Sorbonne Université

Rapport de Projet TPALT

28 mars 2024

# Exploration de Swift - Application de gestion de budget SophistiSpend

*Auteur :*
Alina Novikova

*Encadrant :*
Katia Amichi

## Table des matières

# 1 Introduction

Dans le cadre de mes études, j'ai entrepris un projet de recherche visant à explorer et à approfondir mes connaissances dans le domaine du développement d'applications mobiles. Cette initiative était motivée par mon désir de découvrir de nouveaux langages de programmation et d'acquérir une expérience pratique dans la conception et le développement d'applications modernes.

Le principal objectif de ce projet était d'explorer une technologie que je n'avais pas encore eu l'occasion d'explorer. Alors que Flutter, développé par Google, offre une alternative pour le développement d'applications Android et iOS, j'ai choisi de me tourner vers Swift pour ses caractéristiques uniques. Je souhaitais découvrir les avantages qu'il offre en termes de performance, de sécurité et de soutien communautaire, notamment grâce au support d'Apple. Mon objectif était donc double : apprendre Swift et l'utiliser pour créer une application qui intègre des concepts d'architecture et des outils de pointe. Ce parcours m'a conduit à une immersion intense dans l'apprentissage de Swift et au développement d'une application fonctionnelle, concrétisée par la réalisation de SophistiSpend.

Dans ce rapport, je vais présenter en détail le contexte de mon projet, les objectifs que j'ai définis, la méthodologie que j'ai suivie pour atteindre ces objectifs, ainsi que les résultats obtenus et les leçons apprises au cours de ce processus. Je vais également discuter des défis rencontrés et des stratégies que j'ai utilisées pour les surmonter, ainsi que des perspectives pour des travaux futurs et des recommandations pour d'autres étudiants intéressés par des projets similaires.

Ce rapport est structuré de manière à offrir une vue d'ensemble complète de mon projet de recherche, en mettant en lumière les aspects clés de mon parcours d'apprentissage et de développement. J'espère que ce travail servira de guide et d'inspiration pour d'autres étudiants qui envisagent de se lancer dans des projets de recherche similaires et contribuera à enrichir notre compréhension collective des technologies émergentes et des meilleures pratiques en matière de développement logiciel.

# 2 Choix du sujet du projet

Le choix du sujet du projet s'est fait en prenant en compte plusieurs contraintes temporelles et de compétences. Avec une durée limitée de sept semaines et une disponibilité de travail maximal de six heures par semaine, il était essentiel de sélectionner un projet réaliste et réalisable dans ces limites. Étant donné que je travaille seul sur ce projet, sans coéquipier, il était important de choisir un sujet qui ne soit pas trop complexe à mettre en œuvre.

De plus, j'ai pris en considération le fait que je suis également un utilisateur d'un système de gestion de budget personnel. Cela m'a incité à choisir une application de gestion de budget simple comme sujet du projet. Cela correspondait également à l'idée de développer une application qui répondait à un besoin personnel, ce qui rendait le projet d'autant plus motivant.

Enfin, le choix de cette application s'alignait avec l'objectif global du projet, qui était de découvrir une nouvelle technologie, en l'occurrence le langage Swift. En optant pour un projet concret et pertinent comme la gestion de budget, j'ai pu mettre en pratique mes nouvelles compétences en Swift tout en répondant à un besoin personnel.

Ces considérations ont donc conduit au choix du sujet du projet de gestion de budget simple, qui a permis d'allier efficacement les contraintes de temps, de compétences et de pertinence personnelle.

# 3 L'architecture de haut niveau

Dans la conception de l'architecture pour mon application de gestion de budget, j'ai exploré plusieurs options traditionnelles pour le stockage et le traitement des données. Parmi celles-ci figuraient `Firebase, SwiftData et Core Data`, chacune offrant ses propres avantages et défis.

Initialement, j'ai envisagé l'utilisation de Firebase en raison de sa facilité d'intégration et de son stockage NoSQL dans le cloud. Cependant, après avoir examiné de plus près les besoins spécifiques de mon application, j'ai décidé de ne pas poursuivre cette voie. Bien que je possède une certaine expérience avec Firebase, j'ai estimé que son modèle NoSQL n'était pas le plus adapté pour les données relationnelles et structurées requises par une application de gestion de budget.

En ce qui concerne SwiftData et Core Data, bien que ces solutions offrent des fonctionnalités intéressantes pour le stockage local sur l'appareil, leur apprentissage représentait un défi supplémentaire. Étant donné les contraintes de temps du projet et le besoin de se concentrer sur des technologies déjà maîtrisées, j'ai décidé d'explorer d'autres options.

C'est ainsi que j'ai choisi d'adopter **Go** pour le développement du backend de l'application. Ayant déjà une certaine expérience dans la construction d'architectures REST API avec Go, j'ai vu cette décision comme une opportunité d'approfondir mes connaissances et d'explorer pleinement les capacités de ce langage.

La combinaison de Swift pour le développement frontal, de Go pour le backend robuste et de PostgreSQL pour la gestion efficace des données relationnelles est apparue comme la solution optimale. Cette approche me permet de capitaliser sur mes connaissances existantes tout en relevant de nouveaux défis d'apprentissage et d'intégration.

# 4 Wireframes

La phase de conception de mon projet a débuté par la création de wireframes détaillés pour l'interface utilisateur de l'application. Ces wireframes ont servi de schéma directeur pour la disposition des éléments sur les différentes vues de l'application.

J'ai utilisé l'outil de conception Figma pour élaborer mes wireframes, permettant ainsi une visualisation précise de la structure et du flux de l'application. Les wireframes comprenaient des représentations simplifiées des écrans principaux de l'application, tels que l'écran d'accueil, l'écran de gestion des transactions et l'écran de détails de compte.


FIGURE 1 – Color schema

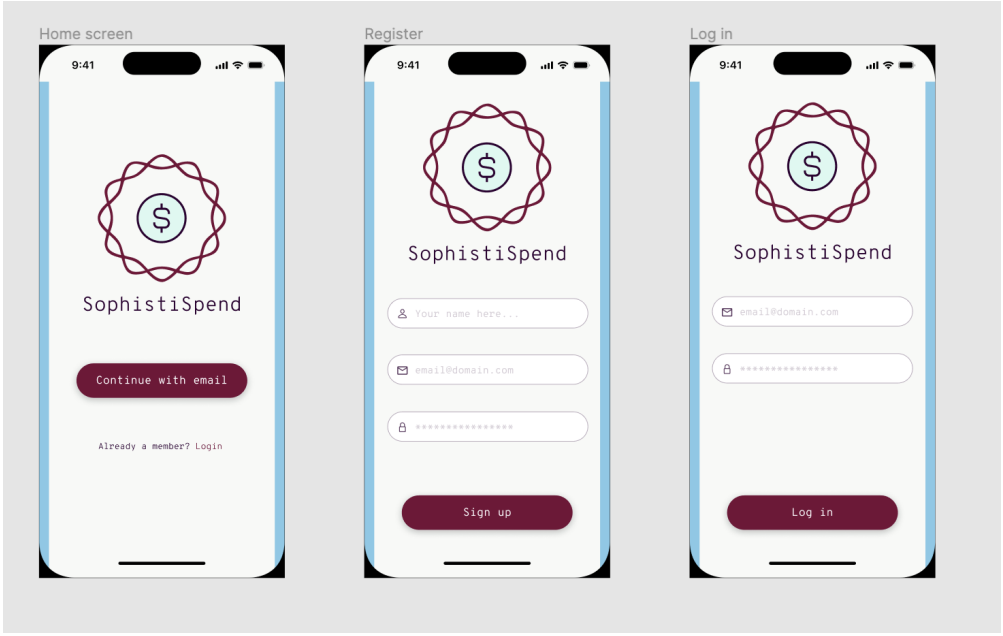


FIGURE 2 – Login

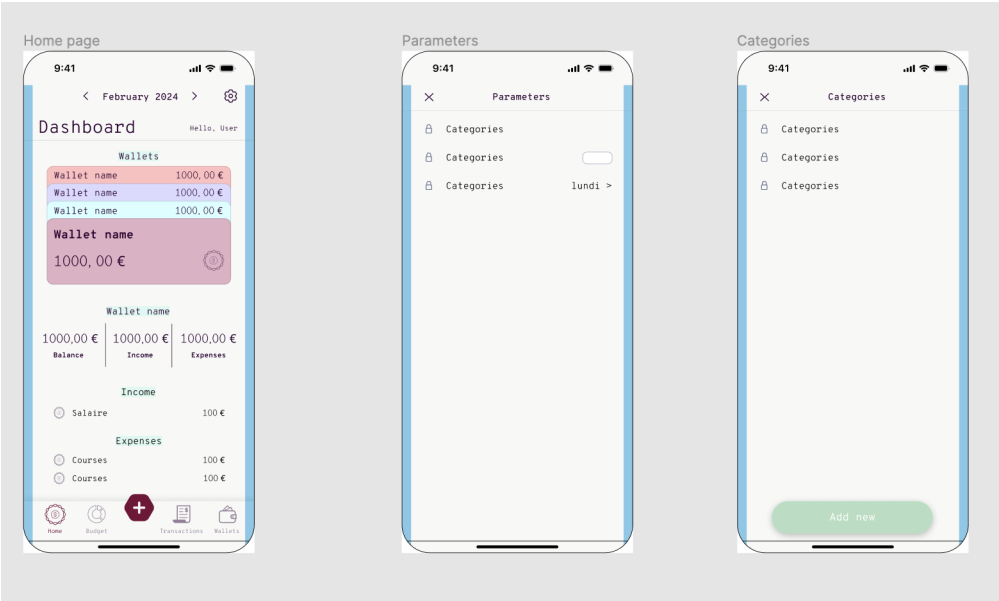


FIGURE 3 – Home

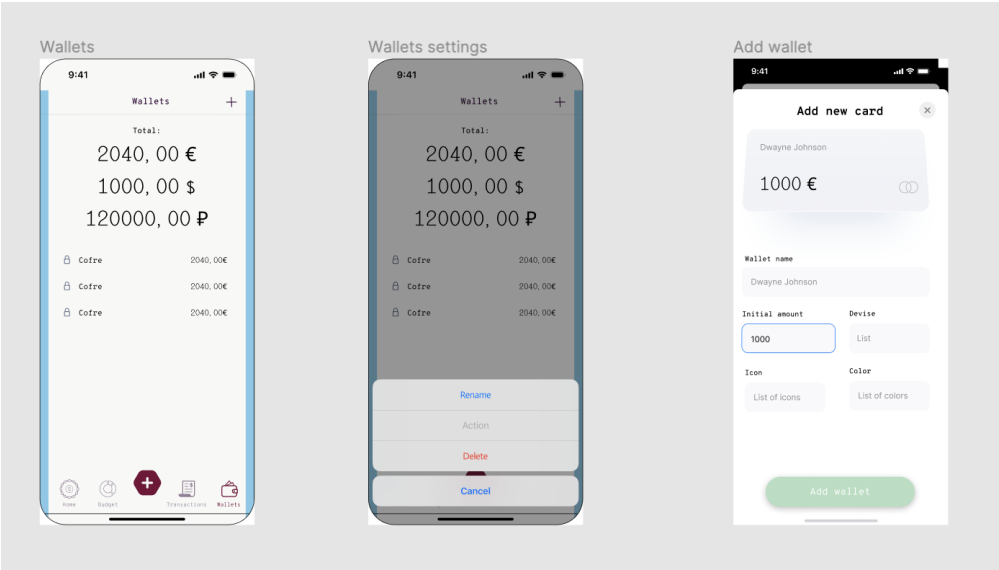


FIGURE 4 – Wallets

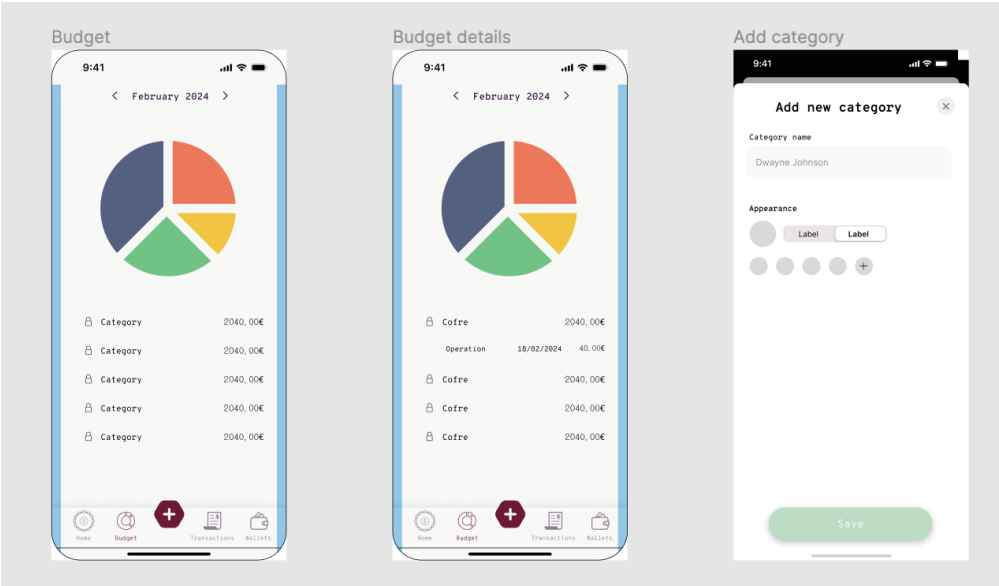


FIGURE 5 – Budget

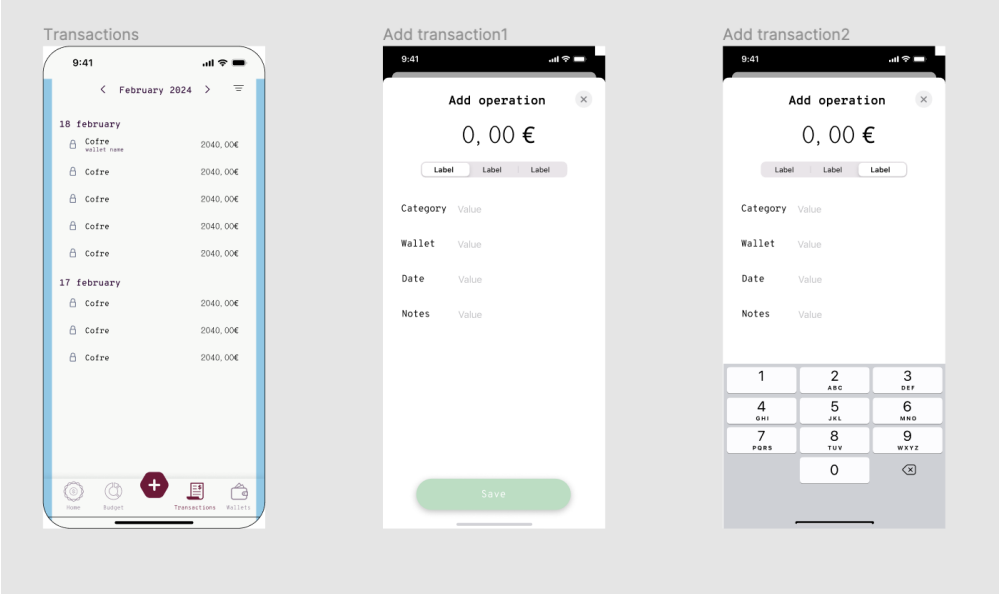


FIGURE 6 – Enter Caption

Chaque wireframe était accompagné d'annotations décrivant les fonctionnalités spécifiques et les interactions utilisateur prévues. Cela a permis de clarifier les attentes et les exigences pour le développement ultérieur de l'interface utilisateur.

Les wireframes ont été essentiels pour aligner les attentes de conception avec les fonctionnalités souhaitées de l'application, tout en facilitant la communication avec les membres de l'équipe de développement. En intégrant les retours et les suggestions de l'équipe, j'ai pu affiner et améliorer les wireframes pour mieux répondre aux besoins des utilisateurs finaux.

# 5 Frontend

Pour la partie frontend de l'application SophistiSpend, j'ai choisi d'exploiter les capacités de SwiftUI couplées à l'architecture MVVM. SwiftUI, grâce à son approche déclarative, a permis de construire des interfaces utilisateurs fluides et réactives avec moins de code et une plus grande lisibilité.

L'adoption du pattern MVVM (Modèle-Vue-ViewModel) était une décision naturelle pour travailler avec SwiftUI. Cette architecture favorise une séparation nette entre la présentation visuelle et la logique métier, facilitant ainsi le développement, les tests, et la maintenance du code. Le ViewModel agit comme un intermédiaire entre la Vue et le Modèle, gérant la logique d'affichage et les interactions, ce qui permet de garder la couche UI propre et concentrée sur la présentation.

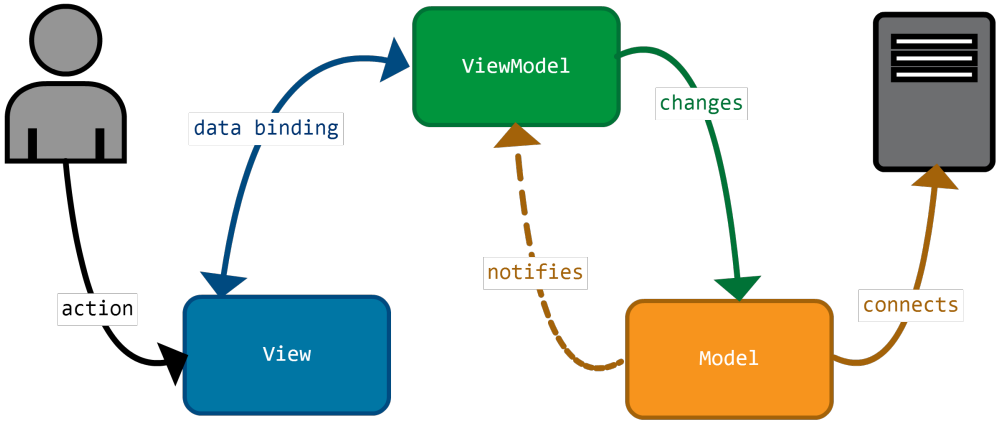


FIGURE 7 – Enter Caption

Pour interagir avec le backend, j'ai utilisé des appels réseau REST API. SwiftUI offre des fonctionnalités intégrées pour effectuer des requêtes réseau et traiter les réponses JSON de manière efficace.

# 6 Backend

Dans la conception du backend de mon projet, j'ai choisi d'utiliser une architecture basée sur les structures de handlers, de services et de repositories, tout en implémentant le JWT (JSON Web Token) pour la gestion de l'authentification. Cette approche permet de séparer clairement les différentes responsabilités du backend et de favoriser la modularité, la réutilisabilité et la testabilité du code.

Les handlers sont des composants chargés de recevoir les requêtes HTTP et de les router vers les services appropriés. Ils servent également à formater les réponses renvoyées au client. En structurant les handlers de manière logique, j'ai pu organiser efficacement les points d'entrée de mon API et rendre le code plus maintenable.

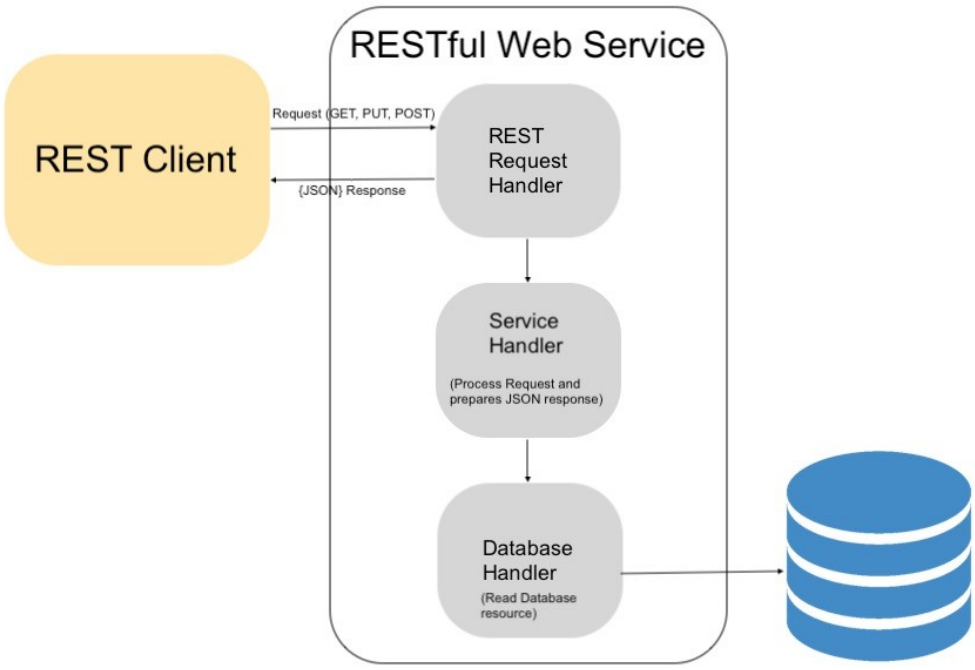


FIGURE 8 – Enter Caption

Les services sont des composants responsables de la logique métier de l'application. Ils effectuent les opérations nécessaires pour traiter les requêtes reçues par les handlers, en interagissant avec les repositories pour accéder aux données. En isolant la logique métier dans des services distincts, j'ai pu rendre mon backend plus modulaire et faciliter les tests unitaires.

Les repositories sont des composants chargés de l'accès aux données. Ils encapsulent la logique d'accès à la base de données et fournissent des méthodes pour effectuer des opérations de lecture, d'écriture, de mise à jour et de suppression sur les données. En utilisant des repositories, j'ai pu centraliser la gestion des données et réduire la duplication de code dans mon backend.

# 7 Conception et schéma de base de données

Dans la conception de la base de données pour mon application de gestion de budget, j'ai opté pour une approche relationnelle en utilisant PostgreSQL. Cette décision a été motivée par la nécessité de stocker des données structurées et interconnectées, telles que les transactions, les catégories de dépenses et les utilisateurs.

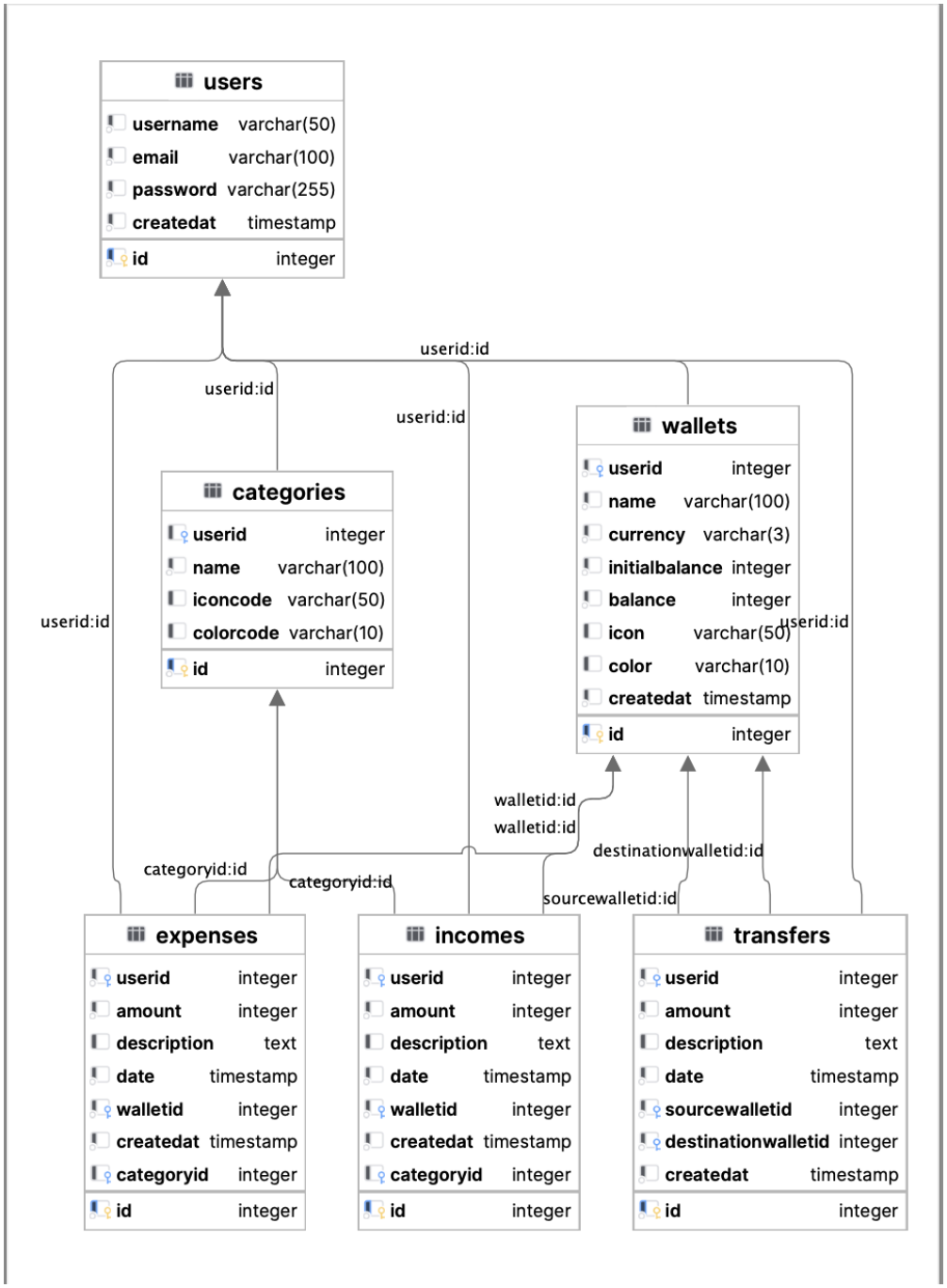


FIGURE 9 – Enter Caption

Le schéma de base de données se compose de plusieurs tables interdépendantes, chacune représentant une entité spécifique de l'application. Voici les principales tables utilisées dans la base de données :

1. **Users (Utilisateurs)** : Cette table stocke les informations relatives aux utilisateurs de l'application, telles que leurs identifiants, noms d'utilisateur et mots de passe (hashés). Cela permet de gérer l'authentification des utilisateurs et de sécuriser l'accès aux données.

2. **Wallets (Portefeuilles)** : Cette table contient les détails des portefeuilles financiers des utilisateurs, tels que leur nom, leur solde et leur devise. Chaque portefeuille est associé à un utilisateur spécifique, ce qui permet de suivre les finances individuelles de chaque utilisateur.

3. **Income (Revenus)** : Cette table enregistre les revenus financiers générés par les utilisateurs, tels que les salaires, les revenus de location, etc. Chaque revenu est associé à un portefeuille spécifique et peut inclure des informations telles que le montant, la date et la source.

4. **Expense (Dépenses)** : Cette table répertorie les dépenses effectuées par les utilisateurs, telles que les achats, les factures, etc. Chaque dépense est associée à un portefeuille spécifique et peut inclure des informations telles que le montant, la date et la catégorie de dépenses.

5. **Transfer (Transferts)** : Cette table enregistre les transferts de fonds effectués entre les portefeuilles des utilisateurs. Chaque transfert est associé à un portefeuille source et à un portefeuille de destination, ainsi qu'à des détails tels que le montant et la date.

6. **Categories (Catégories)** : Cette table répertorie les catégories de dépenses prédéfinies utilisées pour classer les transactions. Chaque catégorie peut avoir un nom et une couleur associés, ce qui permet aux utilisateurs de catégoriser leurs dépenses de manière efficace.

Dans la conception de la base de données pour mon application de gestion de budget, j'ai initialement choisi d'organiser les données en trois tables distinctes : `Revenus (Income), Dépenses (Expenses) et Transferts (Transfers)`. Cette approche semblait logique à première vue, car elle permettait de différencier clairement les différents types de transactions financières.

Cependant, au fur et à mesure que le développement avançait, j'ai rencontré quelques limitations avec cette structure. Par exemple, certaines opérations, telles que les remboursements de dépenses ou les virements entre différents types de comptes, devenaient difficiles à modéliser de manière efficace.

De plus, la gestion des rapports et des analyses financières devenait plus complexe en raison de la nécessité de fusionner et de traiter séparément les données provenant de différentes tables.

En rétrospective, je réalise que la création d'une table unique pour toutes les transactions financières aurait pu simplifier la modélisation des données et rendre les opérations de traitement et d'analyse plus fluides. Une approche basée sur une table unique `"Transactions"` aurait permis de regrouper toutes les opérations financières, quel que soit leur type, dans une seule entité.

Ainsi, dans les futures itérations de l'application, je prévois de migrer vers une structure de base de données basée sur une table unique `"Transactions"`, ce qui offrira une plus grande flexibilité et une meilleure extensibilité pour gérer les opérations financières de manière plus efficace.

# 8 Feedback

Dans le cadre de ce projet, j'ai acquis une expérience précieuse dans le développement d'applications iOS en utilisant le langage Swift et le framework SwiftUI. Voici quelques-uns des principaux points de rétroaction que j'ai recueillis tout au long du processus :

1. **Apprentissage de Swift et SwiftUI** : Travailler avec Swift et SwiftUI a été une expérience enrichissante. J'ai trouvé que Swift était un langage élégant et expressif, offrant de nombreuses fonctionnalités modernes pour le développement d'applications. De même, SwiftUI a simplifié le processus de création d'interfaces utilisateur dynamiques et interactives, grâce à sa syntaxe déclarative et à sa prise en charge des fonctionnalités telles que la gestion de l'état et la liaison de données.

2. **Architecture MVVM** : L'utilisation de l'architecture Modèle-Vue-VueModèle (MVVM) s'est avérée être un choix judicieux pour structurer mon application. La séparation claire des préoccupations entre les modèles, les vues et les vues-modèles a permis une meilleure organisation du code et une facilité de maintenance. Cela m'a également aidé à rendre mon code plus testable et évolutif.

3. **Développement Backend en Go** : Le choix de Go pour le développement du backend a été motivé par sa performance, sa concurrence native et sa syntaxe simple. Bien que j'aie déjà une certaine expérience avec Go, développer un backend complet pour une application iOS était un nouveau défi. J'ai trouvé que Go était bien adapté à cette tâche, offrant une prise en charge native des routines de serveur HTTP et une facilité d'intégration avec d'autres bibliothèques et frameworks.

4. **Gestion de Base de Données avec PostgreSQL** : Utiliser PostgreSQL pour la gestion des données a été une expérience positive. Sa prise en charge des transactions ACID, des requêtes complexes et des contraintes de clé étrangère a facilité la modélisation et la gestion des données financières de l'application. Cependant, la conception initiale de la base de données basée sur les tables Income, Expenses et Transfers s'est avérée limitée et peu flexible dans la gestion des transactions. Cela m'a conduit à envisager une approche alternative avec des tables de transactions distinctes pour une meilleure extensibilité et un traitement plus efficace des données.

# 9 Améliorations Futures et Conclusion

Dans le cadre de ce projet, plusieurs améliorations et évolutions futures peuvent être envisagées pour améliorer l'application SophistiSpend et enrichir l'expérience utilisateur. Voici quelques-unes des améliorations potentielles :

**Ajout de fonctionnalités supplémentaires** : L'application pourrait être étendue pour inclure des fonctionnalités telles que la planification de budget, la génération de rapports financiers, et la synchronisation des données entre les appareils. Ces fonctionnalités ajoutées permettraient aux utilisateurs de mieux gérer leurs finances et de suivre leurs dépenses de manière plus efficace.

**Amélioration de l'interface utilisateur** : L'interface utilisateur de l'application pourrait être améliorée pour offrir une expérience plus fluide et intuitive. Cela pourrait impliquer la refonte des éléments d'interface utilisateur, l'ajustement de la disposition et de la navigation, ainsi que l'ajout d'animations et de transitions pour rendre l'application plus attrayante visuellement.

En conclusion, ce projet m'a permis de plonger dans le monde passionnant du développement d'applications iOS en utilisant Swift et SwiftUI. J'ai pu explorer de nouvelles technologies, relever des défis techniques et renforcer mes compétences en développement logiciel. Bien que l'application SophistiSpend soit encore en phase de développement, je suis confiant dans sa capacité à évoluer et à s'améliorer grâce à ces améliorations futures. Je suis reconnaissant pour cette opportunité d'apprentissage et je suis impatient de poursuivre ma progression dans le domaine du développement d'applications mobiles.