МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
__ _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна система контролю фінансів»
здобувача групи ІН-06-2 – Боровика Олександра Олеговича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Олександр БОРОВИК
(підпис)

Керівник, доцент,
кандидат фізико-математичних наук,
доцент Галина ОЛЕКСІЄНКО _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-06-2 Боровика Олександра Олеговича

1. Тема роботи: «Інформаційна система контролю фінансів»
затверджую наказом по СумДУ від *«23» травня 2024 р. №0570-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналітичний огляд предметної області, аналіз аналогів та постановка задачі. 2) Огляд технологій для розробки інформаційної системи контролю фінансів. 3) Розробка інформаційної системи контролю фінансів. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____202_ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналітичний огляд предметної області, аналіз аналогів та постановка задачі*		
2	*Аналіз технологій для створення інформаційної системи фінансового контролю*		
3	*Розробка інформаційної системи контролю фінансів*		
4	*Тестування інформаційної системи контролю фінансів*		
5	*Аналіз результатів, оформлення пояснювальної записки*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 59 стр., 28 рис., 2 додатки, 21 використаних джерел.

Обґрунтування актуальності теми роботи — тема кваліфікаційної роботи є важливою та актуальною, оскільки спрямована на вирішення практичної проблеми управління особистими фінансами шляхом розробки моделей, макетів та інформаційної системи.

Об’єкт дослідження — процес управління особистими фінансами.

Предмет дослідження — система для контролю фінансів, що допомагає управляти та аналізувати витрати і доходи.

Мета роботи — створення інформаційної системи контролю фінансів, яка дозволяє зручно додавати транзакції витрат та надходжень, переглядати їх згідно хронологічного порядку та аналізувати їх.

Методи дослідження — аналіз існуючих інформаційних систем, моделювання системи, аналіз інструментів розробки, створення макету інтерфейсу, проектування бази даних та розробка програмної частини.

Результати — створено інформаційну систему контролю фінансів для додавання та редагування транзакцій та категорій, перегляду витрат та аналізу за місяць. Проведено аналіз актуальності системи та існуючих мобільних застосунків. Розроблено макет інтерфейсу, базу даних та програмну частину. Застосунок успішно пройшов тестування на iOS та Android.

ІНФОРМАЦІЙНА СИСТЕМА, КОНТРОЛЬ ФІНАНСІВ, МОБІЛЬНИЙ ЗАСТОСУНРОК, FLUTTER, SQLITE

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, управління фінансами є важливою частиною свідомого життя. Здатність ефективно керувати ресурсами часто визначає різницю між розвитком і стагнацією. Фінансова грамотність, розуміння того, як працюють гроші в різних сферах, стала не просто бажаною, а необхідною навичкою [1]. Зважаючи на велику кількість транзакцій, які відбуваються щодня, від дрібних покупок до великих інвестицій, для людей важливо мати необхідну інформацію про свої витрати та бути свідомими у своїх фінансових рішеннях.

Об'єкт дослідження. Процес керування фінансовими ресурсами та розробка інструментів для їх контролю.

Предмет дослідження. Розробка та впровадження інформаційної системи для контролю фінансів, яка допоможе користувачам усвідомлено керувати та аналізувати власні витрати та надходження.

Гіпотеза. Використання інформаційної системи контролю фінансів сприятиме покращенню фінансової грамотності користувачів та допоможе їм ефективніше керувати власними фінансами.

Новизна. Комбінація функціональності інформаційної системи контролю фінансів з акцентом на простоту використання та інтуїтивність інтерфейсу, що дозволить користувачам легко керувати та аналізувати власні витрати та надходження.

Структура. Робота складається з вступу, аналітичного огляду, вибору методів розв’язання задачі, реалізації, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Актуальність системи контролю фінансів

Кожного дня користувачів смартфонів стає все більше [4] і разом з цим більшість компаній, що надають послуги та продають товари, намагаються привернути увагу до свого продукту. Можна впевнено сказати що цифровізація послуг та продуктів змінює життя кожного з нас. Здається, що тільки вчора потрібно було йти у відділення за довідкою про місце проживання, а сьогодні це вже можна зробити з мобільного застосунку за лічені хвилини. Хоч може здаватись, що в електронний вигляд перенесено все що тільки можна уявити, але кожного дня додаються нові застосунки та функції.

Багато змін, що відбулися в останні роки, пов’язані саме з наявністю смартфонів та їх застосунками. Щоб вже існуючі послуги змогли конкурувати в сучасних умовах, необхідно було адаптувати їх до змін. Наразі вже важко уявити ресторан, в якому не буде електронного меню, сторінки в соціальних мережах чи сайту, де можна залишити відгук, подивитись де вони знаходяться та дізнатись час їх роботи. Також з’явились і зовсім нові сервіси, щоб задовільнити потреби – послуги з кібербезпеки, VPN-сервіси, крипто валюта та багато іншого.

Категорія фінансових застосунків є одна з найважливіших та найпопулярніших серед користувачів смартфонів [5]. Застосунки, що дозволяють взаємодіяти з власними грошима, робити платежі, переводити кошти, відстежувати свої витрати та доходи, стали невід'ємною частиною життя багатьох людей.

Основні можливості фінансових застосунків:

1) Особисті фінанси: застосунки для контролю фінансів допомагають планувати власний бюджет. Користувачі можуть вносити свої щоденні

витрати та доходи до відповідної категорії та отримувати детальний аналіз своїх фінансів.

2) Банківські послуги: більшість банків мають власні мобільні застосунки для здійснення платежів, переказів грошей, перегляду балансу та історії транзакцій. Наразі вже існують банки у яких не має фізичних відділень, що дозволяє зменшити витрати та мати конкурентні умови.

3) Інвестиції: застосунки, що дозволяють користувачам інвестувати гроші в акції, облігації або криптовалюту, надаючи інструменти для аналізу ринку та вибору стратегії інвестування.

4) Валютні операції: застосунки для конвертації валют, перевірки актуальних курсів, а також здійснення міжнародних переводів.

5) Податки та рахунки: застосунки, які спрощують процес сплати податків або ведення обліку для підприємців.

Для забезпечення захисту інформації користувача, більшість фінансових застосунків, що мають доступ до особистої інформації та грошей власника, використовують двофакторну автентифікацію [6].

Інтеграція з іншими застосунками та сервісами є ще однією важливою характеристикою. Наприклад, застосунки для контролю фінансів можуть синхронізуватися з банківськими рахунками користувача, що автоматично заносить нові транзакції до потрібного сервісу.

Зростаюча популярність фінансових мобільних застосунків спонукає розробників постійно вдосконалювати їх, пропонуючи нові функції та можливості для користувачів. На ринку існує велика конкуренція [7], тому ключем до успіху є зручний інтерфейс, висока безпека даних та інноваційні рішення.

1.2 Аналіз існуючих мобільних застосунків

Щоб повноцінно проаналізувати та дослідити існуючі аналоги мобільних застосунків для керування фінансами, треба протягом деякого часу їх використовувати та визначити основні функції, переваги та недоліки.

Для аналізу були вибрані три застосунки, які були найпопулярніші за запитом «контроль фінансів» [8], а саме: «Money manager & expenses» [9], «Wallet: Budget Planner Tracker» [10] та «Monefy – Budget & Expenses app» [11], які зображено на рисунку 1.1.

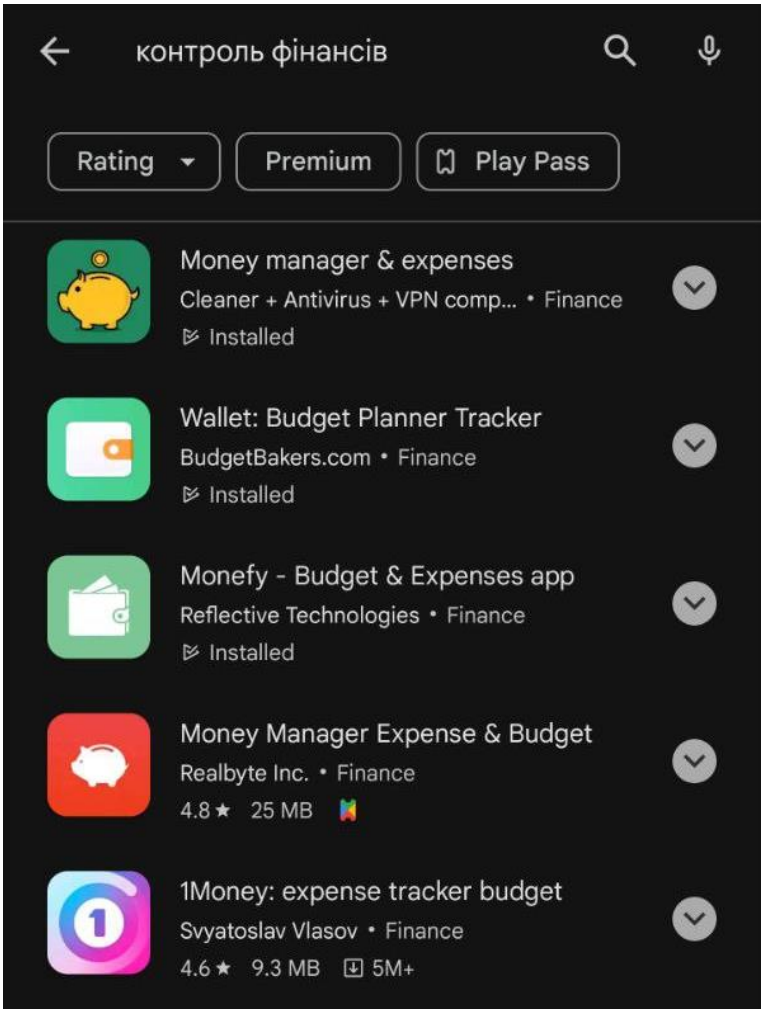

Рисунок 1.1 – Результат пошуку

- Money manager & expenses

Першим застосунком у пошуковій видачі є «Money manager & expenses». Він має більше 5 мільйонів завантажень і оцінку 4.9 з 5. Застосунок є безкоштовним для встановлення і використання, але може містити рекламу та платні функції. Запустивши застосунок не обов’язково авторизуватись і його можна використовувати без акаунта. Темне оформлення автоматично підтягнулась з налаштувань смартфону. З основних відмінностей також можна

зазначити приємний дизайн (рис. 1.2). Важливо відмітити, що навігація по застосунку та пошук потрібних функцій потребують багато часу і випадкові свайпи чи кліки можуть спричинити не потрібний результат.

Основні функції:

- Створення та редагування категорій;
- Додавання витрат та надходжень;
- Перегляд транзакцій за певний період;
- Формування діаграм.

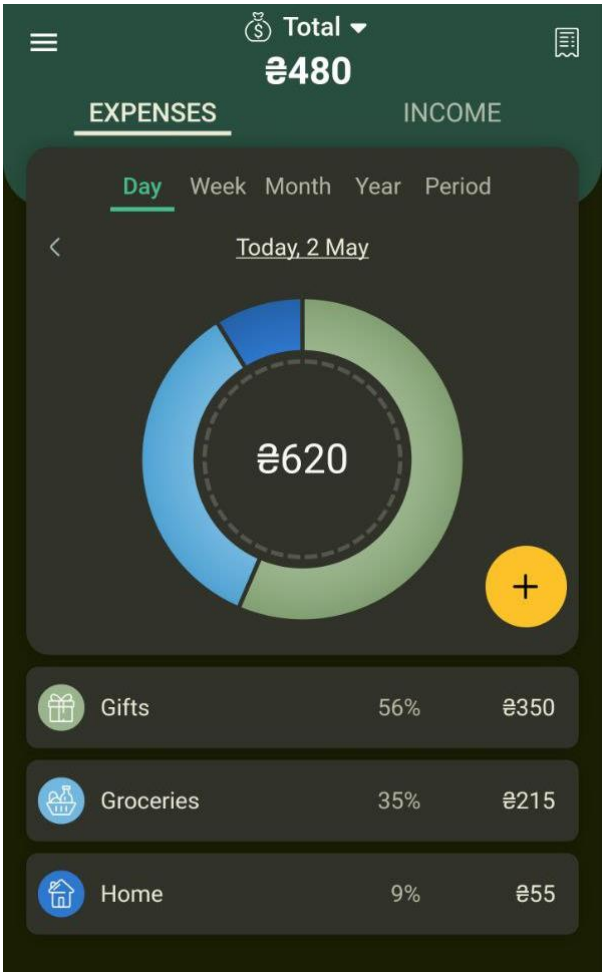

Рисунок 1.2 – Головний екран «Money manager & expenses»

- Wallet: Budget Planner Tracker

Другим застосунком у пошуковій видачі є «Wallet: Budget Planner Tracker». Він має більше 5 мільйонів завантажень і оцінку 4.7 з 5. Застосунок є безкоштовним для встановлення і використання, але має платні функції.

Вперше зайшовши у застосунок необхідно авторизуватись, без акаунту не можливе його використання. Оформлення та зовнішній вигляд можна описати, як перенавантажений та застарілий (рис. 1.3). Серед розглянутих застосунків, це єдиний, де анімації працювали з помітним відставанням. Користуючись застосунком декілька днів, часто стикався з тим, що важко знайти, а потім і відкрити потрібний розділ.

Основні функції:

- Створення та редагування категорій;
- Додавання витрат та надходжень;
- Створення бюджету та цілей;
- Наявність гаманців.

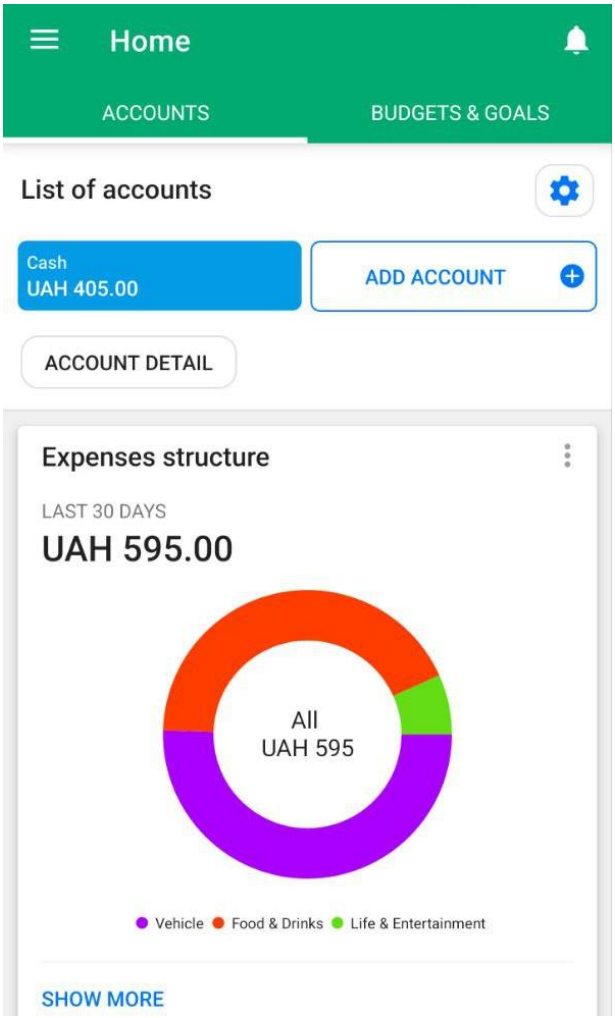

Рисунок 1.3 – Головний екран «Wallet: Budget Planner Tracker»

- Monefy – Budget & Expenses app

Третім застосунком у пошуковій видачі є «Monefy – Budget & Expenses app». Він має більше 5 мільйонів завантажень і оцінку 4.1 з 5. Застосунок є безкоштовним для встановлення і використання, але може містити рекламу та має платні функції. Найменше проблем було саме з використанням даного застосунку, що досягнуто завдяки мінімалістичному інтерфейсу, можливості використання без авторизації та зрозумілому функціоналу (рис. 1.4). Хоч обраний застосунок має найменше функцій серед зазначених вище, але концепція простого інтерфейсу, який не перенавантажений функціоналом є дуже важливим.

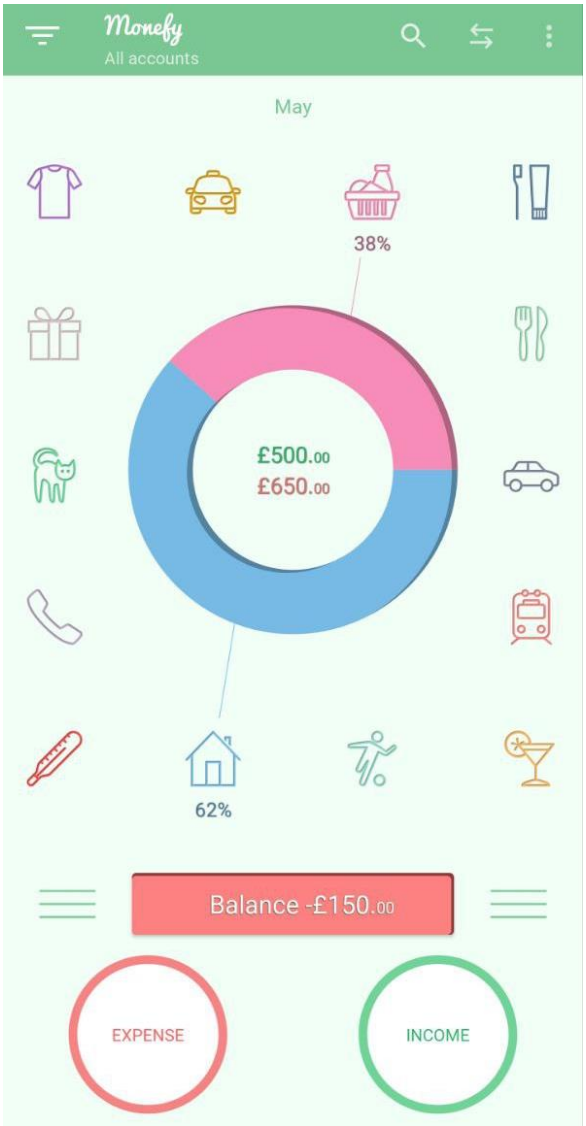

Рисунок 1.4 – Головний екран «Monefy – Budget & Expenses app»

Основні функції:

- Створення та редагування категорій;
- Додавання витрат та надходжень;
- Формування діаграми витрат та надходжень.

Провівши аналіз найпопулярніших застосунків для керування фінансів було визначено їх основні переваги та недоліки. Для наглядного порівняння дані зображено в таблиці 1.1.

Таблиця 1.1 – Основні переваги та недоліки популярних аналогів

	Money manager & expenses	Wallet: Budget Planner Tracker	Monefy – Budget & Expenses app
Використання без реєстрації	+	-	+
Зручний та зрозумілий інтерфейс	-	-	+
Сучасний дизайн	+	-	-
Плавність роботи	+	-	+
Відсутність платних обмежень	-	-	-
Перегляд витрат	+	+	+
Формування аналітики за певний період	+	+	-

Згідно проведеного аналізу застосунків, можна зробити висновок, що кожен має свої переваги, але їх недоліки інколи є критичними для повсякденного використання. Саме відштовхуючись від основних переваг та недоліків, прийнято рішення створити мобільний застосунок керування фінансами. Основними критеріями для власної розробки, можна відмітити: легкість у використанні, зрозумілий інтерфейс, сучасний дизайн та плавність роботи.

1.3 Постановка задачі

Згідно проведеного дослідження предметної галузі, задача розробки мобільного застосунку для контролю фінансів є актуальною та важливою для багатьох користувачів в сучасному світі. Метою даного проекту є створення мобільного застосунку, який дозволить користувачам ефективно відстежувати власні фінанси та аналізувати витрати.

На основі розглянутої інформації буде розроблений мобільний застосунок для контролю фінансів:

1) Створення структурно-функціональної діаграми
2) Створення діаграми варіантів використання
3) Вибір інструментів розробки
4) Вибір середовища розробки
5) Створення макету інтерфейсу
6) Програмна реалізація мобільного застосунку
7) Тестування мобільного застосунку

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Структурно-функціональне моделювання

Щоб описати інформаційну систему, визначити її основні процеси, дані для входу та кінцевий результат, була обрана методологія IDEF0 [12].

IDEF0 – це одна з найбільш зручних та зрозумілих методологій моделювання процесів. Процеси зображають у вигляді прямокутників, в які входять стрілки з певними ресурсами та даними і справа виходять з очікуваним результатом. У ліву грань прямокутника мають входити стрілки з даними, які змінюються та використовуються під час процесу. З правої грані прямокутника йдуть стрілки з результатом виконання процесу. Згори у прямокутник входять стрілки з конкретними правилами та обмеженнями щодо виконання. У нижню грань прямокутника входять стрілки з ресурсами, які потрібні для виконання процесу. Для більш точного опису процесів, використовується декомпозиція, де на окремій діаграмі зображають детальний

Починати моделювання потрібно з головного процесу. У нашому випадку це «Контролювання фінансів» (рис. 2.1). Для виконання процесу з контролювання фінансів необхідно мати наступну інформацію: назву категорії, вид транзакції (надходження чи витрата), кількість коштів, нотатки або коментарі та дата транзакції. Створення категорій, до яких відносяться транзакції, потребують шаблон редагування, в якому є конкретний перелік потрібної інформації та критерії її перевірки. Щоб додати транзакцію також треба шаблон його створення. Необхідними частинами процесу є користувач, який буде заповнювати необхідну інформацію і отримувати результат, а також мобільний застосунок. Як результат отримуємо список та аналіз транзакцій.

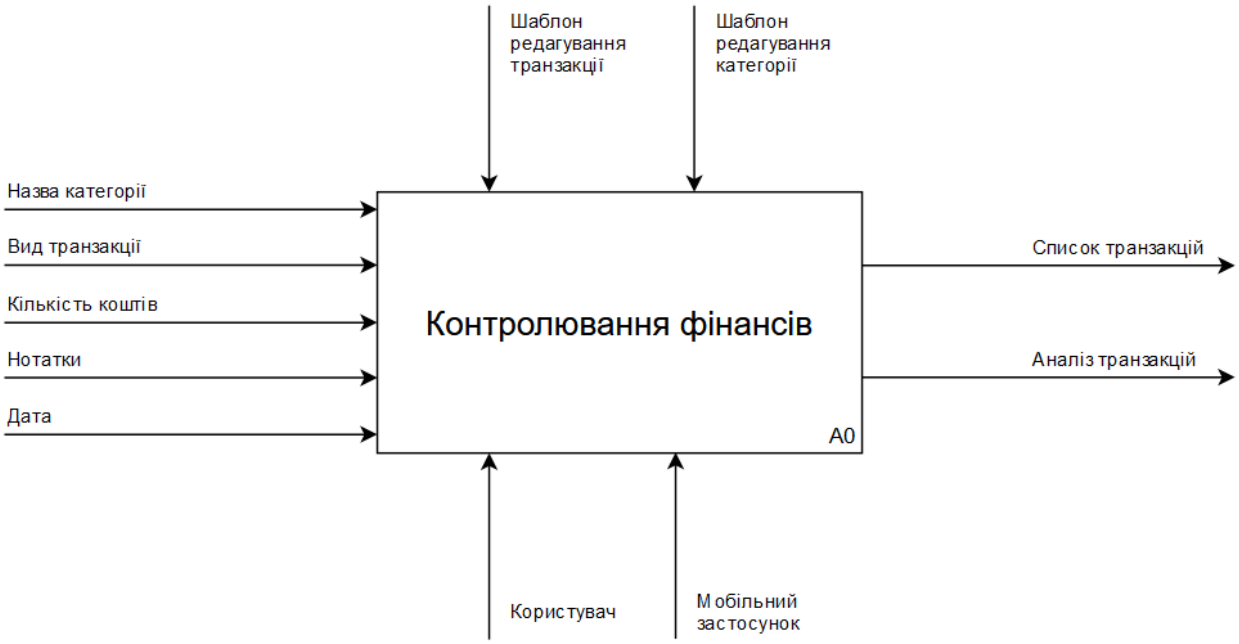

Рисунок 2.1 – Контекстна діаграма

Щоб більш детально розглянути процес роботи інформаційної системи з контролю фінансів потрібно зробити декомпозицію головного процесу (рис. 2.2). З декомпозиції можна побачити, що головний процес було поділено на чотири, а саме: створення категорії, створення транзакції, формування списку транзакцій і формування статистика транзакцій. Згідно IDEF0 методології дані, що використовуються системою, залишилися у тому ж обсязі та переліку, що і в головному процесу.

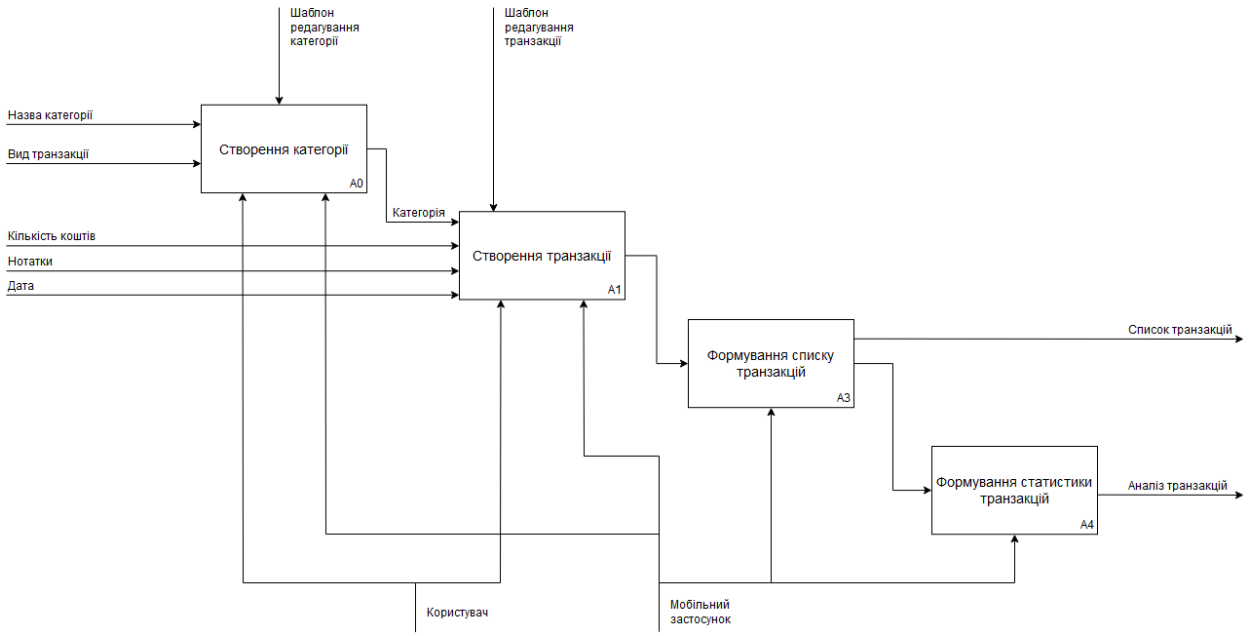

Рисунок 2.2 – Декомпозиція процесу «Контролювання фінансів»

2.2 Моделювання варіантів використання

Невід’ємна частина моделювання інформаційної система є моделювання варіантів використання [13] цієї системи користувачем. Щоб змоделювати варіанти використання, треба зрозуміти, у якій послідовності та з якими цілями користувач буде використовувати створену інформаційну систему.

Згідно з розглянутими аналогами можна виділити декілька основних варіантів використання та їх послідовності:

1) Створення категорії;
2) Створення транзакції відповідно до категорії;
3) Перегляд транзакцій за певний місяць;
4) Перегляд аналітики, що була зроблена згідно доданим транзакціям.

Для наглядного зображення було створено діаграму (рис. 2.3), на якій зображено: користувача, мобільний застосунок для контролю фінансів та їх взаємодію.

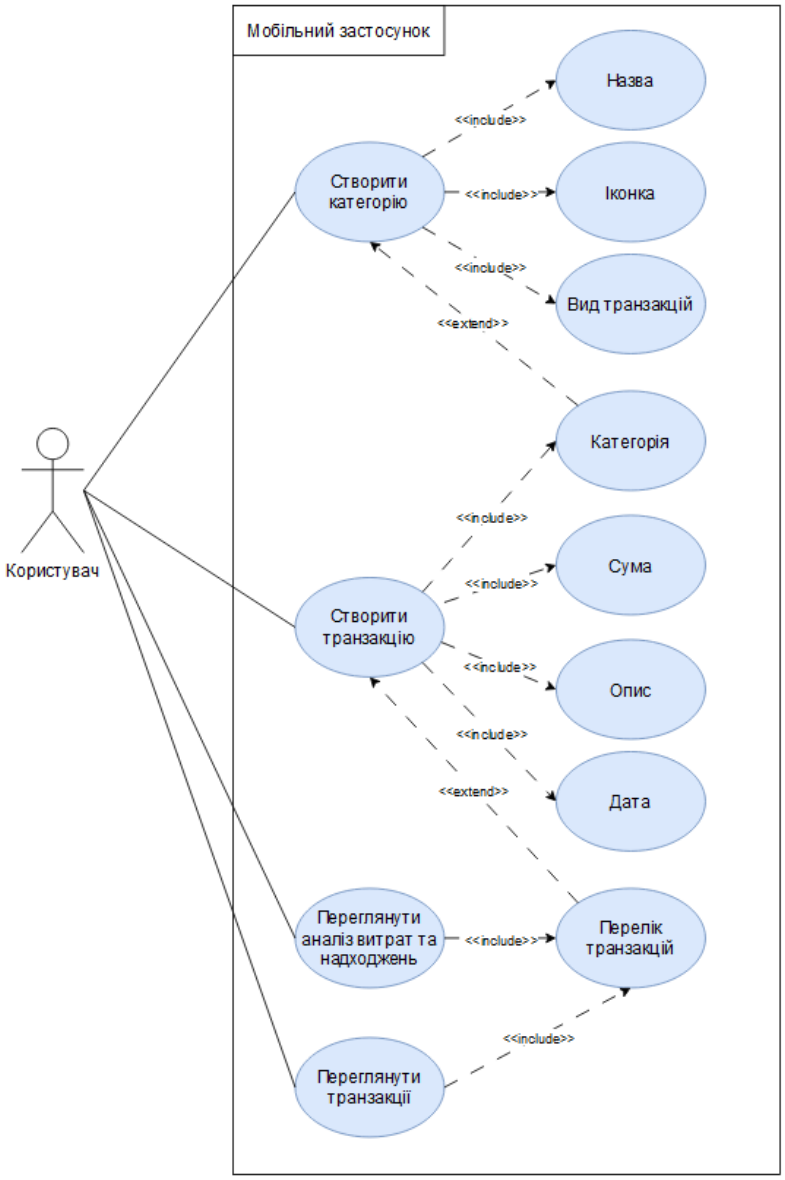

Рисунок 2.3 – Діаграма варіантів використання

2.3 Інструменти розробки

2.3.1 Фреймворк та мова програмування

Згідно аналізу предметної області та моделювання системи, необхідно обрати інструменти, які будуть підходити для розробки мобільного застосунку. Наразі є дві основні операційні системи для смартфонів, а саме Android та iOS, вони займають більше 99% від загального ринку смартфонів [14], тому орієнтуватись треба саме на ці операційні системи.

Для Android існує декілька нативних та популярних рішень, а саме використання Java та Kotlin для розробки застосунків, але ці рішення не підходять для iOS. Для iOS використовують найчастіше Swift та Objective-C. З нативних рішень немає інструментів, які можуть підійти для обох операційних систем. Ігнорувати одну з них теж не є правильним, бо частка Android є близько 71%, а iOS - 28% [14]. Розробляти два окремих застосунки набагато складніше, бо треба вміти програмувати на двох мовах та підтримувати в актуальному стані два застосунки.

Щоб вирішити питання з можливістю запуску застосунку на обох операційних системах, можна використати мультиплатформну мову програмування, таким чином, і знати треба лише одну мову програмування, і підтримувати в актуальному стані лише одну кодову базу.

На ринку мультиплатформних рішень для мобільних застосунків під Android та iOS є два основних конкуренти: Flutter [15] (рис. 2.4) та React Native [16] (рис. 2.5).

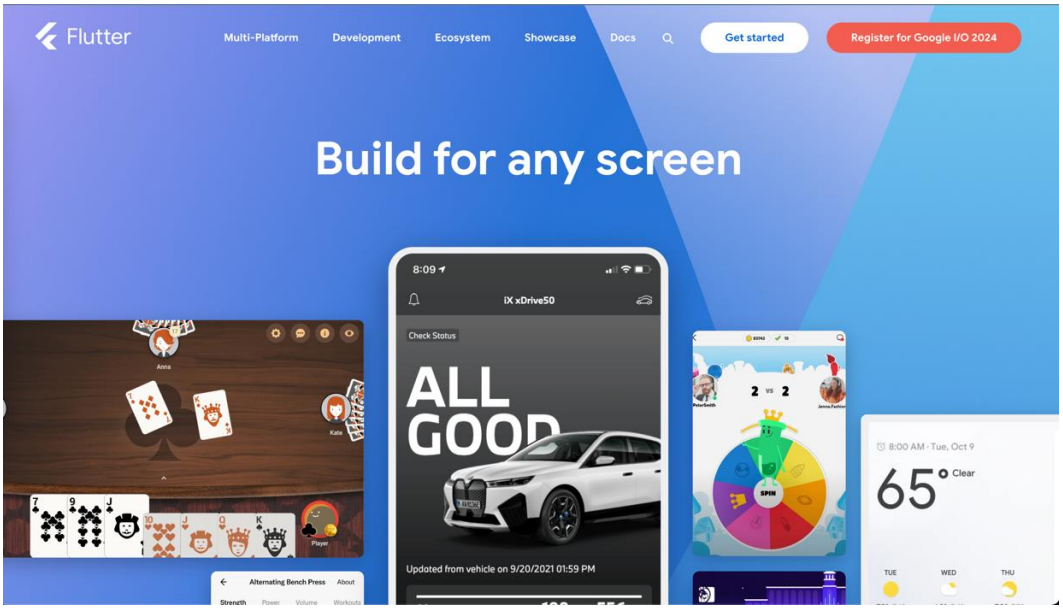

Рисунок 2.4 – Головна сторінка Flutter

Рисунок 2.5 – Головна сторінка React Native

Обираючи серед Flutter та React Native треба звертати увагу на розробників, що підтримують ці інструменти, їх сучасність, продуктивність та поширеність серед розробників.

Flutter – це фреймворк з відкритим програмним кодом, який створила і підтримує компанія Goolge. Мова програмування для даного фреймворку є Dart, що легкий у використанні, має всі потрібні інструменти і безпечний з точки зору типів даних та їх наявності.

React Native – це фреймворк з відкритим програмним кодом, який створила і підтримує компанія Facebook. Мова програмування для даного фреймворку є JavaScript, що найчастіше використовують саме для web. JavaScript дуже поширений, але його проблеми з динамічною типізацією і відсутності безпеки обов'язкової наявності змінних є дуже неприємними. Звісно зараз багато розробників використовують TypeScript разом з JavaScript, щоб нівелювати мінуси JavaScript, але це тільки ускладнює початковий вхід розробників, які не знайомі з даними інструментами.

Згідно критеріїв було обрано саме Flutter, бо компанія, що розробляє є Google, тобто розробники Android, що гарантує чудову інтеграцією. По сучасності та розвитку останніх років Flutter покращується набагато швидше,

у нього вже є вбудоване оформлення Material Design 3 та Cupertino. Також ком'юніті розробників у Flutter більше (рис. 2.6) та присутня офіційна підтримка зі сторони багатьох компаній.

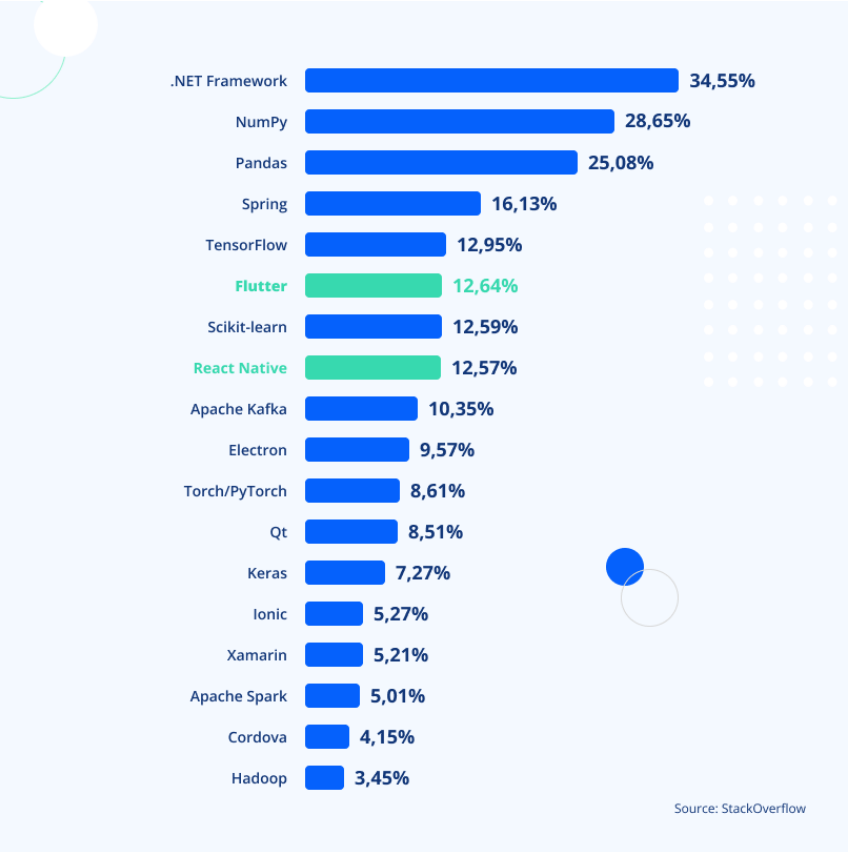

Рисунок 2.6 – Вибір фреймворку серед мультиплатформних рішень серед розробників

2.3.2 Система управління базами даних

Обираючи систему управління базами даних (СУБД), необхідно розглянути найкращі варіанти та перевірити їх на відповідність критеріями мобільного застосунку для контролю фінансів.

Згідно опитування StackOverflow за 2022 рік, що зображений на рисунку 2.7, можна виділити п'ять найпопулярніших рішень серед розробників, а саме: MySQL, PostgreSQL, SQLite, MongoDB, Microsoft SQL Server.

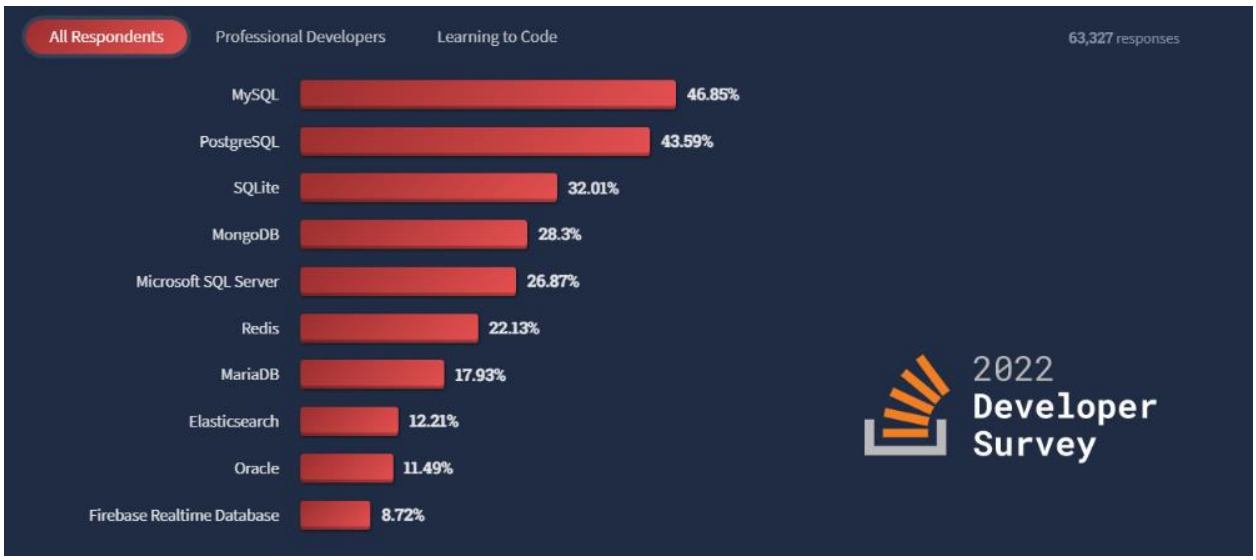

Рисунок 2.7 – Результати опитування розробників про використання СУБД

Для потреб мобільного застосунку з локальним збереженням даних, MySQL, PostgreSQL та Microsoft SQL Server є невиправдано складними рішеннями. Вони найчастіше застосовуються для збереження та керування великою кількістю даних та потребують додаткового налаштування для коректної роботи. Також вони мають велику кількість функціоналу, що не є доцільним для даного проекту.

Серед MongoDB та SQLite є суттєва різниця, бо MongoDB – це документо-орієнтована СУБД, а SQLite є реляційною СУБД. Як можна бачити на рисунку 2.7, більшість розробників використовують саме реляційні бази даних, бо їх легше розширити та оптимізувати.

Було обрано SQLite [17] (рис. 2.8), як один з легких та швидких рішень для мобільних застосунків, що зберігають дані на самому пристрої. Використовуючи SQLite мобільний застосунок створює файл з наповненням бази даних, який зберігається на пристрої. Для зберігання не великої кількості інформації та легкої оптимізації, SQLite чудово підходить для такого рішення.

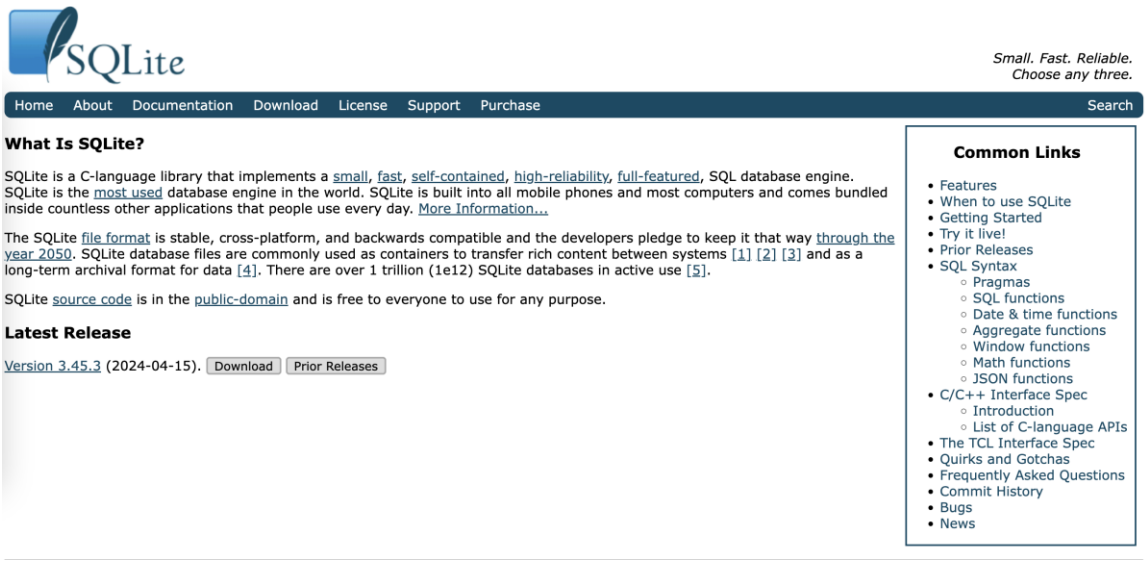

Рисунок 2.8 – Головна сторінка SQLite

2.3.3 Середовище розробки

Обираючи середовище розробки, потрібно враховувати, що розробка відбувається одразу для двох операційних систем, тому треба тестувати одразу, як на Android, так і на iOS. Серед існуючих варіантів для розробки на Flutter, є два середовища розробки: Visual Studio Code [18] та Android Studio [19].

Android Studio – має чудову підтримку та велику кількість функціоналу, але потребує багато ресурсів для роботи та погану оптимізацію. Для розробки краще підходить Visual Studio Code (рис. 2.9), бо займає не багато місця, гарно оптимізований під всі операційні системи та має великий набір плагінів, які можуть пришвидшити розробку. У Visual Studio Code можливе використання терміналу, дебаг консолі, Git та інших інструментів одразу з меню.

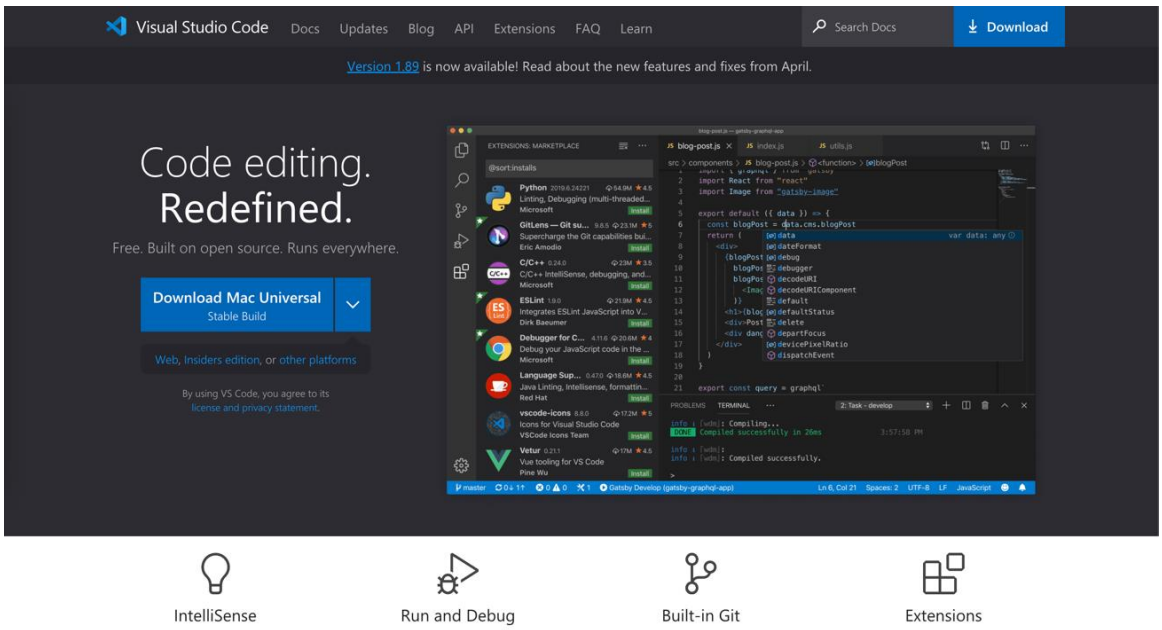

Рисунок 2.9 – Головна сторінка Visual Studio Code

Для тестування можна використати, або пристрої на Android та iOS, або емулятори для цих операційних систем. Щоб зробити емуляцію Android, необхідно встановити SDK для роботи емуляторів. Щоб зробити емуляцію iOS треба встановити XCode [20] та необхідні інструменти.

Для зручної розробки з декількох пристроїв та керування версіями було обрано Git [21]. Зручність використання Git з терміналу, інтеграції з іншими сервісами, такими як GitHub, GitLab та іншими не можливо переоцінити.

3 РЕАЛІЗАЦІЯ

3.1 Макет інтерфейсу

Реалізацію мобільного застосунку розпочинаємо з розробки макету інтерфейсу та необхідних сторінок. Для зручності у використання застосунок було поділено на чотири екрани: Categories, Transactions, Analytics та Settings. Навігація по застосунку відбувається за допомогою нижнього меню, де розташовані піктограми, які відповідають змісту та назві екрану. Вгорі кожного з екранів знаходиться назва екрану, в деякий екранах застосовується вільне місце справа для кнопки.

Categories – головний екран застосунку (рис. 3.1). При запуску застосунку користувач потрапляє саме на цей екран. На даному екрані виводиться таблиця з доданими користувачем категоріями. Категорії можуть бути, як для витрат, так і для надходжень, це налаштування обирає користувач при створенні та редагуванні категорії. Користувач може перемикати режими роботи даного екрану за допомогою кнопки, що розташована справа вгорі. Коли кнопка у вигляді піктограми збереження, то при користувач може додавати та редагувати категорії, а саме вказувати назву категорії, тип транзакцій, піктограму та колір піктограми. Назва категорії – це довільний текст, який вказує користувач. Тип транзакції – це надходження або витрати. Піктограма та її колір використовується для більш помітного та зручного формату зображення. Також користувач може змінювати позицію категорії серед інший, перенісши її у потрібне місце. Коли кнопка у вигляді піктограми олівця, то при користувач може додавати нові транзакції, натискаючи на потрібну категорію. При додаванні транзакції користувач може змінити потрібну категорію, вказати потрібну суму для транзакції, додати нотатки та дату і час транзакції.

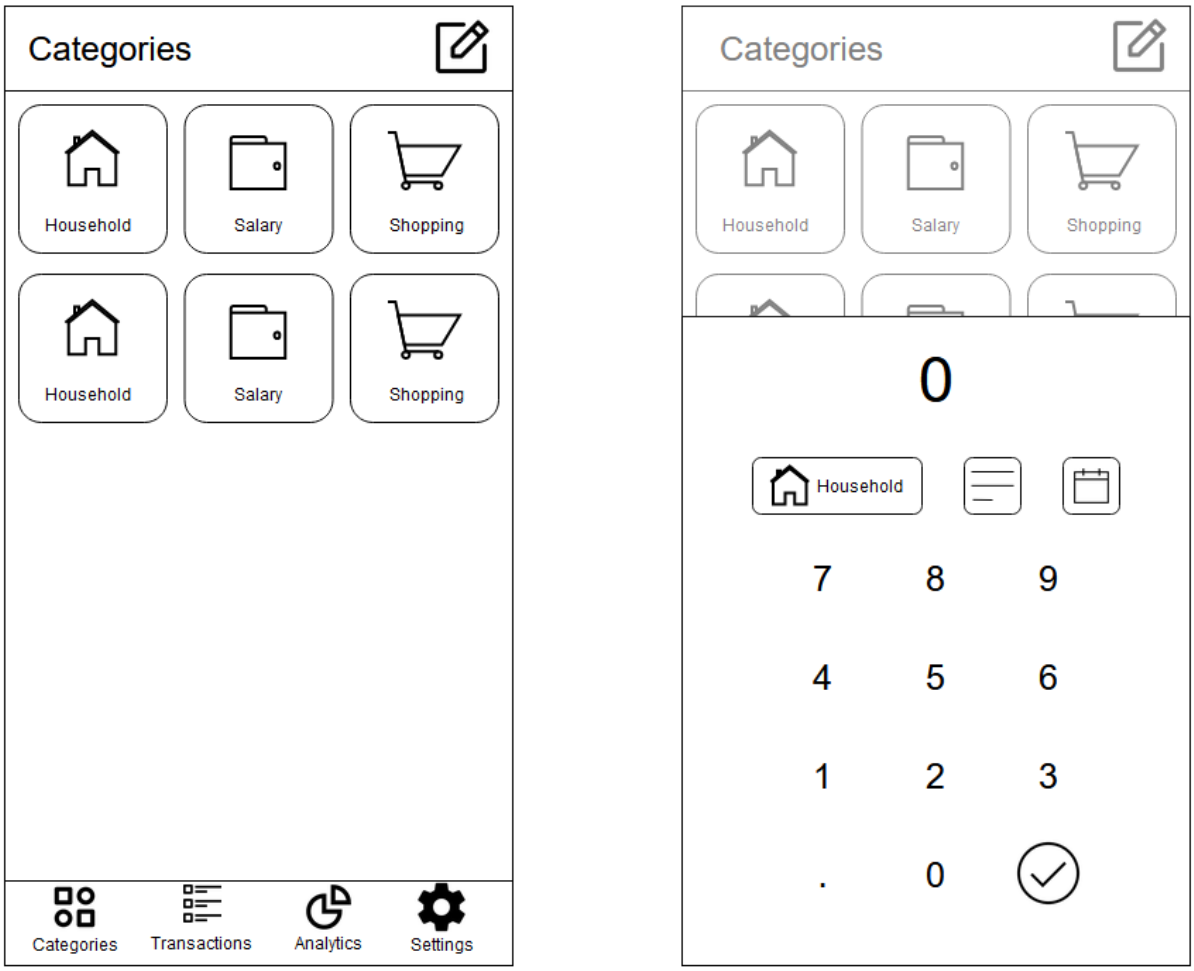

Рисунок 3.1 – Дизайн екрану «Categories»

Щоб мати можливість переглядати та редагувати транзакції, необхідний екран «Transactions», перейти на нього можна за допомогою меню навігації, що знаходиться знизу. При переході на даний екран, користувач бачить поточний місяць і список транзакцій за цей місяць, також транзакції мають бути відсортовані від найновіших до найстаріших. Користувач має змогу змінювати місяць на попередні та майбутні за допомогою кнопок вліво та вправо. Кожна транзакція у списку виглядає як картка з піктограмою, кольором та назвою категорії, нотатками, датою та вартістю транзакції. При натисканні на транзакцію, можна її відредагувати і змінити будь-яке значення. При свайпі транзакції справо у ліво, користувач отримає вікно з підтвердженням видалення. Якщо видалення підтверджено, транзакція буде видалена. Макет екрану «Transactions» зображено на рисунку 3.2.

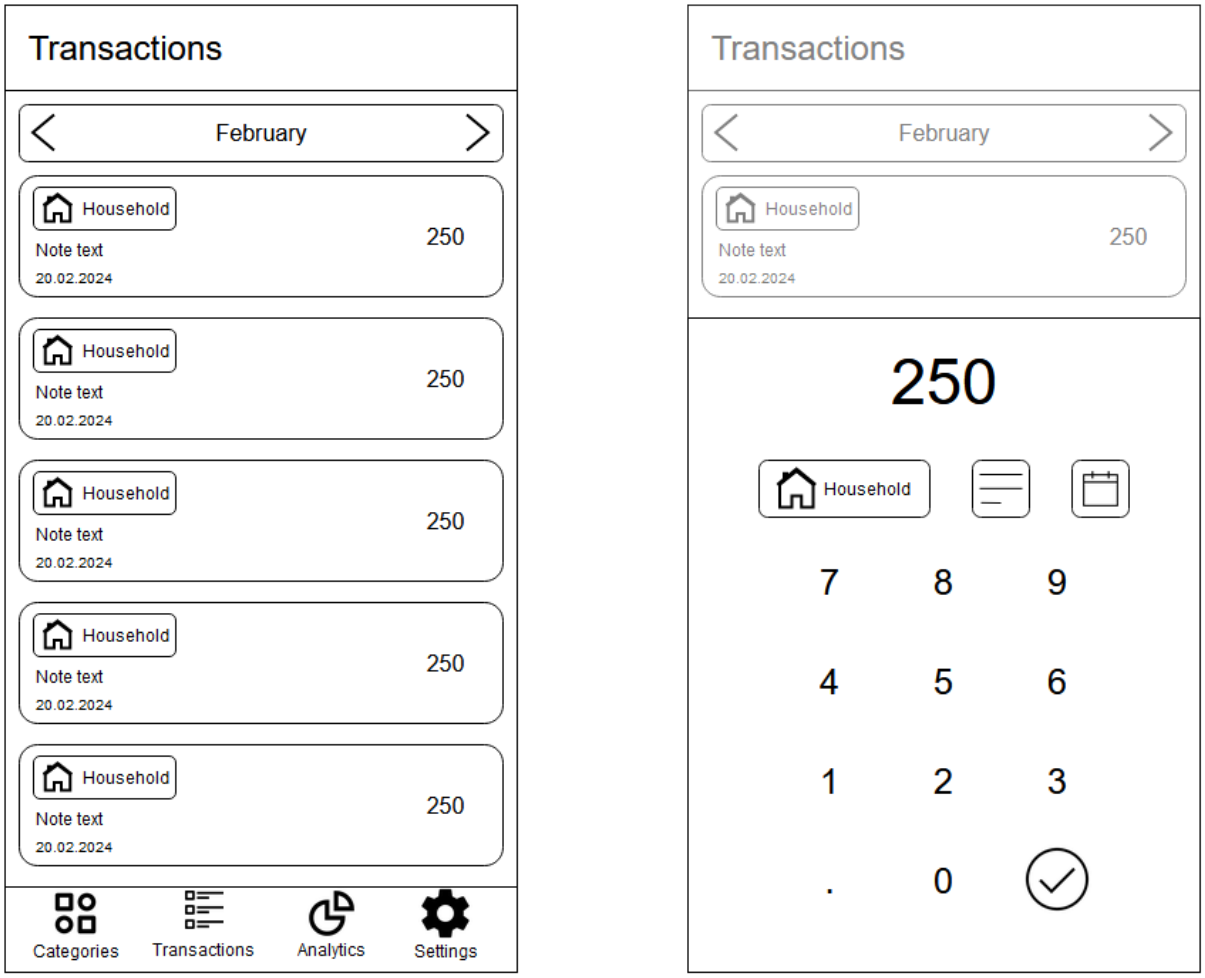

Рисунок 3.2 – Дизайн екрану «Transactions»

Для аналізу витрат та надходжень додано екран «Analytics» (рис. 3.2). Вгорі знаходиться назва місяця, за який проведено аналітику з можливістю перемикання на минулі та наступні місяці. Нижче знаходиться блок з сумою надходжень, витрати та різницею між ними. Також важливим елементом є список категорій та їх сумою. Список категорій відсортовано з спаданням суми.

Для налаштувань застосунку створений екран «Settings», в якому наразі є налаштування кольорового профілю застосунку. Є три варіанти цього налаштування: «Як на пристрої», світле і темне оформлення. Налаштування зберігаються на пристрої користувача.

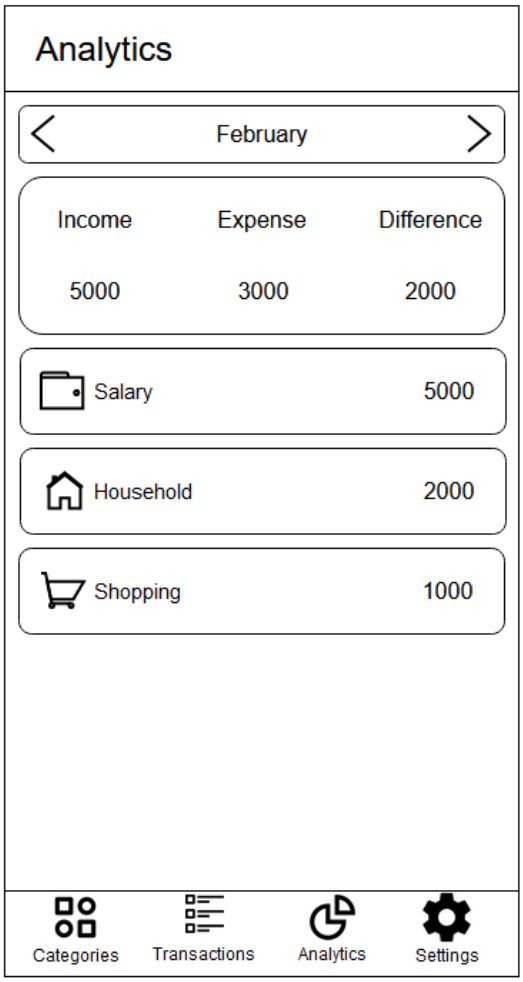

Рисунок 3.3 – Дизайн екрану «Analytics»

3.2 База даних

Для збереження даних, що вносить користувач до системи було вибрано СУБД SQLite. Проаналізувавши дані, які можуть використовуватись у мобільному застосунку, можна розділити їх на дві основних сутності: транзакції та категорії транзакцій.

Транзакція – одноразова операція, що додається користувачем. Транзакція включає у себе наступні дані: категорія, до якої належить, кількість коштів, дата транзакції та нотаток.

Категорія – сутність, що об’єднує та визначає тип транзакції, що їй належать. Категорія включає у себе наступні дані: назва, тип, код піктограми, колір, позиція у сітці та покажчик чи увімкнена категорія.

До однієї категорії може відноситись багато транзакцій, але їх наявність не є обов’язковою. У свою чергу, транзакція може належати тільки до однієї категорії і наявність категорії є обов’язкова. Таку залежність можна назвати «один до багатьох».

Модель даних та взаємодію транзакцій з категоріями зображено у вигляді UML діаграми на рисунку 3.4

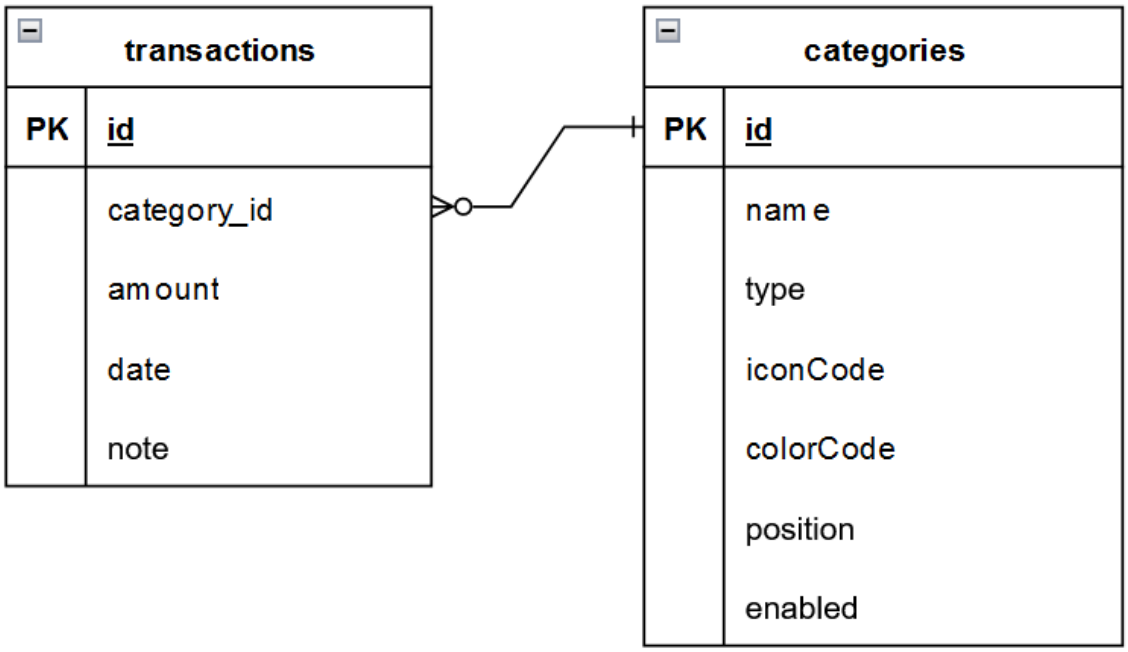

Рисунок 3.4 – UML діаграма бази даних

Для підтримання консистентності даних у таблицях, необхідно задати обмеження по типу та наявності даних. У таблиці 3.1 перелічені назви полів, типи даних та обмеження для таблиці «transactions».

Таблиця 3.1 – Типи даних та обмеження таблиці «transactions»

Назва	Тип даних	Обмеження
id	TEXT	PRIMARY KEY NOT NULL
categoryId	INTEGER	FOREIGN KEY NOT NULL
amount	TEXT	NOT NULL
date	TEXT	NOT NULL
note	TEXT	NOT NULL

У таблиці 3.2 перелічені назви полів, типи даних та обмеження для таблиці «categories».

Таблиця 3.2 – Типи даних та обмеження таблиці «categories»

Назва	Тип даних	Обмеження
id	TEXT	PRIMARY KEY NOT NULL
name	TEXT	NOT NULL
type	TEXT	NOT NULL
position	INTEGER	NOT NULL
enabled	INTEGER	NOT NULL DEFAULT 1
iconCode	INTEGER	NOT NULL
colorCode	INTEGER	NOT NULL

Згідно синтаксису SQLite, наведеним моделям та таблицям з типами даних та обмеженнями було написано код, що призначений для створення таблиць даних, код наведено нижче.

```
CREATE TABLE IF NOT EXISTS categories (
  id TEXT PRIMARY KEY NOT NULL,
  enabled INTEGER NOT NULL DEFAULT 1,
  iconCode INTEGER NOT NULL,
  colorCode INTEGER NOT NULL,
  name TEXT NOT NULL,
  type TEXT NOT NULL,
  position INTEGER NOT NULL
);

CREATE TABLE IF NOT EXISTS transactions (
  id TEXT PRIMARY KEY NOT NULL,
  amount INTEGER NOT NULL,
  categoryId TEXT NOT NULL,
  date TEXT NOT NULL,
  note TEXT,
  FOREIGN KEY (categoryId) REFERENCES categories (id)
);
```

3.3 Програмна реалізація

Для програмної реалізації мобільного додатку використовуємо мову програмування Dart, фреймфорк Flutter, середовище розробки Visual Studio code та стилю оформлення Material Design 3.

Щоб забезпечити можливість навігації по мобільному застосунку, було додана навігаційна панель, що знаходиться знизу, реалізована за допомогою NavigationBar та віджетів, що знаходяться в середині. У навігаційній панелі (рис. 3.5) перелічені всі необхідні користувачу екрани: Categories, Transactions, Analytics, Settings. Переходити між екранами користувач може натиснувши на потрібну піктограму або текст.

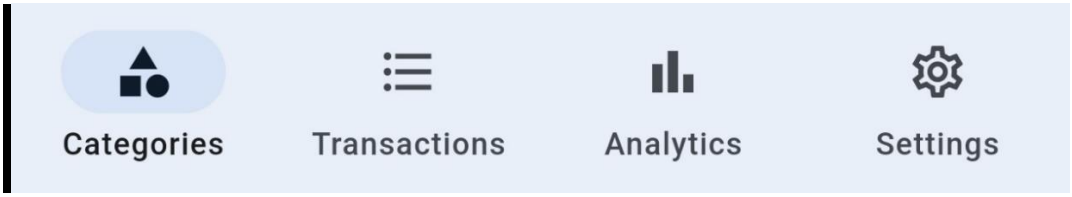

Рисунок 3.5 – Навігаційна панель

Головний екран мобільного застосунку є «Categories», який з’являється одразу після відкриття додатку і на ньому можна додавати нові транзакції та додавати і редагувати категорії, він розташований у файлі categories_page.dart. Для розташування категорій (рис. 3.6) було вибрано віджет DraggableGridView, що дозволяє не тільки розташовувати категорії у вигляді таблиці або решітки, а також переміщувати категорії у потрібне місце, коли користувач знаходиться у режимі редагування категорій. Блок з категорією реалізований за допомогою віджета Card, що закруглює краї, має гарний зовнішній вигляд та дозволяє оброблювати подію onClick.

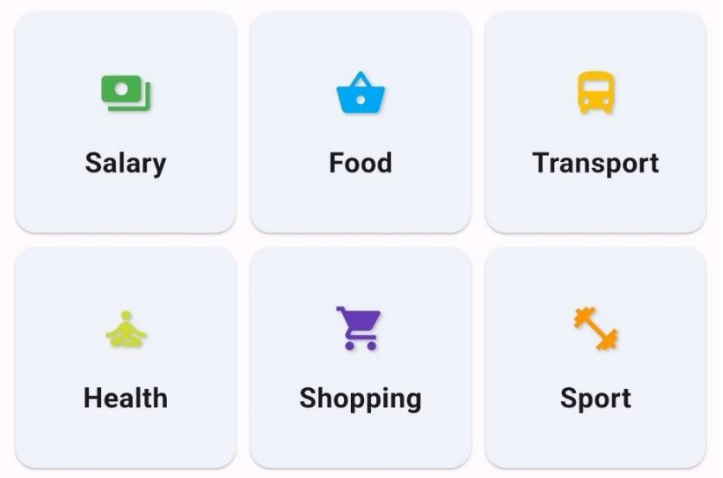

Рисунок 3.6 – Решітка з категоріями

Зміна режиму редагування категорій та додавання транзакцій реалізовані за допомогою кнопки, що розташовується вгорі і справа від назви екрану (рис. 3.7). Піктограма кнопки змінюється згідно режиму, також у режимі редагування категорій з’являється нова картка, де зображена піктограма додавання і текст «Add».

Рисунок 3.7 – Назва екрану та піктограма переходу у режим редагування

У режимі додавання транзакцій при натисканні на категорію, з’являється меню для редагування транзакції (рис. 3.8), що реалізовано за допомогою showModalBottomSheet та SaveTransactionWidget. У даному меню реалізовані всі необхідні елементи, а саме: клавіатура для вводу грошової кількості, кнопка для стирання останнього символу, кнопка вибору категорії, кнопка для введення нотатків та кнопка вибору дати з часом транзакції. Якщо користувач натисне за межами даного меню, то меню закриється і зміни не будуть прийняті. При натисканні на «галочку» транзакція збережеться. Вибір категорії відкриває додатковий попап, де перелічені всі категорії. Нотатки додаються у меню з довільним текстовим вводом. Меню вибору дати та часу реалізовано за допомогою вбудованих механізмів та викликаються за допомогою функцій showDatePicker та showTimePicker, що викликаються послідовно після вибору минулого. Валідація дій користувача відбувається при кожному кліку на клавіатуру, щоб не допустити не валідного значення. Якщо сума транзакції буде 0, то вона не буде зберігатись, бо не є транзакцією, як такою. Збереження суми транзакції відбувається у INTEGER типом даних, можливість вводити не цілі значення реалізовані за допомогою збереження суми множачи її на 100, а відображення користувачу, ділячи на 100. Саме через таку особливість, можливість збереження не цілих значень обмежена до сотих.

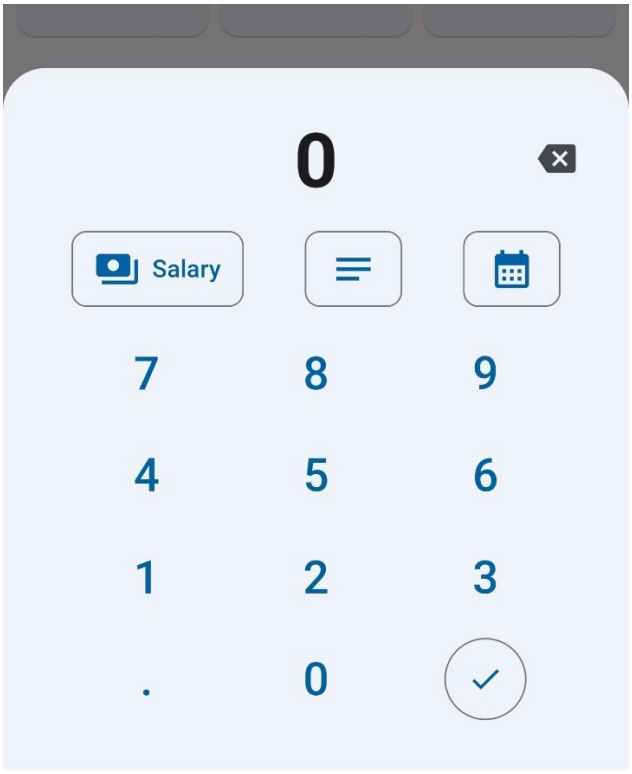

Рисунок 3.8 – Меню редагування транзакції

Редагування категорій реалізовано схожим до транзакцій способом, а саме відображенням меню з потрібними полями (рис. 3.9).

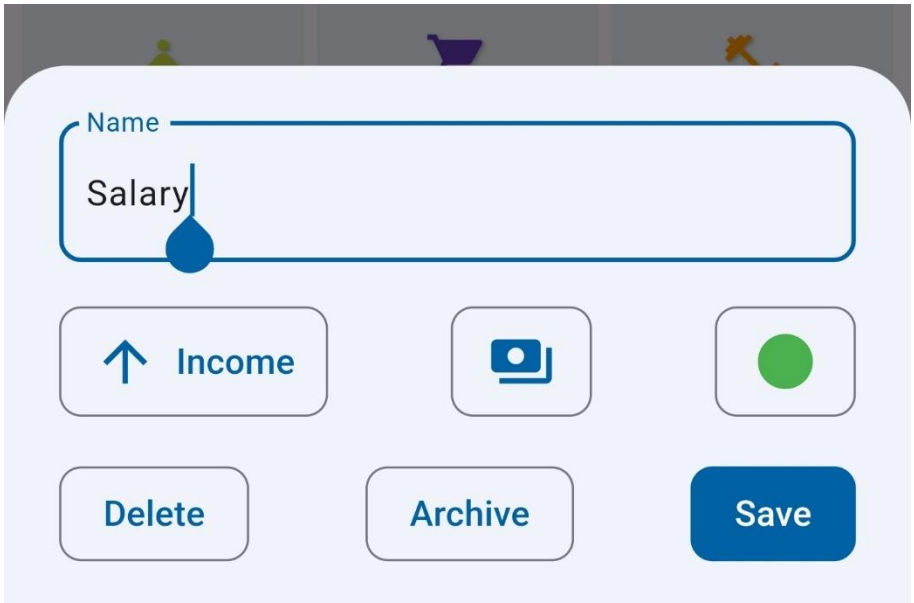

Рисунок 3.9 – Меню редагування категорії

Серед особливостей редагування категорій, можна відмітити можливість архівування категорії. Цей механізм зроблено, щоб не бачити категорію серед їх переліку, але не видаляти всі пов’язані з нею транзакції, що відбувається під час видалення.

Екран з переліком транзакцій (рис. 3.10) знаходиться у файлі transactions_page.dart і реалізований за допомогою вбудованого віджета Column, в якому розташовуються віджети з вибором місяця та список транзакцій, що реалізовані у вигляді карток. Кожна з транзакцій має піктограму та назву категорії, нотаток, дату та суму. Якщо зробити свайп транзакції вліво, то буде вікно підтвердження видалення транзакції. Якщо натиснути на транзакцію, то з’явиться меню з редагування (рис. 3.8). Список транзакцій відсортований за датою, чим вище, тим новіше транзакція.

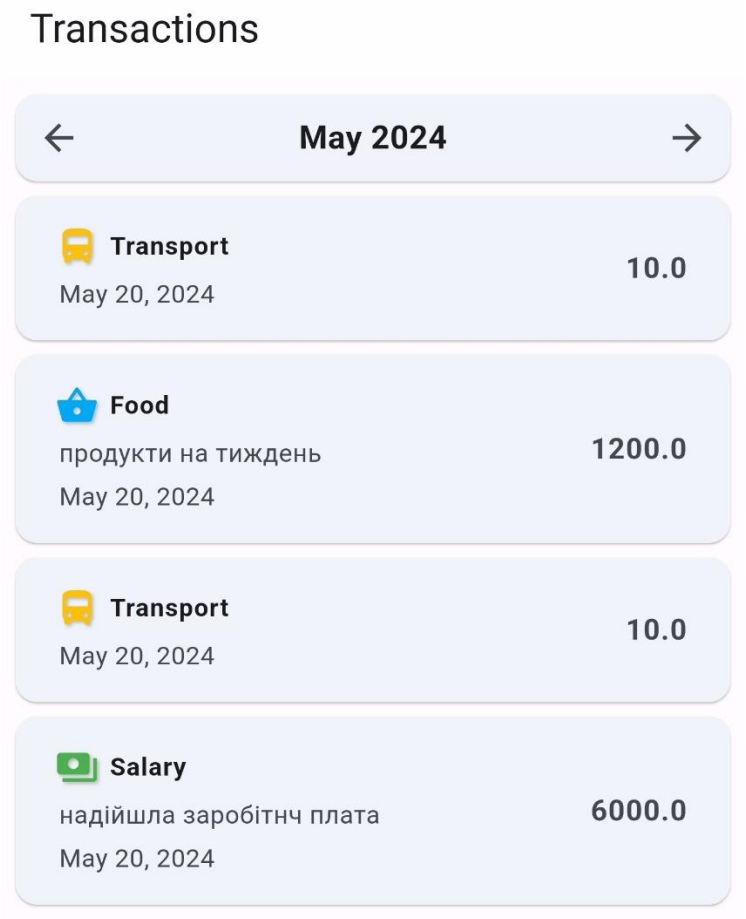

Рисунок 3.10 – Екран транзакцій

Аналітика щодо витрат та надходжень знаходиться на третьому екрані (рис. 3.11), що знаходиться у файлі analytics_page.dart. Вгорі реалізовано механізм вибору місяця, нижче віджет з сумою надходжень, витрат та їх різницею. Також зроблений перелік категорій і їх загальну суму, вони відсортовані за спаданням, що дозволяє користувачу зручно переглянути їх.

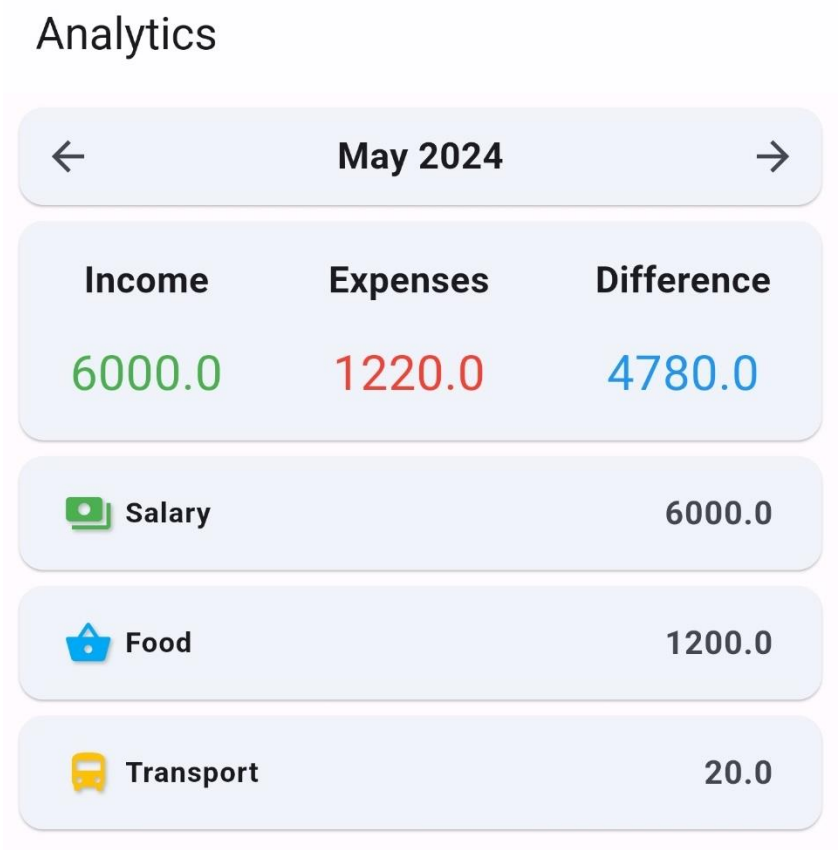

Рисунок 3.11 – Екран аналітики

Кожен користувач має індивідуальний смак щодо теми пристрою та кольорової палітри. Для забезпечення гарного візуального вигляду та зручності користувача, було реалізовано механізм автоматичного визначення теми та кольорової гами пристрою і використання її у застосунку. Тему можна додатково налаштувати за потреби. Для реалізацій вибору теми застосунку, була створена сторінка з налаштуваннями, де можуть у майбутньому знаходитись й інші налаштування додатку. Щоб зробити вибір теми динамічним і реагувати на зміну теми смартфону, було використано ChangeNotifierProvider та Consumer.

Екран налаштувань (рис. 3.12) реалізовано у файлі settings_page.dart і містить перемикач теми, що реалізовано за допомогою SegmentedButton.

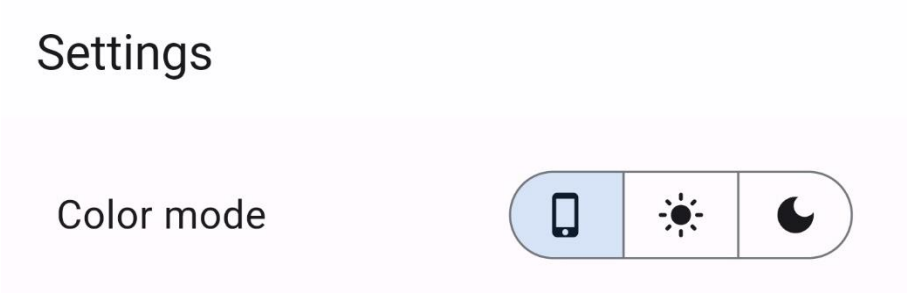

Рисунок 3.12 – Екран налаштувань

Алгоритмічну частину мобільного застосунку можна розділити на збереження даних та на їх обробку. Для збереження даних, було реалізовано сервіс DatabaseService у файлі database_service.dart, що створює файл з локальною базою даних і таблиці, якщо їх немає і виконує запити до бази даних. Для спілкування з даним сервісом виведено публічні методи, до яких сервіси можуть звертатись. Щоб оптимізувати роботу з базою даних та даними в цілому, всі дані зчитуються лише при завантаженні застосунку, а далі використовуються і при їх зміненні записуються у базу даних.

Серсіси, що працюють з обробкою даних – це CategoriesService та TransactionsService. Вони працюють з категоріями та транзакціями, їх сортуванням, зміною індексів, видалення та групуванням.

Моделі, що були реалізовані для застосунку відповідають моделі бази даних і генерують id за допомогою uuid 4.

Під час реалізації було використано наступні бібіліотеки: cupertino_icons – для коректної роботи на iOS, uuid – для генерації та роботи з uuid, provider – для роботи з темою та динамічною її зміню, shared_preferences – для збереження налаштувань, intl та jiffy – для форматування та зручної роботи з датою, flutter_draggable_gridview – для виводу таблиці, елементи якої можна переносити, sqflite та path – для роботи з SQLite.

3.4 Тестування

Щоб забезпечити належну якість інформаційної системи контролю фінансів було проведено функціональне та нефункціональне тестування. Метою тестування було перевірити коректність роботи всіх функцій мобільного застосунку та його стабільність на різних платформах (iOS та Android).

Функціональне тестування включало перевірку наступного функціоналу:

- Додавання транзакцій;
- Редагування транзакцій;
- Додавання категорій транзакцій;
- Редагування категорій транзакцій;
- Перегляд списку витрат за місяцем;
- Перегляд аналітики за місяцем.

Нефункціональне тестування включало:

- Тестування продуктивності;
- Тестування зручності використання;
- Тестування сумісності;
- Тестування надійності.

Результати тестування представлені у таблиці 3.3:

Таблиця 3.3 – Результати тестування інформаційної системи контролю фінансів

Тестовий сценарій	Android	iOS
Додавання транзакції	+	+
Редагування транзакції	+	+
Додавання категорії	+	+
Редагування категорії	+	+
Перегляд списку витрат за місяцем	+	+
Перегляд аналітики за місяцем	+	+

Зручність використання	+	+
Продуктивність при великій кількості даних	+	+
Надійність при тривалому використанні	+	+

Для оцінки зручності та функціональності мобільного застосунку для контролю фінансів, було проведено тестування за участю чотирьох користувачів протягом одного тижня.

Відгуки, що були отримані від користувачів підтвердили, що застосунок виконує свої основні функції, такі як додавання та редагування транзакцій, категорій, перегляд списку витрат та аналітики за місяцем, працювали без збоїв на платформах iOS та Android.

Загалом, користувачі високо оцінили зручність інтерфейсу та корисність представлених функцій. Однак, деякі користувачі висловили побажання щодо розширення функціоналу.

Під час проведення тестування та експлуатації мобільного застосунку було виявлено декілька потенційних можливостей для покращення функціоналу. Серед них:

- Не вистачає підтримки для різних рахунків;
- Відсутність функціоналу для повторюваних витрат;
- Недостатньо налаштувань;

Хоч ці недоліки можна вважати відносно незначними, їх усунення може значно покращити користувацький досвід та функціональність застосунку.

Таким чином, результати тестування та відгуки користувачів свідчать про успішне виконання основних завдань застосунку, але звертають увагу на необхідність подальшого розвитку для покращення його функціональності.

ВИСНОВКИ

Під час аналітичного огляду було виявлено, що системи керування фінансами стають невід'ємною частиною повсякденного життя користувачів. Вони дозволяють зручно та ефективно відстежувати витрати, планувати бюджет та керувати фінансами. Розвиток цього сегменту ринку свідчить про постійну потребу в таких застосунках та можливості для їхнього удосконалення та розширення функціоналу. Реалізація інформаційної системи керування фінансами буде більш ефективна та корисна у вигляді мобільного застосунку.

У результаті структурно-функціонального моделювання було побудовано IDEF0 діаграму та зроблено її декомпозицію. Також створено моделювання діаграми варіантів використання, де описані основні функції, що доступні користувачу.

Під час вибору інструментів розробки, було вибрано Flutter разом з Dart, для програмування мобільного застосунку. Для системи управління базами даних та збереження даних користувача, вибрано SQLite. Щоб процес розробки був приємним для розробника та ефективним у використанні ресурсів комп’ютера, використано Visual Studio Code та емулятор смартфону.

Було спроектовано макет для мобільного застосунку, на якому зображений основний функціонал та зовнішній вигляд застосунку.

На основі макету, структурно-функціональної діаграми та діаграми варіантів використання було реалізовано й протестовано мобільний застосунок для керування фінансами, який відповідає вимогам сучасного користувача. Також застосунок дозволяє зручно та ефективно керувати витратами та доходами. Система має інтуїтивно зрозумілий інтерфейс, яким зручно та легко можна користуватись. Під час тестування застосунок отримав позитивні відгуки від користувачів, що говорить про його ефективність та зручність використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Klapper, L., & Lusardi, A. (2020). Financial literacy and financial resilience: Evidence from around the world. Financial Management, 49(3), 589-614.

2. Bitrián, P., Buil, I., & Catalán, S. (2021). Making finance fun: the gamification of personal financial management apps. International journal of bank marketing, 39(7), 1310-1332.

3. Сорока, Б. (2022). Діджиталізація фінансового ринку України: ключові ризики для індивідуальних інвесторів. Економіка та суспільство, (43).

4. Li, T., Xia, T., Wang, H., Tu, Z., Tarkoma, S., Han, Z., & Hui, P. (2022). Smartphone app usage analysis: datasets, methods, and applications. IEEE Communications Surveys & Tutorials, 24(2), 937-966.

5. Carlin, B., Olafsson, A., & Pagel, M. (2023). Mobile apps and financial decision making. Review of Finance, 27(3), 977-996.

6. Wang, Q., Wang, D., Cheng, C., & He, D. (2021). Quantum2fa: efficient quantum-resistant two-factor authentication scheme for mobile devices. IEEE Transactions on Dependable and Secure Computing, 20(1), 193-208.

7. Білоус, А. Р., Циліцінська, І. В., Чумак, А. В., & Шворак, А. С. РОЗРОБКА МОБІЛЬНИХ ДОДАТКІВ: ПЕРЕДУМОВИ СТАНОВЛЕННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ БІЗНЕСУ В УКРАЇНІ. Організаційний комітет декади студенської науки «Наукові барви—2021», 79.

8. Google Play – Результат пошуку «контроль фінансів», [Електроний ресурс], - Режим доступу до ресурсу: https://play.google.com/store/search?q=%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C%20%D1%84%D1%96%D0%BD%D0%B0%D0%BD%D1%81%D1%96%D0%B2&c=apps&hl=en&gl=US

9. Google Play – «Money manager & expenses», [Електроний ресурс], - Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=ru.innim.my_finance

10. Google Play – «Wallet: Budget Expense Tracker», [Електроний ресурс], - Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.droid4you.application.wallet

11. Google Play – «Monefy - Budget & Expenses app», [Електроний ресурс], - Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.monefy.app.lite

12. Presley, A., & Liles, D. H. (1995, May). The use of IDEF0 for the design and specification of methodologies. In Proceedings of the 4th industrial engineering research conference (pp. 442-448).

13. Bittner, K., & Spence, I. (2003). Use case modeling. Addison-Wesley Professional.

14. Statcounter – Mobile Operating System Market Share Worldwide, [Електроний ресурс], - Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide

15. Flutter - Build apps for any screen, [Електроний ресурс], - Режим доступу до ресурсу: https://flutter.dev/

16. React Native · Learn once, write anywhere, [Електроний ресурс], - Режим доступу до ресурсу: https://reactnative.dev/

17. SQLite Home Page, [Електроний ресурс], - Режим доступу до ресурсу: https://www.sqlite.org/

18. , Visual Studio Code - Code Editing. Redefined, [Електроний ресурс], - Режим доступу до ресурсу: https://code.visualstudio.com/

19. Download Android Studio & App Tools - Android Developers, [Електроний ресурс], - Режим доступу до ресурсу: https://developer.android.com/studio

20. Xcode 15 - Apple Developer, [Електроний ресурс], - Режим доступу до ресурсу: https://developer.apple.com/xcode/

21. Git, [Електроний ресурс], - Режим доступу до ресурсу: https://git-scm.com/

ДОДАТОК А. ПЛАНУВАННЯ РОБІТ

А.1 Планування змісту роботи

Планування структури виконання робіт – важливий етап у керуванні будь-яким проектом. Цей процес полягає у створенні детального плану робіт з використанням інструментів, таких як WBS діаграма. WBS (Work Breakdown Structure) - це графічне зображення у вигляді ієрархічно пов'язаних пакетів робіт, які потрібно виконати для досягнення мети проекту. Діаграма WBS, що зображена на рисунку А.1, дозволяє розкрити всі деталі проекту і зробити його виконання більш систематичним і контрольованим.

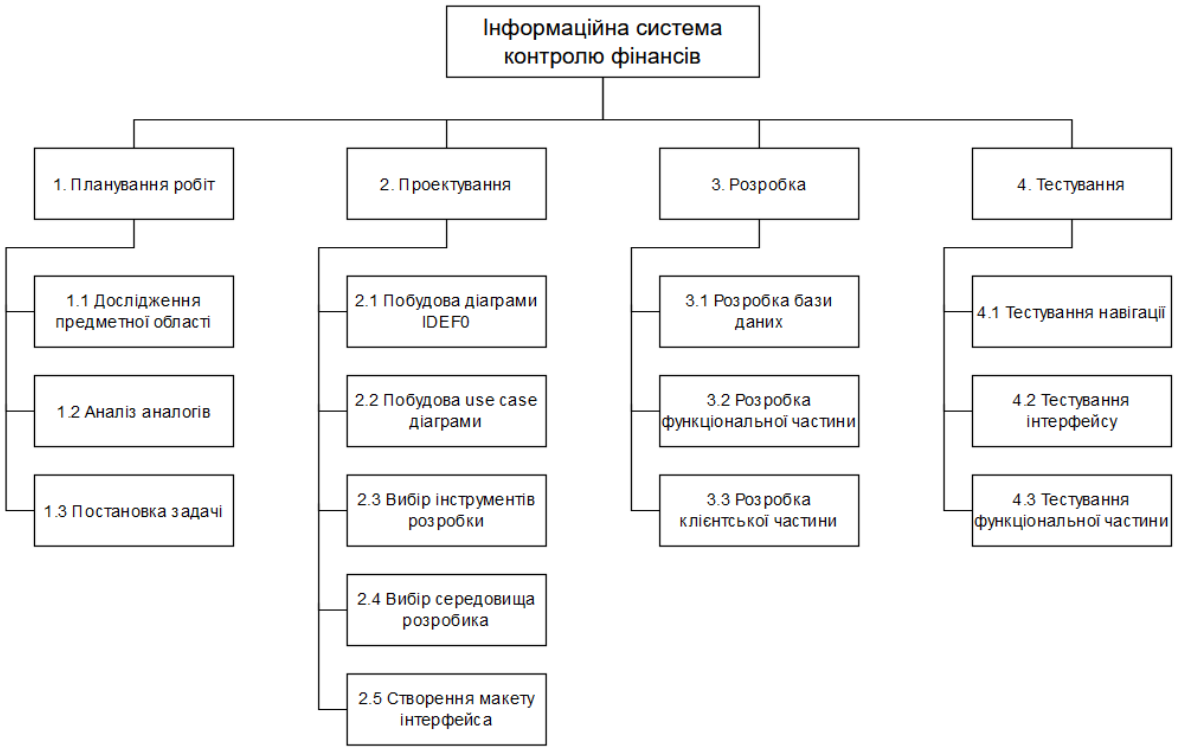

Рисунок А.1 – Діаграма WBS

А.2 Планування структури виконавців

Після створення WBS, наступним кроком є планування структури організації проекту для впровадження готового проекту (OBS). OBS (Organizational Breakdown Structure) - це організаційна структура, яка

відображає взаємозв'язок між виконавцями проекту та їхніми функціями. Створення OBS (рис. А.2) допомагає уникнути змішування завдань та ролей в процесі виконання проекту, а також сприяє кращому розподілу відповідальності та контролю за виконанням робіт.

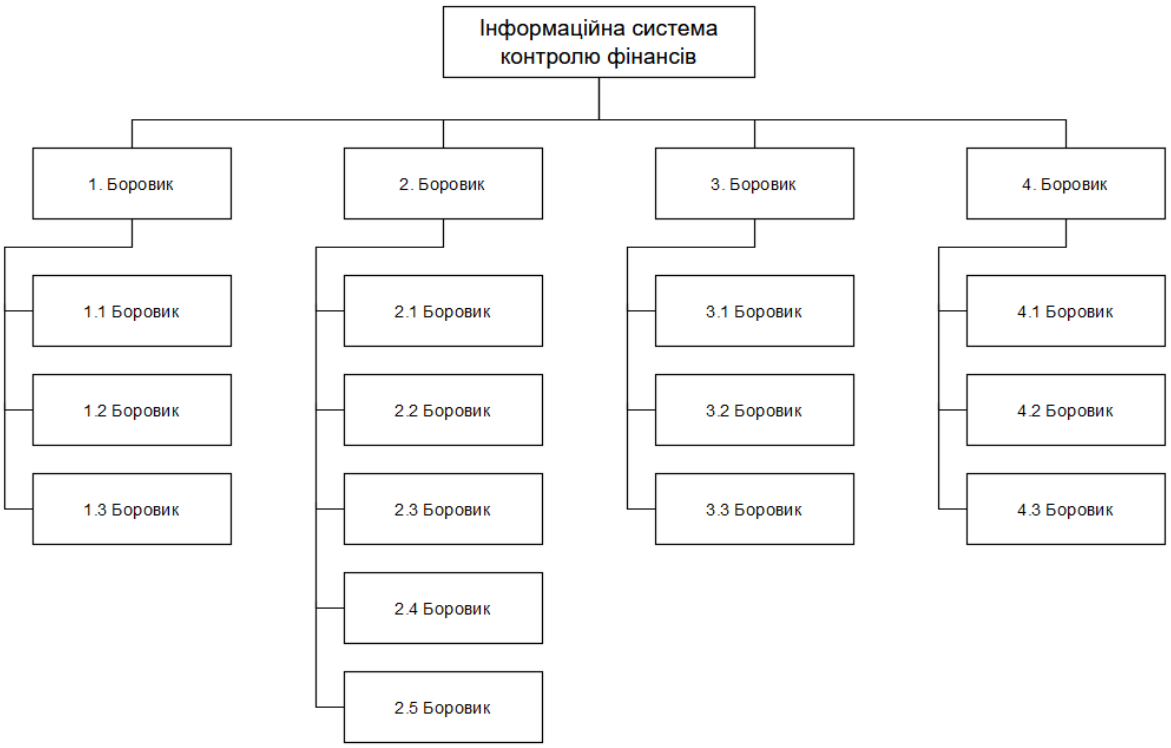

Рисунок А.2 – Діаграма WBS

А.3 Діаграма Ганта

Створення діаграми Ганта - це ще один важливий етап у плануванні та керуванні проектами. Діаграма Ганта дозволяє візуалізувати часовий графік виконання робіт у проекті. Головний принцип цієї діаграми полягає у відображенні роботі на проекті, як поздовжній сегмент на горизонтальній площині, що пропорційна тривалості виконання цієї роботи. Таким чином, можна чітко побачити, коли кожна робота має бути виконана та як вона взаємозв'язана з іншими роботами в проекті.

Щоб створити діаграму Ганта (рис. А.3), треба розпочати зі складання списку всіх завдань, які потрібно виконати в проекті, та визначення тривалості кожного завдання. Потім ці завдання відображаються на горизонтальній осі діаграми як відповідні сегменти. Додавання залежностей між завданнями дозволяє відобразити порядок виконання робіт та можливі затримки, які можуть виникнути у процесі виконання проекту.

	Задача	Кількість	Початок	Кінець
1	Інформаційна система контролю фінансів	49	3.04.2024	21.05.2024
2	Планування робіт	7	3.04.2024	10.04.2024
3	Постановка задачі	1	3.04.2024	4.04.2024
4	Дослідження предметної області	2	4.04.2024	6.04.2024
5	Розробка календарного плану	1	6.04.2024	7.04.2024
6	Аналіз аналогів	3	7.04.2024	10.04.2024
7	Проектування	7	10.04.2024	10.04.2024
8	Побудова діграми IDEF0	1	10.04.2024	11.04.2024
9	Побудова use case діаграми	1	11.04.2024	12.04.2024
10	Вибір інструментів розробки	2	12.04.2024	14.04.2024
11	Вибір середовища розробки	1	14.04.2024	15.04.2024
12	Створення макету інтерфейса	2	15.04.2024	17.04.2024
13	Розробка	32	17.04.2024	19.05.2024
14	Проектування бази даних	2	17.04.2024	19.04.2024
15	Розробка бази даних	4	19.04.2024	23.04.2024
16	Розробка функціональної частини	12	23.04.2024	5.05.2024
17	Розробка клієнтської частини	14	5.05.2024	19.05.2024
18	Тестування	3	19.05.2024	21.05.2024
22	Тестування навігації	1	19.05.2024	20.05.2024
23	Тестування інтерфейсу	1	20.05.2024	21.05.2024
24	Тестування функціональної частини	2	21.05.2024	22.05.2024

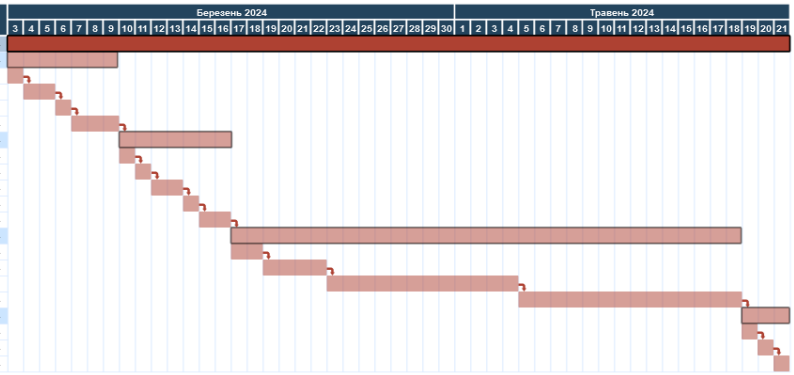

Рисунок А.3 – Діаграма Ганта

ДОДАТОК Б. ПРОГРАМНИЙ КОД

categories_page.dart

```
import 'package:expense_tracker/models/category_model.dart';
import 'package:expense_tracker/models/transaction_model.dart';
import
'package:expense_tracker/screens/categories/widgets/save_category_widget.dart
';
import
'package:expense_tracker/screens/transactions/widgets/save_transaction_widget
.dart';
import 'package:expense_tracker/services/categories_service.dart';
import 'package:expense_tracker/services/theme_provider.dart';
import 'package:expense_tracker/services/transactions_service.dart';
import 'package:flutter/material.dart';
import
'package:flutter_draggable_gridview/flutter_draggable_gridview.dart';

class CategoriesPage extends StatefulWidget {
  const CategoriesPage({super.key});

  @override
  State<CategoriesPage> createState() => _CategoriesPageState();
}

class _CategoriesPageState extends State<CategoriesPage> {
  late CategoryPageEvent _pageEvent;
  late Icon _actionIcon;

  @override
  void initState() {
    super.initState();
    _pageEvent = _getInitialPageEvent();
    _actionIcon = Icon(_getInitialActionIcon());
  }

  List<Category> _getCategories() {
    return CategoriesService.categories.where((category) {
      return category.enabled;
    }).toList();
  }

  void _onAddTransaction(Category category) {
    showModalBottomSheet(
      context: context,
      builder: (ctx) {
        return SaveTransactionWidget(
          transaction: Transaction.create(
            categoryId: category.id,
            amount: 0,
            date: DateTime.now(),
            note: '',
          ),
          onSave: (transaction) {
            TransactionsService.addTransaction(transaction);
          },
        );
      },
    );
```

```
}

void _onSaveCategory(Category category) {
  showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    builder: (ctx) {
      return SaveCategoryWidget(
        category: category,
        onDelete: (category) {
          setState(() {
            CategoriesService.deleteCategory(category);
          });
          Navigator.of(context).pop();
        },
        onArchive: (category) {
          setState(() {
            CategoriesService.disableCategory(category);
          });
          Navigator.of(context).pop();
        },
        onSave: (updatedCategory) {
          setState(() {
            category.name = updatedCategory.name;
            category.icon = updatedCategory.icon;
            category.color = updatedCategory.color;
            category.type = updatedCategory.type;
          });
          CategoriesService.updateCategory(category);
          Navigator.of(context).pop();
        },
        saveMode: CategorySaveMode.edit,
      );
    },
  );
}

void _onAddCategory() {
  showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    builder: (ctx) {
      return SaveCategoryWidget(
        category: Category.create(
          icon: Icons.category,
          color: Colors.grey,
          name: '',
          type: CategoryType.expense,
          position: _getCategories().length,
        ),
        onDelete: (category) => {},
        onArchive: (category) {},
        onSave: (category) {
          setState(() {
            CategoriesService.addCategory(category);
          });
          Navigator.of(context).pop();
        },
        saveMode: CategorySaveMode.create,
      );
    },
  );
}
```

```
CategoryPageEvent _getInitialPageEvent() {
  if (_getCategories().isEmpty) {
    return CategoryPageEvent.editCategory;
  } else {
    return CategoryPageEvent.addTransaction;
  }
}

IconData _getInitialActionIcon() {
  if (_getCategories().isEmpty ||
      _pageEvent == CategoryPageEvent.editCategory) {
    return Icons.save;
  } else {
    return Icons.edit;
  }
}

Card _buildCategoryCardWidget(Category category) {
  return Card(
    child: ConstrainedBox(
      constraints: const BoxConstraints(
        minWidth: 100,
        minHeight: 100,
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 8.0),
            child: Icon(
              category.icon,
              color: category.color,
              size: 30,
              shadows: ThemeProvider().getIconsShadows(),
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(5),
            child: Text(
              category.name,
              maxLines: 1,
              overflow: TextOverflow.ellipsis,
              style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        ],
      ),
    ),
  );
}

Card _buildAddCategoryCardWidget() {
  return Card(
    child: Padding(
      padding: const EdgeInsets.all(10),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 8.0),
            child: Icon(
```

```
                    Icons.add,
                    color: Colors.grey,
                    size: 30,
                    shadows: ThemeProvider().getIconsShadows(),
                  ),
                ),
                const Padding(
                  padding: EdgeInsets.all(5),
                  child: Text(
                    'Add',
                    maxLines: 1,
                    overflow: TextOverflow.ellipsis,
                    style: TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
              ],
            ),
          ),
        );
      }

      List<DraggableGridItem> _buildCategoriesWidgets() {
        List<Category> categories = _getCategories();
        List<DraggableGridItem> categoryWidgets = categories.map((category)
{
          return DraggableGridItem(
            isDraggable: _pageEvent == CategoryPageEvent.editCategory,
            child: GestureDetector(
              onTap: () {
                if (_pageEvent == CategoryPageEvent.addTransaction) {
                  _onAddTransaction(category);
                } else {
                  _onSaveCategory(category);
                }
              },
              child: _buildCategoryCardWidget(category),
            ),
          );
        }).toList();

        if (_pageEvent == CategoryPageEvent.editCategory) {
          categoryWidgets.add(
            DraggableGridItem(
              child: GestureDetector(
                onTap: _onAddCategory,
                child: _buildAddCategoryCardWidget(),
              ),
            ),
          );
        }

        return categoryWidgets;
      }

      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            title: const Text('Categories'),
            actions: [
              IconButton(
```

```
                  onPressed: () {
                    setState(() {
                      if (_getCategories().isEmpty ||
                          _pageEvent == CategoryPageEvent.addTransaction) {
                        _pageEvent = CategoryPageEvent.editCategory;
                        _actionIcon = const Icon(Icons.save);
                      } else {
                        _pageEvent = CategoryPageEvent.addTransaction;
                        _actionIcon = const Icon(Icons.edit);
                      }
                    });
                  },
                  icon: _actionIcon,
                ),
              ],
            ),
            body: DraggableGridViewBuilder(
              isOnlyLongPress: false,
              padding: const EdgeInsets.all(5),
              gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 3,
              ),
              dragCompletion: (list, beforeIndex, afterIndex) {
                setState(() {
                  CategoriesService.reorderCategories(beforeIndex,
afterIndex);
                });
              },
              dragPlaceHolder: (list, index) {
                return PlaceHolderWidget(
                  child: Container(
                    color: Colors.transparent,
                  ),
                );
              },
              children: _buildCategoriesWidgets(),
            ),
          );
        }
      }

      enum CategoryPageEvent {
        addTransaction,
        editCategory,
      }
```

transactions_page.dart

```
      import 'package:expense_tracker/models/transaction_model.dart';
      import
'package:expense_tracker/screens/transactions/widgets/save_transaction_widget
.dart';
      import 'package:expense_tracker/services/categories_service.dart';
      import 'package:expense_tracker/services/theme_provider.dart';
      import 'package:expense_tracker/services/transactions_service.dart';
      import 'package:flutter/material.dart';
      import 'package:intl/intl.dart';
      import 'package:jiffy/jiffy.dart';

      class TransactionsPage extends StatefulWidget {
        const TransactionsPage({super.key});

        @override
```

```
  State<TransactionsPage> createState() => _TransactionsPageState();
}

class _TransactionsPageState extends State<TransactionsPage> {
  DateTime _selectedDate = DateTime(
    DateTime.now().year,
    DateTime.now().month,
  );

  String _buildPeriodText() {
    String text = DateFormat.MMMM().format(_selectedDate);
    text += ' ';
    text += DateFormat.y().format(_selectedDate);
    return text;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Transactions'),
      ),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.only(
              top: 8,
              left: 8,
              right: 8,
            ),
            child: Card(
              margin: const EdgeInsets.all(0),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  IconButton(
                    onPressed: () {
                      setState(() {
                        _selectedDate =
Jiffy.parseFromDateTime(_selectedDate)
                                .subtract(months: 1)
                                .dateTime;
                      });
                    },
                    icon: const Icon(Icons.arrow_back),
                  ),
                  Text(
                    _buildPeriodText(),
                    style: TextStyle(
                      fontSize: 18,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  IconButton(
                    onPressed: () {
                      setState(() {
                        _selectedDate =
Jiffy.parseFromDateTime(_selectedDate)
                                .add(months: 1)
                                .dateTime;
                      });
                    },
                    icon: const Icon(Icons.arrow_forward),
                  ),
```

```
                    ],
                  ),
                ),
              ),
              TransactionsListWidget(
                fromDate: Jiffy.parseFromDateTime(_selectedDate)
                    .startOf(Unit.month)
                    .dateTime,
                toDate: Jiffy.parseFromDateTime(_selectedDate)
                    .endOf(Unit.month)
                    .dateTime,
              ),
            ],
          ),
        );
      }
    }

    class TransactionsListWidget extends StatefulWidget {
      const TransactionsListWidget({
        required this.fromDate,
        required this.toDate,
        super.key,
      });

      final DateTime fromDate;
      final DateTime toDate;

      @override
      State<TransactionsListWidget> createState() =>
_TransactionsListWidgetState();
    }

    class _TransactionsListWidgetState extends State<TransactionsListWidget>
{
      void _onRemoveTransaction(Transaction transaction) {
        setState(() {
          TransactionsService.deleteTransaction(transaction);
        });
      }

      @override
      Widget build(BuildContext context) {
        final transactions = TransactionsService.getTransactionsByDate(
          widget.fromDate,
          widget.toDate,
        );
        return Expanded(
          child: ListView.builder(
            itemCount: transactions.length,
            itemBuilder: (ctx, index) {
              return TransactionCardWidget(
                transaction: transactions[index],
                onRemove: _onRemoveTransaction,
              );
            },
          ),
        );
      }
    }

    class TransactionCardWidget extends StatefulWidget {
      final Transaction _transaction;
      final void Function(Transaction transaction) _onRemove;
```

```
        const TransactionCardWidget({
          Key? key,
          required Transaction transaction,
          required void Function(Transaction transaction) onRemove,
        })  : _transaction = transaction,
              _onRemove = onRemove,
              super(key: key);

        @override
        State<TransactionCardWidget> createState() =>
_TransactionCardWidgetState();
      }

      class _TransactionCardWidgetState extends State<TransactionCardWidget> {
        void _onEditTransaction() {
          showModalBottomSheet(
            context: context,
            isScrollControlled: false,
            builder: (ctx) {
              return SaveTransactionWidget(
                transaction: widget._transaction,
                onSave: (editedTransaction) {
                  setState(() {
                    widget._transaction.categoryId =
editedTransaction.categoryId;
                    widget._transaction.amount = editedTransaction.amount;
                    widget._transaction.date = editedTransaction.date;
                    widget._transaction.note = editedTransaction.note;
                  });
                  TransactionsService.updateTransaction(widget._transaction);
                },
              );
            },
          );
        }

        Widget _buildSubtitle() {
          List<Widget> children = [];
          if (widget._transaction.note.isNotEmpty) {
            children.add(
              Padding(
                padding: const EdgeInsets.only(
                  top: 4,
                  left: 8,
                  right: 8,
                ),
                child: Text(widget._transaction.note),
              ),
            );
          }
          children.add(
            Padding(
              padding: const EdgeInsets.only(
                top: 4,
                left: 8,
                right: 8,
                bottom: 8,
              ),
              child: Text(
                DateFormat.yMMMd().format(widget._transaction.date),
              ),
            ),
          );
```

```
          return Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: children,
          );
        }

        @override
        Widget build(BuildContext context) {
          final category =

CategoriesService.getCategoryById(widget._transaction.categoryId);
          return Padding(
            padding: const EdgeInsets.only(
              top: 8,
              left: 8,
              right: 8,
            ),
            child: Card(
              margin: const EdgeInsets.all(0),
              clipBehavior: Clip.antiAlias,
              child: Dismissible(
                key: ValueKey(widget._transaction.id),
                direction: DismissDirection.endToStart,
                onDismissed: (direction) {
                  widget._onRemove(widget._transaction);
                },
                confirmDismiss: (direction) async => await showDialog(
                  context: context,
                  builder: (ctx) {
                    return const TransactionConfirmationDialogWidget();
                  },
                ),
                background: Card(
                  margin: EdgeInsets.zero,
                  color:
Theme.of(context).colorScheme.error.withOpacity(0.75),
                  child: Container(
                    padding: const EdgeInsets.only(right: 16),
                    alignment: Alignment.centerRight,
                    child: const Icon(
                      Icons.delete,
                      color: Colors.white,
                    ),
                  ),
                ),
                child: ListTile(
                  onTap: _onEditTransaction,
                  title: Padding(
                    padding: const EdgeInsets.only(top: 8),
                    child: Row(
                      mainAxisSize: MainAxisSize.min,
                      children: [
                        Padding(
                          padding: const EdgeInsets.symmetric(horizontal: 6),
                          child: Icon(
                            category.icon,
                            color: category.color,
                            shadows: ThemeProvider().getIconsShadows(),
                          ),
                        ),
                        Text(
                          category.name,
                          style: TextStyle(
                            fontSize: 14,
```

```
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ],
            ),
          ),
          subtitle: _buildSubtitle(),
          trailing: Text(
            '${widget._transaction.amount / 100}',
            style: TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 16,
            ),
          ),
        ),
      ),
    ),
  );
 }
}

class TransactionConfirmationDialogWidget extends StatelessWidget {
  const TransactionConfirmationDialogWidget({
    super.key,
  });

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: const Text('Are you sure?'),
      content: const Text(
        'Do you want to remove the transaction?',
        style: TextStyle(
          fontSize: 15,
        ),
      ),
      contentPadding: const EdgeInsets.all(22),
      actions: <Widget>[
        const TextButton(
          onPressed: () {
            Navigator.of(context).pop(false);
          },
          child: const Text('No'),
        ),
        const TextButton(
          onPressed: () {
            Navigator.of(context).pop(true);
          },
          child: const Text('Yes'),
        ),
      ],
      actionsPadding: const EdgeInsets.symmetric(horizontal: 10),
    );
  }
}
```

analytics_page.dart

```
import 'package:expense_tracker/services/categories_service.dart';
import 'package:expense_tracker/services/theme_provider.dart';
import 'package:expense_tracker/services/transactions_service.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
```

```
import 'package:jiffy/jiffy.dart';

class AnalyticsPage extends StatefulWidget {
  const AnalyticsPage({super.key});

  @override
  State<AnalyticsPage> createState() => _AnalyticsPageState();
}

class _AnalyticsPageState extends State<AnalyticsPage> {
  DateTime _selectedDate = DateTime(
    DateTime.now().year,
    DateTime.now().month,
  );

  String _buildPeriodText() {
    String text = DateFormat.MMMM().format(_selectedDate);
    text += ' ';
    text += DateFormat.y().format(_selectedDate);
    return text;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Analytics'),
      ),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.only(
              top: 8,
              left: 8,
              right: 8,
            ),
            child: Card(
              margin: const EdgeInsets.all(0),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  IconButton(
                    onPressed: () {
                      setState(() {
                        _selectedDate =
Jiffy.parseFromDateTime(_selectedDate)
                                .subtract(months: 1)
                                .dateTime;
                      });
                    },
                    icon: const Icon(Icons.arrow_back),
                  ),
                  Text(
                    _buildPeriodText(),
                    style: TextStyle(
                      fontSize: 18,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  IconButton(
                    onPressed: () {
                      setState(() {
                        _selectedDate =
Jiffy.parseFromDateTime(_selectedDate)
```

```
                            .add(months: 1)
                            .dateTime;
                      });
                    },
                    icon: const Icon(Icons.arrow_forward),
                  ),
                ],
              ),
            ),
            AnalyticsSummaryWidget(
              selectedDate: _selectedDate,
            ),
            AnalyticsChartWidget(
              selectedDate: _selectedDate,
            ),
          ],
        ),
      );
  }
}

class AnalyticsSummaryWidget extends StatelessWidget {
  const AnalyticsSummaryWidget({
    required this.selectedDate,
    super.key,
  });

  final DateTime selectedDate;

  Widget _buildSummaryItem({
    required String title,
    required int value,
    required Color color,
  }) {
    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 8),
      child: Column(
        children: [
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 8),
            child: Text(
              title,
              style: TextStyle(
                fontSize: 18,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 8),
            child: Text(
              '${value / 100}',
              style: TextStyle(
                fontSize: 24,
                color: color,
              ),
            ),
          ),
        ],
      ),
    );
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.only(
        top: 8,
        left: 8,
        right: 8,
      ),
      child: Card(
        margin: const EdgeInsets.all(0),
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 25),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              _buildSummaryItem(
                title: 'Income',
                value: TransactionsService.getIncomeByDate(
                  Jiffy.parseFromDateTime(selectedDate)
                      .startOf(Unit.month)
                      .dateTime,
                  Jiffy.parseFromDateTime(selectedDate)
                      .endOf(Unit.month)
                      .dateTime,
                ),
                color: Colors.green,
              ),
              _buildSummaryItem(
                title: 'Expenses',
                value: TransactionsService.getExpenseByDate(
                  Jiffy.parseFromDateTime(selectedDate)
                      .startOf(Unit.month)
                      .dateTime,
                  Jiffy.parseFromDateTime(selectedDate)
                      .endOf(Unit.month)
                      .dateTime,
                ),
                color: Colors.red,
              ),
              _buildSummaryItem(
                title: 'Difference',
                value: TransactionsService.getBalanceByDate(
                  Jiffy.parseFromDateTime(selectedDate)
                      .startOf(Unit.month)
                      .dateTime,
                  Jiffy.parseFromDateTime(selectedDate)
                      .endOf(Unit.month)
                      .dateTime,
                ),
                color: Colors.blue,
              ),
            ],
          ),
        ),
      ),
    );
  }
}

class AnalyticsChartWidget extends StatelessWidget {
  const AnalyticsChartWidget({
    required this.selectedDate,
    super.key,
  });
```

```
        final DateTime selectedDate;

        @override
        Widget build(BuildContext context) {
          final categoriesSum = TransactionsService.getSortedCateogriesSum(

Jiffy.parseFromDateTime(selectedDate).startOf(Unit.month).dateTime,
            Jiffy.parseFromDateTime(selectedDate).endOf(Unit.month).dateTime,
          );
          return Expanded(
            child: ListView.builder(
              itemCount: categoriesSum.length,
              itemBuilder: (context, index) {
                final categorySum = categoriesSum[index];
                final category =
CategoriesService.getCategoryById(categorySum.key);
                final sum = categorySum.value;
                return Padding(
                  padding: const EdgeInsets.only(
                    top: 8,
                    left: 8,
                    right: 8,
                  ),
                  child: Card(
                    margin: const EdgeInsets.all(0),
                    clipBehavior: Clip.antiAlias,
                    child: ListTile(
                      title: Row(
                        mainAxisSize: MainAxisSize.min,
                        children: [
                          Padding(
                            padding: const EdgeInsets.symmetric(horizontal:
6),

                            child: Icon(
                              category.icon,
                              color: category.color,
                              shadows: ThemeProvider().getIconsShadows(),
                            ),
                          ),
                          Text(
                            category.name,
                            style: TextStyle(
                              fontSize: 14,
                              fontWeight: FontWeight.bold,
                            ),
                          ),
                        ],
                      ),
                      trailing: Text(
                        '${sum / 100}',
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                          fontSize: 16,
                        ),
                      ),
                    ),
                  ),
                );
              },
            ),
          );
        }
      }
```

settings_page.dart

```
import 'package:flutter/material.dart';
import
'package:expense_tracker/screens/settings/widgets/color_mode_widget.dart';

class SettingsPage extends StatelessWidget {
  const SettingsPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Settings'),
      ),
      body: SizedBox(
        width: double.infinity,
        child: Container(
          margin: const EdgeInsets.only(
            top: 25,
            left: 25,
            right: 25,
          ),
          child: Column(
            children: [
              Row(
                mainAxisSize: MainAxisSize.max,
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  Text(
                    'Color mode',
                    style:
Theme.of(context).textTheme.bodyMedium?.copyWith(
                          fontSize: 18,
                        ),
                  ),
                  const ColorModeWidget(),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

settings_service.dart

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';

class SettingsService {
  SettingsService._();

  static late final SharedPreferences _settingsStorage;

  static Future<bool> init() async {
    _settingsStorage = await SharedPreferences.getInstance();
    return true;
```

```
  }

  static Future<bool> setThemeMode(ThemeMode themeMode) {
    return _settingsStorage.setInt('themeMode', themeMode.index);
  }

  static ThemeMode getThemeMode() {
    final themeModeIndex =
        _settingsStorage.getInt('themeMode') ?? ThemeMode.system.index;
    return ThemeMode.values[themeModeIndex];
  }
}
```