НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри інформаційних систем і технологій**
_____ **Ігор БОЛБОТ** (підпис)	_____ **Михайло ШВИДЕНКО** (підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку обліку особистих фінансів з використанням інтелектуального асистента»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи Доцент, канд. пед. наук	_____ (підпис)	**Тетяна ВОЛОШИНА**
Виконав	_____ (підпис)	**Володимир БОЯРІНОВ**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і технологій

к.е.н., доцент _____ Михайло ШВИДЕНКО
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Боярінову Володимиру Івановичу
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Розробка мобільного застосунку обліку особистих фінансів з використанням інтелектуального асистента»

затверджена наказом від «6» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «25 травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Способи обліку особистих фінансів, наявні засоби роботи інтелектуального асистента.

Перелік питань, що підлягають дослідженню:

- Аналіз процесів обліку особистих фінансів та вимог до системи обліку.
- Проєктування та розробка інформаційного забезпечення системи (моделі даних, архітектури, інтеграції інтелектуального асистента).
- Впровадження інформаційної системи для загального доступу

Перелік графічних документів (за необхідності).
Дата видачі завдання «3» лютого 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Тетяна ВОЛОШИНА**

Завдання взяв до виконання _____ (підпис) **Володимир БОЯРІНОВ**

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ – штучний інтелект

API – інтерфейс прикладного програмування

App Store – онлайн магазин застосунків компанії Apple

CRUD – базові операції створення, читання, оновлення та видалення даних

DFD – діаграма потоків даних

ER – модель «сутність-зв’язок»

FDD – функціональна декомпозиція системи

GDPR – загальний регламент захисту даних ЄС

iOS – мобільна операційна система Apple

JSON – текстовий формат структурованих даних

LLM – велика мовна модель

MV – архітектурний підхід розділення інтерфейсу та даних

MVP – мінімально життєздатний продукт

MVVM – архітектурний підхід з проміжним шаром для зв’язку

ORM – підхід відображення об’єктної моделі

SQL – мова структурованих запитів до даних

SwiftData – фреймворк локального зберігання даних в екосистемі Apple

SwiftUI – декларативний фреймворк побудови інтерфейсу користувача

UML – уніфікована мова моделювання

UI – користувацький інтерфейс

UX – досвід взаємодії користувача з інтерфейсом

UUID – універсальний унікальний ідентифікатор

ВСТУП

У сучасних соціально-економічних умовах питання управління особистими фінансами набуває не лише побутового, а й практичного значення. Зростання вартості життя, інфляція, збільшення кількості безготівкових платежів, поява регулярних цифрових підписок та розрахунок у різних валютах формують середовище, у якому людина щоденно приймає фінансові рішення, але не завжди має цілісне уявлення про накопичувальний ефект. Навіть за наявності банківських застосунків користувач часто не бачить повної картини особистих фінансів, оскільки витрати можуть бути розподілені між картками, готівкою та рахунками [1].

Традиційні способи ведення фінансового обліку, зокрема паперові запити, нотатки або електронні таблиці, залишаються корисними як базовий інструмент дисципліни, проте не забезпечують достатньої швидкості, аналітичної глибини, контекстного пояснення та зручності у щоденному використанні. У більшості випадків саме трудомісткість внесення даних є головною причиною відмови користувача від регулярного обліку особистих фінансів. Якщо для додавання однієї транзакції потрібно здійснювати багато дій, відкривати кілька форм або самостійно будувати підсумки, система втрачає практичну цінність. Це створює потребу у новому поколінні персональних фінансових застосунків, які мають бути швидкими, зрозумілими, аналітично корисними та близькими до реальної поведінки користувача [1, 2].

Особливого значення набуває мобільний формат реалізації такої системи, оскільки смартфон є постійно доступним пристроєм, що супроводжує користувача протягом дня та дозволяє фіксувати фінансові події безпосередньо в момент їх виникнення. Саме це дає змогу мінімізувати втрати інформації, пов’язані з відкладеним введенням витрат і доходів.

Важливим технологічним чинником є розвиток інтелектуальних сервісів, і якщо раніше користувачу потрібно було формально описувати кожну фінансову операцію окремими полями, то сьогодні з'являється можливість делегувати інтерпретацію даних ШІ-асистенту. Наприклад, фраза «Купив каву за 120 грн»

може бути перетворена на структуровану транзакцію з сумою, типом операції, датою та категорією. Це не усуває потребу у перевірці та контролі, проте значно зменшує бар’єр входу до обліку фінансів і робить його швидшим для користувача [3].

Попри наявність різних програмних продуктів для обліку особистих фінансів, аналіз ринку показує, що частина популярних рішень орієнтована на хмарну інфраструктуру, інтеграцію з банками або специфічні підходи для ведення бюджетів, які не завжди швидкі і зручні для використання для широкої аудиторії. Додатковими обмеженнями залишаються також відсутність повноцінної української локалізації, обмежена підтримка локальних фінансових реалій та недостатня прозорість щодо зберігання фінансових даних. Це актуалізує розробку локально орієнтованого мобільного застосунку, у центрі якого потрібні швидкість, зручність та конфіденційність.

Об’єктом дослідження є процеси персонального фінансового обліку в мобільному цифровому середовищі. Предметом дослідження є моделі даних, програмні компоненти, архітектурні підходи та методи реалізації мобільної інформаційної системи фінансового обліку.

У рамках проведеного дослідження було розглянуто та проаналізовано цикл розробки інформаційної системи для обліку особистих фінансів з використанням інтелектуального асистента. Послідовність розробки включає в себе аналіз предметної області, формування структури даних, програмну реалізацію, тестування та впровадження інформаційної системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми та проблематика обліку особистих фінансів

Предметною областю досліджень є облік особистих фінансів, тобто сукупність процесів, пов'язаних із фіксацією доходів, витрат, плануванням бюджетів, контролем регулярних платежів, аналізом фінансових звичок та оцінкою фінансового стану користувача. На відміну від бухгалтерського обліку підприємств, персональний фінансовий облік має менший рівень формальності, але вищу залежність від зручності повсякденного використання. Це означає, що ефективність системи визначається не тільки повнотою функціоналу, а й тим, наскільки швидко і зручно людина може додати запис, переглянути результат і зрозуміти його зміст.

У повсякденному житті люди одночасно стикаються з різними типами фінансових подій, такими як регулярні доходи, випадкові надходження, обов'язкові витрати, дрібні імпульсивні покупки, цифрові підписки, боргові зобов'язання і платежі у різних валютах. Усі ці події формують фінансовий профіль людини, але без систематизації залишаються набором фактів. Саме через це люди часто не можуть дати точну відповідь на питання, скільки вони витратили за місяць на харчування, скільки коштів витрачається на заклади харчування і чому залишок коштів не збігається з очікуваним.

Найпростішими формами персонального обліку фінансів залишаються паперові записи, довільні нотатки та електронні таблиці. Їх перевагою є доступність та гнучкість, проте вони мають низку суттєвих недоліків: відсутність швидкого і структурованого введення, складність пошуку й фільтрації даних, відсутність автоматичного підрахунку, складність роботи з різними періодами та слабку придатність для мобільного використання. Для багатьох користувачів

саме необхідність вручну підтримувати структуру таблиць та формул стає причиною припинення ведення обліку.

Додаткову складність створює фрагментарність сучасного фінансового середовища. Частина витрат відображається в банківських застосунках, частина здійснюється готівкою. Банківський застосунок, навіть якщо він надає історію платежів, розбитих на категоріях, не вирішує проблему повністю, оскільки не завжди дозволяє легко змінювати категорії, працювати з кількома рахунками одночасно, поєднувати планування а також враховувати нефінансовий контекст конкретної покупки.

Однією з найважливіших практичних проблем є низька дисципліна довготривалого використання систем фінансового обліку. Навіть технічно функціональний застосунок може виявитися неуспішним, якщо він вимагає від користувача надмірної кількості дій для базових операцій. Тому критичними стають наступні принципи:

- мінімальна кількість кроків для внесення даних;
- збереження автономності базових функцій без інтернет з’єднання;
- поділ витрат і доходів за категоріями;
- прозоре відображення бюджетів і регулярних платежів;
- довіра користувача до способу зберігання даних та відчуття контролю над ними.

Запровадження інформаційної системи не лише спрощує керування фінансами, а й формує новий рівень взаємодії з грошима. Це підвищує контроль, додає прозорості та відповідає сучасним вимогам безпеки та зручності.

1.2 Аналіз існуючих програмних рішень у сфері персональний фінансів

Найпростіші підходи, такі як використання записів на папері або ведення електронних таблиць, приваблюють користувача повним контролем над структурою даних і відсутністю залежності від стороннього програмного

забезпечення. Проте їх слабкість полягає у високому порозі підтримки, так як користувач має самостійно проєктувати структуру, категорії, підтримувати правильність формул, формувати підсумки та нести повну відповідальність за цілісність і точність даних. У мобільному сценарії цей підхід особливо незручний, бо він не пристосований до швидкого введення операцій.

Другий поширений клас рішень представлений банківськими мобільними застосунками. Їх перевага полягає в тому, що історія операцій вже доступна у застосунку без додаткового ручного введення. Однак, ці системи орієнтовані насамперед на банківський сервіс, а не на комплексне персональне фінансове планування. У них часто відсутні гнучкі бюджети, немає окремого механізму для одночасного обліку декількох рахунків або готівки, а аналітика залишається прив’язаною до конкретного банку.

Серед спеціалізованих міжнародних рішень особливу увагу привертає застосунок YNAB [4]. Цей продукт відомий акцентом на дисциплінованому бюджетуванні, розвиненими інструментами планування, синхронізацією між пристроями та достатньо глибокою аналітикою. YNAB є сильним прикладом системи, орієнтованої на серйозне ставлення до власних фінансів. Водночас для частини користувачів його методологія є надто формалізованою, а процес входження в продукт потребує навчання. Крім того, YNAB орієнтований на хмарну інфраструктуру, що не завжди відповідає хвилюванню користувачів щодо безпеки фінансових даних та автономному сценарію використання.

Іншим показовим прикладом є застосунок EveryDollar [5]. Його позиціонування побудоване навколо простоти використання, базового бюджетування та доступного інтерфейсу. Сильними сторонами інформаційної системи є зрозумілий UX і швидкий старт для нових користувачів. Однак цей продукт також орієнтований на хмарну інфраструктуру та підключення банків для автоматичного введення операцій, що, своєю чергою, не завжди може гарантувати безпеку фінансових даних та можливість користуватися застосунком, коли відсутній доступ до інтернету.

Окрім наведених рішень, варто розглянути ще два популярні мобільні застосунки для обліку фінансів, такі як Money Lover [6] та Wallet by BudgetBakers [7].

Застосунок Money Lover орієнтований на щоденний облік витрат із фокусом на швидкість введення та простоту використання. Він пропонує зручну систему категорій, базову аналітику та підтримку планування бюджету. Його перевагою є інтуїтивний інтерфейс і низький поріг входу для нових користувачів. Водночас значна частина функціональності пов’язана з хмарною синхронізацією, що знижує автономність роботи, а розширені можливості часто доступні лише у платній версії.

Застосунок Wallet by BudgetBakers пропонує більш розвинену аналітику та підтримку кількох рахунків, включаючи інтеграцію з банківськими сервісами. Сильними сторонами є деталізовані звіти, візуалізація фінансових потоків та інструменти бюджетування. Проте залежність від інтернет-з’єднання та зовнішніх сервісів обмежує можливість повноцінного використання в офлайн-режимі та створює додаткові питання щодо конфіденційності даних.

У контексті постановки задачі важливо не просто перелічити аналоги, а й співвідносити їх із розроблюваною інформаційною системою за ключовими критеріями.

Таблиця 1.1

Порівняння інструментів для обліку фінансів

	Таблиці / нотатки	YNAB	EveryDollar	Money Lover	Wallet by BudgetBakers	Розроблюваний застосунок
Швидкість введення	Низька	Середня	Висока	Висока	Середня	Висока
Автономність	Висока	Низька	Низька	Низька	Низька	Висока
Українська локалізація	Є	Відсутня	Відсутня	Часткова	Часткова	Є
Облік підписок	Є	Є частково	Відсутній	Є	Є	Є
ШІ-асистент	Відсутній	Відсутній	Відсутній	Відсутній	Відсутній	Є
Конфіденційність даних	Високий	Низький	Низький	Низький	Низький	Високий
Поріг входу	Середній	Високий	Низький	Низький	Низький	Низький

З наведеного порівняння видно, що жодне з існуючих рішень не поєднує одночасно важливі властивості: висока швидкість введення даних, висока автономність у використання, наявність української локалізації, зрозумілий облік підписок, ШІ-асистент, високу конфіденційність даних та низький поріг входу.

1.3 Визначення цілей, завдань і меж дослідження

З огляду на виявлені проблеми предметної області та результати аналізу наявних рішень, основною метод дослідження є створення цілісного мобільного застосунку для обліку особистих фінансів, який забезпечує швидку фіксацію фінансових операцій, структуроване подання даних, зручну аналітику, має додатковий функціонал у вигляді ШІ-асистента, забезпечує конфіденційність фінансових даних і автономність роботи без доступу до інтернету.

Для досягнення цієї мети система повинна вирішувати не одну, а відразу кілька взаємопов’язаних задач. По-перше, потрібно забезпечити стабільне локальне ядро, яке працює без підключення до мережі і забезпечує захист інформації. По-друге, потрібна модель даних, яка відображає не лише транзакції, а й структуру фінансового середовища користувача: рахунки, категорії, бюджети та підписки. По-третє, система має надавати не звичайну виписку операцій, а просту і зрозумілу аналітику, яка допомагає приймати рішення. По-четверте, важливо інтегрувати інтелектуальний інтерфейс, який спростить введення й пояснення даних без руйнування локальної архітектури.

Межі дослідження також необхідно визначити достатньо чітко. Предметом реалізації є локально орієнтований мобільний застосунок для операційної системи iOS. Тому такі можливості, як повноцінна хмарна синхронізація, інтеграція з банківськими API, сімейні акаунти, веб-версія або корпоративне адміністрування, не включаються до базової реалізації і розглядаються як перспективи розвитку інформаційної системи.

Водночас дослідження не обмежується лише створенням набору екранів або структур даних. Система розглядається як інструмент підтримки щоденної фінансової поведінки користувача. Тому важливими є не тільки технічні параметри, а й питання користувацького досвіду, логіка навігації, наочність аналітики та довіра до обробки фінансової інформації.

1.4 Формування вимог до мобільного застосунку для обліку особистих фінансів з використанням інтелектуального асистента

Формування вимог є одним із найважливіших етапів при створенні інформаційної системи, оскільки саме на цьому рівні фіксуються функції, обмеження, очікувані якісні характеристики та технологічні рамки продукту. Для інформаційної системи, вимоги доцільно поділити на функціональні, нефункціональні та загальні вимоги до технологій.

Функціональні вимоги визначають, які саме дії має виконувати інформаційна система та які можливості повинні бути надані користувачу. Інформаційна система призначена для обліку власних фінансів, аналізу витрат та доходів, формування бюджетів і надання простої аналітики.

Функціональні вимоги охоплюю такі оснівні процеси:

- створення транзакцій двох типів: витрати та доходи;
- збереження в транзакції суми, категорії, дати, опису та прив’язку до гаманця;
- редагування та видалення транзакцій;
- автоматичне оновлення балансу гаманця після створення, редагування або видалення гаманця;
- пошук і фільтрація транзакцій за датою, категорією та сумою;
- створення користувацьких категорій витрат і доходів;
- редагування назви, іконки та кольору категорії;
- створення додаткових гаманців;
- редагування та видалення гаманців;

- встановлення бюджетних обмежень;
- відображення попередження при перевищенні встановленого бюджету;
- надання візуальної аналітики;
- підтримка творення транзакцій у різних валютах з автоматичною конвертацією;
- створення транзакцій на основі текстового введення користувача;
- робота ШІ-асистента для відповідей на фінансові запити користувача.

Нефункціональні вимоги визначають якісні характеристики системи, обмеження, технологічні параметри та інтеграційні можливості. Вони описують, як система повинна працювати, а не що вона повинна робити. Нефункціональні вимоги включають в себе:

- забезпечення швидкої роботи інтерфейсу без помітних затримок при виконанні операцій;
- підтримка автономної роботи застосунку без доступу до інтернету для основних функцій;
- гарантування надійного збереження даних між сесіями та обробки збійних сценаріїв без втрати інформації;
- мінімізація передачі користувацьких даних до зовнішніх сервісів;
- виконання основної бізнес-логіки та зберігання даних локально на пристрої;
- можливість подальшого масштабування завдяки модульній архітектурі та розширення функціоналу системи;
- інтуїтивно зрозумілий інтерфейс без потреби у тривалому навчанні користувача;
- оптимізація кількості дій у ключових сценаріях використання;
- відповідність інтерфейсу базовим вимогам контрастності та читабельності;
- підтримка української та англійської мов.

Для розробки інформаційної системи потрібно використовувати сучасні та підтримувані інструменти, що відповідають вимогам стабільності та довгострокової експлуатації. Застосовані технології повинні мати термін підтримки не менше ніж через шість місяців після впровадження системи, що гарантує можливість стабільного користування та подальшого оновлення системи.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Вибір архітектурного підходу та технологічного стеку

Під час проєктування системи було розглянуто кілька можливих архітектурних підходів: класичну клієнт-серверну модель з виділеним бекендом, гібридну модель з локальним кешем і серверною синхронізацією та локально орієнтовану мобільну архітектуру з вибірковим використанням зовнішніх сервісів. З урахуванням теми роботи, джерельних матеріалів і вимог до МВП-фази, найбільш доцільним визнано третій варіант.

Клієнт-серверний підхід має очевидні переваги з точки зору синхронізації між пристроями, централізованого резервування і адміністрування, проте для цієї предметної області він створює низьку проблему. По-перше, ускладнюється стартова архітектура, з'являються окремі серверні сервіси, схеми авторизації, моделі API, питання хостингу, резервування, міграції і підтримки інфраструктури. По-друге, базові сценарії використання стають залежними від мережі. По-третє, зростають ризики, пов'язані з конфіденційністю фінансових даних. Для MVP рішення такий рівень складності не є виправданим.

Гібридна модель із локальним кешуванням і серверною синхронізацією виглядає більш збалансованою. Проте для неї також необхідно спроєктувати правила узгодження конфліктів, механізм відкладеної синхронізації, облік версії записів та відновлення після розривів з мережею. На ранньому етапі це створює значне архітектурне навантаження і також не є виправданим для MVP-рішення.

Локально орієнтована архітектура навпаки оптимально відповідає природі щоденних фінансових сценаріїв. Вона дозволяє зберігати транзакції, категорії, бюджети, підписки та більшість аналітики безпосередньо на пристрої користувача. Такий підхід забезпечує швидкий відгук інтерфейсу, автономність, простіший контроль даних і нижчий період реалізації. При цьому зовнішні сервіси не усуваються повністю, а використовуються там, де вони справді дають цінність: для оновлення валютних курсів і роботи ШІ-асистента.

Вибір локальної моделі підтверджується пріоритетними джерелами другого набору, де послідовно фіксується використання SwiftData, iOS-пристрою як основного вузла розгортання, а також пакетних структур з локальним доменним ядром. Саме тому для цієї інформаційної системи локально орієнтована архітектура приймається як фінальна, а серверно-центровані елементи різних матеріалів розглядаються лише як потенційний напрям майбутнього розширення [8-10, 11-14].

З урахуванням обраної локально орієнтованої архітектури особлива увага приділялася підбору технологій, які максимально ефективно працюють у межах нативного середовища та не створюють зайву інфраструктурну складність. Основний акцент зроблений на інструментах, що дозволяють реалізувати швидкий, автономний і стабільний користувацький досвід без необхідності постійної взаємодії з серверною чистою.

Обраний стек формувався з урахуванням вимог до продуктивності, безпеки та зручності подальшої підтримки. Використання нативної мови програмування та фреймворків екосистеми забезпечує глибоку інтеграцію з платформою, оптимізоване використання ресурсів пристрою та передбачувану поведінку інтерфейсу. Декларативний підхід до побудови UI спрощує розробку складних екранів і дозволяє швидше адаптувати інтерфейс до змін бізнес-логіки. Локальне сховище даних у свою чергу виступає ключовим моментом, що забезпечує автономні системи та швидкий доступ до інформації без мережевих затримок.

Зовнішні сервіси інтегруються точково і виконують допоміжну роль. Вони використовуються лише у тих сценаріях, де локальна реалізація є недоцільною або обмеженою, зокрема, для отримання актуальних валютних курсів та реалізації інтелектуальних функцій асистента. Такий підхід дозволяє зберегти простоту основної архітектури, не обмежуючи функціональні можливості системи.

У результаті сформовано збалансований технологічний фундамент, який підтримує поточні потреби системи та водночас залишає можливість для подальшого розвитку і без необхідності кардинальної перебудови.

Обраний технологічний стєк має такий вигляд:

- Swift як основна мова мого програмування придатна для безпечної реалізації бізнес-логіки та доменних моделей;
- SwiftUI як декларативний спосіб побудови сучасного користувацького інтерфейсу;
- SwiftData як локальне сховище моделей;
- XCode, як інтегроване середовище розробки, тестування і збірки;
- Figma як інструмент попереднього UI/UX-проєктування;
- OpenAI API для реалізації інтелектуального асистента;
- ExchangeRate-API для отримання актуального обмінного курсу.
- Архітектура системи повинна відповідати низці ключових вимог:
- забезпечувати низьку зв'язаність між бізнес-логікою, інтерфейсом та інфраструктурою;
- підтримувати локальну працездатність у базових сценаріях без доступу до мережі;
- передбачати можливість подальшого розширення без кардинальної перебудови ;
- мінімізувати дублювання логіки між екранами;
- забезпечувати прозору інкапсуляцію доступу до зовнішніх сервісів.

Вибраний архітектурний підхід поєднує локальне ядро обробки даних із вибірковими зовнішніми інтеграціями. Саме він найкраще відповідає поставленій задачі розробки мобільного застосунку для обліку особистих фінансів з використанням інтелектуального асистента у форматі реалістичного, технічно цілісного і практично корисного MVP-продукту.

2.2 Логічна і фізична модель даних

Логічна модель даних є центральним елементом проєктування, оскільки саме вона визначає, як предметна область відображається у структурі програмних сутностей, зв’язків і правил цілісності. Для розроблюваної системи модель даних має бути водночас достатньо простою для локального використання та достатньо виразною, щоб підтримати бюджети, підписки, багатовалютність і аналітику.

Сутність Wallet (гаманець) є контейнером фінансового контексту користувача. Вона містить ідентифікатор, назву, колір, іконку та базову валюту. Наявність окремої сутності гаманець дозволяє розділяти фінансові простори, наприклад «Основна карта», «Готівка», «Резерв», «Подорож». Це підвищує гнучкість системи та дає змогу будувати аналітику не тільки за категоріями, а й за середовищем використання коштів.

Сутність Category (категорія) призначена для структуризації транзакцій. Вона має назву, колір, іконку, тип (дохід або витрата) та посилання на гаманець. Прив’язка категорій до гаманця дає можливість підтримувати різний набір категорій у різних фінансових просторах, якщо це необхідно користувачеві. Категорія також може бути пов’язана з бюджетом і множиною транзакцій.

Сутність Budget (бюджет) являє собою ліміт витрат для певної категорії. У базовій моделі бюджет зберігає ідентифікатор, посилання на категорію та суму ліміту. На рівні MVP цього достатньо для побудови поточного контролю використання бюджету. За потреби модель може бути розширена періодом дії, правилами повторення та історією змін.

Сутність Subscription (підписка) описує регулярну витрату. Вона включає назву, періодичність, суму, валюту, іконку, дату початку, стан і посилання на гаманець. Відокремлення підписок від звичайних транзакцій виправдане тим, що вони мають прогнозний характер і важливі для контролю майбутнього бюджетного навантаження.

Сутність Transaction (транзакція) є основним об’єктом системи. Вона зберігає ідентифікатор, посилання на категорію, суму у внутрішньому форматі, оригінальну суму, валюту, коментар і дату. Подвійне зберігання суми є важливим елементом багато валютної логіки оскільки «transactionOriginalAmount» відображає реальне значення, введене користувачем, а «transactionAmount» використовується як нормалізована база для агрегування.

- Логічні зв’язки між сутностями мають такий характер:
- Один гаманець містить багато категорій;
- один гаманець містить багато підписок;
- одна категорія містить багато транзакцій;
- одна категорія може мати один активний бюджет;
- одна транзакція належить одній категорії;
- одна підписка належить одному гаманцю.

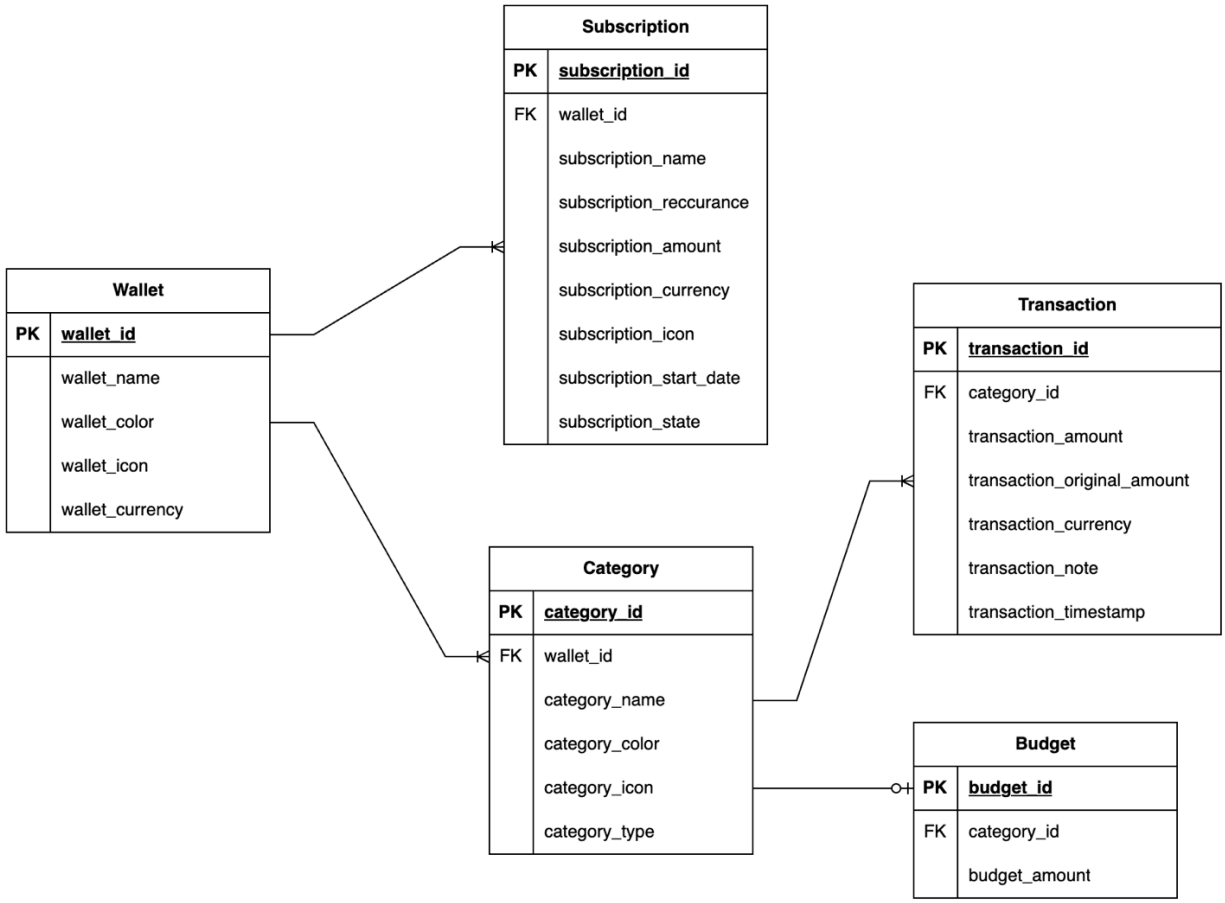

Рис. 2.1 — ER Діаграма інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Для аналітичних сценаріїв логічна модель також є достатньо зручною. Усі потрібні агрегати можуть бути побудовані на основі транзакцій, категорій, бюджетів і часових фільтрів. Завдяки цьому немає потреби дублювати аналітичні таблиці у сховищі. Звіти формуються динамічно на основі локальних даних, що зменшує ризик розсинхронізації.

2.3. Моделювання бізнес-процесів та сценаріїв взаємодії

Проєктування інформаційної системи обліку власних фінансів з використанням інтелектуального асистента передбачає не лише вибір технологій чи формування архітектурної моделі. Значно важливішим є розуміння того, як система поводитиметься у реальних сценаріях використання: під час щоденного ведення витрат, аналізу фінансів або взаємодії з інтелектуальним асистентом. Щоб уникнути абстрактних уявлень, у роботі було застосовано UML як інструмент візуального опису системи.

Використання UML дозволило не просто зафіксувати структуру майбутнього рішення, а фактично описати ключові сценарії ще до етапу розробки. Через діаграми було відображено, як користувач додає транзакції, як система обробляє дані, як формується аналітика та яким чином інтелектуальний асистент взаємодіє з фінансовою інформацією. Такий підхід дав можливість заздалегідь виявити потенційні слабкі місця, продумати поведінку системи у нестандартних ситуаціях і уникнути логічних розривів між вимогами та їх реалізацією.

Особливу цінність UML-моделювання продемонструвало у складних частинах системи, де поєднуються кілька рівнів логіки: обробка фінансових операцій, інтеграція зовнішніх сервісів (наприклад, курсів валют), а також робота інтелектуального асистента. Завдяки цьому вдалося структурувати взаємодію між компонентами та забезпечити узгодженість усіх процесів.

Далі в роботі наведено набір UML-діаграм, створених у процесі проєктування: діаграми прецедентів, діяльності, послідовностей і розгортання. Кожна з них висвітлює систему з окремого ракурсу, від поведінки користувача до внутрішньої організації компонентів. У комплексі ці моделі формують цілісне уявлення про систему як про узгоджену структуру, де всі елементи взаємодіють між собою в межах єдиної логіки.

2.3.1 Діаграма прецедентів

Під час проєктування інформаційної системи основний акцент було зроблено не на розподілі користувачів за ролями, а на структуризації функціональних можливостей навколо єдиного користувача. Такий підхід дозволив зосередитися на реальних сценаріях використання системи без зайвого ускладнення моделі доступу.

На діаграмі змодельована базова взаємодія користувача з ключовими функціями системи. Користувач має доступ до управління категоріями, що дозволяє структурувати витрати та доходи, а також до управління гаманцями для розділення фінансів за рахунками або джерелами. Окремо виділено управління бюджетами, яке відповідає за планування та контроль витрат, і управління транзакціями, що є центральним елементом системи, через який проходить основний облік фінансових операцій. Додатково передбачено можливість керування підписками та перегляду аналітичних звітів, що забезпечує користувачу повний контроль над фінансовим станом (рис. 2.2).

Рис.2.2 — Діаграма прецедентів інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Особливістю даної моделі є розширення базового сценарію управління транзакціями за рахунок інтеграції зовнішніх сервісів. Зокрема, отримання курсів валют реалізовано як додатковий сценарій, що розширює обробку транзакцій у випадках роботи з різними валютами. Це дозволяє системі автоматично враховувати актуальні курси при розрахунках.

Ще одним розширенням є створення транзакцій за допомогою інтелектуального асистента. У цьому випадку користувач може взаємодіяти з системою через природну мову, а обробка запиту делегується зовнішньому ШІ-сервісу. Це спрощує введення даних і зменшує кількість ручних дій.

Зовнішні актори, такі як сервіс курсів валют і API інтелектуального асистента, не є самостійними користувачами системи, але відіграють важливу роль у розширенні її можливостей. Вони підключаються лише в окремих сценаріях і не впливають на базову логіку роботи.

У результаті така структура діаграми дозволяє чітко відобразити ядро системи та показати, як додаткові сервіси підсилюють основний функціонал. Це

робить модель зрозумілою, не перевантаженою і придатною для подальшої реалізації логіки взаємодії та інтеграцій.

2.3.2 Діаграма діяльності

Для детального відображення дій користувача під час додавання фінансової операції було побудовано діаграму діяльності, яка демонструє послідовність створення транзакції в інформаційній системі обліку власних фінансів з використанням інтелектуального асистента. У графічному вигляді показано логіку взаємодії користувача із системою: від моменту ініціації створення запису до його збереження та оновлення інтерфейсу.

Процес розпочинається з ініціації створення транзакції. Далі користувач обирає тип операції (наприклад, дохід або витрата) та вводить суму. На цьому етапі система виконує перевірку коректності введених даних. Якщо сума є некоректною (наприклад, від’ємною або порожньою), користувач отримує повідомлення про помилку та повертається до етапу введення даних. У випадку успішної перевірки процес переходить до наступного блоку дій (рис. 2.3).

Після перевірки суми відбувається розгалуження процесу на кілька паралельних або умовних дій. Зокрема, система визначає, чи необхідно обрати валюту: якщо так, користувач виконує відповідну дію. Аналогічно перевіряється потреба у додаванні опису транзакції: за бажанням користувач може вказати додаткову інформацію. Обов’язковим кроком є вибір категорії, що дозволяє систематизувати фінансові записи.

Після виконання всіх необхідних дій відбувається їх синхронізація, і користувач переходить до підтвердження створення транзакції. Завершальний етап включає збереження даних у системі та оновлення інтерфейсу, що дозволяє миттєво відобразити нову транзакцію у списку операцій.

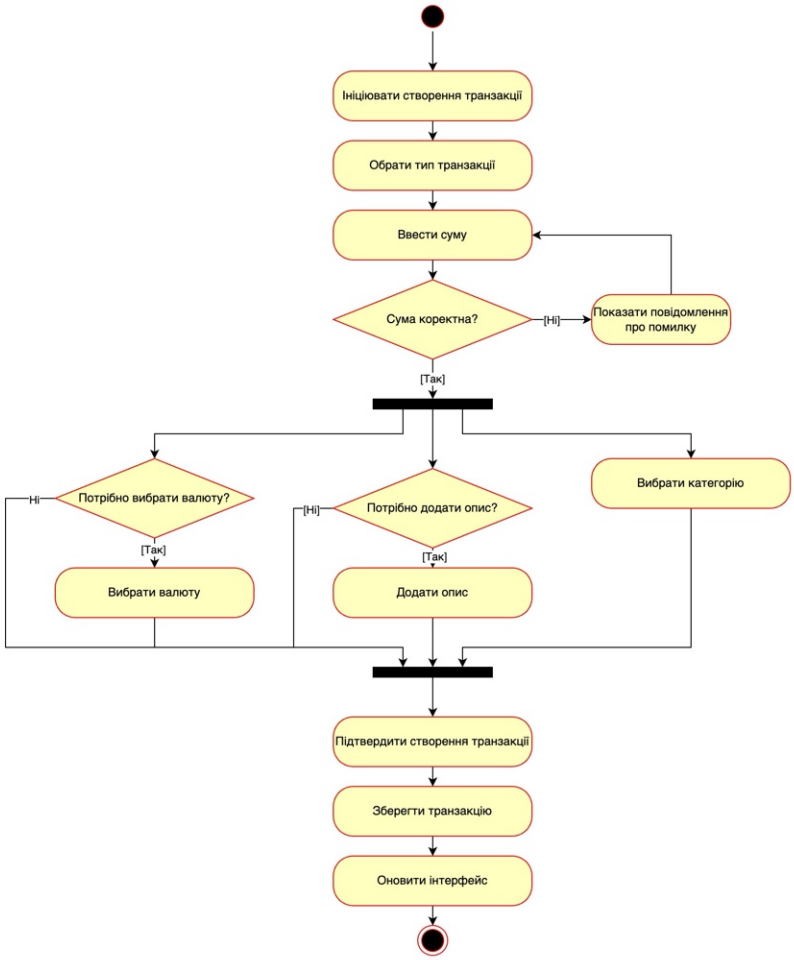

Рис. 2.3 — Діаграма діяльності зі створення транзакції в інформаційній системі обліку власних фінансів з використанням інтелектуального асистента.

Таким чином, діаграма діяльності відображає повний життєвий цикл створення транзакції: від введення даних до їх фіксації в системі. Вона підкреслює важливість перевірки, гнучкості введення інформації та послідовної обробки дій користувача, що забезпечує надійність і зручність використання системи.

2.3.3 Діаграма послідовності

Для відображення внутрішньої логіки взаємодії між компонентами системи під час створення фінансової операції було побудовано діаграму послідовності. Вона демонструє, як саме відбувається обробка запиту на створення транзакції в інформаційній системі: від дії користувача до збереження даних і оновлення інтерфейсу.

Процес ініціюється користувачем, який взаємодіє з формою створення транзакції. Інтерфейс передає запит до відповідного сервісного шару, де починається обробка бізнес-логіки. Першим кроком є перевірка коректності обраного гаманця. Для цього викликається відповідний сервіс, який повертає результат перевірки. Далі відбувається перевірка категорії через окремий сервіс, що забезпечує правильну класифікацію фінансової операції.

Після цього система звертається до сервісу курсів валют для отримання актуальних даних. Запит передається до зовнішнього API, яке повертає актуальний курс. На основі отриманої інформації виконується конвертація суми, якщо транзакція здійснюється в іншій валюті. Таким чином забезпечується точність фінансових розрахунків незалежно від валюти операції.

Після завершення всіх перевірок і обчислень формується запит на створення транзакції, який передається до шару зберігання даних. Дані записуються у локальне сховище, після чого система повертає підтвердження про успішне створення операції. У відповідь інтерфейс отримує оновлені дані та відображає їх користувачу (рис. 2.4).

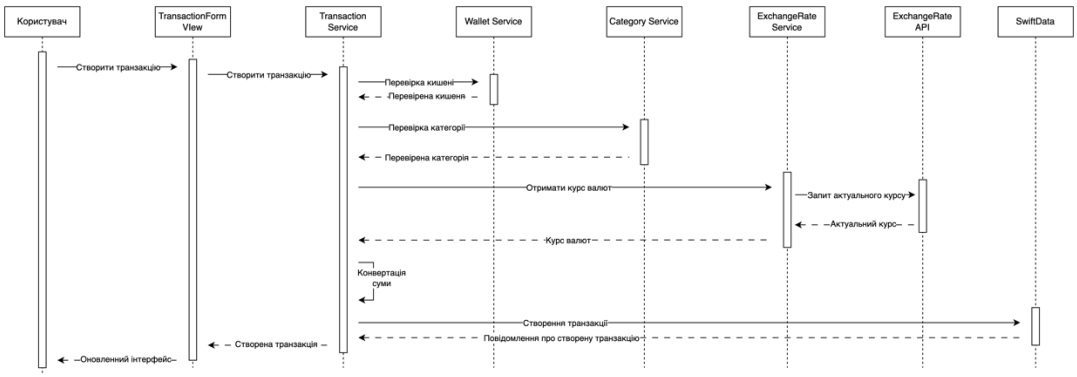

Рис. 2.4 — Діаграма послідовності створення транзакції інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Діаграма послідовності наочно показує не лише порядок викликів між компонентами, а й розподіл відповідальності між ними: інтерфейс відповідає за взаємодію з користувачем, сервіси відповідають за бізнес-логіку, а зовнішні API за надання додаткових даних. Така структура дозволяє зробити систему масштабованою, зрозумілою та легкою для підтримки, оскільки кожен компонент виконує чітко визначену функцію.

2.3.4 Діаграма пакетів

Пакетна архітектура визначає високорівневий поділ системи на логічні області відповідальності. Для розроблюваної системи доцільно використовувати модель, зафіксовану в джерелах другого пріоритету: «Presentation», «Application», «Domain», «Data Access», «Infrastructure». Такий поділ добре узгоджується з принципами чистої архітектури та дозволяє розмежувати користувацький інтерфейс, бізнес-логіку, предметну модель, механізми доступу до даних і технічні інтеграції.

Пакет Presentation відповідає за взаємодію з користувачем. До нього входять екрани головної панелі, історії транзакцій, категорій, гаманців, бюджетів, налаштувань та ШІ-асистента. Завдання цього пакета полягає у відображенні даних та перетворенні дій користувача на зрозумілі запити до прикладної логіки. Presentation не повинен самостійно вирішувати питання збереження, конвертації чи аналітики, інакше інтерфейс буде надмірно зв’язаним із деталями реалізації.

Пакет Application містить бізнес-логіку, реалізовану у вигляді сервісів. Саме тут доцільно розміщувати усі сервіси з бізнес-логікою. Application приймає команди від інтерфейсу, працює з доменними моделями, координує репозиторії та реалізує сценарну логіку. Наприклад, саме на цьому рівні визначається, що після збереження транзакції потрібно перерахувати бюджет і оновити представлення аналітики.

Пакет Domain містить головні сутності предметної області: транзакції, категорії, гаманці, бюджети, підписки, а також перелічувані типи. Domain не повинен залежати від конкретної UI-технології чи конкретного зовнішнього API. Це дає змогу зберегти чистоту предметної моделі та повторно використовувати її в різних контекстах.

Пакет Data Access відповідає за репозиторії та узагальнений доступ до локального сховища. Навіть якщо на рівні моделі SwiftData можна запитувати напряму, окремий шар доступу до даних зменшує кількість дублювання, уніфікує типові операції та полегшує подальше розширення архітектури.

Репозиторії інкапсулюють вибірки, сортування, збереження й видалення сутностей.

Пакет Infrastructure містить технічні реалізації: контейнер SwiftData, механізм запитів до OpenAI API, клієнт валютного сервісу, конфігураційні елементи, кешування зовнішніх відповідей і допоміжні інструменти серіалізації. Саме тут концентруються залежності від мережі, транспортного формату даних та конкретних сервісів.

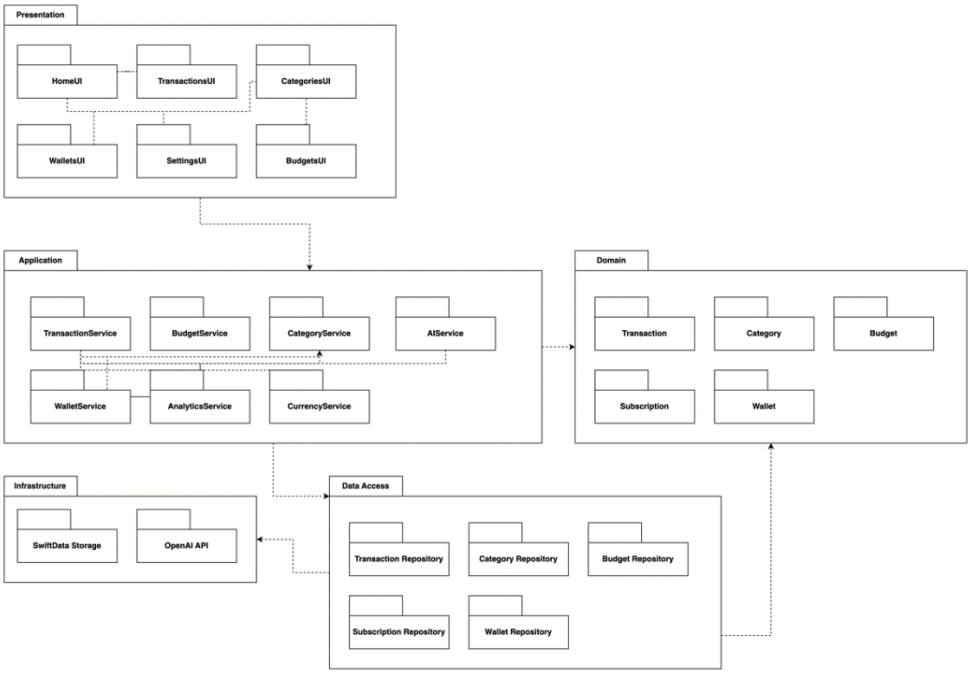

Рис. 2.5 — Діаграма пакетів інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Такий поділ створює кілька суттєвих переваг. По-перше, зменшується зв’язаність між інтерфейсом та інфраструктурою. По-друге, простіше тестувати бізнес-логіку окремо від UI. По-третє, система стає готовою до майбутніх розширень, наприклад, додавання синхронізації або веб-версії. По-четверте, чітко видно, в якому саме шарі повинні реалізовуватися «тригероподібні» операції, які в SQL-системах традиційно винесені в процедури чи тригери, а у запропонованій системі природно належать до сервісного рівня.

2.3.5 Діаграма розгортання

Діаграма розгортання (рис. 2.6) відображає фактичну архітектуру інформаційної системи обліку власних фінансів з використанням інтелектуального асистента та демонструє розміщення основних компонентів на фізичних вузлах. Основний акцент зроблено на взаємодії між мобільним клієнтом, локальним сховищем даних і зовнішніми сервісами.

Користувач працює із системою через мобільний застосунок, встановлений на iOS-пристрої. У межах цього пристрою розгорнуто основний артефакт «Application.app», який містить весь інтерфейс, бізнес-логіку та сервіси обробки даних. Для зберігання інформації використовується локальна база даних, реалізована через SwiftData. Вона забезпечує швидкий доступ до фінансових записів, дозволяє працювати без підключення до мережі та виступає основним джерелом даних для застосунку.

Окрім локальної роботи, застосунок інтегрується із зовнішніми сервісами через захищений протокол HTTPS. Зокрема, для отримання актуальних курсів валют використовується окремий сервер курсів валют, який надає API для доступу до фінансових даних. Це дозволяє системі автоматично оновлювати значення курсів і виконувати коректну конвертацію сум у різних валютах.

Також застосунок взаємодіє з сервером інтелектуального асистента через OpenAI API. Цей компонент відповідає за обробку запитів природною мовою, що дає змогу користувачу створювати транзакції або отримувати фінансову аналітику у більш зручному форматі. Обидві інтеграції виконуються лише за потреби, не впливаючи на базову функціональність системи.

Комунікація між мобільним застосунком і зовнішніми сервісами здійснюється виключно через HTTPS, що гарантує захист переданих даних. Водночас взаємодія між застосунком і локальною базою даних відбувається безпосередньо на пристрої, що мінімізує затримки та підвищує продуктивність.

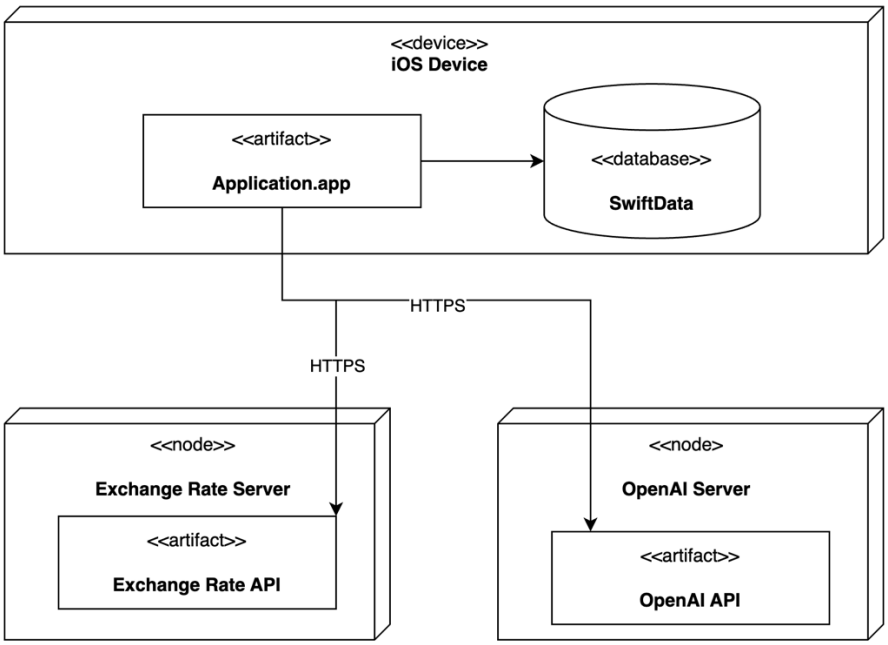

Рис. 2.6 — Діаграма розгортання інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

У підсумку, представлена архітектура поєднує автономність мобільного застосунку з можливостями зовнішніх сервісів. Це дозволяє забезпечити високу швидкодію, стабільність роботи без інтернету та водночас розширити функціональність системи за рахунок інтеграцій, зберігаючи безпеку та масштабованість рішення.

2.4 Інтеграція зовнішніх сервісів

Попри локально орієнтовану архітектуру, розроблювана система потребує взаємодії із зовнішніми сервісами там, де локальна реалізація не забезпечує достатньої цінності. У межах цієї роботи передбачено дві основні інтеграції: сервіс валютних курсів і сервіс штучного інтелекту.

Інтеграція з сервісом валютних курсів необхідна для підтримки багатовалютних транзакцій. Якщо користувач здійснює операцію в валюті, відмінній від валюти гаманця, система повинна мати змогу отримати актуальний курс, виконати конвертацію та зберегти обидва значення: оригінальну суму та нормалізовану внутрішню суму. При цьому важливо, щоб відсутність мережі не

блокувала повністю введення транзакції. Інтеграція з OpenAI API використовується для двох груп задач [3, 15-18]:

- інтерпретація природної мови з метою створення транзакції;
- формування пояснювальних відповідей на фінансові запити користувача.

У першому випадку система повинна перетворювати текст користувача на структурований об’єкт. У другому випадку асистент працює як інтерпретатор локальної аналітики: отримує агреговані дані і формує коротке пояснення або рекомендацію. Важливо підкреслити, що джерелом фінансових фактів залишається локальна база, а ШІ виступає шаром інтерфейсної інтерпретації, а не авторитетною фінансовою базою даних.

Безпековий аспект інтеграцій є принципово важливим. Дані доступу до зовнішніх сервісів не повинні бути жорстко вбудовані в клієнтський код у відкритому вигляді. Крім того, передача даних повинна здійснюватися лише через HTTPS, а в самих ШІ-запитах слід мінімізувати надлишковий контекст. Для сценарію створення транзакції немає потреби передавати весь фінансовий архів користувача, достатньо поточного текстового повідомлення та, за необхідності, переліку доступних категорій.

Інтеграції також повинні бути побудовані з урахуванням керованої відмово стійкості. Якщо зовнішній сервіс недоступний, система має не завершуватися аварійно, а коректно повідомляти користувача про ситуацію і, де це можливо, пропонувати локальний альтернативний сценарій. Це особливо важливо для ШІ-функцій, оскільки вони є розширенням базової функціональності, а не її критичною основою.

Саме така модель інтеграції є технічно виправданою для локального мобільного фінансового застосунку.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Реалізація доменних моделей та локального сховища

Програмна реалізація розроблюваної системи базується на відображенні сутностей предметної області у вигляді моделей SwiftData [11-13]. Такий підхід є логічним продовженням обраної архітектури, оскільки дозволяє мінімізувати розрив між концептуальною моделлю та кодом. На відміну від реляційно-центричних підходів, де потрібно вручну підтримувати схему БД, SQL-запити і трансформаційні шари, SwiftData дозволяє описувати доменну структуру природно для мови Swift і нативної екосистеми Apple.

Для опису моделей використовуються анотації `@Model` і `@Attribute(.unique)`, а також декларативне визначення зв’язків. Це забезпечує унікальність ідентифікаторів, підтримку зв’язків між об’єктами та зручну інтеграцію із запитами на рівні інтерфейсу. Нижче наведено показовий фрагмент реалізації базових доменних типів в Лістингу 3.1.

Лістинг 3.1 – Визначення переліків типів категорій та періодичності підписок

```
import Foundation
import SwiftData
enum CategoryKind: String, Codable, CaseIterable {
   case income, expense }
enum SubscriptionRecurrence: String, Codable, CaseIterable {
   case daily, weekly, monthly, yearly }
```

Ці перелічувані типи фіксують обмежений набір допустимих значень і дозволяють уникати великої кількості логічних помилок на рівні UI та бізнес-логіки. Вони також добре інтегруються з елементами вибору в SwiftUI.

Базова модель гаманця, наведена в Лістингу Б.1, реалізується як окрема сутність, до якої прив’язуються категорії та підписки. Саме гаманець задає

валютний контекст для аналітики, що робить його однією з центральних моделей системи.

Модель «Category» містить атрибути відображення і типу, а також зв’язок із транзакціями та бюджетом. Її роль полягає не лише в класифікації, а й у формуванні основи для бюджетування та аналітики. Саме категорії дозволяють перетворити список розрізнених фінансових фактів на структуровану картину витрат і доходів. Визначення моделі категорії наведено в Лістингу В.1.

Модель «Transaction» має найбільше практичне значення, оскільки працює як первинне джерело аналітичних розрахунків. Збереження двох сум, оригінальної та нормалізованої, є технічно доцільним рішенням для підтримки багатовалютності. Якщо користувач вводить підписку в доларах, а гаманець працює в гривні, застосунок не втрачає реальний контекст операції й одночасно може коректно агрегувати її в статистиці. Визначення моделі транзакції наведено в Лістингу Д.1.

Реалізація локального сховища передбачає створення контейнера моделей SwiftData, який ініціалізується під час запуску застосунку. Контейнер містить усі доменні сутності й підключається до середовища застосунку так, щоб екрани та сервіси могли працювати з єдиним «modelContext».

Особливістю локально-орієнтованої реалізації є відмова від класичних SQL-процедур і тригерів. У джерельних матеріалах прямо зазначено, що для SwiftData основні операції реалізуються на рівні програмної логіки. Це означає, що оновлення балансу, перерахунок бюджету, створення початкових категорій та інші похідні дії повинні бути організовані в сервісному шарі, а не на рівні БД. Такий підхід добре відповідає сучасній мобільній архітектурі, хоча накладає вищі вимоги на дисципліну прикладної логіки.

Окрему роль у локальному сховищі відіграє ініціалізація умовно постійних даних, ініціалізація яких наведена в Лістингу Ж.1. На практиці користувачеві важко починати роботу із зовсім порожньою системою, тому стартовий набір системних і шаблонних категорій створюється автоматично. Це дає змогу одразу почати облік без попереднього ручного конструювання довідника.

3.2 Реалізація сервісного шару та бізнес-логіки

Сервісний шар виконує роль координатора між інтерфейсом, доменними моделями та локальним сховищем [8-10]. Саме тут зосереджуються правила обробки даних, які в іншій архітектурі могли б бути реалізовані як SQL-тригери, stored procedures або серверні use-case-обробники. Для розроблюваної системи такий підхід є принципово важливим, оскільки вся прикладна логіка повинна працювати локально, узгоджено і передбачувано.

«WalletService» відповідає за створення, оновлення, вибір активного гаманцята базові операції з її життєвим циклом. На цьому ж рівні доцільно розміщувати функції отримання балансу гаманця, якщо він обчислюється агреговано на основі транзакцій. В іншому варіанті сервіс може зберігати кешований баланс і підтримувати його актуальність після CRUD-операцій над транзакціями. Реалізація сервісу управління гаманцем наведено в Лістингу К.1.

«CategoryService» виконує подвійну роль. По-перше, він забезпечує звичайне керування категоріями користувача. По-друге, він ініціалізує стартовий набір шаблонних категорій. Саме сервісний шар є правильним місцем для створення цих об’єктів при першому запуску.

«TransactionService» є ключовим сервісом системи. Саме він забезпечує створення транзакції, її редагування, видалення, нормалізацію валюти, прив’язку до категорії та оновлення похідних сутностей. Його логіка повинна гарантувати, що кожна транзакція після збереження коректно відображається в журналі, балансі гаманця, бюджетному прогресі та аналітичних звітах.

«BudgetService» повинен забезпечувати створення і зміну бюджетів, а також обчислення поточного рівня використання бюджету на основі транзакцій категорії. Фактично цей сервіс виступає проміжною ланкою між моделлю «Budget» і модулем аналітики. Його присутність дозволяє не дублювати обчислення в різних екранах.

«SubscriptionService» інкапсулює логіку роботи з регулярними платежами: створення підписок, зміну їхнього стану, визначення поточної або наступної

дати списання, формування аналітичних списків повторюваних витрат. На рівні MVP цього достатньо, щоб виділити підписки в окрему керовану підсистему.

«AnalyticsService» відповідає за підготовку агрегованих даних для дашбордів, категорійних зрізів і місячної динаміки. Він не повинен залежати від конкретного інтерфейсу візуалізації. Його результатом мають бути структуровані набори даних, готові до відображення у SwiftUI-компонентах.

«CurrencyService» інкапсулює логіку оновлення, кешування та використання валютних курсів. Особливо важливо, що цей сервіс повинен мати зрозумілу політику поведінки при відсутності мережі: або використовувати останній доступний курс, або явно сигналізувати про обмежений режим.

«AIService» є окремим сервісом, який працює із зовнішнім API, але з погляду архітектури залишається прикладним модулем верхнього рівня. Він повинен вміти формувати запити на інтерпретацію природної мови, перетворювати відповіді на внутрішні DTO-моделі і передавати ці результати в подальший користувацький сценарій підтвердження.

Загалом, сервісний шар розроблюваної системи реалізує підхід, у якому вся «динамічна» логіка системи винесена з моделей і не змішується з інтерфейсом. Точка входу у застосунок і ініціалізація сервісів наведені в Лістингу Л.1. Це забезпечує вищу підтримуваність коду, кращу тестованість та академічно коректний розподіл відповідальності.

3.3 Реалізація інтерфейсу користувача

Інтерфейс користувача в розроблюваній системі реалізується засобами SwiftUI, що дозволяє побудувати реактивну структуру екранів із прямим зв’язком між локальними даними та представленням [12, 14, 19]. Такий підхід є особливо зручним для фінансового застосунку, де зміни в сховищі повинні швидко відображатися на дашбордах, списках транзакцій, бюджетних індикаторах і картках гаманів.

На головному екрані користувач бачить баланс активного гаманця, блок швидкого переходу до створення транзакції та інших функцій системи. Важливо, щоб головний екран залишався інформаційно компактним: він має мотивувати до подальших дій, а не замінювати всі інші розділи.

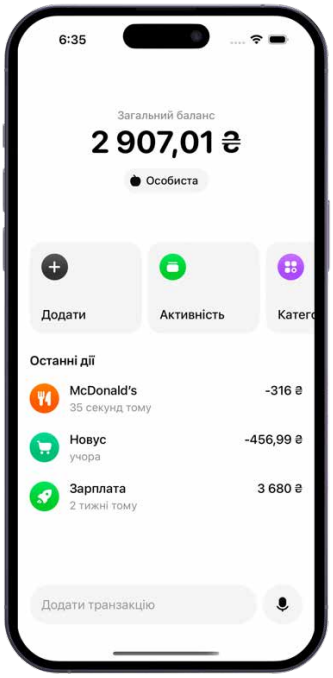

Рис.3.1 – Головний екран інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Екран додавання транзакції реалізується у вигляді форми з пріоритетом для поля суми, вибору категорії та дати. Для мобільного UX доречно використовувати модальну подачу, оскільки користувач зазвичай хоче швидко завершити цю дію та повернутися назад. Окремо важливо, що форма має підтримувати попереднє заповнення частини полів, якщо операція створюється в контексті певної категорії.

Рис.3.2 – Форма створення нової транзакції інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Фрагмент SwiftUI-реалізації форми збереження транзакції наведено в Лістингу М.1.

Екран активності повинен підтримувати перегляд усіх транзакцій, пошук, фільтрацію та перехід до редагування окремого запису. Наприклад, це може бути список, згрупований за датами. У фінансовому застосунку важливо, щоб транзакції мали чітке візуальне кодування: іконку категорії, підпис, дату, суму, позначення валюти та, за можливості, колірне розділення витрат і доходів.

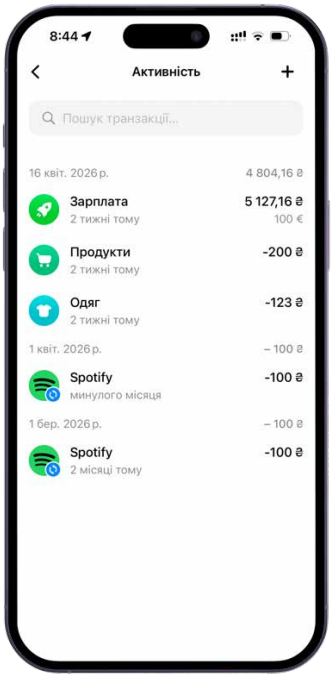

Рис.3.3 – Екран активності інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Екран гаманців повинен надавати огляд доступних фінансових просторів користувача. У картці гаманця доцільно показувати не лише назву та іконку, а й поточний баланс і базову валюту. Це дозволяє використовувати картки як елемент швидкої навігації між контекстами.

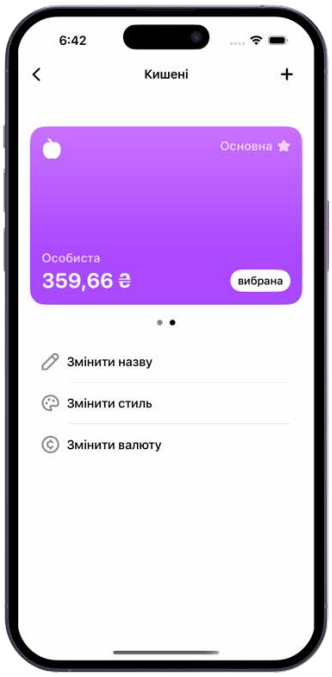

Рис.3.4 – Екран вибору і налаштування гаманця інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Екран бюджетів і категорій повинен бути зорієнтований на контроль, а не просто на довідник. Для бюджету важливо відображати прогрес, залишок, перевищення та зв’язок із відповідною категорією. Для категорій важлива швидкість створення та редагування. Тому інтерфейс цих розділів повинен підтримувати як перегляд, так і швидкі CRUD-операції без зайвого розгалуження.

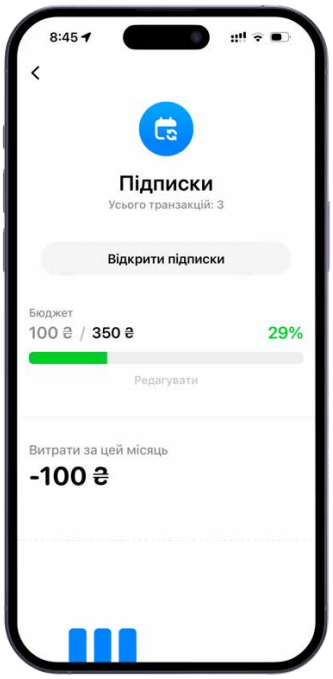

Рис.3.5 – Екран перегляду інформації про категорію інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Екран ШІ-асистента реалізується як текстовий інтерфейс з кнопками для швидкого доступу до інформації. На відміну від звичайного чату, такий екран має тісніше поєднувати повідомлення з реальними діями у застосунку.

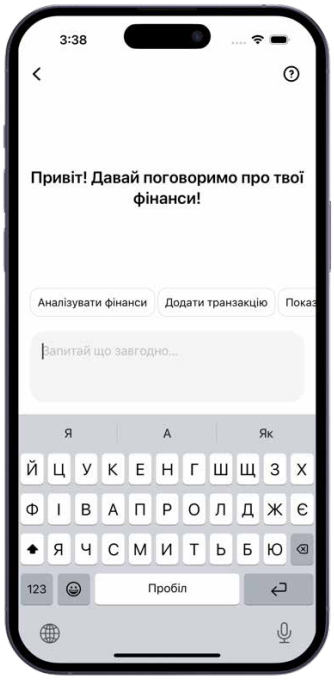

Рис.3.6 – Екран інтелектуального асистента інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

У реалізації UI важливо забезпечити узгодженість між темною та світлою темами, локалізацією, розмірами шрифту і читабельністю числових значень. Для фінансового застосунку дрібні візуальні недоліки швидко перетворюються на реальні помилки сприйняття, тому відображення сум, дат і валют повинно бути максимально чітким.

Отже, інтерфейс користувача розроблюваної системи реалізується як нативний, реактивний та орієнтований на короткі дії мобільний UI, який тісно інтегрований із локальним сховищем і сервісним шаром, але не перевантажений технічними деталями внутрішньої логіки [12, 14, 19].

3.4 Реалізація інтелектуального асистента та роботи з валютними курсами

Модуль ШІ-асистента є однією з ключових відмінностей розроблюваної системи від більшості класичних трекерів витрат. Його призначення полягає не в тому, щоб замінити весь інтерфейс застосунку, а в тому, щоб зменшити бар’єр входу в систему та надати користувачеві більш природний спосіб взаємодії з фінансовими даними.

На рівні створення транзакцій ШІ-асистент виконує задачу семантичного розбору коротких повідомлень. Користувач може описати подію природною мовою, а система повинна виділити з тексту щонайменше такі поля:

- сума;
- валюта;
- тип операції;
- потенційна категорія;
- дата або часовий контекст;
- короткий коментар.

Для цього застосунок формує структурований запит до OpenAI API [3]. У запиті доцільно вказати допустимі категорії, правила відповіді та вимогу

повернути результат у передбачуваному форматі, наприклад, JSON. Це спрощує подальше перетворення відповіді в DTO та зменшує ризик неоднозначного парсингу. Функція створення нової транзакції за допомогою інтелектуального асистента наведена в Лістингу Н.1.

Робота з валютними курсами реалізується як окремий модуль, який отримує актуальні курси через зовнішній API та надає сервісу транзакцій функцію конвертації. У типовому сценарії при створенні транзакції в іншій валюті система визначає, чи потрібна конвертація, отримує або бере з кешу відповідний курс, обчислює нормалізовану суму та зберігає обидва значення. При цьому важливо враховувати кілька аспектів:

- курси валют мають час актуальності;
- користувач повинен бачити, у якій валюті введено операцію;
- аналітика має працювати на єдиній базі порівняння;
- збій мережі не повинен блокувати локальне введення даних.

Реалізація ШІ-асистента та модуля валютних курсів демонструє, як зовнішні сервіси можуть підсилювати локальну систему без руйнування її архітектурного ядра. Обидві інтеграції залишаються допоміжними щодо локального сховища, але суттєво підвищують зручність і аналітичну цінність застосунку.

3.5 Формування аналітики та статистичних даних

Аналітика є тією функцією, яка перетворює застосунок із простого журналу транзакцій на інструмент ухвалення рішень. Для користувача недостатньо лише накопичувати записи про витрати й доходи. Важливо мати змогу зрозуміти структуру фінансової поведінки, виявити проблемні категорії, побачити динаміку змін та співвіднести поточні показники з бюджетними обмеженнями.

Найпростішим і найкориснішим аналітичним екраном є місячний дашборд. На ньому відображаються загальні доходи, витрати, поточний баланс або чистий фінансовий результат, а також статус основних бюджетів. Цей екран дає користувачеві відповідь на питання «Що відбувається з моїми фінансами зараз?» без потреби переходити до глибоких деталей.

Другий рівень аналітики пов’язаний із категорійними зрізами. Користувач повинен мати змогу побачити, яка частка місячних витрат припадає на харчування, транспорт, розваги, підписки чи інші категорії. Саме на цьому рівні виявляються фінансові звички, які складно побачити з окремих транзакцій.

Третій рівень аналітики пов’язаний із часовою динамікою. Для окремої категорії або загальних витрат доцільно показувати порівняння між місяцями. Це дозволяє відстежувати тренди, сезонні зміни та поступове зростання повторюваних витрат.

Такий підхід є цілком достатнім для локального мобільного застосунку, оскільки обсяги даних окремого користувача зазвичай не вимагають важких серверних аналітичних рішень [1, 2]. Перевагою локальної аналітики є миттєвий відгук, приватність і відсутність потреби в окремій звітній інфраструктурі.

Особливу роль відіграє інтеграція аналітики з бюджетами. Для кожної категорії витрат система повинна не лише показувати суму витрат, а й співвідносити її з установленим лімітом. Така форма подання дає користувачеві інструмент оперативного контролю, а не лише постфактум-огляд.

Окремий аналітичний інтерес становлять підписки. На відміну від звичайних категорій, вони формують не тільки історичну, а й прогнозну інформацію. Перегляд сукупного навантаження від підписок за місяць може стати окремим індикатором, який допомагає користувачеві виявляти приховані регулярні витрати.

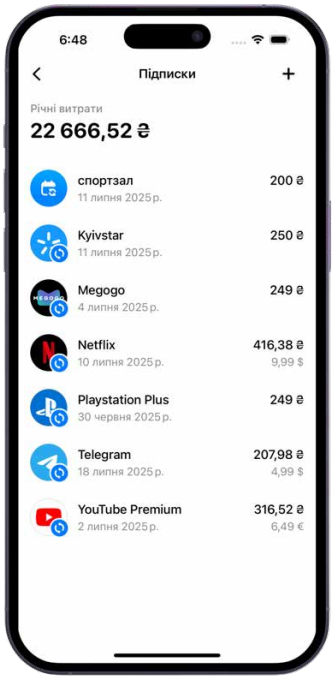

Рис.3.7 – Екран огляду активних підписок інформаційної системи обліку власних фінансів з використанням інтелектуального асистента.

Модуль аналітики у запропонованій системі виконує не допоміжну, а системоутворювальну функцію. Він узагальнює локальні дані, підтримує бюджетний контроль, виявляє структуру витрат і створює контекст для роботи інтелектуального асистента.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Методика тестування функціональних модулів

Тестування мобільного застосунку для обліку особистих фінансів є критично важливим, оскільки система працює з даними, від коректності яких безпосередньо залежить користувацька довіра. Навіть дрібна помилка, пов’язана з неправильним підрахунком суми, невірною категоризацією або втратою даних, у фінансовому контексті сприймається як суттєвий дефект. Тому підхід до тестування повинен охоплювати не лише окремі екрани, а й цілісність доменної логіки, коректність збереження даних, інтеграційні сценарії та передбачуваність поведінки системи в умовах помилок.

Для розроблюваної системи доцільно використовувати комбіновану методику тестування, що включає [2, 20]:

- модульне тестування сервісного шару;
- інтеграційне тестування взаємодії зі сховищем SwiftData;
- UI-тестування основних сценаріїв у Xcode;
- ручне сценарне тестування на симуляторі й реальному пристрої;
- приймальне тестування ключових користувацьких функцій.

Модульне тестування зосереджується на правильності бізнес-логіки. На цьому рівні перевіряються валідація транзакцій, оновлення бюджету, логіка ініціалізації категорій, обробка багатовалютних сум, поведінка сервісів при некоректних вхідних даних. Саме модульний рівень є найкращим місцем для контролю правил, що не залежать від конкретного інтерфейсу.

Інтеграційне тестування необхідне для перевірки того, що моделі, репозиторії, сервіси та локальне сховище працюють узгоджено. Наприклад, після створення транзакції має коректно оновитися журнал, а пов’язаний бюджет повинен змінити свій прогрес. Також перевіряється коректність збереження при повторному запуску застосунку.

UI-тестування важливе для тих сценаріїв, де зручність взаємодії прямо впливає на коректність використання. До таких сценаріїв належать: створення нової транзакції, редагування запису, фільтрація історії, створення бюджету, взаємодія з ШІ-асистентом, зміна активного гаманця. На цьому рівні перевіряється послідовність переходів, доступність кнопок, відображення повідомлень про помилки, коректне оновлення списків після збереження даних.

Окреме значення має тестування відмовостійкості. Система повинна коректно поводитися при відсутності мережі, помилках зовнішнього API, спробі зберегти неповну транзакцію, введенні нестандартної валюти або видаленні об’єкта, на який посилаються інші записи. У локально орієнтованому застосунку критично, щоб збій допоміжної інтеграції не робив недоступним основний фінансовий функціонал.

Для формування тестових даних доцільно використати кілька наборів сценаріїв:

- початковий користувач із порожнім сховищем;
- користувач із кількома каманцями й базовими категоріями;
- користувач із великим набором транзакцій за кілька місяців;
- користувач із підписками, бюджетами та багатовалютними операціями.

Такий підхід дозволяє перевіряти систему не лише на «щасливому шляху», а й на реалістичних комбінаціях станів. Особливо важливо протестувати сценарій першого запуску, бо саме він визначає, наскільки легко новий користувач може почати роботу із системою.

Оскільки застосунок створюється для iOS, тестування виконується у середовищі Xcode на симуляторі та реальному пристрої. Це дозволяє перевірити не лише бізнес-логіку, а й візуальні аспекти: адаптацію до темної теми, поведінку клавіатури, доступність, коректність прокрутки списків, зручність торкання керувальних елементів.

4.2 Тест-кейси та результати перевірки

На основі описаної методики було сформовано набір тест-кейсів, що охоплює критично важливі сценарії системи. Узагальнені результати наведено в табл. 4.1.

Таблиця 4.1

Сценарії тестування з результатами перевірки

№	Сценарій	Очікуваний результат	Результат перевірки
1	Перший запуск застосунку	Створюється локальне сховище, ініціалізуються стартові категорії	Успішно
2	Створення нового гаманця	Гаманець зберігається та відображається в списку	Успішно
3	Створення категорії витрат	Категорія доступна у формі транзакції та в довіднику	Успішно
4	Ручне створення витрати	Транзакцію збережено, журнал оновлено, баланс змінено	Успішно
5	Ручне створення доходу	Дохід відображено в історії та враховано в аналітиці	Успішно
6	Редагування транзакції	Змінені дані збережено, похідні показники оновлено	Успішно
7	Видалення транзакції	Запис видалено, журнал і агрегати перераховано	Успішно

Таблиця 4.1 (закінчення)

8	Створення бюджету для категорії	Ліміт збережено, прогрес бюджету розраховується	Успішно
9	Перевищення бюджету	Відображається візуальне попередження про перевищення	Успішно
10	Створення підписки	Підписка додається до списку регулярних витрат	Успішно
11	ШІ-створення транзакції при наявності мережі	Текст інтерпретовано, сформовано попередній перегляд операції	Успішно
12	Відсутність мережі під час ШІ-запиту	Система показує повідомлення, локальні функції продовжують працювати	Успішно
13	Створення багатовалютної транзакції	Оригінальна сума і нормалізоване значення збережені коректно	Успішно
14	Перегляд аналітики за місяць	Виведено доходи, витрати, категорії та бюджети	Успішно
15	Фільтрація транзакцій за категорією	Список відображає лише релевантні записи	Успішно
16	Перезапуск застосунку після внесення даних	Усі раніше збережені дані залишаються доступними	Успішно

Результати перевірки свідчать, що базовий функціонал системи працює коректно та узгоджено. Особливо важливим є те, що локальна архітектура демонструє стабільність у сценаріях без мережі, а зовнішні інтеграції не блокують роботу основного фінансового ядра.

Під час тестування інтерфейсних сценаріїв було підтверджено, що користувач може пройти повний шлях від першого запуску до внесення транзакцій, створення бюджету та перегляду аналітики без необхідності додаткового навчання. Це опосередковано свідчить про коректність обраного UI-підходу.

Позитивним результатом стала й перевірка сценаріїв локального збереження. Дані після повторного запуску залишаються доступними, а аналітика формується на основі збережених об’єктів без втрати цілісності. Для локального мобільного продукту це є однією з ключових умов практичної придатності.

4.3 Виявлені недоліки та шляхи їх усунення

Попри загальну коректність роботи системи, у процесі тестування та аналітичного перегляду можна виокремити кілька зон ризику, які не руйнують життєздатність застосунку, але потребують уваги в подальшому розвитку.

Першою проблемною зоною є неоднозначність природної мови у ШІ-сценаріях. Повідомлення користувача може бути неповним або двозначним, наприклад: «Заплатив за сервіс учора» без конкретизації суми чи валюти. У такому випадку система не повинна вгадувати дані надто агресивно. Найкращим шляхом усунення ризику є введення обов’язкового екрана підтвердження з можливістю ручного редагування всіх полів перед збереженням.

Друга проблема пов’язана з багатовалютністю. Якщо мережа недоступна, курс валют може бути неактуальним або відсутнім. Практичним шляхом вирішення є використання кешу останніх курсів з часовою позначкою та

виведення індикатора актуальності. Так, користувач не втрачає можливості внести транзакцію, але бачить обмеження точності перерахунку.

Третєю проблемою є можливе зниження читабельності аналітики при зростанні обсягу даних. Якщо транзакцій стає багато, базові графіки й списки можуть перевантажувати екран. Для усунення цього ризику доцільно розвивати багаторівневу аналітику: спочатку показувати узагальнення, а деталізацію відкривати лише на вимогу користувача.

Четверта група недоліків пов’язана з доступністю та інтерфейсними крайовими випадками. Навіть якщо система підтримує темну тему й базову сумісність із VoiceOver, повноцінний аудит доступності має бути окремим етапом перед публікацією. Зокрема, потрібно перевірити достатність контрасту, озвучування іконок категорій, навігацію за допомогою системних засобів доступності та коректність масштабування тексту.

П’ятим обмеженням є відсутність міжпристроєвої синхронізації. Для MVP це свідоме архітектурне рішення, але для частини користувачів воно буде відчутним. Шляхом майбутнього усунення цього недоліку є додавання окремого хмарного шару синхронізації за допомогою iCloud поверх локального ядра без руйнування поточної моделі роботи і міркувань безпеки.

Виявлені недоліки мають переважно характер керованих обмежень початкової версії, а не фундаментальних архітектурних помилок. Це означає, що розроблювана система має сильну основу для подальшого вдосконалення.

4.4 Впровадження програмної системи

Впровадження розроблюваного застосунку повинно відповідати типовому життєвому циклу мобільного застосунку для iOS. Після завершення програмної реалізації виконується збірка проєкту в Xcode, формування інсталяційного пакета та перевірка працездатності на симуляторі й фізичних пристроях.

Інсталяційний пакет системи представлений у вигляді файлу «.ipa», що може поширюватися через TestFlight або App Store.

На фізичному рівні основним вузлом розгортання є iPhone користувача. Саме на ньому розміщено застосунок і локальне сховище SwiftData. Зовнішні вузли, а саме сервер валютних курсів і сервер OpenAI, використовуються лише як допоміжні джерела зовнішньої функціональності. Така схема розгортання є простою, прозорою та добре узгоджується з локально орієнтованою архітектурою.

Мінімальні апаратні вимоги включають:

- пристрій типу iPhone;
- процесор Apple Silicon серії A або новіший;
- не менше 3 ГБ оперативної пам’яті;
- приблизно 100 МБ вільного місця;
- доступ до мережі для роботи ШІ та актуалізації курсів валют.

Програмні вимоги включають використання операційної системи iOS 18 або новішої версії, стандартного середовища виконання iOS, а також наявність необхідних дозволів і відповідних конфігурацій для здійснення мережевих запитів.

Процес впровадження доцільно організувати поетапно:

1. внутрішня перевірка стабільності на тестових даних;
2. збірка тестової версії і встановлення через Xcode на фізичний пристрій;
3. закрите beta-тестування через TestFlight;
4. збір зворотного зв’язку щодо швидкості введення транзакцій, роботи аналітики та асистента;
5. усунення критичних дефектів;
6. підготовка релізної збірки для поширення.

Після інсталяції система автоматично створює локальне сховище, ініціалізує стартові категорії й готова до використання без додаткового

налаштування. Це є однією з головних переваг обраного підходу: користувач фактично отримує робочий інструмент одразу після встановлення.

Впровадження запропонованої системи є технічно нескладним завдяки локально орієнтованій архітектурі, відсутності потреби в окремому серверному середовищі та використанню стандартних каналів доставки iOS-застосунків [3, 11-13, 21].

4.5 Практичне впровадження та публікація застосунку

Розроблюваний застосунок вже пройшов етап практичного впровадження та був опублікований в App Store. Це підтверджує працездатність програмної системи не лише в тестовому середовищі, а й у реальних умовах використання кінцевими користувачами.

Для публікації застосунку було виконано підготовку релізної збірки у середовищі Xcode, налаштовано сертифікат підпису та конфігуровано App Store Connect. Після успішної модерації Apple застосунок був розміщений у магазині App Store.

Процес публікації включає в себе підготовку метаданих застосунку, створення опису та ключових слів, створення скріншотів інтерфейсу застосунку, налаштування політики конфіденційності і перевірку на відповідність вимогам платформи iOS.

Більш детально процес практичного впровадження складався з кількох послідовних етапів, кожен з яких мав окреме технічне і організаційне значення. Після завершення основної розробки і стабілізації функціоналу було сформовано релізний кандидат, у межах якого перевірялися не лише базові сценарії створення транзакцій, роботи з категоріями, бюджетами та аналітикою, а й поведінка застосунку у виняткових випадках: під час порожнього стану бази, при повторному запуску, під час редагування записів і при зміні локальних параметрів пристрою.

Першим практичним етапом підготовки до завантаження в App Store стала ревізія проєкту в середовищі Xcode. На цьому кроці було перевірено коректність ідентифікатора застосунку, конфігурацію, параметри підпису, цільову версію iOS, права доступу та службові описи, які система відображає користувачу під час надання дозволів. Окремо перевірялися назва застосунку, іконка, системні метадані та відповідність локалізованих рядків фактичним сценаріям роботи програми.

Наступним кроком стала підготовка сертифікатів і профілів підпису в екосистемі Apple Developer. Для публікації iOS-застосунку недостатньо лише зібрати робочий проєкт, оскільки кожна фінальна збірка має бути пов’язана з конкретним обліковим записом розробника, цифровим сертифікатом і профілем розповсюдження. У межах цієї роботи було налаштовано обліковий запис розробника, перевірено App ID, активовано автоматичне керування підписом і забезпечено можливість коректного формування архіву без конфлікту сертифікатів або помилок автентифікації.

Після технічної підготовки було сформовано архів-збірку, тобто релізний пакет, придатний для подальшого завантаження через App Store Connect. На відміну від локального запуску на симуляторі чи фізичному пристрої, архівація є окремим етапом, який дозволяє провести фінальну перевірку залежностей, налаштувань конфігурації, оптимізацій компілятора та повноти ресурсів, що входять до складу застосунку. Саме на цьому етапі часто виявляються помилки, які не помітні під час звичайної розробки, наприклад, некоректні чи відсутні ресурси, конфлікти версій або неправильні параметри підпису.

Після успішного створення архіву застосунок було перевірено засобами валідації Apple. Така перевірка дає змогу ще до безпосереднього завантаження виявити критичні порушення вимог платформи. Зокрема, контролювалася коректність структури пакета застосунку, наявність обов’язкових службових полів, сумісність із вимогами App Store та відсутність технічних причин, які могли б призвести до автоматичного відхилення збірки. Лише після

проходження цього кроку релізний пакет був завантажений у App Store Connect для подальшого оформлення сторінки застосунку.

Окремий великий блок роботи стосувався саме наповнення сторінки продукту в App Store Connect. Було підготовлено назву застосунку, короткий опис, повний маркетинговий опис, ключові слова для внутрішнього пошуку, категорію продукту, контактні дані, політику конфіденційності, інформацію про вік користувачів і відповіді на запитання блоку App Privacy. Крім цього, були підготовлені скріншоти для різних сценаріїв демонстрації інтерфейсу, які мали не просто показати екрани застосунку, а пояснити цінність продукту: швидке внесення транзакцій, перегляд категорій, аналіз витрат, роботу бюджетів і зручність щоденного контролю фінансів.

На практиці важливим виявився і передрелізний етап ручного контролю якості. Перед відправленням застосунку на модерацію перевірялися стабільність навігації між екранами, коректність локального збереження записів, швидкість відкриття основних розділів, робота інтерфейсних станів за відсутності даних, коректність форматування сум і дат, а також зрозумілість текстів, які бачить користувач. Така перевірка дуже важлива, оскільки модерація Apple оцінює не лише технічну працездатність застосунку, а й його завершеність, узгодженість інтерфейсу та відсутність сценаріїв, які можуть ввести користувача в оману.

Після завершення налаштувань збірка була відправлена на перевірку. Етап модерації в App Store є окремим процесом, у межах якого Apple перевіряє відповідність застосунку правилам платформи, коректність використання, відсутність прихованого або нестабільного функціоналу, а також відповідність заявленого опису реальній поведінці продукту. У разі потреби розробник має бути готовим надати додаткові пояснення щодо логіки роботи, тестового сценарію входу, політики збору даних або причин використання окремих системних дозволів.

Після успішного проходження перевірки було виконано публікацію застосунку для кінцевих користувачів. Сам факт появи продукту в App Store має практичне значення для цієї кваліфікаційної роботи, оскільки підтверджує

завершення повного циклу, від аналітики і проєктування до реального постачання програмного продукту через офіційний канал дистрибуції Apple. Це означає, що інформаційна система була доведена до стану, коли її можна не лише демонструвати як прототип, а й надавати користувачам для реального щоденного використання.

Таким чином, практичне впровадження не обмежувалося одноразовим завантаженням файлу в магазин застосунків. Воно охоплювало серію взаємопов’язаних дій: технічну підготовку релізу, налаштування інфраструктури Apple Developer, складання й перевірку архіву, формування сторінки продукту, контроль якості, проходження модерації та фінальну публікацію. Саме сукупність цих кроків дозволяє розглядати результат розробки як повноцінно впроваджений програмний продукт, а не лише як навчальну реалізацію окремих функцій.

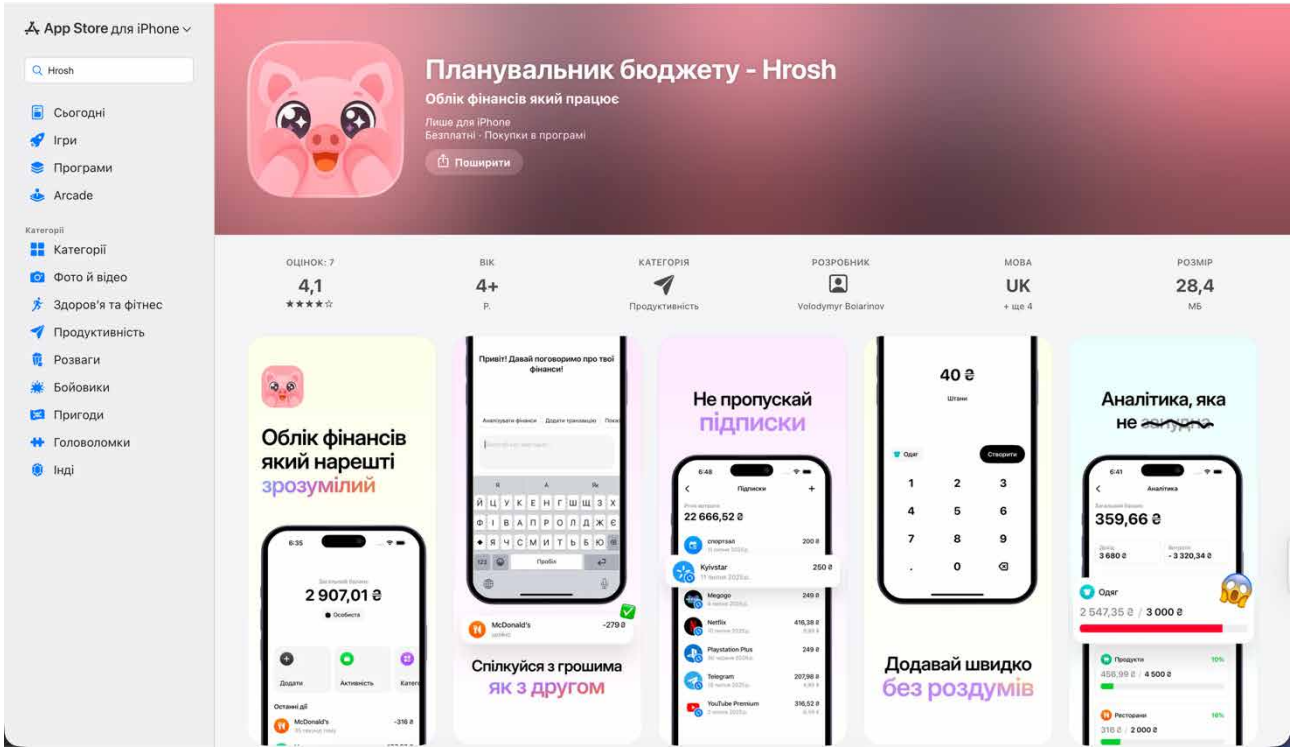

Рис.4.1 – Сторінка мобільного застосунку у магазині App Store

4.6 Оцінка ефективності, продуктивності та зручності використання інформаційної системи

Оцінка ефективності системи повинна спиратися на ті задачі, для яких вона створювалася. Для розроблюваної системи такими задачами є спрощення ведення персонального фінансового обліку, підвищення регулярності внесення даних, покращення прозорості витрат і доходів та надання користувачеві зрозумілої аналітики.

З погляду ефективності основною перевагою запропонованої системи є зменшення операційної складності повсякденного обліку. У порівнянні з паперовими записами або таблицями користувач отримує автоматичний підрахунок, категоризацію, збереження історії та готову аналітику. У порівнянні з банківськими застосунками запропоноване рішення надає ширший контекст: бюджети, підписки, багатовалютність, локальний контроль і текстову взаємодію з інтелектуальним асистентом.

З точки зору продуктивності обрана локальна архітектура є виправданою. Більшість базових операцій виконується без звернення до мережі, а отже, не залежить від зовнішніх затримок. Локальне сховище SwiftData є достатнім для характерних для одного користувача обсягів фінансових даних. Це дозволяє очікувати швидке відкриття головного екрана, оперативне оновлення списків та комфортну роботу з аналітичними зрізами.

Для більш структурованої оцінки доцільно зіставити внесок основних властивостей системи у користувацьку цінність.

Таблиця 4.2

Внесок основних функції інформаційної системи

Критерій	Прояв у розроблюваній системі	Практичний ефект
Швидкість внесення даних	Коротка форма та ШІ-сценарій	Зростає ймовірність регулярного обліку

Таблиця 4.2 (закінчення)

Автономність	Локальна робота без мережі	Дані доступні в будь-який момент
Прозорість фінансів	Дашборди, бюджети, категорії	Користувач бачить структуру витрат
Контроль підписок	Окремий модуль регулярних платежів	Легше виявляти повторювані витрати
Пояснювальний рівень	ШІ-асистент і текстові відповіді	Аналітика стає зрозумілішою
Конфіденційність	Локальне ядро зберігання	Підвищується довіра до системи

Важливо зазначити, що ця оцінка є переважно функціонально-аналітичною, а не результатом масового польового дослідження з великою вибіркою користувачів. Проте навіть на рівні проєктування і тестових сценаріїв видно, що запропонована система має виразну прикладну цінність і добре відповідає заявленим вимогам.

Таким чином, система демонструє достатній рівень ефективності, продуктивності та зручності використання для локального мобільного застосунку персонального фінансового обліку. Її архітектура є виправданою, а функціональне наповнення забезпечує практичну цінність для кінцевого користувача.

4.7 Перспективи масштабування та розвитку

Подальший розвиток розроблюваної системи може відбуватися за кількома напрямами, що логічно продовжують закладену в роботі архітектуру.

Перший напрям пов’язаний із хмарною синхронізацією між пристроями. На рівні MVP локальне ядро є оптимальним, але в міру зростання продукту користувачі можуть очікувати підтримку iPhone, iPad і резервного відновлення

даних. Таке розширення доцільно реалізовувати як надбудову над поточною моделлю, а не як повну заміну локального підходу.

Другий напрям стосується розширення аналітики. Йдеться про довгострокові тренди, прогнозування майбутніх витрат, виявлення аномалій, персоналізовані поради щодо підписок і бюджетів, а також порівняння фінансових періодів на більш глибокому рівні. Саме тут модуль ШІ може отримати нову роль, переходячи від простої інтерпретації запитів до контекстного пояснення патернів поведінки.

Третій перспективний напрям це сімейні або групові бюджети. У такому випадку система працюватиме вже не лише з індивідуальним користувачем, а з кількома учасниками спільного фінансового простору. Це вимагатиме розширення моделі ролей, механізмів синхронізації та правил розмежування доступу.

Четвертий напрям пов’язаний з імпортом даних і банківськими інтеграціями. Повноцінне автоматичне підключення до банківських API є складним із регуляторної та технічної точок зору, але навіть частковий імпорт виписок або напівавтоматичне зіставлення операцій могло б суттєво зменшити ручне навантаження на користувача.

П’ятий напрям стосується підвищення доступності та персоналізації інтерфейсу: віджетів, швидких дій та налаштовуваних дашбордів.

Закладена архітектура не обмежується поточною реалізацією, а створює платформу для подальшого функціонального й ринкового розвитку продукту.

ВИСНОВКИ

У роботі виконано комплексний аналіз, проєктування та опис реалізації мобільного застосунку для обліку особистих фінансів з використанням інтелектуального асистента. Поставлену мету досягнуто, а результати дослідження можуть бути використані як основа для подальшого розвитку реального iOS-застосунку класу менеджеру обліку особистих фінансів.

У межах роботи досліджено предметну область персонального фінансового обліку та виявлено типові проблеми користувачів, зокрема фрагментарність фінансових даних, складність регулярного внесення операцій, обмеженість традиційних інструментів обліку та потребу в локально-орієнтованій аналітичній підтримці. Проведено аналіз програмних аналогів і обґрунтовано конкурентну нішу розроблюваного рішення як локального мобільного застосунку з підтримкою бюджетів, підписок, багатовалютності та взаємодії із ШІ. Показано, що обрана концепція є доцільною для реалізації MVP і краще відповідає вимогам конфіденційності та автономності порівняно з повністю серверно-орієнтованими підходами.

На основі проведеного аналізу сформульовано функціональні, нефункціональні, технологічні та безпекові вимоги до системи, що дозволило спроєктувати її архітектуру, модель даних, бізнес-процеси, пакетну структуру та інтерфейс користувача. Розроблено концепцію програмної реалізації локального ядра системи з використанням Swift, SwiftUI та SwiftData. Показано, що ключові сутності, зокрема Wallet, Category, Budget, Subscription і Transaction, забезпечують підтримку фінансового обліку, аналітики та локального збереження даних, тоді як сервісний шар відповідає за координацію бізнес-логіки застосунку.

Окремо обґрунтовано підхід до вибіркової інтеграції із зовнішніми сервісами, зокрема OpenAI API та сервісами валютних курсів, без порушення локально-орієнтованого характеру системи. Встановлено, що інтелектуальний асистент доцільно використовувати як інтерфейс природної мови, а не як

джерело авторитетних фінансових даних. Також запропоновано методику тестування, сформовано набір тест-кейсів, визначено обмеження MVP-версії та надано рекомендації щодо впровадження, супроводу, охорони праці й безпечної експлуатації системи.

Практичне значення отриманих результатів полягає в можливості безпосереднього використання запропонованих архітектурних, модельних та інтерфейсних рішень під час створення й подальшого розвитку мобільного застосунку для персонального фінансового обліку. Перспективними напрямами розвитку визначено міжпристроєву синхронізацію, розширену аналітику, глибшу персоналізацію інтерфейсу, підтримку сімейних бюджетів і розширення банківських інтеграцій.

Отже, виконана кваліфікаційна робота відповідає поставленим завданням, демонструє цілісний і послідовний зв’язок між аналізом предметної області, проєктуванням, реалізацією, тестуванням і рекомендаціями щодо безпечної експлуатації, а також підтверджує практичну доцільність запропонованого програмного рішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laudon K. C., Laudon J. P. Management Information Systems: Managing the Digital Firm. 16th ed. Harlow : Pearson, 2019. 656 p.
2. Spillner A., Linz T., Schaefer I. Foundations of Software Testing: ISTQB Certification. 4th ed. Boston : Rocky Nook, 2014. 314p
3. OpenAI. API Platform Documentation [Електронний ресурс]. URL: platform.openai.com/docs (дата звернення: 04.01.2026).
4. YNAB. Official Website [Електронний ресурс]. URL: ynab.com (дата звернення: 08.01.2026).
5. EveryDollar. Official Website [Електронний ресурс]. URL: ramseysolutions.com/ramseyplus/everydollar (дата звернення: 08.01.2026).
6. MoneyLover. Official Website [Електронний ресурс]. URL: moneylover.me (дата звернення: 08.01.2026)
7. BudgetBakers. Official Website [Електронний ресурс]. URL: budgetbakers.com/en (дата звернення: 08.01.2026)
8. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2017. 792 p.
9. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill, 2019. 784 p.
10. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Pearson, 2017. 432 p.
11. Apple Inc. Swift Documentation [Електронний ресурс]. URL: developer.apple.com/swift (дата звернення: 15.02.2026).
12. Apple Inc. SwiftUI Documentation [Електронний ресурс]. URL: developer.apple.com/documentation/swiftui (дата звернення: 20.02.2026).
13. Apple Inc. SwiftData Documentation [Електронний ресурс]. URL: developer.apple.com/documentation/swiftdata (дата звернення: 25.02.2026).
14. Apple Inc. Human Interface Guidelines [Електронний ресурс]. URL: developer.apple.com/design/human-interface-guidelines (дата звернення: 28.02.2026).

15. ISO/IEC 27001. Information security, cybersecurity and privacy protection. Information security management systems. Requirements. Geneva : ISO, 2022.
16. Закон України «Про захист персональних даних» [Електронний ресурс]. URL: zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 14.03.2026).
17. Закон України «Про інформацію» [Електронний ресурс]. URL: zakon.rada.gov.ua/laws/show/2657-12 (дата звернення: 14.03.2026).
18. General Data Protection Regulation (GDPR) [Електронний ресурс]. URL: gdpr-info.eu (дата звернення: 14.03.2026).
19. Figma. Product Overview [Електронний ресурс]. URL: figma.com (дата звернення: 23.03.2026).
20. ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models. Geneva : ISO, 2011.
21. Брюгге Б., Дутуа А. Об’єктно-орієнтована інженерія програмного забезпечення з використанням UML, шаблонів та Java. Київ : Діалектика, 2012. 680 с.

Додаток А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій Декларація про академічну доброчесність та використання ШІ

Я, Боярінов Володимир Іванович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Розробка мобільного застосунку обліку особистих фінансів з використанням інтелектуального асистента» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ **Володимир БОЯРІНОВ**

Додаток Б

Лістинг Б.1

```
import Foundation
import SwiftData
@Model final class Wallet: Identifiable {
    @Attribute(.unique) var id: UUID
    var name: String
    var color: String
    var icon: String
    var isPrimary: Bool
    var currency: String = "USD"
    @Relationship(deleteRule: .cascade) var categories: [Category] = []
    @Relationship(deleteRule: .cascade) var transactions: [Transaction] = []
    @Relationship(deleteRule: .cascade) var budgets: [Budget] = []
    @Relationship(deleteRule: .cascade) var subscriptions: [Subscription] = []
    init(
        id: UUID = UUID(),
        name: String,
        color: String,
        icon: String,
        isPrimary: Bool = false,
        currency: String = "USD") {
        self.id = id
        self.name = name
        self.color = color
        self.icon = icon
        self.isPrimary = isPrimary
        self.currency = currency
    }
}
```

Додаток В

Лістинг В.1

```
import SwiftData
import Foundation
@Model final class Category: Identifiable, Equatable, Hashable {
   @Attribute(.unique) var id: UUID
   var name: String
   var color: String
   var icon: String
   var kind: CategoryKind
   @Relationship(inverse: \Wallet.categories) var wallet: Wallet?
   @Relationship(deleteRule: .cascade) var transactions: [Transaction] = []
   @Relationship(deleteRule: .cascade) var budget: Budget?
   @Relationship(deleteRule: .cascade) var summaries: [CategorySummary] =
[]
   init(id: UUID = UUID(),
        name: String,
        color: String,
        icon: String,
        kind: CategoryKind ) {
       self.id = id
       self.name = name
       self.color = color
       self.icon = icon
       self.kind = kind
       self.isFixed = isFixed
       self.isTemplate = isTemplate
       self.wallet = wallet}
}
```

Додаток Д

Лістинг Д.1

```
import Foundation
import SwiftData
@Model final class Transaction: Identifiable, Equatable, Hashable  {
    @Attribute(.unique) var id: UUID
    var kind: TransactionKind
    var amount: Double
    var originalAmount: Double
    var currency: String
    var note: String?
    var timestamp: Date
    @Relationship var subscription: Subscription?
    @Relationship var category: Category
    @Relationship var wallet: Wallet
    init(
        id: UUID = UUID(),
        kind: TransactionKind,
        amount: Double,
        originalAmount: Double,
        currency: String,
        timestamp: Date = .now,
        note: String? = nil,
        subscription: Subscription? = nil,
        category: Category,
        wallet: Wallet
    ) {
        self.id = id
        self.kind = kind
        self.amount = amount
```

```
        self.originalAmount = originalAmount
        self.currency = currency
        self.timestamp = timestamp
        self.note = note
        self.subscription = subscription
        self.category = category
        self.wallet = wallet
    }
}
```

Додаток Ж

Лістинг Ж.1

```
Import Foundation
let fixedCategoryTemplates: [TemplateCategory] = [
    .init(
        name: "other_income_category",
        icon: "arrow.up.circle.fill",
        color: "catGreen",
        kind: .income,
    ),
    .init(
        name: "other_expense_category",
        icon: "arrow.down.circle.fill",
        color: "catRed",
        kind: .expense,
    ),
    .init(
        name: "subscriptions_category",
        icon: "calendar.repeat.fill",
        color: "catBlue",
        kind: .expense,
    ),
]
```

Додаток К

Лістинг К.1

```
import SwiftUI
import SwiftData
import Observation
@Observable
@MainActor
final class WalletService: ObservableObject, DataService {
    let modelContext: ModelContext
    required init(modelContext: ModelContext) {
        self.modelContext = modelContext
    }
    func loadWallet(by id: UUID) -> Wallet? {
        let descriptor = FetchDescriptor<Wallet>(
            predicate: #Predicate { $0.id == id },
        )
        return try? modelContext.fetch(descriptor).first
    }
    func updateWallet(
        _ wallet: Wallet,
        name: String? = nil,
        color: String? = nil,
        icon: String? = nil,
        currency: String? = nil,
    ) throws {
        if let name = name {
            wallet.name = name
        }
        if let color = color {
            wallet.color = color
```

```
    }
    if let icon = icon {
        wallet.icon = icon
    }
    if let currency = currency {
        wallet.currency = currency
    }
    do {
        try modelContext.save()
    } catch {
        print("Failed to update wallet: \(error)")
        throw error
    }
}

func deleteWallet(_ wallet: Wallet) throws {
    guard !wallet.isPrimary else {
        throw WalletServiceError.cannotDeletePrimaryWallet
    }
    _ = wallet.id
    modelContext.delete(wallet)
    try modelContext.save()
    objectWillChange.send()
}
@discardableResult
func createWallet(
    name: String,
    color: String,
    icon: String,
    isPrimary: Bool,
```

```
            currency: String,
            selectedExtras: [TemplateCategory]
        ) throws -> Wallet {
            let wallet = Wallet(name: name, color: color, icon: icon, isPrimary: isPrimary, currency: currency)
            modelContext.insert(wallet)
            attachCategories(from: fixedCategoryTemplates, to: wallet)
            if !selectedExtras.isEmpty {
                attachCategories(from: selectedExtras, to: wallet)
            }
            try modelContext.save()
            return wallet
        }
        private func attachCategories(from templates: [TemplateCategory], to wallet: Wallet) {
            for tpl in templates {
                let category = Category(
                    name: tpl.name,
                    color: tpl.color,
                    icon: tpl.icon,
                    kind: tpl.kind == .income ? .income : .expense,
                    wallet: wallet
                )
                modelContext.insert(category)
            }
        }
    }
    enum WalletServiceError: LocalizedError {
        case cannotDeletePrimaryWallet
        var errorDescription: String? {
```

```
        switch self {
        case .cannotDeletePrimaryWallet:
            return "Cannot delete the primary wallet"
        }
    }
}
```

Додаток Л

Лістинг Л.1

```
import SwiftData
import SwiftUI
@main
struct iosApp: App {
    private let container = ModelContainerProvider.shared
    @StateObject private var appState = AppState()
    @StateObject private var walletService = WalletService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var transactionService = TransactionService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var categoryService = CategoryService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var budgetService = BudgetService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var exchangeRateService = ExchangeRateService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var analyticsService = AnalyticsService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var aiService = AIService(
        modelContext: ModelContainerProvider.shared.mainContext
    )
    @StateObject private var subscriptionService = SubscriptionService(
```

```
    modelContext: ModelContainerProvider.shared.mainContext
)
var body: some Scene {
        ContentView()
    }
    .modelContainer(container)
    .environment(appState)
    .environment(walletService)
    .environment(transactionService)
    .environment(categoryService)
    .environment(budgetService)
    .environment(exchangeRateService)
    .environment(analyticsService)
    .environment(aiService)
    .environment(subscriptionService)
}
```

Додаток М

Лістинг М.1

```
func createTransaction() {
    guard let wallet = wallet else {
        return
    }
    let categories = allCategories
        .filter { $0.wallet?.id == wallet.id }
        .sorted { $0.displayName.localizedCompare($1.displayName) == .orderedAscending }
    guard
        let category = selectedCategory
            ?? categories.first(where: {
                $0.name == "other_expense_category"
            })
    else {
        print("No valid category found.")
        return
    }
    guard let raw = Double(amount), raw > 0 else {
        print("Validation failed: Amount must be positive.")
        return
    }
    let transactionKind: TransactionKind =
        (category.kind == .expense) ? .expense : .income
    let correctAmount: Double
    if selectedCurrency != wallet.currency {
        correctAmount =
            exchangeRateService.convertAmount(
                raw,
```

```
            from: selectedCurrency,
            to: wallet.currency
        ) ?? raw
} else {
    correctAmount = raw
}
switch mode {
case .new, .newWithCategory:
    do {
        let newTransaction = try transactionService.createTransaction(
            kind: transactionKind,
            amount: correctAmount,
            originalAmount: raw,
            currency: selectedCurrency,
            note: note.isEmpty == false ? note : nil,
            timestamp: selectedDate,
            category: category,
            wallet: wallet
        )
    } catch {
        print("Failed to create transaction: \(error)")
    }
case .edit(let oldTransaction):
    do {
        let newTransaction = try transactionService.createTransaction(
            kind: transactionKind,
            amount: correctAmount,
            originalAmount: raw,
            currency: selectedCurrency,
            note: note.isEmpty == false ? note : nil,
```

```
                timestamp: selectedDate,
                category: category,
                wallet: wallet
            )
            try transactionService.deleteTransaction(oldTransaction)
        } catch {
            print("Failed to create transaction: \(error)")
        }
    }
    dismissFullScreenCover()
}
```

Додаток Н

Лістинг Н.1

```
private func processTransactions(
    _ transactions: [TransactionPayload],
    in wallet: Wallet,
    allCategories: [Category],
    exchangeRateService: ExchangeRateService
) async throws {
    var transactionsInsertedCount = 0
    for (index, txnPayload) in transactions.enumerated() {
        if index > 0 {
            try await Task.sleep(for: .milliseconds(100))
        }
        let matchedCategory = allCategories.first(where: {
            UUID(uuidString: txnPayload.category) == $0.id
        }) ?? allCategories.first(where: {
            $0.name.lowercased() == "other \(txnPayload.kind.rawValue.lowercased())" && $0.isFixed
        })
        guard let category = matchedCategory else {
            print("No matching category found for: \(txnPayload.category)")
            continue
        }
        var correctAmount = txnPayload.originalAmount
        if txnPayload.originalCurrency != wallet.currency {
            correctAmount = exchangeRateService.convertAmount(
                txnPayload.originalAmount,
                from: txnPayload.originalCurrency,
                to: wallet.currency,
                at: txnPayload.date
```

```
            ) ?? txnPayload.originalAmount
        }
        let newTransaction = Transaction(
            kind: txnPayload.kind,
            amount: correctAmount,
            originalAmount: txnPayload.originalAmount,
            currency: txnPayload.originalCurrency,
            timestamp: txnPayload.date,
            note: txnPayload.descriptionText,
            category: category,
            wallet: wallet
        )
        modelContext.insert(newTransaction)
    }
```