

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Мобільний застосунок особистого фінансового менеджера»

Виконав студент групи ЗПЗ-22-1
ЦУРЦУМІЯ Костянтин

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
КУШНІР Надія Олександрівна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЦУРЦУМІЯ Костянтин**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Мобільний застосунок особистого фінансового менеджера»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №93/с**

Вихідні дані для роботи: **процес обліку та аналізу особистих фінансів користувача на платформі Android.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	01.03.2026р.	01.03.2026р
2	Вакалюк Т.А.	01.03.2026р	01.03.2026р
3	Вакалюк Т.А.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з мобільного застосунку особистого фінансового менеджера та пояснювальної записки. Пояснювальна записка до випускної роботи містить 69 сторінок, 31 ілюстрації та 11 таблиць.

Мета кваліфікаційної роботи, це створити інструмент для контролю за фінансами, що дозволить вести базовий облік доходів та витрат, переглядати стан рахунків, планувати бюджети, ставити фінансові цілі, керувати регулярними платежами та робити нотатки.

Програма написана мовою Kotlin у середовищі розробки Android Studio. В процесі розробки застосовувалися принципи об'єктно-орієнтованого програмування, підходи до проєктування баз даних та сучасних користувацьких інтерфейсів, а також методи функціонального тестування. Архітектура та візуальна складова побудовані за допомогою Jetpack Compose і Material Design 3, а за роботу з даними відповідають Room, Firebase та Firestore.

Результатом роботи став мобільний застосунок «Finna: Financial Manager», доступний до завантаження на платформі Google Play Market.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, ПЕРСОНАЛЬНІ ФІНАНСИ, БЮДЖЕТИ, ВИТРАТИ, ДОХОДИ.

					ЗІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок особистого фінансового менеджера	Літ.	Арк.	Аркушів
Розроб.		Цурицумія. К						
Керівник		Вакалюк. Т.А					3	
						Житомирська політехніка група ЗІПЗ-22-1		
Н. контр.								
Зав. каф.		Вакалюк. Т.А						

ABSTRACT

The bachelor's thesis consists of a mobile application for personal financial management and an explanatory note. The explanatory note contains 69 pages, 31 illustration and 11 tables.

The goal of the thesis is to create a financial management tool that allows users to keep basic track of income and expenses, view account balances, plan budgets, set financial goals, manage recurring payments, and take notes.

The program is written in Kotlin using the Android Studio development environment. During development, the principles of object-oriented programming, approaches to database design and modern user interfaces, as well as functional testing methods were applied. The architecture and visual components were built using Jetpack Compose and Material Design 3, while data handling is handled by Room, Firebase, and Firestore.

The result of this work is the mobile app “Finna: Financial Manager,” available for download on the Google Play Store.

KEYWORDS: MOBILE APP, PERSONAL FINANCES, BUDGETS, EXPENSES, INCOME.

					3ИП3.КР.Б-121-26-П3	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ.....	9
1.1. Постановка задачі	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Формування вимог до майбутнього мобільного застосунку.....	17
1.4 Обґрунтування вибору інструментальних засобів	20
Висновки до першого розділу	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ	25
2.1 Визначення варіантів використання та структури користувацьких сценаріїв.....	25
2.2 Проєктування архітектури мобільного застосунку	28
2.3. Проєктування структури даних та локальної бази даних.....	31
2.4. Розробка основних модулів та алгоритмів роботи.....	35
2.5. Організація збереження та синхронізації даних.....	39
Висновки до другого розділу	42
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ЗАСТОСУНКОМ	44
3.1 Загальна структура інтерфейсу	44
3.2 Тестування основних сценаріїв роботи	51
Висновки до третього розділу	60
ВИСНОВКИ.....	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62
ДОДАТКИ.....	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - програмний інтерфейс застосунку

DAO - об'єкт доступу до даних у Room

БД - база даних

MD 3 - система принципів і компонентів для мобільного інтерфейсу

Room - бібліотека Android Jetpack для роботи з локальною базою даних

UI - користувацький інтерфейс

UX - користувацький досвід

					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						6

ВСТУП

Люди щоденно стикаються в житті з витратами та надходженням грошей. Для цього вони одночасно використовують банківські картки готівку або різноманітні електронні гаманці з кількома валютами. Через такий сильний розподіл інформації між багатьма джерелами стає дуже складно швидко оцінити свій реальний фінансовий стан. Зробити це без окремого інструмента зведеного обліку майже неможливо.

Звичайні банківські додатки дають змогу переглядати історію операцій лише за конкретним рахунком або випущеною картою. Проте вони не забезпечують користувачу повну картину її загальних статків. На практиці користувач може мати відкриті рахунки одразу в кількох різних банках а частину своїх збережень тримати виключно готівкою.

У подібній ситуації виникає гостра потреба у єдиній мобільній програмі яка успішно об'єднає всі ці розрізнені елементи. Вона має зібрати разом усі рахунки доходи витрати бюджети та регулярні платежі. Головна особливість такого мобільного застосунку полягає в тому що він постійно працює з вкрай чутливими даними користувача. До них належить уся приватна інформація про щомісячні доходи поточні витрати наявні рахунки плани на майбутнє та детальна історія транзакцій. Тому під час проєктування системи критично важливо обов'язково врахувати принципи локального збереження і синхронізації даних.

Метою випускної кваліфікаційної роботи є розробка мобільного застосунку особистого фінансового менеджера, який забезпечує облік усіх вищеперерахованих операцій у зручній для користувача формі.

Для досягнення поставленої мети потрібно виконати аналіз предметної області подібних застосунків, визначити основні задачі користувача, дослідити аналоги на ринку, сформулювати функціональні та нефункціональні вимоги, спроектувати структуру даних і архітектуру застосунку, розробити інтерфейс користувача і провести тестування основних сценаріїв роботи.

Об'єктом дослідження є облік фінансів користувача засобами мобільного програмного забезпечення.

					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						7

Предметом дослідження є методи та засоби розробки Android-застосунку для ведення особистих фінансів з локальним збереженням даних і опціональною хмарною синхронізацією.

У роботі використано методи системного аналізу, об'єктно-орієнтованого проєктування, моделювання структури даних, компонентного проєктування інтерфейсу, функціонального тестування та аналізу нефункціональних вимог.

Практичним значенням є розробка готового мобільного застосунку для фінансового обліку. Користувач отримує змогу вести точний облік усіх своїх джерел коштів та фіксувати нові доходи чи витрати. Програма дозволить переглядати детальну історію та тримати під контролем власні бюджети. Додатково тут можна успішно планувати майбутні фінансові цілі керувати повторюваними платежами та записувати важливі справи. А перевагою застосунку виступає повністю локальне збереження зібраної інформації. Написаний програмний код має потенційно бути готовий до розширення функціоналу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Постановка задачі

Управління особистими фінансами не обмежується лише переглядом залишку на рахунку в банківському застосунку. Сьогодні це норма для користувача мати декілька джерел зберігання грошей, як банки так і численні електронні гаманці або готівки. Користувачу важливо знати звідки надходять кошти і куди вони витрачаються. Для цього потрібен інструмент, що збиратиме усю фінансову інформацію в одному, зручному для перегляду місці.

Базовим джерелом даних у системі має виступати рахунок. Це сутність, яка може бути готівкою, банківською карткою чи електронним гаманцем, з власним типом валюти. Навколо рахунку мають будуватися усі подальші функції, бо кожна операція по суті залежить від джерела коштів, чим і є рахунок. Невдала розробка цього базового елемента може швидко викликати проблеми із роботоспроможністю всього застосунку.

Будь-яка транзакція в додатку має розглядатись як ключова дія, що змінює поточний фінансовий стан рахунків. Кожна операція має містити набір обов'язкових атрибутів, як суму, валюту, дату, прив'язку до конкретного рахунку, тип та опис. Логіка створення та обробки транзакцій, для ефективності, має бути винесена в окремі класи.

Окрему увагу під час розробки слід приділити категоризації операцій. Саме поділ на категорії перетворює хаотичну історію покупок на зрозумілу аналітику. Такий підхід дозволить уникнути надмірного ускладнення інтерфейсу і залишить систему достатньо глибокою для повноцінного використання. Застосунок має пропонувати базовий набір найпопулярніших категорій, але залишить користувачу свободу створювати власні.

Окрім фіксації факту витрат, система має допомагати їх контролювати. Для цього в нагоді стане функція створення бюджетів, яка діятиме як обмежувач для вибраних категорій на заданий період. Користувач зможе вказати ліміт, а

						ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
							9

програма відображатиме прогрес його вичерпання і своєчасно попереджатиме про наближення до встановленої межі.

Планування майбутніх покупок чи формування резервного фонду теж є вагомою частиною фінансового життя. Для цього можна реалізувати функціонал «Цілей». Користувач, вказуючи бажану суму та дедлайн, зможе бачити прогрес накопичення та поточний темп відкладання коштів запланованим термінам.

Також серед користувачів дуже розповсюджені регулярні платежі, однакові суми з визначеною періодичністю, наприклад, оплата комунальних послуг чи підписок. Для цього стане в нагоді функціонал періодичних платежів, які автоматично списуватимуть визначену суму з обраного рахунку в потрібний день. В найкращому випадку тут можна буде використати звичайну транзакцію з елементом відкладеності за допомогою календаря.

Оскільки застосунок працює з чутливою інформацією, встає доречне питання щодо безпеки. Усі записи транзакцій мають бути надійно сховані від третіх осіб. Саме тому найпершим кроком у розробці стало додавання звичайного стартового пароля. Звісно ж чотири цифри не замінять серйозного банківського шифрування. Проте цього цілком достатньо щоб захистити програму від випадкового підглядання.

Іншим критично важливим моментом виявилася повна незалежність програми від інтернету. Сьогодні багато розробників впроваджують примусову реєстрацією ще до того як людина взагалі зрозуміє чи подобається їй продукт. У цьому випадку вирішено принципово відмовитися від такого підходу. Вся ваша статистика зберігається виключно в пам'яті самого смартфона і програма чудово працює повністю без мережі. Резервне копіювання у хмару залишили просто як зручну додаткову опцію для тих хто сам забажає перестрахуватися і добровільно створить свій профіль.

Якщо зібрати всі ці вимоги до купи, головна мета всієї кваліфікаційної роботи стає прозорою. Нам потрібно створити такий кишеньковий фінансовий інструмент який би допомагав планувати бюджет і аналізувати витрати без

зайвого головного болю. Програма розроблялася саме для комфортного щоденного використання без перевантажених налаштувань.

1.2 Аналіз аналогів програмного продукту

На ринку мобільних застосунків є низка готових рішень для персонального фінансового менеджменту, від майже непомітних продуктів до платформ з багатомільйонною аудиторією. З метою проведення об'єктивного дослідження, для порівняльного аналізу було відібрано декілька найбільш популярних програм цього сегмента, що функціонують на базі операційної системи Android. Більшість з них, хоч і об'єднує спільний характер роботи, що базується на транзакціях та менеджменті рахунків, але майже всі можуть пропонувати деякий унікальний функціонал.

Застосунок Wallet (розробник “BudgetBakers”) є одним з найпопулярніших прикладів на мобільному ринку [1]. Головною перевагою цього рішення є унікальний функціонал, зокрема, підтримка просунутого опису транзакцій, навіть з геолокаціями відвіданих закладів. Також на вибір доступні різні фінансові інструменти, на кшталт запланованих платежів, боргів та перегляду курсу валют.

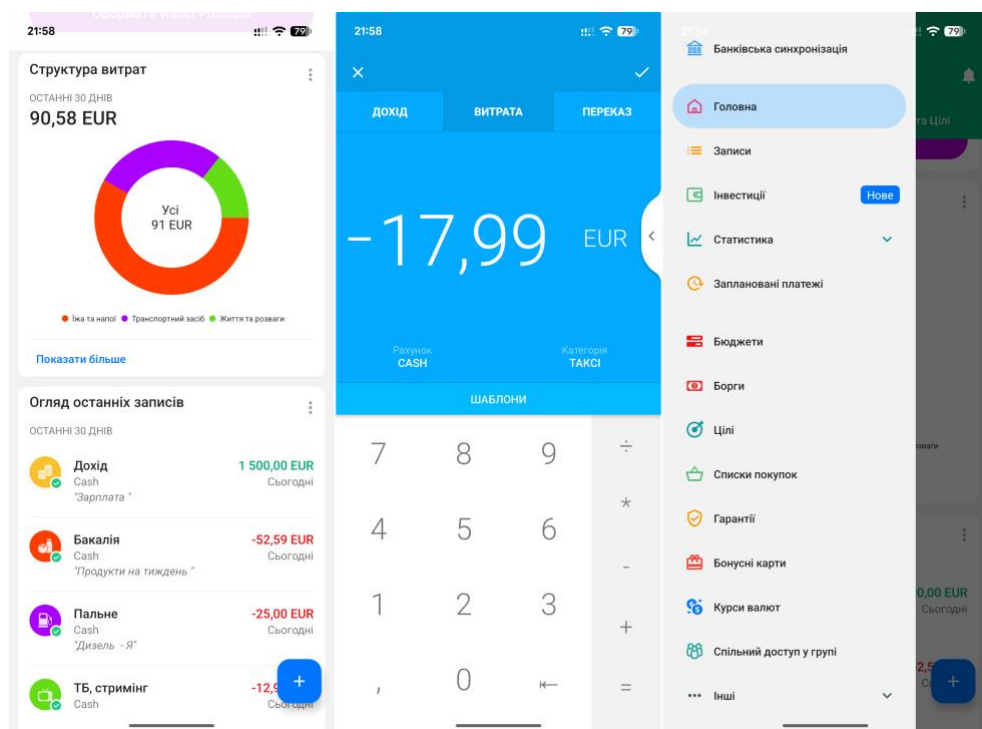


Рис. 1.1. Користувацький інтерфейс Wallet

Wallet є досить просунутою платформою, яка навіть дає змогу синхронізувати акаунт сторонніх банків та електронних гаманців для збору інформації по витратах та надходженнях. Проте деякі окремі функції доступні лише через платну модель використання, що не завадило застосунку набрати понад 10 мільйонів завантажень та хороші відгуки.

Мінімалізм Monefy спрощує інтеракцію з користувачем, чіткі та інтуїтивно зрозумілі шляхи проведення операцій. Проте таке надмірне спрощення позбавляє деталізованої інформації про транзакції, залишаючи лише назви

Money Manager Expense & Budget (розробник Realbyte) [3]. Як і попередній приклад, цей застосунок орієнтований на швидке додавання та перегляд транзакцій. Вся історія витрат зручно розташована на головному меню, а налаштування пропонують широку кастомізацію навіть дрібних деталей, як підкатегорії, регіональні особливості календаря та вибір кольорової палітри користувацького інтерфейсу. Останній, як показано на знімках екрану смартфона (рис.1.3), також досить мінімалістичний та інтуїтивно зрозумілий.

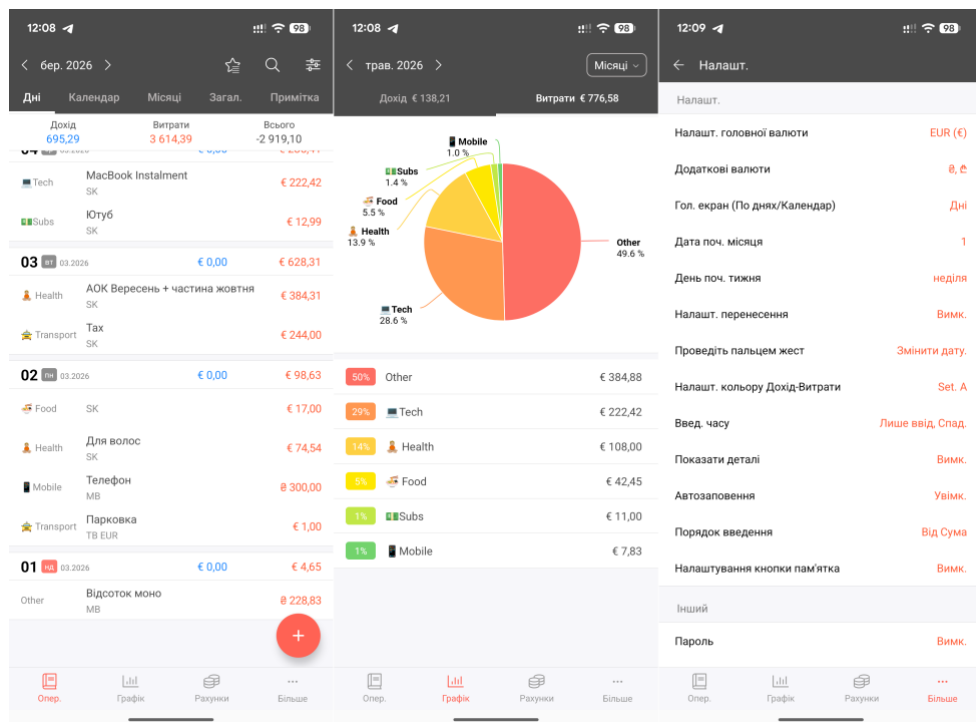
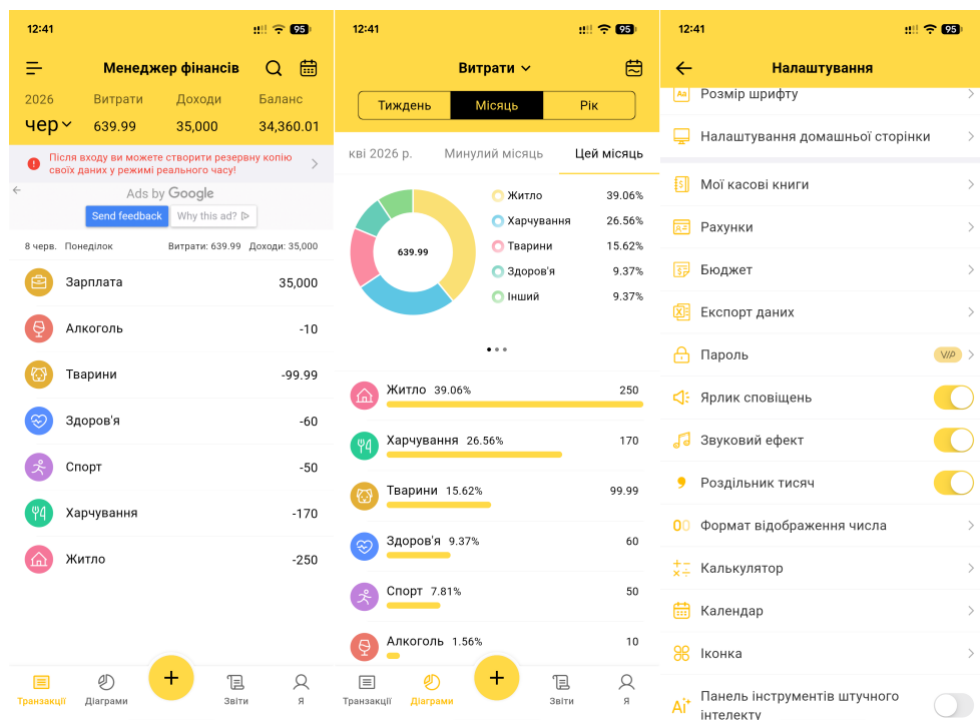


Рис. 1.3. Користувачський інтерфейс Money Manager Expense & Budget

Ще популярний фінансовий застосунок це Money Tracker-Expense & Budget (від Horoscope365) [4]. У програми присутній типове розташування елементів, як і у деяких попередніх прикладів з аналогів, коли основний функціонал доступний з головного екрану. Кругові діаграми, категорії, вибір часового проміжку, із унікального застосунок пропонує інтеграцію інструментів штучного інтелекту для аналізу витрат та більш детальне налаштування користувацького інтерфейсу (рис. 1.4), який також доволі зрозумілий та легкий на вигляд.



Серед іншого варто відзначити можливість синхронізації даних з хмарою, але ця функція доступна лише при авторизації. Також низка можливостей захищені платною підпискою, а на головному екрані є поле для рекламних

банерів, яке неможливо прибрати, що є одним з методів отримання прибутку навіть з непреміальних користувачів. Налаштування кольорової палітри та навіть іконок категорій є непоганим доповненням до основного функціоналу. Окрему увагу варто звернути на маскота застосунку (рис. 1.5), графічне зображення з котиком.



Рис. 1.5. Маскот на іконці застосунку Money Tracker-Expense & Budget

Використання маскота в мобільному застосунку може виконувати не лише декоративну, а й комунікативну функцію. Антропоморфні персонажі допомагають зробити цифровий продукт більш дружнім для користувача, підвищують емоційне залучення та сприяють кращій впізнаваності бренду [5]. Тому наявність маскота-талісмана у фінансовому застосунку може пом'якшити сприйняття теми грошей, яка для частини користувачів асоціюється зі складністю або напруженням. Опираючись на це, можна на цьому етапі й затвердити назву застосунку, аби виділяти його на фоні інших аналогів.

Назва базуватиметься також й на маскоті-талісмані, який буде кішечкою породи Каліко з характерним (білий, чорний, коричневий або помаранчевий) забарвленням, що може й стати ключовими кольорами застосунку, на яких буде акцентуватись дизайн. Назвою стане «Finna: Financial Manager» або ж «Фінна: Фінансовий Менеджер» на честь маскота кішки Фінни, що є скороченням або каламбуром від слова «Фінанси». Дизайн Фінни буде розроблений згодом.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

									Арк.
									15

Таблиця 1.1

Порівняння мобільних застосунків обліку фінансів

Характеристика	Wallet	Monefy	Money Manager	Money Tracker
Облік доходів і витрат	+	+	+	+
Керування декількома рахунками	+	+	+	+
Підтримка різних валют	+	+	+	+
Категоризація операцій	+	+	+	+
Можливість створення власних категорій	+	-	+	+
Детальний опис транзакцій	+	-	+-	+-
Пряме додавання операції з головного екрана	+-	+	+	+
Перегляд історії операцій	+	+	+	+
Фільтрація за періодом	+	+	+	+
Бюджети або обмеження витрат	+	-	-	+
Заплановані або регулярні платежі	+	-	-	+-
Фінансові цілі	+-	-	-	-
Синхронізація з хмарою	+	+-	+	+
Синхронізація з банками	+	-	-	-
Робота без обов'язкової авторизації	+-	+	+	+-
Налаштування зовнішнього вигляду	+-	-	+	+
Наявність маскота або вираженого візуального персонажа	-	-	-	+
Частина функцій доступна лише за підпискою	+	+	+-	+

Спираючись на ці спостереження, було сформовано перелік вдалих архітектурних та інтерфейсних рішень, які об'єктивно варто імплементувати у проєкті, як механізм швидкого створення транзакцій, паралельна підтримка кількох рахунків, а також інструменти для планування бюджетів. Для зручного аналізу витрат знадобляться детальна історія з фільтрацією за періодами та класичні кругові діаграми. Окрім суто технічних можливостей, критичну роль відіграє загальна філософія використання. Застосунок має залишатися дружнім до новачка та обов'язково підтримувати локальний режим роботи, адже перевантаження екранів зайвими налаштуваннями або вимога постійного підключення до мережі гарантовано відштовхнуть користувача від щоденного ведення обліку.

Фактично, ідеальна концепція майбутнього продукту полягає у синергії найкращих ринкових практик. Принципово важливо уникнути головних недоліків конкурентів, як то агресивного обмеження базових функцій платною моделлю або примусової авторизації на старті. Саме таке бачення балансу між функціональністю та зручністю для пересічного користувача може лягти в основу формування остаточних системних вимог до розроблюваного програмного продукту.

1.3 Формування вимог до майбутнього мобільного застосунку

Зрозумівши загальну ситуацію на ринку та вивчивши найпопулярніші аналоги складається чітке уявлення про майбутній програмний продукт. На цьому етапі проектування недостатньо просто вигадати випадковий набір бажаних опцій. Набагато важливіше продумати глибоку логіку взаємодії людини з системою. Розробка вимог до програми спирається саме на попередній аналіз відомих рішень. Проте головна мета полягала не у сліпому копіюванні чужих ідей. Потрібно відібрати найбільш життєздатні інструменти які реально втілити в рамках цієї кваліфікаційної роботи.

Традиційно всі побажання до майбутньої системи діляться на дві великі групи. Перша описує суто практичний бік застосунку. Сюди потрапляє створення гаманців проведення щоденних транзакцій планування бюджетів та налаштування платежів які постійно повторюються.

Друга категорія стосується якості самої роботи. Йдеться про стабільність загальну швидкодію незалежність від підключення до інтернету надійність збереження даних та комфорт графічного інтерфейсу.

Фундаментом усього фінансового обліку виступає концепція рахунків. Користувач не може обмежуватися лише одним загальним балансом бо в реальному житті гроші завжди розподілені між готівкою банківськими картками та різноманітними депозитами. Програма повинна дозволяти створювати необмежену кількість таких джерел фінансування задаючи кожному власну назву стартову суму та валюту. Додатковий поділ за типами гаманців допомагає

людині краще орієнтуватися у власних активах без ускладнення самого процесу фіксації покупок.

Головною рутинною дією завжди залишатиметься запис нових транзакцій. Абсолютно всі зміни балансу відбуваються через доходи витрати або внутрішні перекази між власними рахунками. Кожен такий запис вимагає зазначення точної суми вибору відповідної категорії та дати.

Дуже важливо залишити користувачу можливість вводити час вручну адже люди вкрай рідко записують свої витрати прямо біля каси супермаркету. Іноді виникає потреба змінити вже збережену операцію. У такому випадку система зобов'язана не просто переписати текст у базі даних а повністю перерахувати загальний фінансовий стан з урахуванням нових ввідних значень. Щоб не загубити найважливіші платежі серед тисяч інших дрібних чеків передбачена можливість додавання окремих записів до обраних.

Сучасне життя вимагає постійної взаємодії з різними іноземними валютами. Якщо людина купує щось за долари чи євро застосунок має автоматично перевести цю суму в основну валюту гаманця використовуючи актуальні курси Національного банку України. Для забезпечення безперебійної роботи ці курси обов'язково кешуються в пам'яті смартфона.

Щоб витрати не перетворювалися на суцільний незрозумілий потік цифр їх необхідно групувати за категоріями. Програма одразу пропонує базовий набір популярних напрямків але зовсім не забороняє створювати власні унікальні варіанти. При цьому назви стандартних груп автоматично адаптуються під обрану мову системи що повністю відповідає офіційним правилам розробки під Android.

Знайти стару покупку завжди допоможе зручна історія операцій з розширеними фільтрами. А для візуального розуміння куди саме зникають гроші ідеально підходить класична кругова діаграма яка давно стала стандартом у подібних програмах. Крім аналізу минулого застосунок має допомагати з плануванням майбутнього. Механізм бюджетів дозволяє встановити жорсткий ліміт на обрану категорію і щодня слідкувати за залишком коштів. На противагу

бюджетам існують фінансові цілі які допомагають поступово накопичувати необхідну суму на омріяну річ чи далеку подорож.

Рутинні списання за домашній інтернет чи підписки на різні цифрові сервіси можна легко автоматизувати. Користувач один раз створює шаблон а спеціальний системний компонент WorkManager непомітно проводить потрібну операцію у фоновому режимі у визначений день. Додатково програма пропонує своєрідний список справ пов'язаних виключно з фінансами. Там можна записати нагадування про необхідність перевірити тарифний план чи вчасно погасити кредит.

Щодо технічного боку проєкту найжорсткішою вимогою стала стовідсоткова автономність. Застосунок має ідеально працювати в умовах повної відсутності мережі зберігаючи всі записи локально завдяки потужному інструменту Room. Резервне копіювання у хмару через сервіси Firebase залишається виключно добровільною опцією для тих хто бажає перестрахуватися. Оскільки йдеться про приватну інформацію доступ до програми надійно захищається цифровим паролем. Жодні облікові дані ніде не зберігаються у відкритому вигляді що гарантує відповідність сучасним стандартам мобільної безпеки.

Графічний інтерфейс програми розроблявся з думкою про максимальну простоту. Усі ключові елементи спираються на правила системи Material Design 3 і розташовані так щоб до них було зручно дотягнутися одним пальцем. Внутрішня логіка одразу поділена на незалежні шари для легкого масштабування та додавання нових функцій у майбутньому. Особливу увагу приділили так званім порожнім станам. Якщо користувач ще не додав жодного рахунку чи витрати програма акуратно підкаже з чого саме варто почати замість того щоб лякати порожніми екранами. В результаті отримуємо повноцінний надійний інструмент який закриває всі ключові потреби в щоденному обліку власних грошей.

Всі ці та додаткові вимоги для зручності можна зібрати у таблицю (табл. 1.2).

					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						19

Таблиця 1.2

Основні вимоги до мобільного застосунку

Вимоги	Зміст вимоги
Функціональність	Підтримка рахунків, транзакцій, категорій, бюджетів, цілей, регулярних платежів і завдань
Зручність	Швидке додавання операції, зрозуміла навігація, простий інтерфейс без перевантаження
Автономність	Можливість працювати без постійного доступу до мережі та без обов'язкової авторизації
Безпека	PIN-захист, контроль доступу до локальних даних, використання Firebase Authentication для авторизації
Локалізація	Підтримка української та англійської мов
Масштабованість	Можливість додавання нових модулів без повної переробки архітектури

1.4 Обґрунтування вибору інструментальних засобів

Обґрунтування вибору технологічного стеку є одним із найважливіших етапів проєктування будь-якого програмного забезпечення. Саме на цій стадії закладається той інженерний фундамент від якого безпосередньо залежить довговічність та швидкість роботи майбутнього продукту. Для створення системи щоденного обліку персональних коштів найдоцільнішим рішенням виглядає орієнтація на мобільну платформу Android [6]. Ця операційна система впевнено утримує лідерські позиції на світовому ринку мобільних пристроїв та надає розробникам значно ширші можливості для старту.

Важливим економічним фактором є відсутність жорсткої прив'язки до спеціалізованого апаратного забезпечення на кшталт комп'ютерів Apple що є обов'язковою умовою для створення програм під iOS. Доступність та великий ціновий діапазон сумісних пристроїв суттєво розширюють потенційну аудиторію користувачів. Оскільки сучасний смартфон супроводжує людину протягом усього дня вона отримує можливість фіксувати фінансові операції безпосередньо в момент їх здійснення. Це усуває потребу тримати в пам'яті численні витрати або записувати їх на паперових носіях.

									Арк.
									20

Середовищем для написання коду було обрано програмний комплекс Android Studio від компанії Google [7]. Цей інструмент дозволяє розробнику проектувати починаючи від інтелектуального текстового редактора і закінчуючи інтегрованим віртуальним емулятором пристроїв.

Наявність вбудованого емулятора дозволяє проводити миттєве тестування інтерфейсу та логіки безпосередньо під час написання коду. Також, варто зазначити, що робота в цьому середовищі значно підвищує загальну продуктивність праці завдяки системі динамічного аналізу коду яка моментально вказує на потенційні помилки, як підсвічування деяких функцій, що вже втратили чи скоро втратять свою актуальність. Середовище забезпечує зручне керування мовними ресурсами та конфігураціями збірки програми що вкрай необхідно для створення якісного дипломного проекту.

В ролі основної мови програмування виступає Kotlin яка на сьогодні є офіційним стандартом для створення мобільних додатків у цій екосистемі [8]. Вона демонструє чудову інтеграцію з усіма сучасними системними бібліотеками. Завдяки своєму синтаксису Kotlin дозволяє створювати лаконічніші та безпечніші конструкції порівняно із класичною мовою Java. Це мінімізує появу багів пов'язаних із неочікуваними порожніми значеннями або помилками типізації.

Оскільки у фінансовому менеджері постійно обробляється велика кількість структурних даних про рахунки та ліміти тут корисними виявилися спеціальні класи даних. Використання концепції data class дозволяє розробнику буквально кількома рядками описувати складні інформаційні моделі та автоматично отримувати готові методи для їх копіювання [9]. Такий підхід стає незамінним коли система оновлює поточний баланс або коли користувач редагує деталі старої транзакції адже копіювання об'єктів із модифікацією параметрів захищає систему від критичних збоїв у пам'яті.

Для побудови користувацького інтерфейсу було застосовано прогресивний інструментарій Jetpack Compose [10]. Проєкт повністю відмовляється від класичної та дещо застарілої XML верстки на користь сучасного декларативного

підходу. Головна перевага цієї технології полягає в тому що графічні елементи екрана стають нерозривно пов'язаними із поточним станом даних. Коли в локальній базі відбувається будь-яка зміна інтерфейс миттєво та самостійно перемальовується відображаючи актуальні цифри.

Візуальна концепція додатку базується на рекомендаціях дизайну Material Design 3 [11]. Специфіка фінансових програм вимагає створення максимально спокійного та чистого простору який би не відволікав людину від аналізу грошових потоків. Акуратні інформаційні картки плавні заокруглені форми кнопок та чіткі поля для введення інформації створюють передбачувану та інтуїтивно зрозумілу структуру для щоденної взаємодії.

Для забезпечення роботи в офлайн режимі всі користувацькі записи зберігаються безпосередньо на самому мобільному пристрої. Інструментом для управління локальною базою даних було обрано бібліотеку Room яка входить до складу офіційного пакету Jetpack [12]. Вона створює рівень абстракції над реляційною базою SQLite звільняючи розробника від ручного написання довгих SQL запитів, що гарантує високу швидкість обробки інформації. Користувач може вільно керувати своїми фінансами навіть під час тривалої відсутності підключення до глобальної мережі інтернет.

Внутрішня архітектура коду має бути побудована із чітким розподілом на незалежні логічні шари. Графічні компоненти відповідають виключно за рендеринг елементів на екрані тоді як окремі моделі архітектури ViewModel повністю беруть на себе управління станами додатку та бізнес логікою. Разом із цим тривале збереження інформації лише в пам'яті одного смартфона несе певні ризики через можливість втрати чи механічного пошкодження девайсу. З метою нівелювання цієї проблеми в застосунку було реалізовано додаткову можливість синхронізації із хмарною NoSQL базою даних Cloud Firestore на базі платформи Firebase [13].

Проте в межах даного проєкту авторизація через хмару має суто рекомендаційний характер [14]. Людина може повноцінно користуватися всіма інструментами локально без введення електронної пошти чи прив'язки облікових

записів Google що підвищує рівень приватності продукту. Хмарне сховище задіюється лише тоді коли користувач сам свідомо вирішує створити резервну копію для безпеки даних.

Ще важливою задачею стала організація роботи із різними валютами. Оскільки сучасні користувачі часто здійснюють операції в іноземних грошових одиницях програма отримала функцію автоматичного звернення до серверів Національного банку України. Через парсинг застосунок завантажує актуальні курси у форматі JSON та виконує точний перерахунок транзакцій.

Для автоматизації періодичних витрат на кшталт щомісячних комунальних платежів або регулярних підписок доцільно інтегрувати планувальник фонових завдань WorkManager. Цей компонент автономно виконує перевірки у визначений час та може самостійно виконувати дії у фоновому режимі без залучення уваги користувача.

В ролі мінімальної версії операційної системи було обрано Android 12 що дозволяє охопити переважну більшість актуальних мобільних пристроїв на ринку та повноцінно використовувати сучасні можливості платформи [15]. Створений технологічний стек дозволив отримати збалансований та масштабований програмний продукт який повністю вирішує задачі персонального фінансового менеджменту в сучасних умовах.

Висновки до першого розділу

У першому розділі було детально вивчено сферу управління особистими фінансами та визначено головні завдання майбутньої програми. Дослідження підтвердило неможливість ефективного контролю грошей лише через перегляд одного банківського рахунку. Сучасна людина часто розподіляє свої фінанси між різними картками та готівкою. Тому новий застосунок має створити єдине місце для зручного обліку всіх рахунків та витрат. Базовим елементом усієї системи виступає саме рахунок. До нього завжди прив'язані всі транзакції та поточний баланс. Додатково програма повинна підтримувати фіксацію доходів, переказів, бюджетів, цілей і регулярних платежів.

Огляд конкурентів на ринку показав багато спільного між готовими додатками. Деякі розробники закривають найкорисніші інструменти платною підпискою. Враховуючи чужий досвід вдалося скласти чіткий перелік вимог для власної розробки. В основі програми буде лежати повноцінна робота з різними рахунками та транзакціями. Для зручності туди обов'язково увійдуть бюджети і гнучкі категорії. Крім того враховано правила щодо стабільності та комфорту використання. Головним принципом стане локальне збереження всієї інформації прямо на телефоні. Синхронізація через хмару залишається тільки додатковою опцією за бажанням людини. Також додаток одразу отримає повноцінну підтримку української та англійської мов.

Для технічної реалізації проєкту було обрано операційну систему Android та сучасну мову програмування Kotlin. Головним середовищем розробки виступає Android Studio. Побудова графічного інтерфейсу буде виконуватися за допомогою Jetpack Compose та набору правил Material Design 3. Локальна база даних працюватиме на основі надійної бібліотеки Room.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Визначення варіантів використання та структури користувацьких сценаріїв

Коли всі основні вимоги до майбутнього менеджера фінансів вже сформовані настає час розібратися з його внутрішньою функціональною структурою. Тут визначаємо головні сценарії поведінки та логіку взаємодії людини з готовим продуктом, їх можна графічно описати у вигляді UML-діаграми сценаріїв використання застосунку (рис. 2.1). Особливість таких програм полягає у глибокій пов'язаності всіх елементів між собою. Звичайний гаманець не має жодного сенсу без транзакцій а самі фінансові операції безпосередньо формують загальну історію та постійно змінюють поточний баланс. Водночас створені категорії використовуються для глибокої аналітики і планування бюджетів тоді як налаштовані регулярні платежі здатні автоматично генерувати нові чеки витрат.

Єдиним актором у розробленій системі виступає безпосередній власник смартфона. Жодних складних ієрархій з адміністраторами чи сторонніми спостерігачами. Продукт створюється суто для індивідуального користування де одна людина повністю керує всіма процесами. Користувач самостійно відкриває рахунки додає щоденні покупки планує майбутні цілі та за власним бажанням налаштовує параметри приватності чи хмарного збереження.

Робота з грошима завжди починається зі створення першого фінансового джерела. Без цього кроку система просто не дозволить провести жодну операцію адже кожна витрачена копійка має бути прив'язана до конкретного місця. Це може бути готівка у кишені звичайна зарплатна картка чи навіть банківський депозит. Під час створення такого гаманця обов'язково вказується його назва тип робоча валюта та початковий залишок коштів. Одразу після збереження новий запис з'являється у загальному списку і стає готовим до використання.

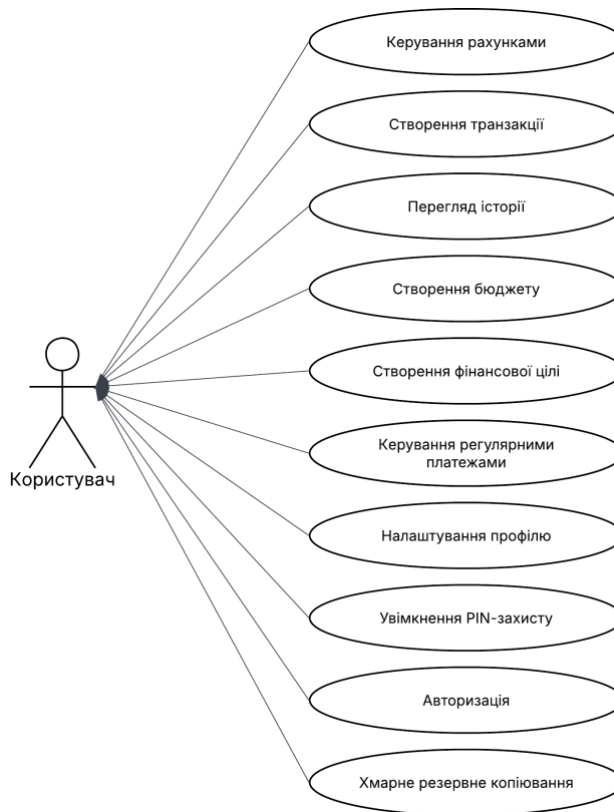


Рис. 2.1. Діаграма сценаріїв використання мобільного застосунку

Цікавим з технічної точки зору є процес редагування вже збережених записів. Не можна просто змінити цифру в базі даних як звичайний текст. Якщо людина вирішила виправити суму чека чи змінити категорію система зобов'язана спочатку повністю скасувати вплив старої транзакції на баланс. Лише після цього застосовуються оновлені значення що надійно захищає загальну статистику від поступового накопичення прихованих розрахункових помилок.

Також, наприклад, аби не тримати в голові постійні рутинні витрати на кшталт оплати за інтернет чи оренду житла існує блок регулярних платежів. Тут достатньо один раз вказати суму з періодичністю і за потреби цей автоматичний платіж завжди можна тимчасово призупинити без повного видалення. Роль своєрідного кишенькового щоденника виконують дрібні фінансові справи де зручно записувати нагадування про необхідність перевірити тариф чи вчасно погасити кредит.

Головна особливість полягає у тісному взаємозв'язку всіх її функцій. Рахунок вважається базовим фінансовим об'єктом у системі. Кожна нова транзакція відразу змінює його поточний баланс. Обрана категорія потрібна для

зручного аналізу витрат. Бюджет встановлює контроль за грошима у межах певної категорії. Фінансова ціль допомагає правильно планувати майбутні накопичення. Саме тому під час проектування архітектури важливо описати не тільки окремі вікна інтерфейсу.

Таблиця 2.1

Основні сценарії використання застосунку

№	Сценарій використання	Короткий опис	Очікуваний результат
1	Створення рахунку	Користувач додає рахунок із назвою, типом, валютою та початковим балансом	У системі з'являється нове джерело коштів, яке може використовуватися у транзакціях
2	Додавання транзакції	Користувач обирає тип операції, рахунок, категорію, валюту, суму, дату та за потреби опис. За потреби він може редагувати транзакцію після її створення	Створюється фінансовий запис, а баланс відповідного рахунку оновлюється
3	Перегляд історії	Користувач переглядає операції, згруповані за датами, і застосовує фільтри	Відображається потрібна частина фінансової історії за рахунком, категорією або періодом
4	Створення бюджету	Користувач встановлює ліміт витрат для обраної категорії	Застосунок показує прогрес використання бюджету та попереджає про наближення до межі
5	Створення фінансової цілі	Користувач задає бажану суму, валюту, дедлайн і поточний прогрес	Система дає змогу відстежувати накопичення та оцінювати наближення до цілі
6	Керування регулярними платежами	Користувач створює платіж, який повторюється у визначений день	Застосунок може автоматично створити відповідну операцію або нагадати про платіж
7	Керування фінансовими справами	Користувач створює фінансову задачу з дедлайном і пріоритетом	Фінансові дії не губляться і можуть бути виконані в потрібний час
8	Захист PIN-кодом	Користувач вмикає або змінює PIN-код у профілі. Програма просить PIN-код при запуску	Під час запуску застосунок запитує код доступу до локальних даних

продовж. табл. 2.1

№	Сценарій використання	Короткий опис	Очікуваний результат
9	Авторизація	Користувач може авторизуватися за допомогою Google акаунту	Користувач авторизований та має змогу зберігати інформацію у хмарі
10	Хмарне резервне копіювання	Після авторизації користувач створює або відновлює хмарну копію даних	Фінансові записи можна зберегти або відновити за потреби

У таблиці 2.1 зібрані ключові сценарії використання створеної системи. Описані саме ті операції, які потрібні людині для щоденного ведення фінансового обліку. Базові функції працюють одразу після встановлення програми і не потребують створення облікового запису. Разом з тим у додатку передбачені додаткові інструменти. Наприклад збереження резервної копії бази даних у хмарному сховищі стає доступним тільки після успішної авторизації.

2.2 Проєктування архітектури мобільного застосунку

Архітектура застосунку побудована за модульним принципом із чітким розділенням відповідальності між шарами інтерфейсу, збереження стану, логіки роботи та даних. Такий підхід відповідає рекомендаціям Android щодо розподілу логіки між UI layer, ViewModel та data layer [16]. У ній одночасно працюють вікна введення інформації, різноманітні списки, фільтри, локальна база та фонові задачі. Додатково сюди входить авторизація та необов'язкова синхронізація. Збереження чітких меж між різними частинами коду суттєво зменшує загальну складність проєкту і робить його набагато зрозумілішим для майбутнього оновлення.

На рівні графічного інтерфейсу працює технологія Jetpack Compose. Усі екрани програми описані як функції компонування, що відповідає підходу UI layer в архітектурі Android [17]. Вони постійно відображають поточний стан системи і миттєво реагують на дії користувача. Кожен складний екран має власний клас ViewModel. Він повністю відповідає за підготовку інформації, правильну обробку подій та формування візуального стану [18]. За роботу з

джерелами даних відповідає окремий data layer, який відокремлює логіку отримання даних від інтерфейсу [19].

Для прикладу головний екран гаманця успішно отримує список рахунків та останні транзакції разом з підсумками доходів і витрат. Екран історії своєю чергою працює з фільтрами та сортуванням за датами для правильної побудови кругової діаграми та показу статистики.

Основна навігація програми організована всередині головного компонента FinnaApp. Він повністю відповідає за плавне перемикання між вікнами програми, як зображено на схемі навігації (рис. 2.2). До цього переліку входять гаманець, рахунки, сторінка створення транзакції, історія, профіль користувача, бюджети, фінансові цілі та регулярні платежі. Окремо реалізовано стартову перевірку ПІН-коду. Якщо захист увімкнений, то перед переходом до меню користувач обов'язково вводить чотири цифри коду. Якщо такий пароль не налаштований, на екрані просто відображається коротка стартова заставка.



Рис. 2.2. Схема навігації застосунку

Робота зі збереженням та створенням усіх головних об'єктів у програмі побудована через класи репозиторіїв. Вони працюють як зручний проміжний шар між компонентами ViewModel та самою базою даних. Кожен такий репозиторій має свою чітку зону відповідальності.

- Клас AccountRepository повністю керує інформацією про рахунки.

- За правильну роботу з операціями відповідає TransactionRepository.
- Для керування бюджетами та цілями створено окремі компоненти BudgetRepository і GoalRepository.
- Також існують спеціальні репозиторії для роботи з регулярними платежами та дрібними фінансовими справами, це класи RecurringPaymentRepository та FinancialTaskRepository.
- Водночас усі загальні параметри самої програми зберігаються за допомогою AppSettingsRepository.

Крім репозиторіїв важливу роль у системі відіграють спеціальні сервісні класи. Компонент `TransactionService` бере на себе створення та редагування всіх грошових операцій. Це зроблено навмисно для правильного та безпечного перерахунку загального балансу рахунків під час кожної зміни. Роботу зі створення та відновлення резервних копій у хмарі виконує `CloudBackupService`.

Загальну відповідність модулів коду їхньому призначенню наведено в табл. 2.2.

Таблиця 2.2

Основні модулі застосунку

Модуль	Призначення	Пов'язаний функціонал
FinnaApp	Організація основної навігації та перемикання між екранами	Гаманець, рахунки, транзакції, історія, профіль, бюджети, цілі, платежі, задачі
MainActivity	Точка входу в Android-застосунок і підключення теми інтерфейсу	Запуск застосунку, застосування FinnaTheme
AppContainer	Створення бази даних, репозиторіїв і сервісів застосунку	Ініціалізація залежностей без складного DI-фреймворку
AppDatabase	Опис локальної бази даних Room і підключення DAO	Збереження рахунків, транзакцій, бюджетів, цілей, платежів і задач
TransactionService	Обробка створення та редагування фінансових операцій	Перерахунок балансу раунків після доходів, витрат і переказів
PinSecurityManager	Збереження стану PIN-захисту та перевірка введеного коду	Стартовий PIN-екран і захист локальних даних

Модуль	Призначення	Пов'язаний функціонал
CloudBackupService	Резервне копіювання та відновлення даних через хмарне середовище	Опціональна синхронізація після авторизації
RecurringPaymentWorker	Фонова обробка регулярних платежів	Автоматичне створення операцій за визначеним графіком.
CategoryUi	Локалізоване відображення назв та іконок категорій	Категорії у транзакціях, бюджетах, історії та платежах

Запропонована архітектура свідомо уникла надмірної складності. При цьому вона дуже чітко розподіляє зони відповідальності між усіма системними компонентами. Цей аспект є вкрай важливим для формату дипломного проєкту. Розроблений продукт має бути не просто функціональним на практиці. Його внутрішня логіка повинна легко пояснюватися та демонструватися під час публічного захисту перед комісією. Обрана структура чудово закриває цю потребу. Вона дає змогу наочно показати конкретні місця збереження даних та шляхи їхньої передачі до користувацького інтерфейсу. Також вона чітко виділяє ті класи коду, які беруть на себе виконання ключових фінансових операцій.

2.3. Проєктування структури даних та локальної бази даних

Головним місцем для надійного збереження всієї інформації у програмі виступає локальна база даних Room. Ця бібліотека надає рівень абстракції над SQLite та підходить для локального збереження структурованих даних у Android-застосунках [20]. Вона потрібна для забезпечення стабільної автономної роботи пристрою. Завдяки цьому користувач може легко створювати нові рахунки або записувати операції та бюджети навіть без постійного доступу до мережі Інтернет.

Вся внутрішня структура даних розробляється навколо кількох головних сутностей. До цього списку увійшли класи Account та Category разом із Transaction і Budget. Також сюди відносяться Goal та RecurringPayment а також FinancialTask і AppSettings.

									Арк.
									31

Кожен із цих компонентів чітко відображає конкретний елемент обраної предметної області. Наприклад сутність Account повністю описує поточне джерело грошей користувача. Клас Transaction бере на себе фіксацію кожної окремої фінансової операції. Сутність Budget встановлює необхідний ліміт для витрат. Клас Goal допомагає відстежувати прогрес накопичення на конкретну фінансову ціль. За роботу з повторюваними платежами відповідає компонент RecurringPayment. Дрібні фінансові справи та нагадування записуються через FinancialTask. Водночас усі загальні параметри програми контролюються за допомогою AppSettings.

Модель рахунку (табл. 2.3) зберігає базові дані про його назву та валюту разом із поточним балансом. Також обов'язково вказується конкретний тип рахунку. Це потрібно не просто для красивого відображення в меню програми. Такий поділ допомагає краще організувати всі фінансові джерела людини. На практиці готівка банківська картка чи рахунок мають зовсім різну логіку використання. Хоча під час математичних розрахунків усі вони працюють як звичайні джерела коштів.

Таблиця 2.3

Опис таблиці Account

Поле	Тип	Призначення
id	Long	Ідентифікатор
name	String	Назва рахунку
type	String	Тип рахунку
balanceMinorUnits	Long	Поточний баланс рахунку в мінімальних одиницях валюти
currency	String	Валюта рахунку
createdAt	Long	Дата створення запису у форматі

Категорії (табл. 2.4) необхідні для грамотного впорядкування всіх фінансових потоків. Одразу після встановлення програма пропонує стандартний набір популярних категорій. Але користувач не обмежений цим списком і може вільно створювати власні варіанти. Існують категорія як для витрат так і для доходів.

Опис таблиці Category

Поле	Тип	Призначення
id	Long	Ідентифікатор
name	String	Назва категорії
type	String	Тип категорії
iconName	String	Назва іконки, яка використовується для відображення категорії в інтерфейсі
isDefault	Boolean	Ознака стандартної категорії, створеної застосунком
createdAt	Long	Дата створення запису у форматі timestamp

Сутність транзакції (табл. 2.5) вважається однією з найважливіших деталей системи. Вона фіксує точну суму у вибраній валюті та дату проведення операції. Кожен такий запис також має жорстку прив'язку до конкретної категорії та рахунку. Користувач може додати власний текстовий опис або позначити операцію як вибрану.

Під час внутрішніх переказів логіка трохи змінюється і система враховує одразу два рахунки початковий та цільовий. Окрім того програма дозволяє вводити суму у валюті яка відрізняється від валюти самого рахунку. У таких випадках для правильної статистики обчислюється автоматичний еквівалент.

Варто зазначити один важливий нюанс щодо відображення назв. Текст на екрані не залежить напряду від збереженого значення в базі даних. Для системних категорій працює спеціальний механізм локалізації. Він автоматично перекладає назви українською або англійською мовами залежно від системних налаштувань смартфона.

Таблиця 2.5

Опис таблиці Transaction

Поле	Тип	Призначення
id	Long	Ідентифікатор
accountId	Long	Ідентифікатор рахунку, з якого списуються або на який зараховуються кошти

Поле	Тип	Призначення
accountAmountMinorUnits	Long	Еквівалент суми у валюті рахунку
date	Long	Дата й час виконання операції у форматі timestamp
description	String	Необов'язковий опис транзакції
isFavorite	Boolean	Ознака обраної транзакції
createdAt	Long	Дата створення запису у форматі timestamp
targetAccountId	Long?	Ідентифікатор цільового рахунку для операцій переказу
categoryId	Long?	Ідентифікатор категорії операції
type	String	Тип операції: дохід, витрата або переказ
amountMinorUnits	Long	Сума транзакції у мінімальних одиницях вибраної валюти
currency	String	Валюта суми, яку ввів користувач

Обмеження витрат описується через сутність бюджету (табл. 2.6). Там вказується бажаний грошовий ліміт для обраної категорії на конкретний період часу. Під час відкриття цього екрана програма самостійно перевіряє всі відповідні транзакції та вираховує залишок вільних коштів.

Таблиця 2.6

Опис таблиці Budget

Поле	Тип	Призначення
id	Long	Ідентифікатор
name	String	Назва бюджету
categoryId	Long?	Ідентифікатор категорії, для якої створено бюджет
limitMinorUnits	Long	Ліміт витрат у мінімальних одиницях валюти
currency	String	Валюта бюджету
month	Int	Місяць, для якого діє бюджет
year	Int	Рік, для якого діє бюджет
createdAt	Long	Дата створення запису у форматі timestamp

Свою чергою фінансова ціль (табл. 2.7) працює трохи інакше. Вона містить бажану фінальну суму та орієнтовну дату завершення накопичення. Завдяки цій моделі людина завжди бачить свій реальний прогрес і розуміє наскільки вона наблизилася до запланованої покупки.

Опис таблиці Goal

Поле	Тип	Призначення
id	Long	Ідентифікатор
title	String	Назва фінансової цілі
targetAmountMinorUnits	Long	Цільова сума в мінімальних одиницях валюти
currentAmountMinorUnits	Long	Поточна накопичена сума
currency	String	Валюта фінансової цілі
deadline	Long	Дата завершення цілі у форматі timestamp
createdAt	Long	Дата створення запису у форматі timestamp

2.4. Розробка основних модулів та алгоритмів роботи

Безпосередньо при написанні коду довелося одразу розмежовувати відповідальність різних частин програми. Усю архітектуру побудовано так, аби візуальні екрани не займалися складною математикою. За інтерфейсом залишилося тільки відображення кнопок та полів вводу. Водночас спеціальні моделі екранів формують поточний стан а репозиторії відповідають виключно за збереження у локальній базі. Усі найважчі операції де треба одночасно оновлювати кілька пов'язаних таблиць обережно винесені в окремі сервісні класи.

Виділені окремі блоки для роботи з рахунками, історією та бюджетами. Сюди ж додалися компоненти для регулярних платежів, дрібних справ і налаштувань профілю. Звичайно ж серцем усього цього механізму виступають саме рахунки та щоденні транзакції бо вони формують реальну картину фінансів людини.

Розпочинається все з управління джерелами коштів. Людина створює свій перший гаманець дає йому назву вибирає потрібну валюту і вписує початкову суму. Це може бути готівка чи звичайна картка. Хоча технічно всі ці сутності працюють абсолютно однаково візуальний поділ надзвичайно сильно допомагає користувачу орієнтуватися у власних грошах.

Абсолютно всі суми конвертуються і лежать у базі як цілі числа у мінімальних одиницях тобто в копійках. Такий крок дозволив повністю позбутися помилок округлення які часто вилазять при роботі з фінансами і з часом можуть серйозно спотворити фінальні суми.



Рис. 2.3. Схема створення або редагування транзакції

Центральне місце у роботі всієї програми займає механізм обробки транзакцій, схему якої можна побачити на рис. 2.3. Під час додавання нового фінансового запису користувач має обрати один із трьох доступних варіантів. Це може бути звичайний дохід витрата або внутрішній переказ. Для фіксації доходів та витрат достатньо вказати потрібний рахунок категорію суму у відповідній валюті та поточну дату. За бажанням людина може додати власний текстовий опис. У випадку переказу коштів програма додатково попросить вибрати

цільовий гаманець для зарахування грошей. Якщо валюта відрізняється, сума конвертується через курси НБУ, які витягуються за допомогою парсингу з сайту Національного банку [21].

Кожне успішне збереження операції миттєво оновлює загальний баланс. Процес редагування вже створеного запису працює трохи складніше. Спочатку система повністю скасовує попередній вплив цієї операції на гаманець і тільки після цього застосовує нові введені значення.

Редагування реалізовано обережніше, ніж просте перезаписування даних. Спочатку система скасовує вплив попередньої версії операції на баланс, а потім застосовує оновлену транзакцію. Це захищає фінансові показники від накопичення помилок після багаторазових змін. Обидва функції застосування нової транзакції на баланс рахунку або його видалення та редагування показані наглядно у лістингу коду нижче.

```
private suspend fun applyTransactionBalance(transaction: Transaction):
    OperationResult {
    return when (transaction.type) {
        TransactionType.INCOME -> applyIncome(transaction)
        TransactionType.EXPENSE -> applyExpense(transaction)
        TransactionType.TRANSFER -> applyTransfer(transaction)
    }
}

private suspend fun rollbackTransactionBalance(transaction: Transaction):
    OperationResult {
    return when (transaction.type) {
        TransactionType.INCOME -> rollbackIncome(transaction)
        TransactionType.EXPENSE -> rollbackExpense(transaction)
        TransactionType.TRANSFER -> rollbackTransfer(transaction)
    }
}
```

Для аналізу минулих фінансових дій розроблено сторінку історії. Вона виводить загальний список операцій з можливістю детальної фільтрації та групування за днями. Знайти потрібний запис можна за певним типом операції рахунком обраною категорією або вказаним періодом.

Механізм бюджетів тісно взаємодіє з даними про витрати у конкретних категоріях. Після створення нового ліміту система запам'ятовує доступну суму та часові рамки. Під час кожного відкриття цього екрана програма автоматично рахує фактичні витрати за вказаний період.

Інструмент фінансових цілей оперує даними про бажану кінцеву суму поточний стан накопичення та крайню дату. Користувач отримує можливість вільно створювати нові плани редагувати їх та слідкувати за власним прогресом. На екрані смартфона цей процес відображається у вигляді смуги завантаження або точного відсоткового показника.

Усі повторювані операції згруповані у спеціальному розділі регулярних платежів. Цей інструмент відрізняється від звичайної транзакції наявністю чіткого правила для повторення. Технічно ця функція працює через фоновий механізм системи, для якого в Android доцільно використовувати WorkManager [22]. Він періодично перевіряє активні платежі та самостійно створює відповідні списання у визначений день. Подібна автоматизація зводить до мінімуму ризик забути про оплату важливої підписки чи комунальної послуги. Асинхронне виконання таких операцій у Kotlin може спиратися на корутини Android, що зручно для фонові роботи та звернень до репозиторіїв [23].

За налаштування програми також відповідає окремий блок. Тут зберігається інформація про обрану валюту та мову інтерфейсу. Модуль контролює активацію ПІН-коду і керує підключенням до хмарного сховища. Усі ці параметри записуються в локальну пам'ять пристрою.

Останнім вагомим елементом виступає перелік фінансових справ який гармонійно доповнює основний облік. Він створений не для дублювання транзакцій а для планування важливих кроків. Тут можна створити нагадування про необхідність перевірки платних підписок поповнення скарбнички або перегляду встановлених лімітів. Кожна така справа має власну назву текстовий опис крайній термін виконання пріоритет та поточний статус.

Взагалі всі розрізнені архітектурні елементи функціонують як єдиний добре налаштований механізм. Гаманці просто зберігають інформацію про наявні гроші а фінансові операції постійно оновлюють поточний баланс. Блок історії наочно показує рух коштів за минулі періоди.

Для планування майбутнього існують бюджети які обмежують витрати та спеціальні цілі що допомагають планомірно відкладати гроші на важливі

покупки. Автоматизовані платежі успішно беруть на себе всю рутину з регулярними списаннями. Водночас розділ фінансових справ допомагає тримати в голові дрібні задачі.

Розподіл складних алгоритмів між незалежними сервісами та репозиторіями виявився надзвичайно правильним рішенням. Програмний код залишився максимально чистим. Готовий застосунок завдяки цьому працює абсолютно стабільно під час інтенсивного щоденного використання.

2.5. Організація збереження та синхронізації даних

Програма постійно працює з особистою фінансовою інформацією. Тому під час розробки слід враховувати загальні рекомендації OWASP MASVS щодо мобільної безпеки [24]. Вони наголошують на важливості безпечного збереження даних, правильній автентифікації та контролі доступу до чутливої інформації. У програмі ці принципи реалізовано через відмову від обов'язкового хмарного режиму, використання Firebase Authentication для входу, локальний PIN-захист і обмеження доступу до резервного копіювання лише для авторизованого користувача.

Архітектура поєднує надійну локальну базу додаткову хмарну синхронізацію та базовий захист входу. Головним місцем для запису всієї інформації обрано локальне сховище Room. Ця технологія дає змогу комфортно користуватися всіма функціями взагалі без доступу до мережі Інтернет.

Використання локальної моделі також дуже спрощує перше знайомство з продуктом. Користувач може просто відкрити додаток створити новий гаманець і одразу почати записувати свої витрати. При цьому система не вимагає проходити довгу процедуру реєстрації. Такий підхід вигідно відрізняє програму від багатьох конкурентів де створення облікового запису є обов'язковою умовою ще до першого запуску.

Локальна база даних має кілька переваг. По-перше, вона забезпечує швидкий доступ до записів. Дані не потрібно щоразу завантажувати з віддаленого сервера, тому екрани відкриваються швидко. По-друге, локальна база підвищує автономність. Користувач може працювати із застосунком у

						ЗПЗ.КР.Б-121-26-ПЗ	Арк.
							39

Усі основні операції з даними проходять через репозиторії. Це зроблено для того, щоб екрани та ViewModel не зверталися до DAO напряму. Наприклад, AccountRepository відповідає за роботу з рахунками, TransactionRepository за транзакції, BudgetRepository за бюджети. Якщо в майбутньому буде потрібно змінити спосіб збереження або додати синхронізацію, логіку можна буде оновити в репозиторіях, не переписуючи всі екрани.

Також у консолі видно й дані, які зберігають ці користувачі на прикладі рисунку 2.5, де перелічені усі транзакції користувача. Як приклад нижче лістинг фрагменту з класу `CloudBackupService` для очищення хмарного сховища. За бажанням користувач проходить авторизацію через електронну пошту Google. Після цього він може зробити резервну копію.



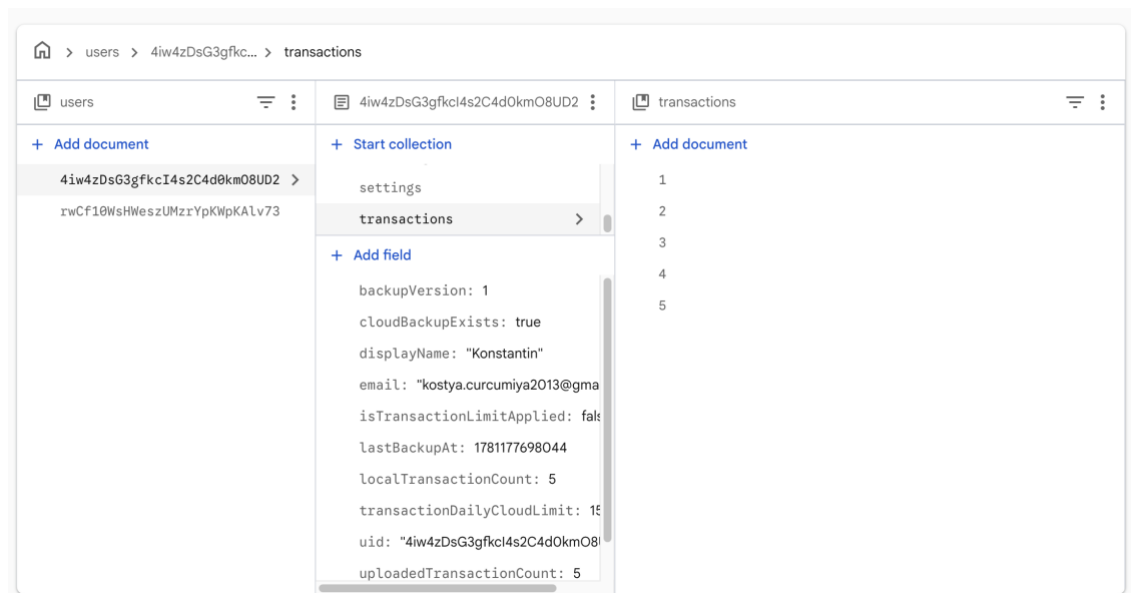


Рис. 2.5. Список транзакцій користувача у базі Firestore

```
suspend fun clearCloudBackup(): CloudBackupOperationResult {
    val userSession = googleAuthRepository.getCurrentUserSession()
    ?: return CloudBackupOperationResult(
        success = false,
        message = CloudBackupOperationMessage.SIGN_IN_REQUIRED
    )

    return runCatching {
        clearUserBackup(userSession.uid)

        firestore
            .collection(USERS_COLLECTION)
            .document(userSession.uid)
            .set(
                mapOf(
                    "uid" to userSession.uid,
                    "email" to userSession.email,
                    "displayName" to userSession.displayName,
                    "lastCloudClearAt" to System.currentTimeMillis(),
                    "lastBackupAt" to null,
                    "backupVersion" to BACKUP_VERSION,
                    "transactionDailyCloudLimit" to TRANSACTIONS_PER_DAY_LIMIT,
                    "localTransactionCount" to 0,
                    "uploadedTransactionCount" to 0,
                    "verifiedUploadedTransactionCount" to 0,
                    "isTransactionLimitApplied" to false,
                    "cloudBackupExists" to false
                )
            )
            .await()
    }
}
```

Проте основним місцем зберігання все одно залишається пам'ять самого смартфона. Хмарне середовище виступає лише зручним інструментом для безпечного перенесення бази на інший пристрій. Завдяки такій моделі вдалося зберегти повну автономність програми і водночас захистити людину від раптової втрати даних.

Захист доступу побудовано на основі перевірки чотиризначного пароля. Якщо користувач активує цю функцію у своєму профілі програма завжди вимагатиме ввести код під час кожного запуску. У випадку вимкненого захисту після стартової заставки одразу відкривається вікно гаманця. Очевидно що такий метод не може повністю замінити складне шифрування даних. Але він чудово працює як перший бар'єр проти доступу сторонніх осіб до особистої фінансової інформації. Такий підхід доповнює загальні рекомендації Android щодо безпечної роботи з локальними даними [27].

Додатковий комфорт використання забезпечується повною підтримкою української та англійської мов [28]. Усі текстові написи ніколи не вписуються прямо у код графічних вікон. Натомість вони зберігаються у спеціальних файлах ресурсів і підтягуються звідти за потреби. Особливо ретельно реалізовано переклад назв системних категорій та форматів дат. Саме ці елементи постійно зустрічаються майже на кожному кроці. Користувач бачить їх у записах витрат історії бюджетів та переліку постійних платежів.

Описана структура збереження та безпеки повністю задовольняє всі вимоги з першого розділу роботи. Створена програма вміє стабільно працювати в локальному режимі без обов'язкової попередньої реєстрації. Разом з тим вона дозволяє легко зробити копію бази даних після добровільного входу в систему. Наявність стартового пароля та якісний переклад інтерфейсу роблять щоденний контроль за власними грошима максимально безпечним і зручним.

Висновки до другого розділу

Другий розділ повністю присвячений проектуванню базової структури мобільного фінансового менеджера. Опираючись на попередні вимоги вдалося сформувати головні сценарії використання програми. Користувач зможе створювати нові рахунки та постійно записувати свої доходи чи витрати. Також передбачено зручні внутрішні перекази та перегляд загальної історії операцій.

Окремо реалізована робота з бюджетами та різними фінансовими цілями. Додатково програма підтримує керування регулярними платежами та дрібними

справами. Для загальної безпеки створено перевірку цифрового пароля та збереження резервної копії в хмарі.

Сама архітектура додатка побудована за модульним принципом. Візуальна частина графічного інтерфейсу працює завдяки інструментам Jetpack Compose. Взаємодія з локальними даними відбувається виключно через репозиторії. Вони надійно відокремлюють загальну логіку програми від прямого звернення до бази.

Значну увагу приділено правильній будові локальної бази даних Room. У ній створені окремі таблиці для збереження рахунків та категорій витрат. Також там зберігаються всі транзакції бюджети та фінансові цілі. Для регулярних платежів дрібних справ та загальних налаштувань передбачені свої місця. Обрана структура ідеально підходить для фінансової сфери та гарантує надійне збереження інформації без доступу до мережі Інтернет. Використання інструментів Firebase Authentication має виключно опційний характер.

У результаті проведеного проєктування вдалося сформулювати технічну основу програмного продукту. Запропонована структура залишається достатньо простою для легкого пояснення та подальшого оновлення коду. Водночас вона успішно охоплює абсолютно всі ключові функції сучасного фінансового менеджера.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ЗАСТОСУНКОМ

3.1 Загальна структура інтерфейсу

Користувацький інтерфейс мобільного застосунку розроблявся з урахуванням щоденного використання. Користувач повинен мати змогу миттєво додати нову операцію без зайвих кроків. Програма працює з особистими фінансами, тому її дизайн свідомо не перевантажений великою кількістю елементів. Усі найважливіші дії знаходяться на видному місці, а другорядні функції акуратно згруповані в окремих меню.

Графічне оформлення програми виконано у світлих та приємних кольорах. Використання спокійних відтінків чудово підтримує концепцію надійного та сучасного фінансового інструменту. Інтерфейс побудовано з використанням акуратних карток, плавно заокруглених кнопок та зрозумілих полів для введення тексту. Всі переліки та списки виглядають компактно і не перевантажують екран. Таке рішення узгоджується з рекомендаціями Material Design щодо зрозумілості й доступності інтерфейсу [29].

Уся структура візуальної частини спирається на філософію швидкого доступу до ключових операцій. Головне вікно миттєво показує стан фінансів, а нижня панель відкриває шлях до інших важливих розділів. Усі другорядні інструменти обережно сховані у відповідних меню. Такий компроміс дозволив зберегти ідеальний баланс між великою кількістю функцій та загальною простотою використання. Людина не витрачає зайвий час на пошук потрібної кнопки.

Одразу після запуску програми на екрані з'являється коротка стартова заставка (рис 3.1). Саме на цьому етапі програма перевіряє ПІН-код. Якщо користувач не вмикав цю функцію, то після заставки він одразу бачить головне вікно гаманця.

Сторінка перевірки пароля вписана у загальну структуру дизайну, хоча і не належить до головної навігації. Людина вводить чотири цифри свого коду і після успішної перевірки потрапляє на головний екран.



Рис. 3.1. Стартовий екран з ПІН-кодом

Центральним елементом додатка виступає вікно гаманця (рис. 3.2), воно відкривається першим і одразу показує всю загальну картину по рахунках. Тут відображається доступний баланс разом із короткими підсумками доходів та витрат. Нижче виводиться перелік останніх проведених транзакцій.

Крім того, на цьому ж екрані розміщені кнопки для переходу до додаткових інструментів. Серед них присутні бюджети, фінансові цілі, регулярні платежі та дрібні справи. Це дозволяє людині миттєво відкривати потрібні функції без довгого пошуку у вкладених меню.

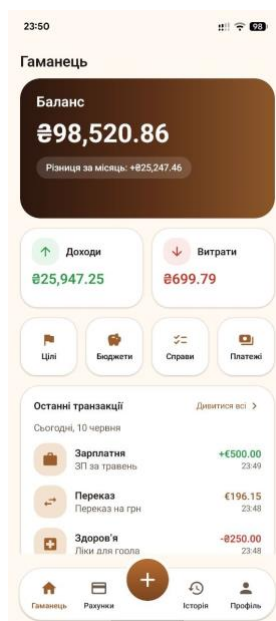
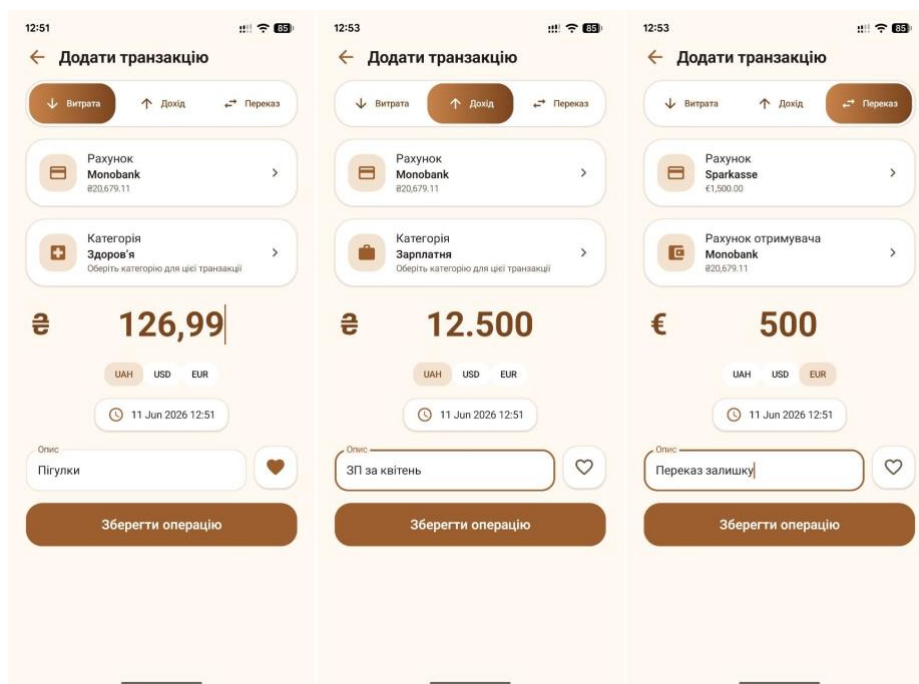


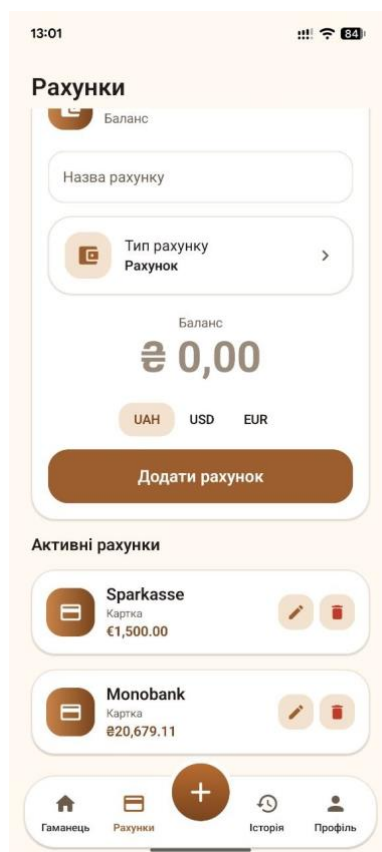
Рис. 3.2. Екран Гаманець

На екрані транзакції (рис. 3.3) можна побачити так звані toggle buttons перемикання типу операції між витратою, доходом та переказом. Далі йде кнопка вибору рахунку, кнопка вибору категорії і поле для введення суми. Знизу під сумою зручно можна обрати валюту транзакції та час її проведення. Опціональний опис та кнопка помітки транзакцію обраною в кінці перед кнопкою збереження транзакції.

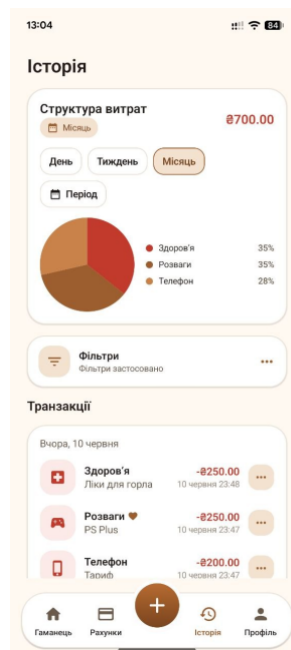


Сторінка рахунків (рис. 3.4) повністю присвячена управлінню джерелами грошей. У цьому вікні людина може переглянути перелік створених гаманців, побачити їхні назви, обрані валюти та точні баланси. Кожен такий рахунок працює як окреме фінансове джерело і постійно фігурує під час запису нових транзакцій чи переказів.

Вкладка історії (рис. 3.5) створена для зручного перегляду всіх минулих операцій. Всі записи тут автоматично групуються за днями для кращого візуального сприйняття. За потреби користувач може відфільтрувати список за конкретним періодом, обраним рахунком або окремою категорією. Важливим доповненням цього вікна стала наочна кругова діаграма. Вона допомагає миттєво оцінити головні напрямки витрат грошей. Таким чином звичайний список покупок перетворюється на повноцінний інструмент базової фінансової аналітики.



Меню фільтрів (рис. 3.6) дозволяє сортувати операції за назвою, типом операції, категорією, рахунком та часовим проміжком. Це не є повноцінним окремим екраном, а всього діалоговим вікном поверх сторінки з історією, яка затемняється, аби вікно з фільтром краще виділялось на фоні решти інтерфейсу.



Розділ профілю (рис. 3.7) має всі системні параметри та додаткові можливості. Саме тут можна змінити валюту, обрати мову інтерфейсу або налаштувати цифровий пароль для входу. Підтримка мовних налаштувань відповідає підходу Android до мовних параметрів застосунку [24]. Також на цій сторінці відбувається добровільна авторизація через сервіси Google для подальшого створення резервної копії в хмарі.

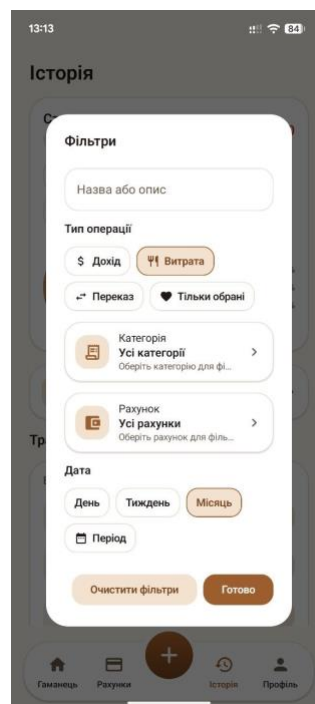


Рис. 3.6. Діалогове вікно з фільтрами транзакцій

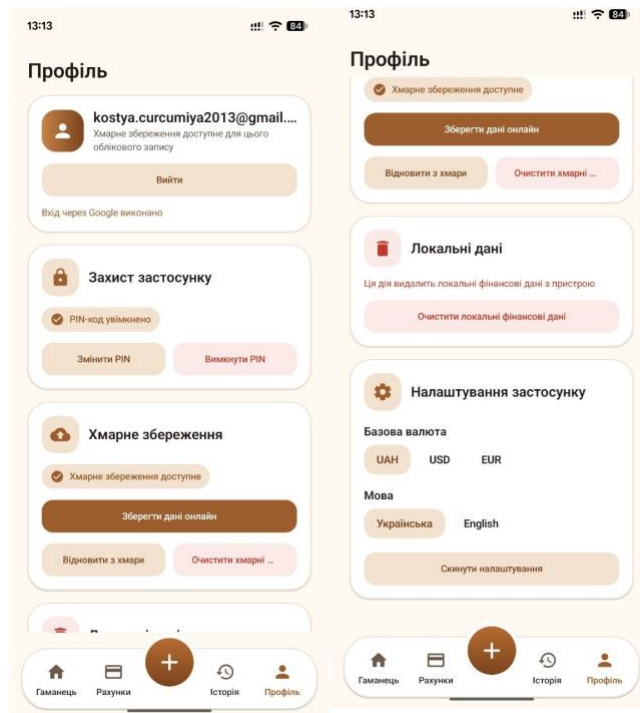


Рис. 3.7. Екран профілю та налаштувань

Усі налаштування згруповані на відокремлені для кращого сприйняття користувачем. Таке компонування є абсолютно логічним і правильним. Перелічені функції не потрібні для щоденного запису витрат, але вони суттєво впливають на загальний комфорт та безпеку роботи з програмою.



Рис. 3.8 Екран керування бюджетами

Розділ для роботи з бюджетами (рис. 3.8) створений спеціально для встановлення розумних обмежень на щоденні покупки. Людина вільно обирає потрібну категорію та вказує бажаний грошовий ліміт на певний проміжок часу. Далі програма автоматично порівнює цей показник із реальними витратами та малює наочний графік прогресу. Це дає змогу вчасно помітити наближення до встановленої межі та трохи зменшити свої витрати.

Планування майбутніх накопичень відбувається на сторінці цілей (рис. 3.9). Тут можна легко створити новий план збереження грошей із вказівкою бажаної фінальної суми та орієнтовної дати завершення. Інтерфейс додатка дуже чітко показує поточний відсоток виконання мети. Подібний інструмент ідеально підходить для планування дорогих покупок або поступового формування особистого резервного фонду.

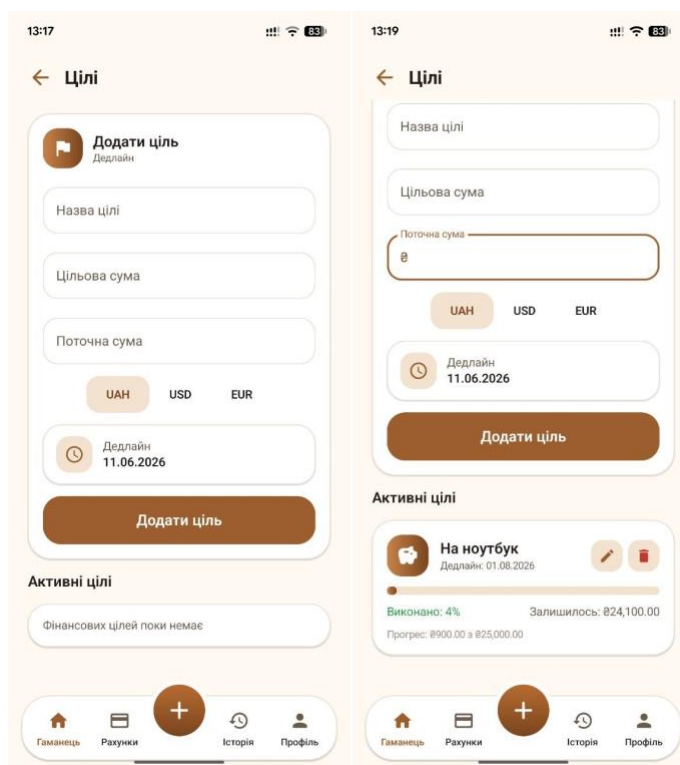


Рис. 3.9 Екран фінансових цілей

Контроль за всіма повторюваними списаннями реалізовано у вікні регулярних платежів. Сюди зазвичай відносять оплату комунальних послуг різноманітні платні підписки або щомісячну оренду житла. Під час налаштування такого платежу необхідно вказати прив'язаний рахунок відповідну категорію та точний день списання грошей.

У регулярного платежу також є функція оплати зараз або призупинки платежу, який стоятиме у черзі на паузі допоки його знову не активують. Завдяки цьому простому інструменту користувач ніколи не забуде про важливі фінансові зобов'язання і зможе краще прогнозувати майбутні витрати.

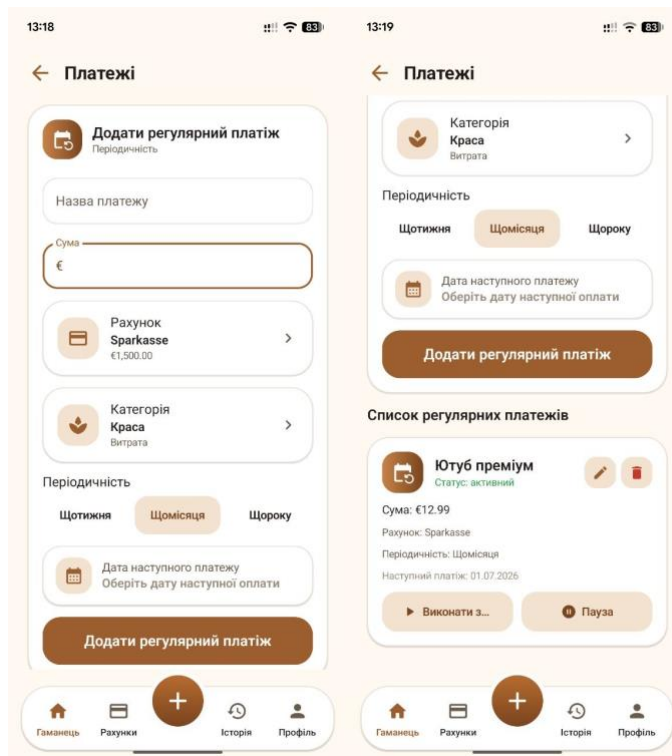


Рис. 3.10 Екран регулярних платежів

Останнім у цій групі виступає розділ дрібних фінансових справ (рис. 3.11). Він створений для запису важливих дій які не є прямими транзакціями. Для прикладу тут зручно створювати нагадування про необхідність перевірки тарифного плану або поповнення скарбнички. Кожен такий запис містить короткий опис крайній термін виконання та рівень пріоритету. Саме ця функція перетворює програму зі звичайного журналу витрат на повноцінного помічника для організації майбутніх дій.

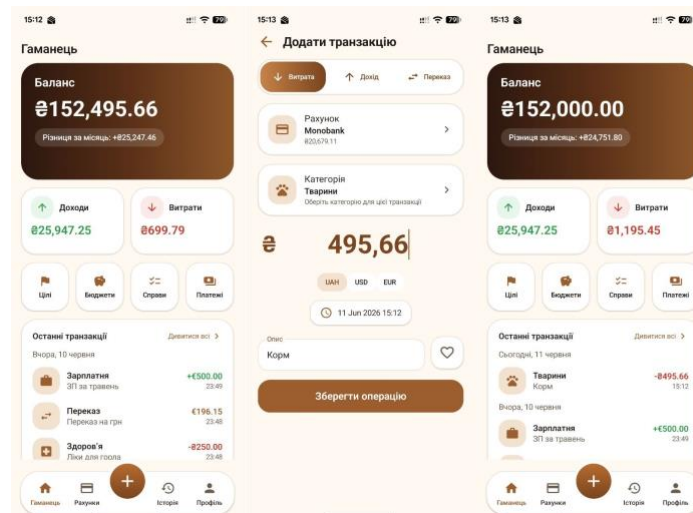
3.2 Тестування основних сценаріїв роботи

Після завершення розробки інтерфейсу настав важливий етап перевірки головних сценаріїв використання програми. Тестування Android-застосунків має охоплювати правильність функцій, поведінку інтерфейсу та стабільність користувацьких сценаріїв [25]. Процес тестування відбувався з позиції

Для мобільного фінансового інструменту вкрай важливо тестувати не просто окремі кнопки а цілі логічні ланцюжки дій. Для інтерфейсів на Jetpack Compose такі перевірки можуть охоплювати відображення елементів, натискання кнопок і зміну стану екрана [30]. Наприклад збереження витрати має супроводжуватися не лише появою запису в історії а й правильним зменшенням залишку на обраному рахунку. Своєю чергою редагування транзакції повинно коректно перераховувати загальний фінансовий стан без помилкового накладання нового значення поверх старого.

					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						52

На рис 3.13 дуже проілюстровано процес збереження витрати та оновлення загального балансу. На самому початку сценарію головний екран показує поточний стан коштів користувача. Далі відбувається перехід до спеціальної форми створення транзакції де вказується потрібний гаманець обрана категорія точна сума дата та невеликий текстовий опис покупки. Після успішного збереження даних ця витрата одразу додається до списку найсвіжіших операцій.



					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						53

The image displays four mobile app screens for a budgeting application, showing a balance of 152,000.00 and a list of transactions. The interface is in Ukrainian and includes a pie chart showing the distribution of expenses.

Screen 1: Main Dashboard

- Баланс:** 152,000.00
- Розмір на місяць:** +24,751.80
- Доходи:** 25,947.25
- Витрати:** 81,195.45
- Категорії витрат:** Ціни, Бюджет, Споживання, Платіжні
- Останні транзакції:**
 - Сьогодні, 11 червня: Тварини Корн -8495.66
 - Вчора, 10 червня: Зарплатня ЗП за травень +5500.00

Screen 2: Transaction History

- Історія:**
 - Сьогодні, 11 червня: Тварини Корн -8495.66
 - Вчора, 10 червня: Зарплатня ЗП за травень +5500.00
 - Вчора, 10 червня: Розваги PS Plus -8250.00
 - Вчора, 10 червня: Телефон Тариф -8200.00

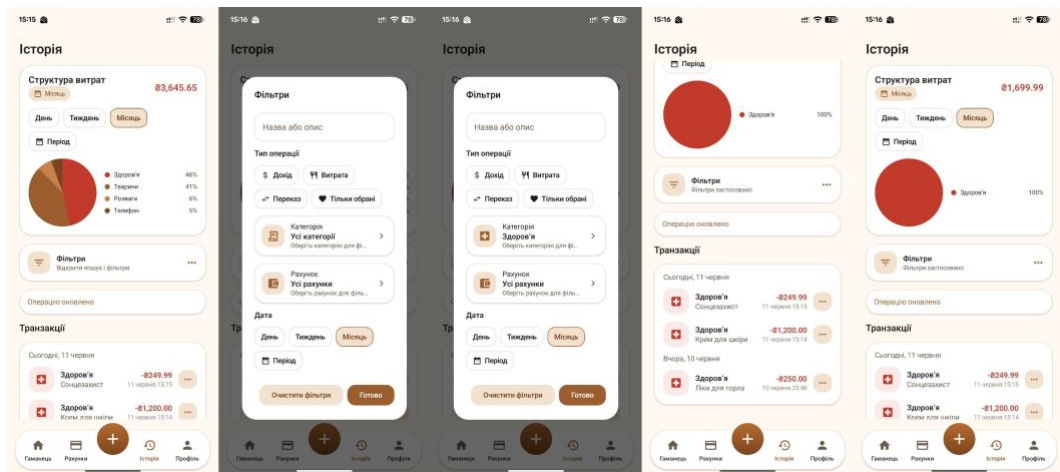
Screen 3: Expense Breakdown

- Історія:**
 - Місяць: 21,195.66
 - Діаграма витрат:
 - Тварини: 68%
 - Здоров'я: 11%
 - Розваги: 11%
 - Телефон: 9%
 - Фільтри: Відібрати пошуки і фільтри
 - Операції: Оновити

Screen 4: Transaction History

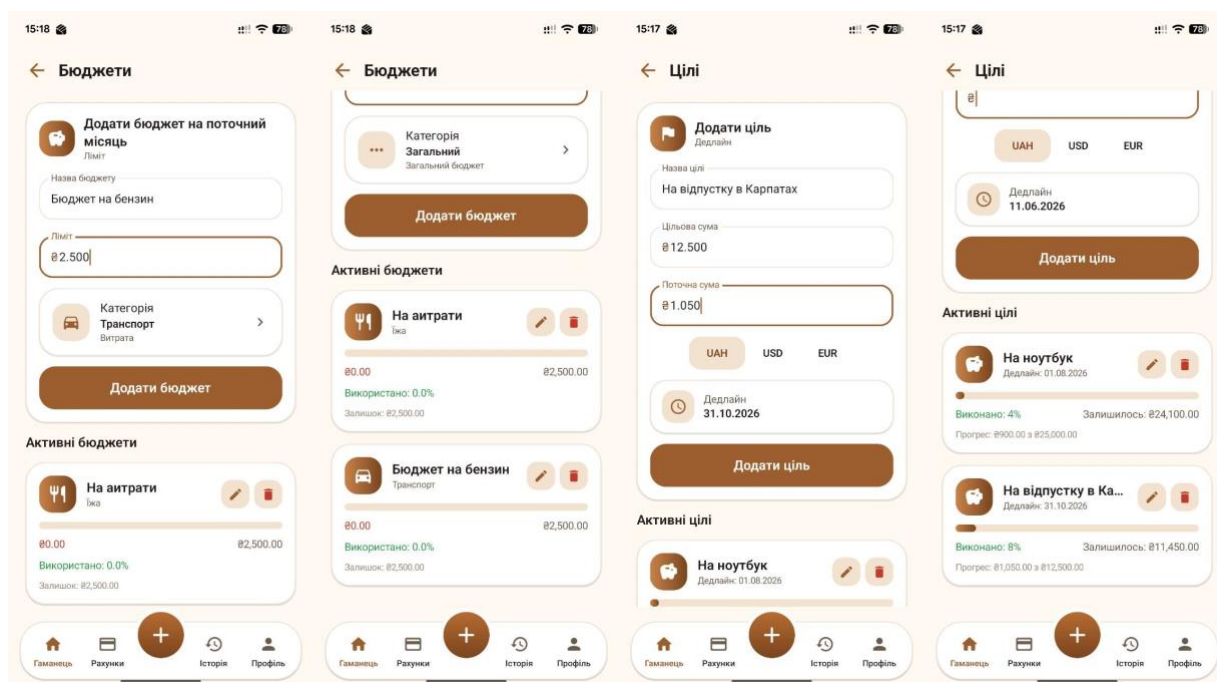
- Історія:**
 - Сьогодні, 11 червня: Тварини Корн -81,495.66
 - Вчора, 10 червня: Зарплатня ЗП за травень +5500.00

На рис 3.15 наочно зображено результати перевірки системи фільтрації у загальній історії платежів. Користувач заходить у відповідний розділ та відкриває приховану панель розширених фільтрів. Там він задає специфічні параметри пошуку і обирає конкретний тип операції категорію гаманець валюту або бажаний часовий проміжок. Одразу після підтвердження вибору список очищується від зайвого залишаючи лише релевантні записи.



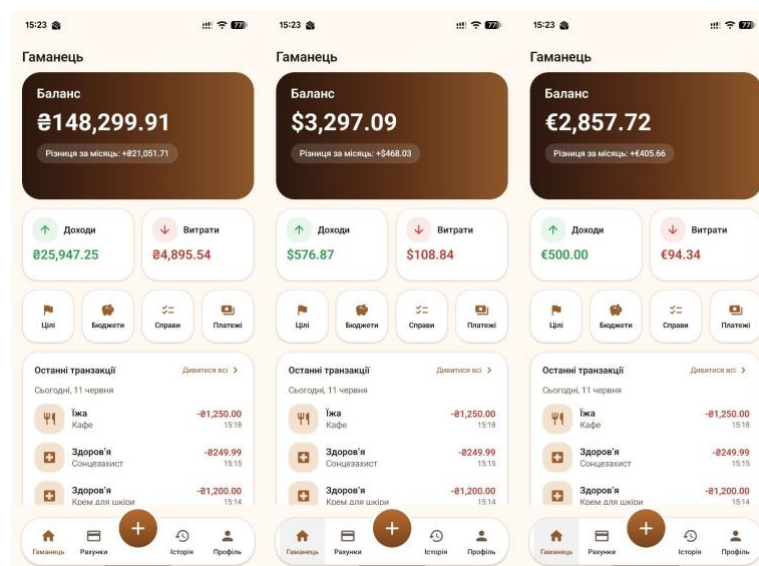
					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						54

Друга частина ілюстрації розкриває кроки для планування майбутніх покупок. Користувач створює нову фінансову ціль вказуючи її ім'я бажану фінальну суму поточний стан накопичень та орієнтовну дату завершення збору. Після збереження цей план з'являється у меню разом зі зручним візуальним індикатором прогресу.



Можливості роботи з фінансовими справами та нагадуваннями розкрито на рис 3.17. Тут людина записує заголовок завдання додає розгорнутий текстовий опис та викликає календар для вибору крайньої дати виконання. За необхідності також визначається рівень пріоритету та вмикається сповіщення. Сформована задача миттєво потрапляє до загального переліку справ із зазначенням термінів та поточного статусу активності.

Перевірка механізму зміни основної системної валюти що зображено на рис 3.20. На прикладі головного вікна чудово видно як одні і ті самі фінансові показники можуть отримувати зовсім інший вигляд у гривнях доларах чи євро. Одразу після вибору нової базової валюти додаток проводить перерахунок.



					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						57

На рис 3.21 зображено результати перевірки англійської локалізації вбудованого інтерфейсу. Щойно користувач змінює мову в головних налаштуваннях елементи екранів миттєво перекладаються англійською.

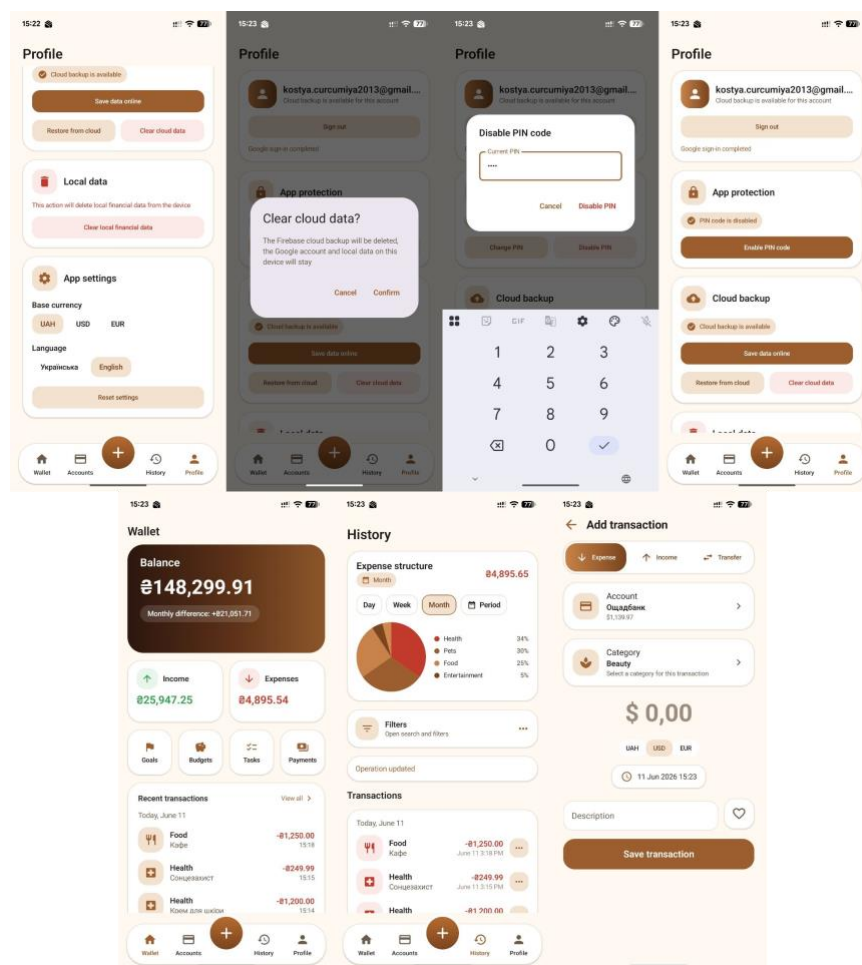


Рис. 3.21. Перевірка англійської локалізації застосунку

Детальні результати функціонального тестування зібрано у таблиці 3.1. Вона містить опис ключових перевірених сценаріїв разом із очікуваними та фактичними результатами після виконання всіх дій.

Таблиця 3.1

Результати функціонального тестування

№	Сценарій перевірки	Очікуваний результат	Результат
1	Запуск застосунку без увімкненого PIN-коду	Після заставки відкривається головний екран гаманця	Виконано
2	Запуск застосунку з увімкненим PIN-кодом	Перед головним екраном відкривається форма введення коду	Виконано

3	Створення рахунку	Новий рахунок з'являється у списку та доступний для транзакцій	Виконано
4	Створення доходу	Баланс вибраного рахунку збільшується на суму операції	Виконано
5	Створення витрати	Баланс вибраного рахунку зменшується на суму операції	Виконано
6	Переказ між рахунками	Сума списується з одного рахунку та додається до іншого	Виконано
7	Редагування транзакції	Старий вплив операції скасовується, після чого застосовуються нові дані	Виконано
8	Фільтрація історії	Список операцій змінюється відповідно до вибраних умов	Виконано
9	Створення бюджету	Бюджет зберігається та відображає прогрес витрат	Виконано
10	Створення фінансової цілі	Ціль зберігається та показує прогрес накопичення	Виконано
11	Створення регулярного платежу	Платіж зберігається з вибраним днем виконання	Виконано
12	Створення фінансової справи	Задача з'являється у списку з дедлайном і пріоритетом	Виконано
13	Зміна мови інтерфейсу	Тексти інтерфейсу відображаються вибраною мовою	Виконано
14	Авторизація та резервне копіювання	Після входу користувач може створити хмарну копію даних	Виконано

Успішно проведені перевірки довели що реалізована програма повністю підтримує всі головні сценарії які були визначені ще на етапі проєктування. Вся інформація надійно зберігається локально а загальний баланс рахунків завжди змінюється відповідно до типу операції. Створений інтерфейс дозволяє дуже

комфортно виконувати щоденні фінансові дії без жодного складного налаштування.

Висновки до третього розділу

Третій розділ присвячено аналізу UI/UX програми та взаємодії з її інструментами. Дизайн додатку орієнтований на максимально швидкий доступ до фінансових дій починаючи від заставки і перевірки PIN коду до головного вікна навігації. Особливу увагу приділено рахункам, бо вони є базою фінансової моделі. Користувач може створювати та редагувати гаманці у різних валютах.

Детально вивчено процес фіксації доходів витрат та внутрішніх переказів. Акцент зроблено на безпечному перерахунку балансу під час редагування транзакцій що гарантує точність збережених даних. Також розглянуто інструменти аналізу та планування. Сторінка історії з фільтрами бюджету фінансові цілі регулярні платежі та дрібні справи перетворюють програму на потужний засіб системного контролю фінансів.

В огляді профілю розібрано налаштування локалізації базової валюти та хмарного копіювання. Додаток здатен працювати повністю автономно а всі хмарні можливості активуються лише за бажанням користувача. Фінальне тестування підтвердило стабільність абсолютно всіх функцій від створення операцій до роботи захисту. Створений продукт повністю задовольняє поставлені вимоги і є надійним мобільним інструментом для щоденного контролю особистих грошей.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено мобільний застосунок особистого фінансового менеджера. Під час розробки було досліджено предметну область визначено потреби користувачів та успішно обґрунтовано вибір технологій. Цільовою платформою стала система Android, а написання коду відбувалося мовою Kotlin. Використання інструментів Jetpack Compose разом із Room та Firebase дозволило створити зручний для користування продукт.

Базовий функціонал програми спирається на постійну роботу з рахунками транзакціями та категоріями. Користувач, для фіксування щоденних витрат, отримує змогу легко та ефективно вести їхній облік, та загальний стан рахунків. Увагу в коді приділено й перерахунку балансу після всіх змін для гарантування правильної статистики. А за повністю автономне збереження інформації відповідає локальна пам'ять пристрою. Водночас інтеграція сервісів Google надає добровільну можливість безпечного створення хмарних резервних копій.

Користувацький інтерфейс має зрозумілу структуру та навігацію. Для захисту інформації додано перевірку стартового цифрового пароля. Загальний комфорт використання підтримується двомовною локалізацією та вільним вибором головної валюти. Проведене фінальне тестування підтвердило абсолютно коректну роботу всіх створених модулів та алгоритмів.

З огляду на отримані результати поставлену мету роботи досягнуто повною мірою. Розроблений додаток відповідає всім початковим вимогам і чудово демонструє практичне застосування сучасних інструментів та принципів інженерії програмного забезпечення. Створена система виступає повноцінним помічником для контролю грошей і має надійний фундамент для подальшого покращення та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wallet [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.droid4you.application.wallet>
2. Monefy – Budget & Expenses app [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.monefy.app.lite>
3. Money Manager Expense & Budget [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.realbyteapps.moneymanagerfree>
4. Money Tracker – Expense & Budget [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.freeman.moneymanager>
5. Bennett D. E., Thompson P. Use of Anthropomorphic Brand Mascots for Student Motivation and Engagement: A Promotional Case Study With Pablo the Penguin at the University of Portsmouth Library. New Review of Academic Librarianship. 2016. DOI: 10.1080/13614533.2016.1162179.
6. Develop for Android [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/develop>
7. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/studio>
8. Kotlin and Android [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/kotlin>
9. Data classes. Kotlin Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://kotlinlang.org/docs/data-classes.html>
10. Jetpack Compose Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/develop/ui/compose/documentation>
11. Material Design 3 in Compose [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/develop/ui/compose/designsystems/material3>
12. Save data in a local database using Room [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/training/data-storage/room>

13. Cloud Firestore [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/firestore>
14. Firebase Authentication [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/auth>
15. Android Distribution Chart [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/about/dashboards>
16. Guide to app architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/topic/architecture>
17. UI layer. App architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/topic/architecture/ui-layer>
18. ViewModel overview [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/topic/libraries/architecture/viewmodel>
19. Data layer. App architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/topic/architecture/data-layer>
20. Define data using Room entities [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/training/data-storage/room/defining-data>
21. API Національного банку України [Електронний ресурс] – Режим доступу до ресурсу: <https://bank.gov.ua/en/open-data/api-dev>
22. Getting started with WorkManager [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/develop/background-work/background-tasks/persistent/getting-started>
23. Kotlin coroutines on Android [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.android.com/kotlin/coroutines>
24. OWASP Mobile Application Security Verification Standard [Електронний ресурс] – Режим доступу до ресурсу: <https://mas.owasp.org/MASVS/>
25. Cloud Firestore data model [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/firestore/data-model>
26. Authenticate with Google on Android [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/auth/android/google-signin>

27. Android security best practices [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.android.com/privacy-and-security/security-best-practices>

28. Per-app language preferences [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.android.com/guide/topics/resources/app-languages>

29. Accessibility overview. Material Design 3 [Электронный ресурс] – Режим доступа до ресурсу: <https://m3.material.io/foundations/accessible-design/overview>

30. Testing apps on Android [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.android.com/training/testing>

ДОДАТКИ

					ЗІПЗ.КР.Б-121-26-ПЗ	Арк.
						65

```

suspend fun addTransaction(
    type: TransactionType,
    sourceAccountId: Long?,
    targetAccountId: Long?,
    categoryId: Long?,
    amountValue: Double?,
    amountCurrency: CurrencyCode,
    description: String,
    date: Long,
    isFavorite: Boolean
): OperationResult {
    val transaction = buildTransaction(
        id = 0,
        type = type,
        sourceAccountId = sourceAccountId,
        targetAccountId = targetAccountId,
        categoryId = categoryId,
        amountValue = amountValue,
        amountCurrency = amountCurrency,
        description = description,
        date = date,
        isFavorite = isFavorite,
        createdAt = System.currentTimeMillis()
    ) ?: return OperationResult(false, lastError)

    return runAtomicOperation {
        applyTransactionBalance(transaction).throwIfFailed()
        transactionRepository.addTransaction(transaction)

        when (type) {
            TransactionType.INCOME -> OperationResult(true,
TransactionOperationMessage.INCOME_ADDED)
            TransactionType.EXPENSE -> OperationResult(true,
TransactionOperationMessage.EXPENSE_ADDED)
            TransactionType.TRANSFER -> OperationResult(true,
TransactionOperationMessage.TRANSFER_COMPLETED)
        }
    }
}

suspend fun updateTransaction(
    oldTransaction: Transaction,
    type: TransactionType,
    sourceAccountId: Long?,
    targetAccountId: Long?,
    categoryId: Long?,
    amountValue: Double?,
    amountCurrency: CurrencyCode,
    description: String,
    date: Long,
    isFavorite: Boolean
): OperationResult {
    val newTransaction = buildTransaction(
        id = oldTransaction.id,
        type = type,
        sourceAccountId = sourceAccountId,
        targetAccountId = targetAccountId,
        categoryId = categoryId,
        amountValue = amountValue,
        amountCurrency = amountCurrency,
        description = description,
        date = date,

```

```

        isFavorite = isFavorite,
        createdAt = oldTransaction.createdAt
    ) ?: return OperationResult(false, lastError)

    return runAtomicOperation {
        rollbackTransactionBalance(oldTransaction).throwIfFailed()
        applyTransactionBalance(newTransaction).throwIfFailed()
        transactionRepository.updateTransaction(newTransaction)
        OperationResult(true, TransactionOperationMessage.TRANSACTION_UPDATED)
    }
}

```

Додаток В

```

@Database(
    entities = [
        AccountEntity::class,
        CategoryEntity::class,
        TransactionEntity::class,
        BudgetEntity::class,
        GoalEntity::class,
        RecurringPaymentEntity::class,
        FinancialTaskEntity::class,
        AppSettingsEntity::class
    ],
    version = 6,
    exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {

    abstract fun accountDao(): AccountDao

    abstract fun categoryDao(): CategoryDao

    abstract fun transactionDao(): TransactionDao

    abstract fun budgetDao(): BudgetDao

    abstract fun goalDao(): GoalDao

    abstract fun recurringPaymentDao(): RecurringPaymentDao

    abstract fun financialTaskDao(): FinancialTaskDao

    abstract fun appSettingsDao(): AppSettingsDao
}

```

Додаток С

```

class NbuExchangeRateRepository(
    context: Context
) {

    private val preferences = context.getSharedPreferences(
        PREFERENCES_NAME,
        Context.MODE_PRIVATE
    )

    suspend fun refreshRates(): Boolean { return loadRatesToUah().isSuccess }
}

```

```

suspend fun loadRatesToUah(): Result<Map<CurrencyCode, Double>> {
    return withContext(Dispatchers.IO) {
        runCatching {
            val cachedRates = loadCachedRatesIfValid()
            if (cachedRates != null) { return@runCatching cachedRates }
            val rates = fetchRatesFromNbu()
            saveRatesToCache(rates)
            rates
        }
    }
}

fun convert(
    amount: Double,
    from: CurrencyCode,
    to: CurrencyCode
): Double {
    if (from == to) { return amount }
    val fromRateToUah = getCacheRateToUah(from)
    val toRateToUah = getCacheRateToUah(to)
    val amountInUah = amount * fromRateToUah
    return amountInUah / toRateToUah
}

private fun loadCachedRatesIfValid(): Map<CurrencyCode, Double>? {
    val today = LocalDate.now().toString()
    val cachedDate = preferences.getString(KEY_DATE, null)
    if (cachedDate != today) { return null }
    val usdRate = preferences.getFloat(KEY_USD, -1f).toDouble()
    val eurRate = preferences.getFloat(KEY_EUR, -1f).toDouble()
    if (usdRate <= 0.0 || eurRate <= 0.0) { return null }
    return mapOf(
        CurrencyCode.UAH to 1.0,
        CurrencyCode.USD to usdRate,
        CurrencyCode.EUR to eurRate
    )
}

private fun saveRatesToCache(rates: Map<CurrencyCode, Double>) {
    preferences.edit()
        .putString(KEY_DATE, LocalDate.now().toString())
        .putFloat(KEY_USD, rates[CurrencyCode.USD]?.toFloat() ?:
FALLBACK_USD_TO_UAH.toFloat())
        .putFloat(KEY_EUR, rates[CurrencyCode.EUR]?.toFloat() ?:
FALLBACK_EUR_TO_UAH.toFloat())
        .apply()
}

private fun fetchRatesFromNbu(): Map<CurrencyCode, Double> {
    val connection = URL(NBU_EXCHANGE_RATE_URL)
        .openConnection() as HttpURLConnection

    connection.requestMethod = "GET"
    connection.connectTimeout = 10_000
    connection.readTimeout = 10_000

    try {
        val response = connection.inputStream
            .bufferedReader()
            .use { reader -> reader.readText() }

        val jsonArray = JSONArray(response)
        val rates = mutableMapOf(

```

```

        CurrencyCode.UAH to 1.0,
        CurrencyCode.USD to FALLBACK_USD_TO_UAH,
        CurrencyCode.EUR to FALLBACK_EUR_TO_UAH
    )

    for (index in 0 until jsonArray.length()) {
        val item = jsonArray.getJSONObject(index)
        val code = item.optString("cc")
        val rate = item.optDouble("rate", 0.0)
        if (rate <= 0.0) { continue }

        when (code) {
            CurrencyCode.USD.name -> { rates[CurrencyCode.USD] = rate }
            CurrencyCode.EUR.name -> { rates[CurrencyCode.EUR] = rate }
        }
    }
    return rates
} finally { connection.disconnect() }
}

private fun getCachedRateToUah(currency: CurrencyCode): Double {
    return when (currency) {
        CurrencyCode.UAH -> 1.0
        CurrencyCode.USD -> preferences
            .getFloat(KEY_USD, FALLBACK_USD_TO_UAH.toFloat())
            .toDouble()

        CurrencyCode.EUR -> preferences
            .getFloat(KEY_EUR, FALLBACK_EUR_TO_UAH.toFloat())
            .toDouble()
    }
}

private companion object {
    const val PREFERENCES_NAME = "exchange_rates_cache"
    const val NBU_EXCHANGE_RATE_URL =
        "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json"

    const val KEY_DATE = "date"
    const val KEY_USD = "usd"
    const val KEY_EUR = "eur"

    const val FALLBACK_USD_TO_UAH = 41.0
    const val FALLBACK_EUR_TO_UAH = 44.5
}
}

```