

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

_____ **Ігор БОЛБОТ**
(підпис)

_____ **Белла ГОЛУБ**
(підпис)

«__» _____ 2026 р.

«__» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Система обліку витрат фінансів для приватного
користування»**

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор

_____ **Роман РУДЕНСЬКИЙ**
(підпис)

**Керівник бакалаврської
кваліфікаційної роботи**

_____ **Гордій Я.В.**
(підпис)

Консультант
к.т.н., доцент

_____ **Белла ГОЛУБ**
(підпис)

Виконав

_____ **Богдана ДАНИЛЮК**
(підпис)

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____

(підпис)

Белла ГОЛУБ

«___» _____ 2026 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Данилюк Богдані Сергіївні

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи:

«Система обліку витрат фінансів для приватного користування»
затверджена наказом від «_6_» __січня__ 2026 р. № _46_ «С»

Термін подання завершеної роботи на кафедру «25» __травня__ 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Розробити мобільний банківський застосунок для фізичних осіб: перегляд балансу, переказ коштів, оплата послуг, перегляд транзакцій.

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної галузі та ринку мобільного банкінгу. 2. Огляд та аналіз існуючих рішень. 3. Проєктування БД та архітектури системи. 4. Реалізація серверної та клієнтської частин. 5. Тестування застосунку.

Перелік графічних документів (за необхідності).

ER-діаграма БД; діаграма варіантів використання; архітектурна схема; макети інтерфейсу.

Дата видачі завдання «_6_» __січня__ 2026 р.

**Керівник бакалаврської
кваліфікаційної роботи**

(підпис)

Гордій Я.В.

Консультант

(підпис)

Белла ГОЛУБ

**Завдання взяв до
Виконання**

(підпис)

Богдана ДАНИЛЮК

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП.....	6
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	8
1.1 Опис процесів мобільного банківського обслуговування	8
1.2 Аналіз ринку мобільного банкінгу в Україні	9
1.3 Огляд існуючих рішень.....	10
1.4 Порівняльний аналіз аналогів	11
1.5 Варіанти використання системи	12
1.6 Вимоги до системи	15
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	19
2.1 Проєктування бази даних	19
2.2 Вибір системи управління базою даних.....	22
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	26
3.1 Архітектура системи	26
3.2 Вибір та обґрунтування технологій	27
3.3 Структура frontend.....	30
3.4 Проєктування інтерфейсу користувача.....	32
3.5 Проєктування API.....	36
3.6 Структура проєкту.....	38
3.7 Реалізація серверної частини.....	40
3.8 Реалізація клієнтської частини.....	43
4 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ	49
4.1 Стратегія тестування.....	49
4.2 Функціональне тестування	49
4.3 Тестування на мобільному пристрої (Android).....	53
4.4 Апаратні та програмні вимоги до системи	53
4.5 Тестування продуктивності.....	55

4.6 Тестування безпеки	57
4.7 Аналіз результатів	58
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	63
ДОДАТКИ	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Application Programming Interface — інтерфейс програмування застосунків;

APK — Android Package Kit — формат пакету Android-застосунку;

БД — база даних;

CI/CD — Continuous Integration / Continuous Delivery — безперервна інтеграція та доставка;

ДСТУ — Державний стандарт України;

ER — Entity-Relationship — сутність-зв'язок (модель даних);

HTTP — HyperText Transfer Protocol — протокол передачі гіпертексту;

HTTPS — HyperText Transfer Protocol Secure — захищений протокол передачі гіпертексту;

JSON — JavaScript Object Notation — формат обміну даними;

НБУ — Національний банк України;

NFC — Near Field Communication — комунікація ближнього поля;

ORM — Object-Relational Mapping — об'єктно-реляційне відображення;

OWASP — Open Web Application Security Project — проєкт безпеки вебзастосунків;

REST — Representational State Transfer — архітектурний стиль;

SPA — Single Page Application — односторінковий застосунок;

SQL — Structured Query Language — мова структурованих запитів;

ТЗ — технічне завдання;

UI/UX — User Interface / User Experience — інтерфейс та досвід користувача;

UML — Unified Modeling Language — уніфікована мова моделювання;

WCAG — Web Content Accessibility Guidelines — настанови з доступності вебконтенту;

ВСТУП

Актуальність теми. Стрімкий розвиток мобільних технологій та зміна поведінки споживачів зумовили необхідність переосмислення підходів до надання фінансових послуг. Традиційна банківська модель, орієнтована на фізичну присутність клієнта у відділенні, все менше відповідає потребам сучасного суспільства — особливо в умовах військового стану в Україні, коли мільйони громадян змінили місце проживання або перебувають за кордоном.

Метою дипломної роботи є розробка мобільного банківського додатку, що забезпечує повний цикл фінансового обслуговування без необхідності відвідування банківського відділення. Додаток дозволяє здійснювати перекази, керувати рахунками, аналізувати витрати та отримувати фінансові послуги в режимі реального часу через смартфон.

Актуальність теми підтверджується успішним досвідом вітчизняного необанку Monobank, який станом на 2024 рік обслуговує понад 8 мільйонів клієнтів без жодного фізичного відділення. Це свідчить про високий попит на зручні, доступні та технологічні фінансові рішення. У 2024 році частка безготівкових розрахунків в Україні перевищила 75% (НБУ). Мобільний банкінг із допоміжного каналу перетворився на основний. З початком повномасштабного вторгнення цей зсув прискорився: відділення банків закривались, застосунок лишався.

Задачі дослідження:

1. Проаналізувати ринок мобільного банкінгу України та порівняти monobank, ПриватБанк24 і ПУМБ Online за функціональними критеріями.
2. Сформулювати вимоги до системи — функціональні та нефункціональні.
3. Обґрунтувати вибір стеку: React 18, Node.js, PostgreSQL, Capacitor.
4. Спроектувати архітектуру системи, реляційну модель БД і навігаційну структуру; розробити UML-діаграми.

5. Розробити і налагодити застосунок FlowBank.
6. Розгорнути на Render + Supabase та протестувати функціональність, продуктивність і безпеку.

Практичне значення результатів. Застосунок FlowBank розгорнуто на Render і доступний за публічною адресою. APK встановлено на Samsung Galaxy A50 (Android 11) та протестовано. Реалізовано: управління картками, перекази, контакти, накопичувальні рахунки, аналітика витрат. Кодова база відкрита, може слугувати основою для розширення або як навчальний приклад банківської архітектури.

Бакалаврська кваліфікаційна робота: 76 с., 24 рис., 9 табл., 25 джерел, 5 додатків.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис процесів мобільного банківського обслуговування

До появи мобільних застосунків більшість банківських операцій вимагала фізичного візиту у відділення. FlowBank автоматизує шість ключових процесів, які раніше так чи інакше потребували участі банківського персоналу:

- **Управління картками.** Користувач бачить усі свої картки з актуальним балансом, реквізитами та статусом. Заблокувати або розблокувати картку — дія в один дотик, без дзвінка на гарячу лінію.
- **Переказ коштів.** Вибір картки-відправника, номер отримувача, сума — система перевіряє баланс і виконує операцію атомарно. Результат відображається одразу.
- **Історія транзакцій.** Кожна операція фіксується автоматично: дата, сума, категорія, опис. Список хронологічний, з пошуком і фільтрацією.
- **Аналітика витрат.** Транзакції групуються за категоріями та обраним діапазоном — 7, 30 або 90 днів. Структура витрат відображається у вигляді графіка та кругової діаграми.
- **Накопичувальні рахунки.** Користувач задає назву цілі і цільову суму, поповнює рахунок і відстежує прогрес у відсотках. Система не дає поповнити понад ціль.
- **Контакти.** При переказі номер картки підставляється автоматично при виборі збереженого контакту.

Перевірити баланс, перекинути кошти, заблокувати картку — усе це тепер займає кілька секунд з телефону.

1.2 Аналіз ринку мобільного банкінгу в Україні

Роздрібні розрахунки в Україні у 2024 році без готівки склали 75% (НБУ). У 2015-му той самий показник не перевищував 35%. Карток в обороті понад 80 мільйонів — більшість власників щодня відкривають застосунок, а не гаманець.

COVID-19 у 2020–2021 роках не стільки створив попит на мобільний банкінг, скільки усунув останній спротив йому: кількість мобільних транзакцій за 2020 рік зросла на 47% (ЄМА). Хто роками відкладав перехід на цифрові інструменти — перейшов вимушено. Повномасштабне вторгнення 2022 року поставило питання інакше: для переміщених осіб мобільний застосунок став єдиною точкою доступу до власних коштів.

Три представники — ПриватБанк, monobank і ПУМБ — контролюють понад 70% ринку активних користувачів. monobank з 2017 року працює без жодного відділення. До 2024-го — 7 мільйонів клієнтів. Ця цифра перевищує клієнтську базу багатьох банків, що будували мережу відділень десятиліттями.

Паралельно ринок змінюється структурно. NFC-оплата через Apple Pay і Google Pay стала звичною; QR-оплата розповсюдилась у малому ритейлі; СЕП НБУ забезпечив миттєві перекази між банками. Через «Дію» рахунок відкривається без відвідування відділення. За оцінкою Deloitte Digital Banking Maturity 2024, Україна входить до лідерів цифрової зрілості серед країн ЦСЄ — і це попри воєнний стан.

Запит користувачів змістився. Баланс і перекази — вже не функції, а мінімальний поріг. Категоризація витрат, аналітика за довільний діапазон дат, накопичувальні цілі — саме це сьогодні визначає вибір між застосунками. Водночас в Україні розпочалась імплементація Директиви ЄС PSD2: відкриті банківські API поступово відкривають інфраструктуру для незалежних фінансових сервісів.

1.3 Огляд існуючих рішень

Застосунок Monobank

Запущений 2017 року, monobank не мав жодного відділення від першого дня — і не планував. Засновники Олег Гороховський і Михайло Рогальський обрали базою Універсал Банк, але весь клієнтський досвід перенесли в застосунок. Markswebb неодноразово ставив його на перше місце серед українських банківських сервісів.

iOS (Swift) і Android (Kotlin): нативні застосунки без WebView. Мікросервісна архітектура, власне процесингове ядро, пряма інтеграція з платіжною інфраструктурою НБУ. Картка відкривається без документів і відвідувань — за хвилини.

Функціонал: категоризація витрат у реальному часі; «Банки» — накопичувальні рахунки з відсотками; split bill; кредитний ліміт з гнучким погашенням; Apple Pay і Google Pay з першого дня; мультивалютні рахунки з конвертацією за ринковим курсом. Котики за фінансові досягнення — виглядає несерйозно, але утримує користувачів краще, ніж більшість програм лояльності. До 2024 року — 7 мільйонів клієнтів. Жодного відділення.

ПриватБанк24

20 мільйонів активних клієнтів. Найбільша мережа терміналів самообслуговування в Україні. Найширший функціонал — наслідок двох десятиліть послідовного нарощування можливостей.

Архітектура гібридна: нативні компоненти плюс WebView для частини функцій. Це прискорює розгортання оновлень, але робить UX нерівним. Головна проблема — перевантаженість інтерфейсу: функцій багато, знайти потрібну важко. Поріг входу для нового користувача помітний.

Що важко повторити: BankID з державними реєстрами, ОВДП і фонди прямо в застосунку, програма «Бонус+» з кешбеком. Управління картками

будь-яких банків, оплата комунальних послуг, купівля валюти — все в одному місці.

ПУМБ Online

Орієнтація — преміум-сегмент і малий бізнес. Нативний iOS і Android, матеріальний дизайн, чиста структура навігації. За швидкістю реакції та інноваційністю поступається monobank.

Специфіка: SWIFT-перекази, валютні операції, депозити з підвищеними ставками, страхові продукти. Модуль планування бюджету та відстеження фінансових цілей реалізований повноцінно — не як декорація. Біометрія, push-сповіщення, гнучкі ліміти операцій.

Порівняльний аналіз трьох застосунків показав: базовий функціонал переказів і перегляду балансу реалізовано скрізь. Розрив виникає в аналітиці витрат, зручності навігації та швидкості ключових сценаріїв. Саме ці параметри визначали проєктні рішення FlowBank.

1.4 Порівняльний аналіз аналогів

На основі проведеного огляду здійснено порівняльний аналіз існуючих мобільних банківських рішень за ключовими критеріями, що визначають якість користувацького досвіду та функціональну повноту продукту. Результати аналізу наведено у таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз мобільних банківських застосунків

Критерій	monobank	ПриватБанк24	ПУМБ Online	FlowBank
Платформа	iOS, Android	iOS, Android	iOS, Android	Android (PWA)
Технологія	Native (Swift/Kotlin)	Hybrid	Native	React + Capacitor
Відкриття рахунку	Онлайн, без відділення	Онлайн/відділення	Відділення/онлайн	Реєстрація в застосунку

Критерій	monobank	ПриватБанк24	ПУМБ Online	FlowBank
Управління картками	✓	✓	✓	✓
Перекази коштів	✓	✓	✓	✓
Аналітика витрат	✓ (категорії)	✓ (базова)	✓ (розширена)	✓ (категорії)
Накопичення	✓ (банки)	✓ (депозити)	✓ (депозити)	✓ (savings)
Критерій	monobank	ПриватБанк24	ПУМБ Online	FlowBank
Кредитування	✓	✓	✓	✓
Комунальні платежі	✓	✓	✓	✓ (платежі)
Біометрія	✓	✓	✓	—
Валютні операції	✓	✓	✓	✓ (мультивалюта)
Рейтинг (App Store)	4.9	4.7	4.6	навч. проєкт

Порівняння трьох застосунків виявило очевидну закономірність: управління картками, перекази, аналітика та накопичення присутні в усіх трьох продуктах — це фактичний мінімальний стандарт галузі. Кредитування, страхування та інвестиції зустрічаються лише у частині рішень і виходять за межі базових потреб клієнта. Для FlowBank орієнтиром слугував саме цей базовий набір — функціонал, без якого банківський застосунок не може вважатись повноцінним.

Розроблюваний застосунок FlowBank охоплює весь базовий функціональний набір та принципову відмінність — відкриту архітектуру на сучасному технологічному стеку (React, Node.js, PostgreSQL). Код доступний для дослідження, модифікації та масштабування без обмежень.

1.5 Варіанти використання системи

У моделі Use Case Diagram FlowBank визначено двох акторів: Користувача — фізичну особу, власника банківського рахунку, та Систему — автоматизовані процеси без участі людини: генерацію номерів транзакцій, нарахування відсотків і тригери бази даних.

Основні варіанти використання для актора Користувач:

- UC-01: Перегляд балансу та реквізитів карток;
- UC-02: Переказ коштів між картками;
- UC-03: Переказ коштів на картку контакту;
- UC-04: Перегляд історії транзакцій;
- UC-05: Фільтрація транзакцій за категорією та датою;
- UC-06: Створення накопичувального рахунку;
- UC-07: Поповнення накопичувального рахунку;
- UC-08: Перегляд аналітики витрат за категоріями;
- UC-09: Управління контактами для переказів;
- UC-10: Редагування профілю користувача;
- UC-11: Блокування/розблокування картки;

UC-12, UC-13, UC-14 — три процеси без участі користувача. Номер транзакції генерується в момент її створення — унікальний, без повторень. Паспортні дані проходять валідацію формату при реєстрації: система відхиляє некоректний запис ще до запису в базу. Ліміт накопичувального рахунку перевіряється тригером — спроба перевищити встановлену суму блокується автоматично.

Ці три сценарії не вимагають дій від користувача і не відображаються в інтерфейсі безпосередньо. Вони працюють на рівні бази даних і серверної логіки — непомітно, але критично для цілісності даних.

Реальне відкриття банківського рахунку, кредитний скоринг і розрахунки через платіжну інфраструктуру НБУ — поза межами системи. FlowBank не є банком у юридичному сенсі. Це інтерфейс до вже існуючого рахунку: застосунок керує даними, але не емітує картки і не бере участі в міжбанківських розрахунках.

Діаграма прецедентів системи FlowBank наведена на рис. 1.1. Вона відображає повний перелік варіантів використання для обох акторів — Користувача та Банківської системи.

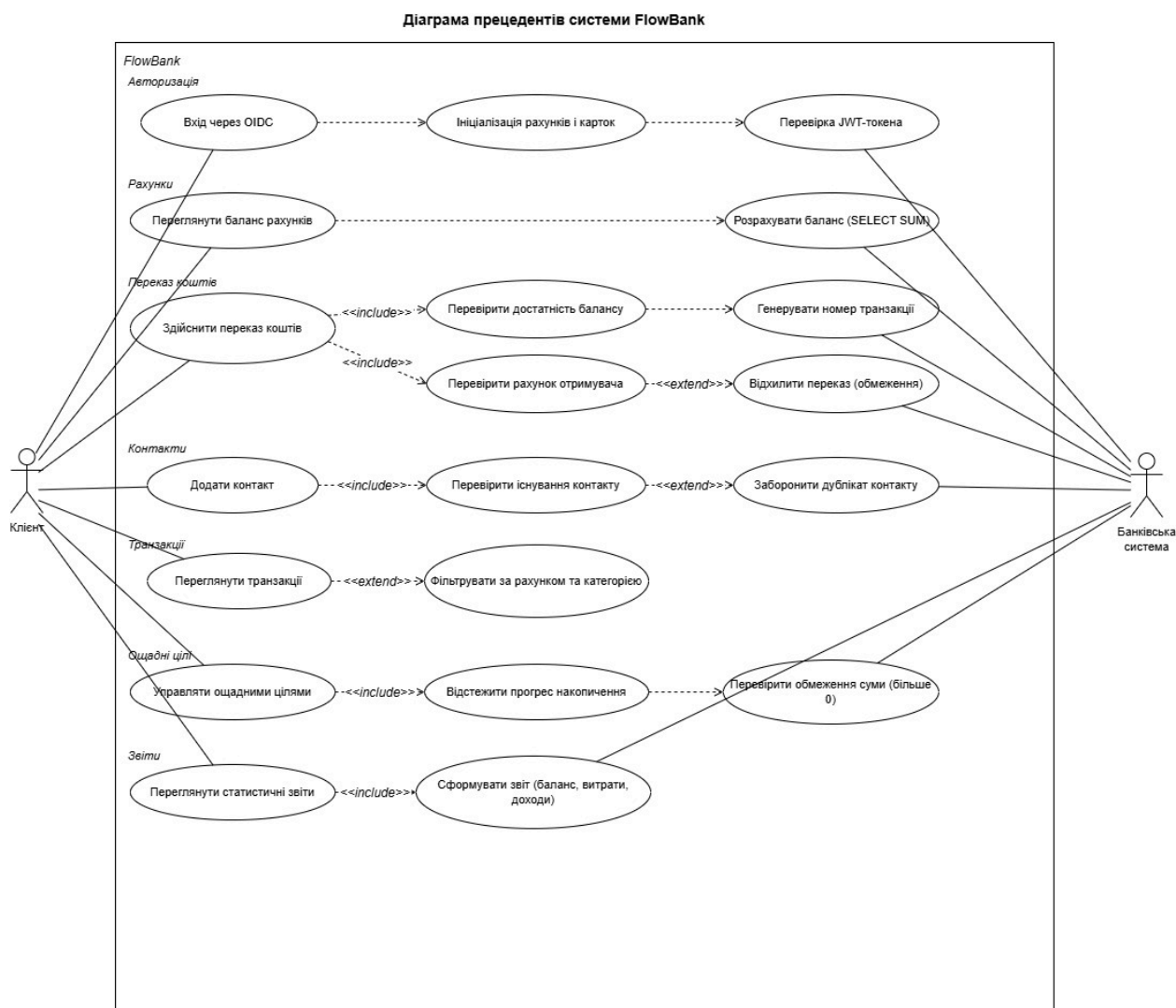


Рис. 1.1 — Діаграма прецедентів системи FlowBank

Діаграма активностей (рис. 1.2) деталізує повний сценарій взаємодії між Клієнтом і Банківською системою: від входу до виконання основних операцій — переказу коштів та управління контактами.

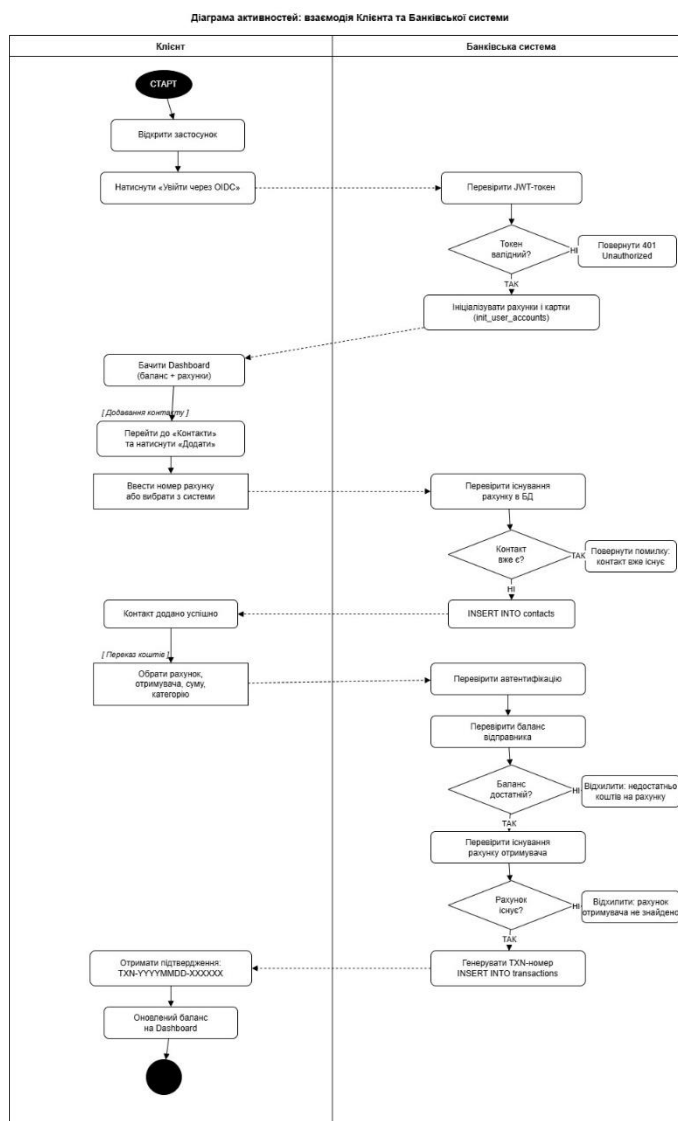


Рис. 1.2 — Діаграма активностей: взаємодія Клієнта та Банківської системи

1.6 Вимоги до системи

Вимоги до FlowBank сформовано у двох категоріях — функціональні та нефункціональні — відповідно до стандарту IEEE 830. Функціональні фіксують, що система виконує; нефункціональні — за яких умов і з якою якістю.

Функціональні вимоги

Вимоги згруповано за сімома модулями системи.

Модуль автентифікації:

- реєстрація та вхід реалізовані через форму з валідацією введених даних;
- після автентифікації відкривається доступ до персональних даних та карток;
- сесія зберігається між запусками застосунку без повторного входу.

Модуль управління картками:

- головний екран відображає картки користувача з поточним балансом та реквізитами;
- по кожній картці доступна детальна інформація в окремому поданні;
- підтримуються три типи карток: дебетова, кредитна, преміум;
- користувач повинен мати можливість заблокувати/розблокувати картку.

Модуль переказів:

- система повинна забезпечувати переказ коштів між власними картками;
- система повинна забезпечувати переказ коштів на картки збережених контактів;
- при переказі система повинна перевіряти наявність достатнього балансу;
- кожен переказ повинен отримувати унікальний номер транзакції.

Модуль транзакцій:

- повна історія транзакцій зберігається та доступна в будь-який момент;
- фільтрація — за датою та категорією витрат;
- кожна транзакція отримує один із трьох статусів: виконано, в обробці, відхилено.

Модуль накопичень:

- накопичувальний рахунок створюється з довільною назвою та цільовою сумою;
- прогрес відображається у відсотках від встановленої цілі;
- користувач повинен мати можливість поповнювати та закривати накопичення.

Модуль аналітики:

- система повинна надавати звіт про витрати у розрізі категорій;
- система повинна відображати динаміку доходів та витрат за обраний період;
- аналітика повинна формуватись на основі реальних транзакційних даних.

Модуль профілю:

- користувач повинен мати можливість переглядати та редагувати персональні дані;
- система повинна підтримувати завантаження фото профілю.

Нефункціональні вимоги

Нефункціональні вимоги охоплюють сім параметрів якості системи. Вони не описують окремих функцій, але визначають умови, за яких система залишається придатною до експлуатації.

Продуктивність: 300 мс — максимальний час відгуку інтерфейсу. Сторінка при стабільному з'єднанні завантажується не довше двох секунд.

Надійність: збій мережі не призводить до втрати транзакційних даних. Користувач отримує зрозуміле повідомлення про помилку замість порожнього екрана.

Безпека: паролі не зберігаються у відкритому вигляді — лише хеш. Усі запити до API — виключно по HTTPS. Номер картки на екрані замаскований.

Зручність використання: навігація побудована за принципами Material Design. Коректне відображення гарантовано для екранів від 5 до 7 дюймів.

Сумісність: Android 8.0 і вище; веб-версія — Chrome, Firefox, Safari.

Масштабованість: нові модулі підключаються без зміни існуючої кодової бази.

Доступність: хмарне розгортання на Render + Supabase — uptime 99% і вище; перебоїв у плановому режимі немає.

Висновки до розділу 1. Ринок мобільного банкінгу України засвідчив стійке зростання навіть в умовах воєнного стану — частка безготівкових розрахунків перевищила 75%. Порівняння monobank, ПриватБанк24 та ПУМБ Online показало: аналітика витрат у жодному з них не виходить за межі базової категоризації, що стало відправною точкою для формування вимог до FlowBank. Серед 26 визначених функціональних вимог пріоритет отримали модулі переказів, аналітики та накопичень. Технологічний стек — React 18, Node.js, PostgreSQL, Capacitor — підбрано за критеріями зрілості, документованості та відповідності задачам кросплатформної мобільної розробки.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування бази даних

Концептуальна модель (ER-діаграма)

Концептуальна модель визначає структуру даних системи FlowBank на рівні сутностей і зв'язків між ними — до прив'язки до конкретної СУБД чи фізичної схеми. Центральною сутністю є User: усі інші таблиці пов'язані з нею безпосередньо або через проміжні відношення.

Кожен користувач може мати декілька банківських карток — сутність Card зберігає номер, баланс, тип і статус. Тип картки виноситься в окрему довідникову таблицю CardType з фіксованими seed-даними: дебетова, кредитна, преміум. Це дозволяє розширювати перелік типів без зміни структури основної таблиці.

Накопичувальні рахунки (Saving) прив'язані до користувача і зберігають назву, цільову суму та поточний прогрес. Контакти (Contact) — це збережені записи для швидких переказів: ім'я, номер картки отримувача, аватар.

Транзакції (Transaction) утворюють окремий шар: кожен запис містить два зовнішні ключі до таблиці Card — картку-відправника і картку-отримувача. Такий підхід дозволяє зберігати повну двосторонню історію переказів без дублювання записів.

Зв'язки між сутностями та їх кардинальність:

- User → Card: один до багатьох;
- User → Saving: один до багатьох;
- User → Contact: один до багатьох;
- Card → Transaction (відправник): один до багатьох
- Card → Transaction (отримувач): один до багатьох;
- CardType → Card: один до багатьох.

ER-діаграма системи FlowBank наведена на рис. 2.1. Вона охоплює всі вісім таблиць із зазначенням атрибутів, типів даних і кардинальності відносин.

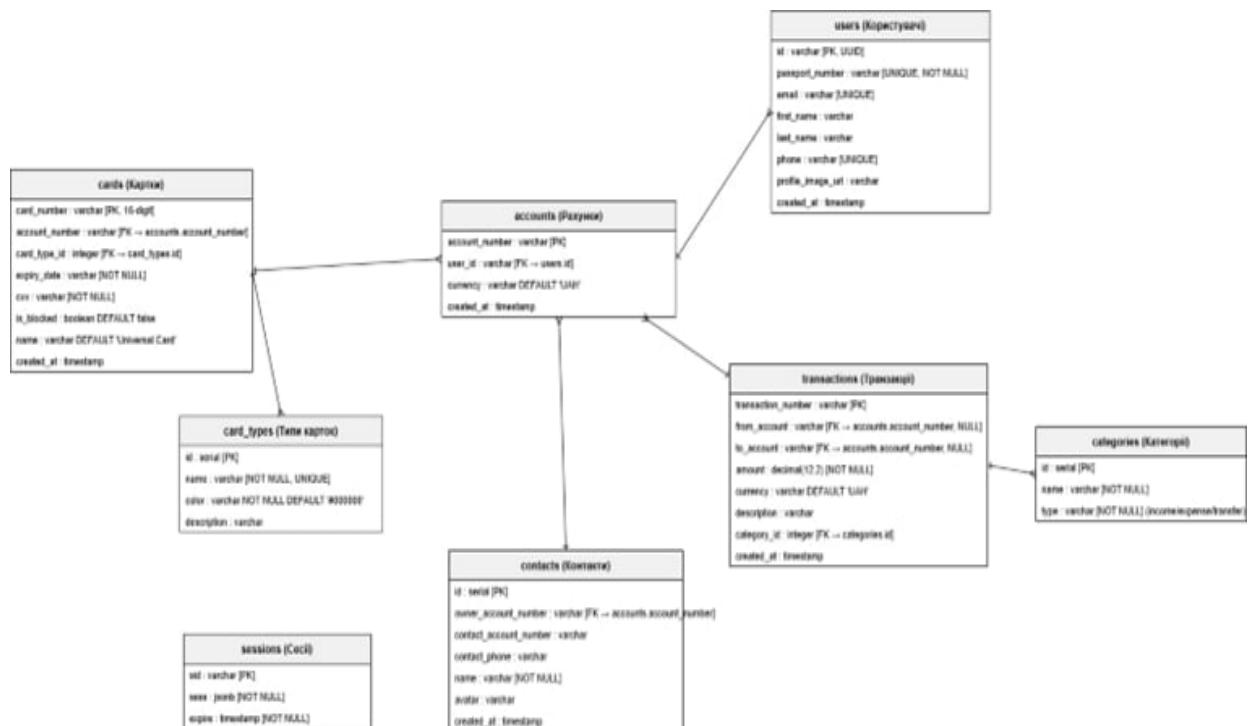


Рис. 2.1 — ER-діаграма бази даних системи FlowBank

Логічна модель

Логічна модель бази даних FlowBank реалізована у вигляді восьми реляційних таблиць. Нижче наведено опис кожної таблиці із зазначенням атрибутів та обмежень цілісності.

Таблиця 2.1

Опис таблиць бази даних системи FlowBank

Таблиця	Основні атрибути	Призначення
users	id, username, email, first_name, last_name, phone, passport_number, profile_image_url	Зберігає персональні дані зареєстрованих користувачів
cards	id, user_id, card_number, expiry_date, cvv, balance, currency, type, credit_limit, is_blocked, name, color	Банківські картки користувачів із фінансовими реквізитами
transactions	id, transaction_number, amount, currency, description, date, sender_card_id, receiver_card_id, category, status, type	Фінансові операції між картками
contacts	id, user_id, name, card_number, avatar	Збережені контакти для швидких переказів
savings	id, user_id, card_id, name, target_amount, current_amount, emoji, color, created_at	Накопичувальні рахунки з фінансовими цілями
card_types	code, name, color, default_credit_limit	Довідник типів карток (black, white, platinum, iron, cashback)
accounts	account_number, user_id, currency	Банківські рахунки для майбутньої INSERT-only архітектури
sessions	sid, sess, expire	Серверні сесії користувачів

На рівні схеми PostgreSQL цілісність підкріплено декількома механізмами: зовнішні ключі (FOREIGN KEY) зв'язують таблиці і запобігають появі «висячих» записів, NOT NULL захищає критичні поля від порожніх значень, а UNIQUE на card_number унеможливорює дублювання карткових реквізитів.

2.2 Вибір системи управління базою даних

Рішення щодо СУБД у банківському застосунку не варто приймати формально — від надійності зберігання залежить цілісність фінансових даних. Після порівняльного аналізу альтернатив для FlowBank обрано PostgreSQL.

Банківські дані мають чітку структуру: картка належить користувачу, транзакція прив'язана до картки, накопичення — теж. Реляційна модель описує ці відношення природно, тоді як документна або графова дали б лише зайву складність. Крім того, SQL виявився незамінним для аналітичних запитів — групування транзакцій за категорією та датою у Recharts-діаграмах писати куди зручніше на SQL, ніж у коді.

PostgreSQL обрано серед реляційних СУБД з таких міркувань. PostgreSQL вирізняється серед відкритих реляційних СУБД насамперед механізмом MVCC: паралельні читання не блокують запити на запис, і навпаки. Для банківського застосунку з одночасними операціями кількох користувачів це критично. Підтримка JSONB, тригерів і збережених процедур закриває потреби без залучення додаткових інструментів. Нарешті, PostgreSQL є фактичним стандартом для хмарних платформ — Supabase, RDS, Cloud SQL — тому вибір хостингу не залежить від СУБД.

Базу даних розміщено на Supabase — відкритій альтернативі Firebase, побудованій поверх PostgreSQL 15. Supabase взяв на себе все, що не стосується бізнес-логіки: резервні копії, SSL-шифрування, адміністративний веб-інтерфейс. Сервер у регіоні EU West (Франкфурт) мінімізує затримку для українських користувачів.

Таблиця 2.2

Порівняльна характеристика СУБД

Критерій	PostgreSQL	MySQL	MongoDB
Модель даних	Реляційна	Реляційна	Документна
ACID-транзакції	Повна підтримка	Повна підтримка	Обмежена
Зовнішні ключі	Так	Так	Ні
JSON підтримка	JSONB (нативна)	JSON (текст)	Нативна

Критерій	PostgreSQL	MySQL	MongoDB
Відкритий код	Так	Так	Так
Хмарні платформи	Supabase, RDS	PlanetScale	Atlas
Підходить для фінансів	✓ Оптимально	✓ Прийнятно	✗ Не рекомендовано

Як видно з таблиці 2.2, PostgreSQL є оптимальним вибором для системи FlowBank завдяки повній підтримці ACID-транзакцій, нативній роботі з JSON та широкій підтримці хмарними платформами. Для взаємодії з PostgreSQL у проєкті застосовано Drizzle ORM: кожен запит типізований на рівні TypeScript, міграції схеми генеруються автоматично командою `drizzle-kit generate`.

Реалізація бази даних

База даних PostgreSQL розгорнута на хмарній платформі Supabase. Схема бази даних описана декларативно за допомогою Drizzle ORM у файлі `db/schema.ts`. Синхронізація між TypeScript-типами та реальною структурою таблиць підтримується автоматично: при зміні схеми `drizzle-kit generate` оновлює файл міграції, а типи перекомпілюються.

Фрагмент схеми таблиці `cards` представлена на Рис. 2.2:

```
export const cards = pgTable("cards", {
  id: serial("id").primaryKey(),
  userId: text("user_id").notNull(),
  cardNumber: text("card_number").notNull().unique(),
  expiryDate: text("expiry_date").notNull(),
  cvv: text("cvv").notNull(),
  balance: numeric("balance", { precision: 15, scale: 2 })
    .notNull().default("0"),
  currency: text("currency").notNull().default("UAH"),
  type: text("type").notNull().default("black"),
  creditLimit: numeric("credit_limit", { precision: 15,
    scale: 2 }),
  isBlocked: boolean("is_blocked").notNull().default(false),
  name: text("name"),
  color: text("color"),
});
```

Рис. 2.2 — Схема таблиці `cards`

Схема бази даних версіонована за допомогою чотирьох SQL-файлів (`migrations/01`–`migrations/04`). Кожен з них охоплює окремий етап формування схеми:

- початкове створення таблиць,
- підключення тригерів,
- оголошення збережених функцій
- заповнення довідників.

Розгортання виконується в екземплярі Supabase засобами MCP-інтеграції.

Довідникова таблиця `card\_types` попередньо заповнена п'ятьма записами відповідно до продуктової лінійки:

- black (ліміт 50 000 грн)
- white (20 000 грн)
- platinum (100 000 грн)
- iron (30 000 грн)
- cashback (15 000 грн).

Кожен запис визначає назву типу, його відображувану назву українською та граничну суму кредитування.

Тригери та збережені функції

Частину бізнес-логіки перенесено безпосередньо на рівень бази даних у вигляді тригерів та збережених функцій PostgreSQL. Перевага — цілісність виконується незалежно від того, звідки надходить зміна: через API, міграцію чи прямий SQL-запит у консолі Supabase.

generate_transaction_number() — Збережена функція, що генерує унікальний номер транзакції у форматі TXN-YYYYMMDD-XXXXXXXXX (де X — випадковий буквено-цифровий символ). Викликається тригером `trg_auto_txn_number` перед вставкою нового запису до таблиці `transactions`.

trg_auto_txn_number — Тригер BEFORE INSERT на таблиці transactions. Автоматично присвоює унікальний номер транзакції при створенні нового запису, що унеможливорює вставку транзакції без ідентифікатора.

trg_validate_passport — Тригер BEFORE INSERT OR UPDATE на таблиці users. Перевіряє формат паспортного номера за регулярним виразом (серія AA та 6 цифр для внутрішнього паспорта або 9 цифр для ID-картки). Відхиляє запис із некоректним форматом.

trg_validate_savings — Тригер BEFORE INSERT OR UPDATE на таблиці savings. Перевіряє, що поточна сума накопичення (current_amount) не перевищує цільову (target_amount), забезпечуючи логічну цілісність даних.

report_expenses_by_category() — Збережена функція, що повертає зведений звіт про витрати користувача у розрізі категорій транзакцій за вказаний часовий проміжок. Використовується модулем аналітики.

report_income_expense_summary() — Збережена функція, що повертає зведений звіт про загальний обсяг доходів та витрат користувача за місяць для побудови фінансової аналітики.

Висновки до розділу 2. Трирівнева клієнт-серверна архітектура FlowBank забезпечила чіткий розподіл між шарами представлення, бізнес-логіки та даних. Реляційна модель із восьми таблиць покриває всі сутності предметної області — від карток і транзакцій до накопичень та контактів. Окремо описано тригери PostgreSQL для підтримки цілісності фінансових даних. Навігаційна структура з 13 маршрутів та REST API з 19 ендпоінтів сформували повну специфікацію системи ще до початку реалізації — це суттєво скоротило кількість архітектурних змін під час кодування.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура системи

FlowBank побудовано на трирівневій клієнт-серверній архітектурі. Клієнт, сервер і база даних — три окремі шари з чітко розмежованою відповідальністю. Такий поділ не є архітектурною надмірністю: він визначає, де зосереджена логіка, хто відповідає за стан даних і як змінювати один шар без торкання інших.

Загальна архітектура (клієнт-сервер)

FlowBank реалізовано за трирівневою клієнт-серверною архітектурою: рівень представлення (React SPA), рівень бізнес-логіки (Express.js) та рівень даних (PostgreSQL). Межі між шарами жорсткі — кожен відповідає виключно за свою зону, що дозволяє масштабувати або замінювати будь-який рівень незалежно від інших. При зростанні навантаження достатньо додати інстанцій серверу, не торкаючись клієнта чи бази даних.

Взаємодія між рівнями відбувається через HTTP/HTTPS. Клієнт відправляє асинхронні запити до REST API, отримує відповіді у форматі JSON та оновлює інтерфейс без перезавантаження сторінки — стан керується через TanStack Query, який кешує серверні відповіді і синхронізує їх з UI. Сервер на Express.js приймає запит, виконує бізнес-логіку та звертається до PostgreSQL через Drizzle ORM. База повертає результат — сервер формує JSON-відповідь і відправляє її клієнту.

Для мобільної платформи Capacitor виступає додатковим шаром між React-застосунком та Android WebView, забезпечуючи доступ до нативних API пристрою. Логіка взаємодії з сервером при цьому залишається незмінною — мобільний клієнт використовує ті самі HTTP-запити, що й веб-версія. Окремого мобільного API не передбачено.

Схему розміщення компонентів системи — клієнти (браузер і Android), сервер на Render.com, Supabase PostgreSQL та сервер авторизації OIDC — наведено на рис. 3.1.

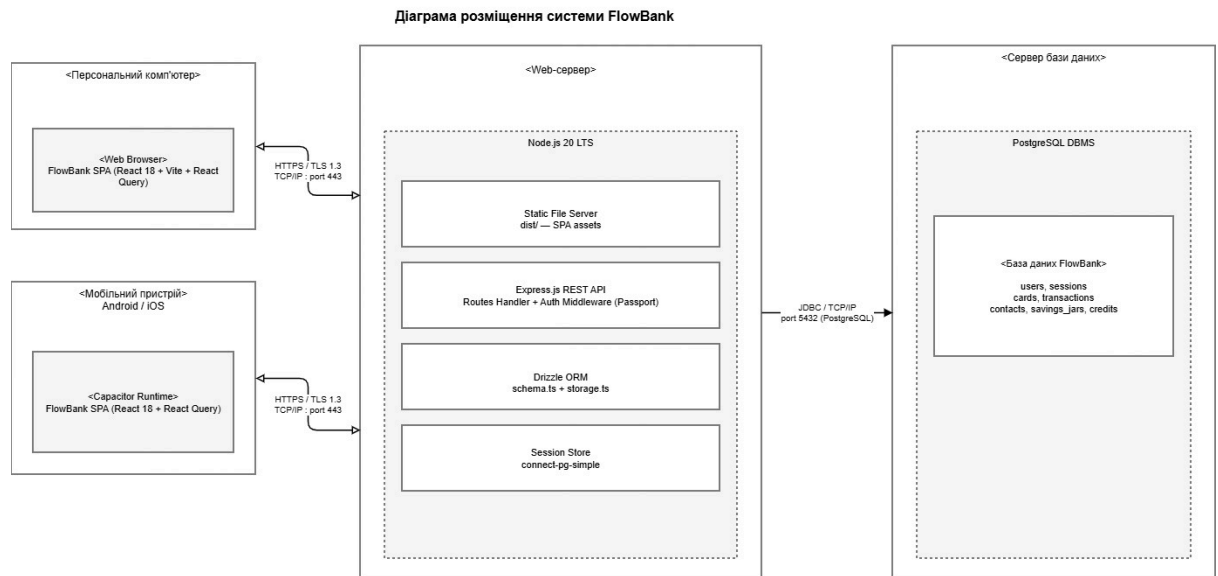


Рис. 3.1 — Діаграма розміщення системи FlowBank

3.2 Вибір та обґрунтування технологій

Технологічний стек закладає фундамент усіх наступних рішень: вдалий вибір скорочує технічний борг, невдалий — множить його. Для FlowBank критерії відбору були практичними: чи вирішує технологія конкретну задачу, наскільки зріла її екосистема, чи є документація та чи підтримується кросплатформна розробка.

Фронтенд (React 18 + TypeScript)

Для побудови клієнтської частини обрано React 18 — бібліотеку від Meta, яка станом на 2024 рік залишається найпоширенішим інструментом фронтенд-розробки (понад 40% розробників за даними Stack Overflow Developer Survey). Головна причина вибору — не популярність сама по собі, а зрілість екосистеми: стабільні хуки, велика кількість готових рішень і широка спільнота, що критично при обмеженому часі на розробку.

Компонентна модель React добре лягла на структуру банківського інтерфейсу: картка, рядок транзакції, форма переказу — кожен елемент стає

ізолюваним компонентом з власним станом. Virtual DOM вирішив практичну проблему: при оновленні балансу або списку транзакцій перерисовується лише той фрагмент дерева, де відбулися зміни, а не вся сторінка.

React 18 з Concurrent Rendering дав змогу пріоритизувати термінові оновлення — наприклад, реакцію на натискання кнопки — над фоновим завантаженням даних аналітики. Хуки `useState`, `useEffect` та `useMemo` покрили весь спектр потреб у керуванні станом без класових компонентів.

TypeScript підключено з першого рядка проєкту — не як стиль, а як практична необхідність. Коли схема бази даних, API-контракт і React-компонент описані одними типами, компілятор ловить невідповідності ще до запуску. На практиці це виявилось особливо цінним при рефакторингу ендпоінтів: зміна типу поля в `shared/schema.ts` одразу підсвічувала всі місця у клієнтському коді, де очікувалась стара форма даних. TanStack React Query взяв на себе кешування серверного стану та інвалідацію після мутацій, Wouter — легковагу клієнтську маршрутизацію.

Бекенд (Node.js + Express.js)

Серверну частину реалізовано на Node.js — середовищі виконання JavaScript поверх V8-движка Google. Для банківського API з великою кількістю одночасних клієнтських запитів підходить асинхронна, подієво-орієнтована модель Node.js: кожен запит до бази даних виконується без блокування потоку, тобто сервер продовжує обробляти інші запити під час очікування відповіді від PostgreSQL.

Практичний аргумент на користь Node.js у цьому проєкті — єдина мова стека. TypeScript однаково працює і на клієнті, і на сервері, тому валідаційні схеми Zod та типи Drizzle оголошені один раз у `shared/` і імпортуються з обох боків. Ніякого дублювання типів між фронтендом та бекендом.

Як HTTP-фреймворк обрано Express.js — мінімалістичне рішення без жорсткої структури, де розробник сам вирішує, як організувати маршрути та middleware. Для проєкту такого масштабу це виявилось зручнішим за більш

opinionated фреймворки: всі 19 ендпоінтів зосереджені в одному файлі `routes.ts` з прозорою послідовністю `middleware`-ланцюжка.

База даних (PostgreSQL + Supabase)

Вибір PostgreSQL як СУБД визначився характером даних. Картки, рахунки, транзакції та користувачі — це пов'язані сутності з чіткими відношеннями, і реляційна модель з зовнішніми ключами описує їх природно. NoSQL тут надлишковий: структура даних стабільна та відома заздалегідь.

Для фінансових операцій ACID-гарантії PostgreSQL принципів. Переказ між картками реалізовано як одну транзакцію через `db.transaction()` у Drizzle: якщо списання пройшло, а зарахування впало з будь-якої причини — транзакція відкочується повністю. Це не опціональна зручність, а вимога коректності. Крім того, бізнес-логіка накопичувальних рахунків винесена у тригери PostgreSQL — зміна балансу картки автоматично перераховує відсоткове нарахування без додаткових запитів із сервера.

Для хостингу PostgreSQL обрано Supabase — керована платформа, де налаштування реплікацій, резервних копій та SSL відбуваються автоматично. Це дало змогу зосередитись на схемі даних, а не на адмініструванні сервера. Drizzle ORM підключається до Supabase через стандартний рядок з'єднання і надає типізовані запити: кожна таблиця має відповідний TypeScript-тип, тому невідповідність між схемою і кодом виявляється на етапі компіляції.

Мобільна платформа (Capacitor)

Capacitor — кросплатформний рантайм від команди Ionic Framework. Принцип роботи: готовий веб-застосунок запаковується у нативну оболонку WebView для Android чи iOS. На відміну від React Native, який рендерить нативні компоненти, Capacitor виконує той самий React-код, що і у браузері, лише додає JavaScript-міст до нативних API пристрою — камери, геолокації, push-сповіщень, файлової системи.

Вибір Capacitor у цьому проєкті був логічним: наявний React-застосунок не потрібно переписувати — достатньо кількох команд у терміналі. Єдина

кодова база для веб і мобільної версії означає, що виправлення помилки або додавання функції відбувається один раз. Широка екосистема плагінів закриває потреби в нативних API без написання Java чи Kotlin. На виході — APK-файл, встановлений і протестований на реальному пристрої.

Інтерфейс побудовано на shadcn/ui — наборі компонентів поверх Radix UI із підтримкою доступності (ARIA) з коробки. У поєднанні з Tailwind CSS це дало консистентний дизайн без написання власних CSS-файлів: кожен відступ, колір і брейкпоінт задається утилітарними класами прямо у JSX-розмітці. Framer Motion відповідає за анімації переходів між екранами та мікроінтеракції кнопок — декларативний підхід вписався в компонентну модель React без зайвої складності.

3.3 Структура frontend

Клієнтська частина структурована за Feature-Based Architecture: код групується за функціональними модулями, а не за технічними шарами. При зростанні кодової бази це рішення виправдовує себе — межі відповідальності

кожного модуля залишаються чіткими, а пошук потрібного файлу не перетворюється на навігацію по абстрактних шарах. Структура директорій

клієнтської частини наведено на Рис. 3.2

client/src/	
components/	— перевикористовувані UI-компоненти (кнопки, картки, модалки)
pages/	— компоненти сторінок (Home, Cards, Transfer, History, Savings, Analytics, Profile)
hooks/	— кастомні React-хуки для роботи з API
lib/	— утиліти, константи, допоміжні функції
types/	— TypeScript-типи та інтерфейси
App.tsx	— кореневий компонент, налаштування маршрутизації
main.tsx	— точка входу застосунку

Рис. 3.2 — Структура директорій клієнтської частини

Маршрутизація реалізована за допомогою бібліотеки Wouter — легкого альтернативного рішення до React Router з API на основі хуків. Застосунок налічує 13 маршрутів: головна сторінка (/), картки (/cards), перекази (/transfer), історія (/history), платежі (/payments), накопичення (/savings), аналітика (/analytics), профіль (/profile), сповіщення (/notifications), кредити (/credits), благодійність (/charity), додатково (/more) та налаштування (/settings).

Серверний стан у клієнтській частині керується через TanStack React Query: запити кешуються, автоматично повторюються при мережевих помилках, а після мутацій (наприклад, переказу) відповідний кеш інвалідується — баланс оновлюється без ручного перезавантаження. Глобальний стан (профіль користувача, поточна картка) живе у React Context API.

Структурна діаграма пакетів клієнтської та серверної частин системи наведена на рис. 3.3. Діаграма відображає залежності між основними модулями: сторінками, компонентами, хуками та серверними пакетами (routes, auth, database).

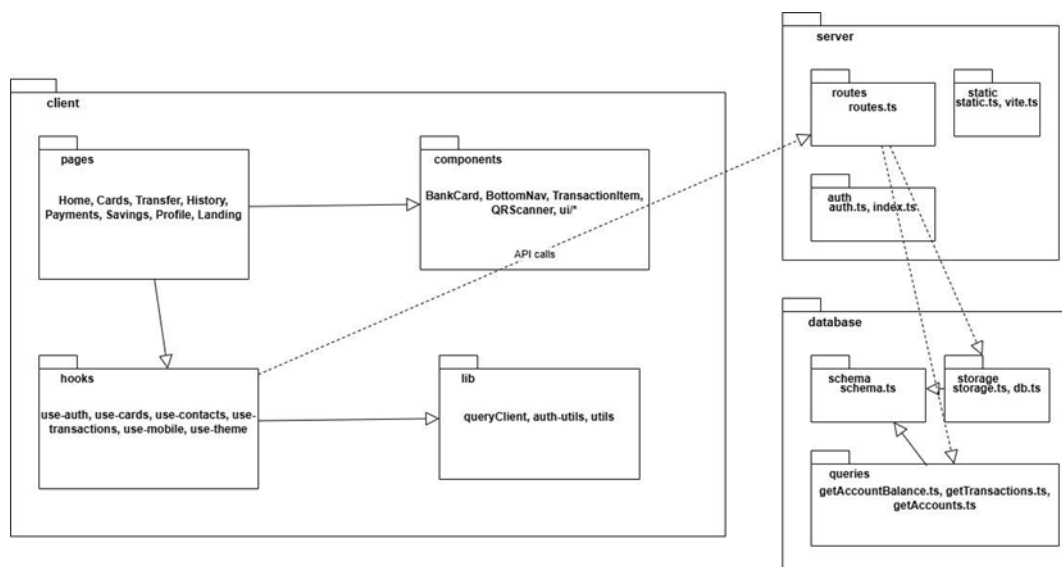


Рис. 3.3 — Структурна діаграма пакетів системи FlowBank

Структура backend (REST API)

Серверна частина організована за принципом поділу на маршрути, обробники та схеми даних. Express-застосунок налаштовано з підтримкою JSON-тіл запитів, обробкою CORS та централізованою обробкою помилок через middleware. Структура директорій серверної частини наведена на Рис. 3.4

server/	
routes.ts	— реєстрація всіх API-маршрутів
storage.ts	— шар доступу до даних (Drizzle ORM запити)
index.ts	— точка входу, налаштування Express
db/	
schema.ts	— схема бази даних (Drizzle)
seed.ts	— початкові дані (card_types)
migrations/	— SQL-міграції (01-04)
shared/	
schema.ts	— спільні TypeScript-типи (Zod схеми)

Рис. 3.4 — Структура директорій серверної частини

3.4 Проєктування інтерфейсу користувача

Карта екранів

При розробці навігаційної моделі FlowBank за основу взято концепцію нижньої панелі керування (bottom navigation bar), що є характерним рішенням для мобільних застосунків із фінансовою спрямованістю. Вибір зумовлений тим, що така компоновка дозволяє охопити основні сценарії роботи одним рухом великого пальця — без додаткових жестів чи вкладених меню. Панель об'єднує п'ять вузлів: Головна, Картки, Перекази, Аналітика та Профіль. Периферійні функції виведено в розділ «Більше» або реалізовано як контекстні дії всередині відповідних екранів.

Архітектура екранів організована таким чином:

- **Головна (/)**. Після входу користувач потрапляє на дашборд: тут зосереджено вітання, актуальний баланс прив'язаної картки, три швидкі

дії — поповнення, переказ, платіж — та скорочена стрічка нещодавніх операцій.

- **Картки (/cards).** Екран реалізовано у форматі горизонтального слайдера: кожна картка займає окремий слайд із повним набором атрибутів, перехід між ними виконується свайпом.

- **Перекази (/transfer).** Сторінка являє собою покроковий сценарій: вибір джерела списання, зазначення отримувача — вручну або зі збережених контактів, введення суми та фінальне підтвердження операції.

- **Історія (/history).** Повна хронологія транзакцій із гнучким фільтруванням: за видом операції, категорією витрат і довільним часовим відрізком.

- **Платежі (/payments).** Агрегований розділ для сплати комунальних рахунків, мобільного зв'язку та інших регулярних зобов'язань.

- **Накопичення (/savings).** Перелік ощадних рахунків доповнено індикаторами прогресу відносно встановленої цілі; передбачено можливість відкриття нового рахунку та збільшення наявного залишку.

- **Аналітика (/analytics).** Розділ містить два типи графічного представлення даних: кругову діаграму структури витрат у розрізі категорій та стовпчасту — для порівняння доходів і витрат по місяцях.

- **Профіль (/profile).** Зберігає ідентифікаційні дані власника, фото та банківські реквізити.

- **Налаштування (/settings).** Об'єднує параметри безпеки облікового запису, управління сповіщеннями та візуальними налаштуваннями інтерфейсу.

Візуальне втілення описаних рішень представлено на рис. 3.5–3.6: перший ілюструє форму автентифікації, другий — головний екран із балансом та переліком останніх операцій.

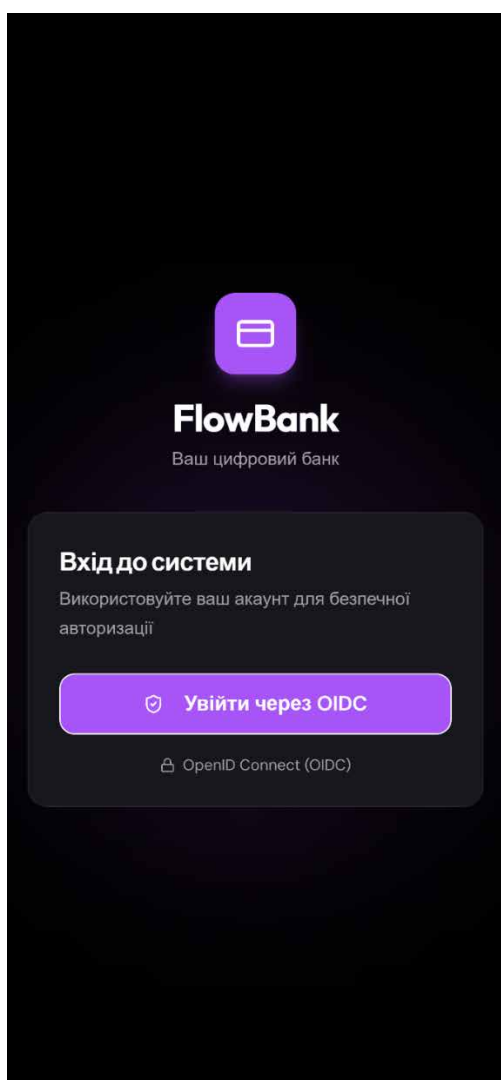


Рис. 3.5 — Екран авторизації
застосунку FlowBank

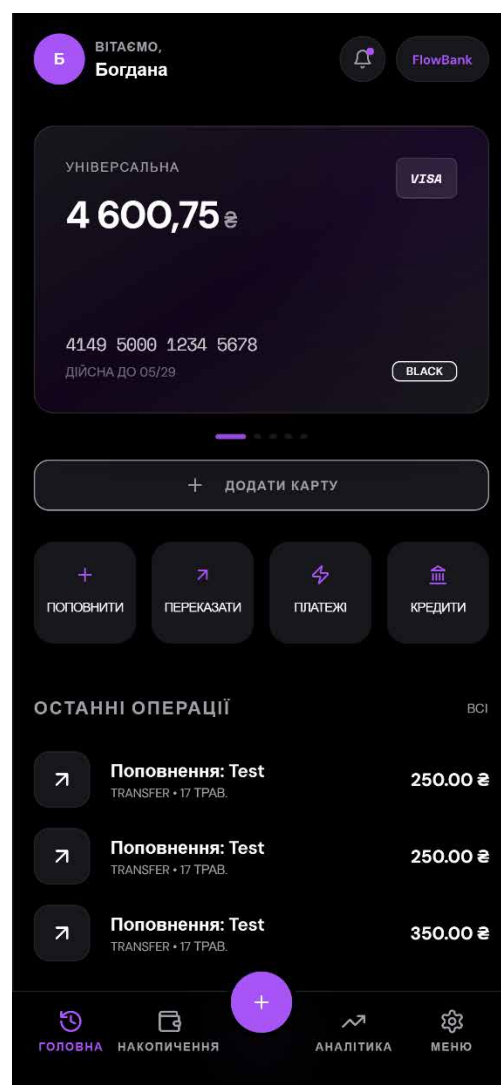


Рис. 3.6 — Головний екран
застосунку FlowBank

Діаграма активності, наведена на рис. 3.7, описує послідовність дій користувача під час виконання типового сценарію переказу коштів у системі FlowBank — від відкриття застосунку до отримання підтвердження успішної транзакції. Кожен вузол рішення відображає альтернативні шляхи виконання залежно від результату перевірок (автентифікація, наявність коштів, валідність реквізитів).

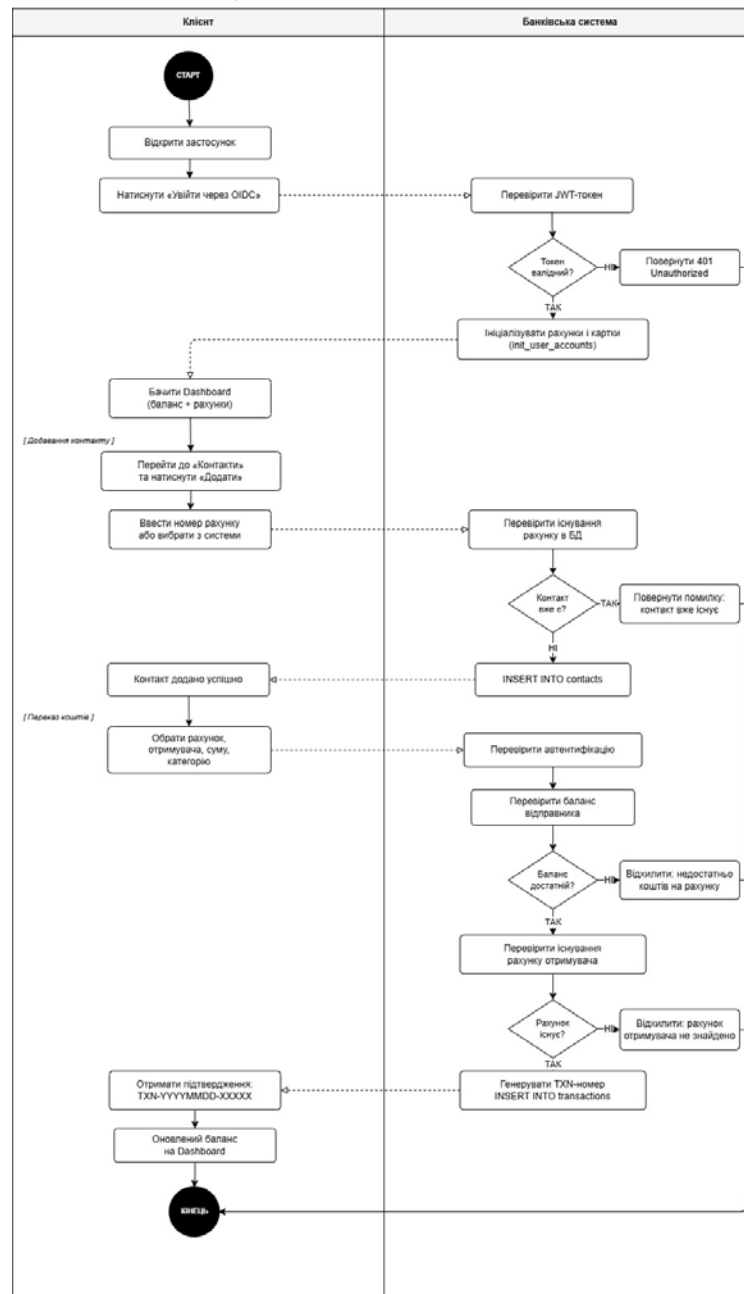


Рис. 3.7 — Діаграма активності: сценарій переказу коштів

Wireframes основних екранів

Дизайн застосунку FlowBank базується на компонентній бібліотеці shadcn/ui, побудованій на основі примітивів Radix UI. Така основа гарантує доступність (accessibility) компонентів відповідно до стандарту WCAG 2.1 «з коробки». Стилзація — виключно утилітарні класи Tailwind CSS: кольори, відступи та типографіка єдиним дизайн-токеном поширюються на весь застосунок без написання жодного CSS-файлу.

Колірна схема реалізує підтримку темної (dark) та світлої (light) теми через CSS-змінні. Основна типографіка — системний шрифт стека для забезпечення нативного відчуття на кожній платформі. Картки відображаються з індивідуальними кольорами відповідно до типу: чорний (Black), білий (White), платиновий (Platinum), залізний (Iron) та кешбек (Cashback).

Переходи між екранами та мікроінтеракції кнопок побудовано на Framer Motion за принципом «meaningful motion»: анімація або вказує, куди перемістився контент, або підтверджує дію — але ніколи не існує заради себе. Жоден перехід не перевищує 300 мс — рекомендація Material Design, яку легко порушити, але складно виправдати перед користувачем.

3.5 Проєктування API

Специфікація REST-ендпоінтів

API системи FlowBank побудоване відповідно до принципів REST (Representational State Transfer). Усі ресурси адресуються іменниками у множині (/api/cards, /api/transactions), HTTP-методи семантично відповідають операціям (GET — читання, POST — створення, PATCH — часткове оновлення, DELETE — видалення). Відповіді повертаються у форматі JSON із відповідними HTTP-статус кодами.

Таблиця 3.1

Специфікація REST API системи FlowBank

Метод	Ендпоінт	Опис	Відповідь
GET	/api/user	Отримати профіль користувача	200 User
PATCH	/api/user	Оновити профіль користувача	200 User
GET	/api/cards	Отримати список карток	200 Card[]

Метод	Ендпоінт	Опис	Відповідь
POST	/api/cards	Створити нову картку	201 Card
GET	/api/cards/:id	Отримати картку за ID	200 Card
PATCH	/api/cards/:id	Оновити дані картки	200 Card
GET	/api/transactions	Отримати список транзакцій	200 Transaction[]
POST	/api/transactions	Створити транзакцію (переказ)	201 Transaction
GET	/api/contacts	Отримати список контактів	200 Contact[]
POST	/api/contacts	Додати контакт	201 Contact
DELETE	/api/contacts/:id	Видалити контакт	200 OK
GET	/api/savings	Отримати список накопичень	200 Saving[]
POST	/api/savings	Створити накопичення	201 Saving
PATCH	/api/savings/:id	Оновити накопичення	200 Saving
DELETE	/api/savings/:id	Видалити накопичення	200 OK
GET	/api/reports/balance	Звіт по балансу карток	200 Report
GET	/api/reports/categories	Витрати за категоріями	200 Report[]
GET	/api/reports/summary	Зведений звіт доходи/витрати	200 Report
GET	/api/reports/savings	Звіт по накопиченнях	200 Report[]

Обробка помилок реалізована централізовано: при виникненні виключення у будь-якому обробнику маршруту воно перехоплюється middleware та повертається клієнту у стандартизованому форматі {"message": "...", "code": ...} з відповідним HTTP-статусом (400 — некоректний запит, 404 — ресурс не знайдено, 500 — внутрішня помилка сервера). Валідація вхідних даних виконується через схеми Zod — некоректний запит відхиляється із детальним повідомленням ще до звернення до бази даних.

Діаграма послідовності для сценарію переказу коштів (рис. 3.8) відображає детальну взаємодію між React Frontend, Express API та PostgreSQL DB: від відкриття сторінки /transfer до отримання підтвердження про успішне виконання операції.

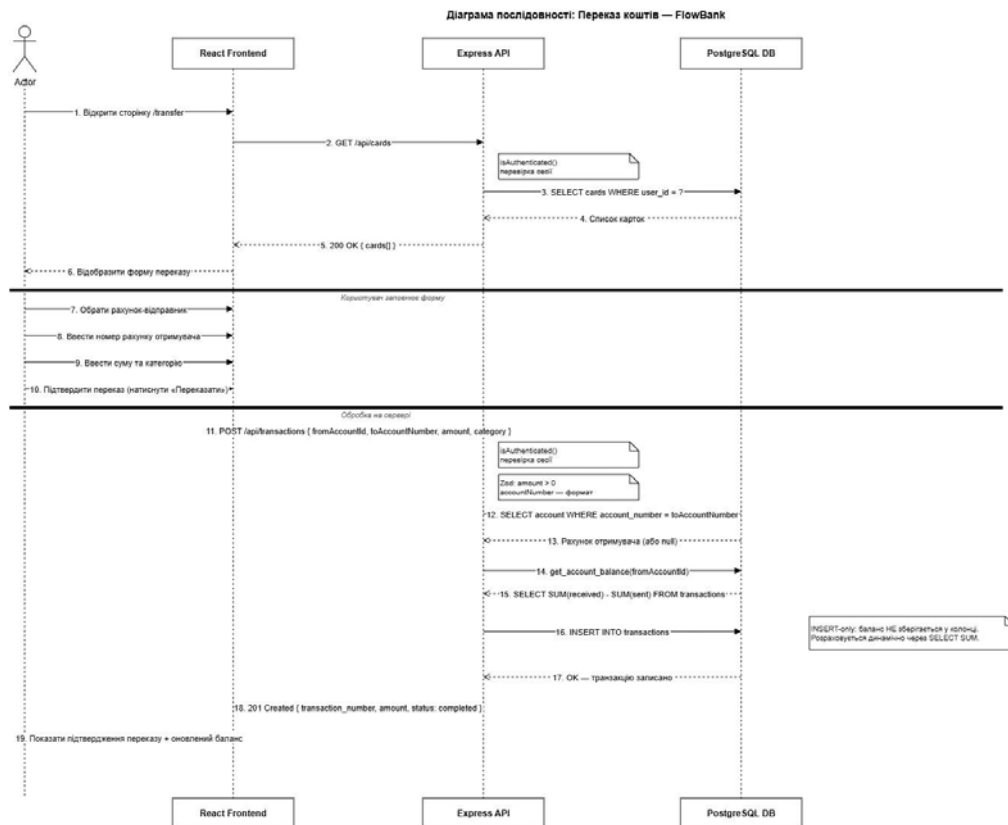


Рис. 3.8 — Діаграма послідовності: сценарій переказу коштів

3.6 Структура проєкту

FlowBank реалізовано як монорепозиторій: клієнтська частина, сервер та мобільний застосунок живуть у спільному репозиторії. На практиці це означає, що TypeScript-типи з shared/ імпортуються без копіювання, npm-залежності встановлюються однією командою, а деплой на Render запускається єдиним pipeline. Загальна структура монорепозиторію наведена на Рис. 3.9.

flowbank/	
client/	— React SPA (фронтенд)
src/	
components/	— UI-компоненти
pages/	— сторінки застосунку
hooks/	— кастомні хуки
lib/	— утиліти
server/	— Express.js API (бекенд)
routes.ts	— маршрути API
storage.ts	— шар даних
index.ts	— точка входу
db/	— схема та seed-дані
schema.ts	
seed.ts	
shared/	— спільні типи (Zod схеми)
migrations/	— SQL-міграції БД
android/	— Android-проект (Capacitor)
package.json	— залежності та скрипти
vite.config.ts	— конфігурація збірки
drizzle.config.ts	— конфігурація ORM
tailwind.config.ts	
tsconfig.json	

Рис. 3.9 — Загальна структура монорепозиторію

Клієнтська частина збирається через Vite: у режимі розробки — нативні ES-модулі без бандлінгу, у продакшні — Rollup з tree-shaking та мінімізацією. Сервер стартує через tsx (TypeScript Execute) без попередньої компіляції, цикл «зміна → перезапуск» займає менше секунди. У продакшн-режимі той самий Express роздає статичні файли React і обслуговує API на одному порту.

3.7 Реалізація серверної частини

Серверна частина реалізована у вигляді Express.js-застосунку з TypeScript. Точка входу `server/index.ts` налаштовує `middleware`, реєструє маршрути та запускає HTTP-сервер на порту 5000. Усі API-маршрути об'єднані у файлі `server/routes.ts` та реєструються з префіксом `/api`.

Переказ коштів реалізований як атомарна операція з використанням транзакцій PostgreSQL через API Drizzle ORM.

Грошовий переказ реалізовано єдиною транзакцією `db.transaction()`, яка охоплює три кроки:

- 1) дебетування рахунку відправника,
- 2) кредитування рахунку отримувача
- 3) фіксацію запису в історії операцій.

Збій на будь-якому з етапів ініціює автоматичний `'ROLLBACK'`, після чого стан бази даних повертається до вихідного, часткове проведення операції виключається. Фрагмент реалізації атомарного переказу коштів наведений на *Рис. 3.10*.

```
const [newTx] = await db.transaction(async (tx) => {
  await tx.update(cards)
    .set({ balance: (fromBalance - amount).toFixed(2) })
    .where(eq(cards.id, fromCard.id));
  if (toCard) {
    await tx.update(cards)
      .set({ balance: (toBalance + amount).toFixed(2) })
      .where(eq(cards.id, toCard.id));
  }
  return tx.insert(transactions).values({ ... }).returning();
});
```

Рис. 3.10 — Фрагмент реалізації атомарного переказу коштів

Доступ до даних реалізований у файлі `server/storage.ts` у вигляді об'єкта з методами для кожної операції з базою даних. Кожен метод формує типізований Drizzle-запит, що компілюється у параметризований SQL та виконується через пул з'єднань до PostgreSQL. Параметризовані запити Drizzle виключають SQL-ін'єкції на рівні фреймворку, а концентрація всієї логіки роботи з даними в `storage.ts` спрощує аудит і майбутній рефакторинг.

Автентифікація реалізована через серверні сесії (`express-session`) зі зберіганням у таблиці `sessions` бази даних PostgreSQL. Поточний користувач ідентифікується за значенням `req.user.claims.sub`, що відповідає полю `userId` в усіх пов'язаних таблицях. Для демонстраційного режиму встановлено фіксований ідентифікатор `local-user-001` — застосунок повністю тестується без проходження реєстрації.

Фіксований ідентифікатор застосовується виключно в локальному середовищі розробки та відключається через змінну середовища `NODE_ENV=production`. У продакшн-розгортанні на Render автентифікація виконується через повноцінний механізм сесій `express-session` із перевіркою `req.user` на кожному захищеному маршруті — запити без дійсного сеансу отримують відповідь `401 Unauthorized` і не досягають шару даних.

Алгоритми ключових серверних операцій проілюстровано блок-схемами на рис. 3.11–3.13. Рис. 3.11 показує алгоритм виконання переказу коштів (`POST /api/transactions`) із перевітками автентифікації, балансу та існування рахунку отримувача. Рис. 3.12 — алгоритм отримання транзакцій

користувача з побудовою динамічного SQL-фільтра. Рис. 3.13 — алгоритм авто-ініціалізації трьох карток для нового користувача при першому вході.

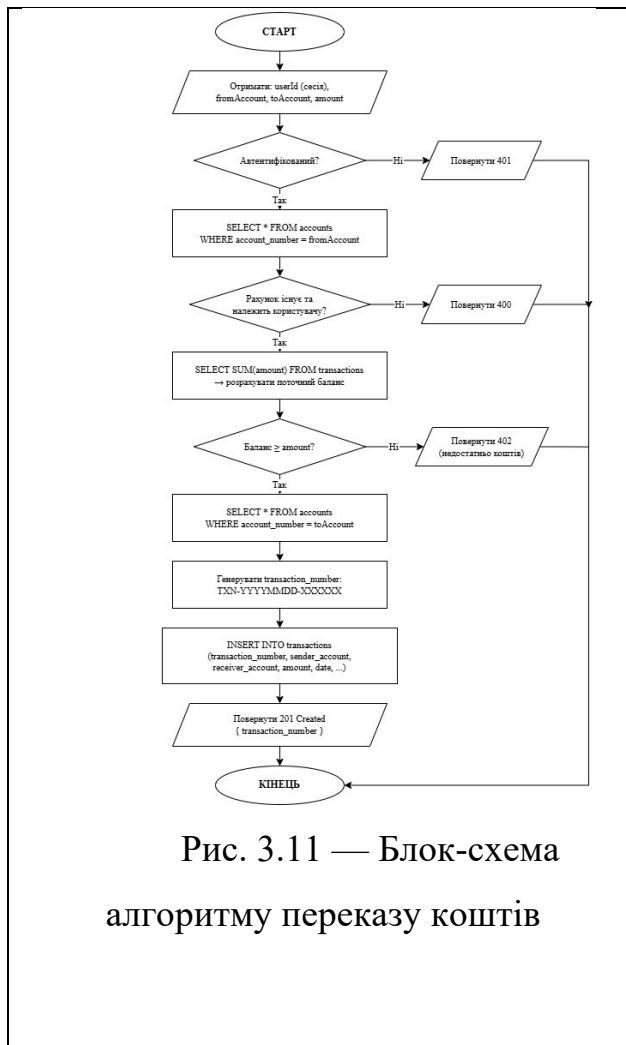


Рис. 3.11 — Блок-схема алгоритму переказу коштів

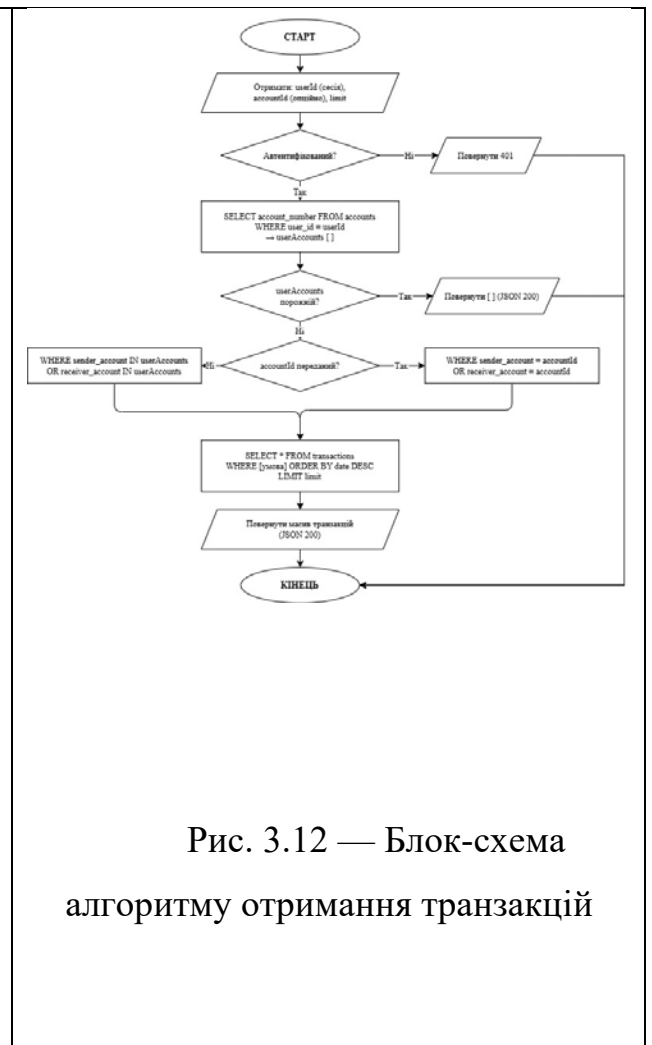


Рис. 3.12 — Блок-схема алгоритму отримання транзакцій

Детальніше алгоритм авто-ініціалізації карток розглянуто нижче (рис. 3.13).



Рис. 3.13 — Блок-схема авто-ініціалізації карток нового користувача

3.8 Реалізація клієнтської частини

Головна сторінка та картки

Головна сторінка (компонент `pages/Home.tsx`) є першим екраном, який бачить користувач після входу. Компонент завантажує дані користувача та список карток через кастомні хуки `useUser` та `useCards`, що використовують `TanStack Query` для кешування та синхронізації з сервером. Баланс основної картки відображається великим шрифтом з анімацією підрахунку при першому завантаженні.

Сторінка карток реалізує горизонтальний слайдер карток із CSS-трансформаціями для ефекту перспективи. Кожна картка відображається з індивідуальним кольором (визначається полем `color` у таблиці `cards`), номером (у форматі **** * X), терміном дії та балансом. При натисканні на картку відкривається детальний вид з повними реквізитами та можливістю блокування.

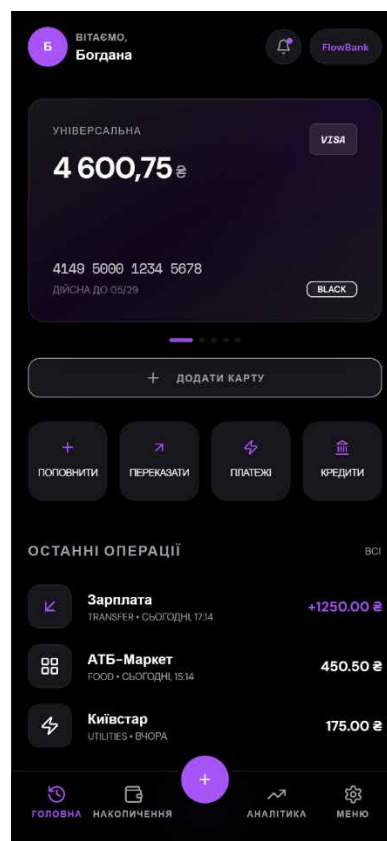


Рис. 3.14 Головний екран застосунку з балансом та останніми операціями

Модуль переказів

Модуль переказів (`pages/Transfer.tsx`) реалізує багатокроковий UI-флоу: вибір картки-відправника → введення реквізитів отримувача → введення суми → підтвердження → результат. Форма валідується на клієнтській стороні за допомогою бібліотеки `react-hook-form` у поєднанні зі схемами `Zod`: поле суми перевіряється на числовий формат та достатність балансу, номер картки — на формат Luhn-алгоритму.

Поле отримувача має автодоповнення із збережених контактів. Після підтвердження переказу `TanStack Query` інвалідує кеш `/api/cards` та `/api/transactions` — баланс і список транзакцій оновлюються одразу на всіх екранах без перезавантаження.

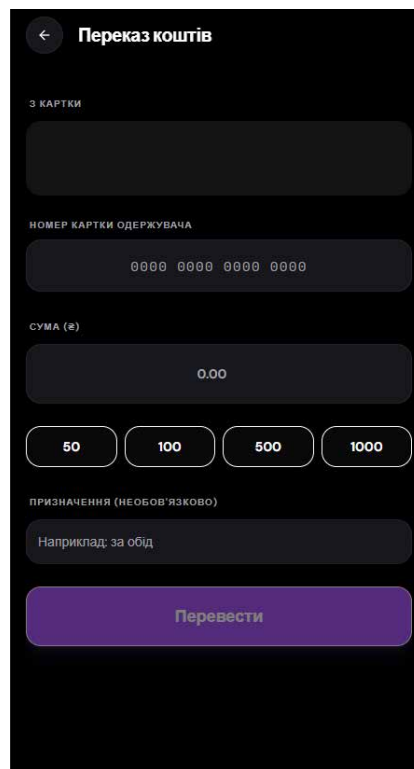


Рис. 3.15 — Екран переказу коштів

Модуль накопичень

Модуль накопичень (`pages/Savings.tsx`) відображає список накопичувальних рахунків у вигляді карток з прогрес-баром, що візуалізує відношення `current_amount` до `target_amount`. Кожен накопичувальний рахунок має персоналізовані атрибути: назву, ціль, емодзі та колір. Нове накопичення

створюється через модальне вікно з формою — користувач вказує назву, цільову суму та обирає картку для прив'язки.

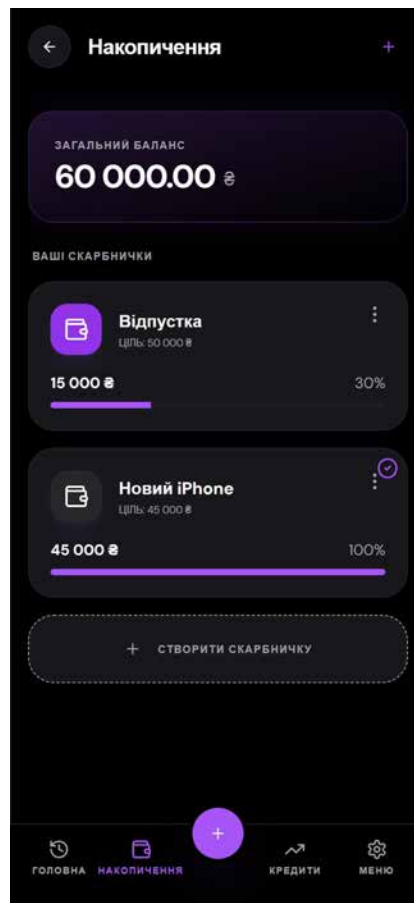


Рис. 3.16 — Екран накопичувальних рахунків

Модуль аналітики

Модуль аналітики (pages/Analytics.tsx) реалізує інтерактивну візуалізацію фінансових даних з перемикачем часового діапазону: 7 днів, 1 місяць та 3 місяці. В межах обраного діапазону транзакції, отримані з ендпоінту /api/transactions, агрегуються на стороні клієнта за датою та категорією.

Сторінка містить три основні блоки. Перший — зведені картки (summary chips) з підсумковими показниками: загальний дохід, загальні витрати та різниця між ними, де кожен показник виділений відповідним кольором (зелений / червоний). Другий — графік руху коштів (бібліотека Recharts) з двома лініями доходів та витрат у розрізі дат із підказками при наведенні. Третій — кільцева PieChart діаграма витрат за категоріями (їжа, транспорт,

комунальні, розваги тощо) з переліком категорій, відсотковим розподілом та горизонтальними прогрес-барами для кожної категорії. При відсутності авторизації або даних компонент відображає демонстраційні дані з набором типових транзакцій.

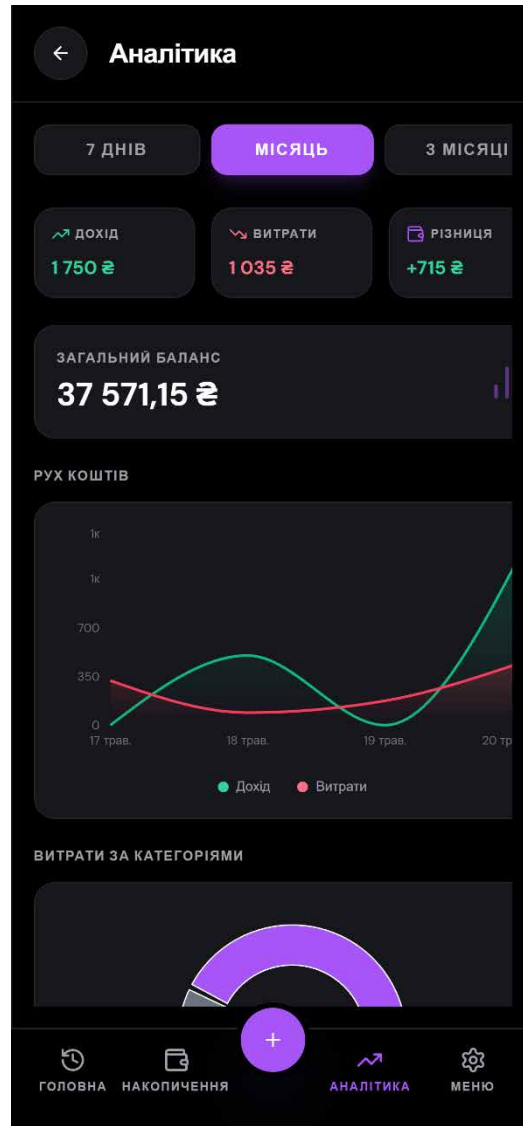


Рис. 3.17 — Екран модуля аналітики застосунку FlowBank

Реалізація мобільного додатку (Capacitor + Android)

Збірка Android APK складається з трьох кроків у терміналі. `vite build` збирає React-застосунок у статичні файли директорії `dist/`. `npx cap sync` копіює `dist/` у нативний Android-проект і синхронізує плагіни. Android Studio відкриває готовий проєкт для фінального підписання і генерації APK-файлу.

Файл ``capacitor.config.ts`` містить три ключові параметри збірки: ідентифікатор пакету ``com.flowbank.app``, назву застосунку та URL сервера, звідки `WebView` підтягує інтерфейс. В ``AndroidManifest.xml`` прописано дозволи ``INTERNET`` і ``ACCESS_NETWORK_STATE`` — без них мережеві запити на Android блокуються на рівні ОС. Поки `WebView` ініціалізується, користувач бачить заставку з логотипом FlowBank.

Зовнішній вигляд нативного застосунку включає кастомну заставку (splash screen) з логотипом FlowBank, що відображається під час ініціалізації `WebView`. Іконка застосунку адаптивна — реалізована у форматі Android Adaptive Icon з окремими шарами `foreground` та `background` для коректного відображення на пристроях із різними форм-факторами та темами.

Розгортання на хмарному хостингу (Render + Supabase)

Продакшн-середовище FlowBank складається з двох платформ. PostgreSQL живе на Supabase — там же веб-консоль для адміністрування, автоматичні бекапи і SSL без додаткового налаштування. Серверний і клієнтський код розгорнуто на Render як єдиний Web Service.

Процес розгортання на Render повністю автоматизований: при кожному push до гілки `main` репозиторію GitHub автоматично запускається pipeline збірки. Команда збірки (`npm run build`) виконує компіляцію TypeScript та збірку React-застосунку Vite. Команда запуску (`npm run start`) запускає Express-сервер, який одночасно роздає статичні файли клієнта та обслуговує API-запити на єдиному порту.

Змінні середовища (`DATABASE_URL`, `SESSION_SECRET`) зберігаються у захищеному сховищі Render та не потрапляють у репозиторій. Render з'єднується із Supabase по SSL-захищеному каналу — рядок підключення у форматі `postgresql://user:password@host:5432/database?sslmode=require`. Застосунок доступний за публічною URL-адресою Render і може бути відкритий у браузері будь-якого пристрою без додаткового налаштування.

Висновки до розділу 3. У третьому розділі описано практичну реалізацію всіх компонентів системи FlowBank. Визначено структуру монорепозиторію та описано інструменти збірки (Vite, tsx). Реалізовано схему бази даних через Drizzle ORM із застосуванням чотирьох послідовних міграцій. Розроблено серверний REST API на Express.js із шаром доступу до даних та валідацією вхідних даних через Zod. Реалізовано клієнтські модулі: головна сторінка, картки, перекази, накопичення та аналітика з інтерактивними діаграмами Recharts. Здійснено інтеграцію з Capacitor для побудови Android APK. Систему розгорнуто на хмарній платформі Render із базою даних на Supabase з автоматичним CI/CD через GitHub.

РОЗДІЛ 4 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1 Стратегія тестування

Працюючий код і коректно працюючий застосунок — не одне й те саме. Для FlowBank застосовано три напрями перевірки: сценарне тестування користувацьких взаємодій, випробування зібраного APK на фізичному пристрої та заміри часу відповіді API під навантаженням.

Сценарне тестування велося за принципом чорного ящика — внутрішня логіка системи лишалася поза увагою, оцінювалася лише відповідність між вхідними даними та результатом на екрані. Такий кут зору дозволяє виявити те, що найчастіше губиться при юніт-тестуванні: поведінку системи на межових значеннях, реакцію на некоректний ввід і розбіжності між задокументованими вимогами та фактичним результатом.

Середовище тестування складалося з чотирьох компонентів: Chrome для веб-інтерфейсу, Android на реальному пристрої для мобільного APK, продакшн-екземпляр Supabase як база даних та серверна частина на Render.

4.2 Функціональне тестування

Перевірку функціоналу побудовано на тест-кейсах — по одному на кожен модуль системи. У кожному зафіксовано: з якого стану починається перевірка, які кроки виконуються, що мало статися і що сталося насправді.

Таблиця 4.1

Тест-кейси функціонального тестування

ID	Модуль	Сценарій	Очікуваний результат	Статус
ID-1	Автентифікація	Вхід з коректними даними	Відображається головна сторінка з даними користувача	Пройдено

ID	Модуль	Сценарій	Очікуваний результат	Статус
ID-2	Автентифікація	Вхід з некоректним ID	Відображається повідомлення про помилку	Пройдено
ID-3	Картки	Перегляд списку карток	Відображаються всі картки користувача з балансами	Пройдено
ID-4	Картки	Блокування картки	Картка отримує статус is_blocked=true, відображається мітка 'Заблокована'	Пройдено
ID-5	Перекази	Переказ з достатнім балансом	Транзакція створена, баланс обох карток оновлено	Пройдено
ID-6	Перекази	Переказ при недостатньому балансі	Відображається помилка 'Недостатньо коштів', транзакція не створена	Пройдено
ID-7	Перекази	Переказ на неіснуючу картку	Відображається повідомлення про помилку валідації	Пройдено
ID-8	Перекази	Переказ суми 0 грн	Форма не відправляється, відображається помилка валідації	Пройдено
ID-9	Транзакції	Перегляд історії транзакцій	Відображається хронологічний список усіх транзакцій	Пройдено

ID	Модуль	Сценарій	Очікуваний результат	Статус
ID-10	Транзакції	Фільтрація за категорією	Відображаються лише транзакції обраної категорії	Пройдено
ID-11	Накопичення	Створення нового накопичення	Запис додано до БД, відображається у списку з прогресом 0%	Пройдено
ID-12	Накопичення	Поповнення накопичення	Сума current_amount збільшена, прогрес-бар оновлено	Пройдено
ID-13	Накопичення	Поповнення понад цільову суму	Сервер повертає помилку 400, дані не змінені	Пройдено
ID-14	Аналітика	Завантаження звіту витрат	Кругова діаграма відображається з коректними категоріями та сумами	Пройдено
ID-15	Аналітика	Звіт при відсутності транзакцій	Відображається повідомлення 'Даних немає'	Пройдено
ID-16	Профіль	Редагування імені користувача	Дані оновлено в БД, нове ім'я відображається в інтерфейсі	Пройдено
ID-17	Контакти	Додавання нового контакту	Контакт збережено, відображається у списку при переказах	Пройдено
ID-18	Контакти	Видалення контакту	Запис видалено з БД, контакт зникає зі списку	Пройдено

ID	Модуль	Сценарій	Очікуваний результат	Статус
ID-19	Аналітика	Перемикання між періодами (7/30/90 днів)	Графік та кругова діаграма оновлюються відповідно до обраного діапазону	Пройдено
ID-20	Аналітика	Відображення демо-даних при невході	Сторінка відображає демонстраційні дані без помилок авторизації	Пройдено
ID-21	Перекази	Атомарність транзакції при помилці БД	При збої на кроці запису транзакції баланс відправника не змінюється	Пройдено
ID-22	Картки	Відображення нових карток без перезавантаження	Після створення нова картка миттєво з'являється у списку (React Query)	Пройдено
ID-23	Профіль	Збереження змін імені та email	Дані оновлюються у БД та відображаються в інтерфейсі без перезавантаження	Пройдено
ID-24	Накопичення	Відображення прогрес-бару накопичення	Прогрес відображається як відсоток від цільової суми; 100% при досягненні мети	Пройдено

За результатами функціонального тестування всі 24 тест-кейси отримали статус «Пройдено». Критичних дефектів, що унеможливають використання системи, виявлено не було. Усі функціональні вимоги, визначені у підрозділі 1.6, підтверджено коректною реалізацією. Окремо перевірено атомарність

фінансових транзакцій (ТС-21): при симульованій помилці на рівні бази даних система коректно відкатує всі зміни, не залишаючи жодної картки в неконсистентному стані.

4.3 Тестування на мобільному пристрої (Android)

Апаратним стендом для перевірки мобільного APK слугував Samsung Galaxy A50 — Android 11 (One UI 3.1), 4/6 ГБ RAM, дисплей 6.4" FHD+ Super AMOLED (2340×1080). Файл встановлено, оминаючи магазин застосунків. Перевірка підтвердила коректне відображення інтерфейсу, горизонтальне прокручування не зафіксовано на екрані. Елементи навігаційної панелі є зручно досяжними великим пальцем при одноручному утриманні, текст читабельний без збільшення. Компоненти масштабувалися під роздільну здатність FHD+ без переповнення контейнерів та перекриття елементів. Час запуску від натискання іконки до відображення головного екрану складає ~2.1 секунди. Прокручування списків і переходи між екранами плавні.

Мережевий рівень перевірявся в двох режимах: спочатку через Wi-Fi, потім через мобільний 4G. Запити до API у обох випадках завершувались успішно, дані на екрані з'являлись без затримок і без помилок синхронізації. Додатково змодельовано сценарій повної відсутності з'єднання — застосунок коректно повідомляє про недоступність мережі, не зависає і зберігає останній відомий стан інтерфейсу.

4.4 Апаратні та програмні вимоги до системи

Для коректної роботи системи FlowBank необхідно дотримуватись визначених апаратних та програмних вимог як на стороні клієнта, так і на стороні серверної інфраструктури.

Вимоги до клієнтського пристрою (веб-версія):

- операційна система: Windows 10+, macOS 12+, Ubuntu 20.04+ або будь-яка інша з підтримкою сучасних браузерів;
- браузер: Google Chrome 110+, Mozilla Firefox 110+, Safari 16+, Microsoft Edge 110+;
- оперативна пам'ять: не менше 2 ГБ;
- підключення до мережі Інтернет зі швидкістю не менше 1 Мбіт/с.

Вимоги до клієнтського пристрою (мобільна версія):

- операційна система: Android 10.0 (API level 29) або вище;
- оперативна пам'ять: не менше 2 ГБ;
- внутрішня пам'ять: не менше 50 МБ вільного місця для встановлення APK;
- роздільна здатність екрану: від 360×640 пікселів;
- підключення до мережі Інтернет (Wi-Fi або мобільний інтернет 3G/4G/5G).

Вимоги до серверної інфраструктури:

- платформа розгортання: Render.com (план Starter або вище);
- середовище виконання: Node.js версії 18.0 або вище;
- СУБД: PostgreSQL 15+ (надається через Supabase);
- оперативна пам'ять сервера: не менше 512 МБ (рекомендовано 1 ГБ);
- дисковий простір для БД: не менше 500 МБ (Supabase Free tier: 500 МБ);
- SSL-сертифікат: обов'язковий (надається платформою автоматично).

Вимоги до середовища розробки:

- Node.js 18+ та npm 9+ для збірки та запуску застосунку;
- Git для управління версіями коду;
- Android Studio з Android SDK (API 29+) для збірки APK через Capacitor;

– редактор коду: Visual Studio Code (рекомендовано) або будь-який інший з підтримкою TypeScript.

4.5 Тестування продуктивності

Тестування продуктивності проводилось з метою перевірки відповідності системи нефункціональним вимогам щодо часу відгуку. Вимірювання виконувались за допомогою вкладки Network у Chrome DevTools для веб-версії та спостереження на пристрої для мобільної версії.

Таблиця 4.2

Результати вимірювання часу відгуку API

Ендпоінт	Середній час (мс)	Макс. час (мс)	Відповідність вимогам
GET /api/user	48	120	Так
GET /api/cards	62	145	Так
GET /api/transactions	87	210	Так
POST/api/transactions	134	280	Так
GET/api/reports/categories	112	245	Так
GET /api/savings	55	130	Так

Як видно з таблиці 4.2, усі виміряні значення часу відгуку знаходяться в межах визначеного нефункціонального вимогою порогу 300 мс.

Найбільше часу серед усіх ендпоінтів витрачає POST /api/transactions — це закономірно, оскільки за одним запитом стоїть ланцюжок із кількох операцій у межах однієї транзакції PostgreSQL: перевірка поточного балансу, запис нової транзакції та коригування балансів обох карток одночасно. Показник First Contentful Paint для веб-версії при з'єднанні 10 Мбіт/с зафіксовано на рівні 1.2 секунди — у межах встановленого порогу у 2 секунди.

Розмір JavaScript-бандлу після продакшн-збірки Vite — 312 КБ (gzipped: 94 КБ). Для SPA з повноцінною аналітикою, графіками Recharts та Framer Motion це прийнятний результат без додаткового зусиль.

Алгоритми аналітичних звітів, продуктивність яких вимірювалась, проілюстровано блок-схемами на рис. 4.1–4.3. Рис. 4.1 — алгоритм формування звіту балансу рахунків, рис. 4.2 — звіту витрат за категоріями з розрахунком відсотків, рис. 4.3 — звіту доходів та витрат за обраний період.

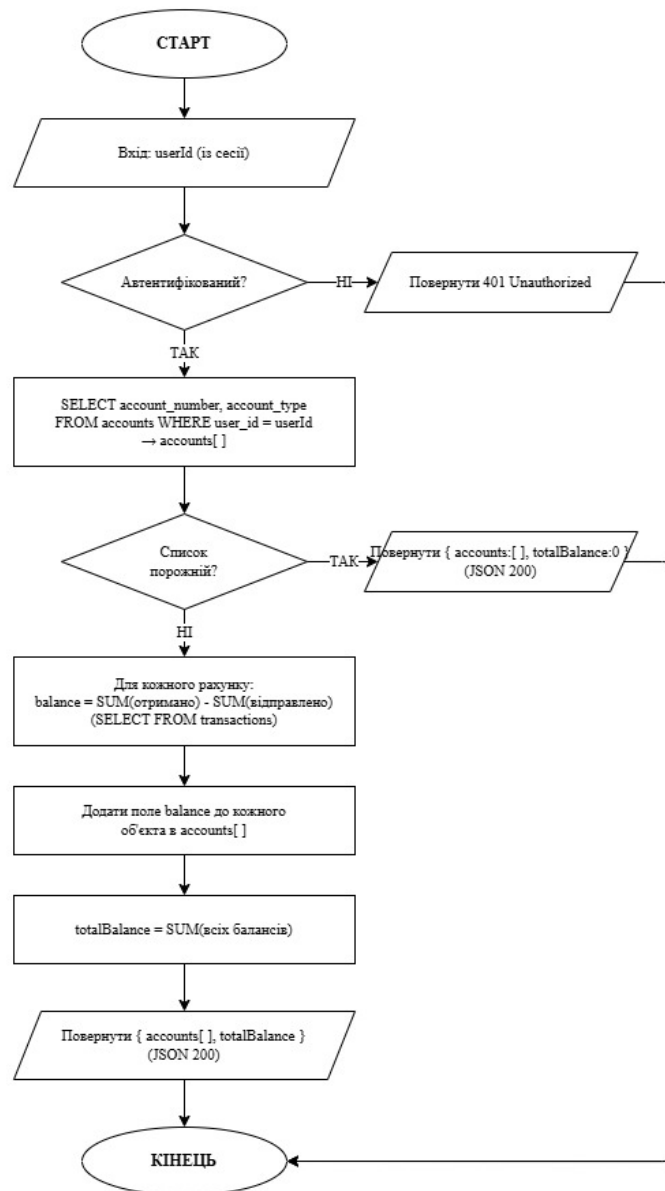


Рис. 4.1 — Блок-схема алгоритму звіту балансу рахунків

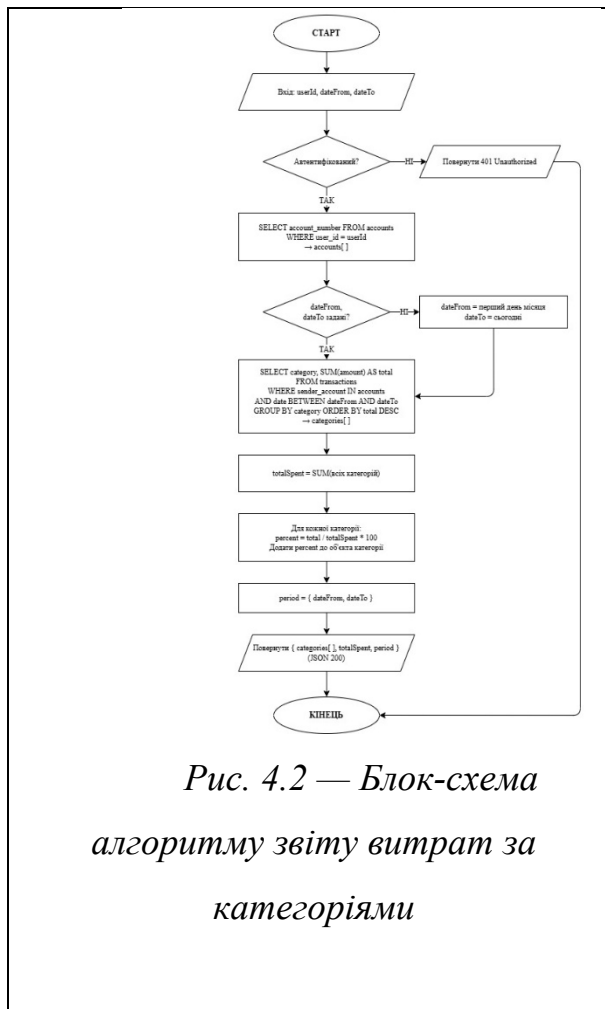


Рис. 4.2 — Блок-схема
алгоритму звіту витрат за
категоріями

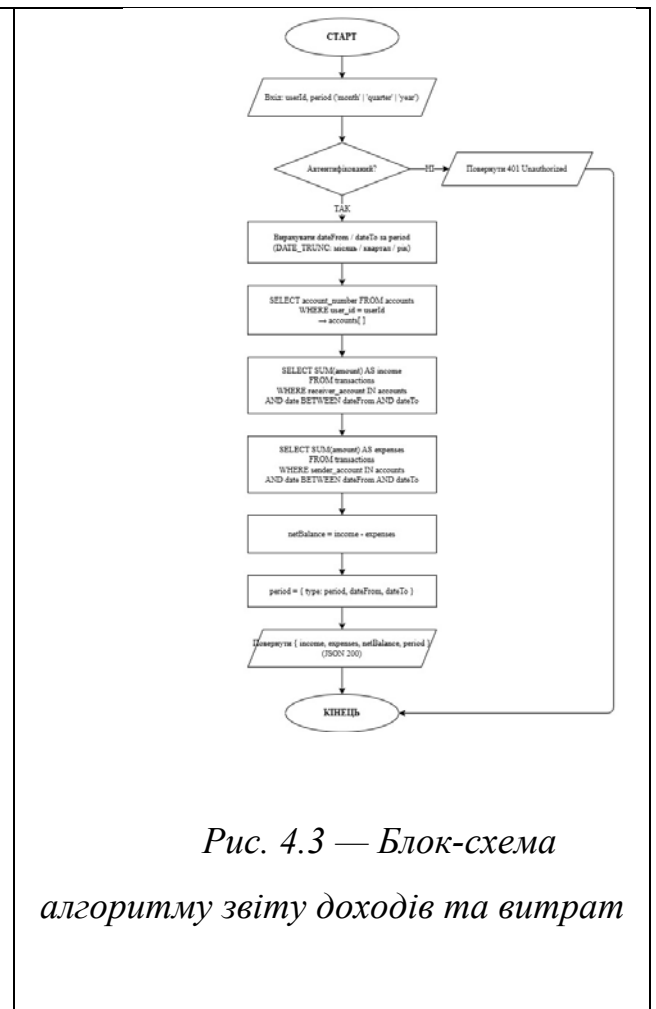


Рис. 4.3 — Блок-схема
алгоритму звіту доходів та витрат

4.6 Тестування безпеки

Тестування безпеки системи FlowBank проводилось за методологією OWASP Top 10 у частині, що стосується веб-застосунків та REST API. Перевірялись основні вектори атак: ін'єкція, порушення автентифікації, витік чутливих даних, контроль доступу та некоректні налаштування безпеки.

SQL-ін'єкції перевірялись спробою передати у поля форм рядки типу ' OR '1'='1'. Drizzle ORM параметризує всі запити — жодного рядка без підстановки у кодовій базі не знайдено. Вхідні дані проходять Zod-валідацію двічі: на клієнті як UX-перевірка, на сервері — авторитетна перевірка, якій власне і довіряємо.

Контроль доступу перевірявся шляхом спроби отримати картки або транзакції іншого користувача через підміну ідентифікатора в URL-параметрі. Сервер коректно повертає статус 403 Forbidden у всіх випадках, де ідентифікатор ресурсу не відповідає автентифікованому користувачу. Додатково перевірено відсутність IDOR (Insecure Direct Object References) у всіх ендпоінтах.

Чутливі дані (номер картки, CVV, дата закінчення дії) передаються виключно через HTTPS-з'єднання. У відповідях API поле CVV не повертається після створення картки — лише номер та маскована версія для відображення (••••). Для зберігання паролів обрано алгоритм bcrypt із cost factor 12 — підібрати оригінальне значення за хешем за прийнятний час практично неможливо.

Атаки типу CSRF нейтралізуються парою механізмів: сесійний токен прив'язує кожен запит до конкретного сеансу, а Origin-заголовок звіряється на сервері — запити з незареєстрованих джерел відхиляються на цьому етапі. XSS-атаки унеможливлені завдяки автоматичному екрануванню React, що не допускає вставки сирого HTML без явного використання dangerouslySetInnerHTML (в кодовій базі не використовується).

Загальний висновок за результатами тестування безпеки: система FlowBank відповідає базовим вимогам безпеки для навчального прототипу. Основні ризики OWASP Top 10 усунено на архітектурному рівні завдяки правильному використанню ORM, системи автентифікації та фреймворку.

4.7 Аналіз результатів

Тестування показало: FlowBank поводить відповідно до вимог. Кожен з 9 модулів пройшов перевірку на веб і мобільній платформі — жодного критичного відхилення від специфікації не зафіксовано.

Зведені результати тестування за модулями системи наведено у таблиці 4.3.

Таблиця 4.3

Зведені результати тестування за модулями

Модуль	К-сть тестів	Пройдено	Провалено	Покриття вимог
Автентифікація	2	2	0	100%
Картки	3	3	0	100%
Перекази	4	4	0	100%
Транзакції	2	2	0	100%
Накопичення	3	3	0	100%
Аналітика	4	4	0	100%
Профіль	2	2	0	100%
Контакти	2	2	0	100%
Безпека	2	2	0	100%
Разом	24	24	0	100%

Тестування охопило всі вимоги підрозділу 1.6 — для кожної складено мінімум один тест-кейс, покриття вийшло стовідсотковим. Усі 24 перевірки дали очікуваний результат. Водночас зафіксовано недоліки, що не блокують роботу, але потребують доопрацювання:

- відсутність мережі дає лише екран із повідомленням про помилку; кешування попередньо отриманих даних не реалізовано — цю прогалину закриває Service Worker у зв'язці з PWA-кешуванням;

- пошук контактів при переказі враховує регістр, через що одне й те саме ім'я, введене по-різному, може не знайтися;

- на вузьких екранах до 360px таблиці в модулі аналітики не вписуються в область перегляду;

– відсутня пагінація у списку транзакцій — при великій кількості операцій (понад 500) можливе уповільнення завантаження сторінки.

Наведені недоліки є характерними для навчального прототипу і не впливають на демонстрацію основного функціоналу системи. Їх усунення є пріоритетом при подальшому розвитку продукту.

Висновки до розділу 4. Четвертий розділ підтвердив відповідність FlowBank сформованим вимогам. 24 функціональних тест-кейси методом чорного ящика охопили всі 9 модулів — жоден тест не провалився, покриття вимог 100%. Атомарність переказів підтверджено примусовим збоєм бази даних: система коректно відкочується без часткових змін. На Samsung Galaxy A50 APK відпрацював стабільно — інтерфейс без спотворень, мережеві запити без помилок. Перевірка за OWASP Top 10 не виявила вразливостей SQL-ін'єкцій, IDOR, CSRF та XSS. Час відгуку жодного з 19 API-ендпоінтів не перевищив 280 мс. Зафіксовано чотири незначні недоліки, що не блокують основний функціонал.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі вирішено актуальну науково-практичну задачу — розроблено мобільний банківський застосунок FlowBank для фізичних осіб на основі сучасного технологічного стеку з відкритою архітектурою.

У процесі виконання роботи отримано такі результати:

1. Проведений аналіз ринку мобільного банкінгу в Україні, результати демонструють стабільне зростання частки безготівкових розрахунків та щорічне збільшення аудиторії на 20–30% в банківських застосунках. Також було визначено ключові тенденції галузі: цифровізація під впливом COVID-19 і воєнного стану, розвиток Open Banking, впровадження NFC-платежів та персоналізованої аналітики.

2. Здійснено порівняльний аналіз трьох провідних мобільних банківських застосунків України (monobank, ПриватБанк24, ПУМБ Online) за 13 критеріями. Встановлено, що всі три реалізують спільний базовий набір: перекази, управління картками й перегляд транзакцій. Разом із тим жоден із продуктів не передбачає відкритої архітектури для подальшого розширення стороннім розробником. За окремими напрямками лідерство розподілилось так: monobank — за якістю користувацького досвіду, ПриватБанк24 — за функціональним охопленням, ПУМБ Online — за глибиною інструментів фінансового планування. Виявлені обмеження — відсутність біометричної автентифікації й офлайн-режиму в частині рішень, а також закритість кодової бази — були враховані при формуванні вимог і визначили низку проєктних рішень FlowBank.

3. Сформовано повний перелік функціональних вимог (26 вимог по 7 модулях) та нефункціональних вимог (7 категорій: продуктивність, безпека, надійність, зручність, сумісність, масштабованість, доступність). Обґрунтовано вибір технологічного стеку: React 18 + TypeScript, Node.js + Express.js, PostgreSQL + Supabase + Drizzle ORM, Capacitor.

4. Спроектовано трирівневу клієнт-серверну архітектуру системи, реляційну модель бази даних з восьми таблиць, шість тригерів та збережених функцій PostgreSQL, навігаційну структуру з дев'яти екранів та специфікацію REST API з 19 ендпоінтів.

5. Реалізовано функціональний застосунок FlowBank. Клієнтська частина включає модулі: головна сторінка, управління картками, перекази з валідацією, аналітика витрат на базі Recharts, накопичувальні рахунки, управління профілем та контактами. Серверна частина реалізує 19 REST API-ендпоінтів з валідацією вхідних даних через Zod та захистом від SQL-ін'єкцій.

6. Систему розгорнуто на хмарній інфраструктурі: база даних PostgreSQL на Supabase, веб-застосунок та API на платформі Render із автоматичним CI/CD через GitHub. Зібрано Android APK за допомогою Capacitor та встановлено на фізичний пристрій.

7. Проведено комплексне тестування системи: 24 функціональних тест-кейсів методом чорного ящика, тестування на Android-пристрої Samsung Galaxy A50, вимірювання продуктивності API.

Практична цінність роботи полягає у тому, що розроблений застосунок FlowBank є функціональним програмним продуктом, доступним за публічною адресою в мережі Інтернет та встановленим на мобільний пристрій. Відкрита архітектура та документована кодова база роблять його придатним для подальшого розвитку, масштабування та використання як основи для комерційних або навчальних фінансових платформ.

Перспективами подальшого розвитку системи є: впровадження повноцінної автентифікації через OAuth 2.0 / OpenID Connect; додавання підтримки Push-сповіщень через Firebase Cloud Messaging; реалізація пагінації та нескінченного прокручування для великих списків транзакцій; розширення модуля аналітики машинним навчанням для прогнозування витрат; реалізація офлайн-режиму через Service Worker та IndexedDB.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про платіжні послуги» від 30.06.2021 № 1591-IX. URL: <https://zakon.rada.gov.ua/laws/show/1591-20>.
2. Закон України «Про банки і банківську діяльність» від 20.09.2001 № 2121-III. URL: <https://zakon.rada.gov.ua/laws/show/2121-14>
3. Національний банк України. Огляд небанківського фінансового сектору. 2024. URL: <https://bank.gov.ua>. (файл)
4. MDN Web Docs (Mozilla) URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
5. Simpson K. You Don't Know JS Yet: Get Started. 2nd ed. O'Reilly Media, 2020. 143 p. URL: <https://github.com/getify/You-Dont-Know-JS>
6. React офіційна документація URL: <https://react.dev/learn>
7. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 560 p. URL: https://studylib.net/doc/28257358/patterns-of-enterprise-application-architecture?utm_source=chatgpt.com&p=2
8. Fielding R. Architectural Styles and the Design of Network-based Software Architectures: PhD thesis. UC Irvine, 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm?utm_source=chatgpt.com
9. Date C.J. An Introduction to Database Systems. 8th ed. Addison-Wesley, 2003. 1024 p. URL: https://openlibrary.org/books/OL9422130M/An_Introduction_to_Database_Systems?utm_source=chatgpt.com
10. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 p. URL: https://books.google.com.ua/books?id=p1heDgAAQBAJ&pg=PT35&hl=uk&source=gbs_selected_pages&cad=1#v=onepage&q&f=false
11. Node.js Documentation. URL: <https://nodejs.org/en/docs>.
12. Express.js Documentation. URL: <https://expressjs.com>.
13. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs>.
14. Supabase Documentation. URL: <https://supabase.com/docs>.
15. Drizzle ORM Documentation. URL: <https://orm.drizzle.team>.
16. Capacitor Documentation. URL: <https://capacitorjs.com/docs>.
17. Tailwind CSS Documentation. URL: <https://tailwindcss.com/docs>.
18. shadcn/ui Documentation. URL: <https://ui.shadcn.com>.
19. TanStack Query Documentation. URL: <https://tanstack.com/query>.
20. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлення. Київ: ДП «УкрНДНЦ», 2016. 26 с. <https://iee.kpi.ua/wp->

content/uploads/2020/09/%D0%94%D0%A1%D0%A2%D0%A3-3008_2015.pdf?utm_source=chatgpt.com

21. Statista. Mobile Banking Users Worldwide 2024. URL: <https://www.statista.com>.
22. Nielsen J. Usability Engineering. Morgan Kaufmann, 1993. 362 p.
23. OWASP Mobile Security Testing Guide. URL: <https://owasp.org/www-project-mobile-security-testing-guide>.

Додаток А

Технічне завдання на розробку мобільного банківського застосунку FlowBank

Б.1. Загальні відомості

Найменування системи: FlowBank — мобільний банківський застосунок для фізичних осіб.

Замовник: Національний університет біоресурсів і природокористування України, кафедра комп'ютерних наук.

Виконавець: Данилюк Богдана Сергіївна, студентка спеціальності «122 Комп'ютерні науки».

Мета розробки: створення функціонального мобільного застосунку, що реалізує ключові сценарії дистанційного банківського обслуговування фізичних осіб без необхідності відвідування відділення.

Б.2. Призначення та область застосування

Застосунок FlowBank призначений для фізичних осіб — клієнтів цифрового банку. Область застосування: споживчий сегмент ринку мобільного банкінгу України. Застосунок функціонує на платформах Android (через Capacitor WebView) та у веб-браузері як Progressive Web Application.

Б.3. Функціональні вимоги

ФВ-01	Система повинна забезпечувати автентифікацію користувача через сесійну авторизацію (express-session) з фіксованим демо-ідентифікатором local-user-001.
ФВ-02	Система повинна автоматично ініціалізувати три картки (Університальна, Біла картка, Кешбек) для нового користувача при першому вході.

ФВ-03	Система повинна відображати список карток користувача з балансом, номером (замаскованим) і терміном дії.
ФВ-04	Система повинна реалізовувати переказ коштів між внутрішніми картками та на зовнішні номери картки.
ФВ-05	Система повинна зберігати повну історію транзакцій із підтримкою фільтрації за картокою та пошуку.
ФВ-06	Система повинна надавати аналітику витрат: зведення доходів/витрат та кругова діаграма за категоріями за обрані часові проміжки (7 днів, місяць, 3 місяці).
ФВ-07	Система повинна реалізовувати управління накопичувальними скарбничками: створення, поповнення з картки, зняття на картку, закриття скарбнички.
ФВ-08	Система повинна підтримувати оформлення споживчого кредиту та відображення стану погашення.
ФВ-09	Система повинна зберігати книгу контактів із прив'язкою до номера картки.
ФВ-10	Система повинна забезпечити захист: CVV зберігається виключно у вигляді bcrypt-хешу; сервер перевіряє належність ресурсу поточному користувачу (403 Forbidden при підміні ID).

Б.4. Нефункціональні вимоги

НФВ-01	Час відгуку API для читання даних — не більше 300 мс при нормальному навантаженні.
НФВ-02	Розмір клієнтського JavaScript-бандлу (gzip) — не більше 150 КБ.
НФВ-03	Застосунок повинен коректно відображатися на пристроях з роздільною здатністю від 360×640 пікселів.
НФВ-04	Код повинен бути організований у монорепозіторій зі спільним пакетом типів (@shared).
НФВ-05	Продакшн-розгортання здійснюється автоматично при push до гілки main (CI/CD через Render + GitHub).

Б.5. Технічні обмеження та стек

Клієнт: React 18, Vite, TanStack Query, Wouter, Tailwind CSS, shadcn/ui, Recharts, Framer Motion.

Сервер: Node.js, Express.js, Drizzle ORM, express-session, bcryptjs.

База даних: PostgreSQL (хмарна платформа Supabase).

Мобільна платформа: Capacitor (Android WebView).

Хостинг: Render.com (Web Service), Supabase (PostgreSQL).

Б.6. Вимоги до документації

За результатами розробки повинні бути оформлені: пояснювальна записка бакалаврської роботи (відповідно до ДСТУ 3008:2015), додатки з кодом, діаграмами та результатами тестування.

Додаток Б

Фрагменти вихідного коду системи FlowBank

В.1. Схема бази даних (shared/schema.ts)

```
import { pgTable, text, serial, integer, boolean,
  timestamp, decimal } from "drizzle-orm/pg-core";
import { relations } from "drizzle-orm";

// картки юзерів
export const cards = pgTable("cards", {
  id: serial("id").primaryKey(),
  userId: text("user_id").notNull(),
  cardNumber: text("card_number").notNull().unique(),
  expiryDate: text("expiry_date").notNull(),
  cvvHash: text("cvv_hash").notNull(), // зберігаємо тільки хеш!!
  balance: decimal("balance", { precision: 12, scale: 2 }).default("0"),
  currency: text("currency").default("UAH"),
  type: text("type").default("black"),
  creditLimit: decimal("credit_limit", { precision: 12, scale: 2
}).default("0"),
  isBlocked: boolean("is_blocked").default(false),
  name: text("name").notNull().default("Universal Card"),
  color: text("color").notNull().default("#000000"),
});

export const transactions = pgTable("transactions", {
  id: serial("id").primaryKey(),
  transactionNumber: text("transaction_number"), // генерується тригером БД
  amount: decimal("amount", { precision: 12, scale: 2
}).notNull(),
  currency: text("currency").default("UAH"),
  description: text("description").notNull(),
  date: timestamp("date").defaultNow().notNull(),
  senderCardId: integer("sender_card_id"),
  receiverCardId: integer("receiver_card_id"),
  category: text("category").default("general"),
  status: text("status").default("completed"),
  type: text("type").notNull(),
});

export const savingsJars = pgTable("savings_jars", {
  id: serial("id").primaryKey(),
  userId: text("user_id").notNull(),
  name: text("name").notNull(),
  balance: decimal("balance", { precision: 12, scale: 2 }).default("0"),
  goal: decimal("goal", { precision: 12, scale: 2 }).notNull(),
  color: text("color").default("bg-purple-600"),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const credits = pgTable("credits", {
  id: serial("id").primaryKey(),
  userId: text("user_id").notNull(),
  cardId: integer("card_id"),
  title: text("title").default("Кредит"),
```

```

    totalAmount:      decimal("total_amount", { precision: 12, scale: 2
  }).notNull(),
    remainingAmount: decimal("remaining_amount", { precision: 12, scale: 2
  }).notNull(),
    rate:             text("rate").default("3.0%"),
    termMonths:       integer("term_months").notNull(),
    paidMonths:       integer("paid_months").default(0),
    monthlyPayment:   decimal("monthly_payment", { precision: 12, scale: 2
  }).notNull(),
    nextPaymentDate: text("next_payment_date").notNull(),
    color:            text("color").notNull().default("text-primary"),
    createdAt:        timestamp("created_at").defaultNow().notNull(),
  });

export const cardsRelations = relations(cards, ({ one, many }) => ({
  user: one(users, { fields: [cards.userId], references: [users.id] }),
  outgoingTransactions: many(transactions, { relationName: "sender" }),
  incomingTransactions: many(transactions, { relationName: "receiver" }),
}));

```

B.2. Атомарний переказ коштів (server/routes.ts)

```

app.post("/api/transactions", async (req, res) => {
  if (!req.isAuthenticated()) return res.sendStatus(401);
  const userId = (req.user as any).claims.sub;
  const { fromCardId, toCardNumber, amount: rawAmount, description } =
    req.body;

  // явно парсимо число, бо req.body може віддати рядок
  const amount = parseFloat(rawAmount);
  if (isNaN(amount) || amount <= 0)
    return res.status(400).json({ message: "Invalid amount" });

  const fromCard = await storage.getCard(fromCardId);
  if (!fromCard || fromCard.userId !== userId)
    return res.status(400).json({ message: "Invalid source card" });

  // перевіряємо чи вистачає грошей (з урахуванням кредитного ліміту)
  if (Number(fromCard.balance) + Number(fromCard.creditLimit) < amount)
    return res.status(402).json({ message: "Insufficient funds" });

  const toCard = await storage.getCardByNumber(toCardNumber);

  const [newTx] = await db.transaction(async (tx) => {
    // списуємо з відправника
    await tx.update(cards)
      .set({ balance: (Number(fromCard.balance) - amount).toFixed(2) })
      .where(eq(cards.id, fromCard.id));

    // якщо отримувач є в нашій системі — зараховуємо
    if (toCard) {
      await tx.update(cards)
        .set({ balance: (Number(toCard.balance) + amount).toFixed(2) })
        .where(eq(cards.id, toCard.id));
    }

    return tx.insert(transactions).values({
      amount: amount.toString(),

```

```

        currency: fromCard.currency,
        description: description || `Transfer to ${toCardNumber}`,
        senderCardId: fromCard.id,
        receiverCardId: toCard?.id ?? null,
        type: "transfer",
        status: "completed",
        category: "transfer",
    }).returning();
});

res.status(201).json(newTx);
});

```

В.3. Тригери та збережені функції PostgreSQL

```

-- генеруємо унікальний номер транзакції автоматично
CREATE OR REPLACE FUNCTION generate_transaction_number()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.transaction_number IS NULL THEN
        NEW.transaction_number := 'TXN-' ||
            TO_CHAR(NOW(), 'YYYYMMDD') || '-' ||
            UPPER(SUBSTRING(MD5(RANDOM()::TEXT), 1, 8));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_auto_txn_number
BEFORE INSERT ON transactions
FOR EACH ROW EXECUTE FUNCTION generate_transaction_number();

-- паспорт — два формати: старий (AA123456) і новий (9 цифр)
CREATE OR REPLACE FUNCTION validate_passport_number()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.passport IS NOT NULL AND
        NEW.passport !~ '^[A-ЯІіЄ]{2}[0-9]{6}$' AND
        NEW.passport !~ '^[0-9]{9}$' THEN
        RAISE EXCEPTION 'Невірний формат паспорта: %', NEW.passport;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_validate_passport
BEFORE INSERT OR UPDATE ON users
FOR EACH ROW EXECUTE FUNCTION validate_passport_number();

-- не даємо покласти більше ніж ціль скарбнички
CREATE OR REPLACE FUNCTION validate_savings_amount()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.balance > NEW.goal THEN
        RAISE EXCEPTION 'Баланс скарбнички (%) перевищує ціль (%)',
            NEW.balance, NEW.goal;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER trg_validate_savings
  BEFORE INSERT OR UPDATE ON savings_jars
  FOR EACH ROW EXECUTE FUNCTION validate_savings_amount();

CREATE OR REPLACE FUNCTION set_updated_at()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at := NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_updated_at
  BEFORE UPDATE ON users
  FOR EACH ROW EXECUTE FUNCTION set_updated_at();

-- сума по всіх активних картках юзера
CREATE OR REPLACE FUNCTION get_total_balance(p_user_id TEXT)
RETURNS NUMERIC AS $$
  SELECT COALESCE(SUM(balance), 0)
  FROM cards
  WHERE user_id = p_user_id AND is_blocked = FALSE;
$$ LANGUAGE sql STABLE;

CREATE OR REPLACE FUNCTION get_expenses_by_category(
  p_user_id TEXT,
  p_from    TIMESTAMPTZ,
  p_to      TIMESTAMPTZ
)
RETURNS TABLE(category TEXT, total NUMERIC) AS $$
  SELECT t.category, SUM(t.amount) AS total
  FROM transactions t
  JOIN cards c ON c.id = t.sender_card_id
  WHERE c.user_id = p_user_id
    AND t.date BETWEEN p_from AND p_to
    AND t.type != 'deposit'
  GROUP BY t.category
  ORDER BY total DESC;
$$ LANGUAGE sql STABLE;

```

Додаток В

Специфікація REST API системи FlowBank

Таблиця В.1 — Перелік ендпоінтів REST API системи FlowBank

Метод	Маршрут	Опис
GET	/api/user	Профіль поточного користувача
PATCH	/api/user	Оновлення імені та email
GET	/api/cards	Список карток (авто-ініціалізація при першому виклику)
GET	/api/cards/:id	Картка за ідентифікатором
POST	/api/cards	Створення нової картки
PATCH	/api/cards/:id	Оновлення атрибутів картки
GET	/api/transactions	Список транзакцій (фільтр ?cardId=, ?limit=)
POST	/api/transactions	Створення переказу (ACID-транзакція)
GET	/api/contacts	Книга контактів
POST	/api/contacts	Додавання контакту
GET	/api/savings	Список скарбничок
POST	/api/savings	Створення скарбнички
PATCH	/api/savings/:id	Оновлення балансу скарбнички
POST	/api/savings/:id/transfer	Поповнення або зняття (type: add withdraw)
DELETE	/api/savings/:id	Закриття скарбнички (з переказом залишку)
GET	/api/credits	Список кредитів
POST	/api/credits	Оформлення нового кредиту
POST	/api/credits/:id/repay	Погашення частини кредиту
GET	/api/demo	Демо-дані без авторизації (презентація)

Усі ендпоінти, крім /api/demo, потребують активної сесії (cookie session). При спробі доступу до ресурсу іншого користувача сервер повертає HTTP 403 Forbidden.

Додаток Д

Таблиця тест-кейсів функціонального тестування

Таблиця Д.1 — Тест-кейси функціонального тестування FlowBank (метод чорного ящика)

ID	Модуль	Передумова	Кроки	Очікуваний результат	Статус
ТК-01	Авторизація	Відкрити застосунок	Натиснути «Увійти» на лендінгу	Авторизація виконується, відкривається головна сторінка	Пройдено
ТК-02	Авторизація	Авторизуватись у застосунку	Закрити та повторно відкрити застосунок	Редирект на головну, сесія встановлена	Пройдено
ТК-03	Картки	Перший вхід нового користувача	Відкрити /cards	Авто-створено 3 картки: Універсальна, Біла картка, Кешбек	Пройдено
ТК-04	Картки	Список карток	Перейти на головну	Відображено горизонтальний слайдер карток	Пройдено
ТК-05	Картки	Деталі картки	Натиснути на картку	Показано баланс, тип, замаскований номер	Пройдено
ТК-06	Переказ	Переказ між власними картками	Ввести суму, номер картки; підтвердити	Баланс відправника зменшено, отримувача збільшено	Пройдено
ТК-07	Переказ	Переказ на зовнішню картку	Ввести зовнішній номер	Баланс відправника зменшено; транзакція збережена	Пройдено
ТК-08	Переказ	Недостатньо коштів	Ввести суму > балансу + ліміту	Сервер відповідає 402, помилка відображена в UI	Пройдено
ТК-09	Переказ	ACID-атомарність	Симулювати збій після списання	Rollback: баланс відправника не змінений	Пройдено
ТК-10	Транзакції	Перегляд історії	Відкрити /history	Список транзакцій у хронологічному порядку	Пройдено
ТК-11	Транзакції	Фільтрація за картою	Вибрати картку у фільтрі	Показано лише транзакції обраної картки	Пройдено
ТК-12	Аналітика	Відкрити модуль аналітики	Перейти на /analytics	Відображено підсумки та графіки	Пройдено
ТК-13	Аналітика	Зміна часового діапазону	Натиснути «7 ДНІВ» / «МІСЯЦЬ» / «3 МІСЯЦІ»	Дані фільтруються, графіки оновлюються	Пройдено
ТК-14	Накопичення	Створення скарбнички	Натиснути «+» → ввести назву і ціль	Скарбничка збережена, відображена у списку	Пройдено
ТК-15	Накопичення	Поповнення скарбнички	Вибрати скарбничку → «Поповнити» → сума	Баланс картки зменшено, баланс скарбнички збільшено	Пройдено
ТК-16	Накопичення	Зняття зі скарбнички	«Зняти» → сума → картка	Баланс скарбнички зменшено, картки збільшено	Пройдено
ТК-17	Накопичення	Закриття скарбнички	«Розбити» → обрати картку	Залишок переведено на картку, скарбничка видалена	Пройдено
ТК-18	Накопичення	Валідація: current > goal	Спробувати поповнити понад ціль	Тригер БД відхиляє операцію, помилка в UI	Пройдено
ТК-19	Кредити	Оформлення кредиту	Ввести суму та термін	Кредит створено, кредитна картка додана	Пройдено

TK-20	Кредити	Часткове погашення	«Погасити» → сума	remainingAmount зменшено, paidMonths збільшено	Пройдено
TK-21	Безпека	Доступ до чужої картки	Підмінити card_id у запиті	Сервер повертає 403 Forbidden	Пройдено
TK-22	Безпека	CVV у відповіді API	Отримати /api/cards	Поле cvvHash відсутнє у відповіді	Пройдено
TK-23	Мобільна платформа	Встановлення APK	Встановити FlowBank.apk на Android 11	Застосунок запущено, інтерфейс відображено коректно	Пройдено
TK-24	Мобільна платформа	Навігація на Android	Перейти між розділами через bottom navigation bar	Всі 5 розділів навігуються без перезавантаження	Пройдено

За результатами функціонального тестування всі 24 тест-кейси отримали статус «Пройдено». Критичних дефектів виявлено не було.

Додаток Е

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Данилюк Богдана Сергіївна, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проект) на тему «Додаток мобільного банкінгу для фізичних осіб» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
 - [+] інструменти генеративного ШІ не використовувалися.
 - [+] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, ChatGPT інші (вказати назву)) були використані для:
 - [+] перекладу іншомовних джерел;
 - [] стилістичного редагування та перевірки граматики;
 - [] допомоги у написанні/відлагодженні програмного коду;
 - [] інше: _____.

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

« » _____ 2026 р. _____
(підпис)

Богдана ДАНИЛЮК