

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ
МОНІТОРИНГУ ФІНАНСОВИХ ВИТРАТ
SOFTWARE IMPLEMENTATION OF A MOBILE APPLICATION TO MONITOR
FINANCIAL EXPENSES

Виконав: студент(ка) 4 курсу, групи КНТ-141

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПАЛАМАРЧУК А.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

Сергій СУББОТІН

«_____» _____ 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПАЛАМАРЧУКА Артема Вікторовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація мобільного застосунку для моніторингу фінансових витрат. Software Implementation of a Mobile Application to Monitor Financial Expenses

керівник проєкту (роботи) д.т.н., професор, ПОЛЯКОВ Михайло Олексійович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПОЛЯКОВ М.О., професор		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Артем ПАЛАМАРЧУК
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Михайло ПОЛЯКОВ
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
78 с., 40 рис., 4 табл., 3 дод., 18 джерел.

МОНІТОРНИГ ВИТРАТ, ЗБЕРЕЖЕННЯ, ФІНАНСИ, ЕКОНОМІЯ,
ЗАСТОСУНОК.

Об'єкт дослідження – моніторинг фінансових витрат і їх аналіз.

Предмет дослідження – програмне забезпечення мобільного застосунку для моніторингу фінансових витрат.

Мета роботи – розробка програмного забезпечення для моніторингу фінансових витрат

Матеріали, методи та технічні засоби: мова програмування Kotlin, система керування базами даних Room(SQLite).

Результати. Основним результатом роботи є створене програмне забезпечення, що надає інструмент для перегляду інформації про витрати, категорії, обмеження та спонукання про раціональне використання фінансів.

Висновки. Розроблене програмне забезпечення надає можливість користувачу розуміти, як правильно використовувати кошти для досягнення своїх цілей або мінімізації витрат.

Галузь використання – це люди які маюь за мету накопичувати та раціонально використовувати фінанси в наш час.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 78 pages, 40 figures, 4 tables, 3 appendixes, 15 sources.

PUBLIC FINANCING, CROWDFUNDING, CREATIVE FUNDING PROJECTS, CHARITY, CROWD FUNDING PLATFORMS.

The object of the study is the monitoring of financial costs and their analysis.

The subject of the study is the software of a mobile application for monitoring financial expenses.

The purpose of the work is to develop software for monitoring financial costs

Materials, methods and technical means: Kotlin programming language, Room database management system (SQLite).

Results. The main result of the work is the created software that provides a tool for viewing information about expenses, categories, restrictions and incentives for the rational use of finances.

Conclusions. The developed software provides an opportunity for the user to understand how to properly use funds to achieve their goals or minimize costs.

The field of use is people who aim to accumulate and rationally use finances in our time.

ЗМІСТ

С.

Перелік скорочень та умовних позначок	7
Вступ	8
1 Аналіз предметної області	9
1.1 Загальний стан проблеми обліку та моніторингу особистих фінансів	9
1.2 Огляд аналогів програмного забезпечення для моніторингу та обліку особистих фінансів	9
1.3 Висновки за розділом 1	17
2 Проєктування програми	19
2.1 Вибір засобів проєктування і розробки	19
2.2 Визначення функціональних вимог до програмного забезпечення	21
2.3 Проєктування структури бази даних	23
2.4 Структурна схема програмного проєкту	25
2.5 Висновки за розділом 2	29
3 Розробка програми	30
3.1 Алгоритм функціонування програмного забезпечення	30
3.2 Опис реалізованих класів	32
3.3 Висновки за розділом 3	38
4 Експлуатація та тестування програми	39
4.1 Призначення програми	39
4.2 Умови виконання програми	39
4.3 Виконання програми	39
4.4 Тестування програми	49
4.5 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилання	53
Додаток А Технічне завдання	55
Додаток Б Текст програми	59
Додаток В Слайди презентації	71

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
MVVM	– Model-View-ViewModel;
YNAB	– You Need A Budget.
БД	– база даних;
СУБД	– система управління базами даних;

ВСТУП

У сучасному світі управління особистими фінансами стає все більш актуальним. Багато людей прагнуть контролювати свої доходи та витрати, щоб ефективніше розподіляти бюджет і уникати зайвих витрат.

На сьогоднішній день існує безліч мобільних додатків для фінансового моніторингу, таких як Monefy, YNAB (You Need A Budget), Wallet by BudgetBakers. Проте вони мають низку недоліків, зокрема обмежений функціонал у безкоштовних версіях, відсутність експорту даних у зручні формати, відсутність підтримки конвертації валют тощо [1].

Метою даної дипломної роботи є розробка програмного забезпечення у вигляді мобільного додатку для моніторингу фінансових витрат, що дозволяє користувачам вести облік доходів та витрат, аналізувати статистику, встановлювати ліміти на категорії витрат та експортувати дані у формат Excel.

У даній роботі необхідно провести аналіз аналогів, спроектувати програмне забезпечення, розробити та протестувати мобільний застосунок. Результати роботи будуть представлені у вигляді пояснювальної записки та створеного програмного продукту.

Таким чином, запропоноване програмне забезпечення стане корисним інструментом для користувачів, які прагнуть раціонально управляти своїми фінансами та контролювати бюджет.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний стан проблеми обліку та моніторингу особистих фінансів

На сьогоднішній день існує безліч мобільних застосунків, які дозволяють користувачам здійснювати моніторинг власних фінансових витрат. Такі застосунки набули популярності завдяки їх зручності, доступності та можливості ефективного керування особистими фінансами.

Моніторинг фінансових витрат – це процес збору, реєстрації та аналізу даних про витрати користувача з метою покращення його фінансового стану та підвищення рівня фінансової дисципліни [2]. Зазвичай, такі застосунки пропонують користувачу можливість створення категорій витрат, ведення статистики та відображення фінансових звітів у зручному для сприйняття вигляді.

Мобільні застосунки для контролю витрат працюють за певним алгоритмом: користувач вносить інформацію про свої витрати, яка зберігається у базі даних, після чого застосунок аналізує дані та формує звіти. Це дозволяє користувачу відслідковувати свої фінансові операції та оптимізувати витрати відповідно до своїх фінансових цілей [3].

Такий підхід до моніторингу фінансів дозволяє не лише отримати чітке уявлення про структуру витрат, а й вчасно виявити непотрібні витрати та внести відповідні корективи у фінансове планування. Це є важливим інструментом для особистого та сімейного бюджетування, особливо в умовах економічної нестабільності.

1.2 Огляд аналогів програмного забезпечення для моніторингу та обліку особистих фінансів

На сучасному ринку існує значна кількість всесвітньо відомих мобільних застосунків для моніторингу фінансових витрат. Проведемо

детальний аналіз найпопулярніших рішень та порівняємо їх з нашим застосунком.

Один із популярних мобільних застосунків для ведення особистих фінансів – Monefy (рис. 1.1). Його розроблено з акцентом на простоту, швидкість і зручність ведення обліку витрат без зайвих налаштувань. Застосунок доступний на платформах Android та iOS і має мільйони завантажень у всьому світі [4].

Основна функціональність Monefy полягає у швидкому додаванні витрат і доходів вручну. Інтерфейс реалізовано таким чином, щоб користувач міг у кілька натискань внести нову транзакцію, вибравши відповідну категорію (їжа, транспорт, розваги тощо) та вказавши суму. Для зручності додано можливість ведення обліку у кількох валютах і налаштування синхронізації через Google Drive або Dropbox, що дозволяє зберігати дані та переносити їх між пристроями [5].

У розділі статистики Monefy відображає витрати у вигляді кругових діаграм, що дає змогу миттєво оцінити розподіл бюджету за категоріями. Також реалізовано функцію встановлення бюджетів, хоча й у спрощеному вигляді, без складних фінансових звітів або прогнозів. Застосунок більше підходить для тих, хто хоче просто і швидко фіксувати свої фінансові дії без глибокої аналітики [5-6].

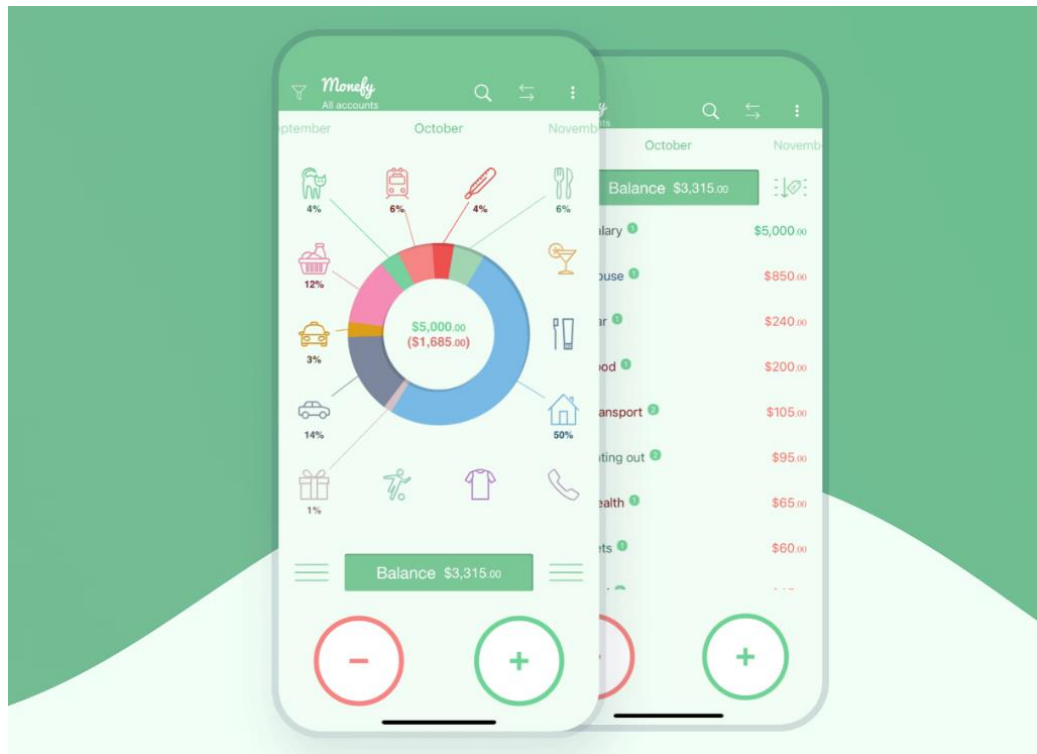


Рисунок 1.1 – Інтерфейс Monefy [7]

До переваг Monefy відносять:

- інтуїтивно зрозумілий інтерфейс;
- швидке додавання витрат;
- підтримка кількох валют і хмарної синхронізації;
- висока швидкість навіть на старих пристроях;
- захист даних за допомогою пароля.

До недоліків платформи відносять:

- обмежений функціонал у безкоштовній версії;
- відсутність автоматичної інтеграції з банками;
- слабо розвинені аналітичні інструменти;
- відсутність підтримки кредитного рейтингу або складних бюджетів;
- відсутність нагадувань про регулярні платежі.

YNAB (рис. 1.2) – це застосунок, що спеціалізується на бюджетуванні за методом «нульового бюджету», де кожен долар має чітке призначення. Головна ідея полягає в тому, щоб планувати витрати наперед, розподіляючи

весь доступний дохід по категоріях ще до моменту фактичного витрачання коштів. Це допомагає уникнути імпульсивних покупок і краще контролювати фінансові звички [8].

Завдяки цьому застосунку користувачам відкривається можливість ретельно планувати особистий чи сімейний бюджет. Наприклад, маючи щомісячний дохід у 2000 доларів, можна заздалегідь передбачити витрати на оренду, продукти, транспорт, дозвілля, заощадження та інші потреби, так, щоб наприкінці місяця не залишалося «вільних» коштів, що не мають конкретного призначення. У результаті створюється система, в якій кожна гривня чи долар працює на досягнення певної фінансової мети.

YNAB підтримує синхронізацію з банківськими рахунками, що дозволяє автоматично імпортувати транзакції, а також передбачає ручне введення витрат для точнішого контролю. Застосунок допомагає формулювати і досягати фінансові цілі, наприклад, накопичення на велику покупку чи створення фінансової «подушки». Окрему увагу приділено освітньому компоненту – платформа пропонує користувачам доступ до вебінарів, порад, інтерактивних курсів і статей із фінансової грамотності. Крім того, реалізовано можливість спільного доступу до бюджету для кількох осіб, що зручно для сімейного використання [9].

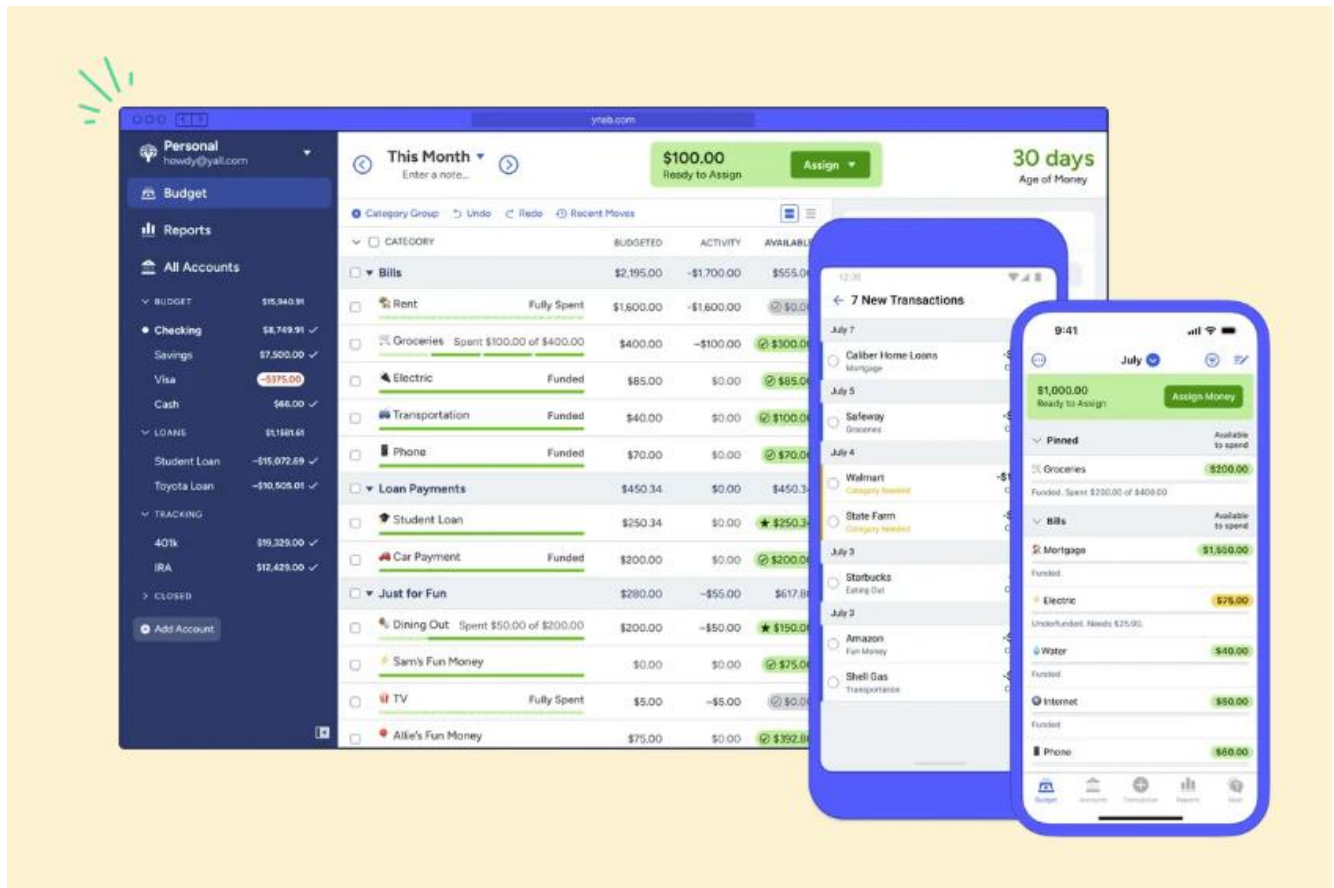


Рисунок 1.2 – Інтерфейс YNAB [10]

До переваг платформи YNAB відносять:

- ефективну методологію бюджетного планування;
- освітній компонент для підвищення фінансової грамотності;
- високий рівень захисту даних;
- детальну аналітику витрат;
- зручність сімейного використання;
- можливість планування довгострокових фінансових цілей;
- регулярне оновлення функціоналу з урахуванням потреб користувачів.

До недоліків YNAB відносять:

- платна підписка (\$14.99 на місяць або \$98.99 на рік);
- складна крива навчання для новачків;
- обмежена функціональність безкоштовної версії;
- відсутність автоматичної категоризації транзакцій;

- слабка інтеграція з деякими європейськими банками;
- перенасиченість інтерфейсу для користувачів, які шукають просто рішення.

Платформа Wallet by BudgetBakers (рис. 1.3) – це зручний багатофункціональний застосунок для керування особистими фінансами, який підтримує інтеграцію з понад 15 000 банків у більш ніж 60 країнах світу. Завдяки широкій міжнародній підтримці, Wallet вважається одним із найгнучкіших рішень для користувачів, які подорожують або мають рахунки у кількох країнах [11].

Одна з ключових переваг Wallet – підтримка понад 160 валют, включно з автоматичним оновленням валютного курсу. Це особливо зручно для тих, хто часто здійснює транзакції в іноземній валюті. Дані автоматично синхронізуються з банківськими рахунками, а за потреби транзакції можна вводити вручну – наприклад, витрати на базарі, готівкові розрахунки або приватні перекази, що не фіксуються банком.

Wallet пропонує гнучке налаштування категорій витрат, можливість додавати персоналізовані підкатегорії, планувати бюджети, формувати фінансові цілі, наприклад, накопичення на відпустку або покупку авто. Усе це супроводжується зручною візуалізацією даних – аналітика подається у вигляді діаграм, графіків та звітів, що дозволяє оцінити фінансову картину в динаміці. Наприклад, можна побачити, які категорії «з’їдають» найбільше грошей протягом місяця, або які витрати повторюються.

Wallet також підтримує додавання фотографій чеків, квитанцій та заміток до транзакцій, що зручно для детального обліку витрат. Завдяки нагадуванням про платежі та автоматичному дублюванню регулярних витрат (наприклад, оренди чи абонементи) можна уникнути прострочень і непередбачених витрат [12].

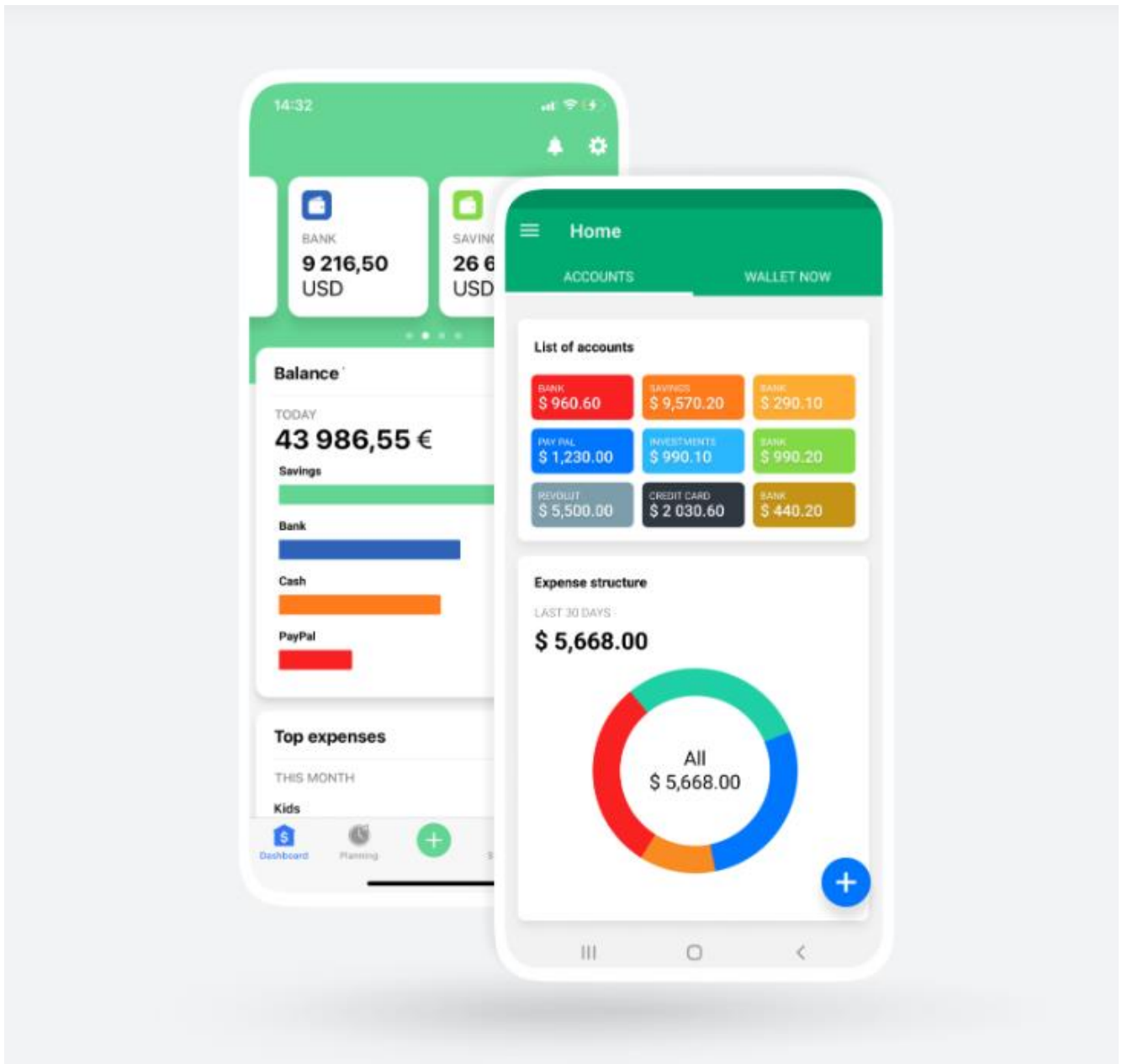


Рисунок 1.3 – Інтерфейс Wallet by BudgetBakers [13]

Переваги платформи:

- міжнародна підтримка: працює з банками у понад 60 країнах;
- мультивалютність: підтримує понад 160 валют для обліку витрат;
- гнучкі налаштування категорій, можливість створювати власні типи витрат;
- зручний інтерфейс, інтуїтивно зрозуміле керування і сучасний дизайн;
- функція нагадувань та регулярних транзакцій.

Недоліки платформи:

- обмежена функціональність безкоштовної версії, багато можливостей доступні лише за підпискою;
- платна підписка для розширених функцій, преміум-функції потребують щомісячної або річної оплати;
- обмежені можливості для інвестиційного планування: застосунок більше зосереджений на витратах, ніж на управлінні активами;
- відсутність функції фінансового консультування: немає персональних рекомендацій чи порад від фінансових експертів;
- відносно висока вартість преміум-підписки порівняно з конкурентами.

За результатами огляду трьох основних програмних застосунків для управління особистими фінансами було виконано порівняльний аналіз за ключовими критеріями (табл. 1.1).

Таблиця 1.1 – Порівняння з нашим рішенням FinanceTrack

Функція	Monefy	YNAB	Wallet	FinanceTrack
Ручне додавання транзакцій	Присутні	Присутні	Присутні	Присутні
Категоризація витрат	Автоматична	Ручна	Автоматична	Напів-автоматична
Встановлення бюджетів	Базова	Базова	Базова	Базова
Сповіщення про перевищення бюджету	Базова	Базова	Базова	Базова
Аналітика та звіти	Розширена	Розширена	Розширена	Базова
Підтримка різних валют	Обмежена	Обмежена	Розширена	Обмежена

Кінець таблиці 1.1

Функція	Monefy	YNAB	Wallet	FinanceTrack
Інвестиційний менеджмент	Базовий	Відсутній	Розширений	Відсутній
Офлайн-режим	Відсутній	Присутній	Відсутній	Присутній
Безпека даних	Стандартна	Висока	Висока	Висока
Локалізація для України	Відсутня	Відсутня	Відсутня	Присутня
Ціна	Безкоштовно з рекламою	Локалізація для України	Безкоштовно + платні послуги	Безкоштовно

Виходячи з аналізу, програмне забезпечення має забезпечувати такі властивості і можливості:

- зручний інтерфейс програмного забезпечення;
- ліміти на витрати з повідомленням;
- статистика та графіки;
- експорт у ексель;
- введення та аналіз доходів та витрат.

При визначенні концепції розроблюваного програмного забезпечення не ставилася за необхідність заміна аналогічних програмних застосунків, а використовувалася ідея втілення зручних методів для зручного використання та менеджменту фінансів.

1.3 Висновки за розділом 1

У цьому розділі було проаналізовано найпопулярніші мобільні застосунки для управління особистими фінансами: Mint, YNAB та Wallet. На основі їхнього функціоналу, переваг і недоліків визначено ключові орієнтири для реалізації власного рішення. Серед основних вимог – простота

використання, локалізація під українські реалії, а також можливість масштабування з урахуванням подальшого розвитку функціоналу.

Завданнями дипломної роботи є:

- провести аналіз предметної області та актуальних підходів до моніторингу фінансових витрат;
- сформулювати функціональні та нефункціональні вимоги до мобільного застосунку;
- розробити архітектуру та спроектувати основні компоненти програмного забезпечення;
- реалізувати та протестувати мобільний застосунок для моніторингу особистих фінансів з урахуванням потреб.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Для створення програмного забезпечення моніторингу фінансових витрат було обрано мову програмування Kotlin.

Kotlin – це сучасна статично типізована мова програмування, розроблена компанією JetBrains та офіційно підтримувана Google як пріоритетна мова для розробки Android-застосунків. Її синтаксис є лаконічним і зручним, що дозволяє зменшити обсяг коду до 30% у порівнянні з Java, без втрати функціональності чи читабельності [14].

Одна з ключових переваг Kotlin – повна сумісність із Java, що дає змогу легко інтегрувати існуючі Java-бібліотеки та компоненти. Крім того, мова підтримує як об'єктно-орієнтоване, так і функціональне програмування, має гнучку систему типів, включає механізми обробки нульових значень, що істотно знижує ризик помилок у рантаймі.

Завдяки активній підтримці з боку Google, екосистема Kotlin постійно розширюється [15]. Платформа Android поступово переходить на Kotlin як на основний інструмент, що спрощує використання таких сучасних інструментів, як:

- Jetpack Compose – декларативний фреймворк для створення UI;
- Room – зручна ORM-бібліотека для роботи з базами даних SQLite;
- Kotlin Coroutines – сучасний підхід до роботи з асинхронними операціями.

Таблиця 2.1 показує порівняння Kotlin з іншими мовами програмування, які також можуть використовуватися для розробки мобільних застосунків.

Таблиця 2.1 – Порівняння Kotlin з іншими мовами програмування

	Kotlin	Java	Flutter(Dart)
Кросплатформеність	Високий рівень	Низький рівень	Середній рівень
Швидкість розробки програми	Високий рівень	Високий рівень	Низький рівень
Швидкість роботи кінцевого продукту	Високий рівень	Низький рівень	Середній рівень
Легкість вивчення	Високий рівень	Низький рівень	Середній рівень
Підтримка різних інструментів	Високий рівень	Середній рівень	Низький рівень
Сучасність	Високий рівень	Низький рівень	Середній рівень
Зрозуміла документація	Високий рівень	Високий рівень	Високий рівень

Як ми можемо бачити в таблиці 2.1, Kotlin пропонує певні переваги порівняно з класичною Java, особливо щодо зручності написання коду та підтримки сучасних моделей [16]. Порівняно з DART (використовується у Flutter Framework), Kotlin дещо нижчий, ніж попередня платформа, але досягає стабільності та продуктивності на платформі Android.

Таблиця 2.2 – Порівняння Jetpack Compose з іншими Kotlin-фреймворками

	Jetpack Compose	XML UI (Android)	Flutter UI
Швидкість розробки	Висока	Висока	Висока
Гнучкість створення	Висока	Низька	Середня

Кінець таблиці 2.2

	Jetpack Compose	XML UI (Android)	Flutter UI
Інтеграція	Висока	Висока	Низька
Продуктивність	Висока	Низька	Середня
Масштабованість	Висока	Низька	Середня

Отже, було вирішено використовувати мову програмування Kotlin та вебфреймворк Jetpack Compose для реалізації програмного мобільного застосунку для моніторингу фінансових витрат.

2.2 Визначення функціональних вимог до програмного забезпечення

Програмний застосунок для обліку особистих фінансів розроблений для користувача мобільного пристрою, який буде основним актором системи.

Програма повинна задовольняти наступні функціональні вимоги для користувача:

- додавання, редагування та видалення транзакцій (доходів та витрат);
- перегляд історії транзакцій з можливістю фільтрації;
- встановлення, редагування та виділення лімітів витрат за категоріями;
- перегляд поточного балансу та фінансової статистики;
- експорт фінансових даних у формат Ексель;
- отримання сповіщення про наближення до встановлених лімітів витрат.

Для візуального сприйняття функціональних вимог для користувача було створено діаграму прецедентів на рис. 2.2

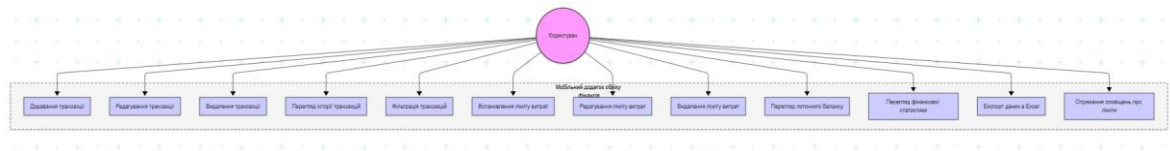


Рисунок 2.2 – Діаграма прецедентів користувача

Програма також має системні функції, які виконуються автоматично без безпосередньої участі користувача:

- періодична перевірка лімітів витрат;
- генерація сповіщень при досягненні порогових значень лімітів;
- збереження даних у локальній базі даних;
- автоматичне оновлення статистики та балансу при додаванні нових транзакції.

Для візуального сприйняття функціональних вимог для системних процесів було створено діаграму прецедентів на рис. 2.3.

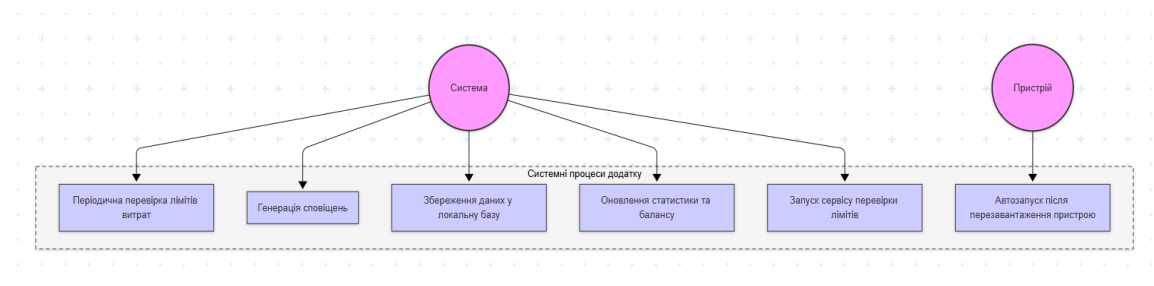


Рисунок 2.3 – Діаграма прецедентів системних процесів

Взаємодія між користувачем та системою відбувається через графічний інтерфейс мобільного застосунку, який забезпечує доступ до всіх функціональних можливостей. Система автоматично обробляє дані, зберігає їх у локальній базі даних та надає користувачу актуальну інформацію про його фінансовий стан.

Окрім основних функцій, застосунок також реалізує важливі нефункціональні вимоги:

- збереження конфіденційності фінансових даних (дані зберігаються лише на пристрої користувача);

- швидкодія та оптимізація використання ресурсів пристрою;
- інтуїтивно зрозумілий інтерфейс для зручного управління особистими фінансами;
- адаптація до різних розмірів екранів мобільних пристроїв;

Весь функціонал додатку спрямований на забезпечення зручного та ефективного інструменту для контролю особистого бюджету, планування витрат та аналізу фінансового стану користувача.

2.3 Проєктування структури бази даних

Система управління базами даних була Room – одна з Android Architecture Components, що надає високорівневу абстракцію над SQLite для більш безпечного доступу до бази даних з повним використанням переваг компіляції [17].

Переліки транзакцій користувача зберігаються в таблиці БД transactions з наступними полями:

- id – номер проєкту, який бере участь у зборі коштів;
- amount – сума транзакції;
- category – категорія транзакції;
- type – тип транзакції;
- date – дата здійснення транзакції;
- description – опис транзакції.

Ліміти витрат зберігаються в таблиці БД expense_limits з наступними полями:

- id – унікальний ідентифікатор ліміту;
- amount – сума ліміту;
- category – категорія витрат, для якої встановлено ліміт;
- period – період дії ліміту;
- startDate – дата початку дії ліміту;
- isActive – статус активності ліміту.

Для правильної роботи з типами даних в Room використовуються спеціальні конвертери, які зберігаються у відповідних класах:

- `DateConverter` – конвертер для перетворення дат у формат, доступний для зберігання в SQLite;
- `TransactionTypeConverter` – конвертер для перетворення типу транзакції (`TransactionType`) у формат для SQLite;
- `LimitPeriodConverter` – конвертер для перетворення періоду ліміту (`LimitPeriod`) у формат для SQLite.

Переліки статистичних даних не зберігаються в базі даних напряму, а розраховуються на основі існуючих транзакцій. Для представлення статистичних даних використовуються наступні класи моделей:

- `CategorySummary` – інформація про суму, кількість і відсоток транзакцій по категорії;
- `BalanceSummary` – інформація про загальні суми доходів, витрат і баланс за період.

Окремо необхідно розглянути зв'язки між сутностями БД.

Зв'язок між сутністю `Transaction` та `TransactionType` один-до-одного: кожна транзакція має один тип (дохід або витрата).

Зв'язок між сутністю `ExpenseLimit` та `LimitPeriod` один-до-одного: кожен ліміт витрат має один період дії (щоденний, щотижневий, щомісячний, щорічний).

Зв'язок між сутністю `Category` та `Transaction` один-до-багатьох: одна категорія може мати багато транзакцій.

Зв'язок між сутністю `Category` та `ExpenseLimit` один-до-одного: для кожної категорії може бути встановлений один ліміт витрат для кожного типу періоду.

Взаємодія з базою даних відбувається через відповідні `Data Access Object (DAO)` інтерфейси:

- `TransactionDao` – містить методи для роботи з транзакціями (додавання, видалення, отримання списку, фільтрація);

– ExpenseLimitDao – містить методи для роботи з лімітами витрат (додавання, видалення, редагування, отримання списку активних лімітів).

Для кешування даних та отримання даних з різних джерел (база даних, API) використовується патерн Repository:

– TransactionRepository – сховище для роботи з транзакціями та лімітами витрат;

– StatisticsRepository – сховище для отримання статистичних даних на основі транзакцій.

Основна база даних створюється за допомогою абстрактного класу AppDatabase, який розширює RoomDatabase і визначає необхідні DAO та сутності. Міграції бази даних виконуються автоматично через механізм fallbackToDestructiveMigration, який відповідає за створення нової версії бази даних при зміні схеми.

2.4 Структурна схема програмного проєкту

Для розробки мобільного додатку було використано архітектурний патерн Model-View-ViewModel (MVVM), який забезпечує чітке розділення відповідальності між компонентами та покращує тестованість коду (рис. 2.6) [18].

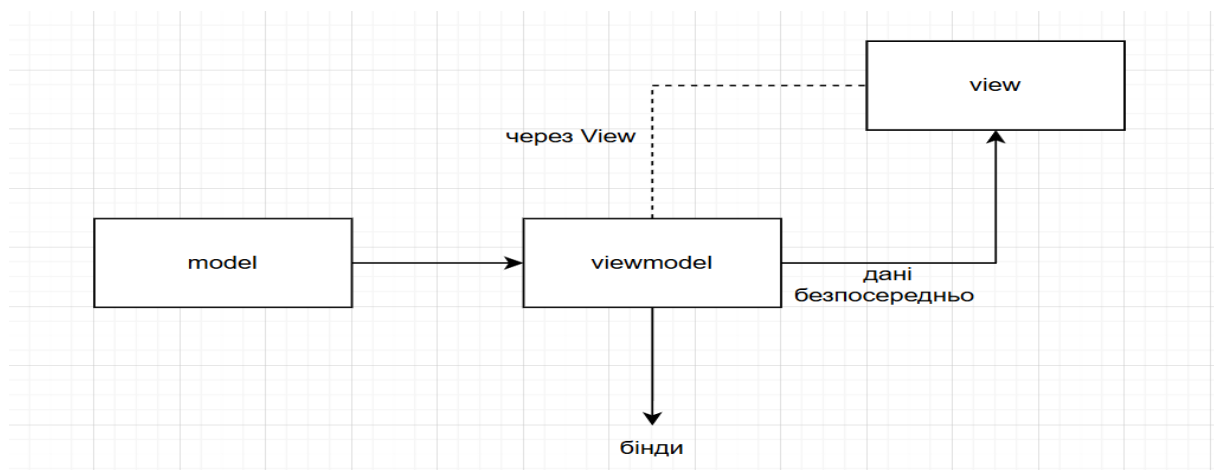


Рисунок 2.6 – Архітектурний патерн Model-View-ViewModel

Патерн MVVM складається з таких компонентів:

- model – відповідає за основні дані та бізнес-логіку програми;
- view – відповідає за візуалізацію даних, які зберігаються в моделі;
- viewmodel – проміжний шар між моделями і переглядом, який готує

дані для відображення та обробляє користувацьке введення.

Архітектура додатку базується на сучасних компонентах Android Architecture Components і складається з кількох ключових пакетів, які призначені для внутрішньої роботи програми [18].

Пакет model (рис. 2.7) містить класи даних, які представляють основні сутності застосунку, такі як Transaction (транзакція) та ExpenseLimit (ліміт витрат). Transaction зберігає інформацію про фінансові операції користувача: суму, категорію, тип, дату та опис. ExpenseLimit представляє обмеження витрат за категоріями з вказанням максимальної суми та періоду дії.

Ці класи використовуються для зберігання даних та визначення відносин між ними.

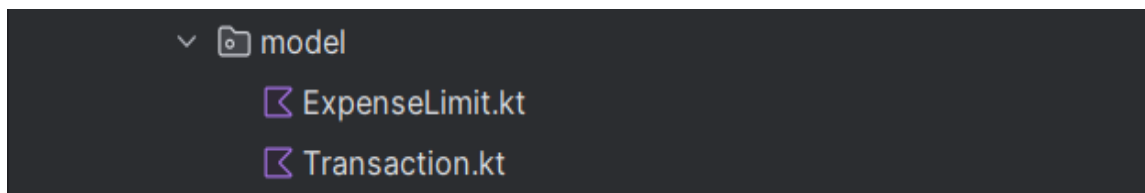


Рисунок 2.7 – Директорія model

Пакет ui (рис. 2.8) містить компоненти користувацького інтерфейсу, створені з використанням Jetpack Compose. Тут знаходяться всі екрани додатку, включаючи головний екран, екран додавання транзакцій, екран статистики та інші. Пакет організовано в підкаталоги: limits для роботи з лімітами витрат, statistics для екранів статистики, theme для стилізації інтерфейсу та transactions для управління транзакціями.

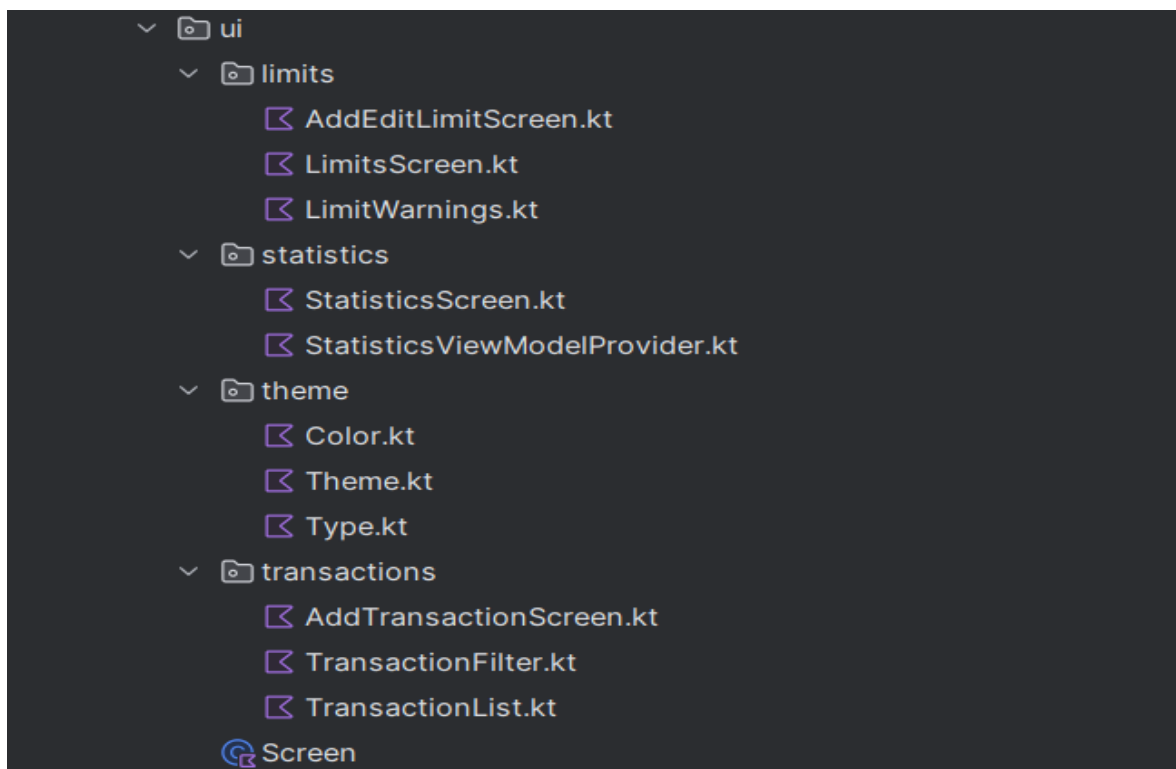


Рисунок 2.8 – Директорія ui

Пакет data (рис. 2.9) містить класи для роботи з даними та їх зберіганням. Тут знаходяться DAO-інтерфейси (Data Access Object) для транзакцій та лімітів витрат, клас бази даних AppDatabase та репозиторії для доступу до даних. Також пакет включає конвертери типів для забезпечення правильного збереження дат та перелічень у базі даних Room.

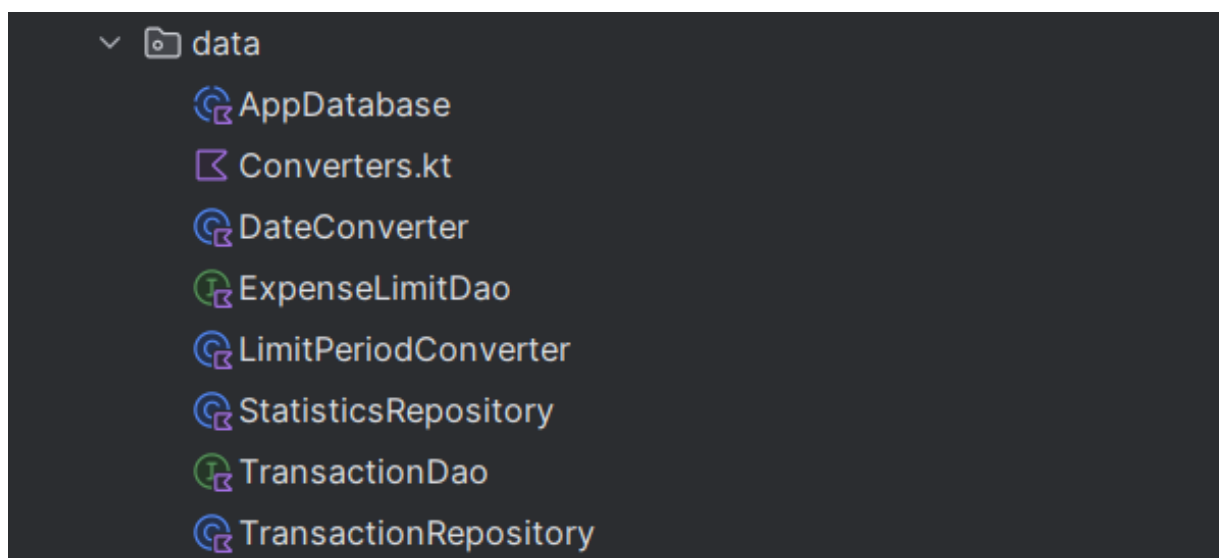


Рисунок 2.9 – Директорія data

Пакет `viewmodel` (рис. 2.10) містить класи `ViewModel`, які служать з'єднувальною ланкою між шаром даних та користувацьким інтерфейсом. Вони відповідають за підготовку даних для відображення та обробку дій користувача.

`MainViewModel` відповідає за основну функціональність застосунку, включаючи управління транзакціями, лімітами витрат та експортом даних. Він надає `LiveData`-об'єкти для спостереження за змінами в даних та містить методи для додавання, видалення та редагування записів. Також цей `ViewModel` включає логіку для розрахунку поточного стану лімітів витрат та формування сповіщень.

`StatisticsViewModel` спеціалізується на аналізі фінансових даних та підготовці статистичної інформації. Він забезпечує функціональність для вибору періоду аналізу (тиждень, місяць, квартал, рік) та формування відповідних аналітичних даних у вигляді категорійних сумарних показників та балансових звітів.

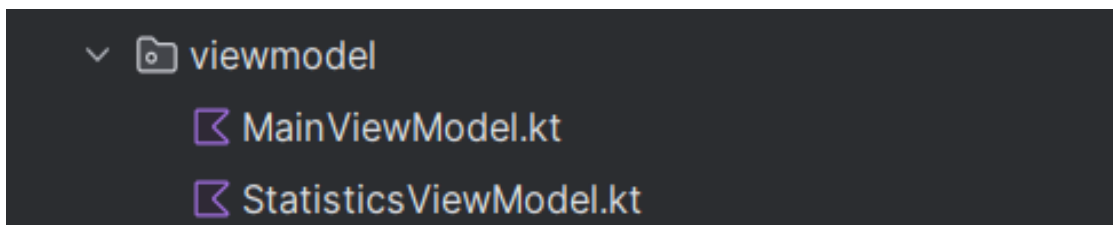


Рисунок 2.10 – Директорія `viewmodel`

Пакет `notification` (рис. 2.11) містить компоненти для роботи з системою сповіщень Android, включаючи `LimitCheckWorker` та `NotificationHelper`, які відповідають за перевірку лімітів витрат та надсилання сповіщень користувачу.

`LimitCheckWorker` використовує `WorkManager` для періодичних перевірок витрат у фоновому режимі, а `BootCompletedReceiver` забезпечує запуск сервісу після перезавантаження пристрою.

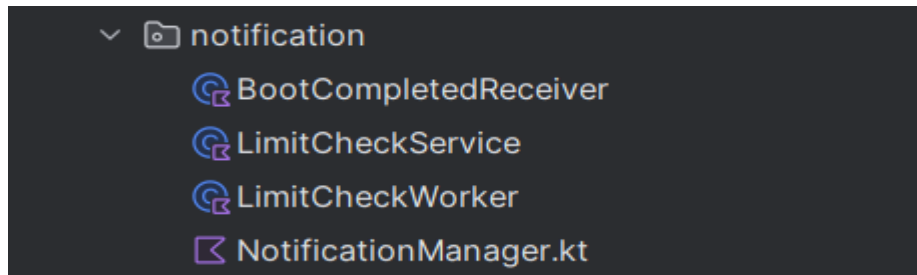


Рисунок 2.11 – Директорія notification

Використання архітектури MVVM у поєднанні з Jetpack Compose та іншими сучасними бібліотеками Android дозволило створити добре структурований, масштабований та легкий у підтримці застосунок. Такий підхід забезпечує чітке розділення відповідальності між компонентами, покращує тестованість та дозволяє ефективно керувати життєвим циклом даних.

2.5 Висновки за розділом 2

В цьому розділі було обрано такі засоби проєктування та подальшої розробки: мову програмування Kotlin, фреймворк Android Jetpack, архітектурний шаблон Model-View-ViewModel (MVVM), локальну базу даних Room (SQLite). Представлено діаграми прецедентів користувача та системних процесів. Реалізовано проєктування бази даних, розроблено архітектуру додатку та інтерфейс користувача.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програмного забезпечення

Алгоритм функціонування програмного забезпечення розрахований на взаємодію користувача з мобільним додатком обліку фінансів, який було розроблено для платформи Android.

Після запуску мобільного додатку користувачеві відразу відображається головний екран із поточним балансом та списком останніх фінансових транзакцій. Користувач може виконувати різноманітні дії для управління своїми фінансами.

Основними функціональними можливостями додатку є:

- управління транзакціями;
- робота з лімітами витрат;
- перегляд статистики;
- експорт даних.

Порядок роботи з функціями програми всіх типів користувачів показаний на рис. 3.1

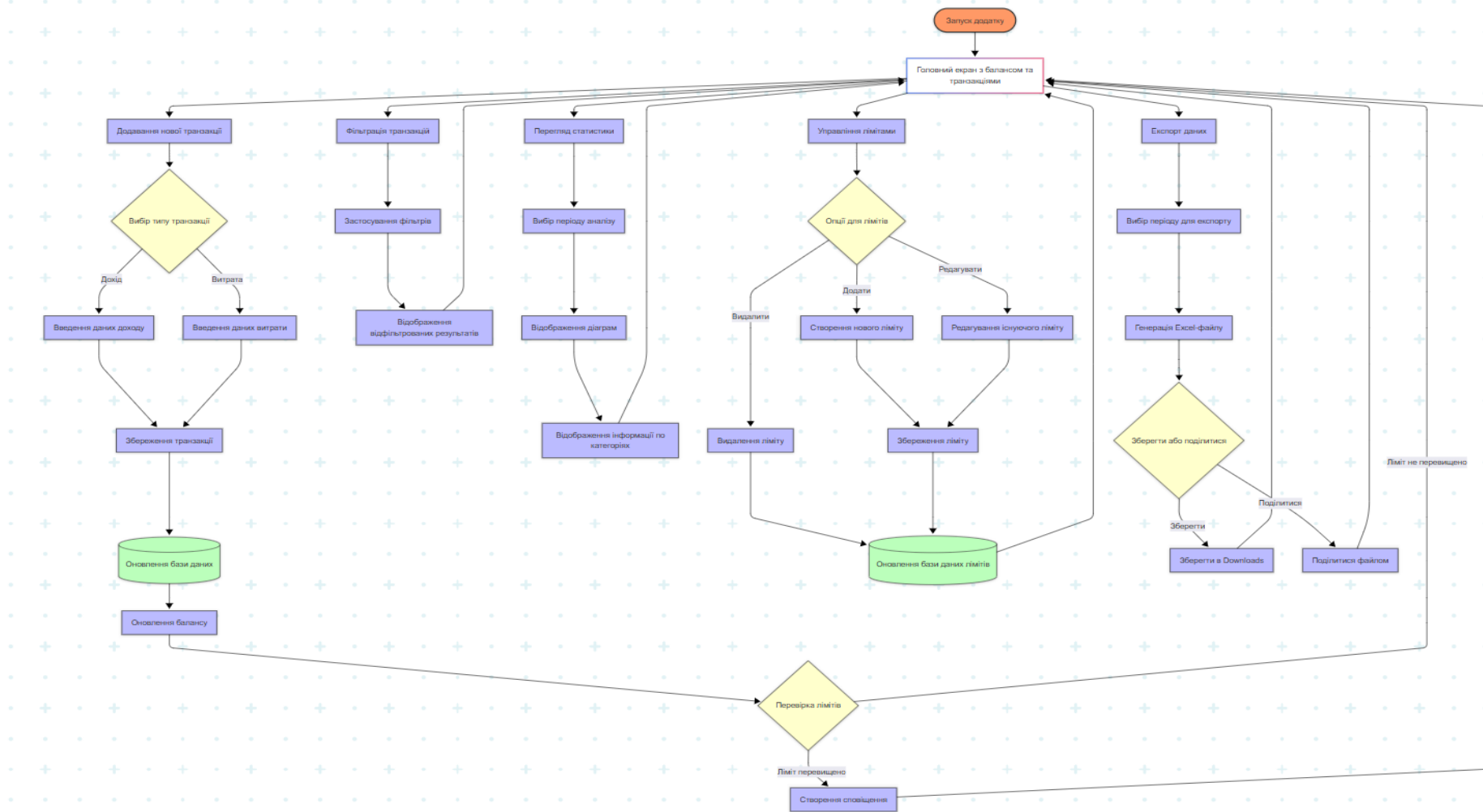


Рисунок 3.1 – Схема функціонування

3.2 Опис реалізованих класів

При розробці мобільного застосунку для моніторингу фінансових витрат було застосовано архітектурний шаблон MVVM, що забезпечило чітке розділення відповідальності між різними компонентами системи. Розглянемо основні класи, що були реалізовані в проєкті.

MainActivity головний клас застосунку, що відповідає за ініціалізацію компонентів та забезпечує життєвий цикл програми:

- onCreate() – метод ініціалізації, де відбувається налаштування інтерфейсу, перевірка дозволів та запуск сервісів;
- startLimitCheckService() – запускає фоновий сервіс для моніторингу лімітів витрат;
- checkNotificationPermission() – перевіряє та запитує дозвіл на показ сповіщень;
- checkStoragePermissions() – перевіряє та запитує дозвіл на доступ до сховища.

Клас ініціалізує два ключових ViewModel:

- MainViewModel – для управління транзакціями та лімітами;
- StatisticsViewModel – для роботи зі статистичними даними;
- allTransactions – список усіх транзакцій як LiveData;
- balance – поточний баланс як LiveData;
- allLimits – активні ліміти витрат як LiveData;
- insert(transaction) – додає нову транзакцію та перевіряє ліміти;
- delete(transaction) – видаляє транзакцію та оновлює стан лімітів;
- saveLimit(limit) – зберігає або оновлює ліміт витрат;
- deleteLimit(limit) – видаляє ліміт витрат;
- getLimitInfo(category, limitPeriod) – отримує детальну інформацію про використання ліміту;
- checkLimitsForNotifications() – перевіряє стан лімітів та надсилає сповіщення;

- `exportTransactionsToExcel(...)` – експортує дані про транзакції у формат Excel;

- `calculatePeriodDates(limit)` – обчислює дати початку та кінця періоду для ліміту.

`StatisticsViewModel` відповідає за обробку статистичних даних:

- `TimePeriod` – перелік доступних періодів для аналізу (тиждень, місяць, квартал, рік);

- `selectedPeriod` – поточний обраний період як реактивний потік;

- `expensesByCategory` – статистика витрат за категоріями;

- `incomesByCategory` – статистика доходів за категоріями;

- `balanceSummary` – загальний фінансовий звіт за період;

- `setPeriod(period)` – змінює поточний період аналізу;

- `calculatePeriodDates(period)` – обчислює дати початку та кінця обраного періоду.

`BootCompletedReceiver` приймач широкомовних повідомлень, який запускає сервіси після завантаження пристрою:

- `onReceive(context, intent)` – обробляє подію завантаження пристрою та запускає сервіс перевірки лімітів;

- `LimitCheckService` сервіс, що працює у фоновому режимі для моніторингу лімітів витрат;

- `onCreate()` – ініціалізує сервіс та допоміжні компоненти;

- `onStartCommand(...)` – запускає перевірку лімітів при старті сервісу;

- `checkLimits()` – перевіряє стан усіх активних лімітів витрат;

- `calculatePeriodDates(period)` – обчислює дати початку і кінця періоду для ліміту;

- `LimitCheckWorker` компонент `WorkManager` для періодичної перевірки лімітів витрат;

- `doWork()` – основний метод, що виконується за розкладом;

- `calculatePeriodDates(period)` – обчислює дати періоду для аналізу

витрат.

NotificationHelper клас для роботи з системними сповіщеннями:

- createNotificationChannel() – створює канал для сповіщень (необхідно для Android 8+);
- showLimitWarningNotification(limit, percentage) – показує сповіщення про наближення до ліміту.

AppDatabase центральний клас системи зберігання даних, що забезпечує доступ до локальної бази:

- @Database(entities = [Transaction::class, ExpenseLimit::class], version = 2, exportSchema = false) – анотація, що визначає основні сутності бази даних та версію схеми;
- @TypeConverters(DateConverter::class, TransactionTypeConverter::class, LimitPeriodConverter::class) – визначає класи-конвертери для типів, які не підтримуються SQLite безпосередньо;
- abstract fun transactionDao(): TransactionDao – абстрактний метод для отримання об'єкту доступу до транзакцій;
- abstract fun expenseLimitDao(): ExpenseLimitDao – абстрактний метод для отримання об'єкту доступу до лімітів витрат;
- companion object – реалізує шаблон Singleton для бази даних;
- getDatabase(context) – отримує або створює екземпляр бази даних;

Використовує блокування для гарантування потокової безпеки. Застосовує fallbackToDestructiveMigration() для спрощеної міграції схеми.

DateConverter перетворює типи Date для зберігання в базі даних:

- fromTimestamp(value: Long?): Date? – конвертує числове значення Long у тип Date;
- dateToTimestamp(date: Date?): Long? – перетворює Date у Long для зберігання;
- TransactionTypeConverter забезпечує зберігання перелічуваного типу TransactionType;

- `fromTransactionType(type: TransactionType): String` – перетворює тип транзакції в рядок;
- `toTransactionType(value: String): TransactionType` – відновлює тип транзакції з рядка.

TransactionDao Інтерфейс для операцій з транзакціями:

- `getAllTransactions(): Flow<List<Transaction>>` – отримує всі транзакції як реактивний потік;
- `insertTransaction(transaction: Transaction)` – додає нову транзакцію до бази;
- `deleteTransaction(transaction: Transaction)` – видаляє транзакцію з бази;
- `getBalance(): Flow<Double>` – обчислює загальний баланс як різницю доходів і витрат;
- `getCategoryExpensesBetweenDates(category, startDate, endDate): Flow<Double>` – обчислює витрати за категорією в заданому періоді;
- `getTransactionsByTypeAndDateRange(type, startDate, endDate): Flow<List<Transaction>>` – отримує транзакції за типом та періодом;
- `getAllTransactionsInRange(startDate, endDate): Flow<List<Transaction>>` – отримує всі транзакції в заданому періоді;
- `getTransactionsAfterDate(date): Flow<List<Transaction>>` – отримує транзакції після вказаної дати;
- `getTransactionsBeforeDate(date): Flow<List<Transaction>>` – отримує транзакції до вказаної дати.

ExpenseLimitDao Інтерфейс для роботи з лімітами витрат:

- `getAllActiveLimits(): Flow<List<ExpenseLimit>>` – отримує всі активні ліміти;
- `insertLimit(limit: ExpenseLimit)` – додає новий ліміт витрат;
- `updateLimit(limit: ExpenseLimit)` – оновлює існуючий ліміт;
- `deleteLimit(limit: ExpenseLimit)` – видаляє ліміт витрат;
- `getLimitForCategory(category): Flow<ExpenseLimit?>` – отримує

ліміт для конкретної категорії.

TransactionRepository центральний репозиторій для роботи з фінансовими даними:

- allTransactions: Flow<List<Transaction>> – потік усіх транзакцій;
- balance: Flow<Double> – потік балансу коштів;
- allLimits: Flow<List<ExpenseLimit>> – потік усіх лімітів витрат;
- insert(transaction: Transaction) – додає нову транзакцію;
- delete(transaction: Transaction) – видаляє транзакцію;
- insertLimit(limit: ExpenseLimit) – додає новий ліміт витрат;
- updateLimit(limit: ExpenseLimit) – оновлює ліміт витрат;
- deleteLimit(limit: ExpenseLimit) – видаляє ліміт витрат;
- getLimitForCategory(category): Flow<ExpenseLimit?> – отримує ліміт для категорії;
- getCategoryExpenses(category, startDate, endDate): Flow<Double> – отримує суму витрат;
- getTransactionsInDateRange(startDate, endDate): Flow<List<Transaction>> – фільтрує транзакції за датами;
- getTransactionsAfterDate(date): Flow<List<Transaction>> – отримує транзакції після дати;
- getTransactionsBeforeDate(date): Flow<List<Transaction>> – отримує транзакції до дати.

Репозиторій інкапсулює логіку доступу до даних, абстрагуючи ViewModel від безпосередньої взаємодії з DAO.

StatisticsRepository репозиторій для формування аналітичних даних:

- getExpenseSummaryByCategory(startDate, endDate);
- Flow<List<CategorySummary>> – обчислює статистику витрат за категоріями;
- getIncomeSummaryByCategory(startDate, endDate): Flow<List<CategorySummary>> – обчислює статистику доходів за категоріями;

- `getBalanceSummary(startDate, endDate): Flow<BalanceSummary>` – формує узагальнений фінансовий звіт;
- `ExcelExporter` основний клас, що відповідає за формування та збереження Excel-файлів;
- `defaultFileName` – базова назва файлу для експорту;
- `dateFormatter` – форматування дат для звіту, адаптоване для української локалізації;
- `filenameDateFormatter` – форматування дати для включення в ім'я файлу;
- `exportTransactionsToExcel(transactions, startDate, endDate): Pair<Uri, String>?` – головний метод експорту, що виконується в контексті введення/виведення (`Dispatchers.IO`). Створює файл у директорії `Downloads`. Формує Excel-документ з використанням бібліотеки `JXL`. Налаштовує локалізацію (українською), компонує структуровані дані в таблицю. Форматує заголовки та дані відповідно до типів (доходи/витрати) та обчислює підсумкові значення (загальний дохід, витрати, баланс). Оптимізує ширину стовпців, повертає `Uri` та ім'я створеного файлу;
- `createShareFileIntent(uri, fileName): Intent` – створює намір для поширення файлу. Визначає тип файлу як `Excel`, додає метадані (тему, опис) та налаштовує дозволи для доступу до контенту;
- `createExcelFile(): File` – створює файл у зовнішньому сховищі. Генерує унікальне ім'я з часовою міткою, використовує публічну директорію `Downloads` та повертає об'єкт `File` для запису.

Методи форматування:

- `getHeaderStyle()` – стиль для заголовків (жирний шрифт, центрування, фон);
- `getIncomeStyle()` – стиль для доходів (зелений колір);
- `getExpenseStyle()` – стиль для витрат (червоний колір);
- `getDateStyle()` – стиль для дат (спеціальне форматування);
- `getTotalStyle()` – стиль для підсумкових рядків (жирний шрифт,

вирівнювання вправо).

ExportScreen – функція, що надає інтерфейс для налаштування та запуску експорту:

- isDateRangeEnabled – прапорець для увімкнення фільтрації за датами;
- startDate та endDate – дати початку та кінця періоду;
- isExporting – прапорець процесу експорту;
- showFileActions – прапорець для відображення дій з файлом;
- lastExportedFileUri та lastExportedFileName – результати експорту.

Screen клас, що визначає різні екрани застосунку:

- Main – головний екран із балансом та транзакціями;
- AddTransaction – екран додавання/редагування транзакцій;
- Limits – екран управління лімітами витрат;
- AddEditLimit – екран додавання/редагування ліміту;
- Statistics – екран статистики та аналітики;
- Export – екран експорту даних.

3.3 Висновки за розділом 3

Було надано підсумки розробки програмного забезпечення, що містять опис архітектури реалізованого рішення відповідно до актуальної структурної діаграми застосунку, а також розкрито особливості організації ключових класів, втілених у коді.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Мобільний застосунок призначений, насамперед, для допомоги користувачам ефективно відстежувати та контролювати свої фінансові витрати, вести облік доходів та витрат, встановлювати ліміти витрат по категоріях, аналізувати фінансовий стан за допомогою статистики та експортувати дані для подальшого використання.

4.2 Умови виконання програми

До мобільного пристрою, на якому виконується програмне забезпечення, висуваються наступні мінімальні вимоги:

- операційна система Android 6.0 або новіша;
- процесор з тактовою частотою не менше 1.2 ГГц (рекомендовано 1.8 ГГц або вище);
- 2 ГБ оперативної пам'яті (рекомендовано 3 Гб або більше);
- 100 Мб вільного місця на внутрішньому накопичувачі для встановлення програми та 50 Мб для зберігання даних.

4.3 Виконання програми

Для виконання програми необхідно запустити локальний емулятор. Після цього програма стає доступною для користувача (рис. 4.1). Користувач може переглядати головний екран, баланс та останні транзакції.

Головна сторінка застосунку має інтуїтивно зрозумілий інтерфейс. У верхній частині екрану розташована назва застосунку "Мої фінанси" та кнопки навігації. Під нею знаходиться поле пошуку, яке дозволяє швидко знаходити транзакції за категорією або описом.

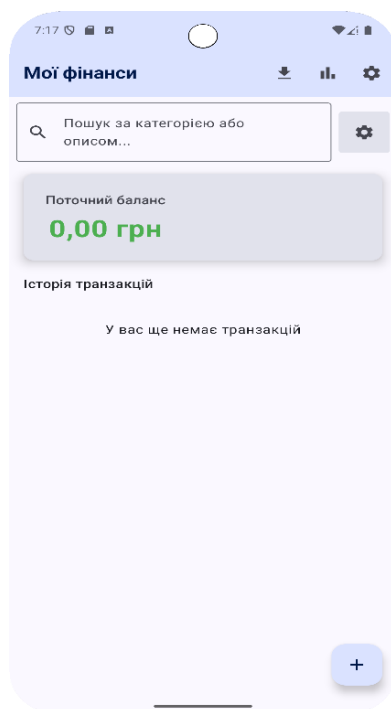


Рисунок 4.1 – Головна сторінка, яка відкривається при запуску

Далі користувач може додати нові транзакції, переглянути статистику або виставити ліміт.

При переході на сторінку додавання транзакції (рис. 4.2), відображається форма для введення суми, категорії, опису (за бажанням) та вибору типу транзакції. Якщо дані введені некоректно, програма повідомляє про це користувача, підкреслює некоректно введене текстове поле (рис. 4.3).

7:18

← Додати транзакцію

Сума
45000

Категорія
Salary

Опис (необов'язково)
|

Тип транзакції

☐ Витрата ☒ Дохід

Додати

Navigation icons: microphone, close, back, home, menu

Рисунок 4.2 – Сторінка авторизації

← Додати транзакцію

Сума
dsgf

Введіть коректну суму більше нуля

Категорія
|

Опис (необов'язково)

Тип транзакції

☒ Витрата ☐ Дохід

Додати

Рисунок 4.3 – Повідомлення про помилку при введенні суми

Після додавання транзакція відображається на головному екрані (рис. 4.4). На оновленому головному екрані тепер відображається поточний баланс, який автоматично розраховано на основі всіх доданих транзакцій. Баланс виділено зеленим кольором, що свідчить про позитивне значення.

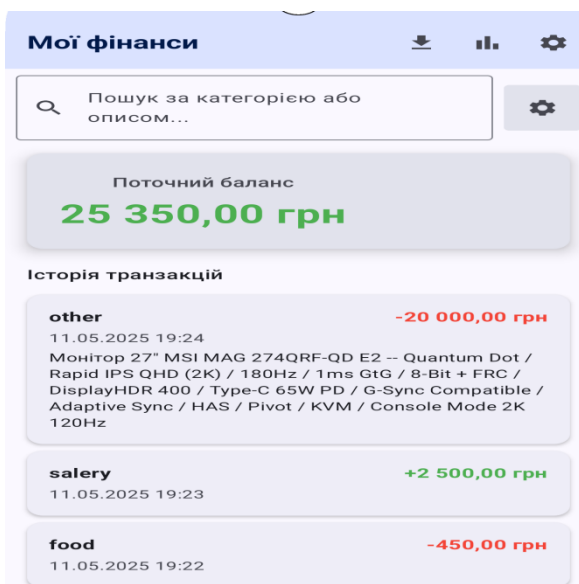


Рисунок 4.4 – Додана транзакція

Після введення коректних даних, користувач може вибрати фільтри для транзакцій (рис. 4.5). Та виконати пошук транзакцій за категорією або описом (рис. 4.6).

На рисунку 4.5 відображено розгорнуту форму фільтрації транзакцій, яка дозволяє користувачам гнучко налаштовувати відображення фінансових операцій за різними параметрами. У верхній частині форми користувач може вибрати типи транзакцій для відображення - "Доходи" та/або "Витрати".

Після застосування фільтрів система автоматично перераховує баланс, показуючи суму лише для відфільтрованих транзакцій.

7:21

Мої фінанси

Пошук за категорією або описом...

Тип транзакції

☐ Доходи ☒ Витрати

Період

Початкова дата Кінцева дата

Діапазон сум

Від: 0,00 грн До: 45 000,00 грн

Скинути все Застосувати

Фільтри активні

Поточний баланс

45 800,00 грн

Баланс за фільтром

-1 700,00 грн

Показано 2 з 4 транзакцій

Історія транзакцій

sport -1 200,00 грн

Рисунок 4.5 – Форма для фільтру транзакцій

food

Поточний баланс

25 350,00 грн

Баланс за фільтром

-950,00 грн

Показано 2 з 6 транзакцій

Історія транзакцій

food -450,00 грн

11.05.2025 19:22

food -500,00 грн

11.05.2025 19:19

mcdonalds

Рисунок 4.6 – Пошук транзакцій за категоріями та описом

Зображено інтерфейс додавання нового ліміту витрат. Користувач може вказати категорію, встановити максимальну суму та вибрати період дії ліміту (рис. 4.7. В даному випадку обрано щомісячний період, що є найпоширенішим варіантом для контролю регулярних витрат на харчування.

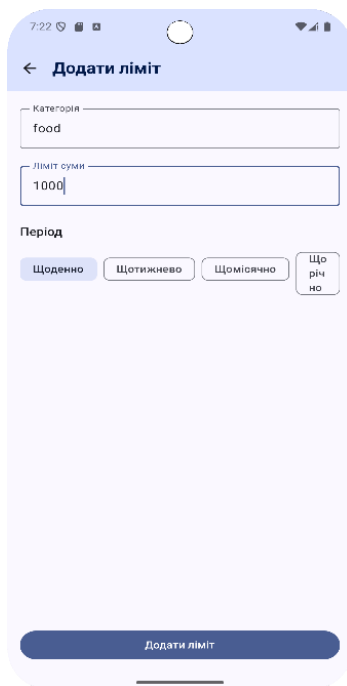


Рисунок 4.7 – Сторінка з додавання ліміту

Головний екран застосунку з відображенням поточного балансу та переліком усіх транзакцій. У верхній частині присутнє попередження про наближення до ліміту витрат, що виділене жовтим кольором для привернення уваги користувача (рис. 4.8).

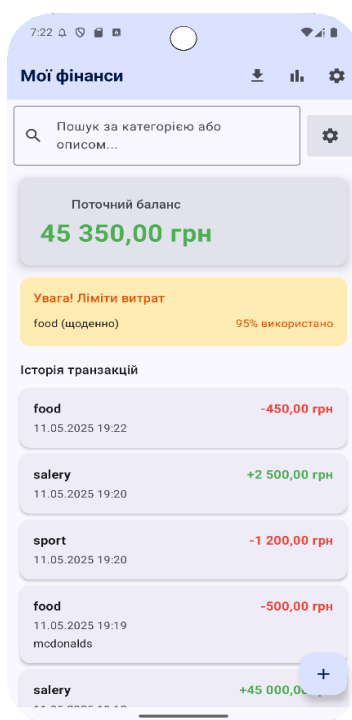


Рисунок 4.8 – Демонстрація ліміту

Показано системне сповіщення, яке з'являється на пристрої, коли витрати в певній категорії наближаються до встановленого ліміту (рис. 4.9). Повідомлення інформує користувача про необхідність контролювати подальші витрати в цій категорії.

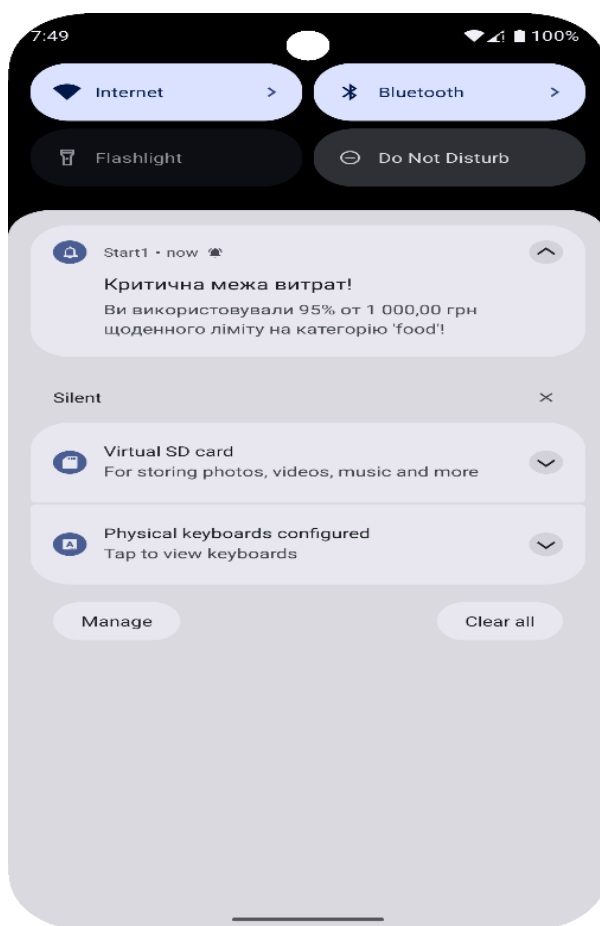


Рисунок 4.9 – Повідомлення про перевищення витрат

Окрім того, користувач може переглядати статистику та вибирати актуальний для нього проміжок часу за який він хоче переглянути статистику (рис. 4.10). Також для наглядності, була реалізована кругова діаграма для візуалізації (рис. 4.11).

На рисунку 4.10 представлено екран статистики фінансів з текстовим відображенням даних. У верхній частині екрану користувач може вибрати період для аналізу: тиждень, місяць, квартал, рік або весь час. У центральній частині показано підсумки за обраний період: доходи, витрати та баланс.

Нижче наведено секцію «Топ категорій витрат», де відображено категорії з найбільшими витратами та їх відсоткове співвідношення від загальної суми витрат.



Рисунок 4.10 – Перегляд статистики

Рисунок 4.11 демонструє той самий екран статистики, але з додаванням кругової діаграми для візуалізації структури витрат. Кольорове відображення дозволяє швидко оцінити пропорції витрат по різних категоріях.

Візуалізація даних у вигляді діаграми значно покращує сприйняття інформації та дозволяє користувачу миттєво визначити, на які категорії припадає найбільша частка витрат, що може допомогти в оптимізації бюджету.



Рисунок 4.11 – Кругові діаграми для перегляду статистики

Наступною є функція експорту всіх даних в файл Excel (рис. 4.12). Користувач може вибрати період за який він хоче завантажити статистику. Також відразу є функція «відкриття файлу» та «поділитись файлом» для зручності.

На рисунку 4.12 зображено екран експорту фінансових даних, який дозволяє користувачам зберігати інформацію про свої транзакції у форматі Excel для подальшого аналізу або зберігання. У верхній частині екрану міститься пояснення, що експортований файл буде збережено в директорії «Download» пристрою.

Ця функція особливо корисна для користувачів, які бажають зробити більш глибокий аналіз своїх фінансів за допомогою розширених можливостей Excel або зберегти резервну копію своїх фінансових даних.

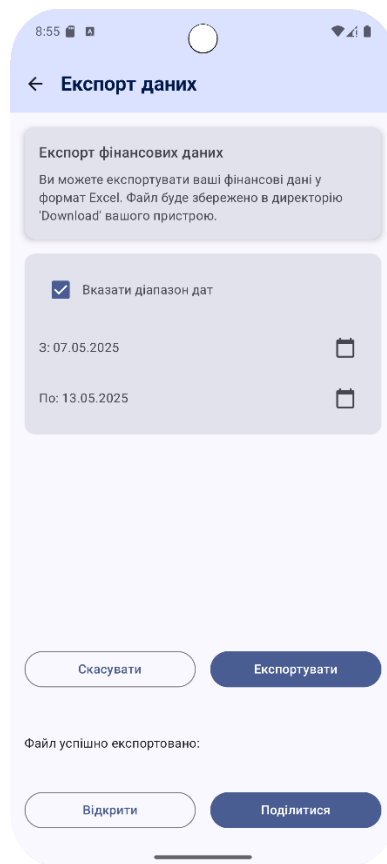


Рисунок 4.12 – Сторінка з експортом даних

4.4 Тестування програми

Для перевірки працездатності та відповідності функціональним вимогам мобільного застосунку для моніторингу фінансових витрат було проведено комплексне тестування. Результати виконаного тестування за перевіркою функціональних вимог до програмного забезпечення представлено в табл. 4.1.

Таблиця 4.1 – Чеклист тестування програмного забезпечення

Перевірка	Результат
1	2
Додавання нової транзакції	Пройдено
Видалення транзакції	Пройдено
Фільтрація транзакцій за типом	Пройдено
Фільтрація транзакцій за датою	Пройдено

Кінець таблиці 4.1

1	2
Фільтрація транзакцій за сумою	Пройдено
Пошук транзакцій за категорією	Пройдено
Пошук транзакцій за описом	Пройдено
Встановлення нового ліміту витрат	Пройдено
Редагування існуючого ліміту витрат	Пройдено
Видалення ліміту витрат	Пройдено
Автоматична перевірка лімітів витрат	Пройдено
Відображення сповіщень про наближення до ліміту	Пройдено
Перегляд поточного балансу	Пройдено
Перегляд статистики витрат за різні періоди	Пройдено
Відображення діаграми витрат за категоріями	Пройдено
Відображення діаграми доходів за категоріями	Пройдено
Експорт даних у формат Excel	Пройдено
Збереження експортованого файлу у зовнішню пам'ять	Пройдено
Відкриття експортованого файлу	Пройдено
Автозапуск сервісу перевірки лімітів після перезавантаження пристрою	Пройдено
Адаптація інтерфейсу до різних розмірів екрану	Пройдено
Коректне відображення кирилических символів	Пройдено
Правильне форматування валюти (гривня)	Пройдено

Додатково було проведено перевірку нефункціональних вимог, включаючи продуктивність додатку, стабільність роботи та використання ресурсів пристрою.

Тестування показало, що застосунок демонструє високу продуктивність навіть на пристроях із мінімальними технічними характеристиками, а використання ресурсів (пам'яті, процесору, акумулятора) знаходиться на оптимальному рівні.

4.5 Висновки за розділом 4

В даному розділі було перевірено розроблене програмне забезпечення, а також було продемонстровано основні функції мобільного застосунку для моніторингу фінансових витрат з підтвердженням реалізації під час тестування програми.

ВИСНОВКИ

У ході виконання дипломної кваліфікаційної роботи було створено мобільний застосунок для відстеження фінансових витрат. Проведено детальний аналіз наявних рішень у даній сфері, визначено їх переваги та недоліки, на основі чого сформовано функціональність власної програмної розробки.

Для реалізації програмного продукту застосовано сучасну мову Kotlin із застосуванням Android Jetpack компонентів, імплементовано архітектурну модель MVVM та інтегровано локальну базу даних Room. Інтерфейсна частина розроблена з використанням декларативного підходу Jetpack Compose, що забезпечило гнучкість та ефективність розробки. В систему інтегровано механізми роботи з фоновими процесами через WorkManager та організовано сповіщення користувачів через NotificationManager.

Розроблений програмний продукт надає користувачам такі можливості:

- внесення та модифікація записів про фінансові операції різних типів;
- застосування багаторівневої системи фільтрів для аналізу запитів;
- встановлення граничних показників витрат за різними категоріями;
- інформування при наближенні до встановлених обмежень;
- візуалізація фінансової інформації через різноманітні графічні представлення;
- категоризація та аналіз структури витрат;
- взаємодія через інтуїтивний інтерфейс з оптимізованою навігацією.

Проведене всебічне тестування підтвердило відповідність розробленого застосунку поставленим функціональним вимогам та технічним характеристикам. Особливу увагу приділено оптимізації використання системних ресурсів та забезпеченню стабільної роботи на різних пристроях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lifehacker : 10 кращих додатків для управління фінансами [Електрон. ресурс]. – Режим доступу : <https://lifehacker.ru/10-money-management-apps/>.
2. Фінансовий моніторинг: огляд популярних інструментів [Електрон. ресурс]. – Режим доступу : <https://finap.com.ua/finansovyyj-monitoryng/>.
3. DOU : Огляд додатків для обліку особистих фінансів [Електрон. ресурс]. – Режим доступу : <https://dou.ua/forums/topic/41422/>.
4. Monefy : Personal finance application [Електрон. ресурс]. – Режим доступу : <https://monefy.com>.
5. Monefy : Огляд мобільного додатку для ведення бюджету [Електрон. ресурс]. – Режим доступу : <https://www.kapusta-online.com.ua/monefy/>.
6. Monefy : Плюси та мінуси додатку для управління фінансами [Електрон. ресурс]. – Режим доступу : <https://irecommend.ua/content/samoe-glavnoe-prilozhenie-na-moem-telephone-plyusy-i-minusy>.
7. Monefy App : Hero image [Electronic recourse]. – Access mode : https://monefy.com/assets/images/hero_image.png.
8. YNAB : Features overview [Electronic recourse]. – Access mode : <https://www.ynab.com/features>.
9. YNAB : Mobile application [Electronic recourse]. – Access mode : <https://apps.apple.com/us/app/ynab/id1010865877>.
10. YNAB : App devices lineup [Electronic recourse]. – Access mode : [https://cdn.prod.website-files.com/640f69143ec11b21d42015c6/6754d6950a834d8433241515_app_devices_lineup_blurple%20\(2\)-p-1080.avif](https://cdn.prod.website-files.com/640f69143ec11b21d42015c6/6754d6950a834d8433241515_app_devices_lineup_blurple%20(2)-p-1080.avif).
11. Wallet by BudgetBakers : What is Wallet? [Electronic recourse]. – Access mode : <https://budgetbakers.com/what-is-wallet/>.

12. Wallet by BudgetBakers : Mobile application [Electronic recourse]. – Access mode : <https://play.google.com/store/apps/details?id=com.droid4you.application.wallet>.
13. Wallet by BudgetBakers: Application interface [Electronic recourse]. – Access mode : https://i0.wp.com/budgetbakers.com/wpcontent/uploads/2022/05/main_img-3.png?resize=1437%2C1536&ssl=1.
14. Kotlin programming language : Official documentation [Electronic recourse – access mode]. – Access mode : <https://kotlinlang.org/docs/kotlin-tour-welcome.html>.
15. Sharma K. Kotlin vs Java : Why Kotlin is the best choice for Android development [Electronic recourse]. – Access mode : <https://medium.com/@krishusharma292/kotlin-vs-java-why-kotlin-is-the-best-choice-for-android-development-59c1153d30b1>.
16. Android Studio : Announcing Studio Bot [Electronic recourse]. – Access mode : <https://android-developers.googleblog.com/2023/05/android-studio-io-23-announcing-studio-bot.html>.
17. Android Developers. Room persistence library [Electronic recourse]. – Access mode : <https://developer.android.com/training/data-storage/room>.
18. Android Developers. Guide to app architecture [Electronic recourse]. – Access mode : <https://developer.android.com/topic/architecture>.

ДОДАТОК А
Технічне завдання

Вступ

У сучасному світі контроль власних фінансів становить важливу складову економічного благополуччя особистості. Проте, багато людей стикаються з труднощами ефективного відстеження та аналізу своїх повсякденних витрат, що ускладнює раціональне планування бюджету.

На сьогодні існує потреба в зручних інструментах, які дозволять користувачам мобільних пристроїв систематично моніторити фінансові витрати та приймати обґрунтовані рішення щодо розподілу коштів.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується за темою «Програмна реалізація мобільного застосунку для моніторингу фінансових витрат» у відповідності з наказом № 209 від 28 квітня 2025 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програмний продукт створюється для надання користувачам мобільних пристроїв інструменту відстеження та аналізу особистих фінансових операцій. Це сприятиме формуванню раціональних фінансових звичок, зменшенню непередбачуваних витрат та ефективному розподілу коштів за різними категоріями потреб.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Розроблений мобільний застосунок повинен забезпечувати реалізацію таких можливостей:

- внесення та редагування даних про фінансові операції;
- категоризація витрат за призначенням та періодами;
- встановлення граничних обмежень на різні категорії витрат;
- інформування користувача при наближенні до встановлених обмежень;
- візуалізація аналітичних даних за допомогою графічних представлень;
- формування статистичних звітів за обраними періодами;
- вивантаження структурованих даних у зовнішні формати;
- інтуїтивно зрозумілий інтерфейс для щоденного використання.

А.3.2 Умови експлуатації

Для функціонування програмного забезпечення необхідний мобільний пристрій під управлінням операційної системи Android. Всі дані зберігаються локально на пристрої користувача, що забезпечує конфіденційність та можливість використання без постійного доступу до мережі.

А.3.3 Вимоги до складу та параметрів технічних засобів

До мобільного пристрою, на якому виконується програмне забезпечення, висуваються наступні мінімальні вимоги:

- пристрій з операційною системою Android версії 6.0 або вище;
- процесор з частотою від 1.2 ГГц;
- оперативна пам'ять обсягом не менше 2ГБ;
- внутрішня пам'ять з вільним простором від 150 МБ.

А.4 Порядок контролю та приймання

Для успішного виконання дипломної кваліфікаційної роботи

визначальним фактором є дотримання поставлених завдань відповідно до затвердженого технічного завдання за визначеною темою в терміни, встановлені графіком виконання роботи.

Підсумкова оцінка роботи здійснюється шляхом представлення та захисту проекту перед екзаменаційною комісією згідно з попередньо оприлюдненим розкладом.

ДОДАТОК Б
Текст програми

Б.1 Текст файла AppDatabase.kt

```

package com.example.start1.data

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import com.example.start1.model.ExpenseLimit
import com.example.start1.model.Transaction

@Database(entities = [Transaction::class, ExpenseLimit::class], version = 2,
exportSchema = false)
@TypeConverters(DateConverter::class, TransactionTypeConverter::class,
LimitPeriodConverter::class)
abstract class AppDatabase : RoomDatabase() {

    abstract fun transactionDao(): TransactionDao
    abstract fun expenseLimitDao(): ExpenseLimitDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "finance_database"
                )
                    .fallbackToDestructiveMigration()
                    .build()

                INSTANCE = instance
            }
            instance
        }
    }
}

```

Б.2 Текстфайла Converters.kt

```
class TransactionTypeConverter {
    @TypeConverter
    fun fromTransactionType(type: TransactionType): String {
        return type.name
    }

    @TypeConverter
    fun toTransactionType(value: String): TransactionType {
        return TransactionType.valueOf(value)
    }
}
```

Б.3 Текстфайла DateConverter.kt

```
class DateConverter {

    @TypeConverter
    fun fromTimestamp(value: Long?): Date? {
        return value?.let { Date(it) }
    }

    @TypeConverter
    fun dateToTimestamp(date: Date?): Long? {
        return date?.time
    }
}
```

Б.4 Текстфайла ExpenseLimitDao.kt

```
@Dao
@TypeConverters(DateConverter::class, LimitPeriodConverter::class)
interface ExpenseLimitDao {

    @Query("SELECT * FROM expense_limits WHERE isActive = 1")
    fun getAllActiveLimits(): Flow<List<ExpenseLimit>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertLimit(limit: ExpenseLimit)

    @Update
    suspend fun updateLimit(limit: ExpenseLimit)
```

```
@Delete
suspend fun deleteLimit(limit: ExpenseLimit)
```

```
@Query("SELECT * FROM expense_limits WHERE category = :category
AND isActive = 1")
fun getLimitForCategory(category: String): Flow<ExpenseLimit?>
}
```

Б.5 Текстфайла LimitPeriodConverter.kt

```
class LimitPeriodConverter {
    @TypeConverter
    fun fromLimitPeriod(period: LimitPeriod): String {
        return period.name
    }

    @TypeConverter
    fun toLimitPeriod(value: String): LimitPeriod {
        return LimitPeriod.valueOf(value)
    }
}
```

Б.6 Текстфайла StatisticsRepository.kt

```
class StatisticsRepository(private val transactionDao: TransactionDao) {

    fun getExpenseSummaryByCategory(startDate: Date, endDate: Date):
    Flow<List<CategorySummary>> {
        return transactionDao.getTransactionsByTypeAndDateRange(
            TransactionType.EXPENSE, startDate, endDate
        ).map { transactions ->

            val categoryGroups = transactions.groupBy { it.category }
            val totalExpense = transactions.sumOf { it.amount }
            categoryGroups.map { (category, categoryTransactions) ->
                val categoryTotal = categoryTransactions.sumOf { it.amount }
                val percentage = if (totalExpense > 0) (categoryTotal / totalExpense *
100).toFloat() else 0f

                CategorySummary(
                    category = category,
                    amount = categoryTotal,
                    transactionCount = categoryTransactions.size,
                    percentage = percentage
                )
            }
        }
    }
}
```

```

    )
    }.sortedByDescending { it.amount }
  }
}

```

```

fun getIncomeSummaryByCategory(startDate: Date, endDate: Date):
Flow<List<CategorySummary>> {
    return transactionDao.getTransactionsByTypeAndDateRange(
        TransactionType.INCOME, startDate, endDate
    ).map { transactions ->
        val categoryGroups = transactions.groupBy { it.category }
        val totalIncome = transactions.sumOf { it.amount }

        categoryGroups.map { (category, categoryTransactions) ->
            val categoryTotal = categoryTransactions.sumOf { it.amount }
            val percentage = if (totalIncome > 0) (categoryTotal / totalIncome *
100).toFloat() else 0f

```

```

            CategorySummary(
                category = category,
                amount = categoryTotal,
                transactionCount = categoryTransactions.size,
                percentage = percentage
            )
        }.sortedByDescending { it.amount }
    }
}

```

```

fun getBalanceSummary(startDate: Date, endDate: Date):
Flow<BalanceSummary> {
    return transactionDao.getAllTransactionsInRange(startDate, endDate).map {
transactions ->
        val incomeTransactions = transactions.filter { it.type ==
TransactionType.INCOME }
        val expenseTransactions = transactions.filter { it.type ==
TransactionType.EXPENSE }

```

```

        val totalIncome = incomeTransactions.sumOf { it.amount }
        val totalExpense = expenseTransactions.sumOf { it.amount }

```

```

        BalanceSummary(
            totalIncome = totalIncome,
            totalExpense = totalExpense,
            netBalance = totalIncome - totalExpense,
            periodStart = startDate,

```

```

        periodEnd = endDate
    )
}
}
}

```

Б.7 Текстфайла TransactionDao.kt

```

@Dao
@TypeConverters(DateConverter::class)
interface TransactionDao {

    @Query("SELECT * FROM transactions ORDER BY date DESC")
    fun getAllTransactions(): Flow<List<Transaction>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTransaction(transaction: Transaction)

    @Delete
    suspend fun deleteTransaction(transaction: Transaction)

    @Query("SELECT COALESCE(SUM(CASE WHEN type = 'INCOME' THEN amount ELSE -amount END), 0) FROM transactions")
    fun getBalance(): Flow<Double>

    @Query("SELECT COALESCE(SUM(amount), 0) FROM transactions WHERE category = :category AND type = 'EXPENSE' AND date BETWEEN :startDate AND :endDate")
    fun getCategoryExpensesBetweenDates(category: String, startDate: Date, endDate: Date): Flow<Double>

    @Query("SELECT * FROM transactions WHERE type = :type AND date BETWEEN :startDate AND :endDate ORDER BY date DESC")
    fun getTransactionsByTypeAndDateRange(type: TransactionType, startDate: Date, endDate: Date): Flow<List<Transaction>>

    @Query("SELECT * FROM transactions WHERE date BETWEEN :startDate AND :endDate ORDER BY date DESC")
    fun getAllTransactionsInRange(startDate: Date, endDate: Date): Flow<List<Transaction>>

    @Query("SELECT * FROM transactions WHERE date BETWEEN :startDate AND :endDate ORDER BY date DESC")

```

```

    fun getTransactionsInDateRange(startDate: Date, endDate: Date):
Flow<List<Transaction>>

    @Query("SELECT * FROM transactions WHERE date >= :date ORDER BY
date DESC")
    fun getTransactionsAfterDate(date: Date): Flow<List<Transaction>>

    @Query("SELECT * FROM transactions WHERE date <= :date ORDER BY
date DESC")
    fun getTransactionsBeforeDate(date: Date): Flow<List<Transaction>>
}

```

Б.8 Текстфайла TransactionRepository.kt

```

class TransactionRepository(
    private val transactionDao: TransactionDao,
    private val expenseLimitDao: ExpenseLimitDao
) {
    val statisticsRepository = StatisticsRepository(transactionDao)
    val allTransactions: Flow<List<Transaction>> =
transactionDao.getAllTransactions()
    val balance: Flow<Double> = transactionDao.getBalance()

    suspend fun insert(transaction: Transaction) {
        transactionDao.insertTransaction(transaction)
    }

    suspend fun delete(transaction: Transaction) {
        transactionDao.deleteTransaction(transaction)
    }

    val allLimits: Flow<List<ExpenseLimit>> =
expenseLimitDao.getAllActiveLimits()

    suspend fun insertLimit(limit: ExpenseLimit) {
        expenseLimitDao.insertLimit(limit)
    }

    suspend fun updateLimit(limit: ExpenseLimit) {
        expenseLimitDao.updateLimit(limit)
    }

    suspend fun deleteLimit(limit: ExpenseLimit) {
        expenseLimitDao.deleteLimit(limit)
    }
}

```

```

    }

    fun getLimitForCategory(category: String): Flow<ExpenseLimit?> {
        return expenseLimitDao.getLimitForCategory(category)
    }

    fun getCategoryExpenses(category: String, startDate: Date, endDate: Date):
    Flow<Double> {
        return transactionDao.getCategoryExpensesBetweenDates(category,
        startDate, endDate)
    }

    fun getTransactionsInDateRange(startDate: Date, endDate: Date):
    Flow<List<Transaction>> {
        return transactionDao.getTransactionsInDateRange(startDate, endDate)
    }

    fun getTransactionsAfterDate(date: Date): Flow<List<Transaction>> {
        return transactionDao.getTransactionsAfterDate(date)
    }

    fun getTransactionsBeforeDate(date: Date): Flow<List<Transaction>> {
        return transactionDao.getTransactionsBeforeDate(date)
    }
}

```

Б.9 Текстфайла ExcelExporter.kt

```

class ExcelExporter(private val context: Context) {

    private val defaultFileName = "finance_export"
    private val dateFormatter = SimpleDateFormat("dd.MM.yyyy", Locale("uk",
    "UA"))
    private val filenameDateFormatter = SimpleDateFormat("yyyy-MM-dd_HH-
    mm", Locale("uk", "UA"))

    suspend fun exportTransactionsToExcel(
        transactions: List<Transaction>,
        startDate: Date? = null,
        endDate: Date? = null
    ): Pair<Uri, String>? = withContext(Dispatchers.IO) {
        try {
            val file = createExcelFile()
            val fileName = file.name
            val workbookSettings = WorkbookSettings()

```

```

workbookSettings.locale = Locale("uk", "UA")
val workbook = Workbook.createWorkbook(file, workbookSettings)
val sheet = workbook.createSheet("Транзакції", 0)
val headerStyle = getHeaderStyle()
val incomeStyle = getIncomeStyle()
val expenseStyle = getExpenseStyle()
val dateStyle = getDateStyle()
val totalStyle = getTotalStyle()

sheet.addCell(Label(0, 0, "Фінансовий звіт", headerStyle))
sheet.mergeCells(0, 0, 4, 0)
// Період звіту
if (startDate != null || endDate != null) {
    val periodText = when {
        startDate != null && endDate != null ->
            "Період: з ${dateFormatter.format(startDate)} по
${dateFormatter.format(endDate)}"
        startDate != null ->
            "Період: з ${dateFormatter.format(startDate)}"
        else ->
            "Період: по ${dateFormatter.format(endDate!!)}"
    }
    sheet.addCell(Label(0, 1, periodText, headerStyle))
    sheet.mergeCells(0, 1, 4, 1) // Об'єднання клітинок для періоду
}

val headers = arrayOf("Дата", "Тип", "Категорія", "Сума", "Опис")
headers.forEachIndexed { index, header ->
    sheet.addCell(Label(index, 2, header, headerStyle))
}

var rowNum = 3
var totalIncome = 0.0
var totalExpense = 0.0

transactions.forEach { transaction ->
    sheet.addCell(DateTime(0, rowNum, transaction.date, dateStyle))
    val typeName = if (transaction.type == TransactionType.INCOME)
        "Дохід" else "Витрата"
    val style = if (transaction.type == TransactionType.INCOME)
        incomeStyle else expenseStyle
    sheet.addCell(Label(1, rowNum, typeName, style))

    // Категорія
    sheet.addCell(Label(2, rowNum, transaction.category, style))

```

```

// Сума
sheet.addCell(Number(3, rowNum, transaction.amount, style))

// Опис
sheet.addCell(Label(4, rowNum, transaction.description, style))

if (transaction.type == TransactionType.INCOME) {
    totalIncome += transaction.amount
} else {
    totalExpense += transaction.amount
}

rowNum++
}

// Підсумковий рядок
sheet.addCell(Label(0, rowNum, "Підсумок:", totalStyle))
sheet.mergeCells(0, rowNum, 2, rowNum)
sheet.addCell(Number(3, rowNum, totalIncome, incomeStyle))
sheet.addCell(Number(4, rowNum, totalExpense, expenseStyle))

// Баланс
rowNum++
sheet.addCell(Label(0, rowNum, "Баланс:", totalStyle))
sheet.mergeCells(0, rowNum, 2, rowNum)
val balance = totalIncome - totalExpense
val balanceStyle = if (balance >= 0) incomeStyle else expenseStyle
sheet.addCell(Number(3, rowNum, balance, balanceStyle))

// Встановлюємо ширину стовпців
sheet.setColumnView(0, 20) // Дата
sheet.setColumnView(1, 15) // Тип
sheet.setColumnView(2, 20) // Категорія
sheet.setColumnView(3, 15) // Сума
sheet.setColumnView(4, 30) // Опис

workbook.write()
workbook.close()

val uri = FileProvider.getUriForFile(
    context,
    "${context.packageName}.fileprovider",
    file
)

```

```

withContext(Dispatchers.Main) {
    Toast.makeText(
        context,
        "Файл збережено в Downloads: $fileName",
        Toast.LENGTH_LONG
    ).show()
}

Pair(uri, fileName)
} catch (e: Exception) {
    e.printStackTrace()
    null
}
}

fun createShareFileIntent(uri: Uri, fileName: String): Intent {
    val shareIntent = Intent(Intent.ACTION_SEND)
    shareIntent.type = "application/vnd.ms-excel"
    shareIntent.putExtra(Intent.EXTRA_STREAM, uri)
    shareIntent.putExtra(Intent.EXTRA_SUBJECT, "Фінансовий звіт")
    shareIntent.putExtra(Intent.EXTRA_TEXT, "Експортовані фінансові дані у файлі $fileName")
    shareIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
    return Intent.createChooser(shareIntent, "Поділитися через")
}

private fun createExcelFile(): File {
    val timestamp = filenameDateFormatter.format(Date())
    val fileName = "${defaultFileName}_${timestamp}.xls" // Розширення .xls
    для JXL

    val downloadsDir =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS)
    return File(downloadsDir, fileName)
}

private fun getHeaderStyle(): WritableCellFormat {
    val headerFont = WritableFont(WritableFont.ARIAL, 12,
WritableFont.BOLD)
    val style = WritableCellFormat(headerFont)
    style.setBackground(Colour.GRAY_25)
    style.setAlignment(Alignment.CENTRE)
    style.setBorder(Border.ALL, BorderLineStyle.THIN)
    return style
}

```

```
}
```

```
private fun getIncomeStyle(): WritableCellFormat {  
    val font = WritableFont(WritableFont.ARIAL, 10)  
    font.setColour(Colour.DARK_GREEN)  
    val style = WritableCellFormat(font)  
    style.setAlignment(Alignment.CENTRE)  
    style.setBorder(Border.ALL, BorderLineStyle.THIN)  
    return style  
}
```

```
private fun getExpenseStyle(): WritableCellFormat {  
    val font = WritableFont(WritableFont.ARIAL, 10)  
    font.setColour(Colour.RED)  
    val style = WritableCellFormat(font)  
    style.setAlignment(Alignment.CENTRE)  
    style.setBorder(Border.ALL, BorderLineStyle.THIN)  
    return style  
}
```

```
private fun getDateStyle(): WritableCellFormat {  
    val dateFormat = DateFormat("dd.MM.yyyy")  
    val style = WritableCellFormat(dateFormat)  
    style.setAlignment(Alignment.CENTRE)  
    style.setBorder(Border.ALL, BorderLineStyle.THIN)  
    return style  
}
```

```
private fun getTotalStyle(): WritableCellFormat {  
    val font = WritableFont(WritableFont.ARIAL, 10, WritableFont.BOLD)  
    val style = WritableCellFormat(font)  
    style.setAlignment(Alignment.RIGHT)  
    style.setBorder(Border.ALL, BorderLineStyle.THIN)  
    return style  
}
```

ДОДАТОК В
Слайди презентації

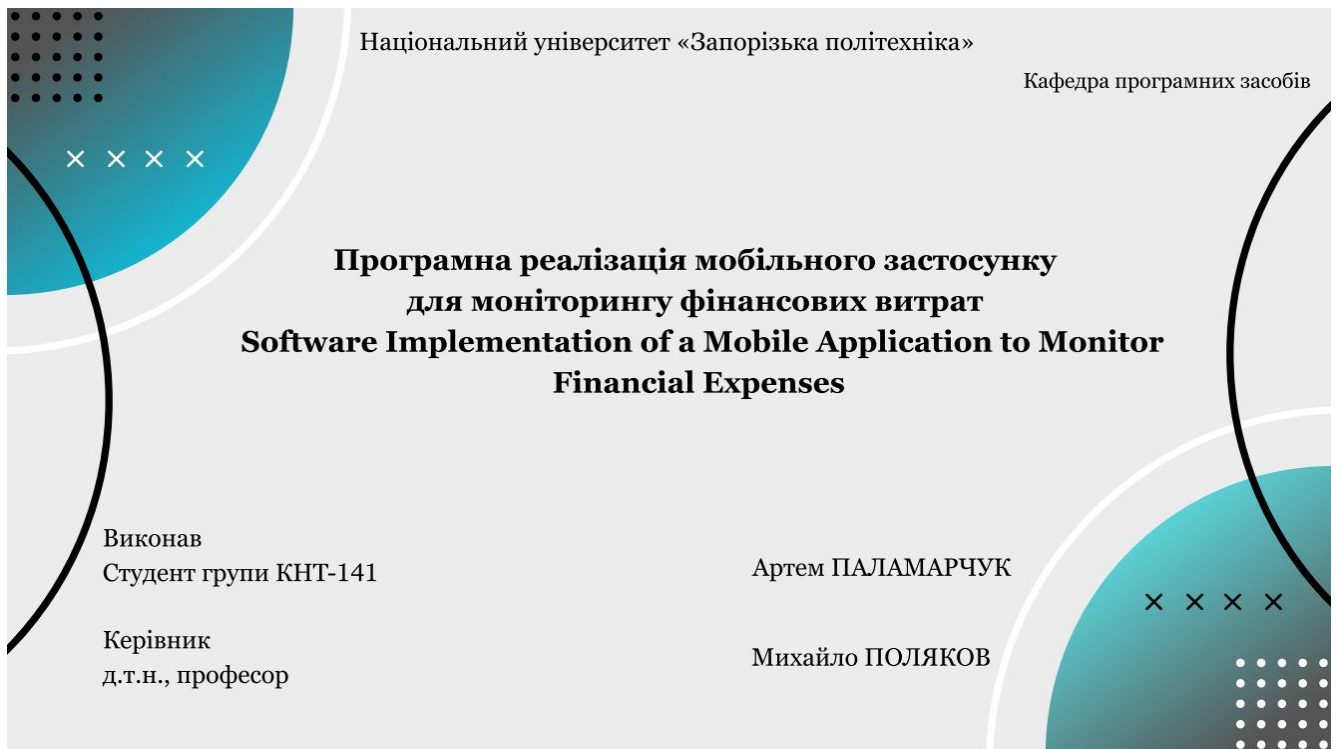


Рисунок В.1 – Слайд №1

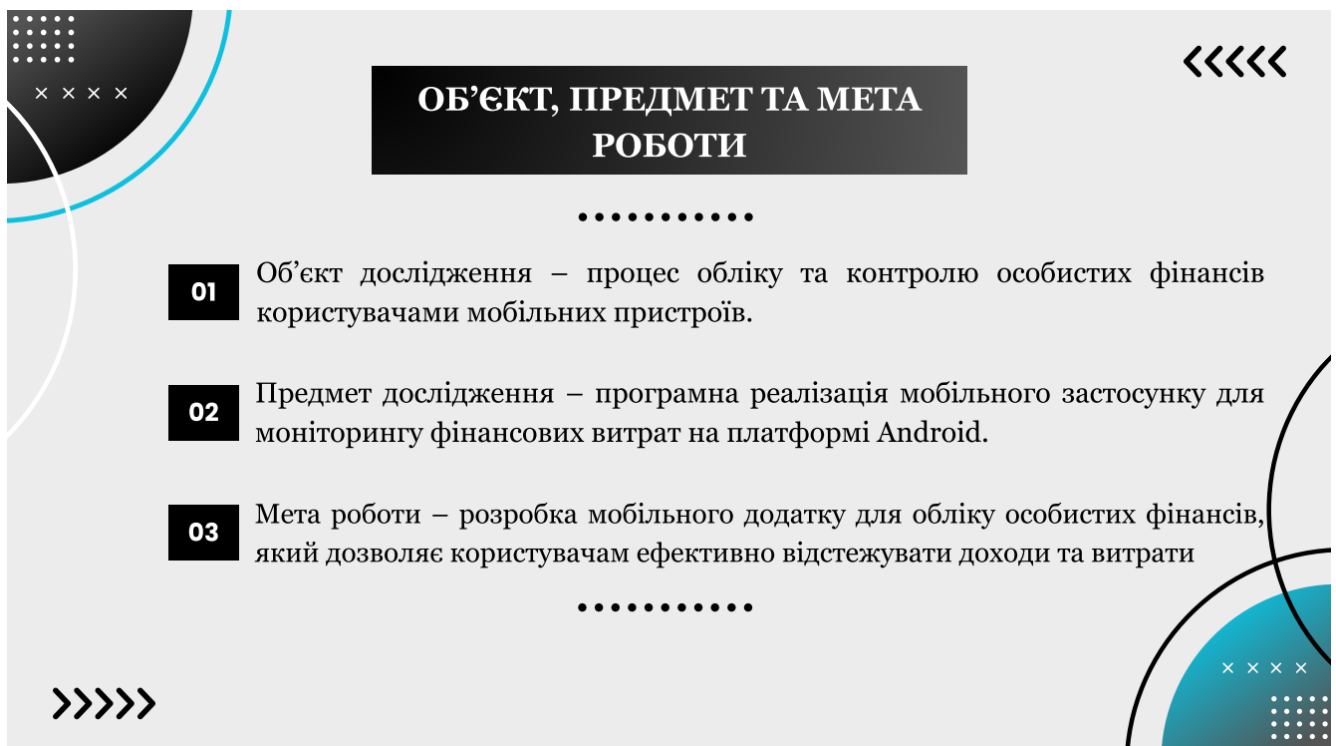


Рисунок В.2 – Слайд №2

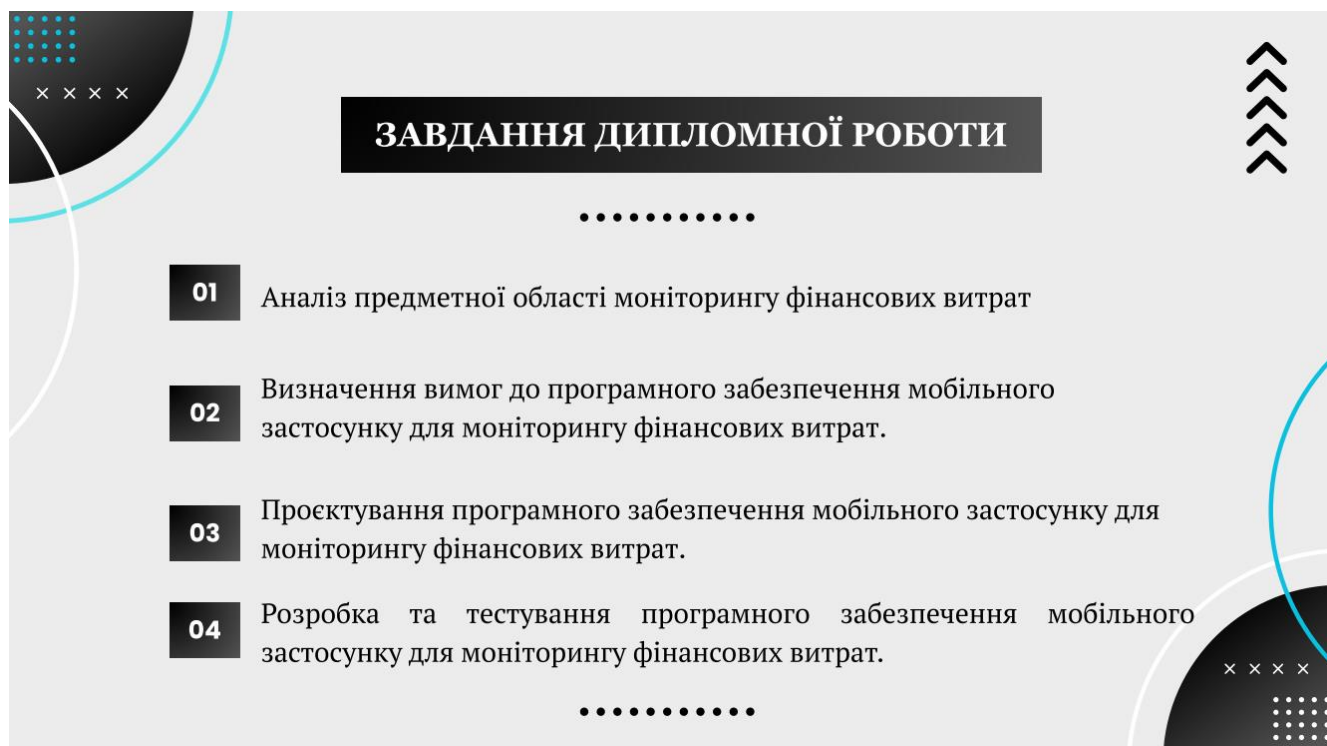


Рисунок В.3 – Слайд №3

Порівняння існуючих аналогів

Таблиця 1.1 – Порівняння з нашим рішенням FinTrack

Функція	Monefy	YNAB	Wallet	FinanceTrack
Ручне додавання транзакцій	Присутні	Присутні	Присутні	Присутні
Категоризація витрат	Автоматична	Ручна	Автоматична	Напів-автоматична
Встановлення бюджетів	Базова	Базова	Базова	Базова
Сповіщення про перевищення бюджету	Базова	Базова	Базова	Базова
Аналітика та звіти	Розширена	Розширена	Розширена	Базова
Підтримка різних валют	Обмежена	Обмежена	Розширена	Обмежена
Інвестиційний менеджмент	Базовий	Відсутній	Розширений	Відсутній
Офлайн-режим	Відсутній	Присутній	Відсутній	Присутній
Безпека даних	Стандартна	Висока	Висока	Висока
Локалізація для України	Відсутня	Відсутня	Відсутня	Присутня
Ціна	Безкоштовно з рекламою	Локалізація для України	Безкоштовно + платні послуги	Безкоштовно

Рисунок В.4 – Слайд №4

Порівняння мов програмування



Таблиця 2.1 – Порівняння Kotlin з іншими мовами програмування

	Kotlin	Java	Flutter(Dart)
Кросплатформеність	Високий рівень	Низький рівень	Середній рівень
Швидкість розробки програми	Високий рівень	Високий рівень	Низький рівень
Швидкість роботи кінцевого продукту	Високий рівень	Низький рівень	Середній рівень
Легкість вивчення	Високий рівень	Низький рівень	Середній рівень
Підтримка різних інструментів	Високий рівень	Середній рівень	Низький рівень
Сучасність	Високий рівень	Низький рівень	Середній рівень
Зрозуміла документація	Високий рівень	Високий рівень	Високий рівень

Рисунок В.5 – Слайд №5

Вибір інструментів розробки мобільного застосунку

Таблиця 2.2 – Порівняння Jetpack Compose з іншими Kotlin-фреймворк

	Jetpack Compose	XML UI (Android)	Flutter UI
Швидкість розробки	Висока	Висока	Висока
Гнучкість створення	Висока	Низька	Середня
Інтеграція	Висока	Висока	Низька
Продуктивність	Висока	Низька	Середня
Масштабованість	Висока	Низька	Середня

Рисунок В.6 – Слайд №6

Діаграма прецедентів

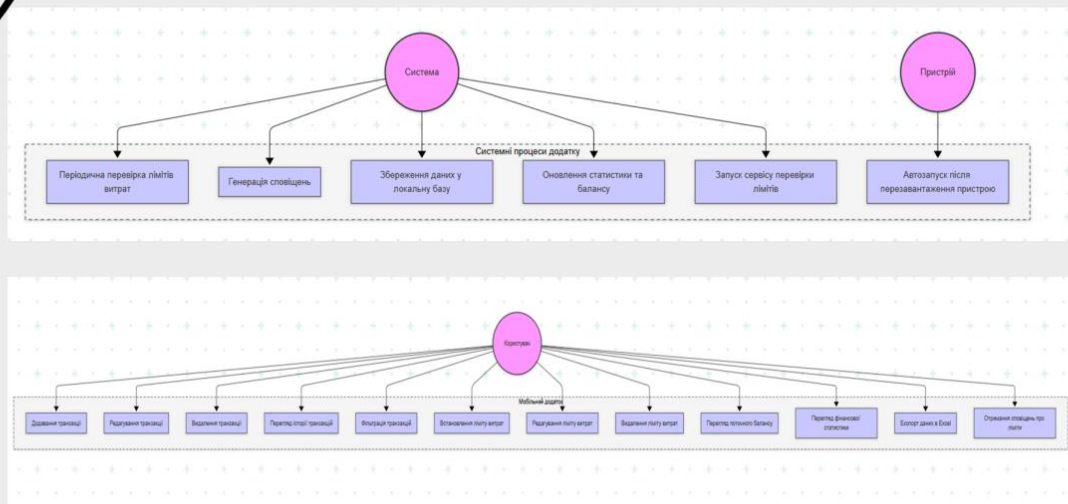


Рисунок В.7 – Слайд №7

Діаграма прецедентів

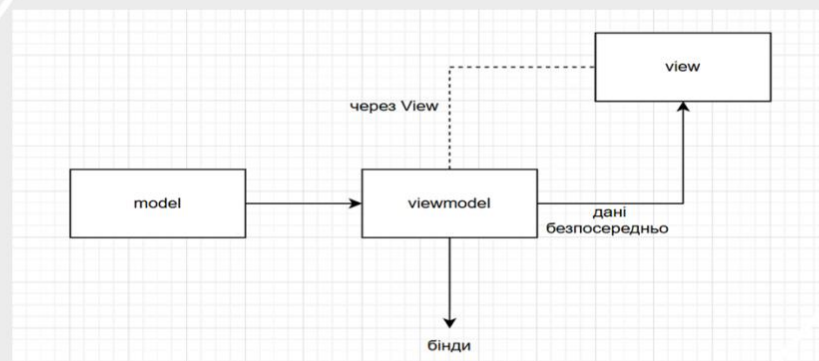


Рисунок В.8 – Слайд №8

Алгоритм функціонування розробленої програми

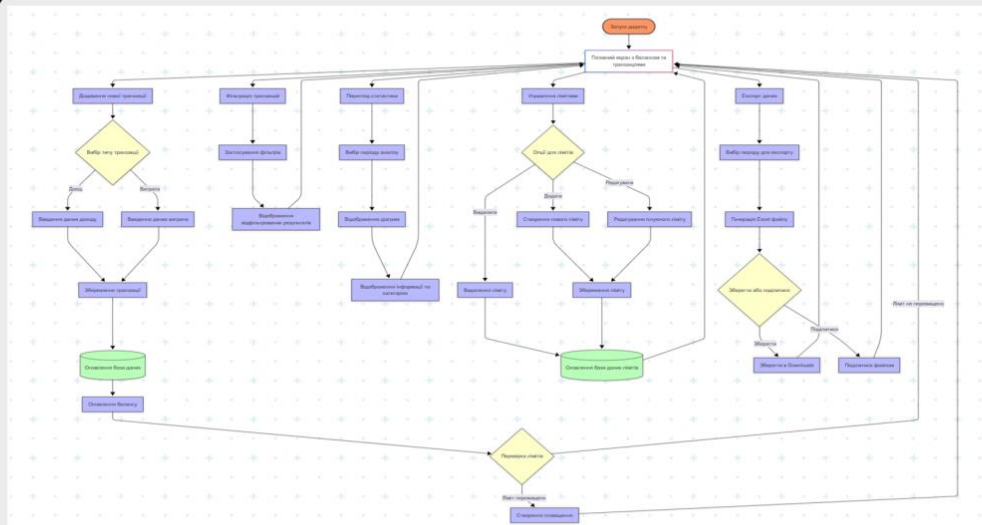


Рисунок В.9 – Слайд №9

Інтерфейси програмного застосунку

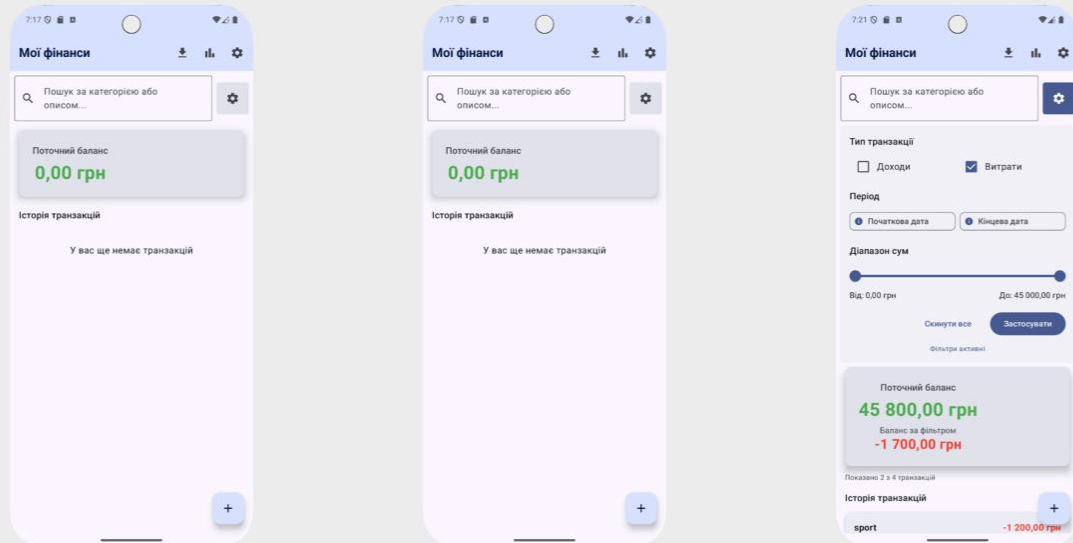


Рисунок В.10 – Слайд №10

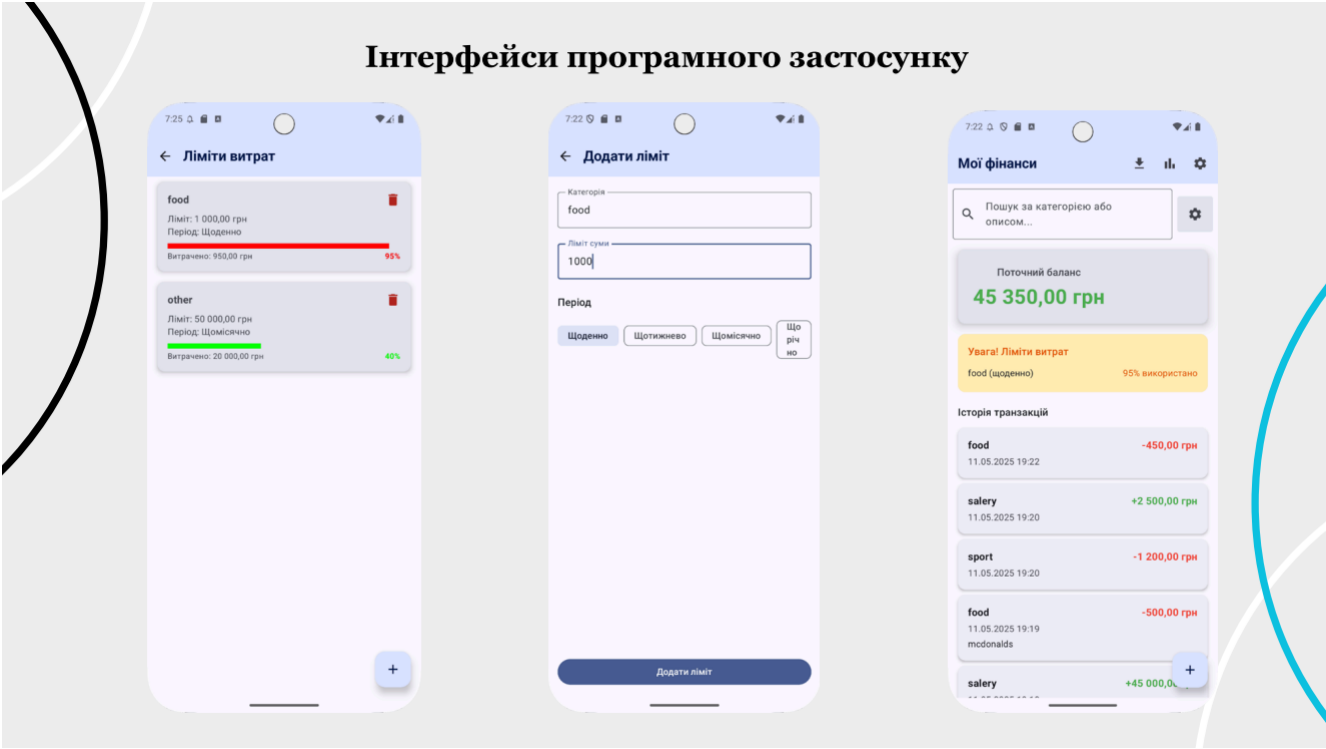


Рисунок В.11 – Слайд №11

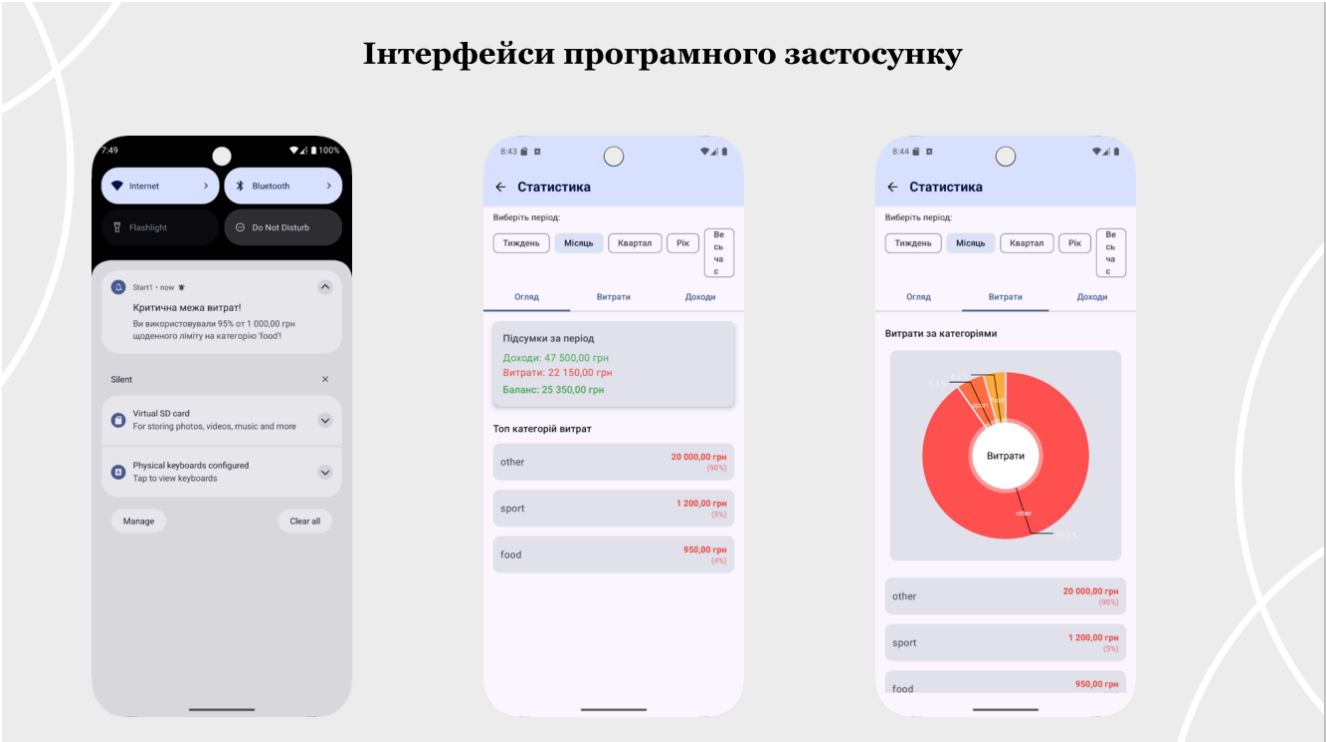


Рисунок В.12 – Слайд №12

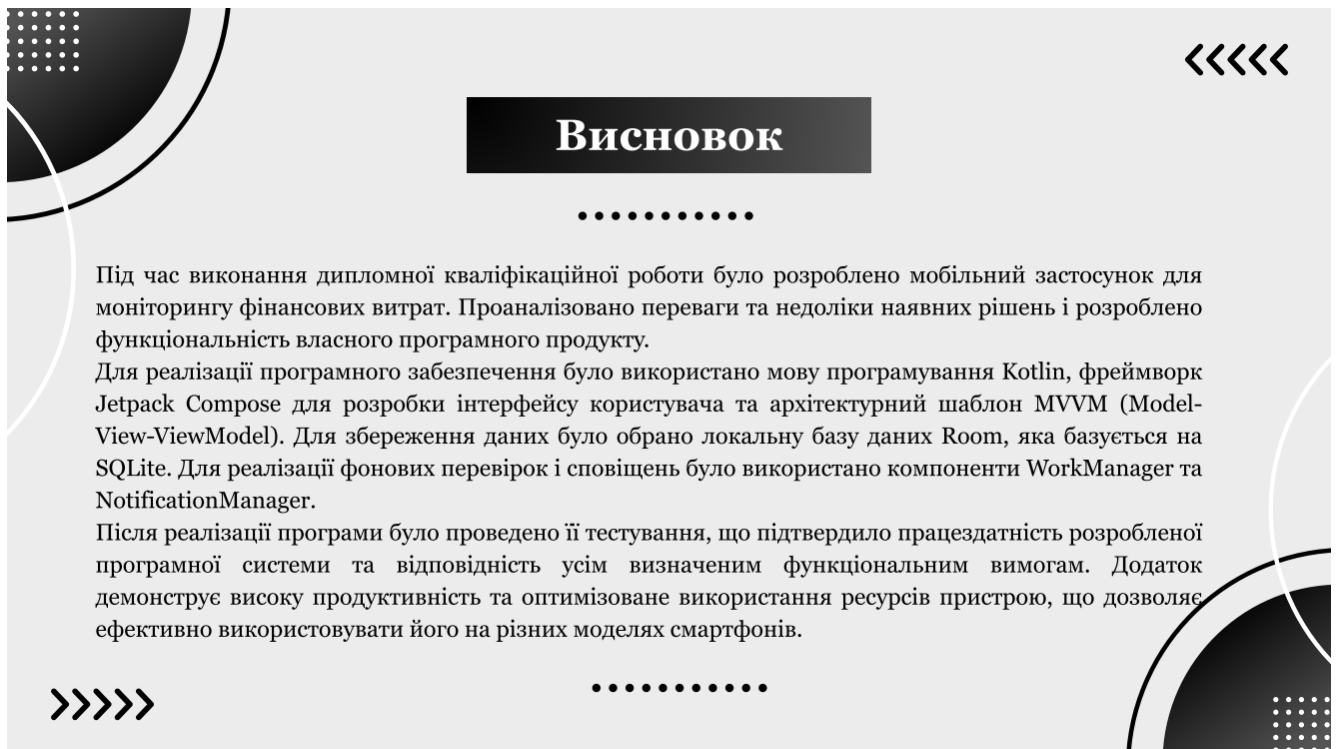


Рисунок В.13 – Слайд №13

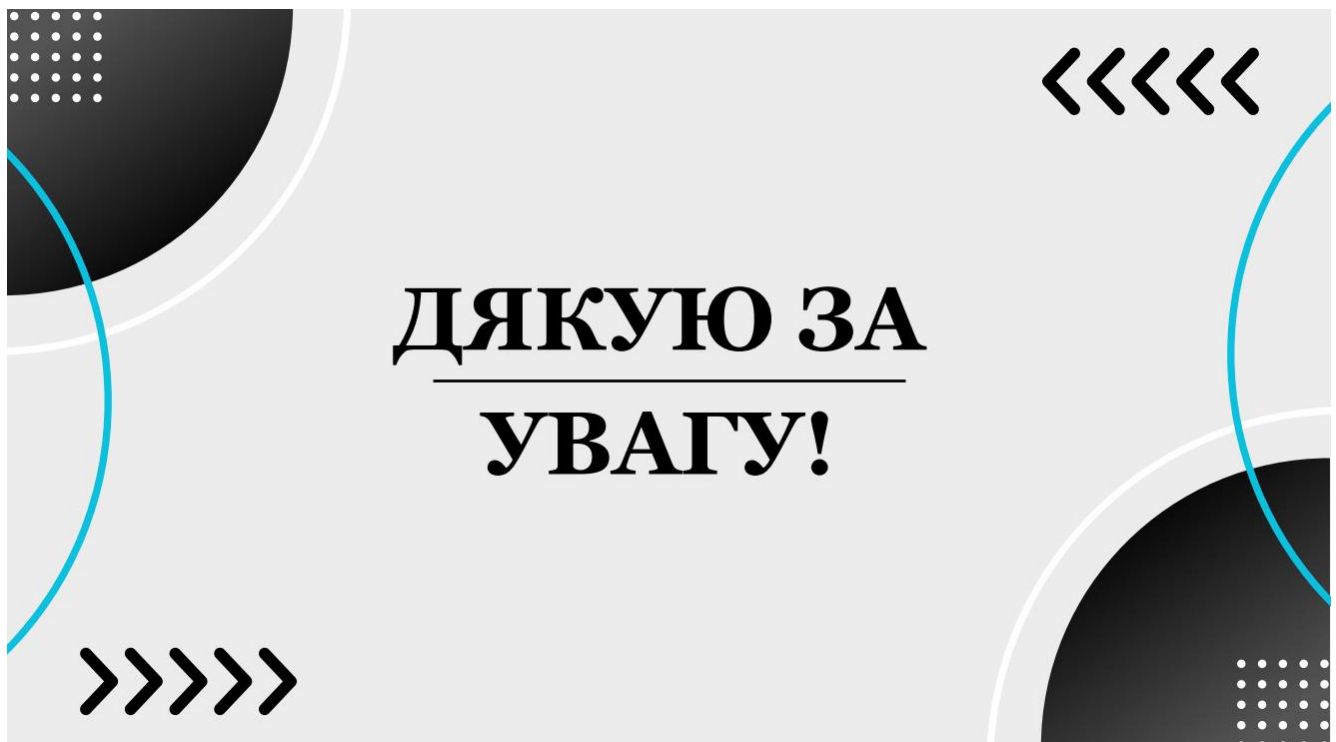


Рисунок В.14 – Слайд №14