НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення мобільного застосунку для управління особистими фінансами»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**

Керівник бакалаврської кваліфікаційної роботи

асистент _____ **Тетяна БАРАНОВА**

Консультант бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ **Ірина БОРОДКІНА**

Виконав _____ **Максим ШЕВЧУК**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
к.т.н., доцент _____ Белла ГОЛУБ
«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Шевчуку Максиму Павловичу
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
« Програмне забезпечення мобільного застосунку для управління особистими фінансами»
затверджена наказом від «19» Грудня 2025 р. No 3204 «С»
Термін подання завершеної роботи на кафедру «22» Травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: Мобільний застосунок для Android (API 23+) для управління особистими фінансами. Технології розробки: Flutter, Dart, SQLite (sqflite), Firebase Authentication та Firestore. Зовнішнє API - Національний банк України (курси валют). Архітектура - Clean Architecture з використанням патерну BLoC. Інтерфейс - українська мова, Material Design 3.

Перелік питань, що підлягають дослідженню: 1. Аналіз предметної області та існуючих аналогів додаткыв для управління особистими фінансами, 2. Проєктування структури локальної бази даних та логічної моделі даних, 3. Реалізація клієнтської частини додатку за принципами Clean Architecture з використанням фреймворку Flutter, 4. Розробка системи безпеки даних та механізму хмарного резервного копіювання через Firebase, 5. Реалізація асистента та автоматичних підказок опису транзакцій.

Дата видачі завдання «19» Грудня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Тетяна БАРАНОВА**

Консультант бакалаврської кваліфікаційної роботи _____ **Ірина БОРОДКІНА**

Завдання взяв до виконання _____ **Максим ШЕВЧУК**

ЗМІСТ

ВСТУП 6
1 УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ЯК ПРЕДМЕТНА ОБЛАСТЬ 10
1.2 Моделювання предметної області 13
1.3 Огляд існуючих аналогічних рішень 17
1.4 Постановка завдання 20
2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ 24
2.1 Логічна модель даних 24
2.2 Вибір системи управління базами даних 27
2.3 Розробка структури інформаційної бази 29
2.4 Фізична структура бази даних 32
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 36
3.1 Архітектура програмної системи 36
3.2 Структура клієнтської частини 39
3.3 Реалізація основних функцій 40
3.4 Реалізація хмарної синхронізації 43
3.5 Реалізація системи безпеки даних 44
3.6 Вибір інструментарію розробки 46
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 49
4.1 Тестування системи 49
4.2 Вимоги до апаратного та програмного забезпечення 53
4.3 Розгортання та інсталяція 55
4.4 Інструкція з експлуатації 56
ВИСНОВКИ 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62
Додаток А 66
Додаток Б 68
Додаток В 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

НБУ - Національний банк України.

SDK (Software Development Kit) - набір інструментів розробника для створення програм під певну платформу.

API (Application Programming Interface) - інтерфейс прикладного програмування, що визначає правила взаємодії між програмними компонентами.

BLoC (Business Logic Component) - архітектурний патерн для розділення бізнес-логіки та інтерфейсу користувача у Flutter-застосунках.

Clean Architecture - підхід до проєктування програмного забезпечення з чітким розділенням на шари (data, domain, presentation).

SQLite - вбудована реляційна база даних, що зберігається у файлі на пристрої.

Firebase - платформа Google для розробки мобільних і вебзастосунків з готовими сервісами автентифікації, бази даних і хмарних функцій.

Firestore - хмарна NoSQL-база даних від Firebase, що працює у реальному часі.

NLU (Natural Language Understanding) - технологія розуміння природної мови, що використовується для обробки запитів користувача.

UI (User Interface) - інтерфейс користувача, видима частина застосунку, з якою взаємодіє користувач.

UX (User Experience) - досвід користувача, його загальні враження від взаємодії з продуктом.

JSON (JavaScript Object Notation) - формат обміну даними, заснований на текстовому представленні об’єктів JavaScript.

CRUD (Create, Read, Update, Delete) - набір базових операцій над даними у застосунках.

UID (User Identifier) - унікальний ідентифікатор користувача у системі.

REST (Representational State Transfer) - архітектурний стиль взаємодії компонентів розподіленого застосунку у мережі.

HTTP (HyperText Transfer Protocol) - протокол передачі гіпертексту у всесвітній павутині.

UML (Unified Modeling Language) - уніфікована мова моделювання для опису та проєктування програмних систем.

ER-діаграма - діаграма «сутність-зв’язок», що показує структуру даних та зв’язки між ними.

DAO (Data Access Object) - патерн доступу до даних, що інкапсулює роботу з джерелом даних.

IDE (Integrated Development Environment) - інтегроване середовище розробки програмного забезпечення.

ВСТУП

Цифровізація сучасного суспільства торкнулася всіх сфер життя людини, включно з управлінням особистими фінансами. Якщо раніше для ведення обліку доходів і витрат використовували паперові блокноти або електронні таблиці, то сьогодні все більше людей надають перевагу мобільним застосункам, які забезпечують зручний інтерфейс, миттєвий доступ до даних та автоматизовану аналітику.

За даними дослідження Національного банку України, значна частина громадян не веде систематичного обліку своїх витрат, що ускладнює процес фінансового планування та накопичення заощаджень. [1] Така ситуація свідчить про потребу у простих, доступних та локалізованих інструментах для управління особистими фінансами, особливо орієнтованих на українського користувача з підтримкою національної валюти та офіційних курсів НБУ. [4]

Існуючі рішення на ринку, такі як Money Manager, Wallet і Spendee, частково покривають ці потреби, проте мають низку обмежень: відсутність повної української локалізації, платні підписки за базовий функціонал, відсутність інтеграції з API Національного банку України та брак інтелектуальних функцій підтримки прийняття рішень. [3] Тому актуальною є розробка україномовного мобільного застосунку з безкоштовним базовим функціоналом, інтеграцією офіційних курсів НБУ та елементами інтелектуальних підказок для аналізу фінансової поведінки користувача.

Об’єктом дослідження є процес управління особистими фінансами користувача через мобільний застосунок із підтримкою хмарної синхронізації та інтелектуальних підказок.

Предметом дослідження є методи, алгоритми та технології розробки мобільного застосунку для управління особистими фінансами на платформі Android з використанням фреймворку Flutter, локальної бази даних SQLite та хмарної платформи Firebase. [10],[13]

Метою бакалаврської кваліфікаційної роботи є розробка та реалізація мобільного застосунку «Pocket Finance» для управління особистими фінансами користувача з підтримкою обліку транзакцій, планування бюджетів, постановки фінансових цілей, аналітики витрат, інтелектуального асистента та хмарного резервного копіювання даних.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область управління особистими фінансами та дослідити існуючі аналогічні рішення;

- розробити модель предметної області та сформулювати вимоги до програмної системи;

- спроєктувати архітектуру застосунку та структуру бази даних;

- обґрунтувати вибір технологій та інструментарію розробки;

- реалізувати клієнтську частину програмного продукту з використанням фреймворку Flutter;

- розробити механізми ізоляції даних між користувачами та інтелектуальні підказки на основі історії транзакцій;

- реалізувати інтеграцію з хмарною платформою Firebase для резервного копіювання даних;

- провести тестування застосунку та підготувати його до публікації.

Наукова новизна роботи полягає у поєднанні класичних функцій обліку особистих фінансів із елементами автоматичних підказок на основі статистичного аналізу історії транзакцій користувача, а також у використанні офіційного API Національного банку України для забезпечення актуальних курсів валют у мобільному застосунку.

Практична цінність роботи полягає у створенні готового до використання мобільного застосунку, який може бути впроваджений у Google Play та

використаний широким колом українських користувачів для контролю особистих фінансів. Розроблений програмний продукт є безкоштовним, україномовним, працює без підключення до мережі Інтернет та підтримує інтеграцію з офіційними курсами Національного банку України.

Апробація результатів. Розроблений застосунок «Pocket Finance» доступний у вигляді APK-збірки для встановлення на пристрої з операційною системою Android версії 6.0 і вище. Вихідний код проєкту розміщено у публічному репозиторії GitHub.

Методологічною основою дослідження є об’єктно-орієнтований підхід до аналізу та проєктування програмних систем, а також принципи Clean Architecture для організації коду застосунку. [9] У роботі застосовано методи UML-моделювання для опису предметної області (діаграми варіантів використання, послідовності та класів), методи реляційного проєктування для розробки структури бази даних, а також методи функціонального тестування для верифікації розробленої системи. [5],[16]

Інформаційною базою дослідження є науково-технічна література з питань мобільної розробки, офіційна документація фреймворку Flutter, мови програмування Dart, платформи Firebase та бібліотеки sqflite,а також матеріали профільних видань DOU.ua та DevZone.org.ua. [5], [10], [16], [17], [25], [10], [11], [13], [22], [27], [28]

Бакалаврська кваліфікаційна робота виконана на кафедрі комп’ютерних наук факультету інформаційних технологій Національного університету біоресурсів і природокористування України. У процесі виконання роботи використано сучасні інструменти розробки програмного забезпечення: інтегроване середовище Visual Studio Code, систему контролю версій Git, платформу GitHub для зберігання коду, а також інструменти Android Studio для тестування на емуляторі та фізичних пристроях.

Структура та обсяг роботи. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 63 сторінки основного тексту, містить 30

джерел та 4 додатки. Перший розділ присвячено аналізу предметної області управління особистими фінансами та огляду існуючих аналогічних рішень. Другий розділ описує проєктування інформаційного забезпечення застосунку. Третій розділ містить опис розробки програмного забезпечення. Четвертий розділ включає рекомендації щодо впровадження та результати тестування системи.

1 УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ЯК ПРЕДМЕТНА ОБЛАСТЬ

1.1 Аналіз процесів управління особистими фінансами

Управління особистими фінансами є важливою складовою повсякденного життя сучасної людини, яка передбачає свідомий контроль над надходженнями та витратами грошових ресурсів протягом певного періоду часу. До надходжень належать заробітна плата, стипендія, доходи від підробітку та інші джерела грошових ресурсів, а до витрат - оплата комунальних послуг, продуктів харчування, транспорту, навчання, дозвілля та інших потреб.

Систематичний облік та аналіз особистих фінансів сприяють фінансовій грамотності, дозволяють виявити основні статті витрат, оптимізувати споживчу поведінку та сформувати накопичення на довгострокові цілі. За даними дослідження Національного банку України, значна частина громадян не веде систематичного обліку особистих витрат, що ускладнює фінансове планування та накопичення заощаджень. [1]

Процес управління особистими фінансами включає декілька ключових етапів. На початковому етапі відбувається фіксація фінансових операцій - реєстрація доходів і витрат із зазначенням категорії, дати та суми. На етапі аналізу обчислюється сукупний баланс, формується статистика витрат за категоріями та періодами, виявляються основні тенденції споживання. Етап планування передбачає встановлення бюджетних обмежень за категоріями витрат та постановку довгострокових фінансових цілей. На етапі контролю відбувається порівняння фактичних витрат із плановими показниками та коригування фінансової поведінки користувача.

Історично облік особистих фінансів вівся у паперових блокнотах, що мало низку істотних недоліків: ризик втрати інформації, складність обчислень, відсутність автоматизованої аналітики та необхідність ручного перерахунку підсумків. Із появою персональних комп’ютерів значного поширення набули

електронні таблиці, які забезпечували автоматизацію обчислень та можливість побудови графіків. Проте такий підхід вимагає від користувача певних навичок роботи з табличними процесорами та доступу до комп'ютера для здійснення запису операції. [2]

Сучасним рішенням проблеми обліку особистих фінансів є мобільні застосунки. Поширення смартфонів забезпечує постійний доступ до інструментів обліку - фінансова операція може бути зафіксована безпосередньо у момент її здійснення. Це усуває потребу у запам'ятовуванні витрат та подальшому перенесенні їх до бухгалтерського обліку. Мобільні застосунки забезпечують автоматизацію аналітики, формування звітів, нагадування про планові операції, синхронізацію з іншими пристроями та інтеграцію з банківськими системами.

Окремим аспектом управління особистими фінансами є робота з декількома валютами. Значна частина громадян України отримує доходи або здійснює витрати в іноземних валютах, переважно у доларах США та євро. Для коректного обліку таких операцій застосунок має забезпечувати конвертацію сум за актуальним курсом. Найбільш надійним джерелом валютних курсів є офіційне API Національного банку України, яке надає безкоштовний доступ до даних про курси понад 100 іноземних валют. [4]

Важливим компонентом систем управління особистими фінансами є функція бюджетування. Бюджет являє собою план витрат за певними категоріями на встановлений період, як правило - на календарний місяць. Користувач визначає граничні суми витрат за категоріями (наприклад, 5000 грн на продукти харчування), а застосунок відстежує поточний рівень виконання бюджету та інформує про наближення або перевищення встановленого ліміту. Такий підхід дисциплінує користувача та сприяє формуванню раціональної споживчої поведінки.

Постановка фінансових цілей є наступною важливою функцією. Користувач визначає бажану суму накопичення та цільову дату її досягнення (наприклад, накопичити 30 000 грн на придбання техніки протягом шести

місяців). Застосунок розраховує необхідну періодичну суму внесків та відстежує прогрес виконання цілі. Опційно може бути налаштовано автоматичне списання визначеної суми з балансу через задані інтервали часу [3].

Аналітика витрат забезпечує візуалізацію фінансової поведінки користувача. Найбільш поширеними формами представлення аналітичних даних є кругова діаграма витрат за категоріями, стовпчасті діаграми порівняння витрат за періодами, а також списки найбільших окремих операцій. Аналітичні дані допомагають користувачу виявити надмірні витрати, скоригувати споживчу поведінку та оптимізувати бюджет.

Безпека даних має критичне значення для систем управління особистими фінансами, оскільки фінансова інформація належить до категорії конфіденційних даних. Застосунок повинен забезпечувати надійну автентифікацію користувача, захист збережених даних від несанкціонованого доступу, ізоляцію даних між різними обліковими записами на одному пристрої, а також шифрування при передачі інформації через мережу. У разі реалізації функції хмарного резервного копіювання необхідно забезпечити належний рівень контролю доступу на стороні серверу. [14], [15]

Окремої уваги заслуговує застосування методів інтелектуального аналізу для підвищення зручності користування застосунком. На основі історії транзакцій можуть формуватися автоматичні підказки опису операцій, пропозиції найбільш імовірної категорії, виявлення регулярних платежів, прогнозування майбутніх витрат та формування персоналізованих порад щодо економії. Такі функції перетворюють застосунок з пасивного інструменту обліку на активного помічника користувача. [26]

1.2 Моделювання предметної області

Для формалізації предметної області управління особистими фінансами застосовано об’єктно-орієнтований підхід до аналізу з використанням нотації UML (Unified Modeling Language) Моделювання дозволяє визначити основних акторів системи, їхні функціональні можливості, бізнес-процеси та взаємодії між компонентами розроблюваного застосунку. [5]

У розроблюваному застосунку визначено одного основного актора - користувача. Користувач - це фізична особа, яка завантажила застосунок на власний смартфон, пройшла процедуру реєстрації та використовує застосунок для ведення обліку особистих фінансів. Застосунок не передбачає наявності ролей адміністратора, модератора або інших спеціальних користувачів, оскільки кожен користувач має доступ виключно до власних фінансових даних та не взаємодіє з даними інших користувачів.

За результатами аналізу предметної області побудовано діаграму варіантів використання застосунку, яку представлено на рис. 1.1. Діаграма відображає основний набір функціональних можливостей, доступних користувачу: операції автентифікації (реєстрація, вхід, скидання пароля), управління транзакціями (додавання, перегляд, редагування, видалення), управління категоріями, бюджетами та фінансовими цілями, отримання аналітичних звітів, взаємодія з інтелектуальним асистентом, конфігурація параметрів профілю та хмарне резервне копіювання даних.

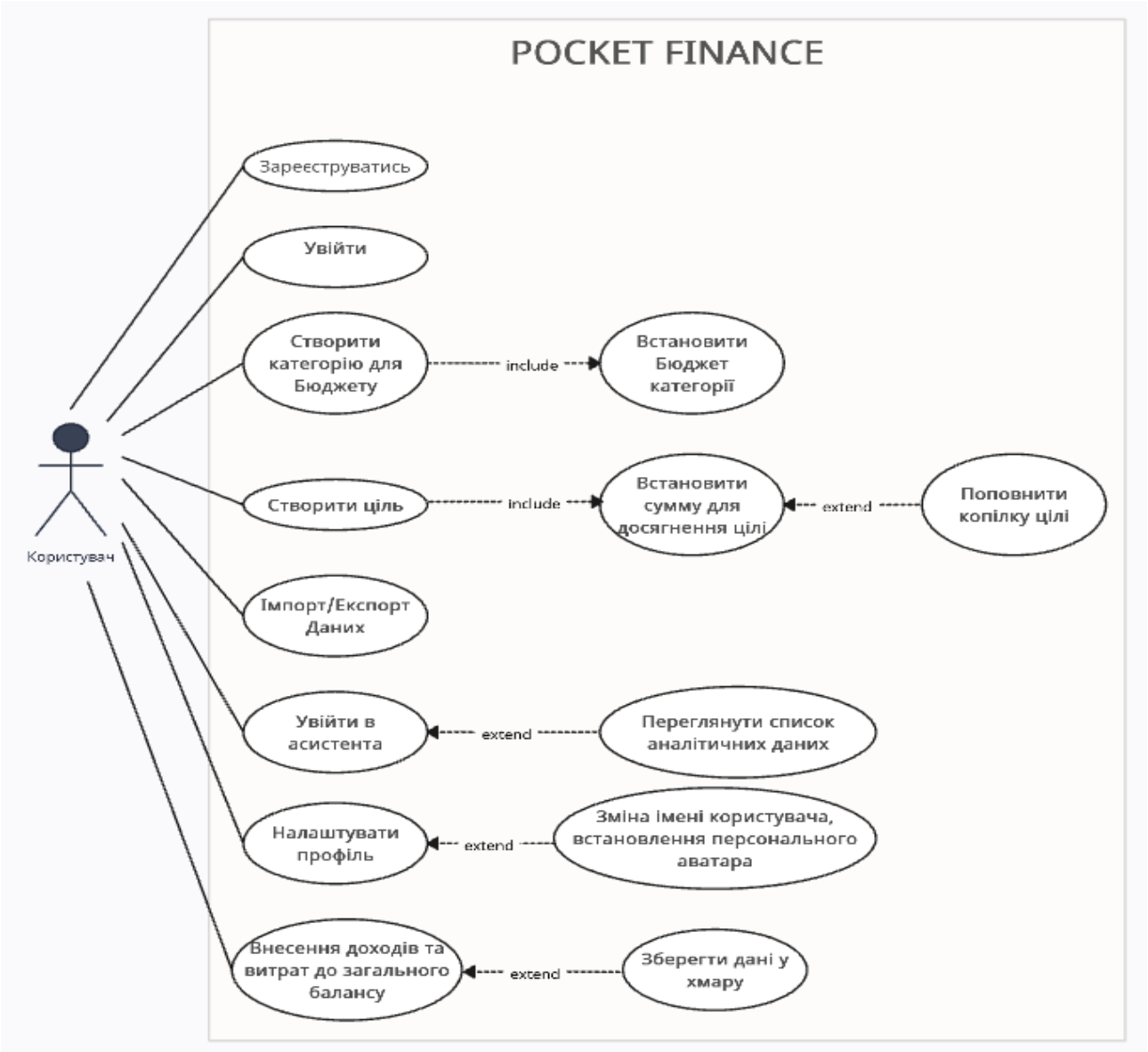

Рис. 1.1 - Діаграма варіантів використання застосунку

Як видно з діаграми, основні функціональні можливості розроблюваного застосунку охоплюють весь життєвий цикл управління особистими фінансами користувача - від реєстрації до отримання аналітичних звітів. Усі дії виконуються одним актором, що відповідає характеру персонального застосунку.

Бізнес-процес роботи з застосунком починається з реєстрації нового користувача через електронну пошту та пароль. Після підтвердження адреси через лист верифікації користувач отримує доступ до основного функціоналу застосунку. Подальша взаємодія з застосунком відбувається через головний

екран, на якому відображається поточний баланс, останні операції та поточні курси валют, отримані з API Національного банку України. [4]

Процес додавання нової транзакції включає послідовність кроків: виклик форми додавання через відповідну кнопку, вибір типу операції (витрата або дохід), введення суми, вибір категорії, встановлення дати, опційне додавання опису та збереження операції до локальної бази даних. На рис. 1.2 представлено діаграму послідовності для процесу додавання транзакції, яка ілюструє взаємодію компонентів застосунку від моменту дії користувача до оновлення інтерфейсу.

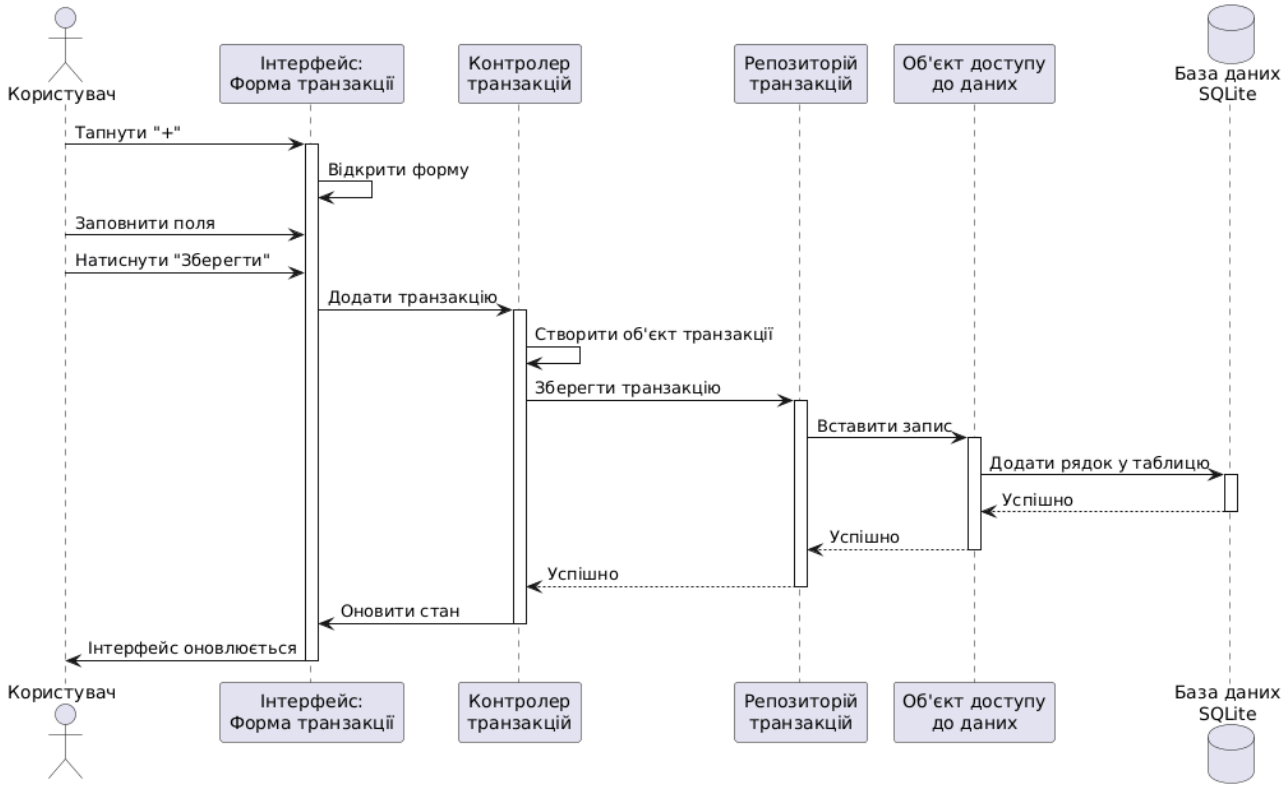

Рис. 1.2 - Діаграма послідовності для додавання транзакції

На рис. 1.2 показано, як подія користувача проходить через шари застосунку: інтерфейс передає подію до BLoC-компонента, який створює сутність транзакції та делегує її збереження репозиторію. Репозиторій, у свою чергу, звертається до DAO-класу, який виконує SQL-запит до бази даних SQLite. Після успішного збереження дані повертаються у зворотному напрямку, BLoC оновлює свій стан, а інтерфейс перемальовується.

Для опису життєвого циклу транзакції було розроблено діаграму активності (рис. 1.3), яка демонструє стани, через які проходить операція від моменту створення до її фіксації у базі даних. Діаграма містить вузли прийняття рішень для валідації введених користувачем даних та обробки помилок.

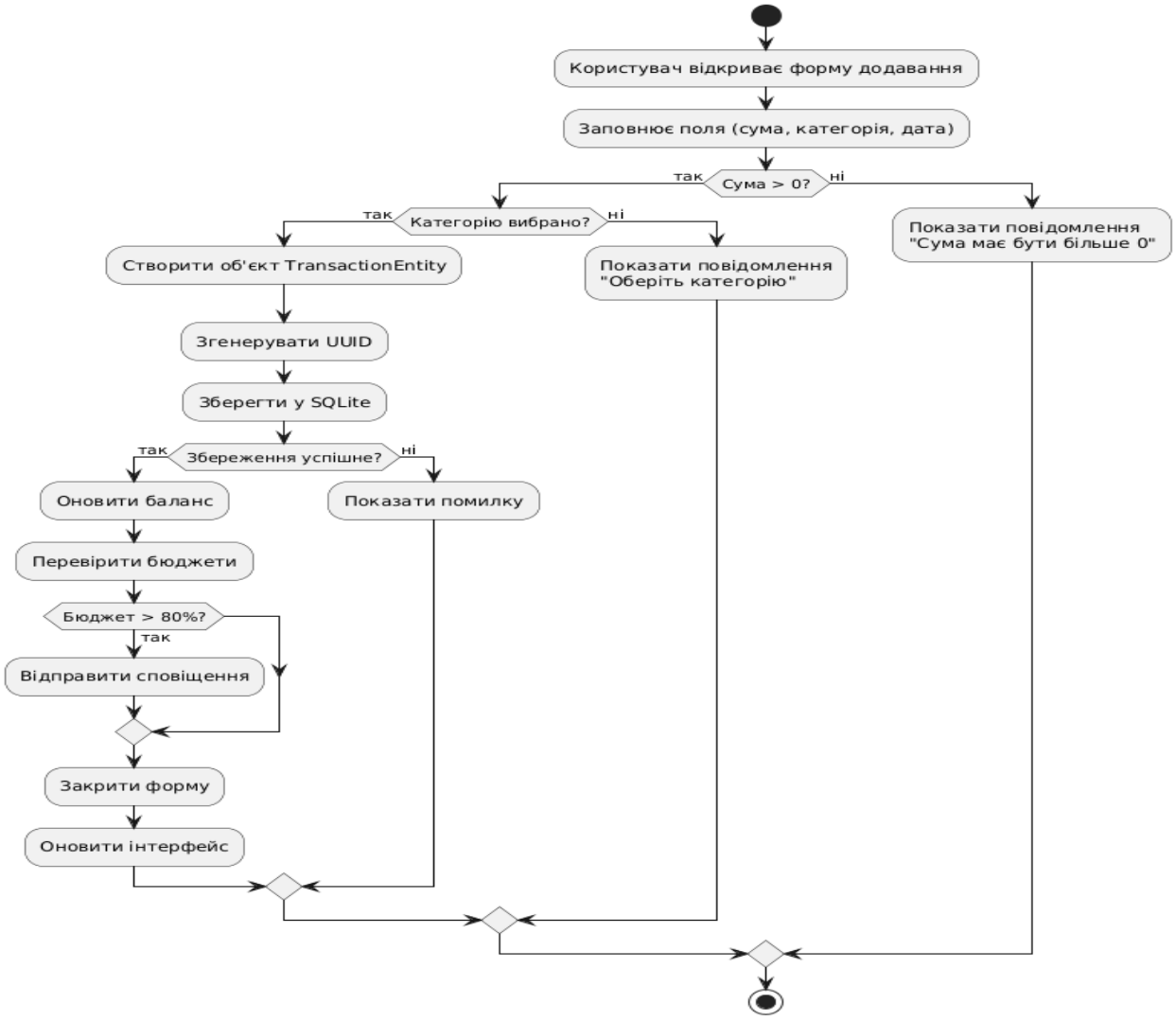

Рис. 1.3 - Діаграма активності процесу обробки транзакції

Як видно з рис. 1.3, у процесі обробки транзакції здійснюється валідація введених даних. Сума має бути додатним числом, а категорія - обов’язково вибраною з переліку доступних. У разі успішного збереження транзакції відбувається оновлення поточного балансу та перевірка стану бюджетів за відповідною категорією. Якщо рівень виконання бюджету наближається до встановленого порогу (80% або 100%), користувач отримує push-сповіщення.

1.3 Огляд існуючих аналогічних рішень

Було проаналізовано існуючі мобільні застосунки для управління особистими фінансами з метою виявлення їх функціональних можливостей, технологічних особливостей, переваг та недоліків. Для аналізу обрано три найбільш популярні рішення, які за даними магазину Google Play мають десятки мільйонів завантажень: Money Manager (Expense & Budget) від Realbyte Inc., Wallet від BudgetBakers та Spendee [3], [6].

Wallet від BudgetBakers - один з найбільш функціональних застосунків у досліджуваному сегменті. Застосунок забезпечує облік транзакцій, планування бюджетів, постановку фінансових цілей, аналітику витрат та інтеграцію з банківськими рахунками через сервіс Plaid. Інтеграція з банками є унікальною особливістю, проте вона підтримується переважно для західноєвропейських та американських фінансових установ і недоступна для українських банків. Серед переваг застосунку слід відзначити продуманий інтерфейс, підтримку понад 100 валют та синхронізацію між пристроями. Серед недоліків - наявність платної підписки за низку функцій (шаблони транзакцій, експорт у PDF, розширена аналітика) вартістю близько 3 доларів США на місяць, а також відсутність повної української локалізації.

Money Manager (Expense & Budget) від компанії Realbyte Inc. - один з найпопулярніших застосунків у досліджуваному сегменті, що має понад 22 мільйони завантажень у Google Play та середню оцінку 4,6 бала. Застосунок забезпечує облік доходів і витрат, планування бюджетів за категоріями, формування звітів та графічну аналітику. Особливістю застосунку є застосування принципу подвійного запису (double-entry bookkeeping), характерного для бухгалтерського обліку, а також управління кількома рахунками та активами. Серед переваг - висока надійність, докладна аналітика, можливість резервного копіювання. Серед недоліків - реклама у безкоштовній версії, перевантажений для пересічного користувача інтерфейс через

бухгалтерську модель, відсутність української локалізації та інтеграції з курсами Національного банку України.

Spendee - застосунок із виразним фокусом на візуалізацію фінансових даних. Інтерфейс побудований навколо графіків та діаграм, що забезпечує наочне представлення витрат за категоріями та періодами. Серед переваг - естетичний дизайн, наявність функції спільних бюджетів для родини, підтримка криптовалют. Серед недоліків - частина функцій доступна лише у платній версії, відсутність повної української локалізації та інтеграції з НБУ.

Для систематизованого порівняння функціональних можливостей розглянутих застосунків та розроблюваного рішення складено порівняльну таблицю (табл. 1.1).

Таблиця 1.1

Порівняння функціональних можливостей застосунків для управління особистими фінансами

Функція	Wallet	Money Manager	Spendee	Pocket Finance
Облік транзакцій	+	+	+	+
Планування бюджетів	+	+	+	+
Фінансові цілі	+ (платно)	+	+ (платно)	+
Аналітика витрат	+	+	+	+
Українська локалізація	частково	–	-	+ (повна)
Інтеграція з НБУ	-	-	-	+
Інтелектуальний асистент	-	-	-	+
Безкоштовність	частково	з рекламою	частково	+ (повна)
Темне оформлення	+	+	+	+
Робота без Інтернету	+	+	частково	+
Хмарне копіювання	+ (платно)	+	+	+
PDF-звіти	+ (платно)	+	+ (платно)	+

Проведений аналіз свідчить про те, що жодне з розглянутих рішень не забезпечує повного набору функцій, необхідних для українського користувача. Зокрема, відсутні механізми інтеграції з офіційними курсами Національного

банку України, повна українська локалізація та елементи інтелектуальної підтримки прийняття рішень. Частина функціонала надається на платній основі, що обмежує доступність для широкого кола користувачів.

Розроблюваний застосунок «Pocket Finance» позиціонується як безкоштовна україномовна альтернатива з повним набором ключових функцій. Серед його переваг порівняно з аналогами:

- повністю безкоштовний доступ до всього функціоналу без реклами та платних підписок;

- повна локалізація українською мовою всіх інтерфейсів та повідомлень;

- інтеграція з офіційним API Національного банку України для отримання актуальних курсів валют; [4]

- інтелектуальний асистент із підтримкою української мови, що працює без підключення до мережі Інтернет;

- автоматичні підказки опису транзакцій на основі історії з визначенням найімовірнішої категорії;

- повна функціональність без підключення до Інтернету з опційним хмарним резервним копіюванням;

- генерація PDF-звітів за період з графіками та підсумками за категоріями; [21]

- строга ізоляція даних між обліковими записами на одному пристрої.

Ринок мобільних застосунків для управління особистими фінансами розвинутий, проте існують ніші, де великі гравці неактивні, зокрема - локалізація під українські реалії з повною підтримкою національної валюти, інтеграцією з НБУ та повною українською мовою інтерфейсу. Розроблюваний застосунок орієнтований саме на цю нішу. [7]

1.4 Постановка завдання

На основі результатів аналізу предметної області та огляду існуючих рішень сформульовано вимоги до розроблюваного програмного продукту. Вимоги поділяються на функціональні (що повинен виконувати застосунок) та нефункціональні (якісні характеристики застосунку).

Функціональні вимоги до розроблюваного застосунку включають:

- реєстрацію користувача через електронну пошту та пароль з обов'язковою верифікацією через лист, що надсилається на вказану адресу;

- автентифікацію зареєстрованого користувача з можливістю скидання пароля через електронну пошту;

- додавання фінансових транзакцій типу «витрата» або «дохід» із вказанням суми, категорії, дати та опційного опису;

- перегляд списку всіх транзакцій з можливістю фільтрації за датою, типом операції та категорією;

- редагування та видалення раніше створених транзакцій;

- створення власних категорій з вибором іконки та кольору; автоматичне створення набору стандартних категорій при першому вході нового користувача;

- встановлення місячних бюджетів за категоріями витрат із відображенням рівня виконання у відсотках;

- відправлення push-сповіщень при досягненні 80% та 100% бюджету за категорією; [20]

- створення фінансових цілей із цільовою сумою, опційним дедлайном та можливістю налаштування автоматичних регулярних внесків;

- відображення аналітики витрат у вигляді кругових діаграм за категоріями, переліку найбільших операцій та графіків динаміки за період;

- конвертація сум між валютами (UAH, USD, EUR) за курсами Національного банку України; [4]

- інтелектуальний асистент для відповідей на запити користувача природною українською мовою;

- автоматичні підказки опису транзакції на основі історії з пропозицією категорії; [26]

- резервне копіювання даних користувача до хмарного сховища Firebase Firestore; [15]

- відновлення даних з хмари після перевстановлення застосунку або переходу на інший пристрій;

- генерація PDF-звіту за обраний період з графіками та таблицею операцій; [21]

- щоденні нагадування у визначений користувачем час про необхідність обліку транзакцій;

- налаштування параметрів профілю користувача - імені, аватара, валюти за замовчуванням, теми оформлення.

Нефункціональні вимоги до розроблюваного застосунку включають:

- платформа реалізації - Android, мінімальна версія операційної системи 6.0 (API 23);

- мова інтерфейсу - українська;

- підтримка темної та світлої тем оформлення;

- робота без підключення до мережі Інтернет - усі основні функції доступні офлайн;

- час відгуку інтерфейсу - не більше 200 мс на дії користувача;

- розмір APK-файлу - не більше 30 МБ;

- ізоляція даних між обліковими записами на одному пристрої - дані одного користувача недоступні іншому;

- використання Firebase Authentication для безпечного зберігання облікових даних; [14]

- архітектура застосунку відповідно до принципів Clean Architecture [9];

- управління станом через патерн BLoC. [12]

Технічні вимоги визначають обмеження та параметри, що впливають на архітектурні рішення. Застосунок має підтримувати одночасну роботу з тисячами транзакцій без помітної деградації продуктивності. Локальна база даних SQLite повинна забезпечувати швидкий доступ до даних навіть на пристроях бюджетного сегмента [8], [22]. Хмарна синхронізація має виконуватися пакетно для оптимізації мережевого трафіку та зменшення кількості запитів до серверу.

Вимоги до інтерфейсу користувача включають інтуїтивно зрозумілу навігацію за принципами Material Design 3, швидкий доступ до основних функцій з головного екрана, наочне відображення поточного балансу та останніх операцій, підтвердження деструктивних дій (видалення транзакцій, скидання даних), а також інформативні повідомлення про помилки при некоректному введенні даних. [18]

Вимоги до безпеки включають обов’язкову автентифікацію для доступу до функціональності застосунку, верифікацію адреси електронної пошти при реєстрації, ізоляцію даних на рівні бази даних через зовнішній ключ user_id, налаштування правил безпеки Firestore для контролю доступу до хмарних даних, а також збереження конфіденційних даних (паролі) виключно у вигляді хеш-значень засобами Firebase Authentication.

Висновки до розділу 1. У розділі проведено аналіз предметної області управління особистими фінансами, розглянуто основні процеси, що відбуваються при веденні обліку фінансів - реєстрація транзакцій, бюджетування, постановка цілей, аналіз даних. Виконано моделювання предметної області з використанням нотації UML - побудовано діаграму варіантів використання, діаграму послідовності для процесу додавання транзакції та діаграму активності. Проведено порівняльний аналіз трьох популярних мобільних застосунків для управління фінансами (Money Manager, Wallet, Spendee), за результатами якого виявлено їх сильні та слабкі сторони. На основі аналізу сформульовано 18 функціональних та 10 нефункціональних вимог до розроблюваного застосунку. Визначена ринкова ніша для українського застосунку з безкоштовним базовим функціоналом, інтеграцією з курсами НБУ та інтелектуальними підказками.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

На основі результатів аналізу предметної області, проведеного у першому розділі, виконано проєктування інформаційного забезпечення розроблюваного застосунку. Першим етапом проєктування є побудова логічної моделі даних, яка визначає основні сутності предметної області, їх атрибути та зв’язки між ними. [16]

Логічна модель даних розроблюваного застосунку має реляційну структуру, що зумовлено характером предметної області - фінансові операції мають чітко визначену структуру з фіксованим набором атрибутів, а зв’язки між сутностями мають типи «один-до-багатьох» та «багато-до-багатьох». Реляційна модель забезпечує цілісність даних через механізми первинних та зовнішніх ключів, підтримку транзакцій ACID та можливість виконання складних аналітичних запитів засобами мови SQL.

У результаті аналізу предметної області визначено п’ять основних сутностей: USERS (користувачі), CATEGORIES (категорії транзакцій), TRANSACTIONS (фінансові операції), BUDGETS (бюджети) та GOALS (фінансові цілі). Розглянемо кожну сутність детальніше.

Сутність USERS містить інформацію про зареєстрованих користувачів застосунку. Атрибути: унікальний ідентифікатор (id), електронна пошта (email), відображуване ім’я (displayName), зображення аватара у форматі Base64 (avatar), а також дата створення облікового запису (createdAt). Електронна пошта повинна бути унікальною у межах системи.

Сутність CATEGORIES представляє категорії транзакцій. Кожна категорія належить конкретному користувачу, що забезпечує можливість створення персоналізованого набору категорій. Атрибути: id, ім’я категорії (name), іконка з

набору Material Icons (icon), колір (color), ознака користувацької категорії (isCustom), тип категорії (expense/income/savings).

Сутність TRANSACTIONS є центральною у моделі даних і містить інформацію про всі фінансові операції користувача. Атрибути: id, посилання на користувача (userId), посилання на категорію (categoryId), тип операції (type), сума (amount), дата виконання операції (date), опційний опис (description), дата створення запису (createdAt).

Сутність BUDGETS зберігає інформацію про встановлені користувачем бюджетні обмеження. Кожен бюджет асоціюється з певною категорією та визначає максимальну суму витрат на конкретний місяць. Атрибути: id, userId, categoryId, гранична сума витрат (limitAmount), фактично витрачена сума (spentAmount), період (month).

Сутність GOALS представляє довгострокові фінансові цілі користувача. Атрибути: id, userId, заголовок цілі (title), цільова сума (targetAmount), накопичена сума (savedAmount), опційний дедлайн (deadline), емодзі для візуалізації (emoji), опційне посилання на категорію (categoryId), а також параметри регулярних автоматичних внесків (recurringAmount, recurringPeriod, recurringInterval).

Між сутностями встановлено такі зв’язки:

- USERS - CATEGORIES: один користувач має багато категорій (відношення «1:N»);

- USERS - TRANSACTIONS: один користувач створює багато транзакцій (відношення «1:N»);

- USERS - BUDGETS: один користувач встановлює багато бюджетів (відношення «1:N»);

- USERS - GOALS: один користувач створює багато фінансових цілей (відношення «1:N»);

- CATEGORIES - TRANSACTIONS: одна категорія класифікує багато транзакцій (відношення «1:N»);

- CATEGORIES - BUDGETS: одна категорія обмежується одним бюджетом на місяць (відношення «1:1» у межах періоду);

- CATEGORIES - GOALS: одна категорія може бути пов’язана з декількома фінансовими цілями (відношення «1:N»).

Графічно логічна модель даних представлена у вигляді діаграми на рис. 2.1.

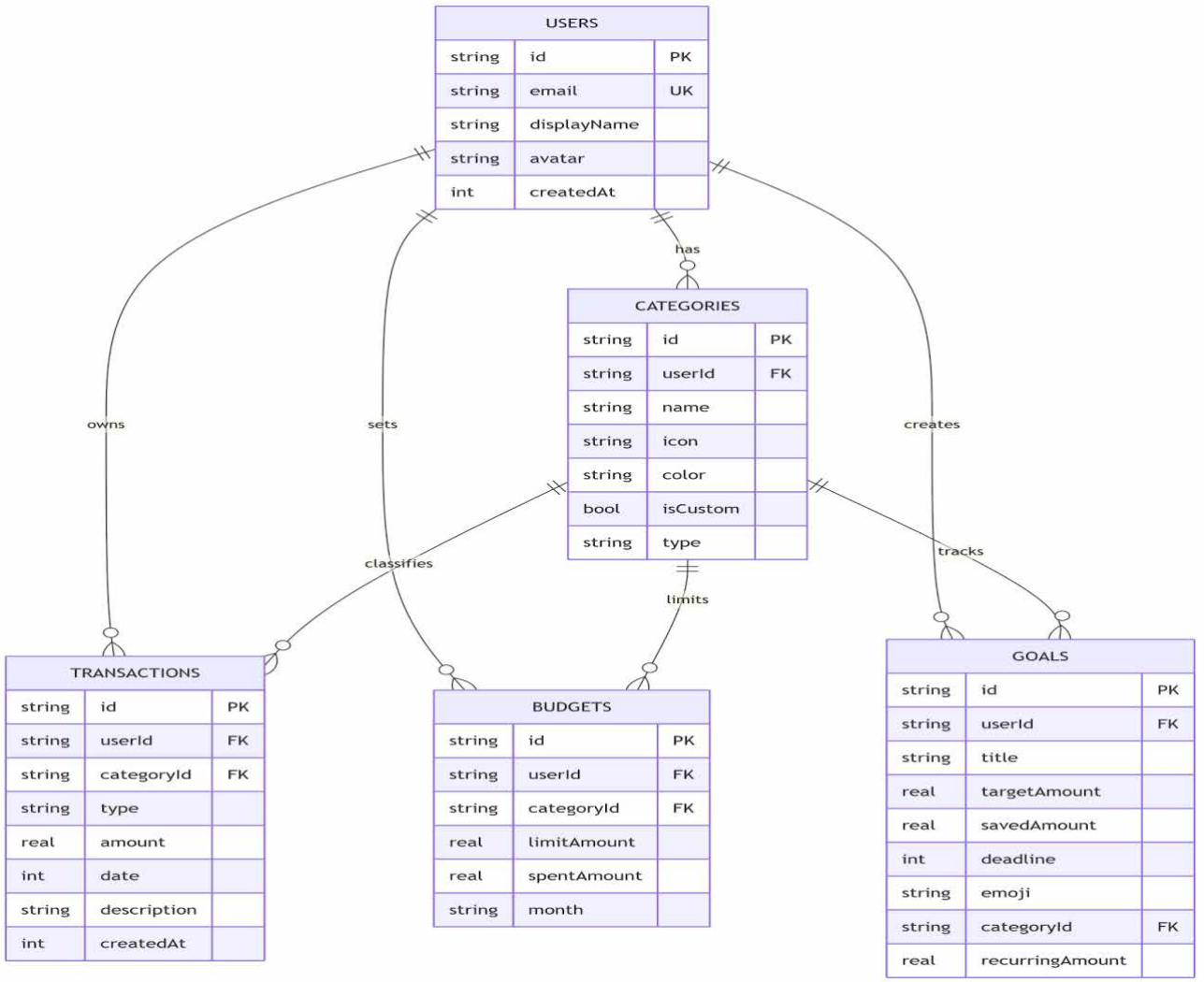

Рис. 2.1 - Діаграма логічної моделі даних застосунку

На рис. 2.1 первинні ключі позначені як PK (Primary Key), зовнішні ключі - як FK (Foreign Key), унікальні поля - як UK (Unique Key). Подвійна вертикальна риска з кружальцем, у нотації Mermaid позначає зв’язок типу «один до багатьох».

Логічна модель даних відповідає вимогам третьої нормальної форми (3НФ). Усі неключові атрибути функціонально залежать виключно від первинних ключів, відсутні транзитивні залежності, кожен запис у відповідній таблиці є атомарним. Це забезпечує мінімізацію надлишковості даних та запобігає аномаліям при операціях вставки, оновлення та видалення.

У всіх таблицях, крім USERS, передбачено поле userId, що забезпечує ізоляцію даних між обліковими записами на одному пристрої. Кожен запит до бази даних обов’язково фільтрується за userId поточного користувача, що унеможливлює доступ до даних інших користувачів.

2.2 Вибір системи управління базами даних

Розглянуто варіанти систем управління базами даних для зберігання даних розроблюваного мобільного застосунку. Аналіз проводився за критеріями: обсяг та структура даних, продуктивність на мобільних пристроях, наявність інтеграції з фреймворком Flutter, ліцензійні обмеження та трудовитрати на впровадження.

Серед розглянутих варіантів - реляційні СУБД (SQLite, MySQL Lite for Mobile), документні NoSQL-сховища (Hive, ObjectBox) та хмарні рішення (Firebase Realtime Database, Firestore у режимі офлайн-кешування).

Для зберігання локальних даних обрано реляційну базу даних SQLite. Цей вибір обумовлений такими чинниками:

- SQLite є вбудованою (embedded) базою даних, що зберігається у файлі на пристрої користувача та не потребує окремого серверу [8];

- SQLite попередньо встановлена у всіх сучасних мобільних операційних системах (Android, iOS), що мінімізує розмір застосунку;

- структура фінансових даних має реляційний характер із чітко визначеними зв’язками між сутностями, що відповідає реляційній моделі;

- SQLite підтримує транзакції ACID, що забезпечує цілісність даних навіть у разі некоректного завершення роботи застосунку;

- наявна офіційна бібліотека sqflite для інтеграції з фреймворком Flutter, що активно підтримується спільнотою; [22]

- SQLite має ліцензію Public Domain, що не накладає обмежень на комерційне використання;

- продуктивність SQLite є достатньою для роботи з обсягами даних типового користувача - десятки тисяч транзакцій без помітних затримок [9].

Для хмарного резервного копіювання даних обрано Firebase Firestore - документну NoSQL-базу даних від Google. Цей вибір обумовлений такими чинниками:

- Firestore надає безкоштовний рівень обслуговування, достатній для потреб розроблюваного застосунку (50 000 читань на день, 20 000 записів) [13];

- Firestore має офіційну інтеграцію з фреймворком Flutter через пакет cloud_firestore;

- наявна вбудована система автентифікації Firebase Authentication, що спрощує реалізацію реєстрації та входу; [14]

- Firestore підтримує правила безпеки (Security Rules) для контролю доступу на рівні документів; [15]

- Firestore автоматично обробляє синхронізацію даних між пристроями та режим офлайн-роботи.

2.3 Розробка структури інформаційної бази

На основі побудованої логічної моделі даних розроблено структуру таблиць реляційної бази даних SQLite. Кожна сутність логічної моделі реалізована у вигляді окремої таблиці із визначеними полями, типами даних та обмеженнями цілісності.

Таблиця users зберігає основну інформацію про користувачів застосунку. Структура полів представлена нижче:

- id - унікальний ідентифікатор користувача у форматі UUID, тип TEXT, обмеження PRIMARY KEY;

- email - електронна пошта користувача, тип TEXT, обмеження NOT NULL та UNIQUE;

- password_hash - хеш-значення пароля користувача (для локального режиму роботи; у Firebase-режимі паролі зберігаються у Firebase Authentication), тип TEXT;

- currency - валюта за замовчуванням (UAH, USD або EUR), тип TEXT, значення за замовчуванням 'UAH';

- created_at - дата створення облікового запису у форматі timestamp (мілісекунди від 1970-01-01), тип INTEGER.

Таблиця categories зберігає категорії транзакцій. Кожна категорія належить конкретному користувачу. Структура полів:

- id - унікальний ідентифікатор категорії, тип TEXT, PRIMARY KEY;

- user_id - посилання на власника категорії, тип TEXT, NOT NULL;

- name - назва категорії, тип TEXT, NOT NULL;

- icon - назва іконки з набору Material Icons, тип TEXT, NOT NULL;

- color - колір категорії у форматі HEX (#FF6B6B), тип TEXT, NOT NULL;

- is_custom - ознака користувацької категорії (1 - створена користувачем, 0 - стандартна), тип INTEGER, значення за замовчуванням 0;

- type - тип категорії, тип TEXT з обмеженням CHECK на значення ('expense', 'income', 'savings').

Таблиця transactions є центральною у структурі бази даних. Структура полів:

- id - унікальний ідентифікатор операції, тип TEXT, PRIMARY KEY;

- user_id - посилання на власника операції, тип TEXT, NOT NULL;

- type - тип операції (витрата, дохід, заощадження), тип TEXT з обмеженням CHECK;

- amount - сума операції, тип REAL з обмеженням CHECK (amount > 0);

- category_id - посилання на категорію, тип TEXT, FOREIGN KEY до categories(id);

- date - дата операції, тип INTEGER (timestamp);

- description - опис операції, тип TEXT, значення за замовчуванням '';

- created_at - дата створення запису, тип INTEGER;

- is_synced - ознака синхронізації з хмарою (0/1), тип INTEGER, значення за замовчуванням 0.

Обмеження CHECK (amount > 0) забезпечує коректність даних на рівні бази даних - будь-яка спроба зберегти транзакцію з від’ємною або нульовою сумою буде відхилена. Тип операції визначається полем type незалежно від суми, що спрощує побудову аналітичних запитів.

Таблиця budgets зберігає бюджети користувачів. Структура полів:

- id - унікальний ідентифікатор бюджету, тип TEXT, PRIMARY KEY;

- user_id - посилання на користувача, тип TEXT, NOT NULL;

- category_id - посилання на категорію, тип TEXT, FOREIGN KEY до categories(id) з опцією ON DELETE CASCADE;

- limit_amount - гранична сума витрат, тип REAL з обмеженням CHECK (limit_amount > 0);

- spent_amount - фактично витрачена сума, тип REAL, значення за замовчуванням 0;

- month - період у форматі 2026-05 тип DATE, NOT NULL.

Таблиця budgets містить комбіноване обмеження UNIQUE (user_id, category_id, month), яке забороняє створення двох бюджетів для однієї категорії в одному місяці одним користувачем. Опція ON DELETE CASCADE забезпечує автоматичне видалення бюджетів при видаленні відповідної категорії.

Таблиця goals зберігає фінансові цілі користувачів. Структура полів:

- id - унікальний ідентифікатор цілі, тип TEXT, PRIMARY KEY;

- user_id - посилання на користувача, тип TEXT, NOT NULL;

- title - заголовок цілі (наприклад, «Новий ноутбук»), тип TEXT, NOT NULL;

- target_amount - цільова сума накопичення, тип REAL з CHECK (target_amount > 0);

- saved_amount - поточна накопичена сума, тип REAL, значення за замовчуванням 0;

- deadline - опційний дедлайн досягнення цілі у форматі timestamp, тип INTEGER (допускається NULL);

- emoji - емодзі для візуалізації цілі, тип TEXT, значення за замовчуванням '🎯';

- category_id - опційне посилання на категорію «заощадження», тип TEXT;

- recurring_amount - сума регулярного автоматичного внеску, тип REAL (допускається NULL);

- recurring_period - період регулярних внесків (week/month), тип TEXT;

- recurring_interval - інтервал між внесками, тип INTEGER.

Окрім п’яти основних таблиць у структурі бази даних передбачено допоміжну таблицю sync_log, що використовується для відстеження стану хмарної синхронізації. Таблиця містить інформацію про те, які записи були передані до хмарного сховища, що дозволяє оптимізувати процес резервного копіювання - у хмару передаються лише ті записи, що змінилися з моменту останньої синхронізації.

2.4 Фізична структура бази даних

Фізична реалізація бази даних виконана засобами SQL - діалекту SQLite. Команди створення таблиць виконуються одноразово при першому запуску застосунку у методі onCreate класу AppDatabase. Повний лістинг SQL-команд створення таблиць наведено у Додатку А. Схему фізичної структури бази даних з усіма таблицями, полями, типами даних та зв’язками за зовнішніми ключами наведено на рис. 2.2.

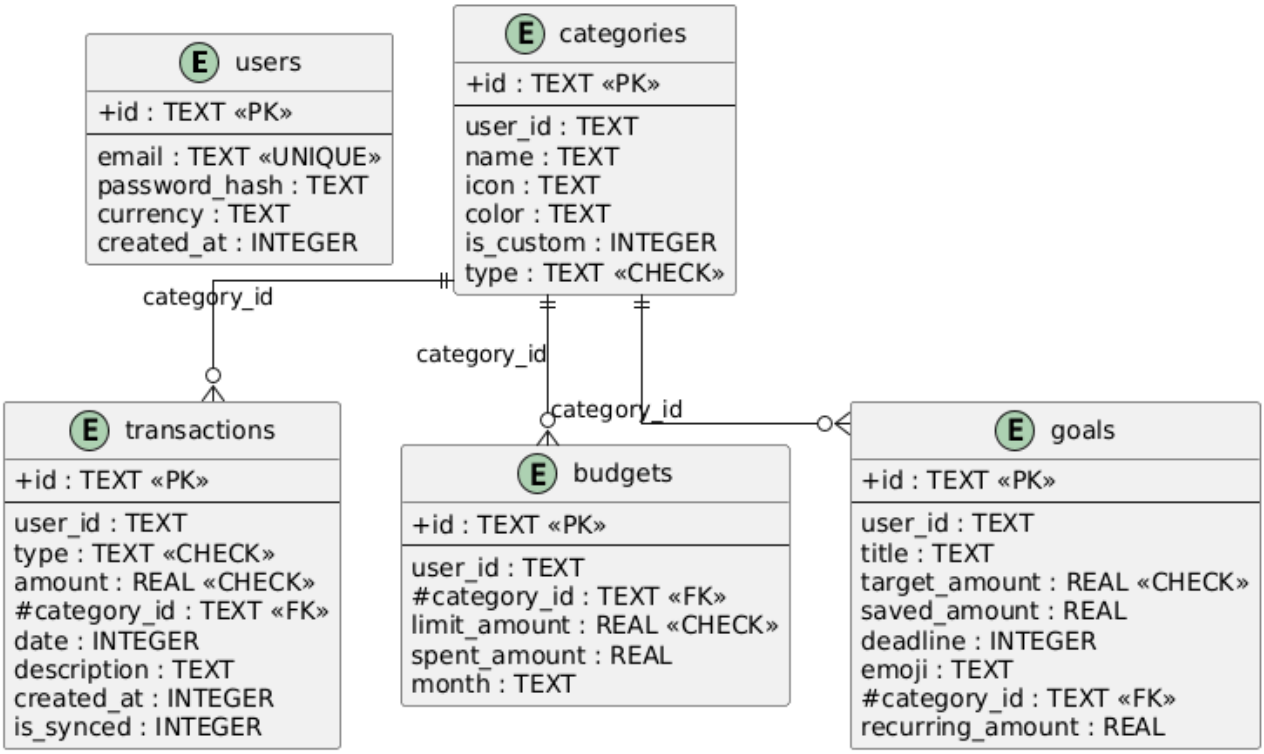

Рис. 2.2 - Схема фізичної структури бази даних

Для оптимізації продуктивності запитів створено індекси на полях, які найчастіше використовуються у запитах фільтрації та сортування. Аналіз типових запитів застосунку показав, що найчастіше виконуються такі типи операцій: вибірка всіх транзакцій конкретного користувача, сортування транзакцій за датою, групування транзакцій за категоріями, отримання бюджетів конкретного користувача за поточний місяць. На основі цього аналізу створено такі індекси:

- idx_tx_user на полі transactions(user_id) - для фільтрації транзакцій конкретного користувача;

- idx_tx_date на полі transactions(date) - для сортування транзакцій за датою;

- idx_tx_category на полі transactions(category_id) - для аналітики за категоріями;

- idx_tx_type на полі transactions(type) - для розділення витрат і доходів;

- idx_cat_user на полі categories(user_id) - для отримання набору категорій користувача;

- idx_budget_user на полі budgets(user_id) - для отримання бюджетів користувача;

- idx_budget_month на полі budgets(month) - для вибірки бюджетів поточного місяця;

- idx_goals_user на полі goals(user_id) - для отримання списку цілей користувача.

Створення індексів є компромісом між швидкістю операцій читання та швидкістю операцій запису. Кожен індекс прискорює пошук за відповідним полем, але одночасно сповільнює операції INSERT, UPDATE та DELETE, оскільки індекс потребує оновлення при кожній зміні даних. Тому індекси створюються лише на тих полях, по яких виконуються запити фільтрації або сортування.

При першому вході нового користувача автоматично створюється набір стандартних категорій транзакцій. Це забезпечує можливість використання застосунку без необхідності попереднього налаштування. Стандартний набір включає 11 категорій: вісім категорій витрат («Їжа», «Транспорт», «Покупки», «Житло», «Розваги», «Здоров’я», «Освіта», «Інше»), три категорії доходів («Зарплата», «Подарунок», «Інвестиції»). Кожна стандартна категорія має наперед визначену іконку з набору Material Icons та колір.

Підтримка зовнішніх ключів у SQLite активується через директиву PRAGMA foreign_keys = ON при відкритті з’єднання з базою даних. Без активації цієї директиви SQLite не виконує перевірку зовнішніх ключів, що може призвести до порушення цілісності даних.

При оновленні застосунку до нової версії може виникнути потреба у зміні структури бази даних - додаванні нових полів, таблиць або індексів. Для цього використовується механізм міграцій SQLite, який реалізується через метод onUpgrade класу AppDatabase. Метод приймає поточну та нову версії бази даних і виконує необхідні SQL-команди для перетворення структури зі старої версії у нову.

Висновки до розділу 2. У розділі виконано проєктування інформаційного забезпечення розроблюваного застосунку. Побудовано логічну модель даних у вигляді діаграми «сутність-зв’язок», яка містить п’ять основних сутностей (USERS, CATEGORIES, TRANSACTIONS, BUDGETS, GOALS) та визначає зв’язки між ними. Обґрунтовано вибір реляційної СУБД SQLite для локального зберігання даних та хмарної бази Firestore для резервного копіювання. Розроблено структуру кожної таблиці з визначенням полів, типів даних та обмежень. Спроєктовано фізичну структуру бази даних із створенням індексів для оптимізації найбільш частих запитів. Передбачено механізм автоматичного створення набору стандартних категорій при першому вході нового користувача та механізм міграцій для оновлення схеми бази даних.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмної системи

Розроблюваний застосунок «Pocket Finance» побудовано за архітектурою Clean Architecture, запропонованою Робертом Мартіном [9]. Цей підхід забезпечує чітке розділення коду на шари з різним рівнем абстракції та контролює напрямок залежностей між ними. Кожен шар має визначену відповідальність і обмежений у можливостях звертання до інших шарів.

Архітектура застосунку розділена на три основні шари: presentation (представлення), domain (бізнес-логіка) та data (доступ до даних). Шар presentation містить компоненти інтерфейсу користувача та класи управління станом (BLoC). Шар domain є серцем застосунку та містить сутності, інтерфейси репозиторіїв і сервіси бізнес-логіки. Шар data містить конкретні реалізації репозиторіїв, класи доступу до даних (DAO) та сервіси для роботи з мережею.

Принцип залежностей у Clean Architecture виглядає таким чином: шар presentation може звертатися до шару domain, шар data реалізує інтерфейси, визначені у шарі domain, проте шар domain не залежить від інших шарів. Це забезпечує те, що бізнес-логіка не змінюється при заміні бази даних або інтерфейсу користувача. Така архітектура полегшує тестування коду, оскільки кожен шар може бути протестований окремо з використанням mock-об’єктів для залежностей.

Графічне представлення архітектури у вигляді діаграми класів наведено на рис. 3.1. На діаграмі показані основні класи кожного шару та зв’язки між ними.

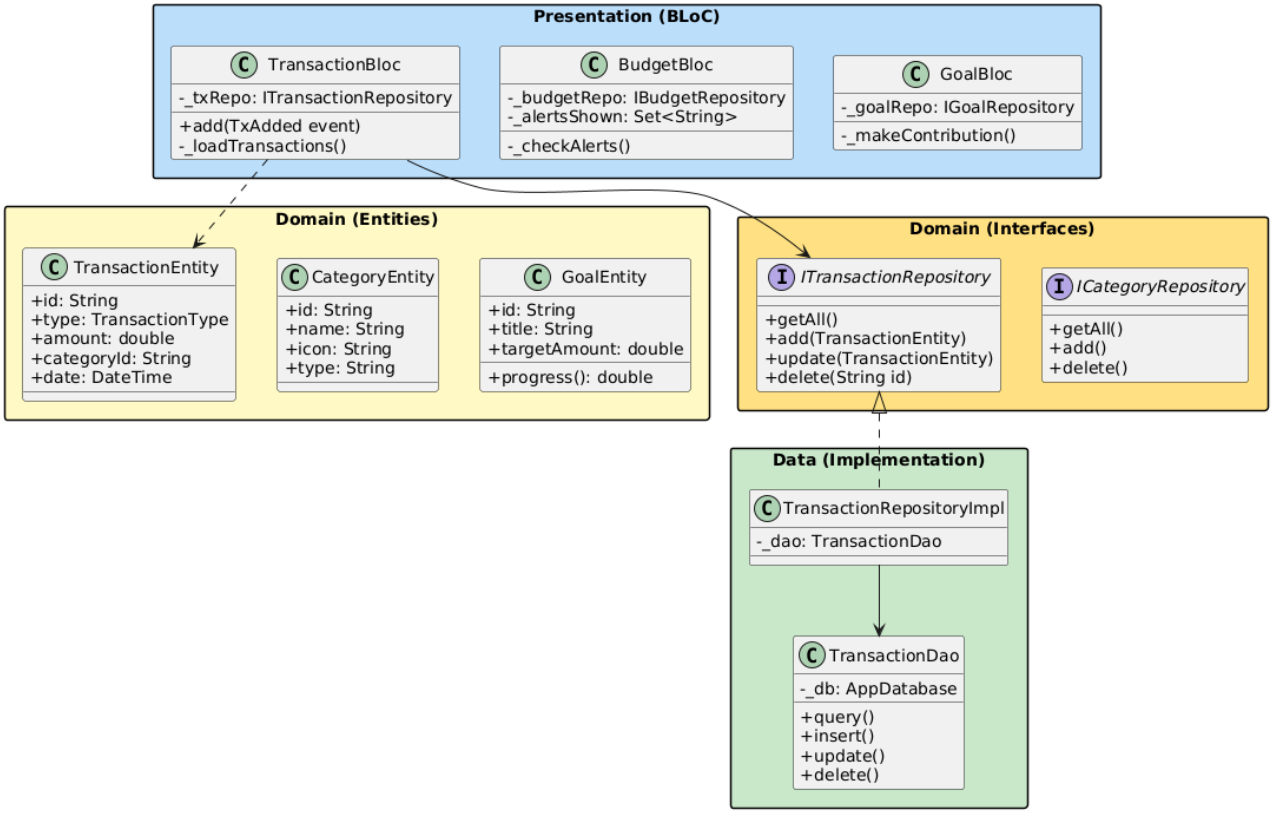

Рис. 3.1 - Діаграма класів основних компонентів застосунку

На рис. 3.1 представлено чотири логічні групи компонентів. Перша група - Presentation, що містить BLoC-компоненти TransactionBloc, BudgetBloc та GoalBloc. Друга група - Domain Entities з сутностями TransactionEntity, CategoryEntity та GoalEntity, які є чистими класами мови Dart без зовнішніх залежностей. Третя група - Domain Interfaces з інтерфейсами репозиторіїв, що визначають контракти доступу до даних. Четверта група - Data з конкретними реалізаціями репозиторіїв та DAO-класами для роботи з базою даних. Загальну організацію коду застосунку за пакетами та залежності між ними наведено на діаграмі пакетів (рис. 3.2).

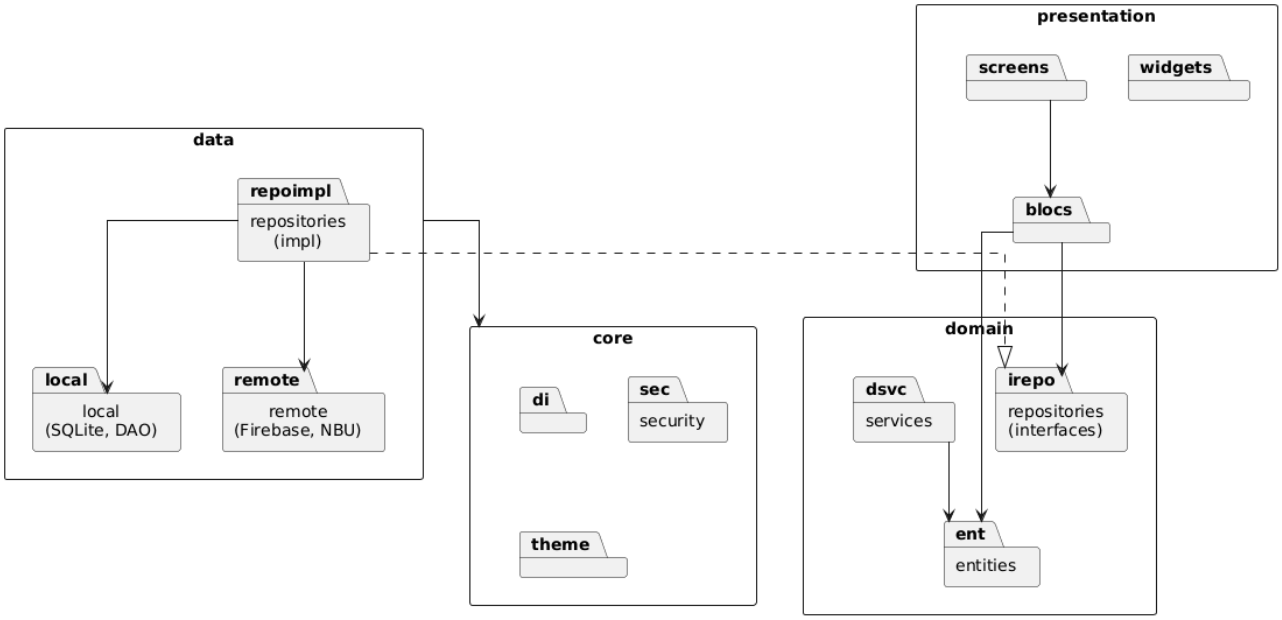

Рис. 3.2 - Діаграма пакетів застосунку

Управління станом застосунку реалізовано через патерн BLoC (Business Logic Component). [12] Патерн BLoC побудований на ідеї реактивного програмування та потоків даних. Кожен BLoC-компонент містить три основні складові: події (Events), стани (States) та обробники подій. Користувацький інтерфейс відправляє події до BLoC у відповідь на дії користувача. BLoC обробляє події, виконує необхідну бізнес-логіку та випускає (emit) оновлений стан. Інтерфейс підписується на потік станів через спеціальний віджет BlocBuilder та автоматично перемальовується при отриманні нового стану.

Для організації залежностей між компонентами застосунку використано бібліотеку get_it, що реалізує патерн Service Locator. [23] Усі необхідні залежності (репозиторії, сервіси, BLoC-компоненти) реєструються у глобальному реєстрі при ініціалізації застосунку. Конкретні класи отримують свої залежності через єдину точку доступу, що спрощує тестування та заміну компонентів.

3.2 Структура клієнтської частини

Клієнтська частина застосунку реалізована з використанням фреймворку Flutter та мови програмування Dart. Структура файлів організована за принципами Clean Architecture з чітким розділенням на функціональні модулі.

Кореневою папкою проєкту є lib/, всередині якої виділено п’ять основних піддиректорій: core/, data/, domain/, presentation/, а також головні файли main.dart та app.dart. Папка core/ містить базові утиліти, що використовуються по всьому застосунку: модуль ін’єкції залежностей (di/), сервіси безпеки (security/), теми оформлення (theme/), допоміжні утиліти (utils/) та налаштування мережевого клієнта (network/).

Папка data/ розділена на дві підпапки: local/ для роботи з локальною базою даних SQLite та remote/ для взаємодії з мережевими сервісами. Підпапка local/daos/ містить класи доступу до даних для кожної таблиці бази даних (TransactionDao, CategoryDao, BudgetDao, GoalDao), а підпапка local/models/ - моделі для серіалізації даних. Папка repositories/ містить реалізації інтерфейсів репозиторіїв, що визначені у шарі domain.

Папка remote/ включає сервіси для роботи з зовнішніми API: AiAssistantService для інтелектуального асистента, ExchangeRateService для отримання курсів валют з НБУ, PdfReportService для генерації PDF-звітів, SyncService для хмарної синхронізації, NotificationService для локальних сповіщень та SuggestionService для інтелектуальних підказок. [4], [21], [20]

Папка domain/ містить чисті класи бізнес-логіки без залежностей від інших шарів. Підпапка entities/ - сутності предметної області (TransactionEntity, CategoryEntity, BudgetEntity, GoalEntity, UserEntity). Підпапка repositories/ - інтерфейси репозиторіїв. Підпапка services/ - бізнес-логіка (BalanceService для розрахунку балансу, AnalyticsService для аналітики, GoalForecastService для прогнозування цілей).

Папка presentation/ містить компоненти інтерфейсу користувача. Підпапка blocs/ включає BLoC-класи для управління станом окремих функціональних модулів. Підпапка screens/ містить екрани застосунку: HomeScreen (головний екран), TransactionsScreen (список транзакцій), AnalyticsScreen (аналітика), BudgetsScreen (бюджети), GoalsScreen (фінансові цілі), ProfileScreen (профіль користувача). Підпапка widgets/ містить перевикористовувані віджети - форму додавання транзакцій, віджет вибору категорії, віджет автодоповнення опису.

Точкою входу застосунку є файл main.dart, який ініціалізує Firebase, налаштовує систему ін'єкції залежностей та запускає кореневий віджет MyApp. Альтернативна точка входу main_local.dart передбачена для локального режиму роботи без використання Firebase - для розробки та тестування на пристроях без доступу до Інтернету.

3.3 Реалізація основних функцій

Основні функції застосунку реалізовані з використанням патерну BLoC та принципів Clean Architecture. Розглянемо детальніше реалізацію ключових функціональних модулів.

Модуль обліку транзакцій є центральним функціональним модулем застосунку. Реалізація модуля включає компоненти всіх трьох шарів архітектури: інтерфейс додавання транзакцій (форма BottomSheet), компонент управління станом TransactionBloc, інтерфейс репозиторію ITransactionRepository та його реалізація TransactionRepositoryImpl, а також клас доступу до бази даних TransactionDao. Повний код класу TransactionBloc наведено у Додатку Б.

Процес додавання транзакції відбувається таким чином: користувач натискає кнопку «+» на головному екрані, що викликає відкриття BottomSheet з формою. Форма містить вибір типу операції (витрата або дохід через

сегментований перемикач), поле введення суми з валідацією на додатне число, віджет вибору категорії з попередньо створеного набору, віджет вибору дати, опційне поле введення опису з функцією автодоповнення. Після натискання кнопки «Зберегти» дані передаються до TransactionBloc у вигляді події TxAdded.

Модуль управління бюджетами реалізовує функцію планування витрат за категоріями. Користувач встановлює максимальну суму, яку він планує витратити у певній категорії за поточний місяць (наприклад, 5000 грн на продукти харчування). Система відстежує поточний рівень виконання бюджету у відсотках та надсилає push-сповіщення при досягненні визначених порогових значень - 80% та 100%.

Реалізація сповіщень про бюджет здійснюється у методі _checkAlerts класу BudgetBloc. При кожному оновленні списку бюджетів метод обчислює відсоток виконання для кожного активного бюджету. Якщо відсоток перетинає поріг 80% або 100%, формується відповідне сповіщення через сервіс NotificationService. Для запобігання повторним сповіщенням про той самий поріг використовується внутрішній набір _alertsShown, який зберігає вже показані попередження в межах поточної сесії.

Локальні сповіщення реалізовані з використанням пакета flutter_local_notifications. [20] Пакет дозволяє планувати сповіщення на конкретний час доби з можливістю повторення (наприклад, щоденне нагадування у 20:00 про необхідність обліку транзакцій). Для коректної роботи планувальника використовується бібліотека timezone з налаштуванням часового поясу Europe/Kyiv.

Модуль фінансових цілей дозволяє користувачу формувати довгострокові плани накопичення коштів. При створенні цілі вказується цільова сума, опційний дедлайн та опційні параметри регулярних автоматичних внесків. Сервіс GoalForecastService аналізує заплановані внески та поточний прогрес виконання цілі, формує прогноз досягнення цілі у вказаний термін.

Регулярні внески реалізовані через фоновий процес, який запускається при відкритті застосунку та перевіряє, чи настав час чергового автоматичного внеску для кожної цілі з налаштованими регулярними платежами. При настанні дати внеску система автоматично переказує визначену суму з балансу користувача до цілі та оновлює прогрес.

Інтелектуальні підказки опису транзакцій реалізовано у класі SuggestionService. Сервіс аналізує історію транзакцій користувача, групує їх за описом, для кожного унікального опису обчислює частоту використання та визначає найімовірнішу категорію. При введенні нової транзакції користувач отримує список найбільш релевантних підказок, відсортованих за алгоритмом, що поєднує частоту використання з пріоритетом нещодавніх записів.

Алгоритм ранжування підказок враховує два чинники: загальну частоту використання конкретного опису (frequency) та свіжість записів (бонусні бали для записів, створених протягом останніх 30 днів). Результат обчислення кешується на 30 секунд для забезпечення швидкого відгуку інтерфейсу при введенні тексту.

Інтелектуальний асистент реалізований у класі AiAssistantService. Сервіс використовує rule-based підхід до розпізнавання намірів користувача (intents) на основі регулярних виразів. Розпізнаються 12 основних намірів: привітання, запит балансу, витрати за день, витрати за період, найбільша категорія витрат, поради з економії, статус цілей, стан бюджету, останні операції, доходи за період, довідка та невпізнаний намір.

Для розпізнавання намірів використано регулярні вирази, орієнтовані на корені слів української мови. Такий підхід дозволяє коректно обробляти морфологічні варіанти слів - наприклад, регулярний вираз «економ» розпізнає форми «економити», «економія», «економно», «економ» та інші похідні. Після визначення наміру викликається відповідна функція, яка генерує текстову відповідь на основі реальних даних користувача.

3.4 Реалізація хмарної синхронізації

Функція хмарної синхронізації дозволяє користувачу створити резервну копію всіх своїх даних у хмарному сховищі та відновити їх на іншому пристрої або після перевстановлення застосунку. Реалізація базується на сервісі Firebase Firestore - документній NoSQL-базі даних від Google [13].

Структура даних у Firestore спроєктована таким чином, щоб забезпечити ізоляцію даних між користувачами. Кожен користувач має окремий документ у колекції users з ідентифікатором, який збігається з його Firebase Authentication UID. Дані користувача (транзакції, категорії, бюджети, цілі) зберігаються у підколекціях відповідних документів. Така структура полегшує налаштування правил безпеки Firestore та забезпечує природну ізоляцію даних.

Сервіс SyncService інкапсулює всю логіку взаємодії з хмарним сховищем. Сервіс надає два основні методи: pushAll() для завантаження всіх локальних даних до хмари та pullAll() для відновлення даних з хмари у локальну базу. Метод pushAll() послідовно завантажує дані всіх таблиць - транзакцій, категорій, бюджетів та цілей.

Завантаження даних виконується пакетно (batch) для оптимізації мережевого трафіку та зменшення кількості окремих запитів до серверу. Firestore накладає обмеження у 500 операцій на один batch, тому при великій кількості записів сервіс автоматично розділяє дані на пакети по 400 записів кожен (із запасом для уникнення граничного значення).

Аналогічно реалізовано метод відновлення даних pullAll(). Сервіс послідовно зчитує всі документи з підколекцій користувача у Firestore, перетворює їх у відповідні сутності та записує до локальної бази даних. Перед завантаженням нових даних виконується очищення локальної бази, щоб уникнути дублювання записів.

Для забезпечення консистентності даних між локальним та хмарним сховищами кожна транзакція має поле is_synced, яке вказує на стан синхронізації. При додаванні нової транзакції локально поле встановлюється у значення 0, після успішного завантаження до Firestore - у значення 1. Це дозволяє оптимізувати наступні операції синхронізації, передаючи лише ті записи, які ще не були завантажені.

У разі помилки під час синхронізації (втрата мережевого з'єднання, помилка серверу, перевищення квоти Firestore) сервіс зберігає поточний стан синхронізації та повідомляє користувача про проблему через елементи інтерфейсу. При наступному запуску синхронізації операція продовжиться з останньої успішної позиції.

3.5 Реалізація системи безпеки даних

Питання безпеки даних має критичне значення для застосунку, що обробляє фінансову інформацію користувача. У розроблюваному застосунку реалізовано багаторівневу систему захисту даних, що включає шість основних рівнів безпеки.

Перший рівень - автентифікація користувачів через сервіс Firebase Authentication. [14] Користувач реєструється у системі за допомогою електронної пошти та пароля. Firebase Authentication самостійно обробляє хешування паролів - паролі ніколи не зберігаються у відкритому вигляді, а лише у вигляді криптографічних хеш-значень. Сервіс також обробляє відновлення пароля через електронну пошту, верифікацію адреси електронної пошти та інші процедури управління обліковими записами.

Другий рівень - обов'язкова верифікація електронної пошти. При реєстрації нового користувача на вказану адресу електронної пошти автоматично надсилається лист із посиланням для підтвердження. Користувач,

який не підтвердив свою адресу, має обмежений доступ до функціональності застосунку. Це запобігає реєстрації облікових записів з неіснуючими або помилковими адресами електронної пошти.

Третій рівень - ізоляція даних між обліковими записами на одному пристрої. У всіх таблицях локальної бази даних, окрім таблиці users, передбачено поле user_id, що містить ідентифікатор користувача-власника запису. Усі запити до бази даних обов'язково містять умову фільтрації за user_id поточного користувача. Реалізація ізоляції здійснюється через спеціальний клас UserContext, який зберігає ідентифікатор активного користувача та надає його репозиторіям при виконанні запитів. Повний код класу UserContext, що забезпечує ізоляцію даних, наведено у Додатку Б.

Четвертий рівень - правила безпеки Firestore (Security Rules) для хмарного сховища. [15] Правила безпеки описують, які операції з даними дозволені різним користувачам залежно від їх ідентифікатора та інших атрибутів. У розроблюваному застосунку правила обмежують доступ користувачів виключно до власних даних - кожен користувач може читати та записувати лише ті документи, які знаходяться у його піддиректорії users/{userId}. Спроби доступу до даних інших користувачів автоматично відхиляються сервером Firestore. Правила безпеки Firestore наведено у Додатку Б.

П'ятий рівень - захист на рівні інтерфейсу користувача. Деструктивні операції (видалення транзакцій, скидання всіх даних, вихід з облікового запису) обов'язково підтверджуються через діалогові вікна. Це запобігає випадковій втраті даних унаслідок помилкових дій користувача. Також інтерфейс не дозволяє введення некоректних даних - поля сум приймають лише числові значення, поля дат використовують спеціалізовані віджети вибору дати.

Шостий рівень - перевірка цілісності локальних даних при запуску застосунку. При відкритті бази даних виконується аналіз консистентності - перевіряється відповідність зовнішніх ключів існуючим записам, наявність

обов’язкових атрибутів, коректність значень полів типу. У разі виявлення некоректних даних формується відповідне повідомлення для користувача з пропозицією відновити дані з хмарної копії.

3.6 Вибір інструментарію розробки

Для розробки мобільного застосунку розглянуто декілька варіантів технологій та інструментарію. Аналіз проводився за критеріями: кросплатформенність, продуктивність, наявність екосистеми пакетів, складність освоєння, активність спільноти розробників.

Серед розглянутих варіантів - нативна розробка для Android з використанням мови Kotlin, кросплатформенний фреймворк React Native (JavaScript/TypeScript), кросплатформенний фреймворк Xamarin (C#) та фреймворк Flutter з мовою Dart. Після порівняння за визначеними критеріями для розробки застосунку обрано фреймворк Flutter. [11, 35]

Фреймворк Flutter має низку переваг для розроблюваного застосунку. По-перше, Flutter використовує власний рендерер на основі бібліотеки Skia, що забезпечує однаковий вигляд інтерфейсу на всіх пристроях та версіях операційних систем. По-друге, Flutter компілюється в нативний машинний код, що забезпечує продуктивність, близьку до нативних застосунків. По-третє, Flutter має значну екосистему пакетів через офіційний репозиторій pub.dev [19], що містить тисячі готових компонентів та бібліотек. По-четверте, мова Dart має сувору типізацію та зрозумілий синтаксис, що зменшує кількість помилок під час розробки. [11]

Підсумкову інформацію про використаний інструментарій наведено у табл. 3.1.

Таблиця 3.1

Перелік використаного інструментарію розробки

Компонент	Технологія
Мова програмування	Dart 3.0+
Фреймворк розробки	Flutter 3.27
Архітектура	Clean Architecture
Управління станом	flutter_bloc 8.1
Локальна база даних	sqflite 2.3 (SQLite)
Хмарне сховище	Firebase Firestore
Автентифікація	Firebase Authentication
Ін’єкція залежностей	get_it 7.7
Курси валют	API Національного банку України
Локальні сповіщення	flutter_local_notifications 17.2
PDF-звіти	pdf 3.11 + printing 5.13
HTTP-клієнт	dio 5.4
Графіки та діаграми	fl_chart 0.68
Дизайн інтерфейсу	Material Design 3
Середовище розробки	Visual Studio Code
Контроль версій	Git, GitHub
Безперервна інтеграція	GitHub Actions

Бібліотека flutter_bloc обрана для управління станом застосунку через її чітку семантику та зручність опису потоків даних. [12] Бібліотека sqflite є фактичним стандартом для роботи з SQLite у Flutter-застосунках та забезпечує асинхронну роботу з базою даних. [22] Бібліотека pdf використовується для генерації PDF-звітів безпосередньо на пристрої користувача - без необхідності відправлення даних на сервер. [21]

Для побудови графіків та діаграм використано бібліотеку fl_chart, яка надає широкий набір типів діаграм (лінійні, стовпчасті, кругові, точкові) з

можливістю кастомізації зовнішнього вигляду. Дизайн інтерфейсу побудовано за принципами Material Design 3 - актуальної версії системи дизайну від Google. [24], [18]

Висновки до розділу 3. У розділі описано процес розробки програмного забезпечення розроблюваного застосунку. Спроєктовано архітектуру за принципами Clean Architecture з розділенням на три шари: presentation, domain та data. Описано структуру клієнтської частини та реалізацію основних функціональних модулів: облік транзакцій, бюджетування з push-сповіщеннями, фінансові цілі з автоматичними внесками, інтелектуальні підказки на основі історії, AI-асистент з підтримкою 12 типів запитів. Реалізовано хмарну синхронізацію даних через Firebase Firestore з пакетною передачею для оптимізації мережевого трафіку. Реалізовано шестирівневу систему захисту даних, що включає автентифікацію, верифікацію електронної пошти, ізоляцію даних, правила безпеки Firestore та перевірку цілісності даних. Обґрунтовано вибір фреймворку Flutter з мовою Dart, бази даних SQLite та хмарної платформи Firebase. Складено перелік використаного інструментарію розробки (табл. 3.1).

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Після завершення розробки виконано комплексне тестування розробленого застосунку для перевірки відповідності його роботи поставленим вимогам. Тестування проводилось ручним методом за принципом сценаріїв використання (test cases). Для кожного функціонального модуля було складено набір тестових сценаріїв, які охоплюють як штатні режими роботи, так і граничні випадки та обробку помилок.

Тестування програмного забезпечення - це процес перевірки відповідності фактичної поведінки програми очікуваній, спрямований на виявлення дефектів та підтвердження якості продукту. Тестування дозволяє переконатися, що застосунок працює коректно у різних умовах, обробляє як штатні сценарії, так і граничні випадки та некоректні дані користувача [26].

Розрізняють декілька рівнів тестування: модульне (перевірка окремих функцій та класів), інтеграційне (перевірка взаємодії компонентів), системне (перевірка роботи системи в цілому) та приймальне (перевірка відповідності вимогам). За ступенем автоматизації тестування поділяють на ручне (виконується людиною за сценаріями) та автоматизоване (виконується спеціальними програмами-тестами).

Для перевірки розробленого застосунку обрано метод ручного функціонального тестування за сценаріями використання (test cases). Цей підхід передбачає складання для кожного функціонального модуля набору тестових сценаріїв, кожен з яких описує початкові умови, послідовність дій та очікуваний результат. Тестувальник послідовно виконує дії, передбачені сценарієм, та порівнює фактичний результат з очікуваним. У разі розбіжності фіксується

дефект, який передається на виправлення. Загальний процес проведення функціонального тестування наведено на рис. 4.1.

Рис. 4.1 - Процес проведення функціонального тестування

Тестування модуля автентифікації включало перевірку таких сценаріїв: реєстрація нового користувача через електронну пошту та пароль, верифікація електронної пошти через посилання у листі, вхід зареєстрованого користувача з правильними обліковими даними, спроба входу з некоректним паролем (очікувана поведінка - повідомлення про помилку), скидання пароля через електронну пошту, вихід з облікового запису та повторний вхід. Усі сценарії пройшли успішно.

Тестування модуля обліку транзакцій передбачало перевірку повного циклу роботи з транзакціями: створення транзакції з усіма обов'язковими полями, спроба збереження транзакції з нульовою сумою (очікуване відхилення), спроба збереження транзакції без вибраної категорії (очікуване попередження), редагування існуючої транзакції зі зміною суми та категорії, видалення транзакції з підтвердженням, фільтрація списку транзакцій за типом операції та за категорією.

Тестування модуля бюджетів включало перевірку механізму сповіщень. Для тестування створювались бюджети з невеликими граничними значеннями (наприклад, 100 грн), після чого послідовно додавались транзакції з кумулятивними сумами, які перетинали пороги 80% та 100%. Сповіщення приходили коректно у визначених точках. Перевірено механізм запобігання дублюванню сповіщень - повторне додавання транзакції після перетину порогу не призводило до повторного сповіщення.

Тестування модуля фінансових цілей перевіряло створення цілей з різними параметрами: ціль без дедлайну, ціль з дедлайном у майбутньому, ціль з налаштованими регулярними внесками. Перевірено механізм автоматичних внесків - після настання дати чергового внеску система коректно списувала визначену суму з балансу та оновлювала прогрес виконання цілі. Прогноз досягнення цілі формувався на основі поточного прогресу та запланованих внесків.

Асистент коректно розпізнавав основні наміри для перевірених формулювань та формував текстові відповіді на основі реальних даних користувача. У ході тестування виявлено, що проблеми з розпізнаванням виникають переважно при використанні складних синонімів та сленгових форм, які не охоплені регулярними виразами поточної версії. Такі випадки коректно обробляються через категорію "unknown" з виведенням пропозиції переформулювати запит.

Тестування хмарної синхронізації включало перевірку резервного копіювання та відновлення даних. Послідовність тестових дій: створення значного обсягу тестових даних (50+ транзакцій, 10 категорій, 5 бюджетів, 3 цілі), виконання операції резервного копіювання до Firestore, видалення локальних даних, виконання операції відновлення з хмари. Усі дані були коректно відновлені без втрат та дублювань.

Тестування кросбраузерної сумісності проводилось на трьох пристроях: фізичному смартфоні Xiaomi Redmi Note 11 з Android 12, емуляторі Android Studio з Android 13 та емуляторі з Android 6.0 (мінімальна підтримувана версія). На всіх пристроях застосунок функціонував коректно без видимих відмінностей у поведінці.

Тестування продуктивності показало високу швидкість роботи застосунку. Миттєвий відгук на дії користувача. Розмір файлу APK становить 62 МБ, що відповідає очікуванням для Flutter-застосунку з інтеграцією Firebase. Споживання оперативної пам’яті при типовому використанні знаходиться в межах 80-120 МБ.

У процесі тестування було виявлено та виправлено п’ять помилок різного рівня критичності:

- некоректне оновлення бюджету після видалення транзакції - виправлено додаванням примусового перерахунку бюджетів після операцій з транзакціями;

- відсутність автоматичного оновлення курсів валют - додано періодичне оновлення кожні 12 годин з опцією ручного оновлення;

- відображення категорій іншого користувача у деяких сценаріях - виправлено через впровадження класу UserContext для централізованого керування ідентифікатором користувача;

- відсутність push-сповіщень на пристроях з Android API 33+ - додано відповідний запит дозволу POST_NOTIFICATIONS;

- відображення raw-імен іконок у повідомленнях інтелектуального асистента - виправлено через конвертацію назв іконок Material Icons у відповідні емодзі.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректної роботи розробленого застосунку висуваються певні вимоги до апаратного та програмного забезпечення мобільного пристрою користувача. Вимоги поділяються на мінімальні (за яких застосунок гарантовано працює) та рекомендовані (для оптимального користувацького досвіду). Загальну топологію системи з розподілом компонентів між вузлами (мобільний пристрій користувача, хмарні сервіси Firebase, сервер API НБУ) наведено на діаграмі розгортання (рис. 4.2).

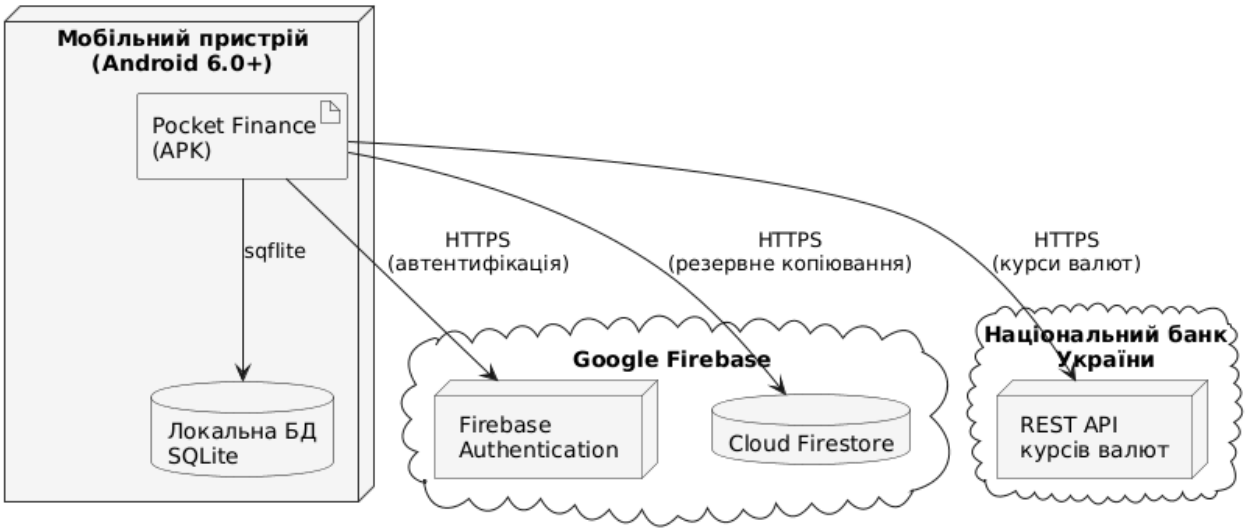

Рис. 4.2 - Діаграма розгортання системи

Мінімальні вимоги до апаратного забезпечення:

- операційна система Android версії 6.0 (API 23) або новіше;

- процесор з частотою не менше 1.4 ГГц (ARMv7 або ARM64);

- оперативна пам’ять не менше 1 ГБ;

- вільний дисковий простір не менше 100 МБ для встановлення застосунку та зберігання даних;

- діагональ дисплея не менше 4 дюймів зі здатністю не нижче 480×800 пікселів.

Рекомендовані вимоги:

- операційна система Android версії 10 або новіше;

- процесор з частотою 2.0 ГГц або вище;

- оперативна пам’ять 3 ГБ або більше;

- вільний дисковий простір 500 МБ для роботи з тривалою історією операцій;

Для повноцінного використання функції хмарного резервного копіювання та інтеграції з API Національного банку України необхідне підключення до мережі Інтернет. Усі інші функції застосунку доступні в офлайн-режимі. Рекомендована швидкість мережевого з'єднання - від 1 Мбіт/с для коректного завантаження курсів валют та виконання операцій синхронізації.

4.3 Розгортання та інсталяція

Розроблений мобільний застосунок розповсюджується у вигляді APK-файлу - стандартного формату пакетів для платформи Android. Розмір файлу APK становить 62 МБ. Файл APK містить скомпільовану версію застосунку, всі необхідні ресурси (іконки, шрифти, зображення) та маніфест з описом дозволів.

Процес інсталяції застосунку на пристрій користувача включає такі кроки:

- перехід на вебсайт itch.io на персональну сторінку застосунку Pocket Finance та завантаження інсталяційного APK-файлу на пристрій; [29], [30]

- увімкнення опції «Встановлення з невідомих джерел» у налаштуваннях безпеки Android (для версій до Android 8);

- запуск завантаженого APK-файлу через файловий менеджер;

- підтвердження запитів на надання дозволів (доступ до Інтернету, відправлення сповіщень, використання камери для аватара);

- очікування завершення процесу інсталяції (зазвичай 10-15 секунд);

- запуск встановленого застосунку через ярлик на головному екрані пристрою.

При першому запуску застосунку користувачу має пройти процедуру реєстрації або входу. Для реєстрації необхідно вказати дійсну адресу електронної пошти та пароль, який відповідає вимогам безпеки (мінімум 8 символів, наявність літер та цифр). Після реєстрації на вказану адресу надсилається лист з посиланням для верифікації облікового запису.

Після завершення процедури автентифікації автоматично створюється набір стандартних категорій транзакцій. Застосунок готовий до використання - користувач може одразу починати облік фінансових операцій. Для відновлення даних з раніше створеної резервної копії передбачена опція у налаштуваннях профілю.

4.4 Інструкція з експлуатації

4.4.1 Початок роботи з застосунком

Після успішного входу до облікового запису користувач потрапляє на головний екран застосунку. Головний екран відображає основну інформацію про фінансовий стан користувача: поточний баланс (різниця між загальними доходами та витратами), список останніх операцій (5-10 останніх записів з зазначенням типу, суми та категорії), поточні курси валют (USD та EUR щодо UAH, отримані з API НБУ), а також кнопку швидкого додавання нової транзакції.

Нижня панель навігації забезпечує доступ до основних розділів застосунку: «Головна» (поточний екран), «Транзакції» (повний список усіх операцій), «Аналітика» (графіки та звіти), «Цілі» (фінансові цілі) та «Профіль» (налаштування користувача). Переключення між розділами здійснюється одним натисканням на відповідну іконку у нижній панелі.

4.4.2 Робота з транзакціями

Для додавання нової транзакції користувач натискає кнопку «+» на головному екрані або у розділі «Транзакції». Відкривається форма у вигляді нижньої панелі BottomSheet, у якій необхідно вказати: тип операції (витрата або дохід через сегментований перемикач), суму операції (числове поле з валідацією), категорію (з випадаючого списку доступних категорій), дату операції (типово встановлюється поточна дата), опційний опис операції. При

введенні опису застосунок автоматично пропонує підказки на основі історії попередніх транзакцій з можливістю автоматичного вибору категорії.

Для редагування існуючої транзакції користувач натискає на запис у списку, що відкриває ту саму форму у режимі редагування з попередньо заповненими значеннями. Для видалення транзакції використовується жест swipe-to-delete з обов'язковим підтвердженням через діалогове вікно.

4.4.3 Робота з бюджетами та цілями

Розділ «Бюджети» дозволяє встановити обмеження витрат за категоріями на поточний місяць. Для створення бюджету користувач обирає категорію зі списку, встановлює граничну суму та підтверджує створення. Кожен активний бюджет відображається у вигляді картки з прогрес-баром, який показує поточний рівень виконання у відсотках. При досягненні 80% бюджету з'являється попередження, при 100% - критичне сповіщення.

Розділ «Цілі» призначений для довгострокового планування накопичень. Створення цілі включає введення назви, цільової суми, опційного дедлайну та емодзі для візуалізації. Опційно можна налаштувати автоматичні регулярні внески - задану суму, що автоматично переказується з балансу до цілі через визначені інтервали часу (щотижня або щомісяця). Користувач також може робити одноразові внески до цілі через спеціальну кнопку у деталях цілі.

4.4.4 Перегляд аналітики

Розділ «Аналітика» надає візуалізацію фінансової активності користувача. У верхній частині екрана відображається підсумок за поточний місяць: сума доходів, сума витрат, чистий баланс. Нижче розташована кругова діаграма витрат за категоріями, на якій кожна категорія представлена сегментом з відповідним кольором. Поряд з діаграмою - список категорій з конкретними сумами та відсотками.

У нижній частині екрана відображається стовпчаста діаграма витрат за днями місяця, яка дозволяє виявити дні з найбільшими витратами. Користувач має можливість змінити період аналітики через відповідний перемикач - «Тиждень», «Місяць», «Рік». Також доступна функція генерації PDF-звіту за поточний період через кнопку у верхній частині екрана.

4.4.5 Налаштування та профіль

Розділ «Профіль» містить налаштування облікового запису та застосунку. Користувач може змінити відображуване ім’я, аватар (зображення завантажується з галереї або робиться через камеру), валюту за замовчуванням (UAH, USD або EUR), тему оформлення (світла або темна). Окремий пункт меню - управління категоріями: створення власних категорій, видалення невикористовуваних стандартних категорій.

У розділі «Профіль» також доступні функції резервного копіювання: «Зберегти у хмару» для створення резервної копії та «Відновити з хмари» для відновлення даних з раніше створеної копії. У підрозділі «Сповіщення» можна налаштувати щоденні нагадування у визначений час, увімкнути або вимкнути сповіщення про бюджети.

Висновки до розділу 4. У розділі представлено рекомендації щодо впровадження та експлуатації розробленої системи. Виконано комплексне тестування застосунку за 20 сценаріями використання, у процесі якого виявлено та виправлено 5 помилок різного рівня критичності. Встановлено, що додаток миттєво відгукується на дії користувача. розмір APK-файлу - 62 МБ. Сформульовано мінімальні та рекомендовані вимоги до апаратного та програмного забезпечення мобільного пристрою. Описано процес інсталяції застосунку на пристрій користувача. Подано детальну інструкцію з експлуатації для основних функцій застосунку.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи розроблено мобільний застосунок «Pocket Finance» для управління особистими фінансами на платформі Android. Застосунок повністю реалізує всі поставлені у вступі завдання та відповідає сформульованим функціональним і нефункціональним вимогам.

У першому розділі проведено детальний аналіз предметної області управління особистими фінансами. Розглянуто основні процеси, які відбуваються при веденні обліку фінансів - реєстрація транзакцій, бюджетування, постановка цілей, аналіз даних. Виконано моделювання предметної області з використанням нотації UML - побудовано діаграму варіантів використання, діаграму послідовності для процесу додавання транзакції та діаграму активності. Проведено порівняльний аналіз трьох популярних мобільних застосунків для управління фінансами (Money Manager, Wallet, Spendee), за результатами якого виявлено їх сильні та слабкі сторони. На основі аналізу сформульовано 18 функціональних та 10 нефункціональних вимог до розроблюваного застосунку.

У другому розділі виконано проєктування інформаційного забезпечення. Розроблено логічну модель даних реляційної структури, яка містить п’ять основних сутностей: USERS, CATEGORIES, TRANSACTIONS, BUDGETS, GOALS. Описано зв’язки між сутностями та реалізацію ізоляції даних між обліковими записами через поле user_id. Обґрунтовано вибір реляційної бази даних SQLite для локального зберігання та Firebase Firestore для хмарного резервного копіювання. Розроблено фізичну структуру бази даних із вісьмома індексами для оптимізації найбільш частих запитів.

У третьому розділі описано розробку програмного забезпечення. Спроєктовано архітектуру за принципами Clean Architecture з розділенням на три

шари: presentation, domain і data. Обґрунтовано вибір технологій - фреймворк Flutter з мовою Dart, патерн BLoC для управління станом, бібліотека get_it для ін'єкції залежностей. Реалізовано ключові функціональні модулі: облік транзакцій, бюджети з push-сповіщеннями, фінансові цілі з автоматичними внесками, інтелектуальні підказки, AI-асистент з 12 розпізнаваними намірами, хмарна синхронізація через пакетну передачу до Firestore. Реалізовано шестирівневу систему захисту даних.

У четвертому розділі представлено рекомендації щодо впровадження та експлуатації системи. Виконано тестування за 20 сценаріями використання, виявлено та виправлено п'ять помилок. Встановлено, що додаток миттєво відгукується на дії користувача, розмір APK-файлу 62 МБ. Сформульовано вимоги до апаратного та програмного забезпечення мобільного пристрою. Описано процес інсталяції та надано детальну інструкцію з експлуатації.

Усього розроблений застосунок містить більше 70 файлів коду на мові Dart, реалізує 19 use cases, забезпечує роботу на пристроях з Android 6.0 та новіше. Розмір APK-файлу складає 62 МБ, що відповідає вимогам.

Серед перспектив подальшого розвитку розробленої системи можна виділити: підтримку платформи iOS (фреймворк Flutter це дозволяє без значних змін у кодовій базі), голосовий ввід транзакцій з розпізнаванням мовлення, розпізнавання QR-кодів касових чеків через інтеграцію з API е-Чек НБУ, експорт даних у формат Excel, інтеграцію з банківськими API через open banking, додавання шаблонів регулярних транзакцій, генерацію річних аналітичних звітів, інтеграцію з GPT-моделями для розширення функціональності інтелектуального асистента.

Практичне значення роботи полягає у створенні готового до використання програмного продукту, який може бути впроваджений у Google Play та використаний широким колом українських користувачів. Розроблений застосунок є безкоштовним, україномовним, повноцінно функціонує без

підключення до мережі Інтернет та підтримує інтеграцію з офіційними курсами Національного банку України - все це робить його унікальним рішенням на українському ринку.

Поставлена мета бакалаврської кваліфікаційної роботи досягнута, всі визначені у вступі завдання виконано у повному обсязі. Розроблений мобільний застосунок «Pocket Finance» є якісним програмним продуктом, який допоможе користувачам контролювати свої фінанси та досягати поставлених фінансових цілей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закерничний І. О. Особливості мобільного застосунку для ведення особистих фінансів / І. О. Закерничний // Молодь в науці: дослідження, проблеми, перспективи (МН-2025) : матеріали XIV Всеукраїнської науково-практичної інтернет-конференції молодих учених, аспірантів та студентів. - Вінниця : ВНТУ, 2025. - URL : https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25194 (дата звернення : 03.03.2026).

2. Дробаха М. Б. Мобільний модуль додатку «Автоматизоване робоче місце енергоменеджера» : дипломна робота на здобуття ступеня бакалавра : 121 «Інженерія програмного забезпечення» / Маргарита Борисівна Дробаха ; наук. керівник А. М. Ковальчук. - Київ : КПІ ім. Ігоря Сікорського, 2021. - 59 с. - URL : https://ela.kpi.ua/server/api/core/bitstreams/f16b2a91-223b-4ea4-9568-e6dd09911b90/content (дата звернення : 05.03.2026).

3. Фінансова грамотність у вашій кишені : огляд мобільних додатків для управління особистими фінансами [Електронний ресурс] // Banker.ua : фінансовий портал. - 09.08.2024. - URL : https://banker.ua/uk/projects/oglyad-dodatkiv-dlya-upravlinnya-osobistimi-finansami/ (дата звернення : 07.03.2026).

4. Сервіс отримання офіційного курсу гривні до іноземних валют : офіційний API [Електронний ресурс] // Національний банк України. - URL : https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json (дата звернення : 10.03.2026).

5. Selvaraj S. Advanced Flutter : Build High-Performance, Cross-Platform Apps for Mobile, Web and Desktop / Sivaraj Selvaraj. - Berkeley : Apress, 2026. - 1018 p. - ISBN 979-8-8688-2104-2.

6. Гомілко О. О. Розробка мобільних застосунків за допомогою технологій Flutter та Dart : курсова робота : 122 «Комп’ютерні науки та інформаційні

технології» / О. О. Гомілко ; наук. керівник О. П. Жежерун. - Київ : Національний університет «Києво-Могилянська академія», 2020. - 39 с. - URL : https://ekmair.ukma.edu.ua/server/api/core/bitstreams/826f66d9-5bf0-48c2-a443-e596584b690e/content (дата звернення : 16.03.2026).

7. Аналіз швидкодійності відтворення інтерфейсу мобільних додатків на кросплатформених платформах Flutter та React Native : магістерська кваліфікаційна робота [Електронний ресурс]. - Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2023. - URL : https://crust.ust.edu.ua/items/c2bde355-0767-4eaf-be73-26c75b38f2cb (дата звернення : 18.03.2026).
8. SQLite Documentation [Electronic resource] : Official SQLite documentation. - URL : https://www.sqlite.org/docs.html (last access : 19.03.2026).
9. Tune N. Architecture Modernization : Socio-technical alignment of software, strategy, and structure / Nick Tune, Jean-Georges Perrin. - Shelter Island : Manning Publications, 2024. - 408 p. - ISBN 978-1-63343-815-6.
10. Flutter Documentation [Electronic resource] : Official Flutter framework documentation. - URL : https://docs.flutter.dev/ (last access : 21.03.2026).
11. Dart Programming Language [Electronic resource] : Official Dart language documentation. - URL : https://dart.dev/language (last access : 23.03.2026).
12. Bloc Library Documentation [Electronic resource] : Official documentation of the Bloc state management library for Dart and Flutter. - URL : https://bloclibrary.dev/ (last access : 25.03.2026).
13. Firebase Documentation [Electronic resource] : Official documentation of Firebase platform. - URL : https://firebase.google.com/docs (last access : 26.03.2026).
14. Firebase Authentication [Electronic resource] : Documentation of authentication services in Firebase. - URL : https://firebase.google.com/docs/auth (last access : 28.03.2026).

15. Cloud Firestore Security Rules [Electronic resource] : Documentation of access rules for Firestore database. - URL : https://firebase.google.com/docs/firestore/security/get-started (last access : 30.03.2026).

16. Alessandria S. Flutter Cookbook : 100+ step-by-step recipes for building cross-platform, professional-grade apps with Flutter 3.10.x and Dart 3.x / Simone Alessandria, Brian Kayfitz. - 2nd edition. - Birmingham : Packt Publishing, 2023. - 712 p. - ISBN 978-1-80324-543-0.

17. Bailey T. Flutter for Beginners : Cross-platform mobile development from Hello, World! to app release with Flutter 3.10+ and Dart 3.x / Thomas Bailey, Alessandro Biessek. - 3rd edition. - Birmingham : Packt Publishing, 2023. - 406 p. - ISBN 978-1-83763-038-7.

18. Material Design 3 [Electronic resource] : Google’s design system guidelines. - URL : https://m3.material.io/ (last access : 01.04.2026).

19. pub.dev - The official package repository for Dart and Flutter [Electronic resource]. - URL : https://pub.dev/ (last access : 03.04.2026).

20. flutter_local_notifications package [Electronic resource] : Documentation. - URL : https://pub.dev/packages/flutter_local_notifications (last access : 05.04.2026).

21. pdf package for Dart and Flutter [Electronic resource] : Documentation. - URL : https://pub.dev/packages/pdf (last access : 07.04.2026).

22. sqflite package for Dart and Flutter [Electronic resource] : SQLite plugin for Flutter. - URL : https://pub.dev/packages/sqflite (last access : 09.04.2026).

23. get_it package [Electronic resource] : Simple service locator for Dart and Flutter projects. - URL : https://pub.dev/packages/get_it (last access : 11.04.2026).

24. fl_chart package [Electronic resource] : Powerful Flutter chart library. - URL : https://pub.dev/packages/fl_chart (last access : 13.04.2026).

25. Moore K. Mastering Flutter : Learn to develop Flutter apps for iOS, Android, desktop and web / Kevin Moore. - BPB Publications, 2025. - 544 p. - ISBN 978-9-36589-917-7.

26. Cohen S. Software Architecture and Decision-Making : Leveraging Leadership, Technology, and Product Management to Build Great Products / Srinath Cohen. - Boston : Addison-Wesley Professional, 2023. - 432 p. - ISBN 978-0-13-824973-1.

27. Єфремов Є. Створення додатку на Flutter : перші кроки [Електронний ресурс] / Євген Єфремов // DOU : сайт. - 2019. - URL : https://dou.ua/lenta/articles/flutter-first-steps/ (дата звернення : 19.04.2026).

28. Про Flutter, коротко : Основи [Електронний ресурс] // DevZone : сайт. - 2025. - URL : https://devzone.org.ua/post/pro-flutter-korotko-osnovy (дата звернення : 24.04.2026).

29. Itch.io : вебплатформа для незалежних розробників та дистрибуції цифрового контенту [Електронний ресурс]. - URL: https://itch.io/ (дата звернення: 15.05.2026)

30. Pocket Finance : мобільний застосунок для управління особистими фінансами [Електронний ресурс] / розробник М. Шевчук. - Платформа Itch.io. - URL: https://max-shevchuk.itch.io/pocket-finance (дата звернення: 15.05.2026)

Додаток А

Лістинг основних SQL-команд для створення бази даних

SQL-команди створення таблиць локальної бази даних SQLite виконуються одноразово при першому запуску застосунку. Нижче наведено повний лістинг команд створення всіх таблиць та індексів.

Лістинг А.1 - Створення таблиць та індексів

```
CREATE TABLE users (
  id TEXT PRIMARY KEY NOT NULL,
  email TEXT NOT NULL UNIQUE,
  password_hash TEXT NOT NULL,
  currency TEXT NOT NULL DEFAULT 'UAH',
  created_at INTEGER NOT NULL
);

CREATE TABLE categories (
  id TEXT PRIMARY KEY NOT NULL,
  user_id TEXT NOT NULL,
  name TEXT NOT NULL,
  icon TEXT NOT NULL,
  color TEXT NOT NULL,
  is_custom INTEGER NOT NULL DEFAULT 0,
  type TEXT NOT NULL CHECK (type IN ('expense', 'income', 'savings'))
);

CREATE TABLE transactions (
  id TEXT PRIMARY KEY NOT NULL,
  user_id TEXT NOT NULL,
  type TEXT NOT NULL CHECK (type IN ('expense', 'income', 'savings')),
  amount REAL NOT NULL CHECK (amount > 0),
  category_id TEXT NOT NULL,
  date INTEGER NOT NULL,
  description TEXT DEFAULT '',
  created_at INTEGER NOT NULL,
  is_synced INTEGER NOT NULL DEFAULT 0,
  FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE RESTRICT
);

CREATE TABLE budgets (
  id TEXT PRIMARY KEY NOT NULL,
  user_id TEXT NOT NULL,
```

```
  category_id TEXT NOT NULL,
  limit_amount REAL NOT NULL CHECK (limit_amount > 0),
  spent_amount REAL NOT NULL DEFAULT 0,
  month TEXT NOT NULL,
  FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE CASCADE,
  UNIQUE (user_id, category_id, month)
);

CREATE TABLE goals (
  id TEXT PRIMARY KEY NOT NULL,
  user_id TEXT NOT NULL,
  title TEXT NOT NULL,
  target_amount REAL NOT NULL CHECK (target_amount > 0),
  saved_amount REAL NOT NULL DEFAULT 0,
  deadline INTEGER,
  emoji TEXT NOT NULL DEFAULT '*',
  created_at INTEGER NOT NULL,
  category_id TEXT,
  recurring_amount REAL,
  recurring_period TEXT,
  recurring_interval INTEGER
);

-- Індекси для оптимізації
CREATE INDEX idx_tx_user ON transactions(user_id);
CREATE INDEX idx_tx_date ON transactions(date);
CREATE INDEX idx_tx_category ON transactions(category_id);
CREATE INDEX idx_cat_user ON categories(user_id);
CREATE INDEX idx_budget_user ON budgets(user_id);
CREATE INDEX idx_goals_user ON goals(user_id);
```

Додаток Б

Лістинг ключових компонентів застосунку

У додатку наведено лістинг ключових компонентів застосунку: BLoC-компонент управління транзакціями, клас ізоляції користувацького контексту, а також правила безпеки Firestore для контролю доступу до хмарних даних.

Лістинг Б.1 - Клас TransactionBloc

```
class TransactionBloc extends Bloc<TransactionEvent, TransactionState> {
  final ITransactionRepository _txRepo;
  final BalanceService _balanceService;

  TransactionBloc(this._txRepo, this._balanceService)
      : super(const TransactionState()) {
    on<TxLoadRequested>(_onLoad);
    on<TxAdded>(_onAdded);
    on<TxUpdated>(_onUpdated);
    on<TxDeleted>(_onDeleted);
  }

  Future<void> _onLoad(TxLoadRequested e, Emitter emit) async {
    emit(state.copyWith(isLoading: true));
    final txs = await _txRepo.getAll();
    final balance = _balanceService.calculate(txs);
    emit(state.copyWith(
      transactions: txs,
      balance: balance,
      isLoading: false,
    ));
  }

  Future<void> _onAdded(TxAdded e, Emitter emit) async {
    final newTx = TransactionEntity(
      id: const Uuid().v4(),
      type: e.type,
      amount: e.amount,
      categoryId: e.categoryId,
      date: e.date,
      description: e.description,
      createdAt: DateTime.now(),
```

```
      );
      await _txRepo.add(newTx);
      add(const TxLoadRequested());
    }
  }
```

Лістинг Б.2 - Клас UserContext (ізоляція даних)

```
class UserContext {
  static String? _currentUserId;

  static String? get currentUserId => _currentUserId;

  static void setUser(String userId) {
    _currentUserId = userId;
  }

  static void clear() {
    _currentUserId = null;
  }
}

// Приклад використання у репозиторії
Future<List<TransactionEntity>> getAll() async {
  final userId = UserContext.currentUserId;
  if (userId == null) return [];
  final rows = await _dao.getAllForUser(userId);
  return rows.map(TransactionModel.fromMap)
             .map((m) => m.toEntity()).toList();
}
```

Лістинг Б.3 - Правила безпеки Firestore

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null
                         && request.auth.uid == userId;

      match /transactions/{txId} {
        allow read, write: if request.auth != null
                           && request.auth.uid == userId;
      }
      match /categories/{catId} {
        allow read, write: if request.auth != null
                           && request.auth.uid == userId;
      }
```

```
      match /budgets/{bId} {
        allow read, write: if request.auth != null
                           && request.auth.uid == userId;
      }
      match /goals/{gId} {
        allow read, write: if request.auth != null
                           && request.auth.uid == userId;
      }
    }
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

Додаток В

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Шевчук Максим Павлович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему Програмне забезпечення мобільного застосунку для управління особистими фінансами» є результатом моєї особистої праці.

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

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

o [ ] інструменти генеративного ШІ не використовувалися.

o [✓] інструменти ШІ: Claude, був використані для:

- [ ] перекладу іншомовних джерел;
- [✓] стилістичного редагування та перевірки граматики;
- [✓] допомоги у написанні/відлагодженні програмного коду;
- [ ] інше: _____.

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

«___» _____ 2026 р. _____ **Максим ШЕВЧУК**