Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ТА АНАЛІЗУ ВИТРАТ»

Виконала: студентка 4 курсу, групи 2ІСТ-21б спеціальності 126 – «Інформаційні системи та технології»
_____ Аліна СНІЦАР
Керівник: к.т.н., доц. каф. САІТ
_____ Георгій ГОРЯЧЕВ
«27» 05 2025 р.
Рецензент: к.т.н., доц. каф. КН
_____ Олександр ШЕВЧУК
«12» 06 2025 р.

Допущено до захисту

Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
«06» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра системного аналізу та інформаційних технологій
Рівень вищої освіти – перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність 126 – «Інформаційні системи та технології»
Освітньо-професійна програма – Прикладні інформаційні технології

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
д.т.н., проф. Віталій МОКІН
"28" 03 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Сніцар Аліні Валентинівні

1. Тема роботи: «Інформаційна системи управління особистими фінансами та аналізу витрат»
керівник роботи: Горячев Г. В., к.т.н., доц. каф. САІТ
затверджені наказом ВНТУ від «20» 05 2025 року № 97
2. Термін подання студентом роботи 31.05.25 р.
3. Вихідні дані до роботи: статистика фінансової грамотності населення.
4. Зміст текстової частини:
1) Аналіз предметної області та огляд існуючих рішень
2) Вибір IT-інфраструктури та проєктування інформаційної системи.
3) Розроблення інформаційної системи.
5. Перелік ілюстративного матеріалу:
1) Графік фінансових знань населення;
2) Структура системи;
3) Діаграма USE CASE інформаційної системи;.
4) UML-діаграми;
5) Діаграма класів.
6) Інтерфейс ІС

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 28.03.25 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області та огляд існуючих рішень	01.04.25	15.04.25	виконано
2	Вибір IT-інфраструктури та проєктування інформаційної системи.	15.04.25	30.04.25	виконано
3	Розроблення інформаційної системи.	01.05.25	25.05.25	виконано
4	Оформлення матеріалів до захисту БКР	25.05.25	05.06.25	виконано

Студент _____ Аліна СНІЦАР

Керівник БКР _____ Георгій ГОРЯЧЕВ

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 82 сторінок формату А4, містить 34 рисунки, список використаних джерел налічує 20 найменувань.

Метою роботи є розроблення інформаційної системи для управління особистими фінансами та аналізу витрат.

У роботі виконано огляд предметної області та аналіз аналогічних мобільних застосунків, визначено функціональні вимоги та технічне завдання. Реалізовано мобільний додаток на платформі Android з використанням Java та SQLite. Описано архітектуру системи, розроблено користувацький інтерфейс, локальну базу даних та основні функції: ведення обліку доходів і витрат, пошук, аналітика та постановка фінансових цілей. Проведено тестування функціоналу та побудовано UML-діаграми.

Розроблена інформаційна система забезпечує автономну роботу користувача без підключення до інтернету та підвищує зручність у контролі персонального бюджету.

Ключові слова: інформаційна система, особисті фінанси, мобільний додаток, Android, Java, SQLite, облік витрат.

ABSTRACT

The Bachelor's qualification work consists of 82 A4 pages, includes 34 figures, and the list of references contains 20 sources.

The aim of the work is to develop an information system for managing personal finances and analyzing expenses.

The study includes an overview of the subject area and analysis of similar mobile applications, definition of functional requirements and technical specifications. A mobile application for the Android platform was implemented using Java and SQLite. The system architecture is described, along with the development of the user interface, local database, and main functions: income and expense tracking, search, analytics, and setting financial goals. Functional testing was conducted and UML diagrams were created.

The developed information system enables autonomous user operation without internet access and improves convenience in managing a personal budget.

Keywords: information system, personal finance, mobile application, Android, Java, SQLite, expense tracking.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах розвитку цифрових технологій та економічної нестабільності зростає потреба в ефективному контролі особистих фінансів. За даними Державної служби статистики України, понад 70% населення користуються смартфонами [1], що створює сприятливі умови для впровадження мобільних рішень, орієнтованих на спрощення фінансового обліку та аналізу.

Особисті фінанси охоплюють низку складових — доходи, витрати, заощадження, інвестиції [2]. Водночас, низький рівень фінансової грамотності населення зумовлює потребу у створенні простих та доступних інструментів управління фінансами.

Мета і задачі дослідження. Метою роботи є розроблення зручної інформаційної системи для особистого фінансового обліку шляхом створення мобільного додатку, яка дозволить користувачам ефективно планувати, аналізувати та контролювати свої фінансові ресурси.

Для досягнення мети дослідження поставлені такі завдання:

- проаналізувати предметну область та рівень фінансової грамотності користувачів;
- дослідити наявні програмні рішення та виявити їхні недоліки;
- визначити вимоги до функціоналу інформаційної системи;
- розробити архітектуру, інтерфейс користувача та структуру бази даних;
- реалізувати інформаційну систему з використанням мови Java та SQLite;
- провести тестування системи й оцінити її ефективність та зручність використання.

Об’єктом дослідження є процес розроблення інформаційної системи управління особистими фінансами за допомогою інформаційних технологій.

Предметом дослідження є методи та інструменти розробки інформаційної системи у вигляді мобільного додатку для управління особистими фінансами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття та структура особистого бюджету

Особисті фінанси — це важлива частина життя кожної людини, адже вони показують, наскільки ефективно ми вміємо керувати своїми грошима. Управління особистими фінансами — це процес, який включає планування, розподіл і контроль доходів, витрат, заощаджень та інвестицій. Головна мета цього процесу — досягти фінансової стабільності та покращити якість життя завдяки розумному використанню своїх ресурсів.

Структура особистих фінансів включає кілька основних складових:

Доходи — це гроші, які людина отримує, наприклад, із зарплати, власної справи, інвестицій, пенсії або соціальних виплат.

Витрати — це гроші, які людина витрачає на щоденні потреби: оплату комунальних послуг, проїзд, їжу, лікування, навчання та інші речі.

Заощадження та інвестиції — це частина доходу, яку людина не витрачає відразу, а відкладає «на потім» або вкладає в щось (наприклад, у нерухомість, акції, золото), щоб у майбутньому отримати прибуток.

Ефективне управління допомагає не тільки покривати щоденні потреби, а й відкладати гроші на майбутнє, бути готовим до несподіваних витрат і досягати своїх фінансових цілей.

Традиційні методи — записування витрат у зошиті чи зберігання квитанцій - поступово відходять на другий план, поступаючись місцем мобільним додаткам для контролю фінансів. Ці інструменти пропонують зручні та ефективні способи управління [3].

1.2 Сучасні проблеми управління особистими фінансами в Україні

В Україні управління особистими фінансами має багато складнощів через економічну нестабільність і соціальні проблеми. Планувати свій особистий або сімейний бюджет важливо завжди. Кожному потрібно вчасно оплачувати комунальні послуги, кредити, страхування та інші обов'язкові витрати [4].

Ось деякі з найбільш актуальних проблем:

– Низький рівень фінансової грамотності. Багато українців не мають достатніх знань про основи фінансів, тому їм важко приймати правильні фінансові рішення. Це стосується не лише ведення бюджету, а й таких речей, як інвестиції чи пенсійне планування.

– Нестабільна економіка. Через інфляцію, кризи та нестабільність у країні людям важко керувати своїми фінансами. Часто вони не можуть планувати майбутнє або стикаються з серйозними фінансовими проблемами.

– Мало доступних фінансових інструментів. Не всі мають змогу користуватись сучасними фінансовими послугами — наприклад, кредитними картками чи інвестиційними продуктами. Це ускладнює ефективне управління грошима.

– Доходів часто не вистачає. У багатьох випадках витрати більші за доходи, тому люди змушені брати кредити або позики. Це особливо відчутно серед малозабезпечених, що ще більше ускладнює загальну фінансову ситуацію в країні.

Фінансова грамотність — це володіння набором навичок і знань, які дозволяють людині приймати обґрунтовані та ефективні рішення, використовуючи свої фінансові ресурси [5].

1.3 Аналіз даних щодо фінансової грамотності населення

Щоб краще зрозуміти, в якому фінансовому становищі знаходиться населення світу, і чому мій додаток є актуальним, я проаналізувала кілька досліджень. Одне з них — це міжнародне дослідження фінансової грамотності

дорослих, яке провела Організація економічного співробітництва та розвитку (ОЕСР). Ця організація об'єднує 38 країн, більшість з яких — розвинені, з високими доходами та рівнем життя.

Згідно з результатами дослідження OECD/INFE 2023, тільки 34% дорослих у 39 країнах мають базовий рівень фінансової грамотності (рис. 1.1). Це означає, що більшість людей у світі недостатньо обізнані у фінансових питаннях.

– Середній рівень фінансової грамотності серед усіх опитаних — 60 зі 100 балів.

– У країнах-членах ОЕСР цей показник трохи вищий — 63 зі 100.

– Цей бал враховує знання, фінансову поведінку та ставлення до грошей.

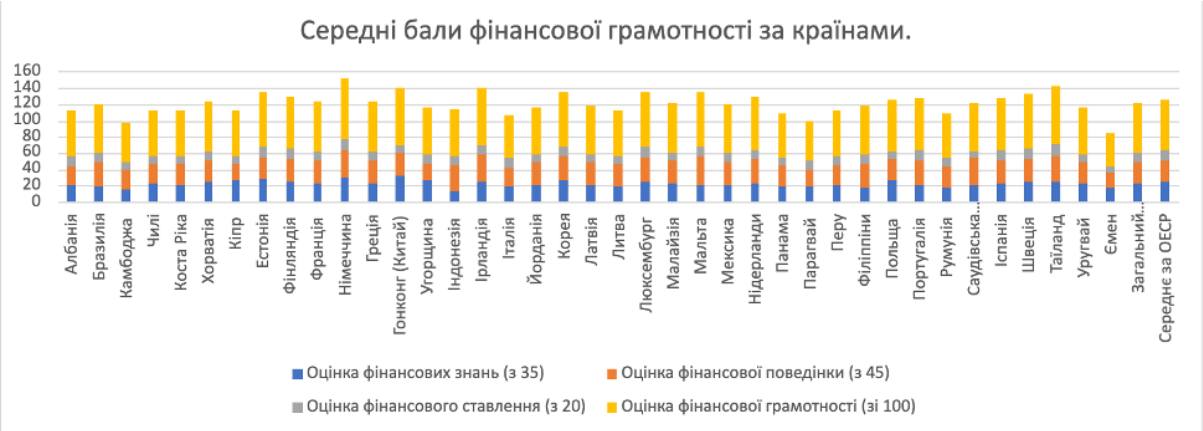

Рисунок 1.1 – Графік фінансової грамотності

Лише половина опитаних дорослих змогли правильно відповісти хоча б на 5 із 7 запитань, тобто досягли базового рівня. Середній результат за фінансовими знаннями склав 63 бали зі 100 (рис.1.2).

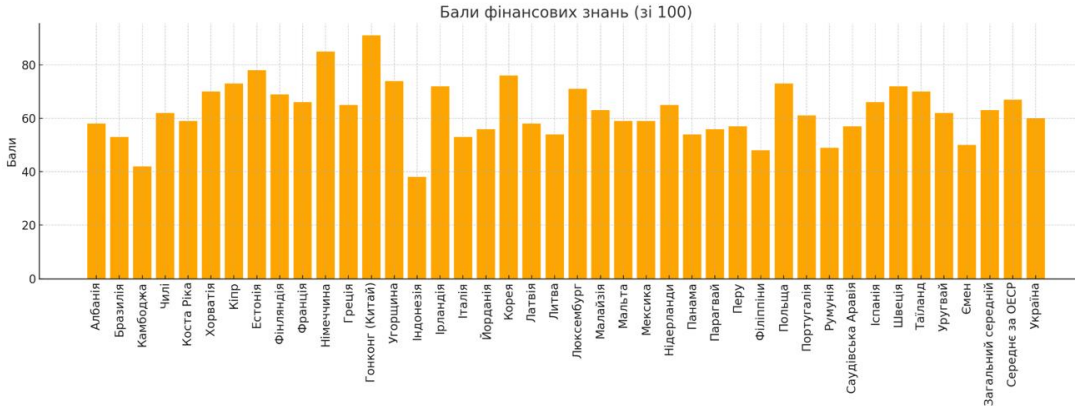

Рисунок 1.2 – Графік балів фінансових знань

Середній бал фінансового добробуту — 46 зі 100, що свідчить про низький рівень фінансової стійкості населення (рис.1.3).

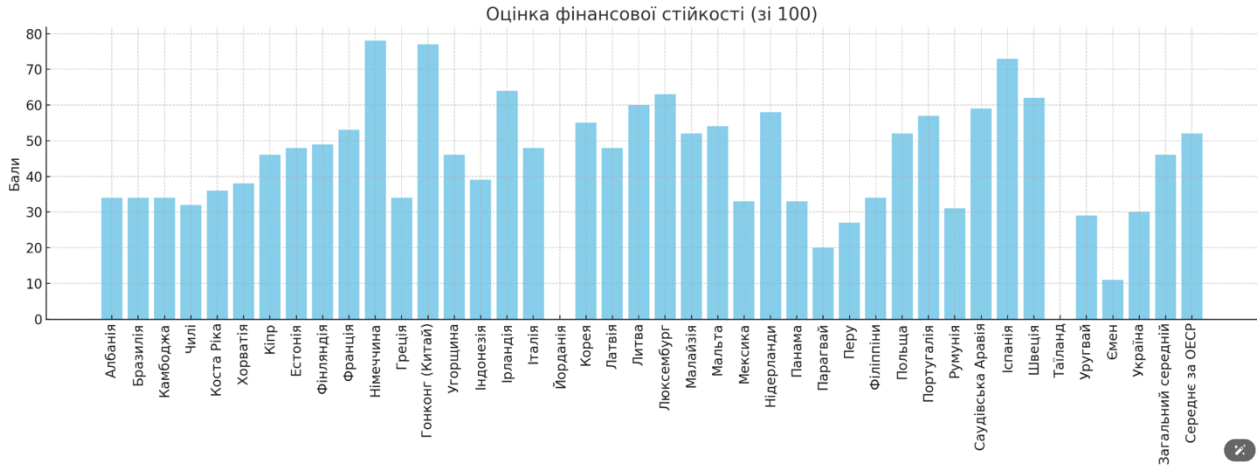

Рисунок 1.3 – Графік оцінки фінансової стійкості

Згідно з міжнародним дослідженням OECD/INFE 2023, середній рівень фінансової грамотності серед дорослих у світі становить 60 зі 100 балів. Тільки половина опитаних мають базові знання у сфері фінансів, а рівень фінансової поведінки теж далекий від ідеального. Це показує, що людям потрібні доступні інструменти для навчання, наприклад, мобільні додатки для ведення особистих фінансів.

На думку лауреата Нобелівської премії з економіки М. Алле, люди часто поводяться нелогічно, коли обирають між фінансовими варіантами. Це пов’язано

з браком знань і впливом стереотипів. У своїй роботі він критикував «теорію очікуваної корисності» й довів, що вона не завжди працює на практиці [6].

Щодо України, згідно з дослідженням USAID та НБУ у 2021 році, фінансова грамотність українців зросла з 11,6 до 12,3 балів (із 21 можливого) за три роки. Це позитивна динаміка, але рівень знань все ще нижчий, ніж у багатьох інших країнах.

Тому створення зручного мобільного додатку, який допоможе людям краще розуміти й контролювати свої фінанси — це справді важливий і потрібний крок. Такий інструмент може покращити фінансову поведінку та загальне благополуччя користувачів.

1.4 Обґрунтування актуальності розробки ІС

Актуальність створення інформаційних систем для управління особистими фінансами в Україні пов’язана з потребою підвищити фінансову грамотність населення. В умовах економічної нестабільності людям потрібні зручні інструменти, які допоможуть контролювати доходи й витрати, уникати фінансових проблем і підвищити свій добробут.

Мобільні додатки для управління фінансами є сучасним і доступним рішенням. Вони дають змогу у будь-який момент переглядати витрати та доходи, планувати бюджет, аналізувати фінансові звички та отримувати поради для економії. Такі додатки також допомагають краще розуміти, куди йдуть гроші, завдяки графікам, діаграмам та інтерактивним підказкам.

Міжнародний досвід показує, що в країнах з високим рівнем фінансової грамотності саме мобільні додатки стали основним інструментом для щоденного управління фінансами. Вони допомагають не тільки контролювати витрати, а й ефективно планувати майбутнє. Тому створення подібного додатку для України є важливим і своєчасним кроком, який може допомогти людям краще керувати своїми фінансами та підвищити фінансову стабільність у країні.

1.5 Аналіз ринку мобільних застосунків для фінансового обліку

Мобільні додатки для управління фінансами стали важливим інструментом для людей, які хочуть краще контролювати свої витрати, доходи та заощадження. В умовах економічної нестабільності та розвитку фінансових технологій такі додатки допомагають користувачам стежити за своїми фінансами в будь-який час і з будь-якого місця, що робить управління грошима зручним і доступним.

Загалом існує багато причин для створення додатку, який допоможе людям легко формувати бюджети, контролювати витрати і накопичення. Такий застосунок може стати надійним помічником для тих, хто хоче досягти фінансової стабільності, краще планувати свої фінанси та ухвалювати обдумані рішення.

Хоча зараз є багато фінансових додатків, більшість з них надають лише базові можливості — запис витрат, просте планування бюджету тощо. Але багатьом користувачам не вистачає більш глибоких функцій для повноцінного фінансового планування. Тому перед створенням власного рішення важливо проаналізувати найпопулярніші додатки, визначити їхні сильні сторони та недоліки, щоб розробити зручний і ефективний інструмент для користувачів.

Аналіз мобільного застосунку "Monefy – Розпорядник бюджету" (рис. 1.4).

Призначення: Monefy — це фінансовий організатор, створений для систематичного відстеження грошей, що пропонує просте додавання нових записів щоразу, коли робиться покупка.

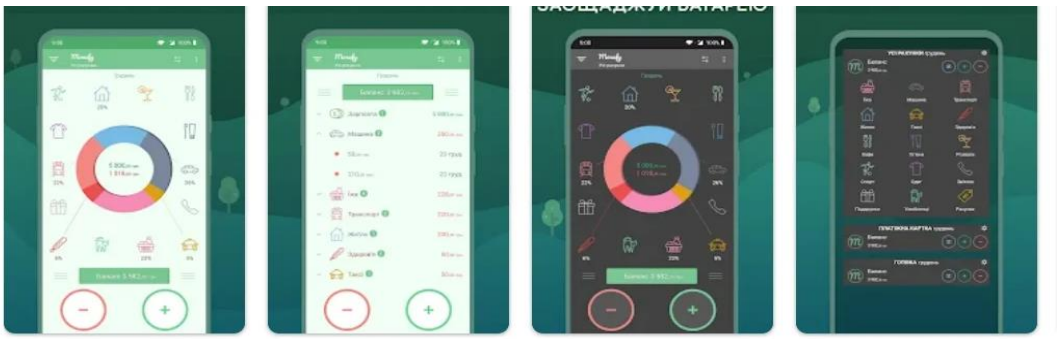

Рисунок 1.4 – Інтерфейс та обкладинки Monefy

Переваги**:**

– Дуже швидка функція додавання витрат.

– Зручна система фільтрування даних.

– Простий у використанні інтерфейс.

Недоліки**:**

– Відсутність функцій керування бюджетом.

– Не підтримує встановлення фінансових цілей.

Аналіз мобільного застосунку "YNAB (You Need a Budget)".

YNAB — це додаток для планування бюджету, який дозволяє користувачам організовувати свої витрати (рис. 1.5), відкладати кошти для досягнення фінансових цілей і стежити за фінансовим станом [7].

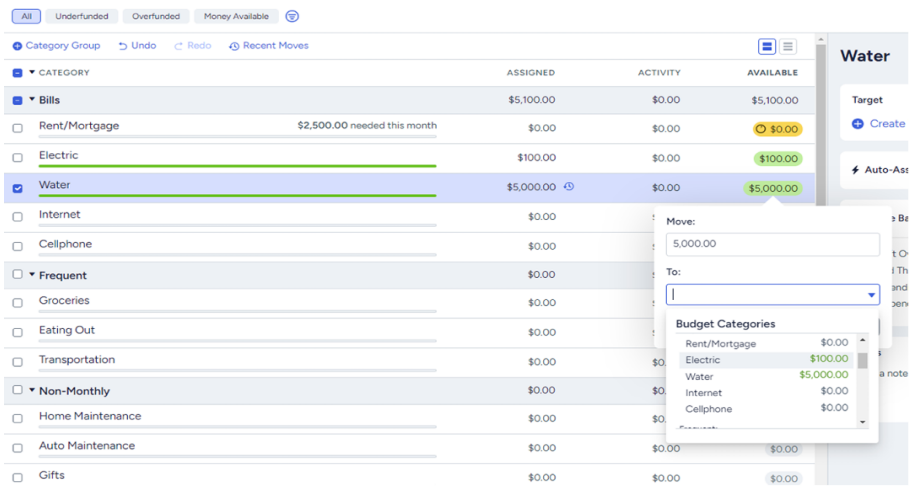

Рисунок 1.5 - вигляд Desktop версії застосунку

Переваги:

– Інтуїтивно зрозуміла система планування бюджету.

– Доступні інструкції для новачків.

– Детальна статистика для аналізу витрат за різні періоди.

Недоліки:

– Висока вартість підписки (15$ на місяць).

– Може бути складним для новачків, оскільки функціонал не завжди простий для розуміння.

Додаток дозволяє встановлювати цілі для різних категорій витрат, зручний розподіл коштів за пріоритетами, та надає детальну статистику витрат за різні часові періоди.

Аналіз мобільного застосунку "GoodBudget"

GoodBudget пропонує класичну систему конвертів для планування витрат, яка дозволяє відстежувати доходи та витрати, прогнозувати майбутні фінансові потреби і створювати резерви (рис. 1.6).

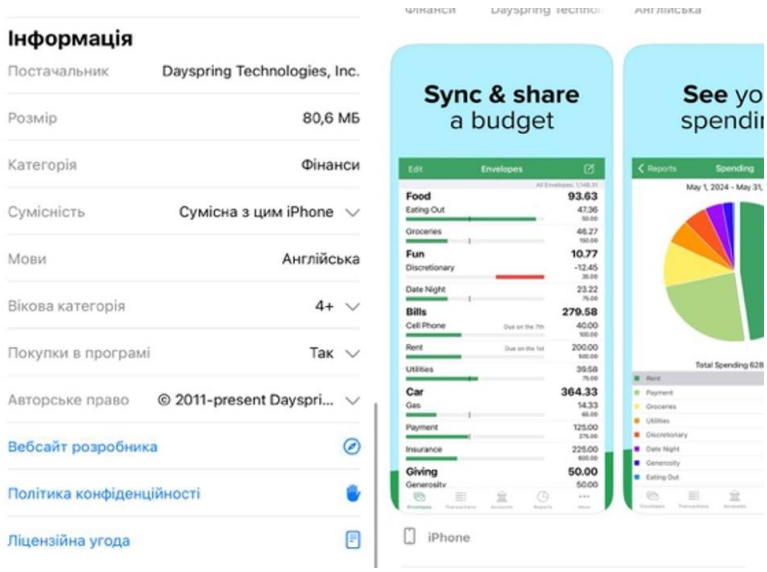

Рисунок 1.6 – Інтерфейс інформація про GoodBudget

Переваги**:**

– Простота використання.

– Безкоштовна версія з базовими функціями.

– Можливість синхронізації бюджетів.

Недоліки**:**

– Відсутність інтеграції з банківськими рахунками.

– Безкоштовна версія має обмеження щодо кількості конвертів.

– Відсутність детального аналізу витрат.

GoodBudget не підтримує підключення до банківських рахунків, що може бути недоліком для тих, хто хоче автоматично імпортувати транзакції. Але додаток є дуже зручним для користувачів, які віддають перевагу простоті.

Аналіз мобільного застосунку "Spendee".

Spendee — це мобільний додаток, який дозволяє відстежувати витрати та створювати бюджет на основі категорій (рис.1.7). Додаток пропонує можливість синхронізувати його з банківськими рахунками (для підтримуваних банків) і планувати свої фінанси.

Рисунок 1.7 – Інтерфейс Spendee

Переваги:

– Можливість синхронізації з банківськими рахунками (для підтримуваних банків).
– Інтуїтивно зрозумілий інтерфейс.
– Множинні варіанти для налаштування бюджету.
– Має дестопну версію.

Недоліки:

– Висока вартість підписки (до 50$ на рік).

1.6 Виявлення недоліків існуючих рішень

Після аналізу існуючих мобільних додатків для ведення фінансів можна виділити кілька основних недоліків:

Занадто дорога підписка. Багато популярних додатків, наприклад YNAB або Spendee, коштують дорого, тому не кожен може собі їх дозволити.

Немає функції фінансових цілей. Додатки, як-от Monefy або GoodBudget, не дають змоги створювати фінансові цілі, а це важливо для тих, хто хоче накопичити кошти на щось конкретне.

Слабкий аналіз витрат. У деяких застосунках немає детального аналізу витрат, тому користувачам важко зрозуміти, на що саме йдуть їхні гроші.

Усі основні переваги й недоліки популярних додатків були зведені в порівняльну таблицю (рис. 1.8), щоб наочно побачити, чого не вистачає і що можна покращити в майбутньому додатку.

Функції	YNAB	Monefy	GoodBudget	Spendee
Облік доходів та витрат	+	+	+	+
Зрозумілий інтерфейс	+	+	-	+
Редагування категорій	+	-	-	+
Інтеграція з банківським картками	-	-	-	-
Швидке введення	+	+	-	+
Генерація звітів в графіках	+	-	-	+
Створення фінансової цілі для накопичення	-	-	-	-
Функції пошуку (фільтрації) витрат	-	+	-	+

Рисунок 1.8 – Порівняльна таблиця

На основі порівняльної таблиці можемо зробити висновок, що перші версії додатку не обов’язково повинні мати функції по роботі з банківськими рахунками, спільного обліку, нагадувань про платежі, розподілу на простори чи бізнес інструменти.

1.7 Визначення вимог до програмного забезпечення та розробка технічного завдання на основі аналізу аналогів

Після проведеного аналізу аналогів мобільних застосунків, було визначено ряд функцій, які потрібно реалізувати в додатку для ефективного управління особистими фінансами.

Згідно з проведеним аналізом, функціонал додатку включатиме наступні можливості (рис.1.9).

Назва функції	Опис функції	Обґрунтування
Додавання витрат і доходів	Функція дозволяє користувачам вводити свої витрати і доходи, розподіляючи їх по відповідних категоріях. Витрати і доходи можна вказувати з точністю до дати та суми.	Ця функція є основою для будь-якого фінансового обліку. Вона дозволяє користувачам систематизувати свої фінанси та контролювати витрати та доходи.
Пошук за категорією, датою та назвою витрати	Дає змогу шукати витрати по категорії, конкретній даті та навіть назві витрати. Це дозволяє швидко знаходити необхідні транзакції.	Дуже зручна функція для користувачів, які хочуть швидко знайти конкретні витрати або доходи, наприклад, для аналізу витрат з картки.
Редагування транзакцій	Користувач може змінювати раніше введені дані про витрати чи доходи, змінювати категорії, дату та суму.	Це дозволяє коригувати введені дані при помилках або змінах у фінансовій ситуації, що робить додаток більш гнучким.
Створення фінансової цілі	Користувач може створювати цілі для накопичення коштів, наприклад, на покупку, подорож чи інші фінансові цілі. З кожним поповненням сума поповнення відображатиметься лінійною шкалою, яка заповнюється при досягненні певного рівня.	Це дозволяє користувачам ставити конкретні фінансові цілі, що мотивує їх до заощаджень та покращує фінансову дисципліну.
Перегляд стовпчикової діаграми витрат і доходів за місяць	Дозволяє переглядати витрати та доходи за певний місяць у вигляді діаграми. Показує загальну суму витрат і доходів, а також різницю між ними (баланс).	Це дає користувачу зручний спосіб для аналізу своїх фінансів за місяць, що дозволяє планувати подальші витрати та заощадження.

Рисунок 1.9 – Перелік основних функцій

Обґрунтування вибору функцій

1. Додавання витрат і доходів, це головна функція будь-якого фінансового додатку. Вона дає змогу користувачу вносити дані про свої фінанси, щоб потім їх аналізувати. Така можливість є в усіх аналогах, але я вирішила зробити її максимально зручною та зрозумілою завдяки простому інтерфейсу.

2. Пошук за категорією, датою та назвою. Дуже зручно, коли можна швидко знайти конкретну витрату або дохід, наприклад, за певною категорією чи вказаною датою. Не всі додатки мають таку функцію, тому її наявність робить аналіз набагато простішим.

3. Редагування транзакцій. У деяких додатках не можна змінювати вже введені дані, і це створює незручності. У моєму додатку буде можливість

редагувати транзакції, щоб виправити помилки або оновити інформацію — це дає користувачам більше контролю.

4. Створення фінансової цілі, ця функція дозволяє користувачу накопичувати гроші на конкретну мету, наприклад, на відпустку або покупку. Вона не дуже поширена серед конкурентів, тому додає унікальності моєму застосунку.

5. Діаграма доходів і витрат, завдяки стовпчиковим діаграмам користувач може краще побачити свою фінансову ситуацію за місяць. У більшості додатків є базові звіти, але не завжди є зручні графіки — я вирішила це врахувати.

6. Баланс витрат і доходів. Користувач зможе бачити різницю між тим, що заробляє, і тим, що витрачає. Це допомагає краще розуміти свою фінансову стабільність та планувати витрати. Для візуалізації основних функцій додатку створено UML-діаграму варіантів використання Use Case (рис.1.10).

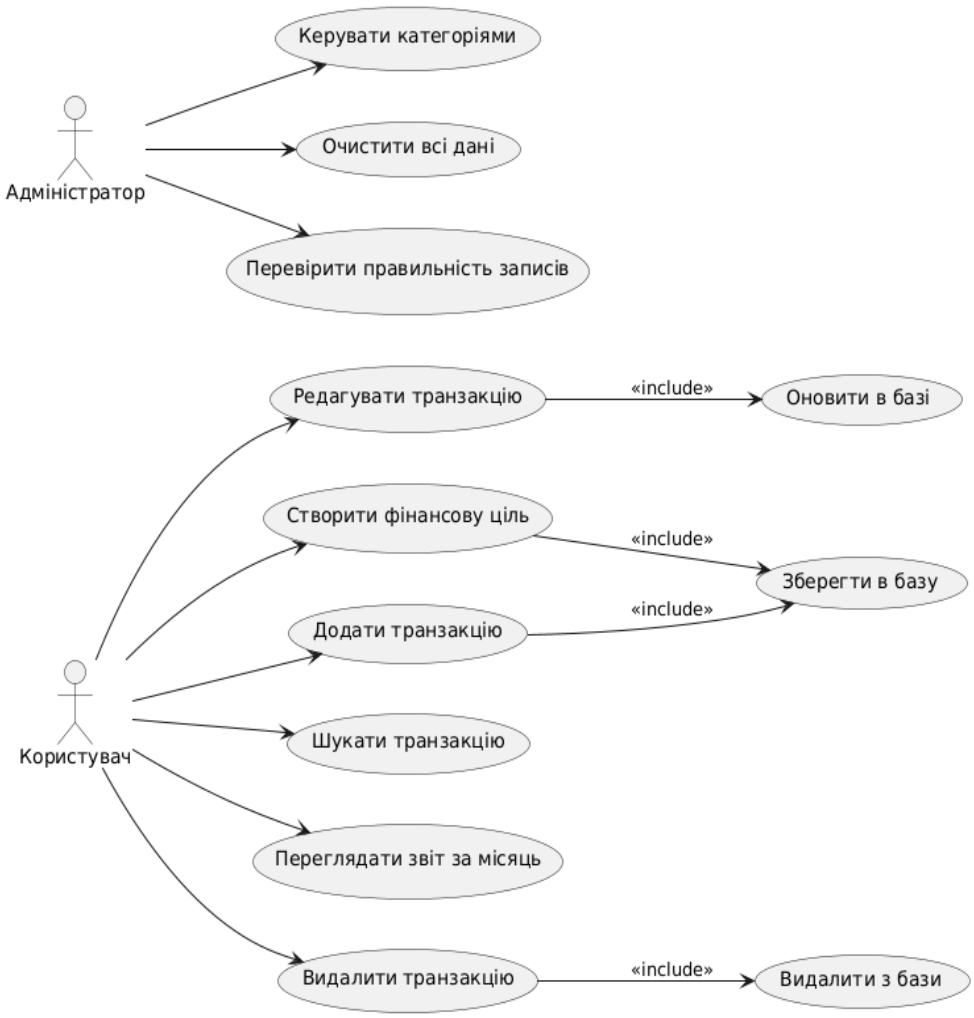

Рисунок 1.10 – Діаграма Use Case

Мета створення мобільного додатку «Finance Manager» — дати користувачеві зручний інструмент для ведення особистого бюджету. За його допомогою можна легко контролювати доходи й витрати, а також аналізувати свої фінансові звички. Додаток створений для індивідуального користування та працює навіть без підключення до інтернету. Це дозволяє користувачеві мати доступ до своїх фінансів у будь-який момент, незалежно від того, є зв’язок чи ні.

Застосунок - це користувацька програма, що дає змогу вирішувати конкретні прикладні задачі користувача. Поняття введене, щоб підкреслити відмінність від операційної системи, драйверів, бібліотек тощо та засобів і середовищ розробки [8].

Головні функціональні завдання, які реалізує додаток:

- зручне внесення та збереження даних про витрати та доходи користувача;
- сортування транзакцій за категоріями, датами та сумами;
- відображення статистики;
- побудова наочних графіків і діаграм для аналізу фінансової активності.

До вимог щодо інтерфейсу користувача належать:

- простота у використанні та інтуїтивно зрозуміле управління;
- логічна структура навігації та меню;
- приємне поєднання кольорів, яке не перевантажує зір;
- якісні візуальні елементи;
- адаптивність до різних типів екранів.

У цьому розділі було проведено дослідження теми, проаналізовано основні проблеми у сфері управління фінансами, які можна вирішити за допомогою програмного забезпечення. Також були визначені потреби майбутніх користувачів, проаналізовано ринок мобільних додатків і його сучасні тенденції. Окремо розглянуто сильні й слабкі сторони вже існуючих програм, які працюють у цій сфері. Уся зібрана інформація допомогла сформувати перелік функціональних вимог до майбутнього додатку, який має вирішувати реальні проблеми у сфері особистих фінансів.

1.8 Висновки

У першому розділі було проаналізовано предметну область управління особистими фінансами, зокрема досліджено рівень фінансової грамотності населення, який, за статистичними даними, залишається низьким. Це підтверджує актуальність розробки інструментів для спрощеного фінансового контролю. Було проведено огляд існуючих мобільних додатків — Monefy, YNAB, Goodbudget, Spendee — з метою виявлення їхніх функціональних можливостей та недоліків. Серед основних обмежень проаналізованих рішень було виявлено: відсутність української мови, складний або перевантажений інтерфейс, обмежена безкоштовна версія, відсутність або слабка реалізація функцій аналітики та постановки фінансових цілей. На основі цього було визначено вимоги до майбутньої системи: простота використання, повна офлайн-робота, підтримка постановки фінансових цілей, категоризація доходів і витрат та доступна аналітика.

2 ВИБІР ІТ-ІНФРАСТРУКТУРИ ТА ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Вибір методу рішення

За даними Statista, у 2021 році понад 6,2 мільярда людей користувалися мобільними телефонами з доступом до інтернету. До кінця 2022 року ця цифра зросла до приблизно 6,57 мільярда, що становить 83% усього населення світу. Це на 308 мільйонів більше, ніж роком раніше (рис. 2.1). Такий швидкий ріст кількості користувачів мобільних пристроїв призводить до зростання попиту на мобільні додатки.

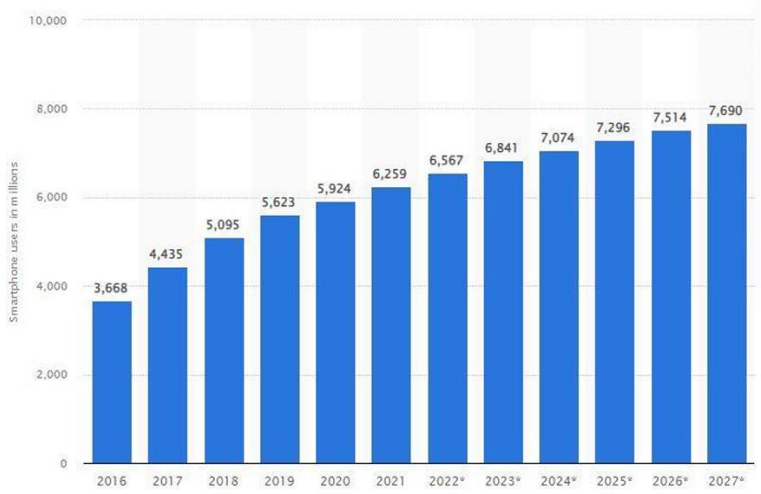

Рисунок 2.1 – Кількість користувачів телефонів з доступом до інтернет

Зважаючи на те, що попит на мобільні додатки постійно зростає, було вирішено створити саме мобільний застосунок для вирішення поставлених задач. Щоб задовольнити потреби користувачів в універсальному рішенні, у додатку будуть поєднані функції збереження фінансових записів та аналізу особистих витрат в одному зручному інтерфейсі.

Вибір операційної системи.

Оскільки Android та iOS є найпопулярнішими операційними системами для смартфонів, для розробки мого додатку я обрала саме Android. За даними Statista, 7 із 10 телефонів у світі працюють на Android. Хоча за останні п’ять років він

трохи втратив частку ринку (приблизно 8%), ця система все одно залишається лідером серед мобільних платформ.

2.2 Аналіз та вибір технологій і методів реалізації застосунку

Після вивчення теми та аналізу предметної області було вирішено створити нативний мобільний додаток для Android. Це дозволить зробити застосунок більш зручним і функціональним, а також краще використовувати ресурси самого пристрою.

Для реалізації програми було обрано Android Studio — це офіційне середовище для розробки Android-додатків. Воно має всі потрібні інструменти для створення, тестування та налагодження програми. Android Studio підтримує зручний візуальний редактор інтерфейсу, автозаповнення коду, інтеграцію з Git і дає змогу створювати додатки, які добре працюють на різних екранах і версіях Android.

Розробка мобільних додатків відбувається за допомогою спеціальних інструментів та програмних платформ, таких як Android Studio, Xcode, React Native, Flutter та інші. Основні мови програмування для розробки мобільних додатків - Java, Kotlin, Swift, Objective-C, JavaScript [9].

Для розробки нативних Android - додатків найчастіше використовуються мови програмування Kotlin та Java. Kotlin - це більш сучасна мова, яка вважається зручною альтернативою Java. Але Java має довгу історію, хорошу підтримку та багато готових бібліотек, тому її й досі широко використовують. Java залишається однією з найпопулярніших мов для Android і забезпечує стабільну та надійну роботу додатків.

Порівняння Java та Kotlin (рис. 2.2):

Характеристи-ка	Java	Kotlin
Популярність	Має великий досвід використання і підтримку. Один із стандартів для Android.	Нова мова, яка швидко набирає популярність. Підтримка від Google з 2017 року.
Сумісність	Повністю сумісна з існуючими бібліотеками Android.	Сумісна з Java, підтримує інтеграцію з існуючим кодом на Java.
Легкість у використанні	Може бути більш багатослівною і вимогливою в порівнянні з Kotlin.	Менше коду, більш компактний і легший для розуміння.
Кількість коду	Потрібно писати більше коду для реалізації однієї задачі.	Меньше коду завдяки функціям та можливостям Kotlin.
Підтримка бібліотек	Величезна кількість бібліотек та інструментів.	Кількість бібліотек зростає, але ще менша порівняно з Java.
Продуктивність	Швидка робота та стабільність.	Подібна до Java, хоча інколи Kotlin може бути трохи повільнішим.

Рисунок 2.2 – Таблиця порівняння Java та Kotlin

Java була обрана як основна мова програмування для розробки додатку через її стабільність, широке розповсюдження та повну сумісність із платформою Android. Вона забезпечує всі необхідні можливості для реалізації бізнес-логіки додатку, роботи з базами даних, а також побудови об'єктно-орієнтованої архітектури, що є важливою складовою якісного програмного забезпечення. Завдяки віртуальній машині JVM код залишається переносимим і легко підтримується [10]. У додатку Java використовується, зокрема, у класах для взаємодії з базою даних і обробки користувацьких дій.

Для збереження даних про доходи та витрати було обрано вбудовану систему керування базами даних — SQLite. Вона є простим, легким та ефективним рішенням, підходить для мобільних пристроїв. SQLite дозволяє створювати локальну базу даних без потреби у підключенні до віддалених серверів, це забезпечує автономну роботу додатку. Система підтримує всі необхідні базові операції з даними (створення, читання, оновлення, видалення).

Основні переваги SQLite:

– Легка вага, тому що, SQLite є вбудованою базою даних, тому не потрібно встановлювати або налаштовувати сервер.

– SQLite дуже швидка для роботи з локальними даними

– Широка підтримка, більшість мобільних платформ, включаючи Android, мають вбудовану підтримку для роботи з SQLite, це забезпечує простоту інтеграції в додаток.

– Багатоплатформність: SQLite працює на багатьох платформах, включаючи Android, iOS, Windows, macOS та інші [11].

Таке рішення дозволяє зберігати важливі дані користувача - безпосередньо на пристрої. Це забезпечує зручність використання, високу швидкість доступу до даних і незалежить від інтернет-з'єднання.

Отже, вибір Java як мови програмування та SQLite як системи керування базами даних є цілком обґрунтованим і логічним. Вони відповідають основним вимогам до стабільності, масштабованості та продуктивності мобільного застосунку.

2.3 Схема IT-інфраструктури інформаційної системи

IT-інфраструктура створеної інформаційної системи

Розроблена інформаційна система функціонує як автономна клієнтська інформаційна система, що не вимагає серверної частини або хмарної інфраструктури для базової роботи. Застосунок реалізовано у вигляді нативного Android-додатку, який розгортається безпосередньо на пристрої користувача.

База даних реалізована за допомогою вбудованої системи SQLite, що дозволяє зберігати фінансові дані локально без підключення до віддаленого сервера, що підвищує стійкість і безпеку.

Безпека забезпечується ізоляцією даних у середовищі Android, де кожен застосунок має власну захищену область пам’яті. Дані не передаються через мережу, що виключає перехоплення або втрату особистої інформації внаслідок мережевих атак.

У випадку розширення функціоналу (наприклад, реалізації синхронізації між пристроями), можливе підключення хмарних сервісів, таких як Firebase або Google Cloud, для зберігання резервних копій та аналітики.

Застосунок підтримує логування ключових дій користувача (через Logcat), що може бути використано під час тестування або в процесі супроводу системи. У майбутньому передбачено додавання модулів збору аналітики для оптимізації взаємодії.

Схематично IT-інфраструктура додатку представлена як однофакторна модель (рис.2.3).

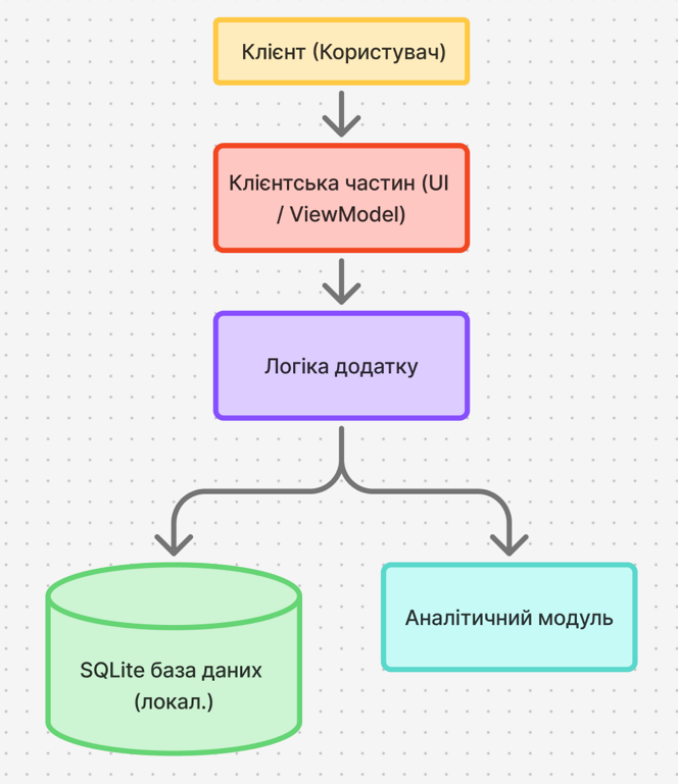

Рисунок 2.3 – Структура системи

Така архітектура є простою, стійкою до відмов, не потребує додаткових витрат на хостинг і забезпечує достатній рівень безпеки для індивідуального користування.

2.4 Архітектура та UML-діаграми інформаційної системи

Для наочного відображення логіки функціонування розроблюваної інформаційної системи було використано нотацію UML (Unified Modeling Language), яка дозволяє візуалізувати структуру, поведінку та взаємодію компонентів системи на різних етапах її роботи. Це допомогло краще уявити, як працюватиме додаток, і заздалегідь врахувати можливі помилки.

Під час роботи над системою я створила кілька UML-діаграм, які відображають основні частини архітектури додатку. Вони допомогли візуально побачити, як реалізовані функції, як користувач взаємодіє із застосунком, і як відбувається передача даних між різними його частинами.

Unified Modeling Language (UML) – це стандартна мова моделювання. Вона використовується для візуалізації, специфікації, конструювання та документування програмних систем [12].

Для повного уявлення про логіку роботи додатку було створено такі UML-діаграми:

Діаграма станів.

Діаграма станів є графічним представленням поведінки системи. Вона демонструє, як система переходить між різними станами у відповідь на певні події або умови. Такий тип діаграми широко використовується в програмуванні та системному аналізі для моделювання динамічних процесів. Діаграма містить стани (етапи функціонування системи), переходи між ними, тригери (події, що спричиняють зміну стану), умови та відповідні дії (рис. 2.4).

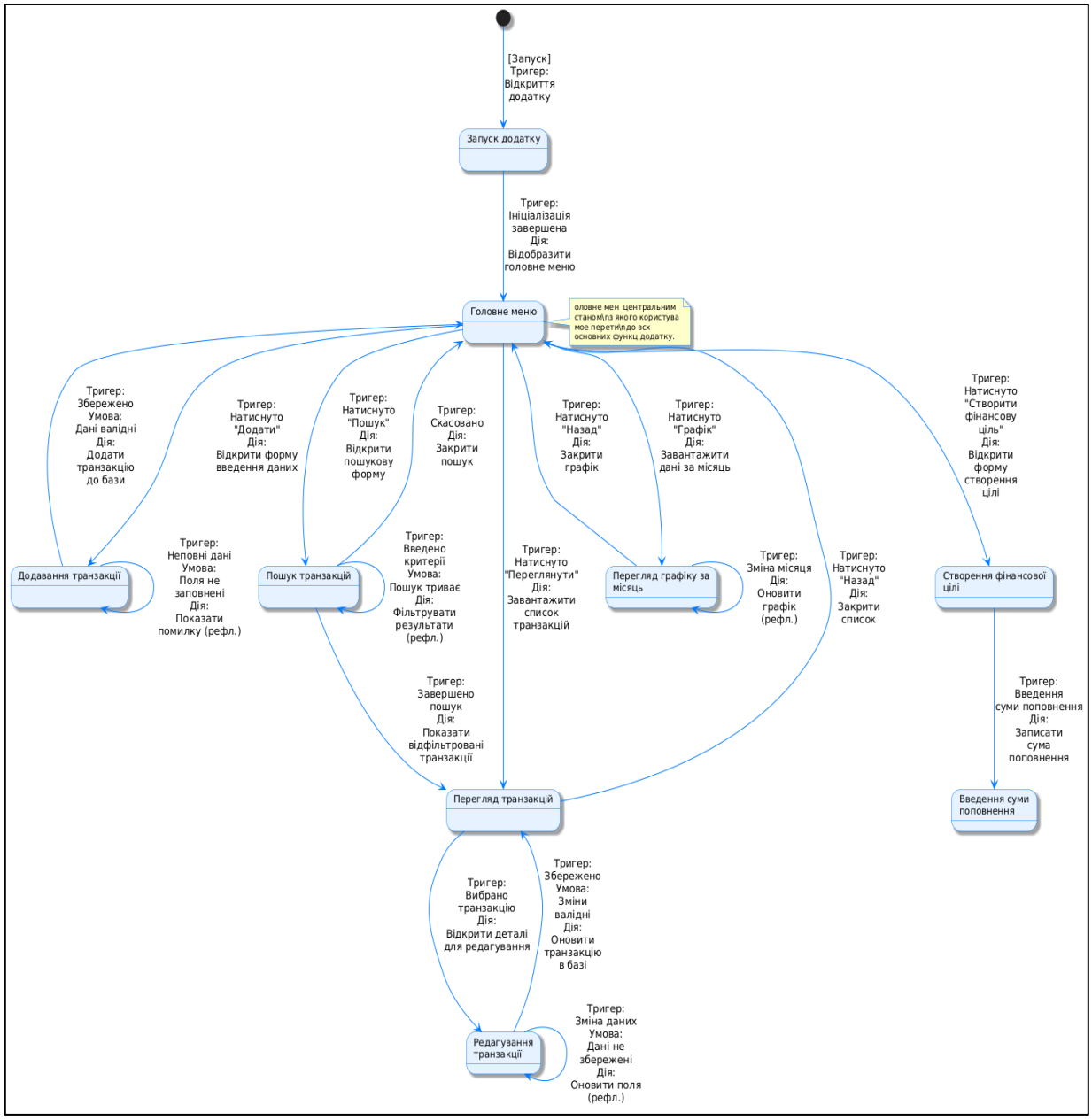

Рисунок 2.4 – Діаграма станів для опису процесу функціонування додатку управління особистими фінансами і аналізу витрат

Запуск додатку. На початковому етапі відбувається запуск додатку, який запускає процес завантаження та ініціалізації. Цей етап є першим у взаємодії з програмою, під час якого система готується до подальшого використання.

Головне меню. Після завантаження додатку відкривається головне меню, що надає доступ до всіх основних функцій. Це центральний елемент навігації, через який можна переходити до додавання транзакцій, перегляду витрат, створення фінансових цілей тощо.

Додавання транзакції. Для створення нової транзакції користувач обирає відповідний пункт меню, після чого відкривається форма для введення даних про витрати або доходи. У разі відсутності обов’язкових полів система повідомляє про помилку. Якщо всі дані введено коректно, інформація зберігається в базі даних, після чого відбувається повернення до головного меню.

Перегляд транзакцій. Ця функція дозволяє переглядати список внесених транзакцій. У разі потреби надається можливість редагування даних — суми, дати, категорії тощо. Після завершення перегляду або редагування здійснюється повернення до головного меню.

Пошук транзакцій. Реалізовано функцію пошуку за критеріями: дата, категорія, назва. Після введення параметрів система виконує фільтрацію й відображає лише ті записи, які відповідають заданим умовам. За потреби користувач може перейти до редагування або повернутись до головного меню.

Створення фінансової цілі. Одна з ключових функцій — можливість створення фінансової цілі. У відповідній формі можна ввести назву мети, бажану суму накопичення та поступово фіксувати поповнення. Додаток відображає прогрес і дозволяє контролювати досягнення встановленої цілі.

Перегляд графіку за місяць. Передбачено перегляд доходів і витрат за певний місяць у вигляді графіка. Можна обрати конкретний місяць для аналізу фінансової активності протягом цього періоду.

Побудова графіків за обраний місяць. Для більш наочного аналізу реалізовано побудову графіків, що відображають зміни доходів і витрат за обраний місяць. Така функція допомагає краще зрозуміти фінансову поведінку та спланувати майбутні витрати.

Діаграма класів

Діаграма класів (рис.2.5) для мобільного додатку для управління особистими фінансами включає різноманітні класи, які забезпечують функціональність додатку. Кожен клас має певні атрибути та методи, які взаємодіють між собою, що дає змогу ефективно працювати з фінансовими даними, здійснювати операції з доходами та витратами, а також створювати та відслідковувати фінансові цілі.

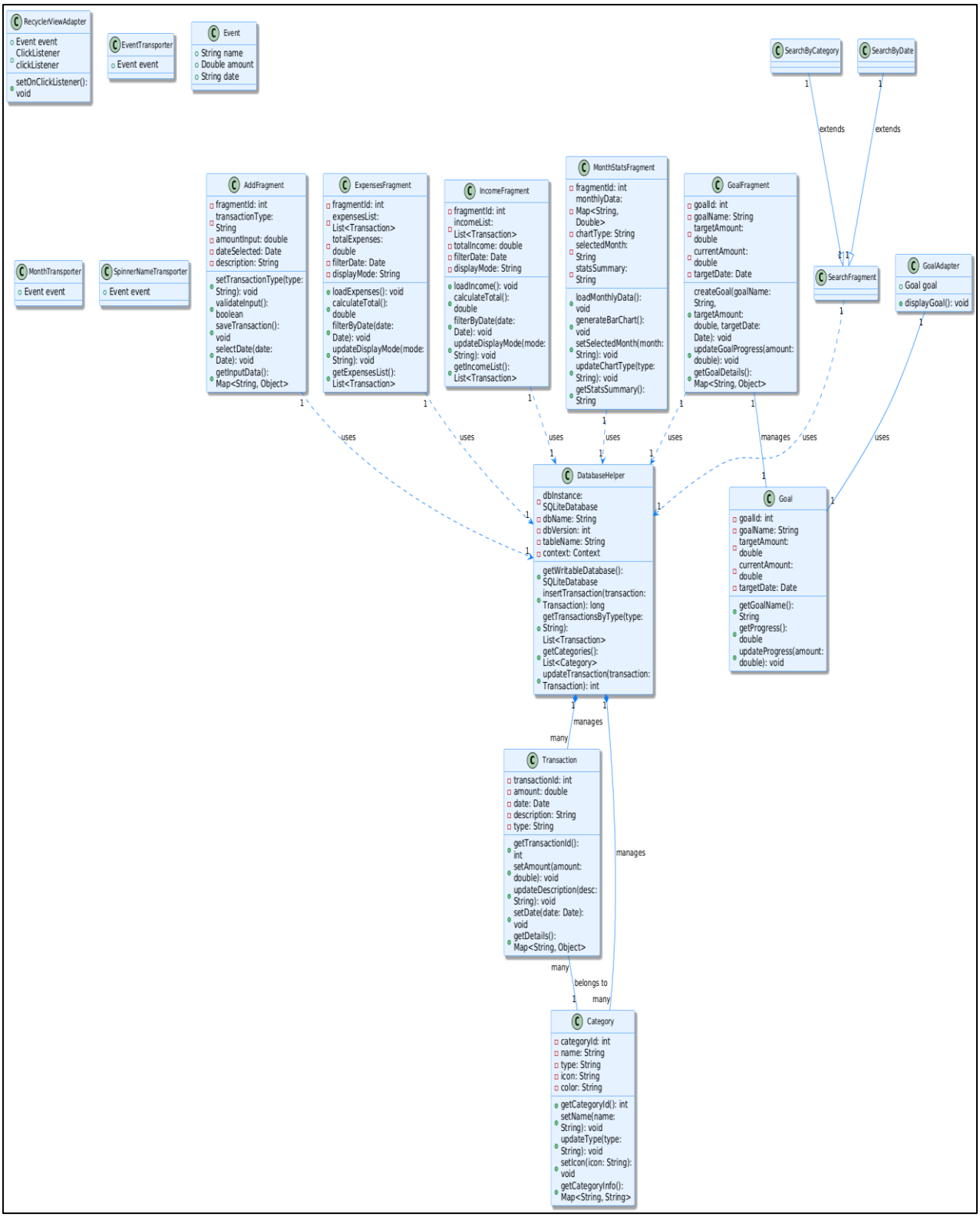

Рисунок 2.5 – Діаграма класів (Class Diagram)

Основні класи:

Клас	Атрибути	Методи	Опис
Transaction	transactionId: int, amount: double, date: Date, description: String, type: String	getTransactionId(), setAmount(), updateDescription(), getDetails()	Клас для зберігання та обробки даних про транзакції: сума, дата, тип (дохід або витрата).
Category	categoryId: int, name: String, type: String, icon: String, color: String	getCategoryId(), setName(), updateType()	Клас для організації категорій транзакцій: дозволяє групувати витрати й доходи за категоріями.
DatabaseHelper	dbInstance: SQLiteDatabase, dbName: String, dbVersion: int	getWritableDatabase(), insertTransaction(), getTransactionsByType(), getCategories(), updateTransaction()	Клас для роботи з базою даних SQLite: додавання, оновлення, отримання транзакцій і категорій.
GoalFragment	goalName: String, targetAmount: double, currentAmount: double, targetDate: Date	createGoal(), updateGoalProgress(), getGoalDetails()	Клас для створення, зберігання та оновлення фінансових цілей користувача.
Goal	goalName: String, targetAmount: double, currentAmount: double, targetDate: Date	getGoalName(), getProgress(), updateProgress()	Клас зберігає інформацію про фінансову ціль і дозволяє оновлювати прогрес накопичення.
GoalAdapter	–	displayGoal()	Клас для відображення цілей у UI. Забезпечує виведення даних у зручному для користувача вигляді.

Рисунок 2.6 – Таблиця основних класів

Відношення між класами:

1. Клас Transaction має зв'язок з Category (множина транзакцій належить до однієї категорії).

2. DatabaseHelper управляє транзакціями та категоріями, дозволяючи зберігати та оновлювати дані в базі.

3. GoalFragment взаємодіє з класами Goal і GoalAdapter, створюючи цілі та оновлюючи їх прогрес.

4. Інші фрагменти, такі як ExpensesFragment, IncomeFragment, та MonthStatsFragment, також використовують DatabaseHelper для взаємодії з даними та відображення інформації в інтерфейсі.

Висновки. Діаграма класів допомагає впорядкувати структуру коду мобільного додатку для управління особистими фінансами. Вона чітко показує взаємозв’язки між класами та їхню участь у виконанні основних операцій — з

транзакціями, категоріями та фінансовими цілями. Така організація дозволяє забезпечити логічну побудову програми, спрощує її розширення та обслуговування, а також сприяє ефективному обліку фінансів та кращому плануванню особистого бюджету.

Діаграма станів. Існують два основні типи діаграм взаємодії — діаграми послідовності та діаграми кооперації. Обидва типи відображають один і той самий процес, але з різних точок зору.

Далі буде розглянуто діаграму послідовності (рис.2.7), яка демонструє, як саме реалізується один із ключових процесів у додатку - додавання транзакції. Ця діаграма наочно показує порядок взаємодії між об’єктами в рамках виконання даної операції.

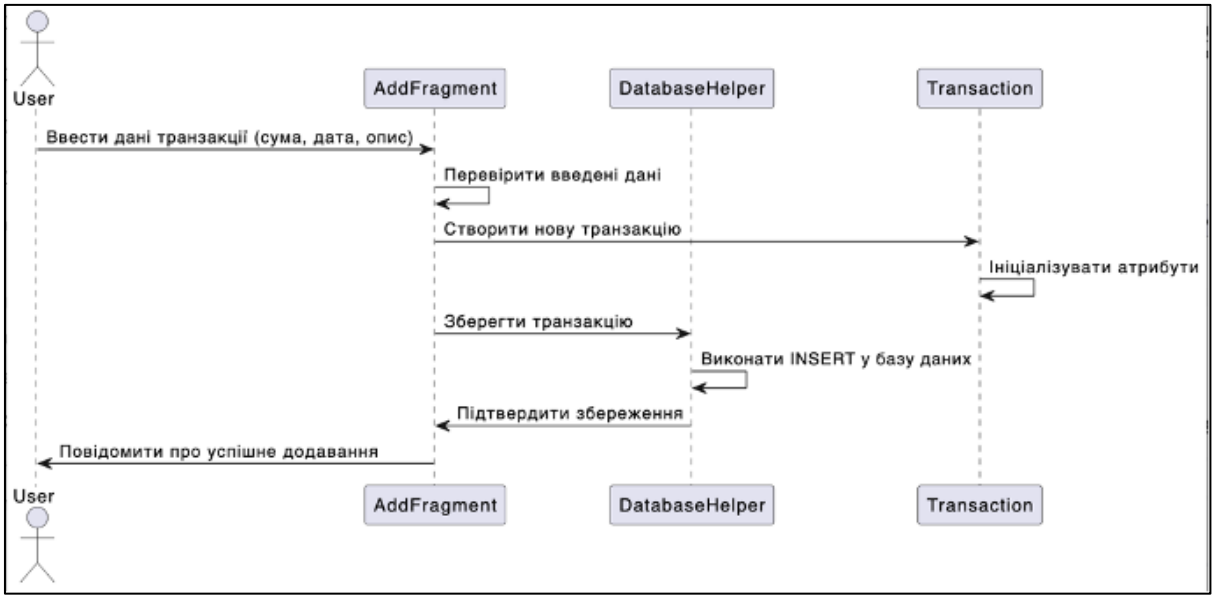

Рисунок 2.7 – Діаграма станів для процесу додавання транзакції

Ця діаграма показує, як у додатку Finance Manager відбувається процес додавання нової транзакції. Користувач вводить потрібні дані через екран додавання (AddFragment). Далі ці дані перевіряються, з них створюється об’єкт транзакції (Transaction), який передається в DatabaseHelper для збереження в базі даних. Після успішного збереження користувач бачить повідомлення про те, що транзакція додана.

Діаграма послідовності для пошуку транзакцій (рис.2.8).

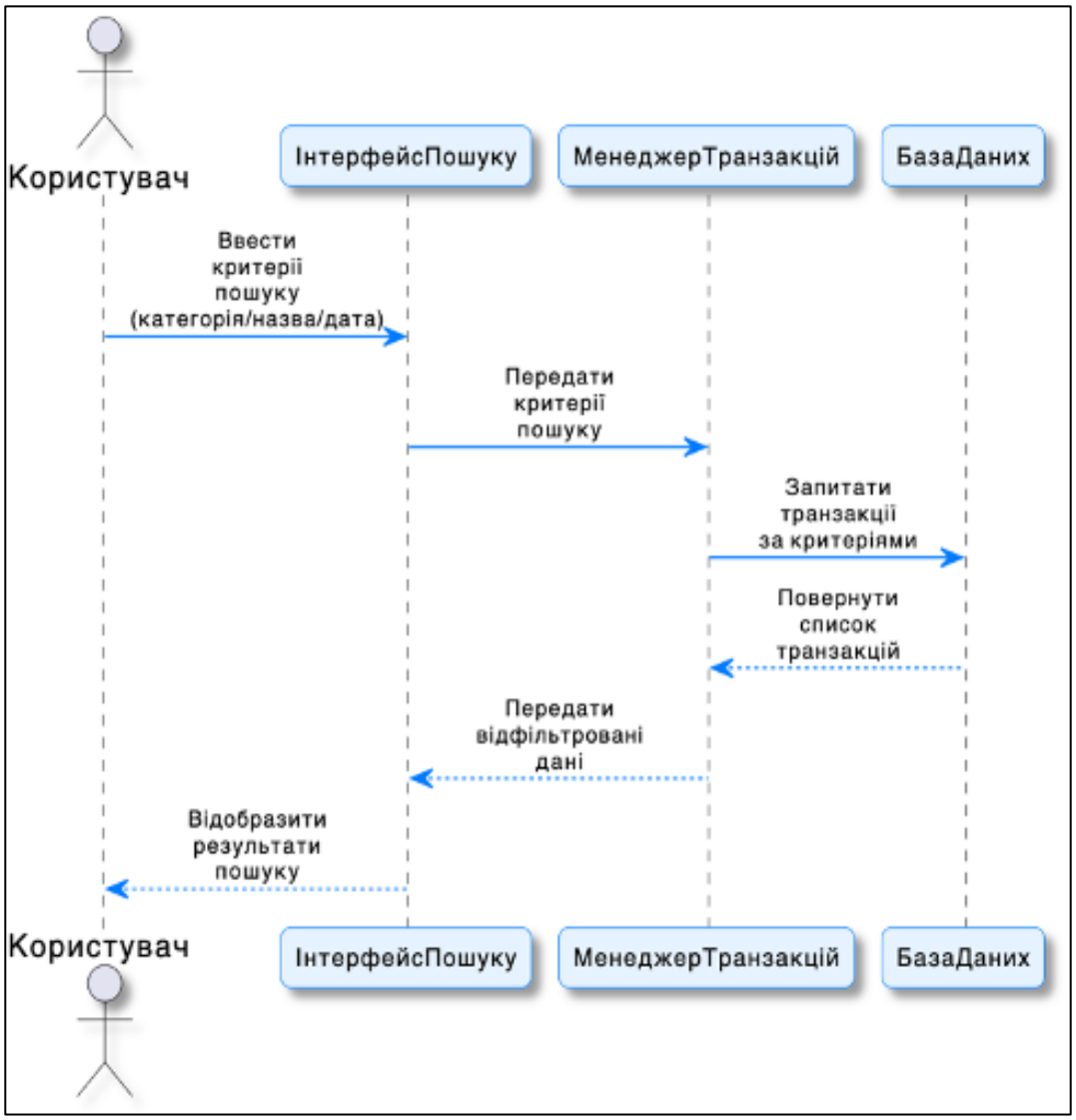

Рисунок 2.8 – Діаграма послідовності для процесу пошуку транзакцій

Ця діаграма показує, як працює пошук транзакцій за категорією, назвою або датою. Користувач вводить потрібні дані через інтерфейс пошуку. Потім ці дані передаються менеджеру транзакцій, який звертається до бази даних, отримує список транзакцій, фільтрує їх за заданими критеріями і повертає результати назад у пошуковий інтерфейс для відображення користувачу (рис.2.9).

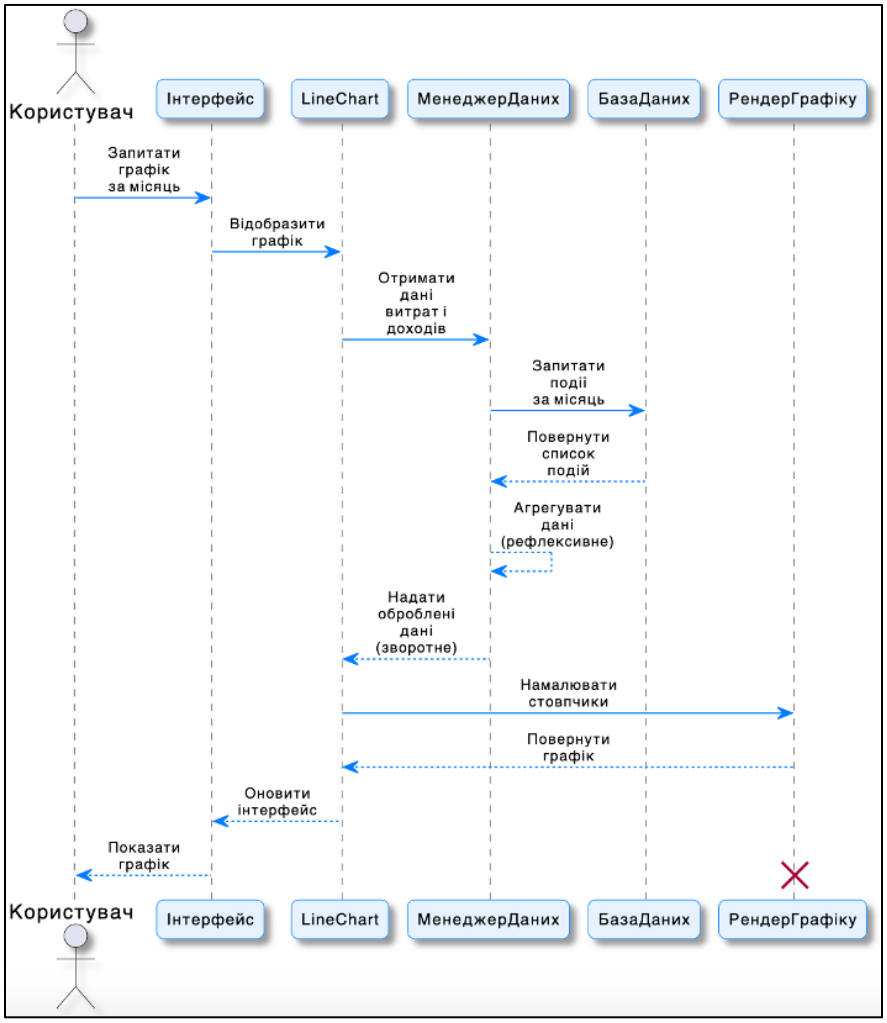

Рисунок 2.9 – Діаграма послідовності для процесу перегляду графіка

Діаграма послідовності показує, як у додатку відбувається процес відображення графіку витрат і доходів за місяць. Усього діаграма включає 11 повідомлень, які охоплюють основні етапи: запит від користувача, отримання даних з бази, їх обробка, побудова графіку та відображення результату на екрані.

Діаграма діяльності. Діаграми діяльності відображають послідовність дій, що виконуються під час роботи з певною функцією додатку або всієї системи в цілому. Вони дуже схожі на звичайні блок-схеми алгоритмів. Як і діаграми станів, ці діаграми показують процес у вигляді графу: дії — це вершини, а переходи між діями — ребра.

У проєкті було виділено три основні сценарії для побудови діаграм діяльності:

– Додавання нової транзакції (рис.2.10).
– Пошук транзакцій.

– Перегляд графіку за місяць (рис.2.11).

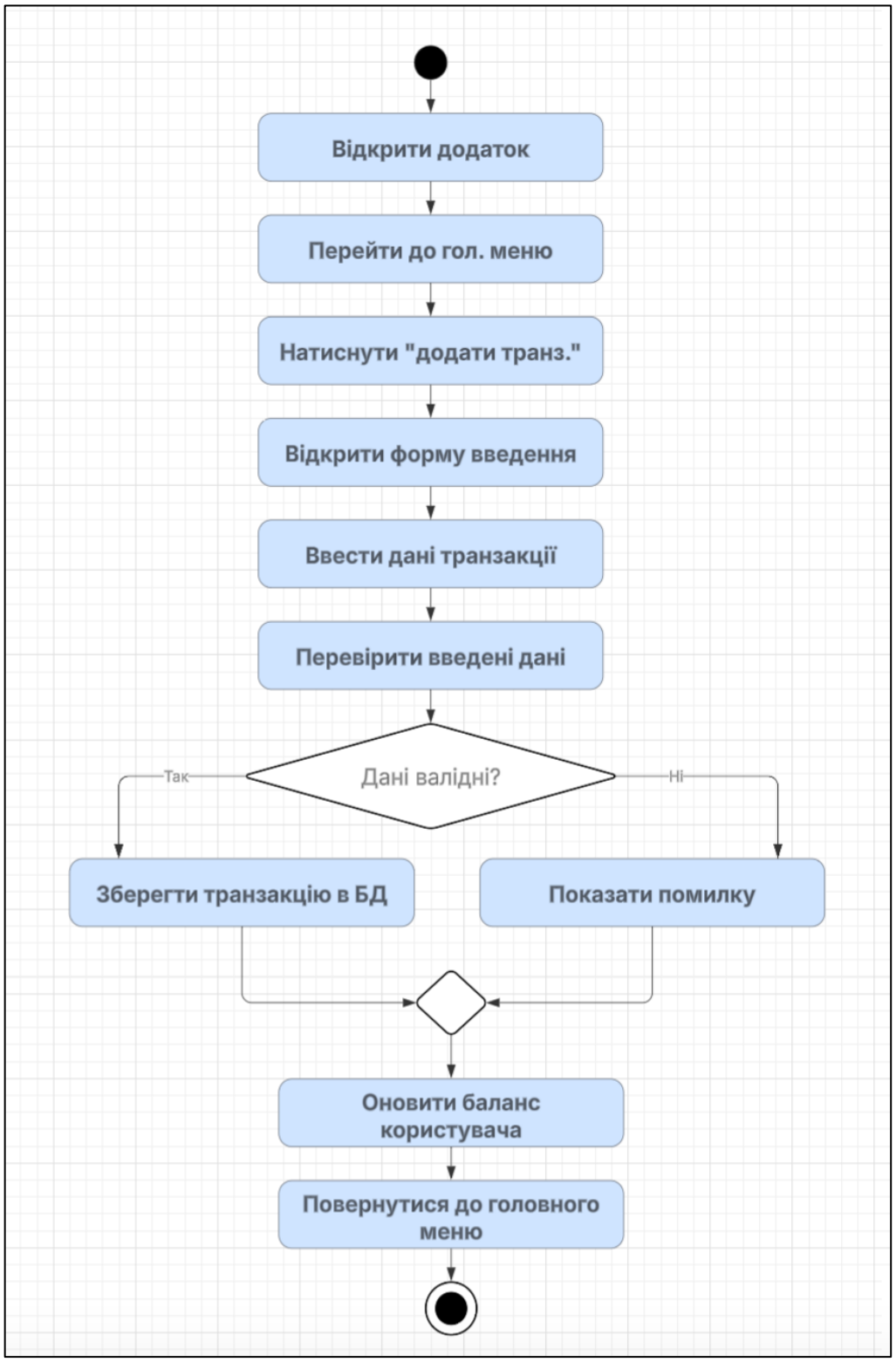

Рисунок 2.10 – Діаграма активності - додавання нової транзакції

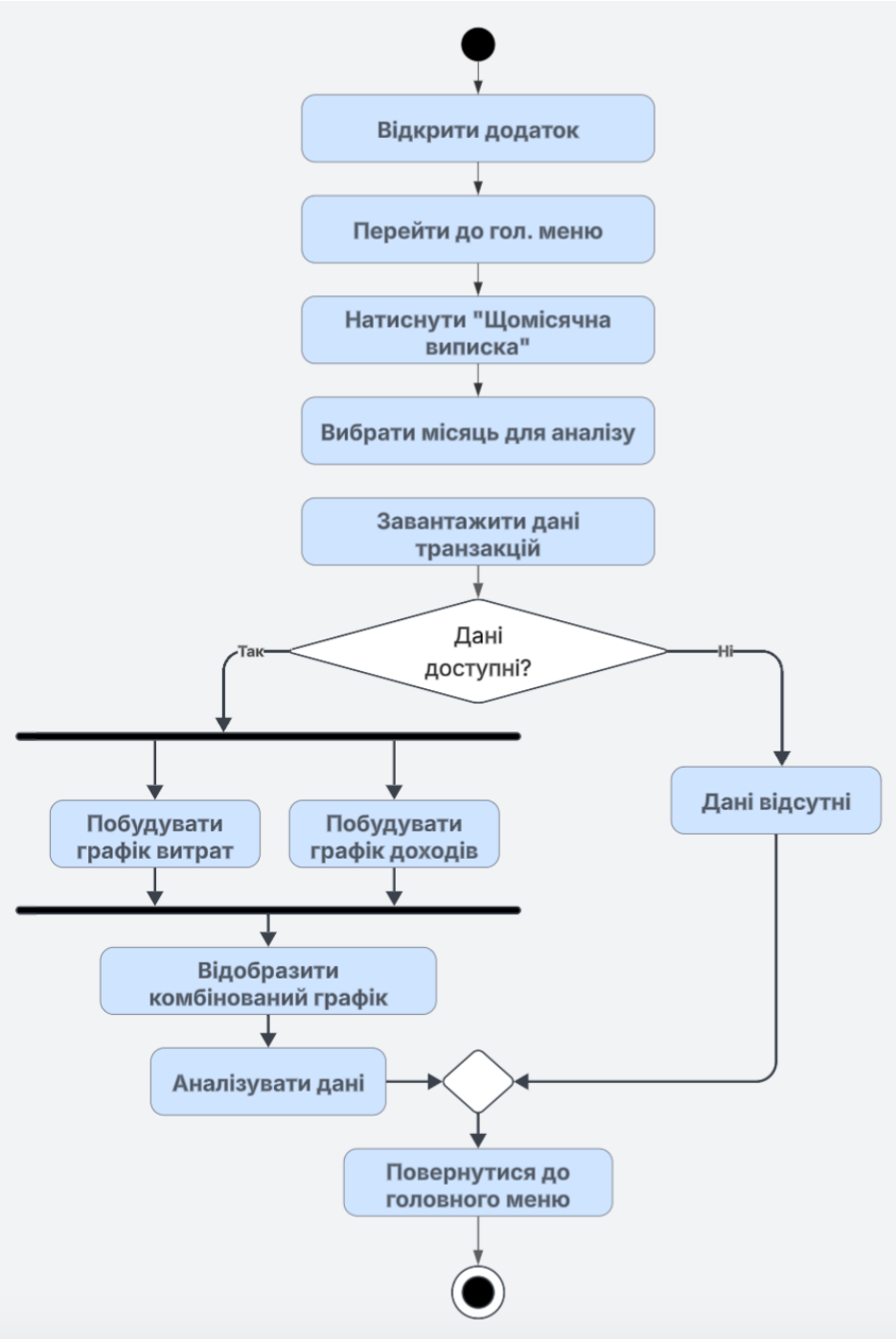

Рисунок 2.11 – Діагама активності - перегляд графіку за місяць

Діаграми відіграють важливу роль у розробці програмного забезпечення, оскільки дають змогу наочно показати, як працює система, з чого вона складається і як взаємодіють її частини. Вони допомагають краще зрозуміти логіку роботи додатку, як розробникам, так і аналітикам чи замовникам. Завдяки діаграмам легше планувати структуру проєкту, помічати слабкі місця ще до початку реалізації та уникати помилок, що дозволяє заощадити час і ресурси під час розробки.

Архітектура:

Для реалізації архітектури додатку обрано MVVM (Model-View-ViewModel) . Ця архітектура дозволяє розділяти логіку обробки даних (Model) від інтерфейсу користувача (View), а ViewModel забезпечує зв'язок між ними, обробляючи запити та взаємодіючи з моделями (рис.2.12) [13].

- Model: має бізнес-логіку, яка відповідає за обробку даних (записи про транзакції, розрахунки тощо).
- View: інтерфейс користувача (екрани додатку), що взаємодіє з ViewModel.
- ViewModel: зв'язує Model і View, передає дані з моделі до інтерфейсу та отримує зворотній зв'язок від користувача.

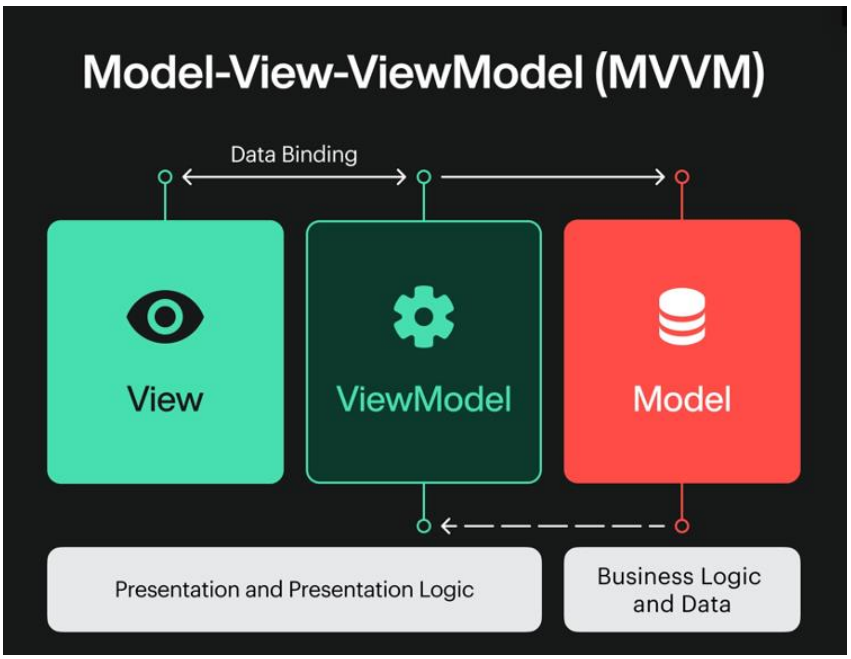

Рисунок 2.12 – схема моделі MVVM

2.5 Вибір апаратного забезпечення ІС

Апаратне забезпечення є важливою складовою будь-якої інформаційної системи, навіть якщо вона функціонує автономно, без серверної частини. У випадку створення мобільного додатку для управління особистими фінансами, основною платформою розгортання є мобільний пристрій користувача — смартфон або планшет з операційною системою Android.

Операційна система Android була обрана через її високу популярність серед користувачів — за даними StatCounter, станом на 2024 рік понад 70% мобільних пристроїв в Україні працюють саме на Android. Крім того, Android надає широкий вибір моделей пристроїв у різних цінових категоріях, що робить розроблену інформаційну систему доступною для більшості користувачів.

Завдяки використанню локальної бази даних SQLite та нативної розробки на мові Java, інформаційна система має невисокі вимоги до ресурсів пристрою. Це дозволяє їй працювати навіть на пристроях із базовими технічними характеристиками.

Мінімальні системні вимоги для роботи додатку:

– Операційна система: Android 8.1 (API Level 27);

– Процесор: 600 МГц (одноядерний);

– Оперативна пам’ять: 400 МБ;

– Вільна пам’ять: щонайменше 20 МБ для додатку та 1 МБ для збереження бази даних;

– Екран: 4.5 дюйма або більше, з роздільною здатністю не нижче 480×800 пікселів.

Рекомендована конфігурація для комфортної роботи:

– Android 10 або новіше;

– Багатоядерний процесор з частотою від 1.2 ГГц;

– Оперативна пам’ять: від 1 ГБ;

– Постійна пам’ять: не менше 100 МБ вільного простору;

– Дисплей з діагоналлю від 5.5 дюйма для кращого відображення графіків і діаграм.

На відміну від багатьох клієнт-серверних систем, розроблена інформаційна система не потребує спеціалізованого серверного обладнання чи хмарної інфраструктури. Вся логіка реалізована локально, а збереження даних здійснюється на пристрої користувача. Такий підхід знижує витрати на обслуговування, забезпечує приватність даних та дає змогу користуватись системою навіть без доступу до Інтернету.

Хоча система наразі реалізована як автономна, її архітектура передбачає можливість майбутнього масштабування — наприклад, додавання синхронізації з хмарними сервісами, використання банківських API або мультикористувацької підтримки. У такому випадку буде необхідне додаткове серверне середовище, проте базова версія залишиться доступною на будь-якому сучасному Android-пристрої.

Таким чином, для використання інформаційної системи не потрібно спеціального апаратного забезпечення — достатньо смартфона середнього рівня. Це робить систему універсальною, доступною та економічно вигідною для широкого кола користувачів.

2.6 Висновки

У другому розділі було здійснено проєктування інформаційної системи, яка реалізована у вигляді мобільного додатку. На основі аналізу було обґрунтовано вибір мови програмування Java, яка забезпечує сумісність із платформою Android, а також Android Studio як офіційного середовища розробки. Для зберігання даних обрано SQLite — локальну базу даних, яка не потребує мережевого підключення і відповідає меті створення автономного додатку. Визначено структуру системи: основні сутності, їх атрибути і взаємозв’язки. Побудовано низку UML-діаграм: варіантів використання, класів, активностей, послідовностей та станів, які відображають логіку функціонування застосунку, його структуру та можливі сценарії взаємодії з користувачем. Архітектура додатку побудована на основі підходу MVVM, що забезпечує розділення логіки, інтерфейсу користувача та доступу до даних, спрощує масштабування та супровід системи.

3 РОЗРОБЛЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Реалізація бази даних

Для розробки додатку було обрано базу даних SQLite. SQLite — це вбудована бібліотека, яка реалізує самодостатню, безсерверну базу даних, SQL без конфігурації [14]. SQLite є вбудованою базою даних, яку підтримують більшість мобільних операційних систем, зокрема Android. Однією з головних переваг її використання є висока швидкість і ефективність роботи з невеликими обсягами даних. Це робить її ідеальним варіантом для мобільних додатків, які мають зберігати інформацію про транзакції, фінансові цілі, категорії витрат тощо.

У базі даних використовуються дві основні таблиці: EVENTS_TABLE - для зберігання транзакцій, та GOALS_TABLE - для збереження даних про фінансові цілі. Кожна таблиця має свої унікальні поля, які допомагають зберігати потрібну інформацію та дозволяють зручно нею керувати.

У програмному коді бази даних реалізовано основні функції для роботи з цими таблицями: створення, додавання нових записів, оновлення, видалення, а також отримання даних для подальшого використання в інтерфейсі додатку. Нижче детально пояснюється, як саме працює кожна з цих частин.

Класи та методи бази даних

1. EVENTS_TABLE (Таблиця подій)

2. GOALS_TABLE (Таблиця цілей)

Метод onCreate (рис.3.1) викликається при створенні бази даних і відповідає за ініціалізацію таблиць.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String createTableStatement = "CREATE TABLE " + EVENTS_TABLE + " (" +
            COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_EVENT_PRICE + " TEXT, " +
            COLUMN_EVENT_NAME + " TEXT, " +
            COLUMN_EVENT_DATE + " TEXT, " +
            COLUMN_EVENT_CATEGORY + " TEXT)";

    String createGoalTableStatement = "CREATE TABLE " + GOALS_TABLE + " (" +
            COLUMN_GOAL_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_GOAL_NAME + " TEXT, " +
            COLUMN_GOAL_TARGET + " REAL, " +
            COLUMN_GOAL_CURRENT + " REAL)";

    sqLiteDatabase.execSQL(createTableStatement);
    sqLiteDatabase.execSQL(createGoalTableStatement);
```

Рисунок 3.1 – Фрагмент коду БД

Цей код створює дві таблиці: events_table та goals_table.

Основні методи роботи з базою даних

Додавання події. Метод addOneEvent (рис.3.2) додає нову транзакцію до таблиці подій.

```
public boolean addOneEvent(Event event){
    SQLiteDatabase database = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();

    contentValues.put(COLUMN_EVENT_NAME, event.getPlace_name());
    contentValues.put(COLUMN_EVENT_PRICE, event.getPrice());
    contentValues.put(COLUMN_EVENT_DATE, event.getDate());
    contentValues.put(COLUMN_EVENT_CATEGORY, event.getCategory());

    long insert = database.insert(EVENTS_TABLE, nullColumnHack: null, contentValues);

    if(insert == -1){
        return false;
    }else{
        return true;
```

Рисунок 3.2 – Фрагмент коду БД

Цей метод використовує клас ContentValues для зберігання значень і вставляє нову транзакцію в базу даних.

Оновлення події: Метод updateOneEvent дозволяє оновити транзакцію.

1) Видалення події: Метод deleteOneEvent (рис.3.3) видаляє запис з таблиці подій за вказаним ID.

```
public boolean deleteOneEvent(Event event){
    SQLiteDatabase database = this.getWritableDatabase();
    String query = "DELETE FROM " + EVENTS_TABLE + " WHERE " + COLUMN_ID + " = " + event.getId();

    Cursor cursor = database.rawQuery(query, selectionArgs: null);
```

Рисунок 3.3 – Фрагмент коду БД

2) Отримання всіх транзакцій: Метод getAllMoney (рис.3.4) отримує всі транзакції для поточного місяця.

```
public double getAllMoney(){
    double returnMoney = 0;

    Calendar cal = Calendar.getInstance();
    SimpleDateFormat month_date = new SimpleDateFormat( pattern: "MM");
    SimpleDateFormat year_date = new SimpleDateFormat( pattern: "yyyy");
    String month = month_date.format(cal.getTime());
    String year = year_date.format(cal.getTime());

    String query = "SELECT * FROM " + EVENTS_TABLE + " WHERE " + COLUMN_EVENT_DATE + " LIKE '" +
    SQLiteDatabase database = this.getReadableDatabase();
    Cursor cursor = database.rawQuery(query, selectionArgs: null);

    if(cursor.moveToFirst()){
        do{
            int id = cursor.getInt( columnIndex: 0);
            String price = cursor.getString( columnIndex: 1);
            String name = cursor.getString( columnIndex: 2);
            String date =cursor.getString( columnIndex: 3);
            String category = cursor.getString( columnIndex: 4);

            Event event = new Event(id, price, name, date, category);
            returnMoney += Double.parseDouble(event.getPrice());
```

Рисунок 3.4 – Фрагмент коду БД

3.2 Реалізація інтерфейсу користувача

Додаток «FinanceManager» розроблено з урахуванням зручності та простоти у використанні. Інтерфейс інтуїтивно зрозумілий, що робить його доступним навіть для тих, хто вперше користується подібними застосунками. Дизайн поєднує сучасний вигляд і практичність, забезпечуючи комфорт під час роботи.

Оформлення інтерфейсу витримане в світлих тонах із м’якими фіолетовими акцентами, які гармонійно підкреслюють важливі елементи, не перевантажуючи візуальне сприйняття. Для тексту використовується шрифт Open Sans, який добре читається та допомагає структурувати інформацію.

Основні фрагменти інтерфейсу:

- AddFragment - це екран, який використовується для додавання нової транзакції (доходу або витрати). Він відповідає за збирання введених користувачем даних і передачу їх у базу даних (рис.3.5). Під час завантаження фрагмента в методі onCreateView підключається макет інтерфейсу, а в onViewCreated — ініціалізуються всі необхідні поля вводу та кнопки. Після заповнення всіх полів і натискання кнопки створюється об’єкт типу Event, який передається до бази даних через клас DatabaseHelper.

```
public List<Event> getAllEventsByCategory(String query_name){
    List<Event> returnList = new ArrayList<>();

    Calendar cal = Calendar.getInstance();
    SimpleDateFormat month_date = new SimpleDateFormat( pattern: "MM");
    SimpleDateFormat year_date = new SimpleDateFormat( pattern: "yyyy");
    String month = month_date.format(cal.getTime());
    String year = year_date.format(cal.getTime());

    String query = "SELECT * FROM " + EVENTS_TABLE + " WHERE " + COLUMN_EVENT_CATEGORY + " LIKE
    SQLiteDatabase database = this.getReadableDatabase();
    Cursor cursor = database.rawQuery(query,  selectionArgs: null);

    if(cursor.moveToFirst()){
        do{
            int id = cursor.getInt( columnIndex: 0);
            String price = cursor.getString( columnIndex: 1);
            String name = cursor.getString( columnIndex: 2);
            String date =cursor.getString( columnIndex: 3);
            String category = cursor.getString( columnIndex: 4);
```

Рисунок 3.5 – частина коду фрагменту «AddFragment»

– EditFragment – екран редагування вже створених записів.

– ShowItemFragment – детальний перегляд інформації про одну транзакцію.

Аналітичні фрагменти:

– BilansFragment – загальний баланс за вибраний період.

– BilansExpensesFragment – баланс по витратах.

– BilansIncomeFragment – баланс по доходах.

– MonthStatsFragment – щомісячна статистика у вигляді графіка та сум.

Цей код описує фрагмент для відображення статистики за місяць у додатку для фінансового менеджменту (рис.3.6).

```
private void findViews(View view) {
    transactionCount = view.findViewById(R.id.transaction_count_monthStats);
    monthBilans = view.findViewById(R.id.money_month_stats);
    income = view.findViewById(R.id.income_month_stats);
    expense = view.findViewById(R.id.expenses_mont_stats);
    bilansCard = view.findViewById(R.id.bilans_card);
    incomeCard = view.findViewById(R.id.income_card);
    expenseCard = view.findViewById(R.id.expenses_card);
    lineChart = view.findViewById(R.id.line_chart);
    spinner = view.findViewById(R.id.month_spinner_selector);

    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(context, R.array.month_s
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);

    int position = adapter.getPosition(MonthTransporter.getMonth());
    if (position >= 0) {
        spinner.setSelection(position);
    }
```

Рисунок 3.6 – частина коду фрагменту MonthStatsFragment

Пошукові фрагменти:

– SearchFragment – головна сторінка пошуку з фільтрами.

– SearchByCategory, SearchByDate, SearchByName – спеціалізовані пошуки за конкретними параметрами.

– Облікові фрагменти:

– IncomeFragment – відображає список транзакцій доходу.

– ExpensesFragment – відображає список транзакцій витрат.

– EventsFragment – загальний список усіх подій (доходів і витрат).

Взаємодія між фрагментами в додатку здійснюється через головне меню або за допомогою кнопок всередині самих фрагментів.

Для передачі даних між екранами, наприклад, обраного місяця чи конкретної транзакції, використовуються спеціальні об'єкти - EventTransporter, MonthTransporter, SpinnerNameTransporter. Вони слугують як "транспортери" і дозволяють зручно передавати дані між фрагментами без втрати стану під час переходів.

Перший екран який ми бачимо після запуску додатку, перед тим, як відобразиться головний екран. На білому фоні розташований логотип додатку.

Наступна сторінка – це сторінка з навігаційним боковим меню, з якого і відбувається всі переходи між сторінками додатку.

Меню в додатку (рис.3.7) реалізовано за допомогою Navigation Component, що дозволяє зручно та логічно переміщатися між основними екранами. Завдяки цій системі користувач легко переходить між різними фрагментами без плутанини чи зайвих дій.

У меню доступні такі екрани:

– Головна сторінка — стартовий екран, де показано всі транзакції.

– Додати транзакцію — форма для внесення нового доходу або витрати.

– Витрати — перелік усіх витрат користувача.

– Доходи — список усіх надходжень.

– Редагування — дозволяє змінювати деталі вже створених транзакцій.

– Пошук — дає змогу знайти потрібну транзакцію за датою або категорією.

– Щомісячна статистика — відображає графік і підсумкову інформацію за вибраний місяць.

Детальний перегляд транзакції - відкривається при перегляді конкретної транзакції з детальною інформацією. Це меню дає змогу зручно управляти персональними фінансами, перемикаючись між ключовими функціями додатку в один клік.

```
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    app:startDestination="@id/events_fragment">
    <fragment
        android:id="@+id/events_fragment"
        android:name="pr.app.com.financemanager.fragments.EventsFragment"
        android:label="Finance Manager"
        tools:layout="@layout/fragment_events"/>

    <fragment
        android:id="@+id/add_fragment"
        android:name="pr.app.com.financemanager.fragments.AddFragment"
        android:label="Додати"
        tools:layout="@layout/fragment_add"/>

    <fragment
        android:id="@+id/expenses_fragment"
        android:name="pr.app.com.financemanager.fragments.ExpensesFragment"
        android:label="Витрати"
        tools:layout="@layout/fragment_expenses"/>
```

Рисунок 3.7 – Фрагмент коду «navigation.xml», реалізація меню

Наступна сторінка на яку користувач може перейти з меню – це «Додати». На цій сторінці можна додавати як витрати так і прибутки (рис. 3.9).

Дизайн виконано в світло-фіолетовому стилі, що відповідає загальному оформленню додатку. Всі елементи мають достатній відступ і висоту для зручного заповнення (рис.3.8).

Рисунок 3.8 – фрагмент коду, та демонстрація дизайну в середовищі “Android Studio”

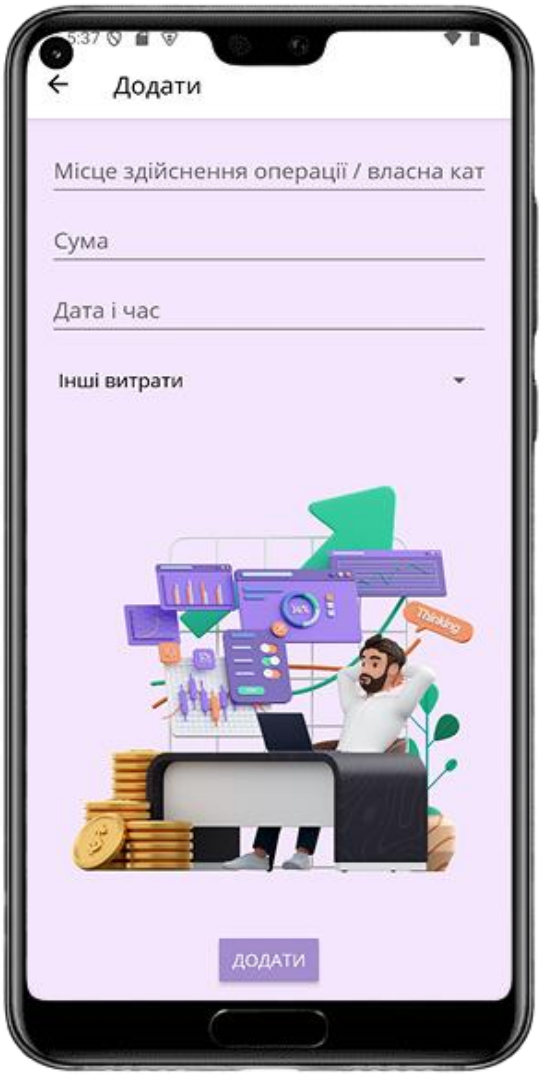

Рисунок 3.9 – Вигляд сторінки «Додати»

Сторінка "Пошук" дозволяє користувачам зручно знаходити потрібні фінансові операції. Вона реалізована у вигляді вкладок і складається з трьох основних частин:

- Пошук за назвою
- Пошук за категорією
- Пошук за датою

Сторінка має вкладений інтерфейс з використанням TabLayout і ViewPager, оформлена у світло-фіолетовому стилі з ілюстрацією на фоні (рис.3.10).

Рисунок 3.10 – Вигляд сторінки на емуляторі

Однією з найважливіших сторінок для аналізу витрат у додатку є «Щомісячна виписка» (рис.3.11). Вона показує короткий фінансовий підсумок за обраний місяць і допомагає користувачу швидко оцінити свою фінансову ситуацію.

Інтерфейс цієї сторінки зручний і зрозумілий. Він складається з кількох основних елементів. Випадаючий список (Spinner), він дає змогу вибрати потрібний місяць для перегляду статистики. Інформаційні картки, яка містять ключову інформацію про витрати, доходи та баланс за місяць. Графік, візуально відображає розподіл доходів і витрат, що полегшує аналіз даних.

Був створений кастомний компонент LineChart, який візуалізує фінансові дані за кожен день місяця. Він реалізований як розширення класу View.

Рисунок 3.11 – Вигляд сторінки на емуляторі

Сторінка «Фінансова ціль» дозволяє користувачу створити власну ціль для накопичення коштів і поступово поповнювати її на будь-яку суму (рис.3.12). На екрані відображається спеціальна лінія, яка поступово заповнюється кольором у міру накопичення грошей. Це не лише зручно, а й має мотиваційний ефект - візуальний прогрес стимулює швидше досягати поставленої фінансової мети.

Рисунок 3.12 – Вигляд сторінки та фрагмент коду “Фінансова ціль”

За такою ж логікою реалізовані і інші сторінки додатку (рис. 3.13).

Усі фрагменти додатку реалізовані за однією логікою: кожна сторінка має окремий XML-файл у папці res/layout. Назви файлів відповідають змісту фрагментів, наприклад: fragment_income.xml, fragment_expenses.xml. Компоненти UI (тексти, кнопки, графіки, списки тощо) розміщені згідно з концепцією зручного користування.

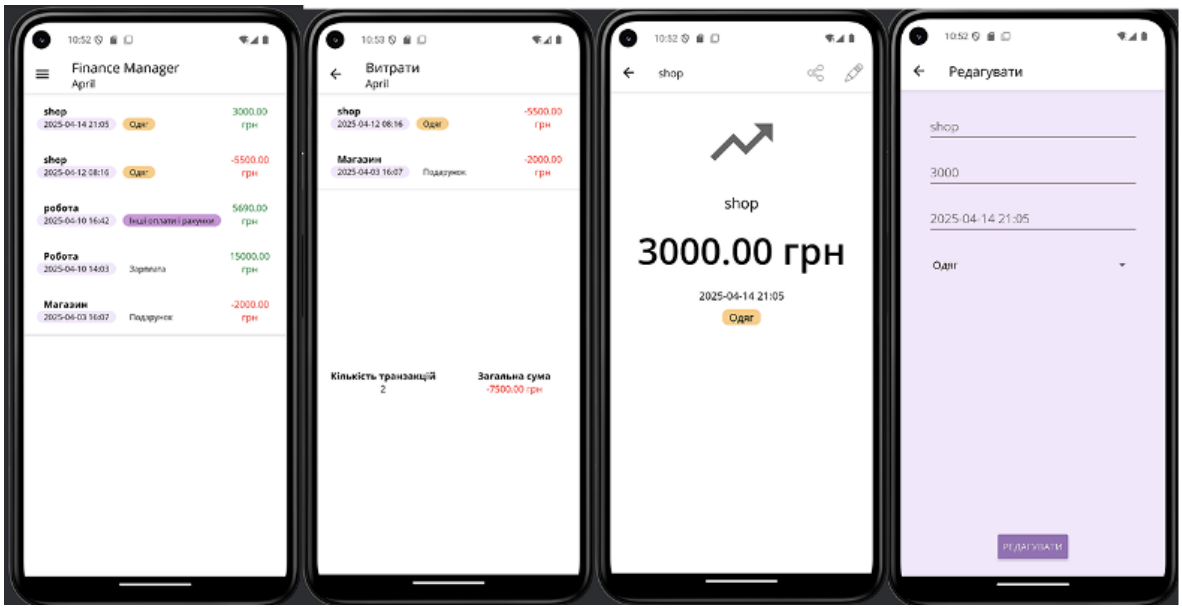

Рисунок 3.13 – Інші сторінки додатку

3.3 Тестування мобільного застосунку

Тестування мобільних додатків — важливий етап у процесі розробки, адже воно допомагає знайти помилки або недоліки, які можуть впливати на стабільність і зручність роботи додатку на різних пристроях та в різних умовах [15].

Для перевірки додатку з управління особистими фінансами було обрано три основні методи тестування, кожен з яких має свої особливості та підходи до перевірки функціоналу.

1) Функціональне тестування

Цей тип тестування дозволяє перевірити, чи працює додаток відповідно до вимог і описаних функцій. Для цього використовуються різні сценарії, які охоплюють основні дії користувача [16].

Було складено список ключових сценаріїв (рис.3.14), які допомагають перевірити найважливіші можливості додатку.

Ідентифікатор	Модуль	Мета	Вхідні значення	Очікуваний результат
К1	Додавання транзакції	Додати нову транзакцію	Назва: Продукти, Сума: 150, Категорія: Їжа	Транзакція додана до бази
К2	Пошук транзакцій	Пошук транзакцій за категорією	Категорія: Їжа	Виведення списку транзакцій по категорії
К3	Редагування транзакції	Оновлення суми транзакції	Транзакція: ID = 1, Нова сума: 200	Транзакція оновлена
Р1	Фінансова ціль	Створити фінансову ціль	Назва: Купити машину, Ціль: 50000	Ціль створена з початковим балансом
Н1	Звіт за місяць	Перегляд загальних витрат	Місяць: травень	Показ загальної суми витрат за місяць

Рисунок 3.14 – Таблиця з сценаріями тестування

Ці сценарії допомогли провести базове функціональне тестування всіх основних функцій додатку, зокрема введення та оновлення транзакцій, пошук за категоріями та датами, а також перегляд статистики витрат.

2) Тестування за допомогою емуляторів. Оскільки Android-пристрої мають велику різноманітність за характеристиками та версіями операційної системи, тестування лише на фізичних пристроях потребує багато часу та ресурсів. Тому було прийнято рішення використовувати емулятори, які дають змогу перевірити роботу додатку на різних типах пристроїв і з різними розмірами екранів.

Такий підхід дозволяє значно зекономити на реальних пристроях і швидше виявляти помилки, особливо ті, що стосуються інтерфейсу користувача [17]. Для перевірки використовувалися емулятори з різними характеристиками — версії Android, роздільна здатність екрана, розміри екрана тощо.

У результаті було протестовано, як додаток адаптується до різних розмірів дисплеїв і версій Android. Перевірено коректну роботу меню навігації та всіх основних елементів інтерфейсу. Також був складений набір тестових сценаріїв (рис.3.15) з результатами у вигляді матриці для кількох емуляторів.

Назва пристрою та його характеристики	**К1**	**К2**	**К3**	**Р1**	**Н1**
Pixel 3, API 29, 1080x2160, 440x800 dps	+	+	+	+	+
Pixel 4a, API 30, 1080x2340, 440x800 dps	+	+	+	+	+
Galaxy S21, API 31, 1440x3200, 480x800 dps	+	+	+	+	+

Рисунок 3.15 – Таблиця результати тестових сценаріїв

3) Третім методом тестування було обрано юзабіліті-тест. Цей підхід дає змогу оцінити, наскільки зручним і зрозумілим є інтерфейс додатку для користувачів. Під час тестування перевірялась легкість навігації, доступність основних функцій і те, наскільки швидко користувач може знайти потрібні розділи або виконати певну дію.

Такий тип тестування допомагає виявити потенційні незручності ще до того, як додатком почнуть активно користуватися [18]. Завдяки цьому можна завчасно усунути недоліки у дизайні або логіці роботи інтерфейсу, що позитивно впливає на загальний досвід користувача.

Після вдалої установки на головному екрані з’явиться ярлик.

Відкривши додаток, користувач спостерігає головний екран, де він може в меню обрати будь яку сторінку яка йому потрібна.

Перейшовши до прикладу на сторінку «Додати», користувач може внести дохід або витрату, вказавши при цьому місце транзакції та вибравши з випадаючого списку категорію своєї витрати або надходження, також вказується час та дата (рис. 3.16).

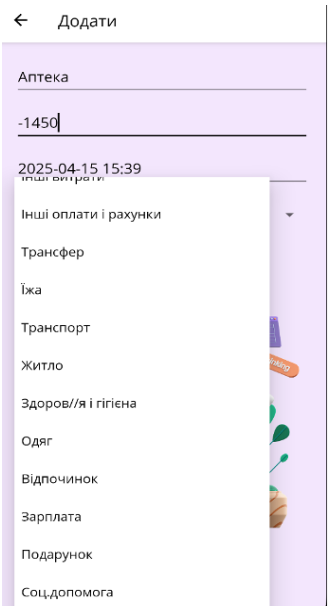

Рисунок 3.16 – Сторінка «Додати»

Після натиснення на кнопку «Додати», ми бачимо як дана витрата з’явилась на сторінці «Витрати».

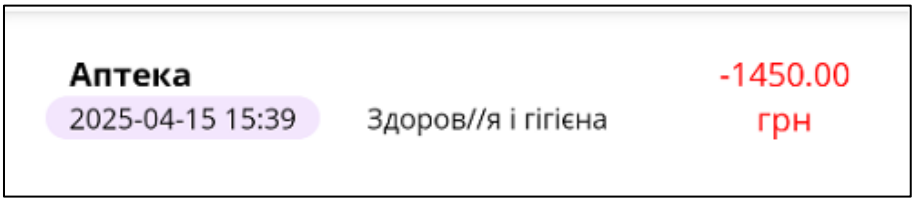

Рисунок 3.17 – Додана витрата

Натиснувши на цю витрату, користувач потрапляє на сторінку де може переглянути її окремо, а також відредагувати натиснувши на іконку редагування (рис.3.18).

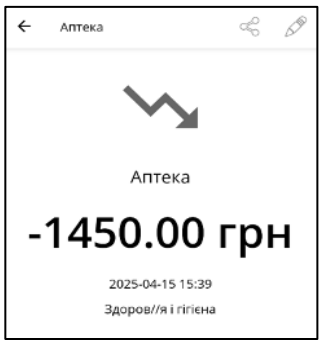

Рисунок 3.18 – Перегляд витрати

Далі повернувшись в меню, користувач може перейти на сторінку, «Щомісячна виписка», дана сторінка дозволяє переглянути графік витрат за конкретний місяць (рис.3.19).

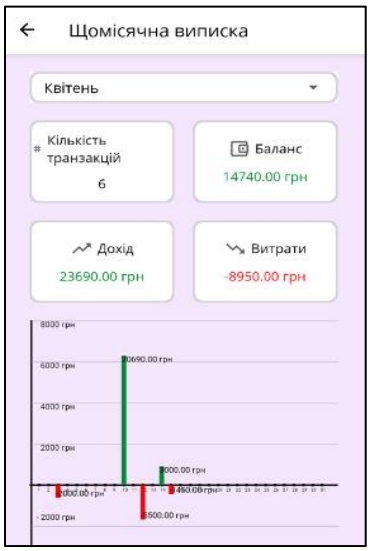

Рисунок 3.19 – Сторінка «Щомісячна виписка»

Далі перейшовши через меню до сторінки «Шукати», користувач має змогу знайти будь яку витрату за назвою, категорією або датою (рис.3.20).

Рисунок 3.20 – Сторінка «Пошуку»

І протестувавши роботу останньої функції “Фінансова ціль”, бачимо що вона успішно працює та виконується поповнення лінійної демонстрації поповнення, при додаванні суми накопичення (рис.3.21).

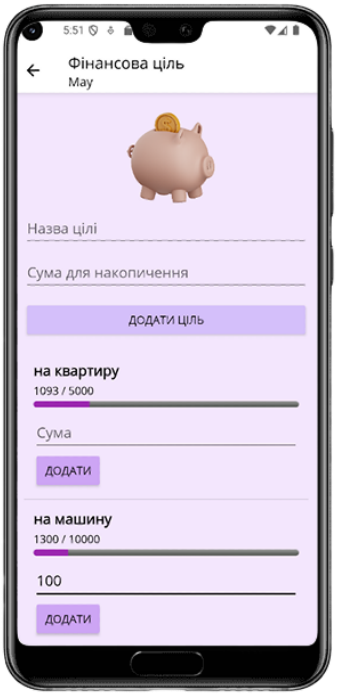

Рисунок 3.21 – Сторінка «Фінансова ціль»

Крім основних методів, для додатку було також проведено інструментальне тестування, зосереджене на перевірці роботи класу DatabaseHelper. Цей клас відповідає за взаємодію з локальною базою даних SQLite і виконує такі ключові операції, як додавання, зчитування та управління фінансовими даними.

Мета тестів - переконатися, що всі функції бази даних працюють стабільно та без помилок. Це особливо важливо, оскільки правильність збереження і обробки даних напряму впливає на коректну роботу всього додатку. Основна задача тестів у DatabaseHelperTest полягає у верифікації ключових методів класу DatabaseHelper. Це включає:

– Перевірку додавання нових записів (наприклад, подій чи фінансових операцій) до бази даних.

– Перевірку коректності отримання даних із бази.

– Забезпечення правильної обробки та збереження інформації відповідно до заданих параметрів.

Тести були запущені на емуляторі Android (модель Pixel_5) з використанням конфігурації "Android Instrumented Tests" у середовищі розробки Android Studio. Під час виконання:

1. Ініціалізовано екземпляр класу DatabaseHelper для доступу до бази даних.

2. Виконано тестові методи для перевірки функціональності, зокрема додавання та отримання даних.

3. Здійснено порівняння отриманих результатів із очікуваними значеннями для підтвердження коректності роботи.

Для цього в Run Configuration я створила нову конфігурацію “DatabaseHelperTest” (рис.3.22).

```
public void setUp() {
    context = InstrumentationRegistry.getInstrumentation().getTargetContext();
    dbHelper = new DatabaseHelper(context);
    // Очищаємо базу перед кожним тестом
    dbHelper.getWritableDatabase().execSQL("DELETE FROM EVENTS_TABLE");
}
@Test
public void testAddEvent() {
    // Створюємо тестову подію
    Event event = new Event();
    event.setPlace_name("Test Transaction");
    event.setPrice("100.50");
    event.setDate("2025-04-24");
    event.setCategory("Food");
    boolean result = dbHelper.addOneEvent(event);
    assertEquals( expected: true, result);
    // Перевіряємо, чи подія додана
    List<Event> events = dbHelper.getAllEvents();
    assertNotNull(events);
    assertEquals( expected: 1, events.size());
    // Перевіряємо значення доданої події
    Event addedEvent = events.get(0);
    assertEquals( expected: "Test Transaction", addedEvent.getPlace_name());
    assertEquals( expected: "100.50", addedEvent.getPrice());
    assertEquals( expected: "Food", addedEvent.getCategory());
    assertEquals( expected: "2025-04-24", addedEvent.getDate());
}
```

Рисунок 3.22 – Фрагмент коду тесту

В результаті тести успішно пройшли, що підтверджується статусом "passed" у звіті про виконання (рис.3.23). Час виконання одного з методів склав 966 мілісекунд. Це свідчить про те, що функціональність управління базою даних працює коректно в поточній реалізації додатку.

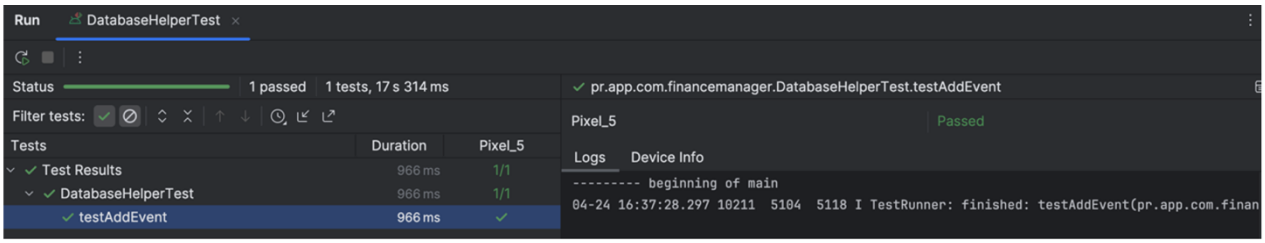

Рисунок 3.23 – Результат тестування

Успішні результати тестів у DatabaseHelperTest підтвердили, що система роботи з базою даних у додатку є надійною. Це означає, що всі дані користувача будуть правильно зберігатися та оброблятися, що є критично важливим для стабільної роботи програми.

Використані методи тестування дали змогу ще на ранніх етапах виявити та виправити помилки, що позитивно вплинуло на загальну якість та стабільність додатку. Проте існують й інші види тестування, які можна застосовувати для ще більш повної перевірки системи у майбутньому.

3.4 Технічні характеристики мобільного застосунку

Додаток розроблявся з підтримкою Android 5.0 Lollipop (minSdkVersion 21), і він оптимізований для Android 11 (targetSdkVersion 30).

Android Studio надає потужні інструменти для аналізу продуктивності мобільних додатків [19]. Одним з таких інструментів є профайлер, який дозволяє оцінити навантаження на операційну систему та ресурси, що споживаються під час роботи додатку.

Для визначення системних вимог скористаємось Профайлером.

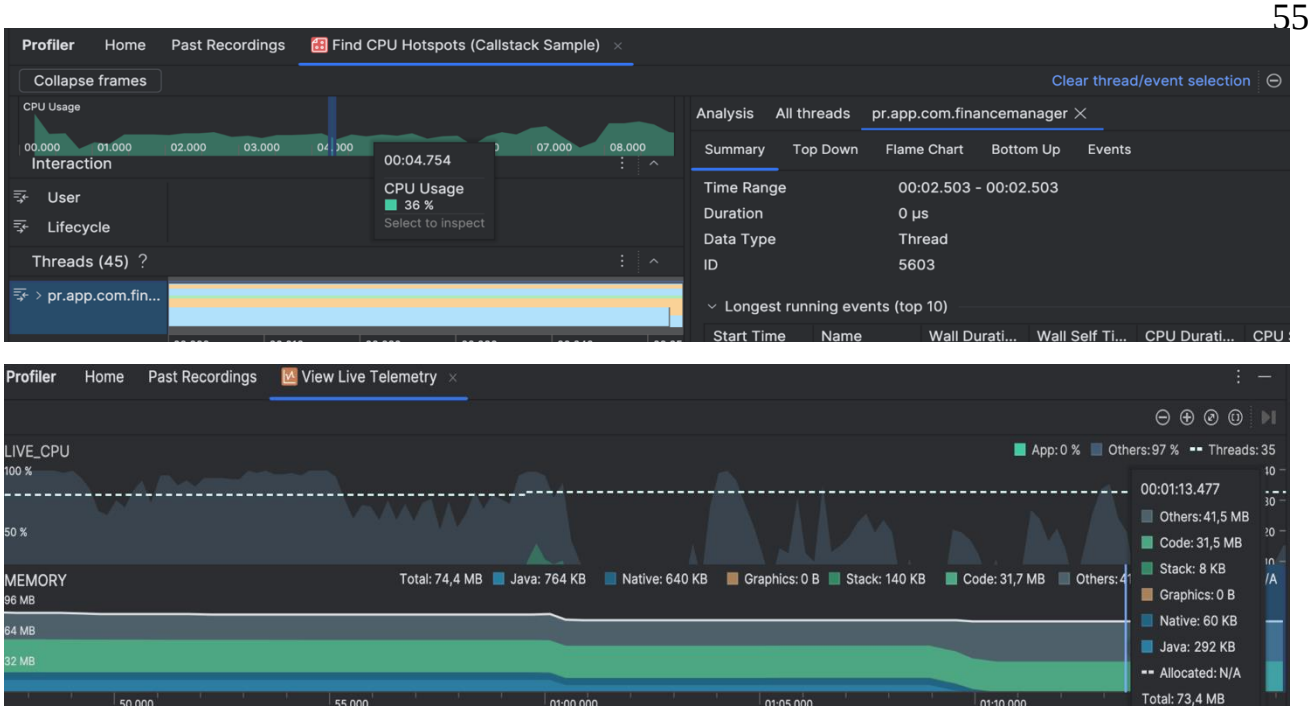

Рисунок 3.25 – Результат тестування

Згідно з отриманими даними з профайлера (рис.3.25) та результатами тестування продуктивності, для мобільного застосунку були визначені наступні характеристики:

Мінімальна конфігурація: Android версії 8.1 або вище, процесор 600 МГц, ОЗП 400 МБ, вільна пам'ять для встановлення додатку 20 МБ, пам'ять для бази даних 1 МБ.

Рекомендована конфігурація: Android версії 8.1 або вище, процесор 600 МГц, ОЗП 400 МБ, вільна пам'ять для встановлення додатку 20 МБ, пам'ять для бази даних 1 МБ.

Ці конфігурації визначають мінімальні та рекомендовані параметри, необхідні для оптимальної роботи застосунку, враховуючи вимоги до використання пам'яті та процесора [20].

У мобільному застосунку були реалізовані всі основні функції, які були визначені ще на етапі постановки задачі в першому розділі. Зокрема, створено зручну систему для додавання, перегляду та розподілу витрат і доходів за категоріями. Також розроблено функціонал для створення фінансових цілей і відстеження прогресу їх досягнення.

Користувач має змогу переглядати детальну фінансову статистику у вигляді графіків і звітів, що дає змогу краще аналізувати свої витрати та доходи.

Крім основного функціоналу, додаток містить низку додаткових можливостей: навігаційне меню, підтримку декількох мов інтерфейсу, а також локальне збереження даних за допомогою бази даних SQLite.

Застосунок має гарний потенціал для розвитку. Серед можливих напрямів покращення.

У цьому розділі детально описано процес розробки та тестування мобільного додатку. Було розглянуто, як створювався інтерфейс, а також показано реалізацію діаграм станів, що описують підготовчі процеси в додатку. Особливу увагу приділено роботі з базою даних і зручності інтерфейсу для користувача, що дозволяє ефективно працювати з різними типами фінансових даних.

Окремо описано архітектуру додатку, структуру його пакетів та проведені етапи тестування. За підсумками тестування встановлено, що мобільний застосунок відповідає поставленим вимогам, працює стабільно та без помилок. Також було оцінено та описано технічні характеристики додатку.

3.5 Висновки

У третьому розділі було реалізовано мобільний застосунок для обліку особистих фінансів на платформі Android з використанням мови Java та бази даних SQLite. Застосунок включає повний функціонал для ведення доходів і витрат: користувач може додавати нові транзакції з вказанням суми, дати, назви, категорії та коментаря, редагувати або видаляти їх при потребі. Уся інформація зберігається локально в базі даних SQLite через реалізований DatabaseHelper. Реалізовано модуль аналітики, який дозволяє переглядати підсумковий баланс, а також зведену інформацію за місяць у вигляді графіків (доходи, витрати, залишок). Окремий фрагмент реалізує пошук транзакцій за трьома критеріями: назвою, датою або категорією. Також реалізовано модуль фінансових цілей, де користувач може створити ціль із заданою сумою. Інтерфейс додатку

побудований у вигляді окремих фрагментів з урахуванням принципів зручності, мінімалізму та інтуїтивності: кожен блок має власне вікно, логічно розділений функціонал і зручну навігацію через нижнє меню. Проведено ручне тестування усіх фрагментів: перевірено коректність збереження та відображення даних, а також адаптивність до різних розмірів екранів. У результаті реалізовано повністю працездатний додаток, готовий до практичного використання в особистому або побутовому фінансовому плануванні.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи — розроблено інформаційну систему для платформи Android, яка дозволяє користувачеві здійснювати облік доходів і витрат, переглядати базовий графік за місяць, та створювати фінансові цілі. Застосунок реалізує інтуїтивно зрозумілий інтерфейс, побудований на архітектурі MVVM, що забезпечує розділення логіки, інтерфейсу та доступу до даних.

Проведено аналіз предметної області, у якому виявлено низький рівень фінансової грамотності серед населення та обґрунтовано необхідність інструментів персонального фінансового контролю. Під час дослідження аналогів (Monefy, YNAB, Spendee, Goodbudget) виявлено типові недоліки: складність інтерфейсу, платна підписка, відсутність підтримки української мови, обмежений функціонал для роботи з фінансовими цілями. Це дозволило чітко визначити вимоги до функціоналу власної інформаційної системи.

В рамках проєктування було розроблено архітектуру застосунку, структуру бази даних на основі SQLite, інтерфейс користувача у вигляді фрагментів та створено набір UML-діаграм, які відображають логіку роботи, взаємодію компонентів та стан системи. Реалізація системи виконана з використанням мови програмування Java, середовища Android Studio та локальної СУБД SQLite, що забезпечує автономну роботу додатку без потреби у зовнішньому сервері.

Функціонал застосунку включає: додавання, редагування та видалення транзакцій, пошук за назвою, датою та категорією, ведення фінансових цілей, перегляд балансу та побудову графіків за місяць. У процесі тестування було перевірено працездатність ІС, коректність записів у базу даних, відображення графіків та стабільність роботи на емуляторі. Оцінено зручність інтерфейсу та підтверджено відповідність реалізованого функціоналу поставленим вимогам.

Розроблений застосунок є завершеним прикладом мобільної інформаційної системи для управління особистими фінансами, яка може бути використана для побутового фінансового контролю окремих осіб, а також адаптована для

впровадження у мікропідприємствах, освітніх проєктах або персональних консультаціях із фінансового планування. фінансів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вернер І. Є. Статистичний щорічник України 2022 / Державна служба статистики. 2023. URL: https://ukrstat.gov.ua/druk/publicat/kat_u/2023/zb/11/year_22_u.pdf (дата звернення: 15.04.2025).

2. Ясинська Н. А. Управління особистими фінансами: навч. посіб. Донецьк: Ноулідж, Донец. від-ня, 2014. 321 с. URL: https://pidru4niki.com/79085/finansi/upravlinnya_osobistimi_finansami (дата звернення: 18.04.2025).

3. П'ять українських додатків для контролю витрат. URL: https://rubryka.com/article/dlya-kontrolyu-vytrat/ (дата звернення: 21.04.2025).

4. Мобільні застосунки для планування бюджету та контролю витрат: які найбільш популярні. Економічна правда. URL: https://www.epravda.com.ua/publications/2023/02/22/697326 (дата звернення: 23.04.2025).

5. Фінансова грамотність. Вікіпедія. URL: https://uk.wikipedia.cc/qysxzw (дата звернення: 26.04.2025).

6. Baumeister R.F., Bushman B.J. Social Psychology and Human Nature. Belmont: Cengage Learning, 2013. 832 p. URL: https://psycnet.apa.org/record/2007-18993-000 (дата звернення: 27.04.2025).

7. What Is YNAB. URL: https://www.ynab.com/features (дата звернення: 29.04.2025).

8. Застосунок. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Застосунок (дата звернення: 30.04.2025).

9. Янчук К. В. Методи і засоби розробки нативних додатків для платформи Android. 2020. URL: https://openarchive.nure.ua/handle/document/23876 (дата звернення: 03.05.2025).

10. Smyth N. Android Studio Electric Eel Essentials – Java Edition. Payload Media, 2023. URL: https://books.google.com (дата звернення: 05.05.2025).

11. Навіщо потрібна SQLite: основні переваги та застосування. URL: https://pravo.home.cx.ua/ukraincyam/navishho-potribnasqlite-osnovni-perevagi-ta-zastosuvannya.html (дата звернення: 07.05.2025).

12. Для чого потрібні UML діаграми? URL: https://foxminded.ua/uml-diagramy/ (дата звернення: 09.05.2025).

13. Model View Model. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Model-View-ViewModel (дата звернення: 11.05.2025).

14. SQLite Home Page. URL: https://sqlite.org (дата звернення: 13.05.2025).

15. Kotenko N., Zhyrova T., Kuleba M. Дослідження особливостей тестування мобільних додатків. Управління розвитком складних систем. 2020. № 41. С. 55–60. DOI: https://doi.org/10.32347/2412-9933.2020.41.55-60 (дата звернення: 15.05.2025).

16. Поворознюк О. А., Іващенко А. В. Автоматизована система функціонального тестування інтерфейсу веб-додатків. 2022. URL: https://repository.kpi.kharkov.ua/handle/KhPI-Press/59454 (дата звернення: 18.05.2025).

17. Satyaputra A., Aritonang E. M., Kom S. Lets Build Your Android Apps with Android Studio. Elex Media Komputindo, 2016. (дата звернення: 20.05.2025).

18. Кравченко С. Методи юзабіліті-тестування для оцінювання мобільного додатку. Technical sciences. 2023. Т. 317. № 1. С. 111–118. DOI: https://doi.org/10.31891/2307-5732-2023-317-1-111-117 (дата звернення: 23.05.2025).

19. Hagos T. Android Studio. Learn Android Studio 3. CA: Apress, 2018. С. 5–17. DOI: https://doi.org/10.1007/978-1-4842-3156-2_2 (дата звернення: 25.05.2025).

20. Бородіна О. О., Філонич М. М. Особливості тестування мобільних додатків та сайтів. 2016. URL: https://reposit.nupp.edu.ua/handle/PoltNTU/3228 (дата звернення: 27.05.2025).

Додаток А

(обов’язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ
Завідувач кафедри САІТ
_____ д.т.н., проф. Віталій МОКІН
“____”_____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ТА АНАЛІЗУ ВИТРАТ

08-34.БКР.020.00.000 ТЗ

Керівник: к.т.н., доц.
_____ Георгій ГОРЯЧЕВ
«__» _____ 2025 р.
Розробила студентка гр. 2ІСТ-21б
_____ Аліна СНІЦАР
«__» _____ 2025 р.

Вінниця 2025

1. Підстава для проведення робіт.

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____2025р., та індивідуальне завдання на БКР, затверджене протоколом №__ засідання кафедри САІТ від «__» _____ 2025р.

2. Джерела розробки:

1) Офіційна документація Android. URL: https://developer.android.com/docs;

2) SQLite Documentation. URL: https://www.sqlite.org/docs.html

3) Ясинська Н. А. Управління особистими фінансами: навч. посіб. Донецьк: Ноулідж, Донец. від-ня, 2014. 321 с.
URL: https://pidru4niki.com/79085/finansi/upravlinnya_osobistimi_finansami

3. Метою дослідження є розроблення інформаційної системи у вигляді мобільного додатку для управління особистими фінансами, що дозволяє здійснювати облік доходів і витрат.

4. Вихідні дані для проведення робіт: Статистика фінансової грамотності населення.

5. Методи дослідження: Аналіз аналогів, розробка UML-діаграм, реалізація архітектури MVVM, моделювання інтерфейсів, створення бази даних SQLite, функціональне тестування, UX-дослідження.

6. Етапи роботи і терміни їх виконання:

a) Аналіз предметної області та огляд існуючих рішень _____ – _____
b) Вибір IT-інфраструктури та проєктування ІС _____ – _____
c) Розроблення інформаційної системи _____ – _____
d) Оформлення матеріалів до захисту БКР _____ – _____

7. Очікувані результати та порядок реалізації

Інформаційна система у вигляді мобільного додатку для Android, що дозволяє керувати особистими фінансами та аналізувати витрати.

8. Вимоги до розробленої документації

Текстова та ілюстративна частини роботи оформлені у відповідності до вимог «Методичних вказівок до виконання бакалаврських кваліфікаційних робіт для студентів спеціальностей: 124 «Системний аналіз», 126 «Інформаційні системи та технології» (освітня програма «Прикладні інформаційні технології»)».

9. Порядок приймання роботи

Публічний захист «___» _____ 2025 р.
Початок розробки «___» _____ 2025 р.
Граничні терміни виконання БКР «___» _____ 2025 р.

Розробила студентка групи 2ІСТ-21б _____ Аліна СНІЦАР

Додаток Б

(обов’язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Інформаційна система управління особистими фінансами та аналізу витрат»

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра САІТ, ФІІТА, гр. 2ІСТ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2.39%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- □ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- □ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Віталій МОКІН, зав. каф. САІТ _____
(підпис)

Євгеній КРИЖАНОВСЬКИЙ, доц. каф. САІТ _____
(підпис)

Особа, відповідальна за перевірку _____ Сергій ЖУКОВ
(підпис)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Георгій ГОРЯЧЕВ, к.т.н., доц. каф. САІТ
(підпис)

Здобувач _____ Аліна СНІЦАР
(підпис)

Додаток В

(довідниковий)

Фрагмент лістингу програми

Код фрагменту додатку AddFragment

```
package pr.app.com.financemanager.fragments;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
import android.content.Context;
import android.os.Bundle;
import android.text.InputType;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.inputmethod.InputMethodManager;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TimePicker;

import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;

import java.text.SimpleDateFormat;
import java.util.Calendar;

import pr.app.com.financemanager.MainActivity;
import pr.app.com.financemanager.R;
import pr.app.com.financemanager.databases.DatabaseHelper;
import pr.app.com.financemanager.models.Event;

public class AddFragment extends Fragment {

    private Context context;
    private EditText add_name, add_price, add_date;
```

```
    private Spinner spinner;
    private Button submit_data;

    public AddFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        context = container.getContext();
        ((MainActivity) getActivity()).getSupportActionBar().setSubtitle(R.string.null_string);
        return inflater.inflate(R.layout.fragment_add, container, false);
    }

    @Override
    public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
        findViews(view);
        setInputTypes();

        add_date.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                showDateDialog(add_date);
            }
        });

        submit_data.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                Event event = new Event();

                try {
                    event = new Event(1, add_price.getText().toString(), add_name.getText().toString(),
add_date.getText().toString(), spinner.getSelectedItem().toString());

                    NavController controller = Navigation.findNavController(view);
                    controller.popBackStack(R.id.events_fragment, false);

                }catch (Exception e){
                    e.printStackTrace();
                }
```

```
                DatabaseHelper databaseHelper = new DatabaseHelper(context);
                databaseHelper.addOneEvent(event);
                dismissKeyboard(getActivity());
            }
        });
    }

    private void findViews(View view){
        add_name = getView().findViewById(R.id.add_name);
        add_price = getView().findViewById(R.id.add_price);
        add_date = getView().findViewById(R.id.add_date);
        submit_data = getView().findViewById(R.id.add_submit_button);

        spinner = getView().findViewById(R.id.add_category_spinner);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(context,
R.array.add_category_spinner, android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spinner.setAdapter(adapter);
    }

    private void setInputTypes(){
        add_date.setInputType(InputType.TYPE_NULL);
        add_price.setInputType(InputType.TYPE_CLASS_NUMBER |
InputType.TYPE_NUMBER_FLAG_DECIMAL | InputType.TYPE_NUMBER_FLAG_SIGNED);
        add_name.setInputType(InputType.TYPE_CLASS_TEXT);
    }

    private void showDateDialog(final EditText add_date) {
        final Calendar calendar = Calendar.getInstance();
        DatePickerDialog.OnDateSetListener dateSetListener = new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker datePicker, int year, int month, int day) {
                calendar.set(Calendar.YEAR, year);
                calendar.set(Calendar.MONTH, month);
                calendar.set(Calendar.DAY_OF_MONTH, day);

                TimePickerDialog.OnTimeSetListener timeSetListener = new TimePickerDialog.OnTimeSetListener()
{
                    @Override
                    public void onTimeSet(TimePicker timePicker, int hour, int minute) {
                        calendar.set(Calendar.HOUR_OF_DAY, hour);
                        calendar.set(Calendar.MINUTE, minute);
```

```
                SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm");
                add_date.setText(simpleDateFormat.format(calendar.getTime()));
            }
        };

        new TimePickerDialog(context, timeSetListener, calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE), true).show();
      }
    };

    new DatePickerDialog(context, dateSetListener, calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH), calendar.get(Calendar.DAY_OF_MONTH)).show();
  }

  private void dismissKeyboard(Activity activity) {
    InputMethodManager imm = (InputMethodManager) activity.getSystemService(Context.INPUT_METHOD_SERVICE);
    if (null != activity.getCurrentFocus())
      imm.hideSoftInputFromWindow(activity.getCurrentFocus()
          .getApplicationWindowToken(), 0);
  }
}
```

Код фрагменту додатку DatabaseHelper

```
import pr.app.com.financemanager.models.Event;

public class DatabaseHelper extends SQLiteOpenHelper {

  private static final String EVENTS_TABLE = "EVENTS_TABLE";
  private static final String COLUMN_EVENT_PRICE = "EVENT_PRICE";
  private static final String COLUMN_EVENT_NAME = "EVENT_NAME";
  private static final String COLUMN_EVENT_DATE = "EVENT_DATE";
  private static final String COLUMN_EVENT_CATEGORY = "EVENT_CATEGORY";
  private static final String COLUMN_ID = "ID";

  public DatabaseHelper(@Nullable Context context) {
    super(context, "events.db", null, 1);
  }

  @Override
  public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String createTableStatement = "CREATE TABLE " + EVENTS_TABLE + " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMN_EVENT_PRICE + " TEXT, " +
```

```
COLUMN_EVENT_NAME + " TEXT, " + COLUMN_EVENT_DATE + " TEXT, " + COLUMN_EVENT_CATEGORY
+ " TEXT)";

        sqLiteDatabase.execSQL(createTableStatement);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {

    }

    public boolean addOneEvent(Event event){
        SQLiteDatabase database = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();

        contentValues.put(COLUMN_EVENT_NAME, event.getPlace_name());
        contentValues.put(COLUMN_EVENT_PRICE, event.getPrice());
        contentValues.put(COLUMN_EVENT_DATE, event.getDate());
        contentValues.put(COLUMN_EVENT_CATEGORY, event.getCategory());

        long insert = database.insert(EVENTS_TABLE, null, contentValues);

        if(insert == -1){
            return false;
        }else{
            return true;
        }
    }

    public boolean updateOneEvent(Event event, String id){
        SQLiteDatabase database = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();

        contentValues.put(COLUMN_ID, event.getId());
        contentValues.put(COLUMN_EVENT_NAME, event.getPlace_name());
        contentValues.put(COLUMN_EVENT_PRICE, event.getPrice());
        contentValues.put(COLUMN_EVENT_DATE, event.getDate());
        contentValues.put(COLUMN_EVENT_CATEGORY, event.getCategory());

        long update = database.update(EVENTS_TABLE, contentValues, "ID = ?", new String[] {id});
        if(update == -1){
            return false;
        }else{
```

```
            return true;
        }
    }

    public boolean deleteOneEvent(Event event){
        SQLiteDatabase database = this.getWritableDatabase();
        String query = "DELETE FROM " + EVENTS_TABLE + " WHERE " + COLUMN_ID + " = " +
event.getId();

        Cursor cursor = database.rawQuery(query, null);

        if(cursor.moveToFirst()){
            return true;
        }else {
            return false;
        }

    }

    public double getAllMoney(){
        double returnMoney = 0;

        Calendar cal = Calendar.getInstance();
        SimpleDateFormat month_date = new SimpleDateFormat("MM");
        SimpleDateFormat year_date = new SimpleDateFormat("yyyy");
        String month = month_date.format(cal.getTime());
        String year = year_date.format(cal.getTime());

        String query = "SELECT * FROM " + EVENTS_TABLE + " WHERE " + COLUMN_EVENT_DATE + "
LIKE '" + year + "-" + month + "-%' ORDER BY " + COLUMN_EVENT_DATE +" DESC";
        SQLiteDatabase database = this.getReadableDatabase();
        Cursor cursor = database.rawQuery(query, null);

        if(cursor.moveToFirst()){
            do{
                int id = cursor.getInt(0);
                String price = cursor.getString(1);
                String name = cursor.getString(2);
                String date =cursor.getString(3);
                String category = cursor.getString(4);

                Event event = new Event(id, price, name, date, category);
                returnMoney += Double.parseDouble(event.getPrice());
```

```
            }while (cursor.moveToNext());
        }else{
            //empty list
        }

        return returnMoney;
    }

    public double getAllMoneyByName(String input){
        double returnMoney = 0;

        String query = "SELECT * FROM " + EVENTS_TABLE + " WHERE " + COLUMN_EVENT_NAME + "
LIKE '%" + input + "%' ORDER BY " + COLUMN_EVENT_DATE +" DESC";
        SQLiteDatabase database = this.getReadableDatabase();
        Cursor cursor = database.rawQuery(query, null);

        if(cursor.moveToFirst()){
            do{
                int id = cursor.getInt(0);
                String price = cursor.getString(1);
                String name = cursor.getString(2);
                String date =cursor.getString(3);
                String category = cursor.getString(4);

                Event event = new Event(id, price, name, date, category);
                returnMoney += Double.parseDouble(event.getPrice());

            }while (cursor.moveToNext());
        }else{
            //empty list
        }

        return returnMoney;
    }

    public double getAllMoneyByCategory(String input){
        double returnMoney = 0;

        String query = "SELECT * FROM " + EVENTS_TABLE + " WHERE " +
COLUMN_EVENT_CATEGORY + " LIKE '%" + input + "%' ORDER BY " + COLUMN_EVENT_DATE +" DESC";
        SQLiteDatabase database = this.getReadableDatabase();
        Cursor cursor = database.rawQuery(query, null);
```

```
        if(cursor.moveToFirst()){
            do{
                int id = cursor.getInt(0);
                String price = cursor.getString(1);
                String name = cursor.getString(2);
                String date =cursor.getString(3);
                String category = cursor.getString(4);

                Event event = new Event(id, price, name, date, category);
                returnMoney += Double.parseDouble(event.getPrice());

            }while (cursor.moveToNext());
        }else{
            //empty list
        }

        return returnMoney;
    }

    public double getAllMoneyByDate(String input){
        double returnMoney = 0;

        String query = "SELECT * FROM " + EVENTS_TABLE + " WHERE " + COLUMN_EVENT_DATE + "
LIKE '%" + input + "%' ORDER BY " + COLUMN_EVENT_DATE +" DESC";
        SQLiteDatabase database = this.getReadableDatabase();
        Cursor cursor = database.rawQuery(query, null);

        if(cursor.moveToFirst()){
            do{
                int id = cursor.getInt(0);
                String price = cursor.getString(1);
                String name = cursor.getString(2);
                String date =cursor.getString(3);
                String category = cursor.getString(4);

                Event event = new Event(id, price, name, date, category);
                returnMoney += Double.parseDouble(event.getPrice());

            }while (cursor.moveToNext());
        }else{
            //empty list
        }
```

```
        return returnMoney;
```

Код фрагменту додатку MainActivity

```
import pr.app.com.financemanager.transporters.MonthTransporter;

public class MainActivity extends AppCompatActivity implements NavigationView.OnNavigationItemSelectedListener {

    private DrawerLayout drawerLayout;
    private NavigationView navigationView;

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

        Calendar cal=Calendar.getInstance();
        SimpleDateFormat month_date = new SimpleDateFormat("LLLL", Locale.forLanguageTag("UA"));
        String month = month_date.format(cal.getTime());
        String month_name = month.substring(0, 1).toUpperCase() + month.substring(1);
        MonthTransporter.setMonth(month_name);

        drawerLayout = findViewById(R.id.drawer_layout);
        navigationView = findViewById(R.id.nav_view);
        ActionBarDrawerToggle drawerToggle = new ActionBarDrawerToggle(this, drawerLayout, R.string.open, R.string.close);
        drawerToggle.syncState();
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        init();
    }

    @Override
    public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
        switch (menuItem.getItemId()) {
            case R.id.main_page: {
                NavOptions navOptions = new NavOptions.Builder()
                        .setPopUpTo(R.id.main, true)
                        .build();
                Navigation.findNavController(this, R.id.nav_host_fragment).navigate(R.id.events_fragment, null, navOptions);
                break;
            }
```

```
        case R.id.add_here:{
            if(isValidDestination(R.id.add_here)){
                Navigation.findNavController(this, R.id.nav_host_fragment).navigate(R.id.add_fragment);
            }
            break;
        }

        case R.id.search:{
            if(isValidDestination(R.id.search)){
                Navigation.findNavController(this, R.id.nav_host_fragment).navigate(R.id.search_fragment);
            }
            break;
        }

        case R.id.expenses:{
            if(isValidDestination(R.id.expenses)){
                Navigation.findNavController(this, R.id.nav_host_fragment).navigate(R.id.expenses_fragment);
            }
            break;
        }

        case R.id.incomes:{
            if(isValidDestination(R.id.incomes)){
                Navigation.findNavController(this, R.id.nav_host_fragment).navigate(R.id.income_fragment);
            }
            break;
        }

        case R.id.monthStats:{
            if(isValidDestination(R.id.monthStats)){
                Navigation.findNavController(this, R.id.nav_host_fragment).navigate(R.id.month_stats);
            }
            break;
        }
    }
    menuItem.setChecked(true);
    drawerLayout.closeDrawer(GravityCompat.START);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem menuItem) {
```

```
        switch (menuItem.getItemId()){
            case R.id.home:{
                if(drawerLayout.isDrawerOpen(GravityCompat.START)){
                    drawerLayout.closeDrawer(GravityCompat.START);
                    return true;
                }else{
                    return false;
                }
            }
            default:
                return super.onOptionsItemSelected(menuItem);
        }
    }

    private void init(){
        NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment);
        NavigationUI.setupActionBarWithNavController(this, navController, drawerLayout);
        NavigationUI.setupWithNavController(navigationView, navController);
        navigationView.setNavigationItemSelectedListener(this);
    }

    private boolean isValidDestination(int dest){
        return dest != Objects.requireNonNull(Navigation.findNavController(this,
R.id.nav_host_fragment).getCurrentDestination()).getId();
    }

    @Override
    public boolean onSupportNavigateUp() {
        return NavigationUI.navigateUp(Navigation.findNavController(this, R.id.nav_host_fragment),
drawerLayout);
    }
}
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ТА АНАЛІЗУ ВИТРАТ

Нормоконтроль: к.т.н., доцент

_____ Сергій ЖУКОВ

«___» _____ 2025 р.

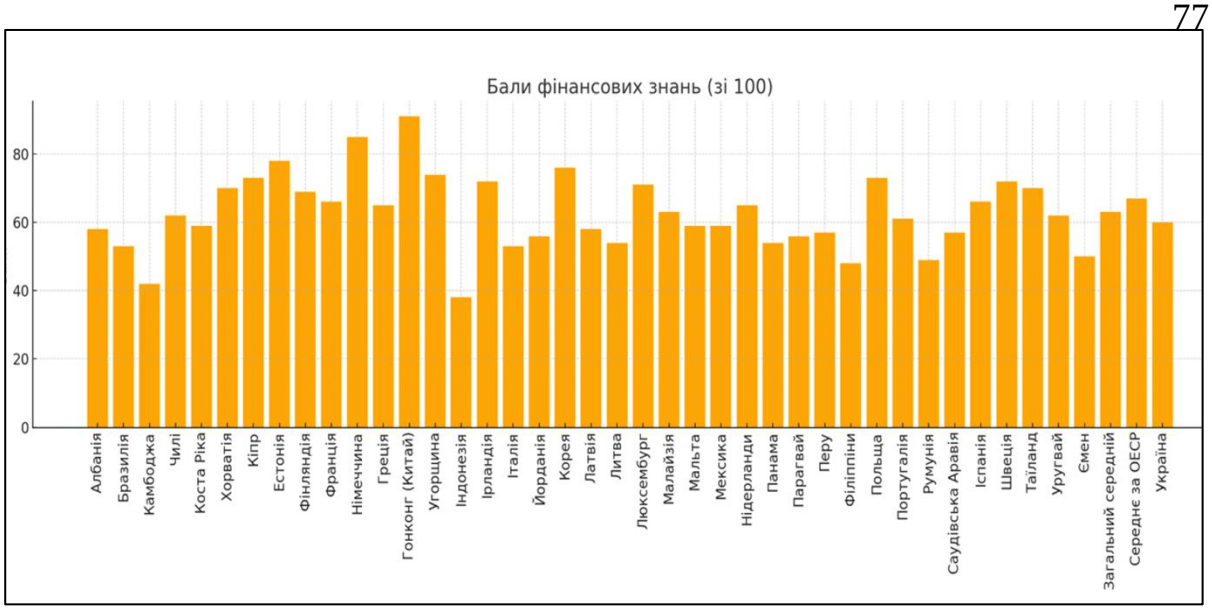

Рисунок Г.1 - Графік балів фінансових знань

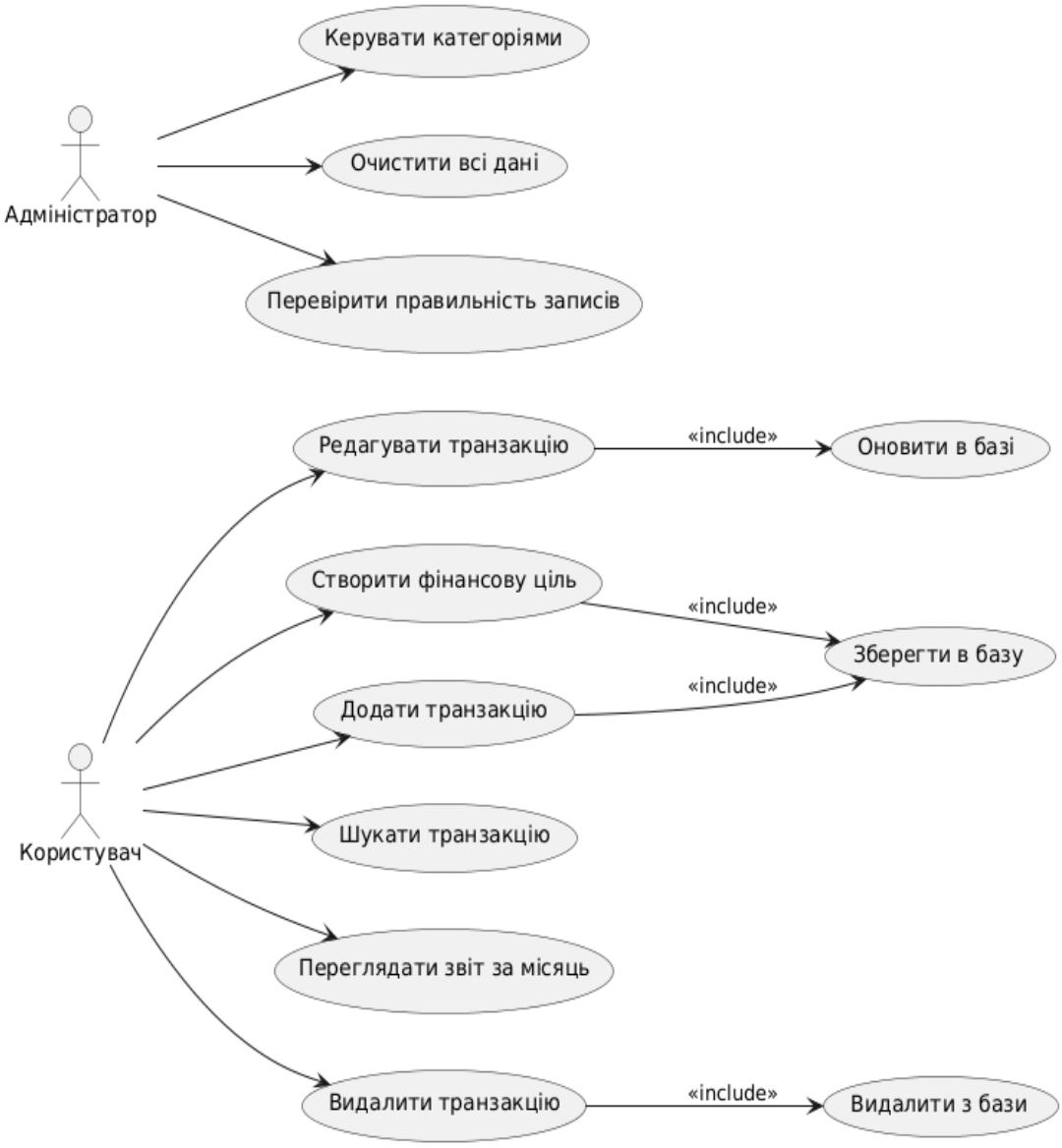

Рисунок Г.2 - Діаграма Use Case

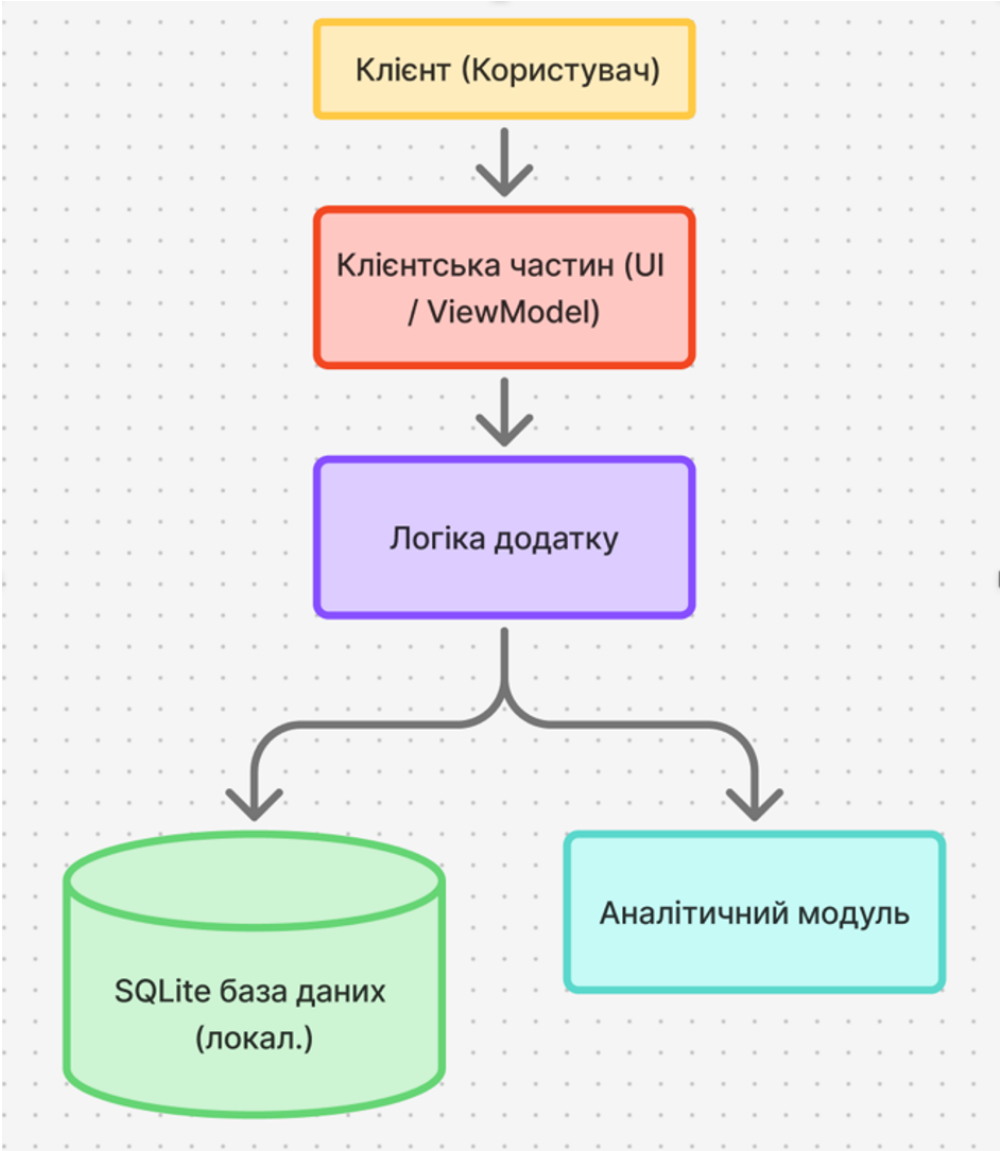

Рисунок Г.3 - Структура системи

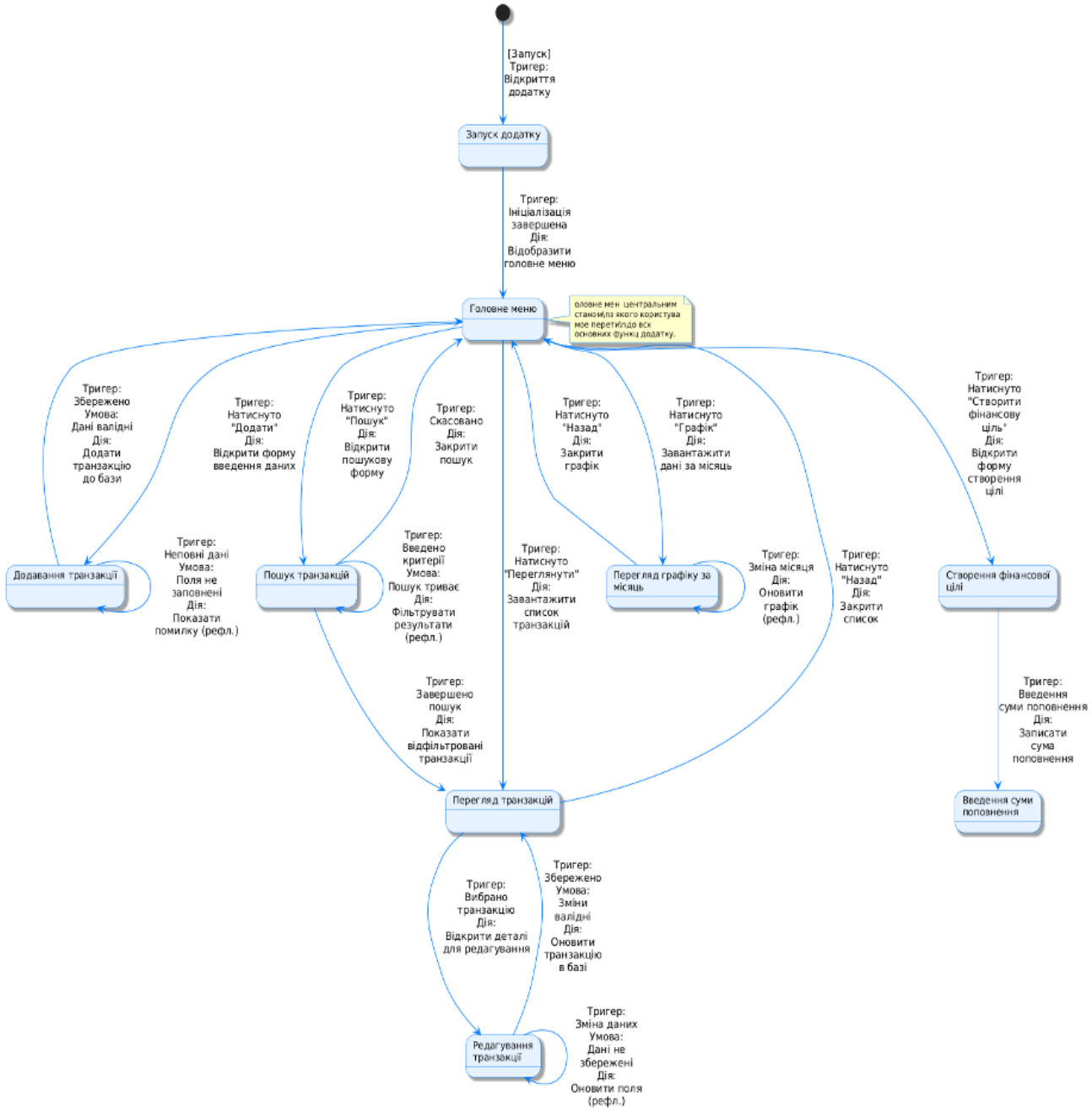

Рисунок Г.4 - Діаграма станів

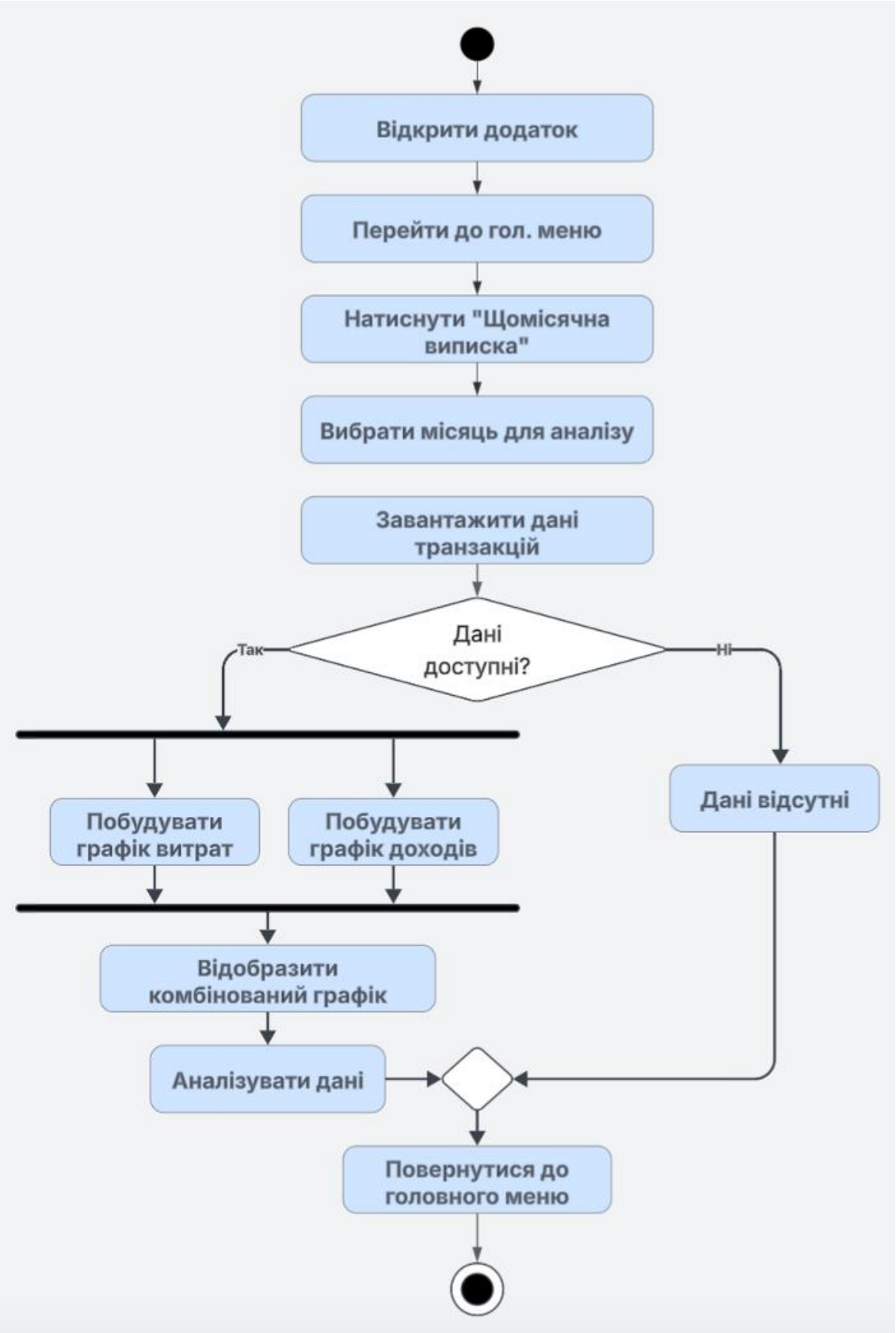

Рисунок Г.5 - Activity Diagram - перегляд графіку за місяць

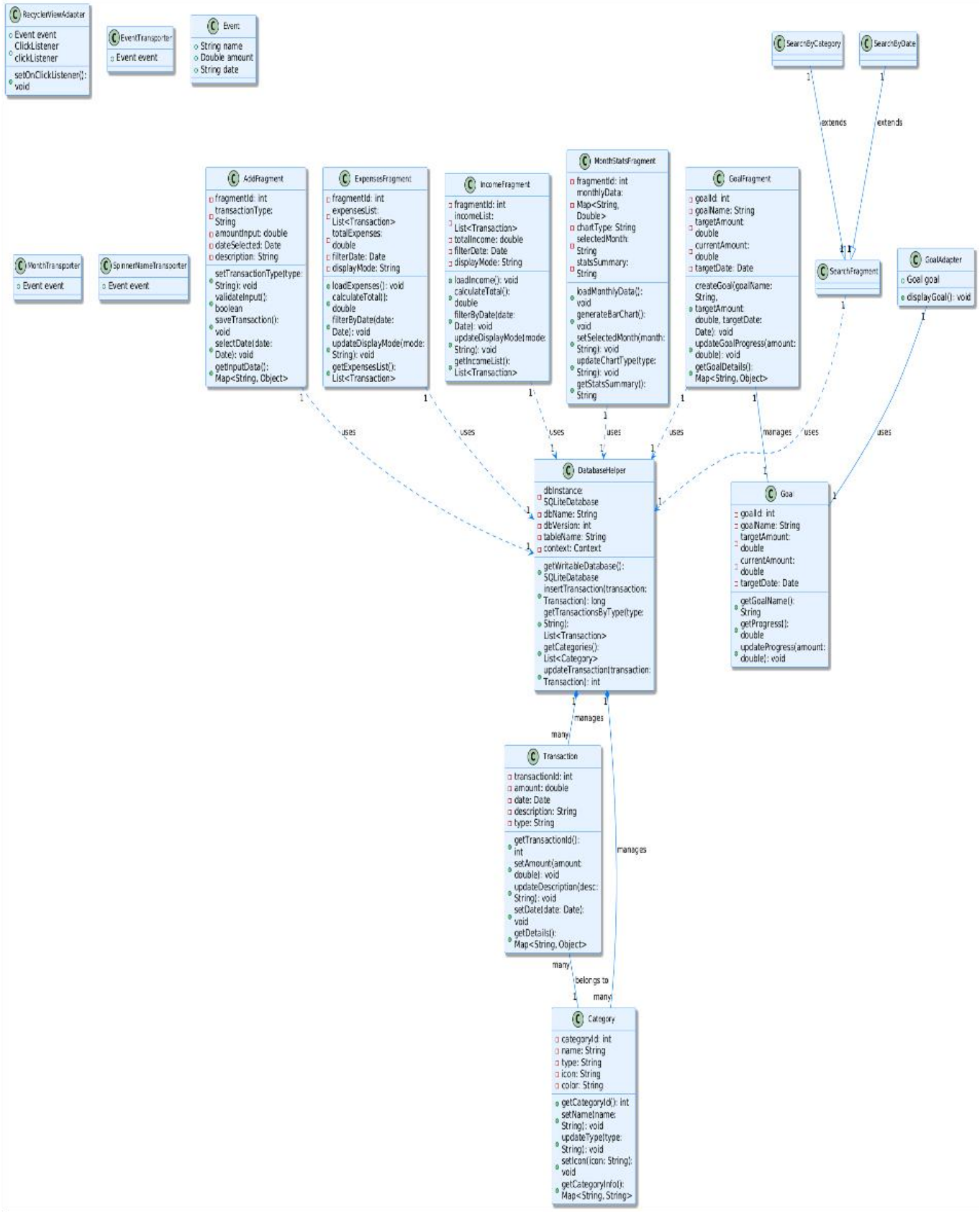

Рисунок Г.5 – Діаграма класів

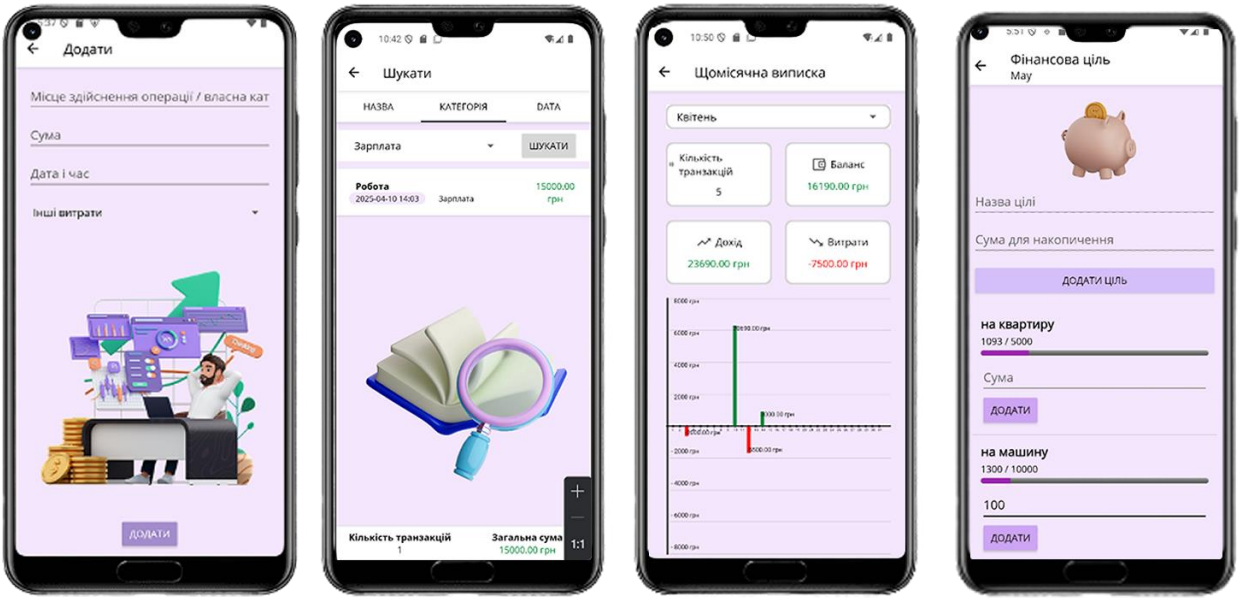

Рисунок Г.6 – Інтерфейс додатку