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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення для моніторингу персональних активів, побудованого на основі мікросервісної архітектури з використанням платформи Node.js

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис)	Ковалишин Т. Е. (прізвище та ініціали)
Керівник (підпис)	Коноваленко І. В. (прізвище та ініціали)
Нормоконтроль (підпис)	(прізвище та ініціали)
Завідувач кафедри (підпис)	Петрик М. Р. (прізвище та ініціали)
Рецензент (підпис)	(прізвище та ініціали)

Тернопіль 2026

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

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

Кафедра програмної інженерії
(повна назва кафедри)

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

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Ковалишину Тимофію Едуардовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення для моніторингу персональних активів, побудованого на основі мікросервісної архітектури з використанням платформи Node.js

Керівник роботи Коноваленко Ігор Володимирович, канд. техн. наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Потреба у розробці серверної системи для моніторингу персональних активів, вихідний код проєкту та результати тестування API-запитів.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
Аналіз потреб користувача та вимог до системи моніторингу персональних активів
Проєктування та реалізація мікросервісної системи на платформі Node.js
Перевірка працездатності, тестування API та розгортання програмної системи
Безпечні умови праці під час розробки та тестування програмного забезпечення
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Use Case діаграма програмної системи моніторингу персональних активів
Схема загальної структури системи з використанням API Gateway
Схема контейнеризованого середовища Docker Compose

Скріншоти тестування API-запитів у Postman

6. Консультанти розділів роботи

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

7. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	**Примітка**
1	Ознайомлення із завданням кваліфікаційної роботи		
2	Збір і аналіз матеріалів за темою роботи		
3	Аналіз предметної області та програмних аналогів		
4	Формування вимог і сценаріїв роботи системи		
5	Проєктування мікросервісної архітектури та REST API		
6	Розробка сервісу авторизації користувача		
7	Розробка сервісу фінансових транзакцій		
8	Розробка сервісу персональних активів		
9	Налаштування MongoDB, Docker Compose та середовища запуску		
10	Тестування API-запитів у Postman		
11	Написання основних розділів пояснювальної записки		
12	Написання розділу про безпечні умови праці		
13	Оформлення висновків, джерел і додатків		
14	Захист кваліфікаційної роботи		

Студент _____ Ковалишин Т. Е.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коноваленко І. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Ковалишин Т. Е. Розробка програмного забезпечення для моніторингу персональних активів, побудованого на основі мікросервісної архітектури з використанням платформи Node.js, ТНТУ ім. Івана Пулюя, Тернопіль 2026.

Пояснювальна записка містить 15 рисунків, 12 таблиць, 27 джерел та 2 додатки – загалом 92 сторінки.

Об’єкт кваліфікаційної роботи: процес розробки програмного забезпечення для моніторингу персональних активів і фінансових операцій користувача.

Мета: створити програмну систему для обліку доходів, витрат і персональних активів користувача з використанням мікросервісної архітектури, платформи Node.js, REST API, MongoDB та механізму JWT-авторизації.

Сучасні програмні системи для обліку особистих фінансів і майнових активів дають змогу впорядкувати дані про доходи, витрати, баланс і цінне майно користувача. Особливої актуальності набуває розробка серверних рішень, у яких фінансова інформація зберігається структуровано, а доступ до неї захищається засобами автентифікації. У межах цієї роботи розглянуто проблему ручного обліку персональних активів, проаналізовано вимоги до програмної системи, визначено сценарії взаємодії користувача та обґрунтовано доцільність використання мікросервісної архітектури.

У кваліфікаційній роботі розроблено мікросервісну систему, що складається з API Gateway, Auth Service, Transaction Service та Asset Service. API Gateway забезпечує єдину точку входу до системи та перенаправлення запитів до відповідних сервісів. Auth Service реалізує реєстрацію користувача, вхід до системи, хешування пароля та формування JWT-токена. Transaction Service забезпечує створення доходів і витрат, роботу з фінансовими транзакціями та розрахунок балансу. Asset Service відповідає за додавання персональних активів, збереження їхніх характеристик і формування зведення за категоріями.

Для збереження даних використано MongoDB, а взаємодію з базою даних організовано через Mongoose. Розгортання програмної системи виконано за

допомогою Docker Compose, що дозволяє запускати API Gateway, три мікросервіси та окремі MongoDB-контейнери як єдине середовище. Перевірку працездатності виконано через Postman. Було протестовано реєстрацію користувача, логін, отримання JWT-токена, додавання доходу, додавання витрати, перегляд балансу, створення активу та отримання зведення активів за категоріями.

Ключові слова: персональні активи, фінансові транзакції, мікросервісна архітектура, Node.js, Express.js, MongoDB, Mongoose, REST API, API Gateway, JWT, Docker Compose, Postman, авторизація, баланс, облік доходів і витрат.

ABSTRACT

Kovalyshyn T. E. Development of software for monitoring personal assets based on microservice architecture using the Node.js platform, Ternopil Ivan Puluj National Technical University, Ternopil 2026.

The explanatory note contains 15 figures, 12 tables, 27 references and 2 appendices – 92 pages in total.

Object of the qualification work: the process of developing software for monitoring personal assets and financial transactions of a user.

Purpose: to create a software system for accounting income, expenses and personal assets of a user using microservice architecture, the Node.js platform, REST API, MongoDB and JWT authorization mechanism.

Modern software systems for accounting personal finances and property assets make it possible to organize data on income, expenses, balance and valuable user property. The development of server-side solutions in which financial information is stored in a structured form and access to it is protected by authentication tools is becoming especially relevant. Within this work, the problem of manual accounting of personal assets was considered, the requirements for the software system were analyzed, user interaction scenarios were defined, and the feasibility of using microservice architecture was substantiated.

In the qualification work, a microservice system consisting of API Gateway, Auth Service, Transaction Service and Asset Service was developed. API Gateway provides a single entry point to the system and redirects requests to the appropriate services. Auth Service implements user registration, login, password hashing and JWT token generation. Transaction Service provides the creation of income and expense records, processing of financial transactions and balance calculation. Asset Service is responsible for adding personal assets, storing their characteristics and generating a summary by categories.

MongoDB was used for data storage, while database interaction was organized through Mongoose. The deployment of the software system was implemented using Docker Compose, which makes it possible to run API Gateway, three microservices and

separate MongoDB containers as a single environment. The system functionality was tested using Postman. User registration, login, JWT token generation, income creation, expense creation, balance retrieval, asset creation and asset summary retrieval by categories were tested.

Keywords: personal assets, financial transactions, microservice architecture, Node.js, Express.js, MongoDB, Mongoose, REST API, API Gateway, JWT, Docker Compose, Postman, authorization, balance, income and expense accounting.

ПЕРЕЛІК СКРОЧЕНЬ

ПЗ – Програмне забезпечення

КРБ – Кваліфікаційна робота бакалавра

БД – База даних

API – Інтерфейс прикладного програмування (Application Programming Interface)

REST – Архітектурний стиль взаємодії вебсервісів (Representational State Transfer)

HTTP – Протокол передавання гіпертексту (HyperText Transfer Protocol)

JSON – JavaScript Object Notation (формат обміну даними)

JWT – JSON Web Token (токен для авторизації користувача)

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

Node.js – Платформа для виконання JavaScript на сервері

Express.js – Фреймворк Node.js для створення вебсерверів та API

MongoDB – Документоорієнтована база даних

Mongoose – Бібліотека для роботи з MongoDB у середовищі Node.js

API Gateway – Шлюз API, єдина точка входу до мікросервісної системи

Auth Service – Сервіс реєстрації, входу та авторизації користувача

Transaction Service – Сервіс обліку доходів, витрат і розрахунку балансу

Asset Service – Сервіс обліку персональних активів і зведення за категоріями

Docker – Платформа для контейнеризації програмного забезпечення

Docker Compose – Інструмент для запуску кількох контейнерів як єдиного середовища

Postman – Інструмент для тестування REST API

.env – Файл для зберігання конфігураційних змінних

ЗМІСТ

АНОТАЦІЯ 4
ABSTRACT 6
ПЕРЕЛІК СКРОЧЕНЬ 8
ВСТУП 11
РОЗДІЛ 1. АНАЛІЗ ПОТРЕБ КОРИСТУВАЧА ТА ВИМОГ ДО СИСТЕМИ МОНІТОРИНГУ ПЕРСОНАЛЬНИХ АКТИВІВ 13
1.1 Особливості обліку персональних активів, доходів і витрат у цифрових фінансових системах 13
1.2 Проблеми ручного ведення фінансових записів і контролю майнових активів користувача 15
1.3 Порівняльний аналіз програмних аналогів для обліку особистих фінансів і активів 17
1.4 Формування функціональних і нефункціональних вимог до розроблюваної системи 19
1.5 Сценарії взаємодії користувача із системою моніторингу активів 21
1.6 Постановка задачі та визначення меж програмного продукту 23
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ МІКРОСЕРВІСНОЇ СИСТЕМИ НА ПЛАТФОРМІ NODE.JS 26
2.1 Обґрунтування вибору мікросервісної архітектури для системи персонального фінансового моніторингу 26
2.2 Побудова загальної структури програмної системи з використанням API Gateway 29
2.3 Реалізація сервісу авторизації користувача та захисту маршрутів за допомогою JWT 32
2.4 Розробка сервісу фінансових транзакцій для обліку доходів, витрат і балансу 35
2.5 Розробка сервісу персональних активів і формування зведення за категоріями 39

2.6 Проєктування моделей MongoDB, REST API та контейнеризованого середовища запуску 42
РОЗДІЛ 3. ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ, ТЕСТУВАННЯ API ТА РОЗГОРТАННЯ ПРОГРАМНОЇ СИСТЕМИ 46
3.1 Підготовка середовища запуску мікросервісів і баз даних через Docker Compose 46
3.2 Перевірка реєстрації, входу користувача та отримання JWT-токена 49
3.3 Тестування захищених API-запитів до сервісів транзакцій і активів 52
3.4 Перевірка створення доходів, витрат і автоматичного розрахунку балансу 55
3.5 Тестування додавання персонального активу та отримання зведення за категоріями 59
3.6 Оцінка результатів тестування, стабільності роботи та можливостей подальшої підтримки системи 62
РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 67
4.1 Моделювання та прогнозування небезпечних ситуацій 67
4.2 Вимоги ергономіки до організації робочого місця оператора ПК 69
ВИСНОВКИ 72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 74
ДОДАТКИ 77
ДОДАТОК А 78
ДОДАТОК Б 90

ВСТУП

Особисті фінанси та майнові активи частіше потребують не епізодичного, а системного контролю. Користувач може мати кілька джерел доходу, заощадження, транспорт, техніку, але інформація про них часто зберігається розрізнено: у нотатках, таблицях, банківських застосунках чи взагалі в пам’яті. У такому форматі складно оцінити фінансовий стан, побачити різницю між доходами й витратами та зрозуміти загальну вартість активів. Через це виникає потреба у програмному продукті, який поєднує облік фінансових операцій і персонального майна.

Актуальність теми пов’язана з тим, що прості способи обліку не забезпечують автоматичного розрахунку балансу, захищеного доступу до даних і зручного поділу функцій системи. Для зберігання персональної фінансової інформації важливо мати авторизацію користувача, ізоляцію даних, зрозумілий REST API та можливість розширення. У цій КРБ розробляється серверна система на платформі Node.js, побудована за мікросервісним підходом. Окремо реалізовано сервіси авторизації, фінансових транзакцій і персональних активів, а взаємодія проходить через API Gateway. Це дозволяє розділити відповідальність між компонентами та не змішувати бізнес-логіку.

Метою роботи є розроблення програмного забезпечення для моніторингу персональних активів, доходів і витрат користувача на основі мікросервісної архітектури з використанням платформи Node.js.

Для досягнення мети передбачено виконати завдання, що відповідають змісту основних розділів КРБ: проаналізувати предметну область обліку персональних активів і фінансових операцій; визначити проблеми ручного ведення доходів, витрат і майнових записів; розглянути програмні аналоги та сформувати вимоги до системи; описати сценарії взаємодії користувача із серверним API; спроєктувати архітектуру з API Gateway, окремими сервісами та MongoDB-базами; реалізувати реєстрацію, вхід, JWT-захист, транзакції, баланс, додавання активів і зведення за категоріями; перевірити роботу запитів у Postman.

Об’єктом роботи є процес програмної підтримки обліку персональних

фінансових операцій і майнових активів користувача. Предметом роботи є засоби проєктування та реалізації серверної мікросервісної системи для моніторингу персональних активів із застосуванням Node.js, Express.js, MongoDB, Mongoose, JWT, bcryptjs, Docker Compose та REST API.

Під час виконання роботи використано методи аналізу предметної області, порівняння програмних аналогів, моделювання сценаріїв використання, проєктування REST API, поділу системи на мікросервіси, моделювання даних у MongoDB, контейнеризації середовища запуску та функціонального тестування API-запитів. Перевірка працездатності виконувалася через Postman із тестуванням реєстрації, входу, додавання доходу, витрати, перегляду балансу, створення активу та отримання зведення.

Особливістю розробки є поєднання двох напрямів обліку: фінансових транзакцій і персональних активів. У межах системи Transaction Service відповідає за доходи, витрати й баланс, Asset Service — за збереження активів і групування їх за категоріями, Auth Service — за автентифікацію та видачу JWT-токена, а API Gateway забезпечує єдину точку входу. Додатковою рисою є використання окремих MongoDB-контейнерів для сервісів.

Практична значущість роботи полягає в можливості використання створеної серверної частини як основи для персональної системи фінансового моніторингу. Розроблене програмне забезпечення дозволяє зареєструвати користувача, виконати вхід, захистити запити токеном, додати дохід або витрату, отримати баланс, зберегти актив і переглянути його у зведенні. Надалі така система може бути доповнена клієнтським інтерфейсом, звітами та підтримкою валют.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі подано аналіз предметної області, аналогів, вимог і сценаріїв використання. У другому розділі описано проєктування та реалізацію мікросервісної системи. У третьому розділі наведено тестування API, запуск середовища та перевірку працездатності. У четвертому розділі розглянуто безпечну організацію роботи розробника, ергономіку робочого місця та профілактику перевтоми.

РОЗДІЛ 1. АНАЛІЗ ПОТРЕБ КОРИСТУВАЧА ТА ВИМОГ ДО СИСТЕМИ МОНІТОРИНГУ ПЕРСОНАЛЬНИХ АКТИВІВ

Перший розділ присвячено аналізу потреб користувача, який веде облік доходів, витрат і персональних активів. Розгляд предметної області, програмних аналогів, вимог і сценаріїв взаємодії дає основу для подальшого проєктування серверної системи.

1.1 Особливості обліку персональних активів, доходів і витрат у цифрових фінансових системах

Облік персональних активів і фінансових операцій є основою для розуміння фактичного фінансового стану користувача. Якщо доходи та витрати показують рух коштів за певний період, то активи відображають майно або цінності, які мають вартість і можуть впливати на загальну оцінку добробуту. До таких активів можна віднести транспорт, нерухомість, електроніку, ювелірні вироби, інвестиційні об’єкти та інші матеріальні або нематеріальні цінності. У межах цієї КРБ облік активів розглядається не як бухгалтерська система для підприємства, а як персональний інструмент, орієнтований на окремого користувача.

Цифрові фінансові системи відрізняються від ручного обліку тим, що дані в них не просто зберігаються, а обробляються автоматично. Після додавання доходу або витрати система може одразу перерахувати баланс, відфільтрувати записи за типом, пов’язати їх із конкретним користувачем і повернути результат через API. Для активів важливо не лише зберегти назву та вартість, а й указати категорію, валюту, дату придбання, опис і активний статус. Це дає змогу надалі формувати зведення, наприклад визначати загальну кількість активів і сумарну вартість за категоріями [3, 4].

У розроблюваній системі персональні фінансові дані поділяються на кілька логічних груп. Такий поділ важливий, оскільки операції з доходами й витратами мають іншу природу, ніж записи про майно. Транзакція описує подію, яка змінює баланс користувача, а актив описує об’єкт, що зберігає вартість у часі. Через це для

них доцільно використовувати різні моделі даних і різні сервіси. Окремо виділяється авторизація, адже доступ до фінансової інформації має бути обмежений лише власником облікового запису [7, 8].

Для кращого розуміння структури предметної області доцільно узагальнити основні типи даних, які використовуються у програмній системі.

Таблиця 1.1 – Основні групи даних у системі моніторингу персональних активів

Група даних	Приклади даних	Роль у системі
Дані користувача	ім'я користувача, email, пароль у захищеному вигляді	забезпечують реєстрацію, вхід до системи та прив'язку фінансових записів до конкретного користувача
Фінансові транзакції	тип операції, сума, категорія, опис, дата створення	використовуються для обліку доходів, витрат і подальшого розрахунку поточного балансу
Персональні активи	назва активу, категорія, вартість, валюта, дата придбання, опис	дають змогу зберігати інформацію про майно користувача та оцінювати його загальну вартість
Аналітичні зведення	загальний дохід, загальні витрати, баланс, кількість активів, вартість активів за категоріями	допомагають швидко отримати підсумкову інформацію без ручних розрахунків

Наведена таблиця показує, що система моніторингу персональних активів не обмежується простим списком записів. Вона повинна об'єднувати ідентифікацію користувача, облік фінансових операцій, збереження майнових об'єктів і формування підсумкових даних. Саме тому в роботі обрано мікросервісний підхід: кожна група функцій може бути реалізована в окремому компоненті, а взаємодія з ними виконується через єдину точку входу [8, 9].

Особливістю предметної області є чутливість інформації. Дані про доходи, витрати та активи не повинні бути відкритими для сторонніх користувачів, тому в програмній системі передбачено автентифікацію та перевірку JWT-токена під час виконання захищених запитів. У результаті кожна транзакція або актив зберігається з прив'язкою до ідентифікатора користувача, що зменшує ризик

змішування даних між різними обліковими записами [13, 14].

Отже, цифровий облік персональних активів і фінансових операцій потребує не лише базового збереження інформації, а й продуманої структури даних, автоматичних розрахунків, захищеного доступу та можливості подальшого розширення. Саме ці особливості визначають вимоги до програмного продукту, який розробляється в межах КРБ [1, 3].

1.2 Проблеми ручного ведення фінансових записів і контролю майнових активів користувача

Ручний облік доходів, витрат і майнових активів на перший погляд здається простим способом контролю особистих фінансів. Для цього можуть використовуватися паперові записи, нотатки в телефоні, електронні таблиці або окремі файли з переліком покупок і цінного майна. Такий підхід не потребує складного програмного забезпечення, однак швидко втрачає зручність, коли кількість операцій зростає. Користувачу доводиться самостійно вносити дані, перевіряти правильність сум, оновлювати залишок коштів і стежити за тим, щоб інформація про активи не була втрачена або застаріла.

Основна проблема ручного обліку полягає в розрізненості даних. Доходи можуть фіксуватися в одному документі, витрати — в іншому, а інформація про транспорт, техніку чи інші активи — в окремих нотатках. У результаті важко отримати повну картину фінансового стану без додаткових розрахунків. Наприклад, навіть за наявності таблиці з доходами й витратами потрібно окремо підсумовувати значення, перевіряти формули та вручну порівнювати отримані результати з фактичними записами. Якщо дані про активи не пов’язані з фінансовими операціями, оцінка загального майнового стану стає ще менш точною [5, 6].

Ще однією проблемою є висока ймовірність помилок. Під час ручного введення можна пропустити операцію, вказати неправильну суму, повторити запис або не оновити дату. Такі помилки не завжди помітні одразу, але вони впливають

на підсумковий баланс. У цифровій системі частину цих дій можна автоматизувати: тип транзакції задається явно, сума зберігається у визначеному форматі, а баланс розраховується на основі всіх записів користувача. Це зменшує залежність від ручних підрахунків і робить результат більш послідовним [4, 6].

Окрему увагу потрібно звернути на безпеку. Персональні фінансові записи містять чутливу інформацію, тому зберігання її у відкритих файлах або незахищених нотатках створює ризики. У розроблюваній системі доступ до транзакцій і активів передбачено лише після автентифікації, а запити до захищених маршрутів виконуються з використанням JWT-токена. Це важливо, оскільки кожен запис повинен належати конкретному користувачу і не бути доступним стороннім особам [13, 14].

Для узагальнення основних недоліків ручного підходу доцільно подати їх у вигляді таблиці.

Таблиця 1.2 – Проблеми ручного обліку персональних фінансів і активів

Проблема	**Прояв під час ручного обліку**	**Врахування в розроблюваній системі**
Розрізненість даних	доходи, витрати й активи можуть зберігатися в різних файлах, нотатках або таблицях	дані логічно поділено між сервісами, а доступ до основних функцій організовано через єдиний API Gateway
Ризик помилок у підрахунках	баланс потрібно розраховувати вручну або через формули, які легко пошкодити чи неправильно налаштувати	баланс формується автоматично на основі створених транзакцій користувача
Складність оновлення інформації	записи про активи можуть втрачати актуальність, якщо вручну не змінювати їхню вартість, опис або статус	для активів передбачено окрему модель із категорією, вартістю, валютою, датою придбання та активним статусом
Недостатній захист доступу	фінансові записи можуть зберігатися у відкритому вигляді або бути доступними стороннім особам	захищені маршрути перевіряють JWT-токен і прив'язують транзакції та активи до конкретного користувача
Обмежена масштабованість	додавання нових можливостей у ручні таблиці або нотатки ускладнює їхню структуру	мікросервісний підхід дає змогу окремо розвивати авторизацію, транзакції та облік активів

Наведена таблиця показує, що ручний облік створює не одну окрему проблему, а цілий набір обмежень. Найбільш суттєвими є відсутність автоматичних підрахунків, слабкий захист персональних даних і складність підтримання

актуальності записів. Саме ці недоліки стали підставою для створення серверної системи, у якій облік транзакцій, активів і доступ користувача реалізовано як окремі логічні частини [1, 3].

Отже, ручне ведення фінансових записів може бути прийнятним лише для невеликої кількості простих операцій. Для системного контролю персональних активів потрібне програмне рішення, яке забезпечує структуроване збереження даних, автоматичний розрахунок балансу, захищений доступ і можливість подальшого розширення функціональності [3, 4].

1.3 Порівняльний аналіз програмних аналогів для обліку особистих фінансів і активів

Для визначення вимог до програмної системи важливо розглянути наявні рішення, які вже використовуються для контролю особистих фінансів. Такі застосунки зазвичай орієнтовані на фіксацію витрат, планування бюджету, перегляд звітів і роботу з кількома рахунками. Частина з них підтримує синхронізацію з банками, спільні гаманці або розширену аналітику. Водночас більшість аналогів подаються як готові клієнтські продукти, а не як відкрита серверна система, у якій можна детально простежити архітектуру, API, структуру сервісів і моделі даних [3, 5].

У межах КРБ аналіз аналогів потрібен не для повного копіювання їхніх функцій, а для виділення корисних підходів, які варто врахувати під час розробки власного програмного продукту. Наприклад, для системи моніторингу персональних активів важливими є облік доходів і витрат, розрахунок балансу, поділ записів за категоріями, захищений доступ користувача та можливість зберігати інформацію про активи окремо від щоденних транзакцій. Саме ці критерії стали основою для порівняння [3, 4].

Для аналізу було обрано поширені програмні продукти у сфері персональних фінансів: Wallet by BudgetBakers, Spendee, Monefy та Money Manager. Wallet орієнтований на бюджетування, облік витрат і синхронізацію з банківськими

рахунками. Spendee поєднує контроль витрат, рахунків, електронних гаманців, бюджетів і спільних фінансів. Monefy робить акцент на швидкому внесенні витрат і простому щоденному контролі бюджету. Money Manager підтримує облік фінансових операцій, звіти, перегляд даних за періодами та роботу з активами.

Для більш наочного порівняння можливостей аналогів доцільно подати їх у вигляді таблиці.

Таблиця 1.3 – Порівняння програмних аналогів для обліку особистих фінансів і активів

Програмний аналог	Основні можливості	Сильні сторони	Обмеження з погляду теми КРБ
Wallet by BudgetBakers	бюджетування, облік витрат, банківська синхронізація, звіти	зручна аналітика, підтримка різних фінансових рахунків, орієнтація на комплексний контроль фінансів	не є навчальним серверним проєктом із відкритою мікросервісною структурою
Spendee	витрати, доходи, бюджети, гаманці, спільні фінанси, кілька валют	зручний користувацький інтерфейс, підтримка спільного обліку та різних фінансових джерел	основний акцент зроблено на клієнтському застосунку, а не на демонстрації власного REST API
Monefy	швидке додавання витрат, контроль бюджету, порівняння витрат і доходів	простота використання, мінімальна кількість дій для внесення запису	обмежена увага до окремого обліку майнових активів і серверної архітектури
Money Manager	облік доходів і витрат, звіти, календар, бюджети, робота з активами	широкий набір функцій для персонального фінансового контролю	функціональність готового застосунку складно адаптувати до навчальної задачі побудови мікросервісів
Розроблювана система	реєстрація, логін, JWT-захист, транзакції, баланс, активи, зведення за категоріями	прозора структура серверної частини, поділ на Auth Service, Transaction Service, Asset Service та API Gateway	відсутній готовий клієнтський інтерфейс, оскільки основний акцент зроблено на серверній реалізації

Наведене порівняння показує, що існуючі аналоги мають сильні сторони у сфері користувацького інтерфейсу, звітності та готових фінансових інструментів. Водночас розроблювана система має іншу спрямованість: вона демонструє побудову серверної частини з чітким поділом відповідальності між мікросервісами. У ній окремо реалізовано авторизацію, роботу з доходами й витратами, розрахунок балансу та збереження персональних активів [7 – 9].

Отже, аналіз аналогів дозволив визначити функції, які є важливими для системи персонального фінансового моніторингу: захищений доступ, структурований облік операцій, підтримка категорій, автоматичні підсумки та можливість розширення. У розроблюваному програмному продукті ці ідеї

адаптовано до задачі створення мікросервісної системи на Node.js, де основна увага приділяється не зовнішньому інтерфейсу, а логіці серверної взаємодії, REST API та організації даних [8, 11, 12].

1.4 Формування функціональних і нефункціональних вимог до розроблюваної системи

Після аналізу предметної області та програмних аналогів було визначено вимоги до системи моніторингу персональних активів. Їх формування є важливим етапом, оскільки саме вимоги задають межі майбутнього програмного продукту, визначають його поведінку та допомагають відокремити основні функції від другорядних [3, 5]. Для цієї КРБ система не розглядається як повноцінний банківський або бухгалтерський застосунок. Її призначення полягає у створенні серверної основи, яка дає змогу користувачу реєструватися, входити в систему, захищено працювати з фінансовими операціями та вести облік персональних активів.

Функціональні вимоги описують дії, які система повинна виконувати. У розроблюваному проєкті до них належать реєстрація користувача, автентифікація, видача JWT-токена, перевірка захищених маршрутів, створення доходів і витрат, перегляд списку транзакцій, розрахунок балансу, додавання активів і формування зведення за категоріями [3, 13]. Кожна з цих функцій пов’язана з окремою частиною мікросервісної структури: Auth Service відповідає за користувача, Transaction Service — за фінансові операції, Asset Service — за активи, а API Gateway забезпечує єдину точку входу [8, 9].

Нефункціональні вимоги визначають не конкретні дії, а якість роботи системи. Для серверного програмного продукту важливими є захищеність даних, зрозумілість API, ізольованість сервісів, стабільність запуску, коректна обробка помилок і можливість подальшого розвитку [4]. Оскільки система працює з персональною фінансовою інформацією, особливе значення має контроль доступу. Запити до транзакцій і активів повинні виконуватися тільки після перевірки JWT-

токена, а записи мають прив’язуватися до конкретного користувача [13, 14].

Для узагальнення сформованих вимог їх доцільно подати у вигляді таблиці, де окремо показано зміст кожної вимоги та її реалізацію в межах програмної системи.

Таблиця 1.4 – Функціональні та нефункціональні вимоги до системи моніторингу персональних активів

Тип вимоги	Зміст вимоги	Реалізація у програмній системі
Функціональна	реєстрація нового користувача	Auth Service створює обліковий запис, зберігає email, ім’я користувача та пароль у хешованому вигляді
Функціональна	вхід користувача в систему	після перевірки email і пароля формується JWT-токен для подальших запитів
Функціональна	облік доходів і витрат	Transaction Service створює транзакції з типом income або expense, сумою, категорією та описом
Функціональна	розрахунок балансу	баланс формується на основі суми доходів і витрат конкретного користувача
Функціональна	облік персональних активів	Asset Service зберігає назву, категорію, вартість, валюту, дату придбання та опис активу
Нефункціональна	захист персональних даних	захищені маршрути перевіряють JWT-токен і не допускають доступу без авторизації
Нефункціональна	модульність системи	авторизація, транзакції та активи винесені в окремі сервіси
Нефункціональна	зручність запуску	сервіси та бази даних запускаються через Docker Compose

У таблиці показано, що вимоги напряму пов’язані з архітектурою системи. Функціональні можливості не зосереджені в одному файлі або одному серверному модулі, а розподілені між сервісами відповідно до їх призначення. Це спрощує розуміння структури проєкту та полегшує перевірку кожної частини окремо [7 – 9].

Отже, сформовані вимоги стали основою для подальшого проєктування програмної системи. Вони визначили склад сервісів, структуру моделей даних, перелік API-запитів і підхід до захисту маршрутів. У результаті система отримує чітку логіку роботи: користувач проходить автентифікацію, після чого може працювати з власними транзакціями, балансом і активами через захищені REST API-запити [11 – 13].

1.5 Сценарії взаємодії користувача із системою моніторингу активів

Сценарії взаємодії користувача із системою потрібні для того, щоб описати практичну поведінку програмного продукту до переходу безпосередньо до реалізації. Якщо вимоги визначають, що саме повинна виконувати система, то сценарії показують послідовність дій: від реєстрації користувача до отримання підсумкових даних про фінансовий стан і активи [1, 3]. Для розроблюваної системи це особливо важливо, оскільки її функції розподілено між кількома сервісами, а доступ до них здійснюється через API Gateway [8, 9].

Головним учасником системи є користувач, який веде облік власних доходів, витрат і персональних активів. Спочатку він створює обліковий запис, після чого виконує вхід і отримує JWT-токен. Цей токен використовується під час звернення до захищених маршрутів, зокрема при створенні транзакцій, перегляді балансу, додаванні активів і отриманні зведення. Без автентифікації такі дії не повинні виконуватися, оскільки фінансові записи мають бути прив’язані до конкретного користувача [13, 14].

У межах програмної системи можна виділити кілька основних сценаріїв. Перший сценарій пов’язаний з реєстрацією: користувач передає ім’я, email і пароль, а Auth Service створює обліковий запис. Другий сценарій — вхід до системи, під час якого перевіряються облікові дані та формується JWT-токен. Наступні сценарії стосуються роботи з фінансовими операціями: додавання доходу, додавання витрати та перегляд балансу. Окремо виділяються сценарії для активів: створення запису про актив і перегляд зведення за категоріями [3, 6].

Для узагальнення основних сценаріїв взаємодії доцільно подати їх у вигляді таблиці.

Таблиця 1.5 – Основні сценарії взаємодії користувача із системою

Сценарій	Дії користувача	Реакція системи	Очікуваний результат
Реєстрація користувача	вводяться ім'я користувача, email і пароль	Auth Service перевіряє дані, хешує пароль і створює обліковий запис	користувач отримує повідомлення про успішну реєстрацію та токен
Вхід до системи	вводяться email і пароль	Auth Service перевіряє пароль і формує JWT-токен	користувач отримує доступ до захищених API-запитів
Додавання доходу	передається тип income, сума, категорія й опис	Transaction Service створює запис транзакції для поточного користувача	дохід зберігається в базі даних і враховується в балансі
Додавання витрати	передається тип expense, сума, категорія й опис	Transaction Service створює запис витрати для поточного користувача	витрата зберігається та зменшує підсумковий баланс
Перегляд балансу	виконується захищений запит на отримання балансу	Transaction Service підсумовує доходи й витрати користувача	система повертає загальний дохід, загальні витрати та баланс
Додавання активу	передаються назва, категорія, вартість, валюта, опис і дата придбання	Asset Service створює запис активу з прив'язкою до користувача	актив зберігається в базі даних і може бути врахований у зведенні
Перегляд зведення активів	виконується захищений запит на отримання summary	Asset Service групує активи за категоріями	система повертає кількість активів, загальну вартість і розподіл за категоріями

У таблиці показано, що кожен сценарій має чітко визначену послідовність: дія користувача, обробка запиту відповідним сервісом і очікуваний результат. Це дає змогу не змішувати різні функції системи між собою. Авторизація не виконує розрахунок балансу, сервіс транзакцій не зберігає активи, а сервіс активів не відповідає за перевірку пароля. Такий поділ відповідає логіці мікросервісної архітектури та робить поведінку системи більш передбачуваною [7 – 9].

Окрему роль у сценаріях відіграє JWT-захист. Після входу користувач отримує токен, який додається до подальших запитів. Якщо токен відсутній або некоректний, доступ до транзакцій і активів не надається. Це дозволяє відокремити відкриті дії, наприклад реєстрацію та логін, від захищених дій, які працюють із персональними фінансовими даними [13, 14].

Отже, сформовані сценарії відображають основний шлях користувача в системі: створення облікового запису, вхід, робота з доходами й витратами, перегляд балансу, додавання активів і отримання зведення. Саме ці сценарії надалі використовуються як основа для побудови Use Case діаграми, проєктування API-запитів і перевірки працездатності програмного продукту [3, 11, 12].

1.6 Постановка задачі та визначення меж програмного продукту

На основі аналізу предметної області, проблем ручного обліку, програмних аналогів, вимог і сценаріїв взаємодії було сформовано задачу розробки програмної системи. У межах КРБ потрібно створити серверне програмне забезпечення, яке забезпечує моніторинг персональних активів і фінансових операцій користувача. Система повинна працювати не як проста таблиця для записів, а як структурований API-продукт із реєстрацією, автентифікацією, захищеними маршрутами, обліком транзакцій, розрахунком балансу та збереженням активів [1, 3].

Головна задача полягає в побудові мікросервісної системи, де кожен компонент має власну зону відповідальності. API Gateway повинен приймати зовнішні запити та перенаправляти їх до потрібного сервісу. Auth Service має відповідати за створення користувача, перевірку облікових даних і формування JWT-токена. Transaction Service повинен забезпечувати роботу з доходами, витратами та балансом. Asset Service має зберігати персональні активи й формувати зведення за категоріями. Такий поділ дозволяє уникнути перевантаження одного серверного застосунку різними функціями та робить структуру проєкту зрозумілішою [7 – 9].

У межах програмного продукту передбачено роботу одного типу користувача — зареєстрованого користувача системи. Адміністративна панель, клієнтський вебінтерфейс, банківська синхронізація, автоматичне оновлення вартості активів і складна фінансова аналітика не входять до меж реалізації. Основний акцент зроблено на серверній частині, REST API, роботі з MongoDB, JWT-захисті та перевірці запитів через Postman [11 – 13, 17, 21]. Це відповідає темі КРБ і дозволяє детально показати саме програмну реалізацію мікросервісної архітектури.

Для чіткого визначення меж системи доцільно подати основні функції, відповідальні сервіси та очікувані результати у вигляді таблиці.

Таблиця 1.6 – Межі програмного продукту та відповідальність сервісів

Функція системи	Відповідальний компонент	Результат виконання	Належність до меж КРБ
Реєстрація користувача	Auth Service	створення облікового запису та хешування пароля	входить до реалізації
Вхід користувача	Auth Service	перевірка email і пароля, формування JWT-токена	входить до реалізації
Маршрутизація запитів	API Gateway	перенаправлення запитів до відповідних сервісів	входить до реалізації
Додавання доходу або витрати	Transaction Service	створення фінансової транзакції з прив'язкою до користувача	входить до реалізації
Розрахунок балансу	Transaction Service	повернення загального доходу, витрат і поточного балансу	входить до реалізації
Додавання персонального активу	Asset Service	збереження активу з категорією, вартістю, валютою та описом	входить до реалізації
Зведення активів	Asset Service	групування активів за категоріями та підрахунок загальної вартості	входить до реалізації
Клієнтський інтерфейс	окремий майбутній компонент	візуальна взаємодія користувача із системою	не входить до реалізації
Банківська синхронізація	зовнішні фінансові сервіси	автоматичне отримання операцій з банку	не входить до реалізації

Таблиця показує, що межі програмного продукту визначено достатньо чітко. До реалізації включено ті функції, які безпосередньо підтверджують тему роботи: мікросервісну побудову, захищений доступ, облік транзакцій, баланс і активи. Водночас функції, які потребують окремої клієнтської частини або інтеграції із зовнішніми банківськими API, залишено поза межами цієї КРБ [3, 8].

Постановка задачі також передбачає, що система повинна бути придатною для тестування через HTTP-запити. Для цього кожна ключова дія має бути представлена окремим REST API-маршрутом. Реєстрація та вхід належать до відкритих запитів, а створення транзакцій, перегляд балансу, додавання активів і отримання зведення виконуються лише після передавання JWT-токена. У результаті програмний продукт має не тільки зберігати дані, а й демонструвати контроль доступу до персональної фінансової інформації [11 – 14].

У підсумку поставлена задача зводиться до розробки серверної мікросервісної системи для моніторингу персональних активів, у якій реалізовано базовий життєвий цикл користувача: реєстрація, вхід, захищена робота з доходами й витратами, розрахунок балансу, додавання активів і перегляд їхнього зведення.

Саме ця постановка задачі є основою для подальшого проєктування архітектури, моделей даних, REST API та середовища запуску через Docker Compose [8, 11, 17 – 20].

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ МІКРОСЕРВІСНОЇ СИСТЕМИ НА ПЛАТФОРМІ NODE.JS

На етапі проєктування було визначено структуру програмної системи та розподіл функцій між її основними компонентами. У розділі описано API Gateway, сервіси авторизації, транзакцій і активів, моделі MongoDB, REST API та запуск середовища через Docker Compose.

2.1 Обґрунтування вибору мікросервісної архітектури для системи персонального фінансового моніторингу

Під час проєктування програмного забезпечення для моніторингу персональних активів потрібно було визначити, як саме організувати серверну частину системи. Для невеликого застосунку можна використати монолітний підхід, коли авторизація, фінансові операції, активи, маршрути й робота з базою даних зосереджені в одному серверному проєкті. Такий варіант є простішим на початку, але зі збільшенням кількості функцій він поступово ускладнює підтримку коду. Зміни в одній частині програми можуть впливати на інші частини, а логіка різних процесів змішується в межах спільної структури [5, 7].

Для теми цієї КРБ більш доцільним є мікросервісний підхід, оскільки система має кілька чітко відокремлених напрямів роботи. Авторизація користувача, облік доходів і витрат, розрахунок балансу та збереження активів виконують різні задачі й працюють із різними моделями даних. Через це їх логічно винести в окремі сервіси. Такий підхід відповідає загальній ідеї мікросервісної архітектури, де складна система складається з невеликих компонентів, кожен з яких має власну відповідальність і взаємодіє з іншими через API [8, 9].

У розроблюваному програмному продукті було виділено чотири основні серверні компоненти. API Gateway приймає зовнішні запити та перенаправляє їх до потрібного сервісу. Auth Service відповідає за реєстрацію, вхід користувача, хешування пароля та формування JWT-токена. Transaction Service обробляє доходи, витрати й баланс. Asset Service працює з персональними активами, їх

категоріями, вартістю та зведенням. Завдяки такому поділу кожен сервіс має зрозумілу роль, а структура проєкту стає ближчою до реальних підходів серверної розробки [7 – 9].

Важливою причиною вибору мікросервісної архітектури є ізоляція даних. У системі передбачено окремі MongoDB-бази для авторизації, транзакцій та активів. Це дозволяє не змішувати облікові записи користувачів із фінансовими операціями або записами про майно. Для навчального програмного продукту такий поділ є корисним, оскільки наочно показує, як різні сервіси можуть мати власні сховища даних і при цьому працювати як єдина система через API Gateway [8, 9, 17].

Ще однією перевагою є можливість незалежної підтримки компонентів. Наприклад, зміни в логіці розрахунку балансу можуть вноситися в Transaction Service без безпосереднього втручання в Auth Service. Аналогічно, розширення категорій активів або додавання нових полів до моделі активу не потребує зміни коду авторизації. Це не означає, що сервіси повністю не залежать один від одного, адже транзакції та активи використовують ідентифікатор користувача з JWT. Однак така залежність є контрольованою і проходить через чітко визначений механізм автентифікації [7, 8, 13].

Для обґрунтування вибору архітектури доцільно порівняти монолітний і мікросервісний підходи саме в контексті розроблюваної системи.

Таблиця 2.1 – Порівняння монолітної та мікросервісної архітектури для системи моніторингу активів

Критерій	Монолітна архітектура	Мікросервісна архітектура в розроблюваній системі
Організація коду	усі функції розміщуються в одному серверному застосунку	авторизація, транзакції, активи та gateway винесені в окремі компоненти
Робота з даними	одна база або спільна структура даних для різних модулів	окремі MongoDB-бази для користувачів, транзакцій і активів
Підтримка та зміни	зміна одного модуля може зачіпати всю систему	кожен сервіс можна змінювати відповідно до його зони відповідальності
Масштабування	масштабування відбувається для всього застосунку одразу	потенційно можна окремо масштабувати сервіс, який має більше навантаження
Тестування API	перевіряється один загальний серверний застосунок	кожен сервіс і маршрути через API Gateway можуть перевірятися окремо
Відповідність темі КРБ	підходить для простого застосунку, але слабше розкриває архітектурний аспект	краще демонструє поділ системи, REST API, контейнеризацію та взаємодію сервісів

У таблиці показано, що мікросервісний підхід краще відповідає задачам цієї КРБ. Він дозволяє не лише реалізувати потрібні функції, а й показати архітектурну організацію програмного продукту. Для системи, де є авторизація, транзакції та активи, такий поділ виглядає природним, оскільки кожна група функцій має власну логіку й власні дані [7 – 9].

Окремо потрібно зазначити, що мікросервісна архітектура має і певні складнощі. Вона потребує налаштування кількох серверів, узгодження портів, змінних середовища, контейнерів і взаємодії між сервісами. У проєкті ці питання вирішуються через Docker Compose, який запускає API Gateway, Auth Service, Transaction Service, Asset Service і три MongoDB-контейнери. Це спрощує підготовку середовища та дає змогу запускати всю систему як єдиний набір пов’язаних компонентів [19, 20].

У результаті вибір мікросервісної архітектури є обґрунтованим для розроблюваного програмного забезпечення. Вона дозволяє чітко розділити відповідальність сервісів, ізолювати дані, організувати доступ через API Gateway, реалізувати JWT-захист і підготувати систему до подальшого розвитку. Для КРБ

такий підхід також має навчальну цінність, оскільки демонструє не лише написання окремих API-запитів, а й побудову повноцінної серверної структури на платформі Node.js [8, 13, 15].

2.2 Побудова загальної структури програмної системи з використанням API Gateway

Загальна структура програмної системи для моніторингу персональних активів побудована навколо ідеї єдиної точки входу. У розроблюваному проєкті такою точкою є API Gateway, який приймає HTTP-запити від зовнішнього клієнта та перенаправляє їх до відповідних мікросервісів. Це дозволяє приховати внутрішню структуру системи від користувача API: замість прямого звернення до окремих сервісів використовується один базовий порт і єдина адреса входу. Такий підхід спрощує організацію маршрутів, полегшує тестування через Postman і робить систему більш впорядкованою [8, 9, 12].

У проєкті API Gateway реалізовано як окремий Node.js/Express.js-сервіс, що працює на порту 3000. Його основне призначення полягає в маршрутизації запитів до трьох внутрішніх сервісів: Auth Service, Transaction Service та Asset Service. Auth Service відповідає за реєстрацію, вхід і перевірку JWT-токена. Transaction Service забезпечує створення доходів і витрат, отримання списку транзакцій та розрахунок балансу. Asset Service працює з персональними активами, їх категоріями та підсумковим зведенням. Для кожного з цих компонентів передбачено окремий порт і власне підключення до MongoDB [15 – 18].

Використання API Gateway є доцільним у мікросервісній системі, оскільки клієнтська частина не повинна знати адреси всіх внутрішніх сервісів. Якщо структура системи зміниться, наприклад буде додано новий сервіс або змінено внутрішній порт, зовнішній формат взаємодії може залишитися стабільним. У межах КРБ це важливо не лише з технічного погляду, а й з погляду пояснення архітектури: API Gateway відокремлює зовнішній рівень доступу від внутрішньої бізнес-логіки [7 – 9].

У розробленій системі маршрути побудовано за логічними групами. Запити, що починаються з /api/auth, передаються до сервісу авторизації. Запити з префіксом /api/transactions спрямовуються до сервісу фінансових транзакцій. Запити з префіксом /api/assets обробляються сервісом персональних активів. Крім цього, у gateway передбачено службовий маршрут /health, який дає змогу перевірити доступність самого шлюзу. Така структура робить API зрозумілим і передбачуваним, оскільки назва маршруту одразу вказує на частину системи, яка його обробляє [11, 12, 16].

У цій структурі зовнішній префікс /api/auth, /api/transactions або /api/assets використовується API Gateway для визначення цільового мікросервісу, а внутрішній префікс /auth, /transactions або /assets належить маршрутизатору самого сервісу. Через це під час тестування через API Gateway використовуються повні маршрути на зразок /api/auth/auth/register, /api/transactions/transactions та /api/assets/assets. Такий підхід дозволяє зберегти однакову структуру внутрішніх маршрутів мікросервісів як під час роботи через gateway, так і під час їх окремого локального запуску [12, 16, 21].

Для перенаправлення запитів використовується механізм проксування. У практичній реалізації це означає, що API Gateway не виконує бізнес-логіку авторизації, транзакцій або активів самостійно. Він приймає запит, визначає його префікс і передає далі до відповідного сервісу. У результаті кожен мікросервіс залишається відповідальним лише за власну предметну область. Це відповідає принципу розмежування відповідальності, який є одним з ключових для побудови підтримуваних серверних систем [7 – 9].

Щоб узагальнити взаємодію компонентів, доцільно подати структуру системи у вигляді таблиці.

Таблиця 2.2 – Загальна структура програмної системи з використанням API Gateway

Компонент системи	Порт	Основне призначення	Приклади маршрутів через API Gateway
API Gateway	3000	приймає зовнішні запити, виконує проксування, застосовує обмеження частоти запитів	`/health`, `/api/auth`, `/api/transactions`, `/api/assets`
Auth Service	3001	забезпечує реєстрацію, вхід користувача та перевірку JWT	`/api/auth/auth/register`, `/api/auth/auth/login`, `/api/auth/auth/verify`
Transaction Service	3002	обробляє доходи, витрати, CRUD-операції з транзакціями та розрахунок балансу	`/api/transactions/transactions`, `/api/transactions/transactions/balance`
Asset Service	3003	забезпечує роботу з персональними активами та формування зведення за категоріями	`/api/assets/assets`, `/api/assets/assets/summary`
MongoDB для авторизації	27017	зберігає облікові записи користувачів	використовується Auth Service
MongoDB для транзакцій	27018	зберігає фінансові операції користувача	використовується Transaction Service
MongoDB для активів	27019	зберігає персональні активи користувача	використовується Asset Service

У таблиці показано, що API Gateway не замінює внутрішні сервіси, а організовує доступ до них. Зовнішній клієнт звертається до порту 3000, тоді як усередині системи запити розподіляються між сервісами на портах 3001, 3002 і 3003. Окремі MongoDB-контейнери не мають прямого API для користувача, але забезпечують збереження даних для відповідних сервісів. Завдяки цьому структура системи залишається логічно розділеною [8, 17, 20].

Додатково в API Gateway використано обмеження частоти запитів. Це не є повноцінною системою захисту від усіх видів атак, але дозволяє зменшити ризик надмірного навантаження на серверну частину. Для навчального програмного продукту така реалізація показує базовий рівень контролю доступу до API. Разом із JWT-захистом у внутрішніх сервісах це створює зрозумілу схему: gateway керує входом до системи, а сервіси перевіряють право користувача на виконання захищених операцій [13, 14].

Важливим елементом побудови структури є Docker Compose. Саме він дозволяє запускати API Gateway, три мікросервіси та три бази даних як пов’язаний набір контейнерів. Усі компоненти підключаються до спільної bridge-мережі, тому

сервіси можуть звертатися один до одного за назвами контейнерів. Це спрощує запуск системи та усуває потребу вручну налаштовувати кожен сервер окремо [19, 20].

У результаті загальна структура програмної системи має чіткий поділ на рівень входу, рівень бізнес-логіки та рівень збереження даних. API Gateway забезпечує приймання і перенаправлення запитів, мікросервіси виконують прикладну логіку, а MongoDB-контейнери відповідають за окремі сховища даних. Така організація добре відповідає темі КРБ, оскільки демонструє не лише роботу окремих REST API-запитів, а й повну взаємодію компонентів у мікросервісній системі на платформі Node.js [8, 11, 15, 17].

2.3 Реалізація сервісу авторизації користувача та захисту маршрутів за допомогою JWT

Сервіс авторизації є одним із ключових компонентів програмної системи, оскільки саме він відповідає за створення облікового запису користувача, перевірку вхідних даних і формування токена доступу. У системі моніторингу персональних активів авторизація має особливе значення, бо транзакції, баланс і записи про активи належать конкретному користувачу та не повинні бути доступними стороннім особам [13, 14]. Через це роботу з користувачами було винесено в окремий мікросервіс Auth Service, який виконує лише функції, пов’язані з автентифікацією та перевіркою особи користувача [8, 9].

Auth Service реалізовано на платформі Node.js із використанням Express.js. Для збереження даних користувачів застосовано MongoDB і Mongoose [15 – 18]. Модель користувача містить основні поля, потрібні для ідентифікації в системі: ім’я користувача, email і пароль. Email використовується як унікальний ідентифікатор під час входу, а пароль не зберігається у відкритому вигляді. Перед записом у базу даних він хешується за допомогою бібліотеки bcryptjs [22]. Такий підхід знижує ризик компрометації паролів, оскільки навіть у разі отримання доступу до бази даних пароль не може бути прочитаний як звичайний текст [14].

Реєстрація користувача виконується через окремий API-маршрут. Користувач передає ім’я, email і пароль, після чого сервіс перевіряє наявність облікового запису з таким email. Якщо користувач ще не зареєстрований, створюється новий запис у базі даних. Після успішного створення облікового запису система повертає повідомлення про успішну реєстрацію, токен і базові дані користувача без відкритого відображення пароля. Така відповідь потрібна для того, щоб клієнтська частина або Postman могли одразу використовувати отриманий JWT-токен у подальших запитах [12, 13, 21].

Вхід до системи реалізовано через перевірку email і пароля. Користувач передає облікові дані, а Auth Service знаходить відповідний запис у MongoDB і порівнює введений пароль із хешем, який зберігається в базі. Якщо перевірка успішна, формується JWT-токен. У токені закладається ідентифікатор користувача, який надалі використовується іншими сервісами для прив’язки транзакцій і активів до конкретного облікового запису [13, 22]. Це дозволяє Transaction Service та Asset Service не працювати безпосередньо з паролями, а отримувати лише підтверджений ідентифікатор користувача.

JWT використовується як механізм передачі підтвердження автентифікації між клієнтом і серверними сервісами. Після входу користувач додає токен до заголовка захищених запитів. Middleware у сервісах транзакцій та активів перевіряє коректність токена, розшифровує його за допомогою спільного секретного ключа та записує ідентифікатор користувача в об’єкт запиту. Якщо токен відсутній або неправильний, доступ до захищеного маршруту не надається. Такий підхід дає змогу відокремити відкриті маршрути, пов’язані з реєстрацією та входом, від маршрутів, які працюють із персональними фінансовими даними [13, 14].

Для узагальнення структури Auth Service та пов’язаних із ним елементів доцільно подати основні маршрути й функції у вигляді таблиці.

Таблиця 2.3 – Основні елементи Auth Service та їх призначення

Елемент реалізації	Маршрут або компонент	Призначення	Результат роботи
Реєстрація користувача	POST /api/auth/auth/register	створення нового облікового запису на основі username, email і password	користувач зберігається в MongoDB, пароль хешується, повертається JWT-токен
Вхід до системи	POST /api/auth/auth/login	перевірка email і пароля зареєстрованого користувача	у разі успішної перевірки формується токен і повертаються дані користувача
Перевірка токена	GET /api/auth/auth/verify	підтвердження дійсності JWT-токена	система повертає результат перевірки авторизації
Модель користувача	User	опис структури користувача в MongoDB	зберігаються username, email, хешований password і службові часові поля
Хешування пароля	bcryptjs	перетворення пароля перед збереженням	пароль не потрапляє до бази даних у відкритому вигляді
Формування токена	jsronethoken	створення JWT на основі ідентифікатора користувача	інші сервіси можуть визначити поточного користувача через токен
Захист маршрутів	middleware protect	перевірка токена в сервісах транзакцій та активів	запити без коректного токена не отримують доступ до персональних даних

У таблиці показано, що Auth Service не обмежується лише маршрутом входу. Він охоплює повний базовий цикл автентифікації: створення користувача, безпечне збереження пароля, видачу токена та можливість перевірки авторизованого доступу. Окремо важливо, що захист маршрутів використовується вже в інших сервісах, але залежить від токена, який формується саме сервісом авторизації [13, 14].

У розробленому проєкті Auth Service не зберігає транзакції та активи, оскільки це не належить до його відповідальності. Його задача полягає в тому, щоб підтвердити особу користувача та надати іншим сервісам надійний спосіб визначити, кому належать фінансові записи. Завдяки цьому Transaction Service може створювати доходи й витрати з прив’язкою до userId, а Asset Service — зберігати активи лише для поточного користувача. Це робить взаємодію сервісів більш упорядкованою та відповідає принципу розділення бізнес-логіки [7 – 9].

Особливістю реалізації є те, що JWT-захист працює без збереження стану сесії на сервері. Після формування токена сервіс не створює окремий серверний запис сесії, а перевірка доступу відбувається на основі самого токена та секретного ключа. Для мікросервісної системи це зручно, оскільки різні сервіси можуть

перевіряти токен самостійно, не звертаючись щоразу до Auth Service [13]. Водночас такий підхід потребує правильного зберігання секретного ключа в змінних середовища та однакового налаштування JWT у сервісах, які виконують перевірку [14].

У результаті сервіс авторизації забезпечує базовий рівень безпеки для всієї системи моніторингу персональних активів. Реєстрація, логін, хешування пароля, формування JWT і перевірка захищених маршрутів утворюють єдиний механізм доступу до персональних даних. Саме цей механізм дозволяє відокремити користувачів один від одного та гарантувати, що фінансові транзакції, баланс і активи обробляються лише в межах відповідного облікового запису [13, 14].

2.4 Розробка сервісу фінансових транзакцій для обліку доходів, витрат і балансу

Transaction Service є окремим мікросервісом, який відповідає за облік фінансових операцій користувача. У межах системи моніторингу персональних активів цей сервіс виконує роль центрального компонента для роботи з доходами, витратами та поточним балансом. Його винесення в окремий сервіс є логічним, оскільки фінансові операції мають власну модель даних, власні маршрути та окремі правила обробки [8, 9]. Auth Service лише підтверджує користувача через JWT, а Transaction Service уже працює з конкретними транзакціями, що належать цьому користувачу [13].

Сервіс реалізовано на основі Node.js та Express.js. Для збереження фінансових записів використовується MongoDB у поєднанні з Mongoose [15 – 18]. Модель транзакції містить поля userId, type, amount, category, description і date. Поле userId потрібне для прив’язки запису до конкретного користувача, а поле type визначає характер операції: income для доходу або expense для витрати. Сума операції зберігається в полі amount, категорія описує призначення запису, а поле description дозволяє додати коротке пояснення. Дата використовується для фіксації часу фінансової операції та подальшого впорядкування записів [17, 18].

Важливою особливістю Transaction Service є те, що всі основні маршрути захищені middleware protect. Перед виконанням запиту сервіс перевіряє JWT-токен, отримує з нього ідентифікатор користувача та записує його в об’єкт запиту. Завдяки цьому під час створення або перегляду транзакцій система працює не з абстрактними фінансовими записами, а саме з даними поточного користувача. Такий підхід зменшує ризик змішування інформації між різними обліковими записами [13, 14].

Створення транзакції відбувається через POST-запит. Користувач передає тип операції, суму, категорію та опис. Якщо створюється дохід, у полі type зазначається income, а якщо витрата — expense. Після перевірки токена сервіс додає до запису userId, отриманий із JWT, і зберігає транзакцію в MongoDB. У тестових запитах через Postman було перевірено додавання доходу із сумою 25000 у категорії «Зарплата» та витрати із сумою 3500 у категорії «Їжа» [21]. Обидва запити повернули відповідь про успішне створення транзакції, що підтверджує коректну роботу маршруту додавання.

Окрім створення нових записів, сервіс підтримує перегляд транзакцій, отримання окремого запису, оновлення та видалення. Такі CRUD-операції потрібні для повноцінного керування фінансовими даними. Наприклад, користувач може переглянути всі власні транзакції або відфільтрувати їх за типом. Це корисно, коли потрібно окремо проаналізувати тільки доходи або тільки витрати. У межах серверної логіки запити завжди обмежуються userId, тому користувач не отримує доступ до чужих фінансових записів [10, 13, 14].

Окремий маршрут передбачено для розрахунку балансу. Його робота базується на підсумовуванні всіх транзакцій поточного користувача за двома групами: доходи та витрати. Сервіс обчислює totalIncome, totalExpense і різницю між ними, яка повертається як balance. Такий підхід дозволяє уникнути ручних підрахунків і забезпечує актуальність балансу після кожної нової операції. У перевірці через Postman після додавання доходу 25000 і витрати 3500 система повернула totalIncome: 25000, totalExpense: 3500 і balance: 21500, що відповідає очікуваному результату [21].

Для наочного відображення роботи Transaction Service доцільно подати схему проходження запиту від користувача до бази даних. Така схема показує, що сервіс фінансових транзакцій отримує запити через API Gateway, виконує перевірку JWT-токена, визначає поточного користувача та залежно від типу запиту створює транзакцію або розраховує баланс.

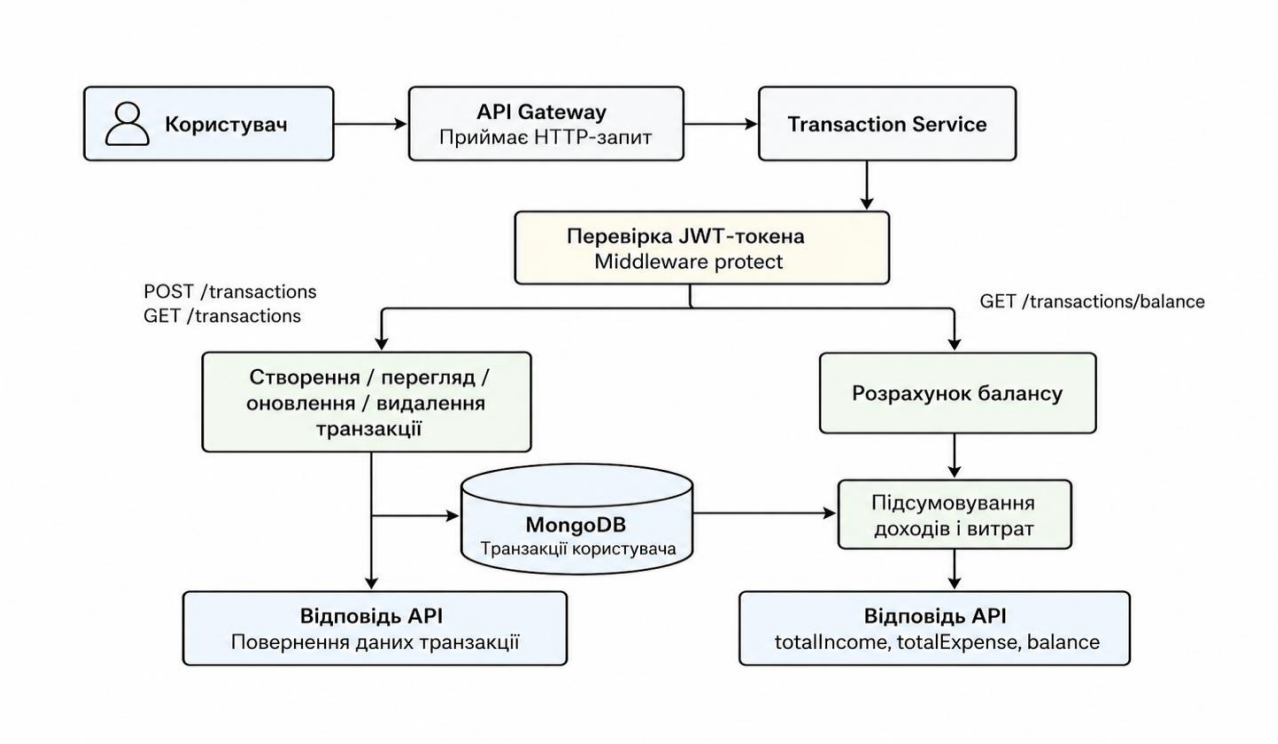

Рисунок 2.1 – Схема обробки запитів у Transaction Service

На рисунку 2.1 показано послідовність обробки запитів у сервісі фінансових транзакцій. Спочатку користувач надсилає запит через API Gateway, після чого він передається до Transaction Service. Далі middleware перевіряє JWT-токен і визначає ідентифікатор користувача. Якщо виконується створення доходу або витрати, сервіс зберігає нову транзакцію в MongoDB. Якщо виконується запит на баланс, система підсумовує доходи й витрати користувача та повертає значення totalIncome, totalExpense і balance [12, 13, 17].

Для узагальнення реалізованих можливостей Transaction Service доцільно подати основні маршрути та їх призначення у вигляді таблиці.

Таблиця 2.4 – Основні маршрути Transaction Service та їх призначення

Маршрут через API Gateway	Метод	Призначення	Результат виконання
/api/transactions/transactions	POST	створення доходу або витрати з прив'язкою до користувача	у MongoDB зберігається нова транзакція з типом "income" або "expense"
/api/transactions/transactions	GET	отримання списку транзакцій поточного користувача	система повертає фінансові записи, що належать авторизованому користувачу
/api/transactions/transactions/:id	GET	перегляд однієї транзакції за ідентифікатором	повертається конкретний запис, якщо він належить поточному користувачу
/api/transactions/transactions/:id	PUT	оновлення фінансової операції	змінюються дані вибраної транзакції
/api/transactions/transactions/:id	DELETE	видалення фінансової операції	транзакція видаляється з бази даних
/api/transactions/transactions/balance	GET	розрахунок балансу на основі доходів і витрат	повертаються totalIncome, totalExpense і balance

У таблиці показано, що Transaction Service охоплює повний набір операцій для роботи з фінансовими записами. Найважливішим для теми КРБ є маршрут балансу, оскільки він перетворює набір окремих доходів і витрат на підсумковий фінансовий показник. Без такого маршруту користувач отримував би лише список операцій, але не мав би швидкого розуміння поточного стану коштів.

Реалізація сервісу транзакцій також демонструє перевагу мікросервісного поділу. Фінансова логіка не змішується з реєстрацією користувача або збереженням активів. Transaction Service працює лише з операціями руху коштів і балансом, а інші частини системи виконують власні задачі. У результаті сервіс можна розвивати окремо: додати фільтрацію за датами, групування за категоріями, статистику витрат або експорт фінансових записів без прямого втручання в Auth Service чи Asset Service [8, 9].

У підсумку Transaction Service забезпечує практичну основу для фінансової частини програмного продукту. Він дозволяє створювати доходи й витрати, зберігати їх у базі даних, захищати доступ через JWT, прив'язувати записи до користувача та автоматично розраховувати баланс. Саме цей сервіс робить систему не просто сховищем персональних активів, а інструментом для контролю поточних

фінансових операцій [13, 17, 18].

2.5 Розробка сервісу персональних активів і формування зведення за категоріями

Asset Service є окремим мікросервісом, призначеним для роботи з персональними активами користувача. У межах програмної системи цей сервіс доповнює фінансовий облік, оскільки доходи й витрати показують рух коштів, а активи відображають майно або цінності, які мають певну вартість. До таких об’єктів у розробленій системі можуть належати транспорт, нерухомість, електроніка, ювелірні вироби, інвестиційні активи та інші категорії. Саме тому функціональність Asset Service не дублює Transaction Service, а розв’язує окрему задачу — збереження і групування майнових записів користувача [8, 9].

Сервіс персональних активів реалізовано на платформі Node.js із використанням Express.js, MongoDB і Mongoose [15 – 18]. Для опису активу створено модель Asset, яка містить поля userId, name, category, value, currency, description, purchaseDate та isActive. Поле userId забезпечує зв’язок запису з конкретним користувачем, а name зберігає назву активу. Категорія дає змогу віднести актив до певної групи, наприклад vehicle, electronics, investment або other. Вартість і валюта потрібні для підрахунку загальної цінності активів, а дата придбання та опис уточнюють зміст запису. Поле isActive дозволяє позначати, чи залишається актив актуальним для користувача [17, 18].

Як і у випадку з транзакціями, маршрути Asset Service захищені JWT-токеном. Перед створенням або переглядом активів виконується middleware-перевірка, після якої з токена отримується ідентифікатор користувача. Завдяки цьому новий актив зберігається не як загальний запис у базі даних, а як запис, прив’язаний до конкретного облікового запису. Такий підхід особливо важливий для системи персонального моніторингу, оскільки дані про майно користувача є конфіденційними та не повинні змішуватися з даними інших користувачів [13, 14].

Створення активу виконується через POST-запит до відповідного маршруту.

Користувач передає назву, категорію, вартість, валюту, опис і дату придбання. У тестовому сценарії через Postman було додано актив Toyota Camry 2020 у категорії vehicle із вартістю 850000 у валюті UAH. Після виконання запиту система повернула повідомлення про успішне додавання активу та об’єкт створеного запису. Це підтверджує, що сервіс коректно приймає вхідні дані, додає до них ідентифікатор користувача та зберігає актив у MongoDB [17, 21].

Asset Service також підтримує перегляд списку активів, отримання одного активу, оновлення та видалення записів. Додатково передбачено фільтрацію за категорією та активним статусом. Це потрібно для того, щоб користувач міг працювати не лише з повним переліком майна, а й з окремими групами. Наприклад, у майбутньому можна швидко переглянути тільки транспортні засоби або лише активні записи, які ще мають значення для поточного фінансового стану [10, 16].

Найважливішою аналітичною функцією сервісу є формування зведення активів за категоріями. На відміну від звичайного перегляду списку, маршрут summary повертає підсумкову інформацію: загальну кількість активів, сумарну вартість і групування за категоріями. У перевірці через Postman після додавання одного активу система повернула totalAssets: 1, totalValue: 850000, а в об’єкті byCategory було сформовано категорію vehicle із кількістю 1 та сумарною вартістю 850000. Такий результат показує, що сервіс не лише зберігає майнові записи, а й виконує базову обробку даних для швидкого отримання підсумків [17, 18, 21].

Для наочного відображення логіки роботи Asset Service доцільно подати схему проходження запиту від користувача до формування зведення. Перед побудовою рисунка наведено дані, які визначають основні етапи цієї схеми.

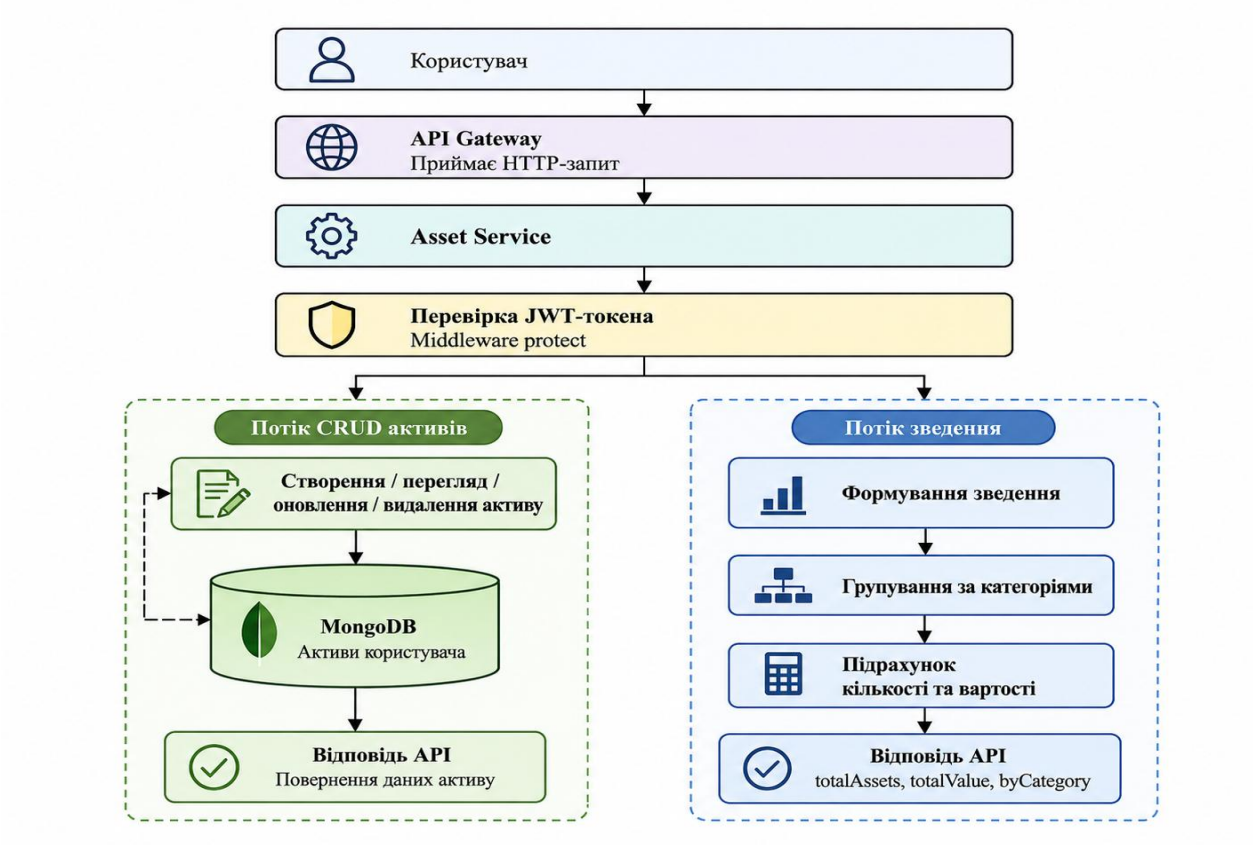

Рисунок 2.2 – Схема роботи Asset Service і формування зведення активів

На рисунку 2.2 показано, що Asset Service отримує запити через API Gateway, після чого виконується перевірка JWT-токена. Якщо запит стосується створення або зміни активу, сервіс працює з окремим записом у MongoDB. Якщо виконується запит на зведення, сервіс отримує активи поточного користувача, групує їх за категоріями та підраховує загальну кількість і вартість. Така схема допомагає зрозуміти різницю між звичайними CRUD-операціями та аналітичним маршрутом summary [12, 13, 17, 18].

Реалізація Asset Service має важливе значення для всієї системи моніторингу персональних активів. Без цього сервісу програмний продукт обмежувався б лише обліком доходів і витрат, тобто показував би рух коштів, але не відображав би майновий стан користувача. Додавання активів розширює можливості системи й дозволяє зберігати дані про об’єкти, які мають довгострокову цінність [3, 4].

У результаті Asset Service забезпечує окремий рівень роботи з майновими даними. Він приймає захищені запити, перевіряє користувача через JWT, зберігає

активи в MongoDB, підтримує фільтрацію та формує зведення за категоріями. Це робить систему більш повною, оскільки користувач може контролювати не лише поточні фінансові операції, а й загальну вартість персональних активів [13, 17, 18].

2.6 Проєктування моделей MongoDB, REST API та контейнеризованого середовища запуску

Для завершення проєктування серверної частини програмної системи потрібно було узгодити між собою три важливі складові: структуру даних у MongoDB, набір REST API-маршрутів і середовище запуску мікросервісів. Саме ці елементи забезпечують практичну роботу системи після реалізації окремих сервісів. Якщо Auth Service, Transaction Service та Asset Service відповідають за власну бізнес-логіку, то моделі даних, API та Docker Compose формують основу, завдяки якій ці сервіси можуть стабільно працювати разом [7 – 9].

Для збереження даних у системі використано MongoDB, оскільки вона добре підходить для роботи з документною структурою записів. У межах проєкту було спроєктовано три основні моделі: User, Transaction та Asset. Модель користувача використовується в Auth Service і містить дані, потрібні для реєстрації та входу. Модель транзакції використовується в Transaction Service і зберігає фінансові операції користувача. Модель активу належить до Asset Service і описує майнові об’єкти, які мають вартість, категорію, валюту та додатковий опис [17, 18].

Важливою особливістю реалізації є те, що кожен сервіс працює з окремою базою даних. Auth Service підключається до MongoDB для користувачів, Transaction Service — до бази транзакцій, а Asset Service — до бази активів. Це відповідає логіці мікросервісної архітектури, де сервіс повинен володіти власними даними та не залежати від прямого доступу до сховища іншого компонента. У результаті облікові записи, фінансові операції та майнові записи не змішуються в одній спільній базі [8, 9, 17].

REST API у проєкті побудовано за ресурсним принципом. Для авторизації використовуються маршрути реєстрації, входу та перевірки токена. Для транзакцій

передбачено створення, перегляд, оновлення, видалення й розрахунок балансу. Для активів реалізовано CRUD-операції, фільтрацію та отримання зведення за категоріями. Зовнішній доступ до цих маршрутів проходить через API Gateway, тому користувач або тестовий інструмент Postman працює з єдиною точкою входу на порту 3000, а не з кожним сервісом окремо [11, 12, 21].

Усі захищені маршрути в сервісах транзакцій та активів використовують JWT-перевірку. Це означає, що структура API не обмежується лише набором HTTP-адрес, а включає також правила доступу. Відкритими залишаються реєстрація та логін, адже користувач ще не має токена. Натомість додавання доходу, витрати, активу або перегляд підсумкових даних вимагають передавання токена в заголовку запиту. Завдяки цьому API працює не просто як набір маршрутів, а як захищений інтерфейс доступу до персональних даних [13, 14].

Контейнеризоване середовище запуску реалізовано за допомогою Docker Compose. У файлі конфігурації описано окремі контейнери для API Gateway, Auth Service, Transaction Service, Asset Service та трьох MongoDB-баз. Такий підхід дозволяє запускати всю систему однією командою, не налаштовуючи кожен сервіс вручну. Кожен Node.js-сервіс має власний порт, власні змінні середовища та підключення до відповідної бази даних. MongoDB-контейнери мають окремі volume, що дозволяє зберігати дані між перезапусками контейнерів [19, 20].

Для кращого розуміння контейнеризованої структури системи доцільно подати схему взаємодії сервісів і баз даних. Перед побудовою рисунка наведено дані, які відображають основні компоненти Docker Compose середовища.

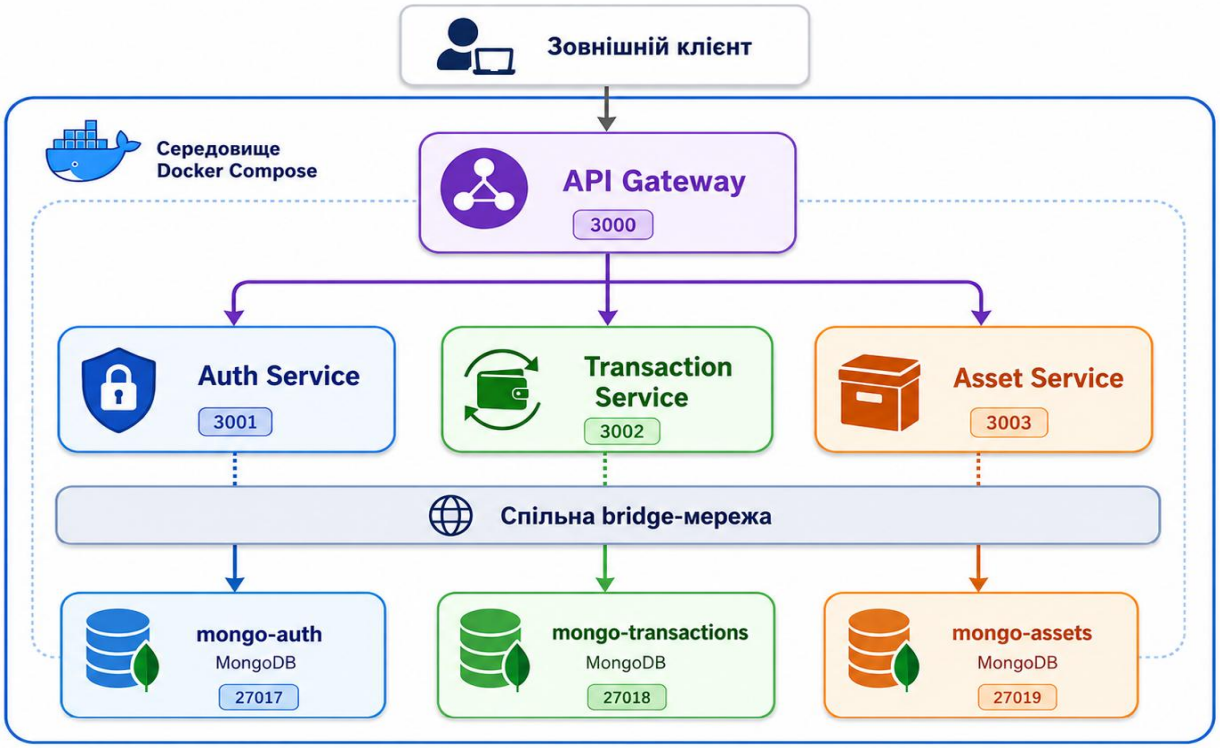

Рисунок 2.3 – Схема контейнеризованого середовища Docker Compose

На рисунку 2.3 відображено, що зовнішній клієнт звертається до API Gateway на порту 3000, після чого запити розподіляються між Auth Service, Transaction Service та Asset Service. Кожен із цих сервісів підключається до власного MongoDB-контейнера: сервіс авторизації працює з mongo-auth, сервіс транзакцій — з mongo-transactions, а сервіс активів — з mongo-assets. Усі компоненти об’єднуються спільною bridge-мережею Docker Compose, що дозволяє контейнерам взаємодіяти між собою за внутрішніми іменами [17, 19, 20].

Такий спосіб організації середовища має практичну перевагу для розробки й тестування. Після запуску Docker Compose всі потрібні частини системи піднімаються разом: шлюз, три мікросервіси та три бази даних. Це зменшує кількість ручних дій і допомагає уникнути ситуації, коли один сервіс запущено, а потрібна база даних або інший компонент недоступні. Крім того, структура контейнерів добре відповідає архітектурній ідеї проєкту: кожен сервіс має власну відповідальність і власне сховище [8, 9, 20].

Проєктування моделей MongoDB, REST API та Docker Compose середовища дозволило сформувати завершену серверну основу програмної системи. Моделі

визначають структуру збережених даних, API-маршрути забезпечують доступ до функцій, а контейнеризація об’єднує всі компоненти в єдину робочу систему. Розробка, тестування, розгортання та подальша підтримка програмної системи розглядаються як пов’язані етапи життєвого циклу програмного забезпечення [2]. У результаті розроблене програмне забезпечення можна запускати, перевіряти через Postman і надалі розширювати без суттєвої зміни загальної архітектури [17 – 21].

РОЗДІЛ 3. ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ, ТЕСТУВАННЯ API ТА РОЗГОРТАННЯ ПРОГРАМНОЇ СИСТЕМИ

Перевірка працездатності є необхідною частиною розробки, оскільки вона показує, чи правильно взаємодіють реалізовані сервіси. У цьому розділі подано запуск середовища та тестування основних API-запитів через Postman.

3.1 Підготовка середовища запуску мікросервісів і баз даних через Docker Compose

Перед перевіркою працездатності програмної системи потрібно підготувати середовище, у якому одночасно запускаються всі компоненти проєкту. Для мікросервісної архітектури це має особливе значення, оскільки система складається не з одного серверного застосунку, а з кількох пов’язаних частин: API Gateway, Auth Service, Transaction Service, Asset Service і трьох окремих MongoDB-контейнерів. Якщо хоча б один із цих компонентів не буде запущений або не зможе підключитися до потрібної бази даних, частина функцій системи стане недоступною [8, 9].

У розробленому програмному забезпеченні запуск середовища організовано через Docker Compose. Такий підхід дає змогу описати всі сервіси в одному конфігураційному файлі та запускати їх як єдину систему. У складі середовища передбачено чотири Node.js-контейнери: api-gateway, auth-service, transaction-service та asset-service. Окремо запускаються три MongoDB-контейнери: mongo-auth, mongo-transactions і mongo-assets. Для кожної бази даних передбачено власне сховище даних, що відповідає принципу розділення відповідальності між мікросервісами [17, 19, 20].

Підготовка середовища починається з перевірки структури проєкту та наявності необхідних конфігураційних файлів. У корені проєкту розміщено docker-compose.yml, який описує контейнери, порти, залежності, змінні середовища, MongoDB-бази та спільну мережу. Також у проєкті наявний файл .env.example,

який показує перелік змінних, потрібних для запуску: порти сервісів, URI підключення до MongoDB, секретний ключ JWT і строк дії токена. Фактичні .env файли не зберігаються в репозиторії, оскільки такі дані не повинні потрапляти до відкритого коду [13, 14, 20].

Після підготовки конфігурації середовище запускається командою Docker Compose. Під час запуску створюються контейнери сервісів, піднімаються MongoDB-бази, налаштовується спільна bridge-мережа та підключаються volume для збереження даних. API Gateway відкривається на порту 3000 і стає основною точкою входу для подальшого тестування. Auth Service працює на порту 3001, Transaction Service — на 3002, Asset Service — на 3003. MongoDB-контейнери доступні всередині мережі Docker та використовуються відповідними сервісами [19, 20].

Для наочного відображення послідовності підготовки середовища запуску доцільно подати дані для схеми, яка показує основні етапи переходу від конфігурації до готової системи.

Рисунок 3.1 – Схема підготовки середовища запуску через Docker Compose

На рисунку 3.1 показано послідовність підготовки середовища: спочатку перевіряється структура проєкту та налаштування змінних середовища, після чого Docker Compose запускає всі контейнери. Далі сервіси підключаються до спільної bridge-мережі, API Gateway стає доступним на порту 3000, а система переходить до етапу тестування через Postman. Така схема важлива для цього підпункту, оскільки вона показує не окремий API-запит, а повний процес підготовки мікросервісної системи до перевірки [20, 21].

Після запуску контейнерів потрібно переконатися, що всі компоненти працюють узгоджено. Для цього спочатку перевіряється доступність API Gateway через службовий маршрут /health. Якщо gateway відповідає успішно, можна переходити до перевірки маршрутів авторизації. Далі виконується реєстрація користувача, вхід до системи та отримання JWT-токена. Після цього перевіряються захищені маршрути Transaction Service та Asset Service. Така послідовність є логічною, оскільки сервіси транзакцій і активів не повинні приймати запити без авторизації [6, 13, 21].

Окрему роль у підготовці середовища відіграють MongoDB-контейнери. Кожен із них обслуговує тільки свій мікросервіс: mongo-auth зберігає користувачів, mongo-transactions — фінансові операції, mongo-assets — персональні активи. Це дозволяє під час тестування чітко розуміти, де саме зберігаються дані, які створюються через Postman. Наприклад, після реєстрації запис має з’явитися в базі авторизації, після додавання доходу або витрати — у базі транзакцій, а після створення активу — у базі активів [17, 18, 21].

У підсумку підготовлене середовище Docker Compose забезпечує повноцінний запуск усієї мікросервісної системи. Воно об’єднує API Gateway, три прикладні сервіси та три бази даних у спільну інфраструктуру, придатну для подальшого тестування. Саме після цього можна переходити до перевірки реєстрації, входу, JWT-захисту, додавання транзакцій, розрахунку балансу, створення активів і формування зведення [8, 19 – 21].

3.2 Перевірка реєстрації, входу користувача та отримання JWT-токена

Після підготовки середовища запуску було виконано перевірку базових функцій авторизації користувача. Цей етап є початковим для подальшого тестування системи, оскільки більшість маршрутів Transaction Service та Asset Service працюють лише після отримання JWT-токена. Без успішної реєстрації та входу неможливо коректно перевірити створення транзакцій, розрахунок балансу, додавання активів і формування зведення за категоріями [13, 21].

Перевірка виконувалася через Postman із використанням API Gateway як єдиної точки входу. Це означає, що запити надсилалися не напряму до Auth Service на внутрішній порт, а через адресу http://localhost:3000. Такий підхід відповідає архітектурі розробленої системи, де API Gateway приймає зовнішні HTTP-запити та перенаправляє їх до потрібного мікросервісу. У цьому випадку запити з префіксом /api/auth передаються до сервісу авторизації [8, 12, 21].

Першим було перевірено сценарій реєстрації користувача. Для цього виконано POST-запит на маршрут http://localhost:3000/api/auth/auth/register. У тілі запиту передавалися ім’я користувача, email і пароль. У тестовому прикладі було використано користувача з іменем student та email student@example.com. Пароль у тексті роботи не відтворюється як чутливе значення, оскільки для опису результату достатньо показати факт його передавання та успішного створення облікового запису [14, 21].

У відповідь система повернула статус 201 Created, повідомлення про успішну реєстрацію, JWT-токен і об’єкт користувача. Наявність токена у відповіді підтверджує, що після створення облікового запису Auth Service одразу формує засіб для подальшої авторизованої взаємодії з API. Об’єкт користувача містить ідентифікатор, ім’я та email, але не повертає пароль у відкритому вигляді. Це є важливим результатом, оскільки пароль повинен зберігатися тільки в захищеному хешованому форматі [13, 14, 22].

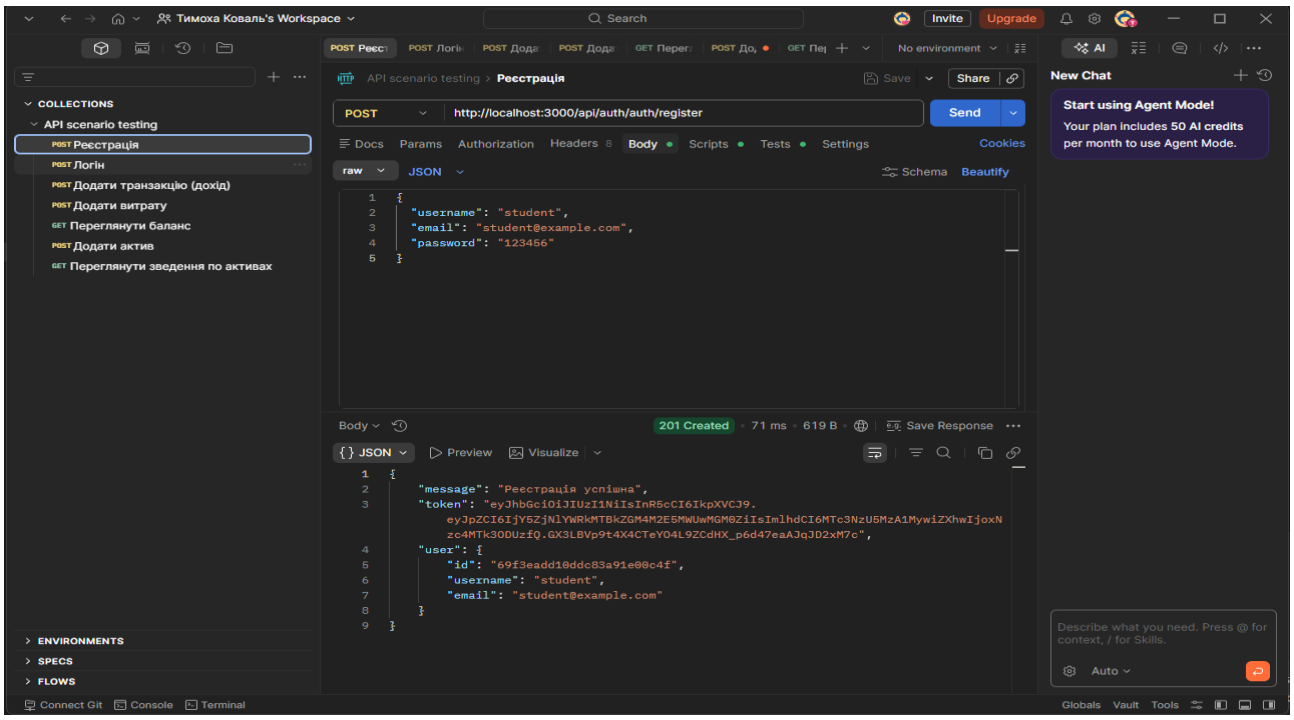

Рисунок 3.2 – Перевірка реєстрації користувача через Postman

На рисунку 3.2 показано виконання POST-запиту до маршруту реєстрації користувача. У верхній частині вікна Postman відображено адресу запиту через API Gateway, а в тілі запиту подано дані нового користувача. У нижній частині видно відповідь сервера зі статусом 201 Created, повідомленням про успішну реєстрацію, JWT-токеном і даними користувача. Це підтверджує, що Auth Service коректно створює обліковий запис і повертає результат через API Gateway [21].

Після реєстрації було перевірено сценарій входу користувача в систему. Для цього виконано POST-запит на маршрут http://localhost:3000/api/auth/auth/login. У тілі запиту передавалися email і пароль зареєстрованого користувача. Сервіс авторизації мав знайти користувача за email, перевірити пароль через механізм порівняння з хешем і в разі успішної перевірки сформувати новий JWT-токен [13, 22].

Результатом виконання запиту став статус 200 OK, повідомлення про успішний вхід, JWT-токен і дані користувача. Це підтверджує, що система не лише створює обліковий запис, а й може повторно автентифікувати користувача за введеними обліковими даними. Отриманий токен надалі використовується в заголовку Authorization для перевірки захищених маршрутів сервісів транзакцій та

активів [13, 21].

Для підтвердження успішного входу користувача в систему в роботі використовується скріншот Postman, на якому показано POST-запит до маршруту логіну та відповідь сервера з JWT-токеном.

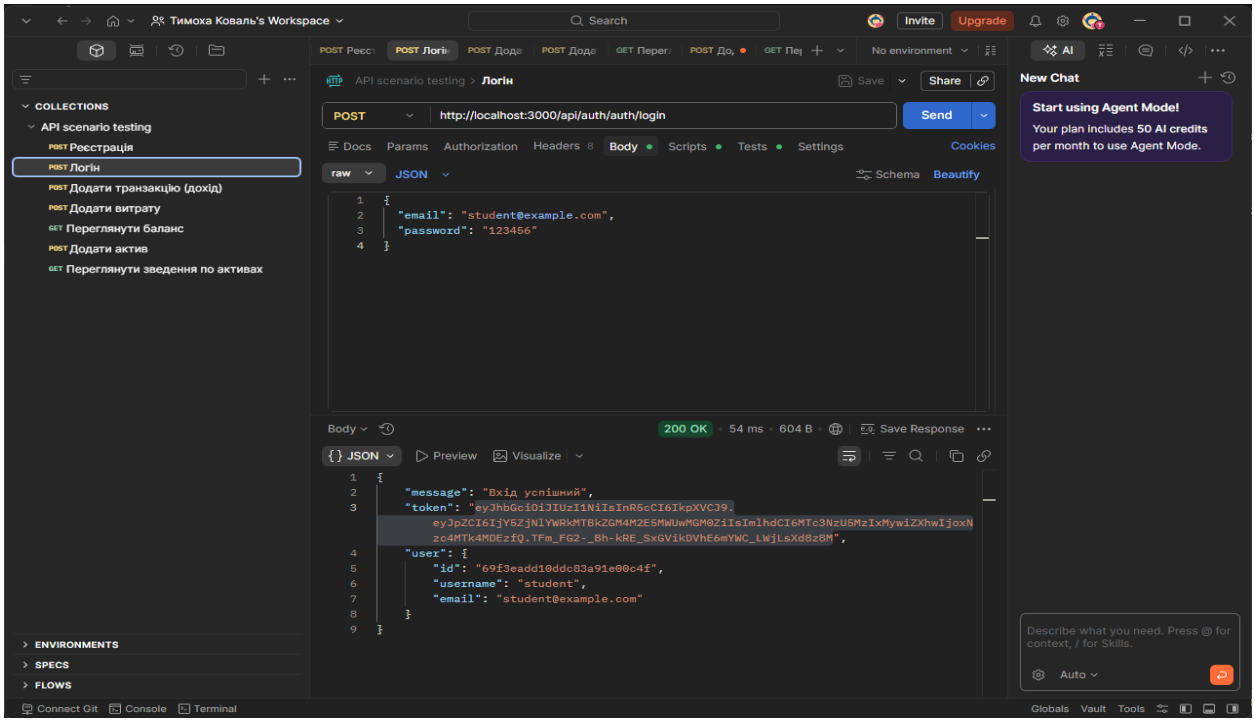

Рисунок 3.3 – Перевірка входу користувача та отримання JWT-токена через Postman

На рисунку 3.3 показано виконання POST-запиту до маршруту входу користувача. У тілі запиту передано email і пароль зареєстрованого користувача. У відповіді сервера відображено статус 200 OK, повідомлення про успішний вхід, JWT-токен і дані користувача. Отримання токена підтверджує, що Auth Service коректно перевіряє облікові дані та формує засіб доступу до захищених маршрутів системи [13, 21].

Для узагальнення результатів перевірки авторизації доцільно подати основні тестові запити у вигляді таблиці.

Таблиця 3.1 – Результати перевірки реєстрації та входу користувача

Перевірювана дія	Метод і маршрут	Передані дані	Очікуваний результат	Фактичний результат
Реєстрація користувача	POST /api/auth/auth/register	username, email, password	створення користувача та повернення JWT-токена	отримано статус 201 Created, повідомлення про успішну реєстрацію, токен і дані користувача
Вхід до системи	POST /api/auth/auth/login	email, password	перевірка облікових даних і повернення JWT-токена	отримано статус 200 OK, повідомлення про успішний вхід, токен і дані користувача

Таблиця 3.1 показує, що обидва базові сценарії авторизації виконалися успішно. Реєстрація підтвердила створення нового користувача, а логін — можливість повторного входу за вже наявними даними. В обох випадках система повернула JWT-токен, що є необхідною умовою для подальшого тестування захищених API-запитів [13, 21].

Отже, перевірка Auth Service показала коректну роботу основних механізмів автентифікації. Через API Gateway було успішно виконано реєстрацію користувача та вхід до системи. Отримання JWT-токена підтвердило готовність програмного продукту до наступного етапу тестування — перевірки захищених маршрутів, які потребують авторизованого доступу [13, 14, 21].

3.3 Тестування захищених API-запитів до сервісів транзакцій і активів

Після перевірки реєстрації та входу користувача було виконано тестування захищених API-запитів. Цей етап є важливим, оскільки Transaction Service та Asset Service працюють із персональними фінансовими даними, які не повинні бути доступними без автентифікації. Якщо користувач не має коректного JWT-токена, система не повинна дозволяти створення доходів, витрат, активів або отримання підсумкової інформації. Саме тому перевірка захисту маршрутів є обов’язковою частиною тестування працездатності програмного продукту [13, 14, 21].

У розробленій системі JWT-токен формується сервісом Auth Service після успішної реєстрації або входу користувача. Надалі цей токен передається в заголовку запиту до захищених маршрутів. Сервіси транзакцій і активів мають власний middleware protect, який перевіряє наявність токена, його коректність і можливість отримати з нього ідентифікатор користувача. Якщо перевірка проходить успішно, ідентифікатор записується в об’єкт запиту та використовується для створення або пошуку даних тільки поточного користувача [13].

Захищеними в системі є маршрути, пов’язані з фінансовими транзакціями та персональними активами. До них належать додавання доходу, додавання витрати, перегляд балансу, створення активу та отримання зведення активів за категоріями. Усі ці дії мають виконуватися тільки після входу користувача в систему. Відкритими залишаються маршрути реєстрації та логіну, оскільки саме вони потрібні для первинного отримання токена [13, 14].

Перевірка JWT-авторизації виконувалася через Postman. Спочатку було отримано токен під час входу користувача, після чого він використовувався в запитах до Transaction Service та Asset Service. У Postman для захищених запитів застосовувався розділ Authorization або відповідний заголовок із токеном. Успішне створення доходу, витрати, активу та отримання балансу підтверджує, що сервіси приймають запити з коректним JWT і правильно визначають поточного користувача [13, 21].

Для систематизації перевірки доцільно подати основні сценарії тестування JWT-захисту у вигляді таблиці.

Таблиця 3.2 – Сценарії перевірки JWT-захисту захищених маршрутів

Перевіряваний сценарій	Маршрут або група маршрутів	Умова виконання запиту	Очікувана поведінка системи	Значення для перевірки
Запит із коректним токеном до транзакцій	`/api/transactions/transactions`	у запиті передано дійсний JWT-токен	сервіс дозволяє створити або отримати транзакції поточного користувача	підтверджує роботу захисту в Transaction Service
Запит із коректним токеном до балансу	`/api/transactions/transactions/balance`	токен отримано після входу користувача	система повертає `totalIncome`, `totalExpense` і `balance`	підтверджує доступ до персональних фінансових підсумків
Запит із коректним токеном до активів	`/api/assets/assets`	у заголовку передано дійсний JWT-токен	сервіс дозволяє створити або переглянути активи користувача	підтверджує захист майнових записів
Запит із коректним токеном до зведення активів	`/api/assets/assets/summary`	токен належить авторизованому користувачу	система повертає кількість активів, загальну вартість і групування за категоріями	підтверджує роботу захищеного аналітичного маршруту
Запит без токена	захищені маршрути транзакцій або активів	заголовок авторизації відсутній	доступ до персональних даних не повинен надаватися	перевіряє базове обмеження неавторизованого доступу
Запит із некоректним токеном	захищені маршрути транзакцій або активів	передано неправильний або пошкоджений JWT	система не повинна виконувати дію з персональними даними	перевіряє стійкість механізму авторизації до помилкових токенів

У таблиці 3.2 показано, що перевірка JWT-захисту охоплює не лише успішні запити, а й ситуації, коли доступ має бути обмежений. Для КРБ особливо важливими є перші чотири сценарії, оскільки вони підтверджуються подальшими запитами в Postman: створенням доходу, створенням витрати, переглядом балансу, додаванням активу та отриманням зведення. Сам факт успішного виконання цих операцій після входу в систему означає, що токен передається коректно та приймається відповідними сервісами [14, 21].

Окремо потрібно зазначити, що JWT-захист у цій системі використовується не тільки для дозволу або заборони запиту. Він також забезпечує прив’язку записів до конкретного користувача. Після перевірки токена middleware отримує userId, який використовується під час створення транзакції або активу. У результаті фінансові операції та майнові записи не створюються як загальні дані, а зберігаються саме для користувача, який виконав авторизований запит [13, 14].

Під час тестування через Postman було важливо дотримуватися правильної послідовності дій. Спочатку виконується логін, потім із відповіді копіюється JWT-токен, після чого він додається до захищених запитів. Якщо пропустити етап отримання токена або передати його неправильно, подальші запити до транзакцій і

активів не мають виконуватися як авторизовані. Така послідовність добре відображає реальний сценарій роботи серверного API [12, 13, 21].

У підсумку перевірка захищених маршрутів показує, що механізм JWT є центральним елементом доступу до персональних фінансових даних. Auth Service видає токен, а Transaction Service та Asset Service використовують його для перевірки користувача. Це дозволяє перейти до наступних етапів тестування, де вже перевіряються конкретні функції: додавання доходів і витрат, розрахунок балансу, створення активів і формування зведення за категоріями [13, 14, 21].

3.4 Перевірка створення доходів, витрат і автоматичного розрахунку балансу

Після перевірки авторизації та захищених маршрутів було виконано тестування створення фінансових транзакцій. Цей етап підтверджує практичну роботу Transaction Service, оскільки саме через нього користувач додає доходи й витрати, які надалі використовуються для розрахунку балансу. У системі моніторингу персональних активів транзакції є основою фінансової частини, тому важливо було перевірити не лише доступність маршруту, а й правильність збереження типу операції, суми, категорії, опису та прив’язки до авторизованого користувача [6, 21].

Тестування виконувалося через Postman із використанням API Gateway. Для створення транзакцій застосовувався маршрут POST http://localhost:3000/api/transactions/transactions. Запит надсилався через порт 3000, після чого API Gateway перенаправляв його до Transaction Service. Оскільки маршрут є захищеним, перед виконанням запиту використовувався JWT-токен, отриманий після входу користувача в систему. Це дало змогу перевірити одночасно дві частини системи: роботу авторизованого доступу та створення фінансового запису [12, 13, 21].

Першим було протестовано додавання доходу. У тілі POST-запиту передавалися тип операції income, сума 25000, категорія «Зарплата» та опис

«Перша зарплата». Після надсилання запиту система повернула статус 201 Created і повідомлення про успішне створення транзакції. У відповіді було сформовано об’єкт транзакції з ідентифікатором користувача, типом операції, сумою, категорією, описом, датою створення та службовими полями MongoDB. Це підтверджує, що сервіс правильно прийняв дані, додав до них userId з JWT-токена та зберіг запис у базі даних [13, 17, 18].

Для підтвердження фактичного виконання запиту на створення доходу в роботі використовується скріншот Postman, на якому показано тіло запиту та відповідь сервера.

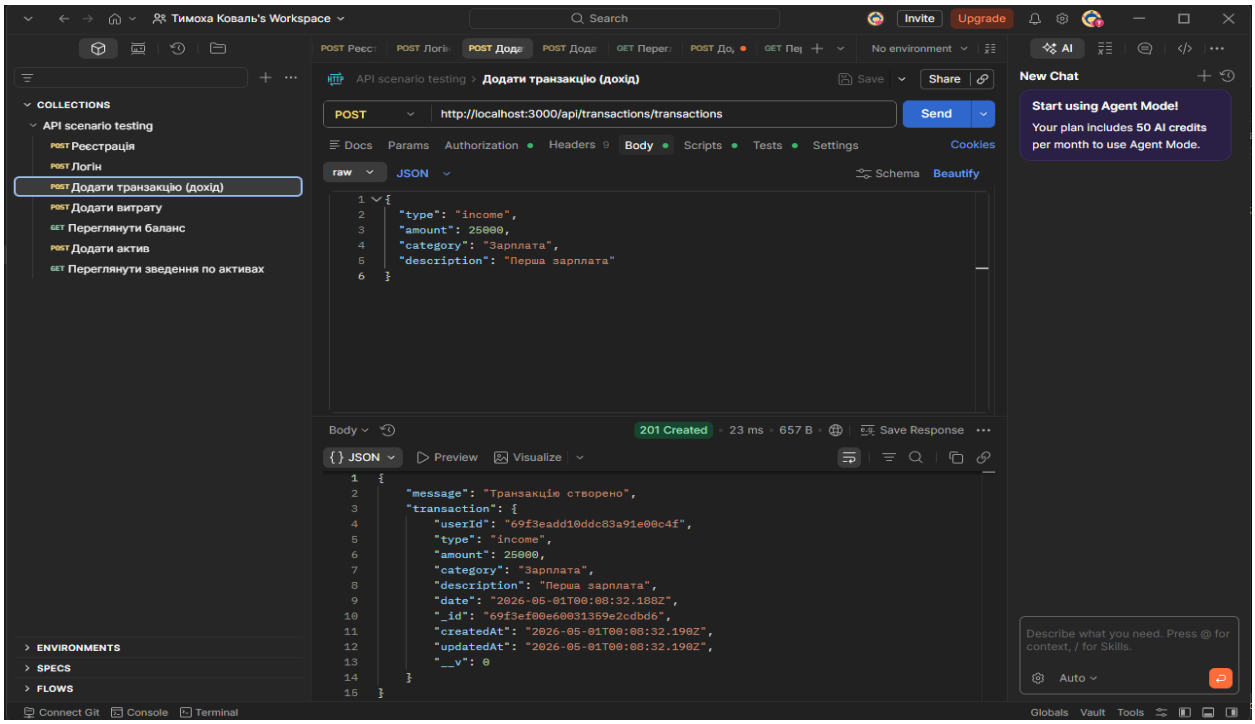

Рисунок 3.4 – Перевірка створення транзакції доходу через Postman

На рисунку 3.4 показано створення транзакції з типом income, сумою 25000, категорією «Зарплата» та описом «Перша зарплата». Відповідь сервера містить статус 201 Created, повідомлення про успішне створення транзакції та об’єкт фінансової операції. Це підтверджує, що Transaction Service коректно обробляє запит на додавання доходу через API Gateway [21].

Другим було протестовано додавання витрати. Для цього використовувався той самий маршрут, але в тілі запиту було передано тип expense, суму 3500,

категорію «Їжа» та опис «Продукти на тиждень». Система також повернула статус 201 Created і повідомлення про створення транзакції. Відповідь містила запис витрати з тими самими ключовими полями, що й у випадку доходу. Різниця полягала в типі операції та сумі, які надалі впливають на підсумковий баланс користувача [12, 21].

Результат створення витрати також зафіксовано за допомогою скріншота Postman.

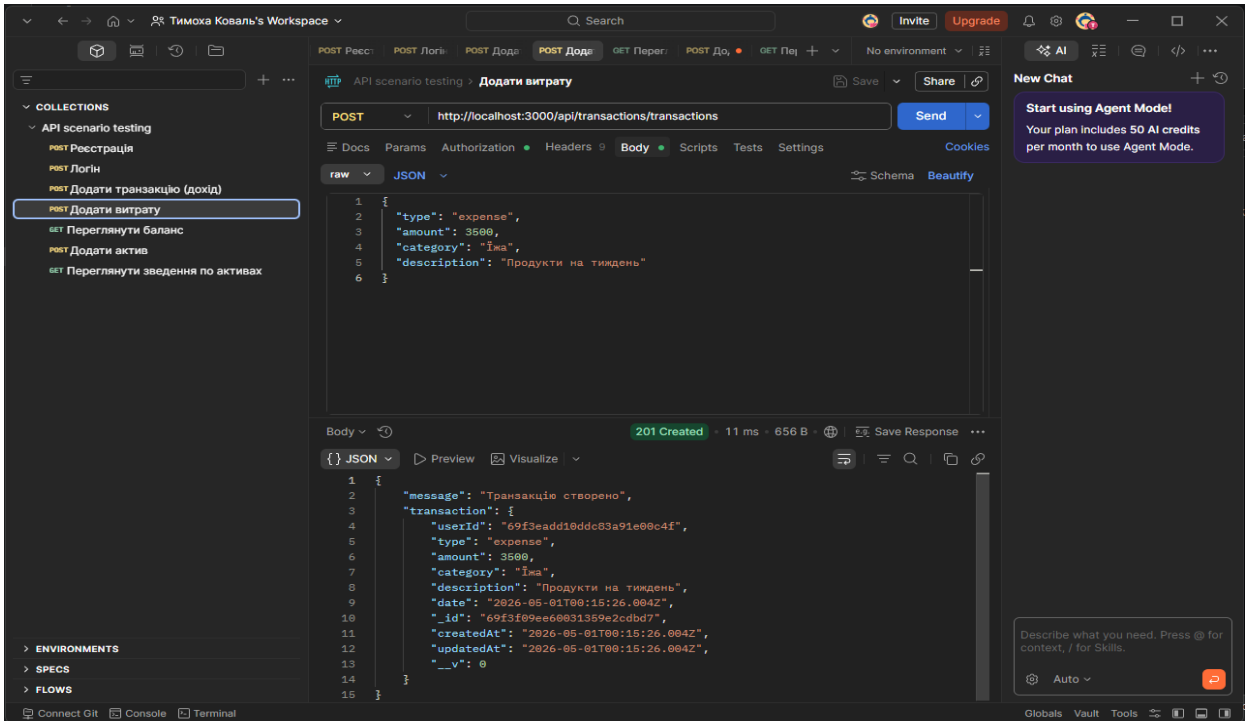

Рисунок 3.5 – Перевірка створення транзакції витрати через Postman

На рисунку 3.5 показано створення транзакції витрати з типом expense, сумою 3500, категорією «Їжа» та описом «Продукти на тиждень». Сервер повернув статус 201 Created, повідомлення про створення транзакції та об’єкт витрати. Це підтверджує, що Transaction Service однаково коректно обробляє обидва типи фінансових операцій [21].

Для додаткового аналізу результатів тестування доцільно подати співвідношення створеного доходу та витрати у вигляді діаграми. Оскільки під час перевірки були отримані реальні числові значення, їх можна використати для побудови стовпчикової діаграми.

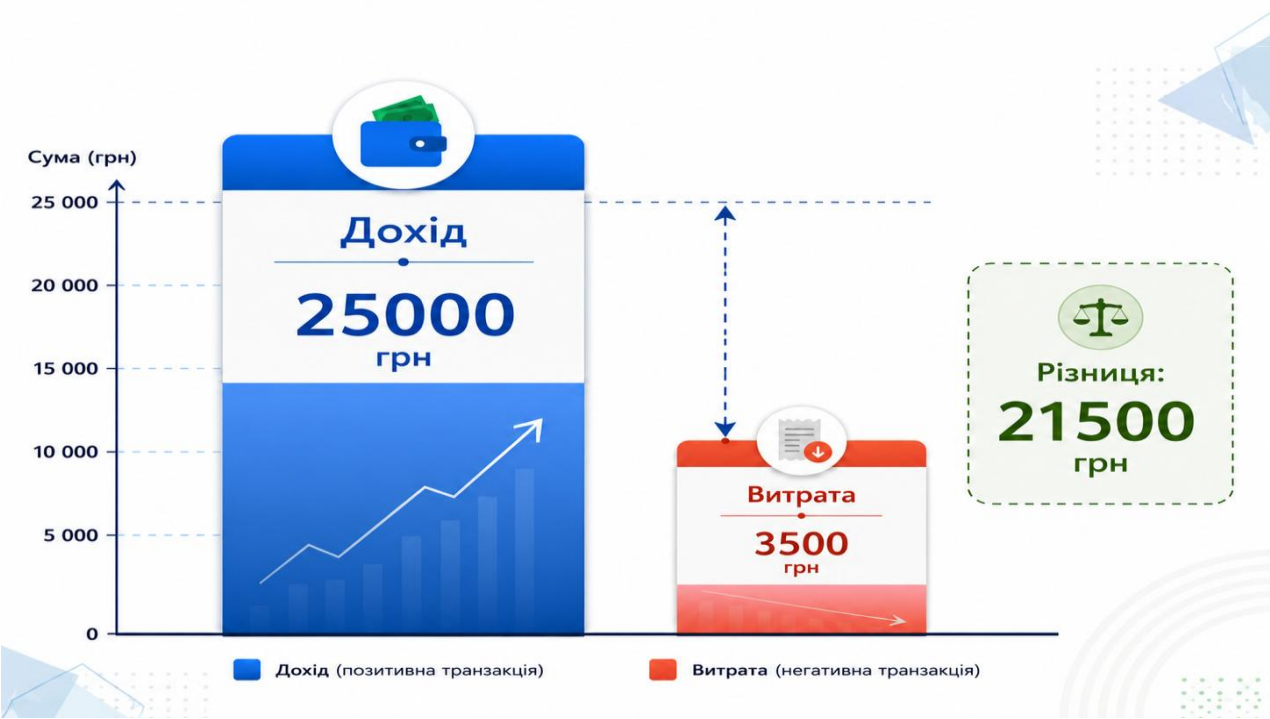

Рисунок 3.6 – Порівняння створених фінансових транзакцій за сумою

На рисунку 3.6 відображено співвідношення між доданим доходом і витратою. Дохід становить 25000 грн, а витрата — 3500 грн, тому на діаграмі добре видно різницю між двома операціями. Такий візуальний елемент доповнює скріншоти Postman і показує, які саме числові дані надалі використовуються під час розрахунку балансу.

Під час тестування важливо було переконатися, що сервіс правильно розрізняє типи транзакцій. Для доходу система має збільшувати загальну суму надходжень, а для витрати — загальну суму витрат. Якщо б поле type оброблялося некоректно, баланс міг би формуватися неправильно. У виконаних запитах обидві транзакції були створені зі статусом 201 Created, що свідчить про успішну роботу маршруту додавання [6, 21].

Окремо варто зазначити, що Transaction Service не створює фінансові записи без прив’язки до користувача. Ідентифікатор користувача береться не з тіла запиту, а з перевіреного JWT-токена. Це важливо для безпеки, оскільки користувач не повинен вручну передавати userId і потенційно підміняти власника транзакції.

Такий підхід зменшує ризик помилок і робить створені записи частиною персонального фінансового профілю [13, 14].

У підсумку тестування підтвердило, що сервіс фінансових транзакцій виконує основну задачу: приймає захищені POST-запити, створює записи доходів і витрат, зберігає їх у MongoDB та повертає коректну відповідь клієнту. Отримані дані стали основою для наступної перевірки — розрахунку балансу користувача на основі вже створених транзакцій [17, 18, 21].

3.5 Тестування додавання персонального активу та отримання зведення за категоріями

Після створення фінансових транзакцій було виконано перевірку розрахунку балансу користувача. Цей етап є логічним продовженням тестування Transaction Service, оскільки окремі записи доходів і витрат мають не лише зберігатися в базі даних, а й правильно використовуватися для формування підсумкових фінансових показників. У розробленій системі баланс не вводиться вручну та не зберігається як окреме незалежне значення. Він обчислюється на основі транзакцій конкретного користувача, що дозволяє отримувати актуальний результат після додавання нових операцій [6, 17, 18].

Для перевірки використовувалися дві транзакції, створені в попередньому підпункті: дохід із сумою 25000 грн у категорії «Зарплата» та витрата із сумою 3500 грн у категорії «Їжа». Обидві операції були додані через захищений маршрут Transaction Service після передавання JWT-токена. Це означає, що під час розрахунку балансу сервіс повинен був врахувати тільки транзакції поточного авторизованого користувача, а не всі записи, які можуть зберігатися в базі даних [13, 14].

Перевірка виконувалася через Postman за допомогою GET-запиту на маршрут http://localhost:3000/api/transactions/transactions/balance. Запит надсилався через API Gateway, після чого перенаправлявся до Transaction Service. Оскільки маршрут балансу є захищеним, у запиті використовувався JWT-токен. Після перевірки

токена middleware визначав userId, а сервіс виконував підсумовування транзакцій, що належать саме цьому користувачу [12, 13, 21].

У результаті виконання запиту система повернула статус 200 OK і JSON-відповідь із трьома основними показниками: totalIncome, totalExpense і balance. Значення totalIncome становило 25000, значення totalExpense — 3500, а підсумковий баланс — 21500. Отриманий результат відповідає очікуваній логіці розрахунку: від загальної суми доходів віднімається загальна сума витрат. У цьому випадку 25000 - 3500 = 21500, тому відповідь сервера є коректною [12, 21].

Для підтвердження фактичного виконання перевірки в роботі використовується скріншот Postman із результатом запиту на отримання балансу.

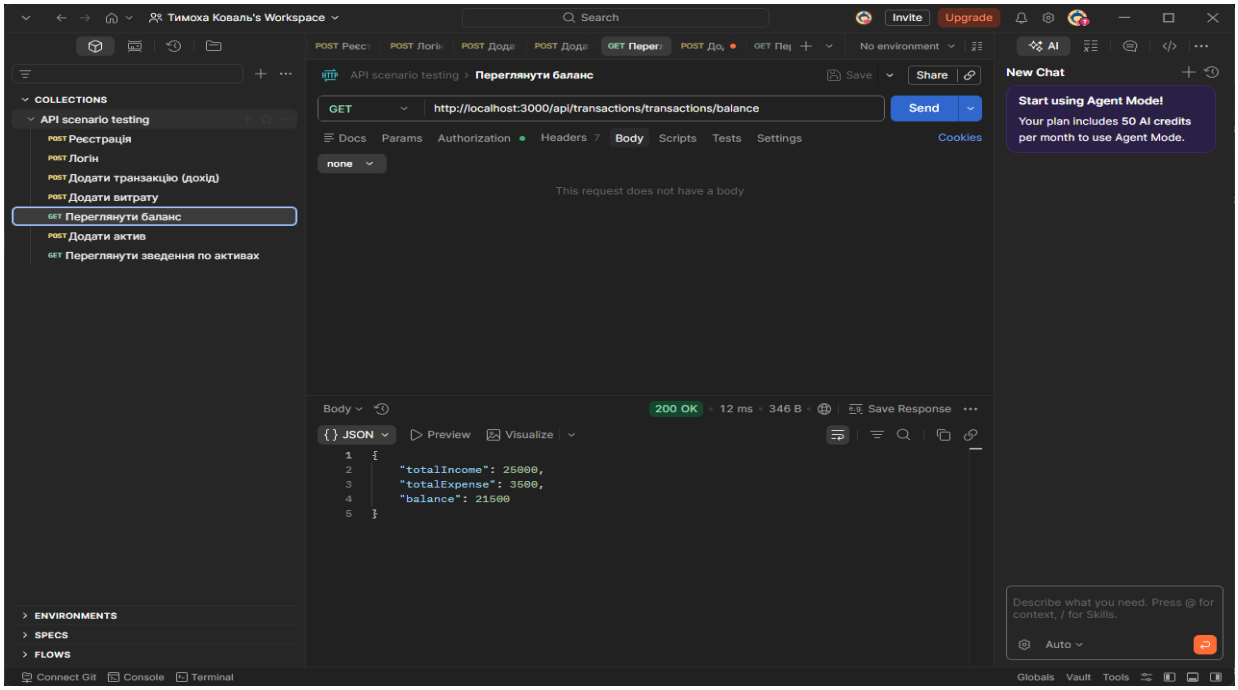

Рисунок 3.7 – Перевірка розрахунку балансу користувача через Postman

На рисунку 3.7 показано виконання GET-запиту до маршруту розрахунку балансу. У відповіді сервера відображено статус 200 OK і значення totalIncome: 25000, totalExpense: 3500, balance: 21500. Це підтверджує, що Transaction Service правильно обробляє захищений запит, отримує транзакції поточного користувача та формує фінансовий підсумок на основі вже створених записів [21].

Для наочного подання результатів розрахунку доцільно використати

діаграму, яка показує співвідношення між загальним доходом, загальними витратами та підсумковим балансом.

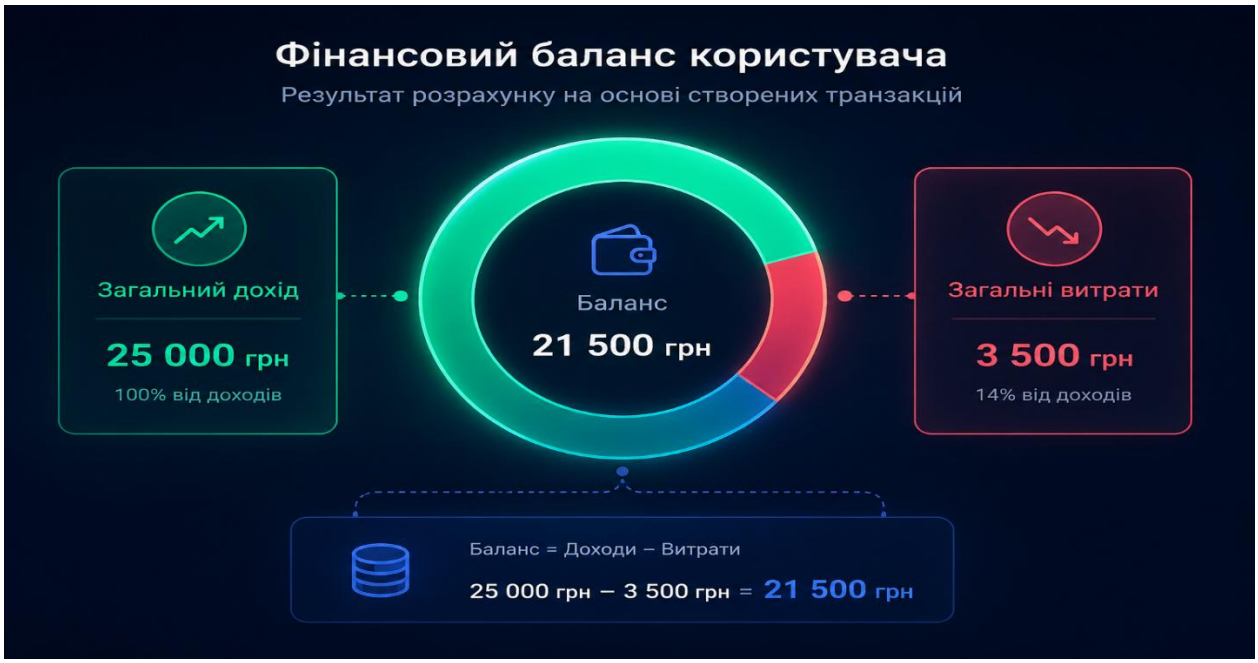

Рисунок 3.8 – Результат розрахунку балансу користувача

На рисунку 3.8 показано три ключові значення, отримані після виконання запиту до Transaction Service. Діаграма дозволяє побачити, що баланс є меншим за загальний дохід саме на суму витрат. Це важливо для перевірки, оскільки система повинна не просто повертати збережені транзакції, а виконувати обчислення на їх основі.

Окрему увагу під час перевірки було приділено тому, що баланс формується не за даними, переданими вручну в запиті, а за записами, які вже збережені в MongoDB. Користувач не передає суму доходів, витрат або готовий баланс. Transaction Service самостійно отримує транзакції за userId, розподіляє їх за типом income і expense, підсумовує відповідні значення та повертає результат. Це зменшує ризик помилок, які могли б виникнути при ручному підрахунку [13, 17, 18].

Перевірка розрахунку балансу також показала зв’язок між попередніми етапами тестування. Реєстрація та вхід забезпечили отримання JWT-токена, захищений доступ дозволив створити транзакції, а маршрут балансу використав ці

транзакції для формування підсумку. У результаті було підтверджено не окрему ізольовану функцію, а послідовну роботу кількох частин системи [6, 13, 21].

У підсумку тестування показало, що Transaction Service коректно розраховує баланс користувача на основі створених доходів і витрат. Отримані значення totalIncome, totalExpense і balance відповідають очікуваному результату, а відповідь сервера через Postman підтверджує працездатність маршруту /balance. Це дає підставу перейти до перевірки роботи Asset Service, який відповідає за додавання персональних активів і формування зведення за категоріями [17, 18, 21].

3.6 Оцінка результатів тестування, стабільності роботи та можливостей подальшої підтримки системи

Після перевірки створення фінансових транзакцій і розрахунку балансу було виконано тестування роботи Asset Service. Цей сервіс відповідає за збереження персональних активів користувача та формування підсумкової інформації за категоріями. Якщо Transaction Service показує рух коштів через доходи й витрати, то Asset Service дозволяє зафіксувати майнові об’єкти, які мають певну вартість і можуть впливати на загальне уявлення про фінансовий стан користувача [3, 4].

Тестування виконувалося через Postman із використанням API Gateway. Для додавання активу застосовувався маршрут POST http://localhost:3000/api/assets/assets. Оскільки цей маршрут є захищеним, перед виконанням запиту використовувався JWT-токен, отриманий після входу користувача в систему. Це дозволило перевірити, що Asset Service не приймає майнові записи як загальні дані, а створює їх із прив’язкою до конкретного авторизованого користувача [13, 14, 21].

У тілі POST-запиту було передано назву активу Toyota Camry 2020, категорію vehicle, вартість 850000, валюту UAH, опис «Особистий автомобіль» і дату придбання 2020-06-15. Після надсилання запиту система повернула статус 201 Created, повідомлення «Актив додано» та об’єкт створеного активу. У відповіді також було відображено userId, назву, категорію, вартість, валюту, опис, дату

придбання, активний статус і службові поля MongoDB. Це підтверджує, що сервіс коректно прийняв дані, доповнив їх ідентифікатором користувача та зберіг у базі даних [17, 18, 21].

Для підтвердження фактичного виконання запиту на створення активу в роботі використовується скріншот Postman.

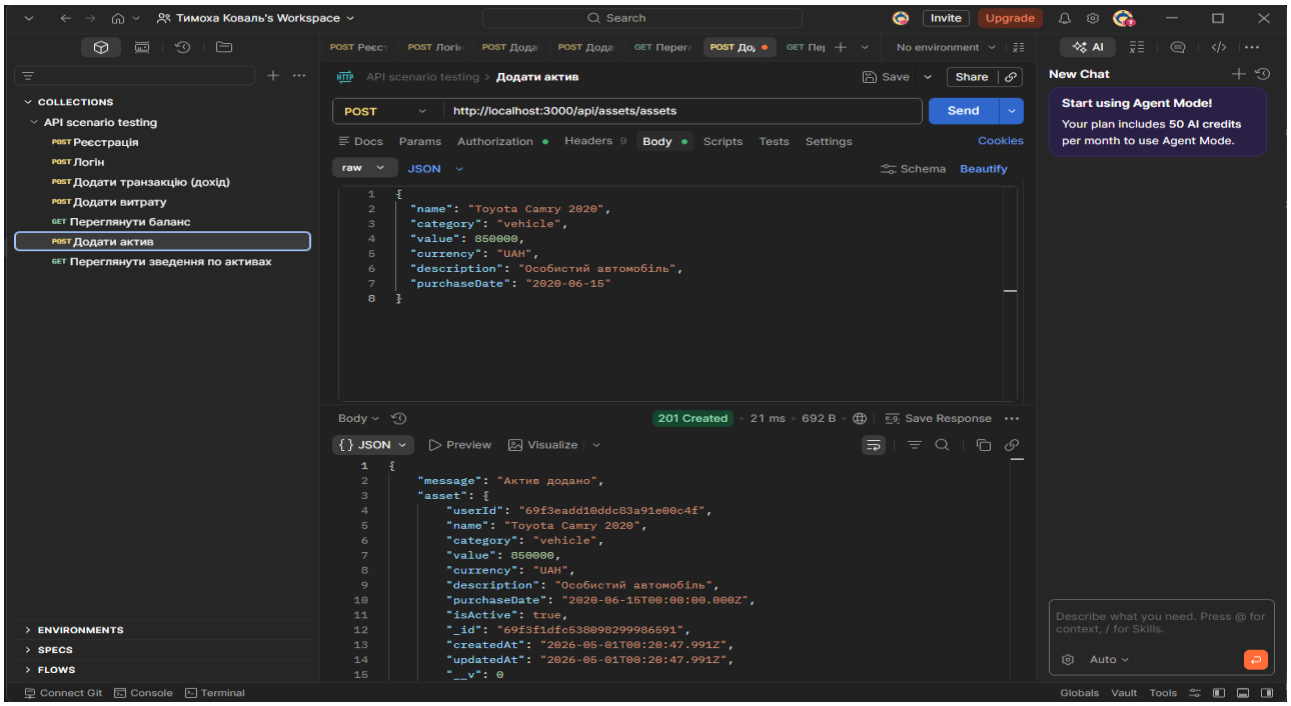

Рисунок 3.9 – Перевірка додавання персонального активу через Postman

На рисунку 3.9 показано виконання POST-запиту до маршруту додавання активу. У тілі запиту передано дані автомобіля Toyota Camry 2020, що належить до категорії vehicle і має вартість 850000 UAH. У відповіді сервера відображено статус 201 Created, повідомлення про успішне додавання активу та створений об’єкт запису. Це підтверджує працездатність маршруту створення активів у Asset Service [21].

Після додавання активу було перевірено отримання зведення за категоріями. Для цього виконано GET-запит на маршрут http://localhost:3000/api/assets/assets/summary. Цей маршрут також є захищеним, тому запит виконувався з JWT-токеном. Його призначення полягає не в поверненні повного списку активів, а у формуванні підсумкової інформації: загальної кількості

активів, сумарної вартості та групування за категоріями [12, 13].

У результаті виконання запиту система повернула статус 200 OK і JSON-відповідь із полями totalAssets, totalValue та byCategory. Значення totalAssets становило 1, а totalValue — 850000. У структурі byCategory було сформовано категорію vehicle, для якої вказано кількість активів 1, сумарну вартість 850000 і валюту UAH. Це показує, що Asset Service правильно обробляє не лише окремий запис активу, а й підсумкові дані за категоріями [17, 18, 21].

Результат отримання зведення активів також зафіксовано за допомогою скріншота Postman.

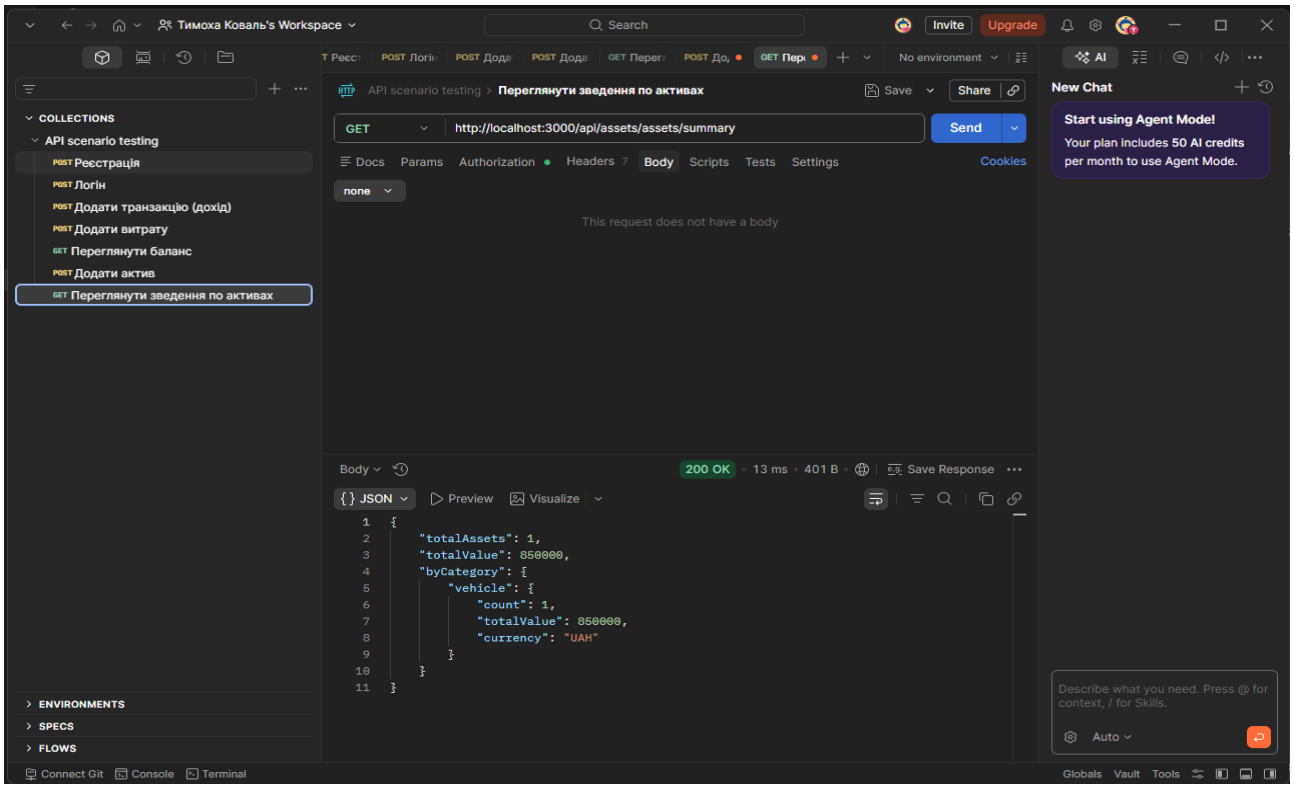

Рисунок 3.10 – Перевірка отримання зведення активів за категоріями через Postman

На рисунку 3.10 показано GET-запит до маршруту зведення активів. У відповіді сервера видно, що система повернула один актив із загальною вартістю 850000 UAH, а також сформувала групування за категорією vehicle. Це підтверджує, що маршрут summary коректно підсумовує майнові записи користувача та повертає дані у структурованому вигляді [21].

Для наочного подання отриманого результату доцільно використати діаграму вартості активів за категоріями. Оскільки під час тестування було створено один актив у категорії vehicle, у діаграмі відображається саме ця категорія та її сумарна вартість.

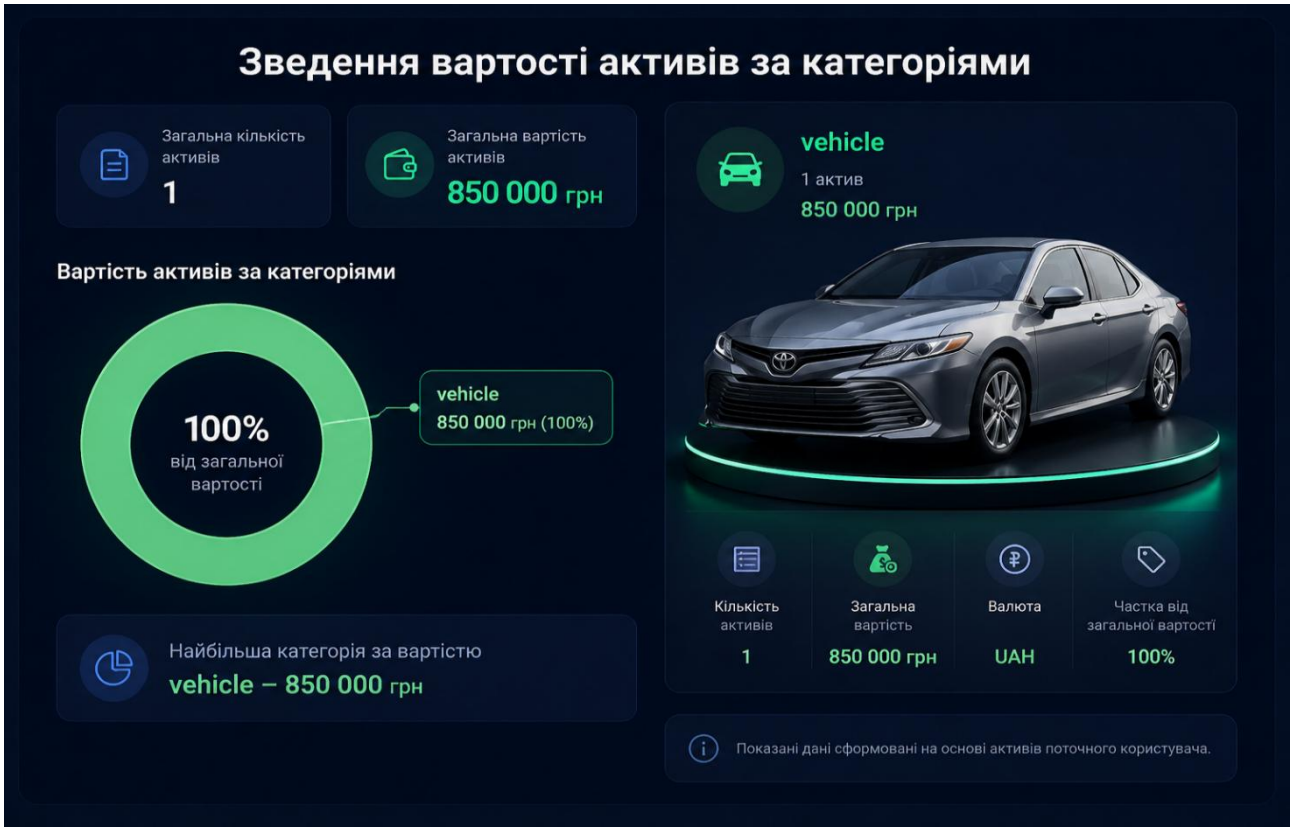

Рисунок 3.11 – Зведення вартості активів за категоріями

На рисунку 3.11 відображено результат групування активів за категоріями. Категорія vehicle містить один актив із загальною вартістю 850000 грн. Такий візуальний елемент доповнює скріншот Postman і показує, як дані з відповіді API можуть бути використані для подальшої аналітики або побудови клієнтського інтерфейсу [17, 21].

Під час тестування було важливо перевірити не тільки факт створення активу, а й логіку його подальшого використання в підсумковому маршруті. Якщо актив успішно зберігається, але не враховується у зведенні, система не виконує повну задачу моніторингу майнових даних. У цьому випадку створений запис було враховано правильно: кількість активів, загальна вартість і категорія відповідають

даним, переданим у POST-запиті [6, 21].

Окремо потрібно зазначити, що активи, як і транзакції, прив’язуються до користувача через userId, отриманий із JWT-токена. Це означає, що зведення формується не за всіма записами в базі даних, а лише за активами поточного авторизованого користувача. Такий підхід є важливим для персонального характеру системи, оскільки кожен користувач повинен бачити тільки власні майнові записи [13, 14].

У підсумку тестування Asset Service підтвердило коректну роботу функцій додавання активу та отримання зведення за категоріями. Система успішно створила запис про автомобіль, зберегла його з прив’язкою до користувача, повернула відповідь зі статусом 201 Created, а потім сформувала зведення з кількістю активів і загальною вартістю. Це завершує перевірку основних функцій програмного продукту, пов’язаних із моніторингом персональних активів [17, 18, 21].

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

В даному розділі розглядається два ключових питання, зокрема моделювання та прогнозування небезпечних ситуацій як інструмент запобігання небезпекам, а також ергономічні вимоги до організації робочого місця оператора ПК.

4.1 Моделювання та прогнозування небезпечних ситуацій

Небезпечна ситуація – це сукупність умов і обставин, за яких створюється реальна загроза життю та здоров'ю людини, матеріальним цінностям і навколишньому природному середовищу. Зважаючи на ймовірнісну природу більшості небезпек, ключовими інструментами їх виявлення та оцінювання у безпеці життєдіяльності виступають моделювання і прогнозування [23].

Моделювання небезпечних ситуацій – це метод дослідження процесів виникнення та розвитку небезпек шляхом побудови й аналізу їх моделей, тобто спрощених формалізованих описів, що відтворюють найістотніші властивості та закономірності реального явища. Мета моделювання полягає у вивченні механізмів зародження небезпеки, встановленні причинно-наслідкових зв'язків між чинниками ризику та можливими наслідками, а також у кількісному оцінюванні ймовірності й масштабів подій без проведення натурних експериментів.

За способом опису та характером отримуваних результатів розрізняють кілька типів моделей небезпечних ситуацій. Детерміновані (фізико-математичні) моделі описують перебіг процесу однозначними залежностями та застосовуються для розрахунку зон ураження, полів концентрацій шкідливих речовин, параметрів ударної хвилі тощо. Імовірнісні (стохастичні) моделі враховують випадковий характер виникнення небезпечних подій і дають оцінку ймовірності їх настання. Особливе місце посідають структурно-логічні моделі – «дерево подій» та «дерево відмов», які у вигляді логічних схем відображають послідовності й комбінації відмов та помилок, що призводять до аварії, і дозволяють виявити найвразливіші ланки системи [23, 24].

Прогнозування небезпечних ситуацій – це науково обґрунтоване передбачення ймовірності виникнення, місця, часу, характеру та можливих наслідків небезпечних і надзвичайних ситуацій. Залежно від завчасності розрізняють довгострокове, середньострокове, короткострокове та оперативне прогнозування. За змістом прогноз має дати відповідь на питання, де і коли може виникнути небезпечна подія, які чинники її спричинять, якими будуть масштаби ураження та який обсяг сил і засобів потрібен для ліквідації наслідків.

Для прогнозування застосовують такі основні групи методів як статистичні, експертні, що ґрунтуються на узагальненні думок фахівців за умов браку статистичних даних, аналітично-розрахункові, які використовують математичні моделі фізичних і хімічних процесів, методи аналогій, що переносять закономірності вже вивчених ситуацій на подібні, а також методи математичного та імітаційного моделювання із застосуванням обчислювальної техніки [24]. Сучасною методологічною основою моделювання та прогнозування є ризик-орієнтований підхід, у межах якого небезпека характеризується кількісно, через ризик. Ризик визначається як добуток імовірності виникнення небезпечної події на величину очікуваних наслідків (збитків). Розрізняють індивідуальний та колективний (соціальний) ризик, а також поняття прийнятного (допустимого) ризику – рівня, з яким суспільство погоджується заради отримання певних благ. Оцінювання та порівняння ризиків дозволяють ранжувати небезпеки за ступенем загрози й спрямовувати обмежені ресурси на запобігання насамперед найбільш значущим з них. Процес визначення та прогнозування небезпечних ситуацій є послідовністю взаємопов'язаних етапів [23]:

1. Виявлення джерел небезпеки та потенційно небезпечних об'єктів, складання їх переліку.

2. Оцінювання ймовірності виникнення небезпечної події на основі статистичних даних та обраних моделей.

3. Розрахунок зон ураження, можливих людських та матеріальних втрат за різними сценаріями.

4. Оцінювання потреби у силах і засобах, необхідних для локалізації та

ліквідації наслідків.

5. Розроблення запобіжних та захисних заходів і планів реагування, спрямованих на зниження ризику до прийнятного рівня.

Таким чином, моделювання та прогнозування є нерозривно пов'язаними складовими управління безпекою, що включають перехід від реагування на вже скоєні події до їх завчасного запобігання. Разом вони створюють інформаційну основу для прийняття обґрунтованих управлінських рішень, планування захисних заходів та раціонального розподілу ресурсів, чим суттєво підвищують рівень захищеності людини й навколишнього середовища від небезпечних та надзвичайних ситуацій.

4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Ергономіка робочого місця оператора персонального комп'ютера є прикладною науково-технічною дисципліною, що вивчає та оптимізує взаємодію людини з технічними засобами й виробничим середовищем з метою досягнення ефективності, безпеки та комфорту праці. Нормативну базу в Україні складають НПАОП 0.00-7.15-18, ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами», ДСТУ 8604:2015 «Ергономіка. Робочі місця з відеодисплейними терміналами», та ДБН В.2.5-28:2018 та ДСН 3.3.6.042-99 [25].

Вимоги до виробничого приміщення. Відповідно до ДСанПіН 3.3.2.007-98 площа на одне робоче місце з ВДТ має становити не менше 6 м², об'єм виробничого приміщення – не менше 20 м³ на одного працівника. Параметри мікроклімату нормуються відповідно до ДСН 3.3.6.042-99 [27]. У холодний період температура повітря 22-24 °С, відносна вологість 40-60 %, швидкість руху повітря не більше 6 0,1 м/с, а у теплий період 23-25 °С, вологість 40-60 %, швидкість повітря до 0,1 м/с. Повітрообмін забезпечується природною або примусовою вентиляцією, кратність якої визначається проєктом відповідно до ДБН В.2.2-28:2010.

Природне освітлення повинно бути лівобічним відносно робочого місця

оператора. Нормований рівень штучної освітленості на горизонтальній робочій поверхні для приміщень з ВДТ має становити не менше 300-500 лк відповідно до ДБН В.2.5-28:2018 [26]. Рівномірність освітленості (відношення мінімальної до максимальної) не повинна бути нижчою за 0,4. Монітор слід орієнтувати перпендикулярно до вікон для уникнення прямих та відбитих відблисків на поверхні екрана, рекомендується застосовувати жалюзі або світлорозсіювальні перегородки, а також монітори з антивідблисковим покриттям.

Геометричні параметри робочого місця є визначальними для профілактики м'язово-скелетних розладів та підтримання стабільної працездатності оператора ПК. На рисунку 4.1 наведено схему раціональної організації робочого місця з позначенням нормованих ергономічних параметрів.

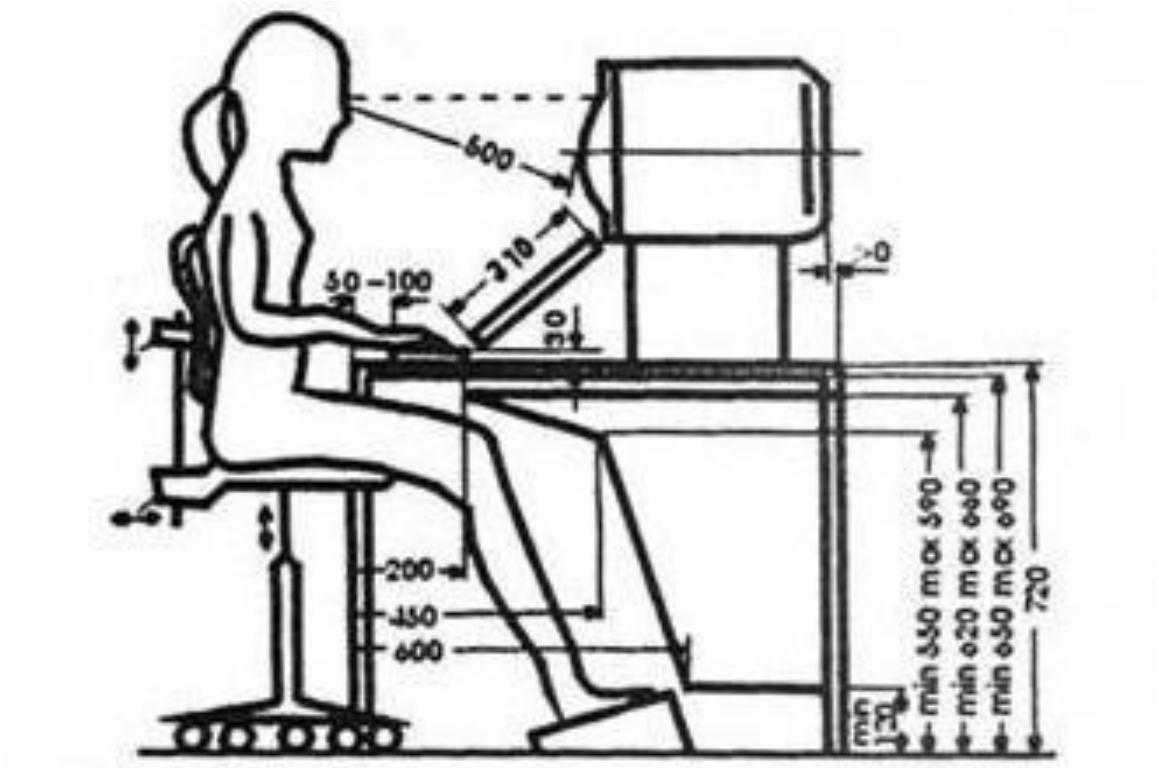

Рисунок 4.1 – Схема організації робочого місця оператора ПК [27]

Відповідно до нормативних вимог та рекомендацій правильна організація робочого місця передбачає дотримання таких параметрів [25]:

− Висота робочої поверхні столу 680-800 мм від рівня підлоги з можливістю регулювання під індивідуальний зріст оператора.

− Відстань від рогівки ока до поверхні екрана монітора 600-700 мм.

− Кут у ліктьовому суглобі при горизонтальному положенні передпліч близько 90°, передпліччя спираються на підлокітники крісла або поверхню столу без статичного напруження.

− Кут між стегном і тулубом (у тазостегновому суглобі) близько 90°, ступні повністю спираються на підлогу або підставку для ніг з регулюванням за висотою та кутом нахилу.

− Клавіатура розміщується на відстані 10-30 мм рахуючи, що він знаходяться від переднього краю столу, для підтримки зап'ясть рекомендується підставка або гелева опора.

Монітор повинен мати роздільну здатність не нижче 1920×1080 пікселів, частоту вертикальної розгортки не менше 60 Гц, регульовані яскравість і контраст, матову поверхню екрана. Клавіатура має бути відокремленою від монітора, із матовим покриттям клавіш і товщиною, що не перевищує 30 мм. Маніпулятор «миша» підбирається відповідно до характеристик долоні оператора.

Робоче крісло має відповідати вимогам ДСТУ 8604:2015 та бути регульованим за висотою сидіння (420-550 мм від рівня підлоги), висотою та кутом нахилу спинки. Спинка крісла повинна мати підтримку поперекового відділу хребта та нахилятися відносно сидіння під кутом 95-110°. Відповідно до вимог НПАОП 0.00-7.15-18 оператор зобов'язаний регулярно змінювати робочу позу, уникаючи статичного положення тривалістю понад 30-45 хвилин поспіль [25].

Правильна ергономічна організація робочого місця оператора ПК є необхідною передумовою збереження здоров'я, підтримання стабільної працездатності та виконання вимог чинного законодавства про охорону праці. Дотримання описаних санітарних норм і ергономічних рекомендацій безпосередньо впливає на якість та ефективність виконання задач з розробки, тестування та промислової експлуатації програмного забезпечення.

Отже, правильна ергономічна організація робочого місця оператора ПК є комплексним завданням, що охоплює вимоги до виробничого приміщення, освітлення, мікроклімату та геометричних параметрів розміщення обладнання.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено програмне забезпечення для моніторингу персональних активів, побудоване на основі мікросервісної архітектури з використанням платформи Node.js. У результаті виконання роботи сформовано серверну систему, яка охоплює реєстрацію користувача, вхід до системи, JWT-захист маршрутів, облік доходів і витрат, розрахунок балансу, додавання персональних активів та отримання зведення активів за категоріями.

У першому розділі було проаналізовано предметну область моніторингу персональних активів і фінансових операцій. Розглянуто проблеми ручного обліку доходів, витрат і майнових записів, зокрема розрізненість даних, ризик помилок у підрахунках, складність оновлення інформації та недостатній рівень захисту. Також було виконано огляд програмних аналогів для обліку особистих фінансів, визначено функціональні й нефункціональні вимоги до системи, сформовано основні сценарії взаємодії користувача з програмним продуктом і встановлено межі реалізації.

У другому розділі було спроєктовано та реалізовано мікросервісну структуру системи. Для організації єдиної точки входу використано API Gateway, який перенаправляє запити до окремих сервісів. Auth Service реалізує реєстрацію, автентифікацію користувача, хешування пароля та формування JWT-токена. Transaction Service забезпечує створення, перегляд, оновлення й видалення фінансових транзакцій, а також розрахунок балансу на основі доходів і витрат. Asset Service відповідає за збереження персональних активів, роботу з категоріями та формування підсумкового зведення. Для збереження даних використано MongoDB, а запуск сервісів і баз даних організовано через Docker Compose.

У третьому розділі було перевірено працездатність розробленої системи. Тестування виконувалося за допомогою Postman через API Gateway. Було перевірено реєстрацію користувача, вхід до системи та отримання JWT-токена. Окремо розглянуто роботу захищених маршрутів, які потребують передавання токена авторизації. Після цього було протестовано створення доходу із сумою

25000 грн і витрати із сумою 3500 грн, а також перевірено розрахунок балансу, який становив 21500 грн. Додатково було протестовано додавання активу Toyota Camry 2020 вартістю 850000 грн у категорії vehicle та отримання зведення активів, де система коректно повернула кількість активів, загальну вартість і групування за категоріями.

У четвертому розділі було розглянуто питання безпеки життєдіяльності та основ охорони праці. Проаналізовано моделювання й прогнозування небезпечних ситуацій як засоби виявлення ризиків, оцінювання ймовірності їх виникнення та визначення можливих наслідків. Також описано ергономічні вимоги до організації робочого місця оператора ПК, зокрема параметри приміщення, освітлення, мікроклімату, розміщення монітора, клавіатури, робочого столу, крісла та підставки для ніг.

Практичним результатом виконання КРБ стала працездатна серверна мікросервісна система, придатна для запуску в контейнеризованому середовищі та перевірки через REST API. Розроблене програмне забезпечення демонструє поділ відповідальності між сервісами, захищену роботу з персональними даними користувача, автоматичний розрахунок фінансових підсумків і збереження інформації про майнові активи. Отриманий результат підтверджує можливість використання Node.js, Express.js, MongoDB, JWT і Docker Compose для побудови структурованої системи моніторингу персональних активів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. URL: https://www.computer.org/education/bodies-of-knowledge/software-engineering (дата звернення: 07.06.2026).

2. ISO/IEC/IEEE 12207:2017. Systems and software engineering — Software life cycle processes. Geneva : International Organization for Standardization, 2017. URL: https://www.iso.org/standard/63712.html (дата звернення: 07.06.2026).

3. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. Geneva : International Organization for Standardization, 2018. URL: https://www.iso.org/standard/72089.html (дата звернення: 07.06.2026).

4. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva : International Organization for Standardization, 2011. URL: https://www.iso.org/standard/35733.html (дата звернення: 07.06.2026).

5. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

6. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill Education, 2020. 704 p.

7. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley Professional, 2021. 464 p.

8. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O’Reilly Media, 2021. 612 p.

9. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

10. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 560 p.

11. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. Irvine : University of California, 2000. URL:

https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 07.06.2026).

12. Fielding R., Nottingham M., Reschke J. RFC 9110: HTTP Semantics. Internet Engineering Task Force, 2022. URL: https://datatracker.ietf.org/doc/html/rfc9110 (дата звернення: 07.06.2026).

13. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 07.06.2026).

14. OWASP Application Security Verification Standard (ASVS). OWASP Foundation. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 07.06.2026).

15. Node.js v20.x Documentation. Node.js. URL: https://nodejs.org/download/release/latest-v20.x/docs/api/ (дата звернення: 07.06.2026).

16. Express 4.x API Reference. Express.js. URL: https://expressjs.com/en/4x/api/ (дата звернення: 07.06.2026).

17. MongoDB Manual. MongoDB Documentation. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 07.06.2026).

18. Mongoose Documentation. Mongoose ODM. URL: https://mongoosejs.com/docs/ (дата звернення: 07.06.2026).

19. Docker Documentation. Docker Inc. URL: https://docs.docker.com/ (дата звернення: 07.06.2026).

20. Docker Compose Documentation. Docker Inc. URL: https://docs.docker.com/compose/ (дата звернення: 07.06.2026).

21. Postman Docs. Postman Learning Center. URL: https://learning.postman.com/ (дата звернення: 07.06.2026).

22. bcryptjs. npm package documentation. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 07.06.2026).

23. Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня „бакалавр". Для студентів всіх форм навчання рівень вищої освіти перший

(бакалаврський) / укл. : О. Я. Гурик , І. Б. Окіпний. – Тернопіль : ТНТУ імені Івана Пулюя, 2021. - 20 с.

24. Запорожець О.І. Безпека життєдіяльності : підручник. 2-ге вид. Київ : Центр учбової літератури, 2020. 448 с.

25. Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

26. ДБН В.2.5-28 : 2018. Природне і штучне освітлення. Київ : Мінрегіон України, 2018. 133 с.

27. Гогіташвілі Г. Г., Лапін В. М. Основи охорони праці : навч. посіб. 4-те вид. випр. і доп. Київ : Знання, 2018. 302 с.

ДОДАТКИ

ДОДАТОК А

Фрагменти програмного коду мікросервісної системи

У додатку А наведено ключові фрагменти програмного коду, які підтверджують реалізацію програмного забезпечення для моніторингу персональних активів. Повний код проєкту не дублюється, оскільки для підтвердження практичної частини достатньо подати основні елементи: API Gateway, MongoDB/Mongoose-моделі, JWT-захист, розрахунок балансу, формування зведення активів і фрагмент контейнеризованого запуску через Docker Compose.

А.1 Фрагмент API Gateway

```
app.use('/api/auth', createProxyMiddleware({
  target: process.env.AUTH_SERVICE_URL,
  changeOrigin: true,
  pathRewrite: {
    '^/api/auth': '/auth'
  }
}));
app.use('/api/transactions', createProxyMiddleware({
  target: process.env.TRANSACTION_SERVICE_URL,
  changeOrigin: true,
  pathRewrite: {
    '^/api/transactions': '/transactions'
  }
}));
app.use('/api/assets', createProxyMiddleware({
  target: process.env.ASSET_SERVICE_URL,
  changeOrigin: true,
  pathRewrite: {
    '^/api/assets': '/assets'
```

```
  }
}));
```

Фрагмент показує, що API Gateway приймає зовнішні запити та перенаправляє їх до Auth Service, Transaction Service і Asset Service. Завдяки цьому користувач або інструмент тестування працює з єдиною точкою входу, а внутрішня взаємодія між сервісами залишається прихованою.

А.2 Модель користувача та хешування пароля

```
const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, 'Ім’я користувача обов’язкове'],
    unique: true,
    trim: true,
    minlength: 3
  },
  email: {
    type: String,
    required: [true, 'Email обов’язковий'],
    unique: true,
    lowercase: true,
    trim: true
  },
  password: {
    type: String,
    required: [true, 'Пароль обов’язковий'],
    minlength: 6
  }
}, {
  timestamps: true
});
```

```
UserSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
UserSchema.methods.comparePassword = async function (candidatePassword) {
  return await bcrypt.compare(candidatePassword, this.password);
};
```

Наведений фрагмент підтверджує, що пароль користувача не зберігається у відкритому вигляді. Перед записом у MongoDB він хешується за допомогою bcryptjs, а під час входу виконується порівняння введеного пароля з хешем.

А.3 Генерація JWT-токена

```
const generateToken = (userId) => {
  return jwt.sign(
    { id: userId },
    process.env.JWT_SECRET,
    { expiresIn: process.env.JWT_EXPIRES_IN }
  );
};
```

Цей фрагмент використовується після успішної реєстрації або входу користувача. JWT-токен містить ідентифікатор користувача та надалі застосовується для виконання захищених запитів до сервісів транзакцій і активів.

А.4 Middleware для перевірки JWT

```
const protect = (req, res, next) => {
  try {
    const authHeader = req.headers['authorization'];

    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      return res.status(401).json({
```

```
        message: 'Доступ заборонено. Токен відсутній'
      });
    }
    const token = authHeader.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userId = decoded.id;
    next();
  } catch (error) {
    return res.status(401).json({
      message: 'Невалідний або прострочений токен'
    });
  }
};
```

Middleware protect використовується в Transaction Service та Asset Service. Після перевірки токена ідентифікатор користувача записується в req.userId, тому фінансові транзакції та персональні активи створюються саме для поточного авторизованого користувача.

А.5 Модель фінансової транзакції

```
const TransactionSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
    index: true
  },
  type: {
    type: String,
    enum: ['income', 'expense'],
    required: [true, 'Тип транзакції обов’язковий']
  },
  amount: {
```

```
    type: Number,
    required: [true, 'Сума обов’язкова'],
    min: [0.01, 'Сума має бути більше 0']
  },
  category: {
    type: String,
    required: [true, 'Категорія обов’язкова'],
    trim: true
  },
  description: {
    type: String,
    trim: true,
    default: ''
  },
  date: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});
```

Модель Transaction описує фінансову операцію користувача. Поле type обмежене значеннями income і expense, що дозволяє відокремлювати доходи від витрат під час розрахунку балансу.

А.6 Фрагмент розрахунку балансу

```
const getBalance = async (req, res) => {
  try {
    const transactions = await Transaction.find({ userId: req.userId });
    let totalIncome = 0;
    let totalExpense = 0;
```

```
    transactions.forEach(t => {
      if (t.type === 'income') {
        totalIncome += t.amount;
      } else {
        totalExpense += t.amount;
      }
    });
    const balance = totalIncome - totalExpense;
    res.status(200).json({
      totalIncome: parseFloat(totalIncome.toFixed(2)),
      totalExpense: parseFloat(totalExpense.toFixed(2)),
      balance: parseFloat(balance.toFixed(2))
    });
  } catch (error) {
    res.status(500).json({ message: 'Помилка сервера' });
  }
};
```

Фрагмент підтверджує, що баланс користувача не вводиться вручну, а обчислюється автоматично на основі створених доходів і витрат.

А.7 Модель персонального активу

```
const AssetSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
    index: true
  },
  name: {
    type: String,
    required: [true, 'Назва активу обов’язкова'],
    trim: true
```

```
  },
  category: {
    type: String,
    enum: ['real_estate', 'vehicle', 'electronics', 'jewelry', 'investment', 'other'],
    required: [true, 'Категорія обов’язкова']
  },
  value: {
    type: Number,
    required: [true, 'Вартість активу обов’язкова'],
    min: [0, 'Вартість не може бути від’ємною']
  },
  currency: {
    type: String,
    default: 'UAH',
    enum: ['UAH', 'USD', 'EUR']
  },
  description: {
    type: String,
    trim: true,
    default: ''
  },
  purchaseDate: {
    type: Date,
    default: null
  },
  isActive: {
    type: Boolean,
    default: true
  }
}, {
```

```
  timestamps: true
});
```

Модель Asset використовується для збереження персональних активів користувача. У ній передбачено назву активу, категорію, вартість, валюту, опис, дату придбання та активний статус.

А.8 Фрагмент формування зведення активів

```
const getAssetsSummary = async (req, res) => {
  try {
    const assets = await Asset.find({
      userId: req.userId,
      isActive: true
    });
    const summary = {};
    let totalValue = 0;
    assets.forEach(asset => {
      if (!summary[asset.category]) {
        summary[asset.category] = {
          count: 0,
          totalValue: 0,
          currency: asset.currency
        };
      }
      summary[asset.category].count += 1;
      summary[asset.category].totalValue += asset.value;
      totalValue += asset.value;
    });
    res.status(200).json({
      totalAssets: assets.length,
      totalValue: parseFloat(totalValue.toFixed(2)),
      byCategory: summary
```

```
    });
  } catch (error) {
    res.status(500).json({ message: 'Помилка сервера' });
  }
};
```

Фрагмент показує формування зведення активів за категоріями. Сервіс підраховує кількість активів, їхню загальну вартість і групує записи за категоріями тільки для поточного авторизованого користувача.

А.9 Основні маршрути сервісів

```
router.post('/register', register);
router.post('/login', login);
router.get('/verify', verify);
router.use(protect);
router.get('/balance', getBalance);
router.post('/', createTransaction);
router.get('/', getAllTransactions);
router.get('/:id', getTransactionById);
router.put('/:id', updateTransaction);
router.delete('/:id', deleteTransaction);
router.use(protect);
router.get('/summary', getAssetsSummary);
router.post('/', createAsset);
router.get('/', getAllAssets);
router.get('/:id', getAssetById);
router.put('/:id', updateAsset);
router.delete('/:id', deleteAsset);
```

Наведені маршрути підтверджують поділ системи на відкриту частину авторизації та захищені частини для роботи з транзакціями й активами.

А.10 Фрагмент Docker Compose

```
services:
```

```
api-gateway:
  build: ./api-gateway
  container_name: api-gateway
  ports:
    - "3000:3000"
  depends_on:
    - auth-service
    - transaction-service
    - asset-service
auth-service:
  build: ./auth-service
  container_name: auth-service
  ports:
    - "3001:3001"
  depends_on:
    - mongo-auth
transaction-service:
  build: ./transaction-service
  container_name: transaction-service
  ports:
    - "3002:3002"
  depends_on:
    - mongo-transactions
asset-service:
  build: ./asset-service
  container_name: asset-service
  ports:
    - "3003:3003"
  depends_on:
    - mongo-assets
```

```
  mongo-auth:
    image: mongo:6
    container_name: mongo-auth
    ports:
      - "27017:27017"
  mongo-transactions:
    image: mongo:6
    container_name: mongo-transactions
    ports:
      - "27018:27017"
  mongo-assets:
    image: mongo:6
    container_name: mongo-assets
    ports:
      - "27019:27017"
networks:
  app-network:
    driver: bridge
```

Фрагмент підтверджує запуск чотирьох Node.js-сервісів і трьох MongoDB-контейнерів у межах одного контейнеризованого середовища.

А.11 Приклад конфігураційних змінних середовища

У проєкті використовується загальний файл .env.example, який показує приклад змінних середовища. Водночас у docker-compose.yml для кожного сервісу підключається окремий .env файл: api-gateway/.env, auth-service/.env, transaction-service/.env та asset-service/.env. Тому під час фактичного запуску значення з прикладу потрібно розподілити між відповідними файлами сервісів.

Приклад файлу api-gateway/.env:

```
PORT=3000
AUTH_SERVICE_URL=http://auth-service:3001
TRANSACTION_SERVICE_URL=http://transaction-service:3002
```

ASSET_SERVICE_URL=http://asset-service:3003

Приклад файлу auth-service/.env:

PORT=3001

MONGO_URI=mongodb://mongo-auth:27017/auth_db

JWT_SECRET=your_super_secret_key_12345

JWT_EXPIRES_IN=7d

Приклад файлу transaction-service/.env:

PORT=3002

MONGO_URI=mongodb://mongo-transactions:27017/transactions_db

JWT_SECRET=your_super_secret_key_12345

Приклад файлу asset-service/.env:

PORT=3003

MONGO_URI=mongodb://mongo-assets:27017/assets_db

JWT_SECRET=your_super_secret_key_12345

Наведені значення є прикладами конфігурації без реальних секретних даних. Фактичні .env файли не публікуються, оскільки вони можуть містити службові або секретні параметри запуску. Для коректної роботи JWT-захисту значення JWT_SECRET має бути однаковим в Auth Service, Transaction Service та Asset Service.

ДОДАТОК Б

Репозиторій програмного проєкту та інструкція запуску

У додатку Б наведено посилання на репозиторій програмного проєкту, короткий опис структури, перелік основних сервісів, порядок запуску через Docker Compose та базову перевірку працездатності після запуску.

Б.1 Посилання на репозиторій

https://github.com/timkoks/personal-assets-monitor

Репозиторій містить серверну частину програмного забезпечення для моніторингу персональних активів, реалізовану на основі Node.js, Express.js, MongoDB, Mongoose, JWT, Docker Compose та REST API.

Б.2 Структура репозиторію

```
personal-assets-monitor-main/
├── api-gateway/
├── auth-service/
├── transaction-service/
├── asset-service/
├── docker-compose.yml
├── .env.example
├── .gitignore
└── README.md
```

Основні папки відповідають окремим мікросервісам. Кожен сервіс має власний Dockerfile, package.json, серверний файл, маршрути, контролери та моделі або middleware відповідно до своєї ролі.

Б.3 Основні сервіси програмної системи

Компонент	Порт	Призначення
API Gateway	3000	єдина точка входу та проксування запитів
Auth Service	3001	реєстрація, логін, формування JWT
Transaction Service	3002	доходи, витрати, транзакції та баланс
Asset Service	3003	персональні активи та зведення за категоріями

mongo-auth 27017 база даних користувачів

mongo-transactions 27018 база даних транзакцій

mongo-assets 27019 база даних активів

Б.4 Підготовка змінних середовища

Перед запуском програмної системи потрібно підготувати окремі .env файли для кожного сервісу. У docker-compose.yml передбачено підключення таких файлів: api-gateway/.env, auth-service/.env, transaction-service/.env та asset-service/.env. Загальний файл .env.example використовується як довідковий приклад, але для фактичного запуску змінні середовища потрібно розмістити відповідно до того, які назви очікує код кожного сервісу.

Для API Gateway використовуються змінні PORT, AUTH_SERVICE_URL, TRANSACTION_SERVICE_URL та ASSET_SERVICE_URL. Приклад:

PORT=3000

AUTH_SERVICE_URL=http://auth-service:3001

TRANSACTION_SERVICE_URL=http://transaction-service:3002

ASSET_SERVICE_URL=http://asset-service:3003

Для Auth Service використовуються змінні PORT, MONGO_URI, JWT_SECRET та JWT_EXPIRES_IN. Приклад:

PORT=3001

MONGO_URI=mongodb://mongo-auth:27017/auth_db

JWT_SECRET=your_super_secret_key_12345

JWT_EXPIRES_IN=7d

Для Transaction Service використовуються змінні PORT, MONGO_URI та JWT_SECRET. Приклад:

PORT=3002

MONGO_URI=mongodb://mongo-transactions:27017/transactions_db

JWT_SECRET=your_super_secret_key_12345

Для Asset Service використовуються змінні PORT, MONGO_URI та JWT_SECRET. Приклад:

PORT=3003

MONGO_URI=mongodb://mongo-assets:27017/assets_db

JWT_SECRET=your_super_secret_key_12345

Секретні значення не публікуються у відкритому вигляді. У пояснювальній записці наведено тільки приклади, достатні для розуміння структури конфігурації. Значення JWT_SECRET повинно збігатися в сервісі авторизації, сервісі транзакцій і сервісі активів, оскільки Auth Service формує JWT-токен, а Transaction Service та Asset Service перевіряють його під час виконання захищених запитів.

Б.5 Перевірка працездатності

Після запуску спочатку перевіряється API Gateway:

GET http://localhost:3000/health

Далі основні функції системи перевіряються через Postman:

POST http://localhost:3000/api/auth/auth/register

POST http://localhost:3000/api/auth/auth/login

POST http://localhost:3000/api/transactions/transactions

GET http://localhost:3000/api/transactions/transactions/balance

POST http://localhost:3000/api/assets/assets

GET http://localhost:3000/api/assets/assets/summary

Для захищених маршрутів використовується JWT-токен, отриманий після входу користувача:

Authorization: Bearer JWT_TOKEN

Під час перевірки було підтверджено створення користувача, вхід до системи, додавання доходу, додавання витрати, розрахунок балансу, створення активу та отримання зведення активів за категоріями.

Б.6 Примітка щодо конфігурації

Фактичні .env файли не публікуються в репозиторії та не виносяться в додатки, оскільки вони можуть містити службові або секретні параметри. Для демонстрації структури налаштувань використовується файл .env.example.