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

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ANDROID-ДОДАТКУ ДЛЯ МОНІТОРИНГУ ВИТРАТ ТА ДОХОДІВ З ВИКОРИСТАННЯМ DART

DEVELOPMENT OF AN ANDROID APPLICATION FOR MONITORING EXPENSES AND INCOME USING DART

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

Виконав: здобувач вищої освіти
групи ІПЗ-41
Пилипчук Ілля Сергійович

Керівник:
Потейчук Михайло Іванович

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

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Пилипчуку Іллі Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка Android-додатку для моніторингу витрат та доходів з використанням Dart»*
Керівник роботи: *асистент Потейчук М. І.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Flutter, мова програмування Dart, Hive, BLoC, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз аналогів, постановка завдання, визначення вимог, проєктування архітектури, вибір технологій, реалізація інтерфейсу, розробка логіки, тестування, створення бази даних, захист даних, розробка документації, формулювання висновків.*

5. Перелік графічного матеріалу: *8 рисунків.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Потейчук М. І.*		
Специфікація вимог до розробленої системи	*Потейчук М. І.*		
Розробка об'єкта проектування	*Потейчук М. І.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,23 %		
Академічна доброчесність	*Потейчук М. І.*		

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

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

Здобувач вищої освіти Ілля ПИЛИПЧУК

Керівник кваліфікаційної роботи Михайло ПОТЕЙЧУК

АНОТАЦІЯ

Пилипчук І. С. Розробка Android-додатку для моніторингу витрат та доходів з використанням Dart. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

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

У першому розділі здійснено аналіз ринку мобільних застосунків для обліку особистих фінансів і сформульовано завдання. У другому розділі визначено вимоги до програмного забезпечення, обґрунтовано вибір архітектури, мови Dart і засобів реалізації. У третьому розділі реалізовано додаток, проведено тестування, розроблено базу даних і супровідну документацію. У висновках підсумовано результати роботи та вказано перспективи подальшого розвитку.

Ключові слова: фінанси, витрати, доходи, мобільний додаток, Dart, Flutter.

ABSTRACT

Pylypchuk I. Development of an Android application for monitoring expenses and income using Dart. Manuscript. Qualification work of the bachelor's program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

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

The first chapter presents an analysis of the mobile application market for personal finance and formulates the objectives. The second chapter defines the software requirements, justifies the choice of architecture, Dart language, and implementation tools. The third chapter implements the application, conducts testing, develops the database, and provides supporting documentation. The conclusions summarize the results and highlight prospects for further development.

Keywords: finance, expenses, income, mobile application, Dart, Flutter.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах розвитку інформаційних технологій та цифровізації щоденних процесів усе більше зростає потреба в автоматизації особистого фінансового обліку. Управління доходами та витратами є невід'ємною частиною фінансової грамотності, що, у свою чергу, впливає на рівень добробуту кожної людини. Особливо актуальним це питання стає в умовах економічної нестабільності, коли зростає значення ефективного контролю за фінансами. У зв'язку з цим розробка мобільних застосунків для моніторингу витрат та доходів набуває практичного значення не лише для окремих користувачів, а й для формування загальної фінансової культури в суспільстві.

Аналіз сучасного стану проблеми свідчить, що на ринку вже існують певні рішення (наприклад, Wallet, Money Manager, Monefy), проте більшість із них або мають надлишковий функціонал, який ускладнює використання для звичайного користувача, або не дозволяють гнучко налаштовувати категорії витрат/доходів, не підтримують локалізацію українською мовою, або передбачають обов'язкову реєстрацію та передавання даних на сторонні сервери, що знижує рівень довіри до таких застосунків. Відтак залишається незаповнена ніша зручного, інтуїтивно зрозумілого й конфіденційного мобільного додатку для українських користувачів.

Світові тенденції демонструють зростання попиту на персоналізовані мобільні рішення для управління фінансами. Все більше користувачів надають перевагу саме мобільним застосункам, які дозволяють здійснювати облік «на ходу», мають дружній інтерфейс та інтеграцію з іншими сервісами. Загалом, розробка сучасного мобільного додатку для моніторингу особистих фінансів є актуальною як у світовому, так і у національному контексті.

Метою кваліфікаційної роботи бакалавра є розробка Android-додатку для обліку витрат і доходів з використанням мови Dart і фреймворку Flutter, який би задовольняв потреби широкої аудиторії користувачів, був зручним у

користуванні, підтримував локалізацію, мав просту структуру й забезпечував наочну візуалізацію фінансових даних.

Об’єктом кваліфікаційної роботи бакалавра є процес автоматизованого ведення особистого бюджету, а предметом – програмний засіб, що реалізує цей процес. Програмний продукт, який розробляється у межах цієї роботи, – Android-додаток для моніторингу витрат і доходів, розроблений мовою програмування Dart із використанням фреймворку Flutter.

Виходячи з мети та об’єкту кваліфікаційної роботи бакалавра були поставлені наступні завдання:

1) здійснити аналіз сучасного стану проблеми обліку особистих фінансів, дослідити наукові джерела, а також проаналізувати наявні мобільні застосунки-аналоги для виявлення їх переваг, недоліків та потенційних можливостей удосконалення;

2) визначити функціональні та нефункціональні вимоги до проєктованого програмного забезпечення, сформулювати основні сценарії використання, враховуючи очікування користувачів, специфіку платформи Android та загальні принципи побудови інформаційних систем;

3) обґрунтувати вибір архітектурного підходу, мови програмування Dart, фреймворку Flutter та допоміжних бібліотек, пояснити переваги такого вибору в контексті мобільної розробки, адаптивності та підтримки автономної роботи;

4) реалізувати Android-додаток, що забезпечує введення, редагування, перегляд та видалення фінансових операцій, категоризацію транзакцій, візуалізацію даних та базові функції бюджетування;

5) провести тестування функціональних модулів та інтерфейсу користувача, здійснити налагодження логіки додатку, виявити та усунути помилки, забезпечивши стабільну роботу програмного продукту на різних пристроях;

6) розробити локальну базу даних для зберігання інформації про транзакції, категорії та бюджети користувача, реалізувати серіалізацію даних через відповідні моделі та адаптери, використовуючи Hive як сховище;

7) створити супровідну документацію, яка містить системні вимоги, інструкцію користувача, довідкові матеріали, а також описує процедуру встановлення додатку та особливості його використання.

РОЗДІЛ 1
АНАЛІЗ МОБІЛЬНИХ ЗАСОБІВ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ ТА ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасний ринок програмного забезпечення пропонує широкий спектр мобільних застосунків для обліку витрат і доходів, що свідчить про високу затребуваність таких рішень серед користувачів. Водночас із розвитком фінансової грамотності та збільшенням частки безготівкових розрахунків виникає потреба в персоналізованих інструментах для ведення бюджету, які мають бути зручними, адаптивними та безпечними.

Загалом, сучасні джерела засвідчують зростаючу роль цифрових інструментів у сфері управління особистими фінансами. У публікації НУ «Львівська політехніка» [1] підкреслено важливість фінансової грамотності та потребу в зручних засобах обліку доходів і витрат. Башовий В. та співавтори [2] розглядають адаптивний вебінтерфейс для фінансового додатку, акцентуючи на гнучкості дизайну для різних пристроїв. Черненко О. та співавтори [3] розробили мобільний додаток з персоналізованим функціоналом для відстеження витрат, що включає категоризацію, авторизацію та аналіз транзакцій. У роботі AstroWealth [4] описано платформу для студентів, орієнтовану на модульну побудову додатку. Stefanov T. та ін. [5] представили Android-додаток з функціями звітності, візуалізації та сканування, що підкреслює важливість інтеграції та зручності.

Також згідно з оглядом літературних джерел та аналітичних звітів, мобільні застосунки для обліку особистих фінансів поділяються на локальні (дані зберігаються лише на пристрої) та хмарні (зберігання через онлайн-сервіси). Найпоширенішими серед користувачів Android є такі програми, як Monefy, Money Manager, Wallet та Spendee.

Monefy (рис. 1.1) – це популярний додаток для Android і iOS, який вирізняється максимально спрощеним інтерфейсом, що дозволяє швидко

вносити витрати вручну без необхідності реєстрації чи підключення до Інтернету. Усі категорії витрат відображаються у вигляді великих іконок, що робить процес фіксації інтуїтивно зрозумілим навіть для новачків. Основний функціонал зосереджений на додаванні, редагуванні та перегляді щоденних витрат. Окремою перевагою є можливість ведення декількох гаманців (рахунків), підтримка кількох валют, а також PIN-захист. Проте функціонал програми обмежений у безкоштовній версії, а можливості для побудови звітності або кастомізації категорій є досить вузькими.

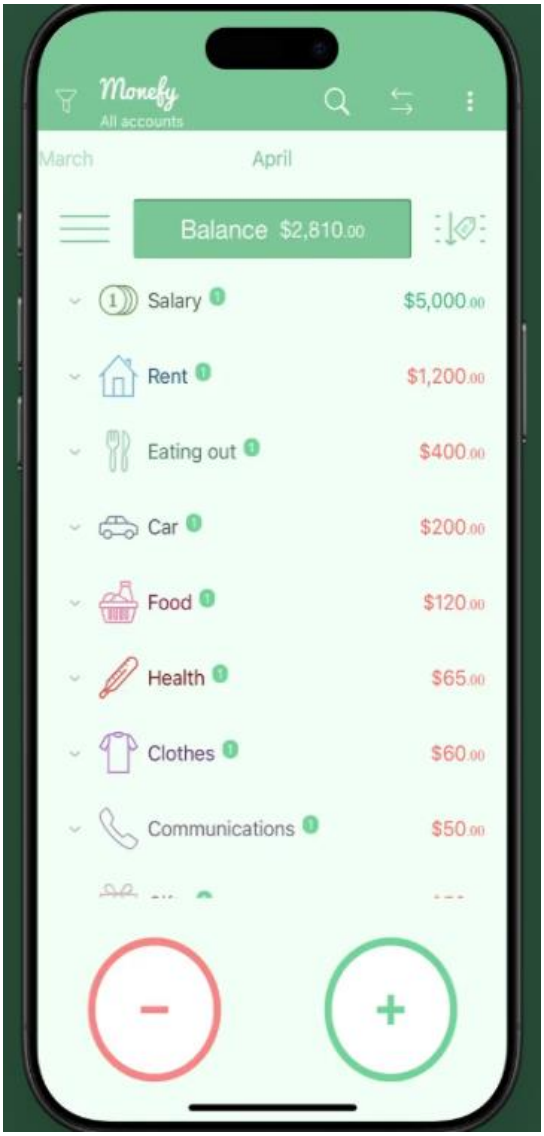

Рисунок 1.1 – Інтерфейс Monefy [6]

Money Manager (рис. 1.2) – більш потужний фінансовий застосунок, який поєднує класичний облік доходів і витрат із гнучкими інструментами бюджетування та графічного аналізу. Програма дозволяє створювати складну структуру категорій і підкатегорій, вести облік за декількома рахунками (у тому числі банківськими), переглядати рух коштів у вигляді графіків та таблиць, експортувати дані у форматах Excel і PDF. Money Manager підтримує функцію планування регулярних платежів, а також має календар для перегляду транзакцій за днями. Водночас деякі основні функції, як-от синхронізація з хмарою або автоматичне резервне копіювання, доступні лише у платній версії.

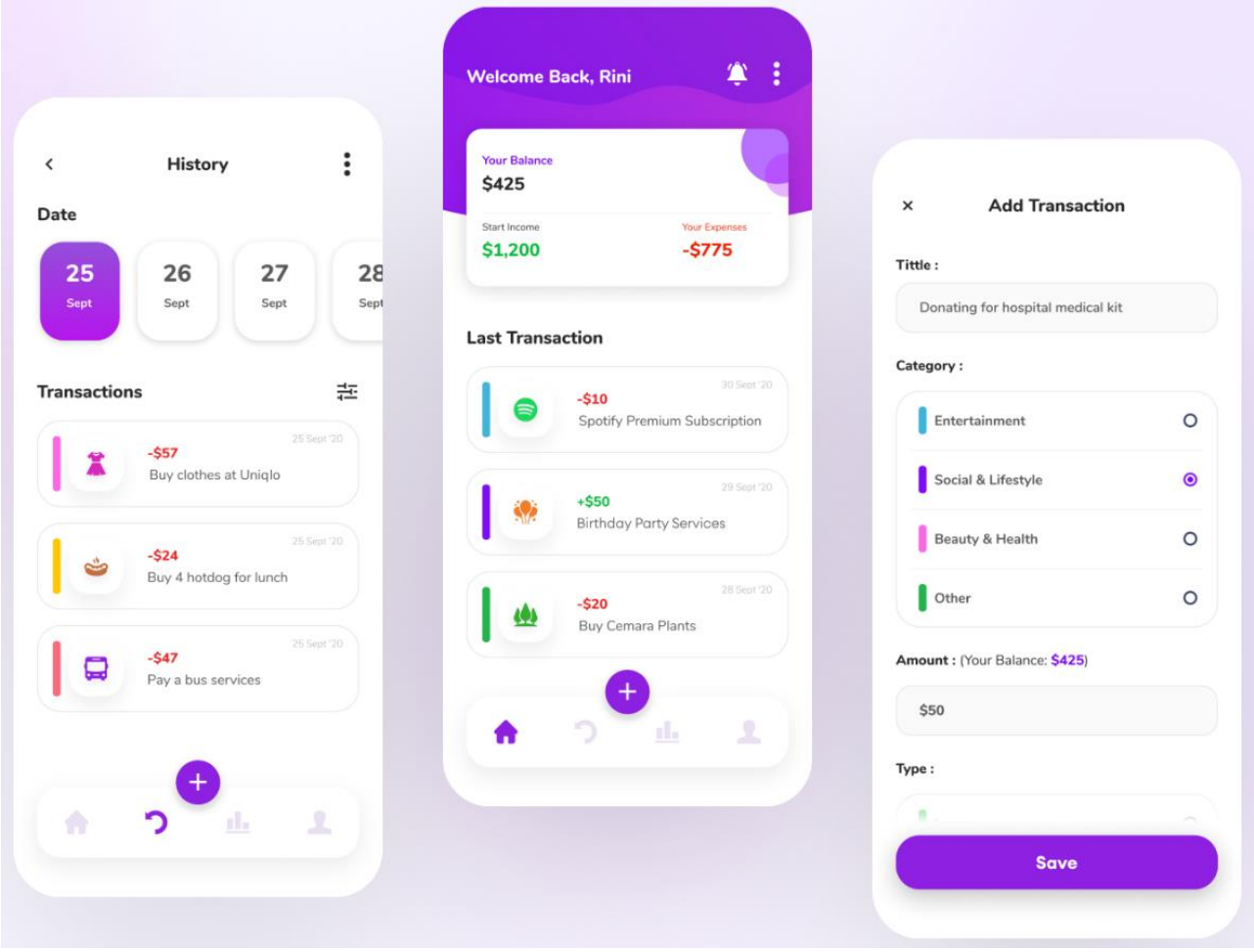

Рисунок 1.2 – Інтерфейс Money Manager [7]

Wallet (рис. 1.3) – це сучасний мобільний фінансовий менеджер, орієнтований на користувачів, які активно користуються банківськими сервісами. Однією з головних переваг Wallet є підтримка автоматичної

синхронізації з банківськими рахунками через API, що дозволяє автоматично імпортувати транзакції та вести статистику без ручного введення. Додаток також дозволяє створювати бюджетні цілі, розраховувати заощадження, прогнозувати витрати та ділитися доступом до гаманця з іншими членами сім'ї. Існує вебверсія, яка синхронізується з мобільним додатком. Проте більшість розширених функцій доступна лише в рамках преміум-передплати, а інтерфейс не має повноцінної української локалізації.

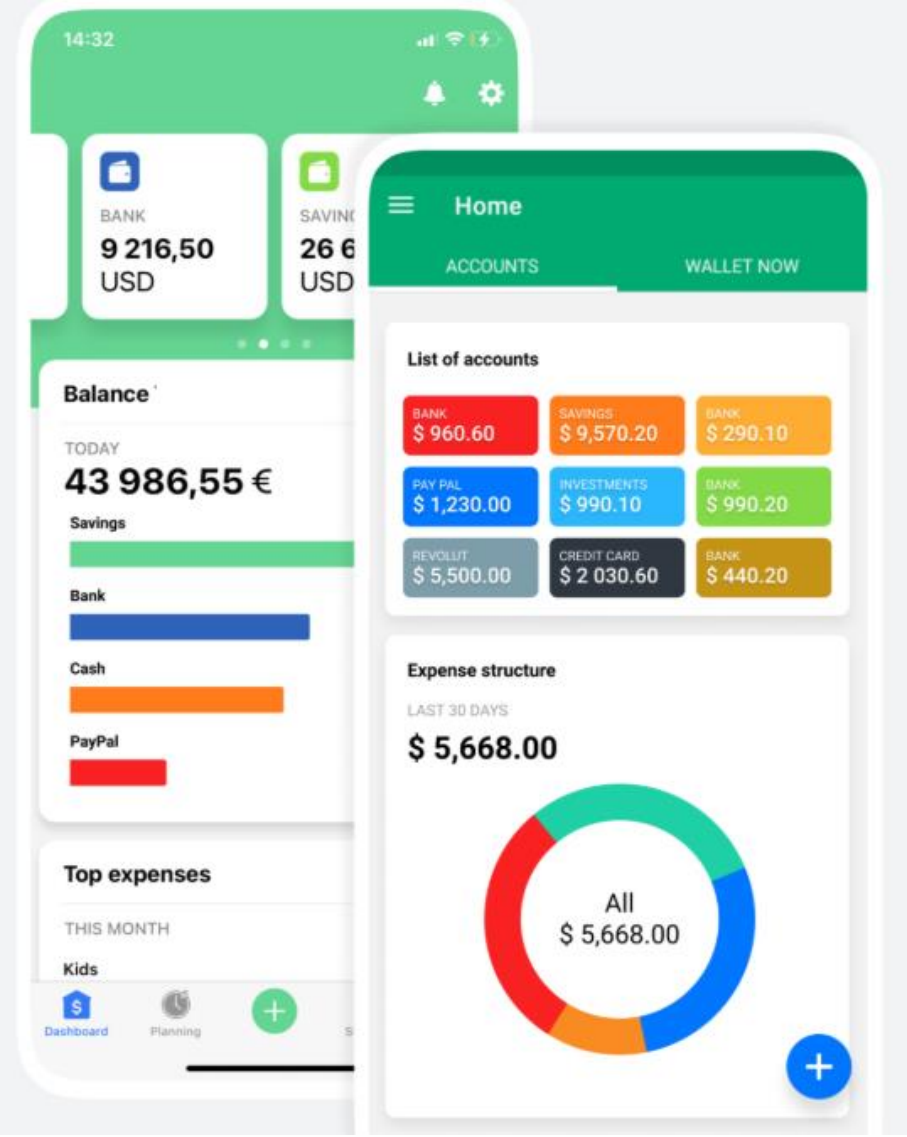

Рисунок 1.3 – Інтерфейс Wallet [8]

Spendee (рис. 1.4) – стильний мобільний додаток з акцентом на естетику, простоту використання та візуалізацію даних. Він підтримує створення

спільних гаманців, що дозволяє кільком користувачам контролювати загальний бюджет – наприклад, у межах родини. Додаток автоматично категоризує транзакції, імпортовані з банківських рахунків або введені вручну. Особливістю Spendee є зручна побудова діаграм витрат, підсвічування перевитрат, а також можливість додавання фото до кожної витрати (наприклад, чеку). Однак застосунок не має широких можливостей локального зберігання: основна частина даних синхронізується через хмару, що може викликати занепокоєння щодо конфіденційності.

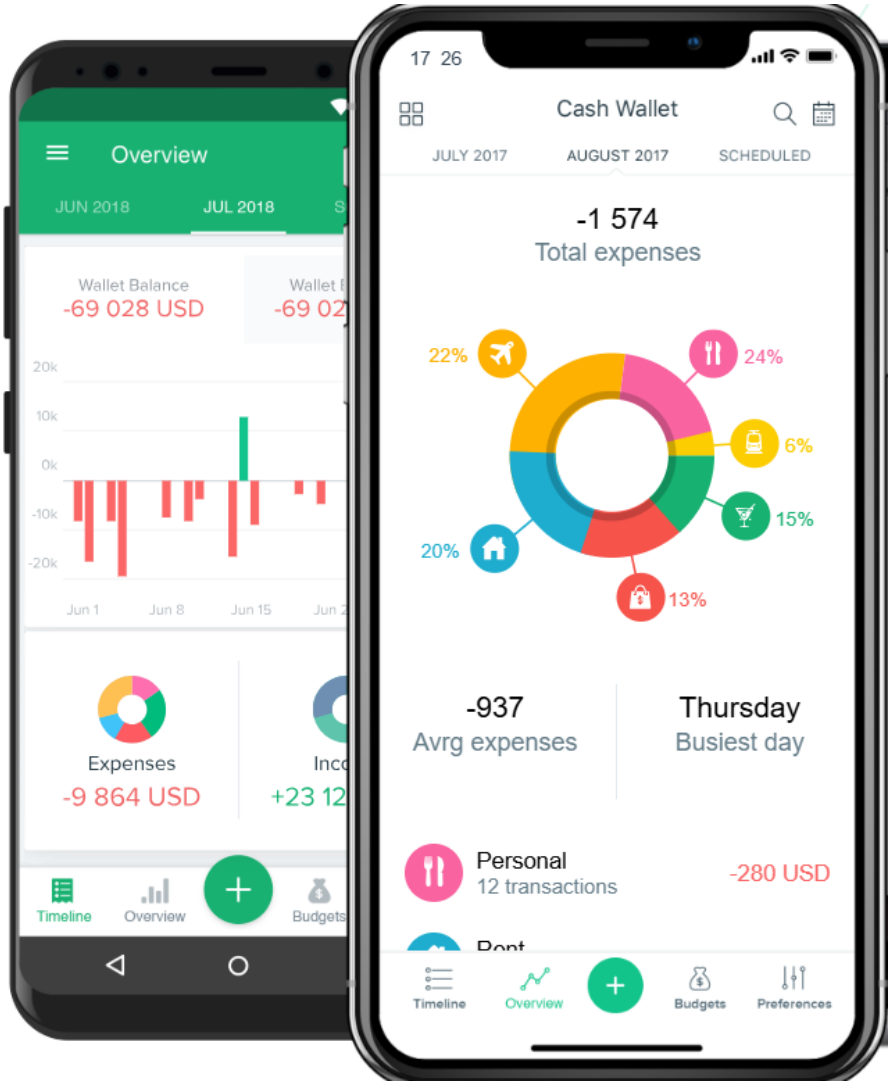

Рисунок 1.4 – Інтерфейс Spendee [9]

Аналіз наявних мобільних застосунків для обліку особистих фінансів дозволив виявити як сильні сторони, так і суттєві обмеження, що мають

значення при формуванні вимог до нового програмного продукту. Основними перевагами більшості популярних рішень є зручність користувацького інтерфейсу, що дозволяє швидко орієнтуватися у функціоналі навіть без попередньої підготовки, а також наявність автоматичної категоризації витрат, яка спрощує аналіз транзакцій. Такі додатки, як правило, підтримують побудову звітів і візуалізацію даних у вигляді діаграм, що дає змогу користувачам відслідковувати фінансову динаміку у зручній графічній формі. Важливою перевагою також є можливість обліку в різних валютах, що є актуальним у випадку користувачів, які мають міжнародні джерела доходів чи витрат. Крім того, частина застосунків пропонує інтеграцію з банківськими сервісами або хмарними сховищами, такими як Google Drive, що дозволяє автоматизувати імпорт даних і зберігати резервні копії.

Разом з тим, детальне вивчення особливостей таких додатків виявило й низку серйозних недоліків, які знижують їхню універсальність та зручність для широкого кола користувачів. Одним з головних обмежень є недостатня підтримка української мови – інтерфейс більшості застосунків переважно англомовний, що ускладнює використання програмного забезпечення для локального споживача. Безкоштовні версії часто мають значні функціональні обмеження, наприклад, заборону на створення додаткових категорій або недоступність повноцінної аналітики. Більшість програм зберігають дані винятково в хмарі, що знижує рівень конфіденційності та безпеки, особливо для користувачів, які не бажають передавати особисту фінансову інформацію на сторонні сервери. Інтерфейси багатьох рішень є перевантаженими, що створює труднощі у навігації для новачків або людей похилого віку. Окремим недоліком є відсутність можливості гнучкого налаштування категорій витрат і доходів, що обмежує адаптацію додатку до індивідуальних потреб користувача. Усі ці недоліки свідчать про необхідність створення нового локалізованого застосунку, який поєднує інтуїтивну простоту, автономність, безпечне зберігання даних та широкі можливості персоналізації.

Проведений патентний пошук не виявив специфічних патентованих рішень на території України, що безпосередньо стосуються Android-додатків, реалізованих мовою Dart з фреймворком Flutter, для ведення обліку особистих фінансів. Це свідчить про відносно низьку конкурентність у даній технологічній ніші на локальному рівні та відкриває перспективи для нових розробок.

У зв’язку з вищевикладеним, розробка нового Android-додатку, орієнтованого на локалізованого українського користувача, з інтуїтивно зрозумілим інтерфейсом, можливістю офлайн-зберігання даних та підтримкою візуального аналізу витрат і доходів, є доцільною та досить актуальною задачею. Така система дозволить користувачам ефективніше управляти особистими фінансами без потреби у складному функціоналі або підключенні до зовнішніх сервісів.

Один із найбільш ефективних підходів до вирішення поставленого завдання полягає у використанні сучасного стеку технологій, що забезпечує розробку швидкодійного, кросплатформного, безпечного та зручного у користуванні мобільного додатку. Зокрема, фреймворк Flutter у поєднанні з мовою програмування Dart дозволяє створювати нативні інтерфейси для Android із високою продуктивністю та широкими можливостями кастомізації. Для забезпечення локального зберігання даних можна використовувати SQLite разом із ORM-бібліотекою Drift, яка надає типобезпечну роботу з базою даних і підтримує реактивне оновлення інтерфейсу. Для автентифікації користувачів та забезпечення базового рівня захисту даних доцільно інтегрувати Firebase Authentication, що дозволяє безпечно керувати доступом без необхідності реалізації власної системи реєстрації. У поєднанні з архітектурними шаблонами на кшталт BLoC (Business Logic Component) і використанням візуалізацій через fl_chart, це створює комплексне технічне рішення, здатне повністю задовольнити функціональні та нефункціональні вимоги до системи обліку особистих фінансів.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У межах кваліфікаційної роботи бакалавра необхідно реалізувати комплекс завдань, що охоплюють усі етапи створення Android-додатку для моніторингу витрат та доходів. Завдання кваліфікаційної роботи бакалавра:

1) здійснити аналіз сучасного стану проблеми обліку особистих фінансів, дослідити наукові джерела, а також проаналізувати наявні мобільні застосунки-аналоги для виявлення їх переваг, недоліків та потенційних можливостей удосконалення;

2) визначити функціональні та нефункціональні вимоги до проєктованого програмного забезпечення, сформулювати основні сценарії використання, враховуючи очікування користувачів, специфіку платформи Android та загальні принципи побудови інформаційних систем;

3) обґрунтувати вибір архітектурного підходу, мови програмування Dart, фреймворку Flutter та допоміжних бібліотек, пояснити переваги такого вибору в контексті мобільної розробки, адаптивності та підтримки автономної роботи;

4) реалізувати Android-додаток, що забезпечує введення, редагування, перегляд та видалення фінансових операцій, категоризацію транзакцій, візуалізацію даних та базові функції бюджетування;

5) провести тестування функціональних модулів та інтерфейсу користувача, здійснити налагодження логіки додатку, виявити та усунути помилки, забезпечивши стабільну роботу програмного продукту на різних пристроях;

6) розробити локальну базу даних для зберігання інформації про транзакції, категорії та бюджети користувача, реалізувати серіалізацію даних через відповідні моделі та адаптери, використовуючи Hive як сховище;

7) створити супровідну документацію, яка містить системні вимоги, інструкцію користувача, довідкові матеріали, а також описує процедуру встановлення додатку та особливості його використання.

Виконання зазначених завдань забезпечить комплексну реалізацію мобільного застосунку, що відповідає сучасним вимогам до персонального фінансового обліку на платформі Android.

Висновки до розділу 1

У першому розділі було проведено комплексний аналіз сучасного стану проблеми обліку особистих фінансів, а також виконано огляд наукових джерел, технічної літератури та існуючих цифрових рішень у цій сфері. Особлива увага була приділена порівнянню функціональних можливостей популярних мобільних застосунків, таких як Monefy, Wallet, Money Manager, Spendee, AstroWealth та інших. У процесі аналізу з'ясовано, що хоча ринок особистих фінансових застосунків є досить насиченим, більшість із них мають суттєві недоліки. Зокрема, це перевантажені інтерфейси, складна навігація, обмеження у безкоштовних версіях, недостатня підтримка української мови, а також обов'язкове використання хмарного зберігання, яке часто не гарантує повної конфіденційності даних.

Узагальнення відгуків користувачів, функціонального аналізу та технічних характеристик аналогів дозволило виявити реальні потреби цільової аудиторії – простий, інтуїтивно зрозумілий додаток із базовими функціями обліку, мінімалістичним дизайном, візуалізацією доходів/витрат, повноцінною локалізацією та збереженням даних локально, без прив'язки до мережі. Ці висновки стали основою для формування чітко визначених завдань кваліфікаційної роботи бакалавра.

Метою стало створення Android-додатку для моніторингу витрат і доходів, який дозволить вести персональний облік фінансів з урахуванням категорій, рахунків, бюджетів та термінів, при цьому забезпечуючи користувачу зручний інтерфейс, адаптований для україномовного середовища, з можливістю фільтрації, перегляду історії та графічного відображення статистики. Також у розділі було окреслено науково-практичне

підґрунтя для виконання проєкту: проаналізовано наявні підходи до управління особистими фінансами, типологію мобільних додатків (локальні та хмарні), переваги автономних рішень, а також охарактеризовано основні виклики при створенні такого програмного продукту.

Загалом, обґрунтовано актуальність і доцільність розробки нового програмного засобу, який відповідатиме запитам сучасного користувача, не залежатиме від зовнішніх серверів, сприятиме підвищенню фінансової грамотності та дозволить ефективно управляти персональним бюджетом у зручному мобільному форматі. Визначені проблеми, вимоги і тенденції стали основою для подальшої постановки задач, проектування архітектури і реалізації технічної частини застосунку в наступних розділах роботи.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Метою розробки Android-додатку є створення персоналізованої мобільної інформаційної системи для обліку особистих фінансів. Застосування сучасних підходів до моделювання та визначення вимог дозволяє забезпечити повноцінне охоплення функціональних і нефункціональних характеристик майбутнього програмного продукту.

Для проєктування системи було обрано модель Clean Architecture, що передбачає багаторівневу структуру з чітким розмежуванням відповідальності між шарами: data, domain, presentation та app [10]. Такий підхід забезпечує легкість тестування, розширення і підтримки коду. Архітектура дозволяє відокремити бізнес-логіку від реалізації користувацького інтерфейсу, а також спростити заміну компонентів у майбутньому.

Вимоги до додатку було сформовано на основі аналізу аналогічних мобільних додатків (Money Lover, Monefy, Spendee, Wallet), опитування цільової аудиторії, експертного оцінювання функцій, необхідних для щоденного обліку фінансів, а також аналізу типових сценаріїв користування додатком.

Функціональні вимоги до системи.

1. Управління транзакціями:
    а) створення, редагування, видалення записів доходів та витрат;
    б) категоризація записів (їжа, транспорт, медичні витрати тощо);
    в) пошук і фільтрація транзакцій за періодом, сумою, типом, категорією.
2. Аналітика:
    а) побудова кругових, стовпчикових та лінійних діаграм;
    б) порівняння доходів та витрат по місяцях;

в) інтерактивна візуалізація витрат за категоріями.

3. Бюджетування:

а) встановлення бюджетів на тиждень або місяць;

б) відображення рівня використання бюджету;

в) повідомлення про перевищення встановлених лімітів.

4. Керування категоріями:

а) створення/редагування власних категорій;

б) використання вбудованого списку категорій.

5. Обробка даних:

а) локальне збереження даних за допомогою Hive;

б) створення резервних копій (опціонально);

в) обробка введеної інформації з перевіркою на валідність.

6. Безпека та доступ:

а) можливість встановлення PIN-коду або біометричної автентифікації;

б) захист даних від несанкціонованого доступу;

в) можливість локального шифрування даних.

Нефункціональні вимоги:

1) платформа – Android (API 21+), Flutter;

2) продуктивність – завантаження транзакцій ≤ 200 мс;

3) інтерфейс – підтримка світлої і темної теми, адаптивність до різних екранів;

4) підтримка майбутніх оновлень з додаванням функцій (наприклад, спільного бюджету).

Інтерфейс реалізовано відповідно до принципів Material Design. Основні екрани:

1) головний екран з короткою аналітикою;

2) журнал транзакцій;

3) сторінка додавання доходу/витрати;

4) бюджетування;

5) профіль користувача;

6) сторінка налаштувань.

Користувач взаємодіє з додатком через інтуїтивно зрозумілі кнопки, випадаючі списки та графіки. Інформаційні потоки побудовано з урахуванням швидкого доступу до основних функцій з мінімальною кількістю кліків.

Для формалізації моделі додатку побудовано UML-діаграми. Use Case діаграма (рис. 2.1) показує взаємодію користувача з основними функціями системи.

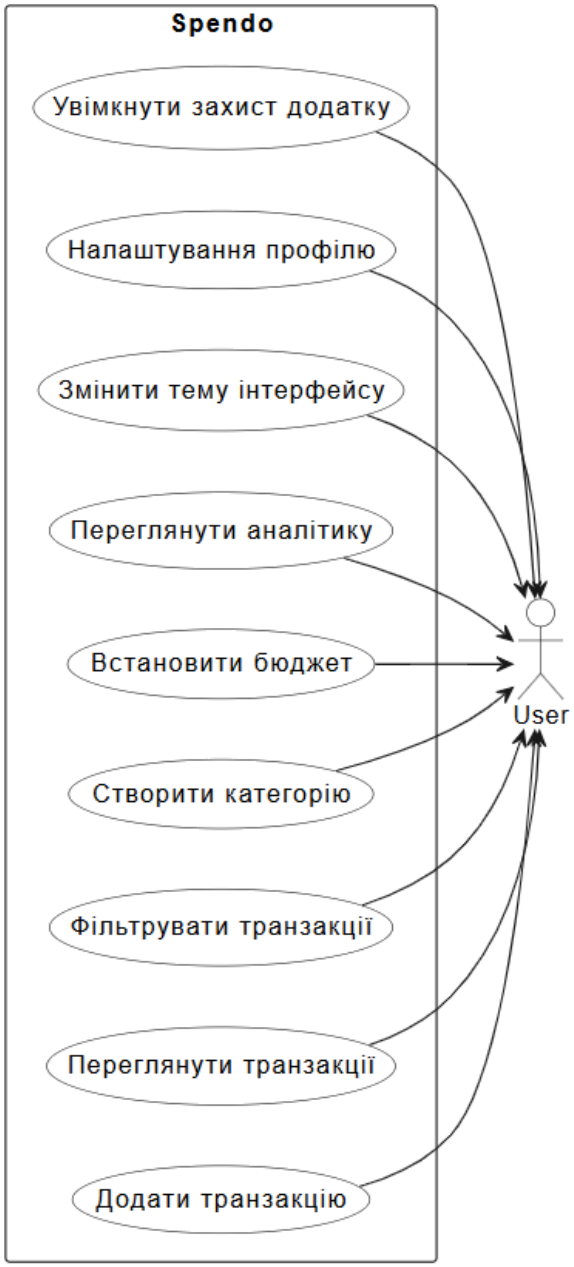

Рисунок 2.1 – Діаграма варіантів використання

Центральним учасником є користувач, який ініціює дії, наприклад, додавання транзакції, фільтрацію, встановлення бюджету або перегляд діаграм. Кожен сценарій позначено окремим use case, що дозволяє структурувати функціональність додатку і зрозуміти вимоги до інтерфейсу. Ця діаграма є основою для побудови моделі інтерфейсу та подальшого програмування логіки взаємодії, оскільки чітко показує обов'язкові компоненти додатку з точки зору користувача.

Діаграма класів (рис. 2.2) ілюструє структуру основних сутностей, які реалізовані в додатку. Клас Transaction описує окрему фінансову операцію, що пов'язана з Category. Клас Budget прив'язується до певної категорії і дозволяє контролювати витрати за обраний період. Клас UserSettings відповідає за збереження персональних налаштувань користувача. Ця діаграма дозволяє логічно структурувати модель даних і закласти основу для зберігання у локальній базі (Hive), а також допомагає при проєктуванні UI-форм введення та редагування об'єктів.

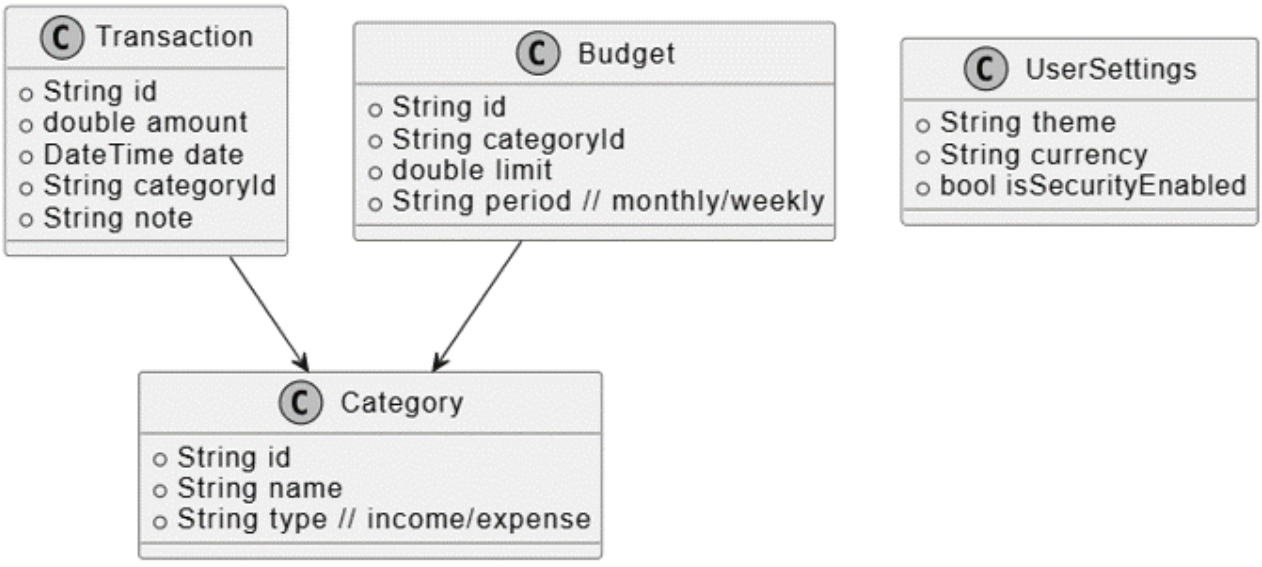

Рисунок 2.2 – Діаграма класів

Діаграма послідовностей (рис. 2.3) демонструє типовий сценарій роботи додатку – додавання нової транзакції. Користувач вводить дані на екрані інтерфейсу, які передаються до логіки бізнес-рівня через Cubit/BLoC. Далі дані

надсилаються до репозиторію, що забезпечує збереження через локальну БД Hive. Після підтвердження запису відображається повідомлення про успішне додавання. Діаграма послідовності чітко ілюструє порядок обміну повідомленнями між компонентами системи. Вона допомагає при проєктуванні архітектури взаємодії між UI, логікою та збереженням даних.

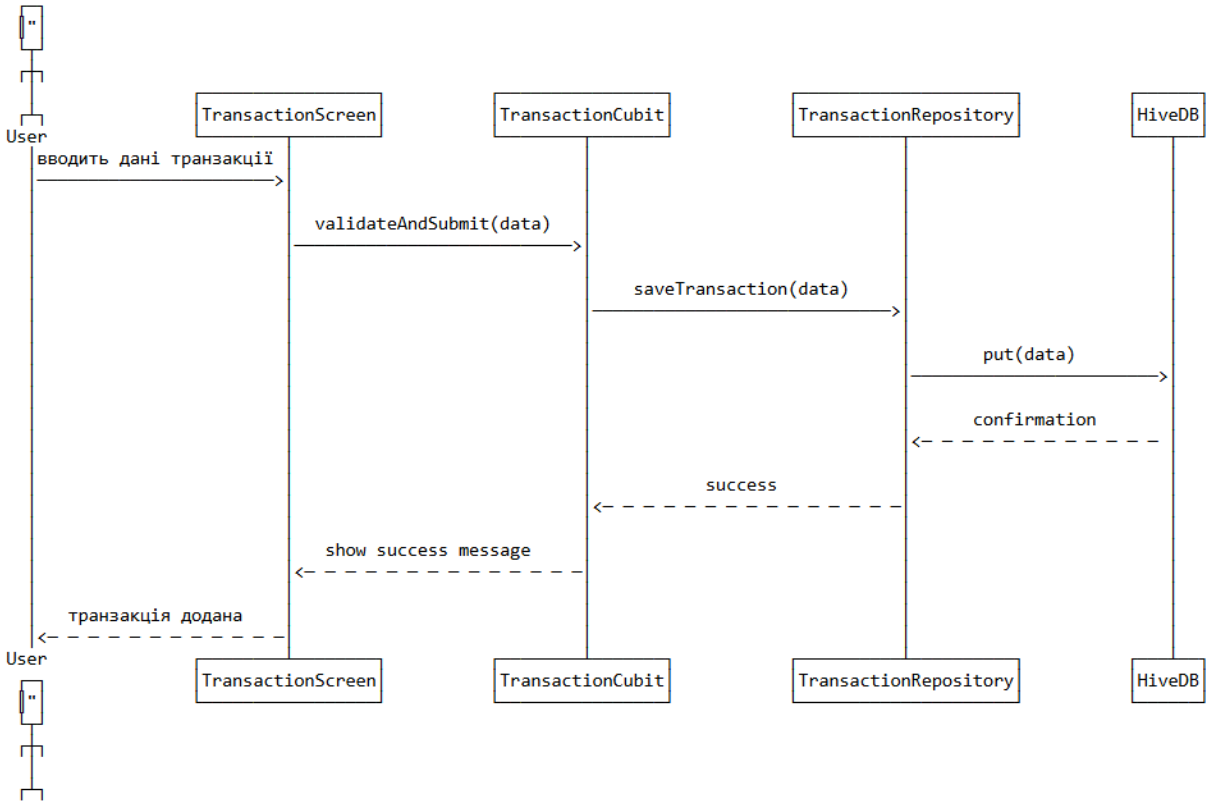

Рисунок 2.3 – Діаграма послідовностей

Можна зробити висновок, що проєктування додатку на основі принципів Clean Architecture дозволило досягти досить високої гнучкості системи, можливості повторного використання коду та легкості масштабування. Поділ на окремі шари забезпечив чітке розмежування логіки і залежностей. А розроблені UML-моделі сприяли уніфікованому розумінню структури системи на етапі реалізації.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі розробки Android-додатку для моніторингу особистих доходів і витрат було проаналізовано широкий спектр сучасних технологій, інструментів і бібліотек. Основною метою вибору є забезпечення зручності для користувача, стабільної роботи додатку, гнучкості архітектури та можливості масштабування в майбутньому. Ретельний підбір інструментів дозволив створити мобільний застосунок, що поєднує естетику інтерфейсу, швидкодію та логічну структуру.

Для реалізації клієнтської частини мобільного додатку було обрано фреймворк Flutter, розроблений компанією Google, який є одним із найпопулярніших інструментів для створення кросплатформених застосунків [11]. Це рішення дозволяє створювати мобільні, веб- та десктопні додатки з єдиною кодовою базою, що значно спрощує підтримку та розвиток проєкту. Однією з основних переваг Flutter є висока швидкість розробки завдяки технології Hot Reload, яка дає змогу миттєво відображати зміни у коді без повного перезапуску застосунку [12]. Інтерфейс будується на основі принципів Material Design, що забезпечує сучасний та естетичний вигляд усіх візуальних компонентів. Важливо також зазначити, що Flutter підтримує створення адаптивного інтерфейсу, здатного коректно відображатися на пристроях з різними розмірами екранів і типами орієнтації [13].

Мова програмування Dart, яка є основною у Flutter, забезпечує зручний об’єктно-орієнтований підхід до розробки та підтримує строгий контроль типів [14]. Це дозволяє уникати великої кількості помилок на етапі компіляції та забезпечує більш надійне функціонування додатку. Крім того, Dart добре інтегрується з архітектурними шаблонами, включаючи clean architecture, що дозволяє структурувати код за принципами високої модульності та розділення відповідальності [15].

Для зберігання даних у додатку було обрано Hive – легковагове NoSQL-сховище, оптимізоване для мобільних пристроїв. Hive не потребує складної

конфігурації або використання ORM-обгорток, оскільки дозволяє серіалізувати та зберігати Dart-об’єкти безпосередньо [16]. Завдяки цьому зберігання та обробка даних відбувається швидко та ефективно. Крім того, Hive підтримує шифрування даних, що особливо важливо для збереження конфіденційності фінансової інформації користувача [17].

У якості засобу управління станами додатку використано BLoC (Business Logic Component) та його спрощену версію Cubit, які реалізують реактивну модель програмування [18]. Ці технології дають змогу чітко розділити користувацький інтерфейс та бізнес-логіку, що позитивно впливає на підтримку та масштабованість проєкту. Вони знижують зв’язність між компонентами застосунку, що дозволяє незалежно змінювати окремі модулі без ризику порушення функціональності всієї системи. До того ж, використання BLoC/Cubit спрощує написання модульних тестів, що сприяє покращенню якості програмного забезпечення та полегшує виявлення і усунення помилок.

Візуалізація даних відбувається за допомогою Syncfusion Flutter Charts. Ця бібліотека дозволяє створювати різноманітні графіки та діаграми, включаючи кругові (pie), стовпчикові (bar), лінійні (line), області (area) тощо. Вона підтримує анімацію, взаємодію з користувачем (наприклад, дотиком можна переглянути деталі значення) та кастомізацію зовнішнього вигляду. У додатку її використано для побудови фінансової аналітики – зокрема, для візуального відображення доходів і витрат за категоріями у вигляді кругових діаграм.

Анімації та мікроінтеракції реалізовані з Lottie, Rive. Lottie та Rive – це потужні інструменти для створення інтерактивної анімації, що не впливає на продуктивність додатку. Lottie дозволяє виводити анімації, згенеровані у форматі JSON через Adobe After Effects, а Rive – створювати інтерактивні UI-анімації, що реагують на події користувача. У додатку ці бібліотеки використовуються для візуального покращення екранів onboarding’у (початкове знайомство з додатком), завантажень, переходів між екранами та інформування про статус операцій.

SVG-графіка – flutter_svg. Ця бібліотека забезпечує відображення зображень у форматі SVG (Scalable Vector Graphics), що особливо важливо для збереження чіткості іконок на екранах з високою роздільною здатністю. Вона підтримує зміну кольору SVG-фігур, адаптацію під тему (темна/світла), а також інтегрується з системою інтерфейсу Flutter. У застосунку її використано для відображення іконок категорій витрат і доходів, стрілок навігації та інших UI-елементів.

Локалізація – intl. Бібліотека intl (Internationalization) відповідає за локалізацію додатку: переклад текстів інтерфейсу, форматування чисел, дат, валют згідно з регіональними стандартами. Вона дозволяє легко додати багатомовну підтримку у Flutter-додаток. У проєкті реалізовано локалізацію українською мовою, з потенційною можливістю масштабування до інших мов.

Унікальні ідентифікатори об’єктів – uuid. Пакет uuid (Universally Unique Identifier) генерує унікальні ідентифікатори для об'єктів, що особливо корисно для транзакцій, категорій, бюджетів тощо. Використання UUID гарантує, що кожен запис має унікальний ключ незалежно від порядку створення або повторного запуску програми. Це мінімізує ймовірність конфліктів або дублювання.

Порівняння сутностей – equatable. Equatable дозволяє спростити порівняння об'єктів у Dart, забезпечуючи правильне визначення рівності між екземплярами. Це особливо важливо при роботі з BLoC, де стан додатку залежить від того, чи змінювався об'єкт. Бібліотека дозволяє зменшити boilerplate-код та підвищити надійність перевірок рівності між станами або моделями.

Залежності – get_it (DI, service locator). get_it – це реалізація шаблону service locator у Flutter, який дозволяє централізовано керувати залежностями між класами. Він використовується для реєстрації та доступу до об’єктів, таких як репозиторії, BLoC-и або сервісні класи, без жорсткого зв’язування. Такий підхід відповідає принципам інверсії залежностей (Dependency Inversion) у

Clean Architecture і значно спрощує підтримку, масштабування та тестування додатку.

У процесі розробки мобільного додатку було реалізовано низку прикладних алгоритмів, які забезпечують виконання основних функціональних задач системи, зокрема збереження, обробки, візуалізації та контролю фінансових даних користувача.

Основним функціональним блоком додатку є алгоритм обліку транзакцій. Алгоритм передбачає формування об’єкта типу TransactionModel, який включає такі поля, як унікальний ідентифікатор (генерується через UUID), тип операції (income або expense), дату транзакції, суму, категорію та примітку користувача. Після введення даних користувачем відбувається валідація введених значень (наприклад, перевірка, щоб сума була додатною, а дата не перевищувала поточну). Далі об’єкт серіалізується і зберігається у сховище Hive.

Для ефективного зчитування інформації алгоритм передбачає фільтрацію транзакцій за певними параметрами, такими як:

1) тип операції (дохід / витрата);
2) діапазон дат (від і до);
3) категорія (наприклад, «Їжа», «Транспорт»);
4) мінімальна або максимальна сума.

Індексація відбувається за допомогою перетворення сховища Hive у список (Box.values.toList()), після чого застосовується фільтрація через методи .where() з переданими критеріями.

Для створення зручної аналітики та візуального сприйняття даних реалізовано набір алгоритмів формування графічних представлень:

1) кругова діаграма (Pie Chart) – показує розподіл витрат або доходів за категоріями. Алгоритм агрегує суми всіх транзакцій за певний період і групує їх за ключем категорії. Кожен сегмент діаграми відповідає одній категорії, а його кутова величина пропорційна сумі витрат/доходів у цій категорії;

2) стовпчикова діаграма (Bar Chart) – порівнює доходи та витрати за різні періоди (дні, тижні, місяці). Алгоритм формує окремі списки транзакцій,

агрегацію яких виконує за часовим маркером, після чого відображає окремі серії даних на діаграмі;

3) лінійна діаграма (Line Chart) – використовується для демонстрації динаміки змін балансу в часі. Вона базується на обчисленні кумулятивного балансу на кожен день/період, що візуалізується у вигляді лінії на графіку.

Ці алгоритми реалізовані через бібліотеку fl_chart з підтримкою анімації, кастомізації кольорів і інтерактивної взаємодії з елементами діаграм (наприклад, підсвічування при натисканні).

Алгоритм контролю бюджету бюджетування базується на логіці регулярного відстеження витрат по кожній категорії з порівнянням встановлених лімітів. Кожен бюджетний запис містить такі параметри:

1) categoryId – категорія, до якої застосовується бюджет;

2) limit – гранична сума;

3) period – тип періоду (щомісячний або щотижневий).

Алгоритм щоденно або після додавання нової транзакції перевіряє суму витрат за відповідну категорію у межах поточного періоду. Якщо фактична сума витрат перевищує встановлений ліміт, виводиться попередження або зміна кольору індикатора бюджету (наприклад, на червоний). У разі близького до вичерпання ліміту значення бюджету підсвічується жовтим.

Цей підхід реалізує реактивну перевірку бюджету, що не потребує ручного оновлення, оскільки використовується stream-зв’язок між джерелом даних і UI-компонентом.

Алгоритм розрахунку залишку є ще одним ключовим елементом є обчислення загального балансу (залишку), який розраховується за формулою. Цей розрахунок виконується щоразу при оновленні списку транзакцій і відображається на головному екрані. Алгоритм враховує як усі транзакції за весь період, так і фільтровані за поточний місяць або тиждень, залежно від обраного режиму перегляду.

Використані алгоритми дозволяють забезпечити не лише базове зберігання та обробку транзакцій, а й надають широкі можливості для

фінансової аналітики, бюджетування та зручного візуального представлення інформації. Їх реалізація у середовищі Flutter з використанням Hive, BLoC та fl_chart забезпечує високу ефективність, масштабованість і стабільність роботи застосунку в реальному середовищі.

Методи, використані при розробці Android-додатку для моніторингу витрат та доходів, охоплюють сучасні архітектурні підходи, принципи програмної інженерії та концепції структурної організації застосунку. Центральним методичним рішенням стало впровадження моделі Clean Architecture, яка забезпечує розмежування логіки на чітко визначені шари: domain (предметна область), data (джерела даних) і presentation (інтерфейс користувача). Така організація коду сприяє масштабованості, повторному використанню компонентів, підвищенню тестованості та зниженню зв’язності між модулями.

Метод використання інверсії залежностей був реалізований за допомогою підходу Dependency Injection через бібліотеку get_it, що дало змогу зручно управляти зв’язками між компонентами додатку, зокрема сервісами, репозиторіями та use case-ами. Це дозволило уникнути жорсткого зв’язування між класами і спростити супровід додатку.

Застосування реактивного програмування на основі шаблонів BLoC (Business Logic Component) та Cubit стало ще одним ключовим методологічним рішенням. Ці патерни дозволяють відокремити інтерфейс від бізнес-логіки, що підвищує стабільність коду та забезпечує предикативність його поведінки в умовах змін стану. Також це спростило тестування логіки, оскільки дозволяє відокремлено перевіряти зміну станів без впливу графічної частини.

Методи типобезпечної роботи з даними, що реалізовані за допомогою бібліотеки Hive, забезпечують надійне зберігання та серіалізацію об’єктів Dart. Кожна модель позначається через відповідні анотації, що дозволяє уникнути помилок типів під час збереження і зчитування даних. Hive також підтримує шифрування сховища, що дозволяє дотримуватися методів захисту інформації згідно з сучасними вимогами конфіденційності.

Важливою частиною методичної бази стала локалізація інтерфейсу, реалізована на основі пакета intl, що дозволяє адаптувати текстовий вміст до різних мовних середовищ. Це забезпечує доступність додатку для українськомовної аудиторії та створює передумови для масштабування проєкту на інші регіони.

Загалом, використані методи у сукупності формують комплексний підхід до побудови мобільного програмного забезпечення, що відповідає сучасним практикам, забезпечує підтримку, розширення й надійну роботу додатку в умовах реального використання.

На рисунку 2.4 зображена UML-діаграма компонентів системи.

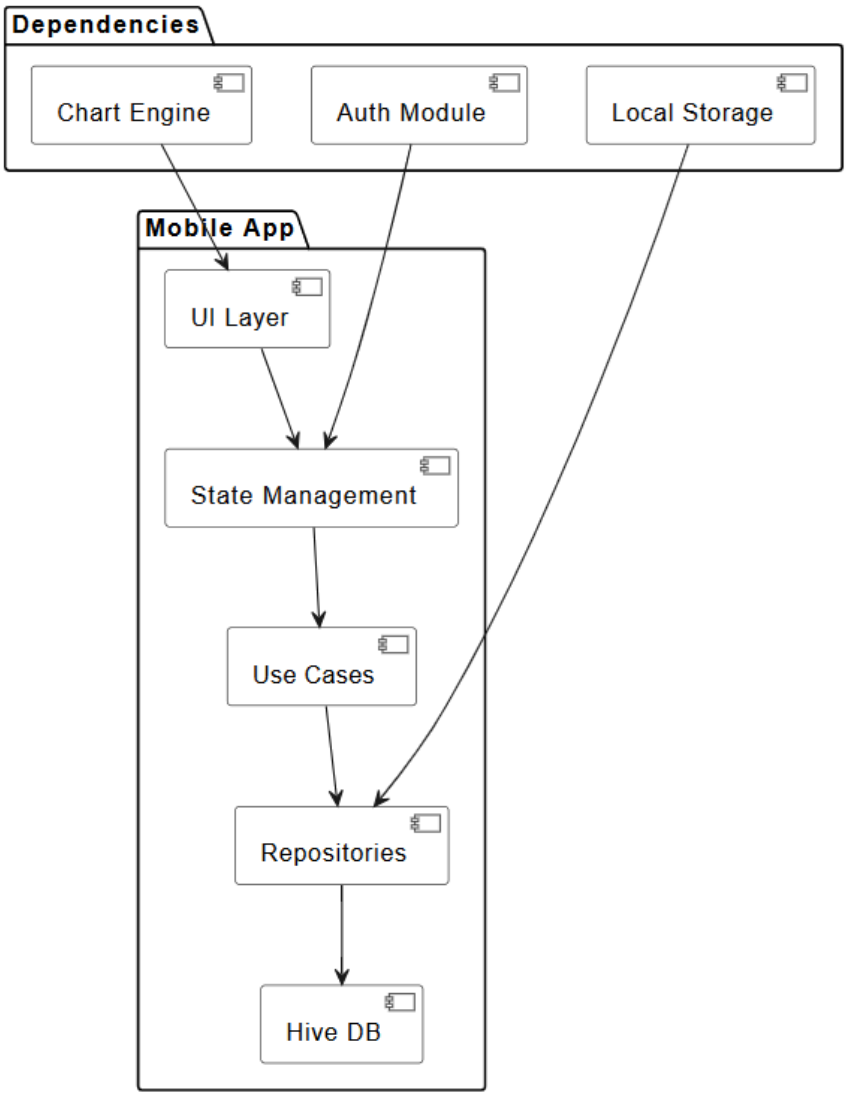

Рисунок 2.4 – Діаграма компонентів

Ця діаграма демонструє взаємозв’язки між основними компонентами мобільного застосунку. Вхідна точка в ній, а саме UI – обробляє події через BLoC або Cubit, передає їх у use cases, що оперують з базою через репозиторії. Вся система будується на модульному підході, що спрощує тестування і супровід.

Діаграма розгортання на рисунку 2.5 ілюструє фізичне розміщення основних компонентів додатку Spendo у середовищі Android-пристрою та за потреби – у хмарному середовищі (Firebase). Основна логіка виконується на мобільному пристрої користувача, де розміщено інтерфейс, бізнес-логіку (через BLoC/Cubit), локальну базу даних (Hive) і файл налаштувань користувача. Firebase Cloud є додатковим компонентом, що використовується для CI/CD через App Distribution (автоматична доставка збірок), а також для push-сповіщень. Взаємодія між компонентами є локальною, швидкодійною, з можливістю асинхронної обробки даних.

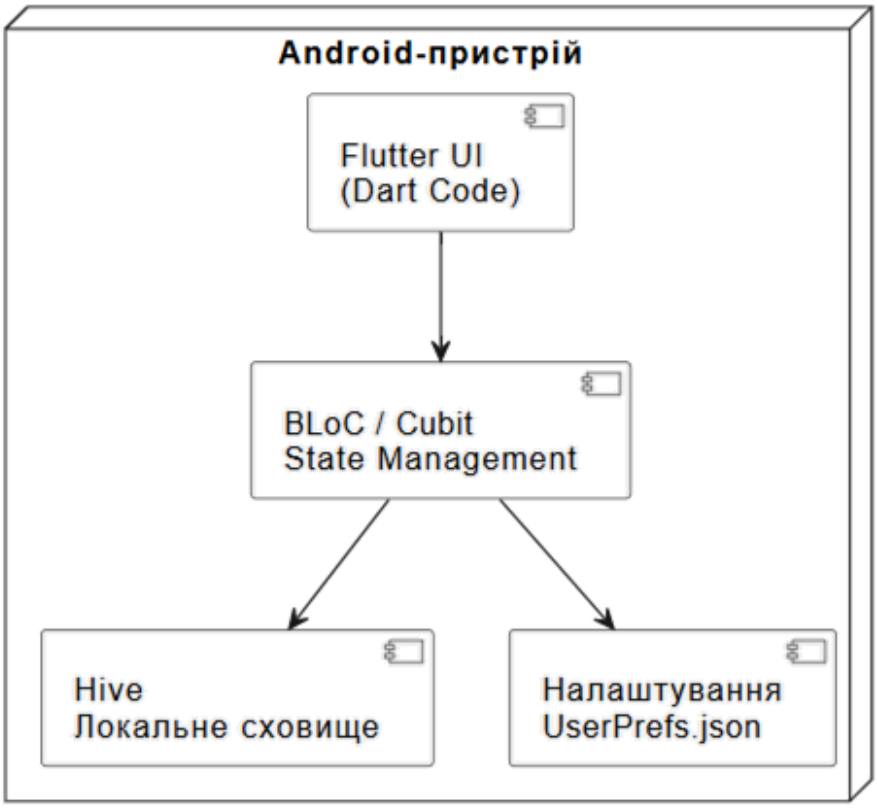

Рисунок 2.5 – Діаграма розгортання

Цей підхід гарантує високу автономність додатку, а також дозволяє забезпечити гнучке оновлення, масштабованість і додаткові сервіси за потреби (наприклад, хмарне резервне копіювання у майбутньому).

Висновки до розділу 2

У другому розділі було проведено комплексну роботу з формування вимог до програмного забезпечення та обґрунтовано вибір архітектурних і технологічних рішень, що є досить важливими для ефективної реалізації мобільного додатку з обліку фінансів. Особливу увагу було приділено застосуванню формалізованих методів моделювання та побудові UML-діаграм, які дозволили чітко структуризувати функціональність, логіку та структуру взаємодії між компонентами додатку.

На основі аналізу предметної області визначено основні функціональні можливості системи: введення, редагування та фільтрація транзакцій, побудова аналітичних графіків, планування бюджету, захист даних і кастомізація інтерфейсу. Всі вимоги систематизовані у вигляді сценаріїв, таблиць і діаграм, що значно полегшує подальшу реалізацію.

У процесі вибору інструментів реалізації було враховано продуктивність, масштабованість, кросплатформеність, рівень спільноти підтримки та можливості інтеграції. В результаті обрано Flutter як основний фреймворк для розробки інтерфейсу, Dart як мову програмування, Hive як швидке NoSQL-сховище, та BLoC/Cubit для управління станами. Додатково було інтегровано бібліотеки Syncfusion, Lottie, flutter_svg, intl, equatable тощо.

Вибрані засоби дозволяють ефективно вирішувати прикладні задачі в межах теми кваліфікаційної роботи бакалавра, зберігаючи високу швидкодію застосунку, гнучкість в адаптації до потреб користувача та можливість масштабування у майбутньому. Побудовані UML-діаграми (сценаріїв використання, класів, послідовностей, компонентів, розгортання) свідчать про

правильну структуризацію логіки додатку і дозволяють мінімізувати ризики при впровадженні.

Загалом, у розділі були закладені надійні концептуальні основи для розробки мобільного застосунку, який відповідає сучасним стандартам проєктування та вимогам до мобільного ПЗ у сфері фінансових технологій.

РОЗДІЛ 3
РОЗРОБКА ANDROID-ДОДАТКУ ДЛЯ ОБЛІКУ ВИТРАТ ТА ДОХОДІВ

3.1 Практична реалізація об’єкта проєктування

Розробка Android-додатку для обліку особистих фінансів реалізована з використанням фреймворку Flutter та мови програмування Dart у середовищі розробки Visual Studio Code. Для зберігання даних застосовано локальну базу Hive, що дозволяє зберігати об’єкти Dart без додаткових конвертацій у SQL-структури.

Проєкт побудований за принципами чистої архітектури (Clean Architecture), що дозволяє відокремити бізнес-логіку, джерела даних та інтерфейс користувача. Це досягається за рахунок розподілу структури на відповідні шари: domain, data, core та features.

Основною точкою входу до застосунку є файл main.dart, у якому ініціалізується Hive, реєструються всі необхідні адаптери для моделей, відкриваються box-и для зберігання транзакцій, категорій, валют і бюджетів, а також виконується конфігурація зовнішнього вигляду системних панелей. У фіналі відбувається запуск віджета App() (ліст. 3.1).

Лістинг 3.1 – Ініціалізація Hive, відкриття box-ів і запуск додатку

```
await Hive.openBox<CategoryModel>('category-database');
await Hive.openBox<CurrencyModel>('currency-database');
await Hive.openBox<BudgetModel>('budget-database');
runApp(const App());
```

кінець лістингу 3.1

Клас App відповідає за конфігурацію інтерфейсу користувача та підключення станів через MultiBlocProvider. Для кожного функціонального модуля додатку створено окремий BLoC або Cubit, що відповідає за обробку подій, логіку взаємодії з use case-ами та передачу стану до UI-компонентів. Наприклад, підключення TransactionBloc відбувається наступним чином (ліст. 3.2).

Лістинг 3.2 – Реєстрація TransactionBloc через BlocProvider

```
BlocProvider(
  create: (_) => sl<TransactionBloc>(),
),
```

кінець лістингу 3.2

Налаштування роутінгу реалізовано у файлі app_routes.dart, де використовується метод onGenerateRoute, який забезпечує навігацію між екранами. Типовий маршрут виглядає наступним чином (ліст. 3.3).

Лістинг 3.3 – Налаштування маршруту splash-екрана через onGenerateRoute

```
case AppRoutes.splash:
  return MaterialPageRoute(
    builder: (_) => const SplashScreen(),
  );
```

кінець лістингу 3.3

Збереження та обробка транзакцій реалізовані за допомогою класу TransactionEntity, який є предметною сутністю. Усі дії з нею виконуються через відповідні use case-и, наприклад AddTransactionUseCase, який викликається з TransactionBloc (ліст. 3.4).

Лістинг 3.4 – Виклик use case для додавання транзакції

```
final result = await
repository.addTransaction(params!);
```

кінець лістингу 3.4

Для зберігання транзакцій використовується клас TransactionModel, що реалізує серіалізацію з допомогою Hive (ліст. 3.5).

Лістинг 3.5 – Оголошення моделі TransactionModel з анотаціями Hive

```
@HiveType(typeId: 4)
class TransactionModel {
  @HiveField(0) final double amount;
  @HiveField(2) final DateTime date;
  @HiveField(6) String? id;
```

```
TransactionModel(this.amount, this.date, {this.id});
}
```

кінець лістингу 3.5

Категорії представлені у вигляді переліку типів, закодованих у перерахуванні CategoryType, та зберігаються у CategoryModel. Їх візуальне відображення реалізовано через компонент CategoryIconWidget, який на основі розширення CategoryTypeExtension підбирає SVG-іконку та відповідне фонове забарвлення (ліст. 3.6).

Лістинг 3.6 – Метод вибору SVG-іконки та кольору для категорії

```
SvgEnumModel get svgEnumModel {
  switch (this) {
    case CategoryType.shopping:
      return SvgEnumModel(
        svgAsset: 'assets/svg/category/shopping_bag.svg',
        backgroundColor: const Color(0xFFFF8A00),
      );
    ...
  }
}
```

кінець лістингу 3.6

Взаємодія з джерелами даних реалізована через класи CategoryLocalDataSourceImpl та TransactionLocalDataSourceImpl. Операції запису і читання даних з Hive відбуваються через відповідні методи (ліст. 3.7).

Лістинг 3.7 – Приклади запису та зчитування даних у Hive

```
await categoryDB.put(category.id, category);
final list = transactionDb.values.toList().reversed.toList();
```

кінець лістингу 3.7

Також реалізовано підтримку кастомного інтерфейсу за допомогою таких елементів, як customAppBar, що забезпечує уніфіковану верхню панель з адаптивною SVG-іконкою та заголовком, залежно від темної або світлої теми інтерфейсу.

На рисунку 3.1 зображено перший слайд onboarding-послідовності, що супроводжує нового користувача під час першого запуску додатку. Тут представлено загальну мету застосунку – відстеження доходів та витрат. Анімація та лаконічний текст сприяють кращому розумінню функціоналу ще до початку використання.

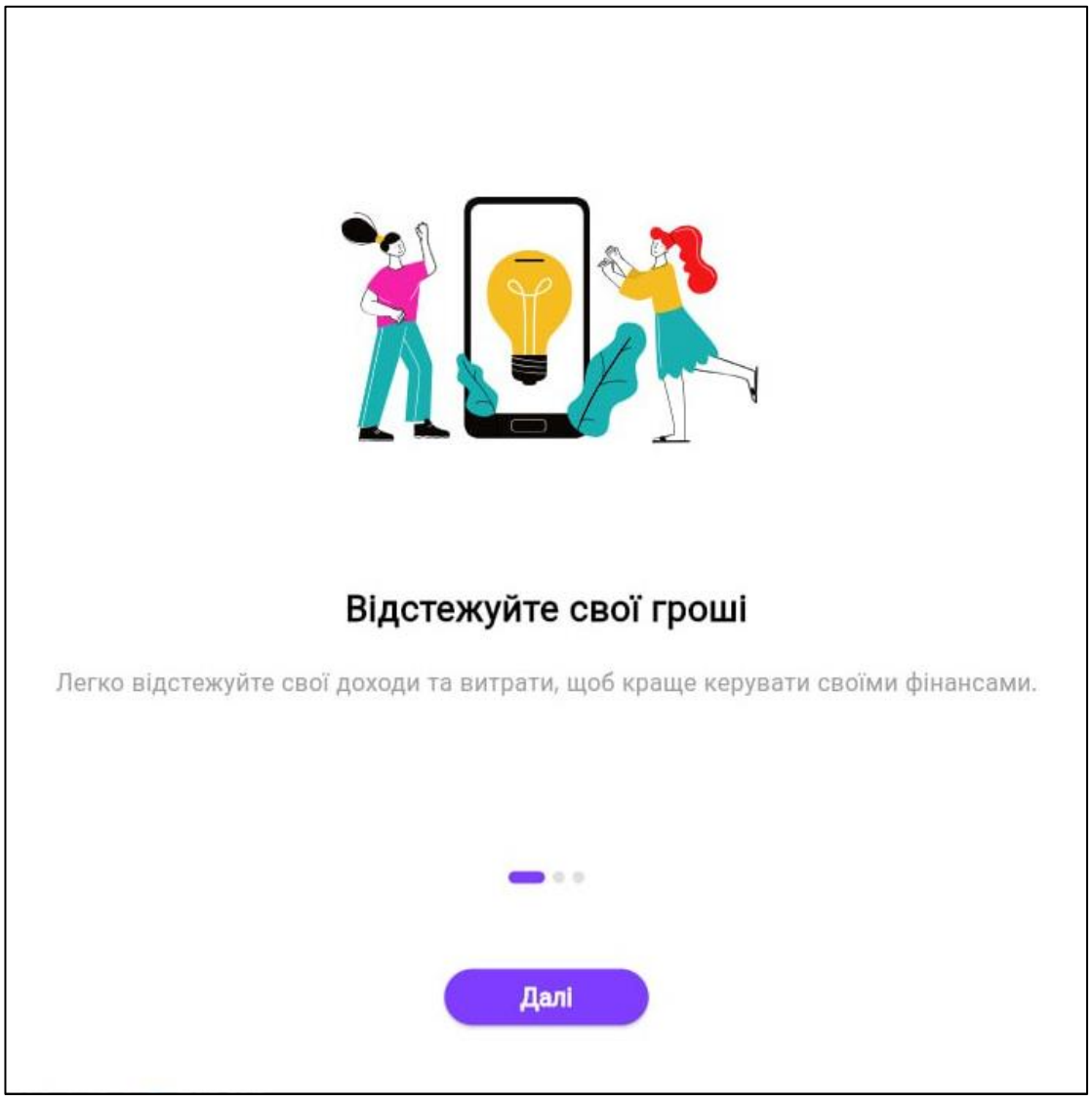

Рисунок 3.1 – Початковий екран для нового користувача

На рисунку 3.2 демонструється аналітична візуалізація даних користувача за останні 30 днів у вигляді кругової діаграми. Категорії доходів і витрат відображаються у вигляді кольорових сегментів, а під діаграмою наводиться підсумкова інформація про загальний дохід, витрати та залишок.

Рисунок 3.2 – Діаграма аналітики доходів і витрат

В свою чергу, рисунок 3.3 демонструє центральний екран додатку, на якому відображається загальний баланс, підсумки доходів та витрат, а також останні транзакції, згруповані за категоріями. Інтерфейс підтримує навігацію за допомогою нижнього меню з доступом до головної сторінки, категорій і профілю користувача.

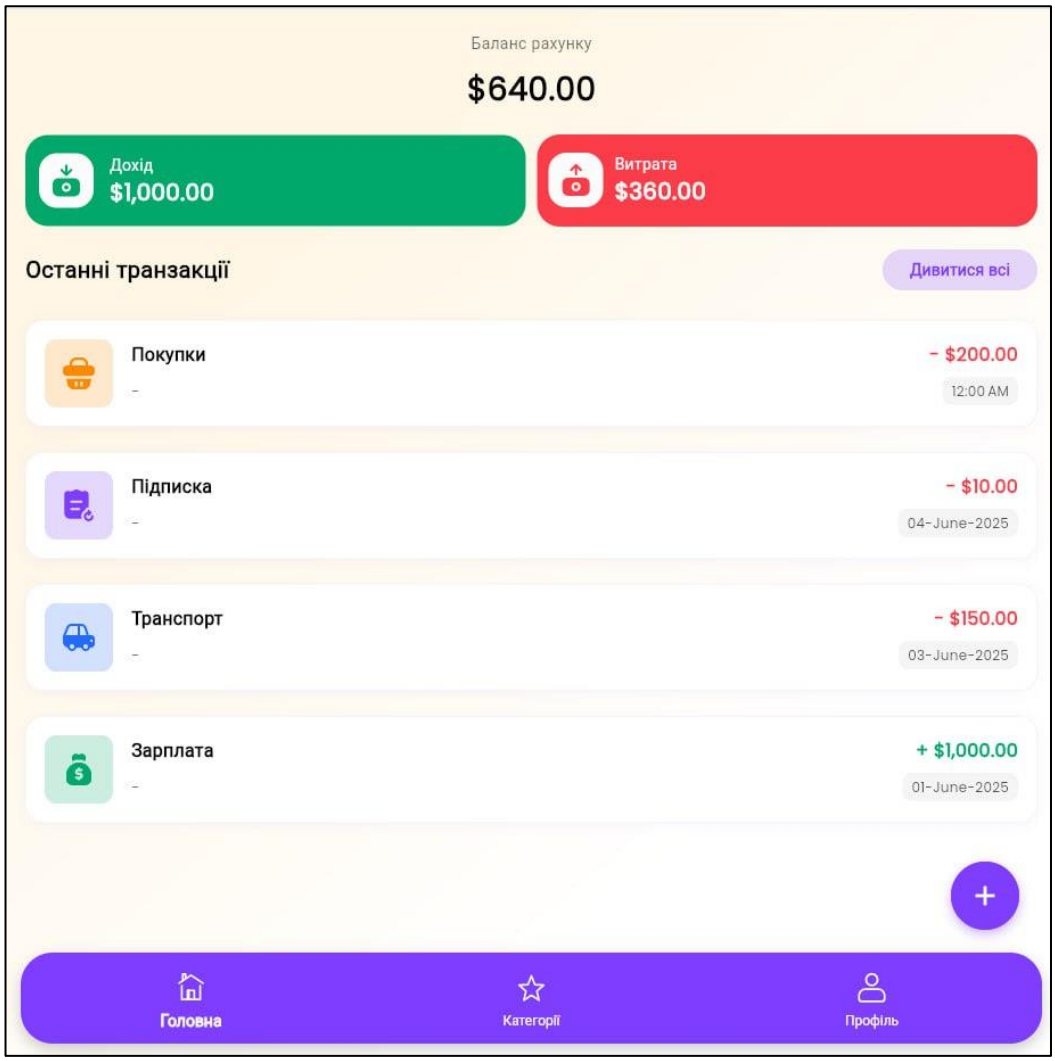

Рисунок 3.3 – Головний екран користувача

Загалом, у межах реалізації проєкту було продемонстровано практичне використання мови Dart та екосистеми Flutter для створення модульного, масштабованого та адаптивного Android-додатку. Структурованість коду, використання шаблонів BLoC, репозиторіїв, use case-ів та локального збереження даних у Hive підтверджує здатність автора застосовувати сучасні технології та принципи інженерії програмного забезпечення на практиці.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

У процесі розробки Android-додатку для обліку витрат і доходів було проведено комплекс тестування, що охоплює як функціональні, так і

нефункціональні характеристики системи. Тестування здійснювалося поетапно, починаючи з перевірки коректності логіки бізнес-операцій на рівні use case-ів, завершуючи валідацією відображення інтерфейсу та стабільності роботи додатку на реальних пристроях.

Значну увагу було приділено модульному тестуванню, яке дозволило перевірити правильність роботи логіки на кожному окремому рівні архітектури. Для цього були створені окремі Dart-тести, що охоплювали функціональність репозиторіїв, джерел даних та окремих use case-ів. Зокрема, тестування передбачало перевірку додавання нових транзакцій, їх успішного зчитування та збереження у локальному сховищі. Аналогічним чином перевірялася коректність роботи з категоріями, включно з можливістю додавання нових категорій та ініціалізацією списку стандартних категорій при першому запуску. Окремі тести стосувалися логіки оновлення бюджету з урахуванням періодичності, а також алгоритмів фільтрації транзакцій за типом (дохід або витрата) та за часовим діапазоном.

Налагодження додатку здійснювалося з використанням інструментів нативного дебагінгу Flutter DevTools, що надають можливість моніторингу станів, перевірки викликів функцій, відстеження часу виконання та навантаження на ресурси пристрою. Крім того, активно використовувався журнал помилок (debug console), який дозволяв оперативно реагувати на виняткові ситуації та логічні помилки в коді.

У ході налагодження було виявлено кілька типових проблем, серед яких – некоректна обробка порожніх полів під час створення нової транзакції, що призводила до збоїв при збереженні даних. Також було зафіксовано дублювання категорій, яке виникало при кожному повторному запуску програми внаслідок неефективної перевірки наявності вже збережених категорій. Ще однією проблемою став конфлікт між темним режимом інтерфейсу та SVG-іконками, зокрема при їх відображенні на контрастному фоні.

Усі виявлені недоліки були усунені шляхом введення додаткової валідації введених даних на рівні UI та використання логіки перевірки isAdapterRegistered(...) перед повторною реєстрацією адаптерів Hive. Також було оптимізовано використання StatefulWidget для окремих компонентів, аби уникнути непотрібного перерендерингу при навігації між вкладками.

З метою забезпечення стабільної роботи додатку, були сформовані мінімальні системні вимоги:

1) операційна система Android 8.0 (API level 26) або новіша;
2) ОЗП пристрою не менше 2 ГБ;
3) вільне місце на диску від 50 МБ;
4) дозвіл екрану 720×1280 px або вище;
5) інтернет-з’єднання не обов’язкове, оскільки додаток працює в офлайн-режимі.

Рекомендовані параметри для максимально комфортного використання:

1) Android 10.0 або новіше;
2) 4+ ГБ оперативної пам’яті;
3) процесор з частотою не менше 1.8 ГГц;
4) AMOLED-дисплей із роздільною здатністю Full HD.

Після фінального етапу тестування додаток стабільно працює в реальному середовищі, включаючи сценарії додавання, редагування, перегляду транзакцій, відображення аналітики, налаштування профілю та теми інтерфейсу. Було також перевірено збереження даних при повторному запуску, відповідність стилів у темному/світлому режимах та адекватність реакції додатку на зміну системної мови.

В цілому, тестування підтвердило функціональну та візуальну цілісність додатку. Всі критичні помилки були своєчасно виявлені й усунуті, що дозволило забезпечити надійну й передбачувану поведінку інформаційної системи в реальних умовах використання.

3.3 Розробка бази даних

У розробленому Android-додатку для моніторингу витрат та доходів реалізація бази даних здійснена не за допомогою серверного реляційного інструменту на кшталт MySQL або PostgreSQL, а через вбудовану локальну NoSQL-базу Hive, що оптимізована для мобільних додатків на Flutter. Hive не використовує SQL-синтаксис, тому структура бази даних описується не через SQL-запити, а через Dart-класи із анотаціями для серіалізації та десеріалізації.

Для реалізації схеми бази даних були створені моделі TransactionModel, CategoryModel, BudgetModel, які містять поля, що відповідають сутностям предметної області: сума транзакції, дата, тип категорії, ID, коментар, а також логічні прапори (наприклад, isDeleted). Ці поля декларуються через анотації @HiveField(index), а самі моделі – через @HiveType(typeId).

Перед використанням моделей у Hive необхідно зареєструвати адаптери типів, які генеруються автоматично за допомогою команди flutter packages pub run build_runner build. Це дозволяє Hive серіалізувати об’єкти класів у двійковий формат для збереження у файлах. Наприклад, оголошення моделі транзакції виглядає наступним чином (ліст. 3.8).

Лістинг 3.8 – Оголошення моделі транзакції

```
@HiveType(typeId: 4)
class TransactionModel {
  @HiveField(0) final double amount;
  @HiveField(1) String? notes;
  @HiveField(2) final DateTime date;
  @HiveField(3) final CategoryType categoryType;
  @HiveField(4) final TransactionType transactionType;
  @HiveField(5) final CategoryModel categoryModel;
  @HiveField(6) String? id;
  ...
}
```

кінець лістингу 3.8

Замість SQL-запитів тут використовуються методи доступу до даних у наступному вигляді (ліст. 3.9).

Лістинг 3.9 – Методи доступу до даних

```
// Додавання запису
await transactionBox.put(transaction.id, transaction);

// Отримання всіх транзакцій
final allTransactions = transactionBox.values.toList();

// Видалення транзакції
await transactionBox.delete(transaction.id);
```

кінець лістингу 3.9

У порівнянні з реляційною базою даних, структура Hive більш гнучка та не потребує явного створення таблиць або написання DDL-операторів (CREATE TABLE, ALTER TABLE). Водночас вона ефективно працює в умовах обмеженого ресурсу мобільного пристрою, дозволяючи здійснювати зберігання та обробку фінансових даних без підключення до Інтернету.

Отже, розробка бази даних у межах даного застосунку полягала в побудові об’єктної схеми, а не традиційної реляційної. Завдяки використанню Hive забезпечується швидкий доступ до даних, підтримка типів Dart, шифрування, та масштабованість на рівні мобільного клієнта.

3.4 Розробка документації для програмного продукту

Документація до розробленого мобільного додатку призначена як для кінцевого користувача, так і для технічного спеціаліста, який відповідає за інсталяцію, обслуговування або супровід програмного продукту. Вона містить основну інформацію про вимоги до системи, етапи встановлення додатку, а також поради з його використання.

Після встановлення додатку користувач має змогу:

1) створювати категорії доходів і витрат (наприклад: зарплата, продукти, транспорт);

2) додавати транзакції із зазначенням суми, дати, коментаря та категорії;

3) переглядати статистику витрат/доходів у вигляді графіків;

4) налаштовувати щомісячний бюджет та отримувати сповіщення про перевищення ліміту;

5) переглядати залишок коштів за поточний період.

Інтерфейс інтуїтивно зрозумілий, підтримує темну та світлу тему, а також локалізацію.

Опишемо також основні кроки інсталяції та розгортання для користувача:

1) завантажити APK-файл на пристрій;

2) дозволити встановлення з невідомих джерел у налаштуваннях Android;

3) встановити додаток через файловий менеджер або термінал;

4) запустити додаток.

В результаті успішної установки має відобразитися на екрані відповідний ярлик. Також у додатку реалізовано базову інтегровану систему довідки, яка подається у вигляді окремого компонента «Help», доступного з головного меню. У подальших версіях може бути реалізовано HTML-індексацію довідки або зовнішнє підключення Markdown-документації через інтернет.

Окрім основного функціоналу, у документації передбачено детальний опис логіки взаємодії користувача з додатком, включно з прикладами типових сценаріїв використання (створення записів, зміна теми, очищення даних), а також короткі інструкції щодо оновлення програми та резервного копіювання бази. Документація містить інтегровані підказки у самій програмі, реалізовані у вигляді діалогових вікон при першому запуску кожного розділу, що робить інтерфейс інтуїтивно зрозумілим навіть для новачків.

Висновки до розділу 3

У третьому розділі було здійснено безпосередню реалізацію Android-додатку для обліку витрат і доходів, що передбачало повний цикл практичної розробки – від написання програмного коду до його тестування, налагодження та підготовки до інсталяції. Було продемонстровано ефективне використання

мови програмування Dart у середовищі Flutter, що дозволило забезпечити кросплатформену сумісність і сучасний адаптивний інтерфейс.

У межах реалізації було застосовано архітектурний підхід із чітким розмежуванням відповідальностей між шарами: представлення (UI), логіки бізнес-процесів (BLoC), доменними моделями та джерелами даних. Особливу увагу приділено роботі з локальним сховищем Hive для збереження об’єктів транзакцій, категорій, бюджетів і валют. Реалізація функціоналу охоплює додавання, редагування, видалення транзакцій, створення категорій, обчислення залишку бюджету, а також побудову графіків для фінансового аналізу.

Проведене тестування логіки застосунку засвідчило його функціональну стабільність. Було усунуто низку виявлених недоліків, серед яких – обробка порожніх значень, дублювання даних і відображення елементів у темній темі.

Описано також процес створення інсталяційного пакета APK та надано документацію з вимогами до системи, інструкціями з розгортання і користувацькою довідкою. Усі вказані компоненти свідчать про завершеність розробки інформаційної системи та її готовність до використання кінцевими користувачами.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра повністю реалізовано поставлену мету – розроблено Android-додаток для моніторингу витрат та доходів, що дозволяє користувачам зручно фіксувати фінансові транзакції, переглядати аналітичні графіки та планувати особистий бюджет. Вибір фреймворку Flutter та мови Dart був зумовлений їхньою кросплатформеністю, швидкістю розробки та підтримкою сучасного реактивного інтерфейсу. Використання Hive як локального сховища забезпечило високу продуктивність, відмовостійкість і автономність додатку без необхідності в зовнішніх базах даних.

Дослідження підтвердило високу актуальність розробки нового Android-додатку для обліку особистих фінансів, орієнтованого на українського користувача. Аналіз наукових джерел та ринку аналогічних рішень показав, що попри наявність низки мобільних застосунків, таких як Monefy, Wallet, Spendee, більшість із них мають суттєві обмеження – зокрема, відсутність локалізації, обов’язкову синхронізацію з хмарою, перевантажений інтерфейс і недоступність повноцінного функціоналу у безкоштовній версії. Виявлені недоліки чітко окреслили вимоги до нового програмного продукту: автономність, конфіденційність, простота у використанні, адаптивний інтерфейс і можливість персоналізації.

Визначено повний набір функціональних та нефункціональних вимог до майбутньої інформаційної системи, сформованих на основі аналізу аналогів, опитування цільової аудиторії та сценаріїв користування. Було обрано архітектурний підхід Clean Architecture, розроблено UML-діаграми варіантів використання, класів і послідовностей, що дозволило сформувати чітку структуру додатку. Ці вимоги й моделі забезпечили концептуальну основу для подальшої реалізації логіки та інтерфейсу додатку з урахуванням принципів модульності, масштабованості та зручності користування.

Також було обґрунтовано вибір інструментів і бібліотек для реалізації мобільного додатку, таких як Flutter, Dart, Hive, BLoC, а також додаткових засобів для анімацій, SVG-графіки й візуалізації аналітики. Вибрані технології дозволили реалізувати швидкий, адаптивний і масштабований додаток з сучасним інтерфейсом. Також було описано алгоритми обліку транзакцій, формування бюджетів, побудови графіків та локального шифрування даних. Обраний технічний стек повністю відповідає цілям проєкту й дозволяє у майбутньому розширювати функціонал без значного перероблення архітектури.

Було здійснено поетапну реалізацію функціоналу Android-додатку, починаючи з ініціалізації локального сховища Hive, налаштування маршрутизації між екранами та створення основної архітектури системи за шаблоном Clean Architecture. Запроваджено використання патернів BLoC і Cubit для управління станом інтерфейсу, що забезпечило чітке розділення відповідальності між логікою додатку, джерелами даних і візуальним представленням. У реалізації було використано засоби Dart, Flutter, Hive, SVG-графіку та кастомізовані елементи UI для побудови адаптивного, інтуїтивно зрозумілого інтерфейсу з підтримкою темної і світлої тем.

Під час тестування було перевірено коректність логіки роботи окремих компонентів, включаючи use case-и, репозиторії та джерела даних. Налагодження проводилося з використанням Flutter DevTools, що дозволило оперативно виявити та усунути ряд помилок, таких як некоректна обробка пустих полів, дублювання категорій та конфлікти SVG-графіки з темною темою. Результатом стало створення стабільного, надійного додатку з підтвердженою функціональною цілісністю, що відповідає заданим мінімальним системним вимогам для комфортного використання на більшості сучасних Android-пристроїв.

Замість класичної реляційної СУБД у додатку було реалізовано об’єктно-орієнтовану структуру зберігання даних за допомогою Hive, що надало змогу серіалізувати об’єкти Dart без використання SQL-запитів. Створені моделі з анотаціями Hive забезпечили чітке структурування даних про транзакції,

категорії, бюджети та інші сутності. Такий підхід дозволив досягти високої продуктивності, офлайн-доступності та простоти масштабування, а також забезпечив типобезпечну взаємодію з даними без зайвих накладних витрат.

У межах розробки було створено супровідну документацію, що включає системні вимоги, інструкції з інсталяції APK-файлу, опис функціоналу додатку та довідкові матеріали для користувача. Передбачено реалізацію інтегрованої довідки у вигляді окремого екрана, що містить основні пояснення з користування. Документація охоплює не лише технічну інформацію, а й аспекти зручності, локалізації та захисту, що дозволяє забезпечити легке впровадження продукту серед цільової аудиторії та його готовність до подальшого супроводу й масштабування.

Надалі перспективним є розширення можливостей додатку за рахунок додавання його у Google Play Market, синхронізації з хмарними сервісами (Firebase, Google Drive), реалізації мультикористувацького режиму, додавання штучного інтелекту для прогнозування витрат, а також інтеграції API банківських систем для автоматичного зчитування транзакцій. Також доцільною є локалізація додатку, розширення аналітичного модуля.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке особисті фінанси і як ними правильно управляти: від комуналки до донатів на перемогу. Національний університет «Львівська політехніка». URL: https://lpnu.ua/news/shcho-take-osobysti-finansy-i-iak-nymy-pravylno-upravliaty-vid-komunalky-do-donativ-na (дата звернення: 27.04.2025).

2. Башовий В. М., Стаценко В. В., Стаценко Д. В. Розробка адаптивного web-інтерфейсу для додатку керування особистими фінансами. Technologies and Engineering. 2023. № 5. С. 9-16. URL: https://doi.org/10.30857/2786-5371.2022.5.1 (дата звернення: 27.04.2025).

3. Черненко О., Лисенко Д., Карабаш В. Створення мобільного додатку для ефективного відстеження та управління особистими фінансами. Науковий вісник Полтавського університету економіки і торгівлі. Серія «Технічні науки», 2024. № 3. С. 50-55. URL: https://doi.org/10.37734/2518-7171-2024-3-9 (дата звернення: 27.04.2025).

4. Jasmin P. AstroWealth: Personal Finance Management Web Application. Applied Information Technology And Computer Science. 2024. Vol. 5, №2. P. 1180-1199. URL: https://doi.org/10.30880/aitcs.2024.05.02.062 (дата звернення: 27.04.2025).

5. Personal Finance Management Application / T. Stefanov et al. TEM Journal. 2024. P. 2065-2075. URL: https://doi.org/10.18421/tem133-34 (date of access: 27.04.2025).

6. Monefy – Handy personal finance management tool for iOS and Android. URL: https://monefy.com/ (date of access: 01.05.2025).

7. Money Manager. URL: https://moneymanager.com.ua/ (date of access: 01.05.2025).

8. Wallet by BudgetBakers. Wallet. URL: https://web.budgetbakers.com/ (date of access: 01.05.2025).

9. Money Manager & Budget Planner – Spendee. URL: https://www.spendee.com/ (date of access: 01.05.2025).

10. The Clean Architecture. Clean Coder Blog. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (date of access: 03.05.2025).

11. Flutter – Build apps for any screen. URL: https://flutter.dev/ (date of access: 10.05.2025).

12. Bailey T., Biessek A. Flutter for Beginners. An introductory guide to building cross-platform mobile applications with Flutter 2.5 and Dart. Packt Publishing Ltd. 2021. 370 p.

13. Bailey T., Biessek A. Flutter for Beginners. Cross-platform mobile development from Hello, World! To app release with Flutter 3.10+ and Dart 3. x. Packt Publishing Ltd. 2023. 406 p.

14. Dart programming language. URL: https://dart.dev/ (date of access: 13.05.2025).

15. Meiller D. Modern App Development with Dart and Flutter 2. A Comprehensive Introduction to Flutter. Walter de Gruyter GmbH & Co KG. 2021. 249 p.

16. Apache Hive. URL: https://hive.apache.org/ (date of access: 13.05.2025).

17. Hive Tutorial – Learn Apache Hive for Big Data. URL: https://www.tutorialspoint.com/hive/index.htm (date of access: 17.05.2025).

18. Bloc Concepts. Bloc. URL: https://bloclibrary.dev/bloc-concepts/ (date of access: 17.05.2025).