НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2026 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Вебдодаток для обліку і аналітики операцій по банківських картках користувача ”

Виконав: студент 4 курсу, групи ТР-21
Пугач Артем Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Олександр Волков _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент доктор філософії, ст. викладач Ольга ВЛАСЕНКО _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2026

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
“___” _____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____ Пугачу Артему Олександровичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи “Вебдодаток для обліку і аналітики операцій по банківським картках користувача ”
Науковий керівник Волков Олександр Володимирович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “ 28 ” травня 2026 року № 1992-с
2. Термін подання студентом роботи 04.06.2026
3. Вихідні дані до роботи мова програмування JavaScript, бібліотека React.js, платформа Node.js, фреймворк Express.js, СУБД PostgreSQL, середовище розробки Visual Studio Code.
4. Перелік питань, які потрібно розробити _____

1) провести аналіз предметної області персонального фінансового обліку та фінансової аналітики
2) виконати аналіз існуючих програмних рішень для обліку та аналітики персональних фінансів
3) визначити функціональні та нефункціональні вимоги до вебдодатка

4) побудувати архітектуру системи та структури бази даних
5) реалізувати підсистему обліку фінансових операцій, банківських карток, категорій і планування бюджету
6) реалізувати модуль аналітики, статистики та графічної візуалізації фінансових даних
7) провести тестування вебдодатка та оцінити результати його роботи
5. Орієнтований перелік ілюстративного матеріалу схема архітектури проєкту, схема структури бази даних, скріншоти роботи користувача з вебзастосунком.
6. Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	13.09.2025	Виконано
2.	Аналіз предметної області персонального фінансового обліку та існуючих програмних рішень та визначення вимог до системи	20-26.04.26	Виконано
3.	Розробка архітектури системи та структури бази даних	27.04-03.05.26	Виконано
4.	Розробка клієнтської та серверної частини	04-07.05.26	Виконано
5.	Реалізація функціоналу обліку фінансових операцій, банківських карток, категорій, аналітики та планування бюджету	08-14.05.26	Виконано
6.	Оформлення пояснювальної записки	15.05-01.06.26	Виконано
7.	Захист програмного забезпечення	15.05.26	Виконано
8.	Передзахист	04.06.26	Виконано
9.	Захист	19.06.26	Виконано

Студент _____ Артем ПУГАЧ
(підпис) (прізвище та ініціали)

Керівник _____ Олександр ВОЛКОВ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 70 сторінках, містить 20 ілюстрацій, 1 додаток, 20 джерел в переліку посилань.

Метою дипломної роботи є розробка такого програмного продукту як «Вебдодаток для обліку і аналітики операцій по банківських картках користувача». Даний програмний продукт дозволяє користувачам полегшити ведення обліку фінансових операцій, аналізувати їх витрати і доходи, а також переглядати фінансову статистику у вигляді графіків, отримувати фінансовий рейтинг місяця та отримувати фінансові рекомендації на основі власної активності.

Клієнтська частина програмного продукту була розроблена за допомогою бібліотеки React.js, для надсилання HTTP-запитів між клієнтською та серверною частинами вебдодатка використовується Axios. Для побудови графіків і візуалізації фінансової статистики користувача використовувалась бібліотека Recharts. Серверна частина розроблена за допомогою Node.js, для реалізації REST API, маршрутизації та роботи з логікою серверної частини використовується фреймворк Express.js, для збереження інформації про користувачів, банківські картки, категорії та фінансові операції використовувалася система керування базами даних PostgreSQL, а для взаємодії Node.js із PostgreSQL використовувалася ORM-бібліотека Sequelize ORM. Для хешування паролів використовувалася бібліотека bcryptjs.

Ключові слова: ВЕБДОДАТОК, АНАЛІТИКА, ОБЛІК ФІНАНСІВ, ФІНАНСОВІ ОПЕРАЦІЇ, ДІАГРАМИ, ВИТРАТИ, ДОХОДИ.

ABSTRACT

The thesis is 70 pages long, contains 20 illustrations, 1 appendix, and 20 sources in the list of references.

The purpose of the thesis is to develop such a software product as "Web application for accounting and analytics of user bank card transactions". This software product allows users to facilitate accounting of financial transactions, analyze their expenses and income, as well as view financial statistics in the form of graphs, receive a financial rating of the month and receive financial recommendations based on their own activity.

The client part of the software product was developed using the React.js library, Axios is used to send HTTP requests between the client and server parts of the web application. The Recharts library was used to build graphs and visualize user financial statistics. The server part was developed using Node.js, the Express.js framework is used to implement the REST API, routing and work with the logic of the server part, the PostgreSQL database management system was used to store information about users, bank cards, categories and financial transactions, and the Sequelize ORM library was used to interact with Node.js and PostgreSQL. The bcryptjs library was used for password hashing.

Keywords: WEB APPLICATION, ANALYTICS, FINANCIAL ACCOUNTING, FINANCIAL TRANSACTIONS, CHARTS, EXPENSES, INCOME.

ЗМІСТ

ВСТУП

На сьогодні через розвиток цифрових технологій та популярність банківських карток, в житті людини збільшується кількість фінансових операцій, які вона виконує, і контролювати витрати та доходи стає важче з кожним днем. Саме тому виникає потреба у зручному додатку, який дозволить зберігати інформацію про фінансові операції, систематизувати її та надавати користувачу наочну статистику й аналітичну інформацію.

.Метою дипломної роботи є розробка такого вебдодатка, який допоможе користувачу вести облік фінансових операцій по його банківським картам, контролювати свої витрати і доходи та переглядати статистику й аналітичну інформацію у графічному вигляді.

Для досягнення мети було поставлено наступні завдання:

1) провести аналіз предметної області персонального фінансового обліку та фінансової аналітики;

2) виконати аналіз існуючих програмних рішень для обліку та аналітики персональних фінансів;

3) визначити функціональні та нефункціональні вимоги до вебдодатка;

4) побудувати архітектуру системи та структури бази даних;

5) реалізувати підсистему обліку фінансових операцій, банківських карток, категорій і планування бюджету;

6) реалізувати модуль аналітики, статистики та графічної візуалізації фінансових даних;

7) провести тестування вебдодатка та оцінити результати його роботи.

Структура роботи включає вступ, чотири основні розділи, що описують аналіз предметної області та існуючих додатків для обліку фінансів, проєктування системи, розробку програмного забезпечення, тестування та демонстрацію роботи системи. Обсяг роботи складає 70 сторінки, містить 20 ілюстрацій, 1 додаток, 20 джерел в переліку посилань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ СИСТЕМ ОБЛІКУ ФІНАНСІВ

Управління персональними фінансами з кожним днем все більше стає актуальним і складним завданням, причиною цього є активне використання банківських карток та електронних платіжних систем, непередбачуваність економічної ситуації, зростання витрат на життєзабезпечення і потреба користувачів у контролі власних витрат і доходів. Через ці причини, традиційні методи обліку та аналізу фінансів стають малоефективними і незручними в сучасних умовах. Сучасні вебдодатки для управління фінансами відкривають нові можливості для користувачів. Вони надають можливість користувачам контролювати свої доходи та витрати, аналізувати їх та переглядати фінансову статистику. Аналіз цих додатків, дає можливість визначити їх переваги та недоліки і сформувати вимоги до розроблюваного програмного продукту.

У даному розділі буде проаналізовано особливості систем обліку та аналітики фінансових операцій, проведено огляд існуючих додатків, визначено основні аспекти постановки задачі створення вебдодатка, а також визначено функціональні та нефункціональні вимоги до системи.

1.1 Особливості систем обліку та аналітики фінансових операцій

Кожного дня люди здійснюють велику кількість фінансових операцій. До таких операцій відносяться витрати на товари та послуги, отримання доходів у вигляді зарплати або стипендії, перекази коштів, оплата підписок та сервісів, покупки в інтернеті, оплата комунальних послуг тощо.

Фінансові операції є основою систем фінансового обліку. Фінансовий облік передбачає систематичне фіксування, впорядкування та узагальнення інформації про фінансові операції користувача [1]. Коли кількість операцій стає дуже великою, виникає потреба у їхній систематизації та контролі.

Щоб вирішити такі задачі, використовують системи обліку та аналітики фінансів. Вони дозволяють користувачу додавати доходи і витрати, бачити історію фінансових операцій та чітко розуміти напрями основних витрат і джерела основних доходів. Також такі системи дозволяють користувачу аналізувати фінансову активність і переглядати статистику.

Фінансова аналітика передбачає збір, аналіз та інтерпретацію фінансових даних для виявлення тенденцій і підтримки прийняття фінансових рішень [2]. Завдяки фінансовій аналітиці користувач має змогу аналізувати свою динаміку витрат і доходів за певний період часу та визначати категорії з найбільшими витратами.

Використання засобів візуалізації даних дозволяє спростити сприйняття складної фінансової інформації, виявляти закономірності та тенденції зміни фінансових показників, а також підвищує ефективність аналізу даних користувачем [3].

Сучасні системи представлені у вигляді вебдодатків або мобільних застосунків. Вони надають користувачеві зручний адаптивний інтерфейс, швидкий доступ до фінансової інформації, можливість зручного перегляду інформації у вигляді таблиць, діаграм і графіків а також мають засоби захисту даних.

Традиційні методи фінансового обліку такі, як ручне ведення записів або використання електронних таблиць втрачають актуальність через ускладнення контролю витрат та відсутності аналізу і статистики, а також не забезпечує наочності фінансової інформації.

1.2 Аналіз існуючих додатків для управління фінансами

На сьогоднішній день існує багато додатків для ведення обліку та аналізу персональних фінансів користувача. Кожен з них має свої переваги і недоліки. Вони доступні у вигляді мобільного за стосунку, вебверсії та декстопній версії. Для проведення аналізу існуючих додатків було розглянуто такі програмні продукти: «Budget Flow», «Money Flow», «Spendee» та «Monefy».

1.2.1 Огляд додатка «Budget Flow»

«Budget Flow» — це мобільний додаток призначений для відстеження доходів і витрат й керування ними. Додаток швидко налаштовується, не потребує реєстрації та може використовуватися офлайн. Даний додаток створений для тих, хто хоче ефективно керувати своїми фінансами, зберегти високий рівень безпеки та конфіденційності даних [4]. Система доступна як на iOS так і на Android, проте недоступна у форматі вебдодатка. Додаток є доволі популярним та має позитивні відгуки серед користувачів.

Функціональні можливості програми передбачають:

— управління фінансами. Користувач може відстежувати свої рахунки, баланси та транзакції. Має багато опцій для сортування, пошуку та фільтрації а також може налаштувати часовий період для всього додатку;

— створення рахунків та категорій. Користувач має можливість створювати кілька рахунків і категорій та організовувати категорії в папки.

— управління транзакціями. Користувач має можливість створювати одноразові та регулярні транзакції. Передбачена автоматична конвертація валют. Користувач може додати нотатки до транзакції та отримувати нагадування про заплановані транзакції;

— планування бюджету. Система забезпечує планування бюджету для категорій, а також вибір різних інтервалів повторення.

— безпечне зберігання даних користувача. Система підтримує локальне використання та засоби біометричної автентифікації.

— графічне представлення інформації. Користувач може побачити зміни витрат і доходів у вигляді кругової та стовпчастої діаграм.

Основними перевагами системи є зрозумілий сучасний інтерфейс, аналітика представлена у вигляді діаграм, планування бюджету, можливість створення регулярних транзакцій та створення запланованих транзакцій.

Основні недоліки програми полягають у слабкому аналітичному модулі. Система забезпечує лише базові засоби фінансової аналітики, що включають в себе перегляд змін витрат та доходів, витрат по категоріям, та аналітика за вибраний період часу. Графічна статистика представлена лише для змін витрат і доходів, що обмежує можливості для наочного аналізу фінансової інформації.

1.2.2 Огляд додатка «Money Flow»

«Money Flow» — це мобільний додаток призначений для швидкого планування, обліку і контролю витрат та доходів [5]. Він надає можливість проводити планування з підтримкою багатьох валют і зручного інтерфейсу. Додаток доступний тільки для платформ iOS та macOS.

Функціональні можливості програми передбачають:

— облік витрат і доходів. Користувач може додавати фінансові операції, налаштовувати рахунки, категорії та обирати іконки і кольори для зручності;

— налаштування бюджету. Система надає гнучкі налаштування бюджету, можливість відстежувати витрати за категоріями та отримування інтерактивних звітів, а також користувач може створити цілі і бачити прогрес накопичення для кожної цілі;

— можливість введення спільного обліку для сім’ї та команди;

— передбачено автоматичну конвертацію валют;

— графічне представлення інформації. Користувач може переглядати у вигляді стовпчастої або кругової діаграм інформацію про доходи та прибутки категорій, міток, одержувачів та місць.

Основними перевагами є планування бюджету та цілей і можливість переглядати прогрес накопичення по ним. Також перевагою є зручний інтерфейс, простота у створенні транзакції де користувач може надати доволі змістовну інформацію про транзакцію вказавши категорію, рахунок, дату, мітки, місце, нотатку і одержувача.

Основні недоліки програми полягають у обмеженій кількості засобів деталізованої фінансової статистики. Система має обмежені можливості формування детальної аналітики , а також відсутні аналітичні висновки на основі активності користувача. Ще одним із недоліків є відсутність реалізації для платформи Android.

1.2.3 Огляд додатка «Spendee»

«Spendee» — це додаток призначений для керування особистими фінансами, який дозволяє відстежувати доходи та витрати користувача, аналізувати фінансові звички та планувати бюджет [6]. Система доступна для мобільних пристроїв як для iOS, так і для Android, а також є вебверсія додатку.

Функціональні можливості програми передбачають:

— ведення обліку доходів і витрат;

— створення бюджету та категорій. Користувач має можливість створювати бюджети для різних категорій витрат, встановлювати ліміти, а також отримувати нагадування у випадку досягнення межі. Користувач може побачити рекомендації щодо щоденної норми витрат, щоб дотримуватися фінансового плану;

— система надає можливість спільного ведення фінансів;

— автоматична класифікація фінансових операцій.

До переваг системи можна віднести інтуїтивний інтерфейс, доступність додатку для різних платформ, підтримку кількох користувачів та автоматичну класифікацію фінансових операцій.

Водночас аналітичний модуль системи є спрощеним, графічне представлення фінансової статистики представлене у базовому вигляді та не забезпечує достатнього рівня візуалізації фінансових даних. Також значним недоліком є відсутність формування персональних фінансових рекомендацій для користувача.

1.2.4 Огляд додатка «Monefy»

«Monefy» — це додаток, який призначений для ведення особистого бюджету, який дозволяє відстежувати витрати, контролювати фінансовий стан користувача, планувати заощадження та досягати фінансових цілей [7]. Доступний тільки для мобільних пристроїв як для систем iOS, так і для Android.

Функціональні можливості програми передбачають:

— ведення обліку витрат і доходів за категоріями. Користувач може проводити облік за допомогою стандартних категорій, а також може вручну створювати власні категорії;

— система має вбудований калькулятор для швидкого введення сум;

— графічне представлення через кругові та стовпчикові діаграми;

— формування базової фінансової статистики;

— підтримка декількох рахунків.

Однією з перших переваг є простий та інтуїтивно зрозумілий інтерфейс користувача. Додаток дозволяє швидко створювати транзакції та забезпечує зручне відображення витрат за категоріями. Також до переваг можна віднести вбудований калькулятор.

Основними недоліками є обмежена аналітика порівняно з конкурентами, відсутні функції фінансового планування та встановлення цілей. Попри наявну графічну аналітику, система не підтримує розширений модуль статистичних показників фінансової активності користувача.

1.3 Постановка задачі розробки вебдодатка

В ході аналізу існуючих систем для обліку та аналітики персональних фінансів користувача, було виявлено що існуючі системи надають користувачу можливості контролювати фінансові операції, бачити по ним базову статистику у вигляді діаграм та графіків і більшість додатків підтримують планування бюджету. Проте більша частина систем має обмежені можливості аналітики та не представляє детальної фінансової інформації для користувача. Також значним недоліком є відсутність формування персональних фінансових рекомендацій для користувача, щоб він розумів як краще керувати своїми коштами. Крім цього, системи не надають оцінки фінансової активності та узагальнення фінансового стану за певний період, що було б дуже корисно для користувача.

Отже, необхідно створити вебдодаток, який дозволить користувачу вести облік фінансів, переглядати історію витрат, бачити розширену фінансову статистику, фінансовий рейтинг місяця з можливістю порівняння з попередніми місяцями та отримувати персональні автоматичні фінансові рекомендації та прогноз витрат до кінця місяця. Також система має надавати користувачу можливість планування бюджету та перегляду різноманітної графічної аналітики фінансових показників.

1.4 Функціональні вимоги до системи

Відносно поставленого завдання було розроблено перелік основних функціональних вимог до вебдодатка. Дані вимоги охоплюють можливості системи та дають змогу забезпечити зручну взаємодію користувача з системою.

Основні функціональні вимоги:

— Реєстрація та авторизація користувача;

— Додавання, перегляд, редагування, видалення банківських карток;

— Додавання, перегляд, редагування, видалення категорій операцій;

— Додавання, перегляд, редагування, видалення фінансових операцій;

— Фільтрація операцій за періодом, типом, категорією та карткою;

— Фільтрація, пошук та сортування категорій;

— Планування та контроль виконання бюджету;

— Відображення фінансової аналітики у вигляді графіків і діаграм;

— Відображення статистичних показників;

— Автоматичне формування фінансових рекомендацій;

— Автоматичне формування аналітичних висновків для фінансового стану користувача;

— Порівняння фінансових показників за різні періоди;

— Оцінювання фінансового стану користувача за місяцями;

Таким чином, функціональні вимоги охоплюють засоби для ведення фінансового обліку, планування бюджету та аналізу фінансових показників, які є необхідними для ефективного контролю особистих фінансів.

1.5 Нефункціональні вимоги до системи

Крім функціональних вимог, було складено список основних нефункціональних вимог до вебдодатка, щоб визначити основні характеристики та ефективність роботи системи.

Основні нефункціональні вимоги:

— Вебінтерфейс має працювати у сучасних браузерах;

— Інтерфейс системи має бути простим і зручним для користувача;

— Інтерфейс системи має бути адаптивним на різних типах пристроїв;

— Система має забезпечити користувача коректною роботою з даними;

— Дані мають зберігатися в реляційній базі даних;

— Система має перевіряти введені дані користувача на коректність;

— Забезпечення доступу до персональних даних тільки для авторизованого користувача;

— Забезпечення захищеного збереження паролів;

— Система має запобігати появі некоректних або пошкоджених даних.

Таким чином, наведені нефункціональні вимоги визначають основні характеристики якості вебдодатка, зокрема зручність використання, надійність, безпеку та коректність обробки даних.

2 ПРОЄКТУВАННЯ СИСТЕМИ ТА МЕТОДИ РЕАЛІЗАЦІЇ ВЕБДОДАТКА

Під час розробки вебдодатка проєктування системи та вибору методів реалізації є важливим етапом створення програмного забезпечення. У цьому розділі детально розглянуто архітектуру програмного продукту, структуру бази даних, моделі даних та зв’язки між ними, принципи побудови REST API, особливості проєктування користувацького інтерфейсу та модуля аналітики.

2.1 Загальна архітектура системи

«Архітектура вебзастосунків — це структурна основа, яка відповідає за їх проєктування і дизайн. Вона охоплює набір принципів і найкращих практик, що визначають, як різні частини застосунку працюють разом, щоб забезпечити виконання його функцій» [8]. Для розробки вебзастосунку було вирішено дотримуватися клієнт-серверної архітектури.

Клієнт-серверна архітектура є мережевою моделлю, у якій клієнти та сервери взаємодіють між собою з метою виконання певних завдань та обміну даними [9]. Використання такого підходу забезпечує розподіл функцій між окремими компонентами системи, спрощує її підтримку і дозволяє незалежно розвивати клієнтську та серверну частини додатка.

Архітектура системи включає в себе три основні компоненти такі, як клієнтська частина, серверна частина та база даних. Клієнтська частина це реалізація взаємодії користувача із системою через графічний інтерфейс. Серверна частина включає в себе обробку запитів користувача, виконання бізнес логіки та обмін даними з базою даних. База даних потрібна для збереження інформації про

користувачів, банківські карти, категорії доходів і витрат, фінансових операцій та встановлених користувачем бюджетних лімітів.

До переваг даної архітектури можна віднести розподіл навантаження між компонентами системи, централізоване управління даними, просте обслуговування програмного забезпечення та можливість незалежно оновлювати окремі модулі. Така архітектура забезпечує масштабованість, що дозволяє без ускладнень збільшувати кількість клієнтів і серверів без впливу на її ефективність. Також дана архітектура дозволяє застосовувати захист на серверній стороні, що сприяє підвищенню рівня безпеки даних.

Однак даний підхід має деякі недоліки. Функціонування клієнтської частини безпосередньо залежить від доступності сервера та стабільності мережевого підключення. Зі збільшення кількості користувачів можливо потрібно буде необхідна додаткова оптимізація серверної частини для того, щоб забезпечити належну продуктивність та стабільність роботи системи загалом. Загальну архітектуру вебдодатка наведено на рисунку 2.1.

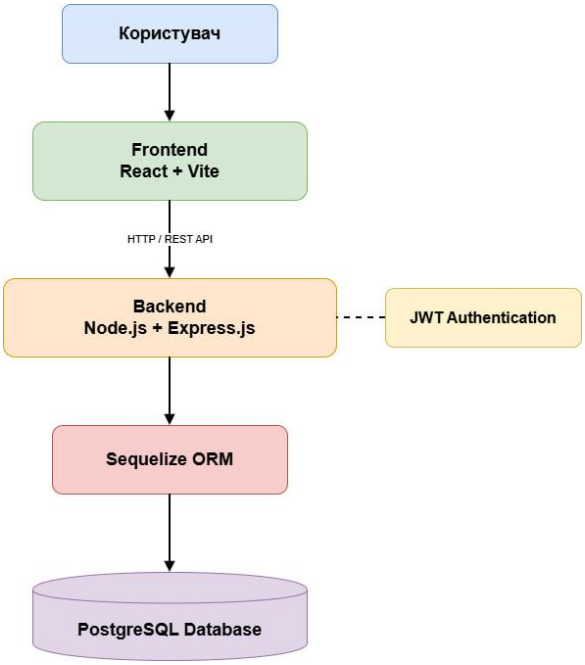

Рисунок 2.1 — Загальна архітектура вебдодатка

Таким чином, використання клієнт-серверної архітектури дозволяє забезпечити ефективну взаємодію між системними компонентами, забезпечує надійність у зберіганні й обробці даних, а також створює перспективи для подальшого розширення і вдосконалення програмного забезпечення.

2.2 Проєктування структури бази даних

База даних є ключовим компонентом вебдодатка, який забезпечує зберігання й обробку інформації, необхідної для функціонування системи. Правильне планування її структури безпосередньо впливає на продуктивність програмного забезпечення, цілісність даних і швидкість виконання операцій.

Для побудови системи було обрано систему керування базами даних PostgreSQL, яка підтримує реляційну модель даних, забезпечує надійне зберігання інформації та надає великі можливості для роботи із запитами різної складності.

Структура бази даних була спроєктована відповідно до функціональних вимог вебдодатка та охоплює сутності користувачів, банківських карток, категорій операцій, фінансових операцій та бюджетних планів. Сутності відповідають об’єктам предметної області й забезпечують організоване зберігання необхідної інформації для реалізації функціоналу системи.

До складу бази даних входять сутності User, яка містить інформацію про користувачів системи, Card, призначена для зберігання даних про банківські картки користувачів, Category, що використовується для зберігання категорій доходів та витрат, Operation, яка містить інформацію про фінансові операції користувачів, а також BudgetPlan, призначена для зберігання даних про бюджетні обмеження та фінансове планування. Структуру бази даних та взаємозв’язки між її основними сутностями наведено на рисунку 2.2.

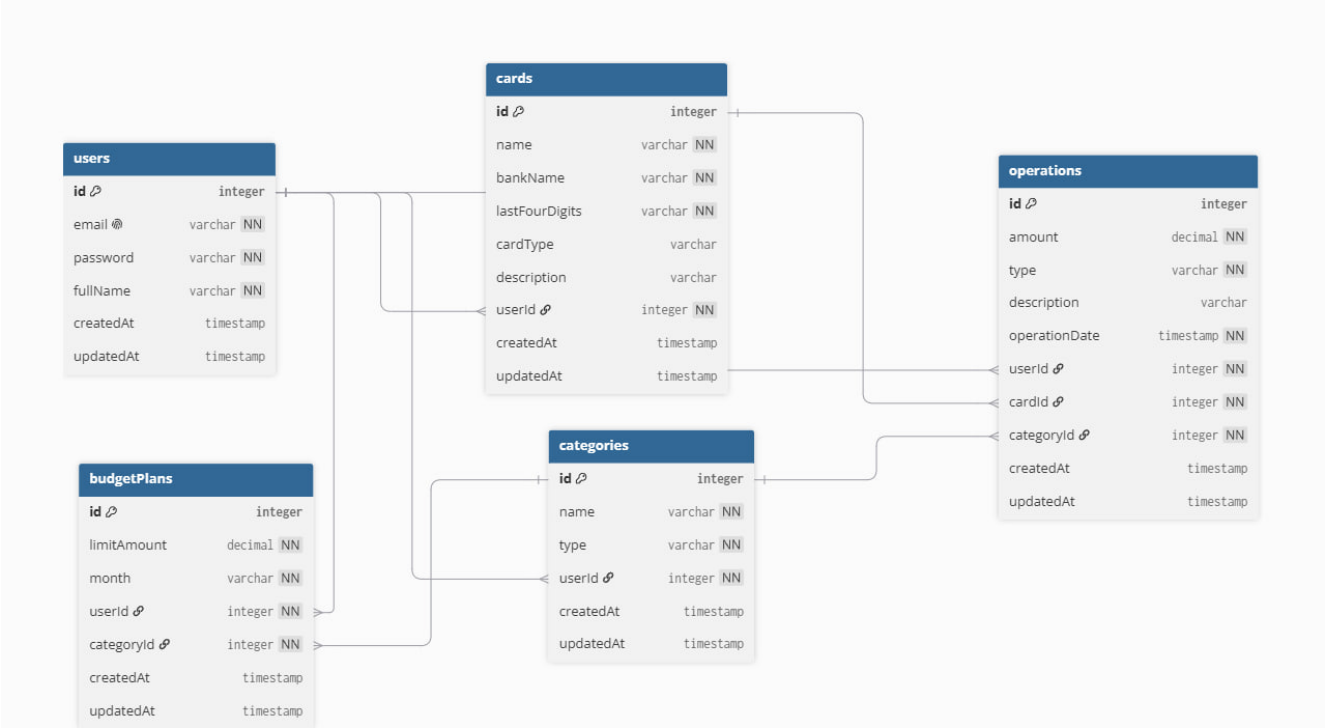

Рисунок 2.2 — Структура бази даних вебдодатка

Таким чином, розроблена структура бази даних забезпечує повноцінне зберігання важливих даних для роботи вебдодатка та служить фундаментом для впровадження інструментів обліку й аналізу фінансових операцій.

2.3 Проєктування моделей даних та зв’язків між ними

Після завершення етапу проєктування структури бази даних були створені моделі даних, які забезпечують ефективне зберігання та обробку інформації в системі. Для кожної моделі визначено набір атрибутів, їх типи даних та взаємозв’язки з іншими моделями.

Модель User відповідає за зберігання інформації про користувачів системи та реалізує функціонал реєстрації, автентифікації й авторизації. Основні атрибути цієї

моделі включають ідентифікатор користувача, ім’я, електронну пошту та хешований пароль.

Модель Card зберігає дані про банківські картки користувачів. Вона містить такі ключові параметри, як ідентифікатор картки, назва картки, назва банку, останні чотири цифри, тип картки та опис.

Модель Category використовується для організації інформації про категорії доходів та витрат. Серед її основних атрибутів — ідентифікатор категорії, назва та тип (доходи чи витрати).

Модель Operation є центральним елементом системи і зберігає дані про фінансові операції користувачів. До основних полів відносяться ідентифікатор операції, сума, тип операції, дата здійснення та опис.

Модель BudgetPlan застосовується для обліку бюджетних обмежень користувачів і управління витратами. Вона включає такі атрибути, як ідентифікатор запису, встановлений бюджетний ліміт і період дії.

У системі застосовано відношення типу «один до багатьох» між моделями даних. Наприклад, одному користувачеві можуть належати кілька банківських карток, категорій, фінансових операцій та бюджетних планів. Кожна фінансова операція пов’язана з відповідною банківською карткою та категорією операцій. Бюджетний план також асоційовано з конкретним користувачем і відповідною категорією витрат для відстеження виконання бюджетних обмежень.

Завдяки розробленим моделям даних та їх взаємозв’язкам система може зберігати всю необхідну інформацію для роботи вебдодатка, підтримувати цілісність даних та слугувати стабільною платформою для реалізації функціоналу обліку й аналізу фінансових операцій.

2.4 Проєктування REST API

REST API забезпечує взаємодію між клієнтом і сервером за допомогою HTTP-протоколу. Обмін даними між компонентами системи зазвичай здійснюється у форматі JSON із використанням стандартних вебпротоколів [10]. Щоб реалізувати взаємодію клієнта з сервером вебзастосунку було розроблено REST API. Даний підхід сприяє незалежності обох компонентів, що спрощує їхню підтримку та подальше масштабування.

Під час проєктування REST API було застосовано основні принципи архітектурного стилю REST. Для отримання даних використовувалися GET-запити, для створення нових записів використовувалися POST-запити, оновлення відбувалося через PUT-запити, а видалення — через DELETE-запити. Такий підхід спрощує взаємодію клієнта з сервером та полегшує реалізацію функціоналу системи.

Використання архітектурного стилю REST забезпечує розділення клієнтської та серверної частин системи, що сприяє підвищенню масштабованості, надійності та незалежності компонентів вебзастосунку [11].

Реалізований REST API організовує передачу даними між клієнтом і сервером. Клієнт надсилає HTTP-запити, сервер їх отримує і опрацьовує, взаємодіє з базою даних та повертає результати у форматі JSON. Це дозволяє швидко та ефективно забезпечити обмін інформацією та працювати компонентам системи незалежно один від одного.

Функціональна структура API розподілена на кілька основних модулів відповідно до завдань вебзастосунку. Серед них: модуль реєстрації та авторизації користувачів, модуль управління банківськими картками, модуль категорій фінансових операцій, модуль обробки транзакцій, бюджетне планування та аналітика. Даний модульний підхід гарантує зрозумілу архітектуру програмного інтерфейсу, що полегшує його оновлення та підтримку в майбутньому.

Щоб забезпечити безпеку даних користувачів було реалізовано механізми автентифікації та авторизації через JWT-токени.

«Вебтокени JSON (JWT) — це надійний спосіб захисту API шляхом вбудовування інформації користувача безпосередньо в токени» [12].

Після того, як користувач успішно входить до системи, сервер генерує для нього токен доступу, який потрібний для виконання запитів до захищених ресурсів. За допомогою JWT-токенів можна ефективно контролювати доступ та забезпечити високий рівень захисту інформації.

Використання JWT-токенів дозволяє реалізувати безстанну автентифікацію, за якої кожний запит містить усю необхідну інформацію для перевірки користувача. Такий підхід спрощує взаємодію між клієнтом і сервером та дозволяє ефективно масштабувати вебзастосунок [13].

Таким чином, спроєктований REST API гарантує надійну взаємодію між клієнтом і сервером, дозволяючи реалізувати функціональні можливості та забезпечити безпечний обмін даними між її компонентами.

2.5 Проєктування користувацького інтерфейсу

Користувацький інтерфейс є ключовим елементом вебдодатка, оскільки від його якості залежить ефективність взаємодії користувача із функціоналом системи. У процесі проєктування було зосереджено увагу на такі два аспекти, як інтуїтивна навігація та швидкий доступ до потрібної інформації

Дизайн інтерфейсу відповідає сучасним стандартам розробки вебзастосунків. Для комфорту користувача було створено логічну структуру сторінок, передбачено зручне розташування елементів керування, а також забезпечено єдиний стиль оформлення. Окремий акцент зроблено на зрозумілому поданні фінансової інформації і мінімізації зусиль, необхідних для виконання ключових операцій.

Інтерфейс побудований так, щоб охопити основні потреби користувача. Він включає сторінки для реєстрації та авторизації, головну сторінку, модулі управління банківськими картками та категоріями операцій, сторінку фінансових транзакцій, інструменти для бюджетного планування та візуалізації аналітики. Кожен розділ проєктувався для виконання чітко визначеного набору завдань та надання швидкого доступу до функціональних можливостей системи.

Для легкого переміщення між модулями використовується система маршрутизації, яка дозволяє швидкий перехід між сторінками без зайвих ускладнень.

Таким чином, спроєктований інтерфейс сприяє комфортній взаємодії користувача із системою та покращує процес доступу до функціональних можливостей вебдодатка, створюючи сприятливі умови для ефективного управління фінансовою інформацією.

2.6 Проєктування модуля аналітики

Аналітичний модуль є одним із ключових компонентів вебдодатка, оскільки його функціонал спрямований на обробку фінансових даних користувача й формування статистичної інформації. Основним завданням модуля є надання користувачу інструментів для аналізу доходів та витрат, моніторингу фінансової активності й прийняття раціональних фінансових рішень.

Під час розробки модуля були визначені ключові показники, що відображають фінансовий стан користувача за певний часовий період. До них належать: загальна сума доходів, витрат, поточний баланс, основні категорії доходів і витрат, найбільші фінансові операції, середньостатистичні коефіцієнти витрат і доходів, кількість проведених транзакцій, а також інші дані, що необхідні для комплексного фінансового аналізу.

Для забезпечення кращого розуміння і наочності даних у модулі аналітики передбачено графічне відображення результатів. Інтеграція діаграм і графіків допомагає користувачу швидко оцінити структуру доходів і витрат, виявити ключові категорії фінансових операцій і простежити динаміку змін показників у різні періоди.

Крім базових статистичних даних, модуль пропонує додаткову інформацію, зокрема: визначення провідних категорій доходів і витрат, фіксацію днів із максимальними обсягами доходів або витрат, виявлення найменш активно використовуваних банківських карток тощо. Такі параметри створюють цілісну картину фінансової діяльності користувача.

З метою підвищення фінансової дисципліни було додано механізм оцінювання фінансового стану. Ця функція базується на аналізі співвідношення надходжень та витрат, дотримання бюджету й інших ключових показників. Результатом такого аналізу стає фінальна оцінка, яка відображає поточний рівень фінансової стабільності користувача.

Функціонування модуля передбачає отримання інформації про транзакції користувача з бази даних, їх подальшу обробку та генерацію статистичних показників. На базі отриманих результатів формується узагальнена статистична інформація та її графічна презентація, яка відображається користувачеві у вебінтерфейсі.

В підсумку, розроблений аналітичний модуль дозволяє проводити усебічний аналіз фінансових даних, забезпечує контроль за станом фінансів і сприяє ефективнішому управлінню особистими грошовими ресурсами.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Після завершення етапу проєктування було створено програмне забезпечення вебзастосунку для обліку та аналізу операцій з банківськими картками користувача. На цьому етапі виконано розробку як серверної, так і клієнтської частин системи, налаштовано їхню взаємодію, а також реалізовано механізми для обробки, зберігання та аналізу фінансової інформації.

У цьому розділі розглянуто обрані технології та засоби розробки, особливості реалізації серверної і клієнтської частин вебдодатка, механізми взаємодії з базою даних та REST API, а також засоби забезпечення безпеки й перевірки даних.

3.1 Вибір технологій та засобів розробки

У процесі створення вебдодатка для обліку та аналізу операцій з банківськими картками був використаний сучасний технологічний стек, що забезпечив високу продуктивність, зручність розробки і можливість майбутнього розширення функціональних можливостей. Вибір технологій базувався на функціональних вимогах системи, потребі інтеграції з базами даних, реалізації клієнт-серверної архітектури та створенні інтуїтивно зрозумілого інтерфейсу для користувачів.

Для серверної частини було обрано платформу Node.js — потужне середовище виконання для запуску JavaScript-коду поза веббраузером, яке дозволяє створювати масштабовані та високопродуктивні застосунки [14].

Щоб реалізувати серверну логіку та створити REST API було використано фреймворк Express.js. Express.js надає перелік функцій для легкого створення вебзастосунків та мобільних застосунків, а також спрощує розробку серверних застосунків, надаючи простий у використанні API для маршрутизації [15].

Зберігання та управління даними реалізовано за допомогою PostgreSQL — реляційна база даних з відкритим кодом, відома своєю надійністю, гнучкістю та підтримкою відкритих технічних стандартів [16]. Щоб спростити роботу з базою даних, було використано ORM-бібліотеку Sequelize, яка надала змогу представити таблиці в базі даних у вигляді об’єктів.

Механізми автентифікації та авторизації користувачів впроваджено за допомогою JWT-токенів, які забезпечують контроль доступу та підвищують безпеку системи.

Для шифрування паролів використовувалася бібліотека bcrypt — криптографічний алгоритм хешування паролів, створений щоб забезпечити безпеку та захист від атак. Основними особливостями є можливість створювати складність, що визначає кількість разів виконання хешування, автоматична генерація випадкової солі для унікальності хеша та забезпечення додаткового захисту від атак грубою силою [17].

Клієнтська частина вебдодатка була реалізована за допомогою бібліотеки React.js — фреймворк і бібліотека JavaScript, що використовується для швидкого створення інтерактивних користувацьких інтерфейсів і вебдодатків з використанням меншої кількості коду, ніж під час використання звичайного JavaScript [18]. За допомогою даного фреймворку можна створювати інтерактивні односторінкові застосунки з оновленням інтерфейсу в режимі реального часу без необхідності повного перезавантаження сторінки. Для організації маршрутизації було використано бібліотеку React Router.

Передача даних між клієнтом і сервером здійснювалася за допомогою бібліотеки Axios. Ця бібліотека потрібна для роботи з HTTP-запитами. Вона полегшує надсилання асинхронних запитів до REST-ендпоінтів, обробку відповідей та роботу з помилками [19].

Для візуалізації статистичних даних та аналітики був застосований Recharts — бібліотека для React, призначена для побудови графіків і діаграм на основі компонентного підходу. Використання Recharts дозволяє створювати візуалізацію даних та інтегрувати її в інтерфейс вебзастосунку [20].

Таким чином, підібрані технології та засоби забезпечили реалізацію всіх необхідних функцій вебдодатка, залишаючись гнучкими для розширення та відповідаючи сучасним вимогам до розробки програмних продуктів.

3.2 Реалізація серверної частини системи

Серверна частина веб-застосунку виконує ключову роль у реалізації бізнес-логіки системи, забезпечуючи обробку запитів користувачів. Вона відповідає за інтеграцію з базою даних, виконання операцій над інформацією, підготовку відповідей для клієнтської сторони, а також за гарантування безпечного та контрольованого доступу до функціоналу застосунку.

Розробка серверної частини реалізована на платформі Node.js із використанням фреймворку Express.js. Для роботи з базою даних PostgreSQL було впроваджено ORM-бібліотеку Sequelize, яка спрощує взаємодію з даними завдяки використанню об'єктних моделей і забезпечує зручне виконання запитів до бази.

Серед ключових функцій серверної частини були впроваджені модулі для реєстрації та авторизації користувачів, управління банківськими картками, адміністрування категорій витрат та доходів, ведення фінансових операцій, планування бюджету та виконання аналітичних розрахунків.

3.2.1 Реалізація авторизації та автентифікації користувачів

Для реалізації доступу до функціональних можливостей веб-додатка було впроваджено систему реєстрації, автентифікації та авторизації користувачів. Цей функціонал розроблено на серверному боці із застосуванням JWT-токенів та бібліотеки bcrypt.

На етапі реєстрації користувач надає ім’я, адресу електронної пошти та пароль. Система перевіряє, чи існує вже користувач із зазначеною електронною поштою. У разі відсутності збігів пароль шифрується за допомогою алгоритму хешування bcrypt і зберігається у базі даних у зашифрованому форматі.

У процесі авторизації користувача здійснюється автентифікація введених облікових даних. Після успішного підтвердження правильності вказаних електронної пошти та пароля сервер генерує JWT-токен, що слугує для забезпечення доступу до захищених ресурсів системи. Використання токенів дозволяє підвищити загальний рівень безпеки шляхом встановлення захищеного каналу комунікації між клієнтською та серверною частинами веб-додатка.

Таким чином впроваджений механізм автентифікації та авторизації забезпечує контроль доступу до персональних даних користувачів та підвищує загальний рівень безпеки системи.

3.2.2 Реалізація роботи з банківськими картками

Одним із ключових функціональних компонентів вебдодатка є модуль управління банківськими картками користувачів. Для його реалізації на серверній стороні розроблено набір REST API-запитів, які забезпечують підтримку основних операцій, таких як створення, перегляд, редагування та видалення карток.

У процесі створення банківської картки користувачеві надається можливість зазначити її назву, найменування банку, останні чотири цифри номера картки, тип картки та додати за потреби додатковий опис. Після отримання відповідного запиту сервер виконує перевірку валідності введених даних та зберігає інформацію у базі даних із прив’язкою до конкретного користувача.

Механізм отримання інформації про картки передбачає надання авторизованому користувачеві доступу до всіх банківських карток, прив'язаних до його облікового запису. Зібрані дані передаються клієнтській стороні для

відображення списку карток, що стає основою для подальшої інтеграції з фінансовими операціями.

Функціональність редагування забезпечує можливість модифікації окремих параметрів банківської картки. Завдяки цьому користувач може оновлювати лише необхідні поля без потреби повторного введення всієї інформації. На серверному боці реалізовано механізм перевірки існування картки та внесення змін лише до переданих даних, що підвищує ефективність і точність обробки запитів.

Модуль видалення передбачає додаткові перевірки на цілісність даних, особливо у випадках, коли до банківської картки прив'язані фінансові операції. У таких ситуаціях система блокує видалення запису та повідомляє користувача про зв’язки з іншими об’єктами.

Отже, створений модуль управління банківськими картками забезпечує комплексний набір функцій для роботи з персональними картками користувачів, гарантує високий рівень цілісності даних та стабільність системи під час виконання операцій створення, редагування і видалення записів.

3.2.3 Реалізація роботи з категоріями операцій

Для вдосконалення організації фінансових операцій у системі було розроблено спеціалізований модуль управління категоріями. Основна функція цього модуля полягає у класифікації доходів і витрат за відповідними категоріями, що значно оптимізує процес ведення обліку фінансових транзакцій та сприяє проведенню детального аналізу статистичних даних.

На серверному рівні впроваджено набір REST API-запитів, які надають можливість здійснювати операції створення, читання, редагування та видалення категорій. При створенні нової категорії користувач повинен зазначити її назву та тип.

Механізм отримання даних із бази реалізовано таким чином, що він враховує приналежність записів до конкретного користувача. Це дозволяє забезпечити ізольоване зберігання інформації та запобігає доступ до даних інших користувачів, що сприяє підвищенню рівня безпеки.

У разі редагування категорії система надає можливість змінювати її ключові атрибути. Для видалення категорій передбачено додаткові перевірки, які гарантують коректність функціонування системи й сприяють збереженню структурної цілісності даних.

Таким чином, розроблений модуль управління категоріями виступає ефективним засобом упорядкування фінансових операцій та створює фундамент для формування аналітичної інформації в рамках аналітичного модуля системи.

3.2.4 Реалізація роботи з фінансовими операціями

Фінансовий модуль виступає одним із ключових компонентів вебдодатка, який відповідає за облік доходів і витрат користувача. Його серверна частина включає інструменти для створення, перегляду, редагування та видалення фінансових операцій, забезпечуючи повний цикл роботи з відповідними даними.

Під час створення нової операції користувач вносить інформацію про тип транзакції, суму, категорію, прив'язану банківську картку, дату здійснення операції та додаткові деталі. Сервер перевіряє правильність зазначених даних і після успішної валідації вносить їх до бази даних.

Функція перегляду реалізує отримання записів із бази даних відповідно до запитів користувача. На основі отриманих даних клієнтська частина формує список операцій та надає статистичну інформацію у вигляді зручних для аналізу графіків або таблиць.

Для редагування або видалення існуючих записів система передбачає ретельну перевірку прав доступу користувача.

У підсумку, модуль фінансових операцій забезпечує комплексне управління доходами і витратами користувачів, виступаючи основним джерелом даних для функціонування інших частин вебдодатка.

3.2.5 Реалізація фільтрації операцій

Для оптимізації роботи з великими обсягами фінансових записів у системі розроблено ефективний механізм фільтрації транзакцій, покликаний забезпечити зручність пошуку необхідних даних та спрощення аналізу фінансової діяльності на основі різноманітних критеріїв.

На рівні серверної частини впроваджено функціональність, яка підтримує фільтрацію за такими параметрами, як тип операції, категорія, банківська карта та часовий інтервал. Після надходження відповідних параметрів система формує специфічний запит до бази даних, який повертає виключно ті записи, що відповідають заданим умовам.

Застосування серверного підходу до фільтрації даних забезпечує зменшення обсягу інформації, яка передається клієнту, що, у свою чергу, сприяє підвищенню ефективності функціонування системи під час обробки великих масивів фінансових операцій.

Отже, розроблений механізм фільтрації суттєво покращує швидкість доступу до необхідної інформації та значно полегшує процес аналізу даних користувача, впроваджуючи більш систематизований і продуктивний підхід до управління фінансовою інформацією.

3.2.6 Реалізація модуля аналітики

Для забезпечення можливості проведення аналізу фінансової активності користувача в серверній частині системи було впроваджено спеціалізований модуль аналітики. Основна функція цього модуля полягає в обробці даних про фінансові операції та формуванні статистичних індикаторів, необхідних для об’єктивної оцінки фінансового стану користувача.

У процесі своєї роботи модуль аналітики отримує інформацію про фінансові трансакції за визначений часовий інтервал і здійснює їх подальшу обробку. На основі цих даних проводиться обчислення ключових показників, зокрема загального обсягу доходів, сукупної суми витрат, поточного балансу, середніх значень, а також визначаються найбільш вагомі категорії доходів і витрат, кількість проведених операцій та інші важливі параметри.

Крім розрахунку базових аналітичних показників, серверна частина забезпечує створення даних для сформування різного роду графіків і діаграм. Ці візуалізації призначені для використання на клієнтській стороні системи, що покращує наочність подання фінансової інформації. Додатково, модуль включає алгоритм оцінки фінансового статусу користувача з урахуванням таких аспектів, як рівень доходів, витрати та відповідність встановленим бюджетним обмеженням.

Результати роботи зазначеного модуля передаються на клієнтську частину за допомогою REST API у форматі JSON. Це сприяє зручному та ефективному використанню отриманих даних у подальшому для відображення статистичних звітів і графічної аналітики.

Таким чином, упроваджений модуль аналітики виконує автоматизовану обробку фінансової інформації й надає користувачам зручні інструменти для комплексного аналізу та управління особистими фінансами.

3.3 Реалізація клієнтської частини

Клієнтська частина вебзастосунку відповідає за організацію взаємодії користувача з функціональністю системи, забезпечуючи відображення отриманої від серверної частини інформації. Для її створення використовується бібліотека React.js, що дає змогу реалізувати сучасний односторінковий застосунок із динамічним оновленням інтерфейсу без необхідності перезавантаження сторінки.

Особливий акцент під час розробки клієнтської частини було зроблено на зручності користування, високій швидкодії роботи інтерфейсу та комфортній взаємодії з фінансовими даними. У рамках розробки було створено набір сторінок, компонентів і механізмів інтеграції із сервером через REST API для повноцінної роботи застосунку.

3.3.1 Реалізація структури React.js-застосунку

Клієнтська частина вебзастосунку була розроблена з використанням бібліотеки React.js, яка ґрунтується на компонентному підході до створення інтерфейсу. Такий підхід дозволяє модульно розподілити функціональність системи, що значно полегшує підтримку та розширення програмного коду.

Архітектура застосунку побудована за принципом логічного розподілу функціональних елементів між окремими каталогами. Взаємодія із серверною частиною реалізована через каталог api, де налаштовано HTTP-клієнт Axios. Серед функціональних можливостей каталогу — механізм автоматичного додавання JWT-токена до запитів, що підвищує безпеку й спрощує аутентифікацію.

Повторно використовувані компоненти інтерфейсу винесено в каталог components. До таких належать ключові елементи, як Header і Sidebar, які відповідають за навігацію між сторінками застосунку та демонстрацію інформації про поточного користувача.

Для управління загальною структурою інтерфейсу застосовується компонент MainLayout. Він об'єднує верхню панель, бокове меню та основну область для відображення контенту сторінок.

Бізнес-логіка користувацького інтерфейсу реалізується через окремі сторінкові компоненти, кожен із яких відповідає конкретному функціональному модулю системи. Бібліотека React Router забезпечує зручну навігацію між сторінками, дозволяючи користувачам перемикатися між розділами без необхідності перезавантаження сайту.

Таким чином, реалізована структура на базі React.js створює чітку та упорядковану архітектуру програмного коду, значно полегшуючи обслуговування системи, сприяючи її стабільності та надаючи можливість для ефективного масштабування функціоналу в майбутньому.

3.3.2 Реалізація сторінок та компонентів

Для забезпечення інтерактивної взаємодії користувачів із функціональними можливостями системи у вебзастосунку було розроблено комплекс сторінок та компонентів інтерфейсу. Кожна сторінка відповідає за реалізацію певного функціонального модуля, що дозволяє отримувати доступ до специфічних даних і виконувати відповідні операції.

З метою керування обліковими записами користувачів у системі були створені сторінки реєстрації та авторизації. Вони забезпечують введення облікової інформації, перевірку її коректності та взаємодію із серверною частиною під час виконання процедур створення нового облікового запису й авторизації.

Основні функціональні можливості вебзастосунку зосереджено на сторінках, присвячених управлінню банківськими картками, категоріями фінансових операцій, плануванню бюджету, аналізу витрат і доходів, а також опрацюванню транзакцій. Такі сторінки надають інструменти для створення, перегляду,

модифікації та видалення відповідної інформації, сприяючи ефективній роботі з даними.

З метою оптимізації розробки й уникнення дублювання програмного коду були використані багаторазові компоненти інтерфейсу. До них належать навігаційні панелі, заголовки застосунку, бокові меню та інші допоміжні елементи, які повторно використовуються на різних сторінках системи. Застосування компонентного підходу підвищило зручність супроводу та модернізації програмного забезпечення, гарантуючи його масштабованість.

В результаті реалізовані сторінки та інтерфейсні компоненти забезпечують зручний і продуктивний досвід взаємодії користувачів із вебзастосунком, ефективно підтримуючи виконання всіх заявлених функціональних можливостей системи.

3.3.3 Реалізація взаємодії з сервером

Для забезпечення ефективного обміну даними між клієнтською та серверною частинами вебзастосунку впроваджено взаємодію через REST API. Передача інформації здійснюється за допомогою HTTP-запитів, а отримані дані проходять обробку та відображаються у користувацькому інтерфейсі.

Для виконання запитів до сервера використовується бібліотека Axios, яка забезпечує асинхронний обмін даними. Це дозволяє отримувати відповіді від сервера без необхідності перезавантаження сторінки. Такий підхід сприяє швидкому оновленню інформації та підвищує зручність користування системою.

Після успішної авторизації JWT-токен, отриманий від сервера, зберігається на стороні клієнта. Він автоматично включається до кожного запиту, що потребує автентифікації. Це дозволяє серверу коректно ідентифікувати користувача й надавати доступ лише до його персоналізованих даних.

Реалізований механізм обміну даними інтегрує всі основні модулі вебзастосунку, включаючи функції роботи з банківськими картками, управління категоріями операцій, облік фінансових транзакцій, бюджетне планування та аналітику.

Завдяки цьому забезпечується актуальність даних усередині системи, а також стабільна та коректна робота всіх функціональних можливостей вебзастосунку.

3.3.4 Реалізація графічної аналітики

З метою покращення наочності у представленні фінансової інформації у вебдодатку було створено модуль графічної аналітики. Його головне завдання полягає у візуалізації статистичних даних та спрощенні процесу аналізу фінансової активності користувачів.

Функціонал модуля реалізовано з використанням бібліотеки Recharts, яка дозволяє будувати інтерактивні графіки та діаграми на основі даних, що надходять із серверної частини. Для створення візуалізацій застосовуються статистичні показники, розраховані аналітичним модулем.

Модуль надає можливість переглянути структуру доходів і витрат за категоріями, аналізувати фінансову активність за обраний часовий період, а також отримувати узагальнені статистичні дані. Завдяки графікам та діаграмам користувач може швидко оцінювати власний фінансовий стан та виявляти ключові тенденції у використанні коштів.

Окрім візуальних елементів, сторінка аналітики включає додаткові ключові статистичні показники, що характеризують фінансову активність, дозволяючи отримати більш детальну інформацію про доходи та витрати.

У результаті створений модуль графічної аналітики надає зручний формат відображення фінансових даних та значно полегшує їх аналіз.

3.4 Забезпечення безпеки та перевірки даних

У процесі розробки вебзастосунку значна увага приділялася забезпеченню безпеки даних користувачів, а також перевірці достовірності інформації, яка надходить у систему. Впроваджені механізми слугують для захисту персональних даних користувачів, попередження несанкціонованого доступу до ресурсів системи та гарантування цілісності інформації, що зберігається у базах даних.

Для захисту облікових записів в системі застосовується автентифікація та авторизація на основі JWT-токенів. Після успішної авторизації користувач отримує токен доступу, який надалі використовується під час запитів до захищених ресурсів вебзастосунку. На етапі обробки запиту серверна частина перевіряє дійсність токена та відповідні права доступу користувача, що дозволяє уникнути несанкціонованих операцій.

З метою захисту конфіденційних даних, паролі користувачів не зберігаються у відкритому вигляді. Перед їхнім збереженням у базі даних застосовується процедура хешування із використанням бібліотеки bcrypt, що значно підвищує рівень інформаційної безпеки облікових записів.

Розробка також включала реалізацію механізмів перевірки вхідних даних для забезпечення коректної роботи системи. Перед виконанням таких операцій як створення, редагування або видалення записів сервер проводить перевірку заповнення обов’язкових полів, типів переданих даних та їх допустимих значень. У разі виявлення помилок система забезпечує користувача детальним повідомленням про невідповідність даних встановленим вимогам.

Окремо було впроваджено функцію перевірки належності даних поточному користувачеві. Наприклад, під час роботи із банківськими картками, категоріями витрат, фінансовими операціями та бюджетними планами система визначає права доступу до відповідних записів.

Для підтримання цілісності даних застосовані механізми перевірки залежностей між сутностями. Наприклад, під час спроби видалити банківську

картку система перевіряє наявність пов’язаних з нею фінансових операцій. У разі виявлення таких зв’язків виконання дії блокується, що попереджає можливість втрати важливих даних.

Таким чином, впроваджені заходи безпеки та механізми перевірки суттєво підвищують захищеність користувацької інформації, підтримують консистентність даних і сприяють стабільній роботі вебзастосунку.

4 ДЕМОНСТРАЦІЯ РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

У цьому розділі представлено системні вимоги для запуску програмного забезпечення, описано процес авторизації та реєстрації користувача, роботу з банківськими картками, категоріями та фінансовими операціями, а також можливості перегляду аналітики, статистики та планування бюджету.

4.1 Системні вимоги для запуску вебдодатка

Для забезпечення запуску та стабільної роботи вебдодатка необхідно підготувати відповідне програмне середовище для функціонування як серверної, так і клієнтської частин системи. Серверна складова потребує попереднього встановлення платформи Node.js, системи керування базами даних PostgreSQL та необхідних програмних залежностей проєкту.

Клієнтська частина вебдодатка функціонує безпосередньо у веббраузері, тому на стороні користувача не потрібна установка додаткового програмного забезпечення. Для безперебійної роботи буде достатньо використання сучасного веббраузера з підтримкою JavaScript, HTML5 і CSS3.

Перед стартом роботи слід виконати встановлення залежностей проєкту за допомогою менеджера пакетів npm, а також налаштувати параметри підключення до бази даних PostgreSQL. У результаті запуску серверної частини система перейде до обробки HTTP-запитів, що дозволить користувачам взаємодіяти з вебдодатком через інтерактивний інтерфейс клієнтської частини.

У процесі розробки та тестування вебдодатка були використані платформа Node.js, система керування базами даних PostgreSQL і веббраузер Google Chrome.

Тож для успішного запуску вебдодатка необхідно мати встановлені Node.js, PostgreSQL і сучасний веббраузер, які забезпечуватимуть безперебійну взаємодію між клієнтською та серверною частинами системи.

4.2 Авторизація та реєстрація користувача

Система забезпечує персоналізований доступ до фінансових даних завдяки використанню механізму реєстрації та авторизації користувачів. Кожен користувач має унікальний обліковий запис, який дозволяє працювати виключно з власними банківськими картками, категоріями витрат, фінансовими операціями та статистикою.

Щоб створити новий обліковий запис, необхідно перейти на сторінку реєстрації та ввести ім’я, адресу електронної пошти та пароль. Після перевірки правильності введених даних система автоматично реєструє користувача, зберігаючи його інформацію в базі даних. Для підвищення рівня безпеки перед збереженням паролі проходять процедуру хешування. На рисунку 4.1 наведено вікно реєстрації користувача.

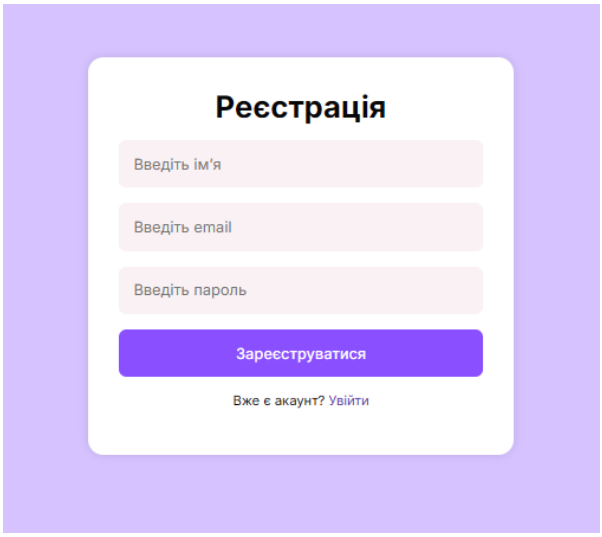

Рисунок 4.1 — Вікно реєстрації користувача

Після успішної реєстрації користувач отримує можливість увійти до системи, використовуючи електронну пошту та пароль. Якщо авторизація проходить успішно, сервер створює JWT-токен, який надає доступ до захищених функцій вебдодатка. На рисунку 4.2 наведено вікно авторизації користувача.

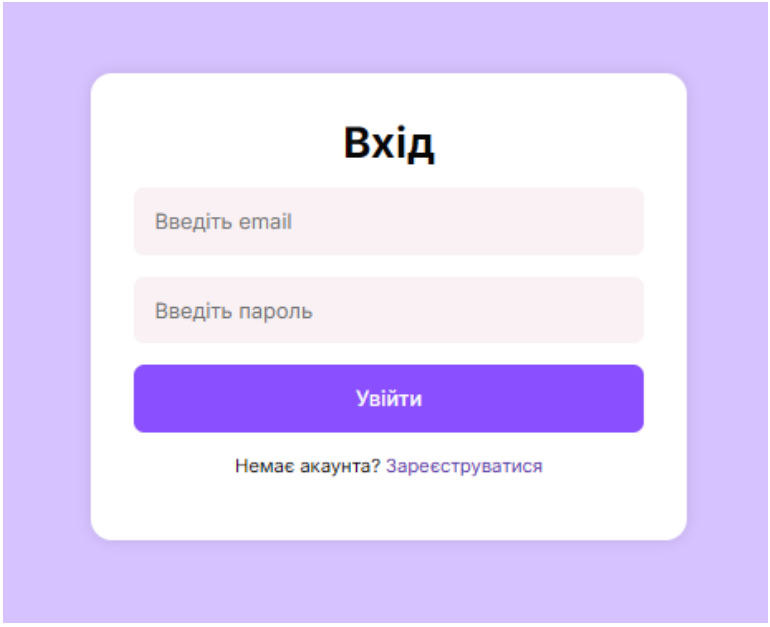

Рисунок 4.2 — Вікно авторизації користувача

Після успішної авторизації користувач автоматично перенаправляється на головну сторінку вебдодатка. На цій сторінці представлено стислий огляд фінансової активності, включаючи дані про доходи, витрати, поточний баланс і статистику за обраний період. Крім того, користувачу надається можливість керувати банківськими картками, категоріями операцій, фінансовими транзакціями, бюджетом і скористатися аналітичними інструментами. На рисунку 4.3 зображено головну сторінку вебдодатка.

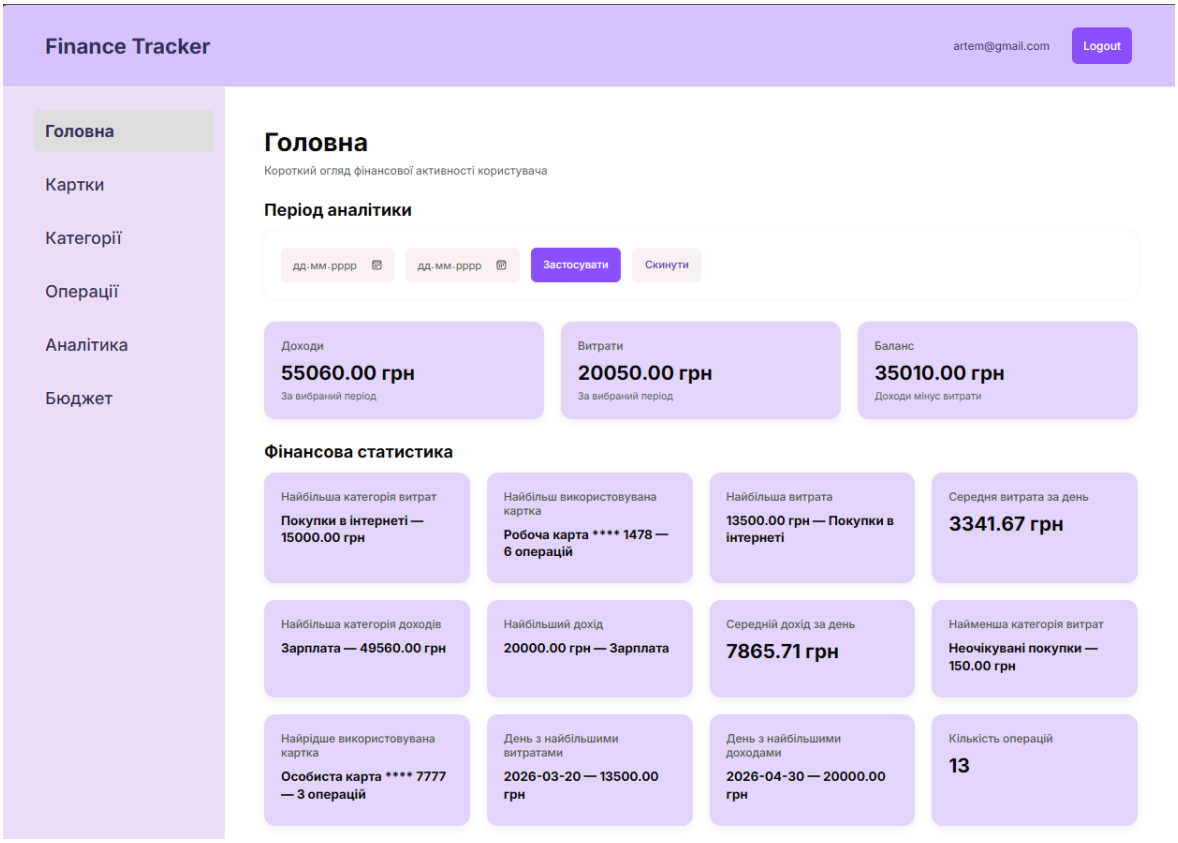

Рисунок 4.3 — Головна сторінка вебдодатка

У разі некоректного введення облікових даних система повідомляє про помилку та пропонує виконати повторну спробу входу.

4.3 Робота з банківськими картками

Функціонал управління банківськими картками створений для збереження даних користувача про картки та використання їх під час виконання фінансових операцій.

У розділі «Картки» користувач має доступ до переліку вже доданих банківських карток. Для кожної з них відображаються назва, банк, останні чотири цифри номера, тип картки та додатковий опис. Сторінка «Картки» показана на рисунку 4.4.

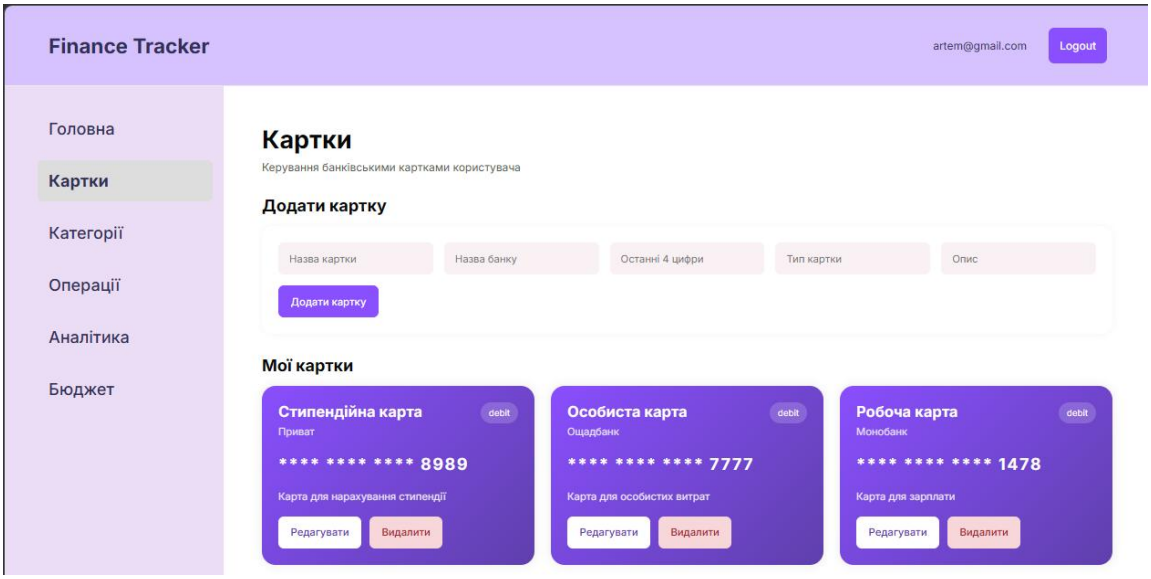

Рисунок 4.4 — Сторінка «Картки»

Щоб додати нову картку, користувач натискає кнопку «Додати картку» й заповнює форму із запитом на введення даних. Під час цієї операції потрібно зазначити назву картки, банк, останні чотири цифри номера, її тип і, за потреби, додаткову інформацію.

Після успішного заповнення і збереження введені дані автоматично додаються до бази та стають доступними у списку карток користувача.

Якщо виникає необхідність змінити інформацію про картку, можна скористатися функцією редагування. Обравши цю дію, користувач отримує можливість внести зміни до всіх доступних параметрів картки та зберегти їх. На рисунку 4.5 наведено форму редагування банківської картки.

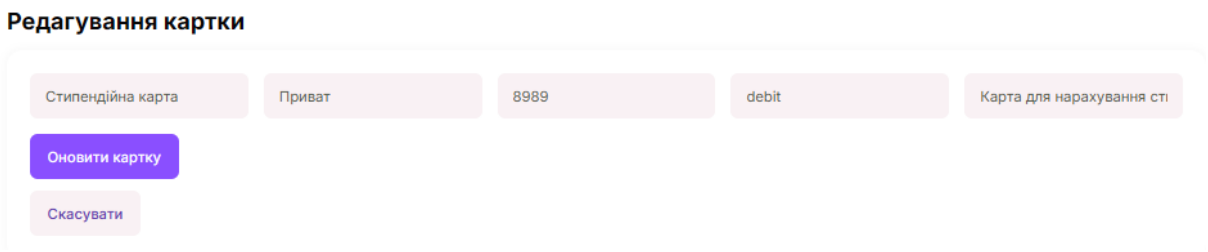

Рисунок 4.5 — Форма редагування банківської картки

Крім того, система підтримує можливість видалення карток. Однак видалення недоступне, якщо до картки прив'язані активні фінансові операції. У такому разі користувач отримає повідомлення про неможливість завершення цієї дії через наявність відповідних залежностей у даних.

4.4 Робота з категоріями операцій

Для організації фінансових операцій у системі передбачено функціонал роботи з категоріями. Ці категорії слугують для класифікації доходів і витрат за різними типами, що дає змогу детально аналізувати фінансову активність користувача.

На сторінці «Категорії» відображається список створених категорій, доступний після переходу до відповідного меню. Для кожної категорії зазначаються її назва та тип, який уточнює, чи належить вона до доходів або витрат. На рисунку 4.6 наведено сторінку роботи з категоріями операцій.

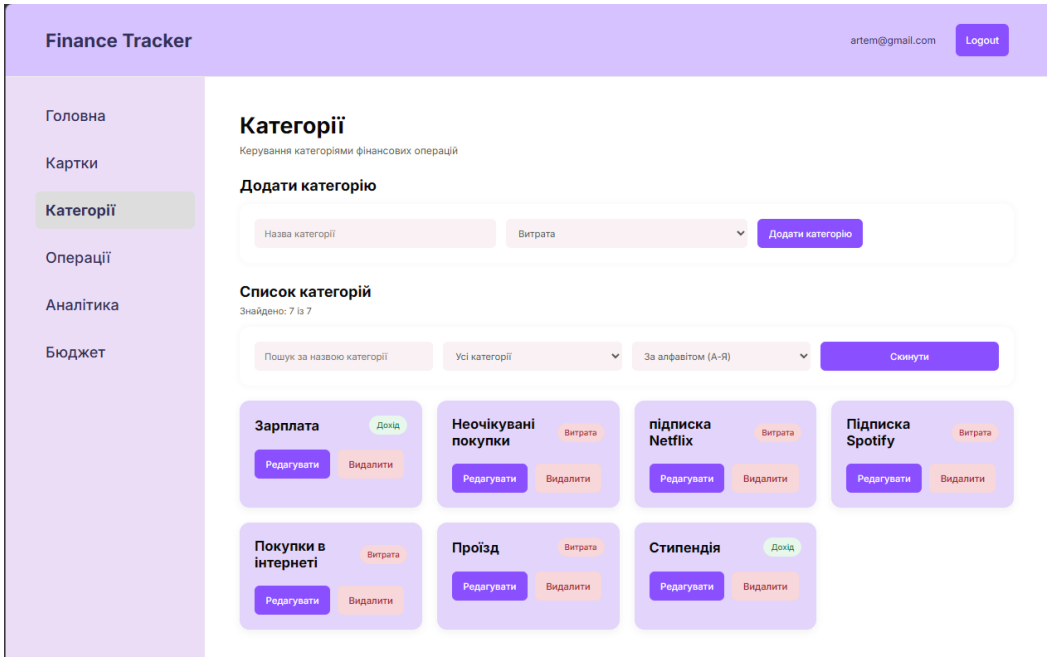

Рисунок 4.6 — Сторінка «Категорії»

Щоб створити нову категорію, користувачеві необхідно вказати її назву та вибрати тип. Після збереження, дані автоматично додаються до бази та стають доступними для використання під час реєстрації фінансових операцій.

Система також дозволяє вносити зміни до існуючих категорій. Зокрема, є можливість скоригувати назву категорії або, якщо до неї не прив’язані фінансові операції, змінити її тип.

З метою забезпечення цілісності даних видалення категорії блокується у випадку її прив’язки до записаних у системі фінансових операцій. У таких ситуаціях система виводить відповідне повідомлення про неможливість видалення.

Для полегшення роботи з великою кількістю категорій у системі впроваджено функції пошуку і сортування. Користувач має можливість шукати категорії за назвою, фільтрувати їх за типом (доходи чи витрати) та сортувати список в алфавітному порядку. Використання цих інструментів дозволяє швидко знаходити потрібні категорії, що значно підвищує зручність використання системи. На рисунку 4.7 наведено приклад сортування категорій за алфавітом.

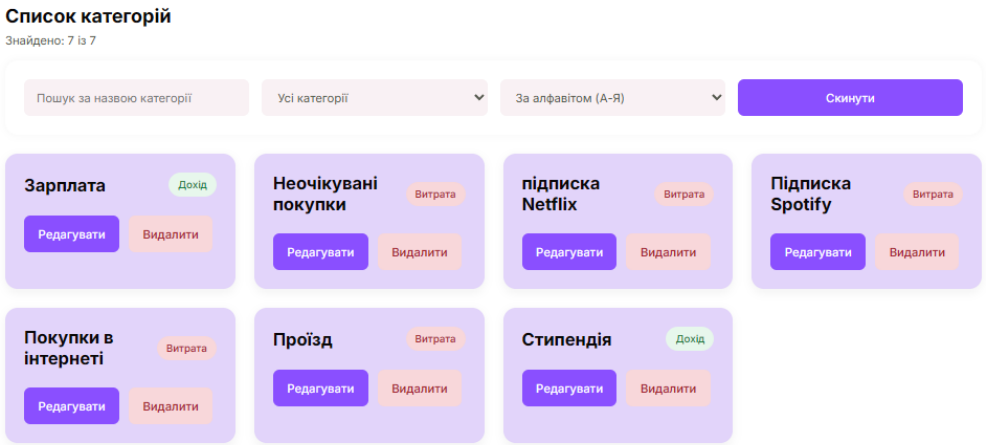

Рисунок 4.7 — Приклад сортування категорій за алфавітом

Отже, підсистема управління категоріями забезпечує зручний спосіб організації фінансових операцій, формуючи базу для подальшого аналізу користувачем своїх доходів і витрат.

4.5 Робота з фінансовими операціями

Основним призначенням вебдодатка є управління фінансовими операціями користувача. Система надає можливість вести облік доходів і витрат, зберігати історію транзакцій та аналізувати накопичені дані для подальшої обробки та прийняття рішень.

Для створення нової фінансової операції користувач заповнює ключові параметри: суму, тип операції, дату виконання, банківську картку та категорію. За потреби можна додати текстовий опис для більш детальної ідентифікації операції. Після збереження всі дані заносяться до бази та з'являються у загальному списку транзакцій. На рисунку 4.8 показано сторінку роботи з фінансовими операціями.

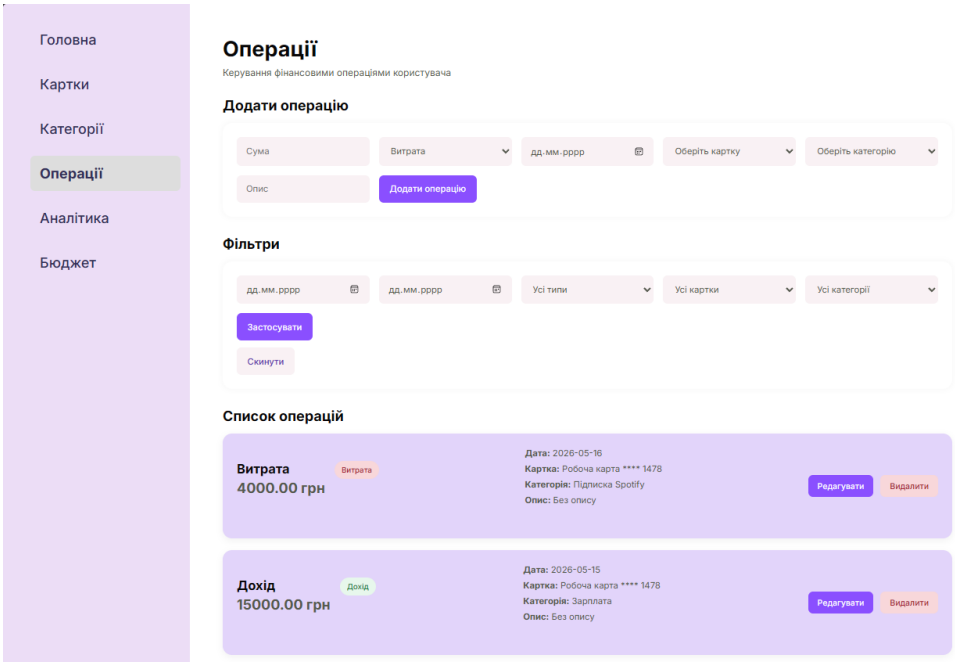

Рисунок 4.8 — Сторінка «Операції»

Для полегшення пошуку необхідних даних у системі впроваджено функцію фільтрації операцій. Користувач має змогу сортувати записи за періодом часу, типом операції, банківською карткою чи категорією. Завдяки цьому інструменту можна швидко знайти відповідну інформацію, що значно спрощує процес аналізу

та управління фінансами. На рисунку 4.9 наведено приклад фільтрації фінансових операцій.

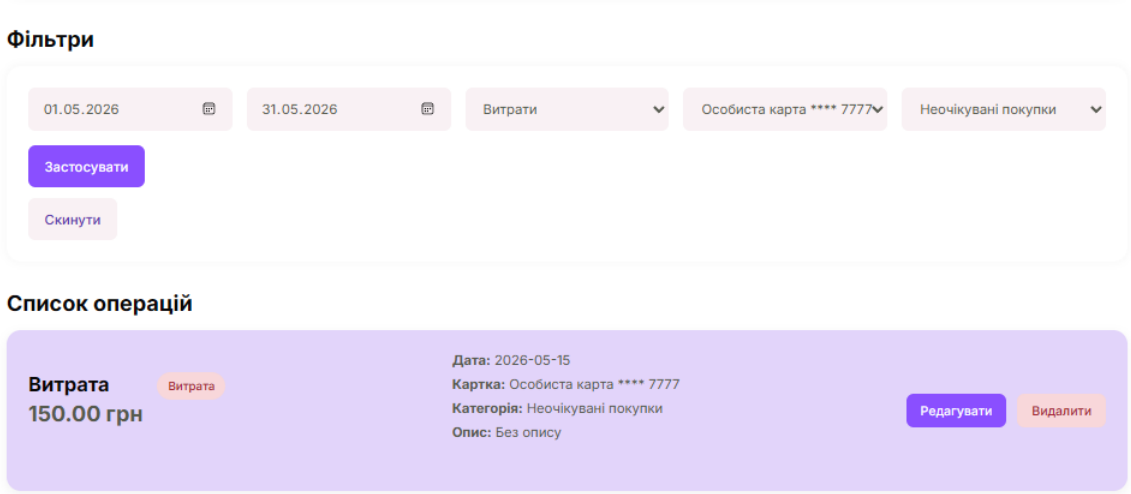

Рисунок 4.9 — Приклад фільтрації фінансових операцій

Кожну фінансову операцію можна змінювати або видаляти за допомогою доступних елементів керування. У режимі редагування користувач отримує змогу оновлювати всі основні параметри транзакції, а внесені зміни автоматично фіксуються у списку операцій.

Отже, передбачений функціонал охоплює всі аспекти роботи з фінансовими операціями: створення, перегляд, редагування, видалення та фільтрацію даних. Це забезпечує зручну платформу для ефективного обліку та аналізу фінансової діяльності користувача.

4.6 Перегляд аналітики та статистики

У вебзастосунку передбачено окремий розділ аналітики, який дозволяє узагальнювати та візуально представляти фінансову інформацію користувача. Цей

модуль автоматично обробляє дані про доходи та витрати, забезпечуючи інструменти для оцінки поточного стану персонального бюджету. Завдяки аналітичним функціям користувач може відстежувати динаміку фінансових показників, аналізувати структуру витрат, виділяти найактивніші категорії чи банківські картки, а також отримувати поради щодо більш ефективного управління власними коштами.

Для зручності сприйняття результати аналізу представлені у формі графіків, діаграм і інформаційних блоків із ключовими статистичними даними. На сторінці аналітики користувач може встановити бажаний період аналізу, після чого система самостійно виконує всі необхідні обчислення. До основних показників, які відображаються, належать: сума доходів і витрат, поточний баланс, загальна кількість операцій, середні доходи і витрати на день, а також підсумкова оцінка фінансового стану за обраний часовий проміжок. На рисунку 4.10 зображено сторінку «Аналітика».

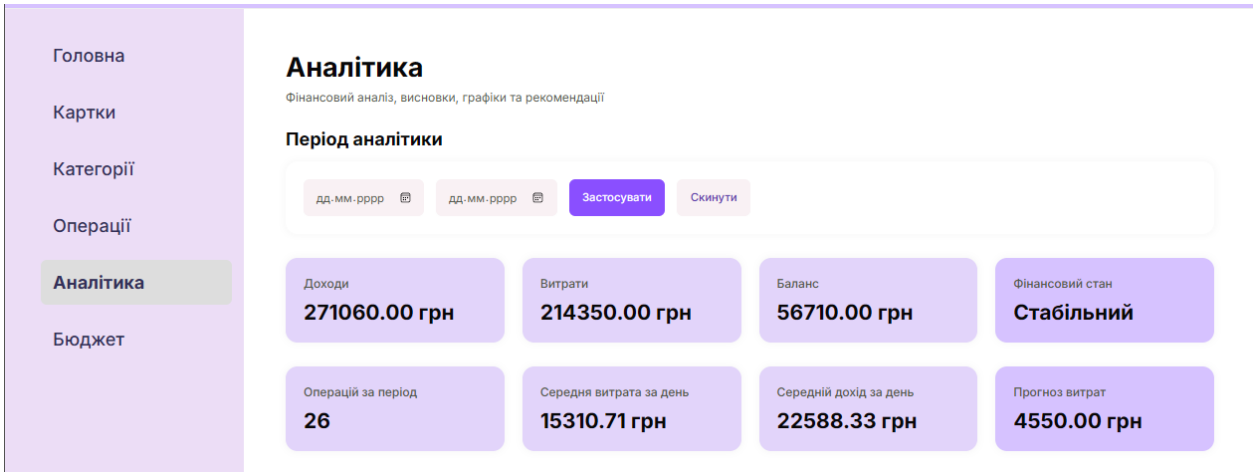

Рисунок 4.10 — Сторінка «Аналітика»

Для всеосяжної оцінки фінансової діяльності інтегровано функцію визначення рейтингу фінансових періодів. Цей механізм аналізує співвідношення доходів та витрат, величину балансу та інші показники за кожен місяць, створюючи підсумкову рейтингову оцінку за шкалою від 1 до 10. Отримана аналітика дозволяє

виявляти більш і менш успішні періоди. На рисунку 4.11 наведено фінансовий рейтинг місяців.

Рисунок 4.11 — Фінансовий рейтинг місяця

Система також передбачає функцію прогнозування витрат до кінця поточного місяця. Розрахунки виконуються із використанням даних про вже здійснені витрати та середні щоденні витрати. Додатково формується аналітичний висновок із рекомендаціями щодо покращення фінансової дисципліни й оптимального використання коштів. На рисунку 4.12 наведено прогноз витрат до кінця місяця, аналітичний висновок та фінансові рекомендації.

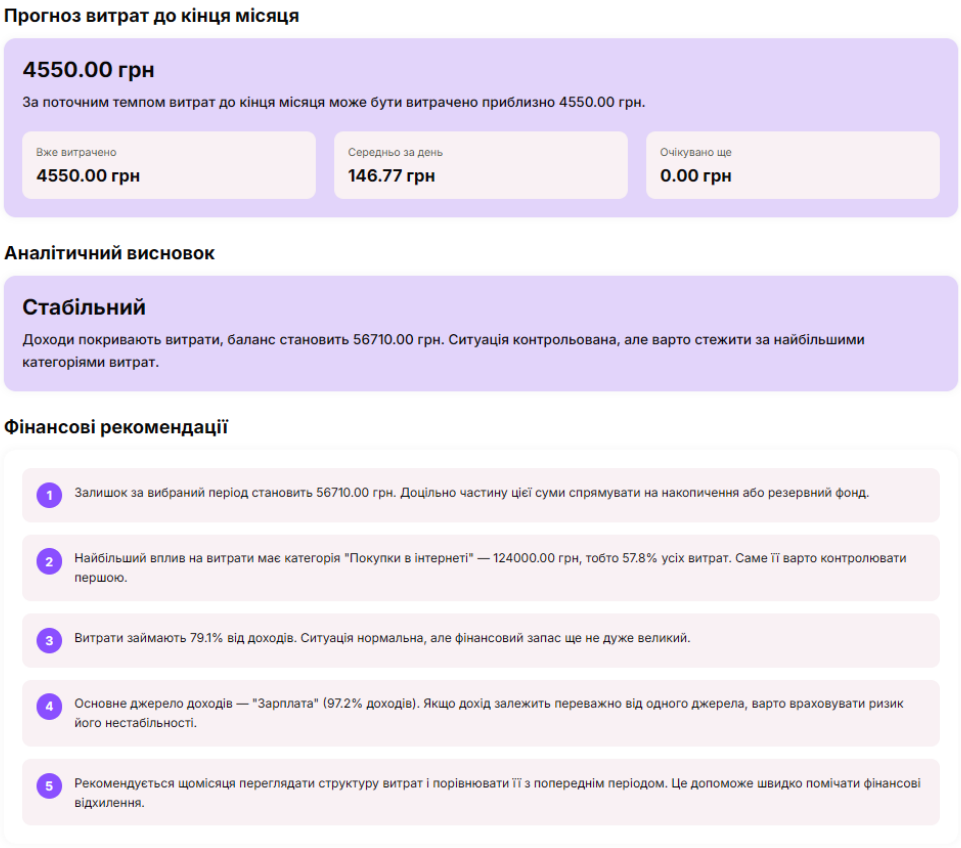

Рисунок 4.12 — Прогноз витрат до кінця місяця, аналітичний висновок та фінансові рекомендації

Для відстеження тенденцій змін у фінансовій активності система пропонує порівняльний аналіз поточних і попередніх періодів. Користувач може переглядати зміни доходів, витрат та балансу у числовому вираженні чи у вигляді відсоткових значень. Для зручності додається графік порівняння доходів і витрат за місяцями, що дозволяє оцінити загальну динаміку. На рисунку 4.13 наведено порівняння з минулим місяцем та помісячне порівняння доходів та витрат.

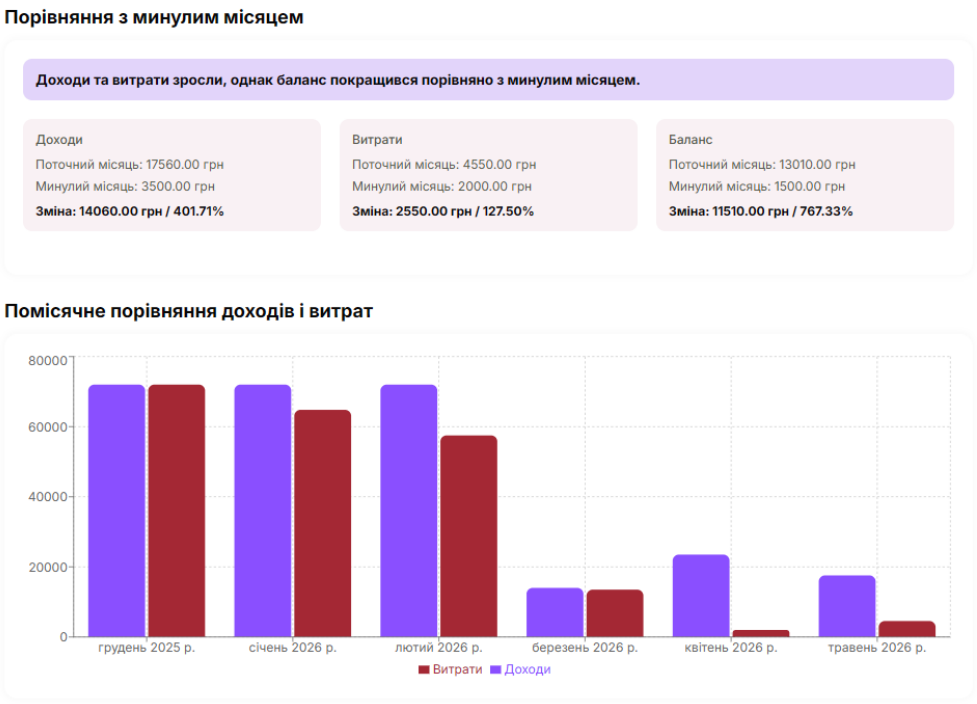

Рисунок 4.13 — Порівняння з минулим місяцем та помісячне порівняння доходів і витрат

Щоб оцінити рівень активності користувача, система окремо відображає кількість виконаних фінансових операцій помісячно. Це допомагає визначати періоди з найбільш інтенсивним використанням додатка і загальну динаміку активності. На рисунку 4.14 наведено графік кількості операцій по місяцях.

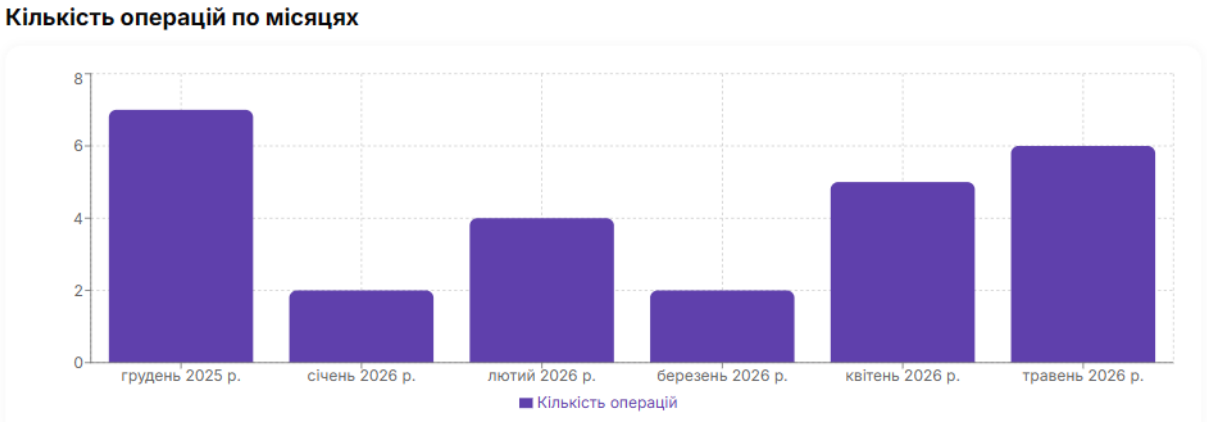

Рисунок 4.14 — Кількість операцій по місяцях\

Для виявлення основних категорій витрат у системі впроваджено аналіз розподілу бюджету. Інформація представлена у вигляді кругових діаграм, що відображають частки кожної категорії у загальних витратах, а також статистику за днями тижня, що допомагає фіксувати піки активності. На рисунку 4.15 наведено статистику витрат за категоріями та витрат за днями тижня.

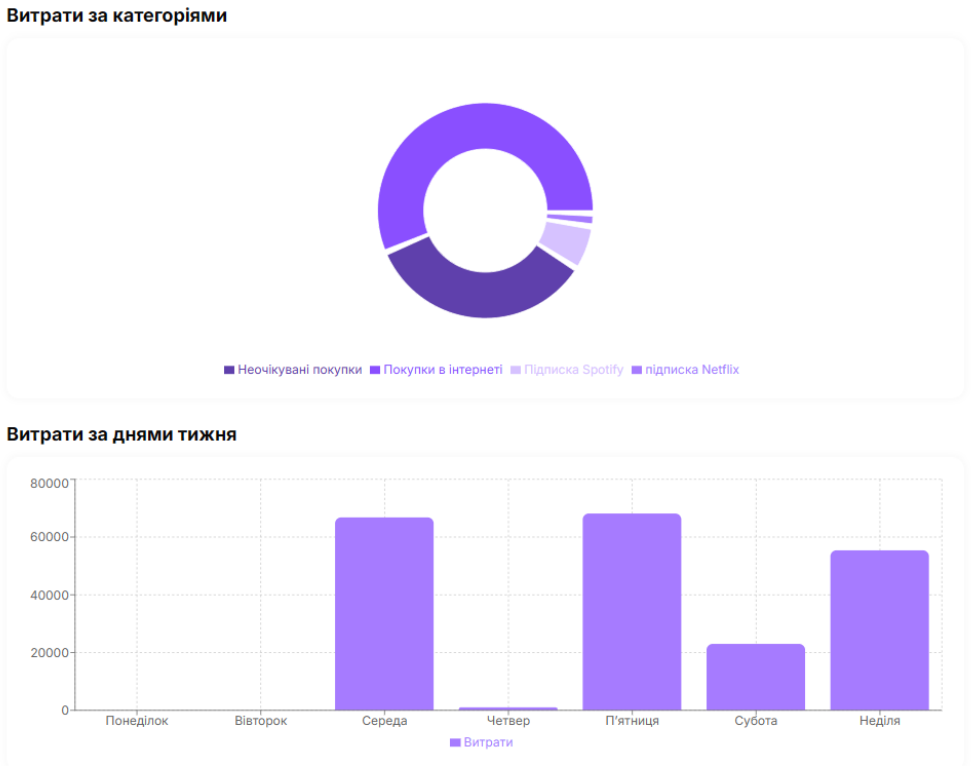

Рисунок 4.15 — Витрати за категоріями та витрати за днями тижня

Окремий розділ аналітики зосереджений на детальному дослідженні використання банківських карток. Програма підсумовує витрати для кожної картки та відображає їх частоту використання під час фінансових операцій. Це допомагає користувачу оцінити, які картки є найактивнішими у використанні та наскільки вони впливають на загальну фінансову діяльність. На рисунку 4.16 зображено статистику витрат за банківськими картками та частоту використання карток.

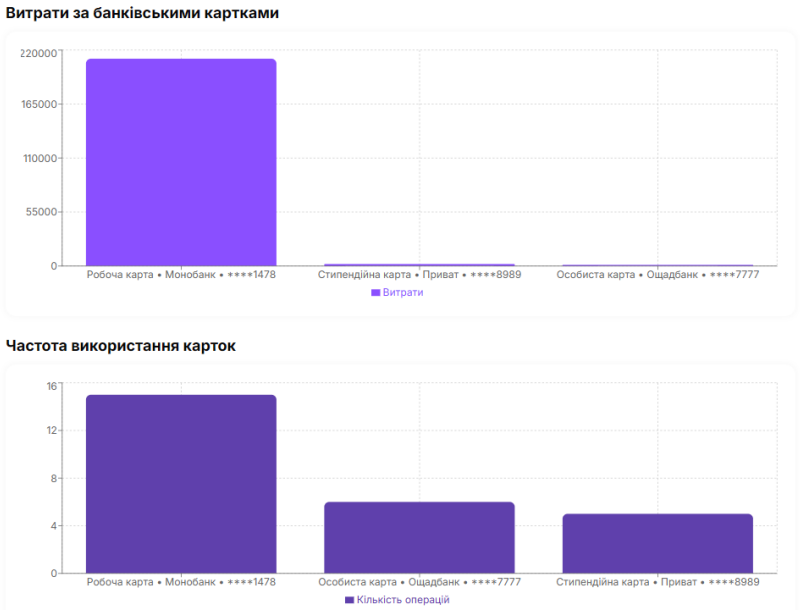

Рисунок 4.16 — Витрати за банківськими картками та частота використання карток

Для детальнішого розуміння руху коштів додатково розроблено графік, що показує динаміку доходів та витрат. Цей інструмент ілюструє зміни фінансових показників у часовій перспективі, що дозволяє аналізувати зв’язок між надходженнями та витратами. Завдяки такому підходу користувач може простежити тенденції у фінансах, визначити періоди підвищеної активності або виявити можливі відхилення від звичайного шаблону поведінки. На рисунку 4.17 наведено графік динаміки доходів і витрат.

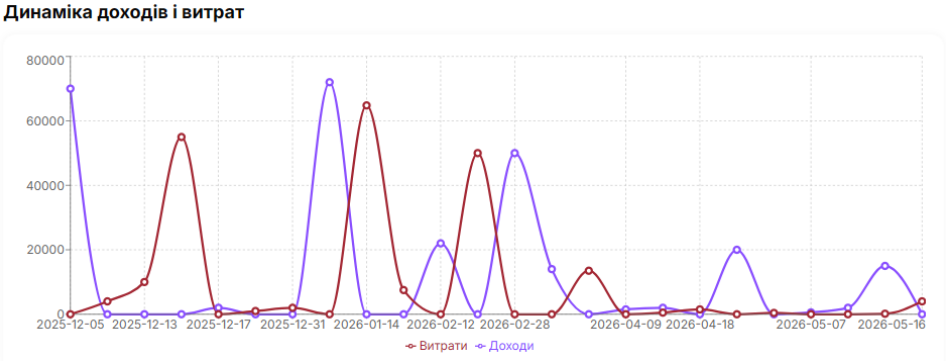

Рисунок 4.17 — Графік динаміки доходів і витрат

Таким чином, аналітичний модуль пропонує користувачеві широкий спектр інструментів для аналізу фінансової активності, оцінки поточного стану бюджету, прогнозування майбутніх витрат і прийняття зважених фінансових рішень.

4.7 Робота з плануванням бюджету

У вебзастосунку наявний модуль бюджетного планування. Його основне завдання — допомогти користувачам встановлювати ліміти витрат для окремих категорій і відстежувати їх дотримання протягом обраного періоду.

У процесі формування бюджету користувач визначає категорію витрат, встановлює ліміт і період його дії. Система в подальшому здійснює автоматичний моніторинг витрат і розраховує відсоток використання заданого обмеження.

На сторінці бюджету користувач відображається заданий ліміт, фактична сума витрат, залишок доступних коштів і частку використаного бюджету у відсотковому вигляді. Для зручності додано графічний індикатор, який дає змогу швидко оцінити стан виконання бюджету. На рисунку 4.18 зображено сторінку планування бюджету.

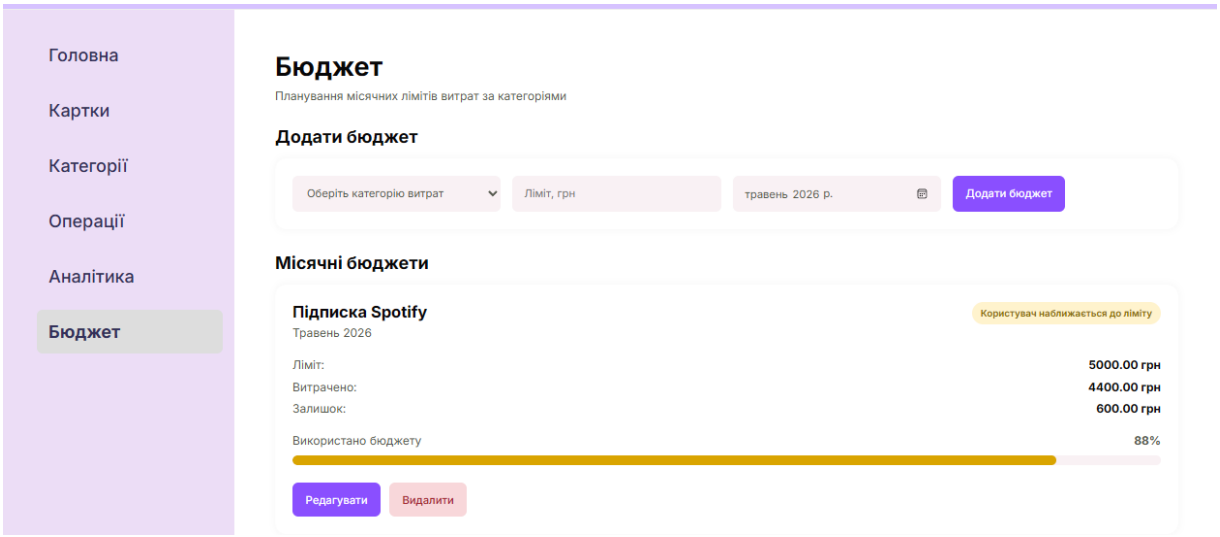

Рисунок 4.18 — Сторінка «Бюджет»

У разі наближення фактичних витрат до встановленого ліміту система генерує попереджувальне повідомлення. Це дозволяє користувачеві оперативно реагувати та уникнути перевищення запланованих витрат, що сприяє більш зваженому управлінню фінансами.

Крім того, користувач має можливість у будь-який момент оновити або видалити створені бюджети. Усі внесені зміни автоматично враховуються системою під час наступних розрахунків і аналізу фінансової активності.

Таким чином, завдяки модулю бюджетного планування стає можливим ефективно контролювати витрати за категоріями та більш розумно керувати особистими фінансами.

ВИСНОВКИ

1. Проведено детальний аналіз предметної області, який охоплював персональний фінансовий облік і фінансову аналітику, з метою виявлення основних проблем у контролі доходів і витрат користувачів та визначення ключових вимог до сучасних систем управління фінансами. У результаті встановлено необхідність впровадження автоматизованих інструментів для обліку фінансових операцій, бюджетування й аналітичної обробки даних, що сприяє підвищенню ефективності управління особистими коштами.

2. Було здійснено аналіз наявних програмних рішень для фінансового обліку та аналітики, зокрема таких продуктів, як Budget Flow, Money Flow, Spendee та Monefy. Під час порівняння їх функціоналу, було виявлено переваги та недоліки кожного додатку, на основі чого було сформовано функції для власної програмної розробки.

3. На основі отриманих даних були визначені функціональні та нефункціональні вимоги до вебзастосунку. Визначені вимоги стали основою для подальшого проєктування системи, що забезпечило ефективну реалізацію необхідного функціоналу роботи з фінансовими даними.

4. У процесі розробки здійснено проєктування архітектури системи, структури бази даних, моделей даних, REST API та користувацького інтерфейсу. У результаті сформовано цілісну структуру вебзастосунку, яка забезпечує взаємодію між клієнтською та серверною частинами і має можливість масштабування для подальшого розширення функцій.

5. Вебзастосунок для обліку персональних фінансів був успішно розроблений, включаючи інтеграцію аналітичного модуля. Система забезпечує такі можливості, як авторизація і автентифікація користувачів, управління банківськими картками, категоризація фінансових операцій, супровід доходів і витрат, а також функціонал для планування бюджету і контролю фінансових лімітів.

6. Також впроваджено модуль аналітики та статистики, який виконує автоматизований розрахунок основних фінансових показників, генерує графіки й діаграми, аналізує структуру витрат, здійснює прогнозування майбутніх фінансових потреб та оцінює загальний стан фінансів користувача. Крім цього, система пропонує рекомендації щодо ефективнішого управління особистими фінансами.

7. Тестування вебзастосунку підтвердило його стабільну роботу: перевірено коректність функціонування всіх модулів, точність обробки та збереження даних, а також відповідність програмного продукту визначеним вимогам.

8. План подальшого розвитку системи передбачає інтеграцію з банківськими сервісами для автоматичного отримання даних про фінансові операції, розширення можливостей аналітики, вдосконалення прогнозування фінансових показників та створення мобільної версії застосунку для зручнішого використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Financial accounting. *Investopedia*. URL:https://www.investopedia.com/terms/f/financialaccounting.asp (дата звернення: 20.05.2026).

2. Was ist Finanzanalytik?. *Actian*. URL: https://www.actian.com/what-is-financial-analytics (дата звернення: 21.05.2026).

3. IBM. What is data visualization? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/data-visualization (дата звернення: 22.05.2026).

4. Budget flow | expense tracker: control all of your finances with ease: enter your budget, get insights and keep track of all your income and expenses. | press kit. *ImpressKit: Best way to create Press Kit for apps*. URL: https://impresskit.net/34bff149-d528-41cd-b496-968a4b0e1557 (дата звернення: 21.06.2026).

5. Money Flow - Облік витрат та доходів. *Money Flow - Spending Tracker and Budget Manager*. URL: https://moneyflow.cloud/uk (дата звернення: 23.06.2026).

6. What is spendee? - spendee help center. *Spendee Help Center*. URL: https://help.spendee.com/article/114-what-is-spendee (дата звернення: 23.05.2026).

7. Monefy | budget & track your money. *Monefy | Budget & Track Your Money*. URL: https://www.monefy.com/ (дата звернення: 23.05.2026).

8. Рогнєда Княжина. Архітектура вебзастосунків 2024: ультимативний гайд для розробників. *robot_dreams - онлайн-курси для фахівців у сфері big data, machine learning, data science | Робот Дрімс*. URL: https://robotdreams.cc/uk/blog/567-arhitektura-vebzastosunkiv (дата звернення: 23.06.2026).

9. Client-Server architecture explained with examples, diagrams and real-world applications. *Medium*. URL: https://medium.com/@devharshgupta.com/client-server-architecture-explained-with-examples-diagrams-and-real-world-applications-407e9e04e2d1 (дата звернення: 23.06.2026).

10. GeeksforGeeks. REST API Introduction - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/node-js/rest-api-introduction/ (дата звернення: 24.05.2026).

11. Fielding R. T. Architectural styles and the design of network-based software architectures : Doctoral dissertation. Irvine, 2000. 162 p. URL: https://dl.acm.org/doi/10.5555/932295 (date of access: 24.05.2026).

12. ambros. Як захистити API за допомогою JWT | Serverion. *Serionion*. URL: https://www.serverion.com/uk/uncategorized/how-to-secure-apis-with-jwts/ (дата звернення: 24.05.2026).

13. JSON web token introduction - jwt.io. *JSON Web Tokens - jwt.io*. URL: https://www.jwt.io/introduction#difference-decoding-encoding-jwt (дата звернення: 24.05.2026).

14. Що таке Node.js? Основи серверної розробки на JavaScript. *DevZone*. URL: https://devzone.org.ua/post/shcho-take-nodejs-osnovy-servernoyi-rozrobky-na-javascript (дата звернення: 25.05.2026).

15. GeeksforGeeks. Express.js Tutorial - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/node-js/express-js/ (дата звернення: 25.05.2026).

16. IBM. What Is PostgreSQL? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/postgresql (дата звернення: 25.05.2026).

17. Klymenko O. Хешування паролів: використання солі та bcrypt. *Друкарня*. URL: https://drukarnia.com.ua/articles/kheshuvannya-paroliv-vikoristannya-soli-ta-bcrypt-fsme-#heading-3-607 (дата звернення: 26.05.2026).

18. Що таке React JS і для чого він потрібен?. *DAN.IT Education*. URL: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/ (дата звернення: 26.05.2026).

19. Axios Library in JavaScript. *IT Notes*. URL: https://www.it-notes.wiki/javascript/axios-library-in-javascript/ (дата звернення: 27.05.2026).

20. Recharts - data visualization made easy awesome react. *Awesome React DEV*. URL: https://awesome-react.dev/library/recharts (дата звернення: 27.05.2026).

ДОДАТОК А

Реалізація підсистеми обліку фінансових операцій

Програмні засоби:

— JavaScript;

— Node.js;

— Express.js;

— Sequelize ORM;

— PostgreSQL.

Програмний код:

```
import { validationResult } from "express-validator"
import { Op } from "sequelize"
import { Operation } from "../models/Operation.js"
import { Card } from "../models/Card.js"
import { Category } from "../models/Category.js"

class OperationController {
  async createOperation(req, res) {
    try {
      const errors = validationResult(req)
      if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() })
      }
      const { amount, type, description, operationDate, cardId, categoryId } = req.body
      const userId = req.user.id
```

```
  const card = await Card.findOne({
    where: { id: cardId, userId }
  })
  if (!card) {
    return res.status(404).json({ message: "Картку не знайдено" })
  }
  const category = await Category.findOne({
    where: { id: categoryId, userId }
  })
  if (!category) {
    return res.status(404).json({ message: "Категорію не знайдено" })
  }
  if (category.type !== type) {
    return res.status(400).json({
      message: "Тип категорії не відповідає типу операції"
    })
  }
  const operation = await Operation.create({
    amount,
    type,
    description,
    operationDate,
    cardId,
    categoryId,
    userId
  })
  return res.status(201).json(operation)
} catch (err) {
  console.log(err)
  return res.status(400).json({ message: "Не вдалося створити операцію" })
```

```
  }
}
async getOperations(req, res) {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() })
    }
    const userId = req.user.id
    const { type, categoryId, cardId, dateFrom, dateTo } = req.query
    if (dateFrom && dateTo && new Date(dateFrom) > new Date(dateTo)) {
      return res.status(400).json({
        message: "dateFrom не може бути пізніше за dateTo"
      })
    }
    const where = { userId }
    if (type) {
      where.type = type
    }
    if (categoryId) {
      where.categoryId = categoryId
    }
    if (cardId) {
      where.cardId = cardId
    }
    if (dateFrom && dateTo) {
      where.operationDate = {
        [Op.between]: [dateFrom, dateTo]
      }
    } else if (dateFrom) {
```

```
      where.operationDate = {
        [Op.gte]: dateFrom
      }
    } else if (dateTo) {
      where.operationDate = {
        [Op.lte]: dateTo
      }
    }

    const operations = await Operation.findAll({
      where,
      include: [
        {
          model: Card,
          attributes: ["id", "name", "bankName", "lastFourDigits", "cardType"]
        },
        {
          model: Category,
          attributes: ["id", "name", "type"]
        }
      ],
      order: [["operationDate", "DESC"]]
    })

    return res.json(operations)
  } catch (err) {
    console.log(err)
    return res.status(500).json({ message: "Не вдалося отримати операції" })
  }
}
```

```
async updateOperation(req, res) {
  try {
    const errors = validationResult(req)
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() })
    }

    const { id } = req.params
    const userId = req.user.id

    const { amount, type, description, operationDate, cardId, categoryId } = req.body

    const operation = await Operation.findOne({
      where: { id, userId }
    })

    if (!operation) {
      return res.status(404).json({ message: "Операцію не знайдено" })
    }

    const updateData = {}

    if (amount !== undefined) updateData.amount = amount
    if (type !== undefined) updateData.type = type
    if (description !== undefined) updateData.description = description
    if (operationDate !== undefined) updateData.operationDate = operationDate

    if (cardId !== undefined) {
      const card = await Card.findOne({
```

```
    where: { id: cardId, userId }
  })

  if (!card) {
    return res.status(404).json({ message: "Картку не знайдено" })
  }

  updateData.cardId = cardId
}

if (categoryId !== undefined) {
  const category = await Category.findOne({
    where: { id: categoryId, userId }
  })

  if (!category) {
    return res.status(404).json({ message: "Категорію не знайдено" })
  }

  const finalType = type !== undefined ? type : operation.type

  if (category.type !== finalType) {
    return res.status(400).json({
      message: "Тип категорії не відповідає типу операції"
    })
  }

  updateData.categoryId = categoryId
}
```

```
if (categoryId === undefined && type !== undefined) {
  const category = await Category.findOne({
    where: { id: operation.categoryId, userId }
  })

  if (category && category.type !== type) {
    return res.status(400).json({
      message: "Новий тип операції не відповідає поточній категорії"
    })
  }
}

if (Object.keys(updateData).length === 0) {
  return res.status(400).json({ message: "Немає даних для оновлення" })
}
await Operation.update(updateData, {
  where: { id, userId }
})
const updatedOperation = await Operation.findOne({
  where: { id, userId },
  include: [
    {
      model: Card,
      attributes: ["id", "name", "bankName", "lastFourDigits", "cardType"]
    },
    {
      model: Category,
      attributes: ["id", "name", "type"]
    }
  ]
```

```
      })
      return res.json(updatedOperation)
    } catch (err) {
      console.log(err)
      return res.status(400).json({ message: "Не вдалося оновити операцію" })
    }
  }
  async deleteOperation(req, res) {
    try {
      const { id } = req.params
      const userId = req.user.id
      const deletedRows = await Operation.destroy({
        where: { id, userId }
      })
      if (deletedRows > 0) {
        return res.status(204).send()
      }
      return res.status(404).json({ message: "Операцію не знайдено" })
    } catch (err) {
      console.log(err)
      return res.status(500).json({ message: "Не вдалося видалити операцію" })
    }
  }
}

export default new OperationController()
```