Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка фінансового менеджера на базі Android з використання Java технологій

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	_____ (підпис)	Коваль Д. В. (прізвище та ініціали)
Керівник	_____ (підпис)	Цуприк Г. Б. (прізвище та ініціали)
Нормоконтроль	_____ (підпис)	Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри	_____ (підпис)	Петрик М. Р. (прізвище та ініціали)
Рецензент	_____ (підпис)	_____ (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра, виконана Коваль Денисом Володимировичем, студентом групи СП-42 Тернопільського національного технічного університету імені Івана Пулюя, присвячена розробці фінансового менеджера на базі Android з використання Java технологій.

Обсяг роботи – 83 сторінок, 29 рисунків, 1 таблиця, 3 додатки, 30 джерел.

На початковому етапі було проведено аналіз предметної області та вимог до фінансового мобільного застосунку. На його основі сформовано основні функціональні модулі: облік транзакцій, категорій, цілей, перегляд статистики та конвертація валют.

Під час виконання роботи були проаналізовані існуючі рішення у сфері мобільних фінансових трекерів, визначені вимоги до системи, розроблена архітектура застосунку за шаблоном MVVM, реалізовано логіку взаємодії з локальною базою даних Room (на базі SQLite), застосовано компоненти ViewModel та LiveData для підтримки реактивного оновлення інтерфейсу.

Додаток розроблено в середовищі Android Studio з використанням мови програмування Java. Особливу увагу приділено модульності, структурованості коду та узгодженому дизайну інтерфейсу. Візуальна частина програми реалізована у вигляді взаємопов’язаних фрагментів із підтримкою адаптивної верстки. Тестування показало коректну роботу додатку в основних сценаріях користування, включно з додаванням і переглядом транзакцій, обробкою помилок введення, навігацією, а також збереженням даних між сесіями.

Ключові слова: Android, база даних, бюджет, категорії, Java, LiveData, мобільний додаток, MVVM, Room, SQLite, транзакція, фінанси.

ABSTRACT

Bachelor's qualification work completed by Koval Denys Volodymyrovych, a student of group SP-42 at Ternopil Ivan Puluj National Technical University, is dedicated to the development of a financial manager based on Android using Java technologies.

The thesis comprises 83 pages, 29 figures, 1 table, 3 appendices, and 30 sources.

At the initial stage, an analysis of the subject area and the requirements for a financial mobile application was conducted. Based on this, the main functional modules were formed: transaction tracking, categories, goals, statistics view, and currency conversion.

During the work, existing solutions in the field of mobile financial trackers were analyzed, system requirements were defined, the application architecture was developed based on the MVVM pattern, interaction logic with the local Room database (based on SQLite) was implemented, and ViewModel and LiveData components were applied to support reactive UI updates.

The application was developed in Android Studio using the Java programming language. Special attention was given to modularity, code structure, and consistent interface design. The visual part of the program was implemented as interrelated fragments with support for responsive layout. Testing showed correct application performance in core usage scenarios, including adding and viewing transactions, input error handling, navigation, and data retention between sessions.

Keywords: Android, database, budget, categories, Java, LiveData, mobile application, MVVM, Room, SQLite, transaction, finance.

ЗМІСТ

ВСТУП

Сучасний світ характеризується високим рівнем динаміки, зростанням обсягів фінансової інформації та необхідністю її постійного контролю. Ефективне управління особистими фінансами є важливою складовою фінансової грамотності, що має безпосередній вплив на добробут людини, її здатність планувати бюджет, досягати фінансових цілей та уникати заборгованості. У зв’язку з цим зростає попит на цифрові інструменти, що дозволяють користувачам контролювати свої витрати та доходи, планувати бюджет та аналізувати фінансову активність.

На ринку існує велика кількість фінансових застосунків, таких як Money Manager, Monefy, Wallet тощо. Вони пропонують широкий спектр функцій: синхронізацію з банками, створення звітів, планування бюджету, мультивалютність. Проте багато з них орієнтовані на просунутих користувачів або вимагають підключення до облікових записів банків, що викликає сумніви в безпеці або створює бар’єр входу. Також більшість популярних рішень є платними або мають обмеження у безкоштовній версії. У зв’язку з цим актуальним є створення простого, безпечного, автономного та безкоштовного застосунку, який задовольнить базові потреби користувача у веденні особистого бюджету.

Метою даної дипломної роботи є розробка Android-застосунку «Finance Manager», що дозволяє користувачеві здійснювати особистий фінансовий облік у зручній формі. Застосунок передбачає базову функціональність: додавання, редагування та видалення транзакцій; створення та управління категоріями доходів і витрат; перегляд списку транзакцій із можливістю фільтрації; базову аналітику; та конвертер валют. Реалізація ґрунтується на використанні архітектурного патерна MVVM (Model-View-ViewModel), що забезпечує розділення логіки представлення даних і взаємодії з користувачем, а також полегшує підтримку та масштабування застосунку. Технологічна основа

проєкту — мова програмування Java, SQLite для збереження даних, а також компоненти ViewModel та LiveData для реалізації архітектури, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних, без необхідності ручного втручання

Актуальність теми зумовлена також тим, що, попри наявність великої кількості фінансових застосунків, користувачі все частіше шукають прості та автономні рішення для повсякденного використання. Запропонований застосунок орієнтований на доступність, легкість у використанні, відсутність прив'язки до мережі або реєстрації, що робить його корисним для широкого кола користувачів — від студентів до пенсіонерів.

У процесі реалізації дипломного проєкту передбачається розв’язання таких завдань:

- Аналіз вимог до функціональності мобільного фінансового застосунку;
- Розробка архітектури застосунку відповідно до патерна MVVM;
- Реалізація інтерфейсу користувача та основного функціоналу (транзакції, категорії);
- Організація зберігання та обробки даних за допомогою SQLite;
- Проведення тестування застосунку та перевірка його на відповідність заданим вимогам.

Дипломна робота охоплює повний цикл створення програмного продукту: від аналізу потреб користувачів і проектування до реалізації, тестування та оцінки результатів. Таким чином, робота має не лише практичну цінність, а й дозволяє закріпити знання і навички, набуті під час навчання, зокрема в галузі об’єктно-орієнтованого програмування, розробки для мобільних платформ, використання шаблонів проєктування та роботи з базами даних. Отриманий застосунок може слугувати основою для подальшого розвитку, включаючи впровадження хмарної синхронізації, захисту даних, статистики у вигляді різних діаграм, нагадувань або підтримки декількох користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз вимог до системи

Першим етапом створення програмного забезпечення є аналіз вимог, що дозволяє визначити основні характеристики майбутньої системи. Від правильності цього етапу залежить успішність реалізації проекту, його зручність для користувачів, продуктивність та надійність [1].

Вимоги до системи поділяються на дві основні категорії: функціональні та нефункціональні.

1.1.1 Функціональні вимоги:

Аутентифікація користувача. Користувач повинен мати змогу створити обліковий запис та входити до системи за допомогою пароля.

Управління фінансами. Застосунок повинен дозволяти створення, редагування та видалення фінансових транзакцій із зазначенням типу (дохід/витрата), категорії, суми, дати та коментаря.

Категорії транзакцій. Повинна бути можливість використовувати вбудовані категорії (їжа, транспорт, зарплата тощо) та створювати власні.

Аналіз і статистика. Додаток має відображати графіки витрат і доходів за період (день, тиждень, місяць) та показувати поточний баланс.

Бюджетування. Користувач може встановити бюджет на категорію або загальний бюджет із повідомленням про перевищення.

Фінансові цілі. Можливість створювати цілі (наприклад, накопичення на покупку) з фіксацією прогресу та термінів.

Резервне копіювання та відновлення даних. Підтримка експорту/імпорту у форматах CSV або JSON для резервного копіювання локально або у хмару.

Конвертер валют. Реалізація можливості перерахунку сум за актуальними курсами валют з підтримкою кількох валют одночасно.

1.1.2 Нефункціональні вимоги

Цільова платформа. Android 8.0 (API 26) або новіше.

Технології розробки. Основна мова — Java; використання Android SDK, Android Studio, SQLite для збереження даних.

Архітектура застосунку. Застосування архітектурного підходу MVVM для розділення логіки інтерфейсу, бізнес-логіки та управління даними, що полегшить тестування та супровід.

Інтерфейс користувача. Інтерфейс має бути інтуїтивно зрозумілим, реалізованим згідно з принципами Material Design та адаптованим до різних розмірів екранів.

Безпека. Зберігання даних має відбуватись із використанням шифрування. Доступ до додатку має бути захищений.

Продуктивність. Додаток має працювати стабільно навіть при великій кількості транзакцій і не споживати надмірно ресурси пристрою.

Локалізація. Підтримка кількох мов інтерфейсу, зокрема української та англійської.

1.2 Опис методології проектування

Розробка мобільного застосунку потребує системного підходу, який охоплює всі етапи життєвого циклу програмного забезпечення: від збору вимог до підтримки готового продукту. З огляду на специфіку проєкту, масштаби та

індивідуальний характер розробки, доцільно обрати гнучку та водночас керовану методологію. Для цього використано ітеративну модель проєктування з елементами методології Agile [11].

1.2.1 Обґрунтування вибору ітеративного підходу

Ітеративна модель дозволяє реалізовувати систему поетапно, із поступовим додаванням функціональності та її перевіркою на кожному кроці. Цей підхід знижує ризик проєктних помилок, оскільки дає змогу виявляти і коригувати недоліки вже на ранніх етапах розробки. Крім того, він дозволяє швидко реагувати на зміну вимог або уточнення з боку замовника чи користувачів.

У нашому випадку, розробка фінансового менеджера передбачає поступову реалізацію таких ключових компонентів:

- Базова структура застосунку та UI (інтерфейс користувача);
- Модулі обліку витрат і доходів;
- Система категорій та фільтрації;
- Статистичні звіти та графіки;
- Модуль бюджету та фінансових цілей;
- Система резервного копіювання;
- Додаткові сервіси: список покупок, конвертер валют тощо.

Кожен з цих етапів розробляється, тестується та вдосконалюється окремо, після чого інтегрується в загальну систему.

1.2.2 Застосовані принципи Agile

Методологія Agile передбачає адаптивне планування, швидке розгортання функцій і постійний зворотний зв’язок. У дипломному проєкті ці принципи застосовуються в спрощеній формі:

- Інкрементна розробка: після кожного етапу користувач отримує нову функцію або вдосконалений модуль.
- Рефакторинг коду: на основі перевірки працездатності попередніх модулів виконується оптимізація коду та усунення виявлених недоліків.
- Фокус на користувачі: інтерфейс та логіка побудовані з орієнтацією на зручність і простоту взаємодії.

1.2.3 Архітектурний підхід

В якості архітектурної моделі обрано MVVM (Model-View-ViewModel) — сучасний підхід до побудови Android-застосунків, який забезпечує чітке розділення відповідальностей:

- Model: робота з базою даних, API та логікою обробки даних.
- View: користувацький інтерфейс, який відображає дані без логіки.
- ViewModel: посередник між Model і View, який забезпечує зв’язок даних і реагує на зміни стану.

Цей підхід підвищує масштабованість та тестованість застосунку, а також спрощує підтримку та розширення функціональності [13].

1.2.4 Інструменти проєктування

Під час розробки використовуються такі інструменти:

- UML-діаграми (Use Case, Class, Activity) — для опису структури та поведінки системи.
- Android Studio — як основне середовище розробки [15].
- Git — для контролю версій та збереження історії змін.

1.3 Обґрунтування вибору платформи Android та мови Java

У процесі створення мобільного застосунку для фінансового обліку важливим етапом є вибір цільової платформи та мови програмування, оскільки саме вони впливають на технічні характеристики продукту, потенційне охоплення користувачів, доступність інструментів розробки та загальну ефективність реалізації проєкту. У межах цієї роботи як основну платформу обрано Android, що забезпечує широке поширення серед користувачів мобільних пристроїв. Для реалізації програмної логіки застосунку використано мову програмування Java, яка є стабільним і підтримуваним інструментом для Android-розробки.

1.3.1 Обґрунтування вибору платформи Android

Android є найпоширенішою мобільною платформою у світі, що робить її логічним вибором для застосунків широкого призначення, зокрема — для особистих фінансових інструментів. Переваги цієї платформи:

- Широка аудиторія користувачів. Станом на 2025 рік частка Android на глобальному ринку мобільних операційних систем перевищує 70%. Це означає потенційно високу доступність та релевантність застосунку для користувачів.
- Відкритість екосистеми. Android базується на відкритому коді, що забезпечує гнучкість у налаштуванні функцій, доступ до системних API та можливість широкої кастомізації.
- Розвинена інфраструктура розробки. Android Studio надає потужне середовище розробки зі зручним інтерфейсом, інтеграцією інструментів відладки, емуляторів, профайлерів, а також автоматичним збірником (Gradle).
- Підтримка великої кількості пристроїв. Android працює на широкому спектрі пристроїв: від бюджетних до флагманських, що дозволяє охопити користувачів з різними технічними можливостями.

Таким чином, Android — це не лише технічно зручна, а й стратегічно доцільна платформа для проєкту, орієнтованого на щоденне використання та широку аудиторію.

1.3.2 Обґрунтування вибору мови Java

Для реалізації застосунку на платформі Android у якості основної мови програмування було обрано Java, незважаючи на наявність новіших альтернатив, таких як Kotlin. Це зумовлено наступними факторами:

- Стабільність і зрілість мови. Java є основною мовою Android-програмування з моменту заснування платформи. Вона має добре документовану базу знань, численні приклади реалізації та широку спільноту підтримки.

- Сумісність з Android SDK. Усі основні бібліотеки Android SDK написані на Java, що дозволяє уникнути труднощів зі сумісністю при використанні сторонніх компонентів.
- Наявність великої кількості навчальних матеріалів. Для Java існує велика кількість освітніх ресурсів, офіційних гайдлайнів та технічної документації, що прискорює розробку, особливо на етапі прототипування і тестування.
- Об’єктно-орієнтована парадигма. Java реалізує чітку об'єктно-орієнтовану модель, яка зручно поєднується з архітектурою MVVM, що використовується в даному проєкті.
- Переносимість і багатоплатформність. Хоча застосунок розробляється під Android, Java як мова залишається універсальною для багатьох середовищ. У разі потреби частину логіки можна повторно використати в інших JVM-сумісних проєктах.

1.4 Цільова аудиторія та сценарії використання додатку

Для створення ефективного та корисного мобільного застосунку важливо чітко визначити його цільову аудиторію, а також типові сценарії використання. Це дозволяє краще адаптувати функціонал, інтерфейс та логіку роботи додатку до реальних потреб користувачів.

У цьому розділі описуються основні групи користувачів, на яких орієнтований застосунок, та типові ситуації його використання — від щоденного обліку витрат до планування бюджету чи досягнення фінансових цілей.

1.4.1 Цільова аудиторія

Основною аудиторією розроблюваного мобільного додатку є користувачі, які мають потребу в особистому фінансовому плануванні та контролі витрат. До таких користувачів належать:

- Студенти та молодь. Особи без стабільного доходу, які прагнуть контролювати щоденні витрати, накопичення, заощадження або оптимізувати кишенькові кошти.
- Працюючі громадяни. Користувачі, які отримують щомісячну заробітну плату, оплачують рахунки, здійснюють регулярні покупки та бажають бачити загальну картину своїх витрат і доходів.
- Фрілансери та самозайняті особи. Люди, що мають нерегулярний дохід і потребують гнучкого інструменту для обліку прибутків і витрат, часто в різних валютах.
- Користувачі без досвіду фінансового обліку. Ті, хто вперше намагається впорядкувати свої фінанси, але не готовий використовувати складні або професійні програми.
- Особливу увагу приділено простоті інтерфейсу, інтуїтивному дизайну та легкості у користуванні, що дозволяє охопити як технічно підготовлених користувачів, так і тих, хто має обмежений досвід у користуванні мобільними застосунками.

1.4.2 Сценарії використання

Щоб сформулювати основний функціонал додатку, доцільно описати типові сценарії використання, які відображають очікувану поведінку користувача. Серед найбільш поширених сценаріїв:

Сценарій 1: Додавання нової витрати або доходу.

Користувач відкриває головний екран додатку, натискає кнопку додавання запису, вводить суму, категорію (наприклад, "їжа", "транспорт", "зарплата"), додає опис і зберігає операцію. Система автоматично оновлює баланс і відображає транзакцію у списку.

Сценарій 2: Перегляд статистики витрат.

Користувач бажає побачити аналітику за останній тиждень або місяць. Він відкриває розділ "Статистика", де дані подані у вигляді графіка або діаграми за категоріями витрат.

Сценарій 3: Створення списку покупок

Перед походом у магазин користувач створює список необхідних товарів, додає їх до списку в додатку та позначає виконані пункти у процесі покупки.

Сценарій 4: Використання конвертера валют

Користувач отримав дохід у валюті або планує подорож, тому переходить до розділу "Конвертер валют", де може перерахувати суму за поточним курсом.

Сценарій 5: Отримання нагадування про регулярну оплату

Користувач встановлює нагадування для оплати абонплати за інтернет або комунальні послуги. У визначений час додаток сповіщає його про майбутній платіж.

Кожен з описаних сценаріїв враховує потреби у швидкому доступі до основних функцій, мінімізацію кількості дій, необхідних для виконання задачі, а також підтримку зрозумілої навігації та зручного інтерфейсу.

1.5 Вибір інструментів і середовища розробки

Успішна реалізація мобільного застосунку значною мірою залежить від правильно обраного набору інструментів та середовища розробки. Вибір повинен

відповідати вимогам проєкту, забезпечувати зручність у розробці, підтримку сучасних технологій, а також відповідати специфіці цільової платформи.

Під час розробки мобільного фінансового застосунку для платформи Android були проаналізовані наявні інструменти програмування, середовища розробки, мови програмування та бібліотеки, які могли б забезпечити ефективну реалізацію функціоналу, зокрема обліку транзакцій, роботи з категоріями, бюджету, конвертера валют та аналітики. Особливу увагу приділено підтримці архітектурних шаблонів, зокрема MVVM, а також можливості масштабування і супроводу застосунку.

У цьому підрозділі розглядаються основні технологічні рішення, які були обрані для реалізації проєкту, з обґрунтуванням їх доцільності та переваг.

1.5.1 Середовище розробки: Android Studio

Android Studio є офіційним інтегрованим середовищем розробки (IDE) для створення додатків під операційну систему Android. Його використання зумовлене рядом переваг:

- Повна підтримка Android SDK та необхідних бібліотек.
- Інтегровані інструменти для створення графічного інтерфейсу (Layout Editor).
- Вбудований емулятор пристроїв, що дозволяє тестувати додаток без фізичного смартфона.
- Засоби для налагодження, аналізу продуктивності та відстеження помилок.
- Можливість використання системи контролю версій Git.
- Android Studio також забезпечує тісну інтеграцію з мовою програмування Java, що робить його ідеальним вибором для даного проєкту.

1.5.2 Додаткові інструменти та технології

Окрім основного середовища, у процесі розробки передбачається використання таких інструментів:

- XML – мова розмітки, яка використовується для опису інтерфейсу користувача в Android-додатках.
- SQLite – вбудована база даних, що використовується для локального зберігання даних, зокрема фінансових записів користувача.

1.5.3 Система контролю версій

Для збереження історії змін, командної роботи та відновлення попередніх версій проєкту використовується Git. Хостингом репозиторію обрано GitHub, що дозволяє зручно управляти версіями, створювати резервні копії та працювати над різними гілками проєкту.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проектування відношень між акторами і прецедентами

На цьому етапі розробки визначаються основні користувачі системи та їх взаємодія з основними функціями програмного забезпечення. Для цього створюється діаграма варіантів використання, яка допомагає наочно представити, які дії може виконувати користувач і як саме працює система. Такий підхід дозволяє краще зрозуміти вимоги до функціональності застосунку та спланувати його реалізацію [2].

2.1.1 Основні актори та їхні сценарії взаємодії із системою

Процес проектування системи починається з визначення основних учасників взаємодії з додатком (акторів) і ключових сценаріїв їхньої поведінки (прецедентів). Це дозволяє зрозуміти функціональні вимоги до системи і правильно організувати її архітектуру. У межах фінансового менеджера на базі Android з використанням Java-технологій виділено чотири основних актора:

- Користувач — основний актор, який взаємодіє з додатком для ведення особистих фінансів. Він має доступ до більшості функцій системи.
- Адміністратор — спеціальний технічний актор, який не взаємодіє безпосередньо з інтерфейсом додатку, але відповідає за оновлення, технічну підтримку і підтримання актуальності довідкових даних.
- Мобільний додаток — умовний посередник між користувачем і технічною частиною системи, який реалізує бізнес-логіку та забезпечує виконання запитів користувача.
- База даних — внутрішній компонент, що виконує зберігання і обробку фінансової інформації, налаштувань, категорій витрат і доходів.

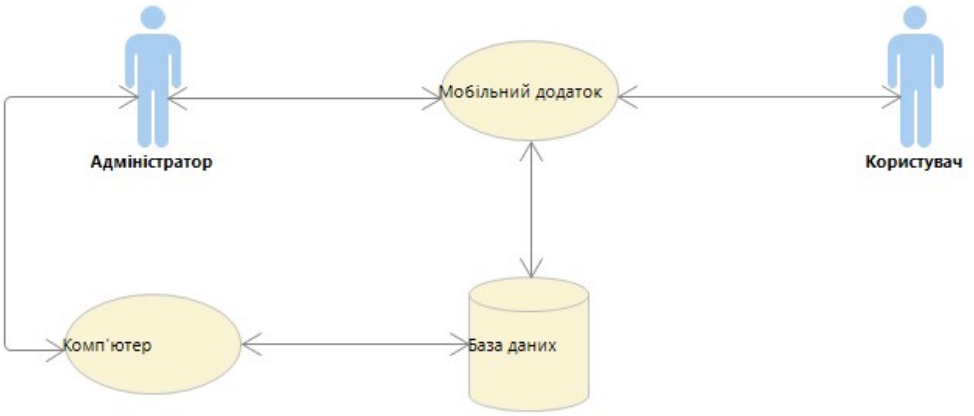


Рисунок 2.1 - Типовий сценарій застосування

Для кожного з акторів можна виділити відповідні варіанти використання:

Користувач:

- додавання, редагування та видалення записів про доходи та витрати;
- створення та налаштування власних категорій;
- перегляд графіків і діаграм для аналізу фінансової активності;
- перегляд щоденного фінансового підсумку;
- перевірка залишку перед здійсненням нової витрати;
- встановлення ліміту витрат на місяць;
- налаштування параметрів додатку (валюта, мова, тема, початковий баланс).

Адміністратор:

- оновлення структури бази даних (наприклад, додавання нових системних категорій);
- публікація нових версій додатку;
- моніторинг стабільності роботи програми та виправлення помилок.
- Мобільний додаток (виконує автоматизовані функції):
- обробка введених користувачем даних і передача їх у базу;
- виведення даних з бази у вигляді звітів, графіків, таблиць;

- надсилання сповіщень про перевищення або наближення до встановленого ліміту витрат;
- автоматичне оновлення інтерфейсу після змін у даних.
- База даних:
- збереження фінансових операцій, категорій та інших налаштувань;
- обробка запитів на зчитування, оновлення та видалення даних;
- забезпечення цілісності та узгодженості інформації.

2.1.2 Типові сценарії використання мобільного додатку

Для кращого розуміння логіки роботи мобільного фінансового менеджера доцільно розглянути кілька типових сценаріїв використання, що відображають реальні потреби користувачів.

Сценарій 1: Додавання нової витрати. Користувач відкриває додаток і натискає кнопку «Додати витрату». З’являється форма, де потрібно ввести суму, вибрати категорію (наприклад, «Продукти»), дату (за замовчуванням — поточна), а також додати коментар (наприклад, «Купівля овочів»). Після підтвердження дані зберігаються в базі, інтерфейс оновлює загальний баланс і, за потреби, виводить повідомлення про наближення до встановленого бюджетного ліміту. Усі зміни одразу відображаються на головному екрані та у звітах.

Сценарій 2: Створення нової категорії. Користувач бажає додати власну категорію витрат — наприклад, «Домашні тварини». У меню «Категорії» він обирає опцію «Додати нову», вводить назву, іконку та, за потреби, колір. Додаток зберігає цю категорію в базу даних, і вона одразу стає доступною у формі додавання витрати або доходу. Таким чином реалізується персоналізація системи під індивідуальні потреби.

Сценарій 3: Зміна валюти обліку. У налаштуваннях користувач обирає іншу валюту (наприклад, з гривні на долар США). Система автоматично перераховує

всі збережені суми за поточним курсом або відображає старі дані з позначкою валюти.

Сценарій 4: Встановлення ліміту витрат на місяць. Користувач заходить у меню «Бюджет» і встановлює місячний ліміт витрат — наприклад, 10 000 гривень. Додаток починає автоматично відстежувати сукупні витрати за поточний місяць і порівнює їх із заданим лімітом. У разі перевищення або наближення до ліміту система надсилає сповіщення, наприклад: «Ви використали 90% вашого бюджету на квітень».

Сценарій 5: Редагування або видалення запису про витрату. Після внесення витрати користувач помічає, що допустив помилку в сумі або категорії. Він відкриває історію транзакцій, знаходить потрібний запис і натискає на нього. З'являється меню з опціями «Редагувати» або «Видалити». Після редагування нові дані зберігаються до бази, а інтерфейс оновлює відповідну інформацію в звітах. Це забезпечує гнучкість і дозволяє уникати накопичення хибної статистики.

Сценарій 6: Перегляд щоденного фінансового підсумку. Щовечора користувач відкриває додаток, щоб переглянути короткий підсумок витрат і доходів за день. На головному екрані або у вигляді спливаючого вікна відображаються загальна сума витрат, кількість транзакцій і залишок бюджету. Це дозволяє швидко оцінити фінансовий стан без глибокого аналізу. За бажанням користувач може натиснути на підсумок і перейти до повного звіту.

Сценарій 7: Перевірка балансу перед витратою. Перш ніж зробити покупку, користувач відкриває додаток і перевіряє залишок у конкретному гаманці. На головному екрані відображається поточний баланс усіх рахунків, а також сума витрат за день або тиждень. Це дозволяє швидко оцінити, чи варто здійснювати нову покупку, не порушуючи фінансової дисципліни.

Сценарій 8: Перегляд фінансової аналітики. Користувач відкриває розділ «Аналітика» в додатку. Тут він переглядає графіки витрат за категоріями, динаміку доходів і витрат за місяцями, а також таблицю з підсумками. За бажанням, користувач може змінити фільтр (наприклад, період або тип операцій) і

побачити оновлену інформацію. Це допомагає оцінити фінансову поведінку та прийняти обґрунтовані рішення.

Сценарій 9: Керування системними категоріями. Адміністратор заходить у панель управління та відкриває розділ «Категорії». Він додає нову системну категорію (наприклад, «Освіта») або редагує існуючі. Зміни зберігаються до бази даних і стають доступними всім користувачам.

Сценарій 10: Налаштування системи та оновлення. Адміністратор розгортає нову версію додатку через системну панель. Він також може змінити загальні налаштування (наприклад, встановити стандартну валюту або мову інтерфейсу). Після збереження змін додаток застосовує нові параметри для всіх користувачів.

Для кращого відображення логіки взаємодії користувачів із функціональністю мобільного застосунку було побудовано діаграму варіантів використання (див. рис. 2.2).

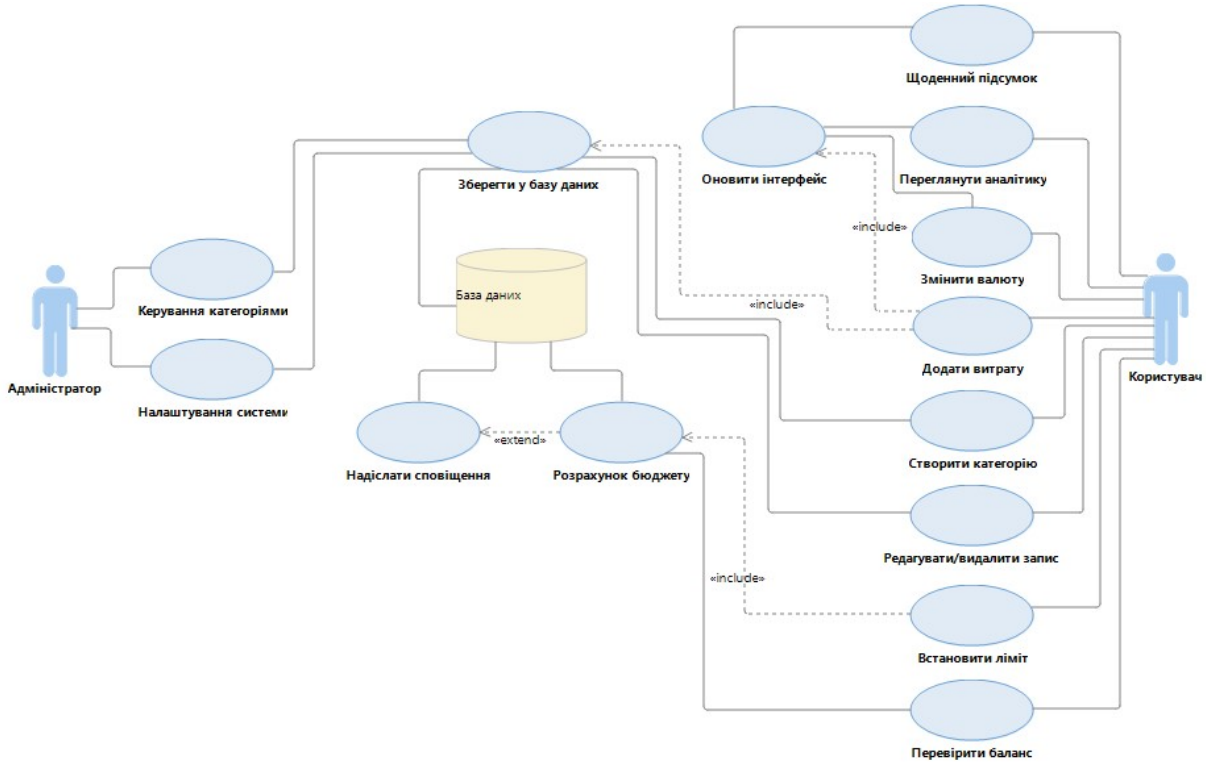


Рисунок 2.2 – Діаграма варіантів використання

Діаграма графічно ілюструє основні сценарії роботи системи, відображаючи взаємозв’язки між суб’єктами та відповідними функціональними можливостями.

2.2 Визначення класів системи

На основі діаграми варіантів використання формується уявлення про основні функції, які повинна виконувати система. Наступним кроком є визначення класів — основних структурних елементів програмного забезпечення, які відображають об’єкти предметної області та логіку взаємодії між ними. Кожен клас описує певну сутність системи, її властивості (поля) та поведінку (методи).

У контексті мобільного застосунку Finance Manager було виділено класи, що відповідають за представлення транзакцій, категорій, цілей, бюджетів, користувацьких даних та інші елементи. Окремі класи також реалізують функціональність взаємодії з базою даних (через Room), обробку подій (ViewModel), а також відображення інформації у користувацькому інтерфейсі.

Визначення класів є важливою частиною проєктування, оскільки дозволяє побудувати логічно узгоджену архітектуру застосунку, розподілити відповідальність між компонентами та забезпечити гнучкість і масштабованість системи під час її подальшого розвитку [5].

В результаті моделювання були визначені наступні класи, що становлять основу системи:

1) MainActivity – головна активність додатку, яка ініціалізує інтерфейс і навігацію між фрагментами.

Методи:

- onCreate() – викликається при створенні активності, налаштовує компоненти та фрагменти.

2) TransactionListFragment – фрагмент, який відображає список транзакцій користувача.

Атрибути:

- viewModel – ViewModel, який постачає список транзакцій.
- adapter – адаптер для прив’язки списку транзакцій до RecyclerView.

Методи:

- onCreateView() – створює та повертає вигляд для фрагмента.

3) TransactionAdapter – адаптер для RecyclerView, що відображає транзакції.

Атрибути:

- transactions – список транзакцій для відображення.

Методи:

- onCreateViewHolder() – створює новий ViewHolder для елемента списку.
- onBindViewHolder() – заповнює даними ViewHolder на заданій позиції.
- getItemCount() – повертає загальну кількість елементів у списку.

4) TransactionViewHolder – контейнер для елементів інтерфейсу однієї транзакції в списку.

Методи:

- TransactionViewHolder() – лише зберігає посилання на елементи інтерфейсу.

5) HomeFragment – фрагмент, який може відображати підсумки фінансів або дашборд.

Атрибути:

- viewModel – ViewModel, який надає дані для головного екрану.

Методи:

- onCreateView() – створює інтерфейс головного фрагмента.

6) AddTransactionActivity – активність, яка містить інтерфейс для додавання нової транзакції.

Методи:

- onCreate() – створює активність і завантажує фрагмент додавання транзакції.

7) AddTransactionFragment – фрагмент, який забезпечує введення даних транзакції користувачем.

Атрибути:

- viewModel – ViewModel, який керує логікою додавання транзакцій.

Методи:

- onCreateView() – створює вигляд фрагмента.
- getCategoryNames() – отримує список назв категорій для випадаючого списку.
- clearFields() – очищає поля введення після збереження транзакції.

8) CategoryListFragment – фрагмент для перегляду або керування категоріями витрат.

Атрибути:

- viewModel – ViewModel для доступу до списку категорій.

Методи:

- onCreateView() – створює інтерфейс для роботи з категоріями.

9) TransactionViewModel – ViewModel, який керує списком транзакцій і взаємодією з базою.

Методи:

- getTransactions() – повертає список транзакцій у вигляді LiveData.
- loadTransactions() – завантажує транзакції з бази даних.
- addTransaction() – додає нову транзакцію до бази.

10) CategoryViewModel – ViewModel, який керує категоріями та оновлює інтерфейс при змінах.

Методи:

- getCategoryList() – повертає список категорій у вигляді LiveData.
- addCategory() – додає нову категорію до бази.
- loadCategories() – завантажує категорії з бази.

11) TransactionDao – інтерфейс для доступу до таблиці транзакцій у базі даних.

Методи:

- getTransactions() – повертає всі транзакції з бази.
- insertTransaction() – вставляє нову транзакцію до таблиці.

12) CategoryDao – інтерфейс для доступу до таблиці категорій.

Методи:

- insertCategory() – вставляє нову категорію до таблиці.
- getCategories() – повертає всі наявні категорії.

13) Transaction – модель даних, яка представляє окрему фінансову транзакцію.

Атрибути:

- id – унікальний ідентифікатор транзакції.
- type – тип транзакції (наприклад, дохід або витрата).
- category – категорія, до якої належить транзакція.
- amount – сума транзакції.
- date – дата здійснення транзакції.
- note – додаткова примітка.
- transactionId – можливо, додатковий ідентифікатор або зовнішній ключ.

Методи:

- getId(), setId() – доступ до ідентифікатора.
- getType(), setType() – доступ до типу транзакції.
- getCategory(), setCategory() – доступ до категорії.
- getAmount(), setAmount() – доступ до суми.
- getDate(), setDate() – доступ до дати.
- getNote(), setNote() – доступ до примітки.
- setCatId() – встановлює категорію за id.

14) Category – модель даних, яка представляє категорію витрат або доходів.

Атрибути:

- id – унікальний ідентифікатор категорії.
- name – назва категорії.

Методи:

- getId(), setId() – доступ до ідентифікатора.
- getName(), setName() – доступ до назви.

15) TransactionDbHelper – клас, що керує створенням і оновленням бази даних.

Атрибути:

- DATABASE_NAME – назва бази даних.
- DATABASE_VERSION – версія бази.
- TABLE_TRANSACTIONS – назва таблиці транзакцій.

Методи:

- TransactionDbHelper() – конструктор для ініціалізації.
- onCreate() – створює структуру бази при першому запуску.
- onUpgrade() – оновлює структуру бази при зміні версії.

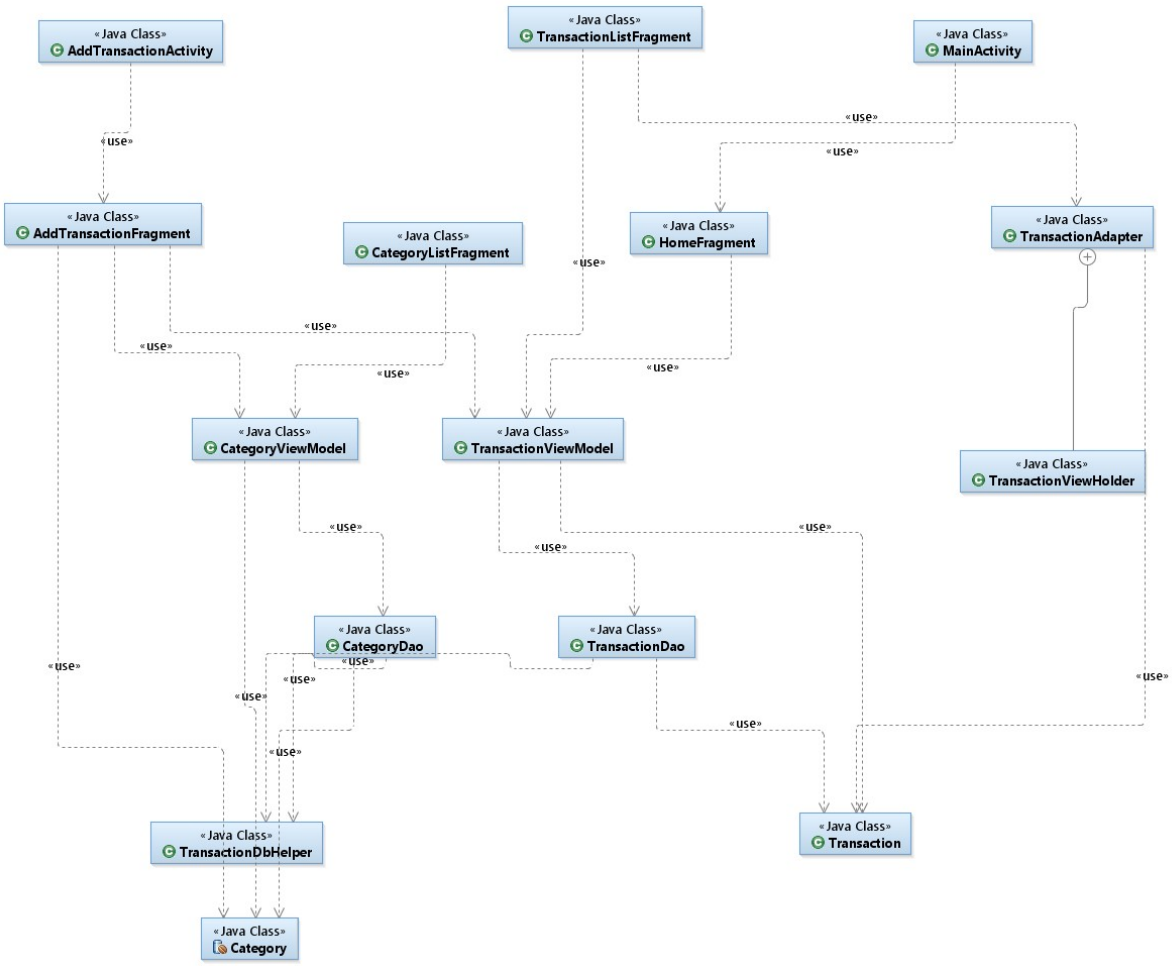


Рисунок 2.3 – Діаграма класів

Визначення класів мобільного застосунку Finance Manager стало результатом моделювання, що враховує функціональність системи та її компоненти. Класи охоплюють логіку і взаємодію з базою, забезпечуючи узгоджену архітектуру.

2.3 Реалізація програмних компонентів

На цьому етапі було здійснено реалізацію ключових програмних компонентів мобільного застосунку Finance Manager для платформи Android. Архітектура застосунку побудована за шаблоном MVVM (Model-View-ViewModel) з використанням таких технологій, як Java, Room (SQLite) для збереження даних, а також ViewModel для розділення логіки і представлення. У цьому розділі розглядається реалізація основних функціональних модулів, включно з додаванням і переглядом транзакцій, керуванням категоріями, реалізацією домашньої сторінки, конвертера валют та інших важливих компонентів застосунку.

Клас AddTransactionActivity відповідає за ініціалізацію інтерфейсу для додавання нової транзакції у мобільному застосунку Finance Manager. Цей компонент є активністю, що відображає відповідний фрагмент AddTransactionFragment, де реалізується основна логіка введення фінансових даних користувачем.

У методі onCreate() здійснюється встановлення макету activity_add_transaction.xml, у якому передбачено контейнер для фрагменту (fragment_container). Якщо активність створюється вперше (тобто savedInstanceState == null), фрагмент AddTransactionFragment динамічно додається до інтерфейсу через транзакцію з використанням FragmentManager.

Завдяки такому підходу реалізується гнучка структура інтерфейсу з можливістю повторного використання фрагментів, що відповідає принципам модульності та повторного використання коду в архітектурі MVVM.

```java
package com.financemanager;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

public class AddTransactionActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add_transaction);

        if (savedInstanceState == null) {
            // Додаємо фрагмент до activity
            FragmentManager fragmentManager = getSupportFragmentManager();
            FragmentTransaction transaction = fragmentManager.beginTransaction();
            transaction.replace(R.id.fragment_container, new AddTransactionFragment());
            transaction.commit();
        }
    }
}
```

Рисунок 2.4 – Код класу AddTransactionActivity

Клас AddTransactionDialogFragment реалізує діалог для додавання нової транзакції. Він наслідує DialogFragment і створює інтерфейс з полями для введення суми, нотатки та вибору категорії.

Діалог побудовано за допомогою AlertDialog, де кнопка «Зберегти» обробляється окремо для валідації даних. При підтвердженні введення створюється об’єкт Transaction, який передається до TransactionViewModel для збереження у базі даних.

У разі помилки введення (наприклад, порожнє поле суми) виводиться відповідне повідомлення. Такий підхід дозволяє швидко додавати транзакції без переходу на окрему активність, що робить інтерфейс зручним і інтерактивним.

```
24    public Dialog onCreateDialog(@Nullable Bundle savedInstanceState) {
25        View view = LayoutInflater.from(getContext()).inflate(R.layout.dialog_add_transaction,  root: null);
26
27        amountEditText = view.findViewById(R.id.editTextAmount);
28        noteEditText = view.findViewById(R.id.editTextNote);
29        categorySpinner = view.findViewById(R.id.spinnerCategory);
30
31
32        AlertDialog dialog = new AlertDialog.Builder(requireContext())
33                .setTitle("Нова транзакція")
34                .setView(view)
35                .setPositiveButton( text: "Зберегти",  listener: null)  // null щоб перехопити пізніше
36                .setNegativeButton( text: "Скасувати", ( DialogInterface d,  int which) -> dismiss())
37                .create();
38
39        dialog.setOnShowListener( DialogInterface d -> {
40            dialog.getButton(AlertDialog.BUTTON_POSITIVE).setOnClickListener( View v -> {
41                try {
42                    String type = "expense";
43                    String category = categorySpinner.getSelectedItem().toString();
44                    double amount = Double.parseDouble(amountEditText.getText().toString().trim());
45                    String note = noteEditText.getText().toString().trim();
46
47                    Transaction transaction = new Transaction( id: 0, type, category, amount,  date: "", note);
48                    TransactionViewModel viewModel = new ViewModelProvider(requireActivity()).get(TransactionViewModel.class);
49                    viewModel.addTransaction(requireContext(), transaction);
50
51                    Toast.makeText(getContext(),  text: "Транзакцію збережено", Toast.LENGTH_SHORT).show();
52                    dismiss();
53                } catch (Exception e) {
54                    Toast.makeText(getContext(),  text: "Помилка введення. Перевірте поля.", Toast.LENGTH_SHORT).show();
```

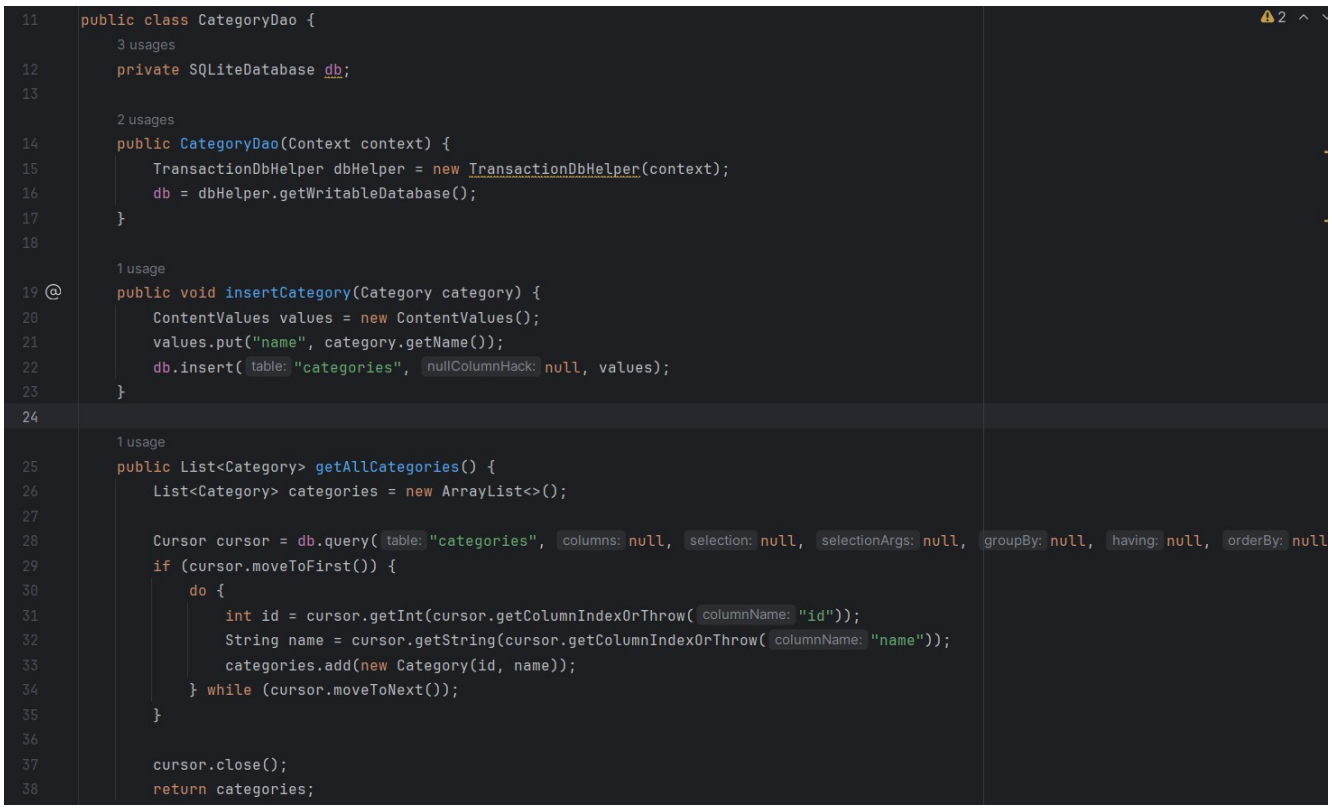
Рисунок 2.5 – Код класу AddTransactionDialogFragment

Клас Category є сутністю Room-бази даних, яка відповідає за зберігання інформації про категорії транзакцій. Він позначений анотацією @Entity, що означає, що об’єкти цього класу зберігаються в таблиці categories. Поле id є первинним ключем і автоматично генерується. Поле name зберігає назву категорії. Конструктор і методи доступу (get та set) забезпечують взаємодію з полями при читанні та записі даних у базу. Клас використовується для побудови списку категорій у додатку.

```java
package com.financemanager;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

9 usages
@Entity(tableName = "categories")
public class Category {

    3 usages
    @PrimaryKey(autoGenerate = true)
    private int id;

    3 usages
    private String name;

    1 usage
    public Category(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    2 usages
    public String getName() { return name; }
    no usages
    public void setName(String name) { this.name = name; }
}
```

Рисунок 2.6 – Код класу Category

Клас CategoryDao відповідає за доступ до даних категорій у локальній SQLite-базі. У конструкторі створюється екземпляр допоміжного класу TransactionDbHelper, після чого відкривається база для запису. Метод insertCategory додає нову категорію до таблиці categories, формуючи об’єкт ContentValues з назвою. Метод getAllCategories читає всі записи з таблиці, створюючи об’єкти Category з отриманих даних курсора. Таким чином, клас дозволяє працювати з категоріями на базовому рівні з використанням SQLite.



```
public class CategoryDao {
    3 usages
    private SQLiteDatabase db;

    2 usages
    public CategoryDao(Context context) {
        TransactionDbHelper dbHelper = new TransactionDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    1 usage
    public void insertCategory(Category category) {
        ContentValues values = new ContentValues();
        values.put("name", category.getName());
        db.insert( table: "categories",  nullColumnHack: null, values);
    }

    1 usage
    public List<Category> getAllCategories() {
        List<Category> categories = new ArrayList<>();

        Cursor cursor = db.query( table: "categories",  columns: null,  selection: null,  selectionArgs: null,  groupBy: null,  having: null,  orderBy: null
        if (cursor.moveToFirst()) {
            do {
                int id = cursor.getInt(cursor.getColumnIndexOrThrow( columnName: "id"));
                String name = cursor.getString(cursor.getColumnIndexOrThrow( columnName: "name"));
                categories.add(new Category(id, name));
            } while (cursor.moveToNext());
        }

        cursor.close();
        return categories;
```

Рисунок 2.7 – Код класу CategoryDao

Фрагмент CategoryListFragment відповідає за відображення списку категорій. У методі onCreateView відбувається прив’язка до відповідного XML-макету, після чого ініціалізується CategoryViewModel через ViewModelProvider. Далі встановлюється спостерігач на LiveData зі списком категорій — коли дані змінюються, тут можна оновити інтерфейс, наприклад, через RecyclerView. Метод loadCategories запускає завантаження категорій із бази, передаючи для цього контекст. Таким чином, фрагмент взаємодіє з ViewModel для динамічного оновлення UI на основі змін у даних.

```java
package com.financemanager;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;

public class CategoryListFragment extends Fragment {

    3 usages
    private CategoryViewModel viewModel;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_category_list, container, attachToRoot: false);
        viewModel = new ViewModelProvider( owner: this).get(CategoryViewModel.class);

        viewModel.getCategoryList().observe(getViewLifecycleOwner(), List<Category> categories -> {
            // Тут можна оновити RecyclerView для категорій
        });

        viewModel.loadCategories(requireContext());

        return view;
    }
}
```

Рисунок 2.8 – Код класу CategoryListFragment

Клас CategoryViewModel відповідає за зберігання та управління даними категорій у рамках архітектури MVVM. Він містить MutableLiveData, яке зберігає список категорій і надається зовні як LiveData для спостереження у фрагментах або активностях.

Метод loadCategories завантажує всі категорії з бази даних через CategoryDao і оновлює LiveData, щоб інтерфейс отримав нові дані. Метод addCategory додає нову категорію до бази, а потім знову викликає loadCategories, щоб актуалізувати список. Таким чином, ViewModel ізолює логіку роботи з базою та забезпечує швидке оновлення інтерфейсу.

```java
package com.financemanager;

import ...

4 usages
public class CategoryViewModel extends ViewModel {
    2 usages
    private final MutableLiveData<List<Category>> categoryList = new MutableLiveData<>();

    2 usages
    public LiveData<List<Category>> getCategoryList() { return categoryList; }

    3 usages
    public void loadCategories(Context context) {
        CategoryDao dao = new CategoryDao(context);
        categoryList.setValue(dao.getAllCategories());
    }

    no usages
    public void addCategory(Context context, Category category) {
        CategoryDao dao = new CategoryDao(context);
        dao.insertCategory(category);
        loadCategories(context);
    }
}
```

Рисунок 2.9 – Код класу CategoryViewModel

Клас HomeFragment відповідає за головний екран додатку, де відображається поточний баланс користувача, сума доходів і витрат. Він використовує TransactionViewModel для завантаження та спостереження за списком транзакцій, на основі яких розраховуються й оновлюються значення балансу, доходів і витрат у відповідних текстових полях. Також фрагмент формує графічне представлення фінансової активності протягом тижня у вигляді горизонтальних прогрес-барів. Кнопка додавання транзакції ініціалізується у методі onViewCreated та очікує реалізації переходу до відповідного екрану.

```java
public class HomeFragment extends Fragment {

    2 usages
    private TextView textBalance, textIncome, textExpenses;
    2 usages
    private FloatingActionButton buttonAddTransaction;
    3 usages
    private TransactionViewModel transactionViewModel;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_home, container, attachToRoot: false);

        // Ініціалізація текстових полів
        textBalance = view.findViewById(R.id.textBalance);
        textIncome = view.findViewById(R.id.textIncome);
        textExpenses = view.findViewById(R.id.textExpenses);

        // ViewModel для транзакцій
        transactionViewModel = new ViewModelProvider( owner: this).get(TransactionViewModel.class);
        transactionViewModel.loadTransactions(requireContext());

        // Спостерігач для оновлення даних балансу
        transactionViewModel.getTransactionList().observe(getViewLifecycleOwner(), List<Transaction> transactions -> {
            double balance = 0.0;
            double income = 0.0;
            double expense = 0.0;

            for (Transaction t : transactions) {
                if (t.getType().equalsIgnoreCase( anotherString: "income")) {
                    income += t.getAmount();
                    balance += t.getAmount();
```

Рисунок 2.10 – Код класу HomeFragment

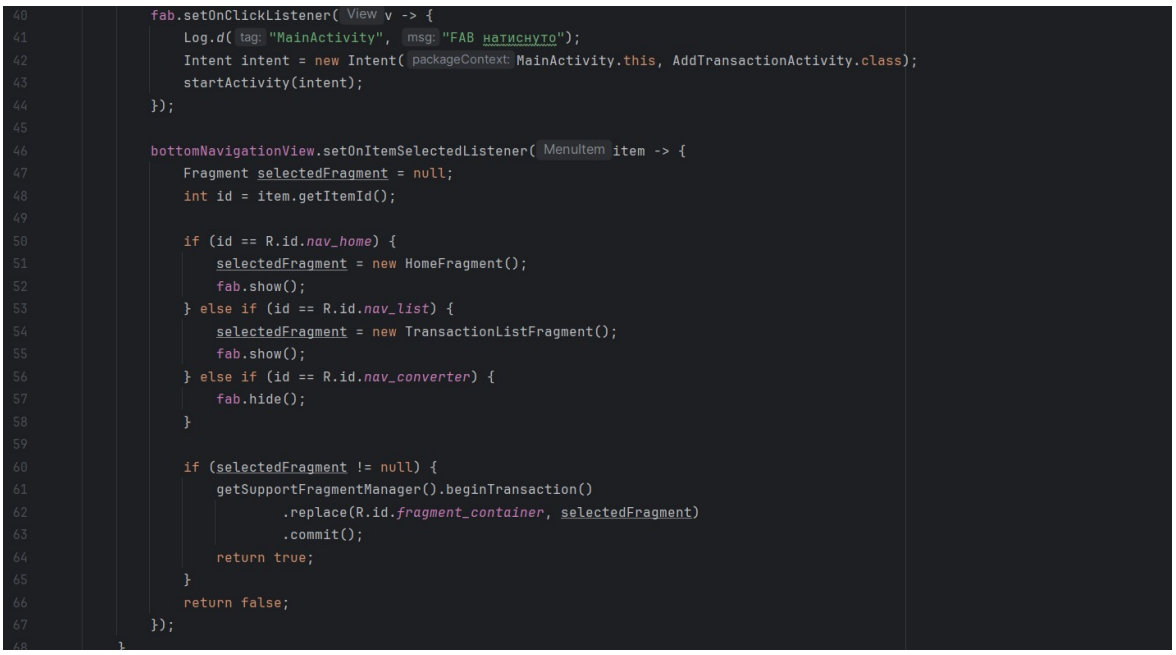
```java
            textBalance.setText(String.format("Поточний баланс: ₴%.2f", balance));
            textIncome.setText(String.format("Загальні доходи: ₴%.2f", income));
            textExpenses.setText(String.format("Загальні витрати: ₴%.2f", expense));
        });

        LinearLayout chartContainer = view.findViewById(R.id.chartContainer);

        String[] days = {"Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Нд"};

        for (String day : days) {

            LinearLayout row = new LinearLayout(getContext());
            row.setOrientation(LinearLayout.HORIZONTAL);
            row.setLayoutParams(new LinearLayout.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.LayoutParams.WRAP_CONTENT));
            row.setPadding( left: 0,  top: 8,  right: 0,  bottom: 8);

            TextView dayLabel = new TextView(getContext());
            dayLabel.setText(day);
            dayLabel.setTextSize(14);
            dayLabel.setWidth(80);

            ProgressBar bar = new ProgressBar(getContext(),  attrs: null, android.R.attr.progressBarStyleHorizontal);
            bar.setMax(500);
            bar.setProgress(amount);
            bar.setLayoutParams(new LinearLayout.LayoutParams( width: 0,  height: 30,  weight: 1f));
            bar.setProgressDrawable(ContextCompat.getDrawable(getContext(), R.drawable.custom_progressbar));

            TextView amountLabel = new TextView(getContext());
```

Рисунок 2.11 – Код класу HomeFragment (частина друга)

Клас MainActivity є головною активністю застосунку, яка керує нижньою навігацією (BottomNavigationView) та плаваючою кнопкою (FloatingActionButton). Під час запуску, якщо savedInstanceState дорівнює null, в контейнер фрагментів підвантажується HomeFragment як екран за замовчуванням, і кнопка fab відображається. Для кнопки fab встановлено слухача подій, який створює і запускає нову активність для додавання транзакції — AddTransactionActivity. У навігаційному меню передбачено перемикання між фрагментами: головним (HomeFragment), списком транзакцій (TransactionListFragment) і конвертером валют (без заміни фрагмента, лише приховування кнопки). У кожному випадку відбувається оновлення вмісту головного контейнера фрагментів.

```
public class MainActivity extends AppCompatActivity {

    2 usages
    private BottomNavigationView bottomNavigationView;
    7 usages
    private FloatingActionButton fab;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        bottomNavigationView = findViewById(R.id.bottom_navigation);
        fab = findViewById(R.id.buttonAddTransaction);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, new HomeFragment())
                    .commit();
            fab.show();
        }


        fab.setOnClickListener( View v -> {
            Log.d( tag: "MainActivity",  msg: "FAB натиснуто");
            new AddTransactionDialogFragment().show(
                    getSupportFragmentManager(),  tag: "AddTransactionDialog"
            );
        });

        fab.setOnClickListener( View v -> {
            Log.d( tag: "MainActivity",  msg: "FAB натиснуто");
```

Рисунок 2.12 – Код класу MainActivity



```
        fab.setOnClickListener( View v -> {
            Log.d( tag: "MainActivity",  msg: "FAB натиснуто");
            Intent intent = new Intent( packageContext: MainActivity.this, AddTransactionActivity.class);
            startActivity(intent);
        });

        bottomNavigationView.setOnItemSelectedListener( MenuItem item -> {
            Fragment selectedFragment = null;
            int id = item.getItemId();

            if (id == R.id.nav_home) {
                selectedFragment = new HomeFragment();
                fab.show();
            } else if (id == R.id.nav_list) {
                selectedFragment = new TransactionListFragment();
                fab.show();
            } else if (id == R.id.nav_converter) {
                fab.hide();
            }

            if (selectedFragment != null) {
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.fragment_container, selectedFragment)
                        .commit();
                return true;
            }
            return false;
        });
    }
```

Рисунок 2.13 – Код класу MainActivity (частина друга)

Клас Transaction представляє модель фінансової транзакції. Він містить основні поля, які описують одну операцію: ідентифікатор, тип (наприклад, витрата чи дохід), назву категорії, суму, дату і примітку. Конструктор ініціалізує ці значення при створенні об'єкта. Для кожного поля надані стандартні гетери і сетери, що дозволяє зчитувати і змінювати дані транзакції у межах програми.

```
    public Transaction(int id, String type, String category, double amount, String date, String note) {
        this.id = id;
        this.type = type;
        this.category = category;
        this.amount = amount;
        this.date = date;
        this.note = note;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    4 usages
    public String getType() { return type; }
    no usages
    public void setType(String type) { this.type = type; }

    3 usages
    public String getCategory() { return category; }
    no usages
    public void setCategory(String category) { this.category = category; }

    7 usages
    public double getAmount() { return amount; }
    no usages
    public void setAmount(double amount) { this.amount = amount; }

    2 usages
    public String getDate() { return date; }
    no usages
    public void setDate(String date) { this.date = date; }
```

Рисунок 2.14 – Код класу Transaction

Клас TransactionAdapter реалізує адаптер для компонента RecyclerView, який забезпечує відображення списку транзакцій у вигляді прокручуваного переліку. Він містить внутрішній клас TransactionViewHolder, що відповідає за зберігання посилань на текстові елементи одного рядка списку: категорію, суму, дату та тип транзакції. У методі onCreateViewHolder створюється нове представлення елемента списку на основі XML-макета item_transaction.xml.

Метод onBindViewHolder заповнює відповідні поля відображення даними конкретної транзакції. Через метод setTransactions список оновлюється, після чого викликається оновлення інтерфейсу. Метод getItemCount повертає кількість транзакцій, що мають бути відображені.

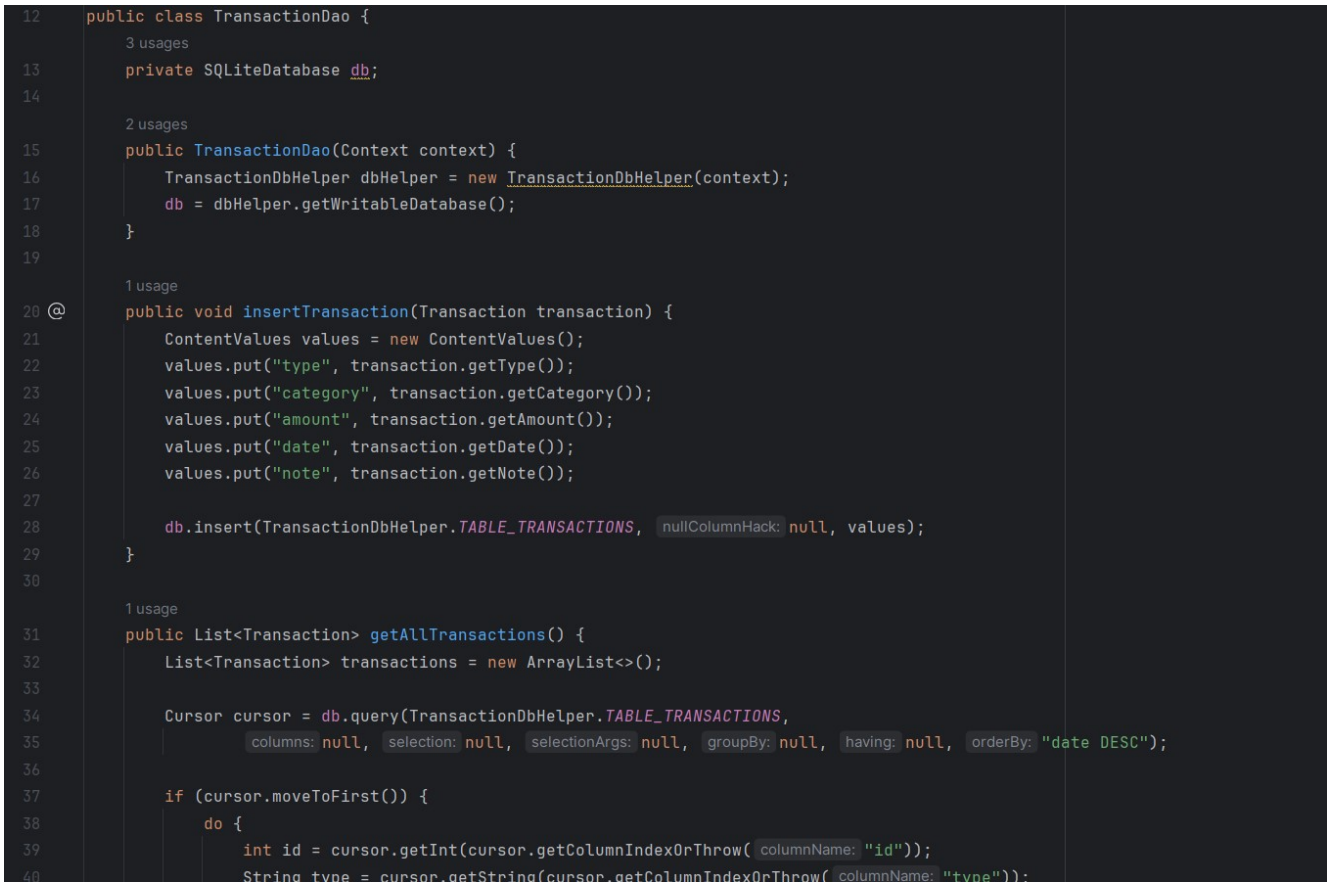
```
12  public class TransactionAdapter extends RecyclerView.Adapter<TransactionAdapter.TransactionViewHolder> {
13
        4 usages
14      private List<Transaction> transactions;
15
        1 usage
16      public void setTransactions(List<Transaction> transactions) {
17          this.transactions = transactions;
18          Log.d( tag: "Adapter",  msg: "Transactions set: " + (transactions != null ? transactions.size() : 0));
19          notifyDataSetChanged();
20      }
21
22
23      @NonNull
24      @Override
25      public TransactionViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
26          View view = LayoutInflater.from(parent.getContext())
27                  .inflate(R.layout.item_transaction, parent,  attachToRoot: false);
28          return new TransactionViewHolder(view);
29      }
30
31      @Override
32      public void onBindViewHolder(@NonNull TransactionViewHolder holder, int position) {
33          Transaction transaction = transactions.get(position);
34          holder.textViewCategory.setText(transaction.getCategory());
35          holder.textViewAmount.setText(String.valueOf(transaction.getAmount()));
36          holder.textViewDate.setText(transaction.getDate());
37          holder.textViewType.setText(transaction.getType());
38      }
39
40      @Override
41      public int getItemCount() {
42          return transactions != null ? transactions.size() : 0;
```

Рисунок 2.15 – Код класу TransactionAdapter

Клас TransactionDao відповідає за взаємодію з базою даних SQLite, у якій зберігаються дані про фінансові транзакції. У конструкторі відбувається ініціалізація бази за допомогою допоміжного класу TransactionDbHelper, що надає доступ до об'єкта SQLiteDatabase.

Метод insertTransaction дозволяє зберігати нову транзакцію, формуючи набір значень і вставляючи їх у таблицю бази даних. Для цього створюється об'єкт ContentValues, куди записуються основні атрибути: тип, категорія, сума, дата та примітка.

Метод getAllTransactions забезпечує зчитування всіх транзакцій із бази даних, відсортованих за спаданням дати. Після виконання SQL-запиту результати обробляються курсором, кожен запис перетворюється на об'єкт Transaction, який додається до списку. В процесі обробки виконується логування для спрощення налагодження. Отриманий список повертається як результат виклику методу.



```
public class TransactionDao {
    3 usages
    private SQLiteDatabase db;

    2 usages
    public TransactionDao(Context context) {
        TransactionDbHelper dbHelper = new TransactionDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    1 usage
    public void insertTransaction(Transaction transaction) {
        ContentValues values = new ContentValues();
        values.put("type", transaction.getType());
        values.put("category", transaction.getCategory());
        values.put("amount", transaction.getAmount());
        values.put("date", transaction.getDate());
        values.put("note", transaction.getNote());

        db.insert(TransactionDbHelper.TABLE_TRANSACTIONS, nullColumnHack: null, values);
    }

    1 usage
    public List<Transaction> getAllTransactions() {
        List<Transaction> transactions = new ArrayList<>();

        Cursor cursor = db.query(TransactionDbHelper.TABLE_TRANSACTIONS,
                columns: null, selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: "date DESC");

        if (cursor.moveToFirst()) {
            do {
                int id = cursor.getInt(cursor.getColumnIndexOrThrow( columnName: "id"));
                String type = cursor.getString(cursor.getColumnIndexOrThrow( columnName: "type"));
```

Рисунок 2.16 – Код класу TransactionDao

Клас TransactionDbHelper є допоміжним інструментом для роботи з базою даних SQLite у застосунку. Він розширює функціональність SQLiteOpenHelper і відповідає за створення та оновлення структури бази.

У конструкторі вказуються назва файлу бази (finance.db) та її версія. При першому запуску застосунку викликається метод onCreate, у якому формується SQL-запит на створення таблиці transactions. Ця таблиця містить поля: ідентифікатор (автоматично зростаючий), тип транзакції, назву категорії, суму, дату і примітку.

```
7   public class TransactionDbHelper extends SQLiteOpenHelper {
8
        1 usage
9       private static final String DATABASE_NAME = "finance.db";
        1 usage
10      private static final int DATABASE_VERSION = 1;
11
        4 usages
12      public static final String TABLE_TRANSACTIONS = "transactions";
13
        2 usages
14      public TransactionDbHelper(Context context) {
15          super(context, DATABASE_NAME,  factory: null, DATABASE_VERSION);
16      }
17
18      @Override
19      public void onCreate(SQLiteDatabase db) {
20          String SQL_CREATE_TABLE = "CREATE TABLE " + TABLE_TRANSACTIONS + " ("
21                  + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
22                  + "type TEXT NOT NULL, "
23                  + "category TEXT NOT NULL, "
24                  + "amount REAL NOT NULL, "
25                  + "date TEXT NOT NULL, "
26                  + "note TEXT)";
27          db.execSQL(SQL_CREATE_TABLE);
```

Рисунок 2.17 – Код класу TransactionDbHelper

У розділі представлено реалізацію ключових компонентів застосунку Finance Manager на базі архітектури MVVM. Описано структуру, взаємодію та функціональність програмних модулів.

3 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є невід’ємною частиною життєвого циклу розробки, оскільки воно дає змогу виявити помилки, перевірити відповідність реалізованого функціоналу визначеним вимогам та забезпечити належний рівень якості кінцевого продукту. У процесі створення мобільного додатку "Finance Manager" для Android було проведено комплексне тестування, зосереджене на перевірці коректності роботи основних функцій, стабільності інтерфейсу користувача, а також надійності взаємодії між компонентами архітектури MVVM [12].

3.1 Функціональне тестування

Особливий акцент було зроблено на функціональному тестуванні, яке є ключовим етапом у процесі контролю якості програмного забезпечення. Воно полягає у перевірці того, чи виконує система всі функції відповідно до вимог та очікуваннях кінцевих користувачів. Такий підхід дозволяє не лише виявити критичні помилки на ранніх етапах, а й переконатися, що кожен компонент системи працює згідно з призначенням та забезпечує очікувану поведінку в типових і граничних сценаріях.

У контексті розробки додатку "Finance Manager" функціональне тестування охопило всі основні аспекти роботи застосунку. Це включало перевірку точності обліку фінансових транзакцій, правильності відображення категорій витрат і доходів, стабільності механізму додавання та редагування записів, а також відповідність логіки взаємодії між елементами інтерфейсу й підсистемами, побудованими за архітектурним шаблоном MVVM [14].

У межах цього процесу було протестовано ключові функціональні можливості, зокрема:

1) Додавання транзакцій

Кроки:

- Перейти на сторінку "Транзакції".
- У полі "Назва транзакції" ввести: Кава.
- У полі "Сума" ввести: 70.
- У полі "Дата" вибрати: 2025-06-05.
- У випадаючому списку "Категорія" вибрати: Продукти.
- Натиснути кнопку "Додати".

Очікуваний результат:

- Транзакція з назвою "Кава", сумою - 70 UAH, датою 2025-06-05 та категорією "Продукти" з'являється у списку транзакцій.
- Поля вводу очищуються після додавання.

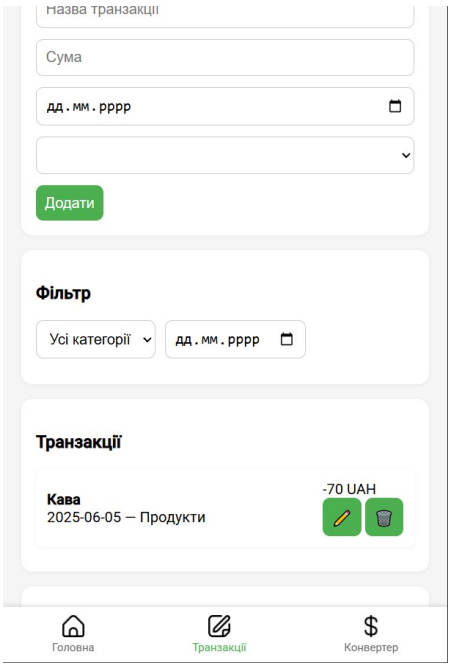


Рисунок 3.1 – Результат виконання сценарію Додавання транзакцій

2) Редагування транзакцій

Кроки:

- Перейти у вікно "Транзакції".

- У списку знайти транзакцію, яку потрібно змінити.
- Натиснути кнопку редагування (іконка).
- У формі редагування змінити назву, суму, дату або категорію.
- Натиснути кнопку "Зберегти".

Очікуваний результат:

- Змінена транзакція з новими даними відображається у списку.
- Дані коректно зберігаються.

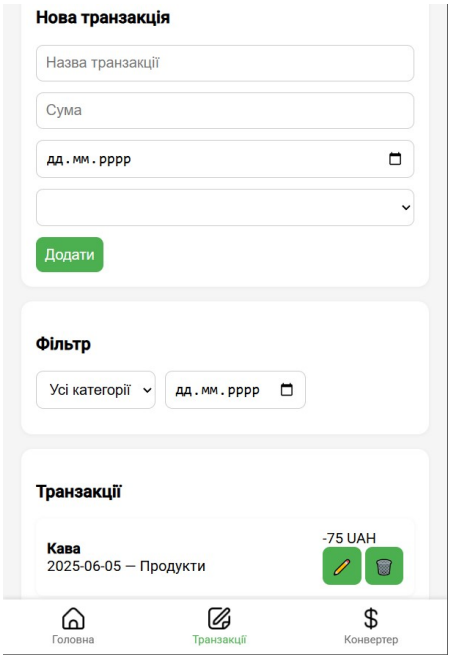


Рисунок 3.2 – Результат виконання сценарію Редагування транзакцій

3) Видалення транзакцій

Кроки:

- Перейти у вікно "Транзакції".
- У списку знайти транзакцію, яку потрібно видалити.
- Натиснути кнопку видалення (іконка).
- Підтвердити видалення у спливаючому діалоговому вікні (якщо є).

Очікуваний результат:

- Вибрана транзакція зникає зі списку.
- Дані видаляються з бази.

- Інтерфейс оновлюється без помилок.

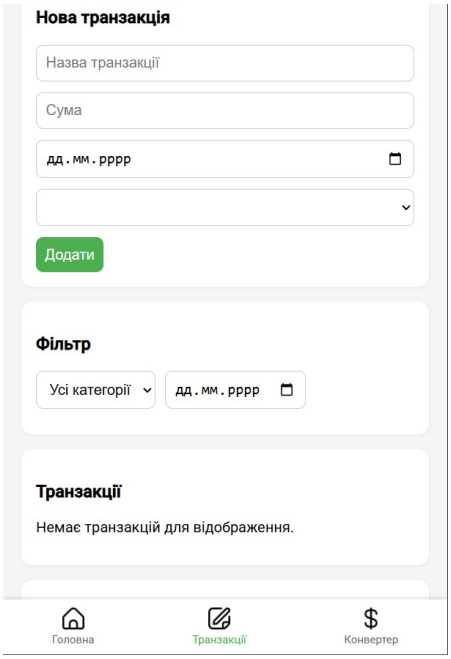


Рисунок 3.3 – Результат виконання сценарію Видалення транзакцій

4) Створення категорій

Кроки:

- Перейти у вікно Транзакції.
- Прокрутити до блоку «Категорії» (нижня частина сторінки).
- У текстове поле з плейсхолдером «Нова категорія» ввести, наприклад, Кафе.
- Натиснути кнопку «Додати категорію».

Очікуваний результат:

- Категорія Кафе з’являється у списку під кнопкою.
- Категорія також з’являється у випадаючому списку вибору категорій при створенні нової транзакції.
- Категорія доступна у фільтрах.
- При повторному введенні тієї самої назви категорії показується повідомлення «Ця категорія вже існує».
- Після додавання категорії поле вводу очищається.

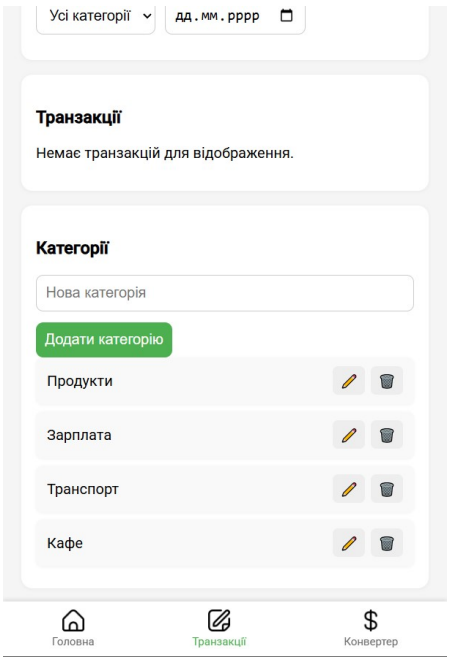


Рисунок 3.4 – Результат виконання сценарію Створення категорій

5) Редагування категорій

Кроки:

- Відкрити сторінку з категоріями.
- Знайти у списку категорій існуючу категорію (наприклад, «Продукти»).
- Натиснути кнопку ✏️ біля цієї категорії.
- У поле вводу «Нова категорія» має автоматично підставитись назва вибраної категорії.
- Внести зміни в назву категорії (наприклад, змінити «Продукти» на «Їжа»).
- Натиснути кнопку «Оновити категорію».
- Перевірити, що назва в списку категорій змінилась на нову.
- Перевірити, що у всіх транзакціях, які мали стару категорію, вона також змінилась на нову.

Очікуваний результат:

- Поле вводу автоматично заповнюється старою назвою категорії.
- Кнопка змінює текст на «Оновити категорію».

- Після оновлення назва категорії змінюється у списку.
- Відповідні транзакції оновлюються з новою назвою категорії.
- Поле вводу очищається, кнопка повертається до стану «Додати категорію».

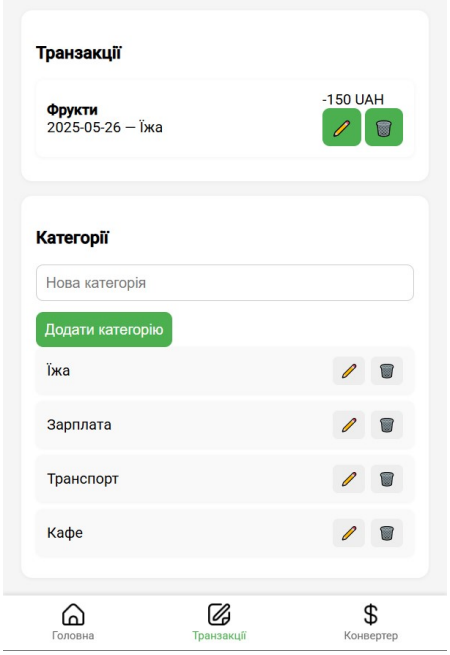


Рисунок 3.5 – Результат виконання сценарію Редагування категорій

6) Видалення категорій

Кроки:

- Відкрити сторінку з категоріями.
- Знайти у списку категорій існуючу категорію (наприклад, «Транспорт»).
- Натиснути кнопку 🗑 біля цієї категорії.
- У спливаючому вікні підтвердження натиснути «ОК» (підтвердити видалення).
- Перевірити, що категорія зникла зі списку.
- Перевірити, що у транзакціях, які мали цю категорію, поле категорії стало порожнім.
- Перевірити, що фільтри також не містять видаленої категорії.

Очікуваний результат:

- Категорія зникає зі списку категорій.
- У всіх пов’язаних транзакціях категорія очищується (стає порожньою).
- Видалена категорія не відображається у випадаючих списках для фільтрації чи додавання транзакцій.

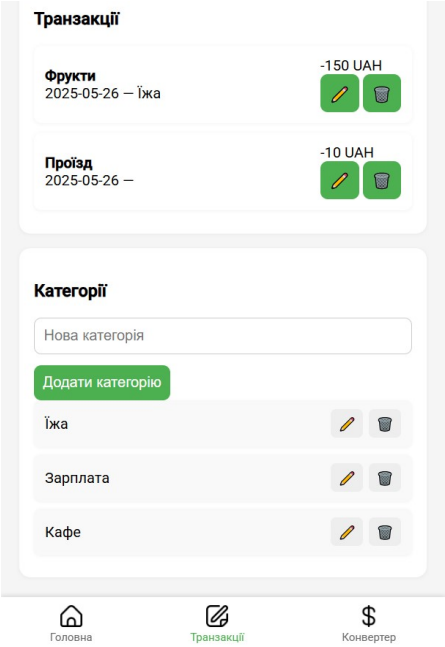


Рисунок 3.6 – Результат виконання сценарію Видалення категорій

7) Фільтрація транзакцій за датою

Кроки:

- Додати щонайменше дві транзакції з різними датами (наприклад, 2025-06-01 і 2025-05-27).
- Переконатися, що обидві транзакції відображаються у загальному списку.
- У полі фільтрації дати (під заголовком «Фільтр») вибрати дату 2025-06-01.
- Перевірити список транзакцій.

Очікуваний результат:

- У списку відображається лише транзакція з датою 2025-06-01.

- Транзакції з іншими датами не відображаються.
- Якщо немає транзакцій з вибраною датою, виводиться повідомлення: «Немає транзакцій для відображення.»

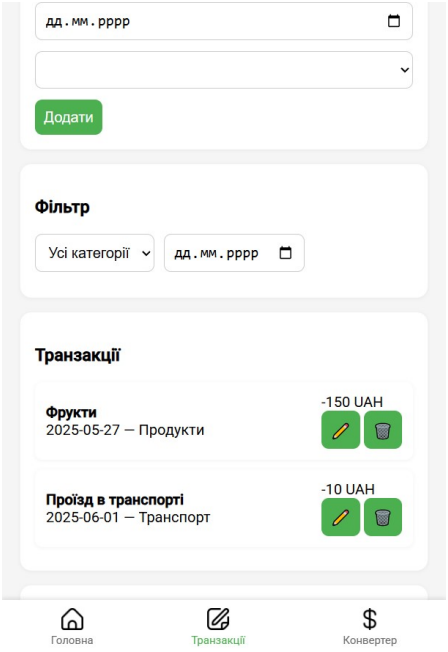


Рисунок 3.7 – Вікно до фільтрації

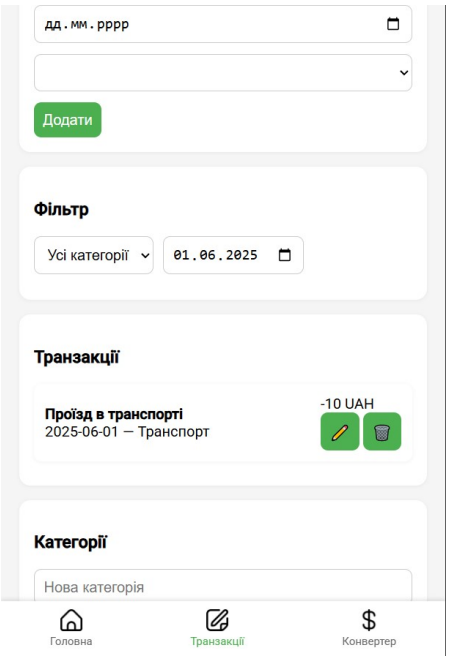


Рисунок 3.8 – Результат виконання сценарію Фільтрація транзакцій за датою

8) Фільтрація транзакцій за категорією

Кроки:

- Додати щонайменше дві транзакції з різними категоріями (наприклад, «Продукти» та «Транспорт»).
- Переконатися, що обидві транзакції відображаються у загальному списку.
- У полі фільтрації категорій (під заголовком «Фільтр») вибрати категорію «Продукти».
- Перевірити список транзакцій.

Очікуваний результат:

- У списку відображається лише транзакція з категорією «Продукти».
- Транзакції з іншими категоріями не відображаються.
- Якщо немає транзакцій у вибраній категорії, виводиться повідомлення: «Немає транзакцій для відображення.»

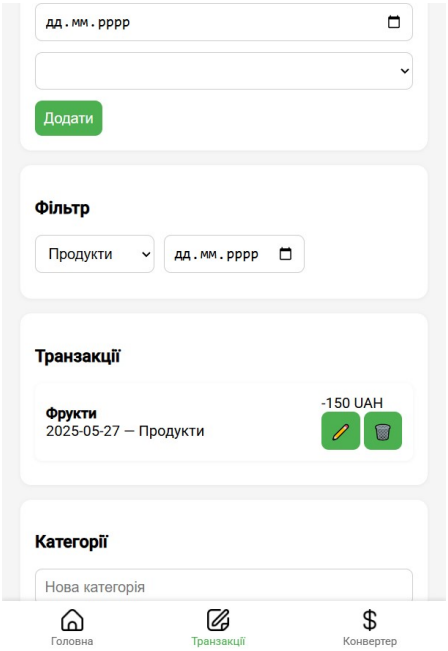


Рисунок 3.9 – Результат виконання сценарію Фільтрація транзакцій за категорією

9) Конвертування валют

Кроки:

- Відкрити вікно "Конвертер"

- У полі "Сума" ввести значення, наприклад: 100.
- У списку "З" вибрати валюту, наприклад: UAH.
- У списку "У" вибрати іншу валюту, наприклад: USD.
- Натиснути кнопку "Конвертувати".
- Переглянути результат конвертації, що відображається під кнопкою.

Очікуваний результат:

- Якщо введена коректна сума (додатнє число), у блоці #result з’являється текст формату:
- 100 UAH = 2.41 USD (значення залежить від курсу).
- Якщо поле суми порожнє, нечислове або менше/дорівнює нулю — з'являється повідомлення: Введіть коректну суму.

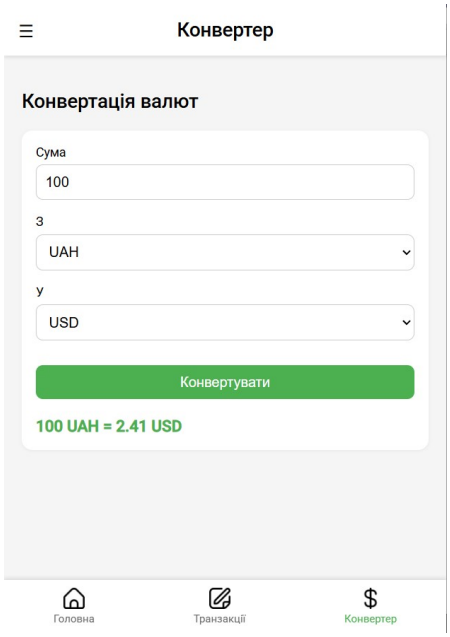


Рисунок 3.10 – Результат виконання сценарію Конвертування валют

Усі функціональні тести було успішно пройдено, що підтверджує коректність реалізації застосунку та відповідність його поведінки заданим вимогам. Це свідчить про надійність роботи основних компонентів і готовність системи до практичного використання.

3.2 Навантажувальне тестування

Навантажувальне тестування є важливим етапом оцінки надійності та стабільності програмного забезпечення в умовах інтенсивного використання. Метою цього виду тестування є визначення поведінки системи при підвищених навантаженнях, що можуть виникати в реальних умовах експлуатації.

Для перевірки стабільності мобільного додатку Finance Manager було проведено тестування за допомогою інструменту Monkey, який входить до складу Android SDK. Monkey генерує велику кількість випадкових подій — дотиків, свайпів, натискань кнопок — що імітують хаотичну поведінку користувача. Це дозволяє виявити помилки, які можуть виникнути внаслідок неочікуваної взаємодії з інтерфейсом, наприклад: збої, зависання додатку (ANR) або нативні помилки.

Було створено спеціальний .bat-файл, що автоматизує запуск тестування на віртуальному пристрої. Перед початком задаються основні параметри: шлях до утиліти adb, ідентифікатор пристрою (emulator-5554), назва пакета додатку (com.financemanager) та лог-файл (monkey-log.txt), у який записуються всі події.

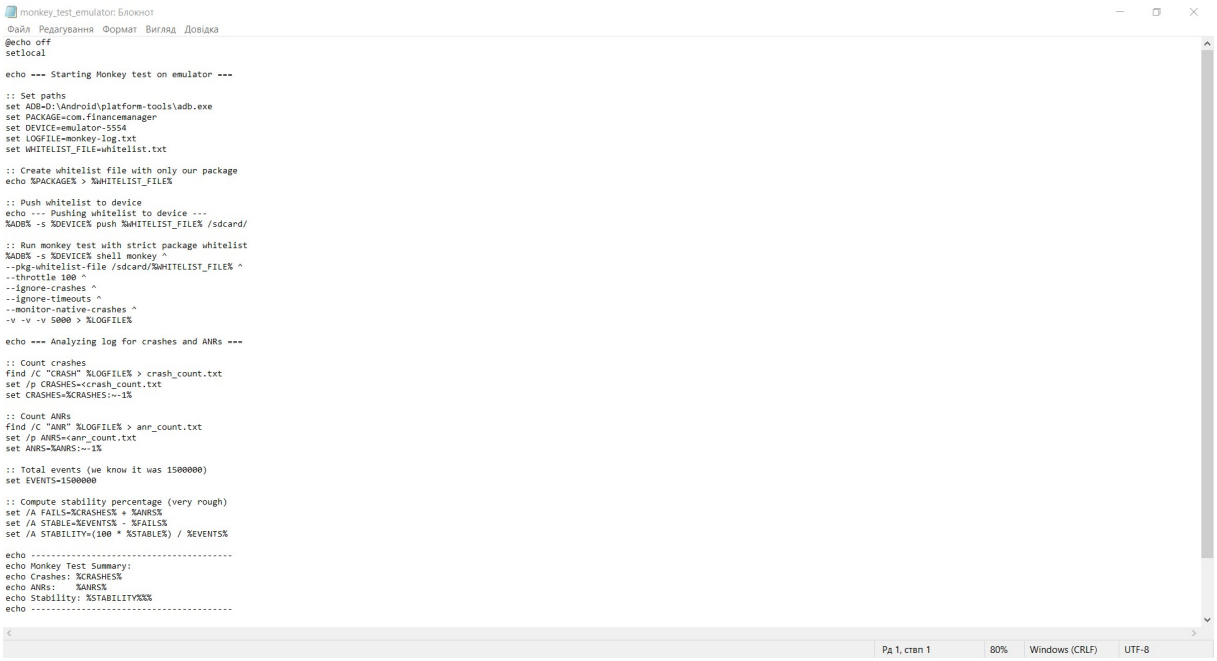


Рисунок 3.11 – Скрипт для запуску інструменту Monkey

Щоб обмежити тестування лише в межах додатку, було створено файл whitelist.txt, у якому вказано дозволений пакет. Цей файл копіюється у файлову систему емулятора і використовується Monkey через параметр --pkg-whitelist-file, що забезпечує тестування лише цільового додатку, без переходу до інших застосунків.

Опції запуску мають наступне значення:

- -p — задає цільовий пакет;
- -v 10000 — кількість подій;
- --throttle 150 — затримка 150 мс між подіями;
- --ignore-crashes, --ignore-timeouts,--monitor-native-crashes — забезпечують продовження тесту у разі збоїв чи зависань;
- --pkg-whitelist-file — whitelist-фільтрація лише дозволених додатків.

Після завершення тесту лог автоматично аналізується для підрахунку кількості збоїв і зависань за допомогою команди find. Дані підсумовуються, після чого приблизно оцінюється стабільність у відсотках за формулою:

STABILITY = ((EVENTS - (CRASHES + ANRS)) / EVENTS) * 100

У результаті тестування стабільність становила 100%, тобто жодних збоїв або зависань виявлено не було. Файл логу, відкритий автоматично у блокноті, підтвердив відсутність проблем.

Це дозволяє впевнено стверджувати, що Finance Manager поводиться стабільно навіть при тривалому та інтенсивному використанні.

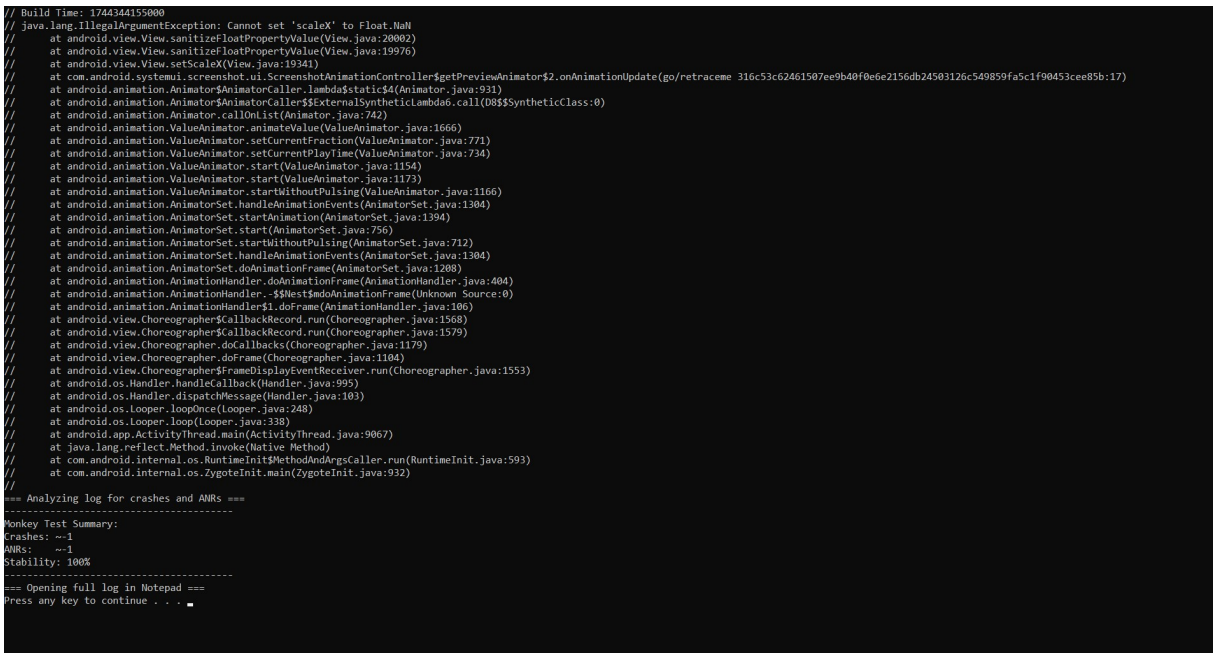


Рисунок 3.12 – Результат тестування

Результати навантажувального тестування підтвердили високу стабільність мобільного застосунку Finance Manager в умовах інтенсивної взаємодії з інтерфейсом. Використання інструменту Monkey дало змогу змоделювати тисячі випадкових подій, які імітують поведінку користувача, виявити потенційні критичні збої та переконатися у відсутності зупинок чи аварійних завершень роботи. Стабільність у 100% без виявлених помилок свідчить про надійність реалізації інтерфейсу та внутрішньої логіки додатку, що є важливим показником готовності до широкого використання у реальному середовищі.

4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Поведінкові реакції населення у надзвичайних ситуаціях

У надзвичайних ситуаціях (НС) поведінка населення є ключовим чинником, що безпосередньо впливає на рівень втрат і загальну ефективність заходів захисту. Реакція людей на кризу визначається поєднанням зовнішніх обставин — таких як рівень загрози, умови місцевості, наявність засобів захисту, організація евакуації — та внутрішніх факторів, зокрема рівня поінформованості, психологічної готовності, досвіду, довіри до органів влади.

Державна система цивільного захисту створена для того, щоб забезпечити населення умовами для адекватного реагування: організовується життєзабезпечення, своєчасне сповіщення та підготовка до евакуації [9]. Сповіщення про загрозу — перший і критично важливий етап, що впливає на поведінку людей. Інформація подається через телебачення, радіо, гучномовці, сирени та інші засоби комунікації. Вона має бути своєчасною, чіткою, зрозумілою та такою, що формує довіру до джерела.

Реакція населення на тривожні повідомлення може варіюватися: від організованої евакуації до паніки або ігнорування небезпеки. Щоб зменшити ризики, велике значення має попередня підготовка — навчання населення правильним діям у надзвичайних ситуаціях, яке проводиться в навчальних і дошкільних закладах, на підприємствах, у громадських місцях. Особливої уваги потребує морально-психологічна підготовка, яка формує стресостійкість, здатність до рішень і колективних дій у кризовій ситуації [8].

Під час евакуації особливо важливими є організованість, дисципліна, готовність дотримуватись інструкцій та підтримувати одне одного. Залежно від ситуації, використовуються різні види транспорту — залізничний, автомобільний, водний, або ж організовуються пішохідні маршрути.

Однак навіть при належній підготовці, в реальних умовах НС людина може зазнавати сильного стресу. Тому важливо діяти швидко, рішуче та усвідомлено.

Насамперед потрібно оглянути місце події, переконатися у власній безпеці — безпека рятівника є пріоритетом. Лише після цього слід надавати допомогу іншим.

Оцінка стану постраждалих передбачає з'ясування, чи є серед них ті, хто потребує невідкладної медичної допомоги, чи існує загроза життю. У разі потреби негайно викликаються екстрені служби — медики, рятувальники. До їхнього прибуття необхідно залишатися з постраждалими, надавати фізичну та психологічну підтримку.

Перед будь-якими діями важливо заспокоїтися, зосередитись, представитися: «Я — волонтер», «Я допомагаю постраждалим», дізнатися ім’я постраждалого. Просте речення «Допомога вже в дорозі» може дати людині відчуття безпеки.

Психологічна підтримка на місці події включає дії, спрямовані на відновлення базових життєвих функцій. Наприклад, ритмічне дихання: «Вдих… затримка… видих…». Встановлення зорового контакту, звернення по імені або прості запитання — «Ви мене чуєте?», «Де ви зараз?» — допомагають людині зорієнтуватися в реальності. Якщо людина розгублена, можна коротко пояснити, що сталося: «Тут був вибух. Ви злякалися». Потім спробувати відновити хронологію подій: «Що було спочатку?».

Фізичний контакт має бути обережним — допустимі дотик до руки чи плеча, уникати доторків до голови або інших вразливих частин тіла. Розмова має відбуватись на одному рівні — присядьте біля людини, не стійте над нею. Дітей дозволено обережно обійняти. В жодному разі не залишайте постраждалих на самоті.

Основні принципи ефективної психологічної підтримки: не звинувачувати, не обіцяти того, в чому не впевнені, не вигадувати факти, не нав’язувати власних історій. Варто запропонувати людині нескладні завдання: «Одягніть куртку», «Випийте води» — це підвищує впевненість у власних силах. Також важливо чесно повідомляти, де й як можна отримати допомогу, залучати інших людей до підтримки, даючи їм чіткі інструкції.

Під час розмови необхідно уважно слухати, підтримувати контакт очима, не перебивати, не квапити. Якщо людина не готова говорити — просто будьте поруч. Пропонуйте практичну допомогу: воду, їжу, ковдру. Поважайте культурні особливості: якщо фізичний контакт може бути недоречним, утримайтеся від нього.

Якщо людина дезорієнтована, допоможіть їй повернутися до реальності: запропонуйте впертися ногами в землю, доторкнутися до колін, назвати кольори предметів навколо, зосередитися на диханні.

Особлива увага — дітям. Їхні реакції залежать від віку, розвитку й поведінки дорослих. Малюки можуть повертатися до інфантильних звичок, школярі — звинувачувати себе чи замикатися в собі, а підлітки — втрачати контакт з оточенням. Усі діти потребують спокійного дорослого поруч, який вислухає, підтримає, не осудить і не залишить на самоті.

Що потрібно / не потрібно робити та говорити під час підтримки постраждалих:

Таблиця 4.1 - Заходи підтримки постраждалого

ПОТРІБНО	НЕДОЦІЛЬНО
Знайти тихе місце для розмови	Змушувати говорити
Дотримуватись дистанції з урахуванням культури	Торкатися без впевненості в доречності
Слухати уважно, демонструвати зацікавленість	Давати оцінки
Бути терплячим і спокійним	Казати: «Ви не повинні себе так відчувати»
Чесно повідомляти наявну інформацію	Вигадувати факти
Використовувати прості слова	Вживати спеціальні терміни
Висловлювати співчуття	Давати хибні обіцянки
Дати можливість побути в тиші	Розповідати про власні труднощі

Продовження таблиці 4.1

ПОТРІБНО	НЕДОЦІЛЬНО
Поважати самостійність людини	Говорити зневажливо або принижувати

Успішна реакція на НС залежить від підготовленості, сповіщення та підтримки. Психологічна стабільність рятує не менше, ніж фізичні дії.

4.2 Розробка конкретних заходів щодо боротьби із статичною електрикою

Статична електрика виникає внаслідок накопичення електричних зарядів на поверхні тіл, особливо при терті, розділенні або механічному впливі на легкоелектризуючі матеріали. У промислових умовах, особливо під час роботи з високовольтними установками постійного струму, синтетичними матеріалами або в середовищах з низькою вологістю, статична електрика може досягати значних величин, створюючи небезпеку виникнення електричних розрядів.

Відповідно до ДСТУ 7302:2013, для боротьби зі статичною електрикою слід розробляти і впроваджувати систему технічних і організаційних заходів, спрямованих на попередження накопичення зарядів та їх своєчасне відведення [10]. До таких заходів належать:

1) Заземлення та зрівнювання потенціалів. Основним технічним заходом є систематичне заземлення елементів обладнання, конструкцій, механізмів, ємностей, які можуть накопичувати статичний заряд. Для цього використовують струмопровідні заземлювальні пристрої, які мають бути розраховані на відповідний опір заземлення згідно з нормативними вимогами.

Зрівнювання потенціалів запобігає іскровому пробою між двома об'єктами з різницею потенціалів, особливо в умовах наявності горючих речовин чи вибухонебезпечного пилу.

2) Використання антистатичних матеріалів і покриттів. У випадках, коли заземлення неефективне або неможливе (наприклад, при роботі з діелектриками), застосовують антистатичні матеріали або антистатичні покриття. Це можуть бути спеціальні просочення, фарби, плівки, які знижують поверхневий опір і сприяють стіканню зарядів.

3) Підтримання відповідного мікроклімату. Відповідно до ДСТУ 7302:2013, одним із факторів, що значно впливає на утворення статичних зарядів, є вологість повітря. Тому важливим заходом є регулювання мікроклімату в приміщеннях, зокрема підтримка відносної вологості повітря не нижче 60 %, що значно знижує електроізоляційні властивості повітря і запобігає накопиченню зарядів.

4) Екранування і еквіпотенціальне з'єднання. У конструкціях, де присутні джерела високої напруги або випромінювання, застосовують екрани з провідного матеріалу, які відводять заряди і перешкоджають їх поширенню. Також використовують еквіпотенціальне з'єднання всіх металевих елементів для унеможливлення виникнення локальних різниць потенціалів.

5) Застосування іонізаторів повітря. В окремих технологічних процесах для нейтралізації зарядів використовуються іонізатори повітря, які генерують потік іонів протилежного знаку до накопиченого заряду. Такі пристрої особливо ефективні в умовах виробництв з високою електроізоляцією поверхонь.

6) Контроль і регламентоване використання спецодягу. Особи, які працюють в умовах утворення електростатичних зарядів, повинні використовувати спеціальний антистатичний одяг і взуття, яке не допускає накопичення зарядів на тілі людини. Такий одяг повинен відповідати вимогам електропровідності та мати спеціальне маркування.

7) Організаційні заходи і навчання персоналу. Не менш важливою складовою є інструктаж і навчання персоналу щодо безпечного поводження з обладнанням, яке здатне накопичувати статичну електрику. Сюди також входить регулярне технічне обслуговування і перевірка заземлювальних пристроїв, вимірювання опору заземлення, вологість повітря тощо.

ВИСНОВКИ

У межах роботи було спроєктовано та реалізовано мобільний фінансовий застосунок "Finance Manager" для Android. Основна мета — створити зручний інструмент для обліку особистих фінансів — досягнута повністю. Застосунок дозволяє користувачам додавати, редагувати та видаляти транзакції, керувати категоріями, планувати бюджети, встановлювати фінансові цілі та переглядати історію операцій.

Архітектура додатку побудована за шаблоном MVVM, що забезпечило чітке розділення логіки, зручність підтримки та масштабованість. Для зберігання даних використано Room/SQLite, а ViewModel і LiveData реалізують реактивну взаємодію між інтерфейсом і даними.

Застосунок має інтуїтивно зрозумілий інтерфейс, орієнтований на щоденне використання. Його структура й логіка були детально спроєктовані та представлені у вигляді діаграм, що дало змогу ефективно організувати розробку.

Було проведено ручне тестування функціоналу: додавання, редагування та видалення транзакцій і категорій, перевірку збереження даних при зміні орієнтації екрана та навігації між фрагментами. Застосунок продемонстрував стабільну й передбачувану роботу без критичних помилок.

Окремо проведено автоматизоване тестування за допомогою Monkey (Android SDK), що згенерував десяти тисяч випадкових подій. За результатами аналізу логів, не було зафіксовано жодного збою або зависання, отже, стабільність додатку становить 100%.

"Finance Manager" виконує важливу практичну функцію: допомагає користувачам краще розуміти та контролювати свої фінанси, планувати витрати й досягати фінансових цілей. Завдяки зручності, надійності та простоті у використанні, застосунок має потенціал стати корисним інструментом у повсякденному житті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Information System for Design of Thin Multilayer Film Processes Parameters Management based on Diffusion Mykhaylo Petryk, Vitalii Chyzh, Halyna Tsupryk, Oksana Petryk, ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23- 25, 2024, Ternopil, Ukraine, Opole, Poland, 2024, pp. 486-493 (Scopus) https://ceur-ws.org/Vol-3896/

2.Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90. (фахове видання України)

3. Яворська, ЄБ, Кінаш, РВ, Цуприк, ГБ, Николайчук ВІ Проблеми виявлення біосигналів у медичних інформаційних системах/ЄБ Яворська, Кінаш РВ, ГБ Цуприк, ВІ Николайчук//IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині»(ІСМ–2021)[Текст]: зб. наук. пр.–Харків: Нац. аерокосм. ун-т ім. МЄ Жуковського «Харків. авіац. ін-т», 2021.– 25-26 с.

4. Дарнобит Н., Цуприк Г. Б. Розробка Android-застосунку для пошуку та збору інформації мовою Java. Google Scholar. URL: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=9pNcfCkAAAAJ&sortby=pubdate&citation_for_view=9pNcfCkAAAAJ:dQ2og3OwTAUC (дата звернення: 08.06.2025).

5. Mathematical Model of the Capacitor Based on Zeolite Material / I. Boyko et al. Google Scholar. URL: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=9pNcfCkAAAAJ&sortby=pubdate&citation_for_view=9pNcfCkAAAAJ:9vf0nzSNQJEC (дата звернення: 08.06.2025).

6. Piezoelectric Properties and Electron-Phonon Interaction in Semiconductor Arsenide GaAs/AlAs Nanosystems of Plane Symmetry / Igor Boyko та ін. Google Scholar. 2022. С. 1–5. URL: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=9pNcfCkAAAAJ&sortby=pubdate&citation_for_view=9pNcfCkAAAAJ:ye4kPcJQO24C (дата звернення: 16.06.2025).

7. Nitride Semiconductor Quantum Dots-Mathematical Models of the Electronic Spectrum and Methods for its Simulation / J. Nestor та ін. Google Scholar. 2022. С. 136–139. URL: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=9pNcfCkAAAAJ&sortby=pubdate&citation_for_view=9pNcfCkAAAAJ:LjlpjdlvIbIC (дата звернення: 08.06.2025).

8. Гурик О., Окіпний І. Безпека життєдіяльності, основи охорони праці. Електронне навчання в ТНТУ. URL: https://dl.tntu.edu.ua/content.php?cid=289173 (дата звернення: 01.06.2025).

9. Дії у надзвичайних ситуаціях. Державна служба України з надзвичайних ситуацій. URL: https://dsns.gov.ua/abetka-bezpeki/diyi-u-nadzvicainix-situaciyax (дата звернення: 01.06.2025).

10. ДСТУ 7302:2013. Статична електрика. Терміни та визначення основних понять. Чинний від 2014-01-01. Вид. офіц. Київ : Мінекономрозвитку України, 2014. 23 с.

11. Panchal K. Step by Step guide to create basic MVVM Application in Android. Medium. URL: https://medium.com/@khush.panchal123/step-by-step-guide-to-create-basic-mvvm-application-in-android-4c1bd83ac628 (дата звернення: 08.06.2025).

12. Code with FK. Build an Expense Tracker App with Jetpack Compose & MVVM | Android Development Tutorial, 2024. YouTube. URL: https://www.youtube.com/watch?v=LfHkAUzup5E (дата звернення: 08.06.2025).

13. Vishal. A Step-by-Step Guide to Android Room Database. *Medium*. URL: https://medium.com/@biz.vishalbarot/a-step-by-step-guide-to-android-room-database-585ae375aae5 (дата звернення: 08.06.2025).

14. GitHub - HongjoLim/My-Financial-Manager: Personal Android mobile application development project. #Java, #SQLite. GitHub. URL: https://github.com/HongjoLim/My-Financial-Manager (дата звернення: 08.06.2025).

15. Build your first app | Get started | Android Developers. Android Developers. URL: https://developer.android.com/get-started/overview (дата звернення: 08.06.2025).

16. A Theoretical Model of Thermal Conductivity for Multilayer Nitride-based Nanosystems / Igor Boyko та ін. Google Scholar. 2022. С. 111–114. URL: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=9pNcfCkAAAAJ&sortby=pubdate&citation_for_view=9pNcfCkAAAAJ:WqliGbK-hY8C (дата звернення: 16.06.2025).

17. Android Jetpack Dev Resources - Android Developers. Android Developers. URL: https://developer.android.com/jetpack (дата звернення: 16.06.2025).

18. Kotlin-Java interop guide | Android Developers. Android Developers. URL: https://developer.android.com/kotlin/interop (дата звернення: 16.06.2025).

19. Home Page. SQLite. URL: https://www.sqlite.org/index.html (дата звернення: 16.06.2025).

20. Save data in a local database using Room | App data and files. Android Developers. URL: https://developer.android.com/training/data-storage/room (дата звернення: 16.06.2025).

21. Material Design. URL: https://m3.material.io/ (дата звернення: 16.06.2025).

22. Android Development. Eclipse, Android and Java training and support. URL: https://www.vogella.com/tutorials/android.html (дата звернення: 16.06.2025).

23. Alcérreca J. ViewModels and LiveData: Patterns + AntiPatterns. Medium. URL: https://medium.com/androiddevelopers/viewmodels-and-livedata-patterns-antipatterns-21efaef74a54 (дата звернення: 16.06.2025).

24. Security checklist | Android Developers. Android Developers. URL: https://developer.android.com/privacy-and-security/security-tips (дата звернення: 16.06.2025).

25. Fundamentals of testing Android apps | Test your app on Android | Android Developers. Android Developers. URL: https://developer.android.com/training/testing/fundamentals (дата звернення: 16.06.2025).

26. GeeksforGeeks. Monkey Software Testing - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/software-testing/monkey-software-testing/ (дата звернення: 16.06.2025).

27. Mario Sanoguera de Lorenzo. Clean Architecture Guide (with tested examples): Data Flow != Dependency Rule. Medium. URL: https://proandroiddev.com/clean-architecture-data-flow-dependency-rule-615ffdd79e29 (дата звернення: 16.06.2025).

28. ViewModel overview | App architecture | Android Developers. Android Developers. URL: https://developer.android.com/topic/libraries/architecture/viewmodel (дата звернення: 16.06.2025).

29. Best practices for SQLite performance | App quality | Android Developers. Android Developers. URL: https://developer.android.com/topic/performance/sqlite-performance-best-practices (дата звернення: 16.06.2025).

30. Saved State module for ViewModel | App architecture | Android Developers. Android Developers. URL: https://developer.android.com/topic/libraries/architecture/viewmodel/viewmodel-savedstate (дата звернення: 16.06.2025).

ДОДАТКИ

ДОДАТОК А – Тези конференції

УДК 004.41
Д.В. Коваль, Г.Б. Цуприк, к.т.н., доц..
Тернопільський національний технічний університет імені Івана Пулюя, Україна

МОДУЛЬНА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ФІНАНСОВОГО КОНТРОЛЮ НА ANDROID ІЗ ЗАСТОСУВАННЯМ ПРИНЦИПІВ ІНВЕРСІЇ УПРАВЛІННЯ ТА ШАБЛОНУ MVVM

D.V. Koval, H.B. Tsupryk, Ph.D., Assoc. Prof.

MODULAR IMPLEMENTATION OF FINANCIAL CONTROL FUNCTIONALITY ON ANDROID USING INVERSION OF CONTROL PRINCIPLES AND THE MVVM PATTERN

Мобільні застосунки для фінансового контролю набули значної популярності серед користувачів, оскільки вони допомагають ефективно управляти особистими фінансами. Такі програми дозволяють стежити за доходами та витратами, створювати бюджети, аналізувати фінансові звіти та досягати фінансових цілей. Однак для забезпечення надійної та ефективної роботи таких застосунків важливо правильно організувати їх внутрішню структуру, щоб забезпечити гнучкість, масштабованість і легкість у підтримці.

Одним з основних підходів до розробки мобільних застосунків є використання модульної архітектури, що передбачає поділ програми на окремі компоненти. Кожен з цих компонентів має свою чітко визначену функцію і може бути розроблений, тестований і оновлений незалежно від інших частин програми. Наприклад, в фінансовому застосунку один модуль може відповідати за облік доходів і витрат, інший — за категоризацію витрат, а ще один — за аналіз бюджету чи формування фінансових цілей.

Такий підхід до структуризації дає можливість працювати з кожною частиною програми окремо, що значно спрощує роботу розробників. Модульність також дає можливість розподіляти завдання між кількома розробниками, зменшуючи час розробки і підвищуючи ефективність роботи над проєктом. Крім того, модульність дозволяє простіше додавати нові функції в застосунок без потреби переписувати вже готовий код, що робить програму більш гнучкою і стійкою до змін.

Шаблон MVVM (Model-View-ViewModel) є одним із найбільш поширених підходів до побудови архітектури мобільних застосунків, оскільки він дозволяє чітко розділити відповідальність між різними частинами програми. У цьому підході є три основні компоненти: Model, View і ViewModel.

Model відповідає за обробку даних. У випадку фінансового застосунку це можуть бути операції з транзакціями, категоріями витрат, обчислення залишку на рахунку тощо. Модель не має відношення до відображення інформації на екрані або взаємодії з користувачем, вона лише обробляє і надає необхідні дані для інших частин програми.

View — це частина програми, яка безпосередньо взаємодіє з користувачем. Вона відповідає за відображення інтерфейсу та приймання введених даних. Важливо, що View не повинна містити складної логіки — вона лише відображає дані, що отримуються з ViewModel. Така структура дозволяє значно спростити тестування інтерфейсу, оскільки View не містить складних залежностей від інших частин програми.

ViewModel є проміжним ланцюгом між Model і View. Вона отримує дані з Model, готує їх для відображення в View та забезпечує зв'язок між цими компонентами. ViewModel дозволяє відокремити бізнес-логіку від інтерфейсу, завдяки чому програма стає більш зрозумілою та зручною для розробників. Замість того, щоб безпосередньо маніпулювати даними, ViewModel відповідає за організацію логіки, яка визначає, які дані і коли повинні бути показані користувачу. Крім того, завдяки використанню механізмів спостереження за даними, таких як LiveData, зміни в даних автоматично призводять до оновлення інтерфейсу, що забезпечує зручну та динамічну взаємодію з користувачем.

Ще одним важливим аспектом є принцип інверсії управління. Це означає, що компоненти застосунку не створюють залежностей самостійно, а отримують їх ззовні. Такий підхід дозволяє значно знизити зв'язаність між компонентами і полегшує тестування та розширення програми. Замість того, щоб кожен модуль самостійно створював потрібні йому об'єкти, ці об'єкти передаються ззовні, що дає змогу централізовано керувати залежностями. В результаті, програмний код стає більш гнучким і зручним для подальших змін.

Завдяки використанню принципу інверсії управління, кожен компонент програми можна тестувати окремо, замінюючи реальні об'єкти на підроблені для цілей тестування. Це дозволяє легко виявляти помилки в кожному модулі без необхідності тестувати всю програму одночасно.

Застосування таких підходів, як MVVM і інверсія управління, в поєднанні з модульною архітектурою, дає значні переваги при розробці мобільних застосунків. Кожен компонент є незалежним, що полегшує їхнє тестування та оновлення. Крім того, цей підхід дозволяє значно спростити масштабування програми. Якщо з'являються нові вимоги або функції, їх можна легко додати до програми, не порушуючи існуючої структури. Також цей підхід дозволяє значно знизити ймовірність виникнення помилок, оскільки кожен компонент має свою чітко визначену роль і не залежить від інших частин програми.

У підсумку, модульна архітектура, шаблон MVVM та інверсія управління є потужними інструментами для створення стабільних, гнучких і зручних для користувача мобільних застосунків. Вони дозволяють розробникам створювати програмне забезпечення, яке легко підтримується, розширюється і адаптується до нових вимог користувачів. Ці підходи сприяють підвищенню якості коду, полегшують тестування та забезпечують надійність і стабільність застосунку в довгостроковій перспективі.

Література

1. Guide to app architecture | App architecture | Android Developers. Android Developers. URL: https://developer.android.com/topic/architecture (дата звернення: 05.05.2025).

2. Android Programming: The Big Nerd Ranch Guide / Б. Філліпс та ін. 4-те вид. Pearson Higher Education & Professional Group, 2019. 624 с.

3. Бойко І.В., Петрик М.Р., Цуприк Г.Б. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

4. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90.

5. Information System for Design of Thin Multilayer Film Processes Parameters Management based on Diffusion M.R., Petryk, Mykhaylo R., V., Chyzh, Vitalii, H.B., Tsupryk, H. B., O.Y., Petryk, Oksana Yu ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23- 25, 2024, Ternopil, Ukraine, Opole, Poland, 2024, pp. 486-493

ДОДАТОК Б – Лістинг коду мобільного додатку

Файл «AddTransactionActivity.java»

```
package com.financemanager;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

public class AddTransactionActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add_transaction);

        if (savedInstanceState == null) {
            // Додаємо фрагмент до activity
            FragmentManager fragmentManager = getSupportFragmentManager();
            FragmentTransaction transaction = fragmentManager.beginTransaction();
            transaction.replace(R.id.fragment_container, new
AddTransactionFragment());
            transaction.commit();
        }
    }
}
```

Файл «AddTransactionFragment.java»

```
package com.financemanager;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;

import java.util.List;

public class AddTransactionFragment extends Fragment {

    private TransactionViewModel viewModel;
    private EditText amountEditText, noteEditText, dateEditText;
    private Spinner categorySpinner;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
```

```
        View view = inflater.inflate(R.layout.fragment_add_transaction, container,
false);
        viewModel = new
ViewModelProvider(requireActivity()).get(TransactionViewModel.class);

        amountEditText = view.findViewById(R.id.editTextAmount);
        noteEditText = view.findViewById(R.id.editTextNote);
        dateEditText = view.findViewById(R.id.editTextDate);
        categorySpinner = view.findViewById(R.id.spinnerCategory);

        CategoryViewModel categoryViewModel = new
ViewModelProvider(requireActivity()).get(CategoryViewModel.class);
        categoryViewModel.getCategoryList().observe(getViewLifecycleOwner(),
categories -> {
            if (categories != null && !categories.isEmpty()) {
                ArrayAdapter<String> adapter = new ArrayAdapter<>(
                        requireContext(),
                        android.R.layout.simple_spinner_item,
                        getCategoryNames(categories)
                );

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
                categorySpinner.setAdapter(adapter);
            }
        });

        categoryViewModel.loadCategories(requireContext());

        view.findViewById(R.id.buttonSaveTransaction).setOnClickListener(v -> {
            try {
                String type = "expense"; // Тимчасово. Потім реалізуємо вибір
типу.
                String category = categorySpinner.getSelectedItem().toString();
                double amount =
Double.parseDouble(amountEditText.getText().toString());
                String date = dateEditText.getText().toString();
                String note = noteEditText.getText().toString();

                Transaction transaction = new Transaction(0, type, category,
amount, date, note);
                viewModel.addTransaction(requireContext(), transaction);

                Toast.makeText(getContext(), "Транзакцію збережено",
Toast.LENGTH_SHORT).show();
                clearFields();
            } catch (Exception e) {
                Toast.makeText(getContext(), "Помилка введення даних",
Toast.LENGTH_SHORT).show();
            }
        });

        return view;
    }

    private List<String> getCategoryNames(List<Category> categories) {
        return categories.stream().map(Category::getName).toList();
    }

    private void clearFields() {
        amountEditText.setText("");
        noteEditText.setText("");
        dateEditText.setText("");
        if (categorySpinner.getAdapter().getCount() > 0) {
            categorySpinner.setSelection(0);
```

```
        }
    }
}
```

Файл «Category.java»

```
package com.financemanager;

import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "categories")
public class Category {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String name;

    public Category(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

Файл «CategoryDao.java»

```
package com.financemanager;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import java.util.ArrayList;
import java.util.List;

public class CategoryDao {
    private SQLiteDatabase db;

    public CategoryDao(Context context) {
        TransactionDbHelper dbHelper = new TransactionDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    public void insertCategory(Category category) {
        ContentValues values = new ContentValues();
        values.put("name", category.getName());
        db.insert("categories", null, values);
    }
```

```
    public List<Category> getAllCategories() {
        List<Category> categories = new ArrayList<>();

        Cursor cursor = db.query("categories", null, null, null, null, null,
null);
        if (cursor.moveToFirst()) {
            do {
                int id = cursor.getInt(cursor.getColumnIndexOrThrow("id"));
                String name =
cursor.getString(cursor.getColumnIndexOrThrow("name"));
                categories.add(new Category(id, name));
            } while (cursor.moveToNext());
        }

        cursor.close();
        return categories;
    }
}
```

Файл «CategoryListFragment.java»

```
package com.financemanager;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;

public class CategoryListFragment extends Fragment {

    private CategoryViewModel viewModel;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_category_list, container,
false);
        viewModel = new ViewModelProvider(this).get(CategoryViewModel.class);

        viewModel.getCategoryList().observe(getViewLifecycleOwner(), categories ->
{
            // Тут можна оновити RecyclerView для категорій
        });

        viewModel.loadCategories(requireContext());

        return view;
    }
}
```

Файл «CategoryViewModel.java»

```
package com.financemanager;

import android.content.Context;
```

```
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;
import java.util.List;

public class CategoryViewModel extends ViewModel {
    private final MutableLiveData<List<Category>> categoryList = new
MutableLiveData<>();

    public LiveData<List<Category>> getCategoryList() {
        return categoryList;
    }

    public void loadCategories(Context context) {
        CategoryDao dao = new CategoryDao(context);
        categoryList.setValue(dao.getAllCategories());
    }

    public void addCategory(Context context, Category category) {
        CategoryDao dao = new CategoryDao(context);
        dao.insertCategory(category);
        loadCategories(context);
    }
}
```

Файл «HomeFragment.java»

```
package com.financemanager;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.core.content.ContextCompat;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;

import com.google.android.material.floatingactionbutton.FloatingActionButton;

public class HomeFragment extends Fragment {

    private TextView textBalance, textIncome, textExpenses;
    private FloatingActionButton buttonAddTransaction;
    private TransactionViewModel transactionViewModel;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_home, container, false);

        // Ініціалізація текстових полів
        textBalance = view.findViewById(R.id.textBalance);
        textIncome = view.findViewById(R.id.textIncome);
        textExpenses = view.findViewById(R.id.textExpenses);
```

```
        // ViewModel для транзакцій
        transactionViewModel = new
ViewModelProvider(this).get(TransactionViewModel.class);
        transactionViewModel.loadTransactions(requireContext());

        // Спостерігач для оновлення даних балансу
        transactionViewModel.getTransactionList().observe(getViewLifecycleOwner(),
transactions -> {
            double balance = 0.0;
            double income = 0.0;
            double expense = 0.0;

            for (Transaction t : transactions) {
                if (t.getType().equalsIgnoreCase("income")) {
                    income += t.getAmount();
                    balance += t.getAmount();
                } else if (t.getType().equalsIgnoreCase("expense")) {
                    expense += t.getAmount();
                    balance -= t.getAmount();
                }
            }

            textBalance.setText(String.format("Поточний баланс: ₴%.2f", balance));
            textIncome.setText(String.format("Загальні доходи: ₴%.2f", income));
            textExpenses.setText(String.format("Загальні витрати: ₴%.2f",
expense));
        });

        LinearLayout chartContainer = view.findViewById(R.id.chartContainer);

        String[] days = {"Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Нд"};


        for (String day : days) {

            LinearLayout row = new LinearLayout(getContext());
            row.setOrientation(LinearLayout.HORIZONTAL);
            row.setLayoutParams(new LinearLayout.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
            row.setPadding(0, 8, 0, 8);

            TextView dayLabel = new TextView(getContext());
            dayLabel.setText(day);
            dayLabel.setTextSize(14);
            dayLabel.setWidth(80);

            ProgressBar bar = new ProgressBar(getContext(), null,
android.R.attr.progressBarStyleHorizontal);
            bar.setMax(500);
            bar.setProgress(amount);
            bar.setLayoutParams(new LinearLayout.LayoutParams(0, 30, 1f));
            bar.setProgressDrawable(ContextCompat.getDrawable(getContext(),
R.drawable.custom_progressbar));

            TextView amountLabel = new TextView(getContext());
            amountLabel.setText(String.format("₴%d", amount));
            amountLabel.setTextSize(14);
            amountLabel.setPadding(12, 0, 0, 0);

            row.addView(dayLabel);
            row.addView(bar);
            row.addView(amountLabel);
```

```
            chartContainer.addView(row);
        }

        return view;
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        buttonAddTransaction =
requireActivity().findViewById(R.id.buttonAddTransaction);
        buttonAddTransaction.setOnClickListener(v -> {
            // TODO: перехід на екран додавання транзакції
        });
    }
}
```

Файл «MainActivity.java»

```
package com.financemanager;

import android.content.Intent;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import android.util.Log;

import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.floatingactionbutton.FloatingActionButton;

public class MainActivity extends AppCompatActivity {

    private BottomNavigationView bottomNavigationView;
    private FloatingActionButton fab;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        bottomNavigationView = findViewById(R.id.bottom_navigation);
        fab = findViewById(R.id.buttonAddTransaction);

        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, new HomeFragment())
                    .commit();
            fab.show();
        }


        fab.setOnClickListener(v -> {
            Log.d("MainActivity", "FAB натиснуто");
            new AddTransactionDialogFragment().show(
                    getSupportFragmentManager(), "AddTransactionDialog"
            );
        });
```

```
        fab.setOnClickListener(v -> {
            Log.d("MainActivity", "FAB натиснуто");
            Intent intent = new Intent(MainActivity.this,
AddTransactionActivity.class);
            startActivity(intent);
        });

        bottomNavigationView.setOnItemSelectedListener(item -> {
            Fragment selectedFragment = null;
            int id = item.getItemId();

            if (id == R.id.nav_home) {
                selectedFragment = new HomeFragment();
                fab.show();
            } else if (id == R.id.nav_list) {
                selectedFragment = new TransactionListFragment();
                fab.show();
            } else if (id == R.id.nav_converter) {
                fab.hide();
            }

            if (selectedFragment != null) {
                getSupportFragmentManager().beginTransaction()
                        .replace(R.id.fragment_container, selectedFragment)
                        .commit();
                return true;
            }
            return false;
        });
    }
}
```

Файл «Transaction.java»

```
package com.financemanager;

public class Transaction {
    private int id;
    private String type;
    private String category;
    private double amount;
    private String date;
    private String note;

    public Transaction(int id, String type, String category, double amount, String
date, String note) {
        this.id = id;
        this.type = type;
        this.category = category;
        this.amount = amount;
        this.date = date;
        this.note = note;
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getType() { return type; }
    public void setType(String type) { this.type = type; }
```

```
    public String getCategory() { return category; }
    public void setCategory(String category) { this.category = category; }

    public double getAmount() { return amount; }
    public void setAmount(double amount) { this.amount = amount; }

    public String getDate() { return date; }
    public void setDate(String date) { this.date = date; }

    public String getNote() { return note; }
    public void setNote(String note) { this.note = note; }
}
```

Файл «TransactionAdapter.java»

```
package com.financemanager;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
import android.util.Log;

public class TransactionAdapter extends
RecyclerView.Adapter<TransactionAdapter.TransactionViewHolder> {

    private List<Transaction> transactions;

    public void setTransactions(List<Transaction> transactions) {
        this.transactions = transactions;
        Log.d("Adapter", "Transactions set: " + (transactions != null ?
transactions.size() : 0));
        notifyDataSetChanged();
    }


    @NonNull
    @Override
    public TransactionViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_transaction, parent, false);
        return new TransactionViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull TransactionViewHolder holder, int
position) {
        Transaction transaction = transactions.get(position);
        holder.textViewCategory.setText(transaction.getCategory());
        holder.textViewAmount.setText(String.valueOf(transaction.getAmount()));
        holder.textViewDate.setText(transaction.getDate());
        holder.textViewType.setText(transaction.getType());
    }

    @Override
```

```
    public int getItemCount() {
        return transactions != null ? transactions.size() : 0;
    }

    public static class TransactionViewHolder extends RecyclerView.ViewHolder {
        TextView textViewCategory, textViewAmount, textViewDate, textViewType;

        public TransactionViewHolder(@NonNull View itemView) {
            super(itemView);
            textViewCategory = itemView.findViewById(R.id.textViewCategory);
            textViewAmount = itemView.findViewById(R.id.textViewAmount);
            textViewDate = itemView.findViewById(R.id.textViewDate);
            textViewType = itemView.findViewById(R.id.textViewType);
        }
    }
}
```

Файл «TransactionDao.java»

```
package com.financemanager;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.content.ContentValues;
import android.database.Cursor;
import java.util.List;
import java.util.ArrayList;
import android.util.Log;


public class TransactionDao {
    private SQLiteDatabase db;

    public TransactionDao(Context context) {
        TransactionDbHelper dbHelper = new TransactionDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    public void insertTransaction(Transaction transaction) {
        ContentValues values = new ContentValues();
        values.put("type", transaction.getType());
        values.put("category", transaction.getCategory());
        values.put("amount", transaction.getAmount());
        values.put("date", transaction.getDate());
        values.put("note", transaction.getNote());

        db.insert(TransactionDbHelper.TABLE_TRANSACTIONS, null, values);
    }

    public List<Transaction> getAllTransactions() {
        List<Transaction> transactions = new ArrayList<>();

        Cursor cursor = db.query(TransactionDbHelper.TABLE_TRANSACTIONS,
                null, null, null, null, null, "date DESC");

        if (cursor.moveToFirst()) {
            do {
                int id = cursor.getInt(cursor.getColumnIndexOrThrow("id"));
                String type =
cursor.getString(cursor.getColumnIndexOrThrow("type"));
```

```
                String category =
cursor.getString(cursor.getColumnIndexOrThrow("category"));
                double amount =
cursor.getDouble(cursor.getColumnIndexOrThrow("amount"));
                String date =
cursor.getString(cursor.getColumnIndexOrThrow("date"));
                String note =
cursor.getString(cursor.getColumnIndexOrThrow("note"));

                Transaction transaction = new Transaction(id, type, category,
amount, date, note);
                transactions.add(transaction);

                Log.d("DAO", "Loaded transaction: " + transaction.getCategory() +
" " + transaction.getAmount());
            } while (cursor.moveToNext());
        } else {
            Log.d("DAO", "No transactions found in DB");
        }

        cursor.close();
        return transactions;
    }
}
```

Файл «TransactionDbHelper.java»

```
package com.financemanager;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class TransactionDbHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "finance.db";
    private static final int DATABASE_VERSION = 1;

    public static final String TABLE_TRANSACTIONS = "transactions";

    public TransactionDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String SQL_CREATE_TABLE = "CREATE TABLE " + TABLE_TRANSACTIONS + " ("
                + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + "type TEXT NOT NULL, "
                + "category TEXT NOT NULL, "
                + "amount REAL NOT NULL, "
                + "date TEXT NOT NULL, "
                + "note TEXT)";
        db.execSQL(SQL_CREATE_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Спрощене оновлення
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_TRANSACTIONS);
```

```
        onCreate(db);
    }
}
```

Файл «TransactionListFragment.java»

```
package com.financemanager;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.annotation.NonNull;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.widget.TextView;

import com.google.android.material.floatingactionbutton.FloatingActionButton;

public class TransactionListFragment extends Fragment {

    private TransactionViewModel transactionViewModel;
    private TransactionAdapter adapter;

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_transaction_list,
container, false);

        RecyclerView recyclerView =
view.findViewById(R.id.recyclerViewTransactions);
        TextView textNoTransactions = view.findViewById(R.id.textNoTransactions);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        adapter = new TransactionAdapter();
        recyclerView.setAdapter(adapter);

        transactionViewModel = new
ViewModelProvider(this).get(TransactionViewModel.class);
        transactionViewModel.loadTransactions(requireContext());

        transactionViewModel.getTransactionList().observe(getViewLifecycleOwner(),
transactions -> {
            adapter.setTransactions(transactions);
            if (transactions == null || transactions.isEmpty()) {
                textNoTransactions.setVisibility(View.VISIBLE);
                recyclerView.setVisibility(View.GONE);
            } else {
                textNoTransactions.setVisibility(View.GONE);
                recyclerView.setVisibility(View.VISIBLE);
            }
        });

        FloatingActionButton fab =
view.findViewById(R.id.buttonAddTransactionFromList);
```

```
        ViewCompat.setOnApplyWindowInsetsListener(fab, (v, insets) -> {
            ViewGroup.MarginLayoutParams params = (ViewGroup.MarginLayoutParams)
v.getLayoutParams();
            int bottomInset =
insets.getInsets(WindowInsetsCompat.Type.systemBars()).bottom;
            params.bottomMargin = bottomInset + 16; // 16dp додатковий відступ
            v.setLayoutParams(params);
            return insets;
        });

        fab.setOnClickListener(v -> {
            requireActivity().getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, new
AddTransactionFragment())
                    .addToBackStack(null)
                    .commit();
        });

        return view;
    }
}
```

Файл «TransactionViewModel.java»

```
package com.financemanager;

import android.content.Context;
import androidx.lifecycle.ViewModel;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import java.util.List;

public class TransactionViewModel extends ViewModel {
    private final MutableLiveData<List<Transaction>> transactionList = new
MutableLiveData<>();

    public LiveData<List<Transaction>> getTransactionList() {
        return transactionList;
    }

    public void loadTransactions(Context context) {
        TransactionDao dao = new TransactionDao(context);
        transactionList.setValue(dao.getAllTransactions());
    }

    public void addTransaction(Context context, Transaction transaction) {
        TransactionDao dao = new TransactionDao(context);
        dao.insertTransaction(transaction);
        loadTransactions(context);
    }
}
```

ДОДАТОК В – Диск із кваліфікаційною роботою бакалавра