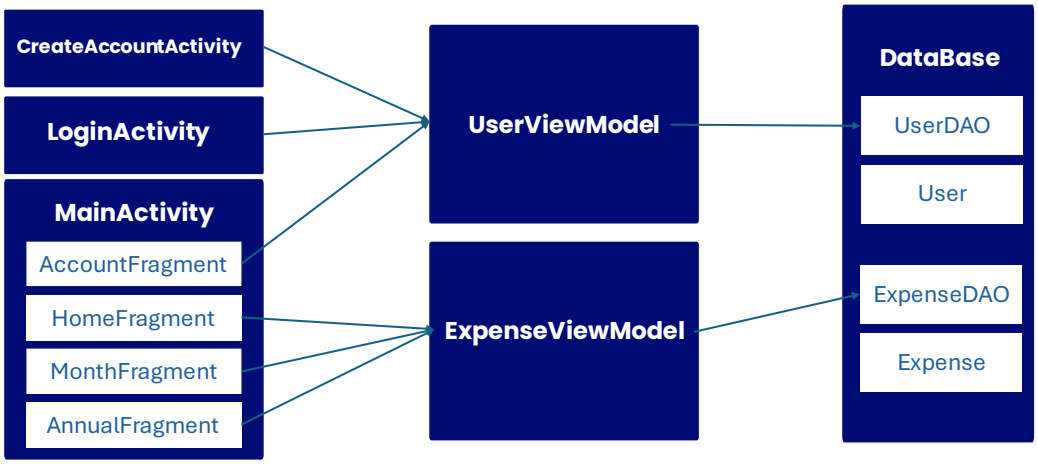

Figure 6 Schéma explicatif de notre adaptation du modèle MVVM

Pour accéder et manipuler ces données nous utilisons des ViewModels. Ce sont des classes contenant les méthodes permettant d'acquérir, ou de stocker, des données dans la base de données et de les manipuler. Pour une meilleure maintenance de notre code nous avons créé un ViewModel par table. Pour pouvez voir sur le diagramme de classes de la Figure 6 que les ViewModel font l'interface entre les Views, les classes d'affichages des informations, et la base de données, classes de stockage de l'information. Vous pouvez observez des classes filles dans la classe MainActivity et dans la classe DataBase : cet agencement sert à mieux structurer notre code. Dans le premier cas, chaque classe contient le code d'affichage d'une page de notre application. Dans le second cas les classes DAO contiennent les requêtes SQL nécessaires à l'acquisition et la manipulation des données, tandis que les classes User et Expense sont des objets instanciable dans le code pour réaliser des opérations logiques avec les données ou les afficher.

V. Outils de développement

1. Persistance des données grâce à Room

La bibliothèque de persistance Room [3] fournit une couche d'abstraction sur SQLite [4] afin de permettre un accès fluide à la base de données, tout en exploitant toute la puissance de SQLite. Room offre, en particulier, les avantages suivants :

- Une vérification des requêtes SQL au moment de la compilation.
- Des annotations pratiques qui réduisent le code récurrent qui peut s'avérer répétitif et être sujet aux erreurs.
- Des chemins de migration simplifiés pour les bases de données.

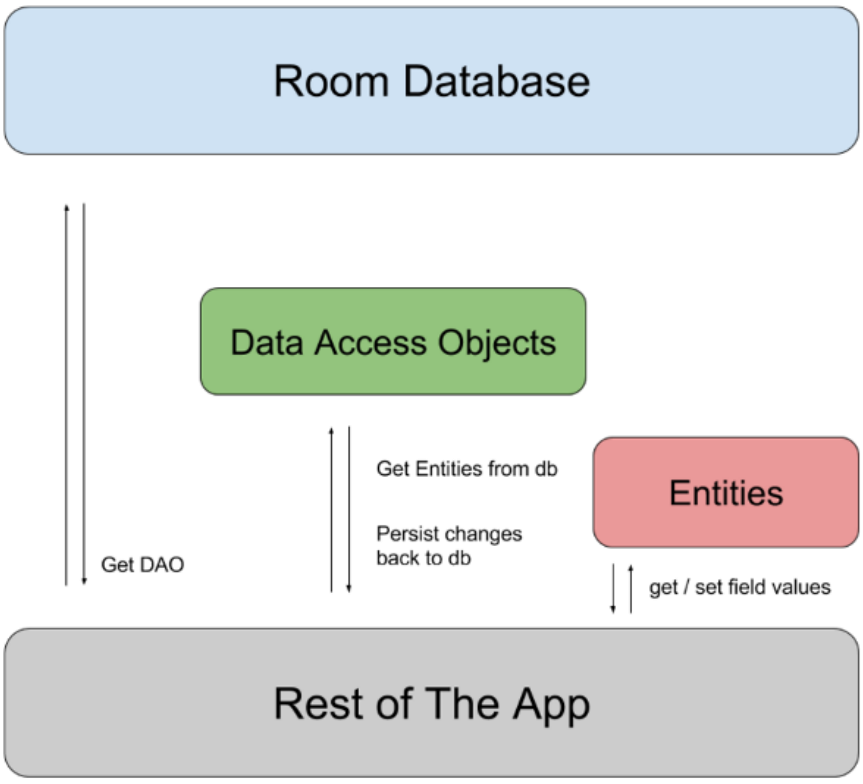

Figure 7 : Relation entre les différents composants de Room.

Room repose sur trois composants principaux :

- La classe Database, qui permet de construire et d'instancier notre base de données et lui fournir les paramètres d'utilisation.

- Les classes « entity », qui représentent les tables de la base de données de notre application, on pourra y définir tous nos champs et leur type. Ces classes pourront être instancier pour manipuler les données.
- Les classes d'objets d'accès aux données (DAO) qui fournissent des méthodes que notre application peut utiliser pour interagir avec la base de données

Afin de faciliter l'utilisation de Room par les autres composants de notre application, nous avons utilisé la structure MVVM détaillé précédemment. Nous avons créé un viewModel par table qui permet d'appeler des méthodes pour réaliser les actions nécessaires au fonctionnement de l'application.

Par exemple, lorsque l'utilisateur clique sur le bouton « se connecter », l'activity va appeler la méthode « login » du UserViewModel qui se chargera de définir l'utilisateur en tant que connecté dans la base de données à partir des informations données en paramètre. Il est nécessaire de créer un constructeur personnaliser pour appeler un ViewModel avec des arguments, d'où la présence des ViewModelFactory.

2. Layout

En développement android, un layout est une structure fondamentale utilisée pour organiser les éléments d'interface utilisateur sur l'écran. Il définit la position, la taille, l'alignement et les relations entre les différents composants visuels (boutons, textes, images, etc). Les layouts permettent de créer une interface claire et cohérente, facilitant l'interaction de l'utilisateur avec l'application. Chaque type de Layout possède ses propres règles de positionnement, offrant des solutions adaptées à différentes configuration d'affichage.

Parmi les layout utilisés dans notre application, linearlayout est souvent employé pour aligner les éléments de manière séquentielle, soit verticalement, soit horizontalement. Cette simplicité de disposition est idéale pour les affichages avec un enchaînement naturel des composants, comme une liste d'éléments à remplir ou des boutons alignés côte à côte.

Le RelativeLayout, également utilisé dans certaines sections, permet de positionner les éléments les uns par rapport aux autres. Par exemple, un bouton peut être aligné à droite d'un texte ou placé en dessous d'une image. Cette flexibilité permet de créer des interfaces dynamiques dîtes « responsive », c'est-à-dire qui s'adapte à toutes les tailles d'écran.

L'utilisation de ces différents Layouts a permis de concevoir une interface harmonieuse et ergonomique. Chaque choix de Layout repose sur la nécessité de répondre à des

exigences spécifiques : faciliter la lisibilité, optimiser l'espace et garantir une expérience fluide quel que soit le type d'appareil.

3. GitHub

Git est un outil de gestion de versions, largement utilisé dans le développement de logiciels pour suivre l'historique des modifications, collaborer efficacement, et maintenir une version stable du projet.

Github est un logiciel en ligne de partage de code construit autour de Git. Il permet aux différents contributeurs d'un projet d'apporter leurs modifications tous en gardant une version stable du projet dans un répertoire commun. L'historique des modifications apportées peut être retracé via les commits, un système obligeant les contributeurs à commenter leurs apports au projet. Ce système garantit la traçabilité des contributions et facilite le retour à une version antérieure en cas de problème. L'utilisation des branches permet de travailler dans des espaces différentes, et de résoudre les conflits de compatibilité tout en préservant la stabilité de la branche principale. Un système de pull request permet de demander la fusion de ses modifications dans la branche principale. Cette demande est soumise à un processus de validation, incluant une revue du code par d'autres membres de l'équipe. Grâce à GitHub, la gestion des versions de l'application a été optimisée et compatible avec les standards du projet avant d'être intégré. En somme, GitHub a été un outil indispensable pour la bonne organisation du développement de notre application, en permettant un travail collaboratif structuré et une bonne gestion du développement. La collaboration entre Git et GitHub sont des outils indispensables pour assurer un développement structuré et fluide.

VI. Problème rencontré

Lors du développement de notre application nous avons rencontré plusieurs défis techniques et organisationnels. Nous avons pu les surmonter au fur et à mesure de l'avancée de l'application.

Le principal obstacle a été la configuration du fichier Gradle [5] qui gère les dépendances et la compilation de notre projet. L'ajout de certaines dépendances s'est révélé particulièrement complexe, notamment pour des bibliothèques comme Room et un graphique circulaire que nous avons réussi à implémenter avec les librairies. Nous avons rencontré des problèmes d'incompatibilité entre les versions des plugins et des dépendances, ce qui a entraîné des erreurs récurrentes lors de la compilation. En particulier, l'intégration de la bibliothèque Room a posé des difficultés liées aux versions de l'API et des configurations spécifiques.

De plus, pour le graphique, nous n'avons pas réussi à faire fonctionner la bibliothèque choisie, malgré plusieurs tentatives d'ajustement. Nous avons dû investir beaucoup de temps pour résoudre ces problèmes, en ajustant les versions des dépendances et en recherchant des solutions dans la documentation officielle. Ce défi a retardé certaines phases du développement, mais après plusieurs tentatives, nous avons pu stabiliser la configuration de Gradle et garantir la compatibilité des dépendances avec le projet.

Nous avons aussi rencontré un problème entre la version de l'API que nous utilisions et la bibliothèque Room. Nous avions initialement prévu de développer l'application en utilisant l'API 25 (Android 7.1.1). Cependant, cette version de l'API ne permettait pas une intégration des coroutines, utilisées par Room. Room offre des fonctionnalités modernes comme la vérification des requêtes SQL, la possibilité d'émettre plusieurs requêtes via des threads, au moment de la compilation et une meilleure gestion des relations entre tables. Malheureusement, en API 25, certaines fonctionnalités avancées de Room ne sont pas compatibles ou sont limitées, ce qui nous a contraints à envisager une mise à jour vers une version d'API plus récente pour profiter pleinement des avantages de cette bibliothèque.

En parallèle, nous avons rencontré un problème avec la gestion de la base de données. En utilisant Room pour la gestion des données locales, nous avons dû faire face à la nécessité de mettre à jour manuellement la base de données en local à chaque modification des schémas ou des entités. Ce processus n'était pas optimal et a entraîné des risques d'erreurs humaines, notamment lors de la mise à jour des versions de la base de données. Pour contourner cela, nous avons dû adopter un workflow spécifique qui consiste à faire des sauvegardes régulières de la base de données et à tester minutieusement chaque changement dans un environnement local avant de le déployer dans l'application.

Nous avons également rencontré des difficultés dans la gestion du travail en groupe. L'utilisation de GitHub, en particulier, a été source de défis. Des problèmes fréquents avec la gestion des branches et des conflits lors des fusions ont ralenti le processus de développement. Nous avons appris à mieux structurer notre travail, notamment en définissant l'ordre des branches à fusionner en fonction des modifications de chacun. Cela nous a permis d'améliorer la coordination au sein du groupe.

Notre architecture initiale reposait sur une activité distincte pour chaque page. Cette approche, bien qu'intuitive au départ, a rapidement révélé ses limites. La gestion des transitions entre activités était complexe, et il était nécessaire de dupliquer la barre de navigation dans chaque activité. Pour résoudre ce problème, nous avons opté pour une architecture basée sur des fragments. Cette solution a considérablement simplifié la gestion de la navigation et a permis de centraliser l'interaction entre les différentes pages de l'application.

VII. Axe d'amélioration

Dans tout projet de développement d'application, l'amélioration continue est essentielle pour répondre aux attentes des utilisateurs et garantir une expérience optimale. Evisual, en tant qu'application en cours de conception, offre une excellente base, mais des optimisations peuvent encore être apportées pour maximiser sa performance, améliorer l'expérience utilisateur et renforcer sa fiabilité.

- Amélioration de l'interface visuel, avec par exemple une meilleure disposition de l'affichage, entre le diagramme et les fragments. Intégrez des illustrations modernes ou des icônes personnalisées pour enrichir l'apparence de l'application. (ex. : bouton qui change de couleur lorsqu'il est cliqué, mettre une icône poubelle pour supprimer un budget).
- Thème adaptatif en proposant un mode clair et un mode sombre pour s'adapter aux préférences des utilisateurs.
- Amélioration de la gestion de la base de données
- Implémentation de la librairie PieChart [6] dans le système, pour faciliter l'implémentation des diagrammes.
- Ajout d'informations utiles, à intégrer dans la page resetcreataccount les sections comme "À propos", "Mentions légales", "Conditions d'utilisation" et "FAQ".
- Intégration d'un page paramètre, avec ajout de fonctionnalité multilingue.
- Intégration d'une table catégorie, personnalisable ne fonction de l'utilisateur.
- Partenariat avec une banque, afin que les dépenses soit automatiquement afficher.

En apportant ces améliorations, Evisual pourra non seulement séduire ses utilisateurs grâce à une esthétique soignée et des fonctionnalités engageantes, mais aussi se positionner comme une application pratique, intuitive et performante.

VIII. Diagramme de GANTT

Afin de gérer au mieux notre projet nous avons choisi de reprendre et d'adapter la méthode très connue du cycle en V. Cette méthode, détaillée sur la figure 8, permet de définir toutes les étapes de développement d'un projet. Nous sommes donc parties de notre besoin, la réalisation d'une application de gestion de budget, pour décomposer notre système en blocs fonctionnels. Une fois la réalisation de chaque bloc effectué, nous avons pu réaliser des tests unitaires. Une fois chaque fonction terminée, nous avons pus assembler notre code pour réaliser un projet complet.

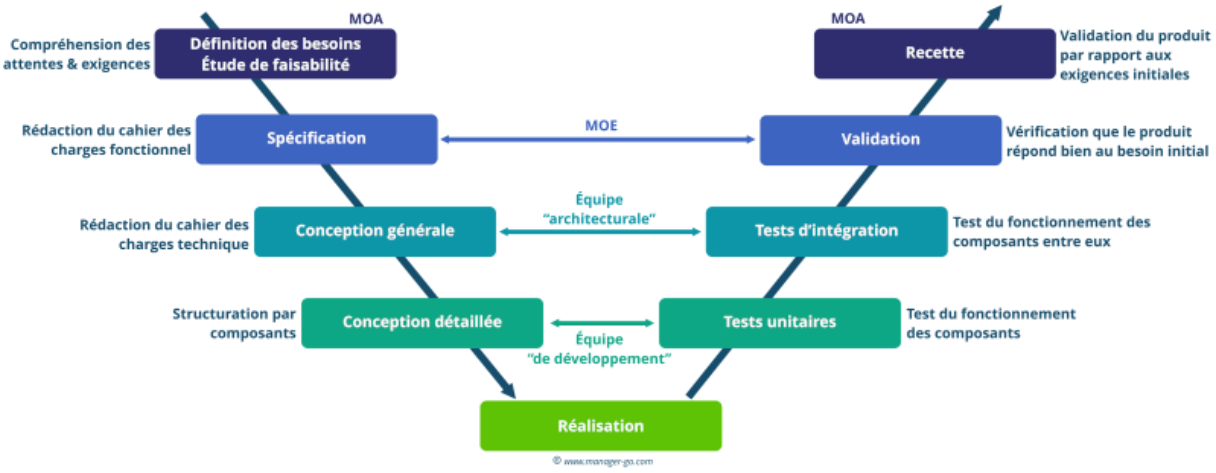

Figure 8 Schéma du cycle en V

Pour réaliser ce projet nous avions à notre disposition 20 heures de travaux pratiques et une équipe de trois étudiants. En tirant leçon de nos précédant projet, nous avons eu une approche beaucoup plus méthodique. Dès le début du semestre nous avons estimé le temps que nous prendrais la réalisation de chaque bloc fonctionnel, l'assemblage des différents blocs et la qualification du système. Le temps de rédaction du rapport n'a pas été négligé non plus, la phase de rédaction peut parfois être longue mais permet d'organiser les idées et de s'assurer de ne pas avoir oublié d'étapes lors de la réalisation de ce projet.

La réalisation des tests unitaires des différents blocs a pu être répartis entre les trois membres du groupe pour paralléliser les tâches, puis les phases de fusion du code et de qualification ont pu être faites tous ensemble.

La figure 9 présente le diagramme de Gantt que nous avons réalisé en début de projet. C'est ce diagramme qui nous a permis de suivre notre avancement semaine après semaine et de nous assurer que nous ne prenions pas de retard sur le planning. L'utilisation d'un diagramme de Gant permet aussi d'arriver en travaux pratique avec un objectif pour les quatre heures qui suivent : essentiel pour ne pas perdre de vue la direction à prendre.

Figure 9 Diagramme de Gant

IX. Conclusion

Le développement de l'application Evisual, une solution mobile pour la gestion de budget personnel, a été une expérience très formatrice. À travers ce projet, nous avons relevé des défis qui nous ont permis de découvrir le développement mobile. Parmi les nombreuses options que propose la programmation mobile, nous avons notamment pu nous familiariser avec le langage Kotlin, l'IDE Android Studio, la bibliothèque Room et l'architecture MVVM. Nous avons construit une application fonctionnelle capable de répondre aux besoins de gestion financière quotidienne sans omettre les standards du développement Android.

Les obstacles rencontrés, qu'ils concernent la configuration de Gradle, les compatibilités des API, la gestion des dépendances ou l'intégration de la navigation par fragments, ont été autant d'opportunités d'apprentissage. Ils nous ont poussés à adopter une approche plus structurée et à approfondir notre compréhension des outils utilisés. Ces défis nous ont également sensibilisés à l'importance d'une planification minutieuse et d'une coordination efficace, notamment dans la gestion du travail en équipe via GitHub.

L'analyse fonctionnelle et les axes d'amélioration proposés dans ce rapport témoignent de notre volonté d'offrir une application non seulement fonctionnelle, mais aussi modulaire. L'intégration de nouvelles fonctionnalités a été facilité par l'adoption de l'architecture MVVM et l'exploitation des objets Kotlin.

En conclusion, Evisual constitue une base solide et prometteuse pour une application de gestion de budget moderne. Avec les améliorations envisagées, elle pourrait se positionner comme une solution incontournable pour les utilisateurs recherchant simplicité, efficacité et flexibilité. Ce projet nous a également permis de renforcer nos compétences en gestion de projet, en résolution de problèmes et en collaboration, des atouts qui pourront être réutilisé lors de nos futurs travaux de développement.

Bibliographie

[1] G. Tekombo, «Medium,» [En ligne]. Available: https://medium.com/androidmood/comprendre-larchitecture-mvvm-sur-android-aa285e4fe9dd.

[2] Google, «Android developpers,» [En ligne]. Available: https://developer.android.com/develop/ui/views/layout/responsive-adaptive-design-with-views?hl=fr.

[3] Google, «Android developer : Room,» [En ligne]. Available: https://developer.android.com/training/data-storage/room?hl=fr.

[4] SQLite, «What is SQLite ?,» [En ligne]. Available: https://www.sqlite.org.

[5] Gradle Build Tool, [En ligne]. Available: https://gradle.org.

[6] PhilJay, «MPAndroidChart,» [En ligne]. Available: https://github.com/PhilJay/MPAndroidChart.