Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ УПРАВЛІННЯ ФІНАНСАМИ З ВИКОРИСТАННЯМ GRADLE KOTLIN DSL

DEVELOPMENT OF A MOBILE APPLICATION FOR FINANCE MANAGEMENT USING GRADLE KOTLIN DSL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Марків Владислав Ігорович

Керівник:
Повстяна Юлія Славомирівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Марківу Владиславу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка мобільного додатку для управління фінансами з використанням Gradle Kotlin DSL».*
Керівник роботи: *доцент Повстяна Ю. С.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Kotlin, Gradle Kotlin DSL, Android Studio, Jetpack Compose, Room, Hilt, MVVM-архітектура, аналіз предметної області, тестування, документування.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *проаналізувати наявні фінансові додатки, спроєктувати архітектуру з базою даних, налаштувати проєкт через Gradle Kotlin DSL, реалізувати основний функціонал, створити сучасний UX/UI, виконати тестування.*
5. Перелік графічного матеріалу: *12 рисунків, 1 додаток.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Повстяна Ю. С.*		
Специфікація вимог до розробленої системи	*Повстяна Ю. С.*		
Розробка об'єкта проектування	*Повстяна Ю. С.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,29%		
Академічна доброчесність	*Повстяна Ю. С.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	виконано

Здобувач вищої освіти _____ Владислав МАРКІВ

Керівник кваліфікаційної роботи _____ Юлія ПОВСТЯНА

АНОТАЦІЯ

Марків В. І. Розробка мобільного додатку для управління фінансами з використанням Gradle Kotlin DSL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі виконано огляд мобільних додатків для управління фінансами, проаналізовано їхні переваги та недоліки, розглянуто патентні аналоги, а також сформульовано мету, завдання, об’єкт і предмет дослідження. У другому розділі визначено функціональні та нефункціональні вимоги до системи, обґрунтовано вибір інструментів і технологій, зокрема Kotlin, Gradle Kotlin DSL, Room та Jetpack Compose. У третьому розділі описано процес розробки додатку FinStat, реалізацію архітектури, основного функціоналу, локальної бази даних, а також тестування системи з використанням емуляторів та інструментальних засобів Android Studio. У висновках підсумовано результати розробки, досягнення поставлених цілей та окреслено напрями подальшого вдосконалення додатку.

Ключові слова: Kotlin, Gradle Kotlin DSL, мобільний додаток, управління фінансами, Android, Room, Jetpack Compose.

ABSTRACT

Markiv V. Development of a Mobile Application for Finance Management Using Gradle Kotlin DSL. Manuscript. Bachelor's Qualification Thesis in the Educational Program "Software Engineering", Specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter provides an overview of mobile applications for finance management, analyzes their advantages and disadvantages, examines relevant patents, and defines the aim, objectives, object, and subject of the research.

The second chapter outlines functional and non-functional system requirements and justifies the choice of tools and technologies, including Kotlin, Gradle Kotlin DSL, Room, and Jetpack Compose.

The third chapter describes the development process of the FinStat mobile application, including the implementation of architecture, core functionality, a local database, and system testing using emulators and Android Studio tools.

The conclusions summarize the results of the development, the achievement of the stated objectives, and suggest directions for further improvement of the application.

Keywords: Kotlin, Gradle Kotlin DSL, mobile application, finance management, Android, Room, Jetpack Compose.

ЗМІСТ

ВСТУП

Актуальність теми. Фінансова грамотність та ефективне управління грошима стають все більш важливими в сучасному цифровому суспільстві. Розвиток мобільних технологій надав користувачам доступ до цілого ряду інструментів, які допомагають їм контролювати свої фінансові транзакції, аналізувати свої витрати, планувати бюджет і досягати своїх фінансових цілей. Мобільні додатки стали невід'ємною частиною цього процесу, забезпечуючи швидкий і легкий доступ до фінансових даних у будь-який час і в будь-якому місці.

Хоча існує низка мобільних додатків для управління фінансами, таких як наприклад Mint, YNAB, PocketGuard і Toshl Finance, але варто підмітити що у даних додатках існує низка проблем, які знижують їхню ефективність та зручність використання. Основними недоліками таких рішень є:

– обмежений функціонал без підписки або додаткових витрат;

– відсутність можливості гнучкого налаштування під потреби користувача;

– недостатня безпека персональних фінансових даних;

– складний або перевантажений інтерфейс, що ускладнює взаємодію з додатком.

Ці проблеми зумовлюють необхідність розробки нового рішення, яке буде враховувати сучасні вимоги до функціональності та продуктивності мобільного додатка для фінансового управління.

Метою кваліфікаційної роботи бакалавра є розробка мобільного додатку «FinStat» для управління фінансами з Gradle Kotlin DSL, яка забезпечує гнучке та ефективне управління проектами, спрощену конфігурацію залежностей, продуктивність та ефективність, легкість вдосконалення та конфігурації. Додаток надає користувачам можливість зручно вводити фінансові операції, додавати різноманітні рахунки, аналізувати доходи та витрати, генерувати детальні фінансові звіти та планувати бюджети.

Об’єктом даної кваліфікаційної роботи бакалавра є мобільний додаток, який допоможе користувачам ефективно керувати своїми домашніми фінансами, отримувати аналітику про свої витрати та доходи, а також покращити фінансову дисципліну.

Предметом даної кваліфікаційної роботи бакалавра є методи, підходи та програмні засоби розробки мобільних додатків для управління фінансами з використанням Gradle Kotlin DSL, а також функціональні, архітектурні та інтерфейсні рішення, що забезпечують зручність користування, надійність обліку фінансових операцій, ефективність аналізу даних та планування бюджету.

Для досягнення мети в роботі будуть розглянуті такі завдання:

– провести аналіз існуючих фінансових додатків, визначити їхні переваги та недоліки для обґрунтування необхідності розробки нового рішення;

– спроєктувати архітектуру додатку, включаючи основні модулі, базу даних та механізми обробки фінансових даних;

– використати декларативний стиль конфігурації Gradle Kotlin DSL для ефективного налаштування проєкту, що забезпечить зручність управління залежностями та конфігурацією;

– реалізувати основні функції, а саме додавання, редагування та перегляд фінансових операцій, категоризація доходів і витрат, візуалізація фінансових даних у вигляді статистики, планування бюджету та контроль фінансових цілей;

– розробити зручний та інтуїтивно зрозумілий UX/UI дизайн, який відповідатиме сучасним стандартам;

– виконати тестування додатку, включаючи функціональне та інтеграційне тестування для забезпечення стабільної роботи.

Таким чином, дана робота спрямована на створення сучасного, гнучкого та безпечного фінансового додатку, який буде корисним інструментом для всіх користувачів.

РОЗДІЛ 1
АНАЛІЗ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ УПРАВЛІННЯ ФІНАНСАМИ З ВИКОРИСТАННЯМ GRADLE KOTLIN DSL І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Оскільки цифрові технології відіграють важливу роль в управлінні фінансами, існує потреба в розробці корисних та ефективних мобільних додатків для управління доходами та витратами, бюджетування та аналізу грошових потоків. Такі додатки допомагають користувачам стежити за своїми фінансами, управляти своїми витратами та досягати своїх фінансових цілей. Одним з ключових аспектів таких додатків є використання сучасних технологій для забезпечення високої продуктивності, безпеки та зручності використання. Однією з таких технологій є Gradle Kotlin DSL, яка використовується для налаштування та управління залежностями в процесі розробки мобільних додатків.

Зі стрімким розвитком мобільних технологій мобільні додатки стали основним інструментом для управління фінансами, забезпечуючи швидкий доступ до фінансових даних у будь-який час і в будь-якому місці. Інтерфейс таких додатків повинен бути інтуїтивно зрозумілим і зручним, що дозволяє користувачам легко адаптуватися до змін у фінансових потоках. Для синхронізації та інтеграції фінансових даних із мобільними додатками активно використовуються веб-технології, такі як REST API та хмарні сервіси.

Аналіз фінансових даних є важливою складовою сучасної фінансової практики. Статистичні методи та алгоритми прогнозування можна використовувати для визначення фінансових тенденцій, виявлення типових витрат і формування рекомендацій щодо бюджетування. У фінансових мобільних додатках історичні дані включають інформацію про транзакції, категорії витрат, баланси рахунків і часові шаблони витрачання коштів. Ці дані дозволяють аналізувати поведінку користувача, формувати прогнози та

надавати персоналізовані фінансові поради. У [1] показано як користувачі Monobank можуть переглядати статистику витрат, накопичень, переказів та інших фінансових операцій через мобільний додаток. Це демонструє, як історичні фінансові дані використовуються для аналізу та управління особистими фінансами в українських фінансових сервісах.

Наукові дослідження в галузі управління фінансами та мобільних застосунків підкреслюють важливість автоматизації фінансових процесів і впровадження аналітичних інструментів для користувачів. Зокрема, у статті [2] автори вказують, що мобільні фінансові застосунки, які аналізують фінансові звички та пропонують персоналізовані поради для оптимізації витрат і заощаджень, значно покращують фінансову обізнаність користувачів. Автори наголошують на важливості інтеграції таких систем у фінансові стратегії для підвищення ефективності управління особистим бюджетом.

Робота Соболєва-Терещенко О. та Жарнікова В. аналізує сучасні методи захисту персональних фінансових даних, що є ключовим аспектом розробки FinStat. Автори зазначають, що впровадження національних стратегій цифрової фінансової грамотності, включаючи захист персональних даних, є необхідним кроком для забезпечення безпеки користувачів у цифровому фінансовому середовищі [3].

У дослідженні [4] пропонується класифікація цифрових рішень для особистих фінансів: від простих бюджетних трекерів до складних аналітичних платформ із функціями прогнозування. Автор підкреслює, що системи, які забезпечують зворотний зв’язок у реальному часі (наприклад, повідомлення про перевищення бюджету), мають більший вплив на фінансову дисципліну користувача.

У межах підготовки до розробки мобільного застосунку для управління особистими фінансами було проведено патентний пошук із метою виявлення існуючих аналогів та оцінки рівня унікальності запропонованого рішення. Пошук здійснювався через відкриті міжнародні бази даних, серед яких Google Patents, Espacenet та United States Patent and Trademark Office.

У процесі пошуку виявлено декілька релевантних патентів. Зокрема, патент US20230267484A1, зареєстрований компанією Capital One Services, LLC, описує мобільний застосунок, який використовує машинне навчання для аналізу транзакцій користувача та надання персоналізованих фінансових порад [5].

Інший приклад – патент US12106383B2, також поданий Capital One Services, LLC [6]. Він стосується системи, яка автоматично виявляє невідповідності між електронними повідомленнями про повернення коштів та фактичними сумами повернення, відображеними в платіжних базах даних. Описана система використовує обробку природної мови для аналізу комунікацій.

Ще одним релевантним документом є патент US11669817B2, заявлений компанією Capital One Services, LLC [7]. У цьому випадку мова йде про систему, яка дозволяє користувачам керувати підписками на різні послуги через мобільний застосунок. Система інтегрується з фінансовими установами для надання користувачам можливості переглядати, змінювати або скасовувати підписки.

Аналіз виявлених патентів свідчить про те, що існуючі рішення частково перегукуються з окремими функціональними можливостями майбутнього продукту, однак не відтворюють його концепцію в повному обсязі. Оскільки жоден із виявлених патентів не охоплює повністю функціональні можливості запропонованого застосунку, це свідчить про наявність вільної ніші для впровадження нового рішення. Розроблювальний мобільний додаток FinStat поєднує в собі елементи фінансового моніторингу та персоналізованого аналізу, що забезпечує зручність, гнучкість і більшу цінність для користувача порівняно з описаними системами.

Хмарні обчислення відіграють важливу роль у сучасних фінансових додатках, дозволяючи зберігати та синхронізувати дані на різних пристроях. Використання хмарних сховищ, таких як Google Firebase та Amazon Web Services, забезпечує надійність та безпеку фінансової інформації, а також дозволяє отримати доступ до даних з будь-якого пристрою. Крім того, хмарні

технології сприяють швидкому масштабуванню додатків та інтеграції з іншими сервісами.

Персоналізація відіграє важливу роль у сучасній фінансовій практиці. Це включає можливість персоналізувати категорії витрат і доходів, створювати особисті фінансові цілі, використовувати різні валюти і синхронізувати з іншими фінансовими продуктами. Це забезпечує найбільш зручні та корисні інструменти, пристосовані до конкретних фінансових звичок і потреб користувача.

У межах аналізу існуючих мобільних рішень для управління особистими фінансами було розглянуто низку популярних застосунків, які вже зарекомендували себе на ринку.

Одним із найбільш відомих є мобільний застосунок Mint Fin (рис. 1.1). Його популярність пояснюється передусім тим, що сервіс надає безкоштовні інструменти для інтеграції з банківськими рахунками, що дає змогу автоматично відстежувати транзакції без потреби ручного введення даних.

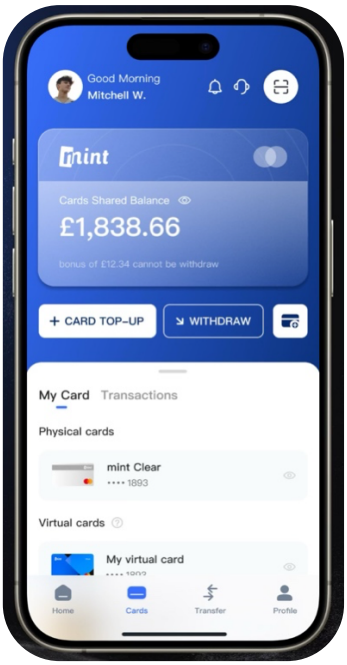

Рисунок 1.1 – Інтерфейс додатку Mint Fin [8]

Крім того, система підтримує автоматичну класифікацію витрат, що полегшує користувачам аналіз своїх фінансових потоків. Водночас застосунок має низку недоліків, які впливають на загальне враження користувачів. Зокрема, можливості кастомізації є досить обмеженими, а саме користувачі не можуть гнучко налаштовувати категорії витрат відповідно до своїх потреб. Додатково у безкоштовній версії присутня реклама, яка може відволікати та знижувати комфорт роботи із застосунком. Однією з найбільших проблем залишається обов'язкова прив'язка до банківського рахунку, що підвищує ризики для безпеки персональних даних, оскільки централізоване зберігання великої кількості фінансової інформації часто стає об'єктом атак зловмисників.

Іншим прикладом є мобільний додаток YNAB (You Need a Budget), який представлено на рисунку 1.2, що позиціонується як потужний інструмент для управління особистим бюджетом.

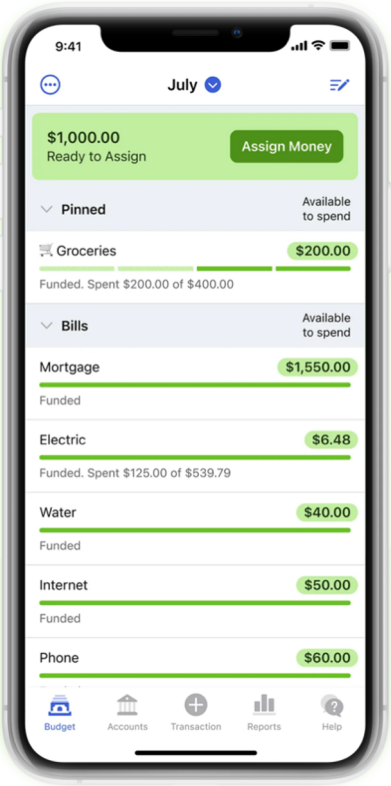

Рисунок 1.2 – Інтерфейс додатку YNAB [9]

Його основною перевагою є систематичний підхід до контролю витрат застосунок допомагає користувачам планувати кожен долар своїх доходів і таким чином формувати здорові фінансові звички. Стратегія YNAB орієнтована на активне управління фінансами, а не пасивне спостереження за балансом рахунку. Однак, незважаючи на ефективність запропонованої методології, застосунок має певні суттєві недоліки. По-перше, він є платним сервісом, що може бути бар'єром для частини користувачів, особливо тих, хто тільки починає працювати над власним бюджетуванням. По-друге, значну частину даних потрібно вводити вручну, оскільки застосунок не має вбудованої автоматичної синхронізації з банківськими рахунками без використання сторонніх сервісів, що ускладнює його використання та вимагає додаткових часових витрат.

Ще одним цікавим прикладом є мобільний застосунок PocketGuard, який орієнтований на спрощення процесу управління особистими фінансами (рис. 1.3). Його головною перевагою є надзвичайно проста та інтуїтивно зрозуміла система обліку витрат. PocketGuard пропонує користувачам у реальному часі бачити, скільки коштів залишилося після врахування всіх необхідних платежів та накопичень, що робить процес фінансового планування більш доступним.

Рисунок 1.3 – Інтерфейс додатку PocketGuard [10]

Проте функціональні можливості цього застосунку є дещо обмеженими. Наприклад, користувачі мають мінімальні можливості для налаштування категорій витрат, що знижує гнучкість сервісу і може не задовольняти потреби тих, хто прагне вести детальний фінансовий облік.

Також було проаналізовано мобільний застосунок Toshl Finance, який вигідно вирізняється серед конкурентів завдяки своїй мультивалютній підтримці, можливості імпорту транзакцій та зручному інтерфейсу, зрозумілому навіть для новачків (рис. 1.4). Використання кількох валют одночасно є особливо корисним для користувачів, які мають доходи або витрати у різних валютах, що часто актуально для фрилансерів або тих, хто подорожує.

Рисунок 1.4 – Інтерфейс додатку Toshl Finance [11]

Однак і цей застосунок не позбавлений недоліків. Безкоштовна версія має значні функціональні обмеження, і багато корисних функцій, таких як автоматичне імпортування банківських даних або детальні аналітичні звіти, доступні лише у платній підписці. Це змушує користувачів або погоджуватися на обмежений функціонал, або здійснювати додаткові витрати.

Підсумовуючи вище сказане, можна виділити як переваги, так і недоліки сучасних фінансових мобільних застосунків. До основних переваг належить автоматичне отримання транзакцій із банківських рахунків, що значно спрощує ведення фінансового обліку. Також користувачам пропонується гнучкість у налаштуванні категорій витрат і формуванні звітів, а зручний інтерфейс і доступність через мобільні пристрої роблять такі додатки привабливими для широкого кола користувачів. Водночас, серед недоліків варто відзначити обмежену можливість кастомізації, що не дозволяє повністю адаптувати застосунок до індивідуальних потреб. Крім того, деякі сервіси є платними або мають функціональні обмеження в безкоштовних версіях. Суттєвими залишаються й питання безпеки, пов’язані зі зберіганням чутливих даних і інтеграцією з банківськими рахунками. Також спостерігається обмежена підтримка нових функцій або інтеграцій з іншими цифровими сервісами.

З огляду на недоліки існуючих рішень і зростаючу потребу в більш кастомізованих та безпечних фінансових інструментах, створення нового мобільного застосунку «FinStat» є цілком доцільним і своєчасним. Однією з ключових переваг цього проєкту є висока гнучкість розробки, використання Gradle Kotlin DSL не лише спрощує процес конфігурації, але й дозволяє створювати модульну архітектуру, яка легко масштабуються й адаптується до змінних потреб. Завдяки цьому розробники зможуть швидко впроваджувати нові функції, що підвищить конкурентоспроможність продукту.

Ще однією важливою перевагою є можливість повного налаштування функціоналу під індивідуальні потреби користувача. На відміну від багатьох існуючих рішень, FinStat орієнтований на персоналізацію – користувач зможе самостійно визначати, які функції й аналітичні інструменти йому потрібні, та створювати власні сценарії використання додатку. Це особливо актуально для людей з різними фінансовими звичками або цілями, які потребують гнучких рішень.

Окрема увага приділяється безпеці користувацьких даних. У FinStat буде реалізовано можливість локального збереження фінансової інформації без

передачі її на сторонні сервери. Це дозволить забезпечити високий рівень конфіденційності й захисту особистої інформації, що є особливо важливим у фінансовому сегменті.

Крім того, застосунок буде оптимізований для стабільної роботи навіть на пристроях із невеликим обсягом ресурсів. Використання Kotlin DSL дозволить досягти високої продуктивності, що забезпечить швидку реакцію інтерфейсу, мінімальне споживання пам’яті та енергоефективність. Таким чином це поєднуватиме технологічну досконалість із орієнтацією на користувача, пропонуючи сучасне рішення для ефективного управління особистими фінансами.

Таким чином, створення FinStat є логічним кроком для вдосконалення існуючих фінансових рішень, оскільки FinStat може усунути недоліки інших додатків і надати користувачам вищий рівень кастомізації, більшу безпеку та ефективне управління фінансами.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи є розробка мобільного додатку FinStat для ефективного управління фінансами з використанням Gradle Kotlin DSL. Додаток має допомагати користувачам зручно вести облік доходів та витрат, аналізувати фінансові показники, планувати бюджети та досягати фінансових цілей. Застосування Gradle Kotlin DSL забезпечить гнучке управління залежностями, пришвидшить розробку та спростить впровадження оновлень.

Для досягнення цієї мети необхідно виконати такі завдання:

– провести аналіз існуючих фінансових додатків, визначити їхні переваги та недоліки для обґрунтування необхідності розробки нового рішення;

– спроєктувати архітектуру додатку, включаючи основні модулі, базу даних та механізми обробки фінансових даних;

– використати декларативний стиль конфігурації Gradle Kotlin DSL для ефективного налаштування проєкту, що забезпечить зручність управління залежностями та конфігурацією;

– реалізувати основні функції, а саме додавання, редагування та перегляд фінансових операцій, категоризація доходів і витрат, візуалізація фінансових даних у вигляді статистики, планування бюджету та контроль фінансових цілей;

– розробити зручний та інтуїтивно зрозумілий UX/UI дизайн, який відповідатиме сучасним стандартам;

– виконати тестування додатку, включаючи функціональне та інтеграційне тестування для забезпечення стабільної роботи.

Реалізація цих завдань дозволить створити FinStat – сучасний мобільний інструмент для управління особистими фінансами, який забезпечуватиме користувачам зручність, швидкість, безпеку та допомагатиме підвищувати фінансову грамотність.

Висновки до розділу 1

У першому розділі було проведено аналіз сучасного стану цифрових рішень для управління особистими фінансами. Розглянуто функціональні можливості, переваги та недоліки популярних мобільних додатків. На основі виявлених обмежень, зокрема недостатньої гнучкості налаштувань, низької безпеки або залежності від підключення до інтернету, обґрунтовано доцільність створення нового мобільного рішення. Також виконано патентний аналіз, який засвідчив, що існуючі системи лише частково реалізують необхідний функціонал, що відкриває можливість для впровадження інноваційного застосунку. У підсумку сформульовано мету, завдання, об’єкт і предмет дослідження, які стали основою для подальшої розробки FinStat.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного

Розробка мобільного додатку FinStat здійснювалася на основі ітеративної моделі, що була обрана через її гнучкість, можливість врахування змінних вимог і послідовне вдосконалення функціоналу. Такий підхід дав змогу розбити процес створення системи на кілька коротких циклів, кожен з яких завершувався робочим прототипом і дозволяв отримати зворотний зв’язок від потенційних користувачів. Завдяки цьому забезпечувалося краще розуміння потреб цільової аудиторії, підвищувалася якість реалізації та спрощувалося виявлення недоліків на ранніх етапах.

Аналіз вимог до системи проводився із залученням різноманітних методів. Були проведені інтерв’ю з потенційними користувачами, що дозволило виявити типові сценарії використання та основні очікування. Також здійснено порівняльний аналіз існуючих аналогів, зокрема мобільних додатків для ведення особистих фінансів, де вивчалися їхні функціональні можливості, зручність використання, дизайн та зворотний зв’язок від користувачів. Отримані результати дали змогу сформувати чітке уявлення про функціональні пріоритети, визначити ключові функції, які мають бути реалізовані у першій версії додатку.

На основі зібраних даних було сформульовано цілі системи, до яких належить забезпечення зручного інструменту для обліку особистих фінансів, планування витрат, перегляду звітів і візуалізації фінансової інформації. Основна задача додатку – дати користувачеві змогу легко відслідковувати свої доходи та витрати, встановлювати бюджетні обмеження та отримувати сповіщення у разі їх перевищення. При цьому важливим аспектом є забезпечення безпеки персональних фінансових даних.

Функціональність системи передбачає можливість додавання, редагування та видалення транзакцій з прив’язкою до категорій, часу та опису а також

користувач має змогу переглядати статистику. Застосунок працює в офлайн-режимі, з можливістю синхронізації після підключення до інтернету. Дані зберігаються у локальній базі з використанням шифрування, що забезпечує конфіденційність і захист інформації.

Проєктування інтерфейсу буде базуватися на принципах мінімалізму та функціональності. Усі елементи навігації розташовано інтуїтивно зрозуміло, що дозволяє користувачеві швидко виконувати основні дії – додавати нові записи, переглядати статистику, змінювати налаштування. Дизайн екранів розроблений таким чином, щоб максимально зменшити кількість дій, необхідних для досягнення результату, що значно підвищує зручність взаємодії з системою.

Інформаційні потоки в додатку мають охоплювати увесь цикл обробки даних: від моменту введення транзакцій користувачем до їх збереження в локальній базі даних та подальшого аналізу й відображення у вигляді фінансових звітів. Усі основні процеси – додавання, редагування, категоризація та перегляд операцій – реалізовані локально, що забезпечує швидкодію додатку та збереження конфіденційності даних. Система не потребує постійного підключення до мережі, що робить її доступною для використання в автономному режимі. У майбутньому передбачається впровадження хмарної синхронізації, яка дозволить зберігати резервні копії, передавати дані між пристроями та розширити можливості взаємодії.

Щоб представити функціональні можливості системи та взаємодію користувачів із її елементами була розроблена UML-діаграма варіантів використання мобільного додатку FinStat. На діаграмі представлені ключові ролі – розробник, дизайнер, тестувальник і кінцевий користувач – а також їхні дії, зокрема додавання фінансових транзакцій, перегляд балансу, створення звітів, редагування рахунків, аналіз витрат і створення бюджету. Такий підхід дозволив формалізувати вимоги до системи та забезпечити чітке уявлення про функціональність додатку ще на етапі проектування, що стало основою для подальшої реалізації та тестування (рис. 2.1).

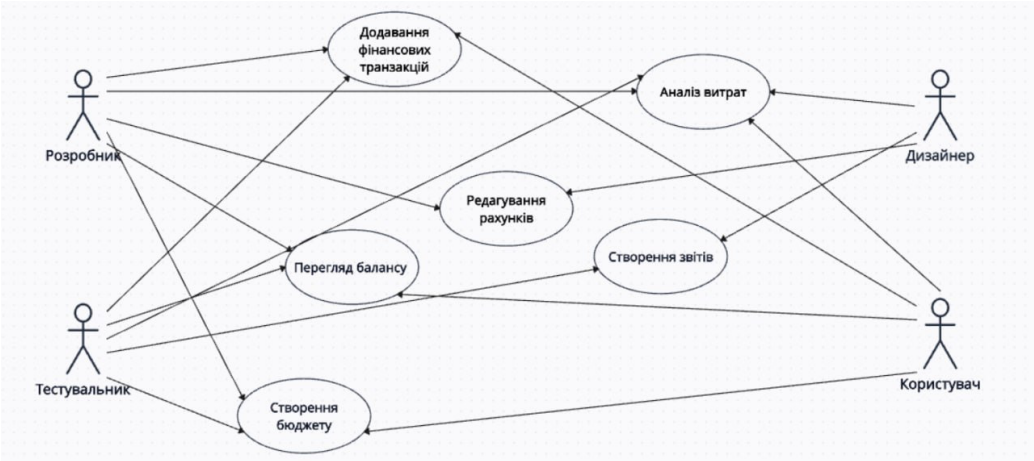

Рисунок 2.1 – UML-діаграма прецедентів

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації проєкту FinStat було обрано набір сучасних технологій, які забезпечують ефективність, надійність та масштабованість мобільного додатку. Центральною мовою розробки є Kotlin, що є офіційно підтримуваною Google мовою для Android-додатків (рис. 2.2). Її переваги полягають у лаконічному синтаксисі, безпечному управлінні нульовими значеннями, повній сумісності з Java і підтримці сучасних підходів до розробки.

Рисунок 2.2 – Логотип Kotlin [12]

Однією з ключових архітектурних рішень стало було вибір використовувати шаблон MVVM (Model-View-ViewModel), який забезпечує чітке розділення логіки представлення, бізнес-логіки та взаємодії з базою даних. Такий підхід підвищує підтримуваність та розширюваність системи, дозволяє

ефективно обробляти дані транзакцій і реагувати на зміни у реальному часі без перевантаження інтерфейсу.

Gradle Kotlin DSL обрано як інструмент для налаштування проєкту та керування залежностями. На відміну від класичного Groovy DSL, Kotlin DSL надає статичну типізацію, покращене автодоповнення в IDE та зменшує кількість помилок на етапі конфігурації. Це дозволяє забезпечити централізоване та зручне управління бібліотеками, модулями та параметрами збірки, що особливо актуально для складних проєктів з великою кількістю компонентів.

Для управління залежностями в системі буде використовуватись Hilt – сучасна бібліотека для впровадження залежностей, інтегрована з Android Jetpack. Вона дозволяє зменшити кількість шаблонного коду та централізовано керувати життєвим циклом компонентів, особливо ViewModel та репозиторіїв.

Для зберігання фінансових даних буде застосовано SQLite у поєднанні з ORM-бібліотекою Room, яка дозволяє працювати з базою через об’єктно-орієнтовану модель, зменшуючи кількість шаблонного коду та підвищуючи безпеку. За допомогою DAO-інтерфейсів реалізується доступ до таблиць транзакцій, категорій і бюджетів. Такий підхід дозволяє гнучко обробляти дані та легко додавати нові функції у майбутньому.

Для інтерфейсу користувача було обрано Jetpack Compose, що дозволяє створювати сучасний, реактивний, декларативний UI. Кожен екран реалізовано як окрема функція-компонент, що динамічно оновлюється при зміні стану ViewModel. Це забезпечує високу продуктивність і чистоту коду.

Міжекранна навігація буде реалізована через Navigation Fragment. Кожен фрагмент відповідає за окрему функціональну частину – перегляд транзакцій, статистику, налаштування тощо. Всі маршрути описано у файлі графу навігації, що централізує керування переходами між екранами.

Для покращення UX при роботі зі списками буде реалізовано підтримку свайп-жестів через бібліотеку RecyclerView SwipeDecorator, що дозволяє редагувати або видаляти записи жестом свайпу з відповідною візуалізацією.

З метою забезпечення стабільної роботи додатку в експлуатаційному середовищі буде впроваджено Firebase Crashlytics, що дозволяє автоматично виявляти збої та надсилати звіти розробникам із повною інформацією про помилку та стан пристрою.

Окрім основних бібліотек, у розробці мобільного додатку FinStat активно використовуватиметься Jetpack-модулі, такі як LiveData, ViewModel та Lifecycle. Завдяки ним буде досягнуто стабільної роботи програми при зміні станів життєвого циклу, а також реалізовано реактивне оновлення інтерфейсу у відповідь на зміни даних. ViewModel дозволяє зберігати стан при зміні конфігурації, тоді як LiveData забезпечує автоматичну синхронізацію між даними та UI без зайвих викликів оновлення вручну.

Під час розробки значна увага має приділятись налагодженню та оптимізації. Зокрема, використовувалися Android Profiler для моніторингу продуктивності, Logcat для перегляду системних логів і відстеження подій під час виконання, а також Lint для автоматичної перевірки якості коду, виявлення помилок, дублювань або порушення стилістичних правил.

Для забезпечення надійності додатку будуть реалізовані як юніт-тести з використанням JUnit, так і інструментальні тести для UI. Юніт-тести допомагають перевірити логіку окремих компонентів, зокрема ViewModel та репозиторіїв. UI-тести дозволяють протестувати поведінку інтерфейсу та взаємодію користувача з додатком.

У частині бізнес-логіки буде реалізовано алгоритми категоризації транзакцій, розрахунку залишку бюджету, агрегації витрат і доходів за певні періоди, а також побудови статистики. Це дозволяє користувачам легко аналізувати структуру витрат, визначати основні фінансові статті та контролювати баланс у часі.

Обробка помилок буде реалізована з дотриманням найкращих практик. Вразливі до збоїв операції мають бути захищені блоками «try-catch», а також має бути передбачена fallback-логіка у випадку помилок з'єднання чи збереження даних. Перед виконанням мережевих запитів перевіряється наявність

підключення до інтернету, що дозволяє уникнути збоїв і покращує стабільність роботи додатку.

Висновки до розділу 2

У цьому розділі визначено ключові функціональні та нефункціональні вимоги до FinStat, обґрунтовано вибір технологій та архітектурного підходу. Застосування UML-діаграм дозволило формалізувати структуру системи, а вибрані алгоритми та засоби реалізації забезпечують стабільну, безпечну та продуктивну роботу додатку в реальних умовах користування.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ УПРАВЛІННЯ ФІНАНСАМИ З GRADLE KOTLIN DSL

3.1 Практична реалізація об’єкта проєктування

Розробка мобільного додатку FinStat здійснювалася поетапно із суворим дотриманням принципів сучасної програмної інженерії. Основною метою стало створення зручного, функціонального та продуктивного інструменту для щоденного управління особистими фінансами. Кожен етап розробки був ретельно спланований і реалізований з орієнтацією на кінцевого користувача, враховуючи технічні особливості мобільної платформи Android, а також обрані засоби – мову Kotlin і систему збірки Gradle Kotlin DSL.

Початковим кроком став етап створення проєкту в середовищі Android Studio. На початковому етапі було створено структуру проєкту з модульним поділом: основний модуль «:app», модуль для роботи з даними та базою, окремий модуль для навігації та UI-компонентів (рис. 3.1).

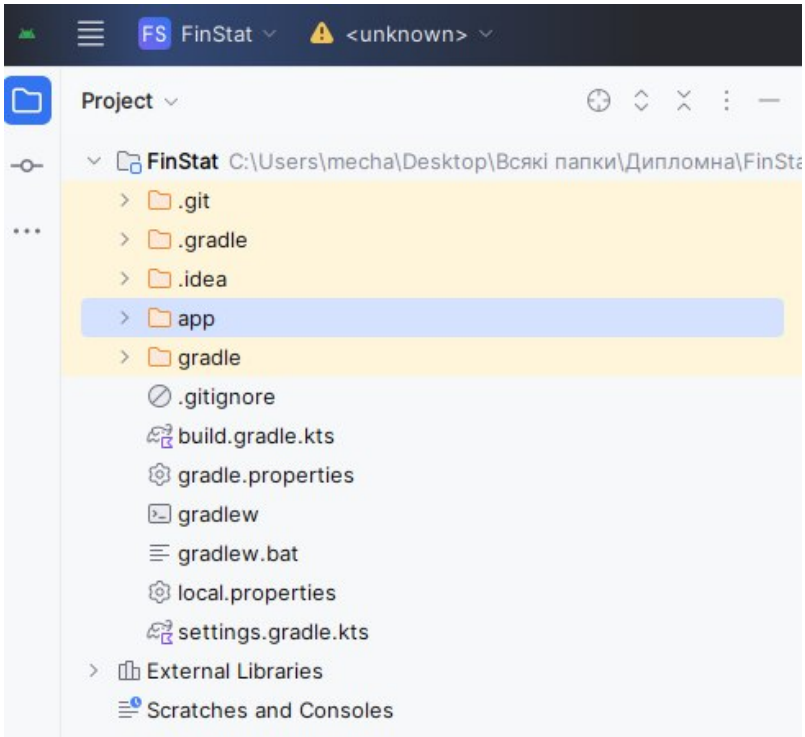

Рисунок 3.1 – Структура проєкту

Після ініціалізації базового проєкту було визначено архітектурну структуру з урахуванням вимог до масштабованості, підтримки модульності та можливості легкого тестування. Було вирішено дотримуватись архітектурного патерну MVVM (Model–View–ViewModel), який дозволяє розділити обов'язки між шарами представлення, бізнес-логіки та обробки даних. У такий спосіб вдалося досягти чистої архітектури та спростити майбутнє розширення функціоналу.

Такий підхід забезпечує масштабованість і зручність підтримки додатку в майбутньому. Було створено файл App.kt, що наслідує Application і позначається анотацією @HiltAndroidApp (ліст. 3.1). Це дозволяє Hilt автоматично згенерувати необхідний код для побудови графа залежностей. Я налаштував глобальний контекст застосунку з використанням Hilt для впровадження залежностей. Я обрав саме Hilt, оскільки він є офіційним інструментом Google, має хорошу документацію та інтегрується з іншими компонентами Jetpack. Його використання дозволяє спростити доступ до бази даних, репозиторіїв та ViewModel у будь-якому фрагменті чи активності.

Лістинг 3.1 – Ініціалізація глобального контексту додатку

```
@HiltAndroidApp
class App : Application() {
    override fun onCreate() {
        super.onCreate()
    }
}
```

кінець лістингу 3.1

Уся логіка розділена за принципами архітектури MVVM (Model–View–ViewModel), що дозволяє ефективно ізолювати бізнес-логіку від інтерфейсу користувача. ViewModel взаємодіє з репозиторієм, який у свою чергу звертається до DAO інтерфейсів бази даних. Усі транзакції, категорії, бюджети, валюти зберігаються у локальній базі даних SQLite, що реалізована за допомогою бібліотеки Room.

Функціональні можливості FinStat включають:

– додавання, редагування та видалення транзакцій;

– категоризацію доходів і витрат;

– перегляд детальної статистики у вигляді звітів;

– встановлення фінансових цілей і бюджетів;

– підтримку різних валют і режиму офлайн-доступу з подальшою синхронізацією.

Основну конфігурацію додатку реалізовано за допомогою Gradle Kotlin DSL, що забезпечило типобезпечність, зручність автодоповнення у IDE та централізоване управління залежностями. Це значно полегшує розгортання проєкту та його масштабування, а також забезпечує простоту у налаштуванні середовища розробки для інших учасників команди.

Інтерфейс додатку реалізовано з використанням технології Jetpack Compose, яка забезпечує декларативний стиль програмування та дозволяє зручно керувати станом елементів (рис. 3.2).

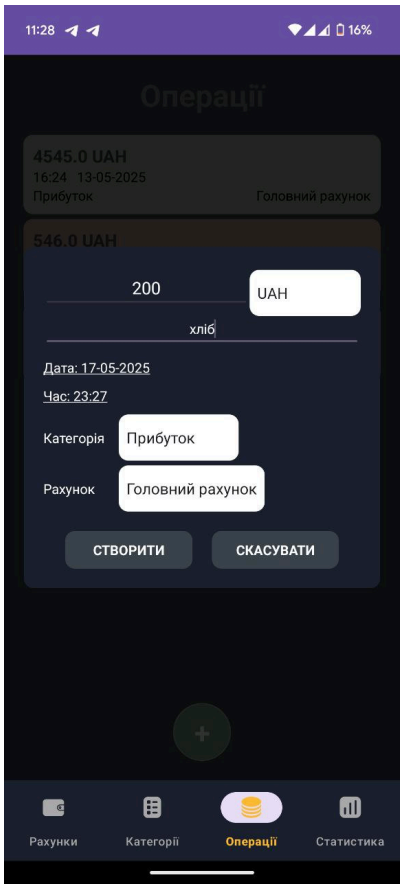

Рисунок 3.2 – Вікно введення транзакції

Наприклад, для введення нової транзакції створено окремий Composable-компонент, де користувач вводить суму, вибирає категорію та дату.

Після реалізації основної архітектури програми, моделі даних і логіки взаємодії з базою даних я перейшов до розробки головного вікна мобільного застосунку – файлу MainActivity.kt. Це центральна точка входу до користувацького інтерфейсу, з якої починається взаємодія користувача з додатком. Вікно головного екрану який зустрічає нас при відкриванні додатку FinStat наведено у рисунку 3.3.

Рисунок 3.3 – Головне вікно програми FinStat

У цьому класі реалізовано побудову навігаційної структури та організацію екранного простору.

Однією з проблем на цьому етапі стало налаштування правильної роботи з NavController при використанні BottomNavigationView. Довелося перепробувати

декілька варіантів – вручну змінювати фрагменти через FragmentTransaction, однак зрештою я зупинився на Navigation Graph, оскільки він краще масштабується та дозволяє легко реалізувати back stack. Приклад побудови навігаційної структури наведено у лістингу 3.2.

Лістинг 3.2 – Побудова навігаційної структури

```
val navHostFragment = supportFragmentManager
    .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
navController = navHostFragment.navController
```

кінець лістингу 3.2

Уся бізнес-логіка додатку зосереджена у ViewModel, що відповідає за обробку подій, зміну станів та взаємодію з джерелами даних. Для організації потоків даних між ViewModel та користувацьким інтерфейсом використовуються реактивні засоби, зокрема LiveData або Flow. Це забезпечує автоматичне оновлення елементів UI у відповідь на зміни даних без необхідності ручного втручання або явного виклику методів оновлення. Завдяки цьому досягається більш динамічна та стабільна взаємодія між даними та інтерфейсом.

Для реалізації додавання нової категорії було створено метод addCategory (ліст. 3.3), який забезпечує унікальність ідентифікатора для кожного нового запису. Після генерації ID та створення об'єкта категорії, інформація зберігається в базу даних через репозиторій. Даний підхід гарантує, що не виникне конфліктів при створенні записів, навіть якщо використовується ручне або псевдовипадкове присвоєння ID. Для зручності UI-компонентів зміни повідомляються через LiveData updateMLD.

Лістинг 3.3 – Додавання нової категорії

```
fun addCategory(message: String) {
    viewModelScope.launch {
        val list = finStatRepository.getAllCategories()
        val listIds = ArrayList<Long>()
        list.forEach {
            listIds.add(it.id)
        }
        var newId = 0L
        while (newId in listIds) {
```

```
            newId = Random.nextInt().toLong()
        }
        val category = Category(newId, message, 1)
        val insert = async {
            finStatRepository.insertCategory(category)
            return@async true
        }
        this@FinStatViewModel.updateMLD.postValue(insert.await())
    }
}
```

кінець лістингу 3.3

У додатку А наведено повний фрагмент коду класу CategoriesFragment, який відповідає за реалізацію функціоналу керування категоріями у мобільному додатку FinStat. Цей фрагмент забезпечує відображення списку категорій, додавання нових, редагування та видалення наявних категорій, а також взаємодію з ViewModel згідно з архітектурним підходом MVVM.

Додавання різноманітних рахунків було реалізовано по схожому принципу, вигляд вікна з додаванням нового рахунку можемо бачити на рисунку 3.4.

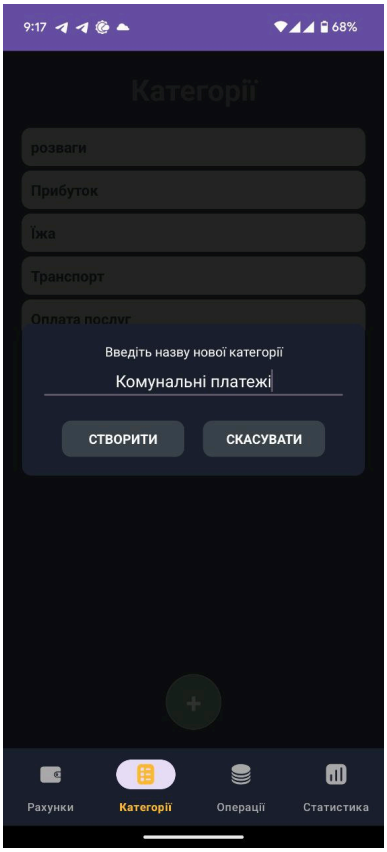

Рисунок 3.4 – Вікно додавання нового рахунку

Клас використовує механізм LiveData для отримання даних, Hilt для впровадження залежностей, а також View Binding для роботи з елементами інтерфейсу. У фрагменті реалізовано функціональність swipe-жестів для швидкого редагування або видалення категорій з використанням компонента ItemTouchHelper. Окрім цього, реалізовано логіку взаємодії з користувачем через кастомні діалогові вікна, які адаптуються під тип дії: додавання, редагування чи підтвердження видалення.

Крім того, залежності між компонентами архітектури керуються за допомогою бібліотеки для впровадження залежностей Hilt, яка інтегрується з Android-архітектурними компонентами та спрощує керування життєвим циклом об'єктів. Це дозволяє зменшити кількість шаблонного коду, полегшує масштабування проєкту та підвищує читаність і підтримуваність коду завдяки чітко визначеній структурі залежностей.

У межах реалізації статистичного модуля було розроблено метод getStatistics. У лістингу 3.4 наведений приклад який забезпечує отримання операцій за обраний період із подальшою фільтрацією за категорією витрат, рахунком та валютою.

Лістинг 3.4 – Метод формування статистики

```
fun getStatistics(
    unixTimeFrom: Long,
    unixTimeTo: Long,
    categoryIndex: Long,
    ordersIndex: Long,
    currencyIndex: Long
) {
    viewModelScope.launch {
        val all = async {
            return@async
finStatRepository.getAllOperationsByParameters(unixTimeFrom, unixTimeTo)
        }
        var list = all.await().sortedByDescending { it.unixtime }

        if(categoryIndex != -1L) {
            list = list.filter { it.categoryId == categoryIndex }
        }
        if(ordersIndex != -1L) {
            list = list.filter { it.orderId == ordersIndex }
        }
        if(currencyIndex != -1L) {
            list = list.filter { it.currencyId == currencyIndex }
        }
```

```
            this@FinStatViewModel.operationsMLD.postValue(list)
        }
    }
```

кінець лістингу 3.4

Цей метод використовує корутини (viewModelScope.launch), що дозволяє виконувати запити до бази даних асинхронно без блокування основного потоку. Після отримання даних, вони сортуються за датою, а потім передаються в LiveData, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних. Вигляд розділу статистики можемо бачити на рисунку 3.5.

Рисунок 3.5 – Розділ з виведенням статистики

У цифрових системах, особливо в базах даних та для обробки статистики, дати часто зберігаються не у вигляді форматованого тексту, а у вигляді UNIX-часу (timestamp) – це кількість секунд, що минули з 00:00:00 UTC 1 січня 1970 року. Метод getUnixTime() який зображений на лістингу 3.5 використовується при створенні запланованої транзакції або нагадування. Наприклад, коли

користувач вибирає дату через календар, вона перетворюється у UNIX-час і зберігається у базі даних для подальшого відображення або аналітики.

Лістинг 3.5 – Отримання UNIX-часу за заданою датою та часом

```
fun getUnixTime(minute: Int, hour: Int, day: Int, month: Int, year: Int): Long {
    calendar.set(Calendar.YEAR, year)
    calendar.set(Calendar.MONTH, month)
    calendar.set(Calendar.DAY_OF_MONTH, day)
    calendar.set(Calendar.HOUR_OF_DAY, hour)
    calendar.set(Calendar.MINUTE, minute)
    return calendar.timeInMillis / 1000
}
```

кінець лістингу 3.5

Це є дуже зручним способом зберігання моменту часу у базі даних, оскільки:

– дозволяє легко виконувати сортування або фільтрацію записів за часовою міткою;

– не залежить від локалізації чи форматів дати;

– сумісний з більшістю інструментів аналітики та баз даних.

Для реалізації фільтрації транзакцій за часовим проміжком було створено метод getAllOperationsByParameters який зображено на лістингу 3.6, який викликає відповідний DAO-запит до бази даних. Завдяки використанню корутин та Dispatchers.IO, забезпечується асинхронна обробка даних без блокування основного потоку. Цей метод є критично важливим для формування статистики у додатку, оскільки дозволяє швидко витягувати тільки релевантні дані на основі тимчасових фільтрів.

Лістинг 3.6 – Отримання операцій за часовим діапазоном

```
suspend fun getAllOperationsByParameters(unixTimeFrom: Long, unixTimeTo: Long):
List<Operation> {
    return withContext(Dispatchers.IO){
       return@withContext finstatDao.getAllOperationsByParameters(unixTimeFrom,
unixTimeTo)
    }
}
```

кінець лістингу 3.6

Окремо варто відзначити налаштування build.gradle.kts, яке зображено на лістингу 3.7 включає всі необхідні плагіни, конфігурацію Compose, Hilt, Room та інші залежності.

Лістинг 3.7 – Конфігурація збірки модуля :app у файлі build.gradle.kts

```
plugins {
    id("com.android.application")
    kotlin("android")
    kotlin("kapt")
    id("dagger.hilt.android.plugin")
}

android {
    compileSdk = 33

    defaultConfig {
        applicationId = "com.finstat.app"
        minSdk = 26
        targetSdk = 33
        versionCode = 1
        versionName = "1.0"
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.4.3"
    }
}

dependencies {
    implementation("androidx.core:core-ktx:1.10.1")
    implementation("androidx.compose.ui:ui:1.4.3")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")
    implementation("androidx.navigation:navigation-compose:2.6.0")
    implementation("com.google.dagger:hilt-android:2.47")
    kapt("com.google.dagger:hilt-android-compiler:2.47")
}
```

кінець лістингу 3.7

Завдяки правильному підключенню необхідних плагінів і залежностей у Gradle, забезпечено стабільну взаємодію всіх компонентів застосунку. Це дозволило спростити конфігурацію проєкту, забезпечити надійність під час збирання та тестування, а також створити умови для подальшого розширення функціональності без ускладнення структури.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

На етапі розробки мобільного застосунку FinStat значну увагу було приділено забезпеченню його стабільної та передбачуваної роботи в різних середовищах. З цією метою було проведено всебічне тестування, а також застосовано методи налагодження з використанням стандартних інструментів Android Studio. Під час перевірки використовувалися як програмні емулятори з різною конфігурацією, так і фізичні пристрої на Android 8.0, 10, 11 та 13.

Забезпечення стабільної та надійної роботи мобільного додатку FinStat здійснювалося через всебічне тестування. Враховуючи специфіку застосунку, було застосовано три основні типи тестування: функціональне ручне тестування, юніт-тести бізнес-логіки, а також інструментальні тести інтерфейсу.

Ручне тестування проводилося на Android-емуляторах та фізичних пристроях із різними характеристиками. Було перевірено:

– створення, редагування та видалення транзакцій;
– валідація введених значень;
– поведінка інтерфейсу в портретному режимі;
– стійкість до повторних швидких дій.

Юніт-тестування реалізоване з використанням JUnit. Для ручного тестування мобільного додатку FinStat було обрано емулятор «BlueStacks це безкоштовна програма для запуску Android-ігор в Windows або Mac OS, має багатомовний інтерфейс, в тому числі й українською мовою. Програма підтримує повноекранний режим, 3D-ігри, а також величезну кількість додатків та має пакет встановлених додатків» [13]. Це рішення базувалося на кількох ключових перевагах цього емулятора. По-перше, BlueStacks підтримує різні версії Android, що дає змогу перевіряти коректність роботи програми на різних платформах і версіях операційної системи. По-друге, BlueStacks характеризується високою швидкодією та стабільністю, що забезпечує швидкий запуск тестів без збоїв і затримок. По-третє, цей емулятор має зручний інтерфейс і дозволяє легко налаштовувати параметри віртуального пристрою, такі як

роздільна здатність екрану та кількість оперативної пам’яті, що важливо для імітації різних умов використання. Також BlueStacks добре інтегрується з інструментами автоматизації тестування, зокрема з JUnit, що спрощує проведення юніт-тестів. Крім того, BlueStacks має широку підтримку користувачів і регулярні оновлення, що забезпечує надійність роботи емулятора. Завдяки цим характеристикам BlueStacks став оптимальним вибором для проведення тестування FinStat. На рисунку 3.6 можемо бачити вигляд програми FinStat на емуляторі BlueStacks.

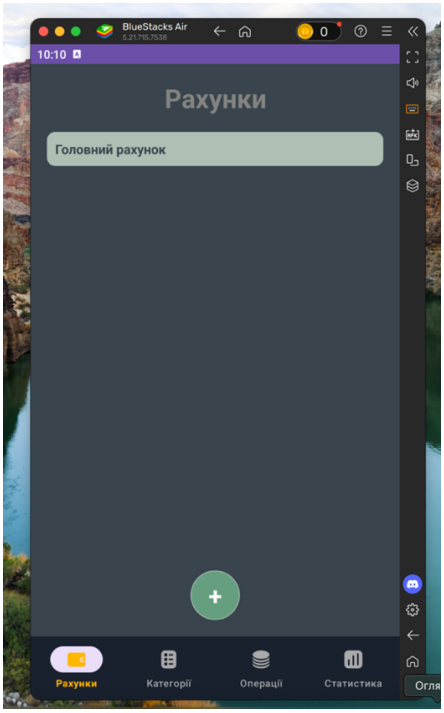

Рисунок 3.6 – Головне вікно програми FinStat на емуляторі BlueStacks

У процесі перевірки функціональності та надійності мобільного застосунку було проведено серію інструментальних тестів із використанням бібліотек AndroidX Test [14] та Espresso [15]. Ці тести дали змогу перевірити коректність навігації між екранами, взаємодію користувача з елементами інтерфейсу (такими як кнопки, текстові поля, списки тощо), а також загальну поведінку UI у різних сценаріях використання. Особлива увага приділялася збереженню введених даних, правильному відображенню фінансової статистики

та стабільній роботі додатку в офлайн-режимі, коли відсутнє підключення до мережі.

У ході тестування було виявлено низку типових помилок, зокрема: некоректне форматування дати при введенні нової транзакції, що могло призводити до неправильного сортування або відображення записів; відсутність перевірки суми транзакції на нульове або від’ємне значення, що дозволяло зберігати некоректні дані; а також неправильне оновлення діаграм після додавання нового запису, внаслідок чого статистика не відображала актуальні значення. Після усунення цих помилок функціональність застосунку була стабілізована, і додаток почав працювати без збоїв у типових сценаріях використання.

Для моніторингу стабільності реліз-версії застосунку та оперативного виявлення критичних помилок було інтегровано сервіс Firebase Crashlytics. Завдяки цьому рішенню стало можливим автоматичне отримання звітів про збої, включно з докладною інформацією про стек викликів, модель пристрою, версію операційної системи та додатку, що значно спрощує діагностику та пришвидшує виправлення помилок на етапі супроводу. Інтеграція Crashlytics також дозволяє відслідковувати частоту виникнення помилок і пріоритезувати їх виправлення відповідно до впливу на користувацький досвід.

3.3 Розробка бази даних

Одним із ключових етапів у створенні мобільного додатку FinStat стала розробка локальної бази даних, яка забезпечує зберігання, обробку та доступ до фінансової інформації користувача. У сучасних мобільних застосунках важливою вимогою є автономність роботи, тобто можливість користуватись функціоналом навіть без підключення до мережі Інтернет. Саме тому було прийнято рішення використовувати локальну базу даних на основі Room Persistence Library – офіційної бібліотеки від Google, яка є обгорткою над SQLite.

Room спрощує роботу з базою даних завдяки використанню анотацій у коді, при цьому зберігаючи повну сумісність із SQL. Архітектура Room передбачає використання трьох основних компонентів:

– entity – клас, що описує структуру таблиці бази даних;

– DAO (Data Access Object) – інтерфейс, який містить методи доступу до даних (запити SELECT, INSERT, DELETE тощо);

– database – абстрактний клас, який об’єднує всі DAO та реалізує єдину точку доступу до БД.

У нашій реалізації доступ до даних здійснюється через інтерфейс FinstatDao. У лістингу 3.8 наведено фрагмент цього інтерфейсу.

Лістинг 3.8 – Інтерфейс FinstatDao

```
@Dao
interface FinstatDao {

    @Insert(entity = Category::class, onConflict = OnConflictStrategy.REPLACE)
    fun insertCategory(category: Category)

    @Query("SELECT * FROM category")
    fun getAllCategories(): List<Category>

    @Query("DELETE FROM category WHERE name = :categoryName")
    fun deleteCategoryByName(categoryName: String)

    @Query("DELETE FROM category WHERE id = :id")
    fun deleteCategoryById(id: Long)

}
```

кінець лістингу 3.8

Усі компоненти, що відповідають за роботу з локальною базою даних, були логічно згруповані у пакеті db, який поділяється на два підпакети (рис. 3.7).

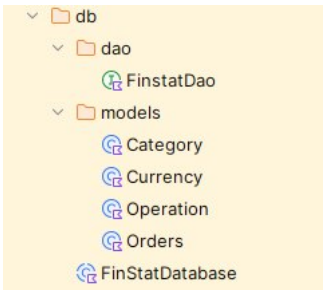

Рисунок 3.7 – Структура бази даних FinStat

В папці dao – міститься єдиний DAO-інтерфейс FinstatDao, що відповідає за виконання всіх CRUD-операцій для таблиць [16].

Папка models – включає всі сутності бази даних (Category, Orders, Operation, Currency).

Ключовим елементом у реалізації бази даних є клас FinStatDatabase, який оголошує конфігурацію всієї бази Room та забезпечує єдину точку доступу до всіх DAO-методів (ліст. 3.9). Він успадковується від базового класу RoomDatabase і позначається анотацією @Database, де вказуються всі сутності та версія бази.

Лістинг 3.9 – Клас FinStatDatabase

```
@Database(
    version = 1,
    entities = [Category::class, Orders::class, Operation::class,
Currency::class]
)
abstract class FinStatDatabase : RoomDatabase() {
    abstract fun getDao(): FinstatDao
}
```

кінець лістингу 3.9

Розробка моделей наприклад модель Currency яка зображена на лістингу 3.10 описує структуру таблиці валют. Вона складається з трьох полів:

– id – унікальний ідентифікатор, що генерується автоматично;
– name – назва валюти;
– editable – ознака, чи можна редагувати запис.

Поля помічені анотаціями @PrimaryKey та @ColumnInfo, що дає змогу Room коректно побудувати структуру таблиці.

Лістинг 3.10 – Модель Currency

```
@Entity
data class Currency(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,

    @ColumnInfo(name = "name")
    val name: String,
```

```
    @ColumnInfo(name = "editable")
    val editable: Int
)
```

кінець лістингу 3.10

Решта моделей реалізовано за аналогічним підходом, що забезпечує уніфіковану структуру проєкту, полегшує масштабування та підтримку. Завдяки модульності та повторному використанню коду, база даних легко адаптується до нових вимог. За потреби можна безболісно додати нові таблиці, розширити DAO-інтерфейси або змінити структуру вже існуючих моделей.

Далі буде детально розглянуто процес розробки інтерфейсу доступу до даних (DAO), який відповідає за логіку взаємодії із чотирма основними таблицями: category, orders, operation та currency.

Категорії – це логічна основа структурування транзакцій у додатку. Вони дозволяють віднести операцію до певного типу витрат або доходів: наприклад, «Продукти», «Зарплата», «Транспорт» тощо. З технічної точки зору, категорії є окремою таблицею у базі даних, яка зберігає унікальні ідентифікатори, назви та, за потреби, інші атрибути, такі як колір або тип. Приклад коду для вставляння категорії зображений на лістингу 3.11.

Лістинг 3.11 – Вставка категорії

```
@Insert(entity = Category::class, onConflict = OnConflictStrategy.REPLACE)
fun insertCategory(category: Category)
```

кінець лістингу 3.11

Цей метод дозволяє вставити нову категорію в таблицю category. У разі, якщо запис із таким самим id вже існує, він автоматично замінюється – завдяки використанню анотації @Insert(onConflict = OnConflictStrategy.REPLACE). Такий підхід дозволяє оновлювати наявні категорії без необхідності попереднього видалення, що спрощує логіку обробки даних та знижує ризик виникнення помилок у разі дублювання. Для отримання списку всіх збережених категорій реалізовано SQL-запит за допомогою анотації @Query, який наведено

у лістингу 3.12. Отримані дані використовуються для відображення категорій у відповідних елементах інтерфейсу.

Лістинг 3.12 – Отримання всіх категорій

```
@Query("SELECT * FROM category")
fun getAllCategories(): List<Category>
```

кінець лістингу 3.12

Цей метод виконує повну вибірку всіх записів із таблиці category. Отриманий список передається на екран користувачу або у ViewModel для подальшої обробки.

Також реалізував можливість видалення категорій за назвою, приклад зображено на лістингу 3.13.

Лістинг 3.13 – Видалення категорій за назвою

```
@Query("SELECT * FROM category")
fun getAllCategories(): List<Category>
```

кінець лістингу 3.13

Решта структур бази даних – замовлення (Orders), операції (Operation) та валюта (Currency) – реалізовані за схожою схемою, що забезпечує уніфікований підхід до розробки, полегшуючи підтримку та тестування системи.

Наприклад, у таблиці orders зберігаються шаблонні фінансові записи, які користувач може повторно застосовувати. Це зручно для щомісячних платежів (оренда, підписки тощо). DAO-методи для цієї таблиці дозволяють додавати шаблон, переглядати всі наявні шаблони та видаляти непотрібні.

У таблиці operation фіксуються фактичні транзакції користувача. Особливістю цієї таблиці є наявність поля з унікальним unixTime, що дозволяє будувати запити з фільтрацією за датами. Це необхідно для статистичних модулів додатку.

Нарешті, таблиця currency зберігає інформацію про підтримувані валюти, включаючи їхні назви, символи, коди. Це дозволяє вести облік у кількох валютах та здійснювати конверсію, якщо така функція буде додана в майбутньому.

Таким чином, підхід до розробки бази даних у додатку FinStat базується на концепції модульності та повторного використання. Одноманітність реалізації DAO для кожної сутності дозволяє зручно масштабувати базу в майбутньому: додати нові таблиці, розширити функціональність запитів, додати нові поля чи зв'язки між таблицями. Крім того, використання бібліотеки Room суттєво знижує ймовірність помилок, покращує безпеку обробки даних та спрощує написання unit-тестів.

Якби база розроблялася у середовищі типу MySQL або phpMyAdmin, структура запитів була б ідентичною, однак вимагала би більше ручної роботи та менше інтеграції з Android-архітектурою. Тож вибір Room як інструмента проєктування бази даних повністю відповідає вимогам сучасної мобільної розробки.

3.4 Розробка документації для програмного продукту

Розробка мобільного додатку FinStat супроводжувалася створенням документації, яка охоплює як технічні вимоги до пристрою, так і інструкції для кінцевого користувача. Оскільки застосунок орієнтований на широке коло користувачів, важливо було забезпечити зрозумілий і простий у використанні інтерфейс, а також підготувати базову інформаційну підтримку. Мобільний додаток призначений для ведення особистого фінансового обліку, зручного контролю витрат і доходів, а також формування базової статистичної звітності. Його основна мета – надати користувачеві інструмент для щоденного моніторингу фінансових операцій, аналізу фінансової поведінки та планування бюджету. Додаток дозволяє створювати категорії витрат і доходів, фіксувати фінансові операції, обирати рахунки та валюти, отримувати аналітику за вибраний період. Завдяки підтримці локальної бази даних, FinStat може

функціонувати без доступу до Інтернету, що робить його зручним у будь-яких умовах.

Основною цільовою аудиторією FinStat є звичайні користувачі, які прагнуть упорядкувати свої фінанси, отримати прозорий огляд витрат та ефективніше керувати особистим бюджетом. Програма не вимагає спеціальних технічних знань або фінансової підготовки, її інтерфейс інтуїтивно зрозумілий і пристосований до повсякденного використання. Також потенційними користувачами можуть бути фрілансери, дрібні підприємці або фінансові консультанти, які ведуть облік кількох джерел доходів і витрат. У перспективі можливе розширення функціоналу для потреб малого бізнесу або додавання мультикористувацького режиму для сімейного бюджету.

Мінімальні системні вимоги додатку базуються на необхідності забезпечення стабільної роботи та сумісності з більшістю сучасних пристроїв. Для коректного функціонування FinStat були визначені мінімальні та рекомендовані системні вимоги.

Мінімальні системні вимоги:

– операційна система: Android 8.0 (Oreo) або новіша;
– оперативна пам’ять (RAM): 2 ГБ;
– процесор: ARMv7 (32-біт) або ARM64;
– вільне місце на пристрої: від 100 МБ;
– дозвіл екрану: HD (1280×720) або вище.

Рекомендовані системні вимоги:

– операційна система: Android 10.0 або новіша;
– оперативна пам’ять (RAM): 3 ГБ або більше;
– процесор: ARM64, з тактовою частотою 1.8 ГГц і вище;
– вільне місце на пристрої: від 200 МБ;
– дозвіл екрану: Full HD (1920×1080) або вище;
– підключення до Інтернету: необов’язкове, але бажане для резервного копіювання (у майбутніх версіях).

Додаток розрахований на автономне використання, тому він не потребує постійного підключення до інтернету. Після встановлення система створює локальну базу даних Room, яка містить всю необхідну інформацію для роботи з транзакціями, категоріями, рахунками та валютами. Перший запуск додатку супроводжується простим інтерактивним поясненням основних дій, таких як додавання нової транзакції або перегляд статистики. Усі іконки в інтерфейсі мають зрозумілі назви, а додаткові пояснення інтегровані у вигляді коротких підказок, що з’являються контекстно.

Для встановлення FinStat достатньо завантажити інсталяційний .apk файл, надати дозвіл на встановлення з невідомих джерел у налаштуваннях пристрою, після чого відкрити файл для інсталяції. Додаткових дій від користувача не потрібно, і додаток одразу готовий до роботи. Всі ключові функції збережені на пристрої, тому користувач може вільно працювати з додатком у режимі офлайн.

У разі розширення функціоналу та інтеграції з хмарними сервісами, документація буде доповнена інформацією про авторизацію, синхронізацію даних та підключення зовнішніх облікових записів. На даному етапі жодних серверних налаштувань або плагінів підключати не потрібно.

3.5 Розробка інсталяційного пакета

Формування інсталяційного пакета для FinStat передбачає створення зручного та компактного .apk-файлу, який містить усі необхідні компоненти додатку. У процесі складання пакета важливо було забезпечити його просту інсталяцію на пристрої кінцевого користувача, без необхідності ручної настройки бази даних чи додаткових системних залежностей.

Після інсталяції додаток розміщується у внутрішньому сховищі Android-пристрою, де автоматично створюються каталоги з локальними даними, фінальний вигляд програми на Android зображено на рисунку 3.7.

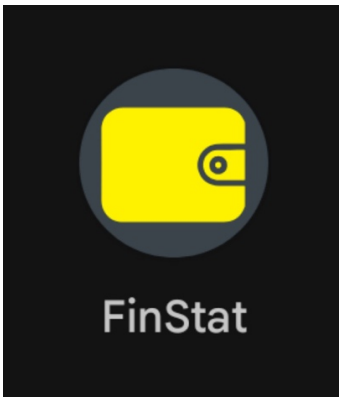

Рисунок 3.7 – Вигляд програми FinStat на Android

Усі транзакції, категорії, рахунки та валюти зберігаються у файлі бази даних finstat.db, який розміщується у внутрішньому каталозі додатку. Окрім цього, у підкаталогах зберігаються налаштування користувача у форматі XML та тимчасові дані, необхідні для кешування зображень і прискорення роботи.

Оскільки Android не використовує реєстр у традиційному розумінні, як це має місце у Windows-системах, усі конфігураційні параметри зберігаються через SharedPreferences. Це забезпечує безпечний, простий і гнучкий спосіб зберігання налаштувань, які можуть змінюватися без потреби перевстановлення додатку.

Під час створення інсталяційного файлу за допомогою Gradle-конфігурації були визначені всі залежності додатку, включно з бібліотеками Jetpack Compose, Room, Hilt та Navigation. Складання відбувалося за допомогою команди ./gradlew assembleRelease, яка формує готовий .apk файл для розповсюдження або публікації на платформі Google Play.

У майбутньому, якщо додаток буде поширюватися через Play Market, інсталяційний пакет зберігатиме свою структуру, однак буде доповнений метаданими, цифровим підписом і політикою безпеки, що вимагається для публікації у Google Store.

Таким чином, розробка інсталяційного пакета для FinStat була реалізована з акцентом на автономність, безпеку та мінімальні вимоги до користувача, що дозволяє легко інсталювати й почати користування додатком навіть без спеціальних технічних знань.

Висновки до розділу 3

У результаті практичної реалізації було створено мобільний додаток FinStat для ефективного управління особистими фінансами. Реалізація виконана з дотриманням сучасних підходів до розробки Android-застосунків, зокрема використанням архітектурного патерну MVVM, бібліотек Jetpack, Kotlin DSL та інструментів автоматичного тестування.

У межах розділу реалізовано такі функціональні можливості, як додавання й редагування транзакцій, категоризація витрат, додавання різноманітних рахунків, генерація статистичних звітів та встановлення бюджету. Особливу увагу приділено реалізації зручного та інтуїтивного інтерфейсу з використанням Jetpack Compose.

Також було проведено комплексне тестування додатку з використанням AndroidX Test, Espresso та емулятора BlueStacks, що дало змогу виявити й усунути типові помилки. Використання Gradle Kotlin DSL суттєво спростило конфігурацію проєкту та покращило продуктивність розробки. Для зберігання даних використано SQLite з бібліотекою Room, що забезпечило надійність і автономність роботи системи.

Таким чином, реалізована система відповідає усім поставленим вимогам: вона є стабільною, зручною у використанні, безпечною та легко масштабованою, що створює підґрунтя для її подальшого розвитку і впровадження.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено детальний аналіз сучасного ринку мобільних застосунків для управління особистими фінансами. Було розглянуто такі популярні рішення, як Mint, YNAB (You Need a Budget), PocketGuard та Toshl Finance, кожне з яких має власні переваги, такі як автоматичне відстеження транзакцій, зручний інтерфейс чи підтримка мультивалютності. Водночас були виявлені суттєві недоліки, зокрема обмежена безкоштовна функціональність, низька кастомізація, складний або перевантажений інтерфейс, а також ризики, пов'язані з безпекою персональних даних. Проведений аналіз підтвердив потребу в новому рішенні, яке б поєднувало функціональну гнучкість, автономність, зручність використання та сучасний підхід до архітектури застосунку.

У межах роботи було спроєктовано архітектуру мобільного додатку FinStat, яка охоплює основні екрани: введення транзакцій, перегляд статистики, налаштування, а також модуль категоризації фінансових операцій і локальне зберігання даних. Для побудови архітектури використано шаблон MVVM, що забезпечує чітке розділення логіки та інтерфейсу, спрощує підтримку і розширення проєкту. Збереження даних реалізовано за допомогою бібліотеки Room, що забезпечує надійну роботу додатку в автономному режимі та швидкий доступ до інформації без підключення до мережі.

Важливим етапом стало налаштування проєкту за допомогою Gradle Kotlin DSL – сучасного підходу до конфігурації Android-проєктів. Застосування DSL на базі мови Kotlin забезпечило прозоре й типобезпечне керування залежностями, що суттєво спростило структуру конфігураційних файлів та зробило проєкт зручним для розширення. Це дозволило забезпечити стабільну збірку проєкту, легке масштабування і простоту інтеграції додаткових бібліотек та модулів у майбутньому.

У процесі реалізації було виконано програмну розробку основного функціоналу: додавання, редагування та видалення транзакцій, їх категоризації

за доходами й витратами, виведення статистики за вибраний період, створення рахунків тощо. Незважаючи на відсутність модулів формування бюджету або фінансових цілей у поточній версії, реалізована система повністю задовольняє базові потреби користувача в обліку особистих фінансів. Для представлення даних застосовано елементи візуалізації у вигляді списків і текстової статистики, а логіка бізнес-операцій реалізована у ViewModel.

Особливу увагу в роботі приділено розробці користувацького інтерфейсу. За допомогою Jetpack Compose було створено зручний, адаптивний та інтуїтивно зрозумілий UI, що відповідає сучасним вимогам до мобільних застосунків. Дизайн забезпечує швидкий доступ до основних функцій, мінімізує кількість зайвих дій і дозволяє користувачам легко орієнтуватися в структурі застосунку, що позитивно впливає на загальний користувацький досвід.

Завершальним етапом стало тестування реалізованої системи. Було проведено ручне, модульне та інструментальне тестування, яке дозволило виявити і усунути критичні помилки, перевірити стабільність роботи додатку на різних емуляторах і пристроях. Для перевірки функціональної цілісності було протестовано основні сценарії взаємодії користувача з додатком: створення транзакцій, навігацію між екранами, збереження та перегляд даних. Тестування підтвердило надійність роботи FinStat в автономному режимі, відсутність збоїв при зміні стану застосунку (перезапуск, зміна орієнтації тощо), а також ефективність реалізованих рішень.

Таким чином, усі поставлені завдання були виконані повністю. У роботі створено стабільний, автономний та зручний мобільний додаток для управління особистими фінансами, побудований на сучасних технологіях Kotlin, Jetpack Compose, Room і Gradle Kotlin DSL. Проєкт має прикладне значення та може бути використаний як основа для подальшого розвитку програмних продуктів у сфері персональних фінансів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Monobank. Як ми аналізуємо ваші витрати. URL: https://monobank.ua/blog/how-we-analyze-your-expenses (дата звернення: 06.02.2025).

2. Gumilar D. W. A., Sangka K. B., Totalia S. Digital Financial Literacy and Digital Financial Inclusion in the Era of Digital Disruption. Formosa Journal of Multidisciplinary Research. 2023. Vol. 3, No. 5. P. 1563-1576. URL: http://bit.ly/3Z74Xvq (дата звернення: 08.02.2025).

3. Соболєва-Терещенко О., Жарнікова В. Current state and prospects for the development of Digital financial literacy in Ukraine. VUZF Review. 2022. Т. 7, № 2. С. 205-215. URL: https://bit.ly/4dDaz6B (дата звернення: 15.02.2025).

4. Шевченко В. Диджіталізація та міжнародний доступ до фінансів. Менеджмент. 2023. № 2(38). С. 151-156. URL: https://er.knutd.edu.ua/handle /123456789/27072 (дата звернення: 08.05.2025).

5. Personalized financial management system based on machine learning: US Patent US20230267484A1. G06Q 40/02 / Capital One Services, LLC. Appl. No. US20230267484A1; filed: 10.02.2022; published: 24.08.2023. 23 p. URL: https://patents.google.com/patent/US20230267484A1 (дата звернення: 09.03.2025).

6. Financial transaction analysis system based on messaging: US Patent US12106383B2. G06Q 20/06 / Capital One Services, LLC. Appl. No. US12106383B2; filed: 12.07.2022; published: 01.10.2024. 18 p. URL: https://patentsgazette.uspto.gov /week40/OG/html/1527-1/US12106383-20241001.html (дата звернення: 15.03.2025).

7. Subscription management methods via mobile application: US Patent US11669817B2. G06Q 30/02 / Capital One Services, LLC. Appl. No. US11669817B2; filed: 11.03.2021; published: 06.06.2023. 20 p. URL: https://patents.google.com /patent/US11669817B2 (дата звернення: 16.03.2025).

8. Mint Fin – мобільний застосунок для фінансового обліку. URL: https://apps.apple.com/us/app/mint-fin/id1634154128 (дата звернення: 27.03.2025).

9. How to Create a Budget Template. YNAB Blog. URL: https://www.ynab.com/blog/how-to-create-a-budget-template (дата звернення: 01.04.2025).

10. PocketGuard – personal finance management service. URL: https://pocketguard.com (дата звернення: 01.04.2025).

11. Toshl Finance – personal budgeting application. URL: https://toshl.com (дата звернення: 15.04.2025).

12. Kotlin documentation. URL: https://kotlinlang.org/docs/home.html (дата звернення: 25.04.2025).

13. Емулятори, симулятори та ферми пристроїв для тестування. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/emulators-simulators-farm-devices/ (дата звернення: 10.05.2025).

14. AndroidX Test official documentation. Android Developers. URL: https://developer.android.com/training/testing (дата звернення: 16.05.2025).

15. Android Espresso official documentation. Android Developers. URL: https://developer.android.com/training/testing/espresso (дата звернення: 17.05.2025).

16. CRUD database validation. QATestLab. URL: https://bit.ly/4dCkTM4 (дата звернення: 19.05.2025).

ДОДАТКИ

Додаток А

Фрагмент коду CategoriesFragment.kt – управління категоріями у мобільному застосунку FinStat

```
@AndroidEntryPoint
class CategoriesFragment : Fragment(), CategoryListener {

    lateinit var binding: FragmentCategoriesBinding
    private val viewModel: FinStatViewModel by viewModels()

    lateinit var adapter: CategoryAdapter
    lateinit var list: List<Category>

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentCategoriesBinding.inflate(layoutInflater, container,
false)
        binding.addCategory.setOnClickListener { addCategory() }
        return binding.root
    }

    override fun onStart() {
        super.onStart()
        viewModel.categoriesLD.observe(viewLifecycleOwner) {
            list = it
            setToUI()
        }
        viewModel.updateLD.observe(viewLifecycleOwner) {
            viewModel.getAllCategories()
        }
        viewModel.getAllCategories()
    }

    private fun setToUI() {
        binding.categoriesRw.layoutManager =
LinearLayoutManager(requireContext())
        adapter = CategoryAdapter(list)
        binding.categoriesRw.adapter = adapter
        swipeToGesture(binding.categoriesRw)
    }

    fun addCategory() {
        binding.dialog.setText(
            message = resources.getString(R.string.add_message),
            dialogType = DialogType.EDIT,
            dialogListener = object : DialogListener {
                override fun accept(message: String?) {
                    if (!message.isNullOrEmpty()) {
                        viewModel.addCategory(message)
                    }
                    hideDialog()
                }

                override fun decline() {
                    hideDialog()
                }

                override fun hideKeyboardOnSimpleDialog() {
```

```
                hideKeyboard()
            }
        }
    )
    showDialog()
}

override fun editCategory(category: Category) {
    binding.dialog.setText(
        message = resources.getString(R.string.edit_category_message,
category.name),
        text = category.name,
        buttonAcceptMessage = resources.getString(R.string.edit),
        dialogType = DialogType.EDIT,
        dialogListener = object : DialogListener {
            override fun accept(message: String?) {
                if (!message.isNullOrEmpty()) {
                    viewModel.editCategory(category.id, message)
                }
                hideDialog()
            }

            override fun decline() {
                hideDialog()
                viewModel.getAllCategories()
            }

            override fun hideKeyboardOnSimpleDialog() {
                hideKeyboard()
            }
        }
    )
    showDialog()
}

override fun deleteCategory(category: Category) {
    binding.dialog.setText(
        message = resources.getString(R.string.delete_category_message,
category.name),
        buttonAcceptMessage = resources.getString(R.string.delete),
        dialogType = DialogType.SIMPLE,
        dialogListener = object : DialogListener {
            override fun accept(message: String?) {
                hideDialog()
                viewModel.deleteCategory(category)
            }

            override fun decline() {
                hideDialog()
                viewModel.getAllCategories()
            }

            override fun hideKeyboardOnSimpleDialog() {
                hideKeyboard()
            }
        }
    )
    showDialog()
}

fun showDialog() {
    binding.shadow.isVisible = true
    binding.dialog.isVisible = true
}
```

```
    fun hideDialog() {
        binding.shadow.isVisible = false
        binding.dialog.isVisible = false
        hideKeyboard()
    }

    fun hideKeyboard() {

(requireActivity().getSystemService(AppCompatActivity.INPUT_METHOD_SERVICE) as?
InputMethodManager)
            ?.hideSoftInputFromWindow(requireView().windowToken, 0)
    }

    private fun swipeToGesture(rv: RecyclerView?) {
        val swipeGesture = object : SwipeGesture(requireContext()) {
            override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction:
Int) {
                val position = viewHolder.adapterPosition
                when (direction) {
                    ItemTouchHelper.LEFT -> {
                        if (list[position].editable == 0) {
                            Toast.makeText(
                                requireContext(),
                                resources.getString(R.string.uneditable),
                                Toast.LENGTH_LONG
                            ).show()
                            viewModel.getAllCategories()
                        } else {
                            deleteCategory(list[position])
                        }
                    }

                    ItemTouchHelper.RIGHT -> {
                        editCategory(list[position])
                    }
                }
            }
        }

        val touchHelper = ItemTouchHelper(swipeGesture)
        touchHelper.attachToRecyclerView(rv)
    }
}
```