

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

КВАЛІФІКАЦІЙНА РОБОТА
на тему: «Веб-застосунок “FlowGuard” для управління
бюджетом та особистими витратами»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень.
Використання ідей, результатів і текстів інших авторів мають посилання на
відповідне джерело*

_____ Мирослав ШЕВЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

_____ Мирослав ШЕВЧУК

_____ Керівник: _____ Вікторія КОРЕЦЬКА
канд.пед.наук, проф.

_____ Рецензент: _____

Київ 2026

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного
забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Шевчуку Мирославу Володимировичу

1. Тема кваліфікаційної роботи: «Веб-застосунок “FlowGuard” для управління бюджетом та особистими витратами».

керівник кваліфікаційної роботи канд.пед.наук, проф. Вікторія КОРЕЦЬКА,
затверджені наказом Державного університету інформаційно-комунікаційних
технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про управління особистими фінансами, бюджетне планування, облік доходів і витрат, категоризацію фінансових операцій, контроль бюджетних лімітів, фінансові цілі, персональні рекомендації, а також технічна документація щодо використання Java, Spring Boot, React, PostgreSQL, JWT, Spring Security, JUnit, Mockito та REST API.

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз предметної галузі управління особистими фінансами, обліку доходів, витрат, бюджетних лімітів, фінансових цілей та персональних рекомендацій.
2. Аналіз існуючих програмних засобів для ведення бюджету, контролю витрат, фінансових цілей та особистої аналітики.
3. Проектування web-застосунку FlowGuard, визначення вимог, архітектури, ролей користувачів, бази даних і модуля «Фінансовий Фокус».
4. Програмна реалізація серверної та клієнтської частин FlowGuard, адміністративної панелі, профілю користувача та аналітики.
5. Тестування backend-частини, перевірка модульних та інтеграційних тестів, аналіз результатів тестування.
6. Оформлення висновків, переліку посилань та додатків.

5. Перелік графічного матеріалу: *презентація*

1. Порівняльний аналіз програмних засобів для управління особистими фінансами.
2. Діаграма варіантів використання web-застосунку FlowGuard.
3. Узагальнена архітектура web-застосунку FlowGuard.
4. ER-діаграма бази даних web-застосунку FlowGuard.
5. Діаграма структури backend-частини web-застосунку FlowGuard.
6. Діаграма структури frontend-частини web-застосунку FlowGuard.
7. Діаграма послідовності авторизації користувача.
8. Діаграма послідовності додавання фінансової операції.
9. Діаграма послідовності формування підказки «Фінансовий Фокус».
10. Екранні форми web-застосунку: сторінка входу, dashboard, сторінка операцій, аналітика, профіль користувача та адміністративна панель.
11. Схема тестування backend-частини та результати виконання тестів.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури за темою	23.02–28.02.2026	

	управління особистими фінансами		
2	Аналіз існуючих програмних засобів для ведення бюджету та контролю витрат	01.03–07.03.2026	
3	Аналіз предметної галузі та визначення вимог до web-застосунку FlowGuard	09.03–15.03.2026	
4	Огляд засобів реалізації web-застосунку FlowGuard	16.03–22.03.2026	
5	Проектування архітектури, бази даних, рольової моделі доступу та діаграм	23.03–05.04.2026	
6	Програмна реалізація серверної та клієнтської частин застосунку	06.04–26.04.2026	
7	Реалізація модуля «Фінансовий Фокус», аналітики, профілю та адмін-панелі	27.04–03.05.2026	
8	Тестування backend-частини та оформлення результатів тестування	04.05–10.05.2026	
9	Оформлення роботи: вступ, висновки, реферат, список джерел і додатки	11.05–17.05.2026	
10	Розробка демонстраційних матеріалів та підготовка до захисту роботи	18.05–22.05.2026	

Здобувач вищої освіти.

(підпис)

Мирослав ШЕВЧУК

Керівник

кваліфікаційної роботи

(підпис)

Вікторія КОРЕЦЬКА

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 53 стор., 1 табл., 28 рис., 24 джерела.

Мета роботи – спрощення процесу управління особистими фінансами за рахунок розробки web-застосунку FlowGuard для централізованого обліку доходів і витрат, контролю бюджетних лімітів, відстеження фінансових цілей, перегляду аналітики та отримання персональних фінансових підказок.

Об'єкт дослідження – процес управління особистим бюджетом, доходами, витратами, бюджетними обмеженнями та фінансовими цілями користувача.

Предмет дослідження – засоби, методи та технології реалізації web-застосунку FlowGuard для управління бюджетом та особистими витратами.

Короткий зміст роботи: у кваліфікаційній роботі проведено аналіз предметної галузі управління особистими фінансами та визначено проблеми, пов'язані з ручним веденням фінансових записів, складністю контролю витрат, відсутністю єдиного середовища для фінансових даних, ризиком перевищення лімітів і недостатньою наочністю фінансової аналітики.

Проаналізовано існуючі програмні засоби для управління особистими фінансами, зокрема Monefy, Wallet, YNAB, Spendee та Money Manager. Визначено їхні переваги й недоліки, серед яких обмеження безкоштовних версій, платні функції, складність окремих інтерфейсів, обмежена web-доступність, відсутність адміністративної панелі та відсутність окремого модуля персональних фінансових підказок.

У роботі сформовано функціональні та нефункціональні вимоги до застосунку FlowGuard, обґрунтовано вибір засобів реалізації клієнтської частини, серверної частини, бази даних, авторизації та тестування. Для реалізації web-застосунку використано Java, Spring Boot, Spring Security, JWT,

Spring Data JPA, PostgreSQL, Liquibase, React, Vite, REST API, Git, JUnit, Mockito, MockMvc та H2 in-memory database.

Спроектовано загальну архітектуру застосунку, структуру бази даних, діаграму варіантів використання, діаграми послідовності авторизації, додавання фінансової операції та формування підказки «Фінансовий Фокус». Реалізовано основні модулі системи: авторизацію, профіль користувача, фінансові операції, категорії, бюджетні ліміти, фінансові цілі, аналітику, адміністративну панель і журнал дій.

Особливістю розробленої системи є модуль «Фінансовий Фокус», який аналізує витрати, бюджети та цілі користувача, визначає фінансові ризики й формує короткі персональні підказки. Модуль дозволяє користувачу швидше помітити перевищення ліміту, низький залишок бюджету, зростання витрат у певній категорії або відставання від фінансової цілі.

Проведено модульне та інтеграційне тестування backend-частини FlowGuard. Перевірено реєстрацію, авторизацію, оновлення профілю, формування фінансових підказок, повний користувацький сценарій через REST API, рольовий доступ USER / ADMIN та адміністративні функції. За результатами тестування виконано 8 тестів, які підтвердили коректну роботу основної backend-логіки.

КЛЮЧОВІ СЛОВА: FLOWGUARD, ОСОБИСТІ ФІНАНСИ, БЮДЖЕТ, ДОХОДИ, ВИТРАТИ, ФІНАНСОВІ ЦІЛІ, ФІНАНСОВИЙ ФОКУС, WEB-ЗАСТОСУНОК, SPRING BOOT, REACT, POSTGRESQL, JWT, REST API, JUNIT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	10
ВСТУП	12
1 АНАЛІЗ WEB-ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ БЮДЖЕТОМ ТА ОСОБИСТИМИ ВИТРАТАМИ.....	14
1.1 Аналіз предметної галузі управління особистими фінансам	14
1.2 Визначення основних функцій web-застосунку FlowGuard	17
1.3 Дослідження існуючих аналогів.....	18
1.4 Порівняльний аналіз аналогів та обґрунтування необхідності розробки FlowGuard.....	23
2.1 Визначення функціональних і нефункціональних вимог до застосунку	26
2.2 Вибір технічних засобів для розробки web-застосунку.....	28
2.2.1 Вибір мови програмування та середовища розробки	29
2.2.2 Вибір backend-фреймворку та засобів роботи з базою даних ...	29
2.2.3 Вибір frontend-фреймворку та програмних бібліотек.....	29
2.2.4 Система контролю версій.....	30
2.3 Архітектура web-застосунку FlowGuard	30
2.4 Проєктування ролей користувачів та прав доступу	33
2.5 Проєктування бази даних web-застосунку FlowGuard.....	34
2.6 Проєктування модуля “Фінансовий Фокус”	36
2.7 Діаграма структури backend-частини FlowGuard	36
2.8 Діаграма структури frontend-частини FlowGuard.....	37
3.1 Проєктування інтерфейсу користувача	39
3.2 Реалізація серверної частини	44
3.3 Реалізація клієнтської частини	45
3.4 Реалізація модуля “Фінансовий Фокус”	46
3.5 Реалізація адміністративної панелі	48
3.6 Завантаження проєкту на GitHub	49
3.7 Тестування web-застосунку	49
ВИСНОВКИ.....	52
ПЕРЕЛІК ПОСИЛАНЬ.....	54
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ.....	56
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ	66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування, що використовується для взаємодії між програмними компонентами або зовнішніми сервісами.

REST – архітектурний стиль організації обміну даними між клієнтською та серверною частинами web-застосунку.

HTTP – протокол передавання даних у мережі Інтернет.

HTTPS – захищена версія протоколу HTTP, що забезпечує безпечний обмін даними між клієнтом і сервером.

JSON – текстовий формат обміну даними, який використовується для передавання структурованої інформації між frontend- і backend-частинами системи.

Frontend – клієнтська частина web-застосунку, з якою безпосередньо взаємодіє користувач.

Backend – серверна частина web-застосунку, яка виконує бізнес-логіку, обробляє запити та працює з базою даних.

Java – об'єктно-орієнтована мова програмування, що використовується для реалізації серверної частини застосунку.

Spring Boot – інструмент екосистеми Spring, який спрощує створення, налаштування та запуск backend-сервісів.

Spring Security – модуль Spring для реалізації авторизації, автентифікації та обмеження доступу до ресурсів.

JWT – формат токена, який використовується для підтвердження авторизації користувача під час виконання API-запитів.

Spring Data JPA – модуль Spring для роботи з реляційними базами даних через об'єктну модель.

PostgreSQL – об'єктно-реляційна система керування базами даних.

Liquibase – інструмент для керування змінами структури бази даних за допомогою міграцій.

React – бібліотека JavaScript для створення компонентного користувацького інтерфейсу.

Vite – інструмент для швидкого створення, запуску та збирання frontend-застосунків.

Git – система контролю версій, що використовується для відстеження змін у програмному коді.

GitHub – web-сервіс для зберігання репозиторіїв, спільної роботи з кодом та контролю версій.

JUnit – фреймворк для модульного та інтеграційного тестування Java-застосунків.

Mockito – бібліотека для створення mock-об'єктів у модульному тестуванні.

MockMvc – інструмент Spring для тестування REST API без запуску повноцінного web-сервера.

H2 – вбудована in-memory база даних, що використовується для тестування.

DTO – об'єкт передавання даних між різними рівнями застосунку.

Dashboard – головна сторінка системи, на якій відображаються ключові показники та підказки.

FlowGuard – web-застосунок для управління бюджетом та особистими витратами.

Фінансовий Фокус – модуль персональних підказок, який аналізує витрати, бюджети та фінансові цілі користувача.

Audit log – журнал дій, у якому зберігаються важливі системні події та адміністративні операції.

USER – роль звичайного авторизованого користувача системи.

ADMIN – роль адміністратора, що має доступ до статистики, користувачів і журналу дій.

ВСТУП

У сучасних умовах розвитку цифрових сервісів управління особистими фінансами є важливою частиною повсякденної діяльності користувача. Користувачі регулярно здійснюють покупки, отримують доходи, оплачують послуги, планують накопичення та контролюють витрати. За відсутності єдиного інструменту для фіксації фінансових операцій виникають труднощі з аналізом бюджету, контролем лімітів і плануванням фінансових цілей [1; 2; 3].

Актуальність теми кваліфікаційної роботи зумовлена необхідністю створення зручного web-застосунку, який дозволяє централізовано вести облік доходів і витрат, розподіляти операції за категоріями, встановлювати бюджетні обмеження, переглядати фінансові цілі та отримувати персональні рекомендації. У реальних умовах користувачі часто використовують таблиці, нотатки або декілька різних додатків, що ускладнює цілісне бачення фінансової ситуації [2; 4].

Мета роботи – спрощення процесу управління особистими фінансами шляхом розробки web-застосунку FlowGuard для обліку доходів і витрат, контролю бюджетних лімітів, відстеження фінансових цілей, перегляду аналітики та отримання персональних фінансових підказок.

Об’єкт дослідження – процес управління особистим бюджетом, доходами, витратами та фінансовими цілями користувача.

Предмет дослідження – засоби, методи та технології реалізації web-застосунку для управління бюджетом та особистими витратами.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну галузь управління особистими фінансами та визначити основні проблеми контролю доходів, витрат, бюджетних лімітів і фінансових цілей [1; 2; 3];
- дослідити існуючі програмні засоби для ведення бюджету та порівняти їхні можливості з майбутнім застосунком FlowGuard [12; 13; 14; 15; 16];

- сформулювати функціональні та нефункціональні вимоги до web-застосунку;
- обґрунтувати вибір технологій для реалізації системи, зокрема Java, Spring Boot, React, PostgreSQL, JWT, JUnit та Mockito [17; 18; 19; 20; 21; 22; 23; 24];
- спроектувати архітектуру застосунку, базу даних, рольову модель доступу та основні діаграми взаємодії компонентів;
- реалізувати серверну частину, клієнтську частину, профіль користувача, адміністративну панель, аналітику та модуль «Фінансовий Фокус»;
- провести модульне та інтеграційне тестування backend-частини й оцінити відповідність розробленого програмного забезпечення поставленим вимогам.

Методи дослідження. У роботі використовуються методи аналізу предметної галузі, порівняльного аналізу програмних засобів, об'єктно-орієнтованого проєктування,

Наукова новизна одержаних результатів полягає у розробці структури web-застосунку FlowGuard, який поєднує стандартний облік особистих фінансів із модулем персональних підказок «Фінансовий Фокус». На відміну від типових додатків для фіксації витрат, запропоноване рішення орієнтоване на автоматичне виявлення фінансових ризиків користувача.

Практична значущість одержаних результатів полягає в тому, що розроблений web-застосунок може бути використаний як інструмент для щоденного контролю особистого бюджету, аналізу витрат, планування фінансових цілей та адміністрування користувачів платформи.

1 АНАЛІЗ WEB-ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ БЮДЖЕТОМ ТА ОСОБИСТИМИ ВИТРАТАМИ

1.1 Аналіз предметної галузі управління особистими фінансами

У сучасних умовах управління особистими фінансами є важливою складовою фінансової грамотності користувача. Щоденні покупки, регулярні платежі, надходження доходів, накопичення коштів, контроль боргових зобов'язань і планування майбутніх витрат потребують системного підходу. Якщо користувач не фіксує фінансові операції або робить це нерегулярно, він поступово втрачає розуміння власного бюджету, не може точно визначити основні напрями витрат і своєчасно помітити перевищення запланованих обмежень [1; 2; 3].

Предметна галузь управління особистими фінансами охоплює процеси обліку доходів і витрат, категоризації фінансових операцій, планування бюджетних лімітів, формування фінансових цілей та аналізу фінансової поведінки користувача. Основною задачею цієї галузі є створення умов, за яких користувач може швидко отримати відповідь на базові фінансові питання.

Особистий бюджет можна розглядати як сукупність доходів, витрат, залишку коштів, фінансових обмежень і запланованих цілей за певний часовий період. Найчастіше користувачі аналізують бюджет у межах місяця, оскільки більшість регулярних платежів і побутових витрат мають саме місячний цикл [2; 4].

У традиційному підході користувач може вести облік витрат у паперовому блокноті, нотатках смартфона або електронних таблицях. Однак такі способи мають низку обмежень. Паперовий облік не дозволяє швидко виконувати пошук, сортування та автоматичний аналіз даних. Нотатки не забезпечують структурованості фінансової інформації. Електронні таблиці є

гнучкими, але потребують ручного налаштування формул, категорій, фільтрів і діаграм.

Використання спеціалізованого web-застосунку дозволяє централізувати фінансові дані користувача в одному середовищі. Такий підхід спрощує введення операцій, зменшує ймовірність помилок у розрахунках, забезпечує автоматичне формування статистики та дозволяє швидко отримувати аналітичну інформацію [7; 9; 10].

Основними об'єктами предметної галузі є користувач, фінансова операція, категорія, бюджетний ліміт, фінансова ціль, аналітичний показник і персональна фінансова підказка. Користувач створює записи про доходи та витрати, обирає або налаштовує категорії, встановлює обмеження для певних напрямів витрат і переглядає результати у вигляді статистики.

Типовими категоріями витрат можуть бути продукти, транспорт, житло, комунальні платежі, медицина, навчання, відпочинок, покупки, підписки, накопичення та інші напрями. Для користувача важливо мати можливість створювати власні категорії, оскільки структура витрат залежить від способу життя, рівня доходу, сімейного стану та фінансових пріоритетів.

Окрему роль у предметній галузі відіграють бюджетні ліміти. Вони дозволяють не просто фіксувати витрати після їх здійснення, а заздалегідь задавати допустимі межі для конкретних категорій. Якщо витрати наближаються до встановленої межі, система повинна показувати попередження, щоб користувач міг своєчасно скоригувати фінансову поведінку.

Важливою складовою управління особистими фінансами є фінансові цілі. Користувач може накопичувати кошти на навчання, подорож, техніку, фінансову подушку або іншу значущу покупку. Для цього система повинна дозволяти створювати ціль, зазначати цільову суму, поточний прогрес і дату досягнення.

Особливою проблемою предметної галузі є те, що користувач часто бачить фінансову інформацію фрагментарно. Саме тому в межах web-

застосунку FlowGuard доцільно передбачити не лише стандартну статистику, а й персональний блок-помічник, який буде виділяти головний фінансовий ризик або пораду на поточний момент.

Таким блоком у системі є модуль «Фінансовий Фокус». Його призначення полягає в автоматичному аналізі фінансової поведінки користувача та формуванні коротких підказок. Такий підхід робить застосунок не просто електронним журналом витрат, а інструментом підтримки фінансових рішень.



Рис. 1.1 Загальна схема процесу управління особистими фінансами у web-застосунку FlowGuard

Цільовою аудиторією web-застосунку FlowGuard є користувачі, які хочуть контролювати власні доходи, витрати, бюджетні обмеження та

фінансові цілі. До цієї аудиторії можна віднести студентів, найманих працівників, фрилансерів, молоді сім'ї, людей із нерівномірним доходом, а також користувачів, які прагнуть сформувати навичку регулярного фінансового обліку.

1.2 Визначення основних функцій web-застосунку FlowGuard

Web-застосунок FlowGuard призначений для комплексного управління особистими фінансами користувача. Його основна ідея полягає в тому, щоб об'єднати в одному середовищі облік доходів і витрат, бюджетне планування, фінансові цілі, аналітику, персональні підказки та адміністративний контроль.

Першою важливою функцією системи є реєстрація та авторизація користувача. Кожен користувач повинен мати окремий обліковий запис, через який він отримує доступ лише до власних фінансових даних. Для захисту інформації використовується авторизація, а доступ до функцій системи обмежується відповідно до ролі користувача [5; 6; 19].

У системі передбачено три основні типи користувачів: гість, авторизований користувач та адміністратор. Гість може переглядати вітальну сторінку, пройти реєстрацію або виконати вхід до системи. Авторизований користувач має доступ до особистого фінансового кабінету. Адміністратор має всі можливості звичайного користувача, а також додатковий доступ до списку користувачів, загальної статистики, журналу дій і системних попереджень.

Однією з основних функцій FlowGuard є керування фінансовими операціями. Користувач може додавати доходи й витрати, зазначаючи тип операції, суму, дату, категорію та короткий опис. Також повинна бути можливість редагувати або видаляти раніше створені операції.

Наступною функцією є робота з категоріями. Категорії потрібні для структурування доходів і витрат. Користувач повинен мати можливість створювати власні категорії відповідно до власних потреб, оскільки універсального набору категорій для всіх користувачів не існує.

Окремий функціональний блок пов'язаний із бюджетними лімітами. Користувач може встановлювати обмеження для певних категорій витрат на місяць. При досягненні 70%, 90% або 100% система може показувати попередження.

Також у FlowGuard передбачено сторінку фінансових цілей. Користувач може створити ціль, зазначити її назву, бажану суму, поточну суму накопичення та кінцеву дату. На основі цих даних система може показувати прогрес виконання цілі та орієнтовну суму, яку потрібно відкласти щомісяця.

Ключовою особливістю системи є модуль «Фінансовий Фокус». Він автоматично аналізує операції, бюджети та цілі користувача, після чого формує коротку персональну підказку. Така функція підвищує практичну цінність застосунку, оскільки користувач отримує не лише статистику, а й конкретний сигнал, на що потрібно звернути увагу.

1.3 Дослідження існуючих аналогів

Для обґрунтування доцільності розробки web-застосунку FlowGuard необхідно проаналізувати існуючі програмні засоби, які використовуються для управління особистими фінансами. На ринку вже існує значна кількість мобільних додатків і web-сервісів для ведення бюджету, однак вони відрізняються за функціональністю, складністю інтерфейсу, доступністю безкоштовних можливостей, підтримкою аналітики, наявністю web-версії та можливістю персоналізованих підказок [12; 13; 14; 15; 16].

Для аналізу було обрано такі програмні засоби: Monefy, Wallet, YNAB, Spendee та Money Manager. Ці застосунки є прикладами різних підходів до управління особистими фінансами: від простого щоденного обліку витрат до повноцінного бюджетного планування з аналітикою та фінансовими цілями.

Monefy є популярним мобільним додатком для швидкого обліку доходів і витрат. Його основна перевага полягає у простому інтерфейсі та швидкому додаванні фінансових операцій. Водночас частина можливостей доступна

лише у платній версії, а сам додаток більше орієнтований на простий облік витрат, ніж на комплексне управління бюджетними лімітами [12].

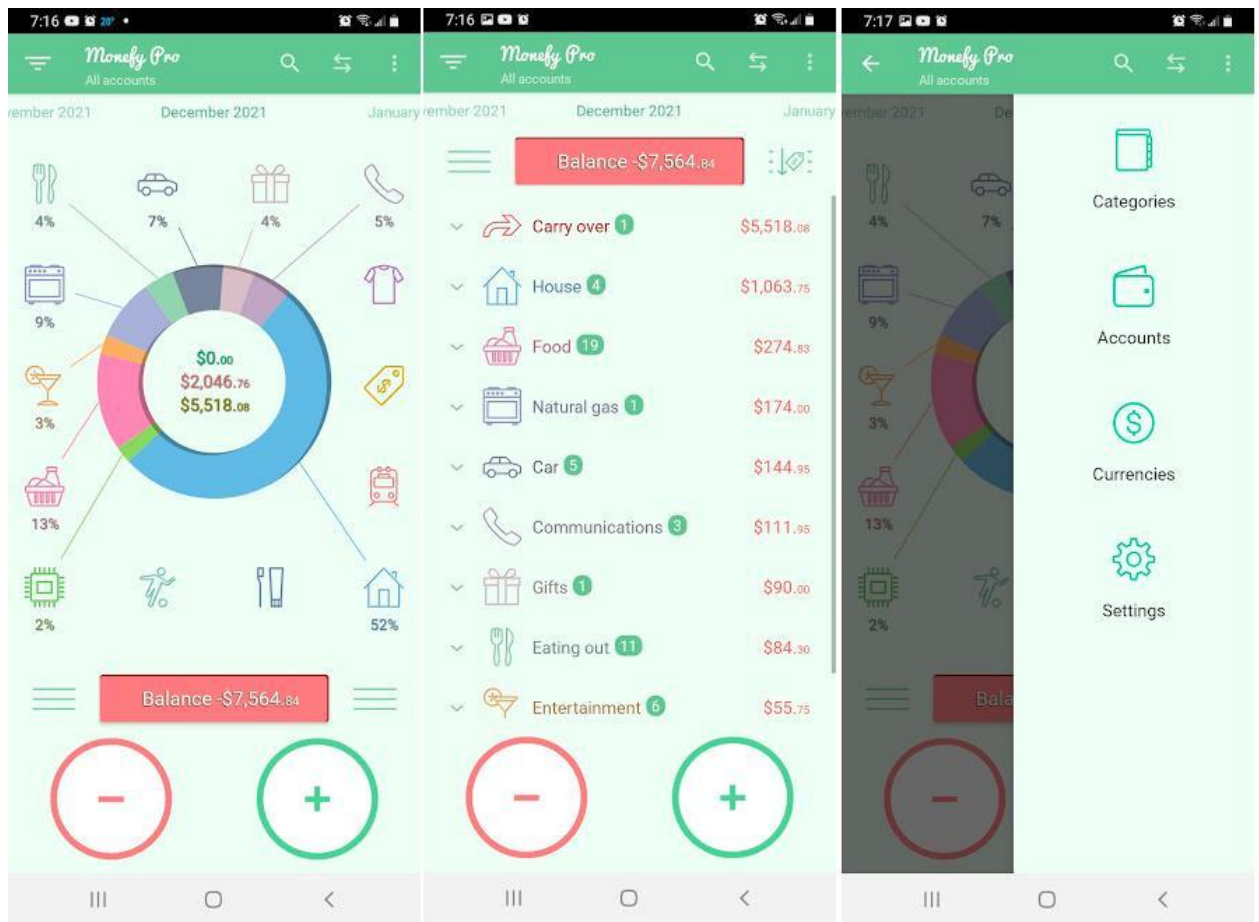


Рис. 1.2 Приклад інтерфейсу додатку Monefy

Wallet є більш функціональним сервісом для управління особистими фінансами. Він підтримує облік доходів і витрат, бюджетне планування, категорії, фінансові цілі, звіти та синхронізацію між пристроями. Однією з переваг Wallet є можливість інтеграції з банківськими рахунками, однак частина функцій доступна лише за платною підпискою [13].

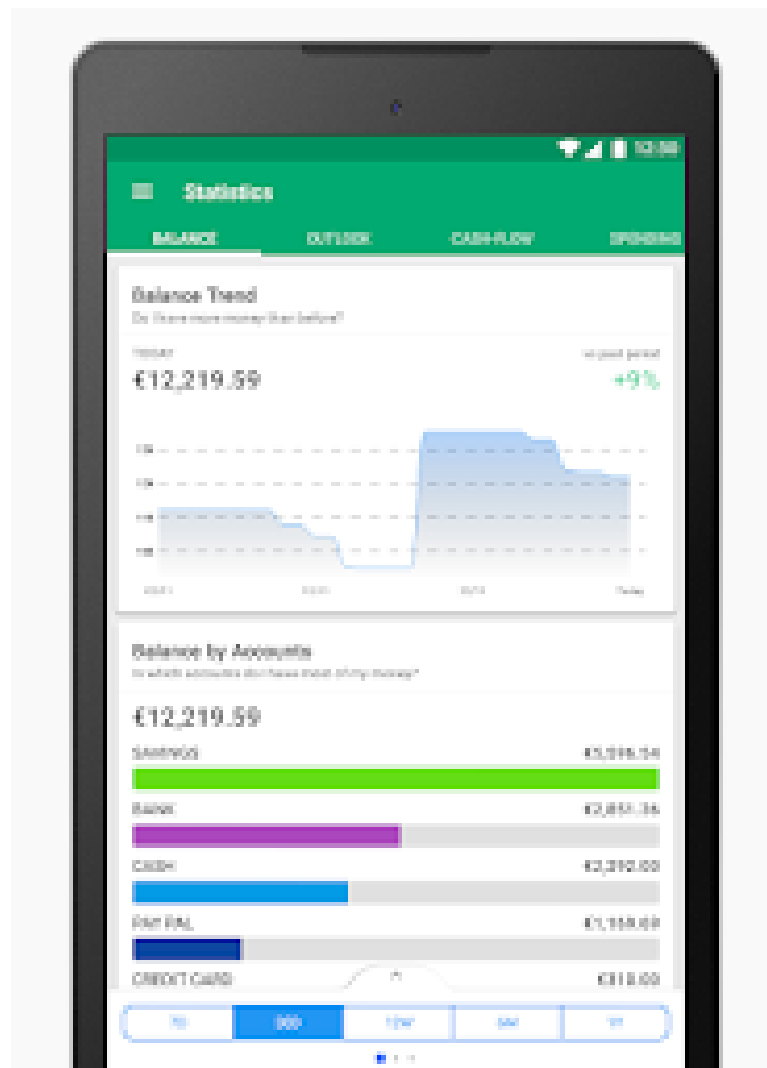


Рис. 1.3 Приклад інтерфейсу сервісу Wallet

YNAB або You Need A Budget є сервісом, який орієнтований на методику активного бюджетування. Його головна ідея полягає в тому, що кожна сума доходу повинна мати конкретне призначення. Перевагою YNAB є сильна система бюджетного планування, але сервіс є платним після пробного періоду та може бути складним для нових користувачів [14].

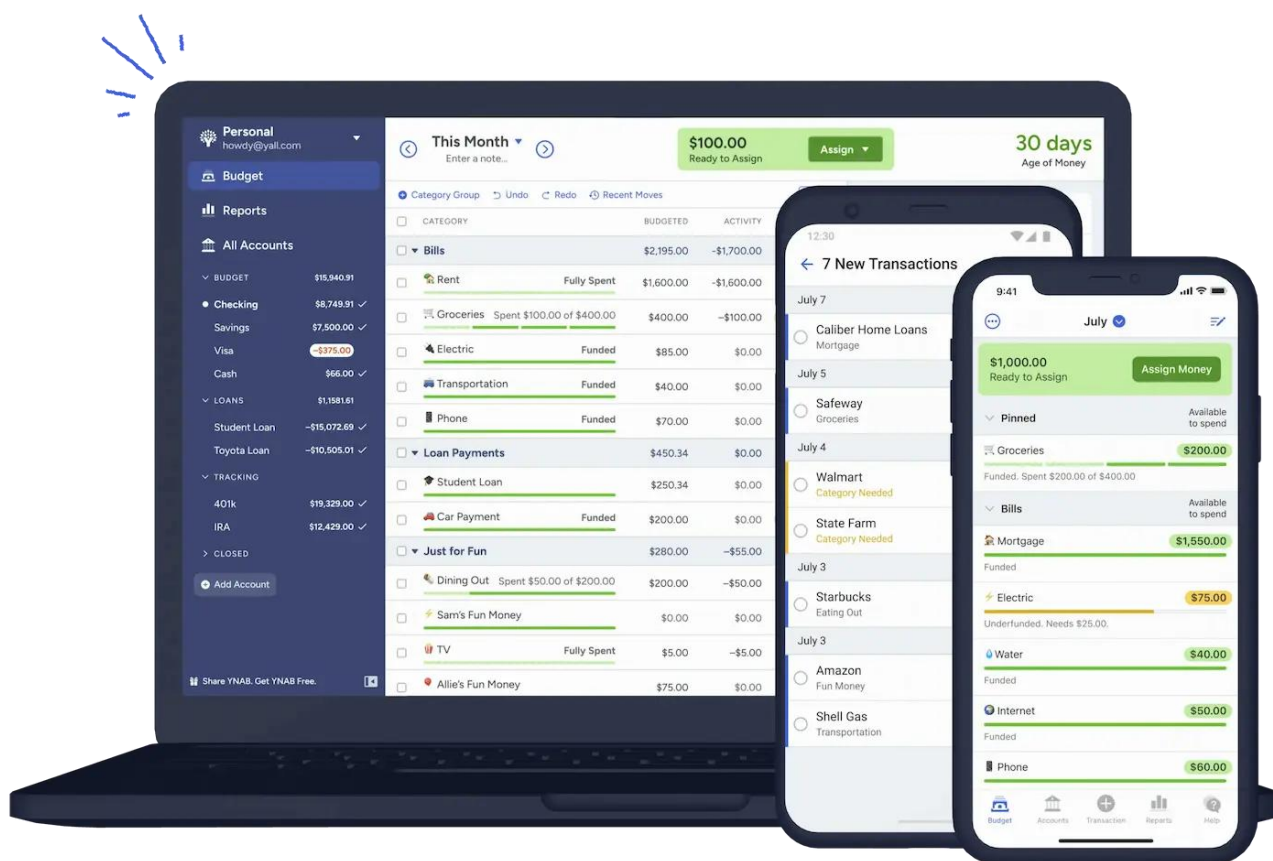


Рис. 1.4 Приклад інтерфейсу сервісу YNAB

Spendee є програмним засобом для контролю особистих і спільних фінансів. Додаток дозволяє створювати гаманці, додавати доходи й витрати, переглядати статистику, працювати з категоріями та планувати бюджет. Проте розширені можливості, зокрема банківська інтеграція і частина аналітичних функцій, доступні переважно у платній версії [15].



Рис. 1.5 Приклад інтерфейсу сервісу Spendee

Money Manager є мобільним додатком для ведення особистого бюджету. Недоліками є менш сучасний інтерфейс, обмежена web-доступність і відсутність повноцінної системи персоналізованих фінансових підказок [16].

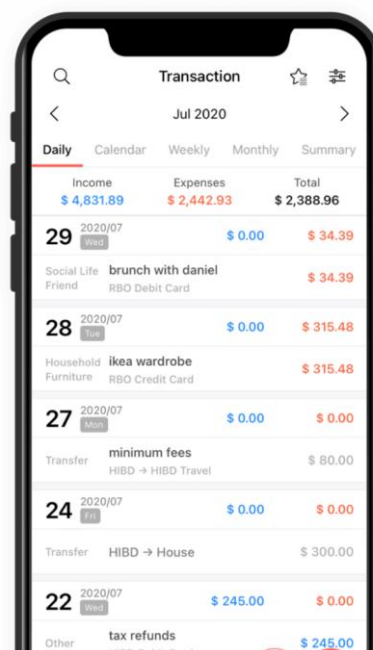


Рис. 1.6 Приклад інтерфейсу додатку Money Manager

Проведений аналіз показує, що більшість існуючих рішень підтримують базовий облік доходів і витрат, категорії, графіки та певний рівень аналітики. Однак не всі засоби мають повноцінну web-версію, розвинену роботу з фінансовими цілями, адміністративну частину, журнал дій і персональні підказки.

1.4 Порівняльний аналіз аналогів та обґрунтування необхідності розробки FlowGuard

Для узагальнення результатів дослідження аналогів доцільно сформулювати порівняльну таблицю. У ній відображено ключові характеристики програмних засобів для управління особистими фінансами. Порівняння проводиться за такими показниками: облік доходів і витрат, категорії, бюджетні ліміти, фінансові цілі, аналітика, графіки, фільтрація, web-версія, рольова модель, адміністративна панель, журнал дій, персональні підказки та наявність модуля «Фінансовий Фокус».

Таблиця 1.1

Порівняльний аналіз програмних засобів для управління особистими фінансами

Показник	Monefy	YNAB	Spendee	Money Manager	FlowGuard
Власні категорії	Доступно переважно у платній версії	+	+	+	+
Бюджетні ліміти	Є лише базовий контроль витрат	+	+	+	+

Продовження таблиці 1.1

Порівняльний аналіз програмних засобів для управління особистими
фінансами

Показник	Monefy	YNAB	Spendee	Money Manager	FlowGuard
Фінансові цілі	Немає повноцінного модуля цілей	+	+	Є базове планування без розширеної логіки	+
Аналітика та звіти	Базова статистика без глибокого аналізу	+	+	Середній рівень аналітики	+
Фільтрація операцій	Обмежена у безкоштовній версії	+	+	Є базова фільтрація	+
Web-версія	-	+	+	-	+
Адміністративна панель	-	-	-	-	+
Персональні підказки	-	Є поради з бюджетування, але в межах методики YNAB	Є базові підказки, але без пріоритетного фінансового фокусу	-	+
Модуль “Фінансовий Фокус”	-	-	-	-	+

Результати порівняння показують, що існуючі програмні засоби мають значну кількість корисних функцій, однак кожен із них має певні обмеження.

Monefy є зручним для швидкого обліку витрат, але не забезпечує повноцінного бюджетного планування.

Wallet має широку функціональність, але частина можливостей залежить від платної підписки та банківських інтеграцій.

YNAB пропонує сильну методику бюджетування, однак може бути складним для нових користувачів.

Spendee має зручний візуальний інтерфейс, але розширені функції також переважно платні.

Money Manager дозволяє вести детальний облік, але має обмежену web-доступність.

FlowGuard орієнтований на поєднання базового обліку фінансових операцій, бюджетного планування, фінансових цілей, аналітики та персональних підказок. Його важливою відмінністю є модуль «Фінансовий Фокус», який не просто відображає статистику, а визначає головний фінансовий ризик або пораду на поточний момент.

Отже, розробка web-застосунку FlowGuard є доцільною, оскільки існуючі аналоги не повністю поєднують простоту щоденного використання, web-доступність, бюджетні ліміти, фінансові цілі, персональні підказки, рольову модель і адміністративну панель.

2 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ FLOWGUARD ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

2.1 Визначення функціональних і нефункціональних вимог до застосунку

На основі аналізу предметної галузі та порівняння існуючих програмних засобів було визначено вимоги до web-застосунку FlowGuard. Система повинна забезпечувати користувачу зручний інструмент для обліку доходів і витрат, контролю бюджетних лімітів, відстеження фінансових цілей, перегляду аналітики та отримання персональних фінансових підказок.

Оскільки застосунок працює з персональними фінансовими даними, важливою вимогою є розмежування доступу. Кожен користувач повинен бачити тільки власні операції, категорії, бюджети, цілі, аналітику та профіль. Для цього у системі передбачено рольову модель, яка включає гостя, авторизованого користувача та адміністратора [5; 6; 19].

До основних функціональних вимог web-застосунку FlowGuard належать:

- реєстрація та авторизація користувача;
- розмежування прав доступу відповідно до ролей гостя, користувача та адміністратора;
- додавання, редагування та видалення фінансових операцій;
- розподіл доходів і витрат за категоріями;
- створення, редагування та видалення категорій;
- встановлення бюджетних лімітів за категоріями;
- створення та відстеження фінансових цілей;
- перегляд аналітики доходів, витрат і балансу;
- редагування профілю користувача;
- формування персональних підказок модулем «Фінансовий Фокус»;

– перегляд адміністративної статистики та журналу дій користувачів.

Нефункціональні вимоги визначають якісні характеристики системи. Для FlowGuard важливо забезпечити швидку роботу інтерфейсу, захист даних, адаптивність, стабільність і зрозумілість для користувача.

- час завантаження основних сторінок системи – до 3 секунд;
- час відповіді стандартного API-запиту – до 2 секунд;
- додавання фінансової операції – за 3–5 дій користувача;
- підтримка не менше 1000 користувачів;
- адаптивність інтерфейсу для екранів від 360 px;
- доступ до персональних фінансових даних тільки після авторизації;
- збереження даних у реляційній базі даних PostgreSQL;
- використання JWT-токена для авторизації запитів;
- можливість освоїти базові функції системи за 5–10 хвилин.

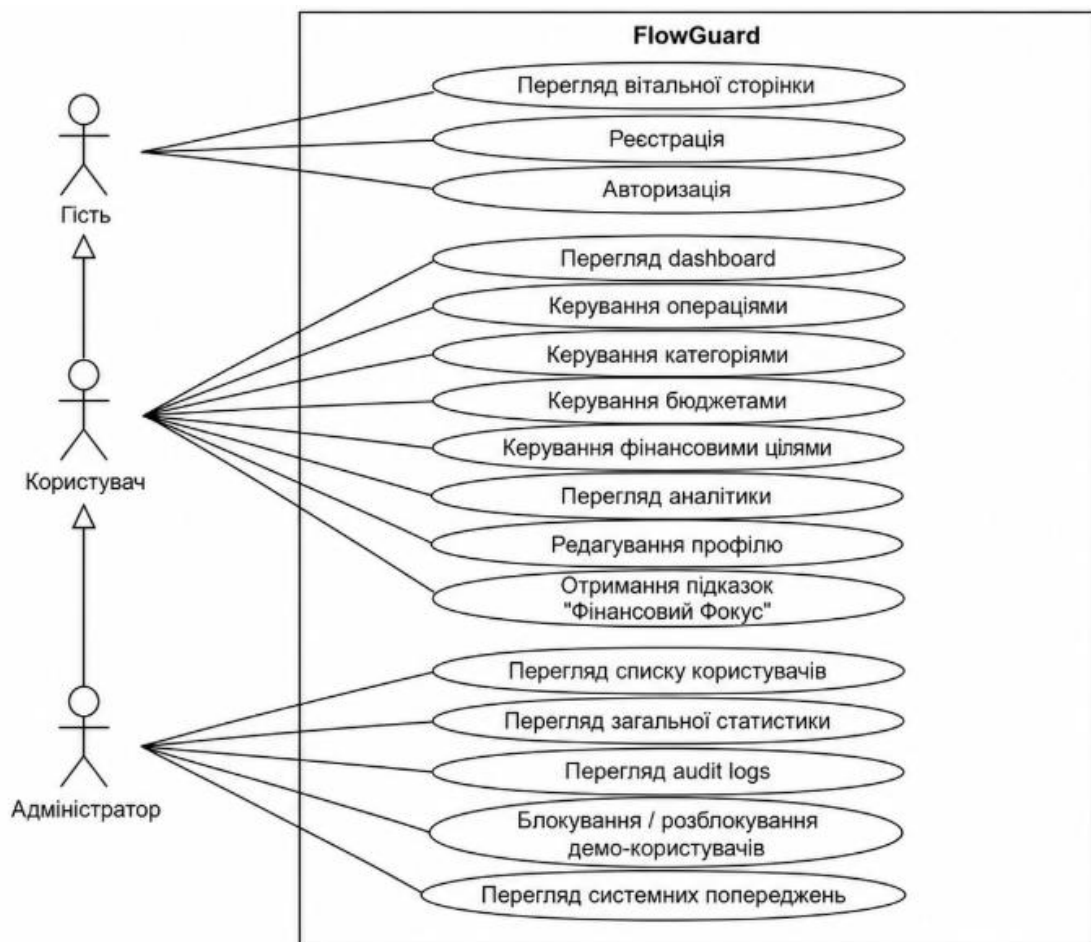


Рис. 2.1 Діаграма варіантів використання web-застосунку FlowGuard

2.2 Вибір технічних засобів для розробки web-застосунку

FlowGuard

Для реалізації FlowGuard було обрано клієнт-серверну архітектуру. Такий підхід дозволяє розділити інтерфейс користувача, серверну бізнес-логіку та збереження даних. Клієнтська частина відповідає за взаємодію з користувачем, а серверна частина обробляє запити, виконує перевірку даних, реалізує бізнес-правила, контролює доступ і взаємодіє з базою даних [9; 10].

- клієнтський рівень, реалізований за допомогою React;
- серверний рівень, реалізований за допомогою Java та Spring Boot;
- рівень даних, реалізований на основі PostgreSQL.

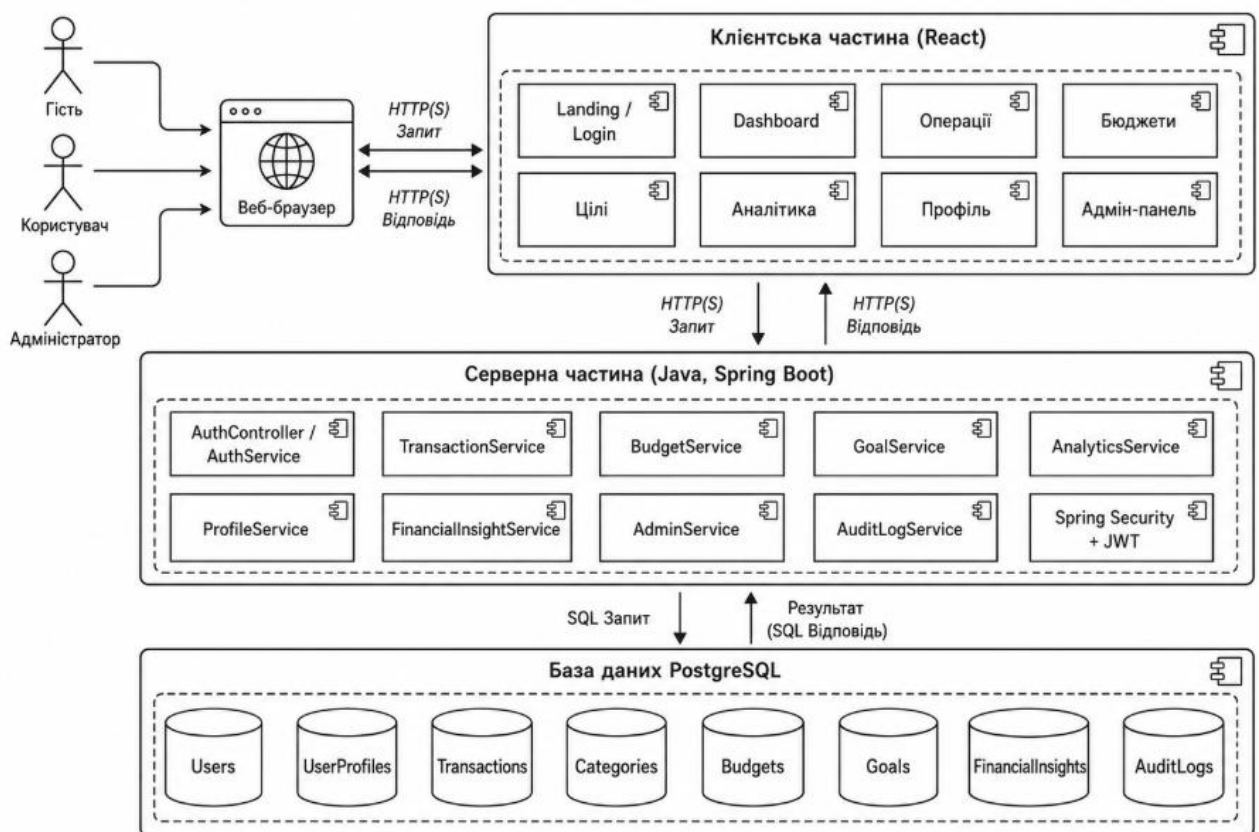


Рис. 2.2 Узагальнена архітектура web-застосунку FlowGuard

2.2.1 Вибір мови програмування та середовища розробки

Для серверної частини застосунку обрано мову програмування Java. Вона добре підходить для створення web-застосунків із чіткою структурою, розділенням логіки на класи, сервіси, контролери та репозиторії. Java дозволяє реалізувати складну бізнес-логіку, працювати з базою даних, обробляти запити та будувати масштабовану серверну частину [17].

У FlowGuard Java використовується для реалізації основної backend-логіки: авторизації, реєстрації, роботи з профілем, фінансовими операціями, бюджетами, цілями, аналітикою, адміністративними функціями та модулем «Фінансовий Фокус».

2.2.2 Вибір backend-фреймворку та засобів роботи з базою даних

Для backend-частини FlowGuard обрано Spring Boot. Цей фреймворк дозволяє швидко створювати web-застосунки, налаштовувати REST API, реалізовувати контролери, сервіси, репозиторії та працювати з базою даних [18]. Для захисту системи використовується Spring Security, який дозволяє обмежувати доступ до endpoint-ів залежно від ролі користувача [19].

Spring Data JPA використовується для роботи з базою даних через репозиторії. Для збереження даних обрано PostgreSQL, яка підходить для роботи зі структурованими фінансовими даними, оскільки підтримує реляційну модель, зв'язки між таблицями, індекси та обмеження цілісності [20].

2.2.3 Вибір frontend-фреймворку та програмних бібліотек

Для клієнтської частини обрано React. Основна перевага React полягає в компонентному підході. Інтерфейс застосунку можна розділити на окремі

компоненти: форми, таблиці, картки, навігаційне меню, графіки, блоки статистики та віджети підказок [21].

Для створення та запуску frontend-проєкту використовується Vite. Він забезпечує швидку збірку, зручний режим розробки та оновлення сторінки під час зміни коду [22].

2.2.4 Система контролю версій

Для контролю версій використовується Git. Він дозволяє зберігати історію змін, працювати з різними гілками, повертатися до попередніх версій коду та контролювати процес розробки. Для збереження репозиторію використовується GitHub [24].

2.3 Архітектура web-застосунку FlowGuard

Архітектура FlowGuard побудована за принципом розділення відповідальності між частинами системи. Клієнтська частина не працює напряду з базою даних, а звертається до backend через REST API. Backend перевіряє запити, виконує бізнес-логіку, звертається до бази даних і повертає результат.

- frontend-застосунок React;
- REST API;
- backend-сервіси Spring Boot;
- репозиторії для роботи з базою даних;
- база даних PostgreSQL;
- механізм JWT-авторизації;
- модуль перевірки ролей;
- модуль формування фінансових підказок.

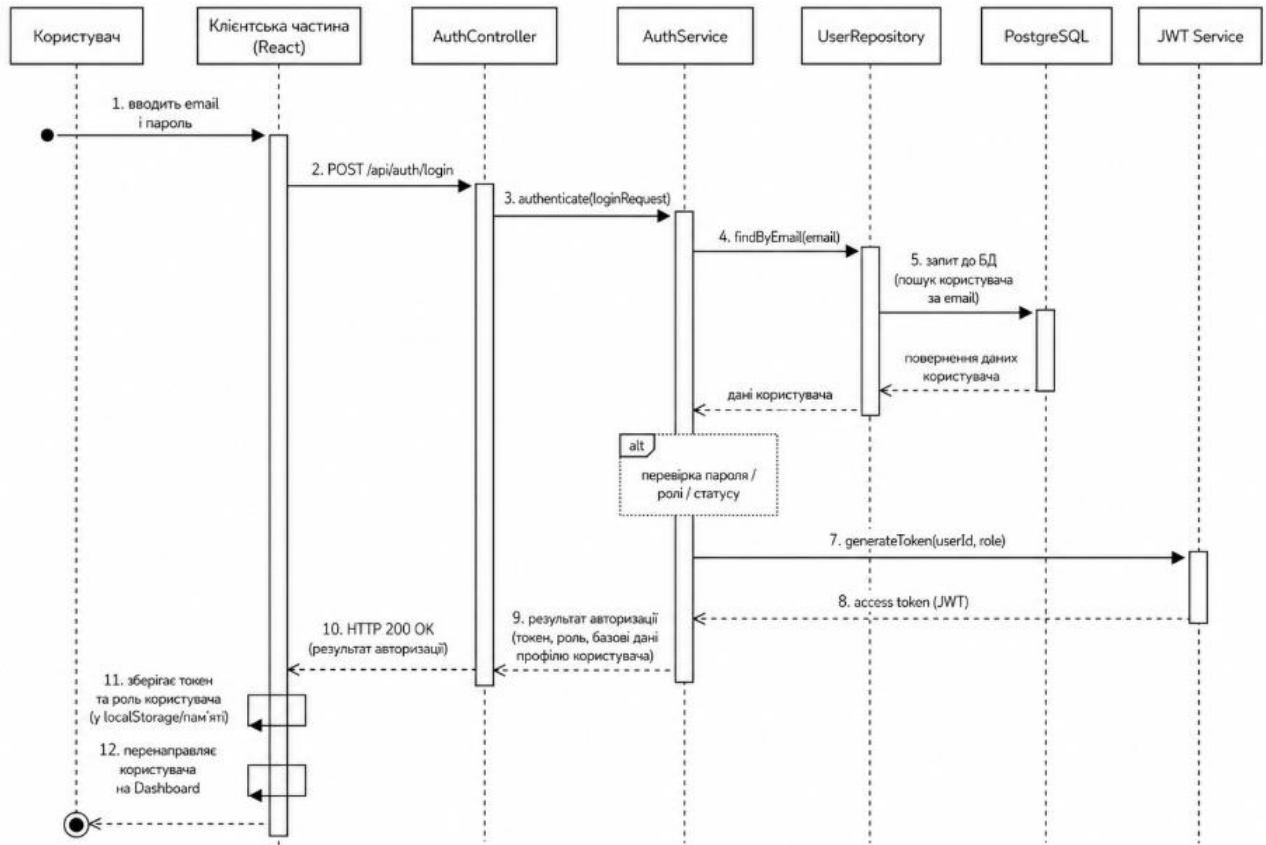


Рис. 2.3 Діаграма послідовності авторизації користувача у FlowGuard

Процес додавання фінансової операції проходить через декілька етапів. Користувач відкриває сторінку операцій, заповнює форму та надсилає дані. Frontend передає запит на backend із JWT-токеном.

Сервер перевіряє користувача, валідність даних, належність категорії цьому користувачу та після цього зберігає операцію в базі даних.

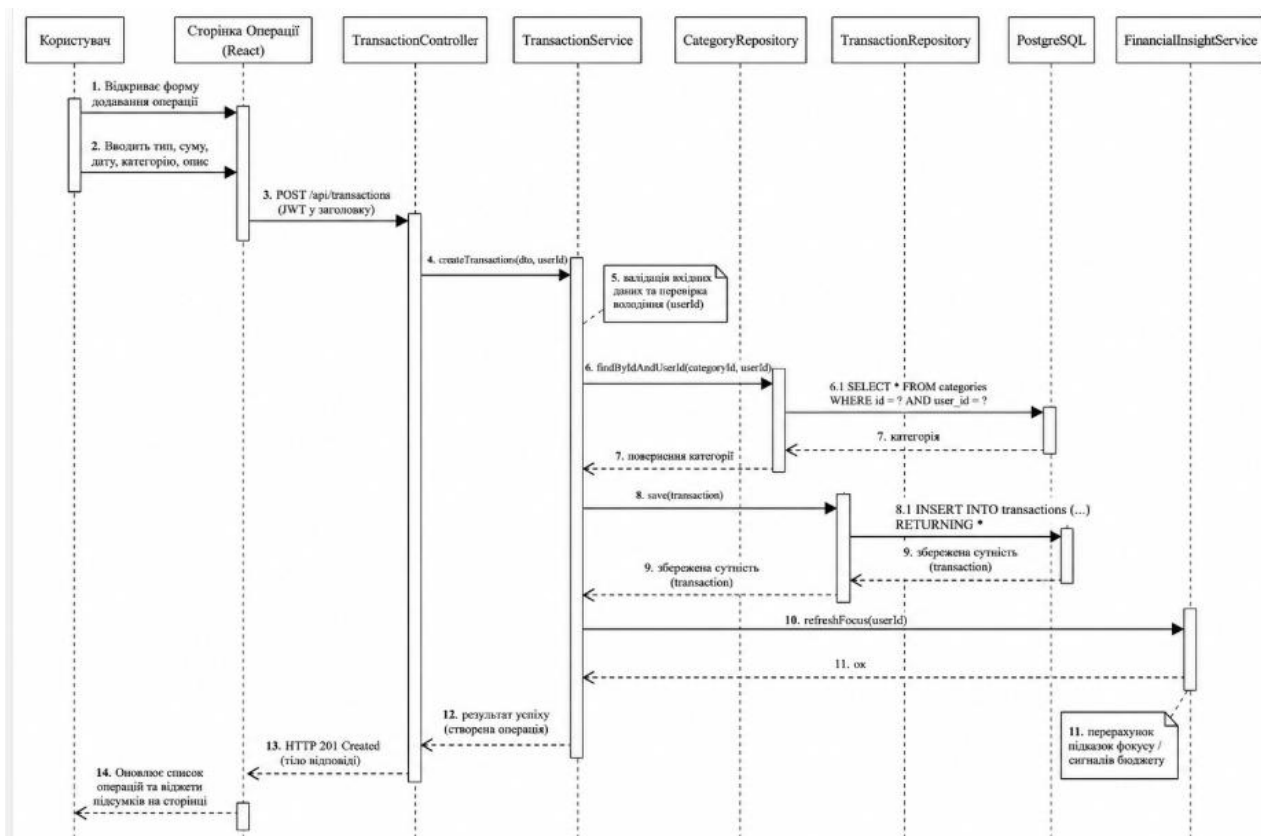


Рис. 2.4 Діаграма послідовності додавання фінансової операції у web-застосунку FlowGuard

Окремим процесом є формування підказки «Фінансовий Фокус». Після відкриття dashboard frontend надсилає запит до endpoint /api/insights/focus. Backend отримує фінансові операції, активні бюджети та цілі користувача. Після цього FinancialInsightService аналізує правила: перевищення 70% або 90% ліміту, низький залишок бюджету, зростання витрат порівняно з минулим місяцем або відставання від фінансової цілі.

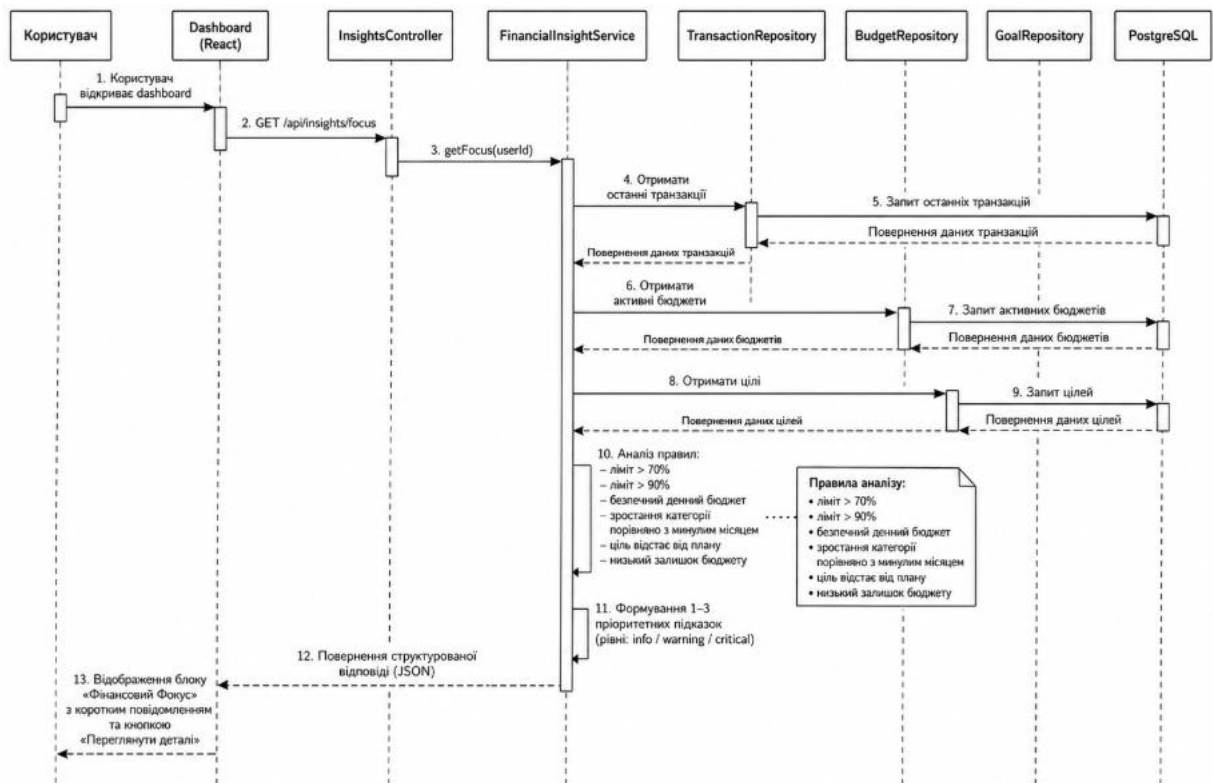


Рис. 2.5 Діаграма послідовності формування підказки “Фінансовий Фокус” у web-застосунку FlowGuard

2.4 Проєктування ролей користувачів та прав доступу

У FlowGuard передбачено рольову модель доступу. Вона необхідна для того, щоб різні типи користувачів мали доступ лише до тих функцій, які відповідають їхнім правам. У системі використовується три типи учасників: гість, користувач і адміністратор.

Гість – це неавторизований відвідувач системи. Він може переглядати вітальну сторінку, перейти на сторінку реєстрації або авторизації. Користувач може керувати власними фінансовими даними: створювати категорії, додавати доходи й витрати, встановлювати бюджети, створювати фінансові цілі, переглядати аналітику, редагувати профіль і отримувати підказки «Фінансовий Фокус».

Адміністратор має всі можливості звичайного користувача, а також доступ до адміністративної панелі. В адміністративній частині адміністратор

може переглядати список користувачів, загальну статистику платформи, audit logs, системні попередження, а також блокувати, розблоковувати або змінювати роль демо-користувачів.

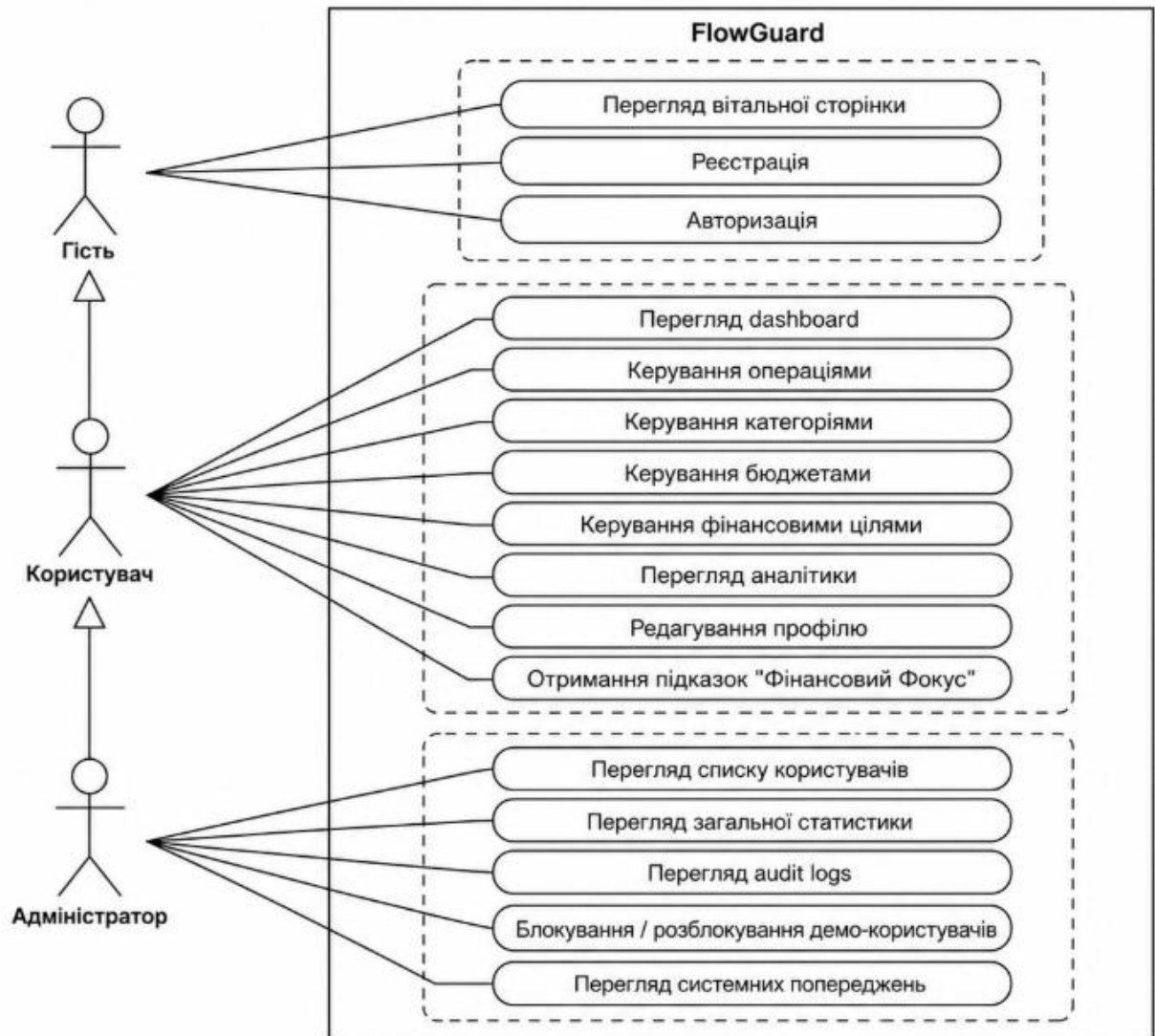


Рис. 2.6 Схема рольової моделі доступу у FlowGuard

2.5 Проєктування бази даних web-застосунку FlowGuard

База даних FlowGuard повинна забезпечувати збереження всієї основної інформації системи: користувачів, профілів, категорій, фінансових операцій, бюджетів, цілей, фінансових підказок та журналу дій. Основною сутністю є

Users, яка містить інформацію про обліковий запис користувача: ідентифікатор, ім'я, email, хеш пароля, роль, статус і дату створення.

Сутність UserProfiles зберігає додаткову інформацію про користувача: телефон, місто, місячний дохід, фінансовий пріоритет, бажану валюту, колір аватара та дату оновлення профілю. Сутність Categories використовується для збереження категорій доходів і витрат. Сутність Transactions зберігає фінансові операції. Сутність Budgets відповідає за бюджетні ліміти, а Goals – за фінансові цілі.

Сутність FinancialInsights зберігає фінансові підказки. Вона містить рівень підказки, заголовок, повідомлення, посилання на деталі, дату створення та ознаку активності. Сутність AuditLogs використовується для збереження журналу дій.

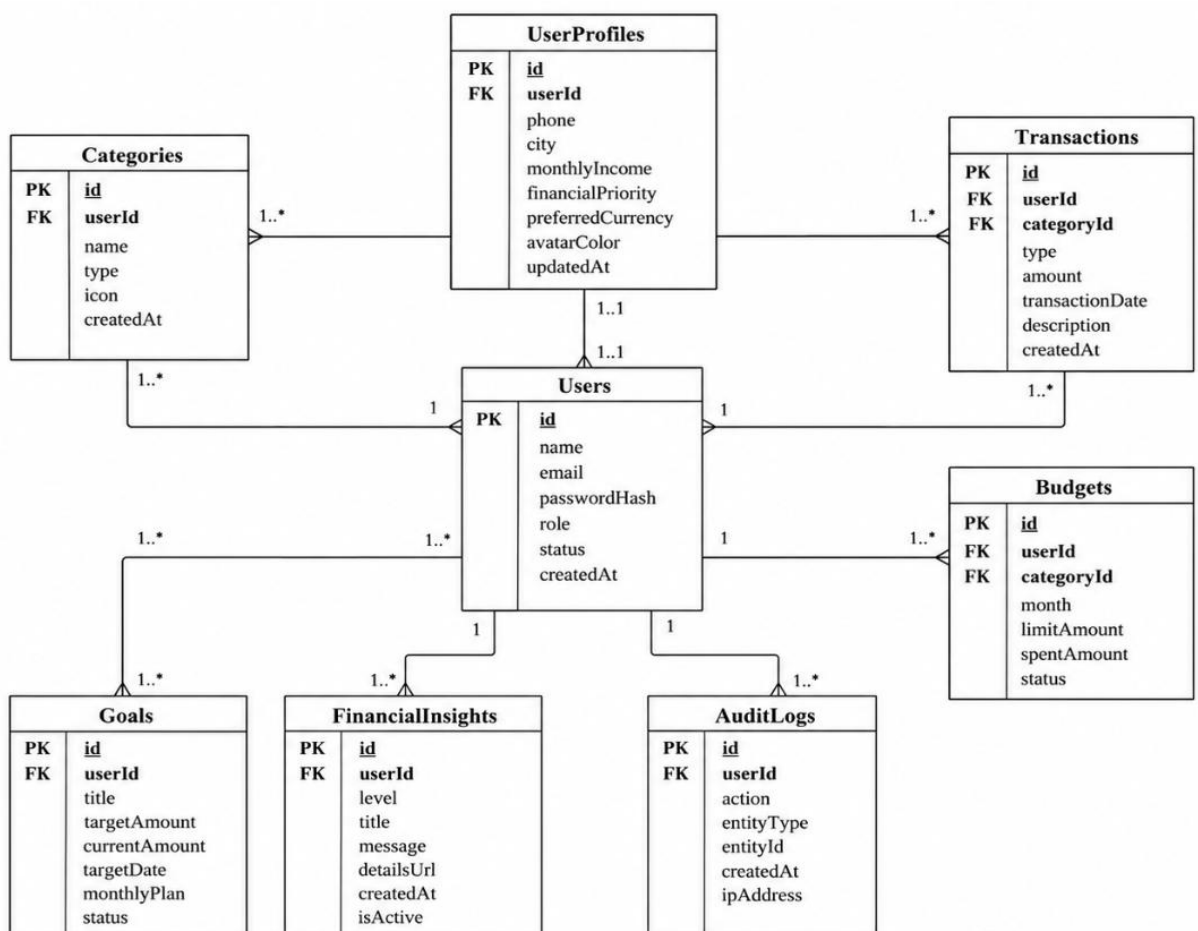


Рис. 2.7 ER-діаграма бази даних web-застосунку FlowGuard

2.6 Проєктування модуля “Фінансовий Фокус”

Модуль «Фінансовий Фокус» є унікальною функцією web-застосунку FlowGuard. Його призначення полягає в тому, щоб не просто показувати користувачу статистику, а самостійно визначати головний фінансовий ризик або пораду на поточний момент.

- категорія витрат наближається до перевищення ліміту;
- ліміт категорії вже майже вичерпано;
- витрати певної категорії зросли порівняно з минулим місяцем;
- залишок бюджету до кінця місяця є низьким;
- фінансова ціль відстає від планового темпу.

На backend-рівні модуль може бути реалізований як FinancialInsightService. Він отримує дані з репозиторіїв фінансових операцій, бюджетів і цілей, застосовує правила аналізу та формує структуровану відповідь для frontend. Frontend відображає результат у вигляді компактного блоку на dashboard.

2.7 Діаграма структури backend-частини FlowGuard

Backend-частина FlowGuard має багаторівневу структуру. Перший рівень – API Layer або рівень контролерів. Контролери приймають HTTP-запити від frontend-частини, перевіряють маршрут і передають дані до відповідного сервісу. Другий рівень – Service Layer, який містить основну бізнес-логіку системи. Третій рівень – Persistence Layer або рівень репозиторіїв.

Окремо виділяється рівень Domain / Entities. Він містить основні сутності предметної галузі: User, UserProfile, Category, Transaction, Budget, Goal, FinancialInsight та AuditLog. Додатково в backend-частині присутні модулі безпеки, DTO та mapper.

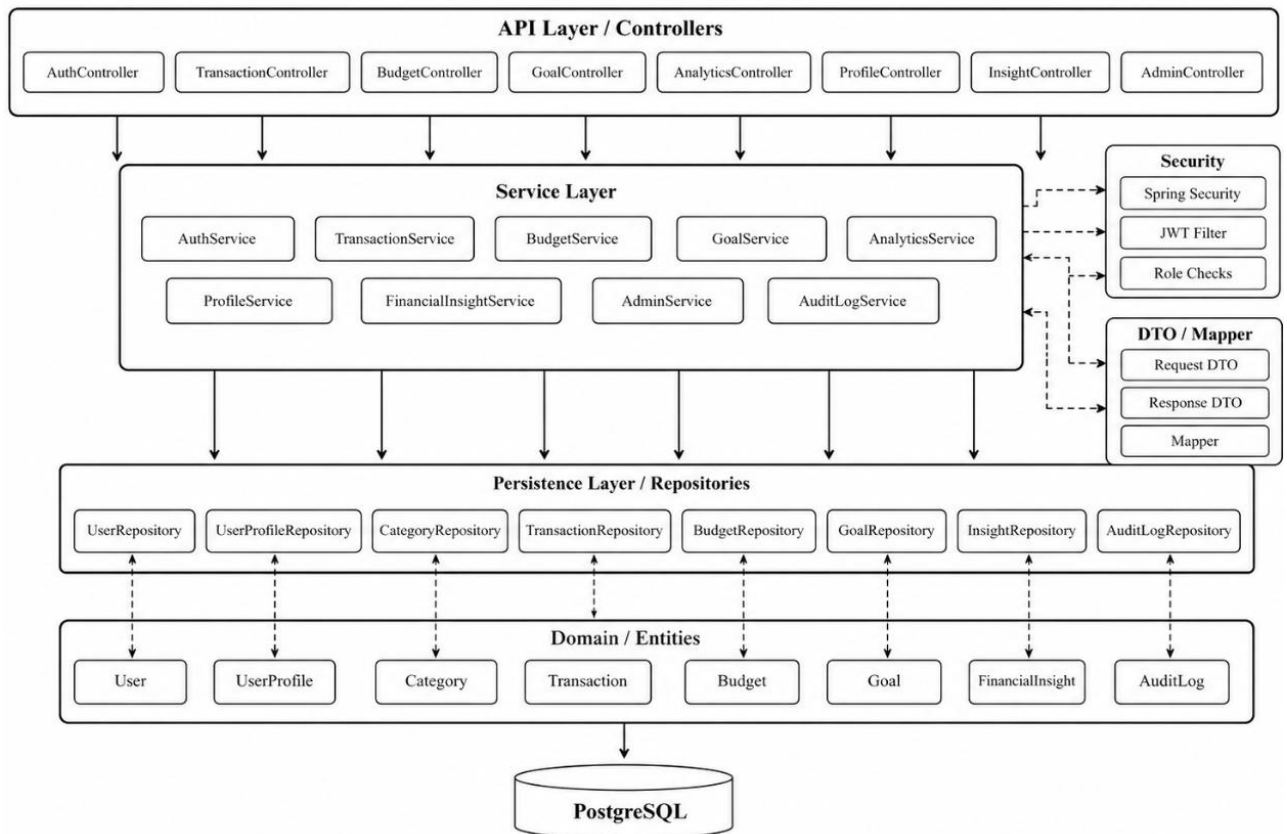


Рис. 2.8 Діаграма структури backend-частини web-застосунку FlowGuard

2.8 Діаграма структури frontend-частини FlowGuard

Frontend-частина FlowGuard побудована за компонентним принципом. Це дозволяє розділити інтерфейс на окремі сторінки, компоненти, сервіси та елементи навігації. Основний рівень frontend-частини – це React-застосунок, який містить маршрутизацію, сторінки, компоненти, сервіси для роботи з API та локальне збереження токена користувача.

Рівень Pages містить основні сторінки системи: LandingPage / LoginPage, DashboardPage, TransactionsPage, BudgetsPage, GoalsPage, AnalyticsPage, ProfilePage та AdminPage. Рівень UI Components містить повторно використовувані елементи інтерфейсу: форми операцій, картки бюджетів, картки цілей, графіки, віджет «Фінансовий Фокус», форму профілю, таблицю адміністратора та бейджі ризику.

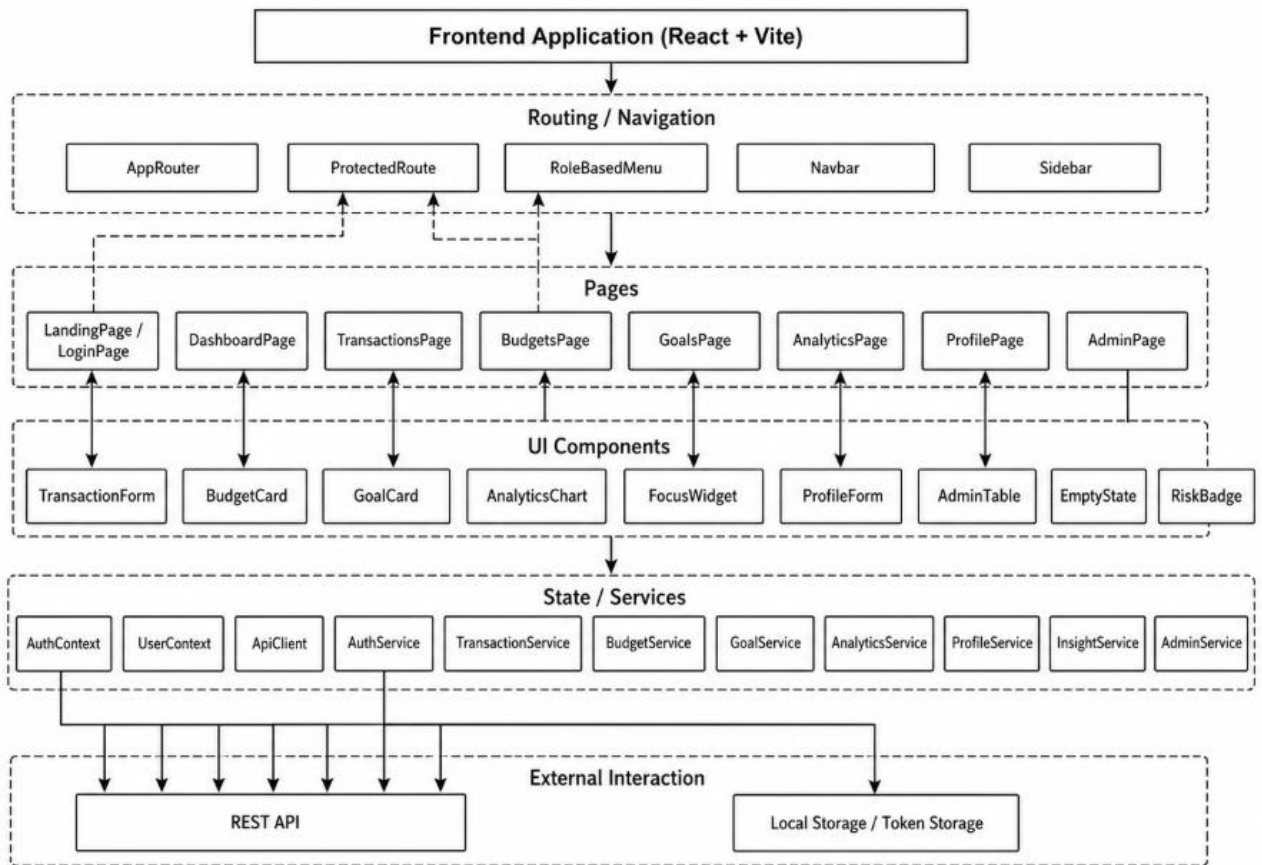


Рис. 2.9 Діаграма структури frontend-частини web-застосунку FlowGuard

Отже, у другому розділі було визначено вимоги до web-застосунку FlowGuard, обрано технічні засоби реалізації, описано архітектуру системи, рольову модель доступу, структуру бази даних, модуль «Фінансовий Фокус», а також структуру backend- та frontend-частин.

3 РОЗРОБКА WEB-ЗАСТОСУНКУ FLOWGUARD ДЛЯ УПРАВЛІННЯ БЮДЖЕТОМ ТА ОСОБИСТИМИ ВИТРАТАМИ

3.1 Проєктування інтерфейсу користувача

Інтерфейс web-застосунку FlowGuard спроектовано як особистий фінансовий кабінет користувача. Основна мета інтерфейсу полягає в тому, щоб користувач міг швидко отримати доступ до ключової фінансової інформації, додавати операції, контролювати бюджетні ліміти, переглядати фінансові цілі та аналізувати власні витрати.

Під час проєктування інтерфейсу було враховано, що застосунок повинен бути зрозумілим для користувачів без спеціальних технічних знань. Тому структура сторінок побудована за принципом розділення основних функцій на окремі розділи. Це дозволяє не перевантажувати одну сторінку великою кількістю форм, таблиць і графіків.

- сторінка входу та реєстрації;
- сторінка Dashboard / Огляд;
- сторінка фінансових операцій;
- сторінка бюджетних лімітів;
- сторінка фінансових цілей;
- сторінка аналітики;
- сторінка профілю користувача;
- адміністративна панель.

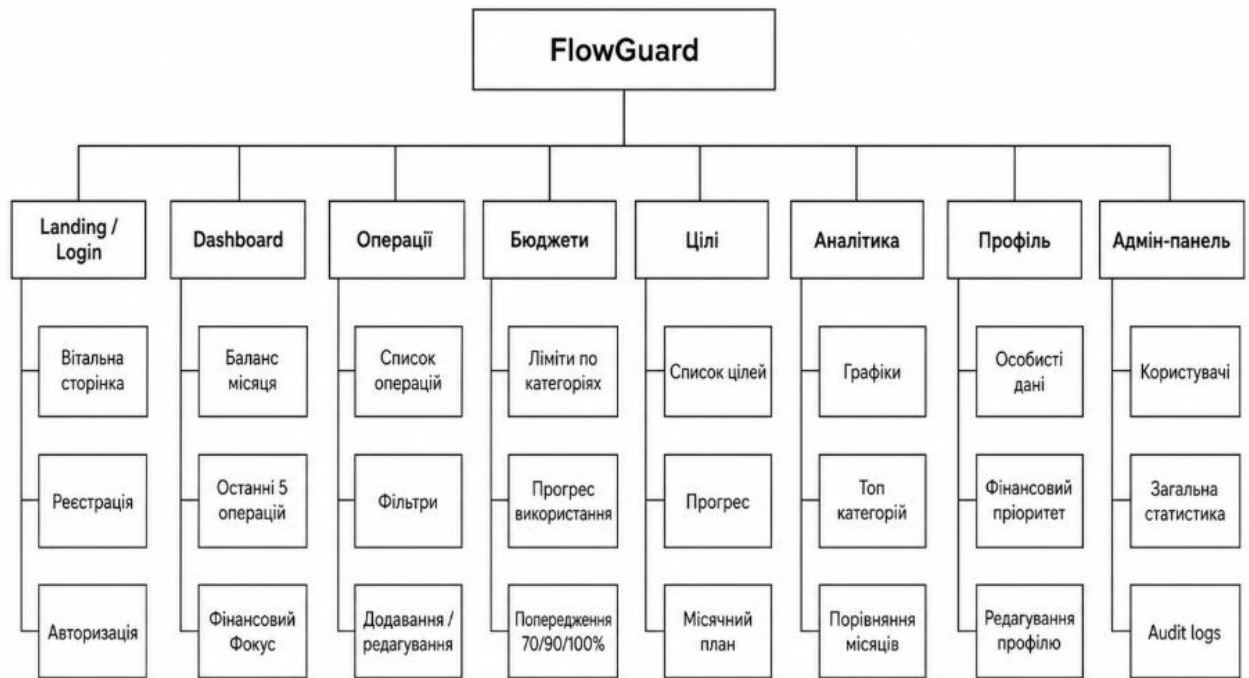


Рис. 3.1 Структура сторінок web-застосунку FlowGuard

Сторінка входу використовується для авторизації користувача. Користувач вводить email і пароль, після чого система перевіряє облікові дані та визначає роль користувача.

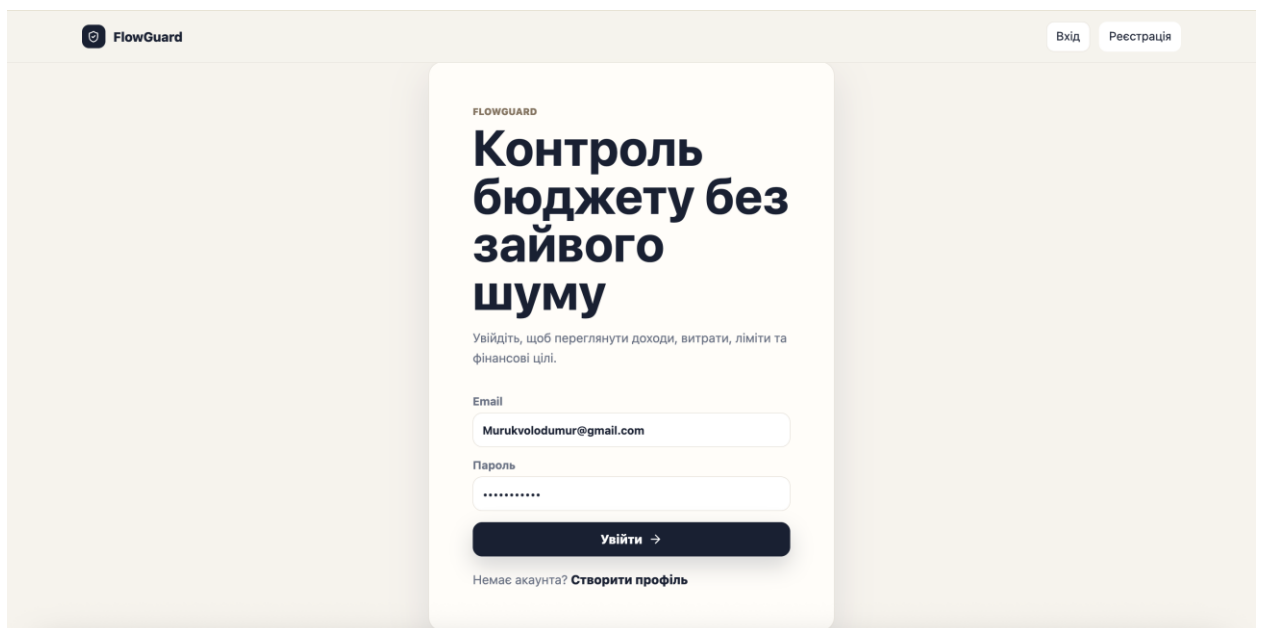


Рис. 3.2 Сторінка входу користувача у web-застосунку FlowGuard

Головна сторінка Dashboard виконує роль короткого фінансового огляду. На ній відображаються основні показники: загальний баланс, доходи та витрати за місяць, останні фінансові операції та блок персональних підказок «Фінансовий Фокус».

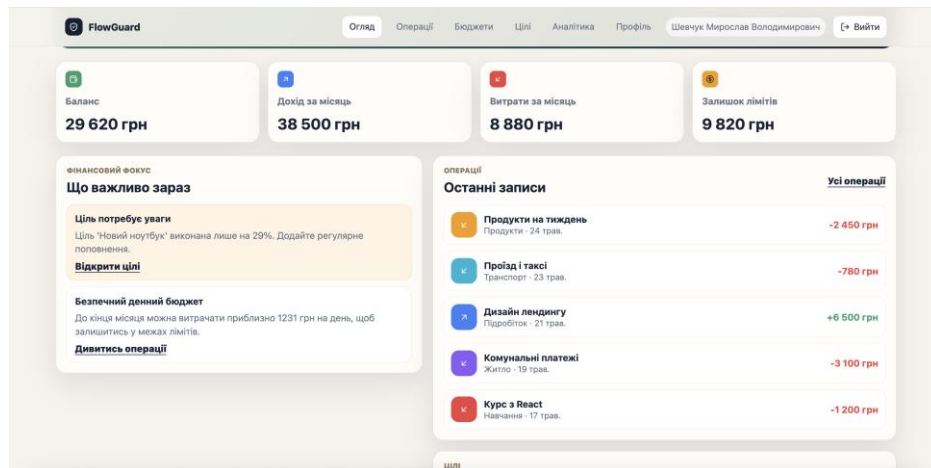


Рис. 3.3 Головна сторінка Dashboard з блоком “Фінансовий Фокус”

Сторінка операцій призначена для повної роботи з доходами та витратами. На ній користувач може додавати нові фінансові операції, редагувати записи, видаляти операції та фільтрувати дані за типом, категорією, датою або сумою.

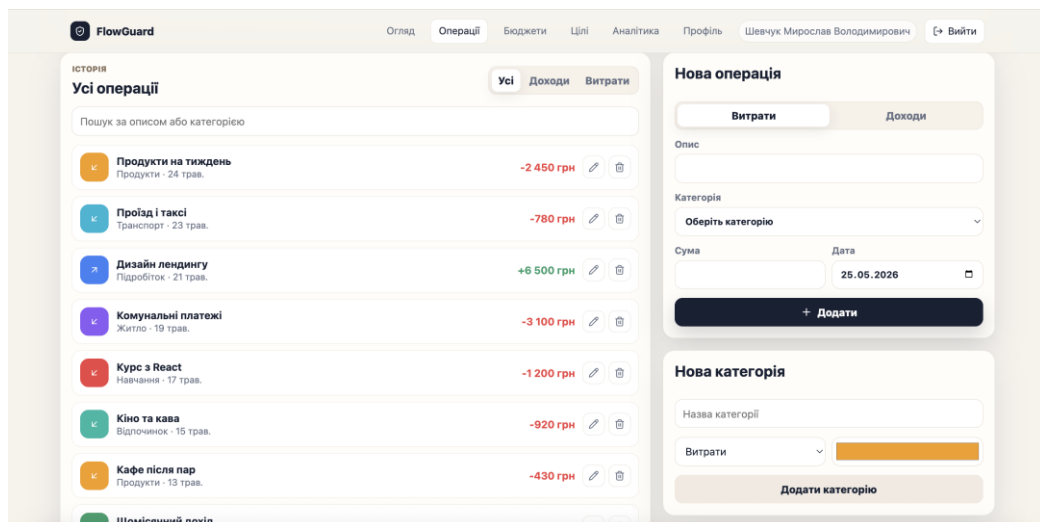


Рис. 3.4 Сторінка фінансових операцій у web-застосунку FlowGuard

Сторінка бюджетів використовується для керування лімітами. Користувач може встановити місячні обмеження для окремих категорій витрат, а система показує прогрес використання кожного ліміту.

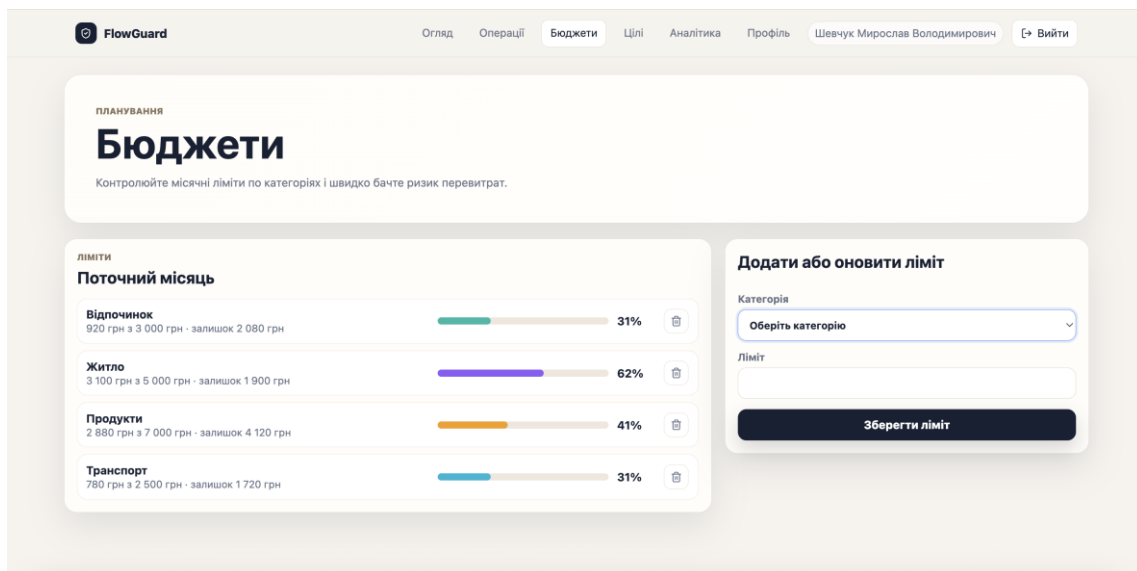


Рис. 3.5 Сторінка бюджетних лімітів у web-застосунку FlowGuard

Сторінка фінансових цілей дозволяє користувачу створювати цілі накопичення, зазначати назву, цільову суму, поточний прогрес і дату досягнення.

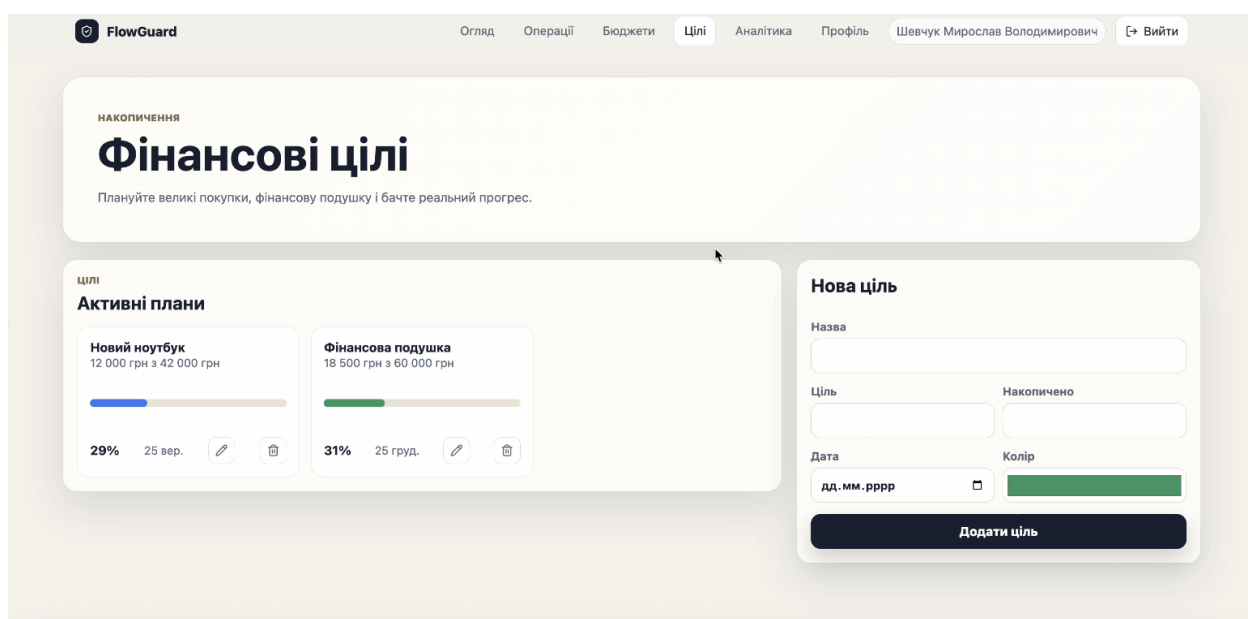


Рис. 3.6 Сторінка фінансових цілей у web-застосунку FlowGuard

Сторінка аналітики містить графіки та фінансові показники. Користувач може переглянути структуру витрат за категоріями, динаміку доходів і витрат, порівняння поточного місяця з попереднім, а також найбільш витратні напрями.

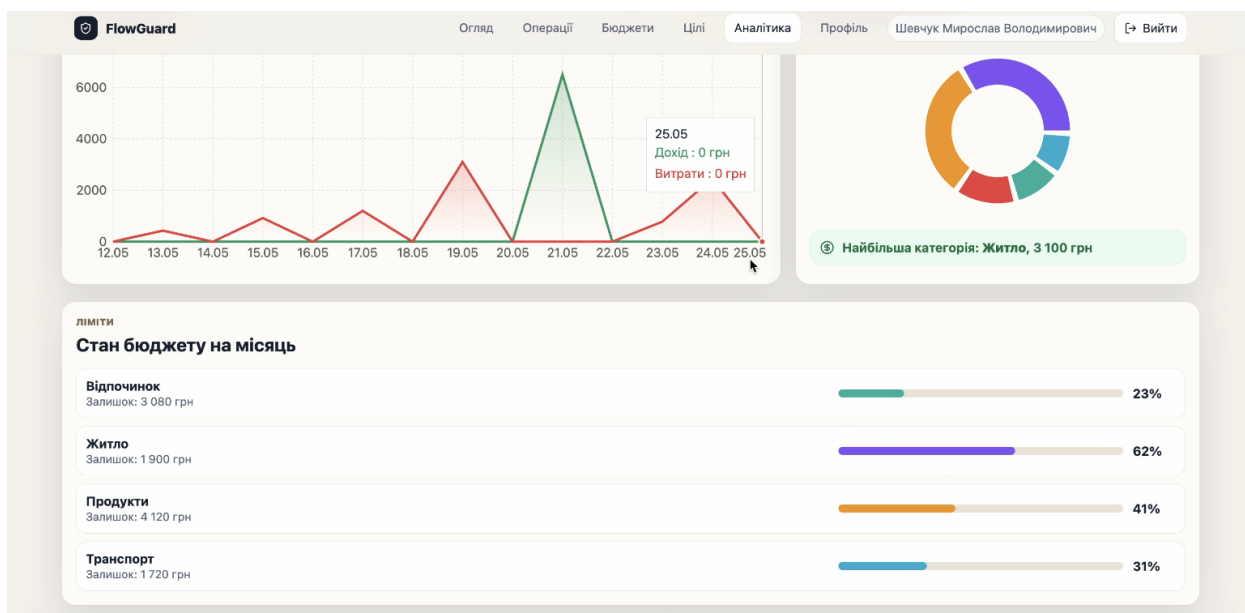


Рис. 3.7 Сторінка аналітики бюджету у web-застосунку FlowGuard

Сторінка профілю містить персональні дані користувача: ім'я, email, телефон, місто, місячний дохід, бажану валюту та фінансовий пріоритет.

The screenshot displays the 'Профіль' (Profile) page in the FlowGuard web application. The top navigation bar is identical to the analytics page.

The main content area features:

- Profile Header:** Displays the user's name 'Шевчук Мирослав Володимирович' and a prompt to edit contact data, financial priorities, and personalization parameters.
- Personal Information Section:** Includes fields for:
 - Ім'я** (Name): Шевчук Мирослав Володимирович
 - Email**: Murukvolodumur@gmail.com
 - Телефон** (Phone): +380 67 420 18 25
 - Місто** (City): Київ
 - Місячний дохід** (Monthly income): 38500
 - Валюта** (Currency): UAH
 - Фінансовий пріоритет** (Financial priority): [Field is present but empty]
 - Колір профілю** (Profile color): [Field is present but empty]
- Account Information Section:** Titled 'Дані та фінанси' (Data and Finance), it includes:
 - Email**: Murukvolodumur@gmail.com
 - Роль** (Role): USER
 - Створено** (Created): 25.05.2026
 - Баланс** (Balance): 29 620 грн
 - Доходи місяця** (Monthly income): 38 500 грн

Рис. 3.8 Сторінка профілю користувача у web-застосунку FlowGuard

Адміністративна панель призначена для користувача з роллю ADMIN. У цьому розділі адміністратор може переглядати список користувачів, загальну статистику платформи, системні попередження та журнал дій.

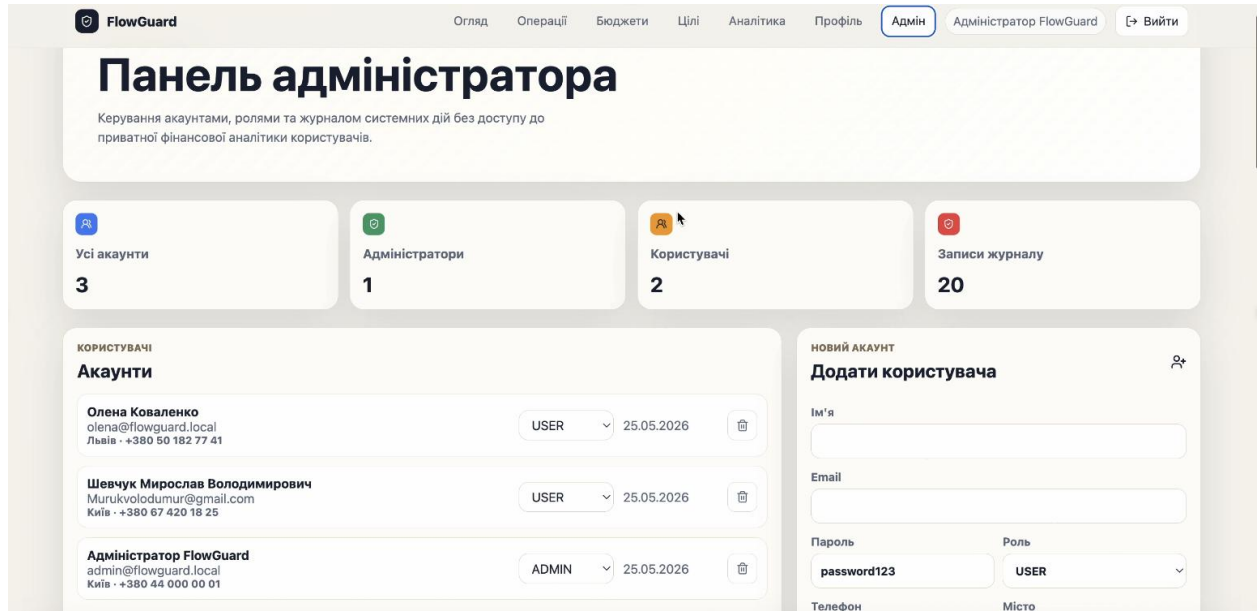


Рис. 3.9 Адміністративна панель web-застосунку FlowGuard

3.2 Реалізація серверної частини

Серверна частина web-застосунку FlowGuard реалізована за допомогою Java та Spring Boot. Backend відповідає за обробку HTTP-запитів, перевірку прав доступу, виконання бізнес-логіки, роботу з базою даних і формування відповідей для клієнтської частини [17; 18; 19].

- рівень контролерів;
- рівень сервісів;
- рівень репозиторіїв;
- рівень сутностей;
- рівень DTO;
- модуль безпеки.

Контролери приймають запити від frontend-частини та передають дані до відповідних сервісів. Наприклад, AuthController відповідає за реєстрацію та

авторизацію, TransactionController – за роботу з фінансовими операціями, BudgetController – за бюджетні ліміти, GoalController – за фінансові цілі, ProfileController – за профіль користувача, а AdminController – за адміністративні функції.

Сервісний рівень містить основну бізнес-логіку. У сервісах виконується перевірка даних, обробка операцій, розрахунок аналітики, формування фінансових підказок і перевірка власника записів. Такий підхід дозволяє не розміщувати бізнес-логіку безпосередньо в контролерах, а зберігати її в окремих класах.

Репозиторії відповідають за взаємодію з базою даних через Spring Data JPA. Для передачі даних між frontend і backend використовуються DTO, які дозволяють відокремити внутрішню структуру сутностей бази даних від даних, які передаються користувачу.

- модуль авторизації;
- модуль профілю користувача;
- модуль категорій;
- модуль фінансових операцій;
- модуль бюджетів;
- модуль фінансових цілей;
- модуль аналітики;
- модуль «Фінансовий Фокус»;
- адміністративний модуль;
- модуль audit logs.

3.3 Реалізація клієнтської частини

Клієнтська частина FlowGuard реалізована за допомогою React та Vite. Вона відповідає за відображення сторінок, роботу форм, взаємодію з користувачем, навігацію та надсилення запитів до backend-частини [21; 22].

Frontend-частина має компонентну структуру. Це означає, що інтерфейс поділяється на окремі повторно використовувані елементи. Наприклад, форми додавання операцій, картки бюджетів, блоки статистики, графіки, таблиці та віджет «Фінансовий Фокус» можуть бути реалізовані як окремі компоненти.

- LoginPage;
- DashboardPage;
- TransactionsPage;
- BudgetsPage;
- GoalsPage;
- AnalyticsPage;
- ProfilePage;
- AdminPage.

Для маршрутизації використовується система routes. Доступ до приватних сторінок обмежується за допомогою protected routes. Якщо користувач не авторизований, він не може відкрити сторінки особистого кабінету. Якщо користувач не має ролі ADMIN, йому не відображається адміністративна панель.

3.4 Реалізація модуля “Фінансовий Фокус”

Модуль «Фінансовий Фокус» є однією з ключових особливостей web-застосунку FlowGuard. Його призначення полягає в тому, щоб автоматично аналізувати фінансові дані користувача та формувати короткі персональні підказки.

На відміну від звичайної аналітики, яка просто показує графіки та числові показники, «Фінансовий Фокус» виділяє найважливіший фінансовий ризик або пораду. Це дозволяє користувачу не витрачати час на самостійний аналіз усіх таблиць і графіків, а одразу побачити, на що потрібно звернути увагу.

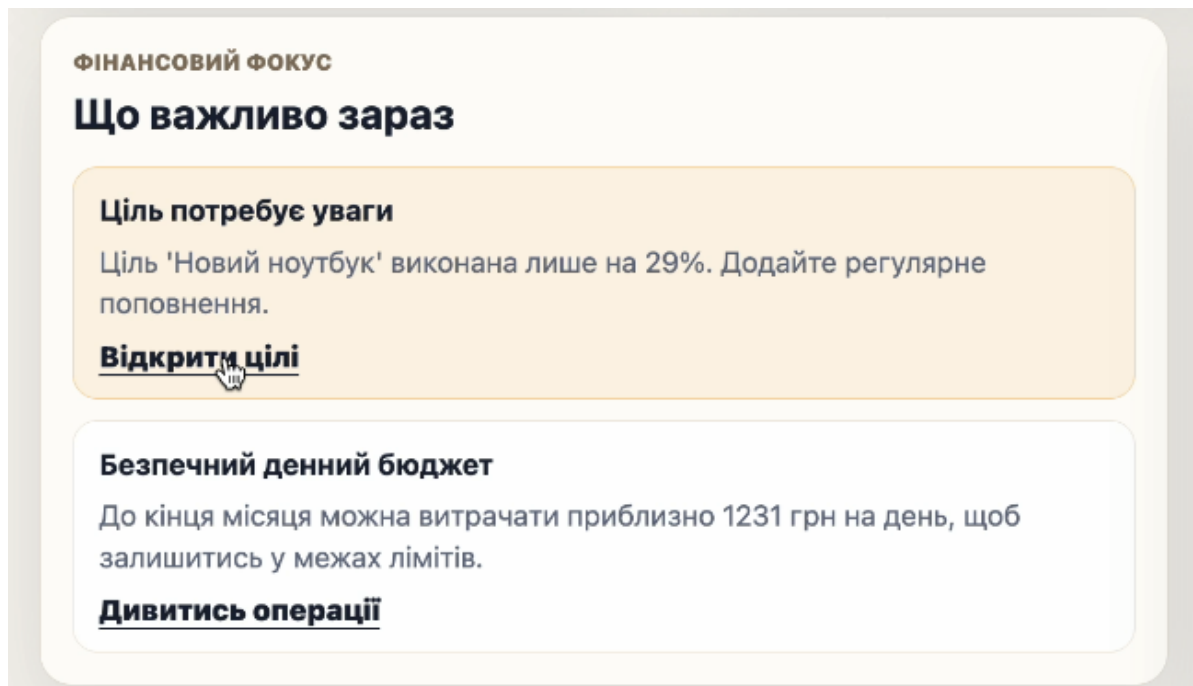


Рис. 3.10 Приклад роботи модуля “Фінансовий Фокус” у web-застосунку FlowGuard

- фінансові операції користувача;
- бюджетні ліміти;
- фінансові цілі;
- витрати за поточний місяць;
- витрати за попередній місяць;
- залишок бюджету до кінця місяця.

На backend-рівні логіка модуля реалізується у сервісі FinancialInsightService. Він отримує дані з репозиторіїв операцій, бюджетів і цілей, застосовує правила аналізу та формує результат. Frontend отримує цей результат через endpoint `/api/insights/focus` і відображає його на dashboard.

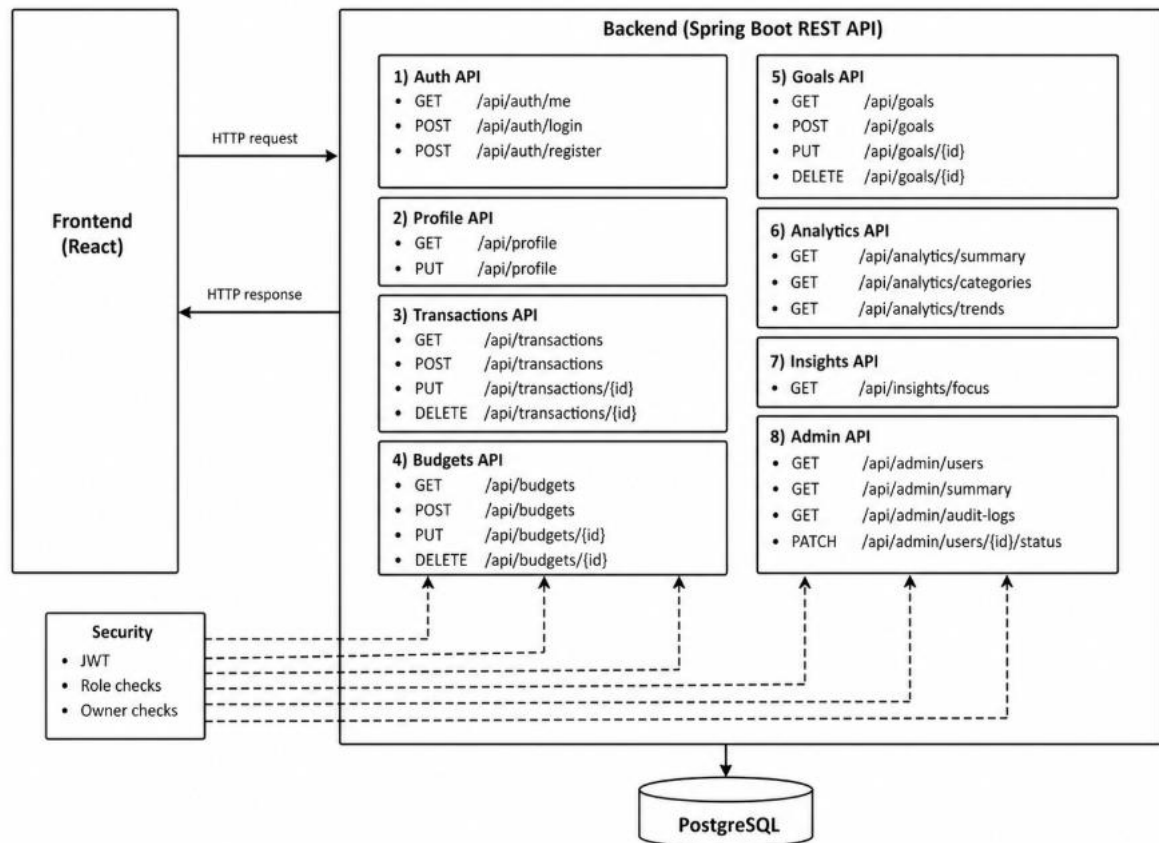


Рис. 3.11 Схема API web-застосунку FlowGuard

3.5 Реалізація адміністративної панелі

Адміністративна панель FlowGuard призначена для користувача з роллю ADMIN. Вона дозволяє переглядати загальну інформацію про роботу системи та виконувати базові адміністративні дії. Адміністратор має доступ до окремого розділу, який недоступний звичайному користувачу.

- перегляд списку користувачів;
- перегляд загальної статистики платформи;
- перегляд журналу дій;
- перегляд системних попереджень;
- зміна ролі користувача;
- блокування або розблокування демо-користувачів.

Адміністративна панель не призначена для прямого керування особистими фінансовими операціями користувачів. Її основна мета – контроль

стану системи, перегляд агрегованої інформації та демонстрація можливостей адміністрування.

3.6 Завантаження проєкту на GitHub

Для збереження коду та контролю версій використовується Git і GitHub. Репозиторій проєкту містить frontend- і backend-частини, що дозволяє зручно підтримувати обидві частини системи в одному середовищі [24].

- backend – серверна частина Spring Boot;
- frontend – клієнтська частина React;
- README.md – інструкція із запуску;
- database або migrations – файли міграцій бази даних;
- docs – додаткові матеріали або діаграми.

3.7 Тестування web-застосунку

Тестування web-застосунку FlowGuard виконувалося для перевірки коректності основної backend-логіки, захищених API, рольового доступу та роботи модуля «Фінансовий Фокус». Для цього були реалізовані модульні та інтеграційні тести [8; 23].

- JUnit;
- Mockito;
- Spring Boot Test;
- MockMvc;
- H2 in-memory database;
- Spring Security Test.

Модульне тестування використовувалося для перевірки окремих сервісів у ізоляції. У межах модульного тестування були перевірені сервіси авторизації, профілю та фінансових підказок. Інтеграційне тестування використовувалося для перевірки спільної роботи компонентів системи,

запуску Spring Boot application context, повного користувацького сценарію через HTTP API, JWT-авторизації та рольового доступу.

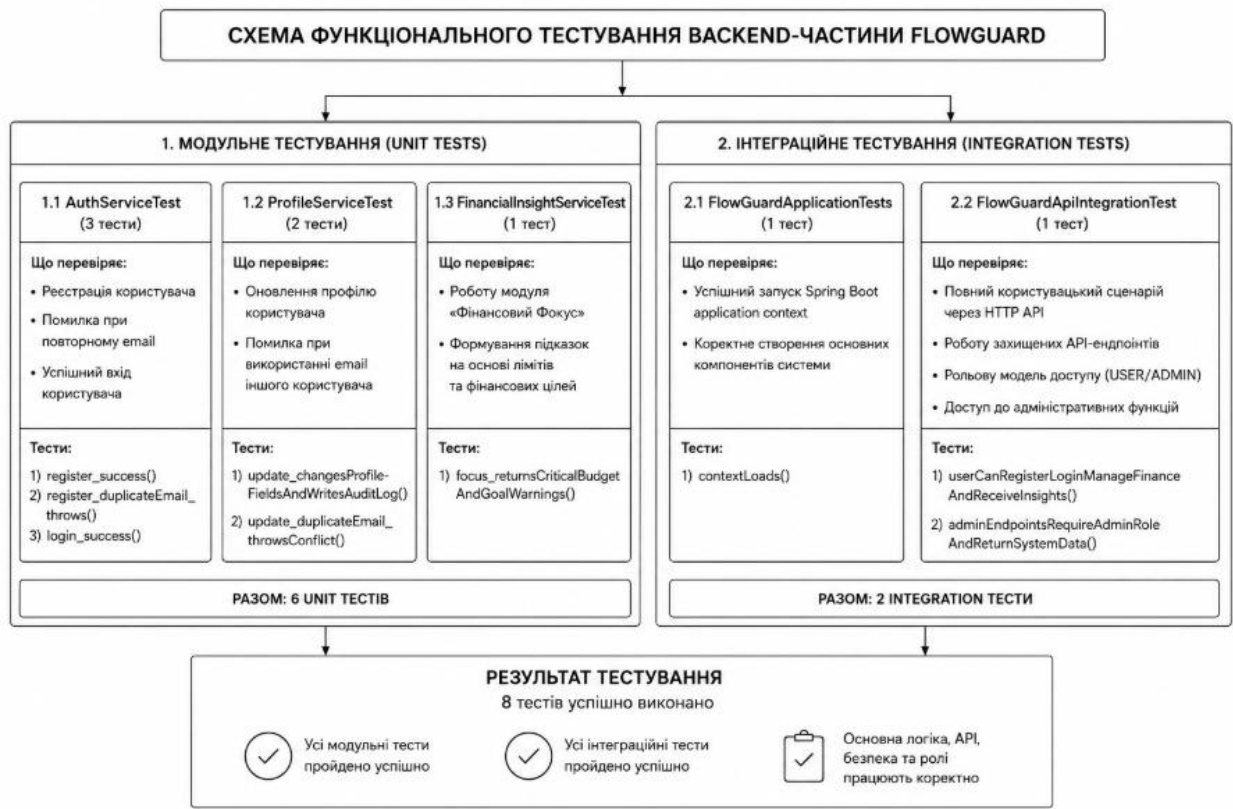


Рис. 3.12 Схема функціонального тестування backend-частини web-застосунку FlowGuard

FlowGuardApplicationTests. Клас AuthServiceTest перевіряє логіку реєстрації та входу користувача. Клас ProfileServiceTest перевіряє оновлення профілю користувача. Клас FinancialInsightServiceTest перевіряє роботу модуля «Фінансовий Фокус». Клас FlowGuardApiIntegrationTest перевіряє повний backend-сценарій через HTTP API.

Окремо в інтеграційному тестуванні перевіряється рольова модель доступу. Звичайний користувач не повинен мати доступ до адміністративних endpoint-ів. У такому випадку система повертає статус 403 Forbidden. Адміністратор, навпаки, отримує доступ до /api/admin/summary, /api/admin/users та /api/admin/audit-logs.

За результатами тестування було виконано 8 тестів. Усі тести пройшли успішно, що підтверджує коректну роботу основної backend-логіки, авторизації, профілю, фінансових підказок, API-запитів, role-based access control та адміністративних функцій.

✓ service (com.example.flowguard)	3 sec 381 ms
✓ AuthServiceTest	2 sec 703 ms
✓ register_duplicateEmail_throws()	2 sec 579 ms
✓ login_success()	108 ms
✓ register_success()	16 ms
✓ ProfileServiceTest	147 ms
✓ update_duplicateEmail_throwsConflict()	135 ms
✓ update_changesProfileFieldsAndWritesAuditLog()	12 ms
✓ FinancialInsightServiceTest	531 ms
✓ focus_returnsCriticalBudgetAndGoalWarnings()	531 ms
✓ FlowGuardApiIntegrationTest (com.example.flowguard)	1 sec 825 ms
✓ userCanRegisterLoginManageFinanceAndReceiveInsights()	1 sec 228 ms
✓ adminEndpointsRequireAdminRoleAndReturnSystemData()	597 ms

Рис. 3.13 Результати виконання тестів backend-частини web-застосунку FlowGuard

Отже, у третьому розділі було описано програмну реалізацію web-застосунку FlowGuard. Розглянуто інтерфейс користувача, серверну та клієнтську частини, роботу модуля «Фінансовий Фокус», адміністративну панель, структуру репозиторію та результати тестування.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено web-застосунок FlowGuard для управління бюджетом та особистими витратами. Метою роботи було спрощення процесу контролю особистих фінансів за рахунок створення системи, яка дозволяє користувачу вести облік доходів і витрат, контролювати бюджетні ліміти.

У першому розділі було проведено аналіз предметної галузі управління особистими фінансами. Визначено, що основними проблемами користувачів є ручне ведення фінансових записів, складність контролю витрат, відсутність єдиного середовища для збереження фінансових даних, недостатня наочність аналітики та ризик несвоєчасного виявлення перевищення бюджетних лімітів.

Також було проаналізовано існуючі програмні засоби для управління особистими фінансами, зокрема Monefy, YNAB, Spendee та Money Manager. У результаті аналізу визначено, що більшість аналогів підтримують базовий облік доходів і витрат, категорії та графіки, однак мають певні обмеження.

У другому розділі було виконано проєктування web-застосунку FlowGuard. Сформовано функціональні та нефункціональні вимоги до системи. Для реалізації застосунку було обрано клієнт-серверну архітектуру. Серверна частина реалізується на основі Java та Spring Boot, клієнтська частина – за допомогою React і Vite, а для збереження даних використовується PostgreSQL.

Окремо було спроектовано рольову модель доступу. У системі передбачено три типи учасників: гість, користувач та адміністратор. Гість має доступ лише до сторінок входу та реєстрації. Користувач працює з власними фінансовими даними, операціями, бюджетами, цілями, аналітикою та профілем. Адміністратор має всі можливості користувача, а також доступ до списку користувачів, загальної статистики, журналу дій та системних попереджень.

У третьому розділі було описано програмну реалізацію web-застосунку FlowGuard. Розглянуто структуру інтерфейсу користувача, серверну частину, клієнтську частину, адміністративну панель, GitHub-репозиторій і тестування системи.

Ключовою особливістю розробленого застосунку є модуль «Фінансовий Фокус». Він аналізує фінансові операції, бюджетні ліміти та фінансові цілі користувача, після чого формує короткі персональні підказки. Модуль може визначати наближення до перевищення ліміту, критичний рівень витрат, низький залишок бюджету.

Для перевірки працездатності системи було проведено модульне та інтеграційне тестування backend-частини. Тестування охоплювало авторизацію, реєстрацію, оновлення профілю, формування фінансових підказок, роботу REST API, рольовий доступ користувача й адміністратора. Загалом було виконано 8 тестів, які підтвердили коректну роботу основної логіки застосунку.

Отже, у результаті виконання кваліфікаційної роботи було створено web-застосунок FlowGuard, який забезпечує комплексне управління особистими фінансами.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Шевчук М.В., Корецька В.О. Розробка системи управління бюджетом для особистих витрат. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.528-531.

2. Шевчук М.В., Корецька В.О. Веб-застосунок "FlowGuard" для управління бюджетом та особистими витратами. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2025, С.118-119.

ПЕРЕЛІК ПОСИЛАНЬ

1. Фінансова грамотність. Фінанси. Що? Чому? Як? : навч. посіб. / авт. кол. Київ, 2019. 272 с. URL: https://events.bank.gov.ua/gmw2021/src/files/Financial_Literacy_Textbook_Aug2019.pdf.
2. Фінансова грамотність, фінансова інклюзія та фінансовий добробут в Україні у 2021 році : звіт за результатами дослідження / Національний банк України. Київ, 2021. URL: https://bank.gov.ua/admin_uploads/article/Research_Financial_Literacy_Inclusion_Welfare_2021.pdf.
3. Національна стратегія розвитку фінансової грамотності до 2030 року / Національний банк України. Київ, 2024. URL: <https://talan.bank.gov.ua/uploads/files/nbu-strategy-fingramotnosti-2030-compressed.pdf?v=6>.
4. Гончаренко І. Г., Бережна Л. В., Березіна С. Б. Інтерактивний посібник з фінансової грамотності : навч. посіб. Черкаси : ЧДТУ, 2024. URL: https://er.chdtu.edu.ua/bitstream/ChSTU/5471/1/Інтерактивний_посібник_Гончаренко_Бережна_Березіна.pdf.
5. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI : станом на 01.01.2026 р. URL: <https://zakon.rada.gov.ua/laws/show/2297-17>.
6. Про електронні довірчі послуги : Закон України від 05.10.2017 р. № 2155-VIII : станом на 01.01.2026 р. URL: <https://zakon.rada.gov.ua/laws/show/2155-19>.
7. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмного забезпечення. Вимоги до якості та оцінювання систем і програмного забезпечення. Моделі якості системи та програмного забезпечення. Київ : ДП «УкрНДНЦ», 2016.
8. ДСТУ ISO/IEC/IEEE 29119-1:2017. Інженерія програмного забезпечення та систем. Тестування програмного забезпечення. Частина 1. Поняття та визначення. Київ : ДП «УкрНДНЦ», 2017.
9. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

10. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2020. 705 p.
11. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
12. Monefy | Budget & Track Your Money : офіційний сайт додатку Monefy. URL: <https://monefy.com/> (дата звернення: 25.05.2026).
13. Wallet by BudgetBakers : офіційний сайт сервісу управління особистими фінансами. URL: <https://web.budgetbakers.com/> (дата звернення: 25.05.2026).
14. YNAB : офіційний сайт сервісу You Need A Budget. URL: <https://www.ynab.com/> (дата звернення: 25.05.2026).
15. Spendee: Money Manager & Budget Planner : офіційний сайт сервісу Spendee. URL: <https://www.spendee.com/> (дата звернення: 25.05.2026).
16. Money Manager Expense & Budget : офіційний сайт додатку Money Manager. URL: <https://www.realbyteapps.com/> (дата звернення: 25.05.2026).
17. Java Documentation / Oracle. URL: <https://docs.oracle.com/en/java/> (date of access: 25.05.2026).
18. Spring Boot Reference Documentation / VMware. URL: <https://docs.spring.io/spring-boot/reference/> (date of access: 25.05.2026).
19. Spring Security Reference Documentation / VMware. URL: <https://docs.spring.io/spring-security/reference/> (date of access: 25.05.2026).
20. PostgreSQL Documentation / The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/> (date of access: 25.05.2026).
21. React Documentation / Meta Open Source. URL: <https://react.dev/> (date of access: 25.05.2026).
22. Vite Documentation / Vite Team. URL: <https://vite.dev/guide/> (date of access: 25.05.2026).
23. JUnit 5 User Guide / JUnit Team. URL: <https://junit.org/junit5/docs/current/user-guide/> (date of access: 25.05.2026).
24. Mockito Documentation / Mockito Team. URL: <https://site.mockito.org/> (date of access: 25.05.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Веб-застосунок "FlowGuard" для управління бюджетом та особистими витратами

Виконав студент 4 курсу
групи ПД-43
Шевчук Мирослав Володимирович
Керівник роботи
канд.пед.наук, проф. Корецька Вікторія Олександрівна

Київ – 2026

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі управління особистими фінансами.
2. Дослідити існуючі програмні засоби для ведення бюджету та контролю витрат.
3. Визначити проблеми ручного обліку доходів, витрат, лімітів і фінансових цілей.
4. Сформулювати функціональні та нефункціональні вимоги до web-застосунку.
5. Спроектувати архітектуру, базу даних та рольову модель доступу.
6. Реалізувати web-застосунок FlowGuard з ролями користувача, адміністратора та гостя.
7. Провести тестування основних функцій системи.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесу управління особистими фінансами шляхом розробки web-застосунку FlowGuard.

Об'єкт дослідження: процес управління особистим бюджетом, доходами, витратами та фінансовими цілями користувача.

Предмет дослідження: засоби, методи та технології реалізації web-застосунку для управління бюджетом та особистими витратами.

3

АНАЛІЗ АНАЛОГІВ

Показник	Monefy	YNAB	Spendee	Money Manager	FlowGuard
Власні категорії	Доступно переважно у платній версії	+	+	+	+
Бюджетні ліміти	Є лише базовий контроль витрат	+	+	+	+
Фінансові цілі	Немає повноцінного модуля цілей	+	+	Є базове планування без розширеної логіки	+
Аналітика та звіти	Базова статистика без глибокого аналізу	+	+	Середній рівень аналітики	+
Фільтрація операцій	Обмежена у безкоштовній версії	+	+	Є базова фільтрація	+
Web-версія	-	+	+	-	+
Адміністративна панель	-	-	-	-	+
Персональні підказки	-	Є поради з бюджетування, але в межах методики YNAB	Є базові підказки, але без пріоритетного фінансового фокусу	-	+
Модуль "Фінансовий Фокус"	-	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Реєстрація та авторизація користувача.
2. Додавання, редагування та видалення фінансових операцій.
3. Розподіл доходів і витрат за категоріями.
4. Створення бюджетних лімітів за категоріями.
5. Відстеження фінансових цілей.
6. Перегляд аналітики бюджету.
7. Редагування профілю користувача.
8. Отримання підказок “Фінансовий Фокус”.
9. Адміністративний перегляд користувачів, статистики та audit logs.

Нефункціональні вимоги

1. Завантаження сторінок — до 3 секунд.
2. Відповідь стандартного API-запиту — до 2 секунд.
3. Додавання витрати — за 3–5 дій користувача.
4. Підтримка не менше 1000 користувачів.
5. Адаптивність інтерфейсу від 360 рх.
6. Доступ до даних лише після авторизації.
7. Збереження даних у PostgreSQL.

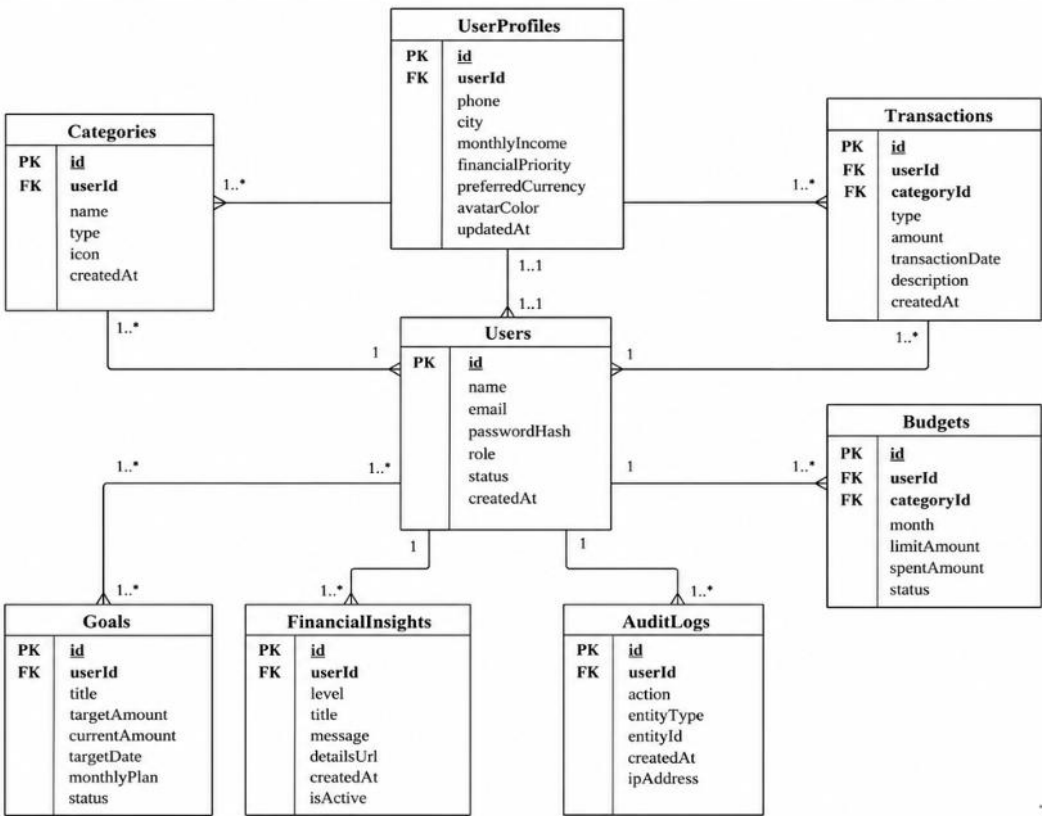
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



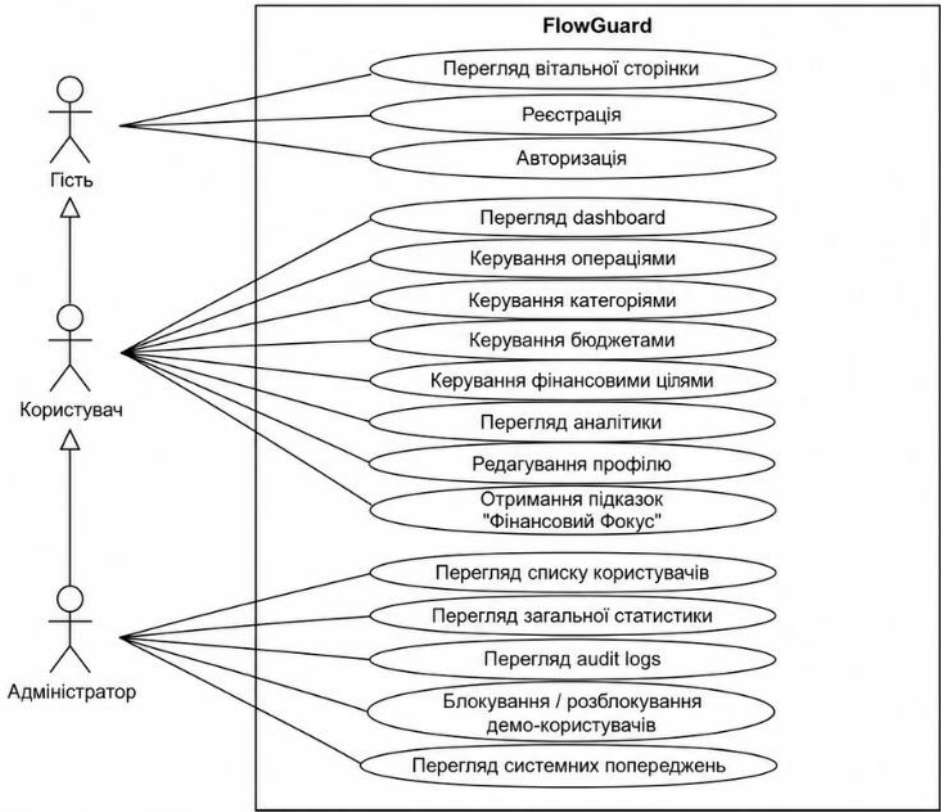
6

ER-ДІАГРАМА БАЗИ ДАНИХ



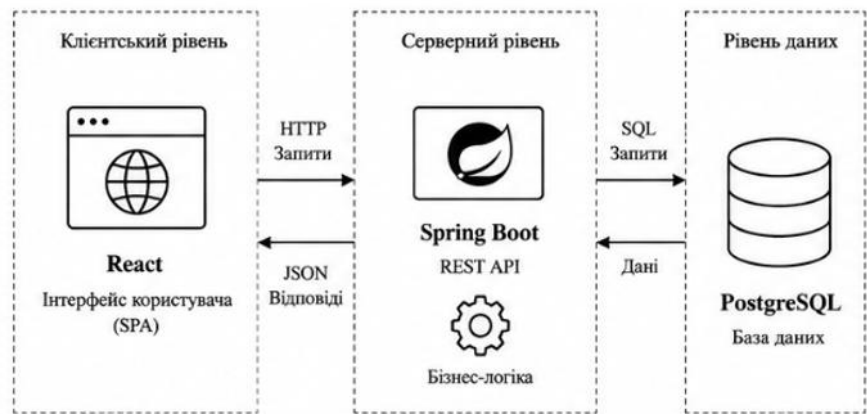
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



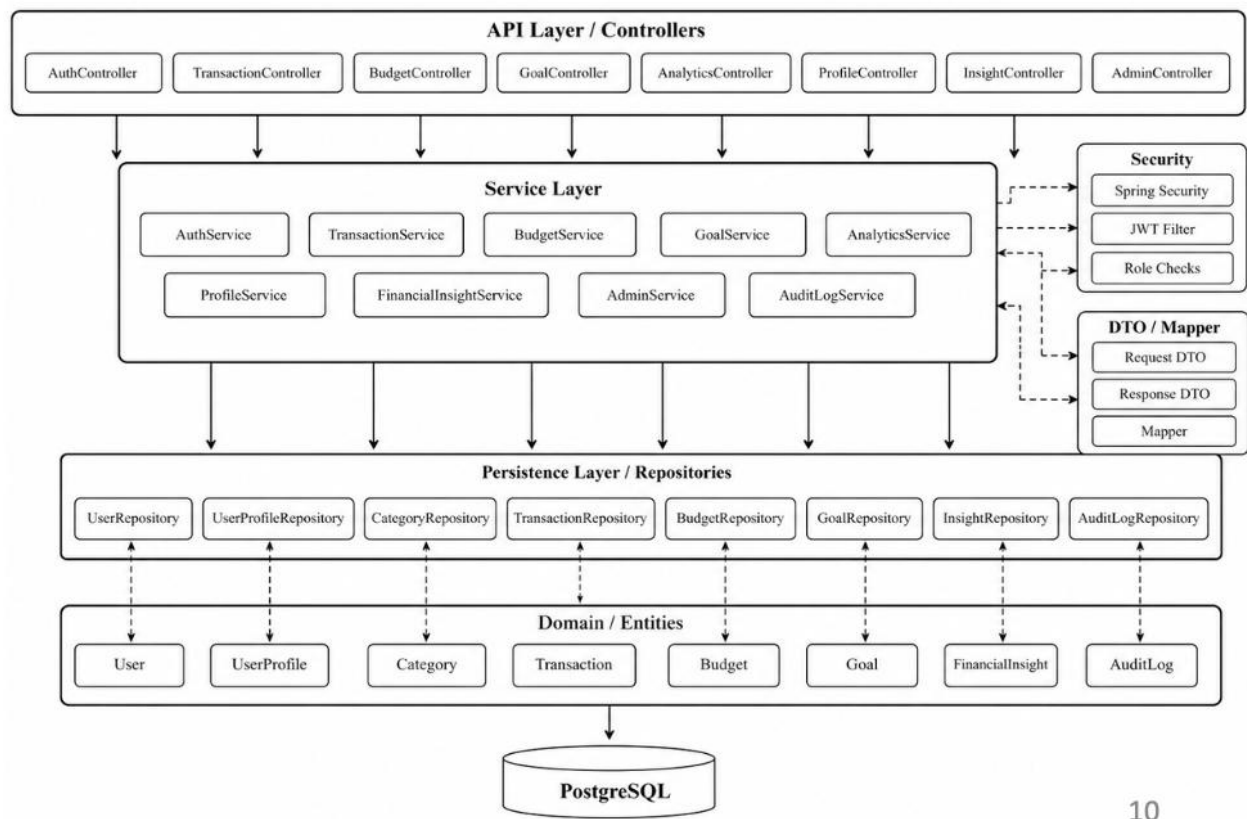
8

ЗАГАЛЬНА СХЕМА ТЕХНОЛОГІЧНОГО СТЕКУ



9

ДІАГРАМА СТРУКТУРИ BACKEND-ЧАСТИНИ



10

ДІАГРАМА ПОСЛІДОВНОСТІ АВТОРИЗАЦІЇ КОРИСТУВАЧА

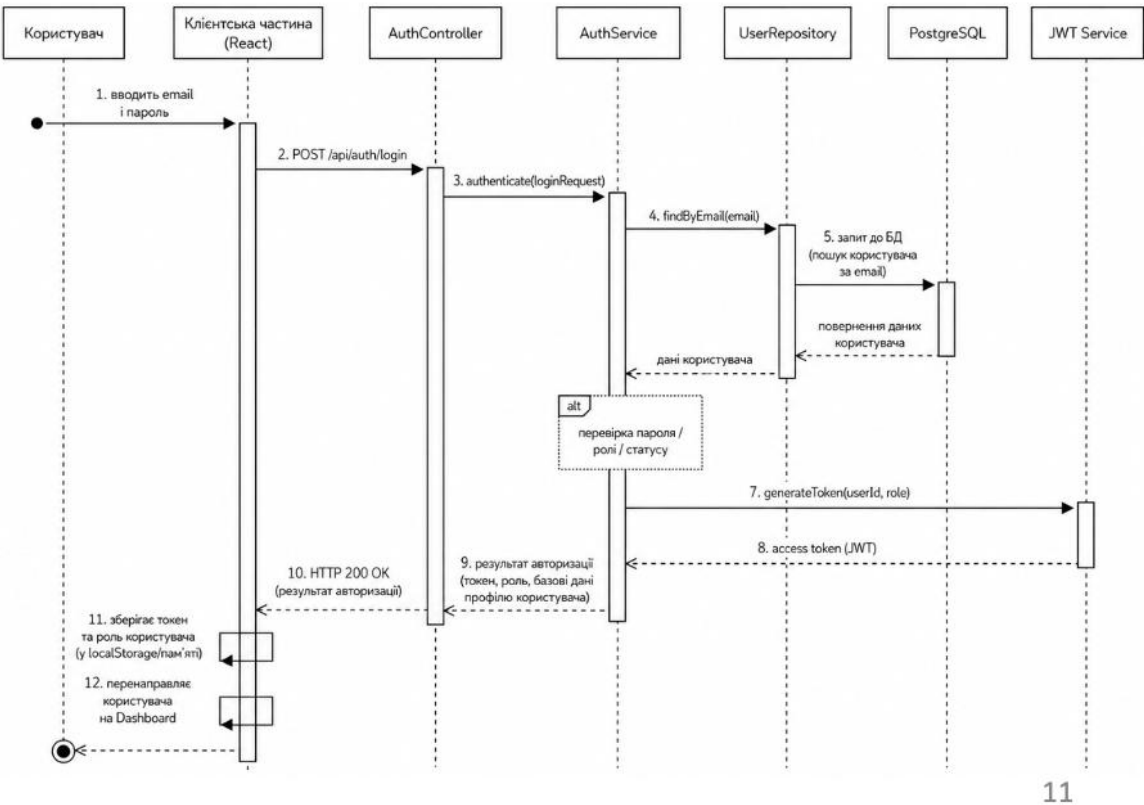
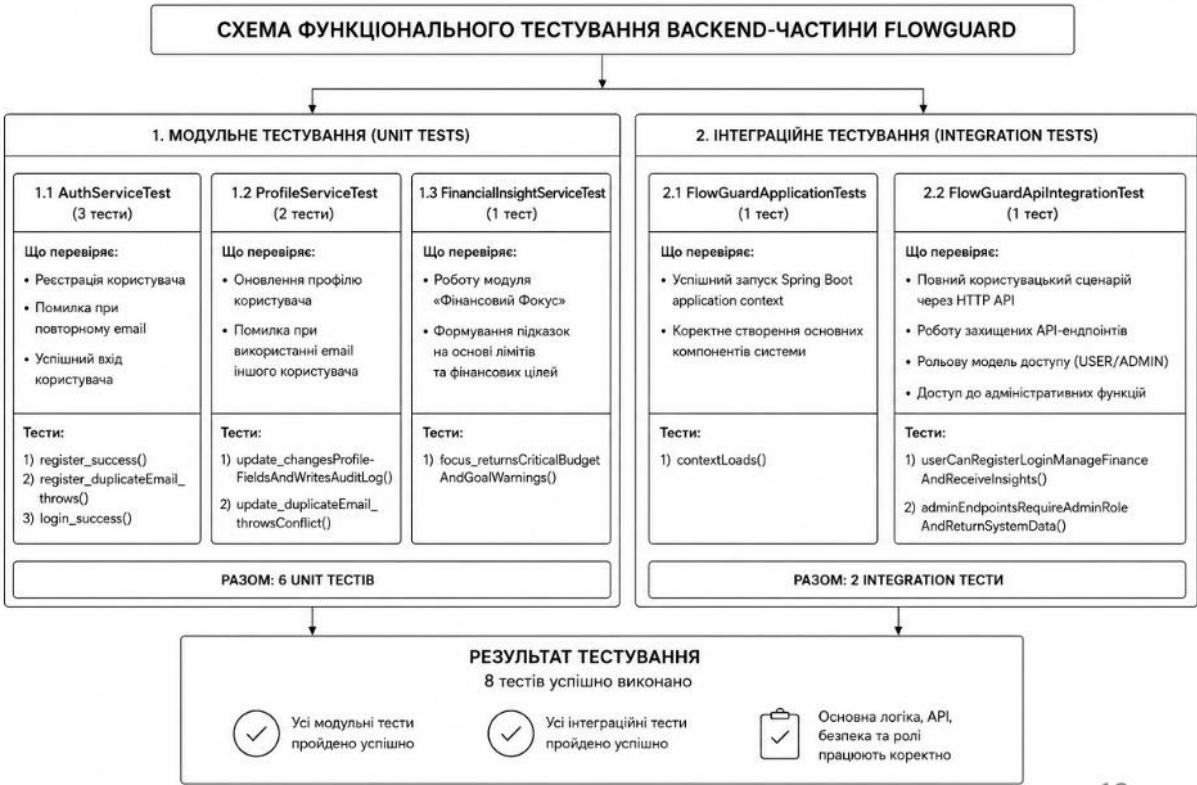


СХЕМА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

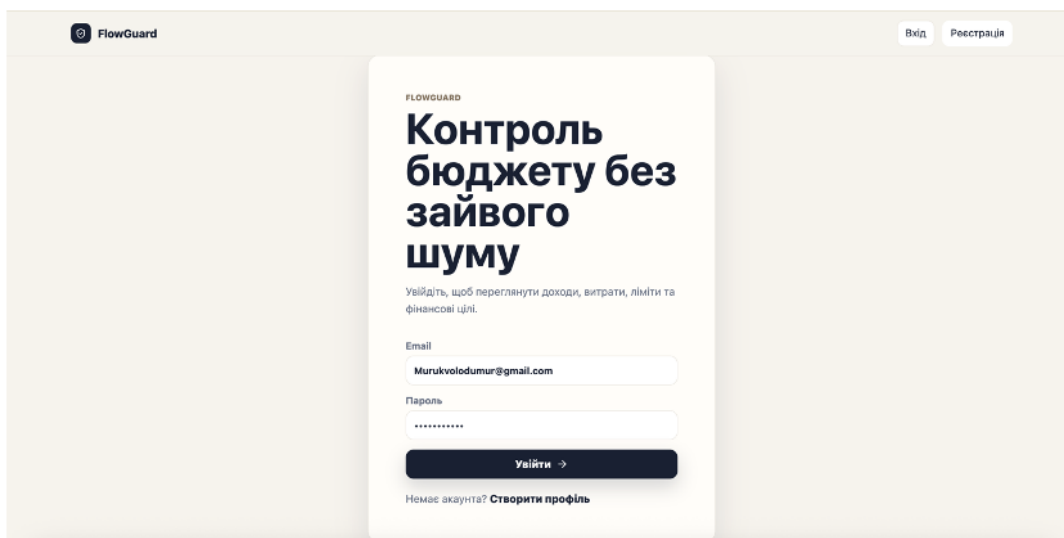


РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

✓ service (com.example.flowguard)	3 sec 381 ms
✓ AuthServiceTest	2 sec 703 ms
✓ register_duplicateEmail_throws()	2 sec 579 ms
✓ login_success()	108 ms
✓ register_success()	16 ms
✓ ProfileServiceTest	147 ms
✓ update_duplicateEmail_throwsConflict()	135 ms
✓ update_changesProfileFieldsAndWritesAuditLog()	12 ms
✓ FinancialInsightServiceTest	531 ms
✓ focus_returnsCriticalBudgetAndGoalWarnings()	531 ms
✓ FlowGuardApiIntegrationTest (com.example.flowguard)	1 sec 825 ms
✓ userCanRegisterLoginManageFinanceAndReceiveInsights()	1 sec 228 ms
✓ adminEndpointsRequireAdminRoleAndReturnSystemData()	597 ms

13

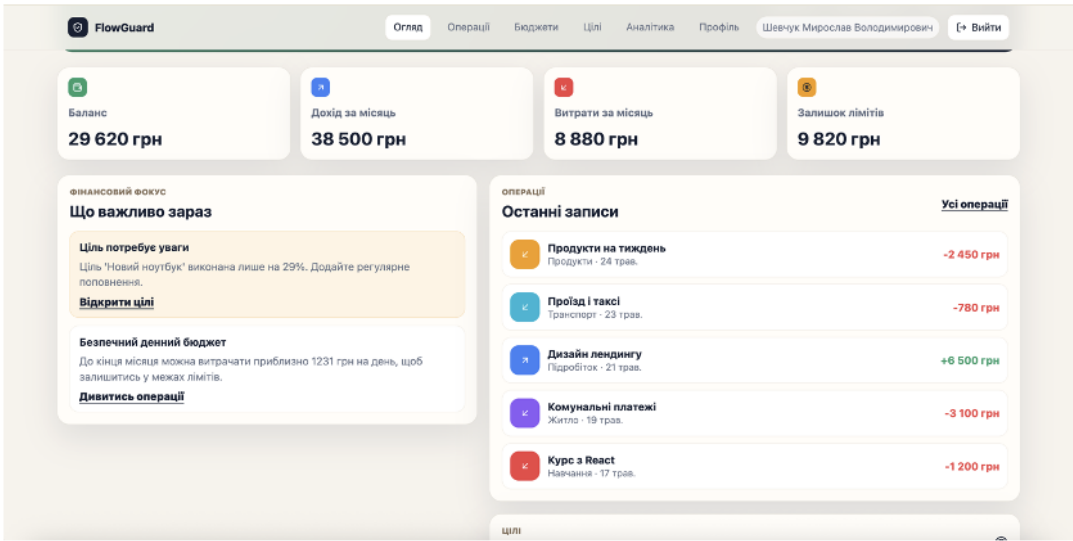
ЕКРАННІ ФОРМИ



Сторінка входу

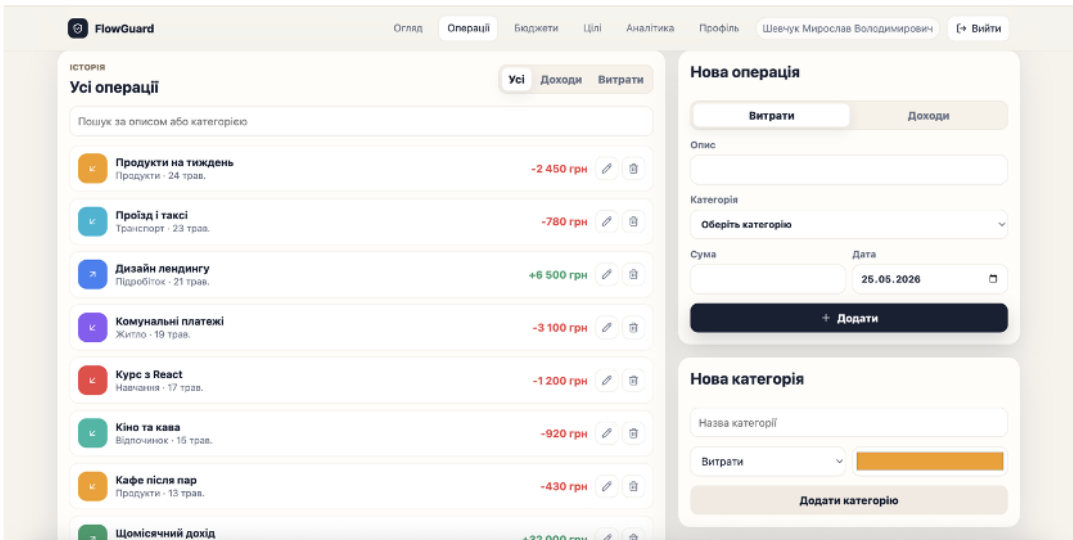
14

ЕКРАННІ ФОРМИ



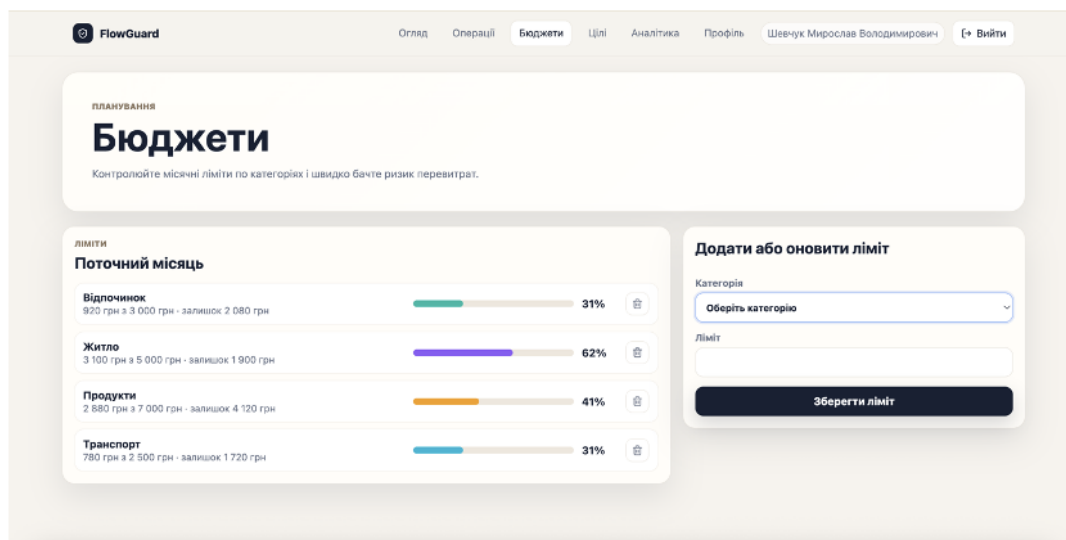
Dashboard з блоком "Фінансовий Фокус"

ЕКРАННІ ФОРМИ



Сторінка операцій

ЕКРАННІ ФОРМИ



Сторінка аналітики бюджету

17

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Шевчук М.В., Корепька В.О. Розробка системи управління бюджетом для особистих витрат. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.528-531.
2. Шевчук М.В., Корепька В.О. Веб-застосунок "FlowGuard" для управління бюджетом та особистими витратами. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. – К.: ДУІКТ, 2025, С.118-119.

19

ВИСНОВКИ

1. Проведено аналіз предметної галузі управління особистими фінансами для визначення основних процесів обліку доходів, витрат, бюджетних лімітів, фінансових цілей та персональних рекомендацій користувача.
2. Проаналізовано програмні засоби Monefy, Wallet, YNAB, Spendee та Money Manager для порівняння їхніх можливостей із розробленою системою та визначення функцій, які доцільно реалізувати у FlowGuard.
3. Визначено проблеми ручного ведення бюджету: складність контролю витрат, відсутність єдиного середовища для фінансових даних, ризик перевищення лімітів і недостатня наочність фінансової аналітики.
4. Спроектовано структуру web-застосунку FlowGuard: за допомогою діаграм описано ролі користувачів, логіку доступу, архітектуру системи, структуру бази даних, взаємодію компонентів та основні сценарії роботи застосунку.
5. Реалізовано web-застосунок FlowGuard для обліку доходів і витрат, керування категоріями, бюджетними лімітами, фінансовими цілями, аналітикою, профілем користувача та адміністративною панеллю.
6. Реалізовано модуль «Фінансовий Фокус», який аналізує витрати, бюджети та цілі користувача, визначає фінансові ризики й формує персональні підказки для швидкого прийняття рішень.
7. Проведено модульне та інтеграційне тестування backend-частини системи для перевірки авторизації, профілю, фінансових підказок, REST API, рольового доступу USER / ADMIN та адміністративних функцій. За результатами тестування підтверджено коректну роботу основної логіки застосунку.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Listing 1. FlowGuardApplication class

```
package com.example.flowguard;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FlowGuardApplication {

    public static void main(String[] args) {

        SpringApplication.run(FlowGuardApplication.class,
            args);
    }

}
```

Listing 2. SecurityConfig class

```
package com.example.flowguard.config;

import com.example.flowguard.security.JwtAuthenticationFilter;
import com.example.flowguard.security.UserDetailsServiceImpl;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
```

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.List;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final UserDetailsServiceImpl userDetailsService;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .sessionManagement(s -> s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(
                    "/api/auth/**",
                    "/api-docs/**",
                    "/swagger-ui/**",
                    "/swagger-ui.html"
                ).permitAll()
                .anyRequest().authenticated()
            )

            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

```

    }

    @Bean
    public CorsConfigurationSource
    corsConfigurationSource() {
        CorsConfiguration config = new
        CorsConfiguration();
        config.setAllowedOrigins(List.of(
            "http://localhost:3001",
            "http://127.0.0.1:3001",
            "http://localhost:3004",
            "http://127.0.0.1:3004",
            "http://localhost:5173",
            "http://127.0.0.1:5173"
        ));
        config.setAllowedMethods(List.of("GET",
            "POST", "PUT", "DELETE", "OPTIONS"));
        config.setAllowedHeaders(List.of("*"));
        config.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source =
        new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return source;
    }

    @Bean
    public DaoAuthenticationProvider
    authenticationProvider() {
        DaoAuthenticationProvider provider = new
        DaoAuthenticationProvider(userDetailsService);

        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager
    authenticationManager(AuthenticationConfiguration
    config) throws Exception {
        return config.getAuthenticationManager();
    }
}

```

Listing 3. JwtTokenProvider class

```

package com.example.flowguard.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;
import java.nio.charset.StandardCharsets;
import java.util.Date;

@Component

```

```

public class JwtTokenProvider {

    @Value("${app.jwt.secret}")
    private String jwtSecret;

    @Value("${app.jwt.expiration-ms}")
    private long jwtExpirationMs;

    private SecretKey key() {
        return
        Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardC
        harsets.UTF_8));
    }

    public String generateToken(String email) {
        Date now = new Date();
        return Jwts.builder()
            .subject(email)
            .issuedAt(now)
            .expiration(new Date(now.getTime() +
            jwtExpirationMs))
            .signWith(key())
            .compact();
    }

    public String getEmailFromToken(String token) {
        return Jwts.parser()
            .verifyWith(key())
            .build()
            .parseSignedClaims(token)
            .getPayload()
            .getSubject();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser().verifyWith(key()).build().parseSignedC
            laims(token);
            return true;
        } catch (JwtException |
        IllegalArgumentException e) {
            return false;
        }
    }
}

```

Listing 4. AuthController class

```

package com.example.flowguard.controller;

import
com.example.flowguard.dto.request.LoginRequest;
import
com.example.flowguard.dto.request.RegisterRequest;
import
com.example.flowguard.dto.response.AuthResponse;
import com.example.flowguard.service.AuthService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

```

```

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@Tag(name = "Аутентифікація")
public class AuthController {

    private final AuthService authService;

    @PostMapping("/register")
    @ResponseStatus(HttpStatus.CREATED)
    @Operation(summary = "Реєстрація нового користувача")
    public AuthResponse register(@Valid
    @RequestBody RegisterRequest request) {
        return authService.register(request);
    }

    @PostMapping("/login")
    @Operation(summary = "Вхід у систему")
    public AuthResponse login(@Valid
    @RequestBody LoginRequest request) {
        return authService.login(request);
    }
}

```

Listing 5. AdminController class

```

package com.example.flowguard.controller;

import
com.example.flowguard.dto.request.AdminCreateUserRequest;
import
com.example.flowguard.dto.request.AdminUpdateUserRoleRequest;
import
com.example.flowguard.dto.response.AdminSummaryResponse;
import
com.example.flowguard.dto.response.AdminUserResponse;
import
com.example.flowguard.dto.response.AuditLogResponse;
import
com.example.flowguard.service.AdminService;
import
com.example.flowguard.service.AuditLogService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/admin")
@RequiredArgsConstructor
@PreAuthorize("hasRole('ADMIN')")
public class AdminController {

```

```

    private final AdminService adminService;
    private final AuditLogService auditLogService;

    @GetMapping("/summary")
    public AdminSummaryResponse summary() {
        return adminService.summary();
    }

    @GetMapping("/users")
    public List<AdminUserResponse> users() {
        return adminService.users();
    }

    @PostMapping("/users")
    @ResponseStatus(HttpStatus.CREATED)
    public AdminUserResponse createUser(@Valid
    @RequestBody AdminCreateUserRequest request) {
        return adminService.createUser(request);
    }

    @PutMapping("/users/{id}/role")
    public AdminUserResponse
    updateRole(@PathVariable Long id, @Valid
    @RequestBody AdminUpdateUserRoleRequest request) {
        return adminService.updateRole(id, request);
    }

    @DeleteMapping("/users/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteUser(@PathVariable Long id) {
        adminService.deleteUser(id);
    }

    @GetMapping("/audit-logs")
    public List<AuditLogResponse> auditLogs() {
        return auditLogService.latest();
    }
}

```

Listing 6. AuthService class

```

package com.example.flowguard.service;

import
com.example.flowguard.dto.request.LoginRequest;
import
com.example.flowguard.dto.request.RegisterRequest;
import
com.example.flowguard.dto.response.AuthResponse;
import com.example.flowguard.entity.User;
import
com.example.flowguard.exception.ConflictException;
import
com.example.flowguard.repository.UserRepository;
import
com.example.flowguard.security.JwtTokenProvider;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authentication.AuthenticationManager;

```

```

import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

@Service
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtTokenProvider tokenProvider;
    private final AuthenticationManager authenticationManager;
    private final AuditLogService auditLogService;

    @Transactional
    public AuthResponse register(RegisterRequest request) {
        if
        (userRepository.existsByEmail(request.getEmail()))
        {
            throw new ConflictException("Email вже
            використовується: " + request.getEmail());
        }

        User user = User.builder()
            .email(request.getEmail())

            .passwordHash(passwordEncoder.encode(request.getPassword()))
            .fullName(request.getFullName())
            .preferredCurrency("UAH")
            .avatarColor("#172033")
            .financialPriority("Контроль витрат")
            .build();

        userRepository.save(user);
        auditLogService.log(user,
            "USER_REGISTERED", "Новий користувач
            зареєструвався", "INFO");

        String token =
        tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(token,
            user.getEmail(), user.getFullName(),
            user.getRole().name());
    }

    public AuthResponse login(LoginRequest request)
    {
        authenticationManager.authenticate(
            new
            UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword())
        );

```

```

        User user =
        userRepository.findByEmail(request.getEmail()).orElseThrow();
        auditLogService.log(user, "USER_LOGIN",
            "Користувач увійшов у систему", "INFO");
        String token =
        tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(token,
            user.getEmail(), user.getFullName(),
            user.getRole().name());
    }
}

```

Listing 7. AdminService class

```

package com.example.flowguard.service;

import
com.example.flowguard.dto.request.AdminCreateUserRequest;
import
com.example.flowguard.dto.request.AdminUpdateUserRoleRequest;
import
com.example.flowguard.dto.response.AdminSummaryResponse;
import
com.example.flowguard.dto.response.AdminUserResponse;
import com.example.flowguard.entity.User;
import
com.example.flowguard.entity.enums.UserRole;
import
com.example.flowguard.exception.ConflictException;
import
com.example.flowguard.exception.ResourceNotFoundException;
import com.example.flowguard.repository.*;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@RequiredArgsConstructor
public class AdminService {

    private final UserRepository userRepository;
    private final AuditLogRepository auditLogRepository;
    private final PasswordEncoder passwordEncoder;
    private final CurrentUserService currentUserService;
    private final AuditLogService auditLogService;

    @Transactional(readOnly = true)
    public AdminSummaryResponse summary() {

```

```

        return new AdminSummaryResponse(
            userRepository.count(),
            userRepository.countByRole(UserRole.ADMIN),
            userRepository.countByRole(UserRole.USER),
            auditLogRepository.count()
        );
    }

    @Transactional(readOnly = true)
    public List<AdminUserResponse> users() {
        return userRepository.findAll().stream()
            .map(user -> new AdminUserResponse(
                user.getId(),
                user.getEmail(),
                user.getFullName(),
                user.getRole().name(),
                user.getCity(),
                user.getPhone(),
                user.getCreatedAt()
            ))
            .toList();
    }

    @Transactional
    public AdminUserResponse
    createUser(AdminCreateUserRequest request) {
        User admin =
        currentUserService.getCurrentUser();
        if
        (userRepository.existsByEmail(request.getEmail()))
        {
            throw new ConflictException("Email вже
            використовується: " + request.getEmail());
        }

        User user = userRepository.save(User.builder()
            .email(request.getEmail())
            .fullName(request.getFullName())

            .passwordHash(passwordEncoder.encode(request.get
            Password()))
            .role(request.getRole() == null ?
            UserRole.USER : request.getRole())
            .phone(request.getPhone())
            .city(request.getCity())
            .preferredCurrency("UAH")
            .avatarColor("#172033")
            .financialPriority("Контроль витрат")
            .build());
        auditLogService.log(admin,
        "ADMIN_CREATED_USER", "Створено акаунт: "
        + user.getEmail(), "INFO");
        return toResponse(user);
    }

    @Transactional
    public AdminUserResponse updateRole(Long id,
    AdminUpdateUserRoleRequest request) {
        User admin =
        currentUserService.getCurrentUser();
        User user = findUser(id);

```

```

        user.setRole(request.getRole());
        auditLogService.log(admin,
        "ADMIN_CHANGED_ROLE", "Змінено роль для
        " + user.getEmail() + " на " + request.getRole(),
        "WARNING");
        return toResponse(user);
    }

    @Transactional
    public void deleteUser(Long id) {
        User admin =
        currentUserService.getCurrentUser();
        if (admin.getId().equals(id)) {
            throw new
            IllegalArgumentException("Адміністратор не може
            видалити власний акаунт");
        }
        User user = findUser(id);
        String email = user.getEmail();
        userRepository.delete(user);
        auditLogService.log(admin,
        "ADMIN_DELETED_USER", "Видалено акаунт: "
        + email, "WARNING");
    }

    private User findUser(Long id) {
        return userRepository.findById(id)
            .orElseThrow(() -> new
            ResourceNotFoundException("Користувача не
            знайдено"));
    }

    private AdminUserResponse toResponse(User
    user) {
        return new AdminUserResponse(
            user.getId(),
            user.getEmail(),
            user.getFullName(),
            user.getRole().name(),
            user.getCity(),
            user.getPhone(),
            user.getCreatedAt()
        );
    }
}

```

Listing 8. FinancialInsightService class

```

package com.example.flowguard.service;

import
com.example.flowguard.dto.response.FinancialGoalR
esponse;
import
com.example.flowguard.dto.response.FinancialInsigh
tResponse;
import com.example.flowguard.entity.BudgetLimit;
import com.example.flowguard.entity.User;
import
com.example.flowguard.entity.enums.TransactionTy
pe;
import
com.example.flowguard.repository.BudgetLimitRepo
sitory;

```

```
import
com.example.flowguard.repository.FinancialGoalRep
ository;
import
com.example.flowguard.repository.TransactionRepos
itory;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transacti
onal;
```

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
```

```
@Service
@RequiredArgsConstructor
public class FinancialInsightService {

    private final CurrentUserService
currentUserService;
    private final BudgetLimitRepository
budgetLimitRepository;
    private final TransactionRepository
transactionRepository;
    private final FinancialGoalRepository
goalRepository;

    @Transactional(readOnly = true)
    public List<FinancialInsightResponse> focus() {
        User user =
currentUserService.getCurrentUser();
        LocalDate today = LocalDate.now();
        LocalDate monthStart =
today.withDayOfMonth(1);
        LocalDate monthEnd =
monthStart.plusMonths(1).minusDays(1);
        List<FinancialInsightResponse> insights = new
ArrayList<>();

        for (BudgetLimit limit :
budgetLimitRepository.findByUserAndMonthStartOr
derByCategoryNameAsc(user, monthStart)) {
            BigDecimal spent =
transactionRepository.sumByCategoryAndTypeBetw
een(
                user,
                limit.getCategory().getId(),
                TransactionType.EXPENSE,
                monthStart,
                monthEnd
            );
            int used = percent(spent,
limit.getLimitAmount());
            if (used >= 90) {
                insights.add(new
FinancialInsightResponse(
                    "critical",
```

```
                    "Критичний ліміт",
                    "Категорія " +
limit.getCategory().getName() + " вже використала
" + used + "% місячного ліміту.",
                    "Відкрити бюджети",
                    "/budgets",
                    100 + used
                ));
            } else if (used >= 70) {
                insights.add(new
FinancialInsightResponse(
                    "warning",
                    "Ліміт наближається",
                    "Категорія " +
limit.getCategory().getName() + " використала " +
used + "% Варто зменшити витрати до кінця
місяця.",
                    "Перевірити ліміти",
                    "/budgets",
                    70 + used
                ));
            }
        }
    }
}
```

```
        BigDecimal monthExpense =
transactionRepository.sumByUserAndTypeBetween(
user, TransactionType.EXPENSE, monthStart,
monthEnd);
        BigDecimal totalBudget =
budgetLimitRepository.findByUserAndMonthStartOr
derByCategoryNameAsc(user, monthStart).stream()
            .map(BudgetLimit::getLimitAmount)
            .reduce(BigDecimal.ZERO,
BigDecimal::add);
        long daysLeft = Math.max(1,
ChronoUnit.DAYS.between(today, monthEnd) + 1);
        if (totalBudget.compareTo(BigDecimal.ZERO)
> 0) {
            BigDecimal safeDaily =
totalBudget.subtract(monthExpense).max(BigDecimal
.ZERO)
                .divide(BigDecimal.valueOf(daysLeft),
0, RoundingMode.HALF_UP);
            insights.add(new FinancialInsightResponse(
                "info",
                "Безпечний денний бюджет",
                "До кінця місяця можна витратити
приблизно " + safeDaily + " грн на день, щоб
залишитись у межах лімітів.",
                "Дивитись операції",
                "/transactions",
                45
            ));
        }

        goalRepository.findByUserOrderByTargetDateAscId
Asc(user).stream()
            .map(goal -> new FinancialGoalResponse(
                goal.getId(),
                goal.getTitle(),
                goal.getTargetAmount(),
                goal.getCurrentAmount(),
```

```

        goal.getTargetDate(),
        goal.getColor(),
        percent(goal.getCurrentAmount(),
goal.getTargetAmount())
    ))
    .filter(goal -> goal.progressPercent() < 35)
    .findFirst()
    .ifPresent(goal -> insights.add(new
FinancialInsightResponse(
    "warning",
    "Ціль потребує уваги",
    "Ціль " + goal.title() + " виконана
лише на " + goal.progressPercent() + "%. Додайте
регулярне поповнення.",
    "Відкрити цілі",
    "/goals",
    80
)));

if (insights.isEmpty()) {
    insights.add(new FinancialInsightResponse(
        "success",
        "Фінанси під контролем",
        "Критичних ризиків немає.
Продовжуйте вести операції та підтримувати
ліміти.",
        "Переглянути аналітику",
        "/analytics",
        20
    ));
}

return insights.stream()

.sorted(Comparator.comparingInt(FinancialInsightRe
sponse::priority).reversed())
    .limit(3)
    .toList();
}

private int percent(BigDecimal value, BigDecimal
total) {
    if (total == null ||
total.compareTo(BigDecimal.ZERO) == 0) {
        return 0;
    }
    return value.multiply(BigDecimal.valueOf(100))
        .divide(total, 0,
RoundingMode.HALF_UP)
        .min(BigDecimal.valueOf(999))
        .intValue();
}
}

```

Listing 9. User class

```

package com.example.flowguard.entity;

import
com.example.flowguard.entity.enums.UserRole;
import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

```

```

import java.math.BigDecimal;
import java.time.LocalDateTime;

```

```

@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length =
255)
    private String email;

    @Column(name = "password_hash", nullable =
false)
    private String passwordHash;

    @Column(name = "full_name", length = 255)
    private String fullName;

    @Column(length = 40)
    private String phone;

    @Column(length = 100)
    private String city;

    @Column(name = "monthly_income", precision =
12, scale = 2)
    private BigDecimal monthlyIncome;

    @Column(name = "financial_priority", length =
80)
    private String financialPriority;

    @Column(name = "preferred_currency", length =
10)
    @Builder.Default
    private String preferredCurrency = "UAH";

    @Column(name = "avatar_color", length = 20)
    @Builder.Default
    private String avatarColor = "#172033";

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    @Builder.Default
    private UserRole role = UserRole.USER;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")

```

```
    private LocalDateTime updatedAt;
```

```
}
Listing 10. Transaction class
```

```
package com.example.flowguard.entity;
```

```
import
com.example.flowguard.entity.enums.TransactionTy
pe;
import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;
```

```
import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
@Entity
@Table(name = "transactions")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Transaction {
```

```
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
```

```
    @ManyToOne(fetch = FetchType.LAZY, optional
= false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
```

```
    @ManyToOne(fetch = FetchType.LAZY, optional
= false)
    @JoinColumn(name = "category_id", nullable =
false)
    private Category category;
```

```
    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    private TransactionType type;
```

```
    @Column(nullable = false, precision = 12, scale =
2)
    private BigDecimal amount;
```

```
    @Column(nullable = false, length = 180)
    private String description;
```

```
    @Column(name = "transaction_date", nullable =
false)
    private LocalDate transactionDate;
```

```
    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
```

```
    @UpdateTimestamp
```

```
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
}
```