ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

(повне найменування вищого навчального закладу)

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

(повне найменування інституту, назва факультету (відділення))

КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи

магістра

(освітній рівень)

на тему:

«Розробка системи управління фінансами та автоматизація особистого бюджету»

Виконав: здобувач 6 курсу, групи 6ПР2

спеціальності

121 – «Інженерія програмного забезпечення»

(шифр і назва спеціальності)

Мусатов Володимир Михайлович

(прізвище та ініціали)

Керівник к.т.н., доцент Доровська І.О.

(прізвище та ініціали)

Рецензент к.т.н., доцент Григорова А.А.

(прізвище та ініціали)

Хмельницький – 2025

Херсонський національний технічний університет

(повне найменування вищого навчального закладу)

Факультет, відділення **Інформаційних технологій та дизайну**

Кафедра **Програмних засобів і технологій**

Освітній рівень **магістр**

Спеціальність **121 – Інженерія програмного забезпечення**

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗіТ

Програмних засобів і технологій

д.т.н. доц. О.Є. Огнєва

“___” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мусатов Володимир Михайлович

(прізвище, ім’я, по батькові)

1.Тема роботи «Розробка систем управління фінансами та автоматизація особистого бюджету»

керівник роботи к.т.н. доцент Доровська І.О.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від 15 .09 .2025 р. № 417–с

2. Строк подання студентом роботи

3. Вихідні дані до роботи літературні та періодичні джерела, матеріали переддипломної практики

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити):

1. Аналіз предметної області систем управління особистими фінансами

2. Аналіз вимог до користувацького режиму та формування технічної специфікації

3. Проектування бази даних та REST API для взаємодії клієнтських застосунків із серверною частиною

4. Розробка, інтеграція та тестування API-системи

5. Перелік додатків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітки
1	Отримання завдання	15.09.2025	Виконано
2	Підбір літератури та джерел по використанню чат ботів в Телеграм	16.09.2025 – 22.09.2025	Виконано
3	Аналіз предметної області та існуючих SEO сервісів	23.09.2025 – 30.09.2025	Виконано
4	Формування технічних вимог і постановки завдання	01.10.2025 – 07.10.2025	Виконано
5	Проектування бази даних та моделювання бізнес-процесів	08.10.2025 – 16.10.2025	Виконано
6	Реалізація основних модулів	17.10.2025 –	Виконано

	системи	31.10.2025	
7	Реалізація автоматизації особистого бюджету	01.11.2025 10.11.2025	–Виконано
8	Аналітика базових механізмів прогнозування	11.11.2025 20.11.2025	–Виконано
9	Тестування системи, оптимізація, перевірка даних	21.11.2025 30.11.2025	–Виконано
10	Оформлення пояснювальної записки	01.12.2025 08.12.2025	–Виконано
11	Підготовка до захисту та захист кваліфікаційної роботи	09.12.2025 22.12.2025	–Виконано

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

Керівник роботи _____ І.О. Доровська
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна магістерська робота : 90 сторінок, 3 додатки, 53 джерел.

Об’єктом дослідження: є процес управління особистими фінансами користувача.

Предметом дослідження: є методи, моделі та програмні засоби автоматизації обліку доходів і витрат, аналіз та планування особистого бюджету.

Мета дослідження: Створення програмної системи, яка забезпечує автоматизований облік особистих фінансів, формування аналітичних звітів та візуалізацію фінансових даних для оптимізації управління особистим бюджетом.

Методи дослідження. Аналіз і систематизацію предметної області, системний підхід до проектування інформаційних систем, моделювання

бізнес-процесів, проектування реляційної бази даних, використання методів об'єктно-орієнтованого програмування, а також методи тестування та валідації програмного забезпечення.

Результати дослідження. В результаті виконання випускної роботи створено api–додаток, для аналітики особистого бюджету, та інтеграції у ваб або мобільний додаток.

КЛЮЧОВІ СЛОВА: ОСОБИСТІ ФІНАНСИ, УПРАВЛІННЯ БЮДЖЕТОМ, API, АВТОМАТИЗАЦІЯ, HTML, LARAVEL, MySQL, ІНФОРМАЦІЙНА СИСТЕМА.

АНОТАЦІЯ

Магістерська робота присвячена розробці системи управління фінансами та автоматизації особистого бюджету на основі серверної API-архітектури. У роботі розглянуто підходи до управління особистими фінансами, проаналізовано існуючі програмні рішення та визначено основні вимоги до інформаційної системи автоматизації бюджету.

У межах роботи спроектовано та реалізовано API-систему управління особистими фінансами з використанням фреймворку Laravel та реляційної бази даних MySQL. Розроблена система забезпечує облік доходів і витрат, класифікацію фінансових операцій, формування аналітичних звітів та можливість інтеграції з клієнтськими веб- і мобільними застосунками.

Практична цінність роботи полягає у можливості використання розробленої API-системи як серверної частини для програмних продуктів з управління особистим бюджетом, а також у її подальшому розширенні та масштабуванні.

Результати роботи можуть бути використані у сфері розробки інформаційних систем, фінансових сервісів та навчальному процесі.

ABSTRACT

The bachelor's thesis has the following structural parts: introduction, four chapters, conclusions, references and appendices.

The introduction reflects the purpose and relevance of developing a personal finance management system and personal budget automation.

In the first chapter of the qualification work, the research and analysis of the subject area of personal finance management were carried out, and existing software solutions were reviewed.

The second section is devoted to the analysis of requirements for the software product and the development of project specifications for an API-based system.

The third chapter describes the software product design process, including the development of the system architecture, database structure, and interaction logic between system components.

The fourth section provides an overview of technical aspects, justifies the choice of development tools and technologies, including the Laravel framework and MySQL database, considers the process of developing, testing, and implementing the API system, analyzes the results obtained, and outlines recommendations for further development.

The conclusions highlight the practical significance of the developed API system, summarize the results of the qualification work, and identify prospects for further improvement and expansion of the personal finance management system.

The list of references contains a list of scientific publications, literature, and information sources used during the development of the project.

The appendices contain fragments of program code, database structure, API specifications, and interface examples.

ЗМІСТ

ВСТУП **10**

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ РІШЕНЬ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ **13**

1.1 Поняття та значення управління особистими фінансами 13

1.2 Автоматизація особистого бюджету як інформаційна система 16

1.3 Аналіз існуючих програмних рішень для управління особистими фінансами 19

1.4 Системи управління фінансами: поняття та класифікація 27

РОЗДІЛ 2. СИСТЕМНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ **31**

2.1 Об’єкт дослідження 31

2.2 Предмет дослідження 35

2.3 Функціональні та нефункціональні вимоги до системи 38

2.4 Алгоритм роботи арі системи 41

РОЗДІЛ 3. МЕТОДИ ТА ЗАСОБИ ВИРІШЕННЯ ПРОБЛЕМИ **45**

3.1 Методи розробки 45

3.2 Архітектура та структура системи 49

3.3 Реалізація бази даних 50

3.4 Реалізація API та механізм авторизації 54

3.5 Визначення та реалізація архітектурних патернів і стандартів кодуванн 59

3.6 Вибір і налаштування інструментарію 61

3.7 Обґрунтування вибору СУБД 64

3.8 Використання Laravel та Laravel Sail у практичній реалізації 66

3.9 Тестування програмного додатку 67

3.10 Робота користувача з програмним додатком 70

3.11 Аналіз отриманих навичок та досвіду в контексті подальшого використання в розробці 72

ВИСНОВКИ 74

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 77

ДОДАТОК А 81

ДОДАТОК Б. 88

ДОДАТОК В 90

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface, програмний інтерфейс прикладного програмування;

БД – база даних;

СУБД – система управління базами даних;

MySQL – реляційна система управління базами даних;

Laravel – PHP-фреймворк для розробки вебзастосунків та API ;

REST – Representational State Transfer, архітектурний стиль побудови вебсервісів;

HTML – мова розмітки гіпертексту (Hyper Text Markup Language) ;

JSON – JavaScript Object Notation, формат обміну даними ;

CRUD – Create, Read, Update, Delete, базові операції роботи з даними ;

MVC – Model–View–Controller, архітектурний шаблон проєктування;

JWT – JSON Web Token, механізм автентифікації та авторизації ;

Eloquent ORM – об’єктно-реляційний мапер фреймворку Laravel;

SQL – Structured Query Language, мова структурованих запитів ;

ВСТУП

Актуальність теми дослідження. У сучасних умовах глобальної цифровізації економіки та стрімкого розвитку фінансових технологій питання ефективного управління особистим капіталом переходить з площини теоретичних знань у категорію життєво необхідних навичок. Щоденне зростання кількості транзакцій, розширення джерел доходів (від основної заробітної плати до пасивних інвестицій та фрилансу) та складність структури витрат роблять традиційні методи обліку малоефективними. Необхідність у миттєвому аналізі фінансового стану та довгостроковому плануванні вимагає впровадження високотехнологічних автоматизованих інформаційних систем. Такі рішення не лише підвищують фінансову дисципліну користувача, а й дозволяють мінімізувати когнітивне навантаження, оптимізувати видатки та приймати стратегічно обґрунтовані фінансові рішення на основі реальних статистичних даних.

Попри широкий спектр існуючих програмних продуктів на ринку, значна частина з них має суттєві недоліки: відсутність гнучких налаштувань під індивідуальні потреби користувача, закриту архітектуру, що унеможливлює інтеграцію з сторонніми сервісами, або надмірну складність інтерфейсу. У зв’язку з цим виникає гостра потреба у розробці універсальної серверної API-системи. Такий підхід дозволяє створити єдине «джерело істини» для фінансових даних, забезпечуючи централізоване зберігання та обробку інформації, що може бути використана як база для побудови екосистеми різних клієнтських застосунків — від мобільних додатків до інтелектуальних вебпанелей. Вибір фреймворку Laravel та СУБД MySQL для реалізації цієї задачі обумовлений їхньою високою надійністю, швидкістю розробки та відповідністю сучасним стандартам безпеки.

Метою магістерської роботи є створення програмної системи, яка забезпечує автоматизований облік особистих фінансів, формування аналітичних звітів та візуалізацію фінансових даних для оптимізації управління особистим бюджетом.

Для досягнення поставленої мети у роботі необхідно розв’язати такі **завдання:**

– Здійснити комплексний аналіз предметної області та виявити ключові закономірності процесу управління особистими фінансами;

– Провести порівняльне дослідження існуючих програмних аналогів, визначивши їхні функціональні переваги та критичні недоліки;

– Сформулювати детальні технічні вимоги (функціональні та нефункціональні) до проектованої системи;

– Спроєктувати логічну та фізичну структуру реляційної бази даних, а також архітектуру взаємодії серверних компонентів;

– Реалізувати серверну частину системи з використанням laravel та mysql;

– Виконати тестування розробленого api-інтерфейсу на предмет коректності обробки запитів, продуктивності та безпеки даних;

Об’єктом дослідження: є процес управління особистими фінансами користувача.

Предметом дослідження: є методи, моделі та програмні засоби автоматизації обліку доходів і витрат, аналіз та планування особистого бюджету.

Методи дослідження. Аналіз і систематизацію предметної області, системний підхід до проектування інформаційних систем, моделювання бізнес-процесів, проектування реляційної бази даних, використання методів об'єктно-орієнтованого програмування, а також методи тестування та валідації програмного забезпечення.

Практичне значення полягає у створенні готового до експлуатації програмного продукту (Backend-частини), який може стати ядром для розробки широкого спектра фінансових сервісів. Завдяки модульній структурі та використанню API, розроблена система легко масштабується, дозволяючи в майбутньому інтегрувати модулі машинного навчання для

прогнозування витрат або підключати банківські виписки через зовнішні сервіси.

Структура роботи. Магістерська робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ РІШЕНЬ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

1.1 Поняття та значення управління особистими фінансами

У сучасній економічній системі управління особистими фінансами виступає не лише базовим елементом індивідуальної фінансової грамотності, а й фундаментом загального добробуту населення. Цей процес охоплює комплексне поєднання стратегічного планування, систематичного обліку, глибокого аналізу та оперативного контролю за всіма грошовими потоками фізичної особи. Фундаментальне значення ефективного менеджменту власних коштів полягає у створенні надійного підґрунтя для фінансової стабільності, що дозволяє мінімізувати вплив економічних коливань та інфляційних процесів. Раціональний розподіл ресурсів і систематичне формування заощаджень стають ключовими інструментами, які трансформують поточні доходи у капітал, необхідний для досягнення стратегічних життєвих цілей, таких як придбання нерухомості, фінансування освіти або забезпечення високого рівня життя у пенсійному віці. Основними складовими управління особистими фінансами є облік доходів і витрат, планування бюджету, аналіз фінансового стану та прийняття фінансових рішень. У сучасних умовах зростання кількості безготівкових операцій та цифрових фінансових сервісів ручне ведення бюджету стає неефективним, що зумовлює необхідність використання автоматизованих інформаційних систем.

Структурна побудова системи управління персональними фінансами базується на декількох взаємопов'язаних етапах, що починаються з детальної фіксації доходів та витрат і закінчуються прийняттям складних інвестиційних рішень. Важливою частиною цього циклу є формування бюджету, яке потребує постійного зіставлення планових показників із фактичними даними для оцінки загального фінансового стану. Проте трансформація сучасної фінансової екосистеми, яка характеризується стрімким зростанням частки безготівкових розрахунків та інтеграцією цифрових банківських сервісів у

повсякденне життя, суттєво змінює підходи до ведення обліку. Традиційні методи ручного внесення даних стають дедалі менш ефективними через високу вірогідність фрагментації інформації, складність обробки великої кількості транзакцій та значні часові витрати, що об'єктивно зумовлює перехід до використання сучасних автоматизованих інформаційних систем.

Впровадження засобів автоматизації у сферу управління особистим капіталом відкриває принципово нові можливості для користувача, забезпечуючи високу точність та швидкість обробки фінансових даних. Цифрові рішення дозволяють майже повністю нівелювати вплив людського фактора, суттєво знижуючи ризик виникнення помилок при класифікації витрат або розрахунку залишків. Окрім того, автоматизовані системи забезпечують миттєву візуалізацію фінансової динаміки, що надає можливість оперативного доступу до актуальної інформації у будь-який час. Такий підхід не тільки вивільняє інтелектуальний ресурс особистості, але й створює умови для більш якісного прогнозування фінансового майбутнього, оскільки аналітичні інструменти автоматизації здатні виявляти приховані закономірності у споживчій поведінці та пропонувати оптимальні шляхи оптимізації бюджету.

Розвиток інноваційних технологій у фінансовому секторі зумовлює трансформацію традиційних підходів до менеджменту капіталу, де особливе місце посідає інтеграція штучного інтелекту та алгоритмів машинного навчання. Сучасні інтелектуальні системи здатні здійснювати глибокий ретроспективний аналіз витрат, виявляючи приховані закономірності у споживчій поведінці, які зазвичай залишаються поза увагою при ручному способі ведення обліку. Це дозволяє користувачеві отримувати персоналізовані рекомендації щодо оптимізації бюджету та виявлення надлишкових витрат, що в довгостроковій перспективі сприяє вивільненню ресурсів для формування інвестиційного портфеля. Важливим аспектом функціонування таких систем є можливість автоматичного налаштування регулярних платежів та відрахувань у резервні фонди, що реалізує стратегію

першочерговості накопичень і забезпечує систематичне зростання добробуту без необхідності постійного вольового контролю з боку індивіда.

Окрім аналітичної складової, автоматизовані рішення вирішують проблему консолідації даних з різних джерел, включаючи банківські рахунки в різних установах, електронні гаманці та активи у криптовалютах. Створення єдиного інформаційного простору для всіх грошових потоків дозволяє сформувати цілісну картину фінансового стану особистості в режимі реального часу, що є критично важливим для прийняття оперативних управлінських рішень. Водночас перехід до цифрових методів контролю суттєво підвищує рівень безпеки фінансової інформації завдяки використанню сучасних протоколів шифрування та багатофакторної автентифікації, що значно надійніше за фізичні носії даних. У підсумку впровадження автоматизованих інформаційних систем у практику управління персональними фінансами стає не просто технічним вдосконаленням обліку, а стратегічним кроком до підвищення фінансової стійкості та адаптивності особистості в умовах динамічних змін глобальної економічної системи.

Процес цифровізації персонального фінансового менеджменту також тісно пов'язаний із концепцією психологічного комфорту та когнітивного розвантаження користувача. Використання інтелектуальних систем дозволяє подолати так званий поріг входу в управління фінансами, оскільки автоматизація знімає необхідність глибоких знань у бухгалтерському обліку або складних математичних розрахунках. Завдяки інтуїтивно зрозумілим інтерфейсам та візуалізації даних у формі динамічних графіків, фінансова інформація стає доступною для сприйняття на якісно новому рівні, що стимулює свідоме ставлення до кожної витрати. Це сприяє формуванню культури відповідального споживання, де кожне фінансове рішення базується на об’єктивних даних, а не на емоційних імпульсах, що є особливо актуальним в умовах агресивного маркетингу та доступності споживчого кредитування.

Додатковим стратегічним вектором розвитку таких систем є їхня здатність до інтеграції з податковими та юридичними сервісами, що мінімізує ризики виникнення заборгованостей перед державою або фінансовими установами. Автоматизований моніторинг термінів сплати обов'язкових платежів та автоматичне нарахування кешбеку чи бонусних балів дозволяють максимізувати корисність кожної транзакції. Більше того, накопичена протягом тривалого часу база даних про доходи та витрати стає надійним фундаментом для довгострокового фінансового прогнозування на десятиліття вперед, що включає планування пенсійного забезпечення або фінансування великих сімейних проектів. Таким чином, сучасна інформаційна система управління особистими коштами перетворюється на повноцінного цифрового асистента, який забезпечує не лише контроль над поточними ресурсами, а й стратегічний розвиток людського капіталу в умовах інформаційного суспільства.

1.2 Автоматизація особистого бюджету як інформаційна система

Автоматизація процесу управління особистим бюджетом базується на впровадженні спеціалізованих програмних засобів, призначених для повного циклу роботи з фінансовою інформацією: від первинного збору та безпечного зберігання до глибокої аналітичної обробки даних. Сучасна архітектура таких систем відзначається мультиплатформенністю, що дозволяє реалізовувати їх у формі кросплатформених мобільних додатків, адаптивних веб застосунків або комплексних серверних рішень. Особливу роль у цьому контексті відіграє розробка серверних API (Application Programming Interface), які виступають сполучною ланкою між базою даних та клієнтськими інтерфейсами, забезпечуючи уніфікований доступ до логіки системи з будь-якого пристрою користувача.

Інформаційна система управління особистим бюджетом повинна відповідати вимогам цілісності та оперативності, що досягається шляхом реалізації наступного функціоналу:

- Систематичний облік доходів та витрат: забезпечення можливості фіксації всіх грошових потоків у режимі реального часу з підтримкою різних валют та типів рахунків (готівкові, карткові, депозитні).Класифікацію фінансових операцій за категоріями;

- Інтелектуальна класифікація операцій: групування транзакцій за категоріями та підкатегоріями, що дозволяє ідентифікувати основні статі витрат та виявляти потенційні резерви для економії.Зберігання історії фінансових операцій;

- Генерація аналітичної звітності: побудова динамічних графіків, діаграм та статистичних зведень, які відображають структуру бюджету за певний період та допомагають оцінити фінансову ефективність користувача.;

- Ретроспективний аналіз та зберігання історії: ведення детального архіву фінансових операцій, що необхідно для порівняння показників у довгостроковій перспективі та прогнозування майбутніх витрат.

- Комплексна безпека та конфіденційність: застосування сучасних протоколів шифрування даних, двофакторної автентифікації та розмежування прав доступу для захисту чутливої фінансової інформації від несанкціонованого втручання.

Використання API-архітектури дозволяє відокремити серверну логіку від клієнтської частини, що підвищує гнучкість, масштабованість та зручність подальшого розвитку системи.

Розглядаючи автоматизацію особистого бюджету як цілісну інформаційну систему, необхідно акцентувати увагу на механізмах обробки даних, що відбуваються на рівні внутрішньої логіки програмного забезпечення. Ефективне функціонування такої системи залежить від якості

побудови моделі даних, де кожна фінансова транзакція розглядається як багатовимірний об'єкт, що містить часову мітку, атрибути валюти, прив'язку до конкретного рахунку та метадані категорії. Саме такий підхід дозволяє системі здійснювати складну агрегацію показників у розрізі довільних часових інтервалів. Застосування хмарних технологій у архітектурі системи забезпечує не лише надійність зберігання інформації через механізми реплікації баз даних, а й гарантує консистентність даних при одночасному доступі з декількох терміналів користувача. Це нівелює проблему розсинхронізації залишків на рахунках, що часто виникає при використанні локальних методів обліку.

Важливим вектором розширення функціональних можливостей є інтеграція зовнішніх фінансових протоколів, які дозволяють автоматизувати процес отримання даних безпосередньо від банківських установ. Це перетворює систему з пасивного інструменту фіксації на активне середовище фінансового моніторингу. Окрім стандартної класифікації, сучасні інформаційні системи впроваджують елементи предиктивної аналітики, що базуються на статистичних методах обробки часових рядів. Такий інструментарій дає змогу виявляти сезонні коливання витрат та завчасно інформувати користувача про потенційні касові розриви або необхідність акумуляції коштів для майбутніх обов'язкових платежів. Таким чином, система трансформується у повноцінну модель підтримки прийняття рішень, де візуалізація даних слугує не лише для констатації фактів, а й для стратегічного моделювання різних сценаріїв фінансової поведінки.

Окрему увагу в структурі інформаційної системи слід приділити рівню інтерфейсу користувача, який має забезпечувати мінімальні часові витрати на введення даних при максимальній інформативності виводу. Використання сучасних фреймворків для розробки клієнтських додатків дозволяє впроваджувати складні інтерактивні елементи керування, що полегшують сприйняття великих обсягів статистичної інформації. Безпека при цьому виступає не просто додатковою опцією, а фундаментальною властивістю

архітектури, де кожний запит до серверного API проходить сувору перевірку на авторизацію. Впровадження технологій токенізації сесій та шифрування на рівні бази даних гарантує, що навіть у разі перехоплення трафіку конфіденційна інформація залишиться недоступною для третіх осіб. В сукупності ці технічні рішення створюють стійку та масштабовану платформу, здатну адаптуватися до зростаючих потреб користувача в умовах глобальної цифровізації фінансових послуг. де візуалізація даних слугує не лише для констатації фактів, а й для стратегічного моделювання різних сценаріїв фінансової поведінки.

1.3 Аналіз існуючих програмних рішень для управління особистими фінансами

Аналіз існуючих рішень на ринку України демонструє чітку тенденцію до монополізації функцій управління капіталом всередині екосистем великих банківських установ. Попри високу технологічність таких сервісів, як monobank чи Приват24, користувач опиняється в межах «закритого саду», де аналітичні можливості обмежені операціями лише всередині однієї фінансової установи. Це створює проблему фрагментації даних для осіб, які використовують рахунки в кількох банках або оперують значними обсягами готівкових коштів та цифрових активів. Відсутність механізмів безшовної консолідації інформації призводить до того, що загальна картина фінансового стану залишається неповною, а процес ручного перенесення даних в інші системи нівелює всі переваги автоматизації.

Більшість незалежних фінансових застосунків, що позиціонують себе як універсальні інструменти, часто стикаються з проблемою складної синхронізації з українськими банками через відсутність єдиних стандартів відкритого банкінгу. Це змушує розробників використовувати проміжні сервіси або методи парсингу сповіщень, що негативно впливає на стабільність роботи системи та ставить під загрозу конфіденційність транзакцій. Крім того, значна частина комерційних рішень орієнтована на платну підписку, що створює додатковий фінансовий бар'єр для

користувачів, які лише починають шлях до фінансової грамотності. Обмеженість налаштувань аналітичних звітів у таких додатках часто не дозволяє адаптувати систему під специфічні потреби користувача, наприклад, для ведення роздільного обліку особистих та сімейних фінансів або інтеграції з бізнес-аналітикою для самозайнятих осіб.

Важливим технологічним викликом залишається також питання кросплатформеності та володіння даними. Більшість популярних рішень існують виключно у форматі мобільних додатків, що суттєво обмежує можливості для глибокої роботи з фінансами на стаціонарних комп'ютерах, де зручніше проводити складне планування та довгострокове прогнозування. Закритість серверної архітектури таких систем означає, що користувач фактично не є власником своєї фінансової історії у форматі, придатному для легкого експорту або міграції. Це зумовлює актуальність розробки відкритих інформаційних систем, які базуються на принципах модульності та надають розширені можливості для розробки власних аналітичних модулів через використання публічних інтерфейсів програмування. Такий підхід дозволяє трансформувати інструмент обліку в гнучку платформу, що здатна еволюціонувати разом із фінансовими потребами користувача, забезпечуючи при цьому максимальну прозорість та безпеку опрацювання інформації. структури бюджету та не передбачає відкритого API для повноцінної інтеграції з зовнішніми системами.

Мобільний застосунок **monobank** надає користувачам можливість перегляду історії транзакцій, автоматичної категоризації витрат, формування статистики доходів і витрат, а також візуалізації фінансових даних у вигляді графіків. До переваг даного рішення належать зручний інтерфейс, швидкий доступ до аналітичної інформації та висока інтеграція з банківськими сервісами. Водночас застосунок має обмежені можливості налаштування структури бюджету та не передбачає відкритого API для повноцінної інтеграції з зовнішніми системами.

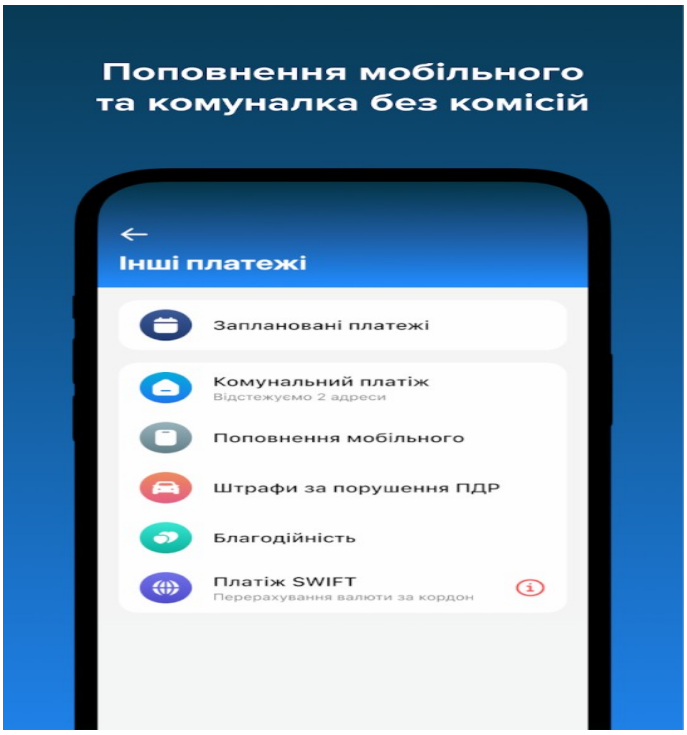

Рис 1.1 Можливість створення платежу у додатку monobank

Додатковим функціональним аспектом monobank є система нагород та ігрових механік, що стимулює користувача частіше взаємодіяти з фінансовим інтерфейсом, проте це не завжди корелює із задачами професійного фінансового планування. Попри наявність інструментів для створення накопичувальних рахунків, таких як сервіс Банка, система автоматизації в межах застосунку зосереджена переважно на аналізі минулих транзакцій, а не на стратегічному прогнозуванні майбутніх періодів. Користувач позбавлений можливості гнучкого редагування автоматично присвоєних категорій для минулих періодів у глобальному масштабі, що іноді призводить до викривлення статистичних даних, якщо алгоритм некоректно розпізнав тип торговельної точки.

Важливим обмеженням є також відсутність інструментарію для врахування готівкових операцій або активів, що знаходяться поза межами банківської системи, у межах єдиного балансу. Це створює ситуацію, за якої застосунок відображає лише частину фінансового життя особи, змушуючи її використовувати додаткові програмні засоби для отримання цілісної

картини. Окрім того, експорт даних для подальшого аналізу у сторонніх програмах часто вимагає ручного вивантаження виписок, оскільки наявне API для фізичних осіб має суттєві обмеження щодо частоти запитів та обсягу переданої інформації. Такий закритий характер екосистеми ускладнює побудову на базі monobank комплексних систем управління капіталом, які потребують синхронізації в реальному часі з іншими фінансовими інструментами чи власними аналітичними платформами користувача.

Додатковим функціональним аспектом monobank є система нагород та ігрових механік, що стимулює користувача частіше взаємодіяти з фінансовим інтерфейсом, проте це не завжди корелює із задачами професійного фінансового планування. Попри наявність інструментів для створення накопичувальних рахунків, таких як сервіс Банка, система автоматизації в межах застосунку зосереджена переважно на аналізі минулих транзакцій, а не на стратегічному прогнозуванні майбутніх періодів. Користувач позбавлений можливості гнучкого редагування автоматично присвоєних категорій для минулих періодів у глобальному масштабі, що іноді призводить до викривлення статистичних даних, якщо алгоритм некоректно розпізнав тип торговельної точки.

Важливим обмеженням є також відсутність інструментарію для врахування готівкових операцій або активів, що знаходяться поза межами банківської системи, у межах єдиного балансу. Це створює ситуацію, за якої застосунок відображає лише частину фінансового життя особи, змушуючи її використовувати додаткові програмні засоби для отримання цілісної картини. Окрім того, експорт даних для подальшого аналізу у сторонніх програмах часто вимагає ручного вивантаження виписок, оскільки наявне API для фізичних осіб має суттєві обмеження щодо частоти запитів та обсягу переданої інформації. Такий закритий характер екосистеми ускладнює побудову на базі monobank комплексних систем управління капіталом, які потребують синхронізації в реальному часі з іншими фінансовими інструментами чи власними аналітичними платформами користувача.

З огляду на це, попри високий рівень технологічності та зручності для повсякденних розрахунків, мобільне рішення від monobank залишається скоріше розширеним банківським інтерфейсом, ніж повноцінною інформаційною системою управління особистими фінансами. Це створює об'єктивну потребу в розробці незалежних рішень, які могли б агрегувати дані з подібних банківських сервісів, але при цьому пропонували б значно ширший інструментарій для персоналізації, глибокої аналітики та довгострокового бюджетування без прив'язки до конкретної фінансової установи.

Система Приват24, будучи найбільш масовим рішенням на вітчизняному ринку, пропонує широкий функціонал для моніторингу витрат, що включає сервіс автоматичного розподілу транзакцій за категоріями та можливість формування детальних виписок. Однією з вагомих переваг цього застосунку є глибока інтеграція з великою кількістю комунальних та державних сервісів, що дозволяє централізувати значну частину обов’язкових платежів в одному інтерфейсі. Проте, попри масштабність системи, вона характеризується певною громіздкістю інтерфейсу та надмірною кількістю маркетингових пропозицій, що іноді ускладнює швидкий доступ до аналітичних розділів. Аналітичний модуль системи фокусується переважно на візуалізації фактичних витрат за минулі періоди, але, як і у випадку з іншими банківськими продуктами, він має обмежений інструментарій для повноцінного планування майбутніх бюджетів з урахуванням складної ієрархії фінансових цілей.

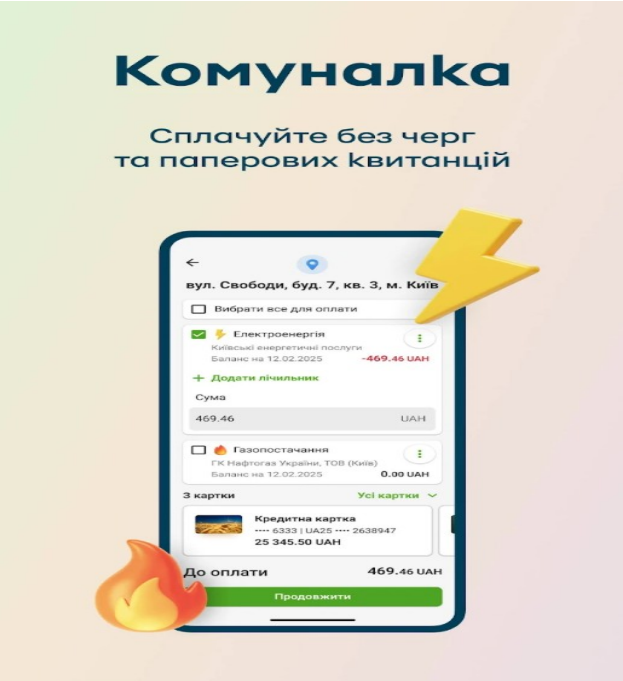

Рис 1.2 Можливість розподілу особистого бюджету у додатку Privat24

Суттєвим недоліком Приват24 у контексті професійного управління персональним капіталом є жорстка прив'язаність до внутрішніх алгоритмів банку, які не завжди коректно ідентифікують специфічні транзакції, а механізми ручного коригування категорій часто є обмеженими або незручними для масового редагування. Окрім того, закритість серверної архітектури та відсутність можливості інтегрувати в систему зовнішні активи, такі як рахунки в інших банках, готівкові заощадження або інвестиції на фондовому ринку, робить цей інструмент вузькоспеціалізованим. Це створює інформаційний вакуум, оскільки користувач бачить лише ту частину своїх фінансів, що проходить через рахунки конкретної установи. Такий підхід суттєво обмежує можливості для проведення комплексного аналізу чистих активів та оцінки реального фінансового стану особи, що є критично важливим для довгострокової фінансової стабільності.

Відсутність публічного API, яке дозволяло б користувачеві безпечно експортувати свої дані до сторонніх аналітичних систем у реальному часі, ще більше підкреслює закритість екосистеми Приват24. Це призводить до

виникнення значних часових витрат при спробах консолідувати звітність вручну, що нівелює головну мету автоматизації — вивільнення інтелектуального ресурсу особистості для прийняття стратегічних рішень. Таким чином, як monobank, так і Приват24, попри свою зручність для операційних задач, не забезпечують користувача повноцінним інструментарієм для системного управління особистим бюджетом як цілісною інформаційною структурою. Це підтверджує необхідність створення незалежного програмного продукту, який базуватиметься на принципах універсальності, відкритості та можливості об'єднання розрізнених фінансових потоків у єдину систему з розширеними аналітичними можливостями.

Спільним недоліком банківських застосунків є їхня закрита архітектура, що не дозволяє користувачам повною мірою контролювати власні фінансові дані або інтегрувати їх з іншими інформаційними системами. Крім того, такі рішення, як правило, обмежуються операціями в межах одного банку та не враховують зовнішні джерела доходів і витрат.

Проведений аналіз свідчить про доцільність розробки окремої API-системи управління особистими фінансами, яка забезпечить незалежність від банківських платформ, гнучкість налаштувань, можливість інтеграції з різними клієнтськими застосунками та повний контроль користувача над фінансовими даними.

Зазначені недоліки обґрунтовують доцільність розробки власної API-системи управління фінансами, яка може бути адаптована під конкретні вимоги та розширена у майбутньому.

Для поглиблення аналізу спільних недоліків існуючих рішень варто звернути увагу на відсутність крос-інституційної аналітики та проблему «цифрової ізоляції» даних. У сучасних умовах, коли фізична особа може одночасно бути клієнтом кількох банківських установ, користуватися послугами необанків та мати активи у децентралізованих фінансових системах, банківські застосунки не здатні забезпечити єдину точку істини.

Кожен окремий додаток формує власну статистику, яка не корелює з даними інших платформ, що призводить до викривлення показників загального капіталу та унеможливлює точне розрахування темпів накопичення чи реальної вартості активів.

Крім того, критичним недоліком є обмеженість інструментів для складного бюджетування та стратегічного планування. Банківські сервіси здебільшого орієнтовані на фіксацію минулих подій, тоді як функціонал для створення сценаріїв майбутнього розвитку фінансового стану — наприклад, моделювання впливу кредитного навантаження на бюджет протягом наступних років або розрахунок досягнення довгострокової фінансової цілі з урахуванням інфляції — практично відсутній. Також спостерігається дефіцит інструментів для спільного управління бюджетом, що є актуальним для домогосподарств, де доходи та витрати розподіляються між кількома особами. Закрита архітектура не дозволяє синхронізувати різні профілі в межах однієї аналітичної моделі, зберігаючи при цьому персоналізований рівень доступу до даних.

Не менш важливою проблемою є відсутність гнучкості в управлінні категоріями та метаданими. Користувач змушений підлаштовуватися під зумовлену банком структуру, яка часто не враховує специфіку особистих бізнес-процесів або складних споживчих звичок. Відсутність можливості створення багаторівневих тегів або складних фільтрів для пошуку транзакцій ускладнює проведення детального аудиту власних фінансів. Впровадження незалежної API-системи дозволить подолати ці бар'єри, надаючи користувачеві можливість самостійно визначати архітектуру своїх даних. Такий підхід забезпечить не лише технічну незалежність від конкретного фінансового оператора, а й створить умови для побудови інтелектуального середовища, де аналітика підлаштовується під потреби людини, а не навпаки, гарантуючи при цьому повну суверенність та конфіденційність приватної фінансової інформації.

1.4 Системи управління фінансами: поняття та класифікація

Поняття системи управління фінансами в сучасній економічній науці розглядається як багатогранна сукупність методів, інструментів та технологічних рішень, спрямованих на реалізацію фінансових цілей суб'єкта через раціональне використання наявних ресурсів. У контексті персонального менеджменту така система виступає інтелектуальною надбудовою над грошовими потоками індивіда, забезпечуючи трансформацію розрізнених даних про транзакції у структуровану інформацію для прийняття управлінських рішень. Фундаментальною характеристикою системи є її здатність до адаптації під мінливі зовнішні економічні умови та внутрішні потреби користувача, що досягається шляхом інтеграції функцій планування, обліку, аналізу та контролю в єдиний інформаційний цикл.

Класифікація систем управління фінансами може здійснюватися за декількома ключовими ознаками, серед яких першочергове значення має ступінь автоматизації процесів. Традиційні або аналогові системи базуються на ручній фіксації даних і мають обмежений аналітичний потенціал, що зумовлює їхню низьку ефективність у сучасному динамічному середовищі. Натомість цифрові системи поділяються на локальні програмні рішення, мобільні застосунки та хмарні платформи. Локальні системи забезпечують високий рівень автономності, проте часто поступаються у зручності доступу та можливостях синхронізації. Хмарні та мобільні рішення, що домінують на сучасному ринку, пропонують кросплатформеність та інтеграцію з банківськими установами, що дозволяє досягти максимальної оперативності в управлінні капіталом.

Іншим важливим критерієм класифікації є функціональна спрямованість систем, де виділяють спеціалізовані та комплексні рішення. Спеціалізовані системи фокусуються на конкретних аспектах, таких як облік щоденних витрат або управління інвестиційним портфелем, тоді як комплексні системи, відомі як Personal Finance Management (PFM)

платформи, намагаються охопити всі сфери фінансового життя користувача. Окрему категорію становлять банківські інтегровані системи, які, попри високу зручність, зазвичай обмежені екосистемою однієї фінансової установи. На противагу їм, незалежні агрегатори дозволяють консолідувати дані з різних джерел, забезпечуючи цілісний підхід до оцінки власного добробуту.

Також системи управління фінансами доцільно розрізняти за архітектурними особливостями побудови. Закриті системи характеризуються монолітною структурою, де користувач має доступ лише до зумовленого розробником функціоналу без можливості розширення чи автоматизованого експорту даних. Відкриті системи, що базуються на API-архітектурі, представляють собою найбільш прогресивний тип програмного забезпечення. Вони дозволяють відокремити інтерфейс користувача від логіки обробки даних, забезпечуючи гнучкість налаштувань та можливість інтеграції з іншими цифровими сервісами. Саме такі системи відповідають вимогам сучасного інформаційного суспільства, оскільки надають користувачеві повний суверенітет над його фінансовою інформацією та створюють умови для побудови персоналізованої стратегії фінансової незалежності.

Додатковим аспектом класифікації систем управління фінансами є їхній поділ за методом обробки вхідної інформації, де виокремлюють детерміновані та адаптивні системи. Детерміновані системи працюють за чітко заданими алгоритмами, де кожна транзакція потребує ручного підтвердження або належить до заздалегідь визначеної категорії без можливості самостійного навчання програми. Натомість адаптивні системи, що базуються на технологіях великих даних, здатні самостійно ідентифікувати аномалії у споживчій поведінці та пропонувати коригування бюджетного плану в режимі реального часу. Це дозволяє системі еволюціонувати від простого цифрового журналу операцій до інтелектуального радника, який оцінює фінансові ризики та прогнозує

ймовірність досягнення стратегічних цілей при збереженні поточних темпів витрат.

Важливою складовою сучасної класифікації є також аналіз систем за рівнем інтеграції в глобальну фінансову інфраструктуру. Окрему групу складають автономні рішення, які орієнтовані на максимальну приватність і працюють без підключення до зовнішніх мереж, що мінімізує ризики кібератак, проте суттєво обмежує оперативність отримання даних. На іншому полюсі знаходяться гібридні системи, які використовують концепцію «відкритого банкінгу» (Open Banking) для безшовної синхронізації з рахунками, податковими кабінетами та інвестиційними платформами. Такі системи забезпечують найвищий рівень автоматизації, оскільки вони автоматично враховують не лише прямі витрати, а й приховані комісії, курсові різниці та нарахування відсотків по депозитах, надаючи користувачеві максимально точне значення його чистої вартості (Net Worth).

Розглядаючи архітектурні принципи, варто виділити системи з централізованим та децентралізованим зберіганням даних. Централізовані системи пропонують високу швидкість обробки запитів та зручність відновлення доступу, але створюють єдину точку відмови та потенційний ризик витоку конфіденційної інформації. Децентралізовані рішення, що часто використовують технологію розподілених реєстрів, забезпечують незмінність історії транзакцій та виключають можливість стороннього втручання в дані. Впровадження таких підходів у персональні фінанси дозволяє створити прозорий та верифікований аудит власної діяльності, що стає основою для побудови довгострокових фінансових стратегій у нестабільному економічному середовищі.

Таким чином, сучасна класифікація відображає перехід від простих облікових інструментів до комплексних інформаційних екосистем, які стають невід'ємним елементом цифрового портрета сучасної людини.

РОЗДІЛ 2. СИСТЕМНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ

2.1 Об’єкт дослідження

Об’єктом дослідження у даній дипломній роботі є процес управління особистими фінансами користувача в умовах використання автоматизованих інформаційних систем

Зазначений процес охоплює сукупність дій, пов’язаних з обліком доходів і витрат, класифікацією фінансових операцій, зберіганням фінансових даних, їх обробкою та аналізом з метою формування інформації для прийняття фінансових рішень. У сучасному цифровому середовищі управління особистими фінансами все частіше здійснюється з використанням програмних засобів, що потребує системного підходу до організації та автоматизації відповідних процесів.

У межах даної роботи об’єкт дослідження розглядається як складна інформаційна система, що включає користувача, фінансові дані, програмне забезпечення та канали взаємодії між ними. Особлива увага приділяється серверній частині системи, яка забезпечує централізоване зберігання та обробку фінансової інформації.

У межах даної роботи об’єкт дослідження розглядається як складна інформаційна система, що включає користувача, фінансові дані, програмне забезпечення та канали взаємодії між ними. Особлива увага приділяється серверній частині системи, яка забезпечує централізоване зберігання та обробку фінансової інформації.

Важливим аспектом досліджуваного об’єкта є механізм семантичного зв’язку між різнорідними фінансовими потоками. Це передбачає вивчення того, як система обробляє метадані категорій, часові мітки та валютні курси для створення уніфікованої звітності. Об’єкт дослідження охоплює також процеси обробки виняткових ситуацій та конфліктів даних, що виникають при паралельному доступі з декількох пристроїв. Це вимагає детального

аналізу методів синхронізації станів, що реалізуються через API, та способів оптимізації навантаження на серверну частину при роботі з великими масивами історичних даних. Таким чином, фокус дослідження зміщується з візуальної оболонки застосунку на внутрішню інженерну структуру, яка визначає надійність та масштабованість системи.

Окрім технічних характеристик, об’єкт дослідження включає інформаційну модель безпеки, яка базується на принципі мінімальних привілеїв. Це означає, що процес управління фінансами розглядається через призму захищеності кожного окремого ресурсу системи, де bearer-токени виступають не лише засобом ідентифікації, а й інструментом розмежування прав доступу до конкретних фінансових рахунків чи звітів. Дослідження цих процесів дозволяє виявити критичні точки в архітектурі, де автоматизація може стикнутися з обмеженнями продуктивності або ризиками несанкціонованого доступу. У підсумку, об’єкт дослідження постає як комплексна цифрова екосистема, функціонування якої безпосередньо впливає на точність фінансового аналізу та якість стратегічного планування життєдіяльності особистості в умовах сучасної інформаційної економіки.

Особлива увага в описі об’єкта приділяється методології побудови абстракцій на рівні серверної логіки, де фінансові операції розглядаються незалежно від їхнього фізичного походження. Це дозволяє об’єкту дослідження охоплювати процеси нормалізації даних, що надходять із різних джерел, приводячи їх до єдиного стандарту зберігання та обробки. Такий підхід забезпечує можливість масштабування системи без необхідності докорінної зміни її архітектури при додаванні нових типів активів або валют. Об’єкт дослідження також включає в себе механізми забезпечення високої доступності системи, що реалізуються через оптимізацію запитів до бази даних та використання ефективних алгоритмів серіалізації інформації у форматі JSON для передачі через API.

Важливим компонентом об’єкта дослідження стає архітектурний патерн розділення відповідальності між фронтенд- та бекенд-частинами, що

дозволяє розглядати систему як незалежний сервіс, здатний обслуговувати різнорідні клієнтські застосунки одночасно. Це вимагає детального вивчення процесів управління станами (state management) та забезпечення консистентності даних в умовах асинхронної взаємодії. Дослідження цих аспектів дозволяє сформувати комплексні вимоги до стійкості інформаційної системи, що безпосередньо впливає на точність відображення поточного фінансового балансу. У такому розширеному розумінні об’єкт дослідження постає як цілісна технологічна платформа, що забезпечує автоматизовану підтримку фінансової діяльності особистості, гарантуючи при цьому високу швидкість відгуку та стійкість до критичних навантажень у моменти пікової активності користувачів.

Кожен API-ендпоїнт представляє окрему точку доступу до ресурсів системи та реалізує визначені операції відповідно до принципів REST-архітектури. Для забезпечення контролю доступу до фінансових даних у системі застосовується механізм автентифікації та авторизації користувачів на основі bearer-токенів. Використання токенів дозволяє ідентифікувати користувача під час кожного запиту до API та забезпечити безпечну взаємодію між клієнтськими застосунками та серверною частиною системи.

Таким чином, дослідження об’єкта спрямоване на визначення особливостей автоматизованого управління особистими фінансами та виявлення вимог до інформаційної системи, що реалізує відповідні функції.

Досліджуваний об’єкт також включає процеси формування аналітичних показників, які базуються на накопичених фінансових даних користувача. До таких показників належать динаміка доходів і витрат у часовому розрізі, структура витрат за категоріями, аналіз фінансових звичок та виявлення повторюваних шаблонів поведінки. Формування подібної аналітики є важливим елементом процесу управління особистими фінансами, оскільки дозволяє користувачеві оцінювати ефективність власних фінансових рішень та коригувати їх у довгостроковій перспективі.

Окрему роль у межах об’єкта дослідження відіграють механізми збереження цілісності та достовірності фінансової інформації. Це передбачає застосування транзакційних операцій на рівні бази даних, контроль коректності введених значень, а також ведення журналів змін фінансових записів. Такий підхід дозволяє мінімізувати ризики втрати або спотворення даних у разі збоїв програмного забезпечення, помилок користувача або нестабільної роботи мережевих з’єднань.

Об’єкт дослідження також охоплює питання адаптації системи до індивідуальних потреб користувача. Це проявляється у можливості налаштування фінансових категорій, валют, періодів звітності та форматів представлення інформації. Гнучкість подібних налаштувань є необхідною умовою ефективного використання автоматизованої системи управління фінансами, оскільки фінансові моделі поведінки різних користувачів можуть суттєво відрізнятися залежно від рівня доходів, стилю життя та фінансових цілей.

Важливою складовою об’єкта дослідження є процес інтеграції з зовнішніми сервісами та джерелами даних. Це можуть бути сервіси отримання актуальних валютних курсів, платіжні платформи або банківські API. Наявність таких інтеграцій розширює функціональні можливості системи та дозволяє автоматизувати введення фінансових операцій, зменшуючи залежність від ручного введення даних і підвищуючи загальну точність обліку.

З точки зору програмної інженерії об’єкт дослідження розглядається як багаторівнева система, у якій кожен рівень виконує чітко визначені функції. Рівень доступу до даних відповідає за взаємодію з базою даних, рівень бізнес-логіки — за обробку фінансових операцій та застосування правил обліку, а рівень API — за комунікацію з клієнтськими застосунками. Подібна структуризація дозволяє підвищити підтримуваність коду, спростити тестування окремих компонентів та забезпечити можливість подальшого розвитку системи.

Окрім цього, об’єкт дослідження включає аспекти продуктивності та оптимізації використання ресурсів серверної інфраструктури. Аналізу підлягають методи кешування, індексації даних та оптимізації запитів, що мають безпосередній вплив на швидкість обробки фінансової інформації. Забезпечення стабільної роботи системи при зростанні кількості користувачів та обсягу збережених даних є одним із ключових викликів у процесі автоматизованого управління особистими фінансами.

2.2 Предмет дослідження

Предметом дослідження у даній кваліфікаційній роботі є методи, моделі та програмні засоби автоматизації управління особистими фінансами, що базуються на сучасній сервісно-орієнтованій архітектурі RESTful API.

У межах роботи основний фокус зосереджено на інженерних підходах до проектування та реалізації серверної частини інформаційної системи, зокрема на розробці ресурсної моделі даних, що відображає фінансові сутності. Це включає організацію REST API, визначення детальної структури та логіки роботи API-ендпоїнтів для операцій CRUD, а також ефективне використання механізмів обробки HTTP-запитів, включаючи валідацію вхідних даних та реалізацію Data Transfer Objects (DTO). Окремо розглядаються способи надійного зберігання та обробки фінансових даних у реляційній базі даних MySQL, де особлива увага приділяється забезпеченню транзакційної цілісності (ACID) при обліку доходів і витрат.

Важливе місце у предметі дослідження займають питання комплексного забезпечення безпеки доступу до конфіденційної фінансової інформації. Зокрема, аналізуються та впроваджуються механізми автентифікації та авторизації користувачів із використанням Bearer-токенів (наприклад, за допомогою Laravel Sanctum), управління сесіями доступу, а також реалізація політик доступу та обмежень до ресурсів системи. Додатково досліджується питання шифрування критичних даних, таких як паролі, що зберігаються у базі даних, для посилення захисту..

Крім того, досліджується застосування високопродуктивного фреймворку Laravel як основного інструменту для реалізації бізнес-логіки системи. Вивчаються методи ефективної взаємодії з базою даних за допомогою Eloquent ORM, включаючи визначення складних відношень між фінансовими сутностями та використання міграцій для управління схемою бази даних. Також предмет дослідження охоплює підходи до автоматизованого тестування API-ендпоїнтів для верифікації коректності обробки фінансових операцій та забезпечення підтримуваності й масштабованості програмного коду.

У межах предмета дослідження також розглядаються підходи до формалізації бізнес-логіки управління фінансовими операціями на рівні серверної частини. Це включає визначення правил обробки доходів і витрат, алгоритмів агрегації даних для формування зведених показників, а також методів обчислення фінансових балансів у різних часових інтервалах. Особлива увага приділяється розмежуванню відповідальності між контролерами, сервісними класами та репозиторіями даних, що сприяє зменшенню зв’язаності компонентів і підвищенню якості програмної реалізації.

Предмет дослідження охоплює питання проєктування структури бази даних з урахуванням оптимізації запитів і масштабування системи. Аналізуються методи нормалізації фінансових таблиць, використання індексів для прискорення доступу до великих обсягів історичних даних, а також підходи до збереження агрегованих показників з метою зменшення навантаження на сервер при формуванні аналітичних звітів. Такі рішення мають суттєвий вплив на продуктивність системи в умовах зростання кількості користувачів та фінансових операцій.

Окремим аспектом предмета дослідження є управління помилками та винятковими ситуаціями при роботі з API. Розглядаються методи централізованої обробки помилок, формування уніфікованих HTTP-відповідей та використання стандартизованих кодів статусу для

інформування клієнтських застосунків про результати виконання запитів. Це забезпечує прогнозовану поведінку системи та спрощує інтеграцію API з зовнішніми сервісами й клієнтськими інтерфейсами.

У межах предмета дослідження також аналізуються підходи до версіонування REST API, що дозволяють забезпечити зворотну сумісність при подальшому розвитку системи. Це включає визначення правил зміни контрактів API, структури URL-адрес та форматів відповідей, а також застосування документації (наприклад, OpenAPI) для формалізованого опису функціональності сервісу. Наявність чітко визначених контрактів є важливою умовою стабільної взаємодії між серверною частиною та клієнтськими застосунками.

Значну увагу в предметі дослідження приділено питанням продуктивності та оптимізації серверних процесів. Розглядаються методи кешування результатів запитів, використання черг для асинхронної обробки фінансових операцій та мінімізації часу відповіді API. Застосування таких підходів дозволяє підвищити масштабованість системи та забезпечити стабільну роботу навіть у періоди пікових навантажень.

Крім того, предмет дослідження включає аспекти логування та моніторингу роботи API-системи. Аналізуються способи фіксації ключових подій, пов’язаних з фінансовими операціями та спробами доступу до системи, а також методи збору технічних метрик для оцінки стану серверної інфраструктури. Це створює передумови для оперативного виявлення помилок, підвищення надійності системи та забезпечення її подальшого розвитку.

Таким чином, предмет дослідження охоплює комплекс технічних, архітектурних та методичних аспектів створення сучасної API-системи управління особистими фінансами, що забезпечує автоматизацію обліку доходів і витрат, надійну та безпечну обробку даних, а також готовність до інтеграції з різними клієнтськими застосунками.

2.3 Функціональні та нефункціональні вимоги до системи

Функціональні вимоги до розроблюваної системи управління особистими фінансами визначають перелік можливостей, які повинні бути реалізовані для забезпечення повноцінної автоматизації обліку та аналізу фінансових даних. Система повинна підтримувати реєстрацію та автентифікацію користувачів із використанням механізму bearer-токенів, що дозволяє ідентифікувати користувача під час кожного звернення до API. Доступ до ресурсів системи має здійснюватися відповідно до прав користувача з урахуванням механізмів авторизації.

Розроблювана система повинна забезпечувати можливість створення, перегляду, редагування та видалення фінансових операцій, що включають доходи та витрати користувача. Фінансові операції мають класифікуватися за категоріями, що дозволяє здійснювати подальший аналіз структури бюджету. Усі фінансові дані повинні зберігатися у реляційній базі даних MySQL з дотриманням вимог цілісності та узгодженості даних.

Система повинна надавати доступ до агрегованої інформації щодо доходів і витрат за обраний період часу, а також формувати аналічні дані для подальшої обробки та візуалізації у клієнтських веб- або мобільних застосунках. Взаємодія клієнтської частини з серверною повинна здійснюватися через REST API з використанням стандартизованих HTTP-запитів і відповідей у форматі JSON. У випадку виникнення помилок система повинна повертати коректні та інформативні повідомлення.

Нефункціональні вимоги визначають загальні характеристики якості та експлуатаційні властивості системи. Розроблювана API-система повинна забезпечувати високий рівень безпеки фінансових даних користувачів шляхом використання сучасних механізмів автентифікації та контролю доступу до API-ендпоінтів. Надійність системи має забезпечуватися коректною обробкою некоректних або неповних запитів, а також збереженням цілісності даних у процесі виконання операцій.

Продуктивність системи повинна відповідати вимогам сучасних

інформаційних систем, забезпечуючи мінімальний час обробки запитів та стабільну роботу при збільшенні кількості користувачів. Архітектура системи повинна бути масштабованою та дозволяти розширення функціональних можливостей без суттєвих змін у наявній структурі. Важливими характеристиками також є підтримуваність і переносимість системи, що досягається завдяки використанню модульної архітектури, стандартизованих технологій та можливості розгортання у різних серверних середовищах.

До переліку функціональних вимог слід додати механізми управління мультивалютними рахунками, що передбачає необхідність інтеграції функціоналу для автоматичного або ручного оновлення курсів валют. Це дозволяє системі коректно перераховувати загальний баланс активів користувача у базову валюту для формування уніфікованої звітності. Крім того, система повинна підтримувати ієрархічну структуру категорій, надаючи можливість створювати підкатегорії для більш детального аналізу витрат, наприклад, розділення категорії транспорт на пальне, обслуговування та громадський транспорт. Важливою вимогою є реалізація функціоналу для роботи з регулярними платежами, що дозволить автоматично генерувати записи про доходи або витрати через задані проміжки часу, суттєво знижуючи навантаження на користувача при обліку постійних витрат.

Функціональна частина також має включати розширені можливості фільтрації та пошуку транзакцій за декількома параметрами одночасно, такими як часовий діапазон, сума, категорія або коментар до операції. Це забезпечує гнучкість при проведенні фінансових ревізій. Для забезпечення коректної роботи аналітичного модуля система повинна реалізовувати алгоритми перевірки даних на дублювання, що запобігає повторному внесенню однієї і тієї ж операції. Окремою вимогою виступає механізм експорту даних у загальноприйняті формати, такі як CSV або PDF, що дозволяє користувачеві використовувати фінансову інформацію поза межами

системи, наприклад, для подання податкової звітності або глибокого аналізу в зовнішніх табличних процесорах.

У контексті нефункціональних вимог особливу увагу слід приділити забезпеченню ідемпотентності API-запитів, що гарантує відсутність повторного створення транзакцій при випадковому дублюванні запитів від клієнтської частини через нестабільне мережеве з’єднання. Вимога до відмовостійкості системи має бути доповнена механізмами логування помилок на серверній стороні, що дозволить розробнику оперативно ідентифікувати та усувати вразливості або технічні збої без переривання сервісу для кінцевого користувача. Стосовно безпеки, окрім використання токенів, необхідно передбачити валідацію всіх вхідних даних на рівні сервера (Data Validation) для захисту від поширених вразливостей, таких як SQL-ін'єкції або XSS-атаки, що є критично важливим при роботі з конфіденційною фінансовою інформацією.

Вимоги до продуктивності повинні включати оптимізацію індексів у базі даних MySQL для прискорення вибірки великих масивів історичних даних, що забезпечує стабільно низький час відгуку (Latency) навіть при накопиченні десятків тисяч записів одним користувачем. Архітектурна гнучкість системи має передбачати можливість легкої інтеграції з зовнішніми сервісами через вебхуки (webhooks), що в майбутньому дозволить системі автоматично отримувати сповіщення про банківські операції в реальному часі. Таким чином, сформований комплекс вимог створює фундамент для розробки професійного програмного продукту, здатного витримувати навантаження сучасного цифрового ринку та задовольняти потреби найбільш вимогливих користувачів.

2.4 Алгоритм роботи api системи

Для забезпечення високої інтелектуальної складової системи управління особистими фінансами особливу увагу варто приділити алгоритмічному забезпеченню, яке перетворює набір статичних записів на

динамічну модель фінансової поведінки. Ключове місце займають алгоритми агрегації та групування даних, що дозволяють у режимі реального часу трансформувати тисячі окремих транзакцій у структуровані звіти. На відміну від простого підсумовування, ці алгоритми повинні враховувати часові чинники, валютні коливання та ієрархічні зв'язки між категоріями. Реалізація методів нормалізації даних забезпечує приведення різнорідних фінансових потоків до єдиного знаменника, що є критично важливим для коректної роботи аналітичного модуля та уникнення логічних помилок при розрахунку загального капіталу.

Окремим важливим компонентом є алгоритми прогнозування фінансових станів, що базуються на аналізі часових рядів та методів статистичної екстраполяції. Використовуючи історичні дані про доходи та витрати, система здатна виявляти сезонні цикли та регулярні патерни споживання, що дозволяє будувати імовірнісні моделі бюджету на майбутні періоди. Це дає змогу реалізувати функціонал раннього попередження про можливий дефіцит коштів або перевищення встановлених лімітів ще до моменту здійснення фактичної транзакції. Такий предиктивний підхід кардинально змінює роль інформаційної системи, перетворюючи її з пасивного реєстратора на активний інструмент підтримки прийняття фінансових рішень.

Також у системі мають бути імплементовані алгоритми автоматичної категоризації на основі текстового аналізу описів транзакцій та ідентифікаторів торгових точок. Використання методів порівняння рядків та евристичних правил дозволяє системі з високою точністю визначати приналежність нових операцій до існуючих категорій без втручання користувача. Окрім цього, важливу роль відіграють алгоритми валідації та контролю цілісності, що працюють на рівні бізнес-логіки сервера. Вони забезпечують перевірку транзакцій на несумісність (наприклад, від’ємні суми там, де це не передбачено логікою) та гарантують ідемпотентність операцій, що запобігає дублюванню даних при повторних запитах. Впровадження

такого комплексу алгоритмів забезпечує високу автономність системи та достовірність наданої користувачеві аналітики.

Додатково до базових аналітичних та предиктивних алгоритмів, у межах інтелектуального ядра системи розглядаються методи адаптивного навчання на основі поведінки конкретного користувача. Такі алгоритми дозволяють поступово коригувати правила обробки фінансових даних, спираючись на історію взаємодії користувача з системою. Наприклад, часті ручні зміни категорій або регулярне ігнорування певних рекомендацій враховуються при подальшій побудові моделей аналізу. Це забезпечує персоналізацію аналітики та підвищує релевантність прогнозів, що є важливою ознакою інтелектуальної інформаційної системи.

Важливим аспектом алгоритмічного забезпечення є обробка неповних, зашумлених або суперечливих даних, які можуть виникати внаслідок помилок введення, асинхронної синхронізації або інтеграції з зовнішніми сервісами. Для цього застосовуються методи фільтрації та згладжування даних, що дозволяють мінімізувати вплив аномальних значень на результати фінансового аналізу. Такі алгоритми забезпечують стабільність аналітичних показників і запобігають різким коливанням звітних значень, які можуть вводити користувача в оману.

Окрему роль відіграють алгоритми виявлення аномалій у фінансовій поведінці, які аналізують відхилення від типових патернів витрат або доходів. На основі порівняння поточних транзакцій з історичними профілями система здатна ідентифікувати нетипові операції, що можуть свідчити як про зміну фінансової поведінки користувача, так і про потенційні помилки або несанкціоновані дії. Реалізація таких алгоритмів підвищує рівень фінансової обізнаності користувача та сприяє своєчасному реагуванню на критичні ситуації.

У межах серверної логіки значна увага приділяється оптимізації алгоритмів обробки великих обсягів даних. Застосування інкрементальних обчислень дозволяє уникнути повного перерахунку аналітичних показників

при кожній новій транзакції, що істотно знижує навантаження на сервер та скорочує час відповіді API. Такий підхід є особливо актуальним для довгострокових користувачів, фінансова історія яких може містити десятки тисяч записів.

Крім того, алгоритмічне забезпечення включає механізми формування сценарних моделей фінансової поведінки. Це дозволяє аналізувати альтернативні варіанти розвитку фінансового стану залежно від змін ключових параметрів, таких як рівень доходів, регулярні витрати або фінансові цілі. Подібні алгоритми створюють основу для реалізації функцій стратегічного планування та оцінки ризиків, що суттєво розширює функціональні можливості системи управління особистими фінансами.

У сукупності використання описаних алгоритмів забезпечує трансформацію системи з класичного інструменту обліку у повноцінну інтелектуальну платформу аналізу фінансової поведінки. Алгоритмічна складова стає ключовим фактором, що визначає точність розрахунків, глибину аналітики та здатність системи адаптуватися до змін фінансового середовища та індивідуальних потреб користувача.

Важливим напрямом алгоритмічного забезпечення є реалізація механізмів контекстного аналізу фінансових операцій. Такі алгоритми враховують не лише числові параметри транзакцій, але й супровідні метадані, зокрема час здійснення операції, географічні ознаки та історію взаємодії користувача з конкретними торговими точками. Контекстний підхід дозволяє більш точно інтерпретувати фінансові події та формувати узагальнені висновки щодо змін у фінансовій поведінці користувача, що підвищує інформативність аналітичних звітів.

Ще одним важливим аспектом є алгоритми пріоритизації фінансової інформації, які визначають релевантність окремих показників залежно від поточного стану користувача та заданих фінансових цілей. На основі таких алгоритмів система здатна динамічно змінювати акценти в аналітичних панелях, висуваючи на перший план ті показники, що мають найбільше

практичне значення у конкретний момент часу. Це зменшує когнітивне навантаження на користувача та сприяє більш усвідомленому прийняттю фінансових рішень.

Крім того, алгоритмічне забезпечення системи включає механізми самодіагностики та оцінки якості обробки фінансових даних. За допомогою внутрішніх метрик система аналізує точність прогнозів, стабільність аналітичних моделей та частоту виникнення виняткових ситуацій. Отримані результати можуть використовуватися для автоматичного коригування параметрів алгоритмів або для інформування користувача про можливі обмеження в точності аналітики. Такий підхід забезпечує прозорість роботи системи та підвищує рівень довіри до результатів фінансового аналізу.

РОЗДІЛ 3. МЕТОДИ ТА ЗАСОБИ ВИРІШЕННЯ ПРОБЛЕМИ

3.1 Методи розробки

Розробка системи управління особистими фінансами здійснювалася із застосуванням методів об’єктно-орієнтованого програмування та принципів Agile-розробки. Основним підходом до побудови системи було поетапне проектування ітеративними циклами, що дозволяло поетапно реалізовувати функціонал, тестувати його та вносити необхідні корективи на ранніх стадіях розробки.

На першому етапі проводилося моделювання бізнес-процесів та формування вимог до системи. Для цього використовувалися UML-діаграми класів, діаграми послідовності та діаграми прецедентів, що дозволяє наочно відобразити взаємодію користувача з системою та структуру компонентів.

Другий етап полягав у проєктуванні REST API-ендпоїнтів та організації серверної частини на базі фреймворку Laravel. Було реалізовано моделі для роботи з базою даних MySQL, маршрути API-ендпоїнтів, контролери для обробки запитів та сервіси для бізнес-логіки. Для забезпечення безпеки користувачів застосовувався механізм автентифікації та авторизації через bearer-токени, що гарантує контроль доступу до фінансових даних. На цьому етапі можна розмістити друге фото, що ілюструє структуру бази даних або механізм авторизації через токени.

Завершальний етап включав тестування реалізованої системи, включаючи функціональне тестування API-ендпоїнтів, перевірку обробки фінансових операцій та забезпечення стабільності роботи при навантаженні. Використання зазначеного методу розробки забезпечує модульність, масштабованість та гнучкість системи, що дозволяє ефективно вирішити проблему автоматизації управління особистими фінансами шляхом створення надійної та безпечної API-системи.

Для реалізації системи управління особистими фінансами були обрані сучасні та надійні засоби розробки, що забезпечують масштабованість,

безпеку та простоту подальшого супроводу програмного продукту. Основним фреймворком для серверної частини системи став Laravel, який реалізує архітектурний шаблон MVC (Model-View-Controller). Використання Laravel дозволило організувати логіку програми в модулі, спростити маршрутизацію API-ендпоїнтів, забезпечити інтеграцію з базою даних та реалізувати механізми автентифікації і авторизації користувачів.

Для зберігання фінансових даних використовується реляційна база даних MySQL, що забезпечує надійне зберігання інформації, підтримку транзакцій та цілісність даних. Модель взаємодії з базою реалізована через ORM Eloquent, що дозволяє працювати з даними як з об’єктами, спрощуючи написання коду та мінімізуючи ймовірність помилок.

Взаємодія клієнтських застосунків із серверною частиною здійснюється через REST API, що забезпечує стандартизовану передачу даних у форматі JSON. Кожен API-ендпоїнт реалізує конкретну функцію системи, наприклад, додавання або редагування транзакцій, управління категоріями доходів та витрат, формування аналітичних звітів. Для контролю доступу до ресурсів застосовується автентифікація через bearer-токени, що гарантує безпеку фінансових даних та обмежує доступ лише авторизованим користувачам.

Для розробки використовувалося сучасне середовище розробки PHP, що дозволяє інтегрувати Laravel з іншими технологіями та забезпечує високу продуктивність при обробці запитів API. Тестування функціональності здійснювалося за допомогою інструментів для перевірки REST API та власних скриптів для перевірки коректності обробки фінансових операцій.

Завдяки обраному набору засобів розробки система забезпечує високу надійність, безпеку, зручність використання та готовність до подальшого розвитку та інтеграції з веб- та мобільними клієнтськими застосунками.

У процесі розробки системи управління особистими фінансами особлива увага приділялася принципам об’єктно-орієнтованого програмування (ООП), таким як інкапсуляція, наслідування та поліморфізм.

Використання цих принципів дозволило створити чітку структуру програмного коду, у якій кожен клас відповідає за конкретну функціональну область системи. Наприклад, окремі класи моделей відповідають за роботу з користувачами, транзакціями та категоріями, тоді як контролери реалізують логіку обробки запитів і взаємодії між клієнтською та серверною частинами.

Методологія Agile-розробки була обрана з огляду на необхідність гнучкого реагування на зміни вимог та можливість поступового вдосконалення функціональності системи. Робота над проєктом здійснювалася короткими ітераціями, у межах яких реалізовувався окремий набір функцій, після чого проводилося тестування та аналіз отриманих результатів. Такий підхід дозволив мінімізувати ризики на пізніх етапах розробки та своєчасно виявляти недоліки в архітектурі або бізнес-логіці системи.

Важливою складовою методів розробки стало дотримання принципів чистої архітектури та розділення відповідальності між компонентами системи. Бізнес-логіка була винесена в окремі сервіси, що дозволяє уникнути перевантаження контролерів та спрощує подальше тестування й модифікацію коду. Такий підхід забезпечує кращу підтримуваність системи та зменшує залежність між окремими модулями.

Під час проєктування REST API використовувався підхід «contract-first», при якому спочатку визначалися структура ендпоїнтів, формат вхідних і вихідних даних, а також можливі коди відповідей сервера. Це дозволило сформувати чіткий інтерфейс взаємодії між серверною та клієнтською частинами ще до початку реалізації, що значно спростило подальшу інтеграцію та тестування системи.

Окрему увагу було приділено забезпеченню якості програмного продукту. У процесі розробки застосовувалися методи модульного та інтеграційного тестування, які дозволили перевірити коректність роботи окремих компонентів та їх взаємодії між собою. Тестування API-ендпоїнтів проводилося шляхом надсилання запитів з різними наборами даних, у тому

числі граничних та помилкових значень, що дало змогу оцінити стійкість системи до некоректного вводу.

Для контролю змін у програмному коді використовувалися засоби керування версіями, що дозволяло відстежувати історію змін, повертатися до попередніх версій та організовувати колективну роботу над проєктом. Це особливо важливо при реалізації складних функціональних можливостей, таких як формування фінансових звітів або реалізація механізмів авторизації та доступу до ресурсів.

У рамках методів розробки також враховувалися вимоги до продуктивності та масштабованості системи. Архітектура серверної частини була спроєктована таким чином, щоб забезпечити ефективну обробку великої кількості запитів без суттєвого зниження швидкодії. Використання REST API та stateless-підходу дозволяє масштабувати систему горизонтально шляхом додавання нових серверних інстансів у разі зростання кількості користувачів.

У результаті застосування описаних методів розробки було створено гнучку та надійну серверну систему, здатну ефективно обробляти фінансові дані та забезпечувати стабільну роботу клієнтських застосунків. Поєднання об’єктно-орієнтованого підходу, Agile-методології та сучасних інструментів розробки дозволило досягти високої якості програмного продукту та створити основу для його подальшого розвитку.

Значну роль у процесі розробки відіграв аналіз потенційних загроз безпеці. На етапі проєктування були визначені основні вектори атак, такі як несанкціонований доступ до фінансових даних або підміна запитів. Для мінімізації ризиків застосовувалися перевірені механізми захисту, зокрема автентифікація на основі токенів, контроль прав доступу та валідація вхідних даних. Це дозволило забезпечити високий рівень захисту персональної інформації користувачів.

Таким чином, обрані методи розробки повністю відповідають поставленим завданням і вимогам до системи управління особистими фінансами. Вони забезпечують не лише реалізацію поточного функціоналу,

але й створюють передумови для розширення системи, інтеграції з зовнішніми сервісами та адаптації до нових потреб користувачів у майбутньому.

3.2 Архітектура та структура системи

Розроблена система управління особистими фінансами побудована за принципом API-орієнтованої архітектури, що забезпечує чітке розділення серверної та клієнтської частин. Серверна частина виконує роль основного ядра системи та відповідає за зберігання даних, обробку запитів і реалізацію бізнес-логіки. Такий підхід дозволяє централізовано керувати фінансовою інформацією користувачів та забезпечує незалежність від конкретних клієнтських застосунків — веб, мобільних або десктопних.

Розроблена система управління особистими фінансами побудована за принципом API-орієнтованої архітектури, що забезпечує чітке розділення серверної та клієнтської частин. Серверна частина виконує роль основного ядра системи та відповідає за зберігання даних, обробку запитів і реалізацію бізнес-логіки. Такий підхід дозволяє централізовано керувати фінансовою інформацією користувачів та забезпечує незалежність від конкретних клієнтських застосунків — веб, мобільних або десктопних.

Взаємодія клієнтських застосунків із серверною частиною здійснюється через REST API. Кожен API-ендпоїнт реалізує конкретну функцію системи та обробляє запити у форматі JSON, що дозволяє легко інтегрувати систему з веб-інтерфейсами, мобільними додатками та сторонніми сервісами. Для контролю доступу використовується механізм Bearer-токенів, який забезпечує автентифікацію користувачів і безпечну роботу з фінансовими даними.

Система спроектована з урахуванням принципів модульності та масштабованості. Кожен модуль є відокремленим компонентом, який можна розширювати або замінювати без впливу на роботу інших частин системи. Така структура дозволяє адаптувати систему під індивідуальні потреби користувача, додавати нові функції та інтегрувати сторонні сервіси,

наприклад, для автоматичного імпорту транзакцій з банківських застосунків, таких як Monobank або Приватбанк.

Завдяки обраній архітектурі та структурі система забезпечує ефективну обробку фінансових даних, високий рівень безпеки та готовність до подальшого розвитку. Центральний серверний компонент у поєднанні з REST API та модульною організацією дозволяє швидко реагувати на зміну вимог, масштабувати систему під більшу кількість користувачів та інтегрувати нові сервіси без суттєвих змін у вже реалізованому функціоналі.

3.3 Реалізація бази даних

Для зберігання фінансових даних користувачів у системі була обрана **реляційна база даних MySQL**, яка забезпечує надійність, цілісність та ефективність обробки інформації. База даних організована з урахуванням нормалізації, що дозволяє уникнути дублювання даних та забезпечує коректність їх зберігання при виконанні транзакцій.

Структура бази даних включає кілька основних таблиць: **Users**, **Categories**, **Transactions** та **Reports**. Таблиця Users містить інформацію про користувачів, включаючи унікальний ідентифікатор, ім’я, електронну пошту, хешований пароль та токен для автентифікації. Таблиця Categories зберігає дані про категорії фінансових операцій, розділені на доходи та витрати, та дозволяє користувачу структурувати бюджет для подальшого аналізу.

Таблиця Transactions є центральною для системи і містить інформацію про всі фінансові операції користувачів, включаючи суму, дату, опис та зв’язок з відповідною категорією і користувачем. Кожна транзакція пов’язана з конкретною категорією через зовнішній ключ, що забезпечує правильну класифікацію даних. Таблиця Reports містить підсумкову аналітичну інформацію про доходи та витрати за визначений період часу, що використовується для формування звітів та графічної візуалізації даних.

Для забезпечення зв’язків між таблицями використані **зовнішні ключі**, які визначають залежності між користувачами, транзакціями та категоріями.

Така організація дозволяє користувачу мати кілька категорій та транзакцій, а також забезпечує правильне агрегування даних при побудові аналітичних звітів.

Взаємодія серверної частини з базою даних реалізована через **ORM Eloquent**, що входить у склад Laravel. ORM дозволяє працювати з таблицями як з об’єктами програмного коду, спрощує виконання CRUD-операцій та автоматично забезпечує відстеження зв’язків між таблицями. Це значно скорочує обсяг коду та зменшує ймовірність помилок при взаємодії з базою даних.

Особлива увага приділена **забезпеченню цілісності даних**. Всі операції запису та редагування транзакцій виконуються у межах транзакцій бази даних, що гарантує коректне збереження даних навіть у разі помилок або збоїв під час обробки запитів. Крім того, реалізовані обмеження унікальності для електронної пошти користувача та назв категорій у межах одного користувача, що запобігає дублюванню даних.

Завдяки продуманій структурі бази даних та використанню ORM, серверна частина системи забезпечує швидкий доступ до інформації, надійне зберігання фінансових даних та можливість масштабування при збільшенні кількості користувачів. Така організація бази даних також дозволяє в майбутньому додавати нові таблиці та функціональні модулі, наприклад, для інтеграції з банківськими сервісами, без необхідності кардинальної зміни існуючої структури.

При проєктуванні структури бази даних особлива увага приділялася забезпеченню логічної цілісності та масштабованості системи. Вибір реляційної моделі даних зумовлений необхідністю чіткого визначення зв’язків між фінансовими сутностями та підтримки складних запитів для аналітичної обробки. Використання нормалізованої структури дозволяє мінімізувати надлишковість даних і забезпечує узгодженість інформації при виконанні операцій додавання, редагування та видалення фінансових записів.

У процесі реалізації бази даних застосовувалися принципи третьої нормальної форми (3NF), що передбачає винесення довідникових даних у окремі таблиці та усунення транзитивних залежностей. Це дозволило розділити дані про користувачів, категорії та фінансові операції таким чином, щоб кожна таблиця відповідала за чітко визначену предметну область. Подібний підхід спрощує супровід системи та полегшує внесення змін у структуру бази даних без ризику порушення цілісності інформації.

Окрему увагу було приділено оптимізації структури таблиці Transactions, оскільки саме вона є найбільш навантаженою в процесі роботи системи. Для підвищення продуктивності доступу до фінансових операцій були реалізовані індекси за ключовими полями, такими як user_id, category_id та date. Використання індексації дозволяє суттєво скоротити час виконання запитів при формуванні звітів, особливо у випадках роботи з великими обсягами історичних даних.

Для зберігання грошових значень у таблиці Transactions використовується тип даних DECIMAL, що забезпечує точність фінансових розрахунків та запобігає похибкам, характерним для чисел з плаваючою комою. Такий вибір є критично важливим для систем управління фінансами, де навіть незначні неточності можуть призводити до викривлення аналітичних результатів та втрати довіри з боку користувача.

У межах реалізації бази даних також враховувалися вимоги до збереження історії змін фінансових записів. Для цього передбачено можливість розширення структури за рахунок додаткових таблиць журналювання або використання м’якого видалення (soft delete), яке підтримується ORM Eloquent. Такий підхід дозволяє зберігати дані про видалені транзакції та забезпечує можливість їх відновлення або аналізу змін у фінансовій поведінці користувача.

Процес управління схемою бази даних реалізовано за допомогою механізму міграцій Laravel. Міграції дозволяють версіонувати структуру бази даних, забезпечують контроль змін та спрощують розгортання системи на

різних середовищах. Використання міграцій також дозволяє автоматизувати створення та оновлення таблиць, що значно знижує ризик помилок при ручному втручанні у структуру бази даних.

Для забезпечення безпеки фінансової інформації у базі даних реалізовано шифрування чутливих даних, зокрема паролів користувачів, із застосуванням сучасних криптографічних алгоритмів хешування. Крім того, доступ до бази даних обмежено на рівні серверної конфігурації, що запобігає несанкціонованому доступу та знижує ризики витоку інформації. Подібний підхід є необхідною умовою для систем, що працюють з конфіденційними персональними та фінансовими даними.

Важливим аспектом реалізації бази даних є забезпечення коректної роботи у багатокористувацькому середовищі. Для цього застосовуються механізми блокування та транзакцій, які гарантують узгодженість даних при паралельному доступі до фінансових ресурсів. Використання транзакцій дозволяє виконувати групу взаємопов’язаних операцій як єдине ціле, що особливо важливо при одночасному додаванні або редагуванні кількох фінансових записів.

З метою підвищення продуктивності системи при формуванні аналітичних звітів у базі даних передбачена можливість зберігання агрегованих даних. Такі дані можуть використовуватися для швидкого відображення підсумкових показників без необхідності повторного обчислення на основі повного набору транзакцій. Це зменшує навантаження на сервер та покращує час відгуку системи при роботі з великими періодами аналізу.

База даних також спроєктована з урахуванням подальшого розширення функціональних можливостей системи. Структура таблиць дозволяє додавати нові атрибути фінансових операцій, такі як валюта, спосіб оплати або джерело доходу, без порушення існуючої логіки роботи. Це створює основу для інтеграції з зовнішніми сервісами, зокрема банківськими API або платіжними системами.

У результаті реалізації описаної структури бази даних забезпечується надійна, масштабована та безпечна платформа для зберігання і обробки фінансових даних користувачів. Використання MySQL у поєднанні з ORM Eloquent дозволяє поєднати високу продуктивність реляційної бази даних із гнучкістю об'єктно-орієнтованого підходу, що є важливим фактором для подальшого розвитку та підтримки системи управління особистими фінансами.

3.4 Реалізація API та механізм авторизації

Серверна частина системи управління особистим бюджетом реалізована у вигляді **RESTful API**, що забезпечує стандартизовану, stateless-взаємодію між клієнтським застосунком (мобільним або веб-інтерфейсом) та бекендом. Архітектура API побудована навколо основних фінансових ресурсів системи, таких як User (Користувач), Transaction (Транзакція), Category (Категорія) та Budget (Бюджет).

REST API підтримує повний набір операцій CRUD (Create, Read, Update, Delete) для ключових фінансових сутностей, використовуючи стандартні методи HTTP:

Таблиця 3.1.

Опис REST API ендпоїнтів системи

/api/transactions	POST	Створення нової транзакції (Дохід/Витрата).
/api/transactions	GET	Отримання списку транзакцій, з підтримкою фільтрації.
/api/transactions/{id}	PUT/PATCH	Оновлення існуючої транзакції.
/api/transactions/{id}	DELETE	Видалення транзакції з бази даних.
/api/categories	GET	Отримання категорій для

		прив'язки до транзакцій.
/api/reports	GET	Генерація фінансового звіту за період.

Розширені можливості запитів: Для забезпечення аналітичних потреб системи, ендпоїнт /api/reports підтримує передачу додаткових параметрів запиту (query parameters). Зокрема, передача часових проміжків (start_date та end_date) у UNIX-форматі дозволяє ефективно фільтрувати та агрегувати інформацію безпосередньо на сервері, оптимізуючи навантаження на клієнтську частину та прискорюючи відображення звітів.

Для забезпечення безпечного доступу використовується сучасний механізм автентифікації на основі токенів (Token-Based Authentication). Після успішного проходження користувачем процесу входу (POST /api/login), сервер, використовуючи пакет Sanctum (для Laravel-бекенду), видає унікальний Bearer-токен. Цей токен клієнтське застосування зберігає та застосовує у заголовку Authorization: Bearer <token> для кожного наступного запиту до захищених ендпоїнтів. Сервер перевіряє дійсність та приналежність токена при кожному запиті, гарантуючи, що лише авторизовані користувачі можуть отримувати або змінювати дані, що належать їм (реалізація принципу Політика доступу до ресурсів/Resource Access Policy). Крім того, для захисту облікових даних користувачів, усі паролі зберігаються виключно у хешованому вигляді із застосуванням сучасних алгоритмів.

Завдяки використанню фреймворку **Laravel**, реалізація API ґрунтується на наступних архітектурних компонентах: Маршрутизація (Routes): Визначення кінцевих точок API та їхнє зіставлення з відповідними методами контролерів.

Контролери (Controllers): Містять бізнес-логіку для обробки вхідних HTTP-запитів, валідації даних та формування відповіді.

Моделі (Models) та Eloquent ORM: Забезпечують взаємодію з базою даних, спрощуючи виконання складних операцій, агрегації та групування даних для звітів.

Додатково до базового набору CRUD-операцій, REST API системи підтримує механізми розширеної фільтрації, сортування та пагінації даних. Це особливо важливо при роботі з великими масивами фінансових транзакцій, коли кількість записів може досягати тисяч або десятків тисяч. Використання параметрів запиту для обмеження вибірки дозволяє зменшити обсяг переданих даних та підвищити швидкість відповіді API, що позитивно впливає на користувацький досвід у клієнтських застосунках.

Для забезпечення коректності обробки фінансових даних на серверній стороні реалізовано багаторівневу систему валідації вхідних запитів. На першому рівні виконується перевірка структури даних відповідно до контракту API, зокрема обов'язковості полів, типів значень та допустимих діапазонів. На другому рівні здійснюється бізнес-логічна валідація, яка враховує правила предметної області, наприклад недопущення створення транзакцій з некоректними сумами або прив'язки до неіснуючих категорій. Такий підхід дозволяє запобігти потраплянню помилкових або неконсистентних даних у базу даних.

Важливим компонентом реалізації REST API є стандартизована обробка помилок. Сервер формує уніфіковані JSON-відповіді з відповідними HTTP-кодами стану, що дозволяє клієнтським застосункам однозначно інтерпретувати результат виконання запиту. Наприклад, у випадку помилки валідації повертається код 422 (Unprocessable Entity), при спробі доступу до чужих ресурсів — 403 (Forbidden), а при відсутності запитуваного ресурсу — 404 (Not Found). Це забезпечує прогнозовану поведінку API та спрощує розробку клієнтської частини.

Окрему увагу приділено питанням продуктивності серверної частини. Для зменшення часу відповіді API використовуються оптимізовані запити до бази даних, вибіркове завантаження зв'язаних сутностей (eager loading) та

обмеження кількості повернених записів. У поєднанні з цим застосовуються механізми кешування результатів обчислень для аналітичних ендпоїнтів, зокрема при формуванні фінансових звітів за фіксовані періоди. Це дозволяє знизити навантаження на сервер та забезпечити стабільну роботу системи при пікових зверненнях користувачів.

Архітектура REST API спроєктована з урахуванням принципу stateless, що означає відсутність збереження стану сесії на сервері між запитами. Уся необхідна інформація для автентифікації та авторизації передається в кожному запиті у вигляді токена доступу. Такий підхід спрощує масштабування системи, оскільки серверні інстанси можуть обробляти запити незалежно одна від одної, що є важливою умовою для горизонтального масштабування у хмарному середовищі.

Для підвищення рівня безпеки реалізовано політики доступу до ресурсів (Policies), які визначають, які дії конкретний користувач може виконувати над певними сутностями. Це дозволяє чітко розмежувати права доступу та гарантує, що користувач має можливість переглядати або змінювати лише ті транзакції та бюджети, які належать йому. Подібний механізм є критично важливим для систем, що працюють з персональними фінансовими даними.

У межах серверної частини також реалізовано логування ключових подій, пов’язаних з роботою API. Фіксуються спроби автентифікації, створення та зміни фінансових транзакцій, а також виникнення виняткових ситуацій. Зібрані журнали подій використовуються для аналізу стабільності роботи системи, виявлення помилок та потенційних спроб несанкціонованого доступу. Це підвищує надійність системи та спрощує її супровід.

Реалізація REST API передбачає можливість подальшого розширення функціональності без порушення існуючих клієнтських інтеграцій. Для цього закладено основу для версіонування API, що дозволяє вводити нові ендпоїнти або змінювати логіку існуючих без втрати зворотної сумісності.

Такий підхід є важливим у контексті довготривалої експлуатації та розвитку системи управління особистими фінансами.

Крім того, серверна частина системи адаптована для інтеграції з зовнішніми сервісами, зокрема платіжними платформами або банківськими API. REST-архітектура дозволяє легко додавати нові точки доступу для імпорту фінансових операцій, синхронізації балансу або отримання актуальних валютних курсів. Це значно розширює можливості системи та підвищує рівень автоматизації управління особистим бюджетом.

Завдяки використанню фреймворку Laravel реалізація REST API відповідає сучасним вимогам до підтримуваності та якості програмного коду. Чітке розділення відповідальності між маршрутами, контролерами, сервісами та моделями забезпечує гнучкість архітектури та спрощує тестування окремих компонентів. У результаті серверна частина системи являє собою надійну, масштабовану та безпечну платформу для автоматизованого управління особистими фінансами користувачів.

3.5 Визначення та реалізація архітектурних патернів і стандартів кодуванн

Розробка програмного забезпечення потребує чітко визначених архітектурних патернів та стандартів кодування, щоб забезпечити якість, масштабованість та підтримку коду. Використання архітектурних патернів дозволяє розробникам створювати системи, що легко адаптуються до змін, забезпечують високу продуктивність та мінімізують можливість виникнення помилок.

Одним з найбільш популярних архітектурних патернів є MVC (Model–View–Controller). Цей патерн розділяє додаток на три взаємозалежні компоненти: модель, вид і контролер. Модель відповідає за управління даними і бізнес–логікою програми. Вид забезпечує представлення даних користувачеві, а контролер обробляє введення користувача і взаємодіє з моделлю для виконання дій. Використання MVC дозволяє забезпечити чітке

розмежування відповідальностей, що полегшує підтримку та розширення коду [18].

Ще одним важливим патерном є REST (Representational State Transfer), який використовується для створення веб–сервісів. REST забезпечує простий, але потужний спосіб взаємодії клієнтів з сервером через HTTP–протокол. Веб–сервіси, побудовані за принципами REST, є легкими, масштабованими та легко інтегруються з іншими системами. У нашому проекті, використовуючи Node.js для бекенду, ми реалізуємо RESTful API для забезпечення взаємодії між клієнтською частиною на основі React та серверною частиною.

Для забезпечення високої якості коду необхідно впровадити стандарти кодування. Стандарти кодування встановлюють правила та рекомендації щодо написання коду, що допомагає забезпечити його читабельність, зрозумілість та підтримуваність. Важливо дотримуватися єдиного стилю кодування, використовувати зрозумілі імена змінних, функцій та класів, а також забезпечувати відповідну документацію коду. Наприклад, у проекті на базі JavaScript ми можемо використовувати стандарт ES6, який забезпечує сучасні можливості мови та підвищує ефективність коду.

Щоб забезпечити додаткову перевірку якості коду, слід використовувати інструменти статичного аналізу коду, такі як ESLint. Цей інструмент дозволяє виявляти потенційні помилки та проблеми в коді ще на етапі розробки, що значно знижує кількість багів у готовому продукті. Використання ESLint у комбінації з іншими інструментами, такими як Prettier, забезпечує автоматичне форматування коду відповідно до встановлених стандартів, що спрощує процес розробки та підтримки.

Тестування також є невід'ємною частиною забезпечення якості програмного забезпечення. Використання модульних тестів, інтеграційних тестів та тестів на рівні системи дозволяє виявляти помилки на ранніх етапах розробки та забезпечувати високу стабільність продукту. У нашому проекті ми можемо використовувати такі інструменти, як Jest та Enzyme для

тестування компонентів React, а також Mocha та Chai для тестування серверної частини на Node.js.

Розробка масштабованих та підтримуваних додатків вимагає також використання архітектурних патернів для управління станом додатку. У випадку з React, ми можемо використовувати патерн Flux або бібліотеку Redux, що дозволяє централізовано управляти станом додатку та забезпечує передбачуваність змін. Використання цих інструментів полегшує роботу з великими та складними додатками, де взаємодія між компонентами може бути важко відстежуваною.

Вибір архітектурних патернів та впровадження стандартів кодування є ключовими аспектами успішної розробки програмного забезпечення. Вони забезпечують структурованість, зрозумілість та підтримуваність коду, а також сприяють підвищенню якості продукту. Впровадження цих практик у нашому проекті дозволить створити надійну, масштабовану та ефективну систему, яка відповідатиме вимогам користувачів та забезпечить високу продуктивність.

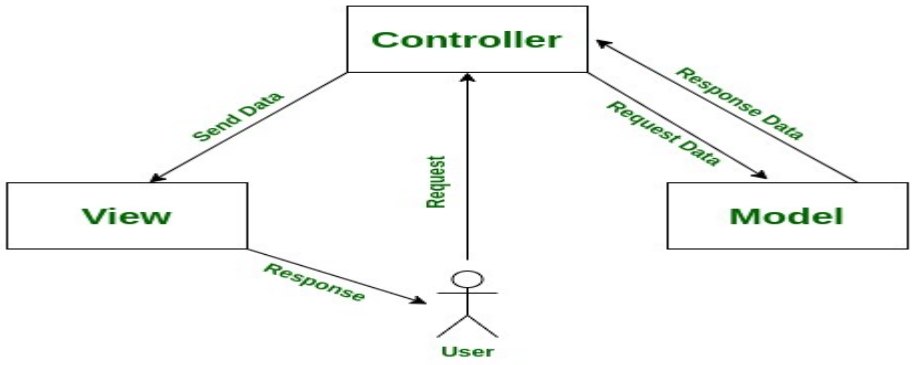

Рис. 3.2. Архітектурний шаблон Model–View–Contoller

3.6 Вибір і налаштування інструментарію

Одним із ключових елементів успішної розробки програмного забезпечення є вибір та налаштування відповідного інструментарію. У нашому проекті ми використовували Visual Studio Code (VSCode) як основне середовище розробки. Це популярне середовище розробки від Microsoft, яке

забезпечує потужні можливості для написання, редагування та тестування коду.

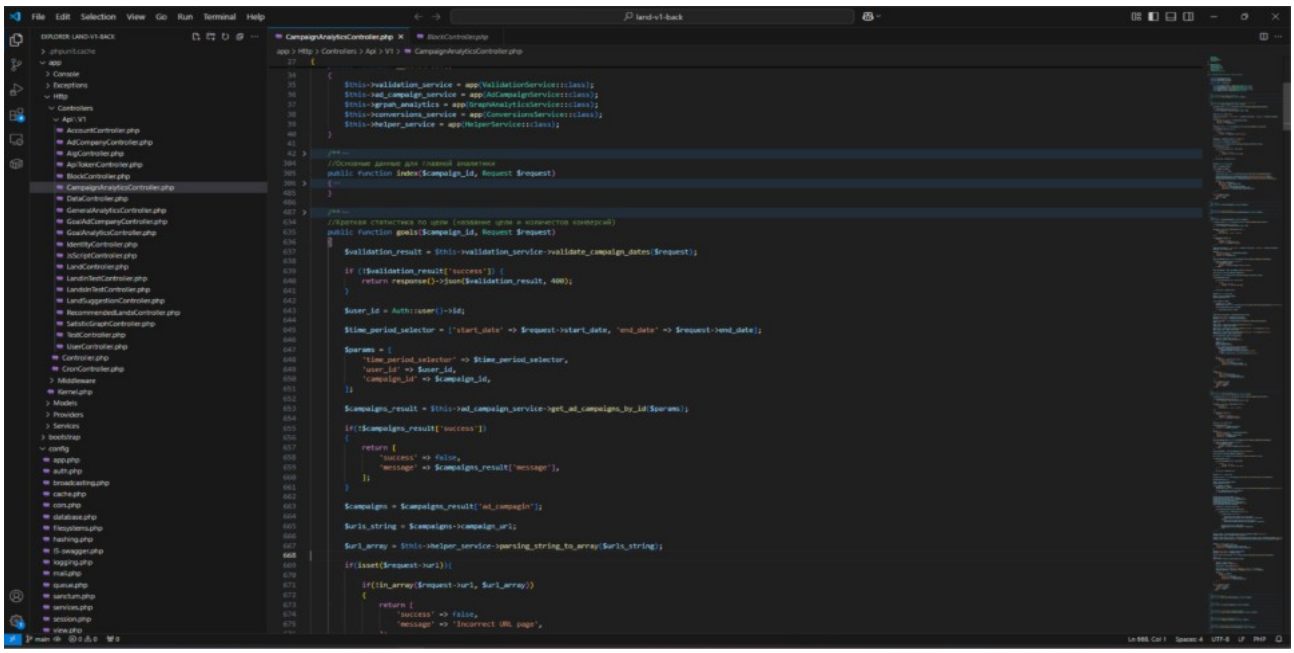

Рис.3.3. Редактор коду VSCode

Visual Studio Code пропонує безліч функцій, які роблять процес розробки зручним та ефективним. Однією з таких функцій є інтегрований термінал, який дозволяє розробникам працювати з командним рядком безпосередньо з середовища розробки. Це особливо корисно для запуску локальних серверів та виконання різних команд, таких як npm або git. У VSCode можна використовувати термінал PowerShell або Git Bash, що забезпечує гнучкість і зручність у роботі.

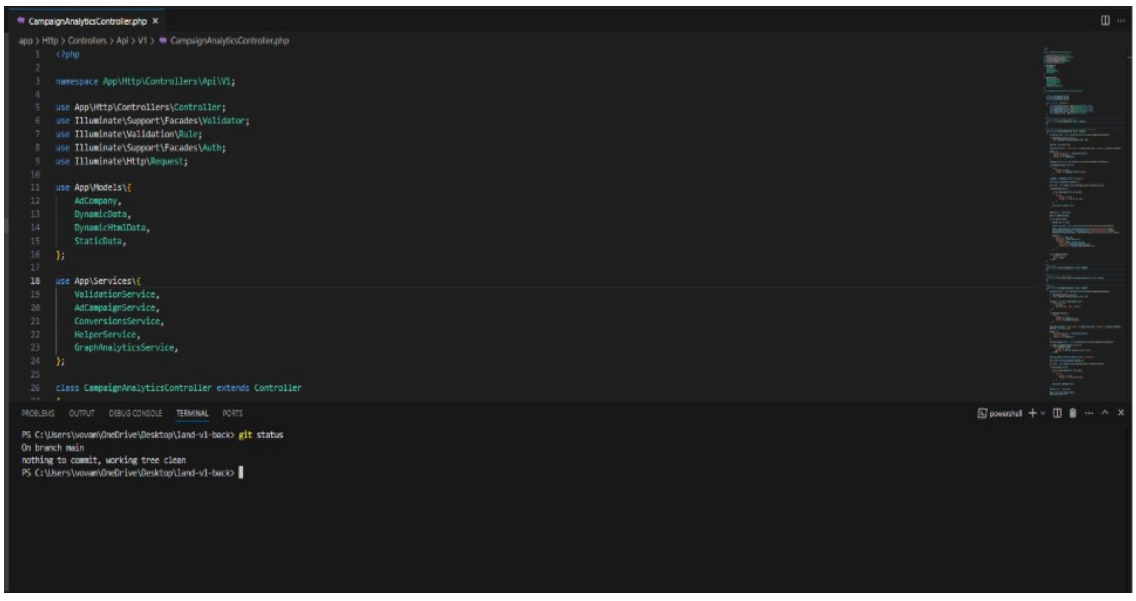

Рис. 3.4. Термінал git bash у VSCode

Налаштування інструментарію включає встановлення Docker Desktop, створення Docker Compose файлу для запуску всіх сервісів проекту, налаштування підключення Laravel до бази даних, а також інтеграцію Visual Studio Code з контейнерами за допомогою плагіна Docker. Це дозволяє безперервно редагувати та тестувати код у стабільному ізольованому середовищі, мінімізуючи ризики конфліктів версій та помилок конфігурації.

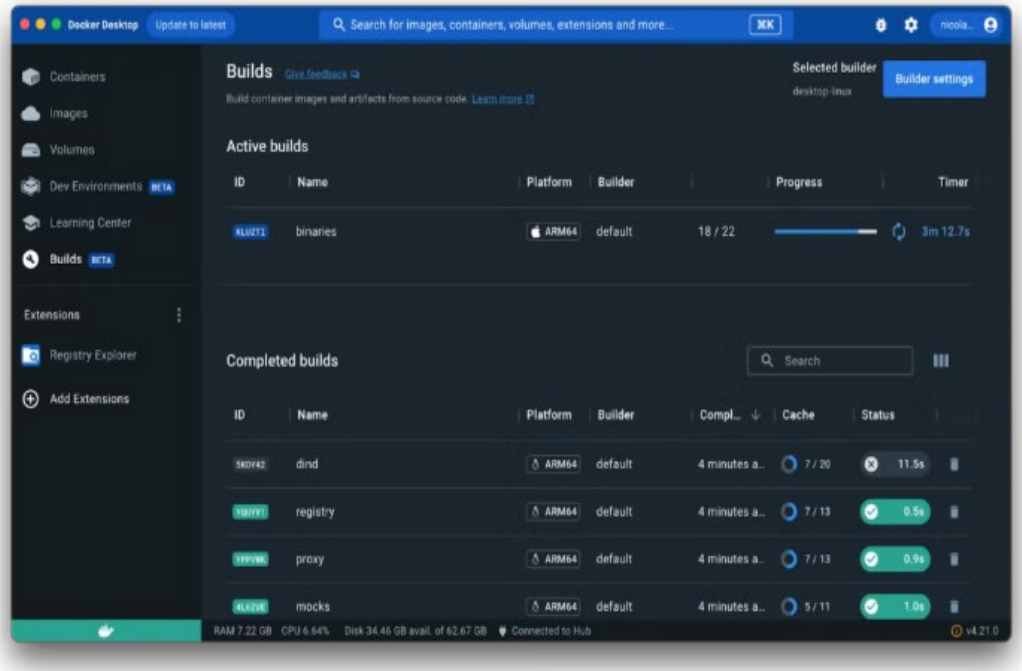

Рис. 3.5 Інтерфейс додатку Docker Desktop

Visual Studio Code також забезпечує чудову підтримку для JavaScript і TypeScript, що є важливим для нашого проекту, оскільки ми використовуємо ці мови для написання як фронтенд, так і бекенд частини додатку. Вбудований автодоповнення, перевірка синтаксису та інтеграція з інструментами налагодження роблять роботу з кодом більш продуктивною і зручною.

Для налаштування середовища розробки під конкретні потреби проекту, Visual Studio Code пропонує безліч конфігураційних можливостей.

Наприклад, можна налаштувати параметри запуску і налагодження додатків, створити завдання для автоматизації процесів або використовувати snippets для швидкого вставлення часто використовуваного коду. Все це сприяє підвищенню продуктивності та зменшенню кількості помилок у коді.

У нашому проекті важливу роль відіграє також інтеграція з системою контролю версій Git. Visual Studio Code забезпечує зручний інтерфейс для роботи з Git, дозволяючи виконувати основні операції, такі як комміти, пуші та пулі, безпосередньо з середовища розробки. Це полегшує командну роботу над проектом, дозволяє швидко відслідковувати зміни в коді та забезпечує кращий контроль над версіями.

Окрім того, Visual Studio Code підтримує налаштування середовища для різних мов програмування та фреймворків. Завдяки цьому, можна легко перемикатися між проектами, які використовують різні технології, і зберігати при цьому комфортний робочий процес.

Загалом, вибір Visual Studio Code як основного середовища розробки для нашого проекту був обумовлений його потужними можливостями, гнучкістю та підтримкою сучасних інструментів розробки. Це середовище забезпечує всі необхідні засоби для ефективної роботи над проектом, включаючи інтеграцію з терміналом, підтримку розширень та зручний інтерфейс для роботи з системою контролю версій.

Для проекту було використано кілька важливих розширень у Visual Studio Code, які допомогли зробити процес розробки більш ефективним і структурованим. Основними з цих розширень були ESLint, Prettier та Tailwind CSS

3.7 Обґрунтування вибору СУБД

Для розробки системи управління особистими фінансами було прийнято рішення використовувати реляційну систему управління базами даних MySQL. Вибір саме цієї СУБД обґрунтований рядом технічних та

практичних факторів, які забезпечують ефективну роботу системи та виконання всіх функціональних і нефункціональних вимог проекту.

По-перше, **MySQL є стабільною та перевіреною СУБД**, яка широко використовується у веб-розробці та інтегрується з популярними фреймворками, включаючи Laravel. Це дозволяє скористатися стандартними бібліотеками та інструментами ORM Eloquent для роботи з даними, спрощує розробку і підтримку коду, а також зменшує ймовірність виникнення помилок при взаємодії з базою даних.

По-друге, MySQL підтримує реляційну модель даних, що є оптимальною для обліку фінансових операцій. Таблиці Users, Categories, Transactions та Reports мають чітко визначені зв’язки через зовнішні ключі, що забезпечує цілісність даних та дозволяє ефективно виконувати запити для аналітики та формування звітів. Реляційна модель спрощує реалізацію складних операцій агрегування, сортування та фільтрації даних, що особливо важливо для системи управління бюджетом.

По-третє, MySQL забезпечує високу швидкість обробки запитів та масштабованість. Навіть при великій кількості користувачів і транзакцій система демонструє стабільну роботу та швидку відповідь на запити. Для подальшого росту проекту MySQL дозволяє застосовувати реплікацію бази, індексацію та оптимізацію запитів, що підвищує продуктивність системи.

По-четверте, MySQL забезпечує надійність та безпеку даних. Підтримка транзакцій, механізмів резервного копіювання та відновлення даних дозволяє гарантувати збереження фінансової інформації користувачів навіть у випадку збою серверного середовища. Крім того, інтеграція з Laravel дозволяє реалізувати додаткові механізми контролю доступу та валідації даних перед записом у базу.

Для зручності візуального управління базою даних та швидкого аналізу структури таблиць використовується TablePlus. Цей інструмент дозволяє переглядати дані в таблицях, редагувати записи, налаштовувати індекси та зовнішні ключі, а також виконувати SQL-запити безпосередньо з графічного

інтерфейсу. TablePlus значно спрощує роботу з базою даних під час розробки, тестування та налагодження системи

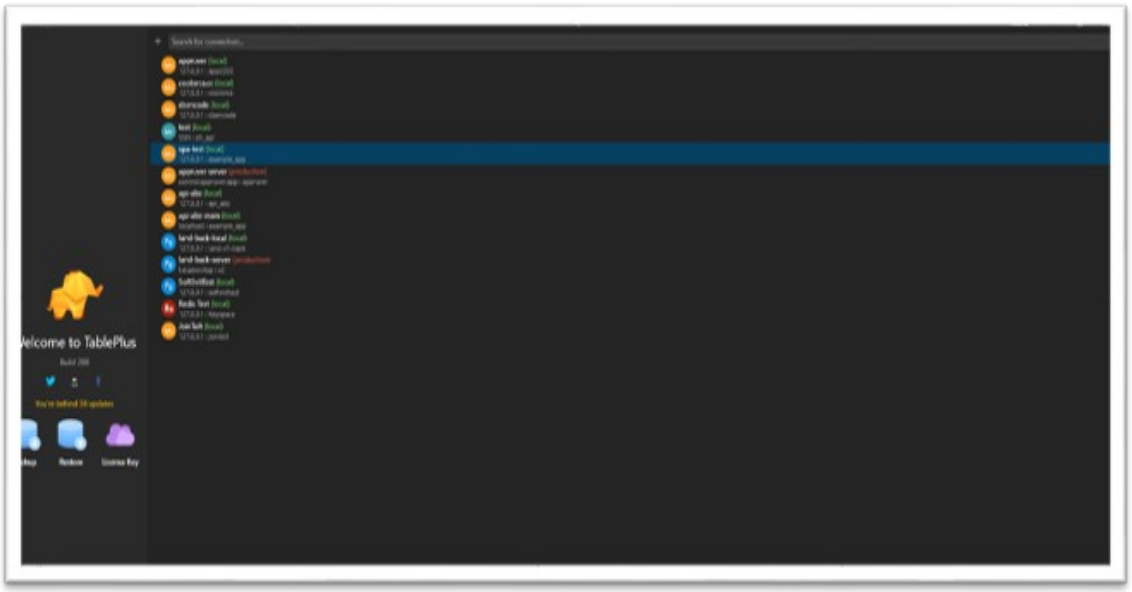

Рис. 3.6. Графічний інтерфейс TablePlus

Таким чином, вибір MySQL як основної СУБД для проекту обґрунтований її стабільністю, швидкістю, безпекою, підтримкою реляційної моделі даних, легкістю інтеграції з фреймворком Laravel та можливістю візуального управління через TablePlus. Ця СУБД повністю відповідає вимогам системи управління особистими фінансами, забезпечує ефективну обробку даних та готовність до подальшого розвитку проекту.

3.8 Використання Laravel та Laravel Sail у практичній реалізації

Для зручного запуску локального середовища розробки був використаний **Laravel Sail** — офіційний інструмент Laravel для роботи з Docker-контейнерами. Sail надає готовий стек сервісів, включаючи PHP, MySQL, Redis, а також веб-сервер, що дозволяє швидко розгортати проект на локальній машині без додаткової ручної конфігурації серверного середовища. Використання Sail дозволяє забезпечити однакове середовище

для всіх розробників, виключає проблеми сумісності версій та спрощує інтеграцію з інструментами CI/CD у майбутньому.

Процес роботи з Laravel Sail включав створення Docker-контейнерів для веб-сервера та бази даних, налаштування середовища через файл **.env**, а також запуск команд для міграції та заповнення бази тестовими даними. Завдяки Sail розробка та тестування API-ендпоїнтів стали простими та контрольованими: усі зміни в коді автоматично доступні всередині контейнерів, а база даних ізольована і зберігається у власному томі.

API-сервіс у Laravel реалізований через маршрути, контролери та моделі Eloquent. Кожен ендпоїнт обробляє запити у форматі JSON, включаючи перевірку вхідних даних, валідацію та авторизацію користувачів за допомогою Bearer-токенів. Тестування функціоналу проводилось за допомогою інтеграційних тестів Laravel та ручної перевірки через Postman, що дозволило переконатися у правильності обробки транзакцій, коректності генерації звітів та стабільності роботи системи.

Використання Laravel та Sail забезпечило швидку та безпечну розробку, контрольоване локальне середовище, легку масштабованість і готовність проекту до перенесення на продуктивні сервери. Такий підхід дозволяє в майбутньому інтегрувати систему з мобільними додатками або сторонніми сервісами, зберігаючи стабільність та безпеку фінансових даних користувачів.

Рис. 3.7. Запуск Laravel Sail

3.9 Тестування програмного додатку

Тестування програмного додатку є важливим етапом розробки системи управління особистими фінансами, оскільки забезпечує перевірку коректності роботи функціональних модулів, стабільності обробки даних та відповідності реалізованого програмного забезпечення встановленим

вимогам. Основною метою тестування було виявлення та усунення помилок на етапі розробки, а також підтвердження працездатності API-системи в реальних умовах використання.

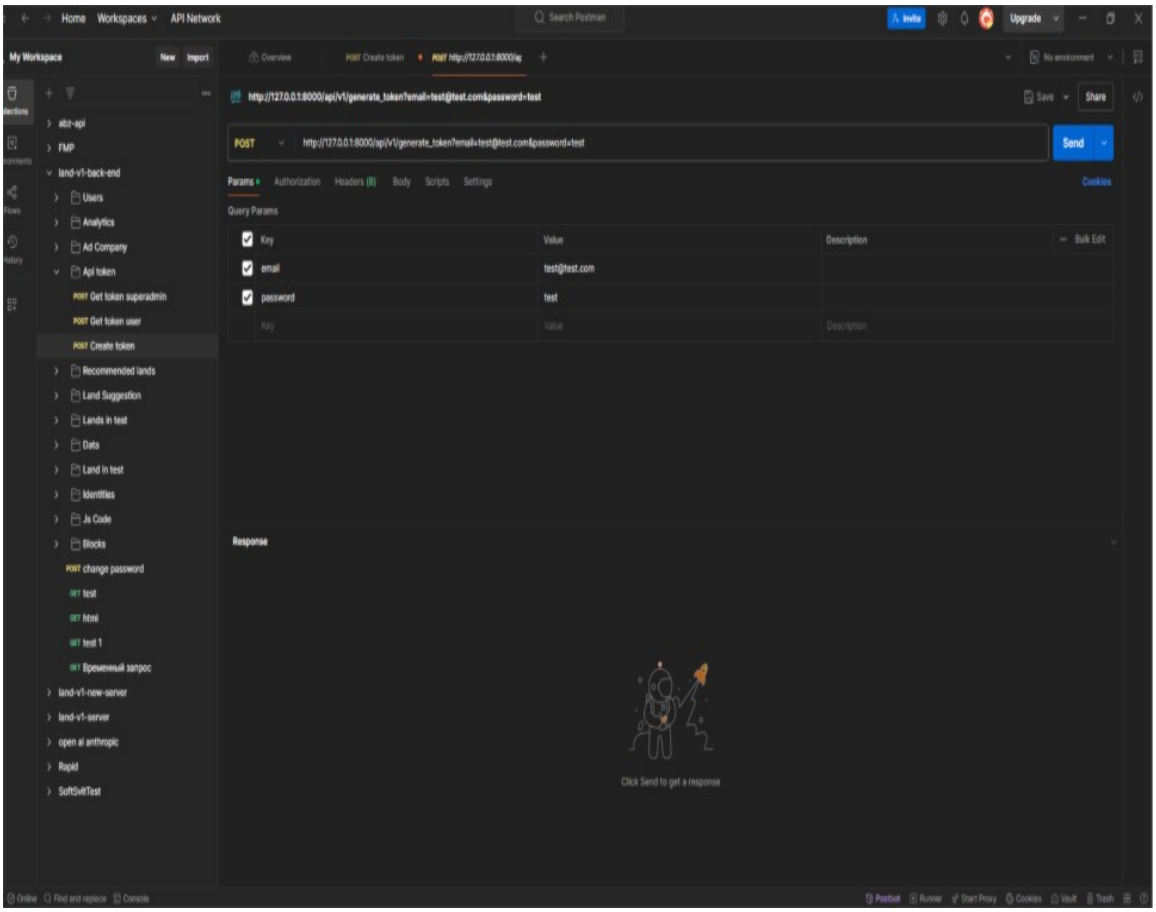

Рис. 3.8 приклад виконання HTTP-запиту до API за допомогою сервісу Postman.

Під час тестування перевірялась коректність обробки запитів методів GET, POST, PUT та DELETE, а також правильність формування відповідей у форматі JSON. Особлива увага приділялася відповідності структури відповіді сервера технічним вимогам та обробці помилкових запитів.

Окрема увага приділялася тестуванню механізмів автентифікації та авторизації користувачів. Перевірялася робота реєстрації та входу в систему, генерація та передача Bearer-токенів, а також обмеження доступу до захищених ресурсів.

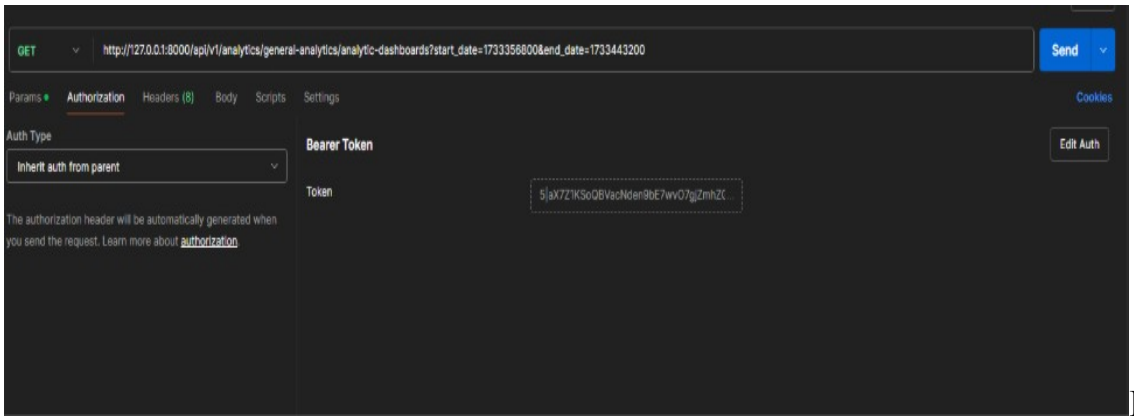

Рис. 3.9 використання Bearer-токена у заголовках запиту в Postman.

Також проводилось тестування бізнес-логіки системи, зокрема додавання, редагування та видалення фінансових транзакцій, робота з категоріями доходів і витрат, а також формування агрегованої інформації для аналізу бюджету. Перевірялася коректність обчислень сум, фільтрація даних за періодами та відповідність отриманих результатів очікуваним значенням.

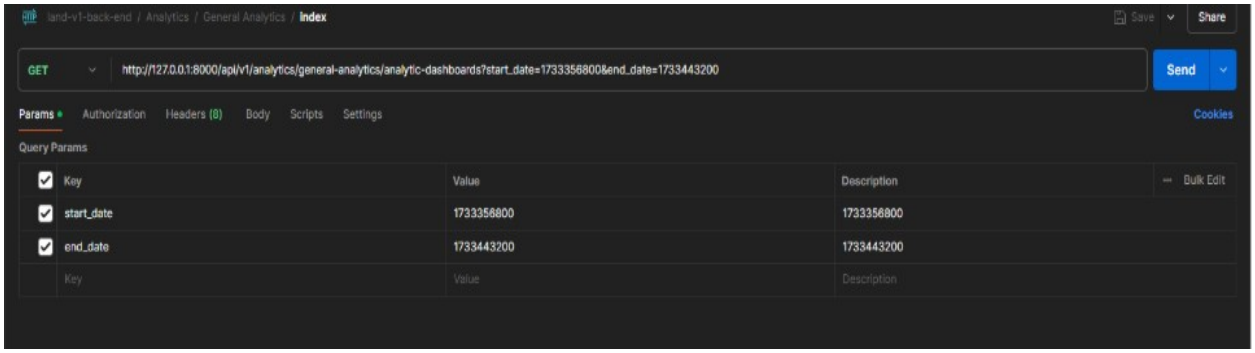

Рис. 3.10 приклад тестування ендпоїнта для додавання фінансової транзакції

Під час тестування аналітичних ендпоїнтів та формування звітів перевірялася робота фільтрації фінансових даних за часовими проміжками. Для цього у відповідних HTTP-запитах використовувались параметри start_date та end_date, які передаються у вигляді часових міток у UNIX-форматі. Такий підхід забезпечує універсальність передачі дати та часу між клієнтською та серверною частинами системи, а також зменшує ризик помилок, пов’язаних з часовими поясами та локальними форматами дат.

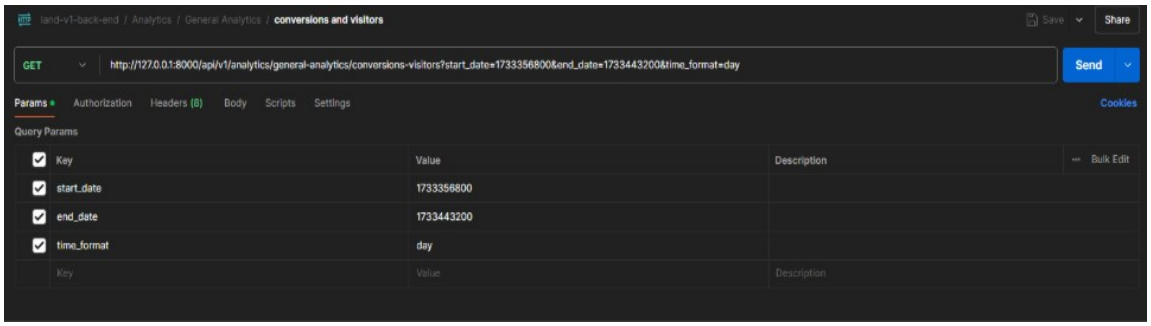

Рис. 3.11 приклад запиту в Postman з параметрами start_date та end_date у UNIX-форматі.

Також проводилось тестування бізнес-логіки системи, зокрема додавання, редагування та видалення фінансових транзакцій, робота з категоріями доходів і витрат, а також формування агрегованої інформації для аналізу бюджету. Перевірялася коректність обчислень сум, фільтрація даних за заданим періодом та відповідність отриманих результатів очікуваним значенням.

У процесі тестування були виявлені незначні помилки, пов’язані з валідацією вхідних даних та обробкою крайових випадків. Усі знайдені недоліки були усунені шляхом доопрацювання логіки контролерів та додавання додаткових перевірок на серверній стороні. Повторне тестування підтвердило стабільну роботу системи та відсутність критичних помилок.

Таким чином, проведене тестування програмного додатку підтвердило коректність реалізації API-системи, стабільність роботи серверної частини та готовність програмного забезпечення до подальшого використання або розгортання у продуктивному середовищі.

3.10 Робота користувача з програмним додатком

Робота користувача з програмним додатком системи управління особистими фінансами здійснюється через взаємодію з серверною частиною за допомогою REST API. Користувач може підключатися до системи за допомогою клієнтського застосунку або сервісів тестування API, зокрема

Postman, що дозволяє перевіряти весь функціонал без необхідності реалізації графічного інтерфейсу.

Початковим етапом роботи користувача є реєстрація в системі. Під час реєстрації користувач передає базові персональні дані, які проходять перевірку на серверній стороні та зберігаються в базі даних. Після успішної реєстрації користувач отримує можливість виконати вхід до системи, у результаті чого генерується токен автентифікації, який використовується для доступу до захищених ресурсів API.

Після проходження автентифікації користувач може виконувати основні операції з управління особистими фінансами. До таких операцій належить створення, перегляд, редагування та видалення фінансових транзакцій, які відображають доходи та витрати. Кожна транзакція пов’язується з відповідною категорією, що дозволяє структурувати фінансові дані та забезпечує зручний аналіз бюджету.

Користувач також має можливість керувати списком категорій доходів і витрат, створюючи власні категорії або редагуючи наявні. Такий підхід дозволяє адаптувати систему під індивідуальні потреби користувача та різні сценарії ведення бюджету.

Окрім роботи з транзакціями, система надає функціонал перегляду фінансової інформації за визначений період. Для цього використовуються запити з параметрами start_date та end_date, які передаються у UNIX-форматі. Це дозволяє користувачу отримувати зведену інформацію про витрати та доходи за обраний часовий проміжок і здійснювати аналіз фінансової активності.У процесі взаємодії з програмним додатком користувач отримує структуровані відповіді сервера у форматі JSON, які містять результати виконання запитів або повідомлення про помилки. У випадку некоректного введення даних або відсутності прав доступу система повертає відповідні коди помилок, що дозволяє користувачу своєчасно виявити та виправити недоліки.

3.11 Аналіз отриманих навичок та досвіду в контексті подальшого використання в розробці

В процесі розробки системи управління особистими фінансами були здобуті численні практичні навички та досвід, що безпосередньо впливають на ефективність майбутньої розробки подібних програмних продуктів. По-перше, значно покращилось розуміння принципів роботи сучасних веб-фреймворків, зокрема Laravel, що дозволяє організовувати код за архітектурним шаблоном MVC, реалізовувати маршрутизацію REST API, а також використовувати ORM Eloquent для роботи з базою даних. Цей досвід забезпечує можливість створювати структуровані та масштабовані веб-додатки з чітким розділенням відповідальності між компонентами.

По-друге, робота з Docker та Laravel Sail дала глибоке розуміння організації ізольованого середовища розробки, що спрощує тестування, налагодження та перенесення проектів між різними комп’ютерами або серверами. Використання контейнерів дозволяє зменшити ризики конфліктів версій програмного забезпечення, а також забезпечує стабільність і передбачуваність середовища, що є критично важливим для комплексних систем, які взаємодіють з базами даних та сторонніми сервісами.

В процесі розробки активно застосовувався Postman для тестування API, що дозволило не лише перевіряти коректність роботи ендпоїнтів, а й здобути досвід у побудові ефективних сценаріїв тестування, організації тестових наборів, роботи з заголовками, параметрами запитів та обробкою токенів авторизації. Здобутий досвід у тестуванні REST API буде безпосередньо корисним при розширенні функціоналу та інтеграції з новими клієнтськими застосунками або сторонніми сервісами.

Також значним досвідом стало використання Unix-формату часу для параметрів start_date та end_date у запитах до API, що дозволяє стандартизувати передачу часових даних між клієнтською та серверною

частинами та забезпечує точність обробки інформації при формуванні фінансових звітів. Це знання стане в нагоді при реалізації аналітичних функцій та агрегування даних у майбутніх проектах.

Крім технічних навичок, значний досвід було здобуто у плануванні та структуризації розробки, включаючи системний аналіз, проектування баз даних, визначення функціональних і нефункціональних вимог, а також організацію тестування програмного забезпечення. Ці компетенції дозволяють більш ефективно управляти процесом розробки, оцінювати ризики та приймати обґрунтовані рішення під час створення нових модулів.

Отже, виконана робота не лише забезпечила реалізацію функціональної системи управління особистими фінансами, а й надала цінний практичний досвід у використанні сучасних технологій веб-розробки, інструментів для тестування та управління середовищем. Отримані навички та знання є безпосередньо застосовними для подальшого розвитку та підтримки аналогічних програмних продуктів

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи над проєктом «**Розробка системи управління фінансами та автоматизація особистого бюджету»** було повністю реалізовано всі поставлені цілі та завдання, визначені технічним завданням. Основною метою роботи було проєктування та розробка серверної частини інформаційної системи, яка дозволяє здійснювати облік доходів і витрат користувачів, забезпечує аналітичну обробку фінансових даних та надає доступ до функціоналу через REST API. У результаті виконаної роботи створено працездатне, масштабоване та безпечне програмне рішення, яке відповідає сучасним вимогам до вебзастосунків.

У межах дипломної роботи було проведено аналіз предметної області, визначено основні функціональні та нефункціональні вимоги до системи управління особистими фінансами. На основі аналізу було спроєктовано логічну та фізичну структуру бази даних, яка забезпечує ефективне зберігання інформації про користувачів, фінансові транзакції, категорії доходів і витрат, а також допоміжні сутності, необхідні для коректної роботи системи. Проєктування бази даних здійснювалося з урахуванням принципів нормалізації, що дозволило уникнути надмірності даних та підвищити цілісність інформації.

Для реалізації серверної частини системи було обрано фреймворк **Laravel**, який надає широкий набір інструментів для розробки вебзастосунків, реалізації REST API, роботи з базами даних та забезпечення безпеки. Використання архітектурного патерну MVC дозволило чітко розділити бізнес-логіку, обробку запитів та взаємодію з базою даних, що позитивно вплинуло на читабельність, підтримуваність і розширюваність коду. Взаємодія з реляційною базою даних **MySQL** була реалізована за допомогою ORM Eloquent, що спростило виконання CRUD-операцій та зменшило кількість низькорівневих SQL-запитів.

У процесі розробки системи було реалізовано повноцінний **REST API**, який забезпечує доступ до основного функціоналу застосунку. Розроблені ендпоїнти дозволяють виконувати операції додавання, редагування, перегляду та видалення фінансових транзакцій, керувати категоріями доходів і витрат, а також отримувати зведену аналітичну інформацію за вибрані часові проміжки. Для передачі часових параметрів було використано формат UNIX timestamp, що забезпечує універсальність та коректну обробку дат незалежно від часових поясів.

Особливу увагу під час реалізації було приділено питанням **безпеки**. Доступ до захищених ресурсів API здійснюється з використанням механізму автентифікації на основі **Bearer-токенів**, що дозволяє обмежити доступ неавторизованих користувачів до фінансових даних. Реалізовано перевірку коректності вхідних даних, обробку помилкових запитів та повернення структурованих HTTP-відповідей із відповідними статус-кодами, що відповідає принципам REST-архітектури та підвищує надійність системи.

Для забезпечення стабільного та відтворюваного середовища розробки було використано технології **Docker** та **Laravel Sail**. Контейнеризація дозволила ізолювати середовище виконання застосунку, забезпечити однакові умови для розробки та тестування, а також значно спростити розгортання проєкту. Використання Docker зменшує залежність від конфігурації операційної системи та дозволяє швидко відновити робоче середовище у разі виникнення помилок або збоїв.

Тестування розробленого API здійснювалося за допомогою інструмента **Postman**. У процесі тестування було перевірено коректність роботи всіх реалізованих ендпоїнтів, правильність обробки параметрів запитів, зокрема start_date та end_date, а також роботу механізмів автентифікації та авторизації. Окрему увагу було приділено тестуванню граничних випадків і помилкових запитів, що дозволило виявити та усунути потенційні недоліки на етапі розробки. Результати тестування підтвердили стабільність та коректність роботи системи.

Реалізована система управління особистими фінансами має практичну цінність та може бути використана як основа для створення повнофункціонального програмного продукту. Архітектура застосунку дозволяє легко розширювати функціонал, зокрема шляхом розробки клієнтської частини у вигляді вебінтерфейсу або мобільного застосунку, інтеграції з платіжними сервісами, імпорту банківських виписок або додавання розширених аналітичних інструментів. Таким чином, система є масштабованою та готовою до подальшого розвитку.

У процесі виконання кваліфікаційної роботи було набуто ґрунтовного практичного досвіду роботи з сучасними технологіями веброзробки. Отримані навички включають проєктування та реалізацію REST API, роботу з фреймворком Laravel, використання ORM для взаємодії з базами даних, контейнеризацію застосунків за допомогою Docker, а також тестування програмного забезпечення. Крім того, було поглиблено знання з проєктування інформаційних систем, аналізу вимог та побудови архітектури серверних застосунків.

Підсумовуючи, можна зробити висновок, що всі поставлені в межах дипломної роботи завдання були успішно виконані, а мета роботи — досягнута в повному обсязі. Результати дослідження та розробки підтверджують доцільність використання обраних технологій та підходів для створення сучасних інформаційних систем у сфері управління особистими фінансами. Розроблений проєкт відповідає вимогам технічного завдання та може бути рекомендований для подальшого практичного використання і розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бейсік А. Веб-розробка на PHP та Laravel : навч. посіб. Київ : Наукова думка, 2021. 320 с.
2. Богачев С. Node.js: практичний посібник для початківців. Харків : Фоліо, 2022. 320 с.
3. Власенко П. В. Безпека веб-застосунків. Харків : УІПА, 2020. 31 с.
4. Глушко С. О. Розробка клієнт-серверних додатків на Node.js. Збірник наукових праць ХНУРЕ. 2021. № 3. С. 45–52.
5. Слободяник М. Ю. Веб-програмування: HTML, CSS, JavaScript. Львів : Новий Світ, 2020.
6. Смирнов І. Реляційні бази даних: проєктування та адміністрування. Харків : ХНУ, 2020. 280 с.
7. Docker Documentation. Docker Engine Overview. URL: https://docs.docker.com/ (дата звернення: 12.09.2025).
8. Express.js Guide. Express.js — Node.js web application framework. URL: https://expressjs.com (дата звернення: 18.09.2025).
9. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2003. 560 p.
10. IEEE Standards Association. IEEE Std 829-2008 — Standard for Software and System Test Documentation. URL: https://standards.ieee.org/ (дата звернення: 25.09.2025).
11. Kelleher J. Node.js: Novice to Ninja. SitePoint, 2018.
12. Laravel Documentation. Laravel — The PHP Framework for Web Artisans. URL: https://laravel.com/docs (дата звернення: 02.10.2025).
13. Martin R. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
14. MySQL Documentation. MySQL Reference Manual. URL: https://dev.mysql.com/doc/ (дата звернення: 08.10.2025).
15. Postman Documentation. Postman Learning Center. URL: https://learning.postman.com/ (дата звернення: 14.10.2025).

16. Pressman R. Software Engineering: A Practitioner's Approach. McGraw-Hill Education, 2019. 900 p.
17. SQLite Documentation. SQLite Reference Manual. URL: https://www.sqlite.org/docs.html (дата звернення: 21.10.2025).
18. TablePlus Documentation. TablePlus User Guide. URL: https://tableplus.com/docs (дата звернення: 27.10.2025).
19. W3Schools. HTML, CSS, and JavaScript Tutorials. URL: https://www.w3schools.com (дата звернення: 30.10.2025).
20. Martin R. *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Prentice Hall, 2017
21. Фрімен Е. Патерни проєктування. Київ: Фабула, 2022. 672 с.
22. Мартин Р. Чиста архітектура. Мистецтво розробки ПЗ. Київ: Основи, 2020. 432 с
23. Шварц Б. MySQL. Оптимізація продуктивності. 4-те вид. O'Reilly, 2022. 380 с.
24. Бьюлі А. Вивчаємо SQL. 3-тє вид. O'Reilly, 2020. 320 с.
25. Фуллер М. Розробка API на Node.js. Спб.: БХВ, 2023. 350 с.
26. Коваленко В. В. Розвиток цифрового банкінгу в Україні. *Вісник НБУ*. 2022. № 12.
27. Васильєва Т. А. Вплив цифровізації на фінансову інклюзію населення. *Фінанси України*. 2024. № 1.
28. Офіційний сайт Національного банку України. URL: https://bank.gov.ua
29. Державна служба статистики України. URL: https://ukrstat.gov.ua
30. Офіційний сайт monobank (АТ "Універсал Банк"). URL: https://www.monobank.ua
31. Офіційний сайт ПриватБанку. URL: https://privatbank.ua
32. Портал Дія.Цифрова освіта: Фінансова грамотність. URL: https://osvita.diia.gov.ua
33. Специфікація OpenAPI (Swagger). URL: https://swagger.io/specification/
34. Документація MySQL 8.0. URL: https://dev.mysql.com/doc/

35. MDN Web Docs: HTTP & REST API. URL: https://developer.mozilla.org
36. Statista: Personal Finance Market Report 2025. URL: https://www.statista.com
37. Gallego Cossio L. Financial Management Models for Digital Ventures. *MDPI*. 2024.
38. Williams D. The Future of Personal Banking and Fintech. *Global Finance Review*. 2025.
39. Smith J. AI-based Personal Financial Management: Opportunities and Challenges. *ComFin Research*. 2025.
40. Про платіжні послуги: Закон України від 30.06.2021 № 1591-IX.
41. Про віртуальні активи: Закон України від 17.02.2022 № 2074-IX.
42. Кузнєцов О. В. Безпека даних у фінансових API-системах. *Кібербезпека*. 2024. № 2.
43. Мельник О. С. Автоматизація обліку доходів і витрат фізичних осіб за допомогою мобільних застосунків. *Науковий погляд*. 2023. № 4.
44. Кізима Т. О. Особисті фінанси: навч. посіб. Тернопіль: ТНЕУ, 2021. 240 с.Клейсон Д. С. Найбагатший чоловік у Вавилоні. Київ: Книголав, 2020. 160 с. (Класика теорії).
45. Пєтухова О. М. Управління фінансами домогосподарств в умовах цифровізації. К.: ЦУЛ, 2024. 192 с.
46. Стратегія розвитку фінансового сектору України до 2025 року: Рішення Правління НБУ від 16.01.2020.
47. Кійосакі Р. Багатий тато, бідний тато. Київ: Світ знань, 2021. 256 с.
48. Стандарт безпеки даних індустрії платіжних карток (PCI DSS), версія 50.

    Нюберг С. Проєктування баз даних. Практичний посібник. Спб.: БХВ, 2023. 412 с.
49. Гриценко А. А., Шевченко О. В. Інформаційні системи та технології у фінансовій сфері. Київ : КНЕУ, 2021. 356 с.
50. Бондаренко М. Ф. Козаченко Г. В. Проєктування та розробка веб-застосунків. Харків : ХНЕУ ім. С. Кузнеця, 2020. 240 с.

51. Мороз С. М. Безпека інформаційних систем і веб-застосунків. Львів : Львівська політехніка, 2022. 198 с.
52. Кравчук О. В. Данилюк І. М. Цифрові фінансові сервіси та фінансові технології. Київ : НАБУ, 2023. 214 с.
53. Петренко В. С. Автоматизація обліку та аналізу особистих фінансів. Тернопіль : ЗУНУ, 2022. 180 с.

ДОДАТОК А

```
private function get_conversions_by_url($url, $time_period_selector, $goals)
{
    $conversions = 0;
    $profit = 0;
    $goal_final_data = [];

    $sub_query = DynamicData::select('session_id')
        ->where('url', $url)
        ->whereRaw('visit_time = (
            SELECT MIN(visit_time)
            FROM dynamic_data AS dd
            WHERE dd.session_id = dynamic_data.session_id
        )')
        ->whereBetween('visit_time', [
            $time_period_selector['start_date'],
            $time_period_selector['end_date']
        ])
        ->groupBy('session_id')
        ->pluck('session_id');

    $sessions = DynamicData::whereIn('session_id', $sub_query)
        ->pluck('session_id');

    foreach ($goals as $goal) {

        $goal_cost = $goal->goal_price;
        $goal_data = [
            'title' => $goal->goal_title,
```

```
    'conversions' => 0
];

if ($goal->goal_type === 'url') {

    $goal_url = optional($goal->urls->first())->url;

    if ($goal_url) {
        $count = DynamicData::where('url', $goal_url)
            ->whereIn('session_id', $sessions)
            ->whereBetween('visit_time', [
                $time_period_selector['start_date'],
                $time_period_selector['end_date']
            ])
            ->select('session_id', 'client_id')
            ->distinct()
            ->count();

        $conversions += $count;
        $profit += $count * $goal_cost;
        $goal_data['conversions'] = $count;
    }
}

if ($goal->goal_type === 'click') {

    $click = $goal->clicks->first();
    $click_count = collect();

    if ($click && $click->element === 'Tag') {
```

```
                    [$operator, $value] = IDENTIFIER_CONDITIONS[$click->identifier];

                                                                $click_count = DynamicHtmlData::whereBetween('time_of_action', [
                        $time_period_selector['start_date'],
                        $time_period_selector['end_date']
                    ])
                    ->where('url', $click->url)
                    ->whereIn('session_id', $sessions)
                    ->where('html_tag', $operator, $value . $click->text . $value)
                    ->select('client_id', 'session_id')
                    ->distinct()
                    ->get();
                }

                $count = $click_count->count();
                $conversions += $count;
                $profit += $count * $goal_cost;
                $goal_data['conversions'] = $count;
            }

            $goal_final_data[] = $goal_data;
        }

        return [
            'conversions' => $conversions,
            'profit' => $profit,
            'goals' => $goal_final_data
```

```
    ];
}
```

ДОДАТОК Б.

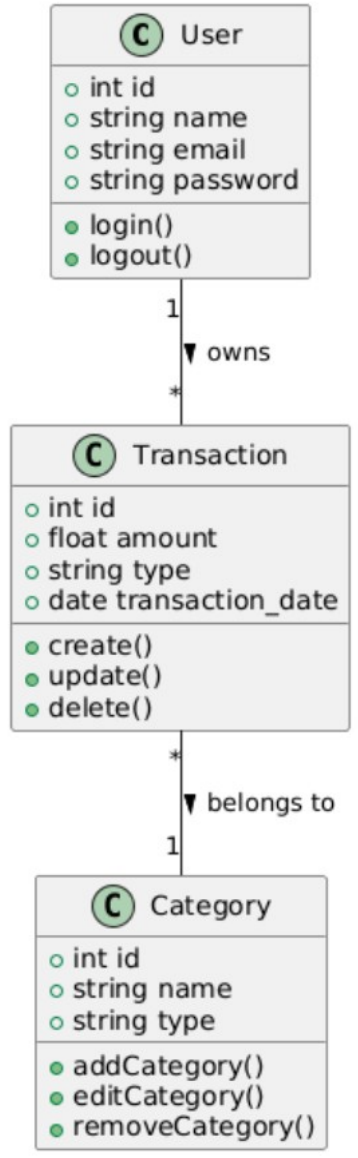
User
int id
string name
string email
string password
login()
logout()
1
owns
*
Transaction
int id
float amount
string type
date transaction_date
create()
update()
delete()
*
belongs to
1
Category
int id
string name
string type
addCategory()
editCategory()
removeCategory()

Рисунок А. клас User пов’язаний з багатьма Transaction, кожна транзакція належить до однієї Category

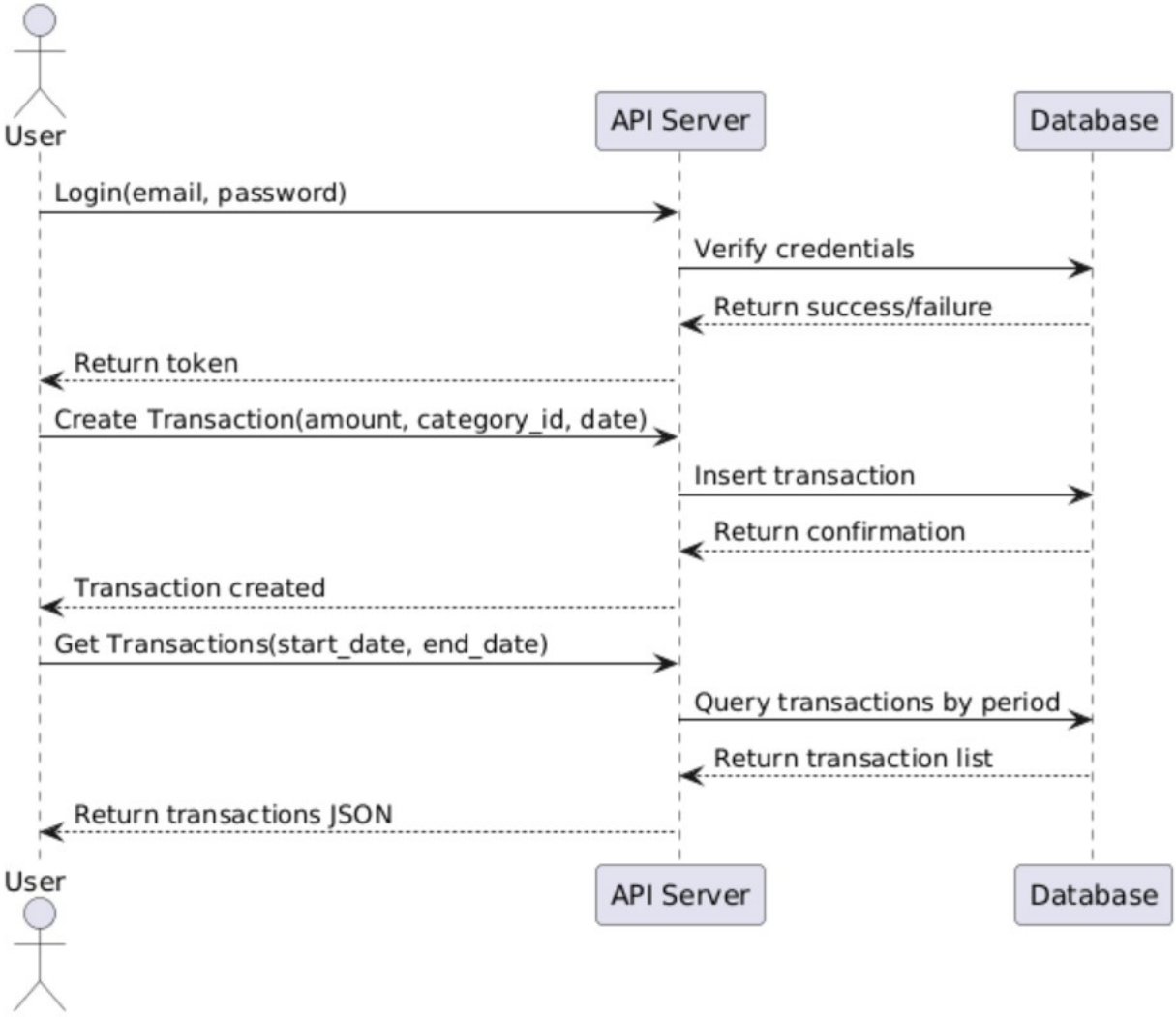

Рисунок Б. показано взаємодію користувача, серверу та бази даних при логіні, додаванні та отриманні транзакцій.

ДОДАТОК В

Таблиця В.

Програмні специфікації

Секція	**Опис**
Реєстрація користувачів	Створення облікового запису користувача з персональною інформацією. Перевірка унікальності електронної адреси. Авторизація користувача для доступу до системи.
Управління транзакціями	Додавання, редагування та видалення фінансових операцій (доходи та витрати). Призначення транзакцій до відповідних категорій.
Категорії фінансів	Створення та редагування категорій доходів і витрат. Можливість налаштування власних категорій користувачем.
Перегляд та аналіз бюджету	Формування звітів за обраний період (параметри start_date та end_date у UNIX-форматі). Відображення зведеної інформації про доходи та витрати.
Розрахунок фінансової статистики	Автоматичний підрахунок сум доходів, витрат, залишку бюджету. Генерація діаграм та графіків для наочного аналізу фінансів.
Безпека	Захист облікових записів користувачів за допомогою Bearer-токенів. Шифрування паролів та критичних даних у базі.
Тестування та стабільність	Перевірка коректності роботи ендпоїнтів через Postman. Виявлення та усунення помилок у бізнес-логіці та обробці даних.
Налаштування середовища	Використання Docker та Laravel Sail для стабільного локального середовища розробки та тестування.
Взаємодія з базою даних	Робота з MySQL через ORM Eloquent. Можливість візуального управління базою даних через TablePlus.