

Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої освіти бакалавра
зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка мобільного застосунку для обліку особистих витрат на платформі Android

Засвідчую, що в цій
кваліфікаційній роботі немає
запозичень із праць інших
авторів без відповідних
посилань.

Студент гр. ІПЗ–22–1

_____ / Д. А. Рибалка /

Керівник кваліфікаційної
роботи

_____ / Д. В. Швець /

В.о. завідувача кафедри

_____ / А. М. Стрюк /

Кривий Ріг

2026

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

В.о. зав. кафедри

_____ А. М. Стрюк

«____» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ–22–1 Рибалці Данилу Андрійовичу

1. Тема: «Розробка мобільного застосунку для обліку особистих витрат на платформі Android» затверджена наказом по КНУ № 284с від «28» травня 2026 р.
2. Термін подання студентом закінченої роботи: «31» травня 2026 р.
3. Вихідні дані по роботі: розроблений додаток має забезпечити контроль фінансових витрат користувача та оцінку їх корисності.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): провести аналіз існуючих на ринку аналогічних програмних продуктів, обґрунтувати функціонал розроблюваного додатку, створити програмне забезпечення, здійснити тестування розробленого додатку.
5. Перелік ілюстративного матеріалу: функціональна схема, блок–схема алгоритму, зображення екранних форм додатку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Розгляд літературних джерел та пошук інтернет-ресурсів з заданої тематики	04.01.26 – 24.01.26
2	Аналіз існуючих методів вирішення проблеми	25.01.26 – 07.02.26
3	Формулювання актуальності роботи і постановка завдань	08.02.26 – 18.02.26
4	Оформлення матеріалів першого розділу роботи	18.02.26 – 02.03.26
5	Створення функціональної системи та алгоритму додатку	03.03.26 – 15.03.26
6	Оформлення матеріалів другого розділу роботи	16.04.26 – 09.04.26
7	Розробка баз даних, інтерфейсу програмного забезпечення, програмних модулів	10.04.26 – 01.05.26
8	Оформлення додатків	02.05.26 – 07.05.26
9	Тестування розробленої програми	08.05.26 – 15.05.26
10	Оформлення пояснювальної записки	16.05.26 – 30.05.26

Дата видачі завдання: «03» січня 2026 р.

Студент: _____ / Д. А. Рибалка /

Керівник роботи: _____ / Д. В. Швець /

РЕФЕРАТ

ANDROID, KOTLIN, JETPACK COMPOSE, MVVM, ROOM, SQLITE, ОБЛІК ВИТРАТ, ОСОБИСТІ ФІНАНСИ, МОБІЛЬНИЙ ДОДАТОК, ЛОКАЛЬНА БАЗА ДАНИХ, АНАЛІТИКА.

Пояснювальна записка: 73 с., 12 рис., 1 табл., 1 дод., 24 джерела.

Дипломна робота присвячена проектуванню та програмній реалізації автономного мобільного додатка для операційної системи Android, призначеного для обліку та поведінкового аналізу особистих фінансів.

Актуальність розробки зумовлена необхідністю створення безпечного інструменту фінансового моніторингу, який гарантує конфіденційність користувацьких даних завдяки відмові від клієнт-серверної взаємодії.

Мета роботи полягає у створенні локального програмного продукту, що поєднує базові функції кількісного бюджетування з механізмами оцінки суб'єктивної корисності транзакцій.

Під час виконання проєкту застосовано нативний стек технологій для платформи Android. Програмний код написано мовою Kotlin. Графічний інтерфейс побудовано за допомогою декларативного фреймворку Jetpack Compose із дотриманням стандартів Material Design 3. Внутрішня структура додатка базується на архітектурному шаблоні MVVM. Збереження фінансової історії забезпечується локальною реляційною базою даних SQLite через систему об'єктно-реляційного відображення Room. Розроблене програмне забезпечення підтримує повний цикл управління транзакціями, організацію системи візуальних категорій, агрегацію статистики та візуалізацію даних за допомогою кругових діаграм. Впроваджено метрику п'ятизіркового рейтингу покупок для автоматичного розрахунку середнього індексу доцільності витрат.

ABSTRACT

ANDROID, KOTLIN, JETPACK COMPOSE, MVVM, ROOM, SQLITE, EXPENSE TRACKING, PERSONAL FINANCE, MOBILE APPLICATION, LOCAL DATABASE, ANALYTICS.

Explanatory note: 73 pages, 12 figures, 1 table, 1 appendix, 24 sources.

The thesis is devoted to the design and software implementation of an autonomous mobile application for the Android operating system, intended for the tracking and behavioral analysis of personal finances.

The relevance of the development is driven by the need to create a secure financial monitoring tool that guarantees user data confidentiality by avoiding client-server interaction.

The purpose of the work is to create a local software product that combines the basic functions of quantitative budgeting with mechanisms for evaluating the subjective utility of transactions.

During the project execution, a native technology stack for the Android platform was utilized. The source code is written in the Kotlin programming language. The graphical user interface is built using the Jetpack Compose declarative framework in compliance with Material Design 3 standards. The internal structure of the application is based on the MVVM architectural pattern. The storage of financial history is ensured by a local SQLite relational database via the Room object-relational mapping system. The developed software supports the full transaction management cycle, the organization of a visual category system, statistics aggregation, and data visualization using pie charts. A five-star purchase rating metric has been implemented to automatically calculate the average expense utility index.

ЗМІСТ

ВСТУП.....	7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОБЛІКУ ОСОБИСТИХ ВИТРАТ НА ПЛАТФОРМІ ANDROID	9
1.1. Аналіз проблеми контролю особистих фінансів	9
1.2. Аналіз сучасних мобільних додатків для обліку витрат	10
1.3. Порівняльний аналіз аналогів.....	14
1.4. Аналіз технологій розробки Android-додатку	15
1.5. Висновки по розділу	17
2 ПРОЕКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ОСОБИСТИХ ВИТРАТ НА ПЛАТФОРМІ ANDROID	19
2.1 Вибір архітектури програмного забезпечення.....	19
2.2 Проектування структури додатку.....	20
2.3 Реалізація бази даних	23
2.4 Проектування інтерфейсу користувача.....	25
2.5 Проектування системи категорій та аналітики.....	26
3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ОСОБИСТИХ ВИТРАТ НА ПЛАТФОРМІ ANDROID	29
3.1 Алгоритм роботи додатку	29
3.2 Налаштування середовища розробки.....	31
3.3 Керівництво користувача та опис роботи програми	33
ВИСНОВКИ	39
ПЕРЕЛІК ПОСИЛАНЬ	41
Додаток А - Програмний код.....	44

ВСТУП

Управління фінансовими ресурсами на індивідуальному рівні вимагає використання надійних інструментів для збору та обробки транзакцій. Масова цифровізація платіжних систем призвела до збільшення кількості щоденних операцій, що робить їх ручний облік неефективним та підвищує ймовірність помилок. Існуючі на ринку мобільні рішення здебільшого функціонують на базі клієнт-серверної архітектури та вимагають передачі персональної історії витрат на віддалені сервери. Для значної частини аудиторії вимога обов'язкової хмарної синхронізації є критичною перешкодою через обґрунтоване занепокоєння щодо конфіденційності даних.

Більшість популярних застосунків фокусується виключно на кількісних показниках сум і категорій, залишаючи поза увагою потребу в якісній оцінці доцільності витрачених коштів. Розробка повністю автономного мобільного інструменту, який поєднує ізольоване збереження бази даних із модулем суб'єктивного оцінювання покупок, є актуальним інженерним завданням.

Метою дипломної роботи є створення мобільного Android-додатка для обліку особистих фінансів, що функціонує без підключення до мережі Інтернет та забезпечує розширені можливості аналізу поведінкових патернів споживача.

Об'єктом дослідження виступає процес управління та програмного аналізу особистих фінансових витрат. Предметом розробки є архітектура, користувацький інтерфейс та функціонал локального мобільного Android-додатка.

Для досягнення поставленої мети передбачено виконання низки технічних завдань. Насамперед планується здійснити огляд наявних рішень, виявити їхні архітектурні вразливості та обґрунтувати вибір технологічного стека.

Практичне значення отриманих результатів полягає у створенні готового до використання програмного забезпечення, яке гарантує

приватність фінансової активності користувача. Розроблений додаток може ефективно застосовуватися для оптимізації персонального бюджету, виявлення нераціональних споживчих звичок та загального підвищення рівня фінансової грамотності без ризику компрометації персональних даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОБЛІКУ ОСОБИСТИХ ВИТРАТ НА ПЛАТФОРМІ ANDROID

1.1. Аналіз проблеми контролю особистих фінансів

Розробка засобів керування особистими фінансами вимагає систематичного обліку доходів та видатків для підтримання економічної стабільності індивіда та домогосподарства загалом [1]. Збільшення кількості безготівкових транзакцій та поширення цифрових платіжних систем призвели до фрагментації фінансової інформації. Користувачі часто здійснюють десятки дрібних операцій щодня, що ускладнює їх запам'ятовування та подальшу ручну категоризацію. Відсутність консолідованої інформації про витрати заважає адекватно оцінювати власні фінансові звички, виявляти поведінкові патерни та планувати бюджет на довгострокову перспективу [2].

Більшість банківських установ надають власні мобільні додатки з вбудованими інструментами аналітики. Їхній функціонал зазвичай обмежений жорстко заданими категоріями, які алгоритми банку визначають автоматично на основі МСС-кодів торгових точок. Цей механізм часто генерує хибні класифікації або узагальнює транзакції до неінформативних груп, позбавляючи користувача можливості самостійно деталізувати витрати. Людина може паралельно користуватися послугами кількох банківських установ, що робить неможливим отримання цілісної картини фінансового стану в межах одного клієнтського застосунку.

На ринку програмного забезпечення існує значна кількість спеціалізованих застосунків для ведення бюджету. Переважна частина таких продуктів побудована за клієнт-серверною архітектурою і вимагає обов'язкової реєстрації, створення облікових записів та постійного підключення до мережі Інтернет. Зберігання конфіденційної фінансової інформації на віддалених серверах створює потенційні ризики витоку

персональних даних внаслідок вразливостей мережових протоколів або цілеспрямованих атак [3]. Для певної частини аудиторії наявність хмарної синхронізації не є критичною вимогою, тоді як гарантія повної приватності та локального зберігання інформації стає пріоритетним критерієм під час вибору програмного продукту.

Іншою неочевидною проблемою існуючих рішень є фокус виключно на кількісних показниках. Традиційні системи обліку фіксують суму та категорію, але повністю ігнорують якісний аспект споживання. Користувач отримує звіт про обсяг витрачених коштів, проте не має інструментів для ретроспективної оцінки доцільності цих витрат. Впровадження механізму оцінювання корисності кожної покупки дозволило б ідентифікувати нераціональні витрати та коригувати фінансову поведінку на основі зібраної локальної статистики.

Вирішення описаних проблем вимагає створення спеціалізованого програмного інструменту, орієнтованого на автономність та відмову від зовнішніх залежностей. Розробка локального мобільного додатку із застосуванням надійних механізмів збереження даних [4] дозволить забезпечити високий рівень конфіденційності, оскільки вся інформація записуватиметься виключно в пам'ять фізичного пристрою. Реалізація функцій гнучкого налаштування категорій та системи рейтингу корисності покупок надасть розширені можливості для аналізу власних фінансових потоків без прив'язки до серверних веб-сервісів чи нав'язаних банківських алгоритмів.

1.2. Аналіз сучасних мобільних додатків для обліку витрат

Ринок мобільного програмного забезпечення пропонує широкий спектр рішень для управління особистими фінансами. Огляд популярних додатків, таких як Monefy, Wallet by BudgetBakers та YNAB (You Need A Budget), демонструє тенденцію до функціонального ускладнення та переходу на сервісну модель розповсюдження. Більшість сучасних застосунків орієнтовані

на екосистемність, що передбачає примусову синхронізацію даних між пристроями користувача через хмарні інфраструктури розробника [5].

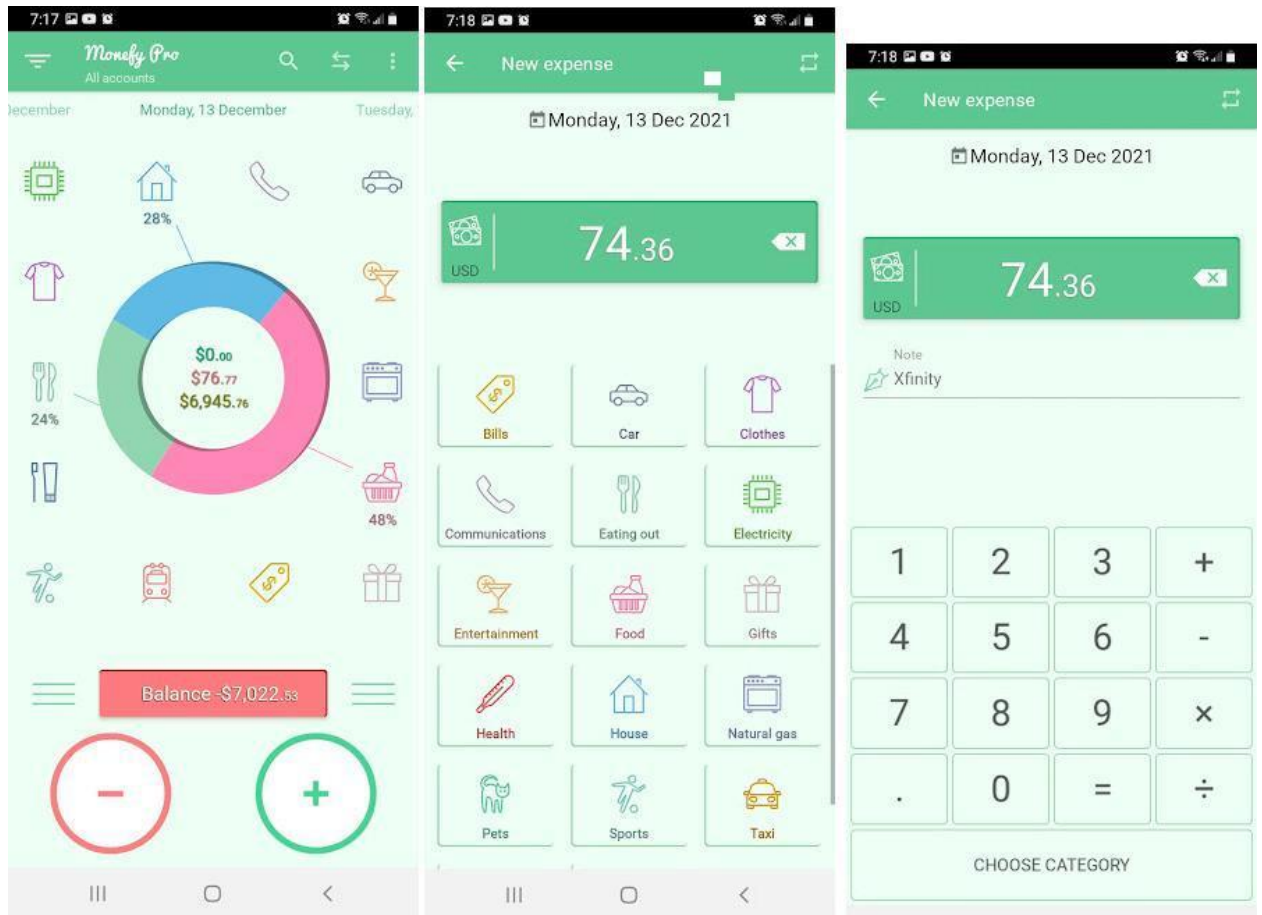


Рисунок 1.1 - Monefy

Архітектурно такі системи побудовані за класичною клієнт-серверною моделлю. Клієнтський мобільний додаток виконує роль інтерфейсу для збору даних та їхньої первинної візуалізації, тоді як основна бізнес-логіка, агрегація статистики та резервне копіювання реалізуються на стороні бекенду. Зазначений принцип забезпечує доступність інформації з різних платформ, проте створює критичну залежність від стабільності інтернет-з'єднання. Користувач змушений передавати третій стороні повний доступ до історії своїх фінансових транзакцій. Враховуючи загальносвітове зростання кількості кіберінцидентів, збереження конфіденційної інформації на віддалених

серверах стає суттєвим недоліком для осіб, які дбають про цифрову безпеку [6].

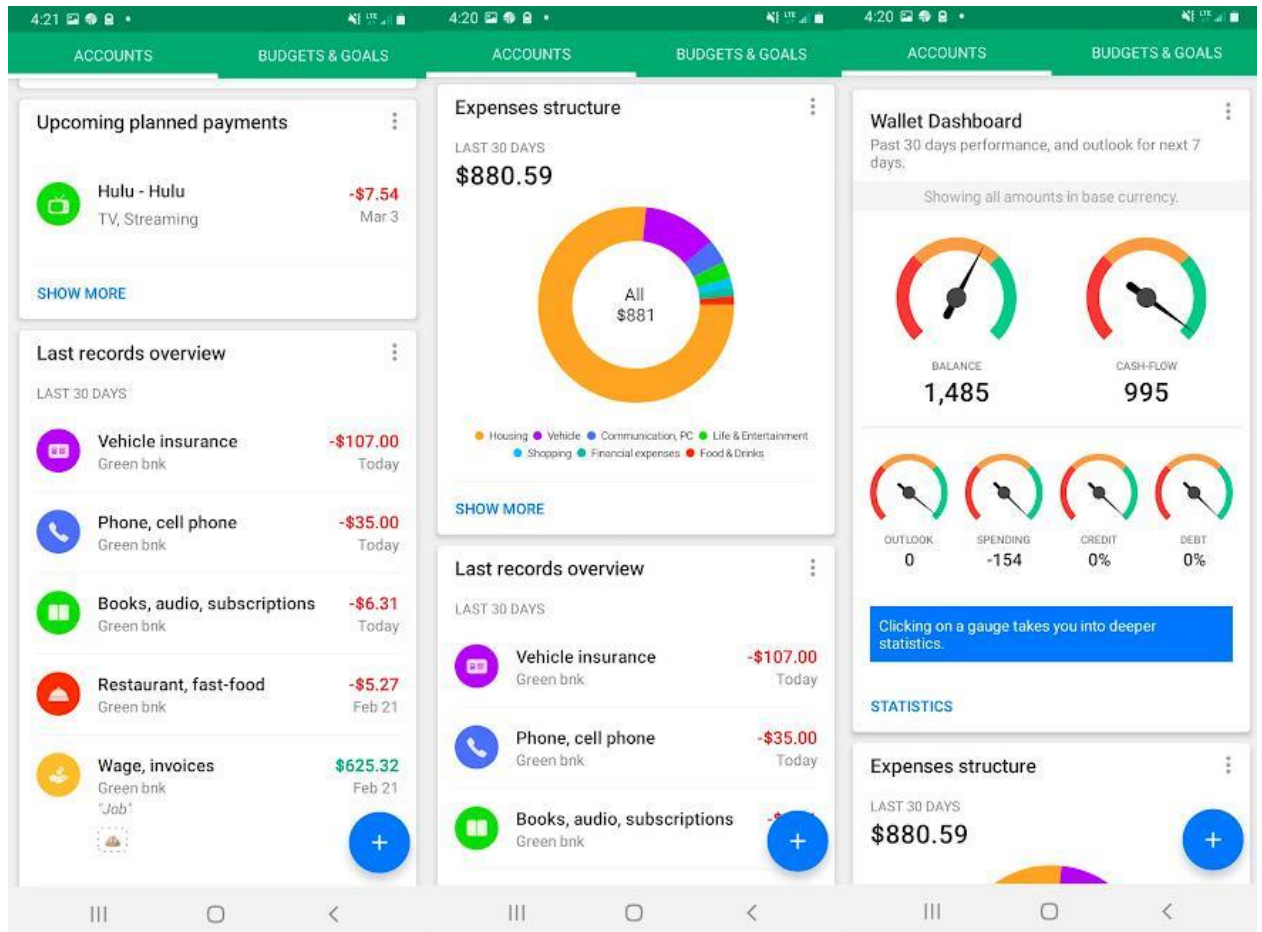


Рисунок 1.2 - Wallet by BudgetBakers

Інтерфейси сучасних фінансових додатків часто перевантажені додатковими модулями. Розробники інтегрують підключення до банківських API, інструменти для інвестування, багаторівневі системи планування бюджету та соціальні функції. Розширення можливостей продукту підвищує поріг входження та ускладнює виконання базового користувацького сценарію, який полягає у швидкій фіксації нової витрати. Надмірне використання складних графічних елементів та глибокої багаторівневої навігації призводить до збільшення часу взаємодії з додатком [7].

Зайве когнітивне навантаження, викликане складним графічним інтерфейсом, негативно впливає на мотивацію користувача регулярно вести облік власних фінансів.

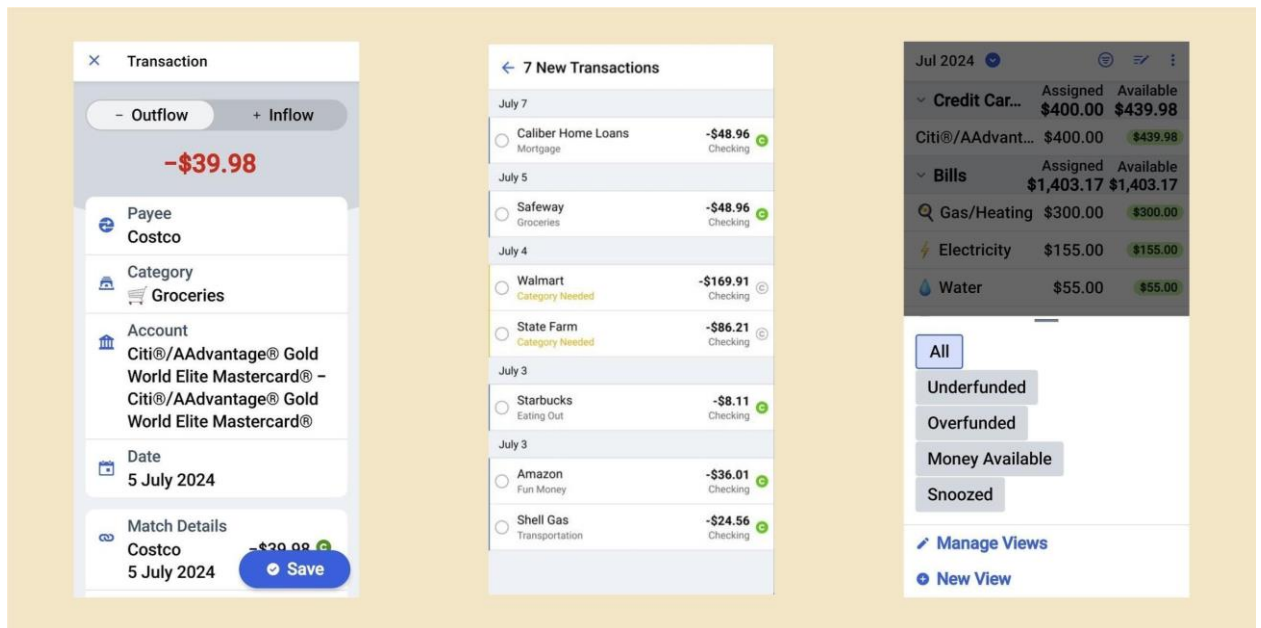


Рисунок 1.3 - YNAB (You Need A Budget)

Окремої уваги потребує підхід до аналітики зібраних даних. У більшості проаналізованих програмних рішень статистика формується виключно на основі кількісних показників, групуючи суми витрат за категоріями у визначений проміжок часу. Жоден з популярних продуктів не пропонує інтегрованих інструментів для якісної ретроспективної оцінки транзакцій. Користувач позбавлений можливості маркувати покупки за рівнем їхньої фактичної корисності чи суб'єктивного задоволення після придбання. Відсутність такого механізму заважає ідентифікувати імпульсивні покупки та знижує загальну ефективність фінансового самоаналізу.

Вивчення існуючих рішень підтверджує наявність вільної ніші для програмного продукту, який поєднує автономність з інструментами якісного аналізу. Розробка додатка на базі локальної бази даних SQLite (бібліотека Room) з використанням архітектурного патерну MVVM дозволить гарантувати повну приватність інформації [8]. Впровадження системи

п'ятизіркового рейтингу оцінки покупок розширить аналітичні можливості без необхідності ускладнення інтерфейсу або підключення зовнішніх аналітичних веб-сервісів.

1.3. Порівняльний аналіз аналогів

Для обґрунтування функціональних вимог до програмного продукту здійснено розгляд існуючих на ринку аналогів. Оцінювання проводилося за критеріями архітектурної автономності, рівня забезпечення конфіденційності користувацьких даних, складності графічного інтерфейсу та наявності інструментів якісного аналізу витрат. Отримані результати дозволяють визначити конкурентні переваги та сформулювати унікальну пропозицію для розроблюваного застосунку [9].

Додаток Monefy вирізняється зрозумілим інтерфейсом, візуальна складова якого побудована навколо інтерактивної кругової діаграми. Незважаючи на можливість локальної роботи, застосунок містить вбудовані модулі синхронізації через сторонні хмарні сервіси, що вимагає надання системних дозволів доступу. Механізм категоризації є обмеженим у базовій версії, а інструментарій для оцінки доцільності витрачених коштів повністю відсутній [10].

Програмний продукт Wallet від компанії BudgetBakers є комплексним рішенням із підтримкою відкритого банкінгу та автоматичної категоризації транзакцій. Застосунок має яскраво виражену клієнт-серверну архітектуру. Повноцінне використання закладених функцій вимагає постійного мережевого підключення та створення персонального облікового запису на серверах компанії. Значна деталізація фінансової інформації супроводжується перевантаженням графічного інтерфейсу, що створює надлишкове когнітивне навантаження під час виконання базових операцій збереження даних.

Застосунок CoinKeeper використовує специфічну візуальну метафору перетягування монет для фіксації транзакцій. Запропонований підхід виділяє продукт серед конкурентів, проте архітектурна модель також базується на

обов'язковій взаємодії з віддаленими серверами. Якісний аналіз фінансової поведінки, як і в попередніх випадках, не передбачений архітектурою бази даних платформи [11].

Таблиця 1.1 – Порівняння розглянутих аналогів

Програмний продукт	Архітектурна модель	Режим офлайн	Оцінка корисності	Складність UI
Monefy	Клієнт-хмара	Частково	Відсутня	Середня
Wallet	Клієнт-сервер	Ні	Відсутня	Висока
CoinKeeper	Клієнт-сервер	Ні	Відсутня	Середня
Власна розробка	Локальна (Room)	Повністю	Наявна	Низька

Аналіз технічних характеристик представлених рішень підтверджує відсутність продукту, який би задовольняв вимогу абсолютної локальності зберігання даних у поєднанні з метрикою оцінки покупок. Проектований додаток компенсуватиме виявлені недоліки шляхом збереження транзакцій у локальній базі даних SQLite та імплементації п'ятизіркового рейтингу корисності для кожного фінансового запису.

1.4. Аналіз технологій розробки Android-додатку

Для створення мобільного додатка пропонується обрати нативний підхід до розробки програмного забезпечення під операційну систему Android.

Незважаючи на значне поширення кросплатформних фреймворків, нативна розробка забезпечує вищий рівень продуктивності, менший розмір інсталяційного пакета та прямий доступ до системних API. Оскільки проєктований застосунок функціонуватиме виключно локально без звернень до віддалених серверів, оптимізація роботи з файловою системою та вбудованою пам'яттю пристрою є важливою умовою [12].

Основною мовою програмування проєкту виступає Kotlin. Ця мова офіційно підтримується корпорацією Google як пріоритетний інструмент для екосистеми Android. На відміну від Java, Kotlin пропонує лаконічний синтаксис та забезпечує вбудовану безпеку типів (null-safety), що кардинально знижує ймовірність виникнення критичних помилок під час виконання програми. Для організації асинхронних операцій застосовується механізм співпрограм (Coroutines). Вони дозволяють виконувати тривалі запити до локальної бази даних у фонових потоках без блокування головного потоку інтерфейсу користувача. Завдяки Coroutines зберігається читабельність програмного коду, оскільки асинхронні виклики записуються у послідовному стилі без надмірного використання функцій зворотного виклику [13].

Проєктування графічного інтерфейсу базується на декларативному фреймворку Jetpack Compose. Традиційний підхід, що передбачав створення статичної розмітки у файлах XML та подальшу програмну маніпуляцію елементами, є застарілим і схильним до помилок синхронізації стану. Jetpack Compose дозволяє описувати користувацький інтерфейс за допомогою функцій Kotlin, які автоматично ініціюють перемальовування екрана при зміні пов'язаних даних. Візуальна складова реалізується з використанням компонентів Material Design 3, що забезпечує сучасний та інтуїтивно зрозумілий вигляд додатка. Організація переходів між екранами виконується засобами бібліотеки Navigation Compose у межах архітектури Single Activity [14].

Зберігання фінансової інформації користувача здійснюється у реляційній базі даних SQLite. Для надійнішої та зручнішої взаємодії з нею

застосовується об'єктно-реляційне відображення (ORM) за допомогою бібліотеки Room. Room створює рівень абстракції над базовими методами SQLite, забезпечуючи перевірку синтаксичної правильності SQL-запитів ще на етапі компіляції. Зазначена бібліотека має нативну інтеграцію з Kotlin Coroutines, що дозволяє реалізувати реактивний підхід до оновлення даних. При додаванні або зміні запису про витрати база даних генерує подію, яка автоматично оновлює відповідні компоненти інтерфейсу без необхідності повторного ручного запиту [15].

Архітектурним фундаментом додатка обрано патерн MVVM (Model-View-ViewModel). У цій парадигмі рівень View, представлений компонентами Jetpack Compose, відповідає виключно за рендеринг графічних елементів та фіксацію дій користувача. Вся бізнес-логіка, агрегація статистики та маніпуляції з локальним сховищем інкапсульовані у класах ViewModel. Такий розподіл обов'язків полегшує подальше тестування програмного коду та запобігає втраті даних під час зміни конфігурації пристрою.

Для візуалізації статистичних даних та побудови кругових діаграм витрат інтегровано бібліотеку MPAndroidChart. Оскільки ця бібліотека розроблялася для класичного імперативного підходу до побудови інтерфейсів, її інтеграція у декларативне середовище здійснюється через спеціальну обгортку AndroidView. Це інженерне рішення дозволяє безшовно поєднувати традиційні візуальні компоненти з сучасним кодом Jetpack Compose. Використання MPAndroidChart гарантує високу швидкість відмальовування графіків та надає необхідний інструментарій для наочної презентації фінансової аналітики в умовах повної автономності застосунку.

1.5. Висновки по розділу

Вивчення предметної області підтвердило обґрунтованість створення автономного інструменту для ведення особистого бюджету. Переважна частина існуючих на ринку мобільних рішень базується на клієнт-серверній архітектурі та вимагає обов'язкової хмарної синхронізації. Зазначена

особливість задовольняє потреби широкого кола споживачів, проте формує потенційні загрози для приватності фінансової інформації. Перевантаженість графічних інтерфейсів банківськими інтеграціями, соціальними функціями та надлишковими модулями планування підвищує поріг входження. Це негативно впливає на виконання базового користувацького сценарію, який полягає у швидкій та безперешкодній фіксації транзакцій.

Детальний розгляд популярних аналогів виявив спільний недолік у підходах до формування аналітики. Традиційні системи обліку фокусуються виключно на кількісних показниках, групуючи витрати за стандартними категоріями. Відсутність механізмів якісного оцінювання фінансових операцій ускладнює виявлення імпульсивних придбань. Впровадження системи оцінки корисності кожної покупки дозволить задовольнити потребу в глибокому самоаналізі власної економічної поведінки без перевантаження інтерфейсу зайвими елементами.

Для технічної реалізації поставлених завдань обрано нативний стек розробки під операційну систему Android. Застосування мови програмування Kotlin у поєднанні з декларативним фреймворком Jetpack Compose забезпечить створення чуйного мінімалістичного інтерфейсу з оптимізованим відмальовуванням компонентів. Збереження конфіденційності даних гарантується використанням локальної бази SQLite через бібліотеку Room. Вибір архітектурного шаблону MVVM дозволить ефективно ізолювати бізнес-логіку обробки транзакцій від візуального представлення, що полегшить подальше тестування програмного коду.

Сформовані вимоги визначають вектор подальшої практичної роботи над проєктом. Проєктування додатка базуватиметься на принципах повної інформаційної автономності, швидкодії взаємодії з локальним сховищем та наданні користувачеві спеціалізованого інструментарію для ведення усвідомленого обліку витрат.

2 ПРОЕКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ОСОБИСТИХ ВИТРАТ НА ПЛАТФОРМІ ANDROID

2.1 Вибір архітектури програмного забезпечення

Розробка надійного та масштабованого мобільного додатка неможлива без попереднього проектування його внутрішньої структури. Правильно обраний архітектурний шаблон визначає правила взаємодії між компонентами системи, спрощує процес тестування та полегшує подальший супровід коду. В екосистемі Android історично застосовувалися різні підходи, починаючи від класичного MVC (Model-View-Controller) і закінчуючи MVP (Model-View-Presenter). Для сучасних застосунків, які активно використовують реактивне програмування та управління станами, офіційна документація рекомендує застосовувати шаблон MVVM (Model-View-ViewModel) [16].

Вибір MVVM для проекту зумовлений його технічною синергією з декларативним фреймворком Jetpack Compose. У цій парадигмі інтерфейс користувача будується на основі поточного стану даних. Шаблон MVVM дозволяє розмежувати візуальне представлення від бізнес-логіки. Рівень View відповідає виключно за відображення графічних елементів, компоновання екранів та реєстрацію дій користувача. Він не містить жодної логіки обробки фінансової інформації, а лише спостерігає за змінами стану, які транслює ViewModel. Такий розподіл відповідальності кардинально зменшує зв'язність коду та запобігає виникненню помилок життєвого циклу компонентів [17].

Ключовою ланкою обраної архітектури виступає ViewModel. Цей клас спроектований таким чином, щоб переживати зміни конфігурації пристрою, зокрема поворот екрана або перемикання системної теми, зберігаючи актуальні дані користувача без необхідності їх повторного завантаження. У контексті розроблюваного застосунку ViewModel бере на себе завдання з обробки введених сум, перевірки коректності даних, розрахунку агрегованої статистики та управління рейтингом корисності покупок. Взаємодія з базою

даних виконується асинхронно у фонових потоках за допомогою механізму Kotlin Coroutines. Ізоляція важких операцій читання та запису гарантує стабільно високу частоту кадрів графічного інтерфейсу навіть під час фільтрації великих масивів фінансових транзакцій [18].

Рівень Model у класичному розумінні зазвичай включає складну систему репозиторіїв для синхронізації локального кешу та віддаленого сервера. Оскільки додаток «SimpleExpenses» спроектовано як повністю автономне рішення без використання мережових запитів, архітектуру було свідомо спрощено до схеми прямих викликів: графічний інтерфейс звертається до ViewModel, яка безпосередньо взаємодіє з локальною базою даних Room. Відмова від проміжних абстракцій, таких як виділений шар репозиторію, є виправданим інженерним кроком для суто локального проєкту. Застосування цієї конфігурації дозволяє уникнути написання надлишкового шаблонного коду, зберігаючи при цьому необхідний рівень модульності системи [19].

Запропонована спрощена архітектура «UI – ViewModel – Room Database» повністю відповідає специфіці та масштабам проєкту. Вона забезпечує високу швидкість розробки, гарантує надійність збереження інформації та створює фундамент для можливого розширення аналітичних можливостей системи без необхідності глобального рефакторингу програмного коду.

2.2 Проєктування структури додатку

Організація навігаційного графа та розподіл користувацьких інтерфейсів формують основу структурної моделі застосунку. Оскільки візуальна частина розробляється з використанням декларативного фреймворку Jetpack Compose, управління переходами між екранами базується на можливостях бібліотеки Navigation Compose. Концепція, яка передбачала створення окремих компонентів Activity або Fragment для кожного вікна, замінена архітектурою єдиної точки входу (Single Activity Architecture). Усі екрани представлені у вигляді компонованих функцій (composable functions),

які монтуються та демонтуються в межах одного активного вікна залежно від поточного стану навігаційного контролера [20].

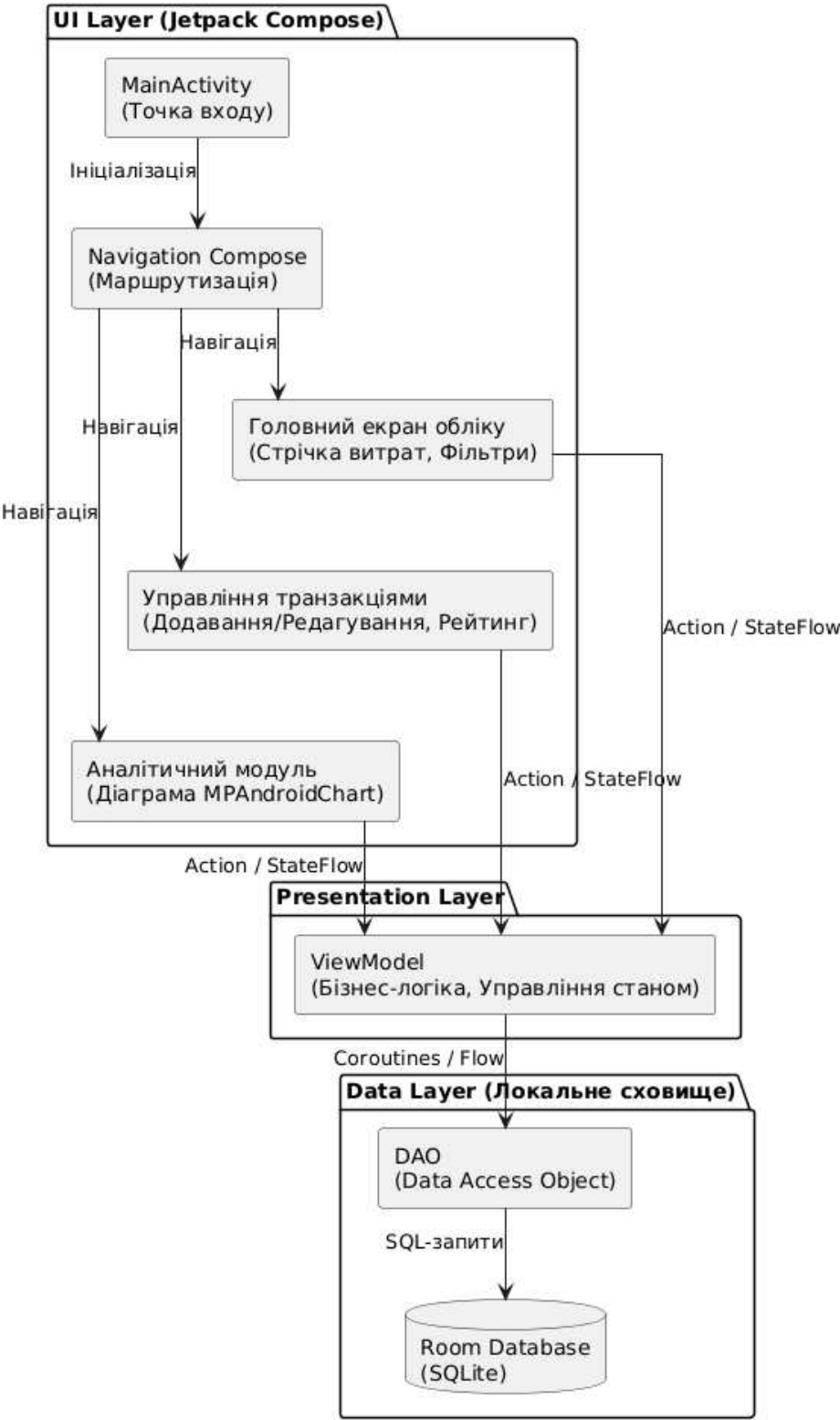


Рисунок 2.1 – Структура додатку

Логічна структура додатка поділяється на три основні функціональні блоки: головний екран обліку, екран управління транзакціями та аналітичний модуль. Головний екран відіграє роль панелі інструментів, де консолідується інформація про останні фінансові операції. Він містить інтерактивну стрічку витрат із відображенням іконки вибраної категорії, відповідного кольорового маркера, суми та дати. Для зручності навігації передбачено механізм швидкого фільтрування записів за часовими періодами, що дозволяє миттєво адаптувати стрічку транзакцій під обраний горизонт планування.

Модуль управління транзакціями призначений для введення нових даних та редагування наявних записів. Екран містить форму з полями для вводу числового значення витрати та вибору відповідної категорії з попередньо встановленого набору. Важливим елементом структури цього модуля є інтерактивний віджет п'ятизіркового рейтингу, за допомогою якого користувач фіксує суб'єктивну оцінку корисності здійсненої покупки. Операції додавання та редагування використовують спільну компоновану функцію, стан якої динамічно змінюється залежно від переданих ідентифікаторів транзакції. Використання єдиного компонента для схожих операцій мінімізує дублювання програмного коду та спрощує подальшу підтримку візуальних елементів.

Аналітичний модуль відповідає за візуалізацію агрегованої фінансової статистики. Його структура включає область відображення кругової діаграми, інтегрованої через обгортку для бібліотеки MPAndroidChart, та текстові блоки з розрахунковими показниками. Графік демонструє відсоткове співвідношення витрат за вибраними категоріями. Нижче розташовується блок обчислення середнього рівня корисності покупок. Цей метричний показник формується шляхом математичного усереднення оцінок, виставлених під час створення записів, і слугує основним інструментом для аналізу раціональності фінансової поведінки.

2.3 Реалізація бази даних

Для забезпечення автономної роботи мобільного додатка механізм збереження даних реалізовано на базі реляційної системи управління базами даних SQLite. Взаємодія з ядром бази даних здійснюється через бібліотеку Room, яка виступає об'єктно-реляційним відображенням (ORM). Застосований інструмент абстрагує розробника від написання низькорівневого коду, гарантує статичну типізацію та виконує перевірку синтаксису SQL-запитів на етапі компіляції. Локальний характер збереження інформації забезпечує миттєвий доступ до фінансової історії без затримок на мережевий обмін та повністю усуває ризики компрометації користувацьких даних на сторонніх серверах [22].

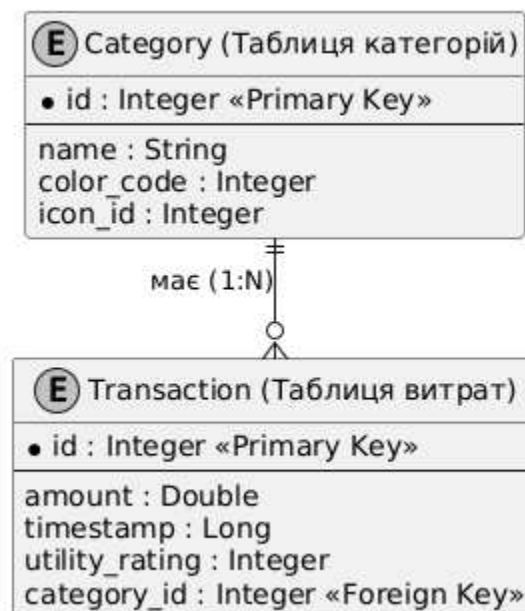


Рисунок 2.2 – Структура бази даних

Логічна структура бази даних проєктується у вигляді двох взаємопов'язаних сутностей, що відображають ключові об'єкти предметної області. Першою сутністю є таблиця категорій, яка зберігає метадані для класифікації фінансових операцій. Кожен запис у цій таблиці містить унікальний первинний ідентифікатор, текстову назву групи, цілочисельний

шістнадцятковий код кольору для візуального маркування та ідентифікатор векторної іконки. Зберігання візуальних атрибутів безпосередньо у таблиці дозволяє системі динамічно генерувати графічний інтерфейс стрічки витрат та закладає архітектурну гнучкість для майбутнього впровадження функції створення користувацьких категорій.

Другою базовою сутністю є таблиця фінансових транзакцій, яка фіксує безпосередні факти витрачання коштів. Структура об'єкта включає первинний ключ, числове значення суми операції, часову мітку транзакції та зовнішній ключ, що вказує на відповідний запис у таблиці категорій. Відмінною рисою спроектованої таблиці є наявність спеціалізованого поля для фіксації оцінки корисності покупки, реалізованого у вигляді цілочисельного значення від одного до п'яти. З метою коректного збереження часових параметрів застосовуються механізми конвертації типів (Type Converters) бібліотеки Room. Вони автоматично перетворюють складні календарні об'єкти мови Kotlin у примітивний формат довгого цілого числа (Long) під час запису у пам'ять та виконують зворотне відновлення об'єктів під час читання [23].

Зв'язок між таблицями організовано за класичною реляційною схемою «один до багатьох», де одній категорії може належати великий масив фінансових записів. Цілісність інформації на рівні файлової системи підтримується обмеженнями зовнішніх ключів. Під час проєктування схеми визначено стратегію каскадного впливу: у разі видалення або модифікації базової категорії механізм SQLite автоматично застосовує відповідні зміни до всіх дочірніх транзакцій, запобігаючи появі осиротілих записів у базі.

Операції читання та запису інкапсульовані в інтерфейсах об'єктів доступу до даних (DAO). Створені інтерфейси містять попередньо скомпільовані запити для вибірки транзакцій за вказаний календарний період, агрегації сумарних витрат у межах кожної групи та математичного обчислення середнього показника корисності. Запропонована структура бази є оптимальною для мобільного застосунку з акцентом на локальну аналітику. Відсутність надлишкової нормалізації та мінімальна кількість таблиць

пришвидшують пошук у фізичній пам'яті мобільного пристрою. Повернення результатів вибірки у форматі реактивних потоків даних гарантує, що будь-яка зміна у таблиці транзакцій миттєво ініціює перерахунок статистики та оновлення діаграм в аналітичному модулі [24].

2.4 Проектування інтерфейсу користувача

Візуальна частина додатка спроектована на основі декларативного підходу за допомогою фреймворку Jetpack Compose та компонентів дизайнерської системи Material Design 3. Відмова від традиційної XML-розмітки на користь написання інтерфейсу мовою Kotlin дозволила інкапсулювати логіку відображення та управління станом у єдиному просторі. Застосування принципів мінімалізму під час проектування забезпечило зниження візуального шуму, дозволяючи користувачеві сфокусуватися безпосередньо на фінансових показниках. Користувацький інтерфейс структурно поділений на три ключові екрани, кожен з яких відповідає за окремий сценарій взаємодії з даними.

Головний екран (HomeScreen) виконує функцію центральної панелі управління фінансами. Основну площу інтерфейсу займає вертикально прокручувана стрічка транзакцій, реалізована за допомогою компонента LazyColumn. Цей елемент забезпечує високу продуктивність відмальовування великих масивів даних шляхом рендерингу лише тих записів, які знаходяться у видимій області дисплея. Кожен рядок списку відображає суму витрати, дату створення, а також відповідну іконку та колір обраної категорії. У верхній частині екрана розташовано панель швидкої фільтрації, яка дозволяє обмежувати вибірку записів за поточний день, тиждень або місяць. Ініціація процесу додавання нового фінансового запису здійснюється через плаваючу кнопку дії (Floating Action Button), яка статично закріплена у нижньому правому куті.

Екран додавання витрати (AddExpenseScreen) спроектовано як форму збору даних із вбудованими механізмами валідації вводу. Введення числового

значення суми здійснюється через спеціалізоване текстове поле, яке автоматично викликає системну цифрову клавіатуру. Вибір категорії реалізовано у вигляді адаптивної сітки компонентів (LazyVerticalGrid), де кожен елемент містить графічний ідентифікатор та текстову назву. Ключовим функціональним елементом цього екрана є інтерактивний віджет п'ятизіркового рейтингу, розроблений для фіксації суб'єктивної оцінки корисності здійсненої покупки. Компонована функція цього модуля є універсальною та адаптується як для створення нових записів, так і для редагування існуючих транзакцій, динамічно змінюючи свій стан на основі отриманих від ViewModel конфігураційних параметрів.

Аналітичний модуль (StatisticsScreen) призначений для візуалізації агрегованої фінансової статистики. Центральним елементом екрана є кругова діаграма, побудована з використанням бібліотеки MPAndroidChart. Оскільки зазначена бібліотека розроблена для імперативної парадигми, її інтеграція у декларативний простір Jetpack Compose виконана за допомогою спеціальної обгортки AndroidView. Це рішення дозволяє повністю контролювати життєвий цикл графічного об'єкта та оновлювати дані при зміні стану застосунку. Діаграма поділена на кольорові сегменти, площа яких пропорційна обсягу витрат за відповідними категоріями. Під графічним блоком розташована зведена текстова інформація, яка включає загальну суму витрат за обраний період та середній показник рейтингу корисності покупок.

Усі спроектовані екрани підтримують динамічну зміну теми оформлення відповідно до глобальних налаштувань операційної системи. Використання палітри кольорів Material 3 гарантує належний рівень контрастності елементів керування та читабельності тексту як у світлому, так і в темному режимах роботи додатка.

2.5 Проєктування системи категорій та аналітики

Класифікація фінансових операцій є базовою вимогою для побудови інформативних аналітичних звітів. У розроблюваному додатку система

категорій спроектована як динамічний об'єктний компонент, керований через реляційну базу даних. Кожен елемент класифікатора містить текстову назву, унікальний колір та графічну іконку. Для оптимізації фізичного зберігання та швидкодії рендерингу інтерфейсу кольорові маркери записуються у форматі цілочисельних значень колірному простору ARGB, а іконки зберігаються як числові ідентифікатори системних векторних ресурсів. Такий формат даних дозволяє функціям Jetpack Compose миттєво зчитувати візуальні параметри без необхідності проміжних конвертацій та будувати адаптивну сітку для вибору категорій під час створення транзакції.

Для навігації масивом накопиченої інформації реалізовано механізм часової фільтрації. Обробка умов вибірки виконується на рівні ядра SQLite через параметризовані запити в DAO-інтерфейсах. Замість завантаження всього обсягу транзакцій у оперативну пам'ять із подальшим програмним сортуванням на боці клієнта, рушій бази даних фільтрує записи за значеннями часових міток (timestamps) і повертає у ViewModel виключно релевантні об'єкти. Зазначена архітектурна оптимізація мінімізує споживання апаратних ресурсів пристрою та пришвидшує відмальовування графічного інтерфейсу під час перемикання між денним, тижневим або місячним періодами відображення.

Ключовою відмінністю проєкту виступає розширений модуль якісної аналітики. Традиційні системи фінансового обліку фокусуються на кількісних показниках витрачених коштів, залишаючи поза увагою психологічний аспект споживання. З метою поглиблення аналітичного потенціалу в систему імплементовано метрику суб'єктивної корисності покупок. Під час фіксації кожної транзакції користувач оцінює її доцільність за п'ятибальною шкалою через інтерактивний віджет. Введене значення зберігається як окремий числовий атрибут у таблиці транзакцій.

Під час генерації фінансового звіту аналітичний алгоритм виконує математичне усереднення зібраних оцінок за вибраний період. Отриманий агрегований індекс корисності демонструється на екрані статистики разом із

круговою діаграмою загальних витрат. Впровадження цього механізму перетворює класичний мобільний трекер на інструмент поведінкового аналізу. Можливість ретроспективного зіставлення витрачених сум із зафіксованим рівнем доцільності придбаних товарів сприяє ідентифікації патернів імпульсивного споживання та допомагає оптимізувати персональний бюджет.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ОСОБИСТИХ ВИТРАТ НА ПЛАТФОРМІ ANDROID

3.1 Алгоритм роботи додатку

Логіка функціонування мобільного додатка побудована на принципах реактивного програмування та подієво-орієнтованої архітектури. Життєвий цикл програми розпочинається з ініціалізації об'єктів бази даних Room та компонентів ViewModel, які відповідають за утримання стану графічного інтерфейсу. Після запуску система зчитує конфігурацію часового фільтра, який за замовчуванням охоплює поточний місяць, і формує відповідний параметризований SQL-запит до локального сховища SQLite.

Отриманий масив фінансових транзакцій повертається у вигляді безперервного потоку даних, що ініціює відмальовування головного екрана додатка. На цьому етапі алгоритм переходить у стан очікування дій користувача. Залежно від вибору оператора, виконання програми розгалужується на кілька незалежних сценаріїв. Сценарій створення або редагування запису передбачає перехід до форми введення, де відбувається збір вхідних параметрів: числового значення суми, ідентифікатора вибраної категорії та оцінки суб'єктивної корисності придбання. Після внутрішньої валідації введеної інформації алгоритм виконує асинхронну операцію запису об'єкта в базу даних через фоновий потік.

Сценарій роботи з аналітикою ініціює перехід до модуля статистики. Програмний код виконує агрегацію збережених показників, групує суми за ідентифікаторами категорій, та математично розраховує загальний індекс корисності покупок для обраного періоду. Результати обчислень передаються до зовнішньої бібліотеки для побудови кругової діаграми. Сценарій зміни умов фільтрації полягає в оновленні параметрів стану в ViewModel. Це провокує формування нового запиту до бази та миттєве оновлення інформації

на екрані без необхідності перезавантаження вікна. Операція видалення транзакції захищена модальним діалоговим вікном, яке вимагає додаткового підтвердження намірів перед фактичним знищенням запису на фізичному носії.

Оскільки архітектура застосунку спирається на принцип єдиного джерела істини (Single Source of Truth), будь-яка успішна операція модифікації даних автоматично ініціює новий виток базового циклу. Ядро бази даних генерує подію оновлення таблиці, потік даних передає актуальну інформацію у ViewModel, а фреймворк Jetpack Compose самостійно перемальовує змінені компоненти екрана.

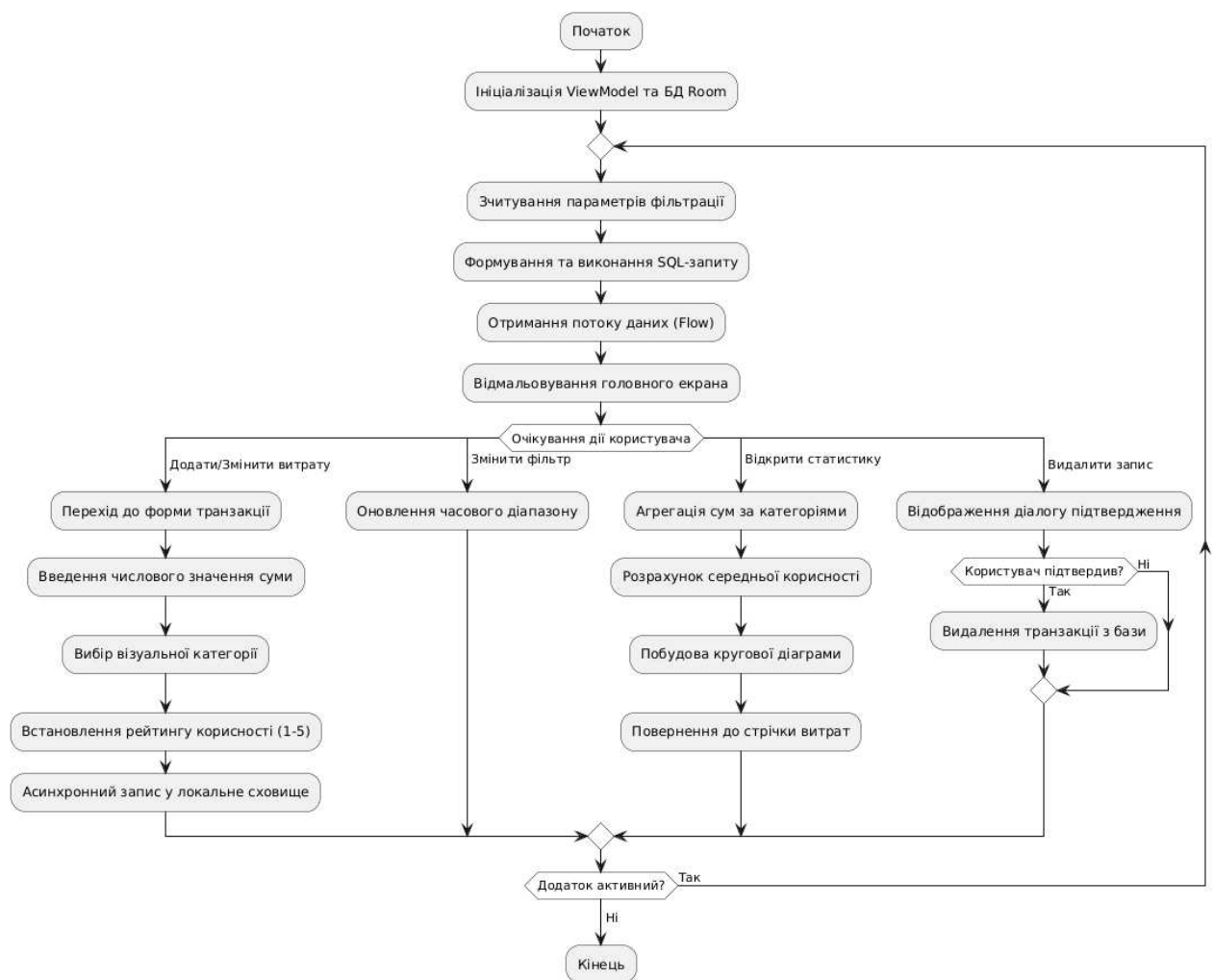


Рисунок 3.1 – Алгоритм роботи додатку

Спроектований алгоритм мінімізує кількість проміжних станів та виключає необхідність ручного оновлення списків, гарантуючи консистентність фінансових даних на всіх екранах додатка під час виконання будь-яких операцій.

3.2 Налаштування середовища розробки

Базовим інструментом для програмної реалізації мобільного додатка обрано інтегроване середовище розробки Android Studio. Це офіційне рішення від корпорації Google, побудоване на ядрі платформи IntelliJ IDEA, яке містить спеціалізований інструментарій для створення нативних застосунків під операційну систему Android. Використання цього середовища забезпечує доступ до засобів статичного аналізу коду (Lint), профілювальників пам'яті та механізмів попереднього перегляду компонентів Jetpack Compose у режимі реального часу. Налаштування робочого простору розпочалося зі встановлення актуальної стабільної версії IDE та конфігурації параметрів віртуальної машини Java (JVM) для підвищення продуктивності процесів компіляції [31].

Основою для взаємодії розроблюваного програмного забезпечення з операційною системою є набір інструментів Android SDK (Software Development Kit). Він містить необхідні бібліотеки прикладного програмного інтерфейсу (API) та системні утиліти, які відповідають за перетворення вихідного коду в бінарний формат. Під час ініціалізації проєкту було зафіксовано мінімально підтримувану версію платформи (MinSdkVersion) на рівні API 24. Цей показник гарантує сумісність із сучасними архітектурними компонентами, зокрема бібліотекою Room, та охоплює переважну більшість активних пристроїв на ринку. Цільову версію (TargetSdkVersion) налаштовано відповідно до новітніх вимог безпеки платформи Android. Через менеджер SDK також були завантажені системні образи для конфігурації емулятора, необхідного для первинного тестування поведінки графічного інтерфейсу на екранах різної роздільної здатності [32].

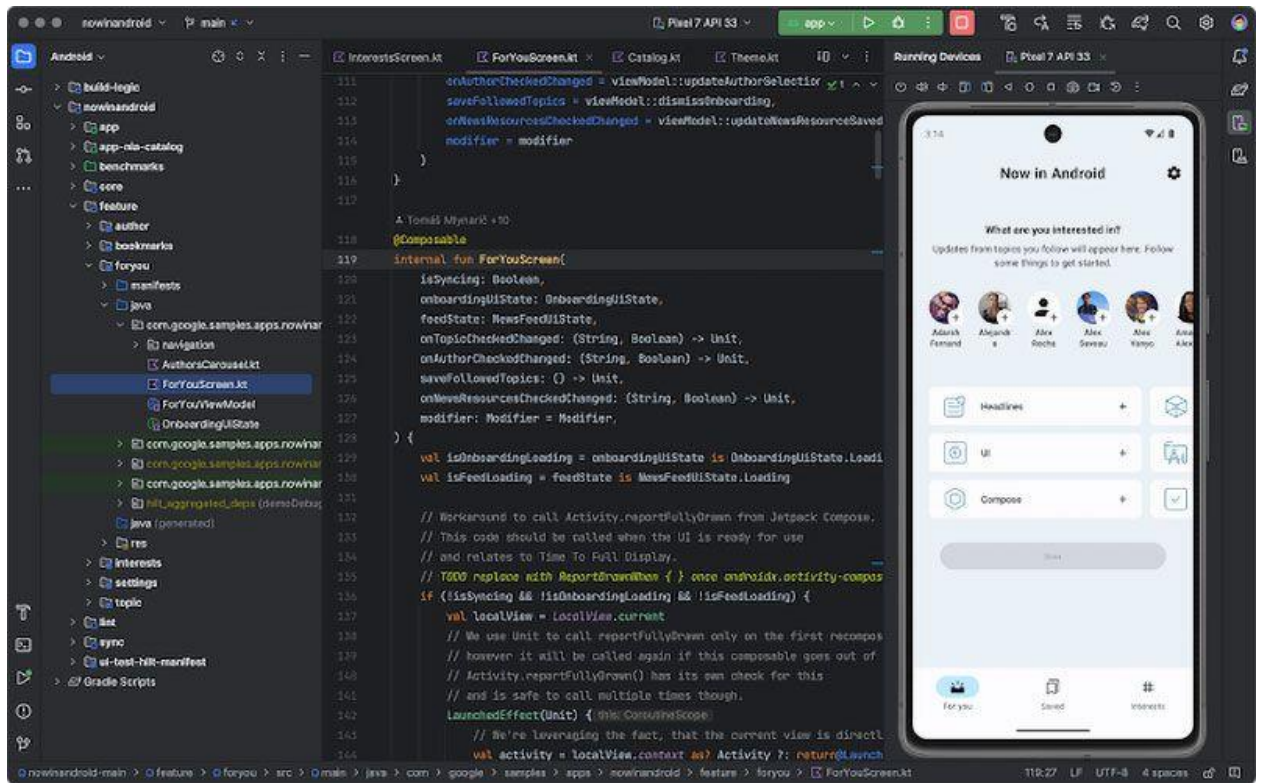


Рисунок 3.2 – Android Studio

Процес компіляції та автоматизації збирання проєкту керується системою Gradle. Для написання складальних скриптів застосовано декларативний синтаксис Kotlin DSL. Це рішення забезпечило єдність мови програмування на всіх рівнях розробки та дозволило повноцінно використовувати можливості автодоповнення коду в межах середовища Android Studio. Файли конфігурації Gradle структурують ієрархію модулів, фіксують версії компіляторів та керують підключенням зовнішніх залежностей. Через цей механізм до проєкту інтегровано бібліотеки Jetpack Compose, Room, Coroutines та MPAndroidChart. Система Gradle самостійно завантажує вказані артефакти з віддалених репозиторіїв, вирішує можливі конфлікти транзитивних залежностей та виконує послідовну трансформацію програмного коду, ресурсів і маніфесту в кінцевий виконуваний файл формату APK для його подальшого розгортання на фізичному пристрої [33].

3.3 Керівництво користувача та опис роботи програми

Запуск мобільного додатка ініціює завантаження головного екрана (HomeScreen), який виконує роль центральної панелі управління фінансами користувача. Інтерфейс демонструє користувачеві стрічку збережених транзакцій, сформованою на основі даних з локальної бази SQLite. Кожен запис у цій стрічці є комплексним інформаційним блоком, що містить суму витрати, дату здійснення операції, довільний текстовий коментар для додаткової деталізації та візуальний ідентифікатор категорії у вигляді унікальної кольорової іконки.

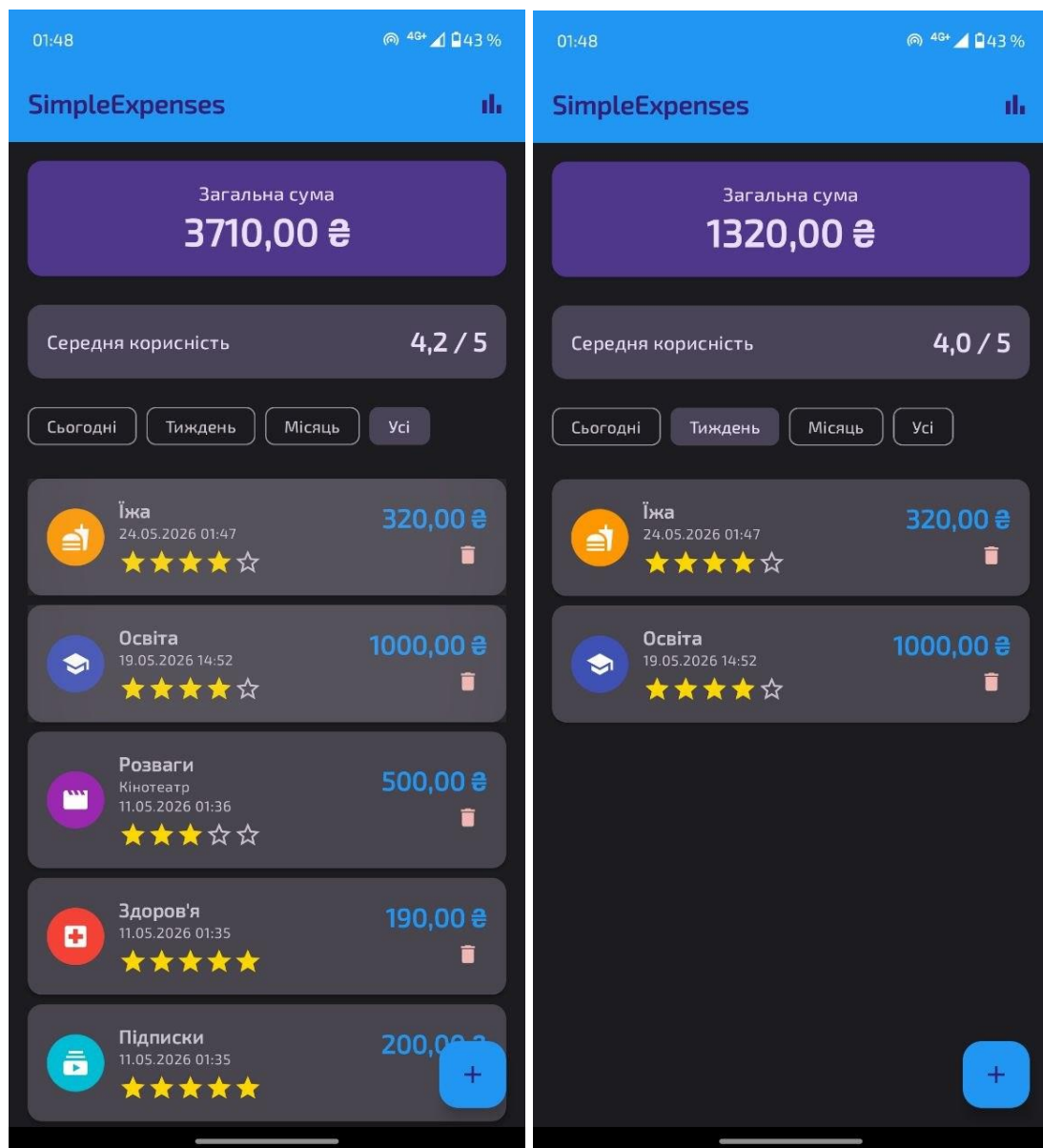


Рисунок 3.3 – Головне вікно додатку

01:48 4G+ 43 %

← Нова витрата

Сума (€)

Категорія

Їжа Транспорт

Дім Покупки

Розваги Здоров'я

Освіта Робота

Підписки Інше

Примітка (необов'язково)

Оцінка корисності покупки

☆☆☆☆☆

✓ Зберегти

Рисунок 3.4 – Додавання нової витрати

Верхня частина екрана відведена під панель часової фільтрації, яка дозволяє перемикаати відображення записів між поточним днем, тижнем або повним календарним місяцем. Навігація масивом транзакцій здійснюється за допомогою вертикального прокручування, а для ініціалізації процесу створення нового фінансового запису передбачена кнопка дії, зафіксована у нижній правій частині дисплея.

Натискання на вказану кнопку відкриває форму реєстрації витрати (AddExpenseScreen). Процес фіксації фінансової операції розпочинається з введення значення суми. Далі користувачеві пропонується взаємодіяти з адаптивною сіткою доступних категорій. Кожен елемент цієї сітки має відповідну графічну іконку, що суттєво пришвидшує пошук потрібного класифікатора.

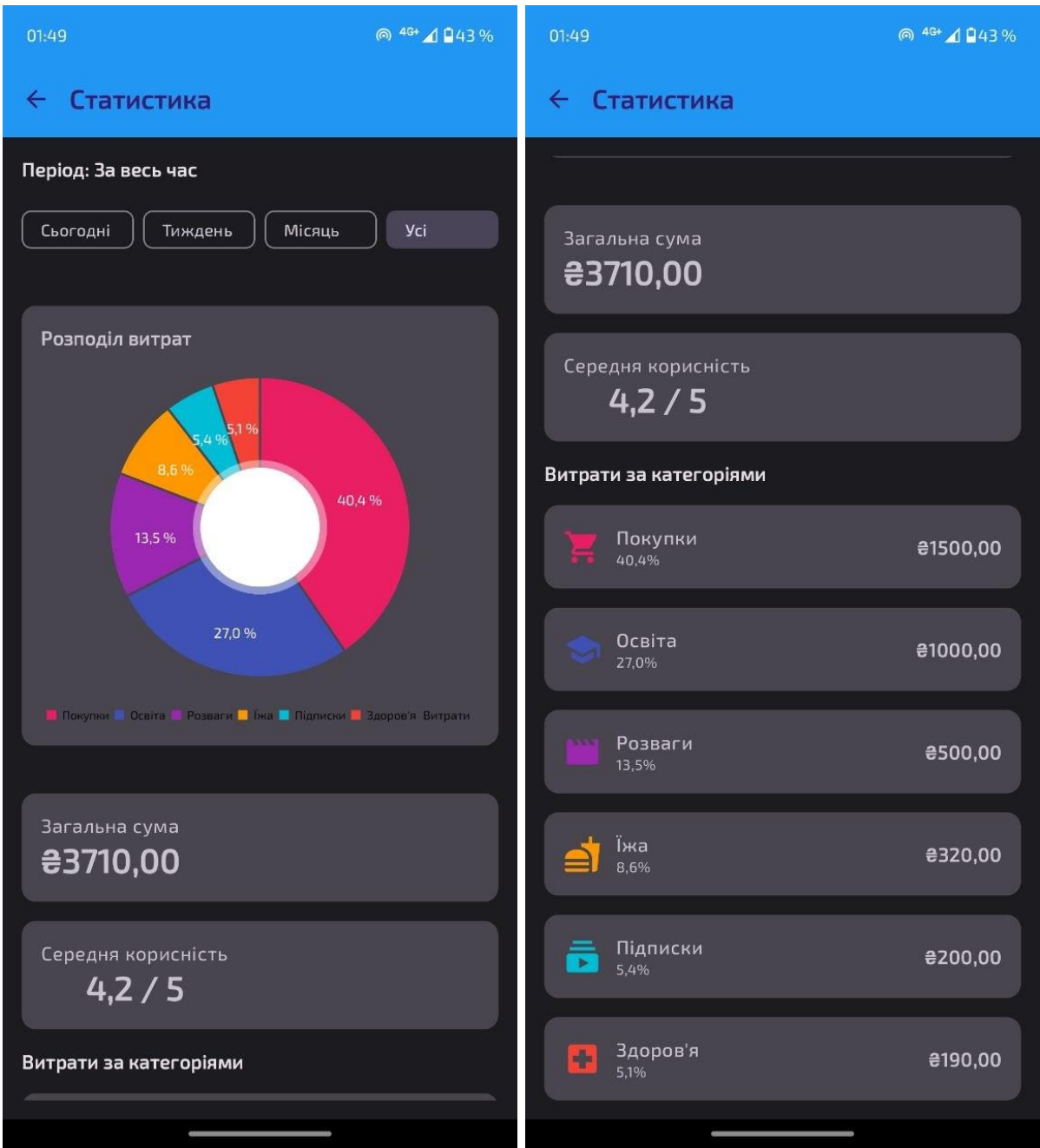


Рисунок 3.5 – Перегляд статистики

Важливим етапом заповнення форми є додавання текстового коментаря, який допомагає уточнити специфічне призначення платежу, та встановлення рейтингу корисності покупки. Процес оцінювання реалізується за допомогою інтерактивного віджета у вигляді п'яти зірок, де користувач фіксує рівень власного задоволення або об'єктивної доцільності витрачених коштів. Завершується процедура натисканням відповідної кнопки збереження, після чого алгоритм записує об'єкт транзакції у пам'ять пристрою та автоматично повертає користувача на головний екран.



Рисунок 3.6 – Виконання фільтрації у вікні статистики

01:49 4G+ 43 %

← Редагування витрати

Сума (€)

320.0

Категорія

Їжа Транспорт

Дім Покупки

Розваги Здоров'я

Освіта Робота

Підписки Інше

Примітка (необов'язково)

Оцінка корисності покупки

★★★★☆

✓ Оновити

Рисунок 3.7 – Редагування витрати

Для комплексного аналізу фінансової поведінки передбачено перехід до аналітичного модуля (StatisticsScreen). Цей екран агрегує накопичені кількісні дані та трансформує їх у наочну графічну форму за допомогою кругової діаграми. Кожен сегмент побудованого графіка відповідає певній групі витрат. Інтерфейс статистики підтримує аналогічні до головного екрана параметри

часової фільтрації, адаптуючи розрахунки під обраний горизонт планування. Крім загальної суми за період, під діаграмою виводиться розрахунковий індекс середньої корисності придбань. Цей показник обчислюється на основі виставлених під час створення транзакцій п'ятизіркових оцінок, надаючи інструментарій для швидкої ідентифікації частки імпульсивних чи нерациональних витрат.

Загальний сценарій взаємодії з програмним продуктом побудований за принципом мінімізації кількості кроків, необхідних для досягнення цільової дії. Повна відсутність процедур створення облікового запису, авторизації та екранів налаштування хмарної синхронізації дозволяє розпочати повноцінний облік бюджету безпосередньо після інсталяції пакета на мобільний пристрій. Концепція графічного інтерфейсу базується на актуальних стандартах проєктування взаємодії з користувачем, що забезпечує інтуїтивно зрозуміле керування функціями застосунку без необхідності попереднього ознайомлення із документацією.

ВИСНОВКИ

Виконана дипломна робота присвячена вирішенню завдання створення безпечного та автономного інструменту для контролю особистих фінансів. Попереднє вивчення предметної області підтвердило, що переважна більшість існуючих програмних продуктів орієнтована на обов'язкову хмарну синхронізацію та сервісну модель розповсюдження. Залежність від зовнішніх серверів створює ризики для конфіденційності персональної інформації та ускладнює графічний інтерфейс надлишковими модулями. Традиційні трекери фіксують виключно кількісні показники витрачених коштів, залишаючи поза увагою якісний аспект споживання та не надаючи інструментів для боротьби з імпульсивними покупками.

Для подолання виявлених недоліків спроектовано та розроблено нативний мобільний додаток під управлінням операційної системи Android. Технічною основою проєкту стала мова програмування Kotlin у поєднанні з декларативним фреймворком Jetpack Compose, що дало змогу створити чуйний мінімалістичний інтерфейс із високою продуктивністю відмальовування. Застосування архітектурного шаблону MVVM забезпечило чітке розмежування візуальних компонентів і бізнес-логіки. Збереження фінансової інформації реалізовано на фізичному носії пристрою через реляційну систему SQLite та бібліотеку абстракції Room. Це інженерне рішення повністю ізолює користувацькі дані від зовнішніх мереж і гарантує працездатність програми без підключення до Інтернету.

Програмна реалізація охоплює повний цикл управління транзакціями, включаючи алгоритми їх створення, редагування, фільтрації за часовими періодами та безпечного видалення. Розроблено гнучку систему класифікації витрат на основі візуальних категорій із застосуванням колірного кодування та векторних ідентифікаторів. Ключовою аналітичною особливістю створеної системи є впровадження механізму якісного оцінювання транзакцій. Завдяки інтерактивному п'ятизірковому рейтингу та можливості залишати текстові

коментарі користувач отримав інструмент для ретроспективного аналізу доцільності придбань. Математична агрегація зібраних оцінок і візуалізація загальної статистики за допомогою обгортки для бібліотеки MPAndroidChart перетворюють базовий облік на засіб виявлення нераціональної фінансової поведінки.

Створений програмний продукт характеризується стабільністю роботи, низьким когнітивним навантаженням на оператора та абсолютною інформаційною автономністю. Відмова від клієнт-серверної взаємодії, відсутність процедур реєстрації та фокус на локальній обробці інформації дозволили реалізувати концепцію приватного фінансового інструменту. Поставлені на початку роботи завдання виконані в повному обсязі, архітектурні та функціональні вимоги дотримані, що підтверджує готовність мобільного додатка до розгортання на реальних пристроях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Кізима Т. О. Фінанси домогосподарств: сучасна парадигма та домінанти розвитку. Київ : Знання, 2010. 431 с.
2. Lusardi A., Mitchell O. S. The Economic Importance of Financial Literacy: Theory and Evidence. *Journal of Economic Literature*. 2014. Vol. 52, No. 1. P. 5–44. DOI: 10.1257/jel.52.1.5.
3. Enck W., Gilbert P., Han S., et al. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems*. 2014. Vol. 32, No. 2. P. 1–29. DOI: 10.1145/2619091.
4. Save data in a local database using Room / Android Developers. URL: <https://developer.android.com/training/data-storage/room> (дата звернення: 21.05.2026).
5. П'ятикоп О. Є. Мобільний застосунок аналізу та прогнозування особистих витрат. *Вісник Приазовського державного технічного університету. Серія: Технічні науки*. 2024. Вип. 48. С. 120–127.
6. Mijić M., Ćebić B. Mobile Applications for Personal Finance Management: Technology Acceptance Perspective. *Digital Transformation of the Financial Industry*. Springer, Cham, 2023. P. 273–295. DOI: 10.1007/978-3-031-23269-5_16.
7. Davis F. D. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*. 1989. Vol. 13, No. 3. P. 319–340. DOI: 10.2307/249008.
8. Guide to app architecture / Android Developers. URL: <https://developer.android.com/topic/architecture> (дата звернення: 22.05.2026).
9. Bass L., Clements P., Kazman R. *Software Architecture in Practice*. 4th ed. Boston : Addison-Wesley Professional, 2021. 448 p.
10. Alqahtani A. S., Al-Badi A. H. Evaluating the Usability of Personal Finance Applications. *International Journal of Computer Science and Network Security*. 2019. Vol. 19, No. 12. P. 165–172.

11. Ghasemi M., et al. Mobile banking security and privacy: A review. *Computers & Security*. 2022. Vol. 113. Article 102551. DOI: 10.1016/j.cose.2021.102551.
12. Спірінцев В., Спірінцева О., Веселов В. Дослідження ефективності використання технологій XML та Jetpack Compose при розробці інтерфейсу мобільних додатків для ОС Android. *Information Technology: Computer Science, Software Engineering and Cyber Security*. 2025. № 2. С. 131–142. DOI: 10.32782/IT/2025-2-14.
13. Jemerov D., Isakova S. *Kotlin in Action*. Shelter Island : Manning Publications, 2017. 360 p.
14. Fjodorovs I., Kodors S. Jetpack Compose and XML Layout Rendering Performance Comparison. *Human. Environment. Technology*. 2021. No. 25. P. 49–54. DOI: 10.17770/het2021.25.6779.
15. Park S., et al. Testing Room Database Interactions: A Case Study. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*. 2022. Vol. 6, No. 3. P. 1–22. DOI: 10.1145/3550304.
16. Griffiths D., Griffiths D. *Head First Android Development: A Brain-Friendly Guide*. 3rd ed. Beijing : O'Reilly Media, 2021. 926 p.
17. Späth P. *Pro Android with Kotlin*. Berkeley : Apress, 2018. 343 p. DOI: 10.1007/978-1-4842-3826-4.
18. Jemerov D., Isakova S. *Kotlin in Action*. Shelter Island : Manning Publications, 2017. 360 p.
19. Phillips B., Stewart C., Hardy B., Marsicano K. *Android Programming: The Big Nerd Ranch Guide*. 4th ed. Atlanta : Big Nerd Ranch Guides, 2019. 696 p.
20. Sin C. *Android UI Development with Jetpack Compose: Bring declarative and reactive UIs to your Android apps*. Birmingham : Packt Publishing, 2022. 354 p.
21. Lee W. M. *Beginning Android Programming with Kotlin*. Hoboken : John Wiley & Sons, 2023. 400 p.
22. Horton J. *Android Programming for Beginners*. 3rd ed. Birmingham : Packt Publishing, 2021. 786 p.

23. Meier R., Lake I. Professional Android. 4th ed. Indianapolis : Wrox, 2018. 1056 p.
24. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2015. 1280 p.

Додаток А - Програмний код

MainActivity.kt

```
package com.example.simpleexpenses

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import androidx.lifecycle.viewmodel.compose.viewModel
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.example.simpleexpenses.data.ExpenseDatabase
import com.example.simpleexpenses.ui.AddExpenseScreen
import com.example.simpleexpenses.ui.HomeScreen
import com.example.simpleexpenses.ui.StatisticsScreen
import com.example.simpleexpenses.ui.theme.SimpleExpensesTheme
import com.example.simpleexpenses.viewmodel.ExpenseViewModel
import
com.example.simpleexpenses.viewmodel.ExpenseViewModelFactory

/**
 * Головна Activity додатку
 * Встановлює Compose UI, тему та навігацію
 */
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            SimpleExpensesTheme {
                // Поверхня для всього контенту
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    // Ініціалізація бази даних
                    val database =
ExpenseDatabase.getDatabase(applicationContext)

                    // Ініціалізація ViewModel
                    val viewModel: ExpenseViewModel = viewModel(
                        factory =
ExpenseViewModelFactory(database)
                    )

                    // Навігаційний контролер
```



```

        val navController = rememberNavController()

        // Навігаційний граф
        NavHost(
            navController = navController,
            startDestination = "home"
        ) {
            // Головний екран
            composable("home") {
                HomeScreen(
                    viewModel = viewModel,
                    onAddExpense = {

navController.navigate("add_expense/null")
                                },
                                onEditExpense = { expenseId ->

navController.navigate("add_expense/${expenseId}")
                                },
                                onShowStatistics = {

navController.navigate("statistics")
                                }
                            )
                        }

            // Екран додавання витрати
            composable("add_expense/{expenseId}") {
backStackEntry ->
                val expenseId =
backStackEntry.arguments?.getString("expenseId")?.toLongOrNull()
                AddExpenseScreen(
                    expenseId = expenseId,
                    viewModel = viewModel,
                    onNavigateBack = {
                        navController.popBackStack()
                    }
                )
            }

            // Екран статистики
            composable("statistics") {
                StatisticsScreen(
                    viewModel = viewModel,
                    onNavigateBack = {
                        navController.popBackStack()
                    }
                )
            }
        }
    }
}

```

```

    }
}
}

```

ExpenseViewModel.kt

```

package com.example.simpleexpenses.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.viewModelScope
import com.example.simpleexpenses.data.Expense
import com.example.simpleexpenses.data.ExpenseDatabase
import com.example.simpleexpenses.utils.DateUtils
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch

/**
 * Enum для фільтрації витрат за періодом
 */
enum class DateFilter {
    TODAY,        // Сьогодні
    WEEK,         // Тиждень
    MONTH,       // Місяць
    ALL           // Усі
}

/**
 * ViewModel для управління витратами
 * Підключається до бази даних через DAO та надає StateFlow для
 * UI
 */
class ExpenseViewModel(private val database: ExpenseDatabase) :
    ViewModel() {

    private val dao = database.expenseDao()

    // Поточний фільтр за датою
    private val _dateFilter = MutableStateFlow(DateFilter.ALL)
    val dateFilter: StateFlow<DateFilter> =
        _dateFilter.asStateFlow()

    // Список витрат з урахуванням фільтра
    val expenses: StateFlow<List<Expense>> = _dateFilter
        .flatMapLatest { filter ->
            when (filter) {
                DateFilter.TODAY ->
                    dao.getExpensesByDate(DateUtils.getStartOfDay())
                DateFilter.WEEK ->
                    dao.getExpensesByDate(DateUtils.getStartOfWeek())
                DateFilter.MONTH ->
                    dao.getExpensesByDate(DateUtils.getStartOfMonth())
            }
        }
    }

```

```

        DateFilter.ALL -> dao.getAllExpenses()
    }
}
.stateIn(
    scope = viewModelScope,
    started = SharingStarted.WhileSubscribed(5000),
    initialValue = emptyList()
)

// Загальна сума витрат з урахуванням фільтра
val totalAmount: StateFlow<Double> = expenses
    .map { expenses -> expenses.sumOf { it.amount } }
    .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed(5000),
        initialValue = 0.0
    )

// Середня корисність покупок з урахуванням фільтра
val averageRating: StateFlow<Double> = expenses
    .map { expenses ->
        val ratedExpenses = expenses.filter {
it.usefulnessRating > 0 }
        if (ratedExpenses.isEmpty()) 0.0
        else ratedExpenses.map { it.usefulnessRating
}.average()
    }
    .stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed(5000),
        initialValue = 0.0
    )

/**
 * Змінює фільтр за датою
 * @param filter новий фільтр
 */
fun setDateFilter(filter: DateFilter) {
    _dateFilter.value = filter
}

/**
 * Додає нову витрату в базу даних
 * @param amount сума витрати
 * @param category категорія витрати
 * @param note примітка до витрати
 * @param usefulnessRating оцінка корисності покупки (1-5)
 */
fun addExpense(amount: Double, category: String, note:
String, usefulnessRating: Int = 0) {
    viewModelScope.launch {
        val expense = Expense(

```

```

        amount = amount,
        category = category,
        note = note,
        date = DateUtils.getCurrentTime(),
        usefulnessRating = usefulnessRating
    )
    dao.insert(expense)
}

/**
 * Оновлює існуючу витрату в базі даних
 * @param id ідентифікатор витрати
 * @param amount нова сума витрати
 * @param category нова категорія витрати
 * @param note нова примітка до витрати
 * @param usefulnessRating нова оцінка корисності покупки
(1-5)
 */
fun updateExpense(id: Long, amount: Double, category:
String, note: String, usefulnessRating: Int) {
    viewModelScope.launch {
        val expense = Expense(
            id = id,
            amount = amount,
            category = category,
            note = note,
            date = DateUtils.getCurrentTime(),
            usefulnessRating = usefulnessRating
        )
        dao.updateExpense(expense)
    }
}

/**
 * Отримує витрату за ідентифікатором
 * @param id ідентифікатор витрати
 * @return об'єкт витрати або null
 */
suspend fun getExpenseById(id: Long): Expense? {
    return dao.getExpenseById(id)
}

/**
 * Видаляє витрату з бази даних
 * @param expense витрата для видалення
 */
fun deleteExpense(expense: Expense) {
    viewModelScope.launch {
        dao.delete(expense)
    }
}

```

```

    /**
     * Видаляє всі витрати з бази даних
     */
    fun deleteAllExpenses() {
        viewModelScope.launch {
            dao.deleteAllExpenses()
        }
    }
}

/**
 * Factory для створення ExpenseViewModel з параметрами
 */
class ExpenseViewModelFactory(private val database:
ExpenseDatabase) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T
    {
        if
        (modelClass.isAssignableFrom(ExpenseViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return ExpenseViewModel(database) as T
        }
        throw IllegalArgumentException("Unknown ViewModel
class")
    }
}

```

HomeScreen.kt

```

package com.example.simpleexpenses.ui

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.BarChart
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.example.simpleexpenses.data.Expense
import com.example.simpleexpenses.ui.components.ExpenseItem
import com.example.simpleexpenses.ui.components.FilterChips
import com.example.simpleexpenses.viewmodel.DateFilter
import com.example.simpleexpenses.viewmodel.ExpenseViewModel
import kotlinx.coroutines.launch

/**

```

```

* Головний екран додатку
* Відображає список витрат з фільтрацією та загальною сумою
* @param viewModel ViewModel для управління даними
* @param onAddExpense callback для переходу на екран додавання
витрати
* @param onEditExpense callback для переходу на екран
редагування витрати
* @param onShowStatistics callback для переходу на екран
статистики
*/
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun HomeScreen(
    viewModel: ExpenseViewModel,
    onAddExpense: () -> Unit,
    onEditExpense: (Long) -> Unit,
    onShowStatistics: () -> Unit
) {
    // Збираємо StateFlow з ViewModel
    val expenses by viewModel.expenses.collectAsState()
    val totalAmount by viewModel.totalAmount.collectAsState()
    val averageRating by
viewModel.averageRating.collectAsState()
    val dateFilter by viewModel.dateFilter.collectAsState()

    // Scaffold для структури екрану
    Scaffold(
        topBar = {
            TopAppBar(
                title = {
                    Text(
                        text = "SimpleExpenses",
                        fontWeight = FontWeight.Bold
                    )
                },
                actions = {
                    IconButton(onClick = onShowStatistics) {
                        Icon(
                            imageVector =
Icons.Default.BarChart,
                            contentDescription = "Статистика",
                            tint =
MaterialTheme.colorScheme.onPrimary
                        )
                    }
                },
                colors = TopAppBarDefaults.topAppBarColors(
                    containerColor =
MaterialTheme.colorScheme.primary,
                    titleContentColor =
MaterialTheme.colorScheme.onPrimary,

```

```

        actionIconContentColor =
MaterialTheme.colorScheme.onPrimary
    )
    ),
    floatingActionButton = {
        FloatingActionButton(
            onClick = onAddExpense,
            containerColor =
MaterialTheme.colorScheme.primary
        ) {
            Icon(
                imageVector = Icons.Default.Add,
                contentDescription = "Додати витрату",
                tint = MaterialTheme.colorScheme.onPrimary
            )
        }
    }
) { paddingValues ->
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
    ) {
        // Загальна сума витрат
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            colors = CardDefaults.cardColors(
                containerColor =
MaterialTheme.colorScheme.primaryContainer
            )
        ) {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp),
                horizontalAlignment =
Alignment.CenterHorizontally
            ) {
                Text(
                    text = "Загальна сума",
                    style =
MaterialTheme.typography.titleMedium,
                    color =
MaterialTheme.colorScheme.onPrimaryContainer
                )
                Text(
                    text = String.format("%.2f ₾",
totalAmount),

```

```

                style =
MaterialTheme.typography.headlineLarge,
                fontWeight = FontWeight.Bold,
                color =
MaterialTheme.colorScheme.onPrimaryContainer
            )
        }
    }

    // Середня корисність покупок
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical =
8.dp),
        colors = CardDefaults.cardColors(
            containerColor =
MaterialTheme.colorScheme.secondaryContainer
        )
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            horizontalArrangement =
Arrangement.SpaceBetween,
            verticalAlignment =
Alignment.CenterVertically
        ) {
            Text(
                text = "Середня корисність",
                style =
MaterialTheme.typography.titleMedium,
                color =
MaterialTheme.colorScheme.onSecondaryContainer
            )
            if (averageRating > 0) {
                Text(
                    text = String.format("%.1f / 5",
averageRating),
                    style =
MaterialTheme.typography.titleLarge,
                    fontWeight = FontWeight.Bold,
                    color =
MaterialTheme.colorScheme.onSecondaryContainer
                )
            } else {
                Text(
                    text = "Немає оцінок",
                    style =
MaterialTheme.typography.bodyMedium,

```



```

        color =
MaterialTheme.colorScheme.onSecondaryContainer
    )
    }
}

// Фільтри за датою
FilterChips(
    currentFilter = dateFilter,
    onFilterChange = { viewModel.setDateFilter(it) }
)

// Список витрат
if (expenses.isEmpty()) {
    // Повідомлення, якщо список порожній
    Box(
        modifier = Modifier
            .fillMaxSize()
            .weight(1f),
        contentAlignment = Alignment.Center
    ) {
        Column(
            horizontalAlignment =
Alignment.CenterHorizontally
        ) {
            Text(
                text = "Немає витрат",
                style =
MaterialTheme.typography.titleLarge,
                color =
MaterialTheme.colorScheme.onSurfaceVariant
            )
            Text(
                text = "Натисніть + щоб додати першу
витрату",
                style =
MaterialTheme.typography.bodyMedium,
                color =
MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
    }
} else {
    LazyColumn(
        modifier = Modifier
            .fillMaxSize()
            .weight(1f),
        contentPadding = PaddingValues(vertical =
8.dp)
    ) {
        items(

```



```

// Список витрат
val expenses by viewModel.expenses.collectAsState()

// Загальна сума
val totalAmount by viewModel.totalAmount.collectAsState()

// Обчислення сум за категоріями
val categorySums = remember(expenses) {
    expenses.groupBy { it.category }
        .mapValues { entry -> entry.value.sumOf { it.amount
} }
        .toList()
        .sortedByDescending { it.second }
}

// Обчислення середньої корисності
val averageRating = remember(expenses) {
    val ratedExpenses = expenses.filter {
it.usefulnessRating > 0 }
    if (ratedExpenses.isEmpty()) 0.0
    else ratedExpenses.map { it.usefulnessRating }.average()
}

Scaffold(
    topBar = {
        TopAppBar(
            title = {
                Text(
                    text = "Статистика",
                    fontWeight = FontWeight.Bold
                )
            },
            navigationIcon = {
                IconButton(onClick = onNavigateBack) {
                    Icon(
                        imageVector =
Icons.Default.ArrowBack,
                        contentDescription = "Назад"
                    )
                }
            },
            colors = TopAppBarDefaults.topAppBarColors(
                containerColor =
MaterialTheme.colorScheme.primary,
                titleContentColor =
MaterialTheme.colorScheme.onPrimary,
                navigationIconContentColor =
MaterialTheme.colorScheme.onPrimary
            )
        )
    }
)

```

```

    ) { paddingValues ->
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(paddingValues)
                .padding(16.dp)
                .verticalScroll(rememberScrollState()),
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            // Фільтр періоду
            Text(
                text = "Період:
${getFilterDisplayName(dateFilter)}",
                style = MaterialTheme.typography.titleMedium,
                fontWeight = FontWeight.Bold
            )

            // Кнопки зміни фільтра
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement =
Arrangement.spacedBy(8.dp)
            ) {
                DateFilter.values().forEach { filter ->
                    FilterChip(
                        selected = dateFilter == filter,
                        onClick = {
viewModel.setDateFilter(filter) },
                        label = {
Text(getFilterShortName(filter)) },
                        modifier = Modifier.weight(1f)
                    )
                }
            }

            Spacer(modifier = Modifier.height(8.dp))

            // Кругова діаграма
            if (categorySums.isNotEmpty()) {
                Card(
                    modifier = Modifier.fillMaxWidth()
                ) {
                    Column(
                        modifier = Modifier.padding(16.dp)
                    ) {
                        Text(
                            text = "Розподіл витрат",
                            style =
MaterialTheme.typography.titleMedium,
                            fontWeight = FontWeight.Bold,
                            modifier = Modifier.padding(bottom =
16.dp)

```

```

        )
        PieChartView(
            categorySums = categorySums
        )
    }
}

Spacer(modifier = Modifier.height(8.dp))

// Загальна сума
Card(
    modifier = Modifier.fillMaxWidth()
) {
    Column(
        modifier = Modifier.padding(16.dp),
        horizontalAlignment =
Alignment.CenterHorizontally
    ) {
        Text(
            text = "Загальна сума",
            style =
MaterialTheme.typography.bodyLarge
        )
        Text(
            text = "€%.2f".format(totalAmount),
            style =
MaterialTheme.typography.headlineMedium,
            fontWeight = FontWeight.Bold
        )
    }
}

// Середня корисність
Card(
    modifier = Modifier.fillMaxWidth()
) {
    Column(
        modifier = Modifier.padding(16.dp),
        horizontalAlignment =
Alignment.CenterHorizontally
    ) {
        Text(
            text = "Середня корисність",
            style =
MaterialTheme.typography.bodyLarge
        )
        if (averageRating > 0) {
            Text(
                text = "%.1f /
5".format(averageRating),

```

```

                style =
MaterialTheme.typography.headlineMedium,
                fontWeight = FontWeight.Bold
            )
        } else {
            Text(
                text = "Немає оцінок",
                style =
MaterialTheme.typography.bodyMedium
            )
        }
    }

    // Список категорій
    Text(
        text = "Витрати за категоріями",
        style = MaterialTheme.typography.titleMedium,
        fontWeight = FontWeight.Bold
    )

    if (categorySums.isEmpty()) {
        Text(
            text = "Немає даних",
            style = MaterialTheme.typography.bodyMedium,
            color =
MaterialTheme.colorScheme.onSurfaceVariant
        )
    } else {
        categorySums.forEach { (category, amount) ->
            CategoryStatItem(
                category = category,
                amount = amount,
                totalAmount = totalAmount
            )
        }
    }
}

}

}

/**
 * Компонент для відображення статистики по категорії
 * @param category назва категорії
 * @param amount сума витрат по категорії
 * @param totalAmount загальна сума для обчислення відсотка
 */
@Composable
fun CategoryStatItem(
    category: String,
    amount: Double,
    totalAmount: Double

```

```

) {
    val percentage = if (totalAmount > 0) (amount / totalAmount
* 100) else 0.0

    Card(
        modifier = Modifier.fillMaxWidth()
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically
        ) {
            // Іконка та назва категорії
            Row(
                horizontalArrangement =
Arrangement.spacedBy(12.dp),
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier.weight(1f)
            ) {
                androidx.compose.material3.Icon(
                    imageVector =
com.example.simpleexpenses.utils.CategoryUtils.getIcon(category)
,
                    contentDescription = null,
                    tint =
com.example.simpleexpenses.utils.CategoryUtils.getColor(category)
),
                    modifier = Modifier.size(32.dp)
                )
                Column {
                    Text(
                        text =
com.example.simpleexpenses.utils.CategoryUtils.getDisplayName(ca
tegory),
                        style =
MaterialTheme.typography.bodyLarge,
                        fontWeight = FontWeight.Medium
                    )
                    Text(
                        text = "%.1f%%".format(percentage),
                        style =
MaterialTheme.typography.bodySmall,
                        color =
MaterialTheme.colorScheme.onSurfaceVariant
                    )
                }
            }

            // Сума
            Text(

```

```

        text = "%%.2f".format(amount),
        style = MaterialTheme.typography.bodyLarge,
        fontWeight = FontWeight.Bold
    )
}
}
}

/**
 * Отримує коротку назву фільтра для відображення на кнопках
 */
fun getFilterShortName(filter: DateFilter): String {
    return when (filter) {
        DateFilter.TODAY -> "Сьогодні"
        DateFilter.WEEK -> "Тиждень"
        DateFilter.MONTH -> "Місяць"
        DateFilter.ALL -> "Усі"
    }
}

/**
 * Отримує повну назву фільтра для відображення в заголовок
 */
fun getFilterDisplayName(filter: DateFilter): String {
    return when (filter) {
        DateFilter.TODAY -> "Сьогодні"
        DateFilter.WEEK -> "Цей тиждень"
        DateFilter.MONTH -> "Цей місяць"
        DateFilter.ALL -> "За весь час"
    }
}

```

AddExpenseScreen.kt

```

package com.example.simpleexpenses.ui

import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material.icons.filled.Check
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import com.example.simpleexpenses.ui.components.RatingStars
import com.example.simpleexpenses.utils.CategoryUtils

/**

```



```

    * Екран додавання нової витрати або редагування існуючої
    * @param expenseId ідентифікатор витрати для редагування (null
для створення нової)
    * @param viewModel ViewModel для доступу до даних
    * @param onBackPressed callback для повернення на попередній
екран
    */
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AddExpenseScreen(
    expenseId: Long?,
    viewModel:
com.example.simpleexpenses.viewmodel.ExpenseViewModel,
    onBackPressed: () -> Unit
) {
    // Стан для полів форми
    var amount by remember { mutableStateOf("") }
    var selectedCategory by remember { mutableStateOf("") }
    var note by remember { mutableStateOf("") }
    var usefulnessRating by remember { mutableStateOf(0) }

    // Визначаємо режим роботи (редагування чи створення)
    val isEditMode = expenseId != null

    // Завантаження даних при редагуванні
    LaunchedEffect(expenseId) {
        if (expenseId != null) {
            val expense = viewModel.getExpenseById(expenseId)
            if (expense != null) {
                amount = expense.amount.toString()
                selectedCategory = expense.category
                note = expense.note
                usefulnessRating = expense.usefulnessRating
            }
        }
    }

    // Стан для відображення помилок
    var amountError by remember { mutableStateOf(false) }
    var categoryError by remember { mutableStateOf(false) }

    // Список категорій
    val categories = CategoryUtils.getAllCategories()

    Scaffold(
        topBar = {
            TopAppBar(
                title = {
                    Text(
                        text = if (isEditMode) "Редагування
витрати" else "Нова витрата",
                        fontWeight = FontWeight.Bold

```

```

        )
    },
    navigationIcon = {
        IconButton(onClick = onNavigateBack) {
            Icon(
                imageVector =
Icons.Default.ArrowBack,
                contentDescription = "Назад"
            )
        }
    },
    colors = TopAppBarDefaults.topAppBarColors(
        containerColor =
MaterialTheme.colorScheme.primary,
        titleContentColor =
MaterialTheme.colorScheme.onPrimary,
        navigationIconContentColor =
MaterialTheme.colorScheme.onPrimary
    )
)
}
) { paddingValues ->
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
            .padding(16.dp)
            .verticalScroll(rememberScrollState()),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        // Поле для введення суми
        OutlinedTextField(
            value = amount,
            onValueChange = {
                amount = it
                amountError = false
            },
            label = { Text("Сума (€)") },
            isError = amountError,
            supportingText = {
                if (amountError) {
                    Text("Введіть суму")
                }
            },
            singleLine = true,
            modifier = Modifier.fillMaxWidth(),
            keyboardOptions =
androidx.compose.foundation.text.KeyboardOptions(
                keyboardType =
androidx.compose.ui.text.input.KeyboardType.Decimal
            )
        )
    }
}

```

```

// Вибір категорії
Text(
    text = "Категорія",
    style = MaterialTheme.typography.titleMedium,
    fontWeight = FontWeight.Bold
)

if (categoryError) {
    Text(
        text = "Оберіть категорію",
        color = MaterialTheme.colorScheme.error,
        style = MaterialTheme.typography.bodySmall
    )
}

// Сітка категорій
CategoryGrid(
    categories = categories,
    selectedCategory = selectedCategory,
    onCategorySelected = {
        selectedCategory = it
        categoryError = false
    }
)

// Поле для примітки
OutlinedTextField(
    value = note,
    onChange = { note = it },
    label = { Text("Примітка (необов'язково)") },
    modifier = Modifier.fillMaxWidth(),
    minLines = 3,
    maxLines = 5
)

// Вибір рейтингу корисності
Text(
    text = "Оцінка корисності покупки",
    style = MaterialTheme.typography.titleMedium,
    fontWeight = FontWeight.Bold
)

RatingStars(
    rating = usefulnessRating,
    onRatingChange = { usefulnessRating = it },
    modifier = Modifier.fillMaxWidth()
)

Spacer(modifier = Modifier.height(16.dp))

// Кнопка збереження

```

```

        Button(
            onClick = {
                // Валідація
                val amountValue = amount.toDoubleOrNull()
                if (amountValue == null || amountValue <= 0)

                    amountError = true
                }
                if (selectedCategory.isEmpty()) {
                    categoryError = true
                }

                // Якщо все валідно, зберігаємо
                if (amountValue != null && amountValue > 0
&& selectedCategory.isNotEmpty()) {
                    if (isEditMode && expenseId != null) {
                        // Режим редагування
                        viewModel.updateExpense(
                            id = expenseId,
                            amount = amountValue,
                            category = selectedCategory,
                            note = note,
                            usefulnessRating =
usefulnessRating
                        )
                    } else {
                        // Режим створення
                        viewModel.addExpense(
                            amount = amountValue,
                            category = selectedCategory,
                            note = note,
                            usefulnessRating =
usefulnessRating
                        )
                    }
                }
                onNavigateBack()
            }
        ),
        modifier = Modifier
            .fillMaxWidth()
            .height(56.dp),
        enabled = amount.isNotEmpty() &&
selectedCategory.isNotEmpty()
    ) {
        Icon(
            imageVector = Icons.Default.Check,
            contentDescription = null,
            modifier = Modifier.size(24.dp)
        )
        Spacer(modifier = Modifier.width(8.dp))
        Text(

```

```

        text = if (isEditMode) "Оновити" else
"Зберегти",
        style = MaterialTheme.typography.titleMedium
    )
    }
}

/**
 * Компонент для відображення сітки категорій
 * @param categories список категорій
 * @param selectedCategory вибрана категорія
 * @param onCategorySelected callback при виборі категорії
 */
@Composable
fun CategoryGrid(
    categories: List<String>,
    selectedCategory: String,
    onCategorySelected: (String) -> Unit
) {
    Column(
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        categories.chunked(2).forEach { row ->
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement =
Arrangement.spacedBy(8.dp)
            ) {
                row.forEach { category ->
                    CategoryChip(
                        category = category,
                        isSelected = selectedCategory ==
category,
                        onClick = { onCategorySelected(category)
},
                        modifier = Modifier.weight(1f)
                    )
                }
                // Якщо в рядку один елемент, додаємо пустий
простір
                if (row.size == 1) {
                    Spacer(modifier = Modifier.weight(1f))
                }
            }
        }
    }
}

/**
 * Компонент для відображення однієї категорії як чіпу

```

```

* @param category назва категорії
* @param isSelected чи вибрана категорія
* @param onClick callback при натисканні
* @param modifier модифікатор
*/
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CategoryChip(
    category: String,
    isSelected: Boolean,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    FilterChip(
        selected = isSelected,
        onClick = onClick,
        label = {
            Row(
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement =
Arrangement.spacedBy(4.dp)
            ) {
                Icon(
                    imageVector =
CategoryUtils.getIcon(category),
                    contentDescription = null,
                    modifier = Modifier.size(18.dp)
                )
                Text(
                    text =
CategoryUtils.getDisplayName(category),
                    maxLines = 1
                )
            }
        },
        modifier = modifier
    )
}

```

PieChartView.kt

```

package com.example.simpleexpenses.ui.components

import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.toArgb
import androidx.compose.ui.unit.dp
import androidx.compose.ui.viewinterop.AndroidView
import com.github.mikephil.charting.charts.PieChart
import com.github.mikephil.charting.data.PieData

```

```

import com.github.mikephil.charting.data.PieDataSet
import com.github.mikephil.charting.data.PieEntry
import com.github.mikephil.charting.formatter.PercentFormatter

/**
 * Компонент для відображення кругової діаграми витрат за
 категоріями
 * @param categorySums список пар (категорія, сума)
 * @param modifier модифікатор для компонента
 */
@Composable
fun PieChartView(
    categorySums: List<Pair<String, Double>>,
    modifier: Modifier = Modifier
) {
    AndroidView(
        factory = { context ->
            PieChart(context).apply {
                // Налаштування діаграми
                description.isEnabled = false
                legend.isEnabled = true
                setDrawEntryLabels(false)
                setUsePercentValues(true)
                isRotationEnabled = false
                isHighlightPerTapEnabled = false

                // Налаштування отвору в центрі
                isDrawHoleEnabled = true
                holeRadius = 40f
                transparentCircleRadius = 45f

                // Налаштування тексту в центрі
                setDrawCenterText(false)
            }
        },
        update = { pieChart ->
            // Підготовка даних
            val totalAmount = categorySums.sumOf { it.second }

            if (totalAmount > 0) {
                val entries = categorySums.map { (category,
amount) ->
                    val percentage = (amount / totalAmount *
100).toFloat()
                    PieEntry(percentage,
com.example.simpleexpenses.utils.CategoryUtils.getDisplayName(ca
tegory))
                }

                val dataSet = PieDataSet(entries,
"Витрати").apply {

```

```

        // Налаштування кольорів для кожної
категорії
        colors = categorySums.map { (category, _) ->
com.example.simpleexpenses.utils.CategoryUtils.getColor(category
).toArgb()
        }

        // Налаштування відступів між секторами
sliceSpace = 2f

        // Вимкнення виділення при натисканні
setDrawValues(false)
    }

    val pieData = PieData(dataSet).apply {
        setDrawValues(true)
    }

    setValueFormatter(PercentFormatter(pieChart))
    setValueTextSize(12f)

    setValueTextColor(android.graphics.Color.WHITE)
}

    pieChart.data = pieData
    pieChart.invalidate()
} else {
    // Якщо немає даних, очищуємо діаграму
    pieChart.clear()
    pieChart.invalidate()
}
},
modifier = modifier
    .fillMaxWidth()
    .height(300.dp)
)
}

```

RatingStars.kt

```

package com.example.simpleexpenses.ui.components

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Row
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Star
import androidx.compose.material.icons.outlined.StarBorder
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color

```



```

/**
 * Компонент для відображення та вибору рейтингу зірок
 * @param rating поточний рейтинг (від 0 до 5)
 * @param onRatingChange callback при зміні рейтингу
 * @param modifier модифікатор для компонента
 * @param enabled чи можна змінювати рейтинг
 */
@Composable
fun RatingStars(
    rating: Int,
    onRatingChange: (Int) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true
) {
    Row(modifier = modifier) {
        // Відображаємо 5 зірок
        for (i in 1..5) {
            Icon(
                imageVector = if (i <= rating)
Icons.Default.Star else Icons.Outlined.StarBorder,
                contentDescription = "Зірка $i",
                tint = if (i <= rating) Color(0xFFFFD700) else
MaterialTheme.colorScheme.onSurfaceVariant,
                modifier = Modifier
                    .clickable(enabled = enabled) {
                        if (enabled) {
                            onRatingChange(i)
                        }
                    }
            )
        }
    }
}

/**
 * Компонент для відображення рейтингу без можливості зміни
(тільки для перегляду)
 * @param rating рейтинг для відображення (від 0 до 5)
 * @param modifier модифікатор для компонента
 */
@Composable
fun RatingStarsDisplay(
    rating: Int,
    modifier: Modifier = Modifier
) {
    Row(modifier = modifier) {
        // Відображаємо 5 зірок
        for (i in 1..5) {
            Icon(
                imageVector = if (i <= rating)
Icons.Default.Star else Icons.Outlined.StarBorder,

```

```

                contentDescription = "Зірка $i",
                tint = if (i <= rating) Color(0xFFFFD700) else
MaterialTheme.colorScheme.onSurfaceVariant
            )
        }
    }
}

```

ExpenseItem.kt

```

package com.example.simpleexpenses.ui.components

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import com.example.simpleexpenses.data.Expense
import com.example.simpleexpenses.utils.CategoryUtils
import com.example.simpleexpenses.utils.DateUtils

/**
 * Компонент для відображення однієї витрати в списку
 * @param expense об'єкт витрати
 * @param onDelete callback для видалення витрати
 * @param onEdit callback для редагування витрати
 */
@Composable
fun ExpenseItem(
    expense: Expense,
    onDelete: () -> Unit,
    onEdit: () -> Unit = {}
) {
    // Стан для відображення діалогу підтвердження
    var showDeleteDialog by remember { mutableStateOf(false) }

    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 4.dp)
            .clickable { onEdit() },
    )
}

```

```

        shape = RoundedCornerShape(12.dp),
        elevation = CardDefaults.cardElevation(defaultElevation
= 2.dp)
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            // Іконка категорії з кольором
            Box(
                modifier = Modifier
                    .size(48.dp)
                    .clip(CircleShape)

                .background(CategoryUtils.getColor(expense.category)),
                contentAlignment = Alignment.Center
            ) {
                Icon(
                    imageVector =
CategoryUtils.getIcon(expense.category),
                    contentDescription = null,
                    tint = Color.White,
                    modifier = Modifier.size(24.dp)
                )
            }

            Spacer(modifier = Modifier.width(12.dp))

            // Інформація про витрату
            Column(
                modifier = Modifier.weight(1f)
            ) {
                Text(
                    text =
CategoryUtils.getDisplayName(expense.category),
                    style =
MaterialTheme.typography.titleMedium,
                    fontWeight = FontWeight.Bold
                )

                if (expense.note.isNotEmpty()) {
                    Text(
                        text = expense.note,
                        style =
MaterialTheme.typography.bodySmall,
                        color =
MaterialTheme.colorScheme.onSurfaceVariant,
                        maxLines = 1,
                        overflow = TextOverflow.Ellipsis
                    )
                }
            }
        }
    }
}

```

```

    }

    Text(
        text =
DateUtils.formatDateTime(expense.date),
        style = MaterialTheme.typography.bodySmall,
        color =
MaterialTheme.colorScheme.onSurfaceVariant
    )

    // Відображення рейтингу корисності
    if (expense.usefulnessRating > 0) {
        RatingStarsDisplay(
            rating = expense.usefulnessRating,
            modifier = Modifier.padding(top = 4.dp)
        )
    }
}

// Сума та кнопка видалення
Column(
    horizontalAlignment = Alignment.End
) {
    Text(
        text = String.format("%.2f ₾",
expense.amount),
        style = MaterialTheme.typography.titleLarge,
        fontWeight = FontWeight.Bold,
        color = MaterialTheme.colorScheme.primary
    )

    IconButton(
        onClick = { showDeleteDialog = true },
        modifier = Modifier.size(32.dp)
    ) {
        Icon(
            imageVector = Icons.Default.Delete,
            contentDescription = "Видалити",
            tint = MaterialTheme.colorScheme.error,
            modifier = Modifier.size(20.dp)
        )
    }
}

}

// Діалог підтвердження видалення
if (showDeleteDialog) {
    AlertDialog(
        onDismissRequest = { showDeleteDialog = false },
        title = {
            Text(text = "Видалити витрату?")
        }
    )
}

```

```

    },
    text = {
        Text(text = "Цю дію неможливо скасувати.")
    },
    confirmButton = {
        TextButton(
            onClick = {
                showDeleteDialog = false
                onDelete()
            }
        ) {
            Text(text = "Видалити")
        }
    },
    dismissButton = {
        TextButton(
            onClick = { showDeleteDialog = false }
        ) {
            Text(text = "Скасувати")
        }
    }
)
}
}
}

```