Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: **Розробка програмного забезпечення сервісу управління фінансами приватних осіб з використанням технологій Flask, React та банківських API**

Виконав: студент 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Берестень М.В. (прізвище та ініціали)

Керівник (підпис) Багрій-Заяць О.А. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Берестеню Михайлу Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення сервісу управління фінансами приватних осіб з використанням технологій Flask, React та банківських API

Керівник роботи Багрій-Заяць Оксана Андріївна к.т.н. доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 2026 року № _____
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи Предметна область, технічне завдання, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області. 2. Проєктування та реалізація програмної системи. 3. Тестування, впровадження та підтримка. 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Перелік використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграма класів, діаграма варіантів використання, діаграми послідовності, ER–діаграма бази даних, слайди презентації

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С. Ю.		
Нормоконтроль	Стоянов Ю. М.		

7. Дата видачі завдання *06.04.2026р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	
2.	*Робота над першим розділом «Аналіз предметної області та вимог до системи»*	*13.04 – 26.04*	
3.	*Робота над другим розділом «Проєктування та розробка програмної системи»*	*27.04 – 03.05*	
4.	*Робота над третім розділом «Тестування, впровадження та підтримка»*	*04.05 – 17.05*	
5.	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*18.05 – 24.05*	
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.06 – 7.06*	
7.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8.	*Попередній захист кваліфікаційної роботи*	*15.06 – 21.06*	
9.	*Захист кваліфікаційної роботи*		

Студент _____ Берестень М. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Багрій-Заяць О. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Берестень Михайло Васильович. Розробка програмного забезпечення сервісу управління фінансами приватних осіб з використанням технологій Flask, React та банківських API. Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СП-41, спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2026. Сторінок 58, рисунків 12, додатків 2, бібліографічних посилань 43.

Метою роботи є розробка та впровадження веб-сервісу управління власними фінансами, який забезпечує автоматизований облік транзакцій, категоризацію витрат, формування бюджетів та візуалізацію фінансової аналітики.

Об’єктом дослідження є процес створення веб-сервісу для управління особистими фінансами із використанням сучасних веб-технологій та банківських API. Предметом дослідження є методи та технології розробки фінансового веб-сервісу, зокрема архітектурні підходи до побудови REST API, інтеграція з зовнішніми банківськими системами, методи візуалізації фінансових даних та забезпечення безпеки персональної фінансової інформації. Серед методів дослідження: аналіз вимог системи, структурне проєктування архітектури, моделювання бази даних, тестування функціональних компонентів та аналіз продуктивності.

У даній кваліфікаційній роботі проведено аналіз предметної області та огляд існуючих рішень, визначено функціональні й нефункціональні вимоги до системи. Розроблено архітектуру на основі клієнт-серверної моделі з використанням Flask REST API та React SPA. Реалізовано інтеграцію з Monobank open API для автоматичного імпорту транзакцій і з PrivatBank API для отримання курсів валют, а також модулі категоризації витрат, бюджетування та аналітичної візуалізації. Наведено опис процесу тестування, документування REST API за допомогою Swagger UI, впровадження системи та аналіз результатів, які підтверджують ефективність запропонованих рішень. Також розглянуто питання охорони праці

та безпеки в надзвичайних ситуаціях. Робота демонструє повний цикл розробки — від аналізу вимог до впровадження.

Ключові слова: особисті фінанси, Flask, React, REST API, Monobank API, PrivatBank API, PostgreSQL, бюджетування, фінансова аналітика.

ABSTRACT

Beresten Mykhailo Vasylovich. Development of software for a personal finance management service using Flask, React, and banking APIs technologies. Bachelor's degree qualification thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, group SP-41, specialty 121 «Software Engineering». Ternopil, 2026. 58 pages, 12 figures, 2 appendices, 43 bibliographical references.

The aim of this work is the development and deployment of a personal finance management web service that ensures automated transaction accounting, expense categorization, budget creation, and visualization of financial analytics.

The object of the study is the process of creating a web service for personal finance management using modern web technologies and banking APIs.

The subject of the study is the methods and technologies for developing a financial web service, in particular architectural approaches to building a REST API, integration with external banking systems, methods of financial data visualization, and ensuring the security of personal financial information. The research methods include system requirements analysis, structural architecture design, database modeling, functional component testing, and performance analysis.

In this qualification work, an analysis of the subject area and a review of existing solutions were carried out, and functional and non-functional requirements were defined. A client-server architecture was developed using a Flask REST API and a React SPA. Integration with the Monobank open API for automatic transaction import and with the PrivatBank API for currency rates was implemented, along with modules for expense categorization, budgeting, and analytical visualization. The testing process, REST API documentation using Swagger UI, system deployment, and analysis of results confirming the effectiveness of the proposed solutions are presented. Labor protection and safety in emergency situations were also considered.

Keywords: personal finance, Flask, React, REST API, Monobank API, PrivatBank API, PostgreSQL, budgeting, financial analytics.

ЗМІСТ

ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ 12
1.1 Аналіз предметної області 12
1.2 Огляд існуючих рішень та їх порівняльний аналіз 14
1.2.1 Monobank та його вбудований функціонал 14
1.2.2 Toshl Finance 15
1.2.3 YNAB (You Need A Budget) 15
1.3 Вимоги до функціональності системи 16
1.3.1 Автоматичний імпорт транзакцій через Monobank API 16
1.3.2 Категоризація витрат та доходів 16
1.3.3 Бюджетування та фінансове планування 17
1.3.4 Аналітика та візуалізація фінансових даних 17
1.3.5 Система сповіщень та рекомендацій 17
1.4 Актори системи та варіанти використання 17
1.5 Нефункціональні вимоги до системи 19
2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ 20
2.1 Вибір процесу розробки та методології 20
2.2 Архітектура системи та вибір технологічного стеку 21
2.3 Діаграма класів системи 23
2.4 Діаграми діяльності системи 25
2.5 Проєктування бази даних 28
2.6 Розробка серверної частини 31
2.7 Реалізація клієнтської частини 33
2.8 Реалізація аналітичного модуля та візуалізації 34
3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА 36
3.1 План тестування системи 36
3.2 Види та методи тестування 36
3.3 Документування та тестування REST API 37
3.4 Тестування функціональних модулів 41

3.5 Тестування інтерфейсу та адаптивності 41
3.6 Аналіз результатів впровадження 42
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 44
4.1 Соціально-політичні небезпеки, їхні види та характеристики 44
4.2 Вимоги до профілактичних медичних оглядів для працівників ПК 46
ВИСНОВКИ 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 51
ДОДАТКИ 55
Додаток А. Тези конференції 56
Додаток Б 58

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface (інтерфейс прикладного програмування)

REST – Representational State Transfer (передача репрезентативного стану)

SPA – Single Page Application (односторінковий додаток)

JWT – JSON Web Token (токен автентифікації)

MCC – Merchant Category Code (код категорії торговця)

JSON – JavaScript Object Notation (формат обміну даними)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

HTTPS – HyperText Transfer Protocol Secure (захищений протокол передачі)

SQL – Structured Query Language (мова структурованих запитів)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

UI – User Interface (інтерфейс користувача)

UX – User Experience (досвід користувача)

СУБД – Система управління базами даних

CSRF – Cross-Site Request Forgery (підробка міжсайтових запитів)

CSV – Comma-Separated Values (текстовий формат даних)

TTL – Time to live (максимальний період часу або кількість ітерацій або переходів, за який набір даних може існувати до свого зникнення)

UML – Unified Modeling Language (уніфікована мова моделювання)

DOM – Document Object Model (специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами)

URL – Uniform Resource Locator (стандартизована адреса певного ресурсу)

ER-діаграма – Entity-Relationship Diagram (графічна модель, яка використовується для проектування баз даних)

ВСТУП

В умовах сучасної цифрової економіки питання ефективного управління особистими фінансами набуває все більшої актуальності. Зростання кількості фінансових інструментів, банківських продуктів та електронних платіжних систем створює потребу в автоматизованих засобах обліку, аналізу та планування фінансових потоків. У межах даної роботи здійснюється розробка сервісу управління власними фінансами з використанням фреймворку Flask для серверної частини, бібліотеки React для клієнтського інтерфейсу та банківських API для автоматичного імпорту транзакцій. Основною інтеграцією виступає Monobank open API, який забезпечує доступ до історії транзакцій, балансу рахунків, а допоміжну роль відіграє PrivatBank API для отримання актуальних курсів валют. Система орієнтована на автоматизацію обліку фінансових операцій, категоризацію витрат і доходів, формування аналітичних звітів та підтримку прийняття фінансових рішень.

Актуальність теми кваліфікаційної роботи обумовлена декількома факторами. По-перше, за даними Національного банку України, кількість безготівкових транзакцій в Україні щорічно зростає, що підвищує складність ручного обліку фінансів. По-друге, більшість існуючих рішень для управління фінансами є або надмірно спрощеними та не забезпечують глибокої аналітики, або є комерційними продуктами з високою вартістю підписки. По-третє, вбудований функціонал банківських додатків, зокрема Monobank, обмежений лише транзакціями одного банку та не дозволяє формувати повноцінні бюджети чи отримувати розширену аналітику. Створення відкритого та гнучкого сервісу, орієнтованого на українського користувача та побудованого на відкритих банківських API, дозволяє вирішити ці проблеми та забезпечити зручний інструмент для фінансового планування.

Мета дослідження полягає в розробці та впровадженні веб-сервісу управління власними фінансами, який забезпечує автоматизований облік транзакцій, категоризацію витрат, формування бюджетів та візуалізацію фінансової

аналітики. Щоб досягти цієї мети, сформульовано кілька задач:

– Провести аналіз предметної області управління особистими фінансами та огляд існуючих рішень.

– Визначити функціональні та нефункціональні вимоги до сервісу з урахуванням потреб цільової аудиторії.

– Розробити архітектуру програмної системи на основі клієнт-серверної моделі з використанням Flask REST API та React SPA.

– Реалізувати інтеграцію з Monobank open API для автоматичного імпорту банківських транзакцій та з PrivatBank API для отримання курсів валют.

– Розробити модуль категоризації витрат, бюджетування та аналітичної візуалізації фінансових даних.

– Створити адаптивний інтерфейс користувача з використанням React та сучасних бібліотек візуалізації.

– Провести тестування системи та впровадження з подальшим аналізом результатів.

Об’єктом дослідження є процес створення веб-сервісу для управління особистими фінансами із використанням сучасних веб-технологій та банківських API. Предметом дослідження є методи та технології розробки фінансового веб-сервісу, зокрема архітектурні підходи до побудови REST API, інтеграція з зовнішніми банківськими системами, методи візуалізації фінансових даних та забезпечення безпеки персональної фінансової інформації.

Наукова новизна роботи полягає у комплексному підході до розробки фінансового сервісу, який поєднує автоматичний імпорт транзакцій через Monobank open API з інтелектуальною категоризацією витрат.

Серед методів дослідження застосовуються: аналіз вимог системи, структурне проєктування архітектури, моделювання бази даних, тестування функціональних компонентів та аналіз продуктивності. Для візуалізації архітектурних рішень використовуються діаграми UML, зокрема діаграми класів, діаграми діяльності та діаграми варіантів використання.

Апробація результатів дослідження відбулася через впровадження системи в

реальних умовах використання серед групи тестових користувачів, а також через публікацію тез доповідей на науково-практичній конференції. Результати роботи були представлені на науково-технічній конференції, де було висвітлено особливості реалізації інтеграції з банківськими API та архітектурні рішення для забезпечення безпеки фінансових даних.

Отже, ця кваліфікаційна робота розкриває важливі питання розробки фінансових веб-сервісів, пропонує ефективні технічні рішення для автоматизації управління особистими фінансами та підтверджує їх практичну цінність через успішне впровадження та апробацію.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ

Аналіз предметної області та формування вимог є фундаментальним етапом розробки сервісу управління власними фінансами. Він визначає як функціональні, так і нефункціональні характеристики майбутнього програмного продукту. Даний розділ систематизує вимоги, які формують основу для проєктування, реалізації та тестування системи. Коректно сформульовані вимоги дозволяють мінімізувати ризики, оптимізувати ресурси та забезпечити злагоджену взаємодію між усіма компонентами сервісу.

1.1 Аналіз предметної області

Управління власними фінансами є комплексним процесом, що охоплює облік доходів та витрат, планування бюджету, контроль фінансових зобов'язань, формування заощаджень та прийняття інвестиційних рішень. В умовах цифрової економіки традиційні методи ведення фінансового обліку, такі як паперові записи або електронні таблиці, поступаються місцем спеціалізованим програмним рішенням, що забезпечують автоматизацію, аналітику та зручний інтерфейс для роботи з фінансовими даними.

Сучасний ринок систем управління фінансами характеризується стрімким розвитком та зростанням конкуренції. За даними аналітичних досліджень, глобальний ринок програмного забезпечення для управління особистими фінансами оцінюється в мільярди доларів і демонструє стабільне зростання, зумовлене підвищенням фінансової грамотності населення та поширенням смартфонів та веб-технологій. Основними трендами ринку є інтеграція з банківськими API для автоматичного імпорту транзакцій, використання штучного інтелекту для категоризації витрат та формування рекомендацій, а також розвиток візуальних засобів аналітики [17].

В Україні ринок таких рішень має свої специфічні особливості. По-перше, українські користувачі активно використовують безготівкові розрахунки через

банківські картки Monobank, ПриватБанку та інших фінансових установ. По-друге, існує потреба в інтеграції саме з українськими банківськими API, що забезпечують доступ до історії транзакцій у національній валюті з урахуванням локальних особливостей описів платежів. По-третє, більшість міжнародних додатків управління фінансів не підтримують українські банки або мають обмежену функціональність для українського ринку. Окремо слід зазначити, що Monobank став першим українським банком, який надав повноцінний відкритий API для сторонніх розробників з доступом до транзакцій, балансу та MCC-кодів операцій [10, 11].

Предметна область даної кваліфікаційної роботи охоплює процеси створення та функціонування веб-сервісу для управління особистими фінансами. До таких процесів належать: автоматичний імпорт банківських транзакцій через відкриті API, категоризація фінансових операцій на основі MCC-кодів та текстових описів, формування та контроль виконання бюджетів, генерація аналітичних звітів, візуалізація фінансових даних у вигляді графіків та діаграм, а також надання персоналізованих рекомендацій щодо оптимізації витрат.

Важливу роль у побудові подібних сервісів відіграє доступ до банківських API. Monobank open API надає сучасний RESTful інтерфейс із JSON-форматом відповідей, що включає отримання інформації про клієнта та його рахунки, виписку за обраний період із детальною інформацією про кожну транзакцію, а також підтримку webhook-сповіщень для отримання нових транзакцій у режимі реального часу. Доповнює інтеграцію PrivatBank API, який забезпечує доступ до актуальних та архівних курсів валют - що необхідно для мультивалютного обліку [3, 6, 7].

Автоматизація у фінансових сервісах передбачає мінімізацію ручних операцій шляхом використання програмної логіки для класифікації транзакцій, розрахунку бюджетних показників та формування сповіщень. Це дозволяє підвищити ефективність фінансового управління, зменшити кількість помилок та забезпечити актуальність фінансової інформації.

Отож, предметна область дослідження охоплює фінансові веб-сервіси як комплексні програмні рішення, що поєднують серверну логіку обробки даних,

клієнтський інтерфейс візуалізації, реляційну базу даних та інтеграцію з зовнішніми банківськими API. У межах цієї роботи подальший аналіз і практична реалізація зосереджуються на розробці такого сервісу з використанням Flask, React, Monobank API та PrivatBank API.

1.2 Огляд існуючих рішень та їх порівняльний аналіз

Для обґрунтування доцільності розробки нового сервісу та визначення конкурентних переваг необхідно провести аналіз існуючих рішень у сфері управління особистими фінансами. Розглянемо найбільш поширені та релевантні додатки [1, 2].

1.2.1 Monobank та його вбудований функціонал

Monobank є одним з найпопулярніших мобільних банків в Україні, який пропонує вбудований функціонал управління фінансами безпосередньо в банківському додатку. Серед ключових можливостей:

– автоматична категоризація витрат на основі MCC-кодів торгових точок;

– візуалізація витрат за категоріями у вигляді кругової діаграми;

– встановлення лімітів на категорії витрат.

Monobank також надає відкритий API, який дозволяє стороннім додаткам отримувати доступ до інформації про клієнта, його рахунки та повну історію транзакцій із MCC-кодами [6].

Однак вбудований функціонал Monobank має суттєві обмеження:

– аналітика обмежена лише транзакціями картки Monobank;

– функції бюджетування є базовими - лише загальний місячний ліміт на категорію;

– відсутня можливість ручного додавання готівкових операцій;

– функціонал доступний виключно через мобільний додаток без повноцінної веб-версії;

– немає підтримки мультивалютного обліку з конвертацією за актуальними курсами

1.2.2 Toshl Finance

Toshl Finance - це міжнародний сервіс управління фінансами, доступний через веб-інтерфейс та мобільні додатки. Він пропонує ручне введення витрат та доходів, гнучку систему тегів та категорій, підтримку декількох валют, формування бюджетів та фінансових цілей, а також експорт даних у форматах CSV та PDF [4].

Серед переваг Toshl Finance варто зазначити інтуїтивний інтерфейс,мультивалютність та багатоплатформність. Водночас, відсутня пряма інтеграція з українськими банками, що вимагає ручного введення всіх транзакцій, розширений функціонал доступний лише у платній версії, а сервер знаходиться за межами України.

1.2.3 YNAB (You Need A Budget)

YNAB - один з найвідоміших світових сервісів для бюджетування, побудований на унікальній методології розподілу коштів: кожна наявна грошова одиниця повинна мати конкретне призначення. Сервіс пропонує детальні бюджети за категоріями, відстеження фінансових цілей, синхронізацію з банками (переважно американськими та європейськими) та розширену аналітику [5].

Головним недоліком YNAB для українського ринку є відсутність інтеграції з українськими банками, висока вартість підписки та англомовний інтерфейс без локалізації.

1.3 Вимоги до функціональності системи

Розроблюваний сервіс призначений для забезпечення повного циклу фінансового обліку: від автоматичного імпорту банківських транзакцій до формування аналітичних звітів та рекомендацій. Вимоги формуються на основі аналізу предметної області, потреб цільової аудиторії та порівняльного аналізу існуючих рішень.

1.3.1 Автоматичний імпорт транзакцій через Monobank API

Ключовою функцією є автоматичний імпорт транзакцій через Monobank open API. Даний API надає RESTful інтерфейс із JSON-відповідями та простою авторизацією через персональний токен з api.monobank.ua. Monobank API надає доступ до: інформації про клієнта, виписки за рахунком за період до 31 доби, а також webhook для отримання нових транзакцій у реальному часі. Кожна транзакція містить: ідентифікатор, час, опис, MCC-код, суму в копійках, код валюти, залишок після операції та коментар.

Реалізація імпорту передбачає два режими: початковий імпорт за максимально доступний період з дотриманням rate limit (1 запит на 60 секунд) та фоновий режим через webhook для оновлення в реальному часі. Токени зберігатимуться у зашифрованому вигляді, передача даних — через HTTPS. Архітектура інтеграційного шару побудована за модульним принципом. Допоміжну роль відіграє PrivatBank API для курсів валют.

1.3.2 Категоризація витрат та доходів

Сервіс підтримуватиме ієрархічну систему категорій з двома рівнями: основні категорії та підкатегорії. Автоматична категоризація використовує MCC-коди з Monobank API. Якщо MCC-код відповідає кільком категоріям, проводиться аналіз текстового опису.

1.3.3 Бюджетування та фінансове планування

Модуль бюджетування дозволятиме встановлювати місячні ліміти витрат за категоріями та відстежувати їх у реальному часі. Функціональність включатиме: створення щомісячних бюджетів, копіювання бюджету попереднього місяця, автоматичний розрахунок щоденного ліміту, візуалізацію прогресу, порівняння планових та фактичних витрат. Також система підтримуватиме фінансові цілі з відстеженням прогресу.

1.3.4 Аналітика та візуалізація фінансових даних

Аналітичний модуль формуватиме: кругові діаграми розподілу витрат, лінійні графіки динаміки доходів та витрат, стовпчикові діаграми порівняння між періодами, зведені таблиці та теплові карти активності витрат. Для візуалізації буде використано бібліотеку Recharts. Для мультивалютних операцій конвертація здійснюватиметься за курсами PrivatBank [18].

1.3.5 Система сповіщень та рекомендацій

Система сповіщень інформуватиме про: наближення до ліміту бюджету великі транзакції, щотижневі та щомісячні зведення, незвичайні транзакції. Завдяки webhook-інтеграції сповіщення надсилатимуться практично миттєво. Крім того, система формуватиме персоналізовані рекомендації щодо оптимізації витрат на основі аналізу фінансової поведінки.

1.4 Актори системи та варіанти використання

Сервіс передбачає взаємодію з кількома типами акторів, кожен з яких виконуватиме чітко визначені ролі та функції.

Незареєстрований користувач – відвідувач, який переглядає інформаційні сторінки сервісу та має можливість зареєструватися. Він може ознайомитися з можливостями сервісу через демонстраційну сторінку.

Зареєстрований користувач – основний актор, який підключає банківські рахунки через Monobank API, імпортує та переглядає транзакції, категоризує

операції, додає готівкові операції вручну, створює бюджети, переглядає аналітику, налаштовує сповіщення та керує профілем.

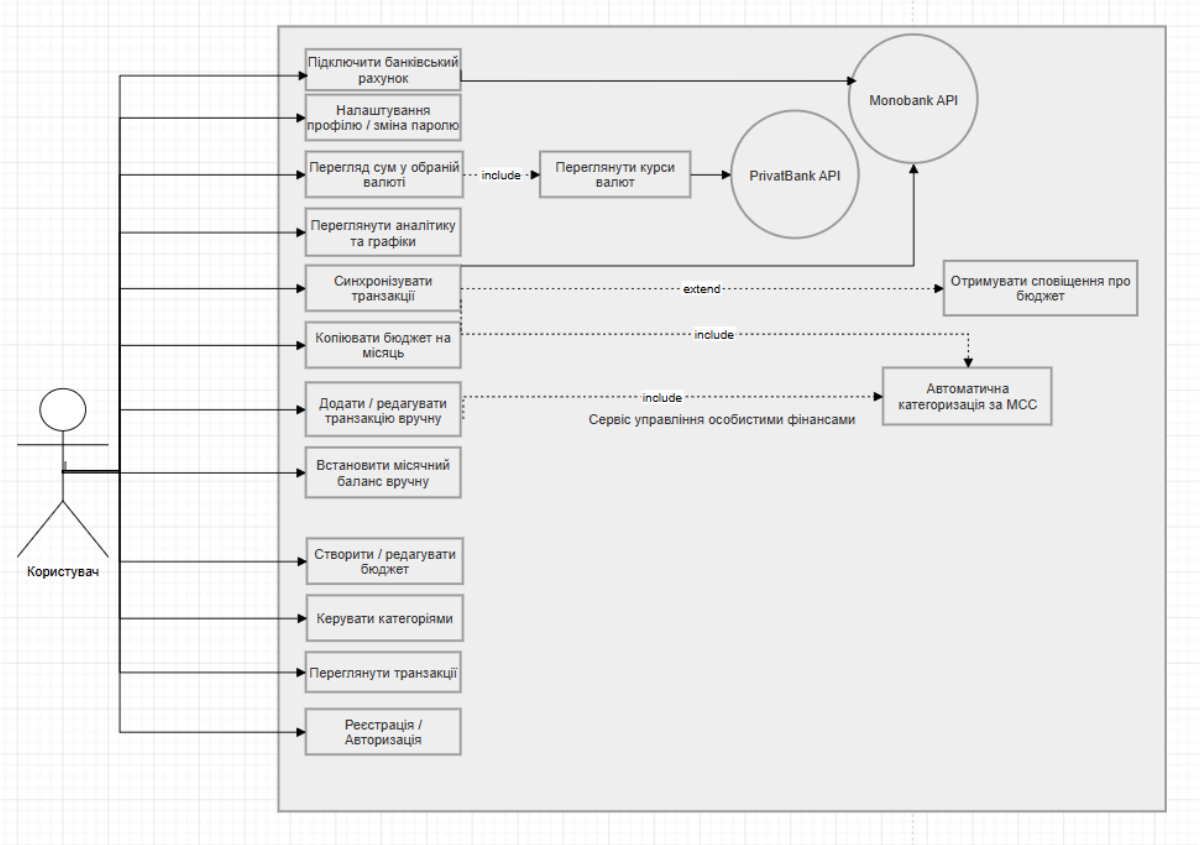

Рисунок 1.1 – Діаграма варіантів використання

Основні варіанти використання зареєстрованого користувача:

- Підключення банківського рахунку – введення токена Monobank API.
- Імпорт транзакцій – автоматично через webhook або за запитом.
- Перегляд та фільтрація транзакцій за датою, категорією, сумою.
- Категоризація транзакцій – підтвердження або зміна автоматичних категорій.
- Додавання готівкових операцій вручну.
- Управління бюджетами – створення, редагування, перегляд прогресу.
- Перегляд аналітики – інтерактивні графіки та звіти.
- Налаштування сповіщень та управління профілем.

Злагоджена робота сервісу базуватиметься на чітко пропрацьованих сценаріях взаємодії, що дозволяє формалізувати вимоги та забезпечити узгодженість між очікуваннями користувачів і реалізованими програмними рішеннями.

1.5 Нефункціональні вимоги до системи

Враховуючи специфіку роботи з фінансовими даними, особливу увагу приділено безпеці, продуктивності та зручності використання [14].

Безпека та захист даних. Необхідно забезпечити:

- шифрування через HTTPS;
- хешування паролів;
- шифрування токенів Monobank API;
- автентифікацію через JWT з обмеженим терміном дії;
- захист від SQL-ін’єкцій, XSS та CSRF;
- валідацію та санітизацію вхідних даних;
- rate limiting для захисту від brute-force атак.

Продуктивність. Час відповіді API — до 200 мс. Завантаження клієнтського інтерфейсу — до 2 секунд. Система повинна обробляти щонайменше 100 одночасних користувачів. Імпорт враховує обмеження Monobank API та виконується у фоновому режимі.

Масштабованість. Модульна структура серверної частини та абстракція інтеграційного шару дозволяють додавати нові банківські конектори та масштабувати окремі компоненти.

Доступність та відмовостійкість. Коректна обробка тимчасової недоступності Monobank та PrivatBank API. Механізми повторного виконання запитів та обробки HTTP 429.

Дотримання нефункціональних вимог є ключовим фактором успіху фінансового сервісу та враховуватиметься на етапі проєктування архітектури.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Розробка і проєктування програмної системи – це етап, який визначає структуру, функції та технічні деталі проєкту. У цьому розділі знаходиться опис методів проєктування, вибір технологічного стеку, а також реалізація основних модулів для забезпечення стабільної роботи, масштабованості та зручності у використанні системи.

2.1 Вибір процесу розробки та методології

Розробка сервісу базується на гібридному підході, що поєднує елементи Agile та Kanban. Вибір зумовлений гнучкими вимогами проєкту, необхідністю швидко реагувати на зміни в пріоритетах та поступовим впровадженням функціоналу.

Процес розробки залишається ітеративним та адаптивним. Завдання поділяються на окремі блоки і передаються на виконання поетапно, враховуючи їх пріоритетність. Це дозволяє швидко реагувати на зміни вимог, поступово впроваджувати функціонал та зменшувати ризики, пов’язані з тривалим циклом розробки.

Розробку системи можна розділити на кілька основних етапів. На етапі аналізу проводиться детальне вивчення вимог до сервісу, визначення структури API-інтеграцій, проєктування бази даних та макетування інтерфейсу. На етапі реалізації відбувається поетапна розробка: спочатку серверна частина з Flask REST API та інтеграція з Monobank API, потім клієнтська частина на React, і нарешті — аналітичний модуль. Етап впровадження передбачає розгортання на продуктивному сервері з подальшим моніторингом та оптимізацією [19].

Для організації робочого процесу використовується система контролю версій Git з хостингом на GitHub, що дозволяє відстежувати зміни коду, створювати гілки для окремих функцій та забезпечувати безпечний відкат у разі помилок.

2.2 Архітектура системи та вибір технологічного стеку

Архітектура сервісу побудована на клієнт-серверній моделі з чітким розділенням на три рівні: клієнтський рівень, серверний рівень та рівень даних. Така архітектура забезпечує незалежність розробки та масштабування кожного рівня, а також дозволяє замінювати окремі компоненти без впливу на інші частини системи [25, 26].

Серверна частина реалізована на Flask — легковісному Python-фреймворку для побудови веб-додатків. Flask обрано з наступних причин: [20, 21]

- мінімалістична архітектура, що дозволяє структурувати проєкт відповідно до потреб;
- багата екосистема розширень;
- вбудована підтримка RESTful API;
- Python має потужні бібліотеки для роботи з даними та аналітикою.

Для бази даних обрано PostgreSQL — потужну реляційну СУБД з відкритим кодом. PostgreSQL забезпечує надійне зберігання фінансових даних, підтримку складних запитів для аналітики, транзакційну цілісність та розширені можливості індексування. Взаємодія з базою даних здійснюється через ORM SQLAlchemy, який спрощує роботу з даними та забезпечує захист від SQL-ін’єкцій [23, 24].

Клієнтська частина реалізована на React — JavaScript-бібліотеці для побудови інтерактивних інтерфейсів. React обрано завдяки компонентному підходу, віртуальному DOM для високої продуктивності, великій екосистемі бібліотек та широкій спільноті розробників. Для управління станом використовується React Context API у поєднанні з хуками useState та useEffect. Маршрутизація здійснюється через React Router. Для стилізації використовується CSS-фреймворк Tailwind CSS, який забезпечує швидку розробку адаптивного інтерфейсу [22].

Для візуалізації фінансових даних обрано бібліотеку Recharts, яка побудована на основі D3.js та забезпечує створення інтерактивних графіків із вбудованою підтримкою адаптивності та анімацій [28].

Для забезпечення безпеки використовуються:

– бібліотека PyJWT для створення та перевірки JWT-токенів;
– bcrypt для хешування паролів;
– cryptography для шифрування банківських токенів у базі даних.

Обмін даними між клієнтом та сервером здійснюється через JSON-формат по HTTPS-з’єднанню.

Таким чином, технологічний стек Flask + PostgreSQL + React + Recharts забезпечує повний цикл розробки фінансового сервісу: від безпечного зберігання та обробки даних до інтерактивної візуалізації для кінцевого користувача.

2.3 Діаграма класів системи

Діаграма класів відображає архітектуру та взаємозв'язки між компонентами серверної частини системи.

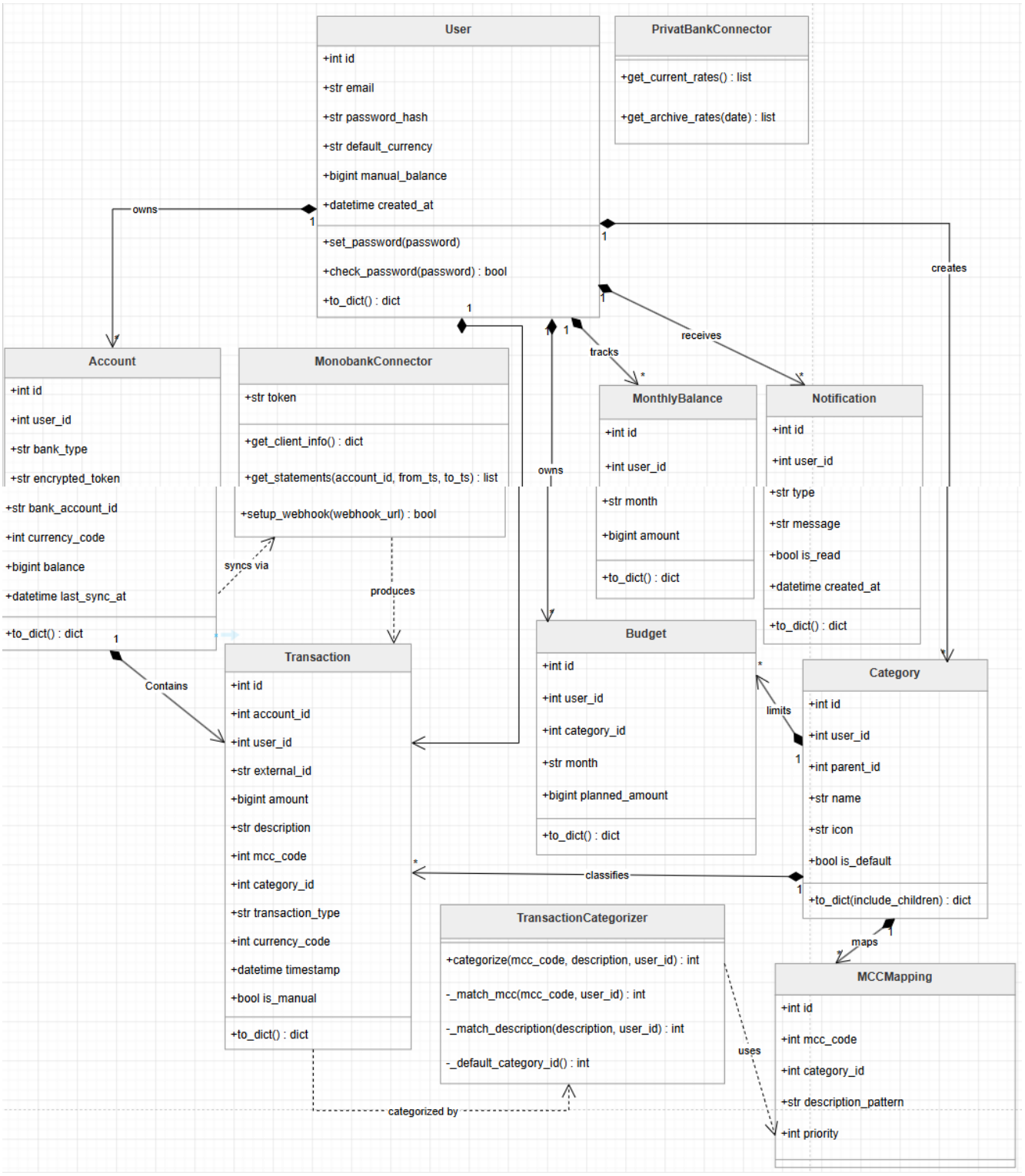

Рисунок 2.1 – Діаграма класів системи

Нижче наведено опис основних класів системи та їх призначення:

– User — модель користувача системи; зберігає облікові дані (email, хеш пароля), базову валюту та ручний баланс, а також реалізує методи автентифікації. Пов’язаний відношенням «один-до-багатьох» із класами Account, Transaction, Budget, Category, MonthlyBalance та Notification.

– Account — банківський рахунок, підключений через API; зберігає зашифрований токен доступу, тип банку, ідентифікатор рахунку в банківській системі, баланс, валюту та час останньої синхронізації. Реалізує розшифрування токена для виконання API-запитів та ініціювання імпорту транзакцій.

– Transaction — центральний клас системи, що зберігає інформацію про фінансову операцію: суму, опис, MCC-код, дату й час, тип операції (дохід/витрата), зовнішній ідентифікатор з банку, валюту та посилання на категорію. Підтримує автоматичну категоризацію на основі MCC-коду й опису.

– Category — категорія витрат або доходів із підтримкою ієрархії через поле parent_id; містить атрибути name, icon, is_default та забезпечує отримання підкатегорій.

– MCCMapping — довідник відповідності MCC-кодів категоріям; зберігає MCC-код, посилання на категорію, шаблон опису та пріоритет, що використовується алгоритмом автоматичної категоризації.

– Budget — бюджет за категорією на визначений місяць; містить планову суму, місяць і посилання на категорію, а також забезпечує розрахунок витрачених коштів та відсотка використання бюджету.

– MonthlyBalance — збережений вручну місячний баланс користувача; зберігає місяць у форматі YYYY-MM та суму, що використовується для коригування зведеної фінансової картини.

– Notification — сповіщення користувача; зберігає тип (попередження або перевищення бюджету, велика транзакція), текст повідомлення, ознаку прочитання та час створення.

– MonobankConnector — конектор інтеграції з Monobank API; реалізує

отримання інформації про клієнта (get_client_info), завантаження виписок (get_statements) та налаштування webhook (setup_webhook).

– PrivatBankConnector — конектор інтеграції з PrivatBank API; реалізує отримання поточних (get_current_rates) та архівних (get_archive_rates) курсів валют.

– TransactionCategorizer — сервіс автоматичної категоризації, що визначає категорію транзакції за MCC-кодом і описом, послідовно застосовуючи зіставлення за MCC, за шаблоном опису та категорію за замовчуванням.

2.4 Діаграми діяльності системи

Діаграми діяльності візуалізують ключові процеси взаємодії користувачів з сервісом та демонструють логічну послідовність дій.

Перша діаграма «Процес підключення банківського рахунку та імпорту транзакцій» відображає послідовність дій від авторизації до отримання транзакцій. Користувач переходить на сторінку налаштувань, обирає банк, переходить за посиланням на api.monobank.ua для генерації токена, копіює та вводить токен у сервіс. Система перевіряє валідність токена через запит GET. Якщо токен валідний, система зберігає його в зашифрованому вигляді, отримує інформацію про рахунки та ініціює початковий імпорт транзакцій. Імпорт виконується послідовно за періодами по 31 добі з дотриманням rate limit. Після завершення імпорту система налаштовує webhook для отримання нових транзакцій у реальному часі. Якщо токен невалідний, система повідомляє про помилку та пропонує повторити введення.

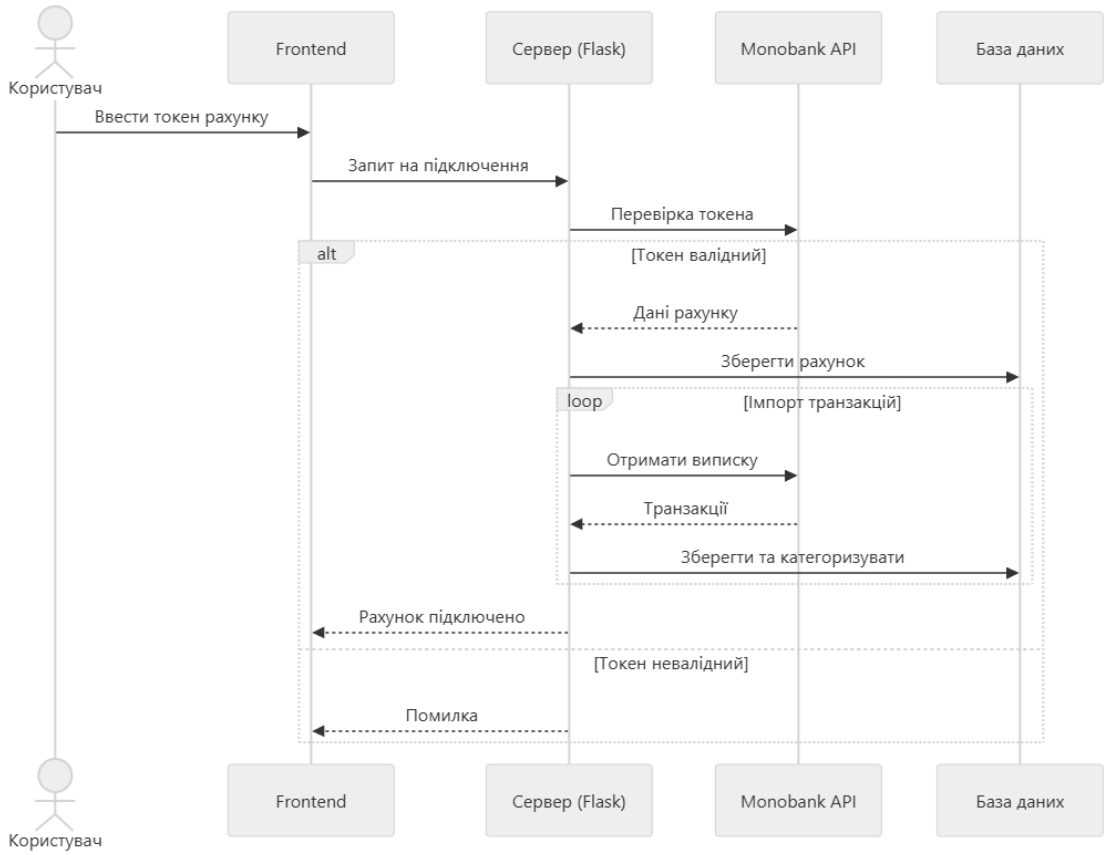

Рисунок 2.2 – Процес підключення банківського рахунку
та імпорту транзакцій

Друга діаграма «Процес перегляду аналітики» показує взаємодію користувача з аналітичним модулем. Користувач переходить на сторінку аналітики, обирає тип звіту, налаштовує параметри. Клієнт надсилає запит до REST API, сервер агрегує дані з бази, при потребі конвертує валюти за курсами PrivatBank API та повертає підготовлені дані. React-компонент відображає інтерактивний графік через Recharts із можливістю масштабування та фільтрації.

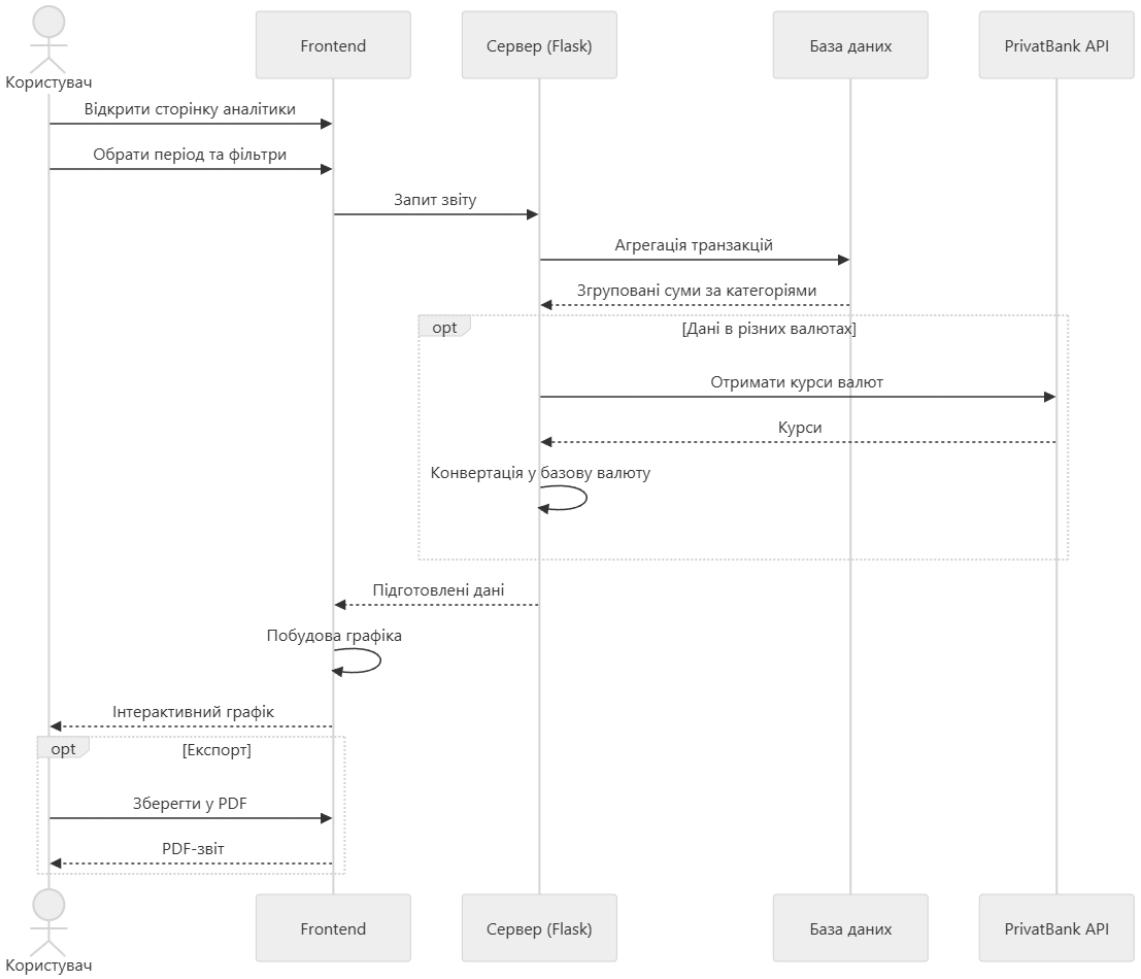

Рисунок 2.3 – Процес перегляду аналітики

Третя діаграма «Процес обробки webhook-сповіщення» ілюструє автоматичний процес при надходженні нової транзакції. Monobank надсилає POST-запит на webhook URL сервісу. Сервер перевіряє автентичність запиту, парсить дані транзакції, перевіряє на дублікати за external_id, виконує автоматичну категоризацію на основі MCC-коду, зберігає транзакцію в базі даних, перевіряє бюджетні ліміти та за потреби створює сповіщення для користувача.

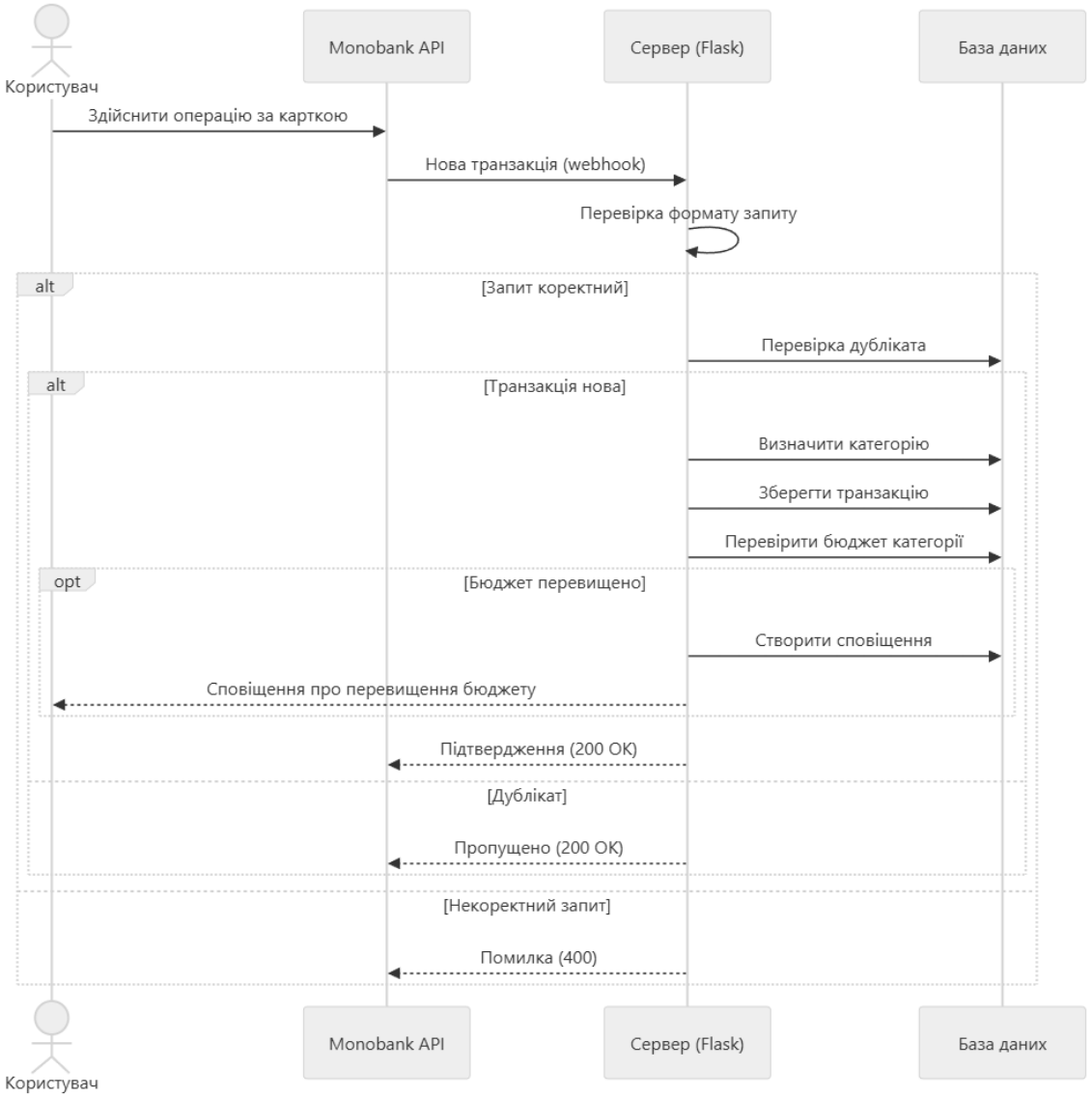

Рисунок 2.4 – Процес обробки webhook-сповіщення

2.5 Проєктування бази даних

Проєктування бази даних здійснюється з використанням PostgreSQL та ORM SQLAlchemy. База даних складається з кількох основних таблиць, об’єднаних у логічні групи: користувачі та автентифікація, банківські рахунки, транзакції, категорії та бюджети [24].

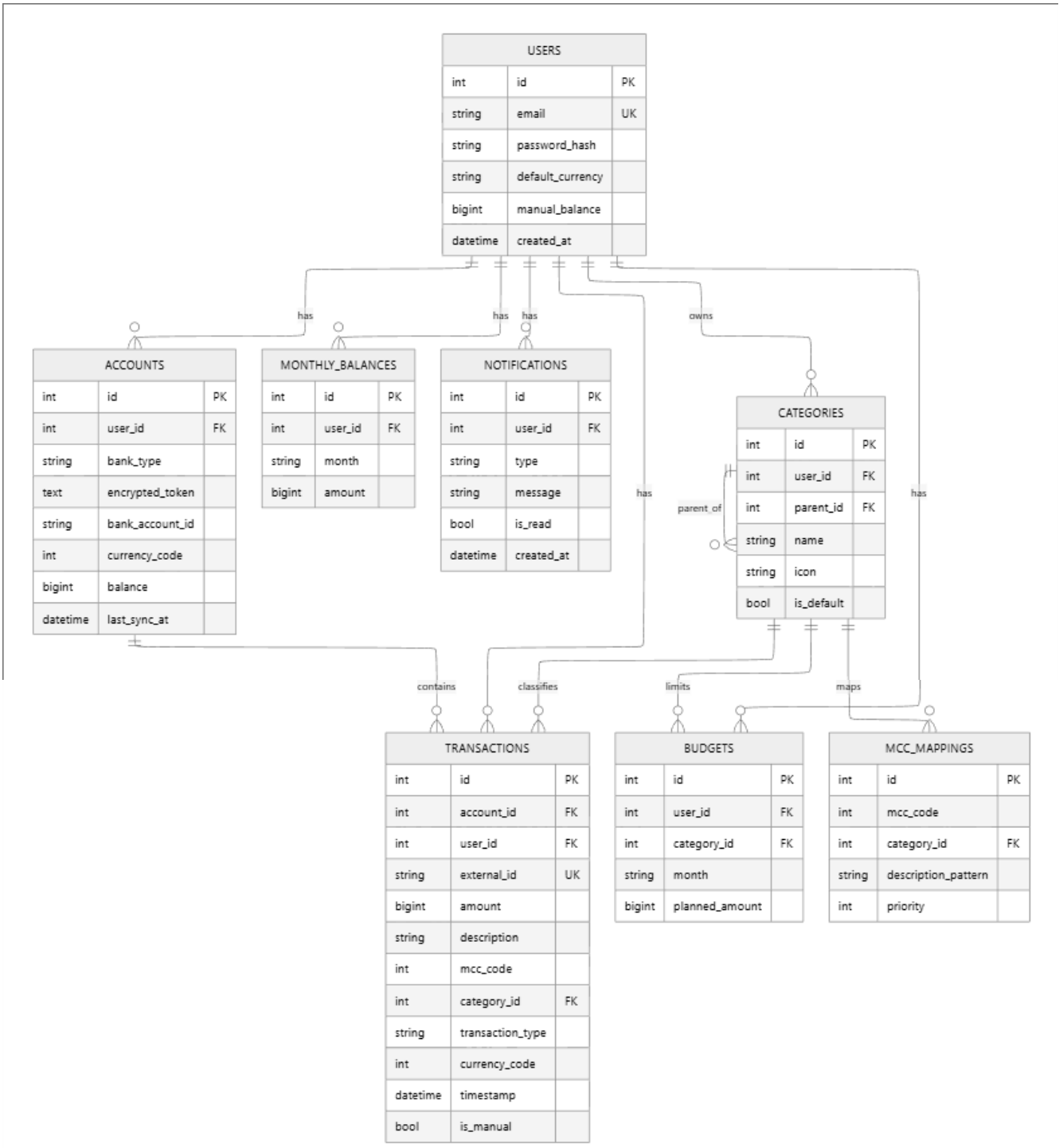

Рисунок 2.5 – ER–діаграма бази даних

Структуру бази даних та зв’язки між таблицями подано у вигляді ER-діаграми на рисунку 2.5.

База даних складається з таких основних таблиць:

– users — облікові записи користувачів. Поля: id (первинний ключ), email (унікальний), password_hash (bcrypt-хеш пароля), default_currency (базова валюта

відображення), manual_balance (введений вручну баланс), created_at (дата реєстрації). Пароль зберігається виключно у вигляді хешу.

– accounts — підключені банківські рахунки. Поля: id, user_id (зовнішній ключ на users), bank_type (тип банку), encrypted_token (зашифрований токен доступу), bank_account_id (ідентифікатор рахунку в банку), currency_code (код валюти), balance (баланс), last_sync_at (час останньої синхронізації).

– transactions — фінансові операції (найбільша таблиця). Поля: id, account_id (зовнішній ключ на accounts), user_id, external_id (унікальний ідентифікатор операції з банку), amount (сума в копійках), description (опис), mcc_code (MCC-код), category_id (зовнішній ключ на categories), transaction_type (дохід/витрата), currency_code, timestamp (дата й час), is_manual (ознака ручного додавання). Створено індекси на полях timestamp, category_id та external_id.

– categories — ієрархічна структура категорій. Поля: id, user_id (NULL для системних категорій), parent_id (самопосилання для підкатегорій), name (назва), icon (піктограма), is_default (ознака категорії за замовчуванням).

– mcc_mappings — довідник відповідності MCC-кодів категоріям. Поля: id, mcc_code (MCC-код), category_id (зовнішній ключ на categories), description_pattern (шаблон опису), priority (пріоритет правила).

– budgets — місячні бюджети за категоріями. Поля: id, user_id, category_id (зовнішній ключ на categories), month (місяць у форматі YYYY-MM), planned_amount (планова сума). Унікальність забезпечується комбінацією (user_id, category_id, month).

– monthly_balances — збережені вручну місячні баланси користувача. Поля: id, user_id, month (формат YYYY-MM), amount (сума). Унікальність забезпечується комбінацією (user_id, month).

– notifications — сповіщення користувача. Поля: id, user_id, type (тип сповіщення), message (текст), is_read (ознака прочитання), created_at (час створення).

Для забезпечення швидкої роботи з великою кількістю транзакцій реалізовано індексування ключових полів: складний індекс на (account_id,

timestamp) для виписок за періодом, унікальний індекс на external_id для запобігання дублюванню, індекс на у таблиці budgets для швидкого пошуку бюджетів.

Міграції бази даних контролюються через Flask-Migrate, що дозволяє версіонувати зміни схеми та безпечно оновлювати продуктивну базу.

2.6 Розробка серверної частини

Серверна частина системи побудована як REST API на основі фреймворку Flask. Її код розділено на окремі логічні модулі, кожен з яких відповідає за свою функціональну область: автентифікацію користувачів, керування банківськими рахунками, роботу з транзакціями, категоріями, бюджетами, формування аналітики та приймання сповіщень від банку. Такий поділ полегшує підтримку проєкту та дозволяє розвивати кожен модуль незалежно [8, 9, 16].

Захист доступу до системи побудовано на механізмі токенів. Під час реєстрації пароль користувача не зберігається у відкритому вигляді, а перетворюється на захищений хеш. Після входу користувач отримує два токени: короткочасний, який підтверджує його особу під час звичайної роботи, і довготривалий, який дозволяє автоматично продовжити сеанс без повторного введення пароля. Завдяки цьому користувач рідше вводить дані для входу, а захищеність облікового запису залишається високою.

Робота з транзакціями організована так, щоб користувач міг швидко знаходити потрібні операції. Список транзакцій можна фільтрувати за періодом, категорією, типом операції та сумою, а також упорядковувати за датою або величиною. Для зручного перегляду великих обсягів даних результати видаються частинами (посторінково), що пришвидшує завантаження сторінки.

Окремий модуль відповідає за аналітику. Він узагальнює дані безпосередньо на рівні бази даних і повертає вже підготовлені показники: розподіл витрат за категоріями, динаміку доходів і витрат за місяцями та зведення витрат за днями для побудови графіків. Завдяки тому, що обчислення виконуються на сервері,

клієнтська частина отримує готові до відображення дані й працює швидко.

Для надійності системи передбачено єдиний підхід до обробки помилок: у відповідь на некоректний запит сервер повертає зрозуміле повідомлення замість технічного збою. Усі вхідні дані перевіряються ще до обробки, що захищає систему від помилкових або шкідливих запитів. Крім того, ведеться журнал роботи сервера, який допомагає відстежувати події та виявляти можливі проблеми.

Інтеграція з Monobank API реалізована через клас MonobankConnector, який інкапсулює всю логіку взаємодії з банківським API. Базова URL-адреса — https://api.monobank.ua. Авторизація здійснюється через заголовок із персональним токеном користувача [6].

Метод get_client_info виконує GET-запит до /personal/client-info та повертає інформацію про клієнта, включаючи масив рахунків з їх ідентифікаторами, балансами та валютами. Ця інформація використовується при першому підключенні для створення записів у таблиці accounts.

Метод get_statements виконує GET-запит до /personal/statement/. API повертає масив транзакцій із полями: id (унікальний ідентифікатор), time (Unix timestamp), description, mcc (MCC-код), originalMcc, operationAmount, currencyCode, commissionRate, cashbackAmount, balance (залишок після операції), comment. Метод враховує обмеження API: максимальний період — 31 доба, мінімальний інтервал між запитами — 60 секунд. При отриманні HTTP 429 (Too Many Requests) система очікує вказаний час та повторює запит.

Для отримання транзакцій у реальному часі реалізовано механізм webhook. При підключенні рахунку сервіс надсилає POST-запит до /personal/webhook із URL-адресою свого ендпоінту. Після цього Monobank автоматично надсилає POST-запити з даними нових транзакцій на вказану адресу. Ендпоінт /api/webhooks/monobank обробляє ці запити: перевіряє формат даних, виконує дедуплікацію за external_id, категоризує транзакцію та зберігає її в базі.

Інтеграція з PrivatBank API реалізована через клас PrivatBankConnector. Використовується публічний API курсів валют за адресою https://api.privatbank.ua. Курси кешуються на рівні сервера з TTL [7].

2.7 Реалізація клієнтської частини

Клієнтська частина реалізована як Single Page Application (SPA) з використанням React та створена за допомогою Vite — сучасного інструменту збірки, що забезпечує швидку розробку та оптимізовану збірку для продакшену [12].

Структура додатку організована за принципом feature-based: кожна функціональна область (auth, dashboard, transactions, budgets, analytics, settings) має свою директорію з компонентами, хуками та утилітами. Маршрутизація реалізована через React Router v6 із захищеними маршрутами, доступними лише авторизованим користувачам.

Головна сторінка (Dashboard) відображає зведену інформацію: загальний баланс по всіх рахунках, витрати та доходи за поточний місяць, прогрес виконання бюджетів, останні транзакції та мініатюрну кругову діаграму розподілу витрат. Сторінка транзакцій надає повний список операцій із фільтрацією за датою, категорією, типом та сумою, а також можливість ручного додавання готівкових операцій.

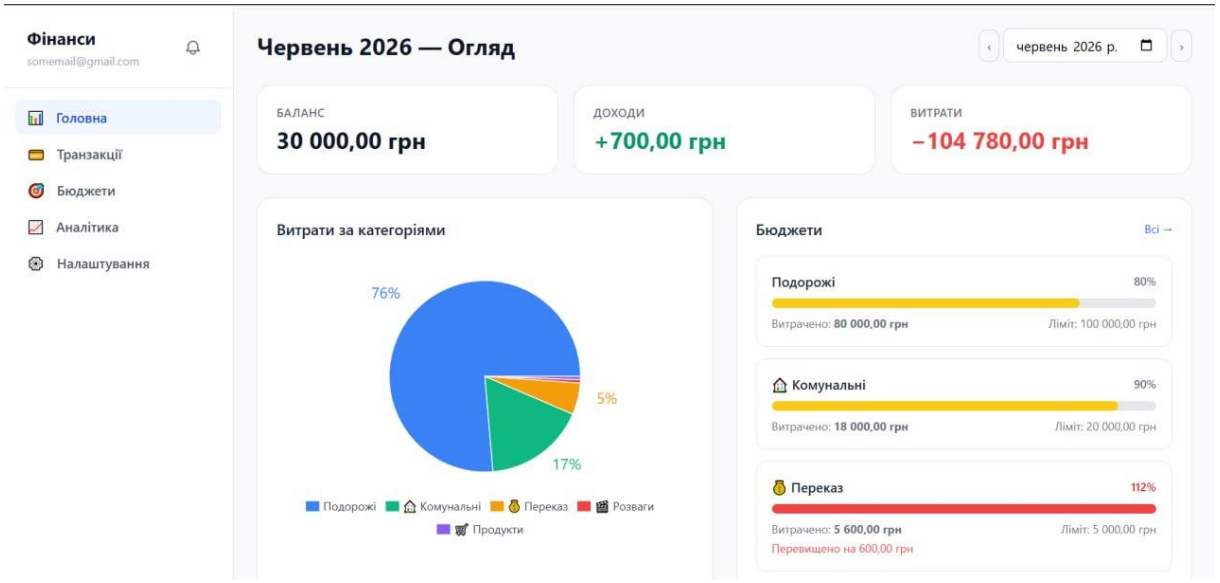

Рисунок 2.6 – Головна сторінка додатка

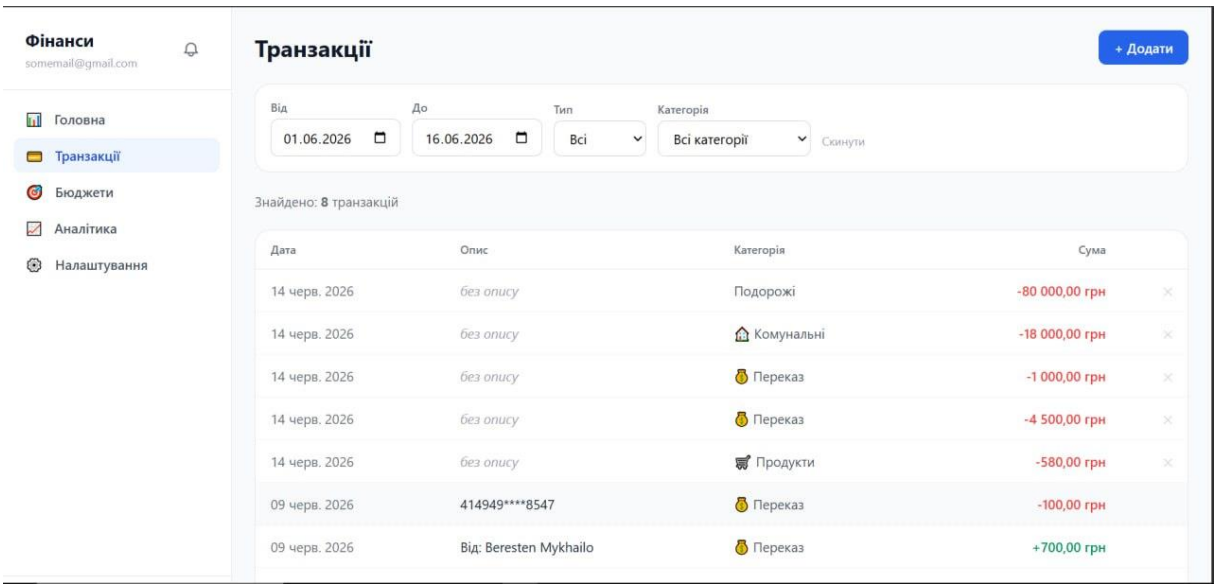

Рисунок 2.7 – Сторінка транзакцій з фільтрацією операцій

Для взаємодії з REST API використовується бібліотека Axios із централізованою конфігурацією: базова URL-адреса, автоматичне додавання JWT-токена до заголовків, interceptor для автоматичного оновлення access token через refresh token при отриманні HTTP 401. Стан автентифікації зберігається в React Context та передається через провайдер на верхньому рівні додатку [29].

Адаптивність інтерфейсу забезпечується Tailwind CSS із використанням утилітарних класів для різних розмірів екранів. Навігація на мобільних пристроях реалізована через випадаюче меню замість бічної панелі на десктопі. Таблиці транзакцій на мобільних пристроях перетворюються на карткове представлення для кращої зручності.

Форми валідуються на клієнтській стороні через бібліотеку React Hook Form із схемами валідації Yup, що забезпечує миттєвий зворотний зв’язок для користувача ще до надсилання запиту на сервер.

2.8 Реалізація аналітичного модуля та візуалізації

Аналітичний модуль є ключовою конкурентною перевагою сервісу, що відрізняє його від вбудованого функціоналу Monobank. Модуль складається із

серверної частини та клієнтської частини.

Серверна агрегація реалізована через SQLAlchemy із використанням SQL-функцій GROUP BY, SUM, COUNT та фільтрації. Для розподілу витрат за категоріями виконується запит з групуванням за category_id та агрегацією суми [24].

Клієнтська візуалізація побудована на Recharts із React-компонентами. Компонент SpendingPieChart відображає кругову діаграму розподілу витрат за категоріями із кольоровим кодуванням, підписами та можливістю кліку для переходу до детальних транзакцій категорії. Компонент MonthlyTrendChart відображає лінійний графік доходів та витрат за 12 місяців із двома лініями різних кольорів та областю під графіком. Компонент BudgetProgressBars відображає прогрес-бари для кожного бюджету із кольоровою індикацією [15, 28].

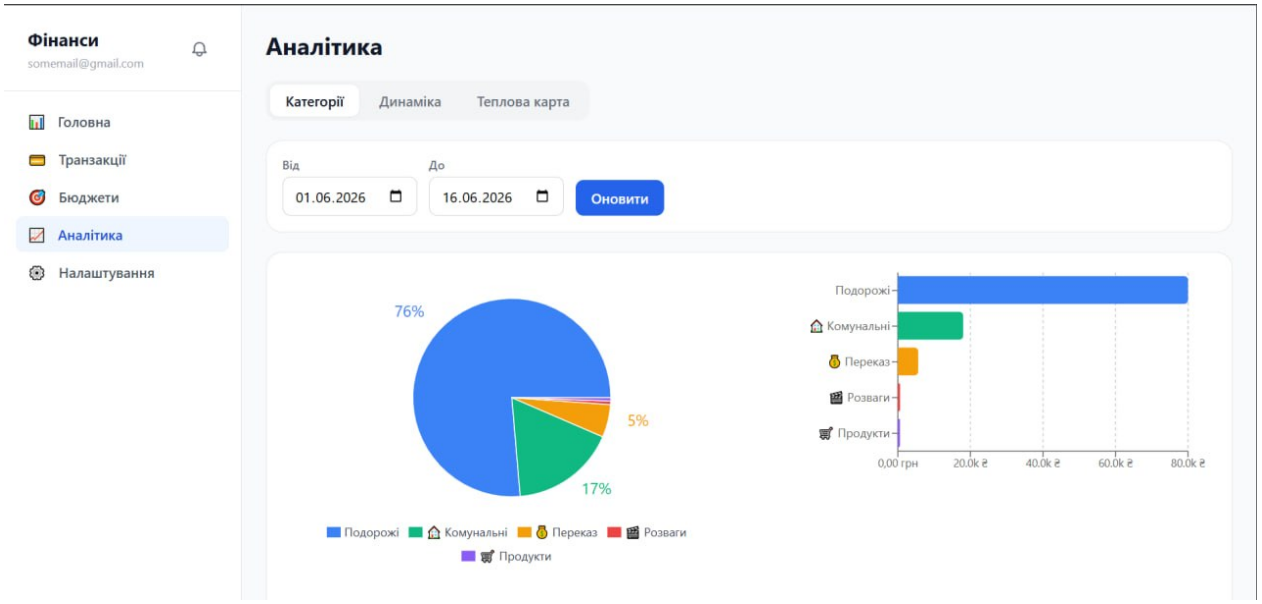

Рисунок 2.8 – Сторінка аналітики з візуалізацією фінансових даних

Для мультивалютних операцій аналітика підтримує конвертацію: якщо користувач має транзакції в різних валютах, серверна частина конвертує їх у обрану базову валюту за курсами PrivatBank API на дату транзакції. Кешування курсів зменшує кількість запитів до зовнішнього API.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Тестування, впровадження та підтримка є етапами життєвого циклу, що забезпечують стабільну роботу програмної системи. Даний розділ присвячено методологіям тестування, процесу впровадження та механізмам підтримки сервісу управління власними фінансами.

3.1 План тестування системи

План тестування враховує практичні вимоги та специфіку фінансового сервісу, де коректність обробки грошових операцій є критичною. Основна мета — забезпечити стабільне функціонування системи, правильність фінансових розрахунків та надійність інтеграцій з банківськими API.

Стратегія тестування ґрунтується на ітеративному підході: кожен модуль проходить перевірку одразу після реалізації. Для серверної частини застосовуються автоматизовані модульні тести з використанням pytest та бібліотеки unittest.mock для імітації зовнішніх API-запитів. Для клієнтської частини використовується Jest разом із React Testing Library [31].

Критерії приймання для функціонального тестування: відсутність критичних помилок у роботі основних модулів, правильність фінансових розрахунків з точністю до копійки, коректна обробка всіх MCC-кодів із довідника. Для тестів продуктивності: час відповіді API не перевищує 200 мс, відсутність витоків пам’яті при тривалій роботі. Для безпеки: відсутність SQL-ін’єкцій та XSS-вразливостей, належне шифрування банківських токенів.

3.2 Види та методи тестування

Функціональне тестування зосереджене на перевірці правильності роботи основних модулів. Серед об’єктів тестування: імпорт транзакцій через Monobank API (парсинг JSON-відповідей, дедуплікація, обробка rate limit); автоматична

категоризація (відповідність MCC-кодів категоріям, обробка невідомих кодів, збереження користувацьких правил); бюджетування (розрахунок витраченої суми, відсоток прогресу, сповіщення при перевищенні); аналітика (агрегація за категоріями, за місяцями, конвертація валют).

Інтеграційне тестування перевіряє взаємодію між компонентами: коректність авторизації через JWT при зверненні до захищених ендпоінтів, та взаємодію React-компонентів із REST API через Axios.

Для перевірки коректності REST API застосовується інтерактивна документація Swagger UI, згенерована на основі специфікації OpenAPI. Вона дозволяє виконувати запити до всіх ендпоінтів безпосередньо з браузера, контролювати структуру відповідей та коди статусів, а також перевіряти правильність роботи JWT-автентифікації через cookie. Це спрощує ручне тестування інтеграційної взаємодії між клієнтом та сервером і дає змогу валідувати контракт API без написання додаткового коду [27].

UX-тестування спрямоване на перевірку відповідності інтерфейсу макетам та зручності використання. Перевіряється адаптивність на різних пристроях, інтуїтивність навігації та інтерактивність графіків Recharts.

Тестування безпеки включає: перевірку неможливості доступу до даних інших користувачів через маніпуляцію з id у запитах; тестування на SQL-ін’єкції через параметри фільтрації; перевірку XSS через поля коментарів та описів категорій; валідацію JWT-токенів із підробленим підписом; перевірку шифрування токенів Monobank у базі даних [13].

3.3 Документування та тестування REST API

Окремим елементом тестування серверної частини є перевірка REST API за допомогою інтерактивної документації Swagger UI. Документація генерується на основі специфікації OpenAPI, описаної у модулі swagger_spec.py, та підключається до Flask-додатка через бібліотеку flasgger [30]. Інтерфейс доступний за маршрутом /docs, а машиночитна специфікація — за адресою /api/swagger.json.

Swagger UI групує ендпоінти за функціональними тегами (Auth, Accounts, Transactions, Categories, Budgets, Analytics, Balance, Rates, Webhooks), що відповідають модулям системи. Для кожного ендпоінта наведено HTTP-метод, шлях, опис, очікувані параметри запиту, схеми тіла та можливі коди відповідей. Це дозволяє розробнику швидко орієнтуватися у структурі API та перевіряти відповідність реалізації до проєктної специфікації.

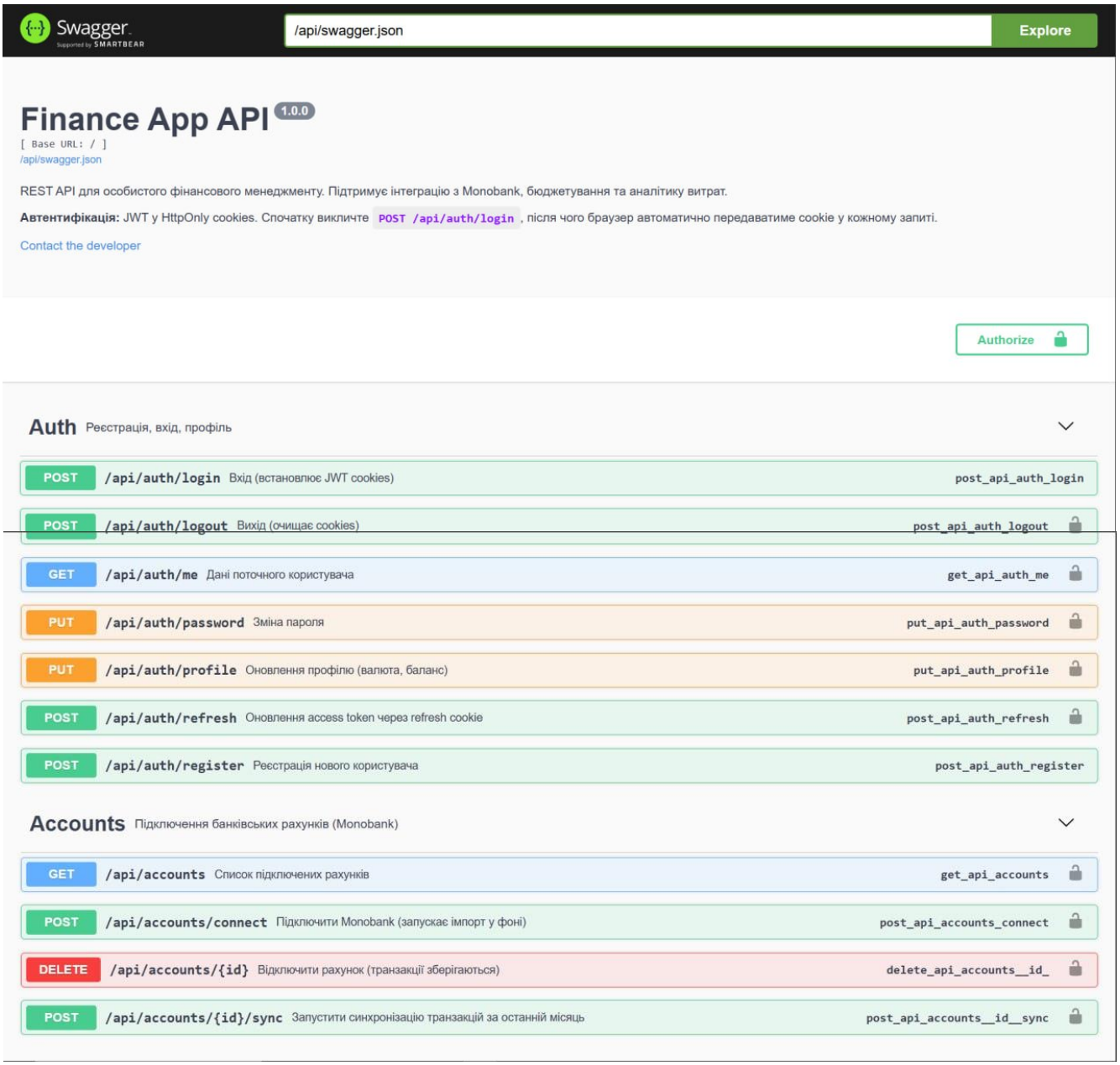

Рисунок 3.1 – Загальний вигляд інтерактивної документації Swagger UI

Swagger UI використовується для ручного функціонального тестування

ендпоінтів безпосередньо з браузера. За допомогою функції «Try it out» формується HTTP-запит із заданими параметрами, виконується звернення до сервера та відображається фактична відповідь разом із кодом статусу й заголовками. Така можливість дає змогу перевіряти сценарії без написання додаткового клієнтського коду: реєстрацію та вхід користувача, підключення банківського рахунку, отримання транзакцій, формування аналітики та роботу з бюджетами.

Автентифікація у Swagger UI реалізована через JWT-токени у HttpOnly cookie. Після виконання запиту POST /api/auth/login браузер автоматично зберігає cookie з access-токеном, які надалі додаються до кожного наступного запиту до захищених ендпоінтів. Для запитів, що змінюють стан, додатково передається CSRF-токен у заголовку X-CSRF-TOKEN. Таким чином Swagger UI дозволяє перевірити не лише бізнес-логіку, а й коректність механізмів авторизації та захисту від CSRF-атак.

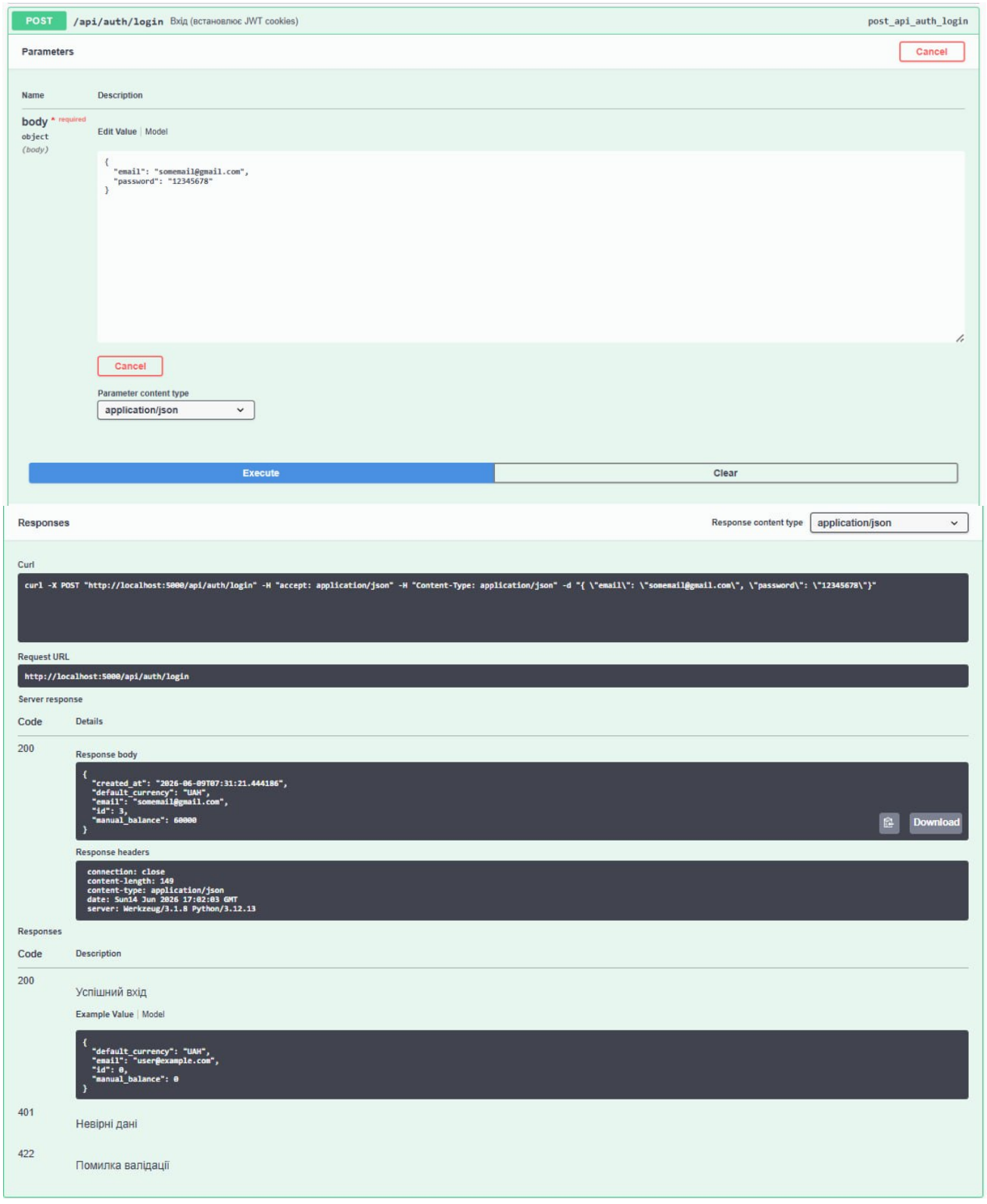

Рисунок 3.2 – Виконання тестового запиту до ендпоінта через Swagger UI

3.4 Тестування функціональних модулів

Тестування модуля імпорту транзакцій зосереджене на коректності парсингу відповідей Monobank API. Створено набір mock-відповідей із різними сценаріями: стандартна транзакція з MCC-кодом, переказ між картками, операція в іноземній валюті, повернення коштів. Перевірено дедуплікацію: повторний імпорт тих самих транзакцій не створює дублікатів у базі.

Тестування бюджетування перевіряє правильність розрахунків: при бюджеті 5000 грн на «Їжу» та витратах 4200 грн прогрес має становити 84%. Перевірено генерацію сповіщень: при досягненні 80% створюється попередження, при 100% — критичне сповіщення. Перевірено коректну роботу з від'ємними та нульовими значеннями.

Тестування аналітики перевіряє коректність агрегації: сума витрат за категоріями має дорівнювати загальній сумі витрат. Перевірено мультивалютну конвертацію: транзакція 100 USD конвертується за курсом PrivatBank API на дату операції.

3.5 Тестування інтерфейсу та адаптивності

Тестування інтерфейсу зосереджене на візуальній відповідності та адаптивності. Для перевірки адаптивності використовуються Chrome DevTools.

Виявлено та виправлено проблеми:

– графіки Recharts потребували додаткового налаштування ResponsiveContainer для коректного відображення на мобільних пристроях;

– таблиця транзакцій замінена на карткове представлення на екранах менше 768px;

– діалогові вікна додавання транзакцій адаптовані для повноекранного відображення на смартфонах.

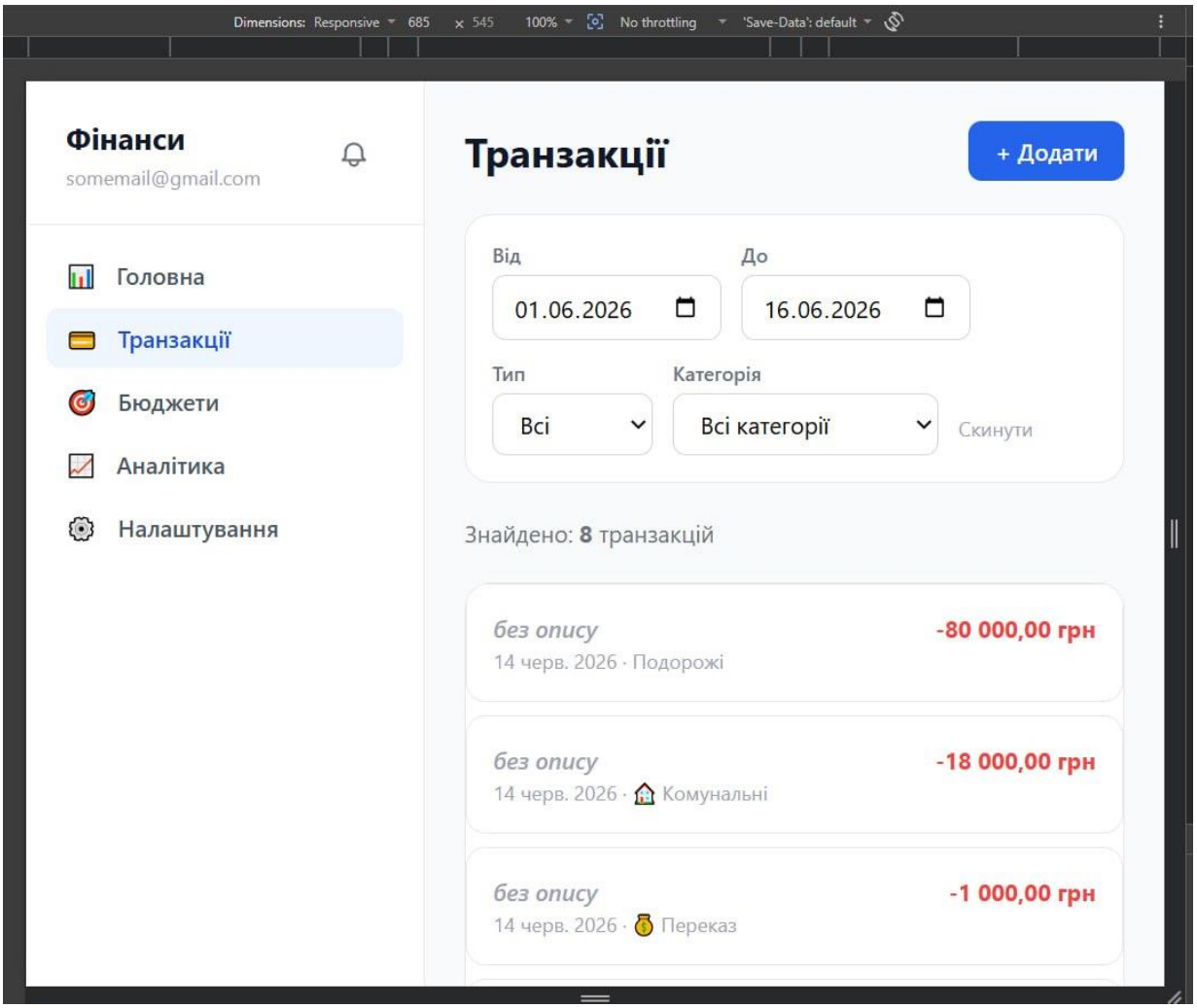

Рисунок 3.3 – Перевірка адаптивності інтерфейсу на різних роздільних здатностях

3.6 Аналіз результатів впровадження

Впровадження сервісу управління власними фінансами проходило серед групи тестових користувачів протягом двох тижнів. Результати підтверджують ефективність реалізованих рішень.

Імпорт транзакцій через Monobank API працює стабільно: середній час початкового імпорту за місяць — 65 секунд. Webhook-сповіщення надходять протягом 1–3 секунд після здійснення операції. Автоматична категоризація на основі MCC-кодів забезпечує точність категоризації відповідно до стандартних

категорій — решта транзакцій потребує ручної категоризації.

Аналітичний модуль отримав позитивні відгуки від тестових користувачів. Кругова діаграма витрат та лінійний графік динаміки виявились найбільш затребуваними типами візуалізації. Функція порівняння витрат між місяцями допомогла виявити аномальні витрати у категоріях.

Модуль бюджетування продемонстрував практичну цінність: користувачі, які встановили бюджети, змогли краще контролювати витрати завдяки сповіщенням про наближення до лімітів. Інтеграція з PrivatBank API для курсів валют забезпечила коректну мультивалютну аналітику.

Серед виявлених проблем: деякі MCC-коди не мали відповідності в довіднику; на мобільних пристроях графіки потребували оптимізації продуктивності. Ці проблеми було оперативно виправлено.

Отже, впровадження сервісу підтвердило працездатність обраної архітектури та технологічного стеку, а також практичну цінність автоматичного імпорту транзакцій та аналітичних інструментів.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У даному розділі висвітлено вимоги нормативно-правових актів України до проведення профілактичних медичних оглядів працівників, які у професійній діяльності використовують персональні комп'ютери, а також розглянуто соціально-політичні небезпеки, їхні види та характеристики. Питання розділу розкрито з урахуванням специфіки кваліфікаційної роботи, виконання якої передбачає тривалу роботу з екранними пристроями під час розробки програмного забезпечення сервісу управління фінансами приватних осіб.

4.1 Соціально-політичні небезпеки, їхні види та характеристики

Соціально-політичні небезпеки — це небезпеки, що виникають у соціальному середовищі внаслідок загострення суперечностей суспільних, міждержавних та міжнаціональних відносин і реалізуються у формі конфліктів соціального та політичного характеру [40, 41]. Джерелом таких небезпек є сама людина та створені нею соціальні інститути, а об'єктом ураження виступають життя, здоров'я, права і свободи людини, матеріальні та духовні цінності суспільства. Особливістю соціально-політичних небезпек є їхня здатність до швидкого поширення, ескалації та поєднання з небезпеками техногенного і природного характеру.

До небезпек політичного характеру належать війни та збройні конфлікти, тероризм, екстремізм, сепаратизм, масові заворушення, конфлікти на міжнаціональному та міжконфесійному ґрунті. До небезпек соціального характеру належать злочинність, шахрайство, алкоголізм, наркоманія, бродяжництво, суїцидальна поведінка та соціальна напруженість, спричинена безробіттям і зниженням рівня життя населення [40]. За масштабом дії соціально-політичні небезпеки класифікуються на глобальні, національні, регіональні та локальні; за тривалістю — на короткочасні та довготривалі.

Найнебезпечнішою формою соціально-політичних небезпек є війна та

збройний конфлікт. Їхніми характерними наслідками є загибель і поранення людей, руйнування житлової та критичної інфраструктури, вимушене переміщення населення, порушення систем життєзабезпечення та погіршення психічного здоров’я суспільства. Кодекс цивільного захисту України визначає правові засади захисту населення і територій від надзвичайних ситуацій, у тому числі воєнного характеру: оповіщення про загрозу, укриття населення у захисних спорудах цивільного захисту, евакуаційні заходи, інженерний, медичний та психологічний захист [37]. В умовах правового режиму воєнного стану виконання робіт з розробки програмного забезпечення організовується з обов’язковим реагуванням на сигнали оповіщення та переміщенням працівників до укриттів на час повітряної тривоги.

Тероризм відповідно до Закону України «Про боротьбу з тероризмом» є суспільно небезпечною діяльністю, яка полягає у свідомому, цілеспрямованому застосуванні насильства шляхом захоплення заручників, підпалів, убивств, тортур, залякування населення та органів влади або вчинення інших посягань на життя чи здоров’я людей для досягнення злочинних цілей [38]. Характерними ознаками тероризму є раптовість, демонстративність, спрямованість проти невизначеного кола осіб та використання страху як інструмента впливу. Окремим сучасним різновидом є кібертероризм та кібератаки на об’єкти критичної інформаційної інфраструктури, зокрема на банківську систему, що безпосередньо стосується сфери функціонування фінансових вебсервісів.

Серед небезпек соціального характеру для теми кваліфікаційної роботи найбільше значення мають злочинність у фінансовій сфері та шахрайство, зокрема фішинг і методи соціальної інженерії, спрямовані на заволодіння коштами та персональними даними користувачів банківських сервісів. Протидія даним небезпекам у розробленій системі реалізована на архітектурному рівні: застосовано шифрування банківських токенів, авторизацію на основі JWT, валідацію вхідних даних та захист від SQL-ін’єкцій і XSS-вразливостей. Таким чином, технічні рішення кваліфікаційної роботи безпосередньо знижують ризик реалізації соціальних небезпек в інформаційній сфері щодо користувачів сервісу.

Захист від соціально-політичних небезпек здійснюється на державному та

особистісному рівнях. На державному рівні основи протидії визначає Закон України «Про національну безпеку України», який встановлює засади державної політики у сферах національної безпеки і оборони, спрямованої на захист людини і громадянина, суспільства і держави від загроз [39]. На рівні особистості захист забезпечується правовою обізнаністю, критичним ставленням до інформації та протидією інформаційно-психологічним маніпуляціям, дотриманням правил поведінки у місцях масового перебування людей і під час надзвичайних ситуацій, володінням навичками домедичної допомоги та дотриманням правил цифрової гігієни під час роботи з фінансовими даними.

Отже, у розділі визначено вимоги законодавства України до проведення попередніх і періодичних медичних оглядів працівників, які працюють з персональними комп'ютерами, та встановлено порядок їх організації стосовно умов виконання кваліфікаційної роботи. Розглянуто види та характеристики соціально-політичних небезпек і сформульовано заходи захисту від них на державному та особистісному рівнях. Дотримання наведених вимог забезпечує збереження здоров'я і працездатності розробника програмного забезпечення та безпеку користувачів створеного фінансового сервісу.

4.2 Вимоги до профілактичних медичних оглядів для працівників ПК

Розробка програмного забезпечення сервісу управління фінансами приватних осіб з використанням технологій Flask, React та банківських API виконується із застосуванням персонального комп'ютера протягом усього робочого дня. Тривала робота з екранними пристроями супроводжується впливом на організм працівника низки шкідливих виробничих факторів: напруження зорового аналізатора, статичного навантаження на опорно-руховий апарат, нервово-емоційного напруження та електромагнітних полів. З огляду на це профілактичні медичні огляди розробників програмного забезпечення є обов'язковим елементом системи охорони праці та проводяться відповідно до вимог чинного законодавства України [32, 33].

Стаття 17 Закону України «Про охорону праці» зобов’язує роботодавця за власні кошти забезпечити фінансування та організувати проведення попереднього і періодичних медичних оглядів працівників, зайнятих на важких роботах, роботах із шкідливими чи небезпечними умовами праці або таких, де є потреба у професійному доборі [32]. Аналогічну норму закріплює стаття 169 Кодексу законів про працю України [34]. За результатами періодичних медичних оглядів роботодавець забезпечує проведення відповідних оздоровчих заходів, а у разі виявлення медичних протипоказань — переведення працівника на іншу роботу.

Порядок проведення медичних оглядів працівників певних категорій затверджено наказом Міністерства охорони здоров’я України від 21.05.2007 № 246 [33]. Попередній медичний огляд проводиться під час прийняття на роботу з метою визначення стану здоров’я працівника, реєстрації вихідних об’єктивних показників та встановлення можливості виконання професійних обов’язків без погіршення стану здоров’я в умовах дії конкретних шкідливих і небезпечних факторів виробничого середовища та трудового процесу. Періодичні медичні огляди забезпечують динамічне спостереження за станом здоров’я працівників, своєчасне виявлення початкових форм професійних захворювань і ранніх ознак впливу шкідливих факторів, а також вирішення питання щодо можливості продовження роботи у відповідних умовах.

Організаційне забезпечення медичних оглядів покладається на роботодавця. Щорічно складається поіменний список працівників, які підлягають періодичним медичним оглядам, визначаються строки їх проведення та укладається договір із закладом охорони здоров’я. Результати огляду заносяться до Картки працівника та оформлюються заключним актом, на підставі якого роботодавець планує і реалізує оздоровчі та профілактичні заходи [33]. Підставою для включення розробника програмного забезпечення до списку є наявність у його трудовому процесі факторів, передбачених Переліком шкідливих та небезпечних факторів виробничого середовища і трудового процесу, зокрема зорово напружених робіт, пов’язаних із спостереженням за екраном відеотермінала, та значного нервово-емоційного навантаження.

Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 встановлюють, що працівники, які професійно працюють з відеотерміналами, проходять обов'язкові попередній та періодичні медичні огляди за участю терапевта, невропатолога та офтальмолога [36]. Періодичність оглядів для даної категорії працівників становить один раз на два роки. Під час огляду офтальмолог визначає гостроту зору, рефракцію та стан акомодаційного апарату ока, оскільки саме зоровий аналізатор зазнає найбільшого навантаження при тривалій роботі з екраном. Особи, у яких виявлено медичні протипоказання, визначені санітарними правилами, до постійної роботи з відеотерміналами не допускаються.

Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, затверджені наказом Міністерства соціальної політики України від 14.02.2018 № 207, передбачають проведення за рахунок роботодавця обстеження зору та стану органів зору працівника перед початком роботи з екранними пристроями, періодично у процесі трудової діяльності та у разі виникнення скарг на порушення зору, які можуть бути наслідком роботи з екранними пристроями [35]. За результатами такого обстеження працівникові за потреби надаються спеціальні засоби коригування зору, призначені для відповідної роботи.

Профілактичні медичні огляди доповнюються раціональною організацією режиму праці та відпочинку. Для розробників програмного забезпечення встановлюються регламентовані перерви тривалістю 15 хвилин через кожну годину безперервної роботи з екранним пристроєм, під час яких виконуються вправи для очей та опорно-рухового апарату [36, 42]. Поєднання періодичних медичних оглядів, обстежень зору та регламентованих перерв запобігає розвитку професійно зумовлених порушень здоров'я, зокрема комп'ютерного зорового синдрому, захворювань хребта та синдрому зап'ястного каналу, і забезпечує збереження працездатності розробника протягом усього періоду створення та супроводу програмної системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи досягнуто поставленої мети — розроблено та впроваджено сервіс управління фінансами приватних осіб, який забезпечує автоматизований облік транзакцій, категоризацію витрат, формування бюджетів та візуалізацію фінансової аналітики. Усі сформульовані у вступі задачі виконано повністю.

Проведено аналіз предметної області управління особистими фінансами та огляд існуючих рішень. Встановлено, що вбудований функціонал банківських застосунків обмежений транзакціями одного банку та не забезпечує повноцінного бюджетування, тоді як комерційні аналоги мають високу вартість підписки й не адаптовані під українського користувача. Це підтвердило доцільність створення відкритого гнучкого сервісу на основі українських банківських API.

Визначено функціональні та нефункціональні вимоги до системи з урахуванням потреб цільової аудиторії, описано акторів та варіанти використання. Сформульовано вимоги до продуктивності, безпеки та адаптивності інтерфейсу, які стали основою для подальшого проєктування й тестування.

Розроблено архітектуру програмної системи на основі клієнт-серверної моделі з чітким розділенням на три рівні. Серверна частина реалізована як REST API на фреймворку Flask, клієнтська — як односторінковий застосунок на React. Для зберігання даних використано PostgreSQL, для кешування та фонових задач — Redis і Celery, що забезпечило масштабованість і відмовостійкість рішення.

Реалізовано інтеграцію з Monobank open API для автоматичного імпорту банківських транзакцій, зокрема через механізм webhook-сповіщень, які надходять протягом 1–3 секунд після здійснення операції. Допоміжну інтеграцію з PrivatBank API реалізовано для отримання актуальних курсів валют, що забезпечило коректну мультивалютну аналітику.

Розроблено модуль категоризації витрат на основі MCC-кодів, модуль бюджетування з генерацією сповіщень про наближення до лімітів та аналітичний модуль із візуалізацією фінансових даних. Створено адаптивний інтерфейс

користувача з використанням React та бібліотеки Recharts, який коректно відображається на пристроях різної роздільної здатності.

Проведено комплексне тестування системи, що охопило модульне, інтеграційне, UX-тестування та тестування безпеки. Для документування й перевірки REST API застосовано інтерактивну документацію Swagger UI на основі специфікації OpenAPI, що дозволило валідувати контракт API та механізми JWT-автентифікації. Впровадження серед групи тестових користувачів протягом двох тижнів підтвердило стабільність роботи системи, ефективність автоматичного імпорту транзакцій та практичну цінність аналітичних і бюджетних інструментів.

Практичне значення роботи полягає у створенні готового до використання сервісу, який автоматизує облік фінансів та підтримує прийняття фінансових рішень українськими користувачами. Результати дослідження пройшли апробацію на науково-технічній конференції. Перспективними напрямами подальшого розвитку є розширення переліку підтримуваних банківських API, впровадження інтелектуальних рекомендацій на основі аналізу витрат та розробка мобільного застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gomber P., Koch J.-A., Siering M. Digital Finance and FinTech: current research and future research directions. Journal of Business Economics. 2017. Vol. 87, No. 5. P. 537–580.

2. Suryono R. R., Budi I., Purwandari B. Challenges and trends of financial technology (Fintech): a systematic literature review. Information. 2020. Vol. 11, No. 12. Article 590. P. 1–20.

3. Zachariadis M., Ozcan P. The API economy and digital transformation in financial services: the case of open banking. SWIFT Institute Working Paper. 2017. No. 2016-001. 29 p.

4. Toshl Finance. Personal finance management service. Official website. URL: https://toshl.com/ (дата звернення: 07.06.2026).

5. YNAB (You Need A Budget). Budgeting method and application. Official website. URL: https://www.ynab.com/ (дата звернення: 07.06.2026).

6. Monobank open API. Документація для розробників. URL: https://api.monobank.ua/docs/ (дата звернення: 07.06.2026).

7. ПриватБанк. API курсів валют. URL: https://api.privatbank.ua/ (дата звернення: 07.06.2026).

8. Fielding R. T., Taylor R. N. Principled design of the modern Web architecture. ACM Transactions on Internet Technology. 2002. Vol. 2, No. 2. P. 115–150.

9. Neumann A., Laranjeiro N., Bernardino J. An analysis of public REST web service APIs. IEEE Transactions on Services Computing. 2021. Vol. 14, No. 4. P. 957–970.

10. Arner D. W., Barberis J., Buckley R. P. The evolution of Fintech: a new post-crisis paradigm? Georgetown Journal of International Law. 2016. Vol. 47, No. 4. P. 1271–1319.

11. Milian E. Z., Spinola M. de M., Carvalho M. M. de. Fintechs: a literature review and research agenda. Electronic Commerce Research and Applications. 2019. Vol. 34. Article 100833. P. 1–21.

12. Tilkov S., Vinoski S. Node.js: using JavaScript to build high-performance network programs. IEEE Internet Computing. 2010. Vol. 14, No. 6. P. 80–83.

13. Mendoza A., Gu G. Mobile application web API reconnaissance: web-to-mobile inconsistencies and vulnerabilities. Proceedings of the IEEE Symposium on Security and Privacy. 2018. P. 756–769.

14. Wang Y., Kogan A., Boland M. Designing confidentiality-preserving Blockchain-based transaction processing systems. International Journal of Accounting Information Systems. 2018. Vol. 30. P. 1–18.

15. Sarikaya A., Correll M., Bartram L., Tory M., Fisher D. What do we talk about when we talk about dashboards? IEEE Transactions on Visualization and Computer Graphics. 2019. Vol. 25, No. 1. P. 682–692.

16. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. Sebastopol : O'Reilly Media, 2013. 406 p.

17. Lee I., Shin Y. J. Fintech: ecosystem, business models, investment decisions, and challenges. Business Horizons. 2018. Vol. 61, No. 1. P. 35–46.

18. Few S. Information Dashboard Design: Displaying Data for At-a-Glance Monitoring. 2nd ed. Burlingame : Analytics Press, 2013. 274 p.

19. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education, 2016. 816 p.

20. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol : O'Reilly Media, 2018. 316 p.

21. Flask Documentation. Pallets Projects. URL: https://flask.palletsprojects.com/ (дата звернення: 07.06.2026).

22. React Documentation. Meta Open Source. URL: https://react.dev/ (дата звернення: 07.06.2026).

23. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/ (дата звернення: 07.06.2026).

24. SQLAlchemy Documentation. The SQLAlchemy authors. URL: https://docs.sqlalchemy.org/ (дата звернення: 07.06.2026).

25. Redis Documentation. Redis Ltd. URL: https://redis.io/docs/ (дата звернення: 12.06.2026).

26. Celery: Distributed Task Queue. Documentation. URL: https://docs.celeryq.dev/ (дата звернення: 07.06.2026).

27. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force, 2015. 30 p. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 07.06.2026).

28. Recharts. A composable charting library built on React components. URL: https://recharts.org/ (дата звернення: 07.06.2026).

29. Axios HTTP client. Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 07.06.2026).

30. OpenAPI Specification v2.0 (Swagger). SmartBear Software. URL: https://swagger.io/specification/v2/ (дата звернення: 07.06.2026).

31. pytest: helps you write better programs. Documentation. URL: https://docs.pytest.org/ (дата звернення: 07.06.2026).

32. Закон України «Про охорону праці» від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 07.06.2026).

33. Про затвердження Порядку проведення медичних оглядів працівників певних категорій : наказ Міністерства охорони здоров'я України від 21.05.2007 № 246. URL: https://zakon.rada.gov.ua/laws/show/z0846-07 (дата звернення: 07.06.2026).

34. Кодекс законів про працю України від 10.12.1971 № 322-VIII. URL: https://zakon.rada.gov.ua/laws/show/322-08 (дата звернення: 07.06.2026).

35. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : наказ Міністерства соціальної політики України від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 07.06.2026).

36. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин : затв. постановою Головного державного санітарного лікаря України від 10.12.1998 № 7. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98 (дата звернення: 07.06.2026).

37. Кодекс цивільного захисту України від 02.10.2012 № 5403-VI. URL: https://zakon.rada.gov.ua/laws/show/5403-17 (дата звернення: 07.06.2026).

38. Закон України «Про боротьбу з тероризмом» від 20.03.2003 № 638-IV. URL: https://zakon.rada.gov.ua/laws/show/638-15 (дата звернення: 07.06.2026).

39. Закон України «Про національну безпеку України» від 21.06.2018 № 2469-VIII. URL: https://zakon.rada.gov.ua/laws/show/2469-19 (дата звернення: 07.06.2026).

40. Желібо Є. П., Заверуха Н. М., Зацарний В. В. Безпека життєдіяльності : навч. посіб. Київ : Каравела, 2011. 344 с.

41. Бедрій І. Я., Нечай В. Я. Безпека життєдіяльності : навч. посіб. Львів : Магнолія 2006, 2007. 499 с.

42. Основи охорони праці : підручник / за ред. К. Н. Ткачука, М. О. Халімовського. Київ : Основа, 2006. 448 с.

43. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 45 с. URL: http://elartu.tntu.edu.ua/handle/lib/50317 (дата звернення: 07.07.2026).

ДОДАТКИ

Додаток А. Тези конференції

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 004.42:004.738.5:336.7
Берестень М. – ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СЕРВІСУ УПРАВЛІННЯ ФІНАНСАМИ ПРИВАТНИХ ОСІБ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ FLASK, REACT ТА БАНКІВСЬКИХ API

Науковий керівник: к.т.н., доц. Багрій-Заяць О. А.

Beresten M.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF A PERSONAL FINANCE MANAGEMENT SERVICE SOFTWARE USING FLASK, REACT AND BANKING APIs

Supervisor: O. A. Bahrii-Zaiats, PhD in Engineering, Associate Professor

Ключові слова: Flask, React, Monobank API
Keywords: Flask, React, Monobank API

В умовах активної цифровізації фінансових послуг та зростання кількості безготівкових операцій постає потреба в автоматизованих засобах управління особистими фінансами. Більшість існуючих рішень або обмежені вбудованим функціоналом банківського додатка, або є комерційними сервісами без інтеграції з українськими банками. Актуальність роботи полягає в розробці відкритого фінансового сервісу, який поєднує автоматичний імпорт транзакцій та інтелектуальну категоризацію.

Об’єктом дослідження є процес створення веб-сервісу для управління особистими фінансами із використанням сучасних веб-технологій та банківських API. Предметом дослідження є методи та технології розробки фінансового веб-сервісу.

Розроблена клієнт-серверна архітектура поєднує серверну частину на Flask, клієнтську частину на React та реляційну базу даних PostgreSQL. Серверна частина організована за модульним принципом із використанням Flask Blueprints, що забезпечує чіткий розподіл функціональних областей: автентифікація, управління рахунками, транзакції, категорії, бюджети та аналітика.

Інтеграція з Monobank open API забезпечує автоматичний імпорт транзакцій з підтримкою webhook-сповіщень для отримання нових операцій у реальному часі. Допоміжну роль відіграє PrivatBank API, який використовується для отримання актуальних та архівних курсів валют.

Розроблений сервіс продемонстрував практичну цінність як інструмент для систематизації фінансового обліку та підтримки прийняття обґрунтованих фінансових рішень.

Список джерел

1. Monobank open API. Документація. URL: https://api.monobank.ua/docs/
2. PrivatBank API. Курси валют. URL: https://api.privatbank.ua/
3. Flask. Official Documentation. URL: https://flask.palletsprojects.com/
4. React. Official Documentation. URL: https://react.dev/

Додаток Б.

Посилання на GitHub репозиторій – https://github.com/Reforde/Personal-finance-manager