БАКАЛАВРСЬКА РОБОТА

БР.ІП – 12.00.00.000 ПЗ
Група ІП-22-4

Путінцев Кирило

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Путінцев Кирило Сергійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка крипто-гаманця засобами Java Spring
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____К. С. Путінцев._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Дмитрик Тарас Богданович, асистент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.___

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

***ЗАТВЕРДЖУЮ*:**
Зав. кафедрою ***ІПЗ***
доц. В.В. Бандура
“ ” 2026 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Путінцев Кирило Сергійович

(прізвище, ім’я, по-батькові)

1. ***Тема проекту (роботи)*** ***“ Розробка крипто-гамания засобами Java Spring ”***
керівник проєкту (роботи) Дмитрик Тарас Богданович, асистент

затверджені наказом закладу вищої освіти від “ 21 ” квітня 2026 р. № 179/7

2. ***Строк подання студентом проєкту (роботи)*** 10 червня 2026 р.

3. ***Вихідні дані до проєкту (роботи)*** Результати і матеріали отримані під час проходження переддипломної практики

4. ***Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)***

1. Огляд існуючих рішень та теоретичних основ проєктування криптовалютних гаманців
2. Аналіз вимог та постановка задачі
3. Проектування системи крипто-гаманця Web Meta Wave
4. Реалізація проєкту крипто-гаманця Web Meta Wave
5. Тестування та впровадження

5. ***Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)***

1. Діаграма варіантів використання системи Web Meta Wave (рис. 2.1, ст. 21)
2. Діаграма послідовності процесу реєстрації користувача (рис. 2.2, ст. 23)
3. Діаграма компонентів архітектури Web Meta Wave (рис. 3.1, ст. 36)
4. Структура проєкту Web Meta Wave в IDE (рис. 4.1, ст. 46)
5. Сторінка реєстрації Web Meta Wave (рис. 4.2, ст. 49)

6. ***Консультанти розділів проекту (роботи)***

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***

7. ***Дата видачі завдання*** ***28 березня 2026***

Керівник _____ Дмитрик Т.Б.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Путінцев К. С.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	***Назва етапів дипломного проєкту (роботи)***	***Строк виконання етапів проєкту***	***Примітка***
1	Визначення та обґрунтування теми роботи	28.03.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	30.03.2026	виконано
3	Побудова моделі або алгоритму власного рішення	10.04.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	25.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	20.05.2026	виконано

Студент – дипломник _____ Путінцев К. С.
(підпис) (розшифровка підпису)

Керівник роботи _____ Дмитрик Т. Б.
(підпис) (розшифровка підпис)

АНОТАЦІЯ

Бакалаврська робота містить 72 сторінки, 37 рисунків, список використаних джерел із 21 найменувань, 2 додатки.

Метою роботи є розробка крипто-гаманця засобами Java Spring, що забезпечує безпечне зберігання та управління криптовалютними активами користувачів.

Об'єкт дослідження: процеси розробки та функціонування серверного застосунку крипто-гаманця.

Предмет дослідження: методи та інструменти розробки крипто-гаманця засобами Java Spring, включаючи проєктування архітектури, структури бази даних та механізмів криптографічного захисту.

Результати дослідження: розроблено серверний застосунок крипто-гаманця з використанням Java та Spring Boot із підтримкою операцій із криптовалютними активами та механізмами автентифікації й авторизації користувачів.

Перший розділ охоплює аналіз як предметної області, так і існуючих системних рішень для криптовалют.

У другому розділі окреслено вимоги та обсяг системи.

У третьому розділі представлені проектні рішення для архітектури та структури баз даних, що використовуються в архітектурі.

У четвертому розділі представлено практичну реалізацію застосунку.

У п'ятому розділі представлено тестування та розгортання розробленої системи. Розроблений криптогаманець відповідає сучасним стандартам безпеки, надійності та функціональності в сучасних програмних застосунках для управління цифровими активами.

КЛЮЧОВІ СЛОВА: КРИПТО-ГАМАНЕЦЬ, JAVA, SPRING BOOT, КРИПТОВАЛЮТА, БЕЗПЕКА ДАНИХ, MYSQL, АВТЕНТИФІКАЦІЯ, БЛОКЧЕЙН.

ANNOTATION

The bachelor's thesis contains 72 pages, 37 figures, a list of used sources with 21 names, 2 appendices.

The purpose of the work is to develop a crypto wallet using Java Spring tools, which provides secure storage and management of users' cryptocurrency assets.

Object of research: the processes of development and functioning of a crypto wallet server application.

Subject of research: methods and tools for developing a crypto wallet using Java Spring tools, including the design of architecture, database structure and cryptographic protection mechanisms.

Research results: a crypto wallet server application was developed using Java and Spring Boot with support for operations with cryptocurrency assets and user authentication and authorization mechanisms.

The first section covers the analysis of both the subject area and existing system solutions for cryptocurrencies.

The second section outlines the requirements and scope of the system.

The third section presents design solutions for the architecture and database structure used in the architecture.

The fourth section presents the practical implementation of the application.

The fifth section presents the testing and deployment of the developed system. The developed crypto wallet meets modern standards of security, reliability and functionality in modern software applications for managing digital assets.

KEYWORDS: CRYPTO-WALLET, JAVA, SPRING BOOT, CRYPTOCURRENCY, DATA SECURITY, MYSQL, AUTHENTICATION, BLOCKCHAIN.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — Application Programming Interface

БД — База даних

ПЗ — Програмне забезпечення

ПЗ — Пояснювальна записка

СУБД — Система управління базами даних

UML — Unified Modeling Language

HTTP — HyperText Transfer Protocol

HTTPS — HyperText Transfer Protocol Secure

REST — Representational State Transfer

JSON — JavaScript Object Notation

SQL — Structured Query Language

ORM — Object-Relational Mapping

MVC — Model-View-Controller

CI/CD — Continuous Integration / Continuous Deployment

JVM — Java Virtual Machine

JPA — Java Persistence API

SHA — Secure Hash Algorithm

AES — Advanced Encryption Standard

SSL — Secure Sockets Layer

TLS — Transport Layer Security

P2P — Peer-to-Peer

CRUD — Create, Read, Update, Delete

DTO — Data Transfer Object

ВСТУП

Цифрова економіка росте, тому користувачі потребують зручних програмних засобів для керування криптовалютою. Ринок криптовалюти зростає. На 2024 рік вартість ринку криптовалюти перевищує два трильйони доларів. Користувачі криптовалютних гаманців — більше 400 мільйонів людей. З цього зрозуміло, що для роботи з криптовалютою потрібні надійні і зручні інструменти.

Актуальність теми зумовлена необхідністю розробки централізованого серверного застосунку криптогаманця Web Meta Wave, який поєднує сучасні стандарти захисту даних, стабільну архітектуру на основі Java Spring Boot та надійне реляційне зберігання даних у MySQL. Серверна частина є ключовим компонентом системи — вона відповідає за автентифікацію користувачів, обробку транзакцій та безпечне централізоване зберігання даних.

Метою роботи є проєктування та реалізація криптогаманця Web Meta Wave засобами Java Spring, що забезпечує безпечне управління криптовалютними активами користувачів через захищений серверний застосунок із централізованим зберіганням даних на сервері MySQL.

Завданнями дослідження є:

— проведення аналізу предметної області криптовалютних систем та існуючих програмних рішень;

— формулювання функціональних та нефункціональних вимог до системи;

— проєктування архітектури серверного застосунку Web Meta Wave та структури бази даних MySQL;

— реалізація ключових модулів системи: автентифікації, управління гаманцями та обробки транзакцій;

— проведення тестування системи та розгортання застосунку на хмарній платформі Google Cloud Run.

Об'єктом дослідження є процеси розробки та функціонування централізованого серверного застосунку криптогаманця Web Meta Wave,

спрямованого на забезпечення безпечного управління криптовалютними активами користувачів.

Предметом дослідження є методи та інструменти розробки серверної частини криптогаманця Web Meta Wave засобами Java Spring Boot, включаючи проєктування архітектури системи, реалізацію механізмів захисту даних, моделювання структури реляційної бази даних MySQL та інтеграцію із зовнішніми сервісами CoinMarketCap API та LiqPay.

Методами дослідження є об'єктно-орієнтоване проєктування, UML-моделювання, модульне та інтеграційне тестування, аналіз алгоритмів захисту даних, а також порівняльний аналіз існуючих програмних рішень у сфері управління криптовалютними активами.

Практичне значення результатів полягає у розробці функціонального серверного застосунку криптогаманця Web Meta Wave, який може використовуватись як самостійний продукт або як основа для подальшого розширення — зокрема у напрямку підтримки децентралізованого зберігання даних та додаткових блокчейн-мереж.

Структура роботи. Бакалаврська робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ КРИПТОВАЛЮТНИХ ГАМАНЦІВ

1.1. Аналіз сучасного стану ринку криптовалютних гаманців

Швидке розповсюдження криптовалют як способу зберігати та переказувати кошти призвело до появи окремої групи програм - криптовалютних гаманців. З 2009 року, коли запрацювала мережа Bitcoin, і вже тоді було кілька сотень програм для управління цифровими грошима, а сьогодні їх тисячі. Але більшість програм орієнтуються лише на тих, хто вже розуміє, як це працює [1]. Інші програми не мають багатьох важливих функцій, потрібних для повного користування криптовалютою.

Найбільш поширені на ринку - централізовані гаманці. У цих сервісах компанія зберігає всю інформацію про рахунки і всі операції на своїх серверах. З централізованими гаманцями зручно, проте користувач залежить від того, наскільки надійна і безпечна компанія.

Якщо подивитися на основні сервіси на ринку, то можна зрозуміти їхні плюси та мінуси.

Binance — найбільша у світі криптовалютна біржа за обсягом торгів. Binance має власний гаманець у системі. Він дозволяє працювати з сотнями валют, швидко міняти одну на іншу, перекидати активи між різними мережами. Але гаманець більше підходить людям, які торгують. Я вважаю, звичайна людина може відчути, що інтерфейс може бути перевантаженим для недосвідченого користувача. І для реєстрації обов’язково треба підтвердити особу (KYC) [2].

Coinbase — один з найвідоміших централізованих гаманців. Coinbase робить сервіс доступним для широкого кола людей. Coinbase простий у використанні і дозволяє поповнювати рахунок банківською карткою. Coinbase підтримує небагато криптовалют і бере велику комісію за перекази. [3]

Blockchain.com - платформа, яка існує вже давно. Платформа дає простий гаманець: можна зберігати, відправляти і отримувати криптовалюту. Платформа

підтримує лише кілька активів. Платформа не дозволяє легко обмінювати валюти чи перекидати їх між різними мережами [4].

Trust Wallet - мобільний гаманець, який підтримує велику кількість різних токенів. В нього можна додавати власні токени, але він тільки для телефону і не має повноцінного серверного API [5].

Порівняння цих платформ та моєї розробки Web Meta Wave розглянемо в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз функціональних характеристик криптовалютних гаманців

Характеристика	**Binance**	**Coinbase**	**Blockchain.com**	**Trust Wallet**	**Web Meta Wave**
Тип зберігання даних	Централізований	Централізований	Централізований	Локальний	Централізований
Підтримка токенів	350+	100+	5+	1000+	Необмежена (CoinMarketCap API)
Поповнення через платіжні системи	Так	Так	Ні	Ні	Так (LiqPay)
Переказ між користувачами	Так	Так	Так	Так	Так
Свап криптовалют	Так	Так	Ні	Так	Так
Брідж криптовалют	Так	Ні	Ні	Так	Так
Історія транзакцій	Так	Так	Так	Так	Так
Підтвердження реєстрації через email	Ні (KYC)	Так	Так	Ні	Так
Новини гаманця	Ні	Ні	Ні	Ні	Так

З даних таблиці 1.1 видно, що Web Meta Wave стоїть окремо від інших рішень. Web Meta Wave підключив CoinMarketCap API, тому підтримує токени

без обмежень. Web Meta Wave бере фіксовану і прозору комісію. Web Meta Wave ще показує новини. Таку суміш можливостей важко знайти у конкурентів [6].

Що стосується активних користувачів на 2024 рік, їхня кількість для кожної з платформ порівняно на рисунку 1.1.

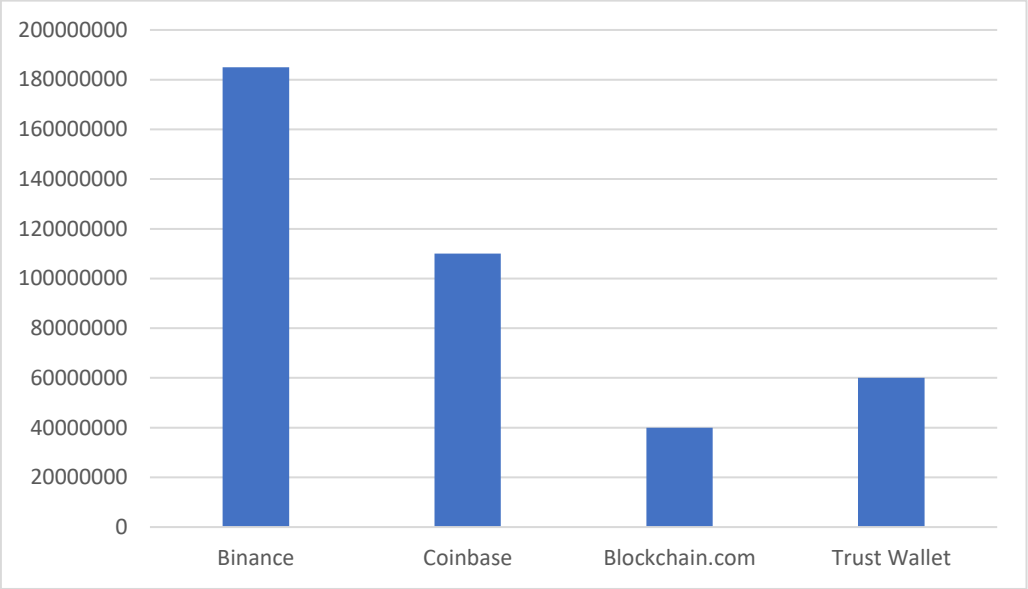


Рисунок 1.1 — Діаграма порівняння кількості активних користувачів криптовалютних платформ (млн осіб, 2024 р.)

Аналітичні дані показують, що ринок переповнений різними пропозиціями. Але серед цих пропозицій немає жодної, що об’єднує все в одному. Потрібна підтримка токенів без обмежень, потрібен зв’язок з LiqPay, потрібна проста фіксована комісія, потрібна можливість інформувати користувачів. Web Meta Wave має цю цінність, бо поєднує всі ці риси.

1.2. Основні поняття та визначення предметної області

Щоб зрозуміти, як працює програма Web Meta Wave і які особливості Web Meta Wave, треба знати головні терміни. Криптовалюта – це цифрові гроші, які не залежать від банку чи держави. Складні алгоритми захищають криптовалюту, і кожна одиниця криптовалюти – це унікальний запис, який не підробити. Перекази криптовалюти між людьми йдуть без банків і без платіжних систем [1].

Токен — це цифровий актив, створений на вже існуючому блокчейні. Токен відрізняється від криптовалюти: токен не має власного блокчейну, токен працює в розумному контракті іншої мережі [7]. У Web Meta Wave користувач може додати будь-який токен у гаманець користувача, просто ввівши ідентифікатор токену з CoinMarketCap.

Рисунок 1.2 – Ідентифікатор токену на сайті CoinMarketCap

Блокчейн — це розподілена база даних, у якій блоки йдуть один за одним. У кожному блоці записані підтверджені операції. Дані в блокчейні не можна змінити, бо кожен блок має хеш попереднього блоку. Якщо змінити будь-який блок, треба переобчислити весь ланцюг [1].

Криптовалютний гаманець — програмний застосунок для зберігання інформації про баланси, формування та підписання транзакцій, а також взаємодії з блокчейн-мережею або централізованим сервером. Розрізняють два типи:

– Децентралізований — зберігає приватні ключі локально на пристрої. Користувач повністю контролює активи, але й несе повну відповідальність за збереження ключів.

– Централізований — зберігає дані на серверах провайдера. Зручніший у використанні та допускає відновлення доступу, однак вимагає довіри до

оператора. Web Meta Wave належить до цього типу: усі дані зберігаються у реляційній базі MySQL на сервері застосунку.

Транзакція – це коли один користувач передає цифровий актив іншому. Web Meta Wave записує кожну транзакцію в MySQL. Запис показує, хто відправляє, хто отримує, скільки, яку операцію і коли це було. Система має чотири типи транзакцій. Переказ – це передача між користувачами з фіксованою комісією. Поповнення – це надходження грошей у гаманець через LiqPay. Свап – це обмін одного активу на інший всередині системи, з фіксованою комісією. Брідж – це передача активу між різними мережами, з фіксованою комісією.

Свап (swap) - це обмін одного цифрового активу на інший за актуальним ринковим курсом. Цей курс визначається на основі даних з CoinMarketCap API [6].

Брідж (bridge) дозволяє перенести токен з однієї блокчейн-мережі в іншу. У Web Meta Wave брідж - це операція всередині застосунку з фіксованою комісією.

Spring MVC виступає як архітектурний патерн Model-View-Controller. Саме на цьому ґрунтується серверна ланка Web Meta Wave: контролери тут приймають HTTP-запити від користувача, сервісний шар доглядає за бізнес-логікою, а Thymeleaf готує HTML-відповідь. Загалом це монолітна архітектура, де backend і frontend не розмежовані на окремі сервіси, а є частиною одного застосунку [8].

CoinMarketCap API - це відкритий програмний інтерфейс, який надає інформацію про криптовалюти: назву, скорочення, поточний курс, загальну вартість тощо. Web Meta Wave користується CoinMarketCap API. Тоді користувачі швидко додають токени в гаманець і отримують актуальні курси для свапу.

LiqPay - українська платіжна система, яка надає API для отримання платежів за допомогою банківських карток [9]. У Web Meta Wave LiqPay – єдиний спосіб додати в гаманець гроші.

1.3. Сучасні технології та тенденції у розробці криптовалютних застосунків

Розвиток індустрії криптовалютних застосунків визначається кількома технологічними тенденціями, які безпосередньо вплинули на архітектурні рішення Web Meta Wave.

Інтеграція з зовнішніми API. Сучасні криптовалютні застосунки відмовляються від власних баз даних курсів на користь спеціалізованих агрегаторів. CoinMarketCap, CoinGecko та аналогічні платформи надають актуальні ринкові дані в режимі реального часу [6]. Підтримка власної системи збору ринкових даних вимагає значних інфраструктурних витрат: підключення до численних бірж, обробка потокових даних та відмовостійкість. Делегування цього завдання спеціалізованому агрегатору дозволяє зосередитися на бізнес-логіці застосунку. Web Meta Wave використовує CoinMarketCap API для отримання актуальних курсів токенів без власної інфраструктури збору даних.

Централізоване зберігання даних на реляційних СУБД. Попри популярність децентралізованих рішень, більшість комерційних гаманців зберігає дані користувачів та транзакцій у реляційних базах. Реляційна модель гарантує цілісність даних, підтримку складних запитів і перевірені механізми резервного копіювання [10]. Підтримка ACID-властивостей транзакцій важлива для фінансових застосунків: атомарність гарантує, що операція переказу або виконається повністю, або не виконається взагалі — часткове списання коштів без їх зарахування отримувачу виключено. У Web Meta Wave цю роль виконує MySQL — одна з найпоширеніших СУБД із широкою підтримкою в екосистемі Java Spring Boot.

Мікросервісна та монолітна архітектури. Великі платформи на кшталт Binance використовують мікросервісну архітектуру заради масштабованості [11]. Мікросервісний підхід дозволяє незалежно масштабувати окремі компоненти та підвищує відмовостійкість — збій одного сервісу не впливає на роботу інших. Проте він суттєво ускладнює розробку: виникає потреба в оркестрації

контейнерів, міжсервісній комунікації та розподіленому трасуванні запитів. Для застосунку рівня Web Meta Wave монолітна архітектура на основі Spring Boot є обґрунтованим вибором: вона спрощує розгортання, налагодження та супровід без надмірного ускладнення інфраструктури. За значного зростання навантаження перехід до мікросервісів є логічним наступним кроком.

Безпека автентифікації. Для серверних застосунків із рендерингом на стороні сервера галузевим стандартом є Spring Security у поєднанні з HTTPS [12]. Сесійна автентифікація зберігає стан користувача на сервері, звільняючи розробника від реалізації власної логіки управління сесіями. У Web Meta Wave додано ще один рівень захисту — обов'язкову верифікацію нових користувачів через електронну пошту при реєстрації. Одноразовий код генерується на стороні сервера та передається виключно на зареєстровану адресу, що унеможливлює реєстрацію з чужими email-адресами та знижує ризик масової автоматизованої реєстрації фіктивних облікових записів.

Інтеграція з платіжними системами. Поєднання криптовалютних гаманців із традиційними платіжними системами дозволяє поповнювати рахунок фіатними коштами — без цього застосунок замкнений на внутрішніх переказах [9]. Інтеграція з платіжними шлюзами вимагає реалізації механізму зворотного виклику (callback): платіжна система надсилає підтвердження оплати на визначений URL застосунку, після чого система зараховує кошти на рахунок користувача. Автентичність callback-запиту підтверджується через верифікацію цифрового підпису із секретним ключем, що захищає від підробки підтверджень. Web Meta Wave реалізує поповнення рахунку через LiqPay — платіжну систему, орієнтовану на український ринок.

Контейнеризація та хмарне розгортання. Сучасні застосунки дедалі частіше розгортаються у Docker-контейнерах на хмарних платформах [17]. Контейнеризація вирішує проблему залежності від середовища виконання: застосунок разом із усіма залежностями упакований в ізольований контейнер із однаковою поведінкою у середовищах розробки, тестування та продакшн. Хмарні платформи надають керовану інфраструктуру для запуску контейнерів,

автоматичне масштабування та моніторинг. Web Meta Wave розгорнуто на платформі Railway з Docker-контейнером та хмарною базою даних MySQL відповідно до сучасних практик розгортання веб-застосунків.

Таблиця 1.2

Порівняння технологічного стеку криптовалютних платформ

Характеристика	Binance	Coinbase	Trust Wallet	Web Meta Wave
Серверна платформа	Java, Go	Ruby, Go	—	Java Spring Boot
База даних	MySQL, NoSQL	PostgreSQL	Локальна	MySQL
Автентифікація	JWT, 2FA	JWT, 2FA	Локальна	Spring Security + Email
Платіжна інтеграція	Visa, SWIFT	Visa, ACH	—	LiqPay
API курсів	Власний	Власний	CoinGecko	CoinMarketCap
Архітектура	Мікросервісна	Мікросервісна	Мобільна	Моноліт

У таблиці 1.2 наведено порівняння технологічного стеку Web Meta Wave із підходами розглянутих платформ. Java Spring Boot — одна з найпоширеніших платформ для серверної розробки у фінансовому секторі [8]; MySQL закриває потребу у надійному реляційному сховищі з підтримкою транзакційності та цілісності даних [10]. Стек відповідає галузевим стандартам.

1.4. Висновки по розділу

У першому розділі розглянуто предметну область криптовалютних гаманців та проаналізовано наявні програмні рішення.

Порівняльний аналіз централізованих платформ — Binance, Coinbase, Blockchain.com, Trust Wallet — показав, що жодна з них не поєднує одночасно необмежену підтримку токенів, інтеграцію з українською платіжною системою, фіксовану прозору комісію та засоби інформування користувачів. Binance та Coinbase пропонують широкий функціонал, проте орієнтовані на міжнародний

ринок і не підтримують поповнення через LiqPay. Trust Wallet є некастодіальним гаманцем без централізованого управління даними. Blockchain.com має обмежений набір підтримуваних активів. Цей збіг відсутніх функцій і визначив практичну мету Web Meta Wave як централізованого гаманця з прозорою моделлю комісій та орієнтацією на потреби українського ринку.

Виділено ключові терміни предметної області: криптовалюта, блокчейн, токен, транзакція, свап, брідж, Spring MVC, Spring Security та централізований гаманець. Розмежування понять свапу та бріджу є архітектурно важливим: свап конвертує різні токени в межах однієї мережі з обчисленням курсу через зовнішній API, брідж переміщує один токен між різними мережами без зміни його виду. Web Meta Wave зберігає дані реляційно у MySQL під сесійною автентифікацією — на відміну від некастодіальних рішень, де приватні ключі зберігаються на стороні користувача.

Огляд технологічних тенденцій підтвердив відповідність обраного стеку галузевим підходам. Java Spring Boot є поширеною платформою для серверної розробки у фінансовому секторі завдяки зрілій екосистемі, вбудованій підтримці безпеки та інструментам інтеграції із зовнішніми сервісами [8]. MySQL підтримує реляційну цілісність даних та ACID-транзакції — обов'язкову вимогу для фінансових застосунків. Інтеграції з CoinMarketCap API та LiqPay відповідають практиці сучасних комерційних гаманців і усувають потребу у власній інфраструктурі для збору ринкових даних та обробки платежів [6, 9, 10]. Контейнеризація засобами Docker та розгортання на Railway відповідають сучасним практикам DevOps і гарантують однакову поведінку застосунку незалежно від конфігурації цільового сервера [17].

Результати аналізу предметної області, існуючих рішень та технологічних тенденцій сформували вимоги до функціоналу системи та обґрунтували вибір технологічного стеку — основу для формулювання функціональних та нефункціональних вимог у наступному розділі.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Збір та аналіз вимог до криптовалютного гаманця Web Meta Wave

Збір вимог є одним із визначальних етапів життєвого циклу розробки програмного забезпечення. Якість зібраних вимог прямо впливає на архітектурні рішення, що закладаються у систему, та на кінцевий результат розробки [13]. У рамках підготовки до проєктування Web Meta Wave проаналізовано потреби цільової аудиторії, визначено ролі користувачів та побудовано сценарії їх взаємодії із застосунком.

Цільова аудиторія

Web Meta Wave створена для широкого кола людей, яким потрібен надійний і простий у користуванні інструмент для криптовалют. На відміну від професійних торгових майданчиків, мій застосунок не вимагає від вас знань у криптографії чи тонкощів блокчейну. Інтуїтивно зрозумілий веб-інтерфейс, реалізований засобами Thymeleaf у складі монолітного Spring Boot застосунку.

У системі передбачено дві ролі.

Перша — звичайний користувач, тобто фізична особа, яка керує своїми активами. Такий користувач реєструється в системі, може поповнити гаманець через LiqPay, додати токени (за допомогою CoinMarketCap API), зробити переказ, провести конвертацію активів за допомогою свапу або бріджу, а також стежити за балансом і транзакціями. При цьому доступ до даних інших учасників йому закритий, кожен працює лише зі своїм гаманцем.

Друга роль — адміністратор. Це уповноважена особа, відповідальна за бездоганну роботу платформи. Вона має повний контроль: може переглядати транзакції будь-якого користувача, управляти обліковими записами (блокувати порушників, надавати права), вести новинний розділ. Права адміністратора передаються іншим адміністратором через спеціальний інтерфейс.

Сценарії використання

Виходячи з цих ролей, ми визначили весь спектр взаємодії з Web Meta Wave.

Функціонал для обох категорій користувачів відображено на діаграмі варіантів використання (див. рис. 2.1).

Рисунок 2.1 — Діаграма варіантів використання системи Web Meta Wave

2.2 Формулювання функціональних та нефункціональних вимог

Вимоги до програмної системи традиційно поділяють на дві категорії: функціональні, що визначають конкретні можливості застосунку, та нефункціональні, що встановлюють якісні характеристики його роботи. Чітке формулювання обох категорій необхідне для коректного проєктування архітектури системи та подальшої оцінки якості розробленого продукту [13].

Функціональні вимоги

Функціональні вимоги до Web Meta Wave сформульовано на основі аналізу сценаріїв використання та згруповано за функціональними модулями застосунку.

1. Модуль автентифікації

Модуль автентифікації відповідає за безпечний вхід користувачів до системи та захист облікових записів. Уся реаліція спирається на Spring Security і підтвердження email-адреси [12].

Модель користувача містить такі атрибути:

- ім'я користувача;
- електронна адреса;
- пароль (зберігається у хешованому вигляді BCrypt);
- роль (USER / ADMIN);
- статус облікового запису (активний / заблокований).

Через модуль автентифікації реалізовано:

- реєстрацію нового користувача з надсиланням підтверджувального коду на електронну пошту;
- активацію облікового запису після введення одноразового коду;
- вхід у систему й відкриття захищеної сесії;
- контроль автентичності при зверненні до будь-яких захищених ендпоінтів через фільтр-ланцюг Spring Security.

На рисунку 2.2 наведено діаграму послідовності процесу реєстрації користувача у системі Web Meta Wave.

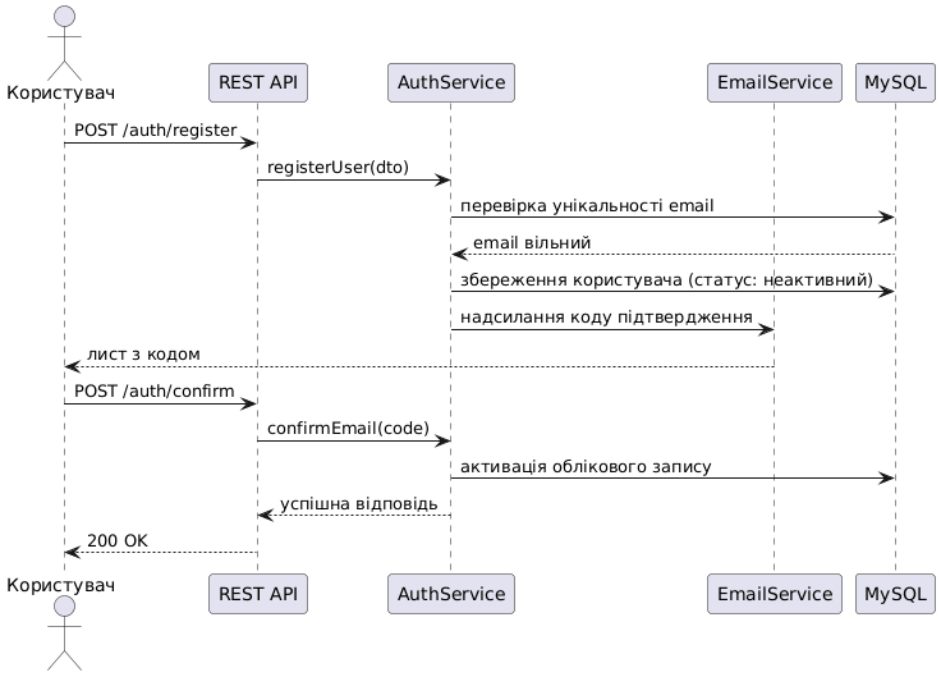


Рисунок 2.2 — Діаграма послідовності процесу реєстрації користувача

2. Модуль гаманця

Модуль гаманця — центральний компонент системи Web Meta Wave, що охоплює повний цикл управління криптовалютними активами користувача: від поповнення рахунку до виконання складних фінансових операцій.

Модель гаманця містить такі атрибути:

- ідентифікатор користувача;
- перелік доданих токенів;
- поточний баланс по кожному токену;
- дата створення гаманця.

Через модуль гаманця реалізовано:

- перегляд поточного балансу у розрізі кожного токена з актуальним курсом у USD через CoinMarketCap API [6];
- поповнення гаманця фіатними коштами через платіжну систему LiqPay [9];
- переказ активів між користувачами системи з фіксованою комісією;
- свап криптовалют за актуальним курсом CoinMarketCap з фіксованою комісією;

- брідж криптовалют з мінімальною сумою операції 10 USD та фіксованою комісією;
- додавання довільного токена до гаманця за його ідентифікатором з CoinMarketCap.

На рисунку 2.3 наведено діаграму послідовності процесу виконання переказу між користувачами.

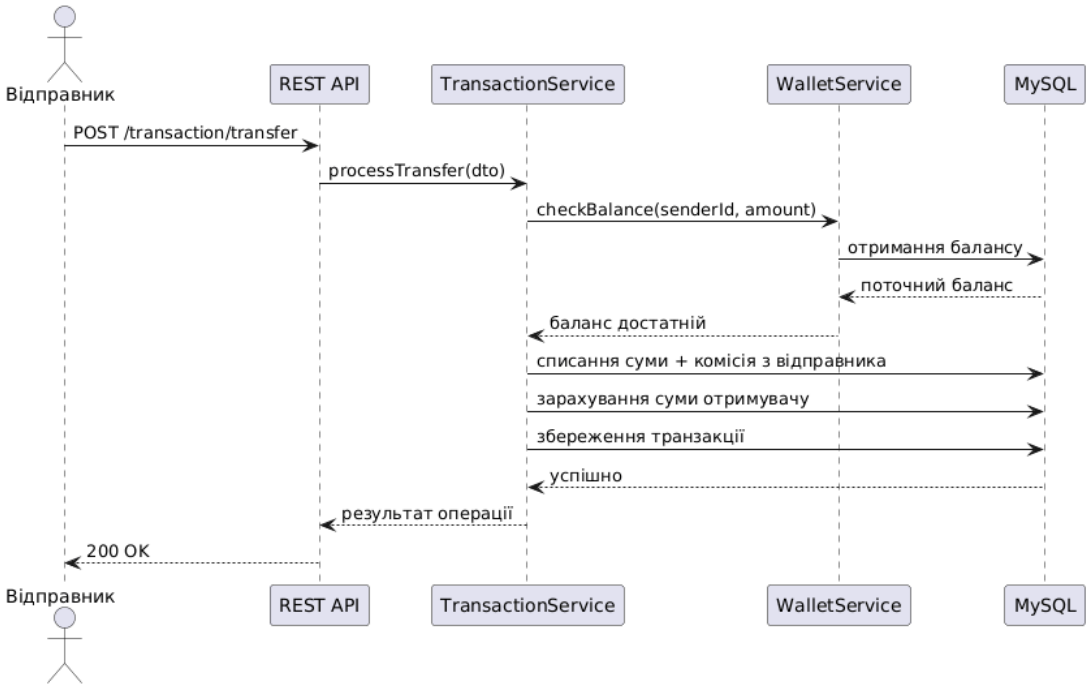


Рисунок 2.3 — Діаграма послідовності процесу переказу коштів

3. Модуль транзакцій

Модуль транзакцій фіксує та відображає повну історію фінансових операцій користувача. Кожна транзакція записується у базу даних MySQL із повним набором атрибутів [10].

Модель транзакції містить такі поля:

- тип операції (переказ / поповнення / свап / брідж);
- ідентифікатор відправника;
- ідентифікатор отримувача;
- сума операції;
- розмір комісії;

- валюта операції;
- дата та час виконання;
- статус (успішно / помилка).

Через модуль транзакцій реалізовано:

- збереження кожної фінансової операції у базі даних незалежно від її результату;
- перегляд хронологічної історії транзакцій користувача з фільтрацією за типом та датою.

На рисунку 2.4 наведено діаграму діяльності процесу свапу криптовалют.

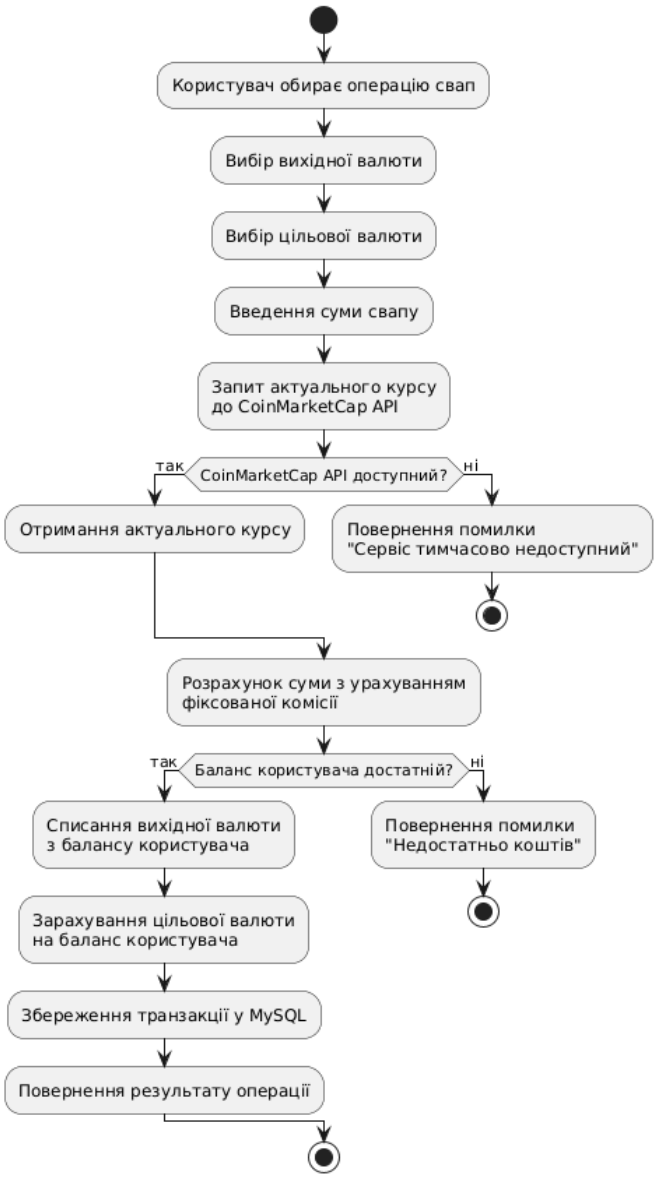


Рисунок 2.4 — Діаграма діяльності процесу свапу криптовалют

На рисунку 2.5 наведено діаграму станів транзакції у системі Web Meta Wave.

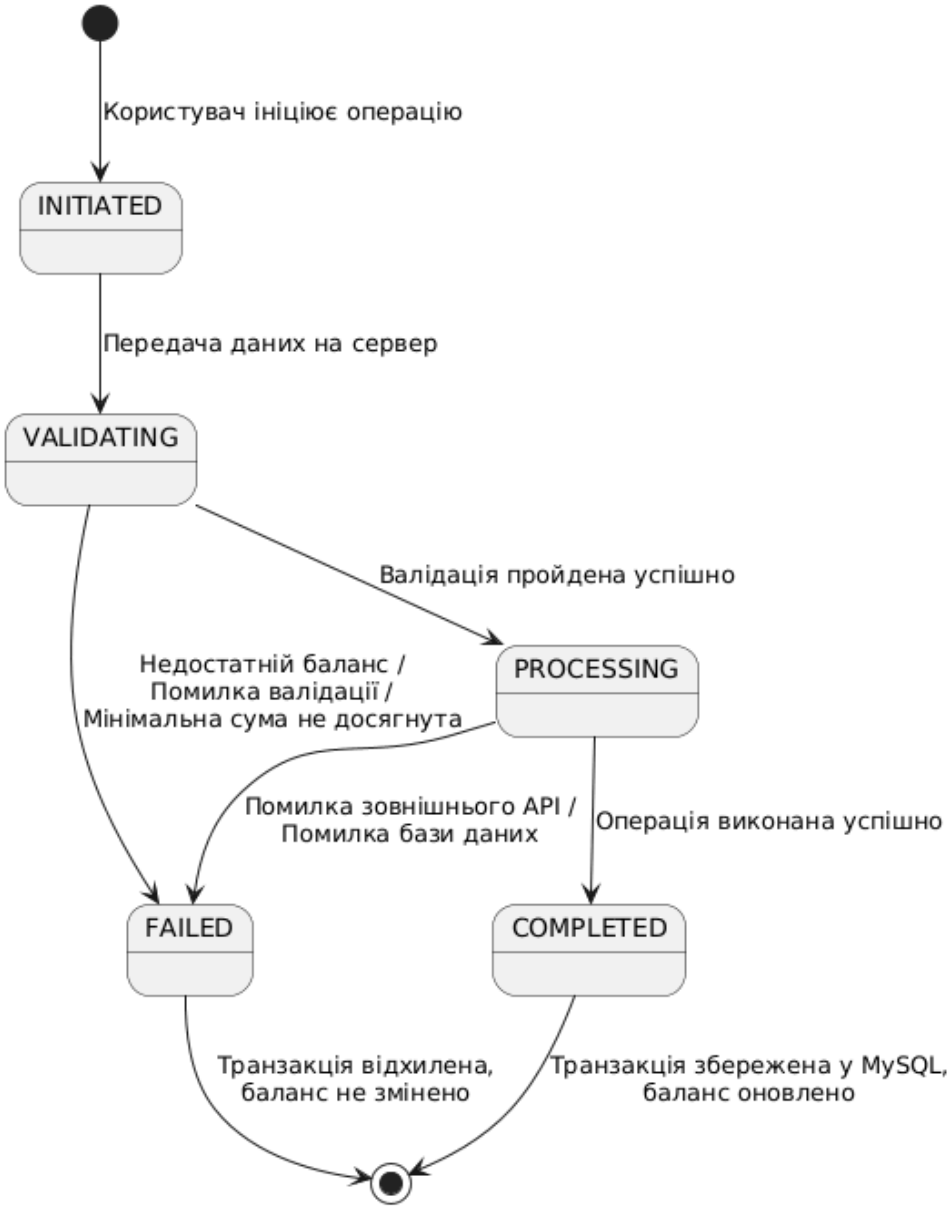


Рисунок 2.5 — Діаграма станів транзакції у системі Web Meta Wave

4. Модуль новин

Модуль новин інформує користувачів про актуальні події платформи Web Meta Wave. Публікації створюються та керуються адміністраторами системи.

Модель новини містить такі поля:

- заголовок публікації;
- текст публікації;

- дата публікації;
- ідентифікатор автора (адміністратора).

Через модуль новин реалізовано:

- перегляд переліку новин у хронологічному порядку для всіх користувачів;
- додавання нової публікації адміністратором;
- редагування існуючої публікації адміністратором;
- видалення публікації адміністратором.

5. Адміністративний модуль

Адміністративний модуль надає уповноваженим особам інструменти контролю за функціонуванням платформи. Доступ до адміністративних функцій обмежено роллю ADMIN, що перевіряється на рівні Spring Security при кожному запиті [12].

Система розмежування прав визначає дві ролі:

- USER — має доступ виключно до власних даних та операцій з гаманцем;
- ADMIN — має повний доступ до даних усіх користувачів та адміністративних функцій.

Через адміністративний модуль реалізовано:

- перегляд повного переліку транзакцій усіх користувачів з фільтрацією;
- перегляд списку зареєстрованих користувачів із їхнім статусом;
- надання або відкликання прав адміністратора;
- блокування облікового запису користувача;
- повне управління розділом новин платформи.

На рисунку 2.6 наведено діаграму діяльності процесу блокування користувача адміністратором.

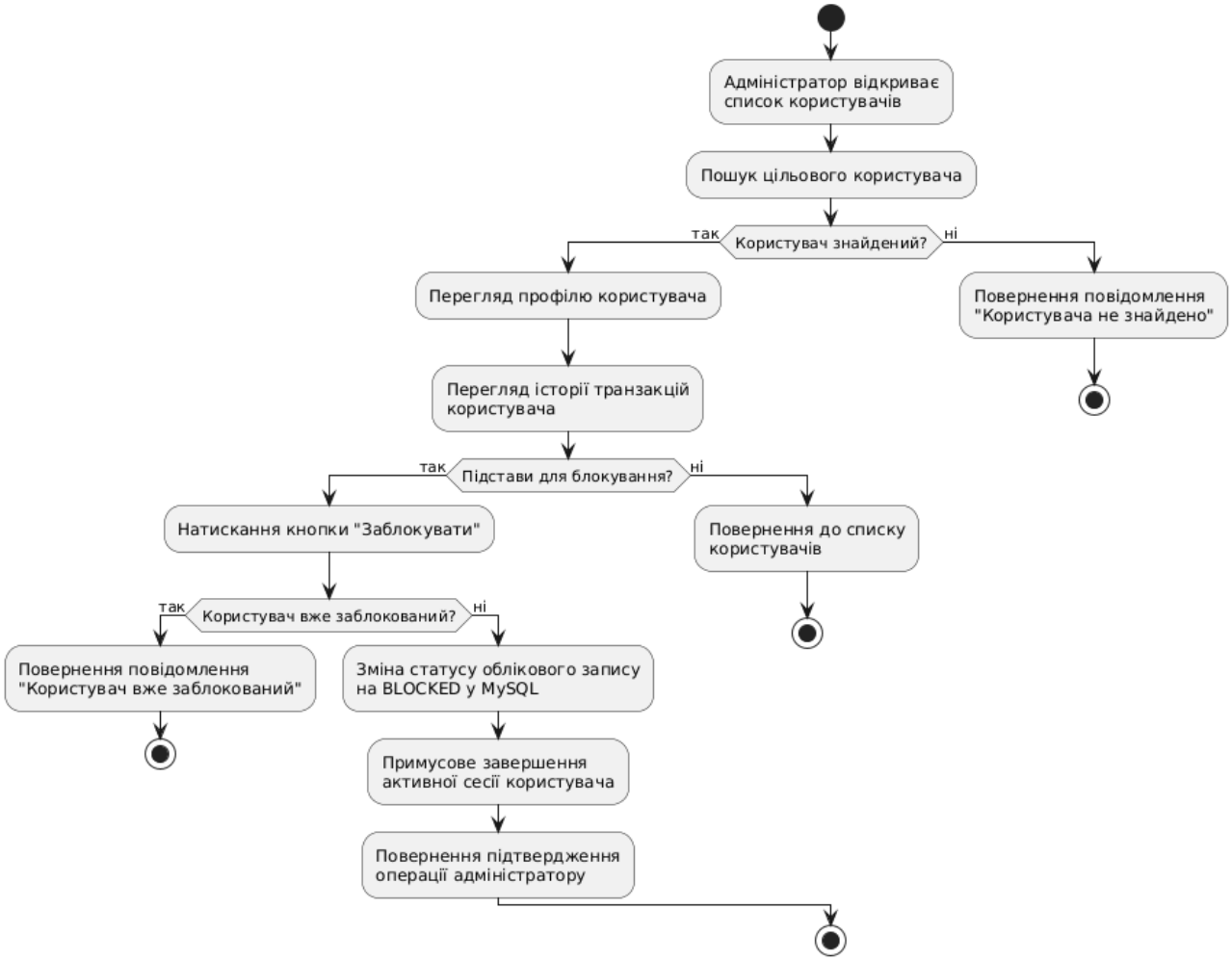


Рисунок 2.6 — Діаграма діяльності процесу блокування користувача адміністратором

Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи Web Meta Wave та критерії оцінки її роботи.

Категорія безпеки

- Хешування паролів. Паролі користувачів зберігаються у базі даних виключно у хешованому вигляді. Застосовується алгоритм BCrypt, що унеможливлює відновлення оригінального пароля навіть у разі компрометації бази даних [12].
- Захищені ендпоінти не обходять цю перевірку. Якщо автентифікація пройшла успішно, сервер формує сесію користувача, а його ідентифікатор клієнт отримує в захищеному cookie JSESSIONID.

- Шифрування передачі даних. Передача даних між клієнтом та сервером здійснюється виключно через HTTPS із використанням SSL/TLS сертифікату.
- Підтвердження реєстрації одноразовим кодом. Реєстрація нового користувача включає обов'язковий етап підтвердження електронної адреси через одноразовий код із терміном дії 15 хвилин.

На рисунку 2.7 наведено діаграму діяльності процесу блокування користувача адміністратором.

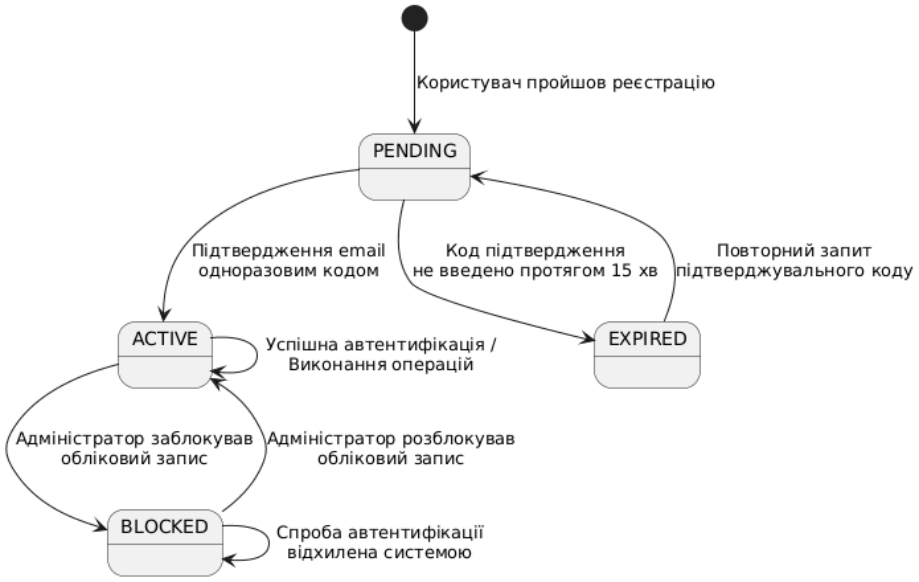


Рисунок 2.7 — Діаграма станів облікового запису користувача

Категорія продуктивності

- Час відповіді API. Час відповіді серверного застосунку на стандартні запити не перевищує 500 мілісекунд для 95-го перцентиля навантаження.

Категорія надійності

- Транзакційність фінансових операцій. Усі фінансові операції у базі даних MySQL виконуються в рамках транзакцій із дотриманням принципів ACID — атомарності, узгодженості, ізольованості та довговічності [10].
- Обробка помилок зовнішніх API. У разі недоступності або помилок зовнішніх сервісів — CoinMarketCap API та LiqPay — система повертає клієнту зрозуміле повідомлення із відповідним HTTP-кодом.

На рисунку 2.8 наведено діаграму діяльності процесу блокування користувача адміністратором.

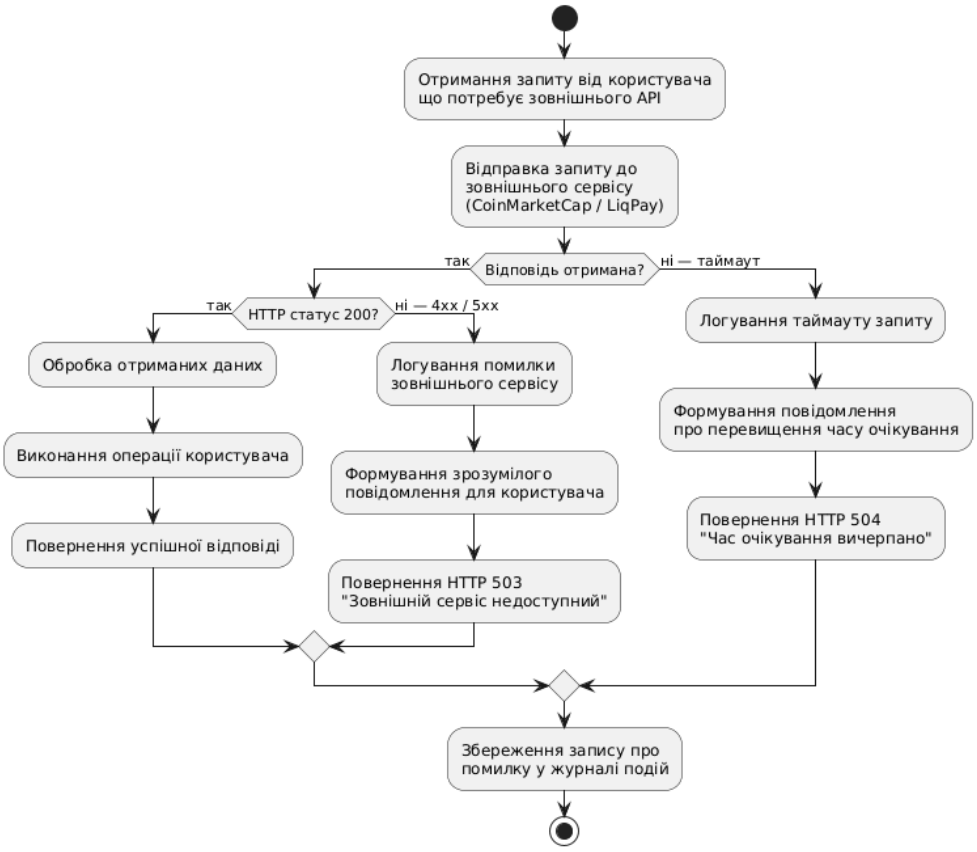


Рисунок 2.8 — Діаграма діяльності системи при обробці помилки зовнішнього API

Категорія масштабованості

- Горизонтальне масштабування. Також архітектура на Spring Boot дозволяє горизонтально масштабувати застосунок. Оскільки сесії тримаються в базі даних MySQL, стан буде доступний для будь-якого екземпляру програми навіть після масштабування [8].

Категорія зручності використання

- Веб-застосунок Web Meta Wave має містити задокументовану структуру MVC-контролерів із описом усіх маршрутів, параметрів запитів та шаблонів Thymeleaf, що забезпечує зручність подальшої підтримки та розширення системи [8].

2.3 Постановка задачі проєктування системи крипто-гаманця Web Meta Wave

Постановка задачі завершує аналіз вимог і слугує вихідним документом для проєктування архітектури системи. На основі зібраних функціональних та нефункціональних вимог сформульовано технічне завдання на розробку застосунку Web Meta Wave.

Опис кінцевого результату

Кінцевим результатом розробки є функціональний монолітний веб-застосунок крипто-гаманця Web Meta Wave, реалізований на платформі Java Spring Boot із використанням архітектурного патерну MVC [8]. Серверна частина побудована на основі Spring Framework, бізнес-логіка зосереджена у сервісному шарі, а веб-інтерфейс формується засобами шаблонізатора Thymeleaf і передається браузеру у вигляді готових HTML-сторінок. Усі дані зберігаються централізовано у реляційній базі даних MySQL [10].

Застосунок охоплює повний цикл управління криптовалютними активами — від реєстрації з підтвердженням через Gmail до виконання фінансових операцій: поповнення рахунку через LiqPay [9], переказу активів між користувачами, свапу та бріджу криптовалют. Актуальні курси отримуються через CoinMarketCap API [6], управління платформою — через вбудовану адміністративну панель.

Межі системи

У таблиці 2.1 наведено розподіл компонентів на внутрішні — ті, що входять до scope розробки — та зовнішні сторонні сервіси.

Таблиця 2.1

Межі системи Web Meta Wave

Компонент	Входить у систему	Є зовнішнім
Серверна частина Java Spring Boot	✓	
База даних Google Cloud SQL (MySQL)	✓	
Веб-інтерфейс Thymeleaf (HTML/CSS)	✓	
Модуль автентифікації Spring Security	✓	

Продовження таблиці 2.1

Модуль управління гаманцями	✓	
Модуль транзакцій (переказ, свап, брідж)	✓	
Адміністративна панель	✓	
Модуль новин	✓	
Docker-контейнер	✓	
Платіжна система LiqPay		✓
API курсів CoinMarketCap		✓
Поштовий сервіс Gmail		✓

Згідно таблиці 2.1, до scope розробки входить повний стек застосунку: серверна частина, веб-інтерфейс, база даних та конфігурація контейнеризації. Зовнішніми по відношенню до Web Meta Wave є платіжна система LiqPay, агрегатор курсів CoinMarketCap та поштовий сервіс Gmail.

Обмеження та припущення

У процесі проєктування та реалізації системи Web Meta Wave діють такі обмеження.

Система є повністю централізованою: усі дані користувачів, гаманців та транзакцій зберігаються у реляційній базі даних MySQL. Децентралізоване зберігання у поточній версії не передбачено.

Операція брідж має мінімальне порогове значення суми — 10 USD. Спроба ініціювати операцію на меншу суму відхиляється із відповідним повідомленням, а факт спроби фіксується у базі даних зі статусом FAILED.

Комісії розраховуються пропорційно до суми операції із фіксованим коефіцієнтом: переказ — 1.8%, свап — 1%, брідж — 1.5%. Коефіцієнти визначено

на рівні моделі Gas і не змінюються у поточній версії застосунку.

Актуальні курси криптовалют отримуються виключно через CoinMarketCap API. У разі недоступності сервісу відповідні операції повертають повідомлення про помилку без зміни балансу користувача.

Для надсилання підтверджувальних листів під час реєстрації використовується Gmail через механізм application password. Недоступність поштового сервісу унеможливлює завершення реєстрації нового користувача.

2.4 Висновки по розділу

У другому розділі зібрано, класифіковано та формалізовано вимоги до монолітного веб-застосунку крипто-гаманця Web Meta Wave.

У підрозділі 2.1 визначено дві ролі користувачів системи — звичайний користувач та адміністратор — з чітко розмежованими наборами функціональних можливостей. Звичайний користувач має доступ до управління власними криптовалютними активами, адміністратор — до контролю за функціонуванням платформи через вбудовану адміністративну панель. Побудовано діаграму варіантів використання, що охоплює 13 сценаріїв взаємодії з системою та показує межі доступу кожної ролі.

У підрозділі 2.2 сформульовано 15 функціональних вимог, згрупованих за п'ятьма модулями: автентифікація, гаманець, транзакції, новини та адміністрування. Для відображення ключових процесів побудовано вісім UML-діаграм: послідовності реєстрації та переказу коштів, діяльності свапу, блокування користувача та обробки помилок зовнішніх API, а також діаграми станів транзакції та облікового запису. Визначено дев'ять нефункціональних вимог у категоріях безпеки, продуктивності, надійності, масштабованості та зручності використання із конкретними метриками оцінки.

У підрозділі 2.3 визначено кінцевий результат розробки, межі системи та діючі обмеження. Web Meta Wave — монолітний веб-застосунок на основі Spring Boot із Thymeleaf-інтерфейсом та централізованим зберіганням даних у MySQL на Google Cloud SQL. До складу системи входять серверна частина, веб-інтерфейс, модулі автентифікації, гаманця, транзакцій, новин, адміністративна панель та конфігурація Docker-контейнера. Зовнішніми є LiqPay, CoinMarketCap API та Gmail.

Визначені вимоги, діаграми та обмеження формують основу для проєктування архітектури системи у наступному розділі.

РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ КРИПТО-ГАМАНЦЯ WEB META WAVE

3.1 Розробка архітектури серверного застосунку

Архітектура застосунку Web Meta Wave побудована на основі монолітного підходу з використанням патерну Model-View-Controller (MVC), реалізованого засобами Spring Boot [8]. Монолітна архітектура відповідає масштабу застосунку: вона спрощує розгортання та налагодження і не створює мережевих затримок між компонентами, характерних для мікросервісного підходу [11].

Застосунок структурований за шістьма пакетами відповідальності відповідно до принципу розділення concerns.

Пакет config містить сім конфігураційних класів. SecurityConfig визначає правила доступу до маршрутів та налаштування Spring Security. AuthHandler обробляє успішну автентифікацію та перевіряє статус облікового запису. CoinMarketCupConfig та EmailConfig налаштовують підключення до зовнішніх сервісів. WalletConfig містить початкові налаштування гаманця. CustomAuthenticationEntryPoint обробляє неавторизовані запити. AppConfig містить загальні налаштування застосунку.

Пакет controller містить десять контролерів, що обробляють HTTP-запити та передають дані у Thymeleaf-шаблони. Кожен контролер відповідає за окремий функціональний модуль:

LoginController — реєстрація та вхід, MainWindowController — головна сторінка гаманця, BridgeController — операції брідж, SwapController — операції свап, SendController — перекази між користувачами, BuyCryptoController — поповнення через LiqPay, ImportTokenController — додавання токенів через CoinMarketCap, AdminController — адміністративна панель, ProfileController — профіль користувача, NotificationsController — перегляд новин.

Пакет service містить шістнадцять сервісних класів, де зосереджена вся бізнес-логіка застосунку. Сервісний шар — єдине місце, де виконується валідація

параметрів, розрахунок комісій та взаємодія з зовнішніми API. Ключові сервіси: BridgeService, SwapService, SendService реалізують основні фінансові операції; BuyCryptoService та LiqPayService відповідають за інтеграцію з платіжною системою; CoinMarketCupService отримує актуальні курси токенів; BanUsersService керує ролями та блокуванням користувачів; UserDetailsServiceImpl реалізує інтерфейс Spring Security для завантаження даних користувача під час автентифікації.

Пакет repository містить шість JPA-репозиторіїв для взаємодії з базою даних MySQL через Spring Data JPA без написання SQL-запитів вручну [14]. Репозиторії реалізовані як інтерфейси, що розширюють JpaRepository; ORM-маппінг між Java-об'єктами та таблицями виконує Hibernate [15].

Пакет model містить дев'ять класів, що представляють сутності предметної області. Шість із них є JPA-сутностями з анотацією @Entity: CustomUser, Balance, Transaction, MetaToken, Network, Notifications. Два класи є переліками: UserRole визначає ролі користувача (USER, ADMIN, BANNED), Status — результат транзакції (CORRECT, FAILED). Клас Gas є службовим об'єктом без анотації @Entity — він не зберігається у базі даних, а використовується для зберігання фіксованих коефіцієнтів комісій під час розрахунків у сервісному шарі. Для усіх Entity-класів застосовується бібліотека Lombok [16]: анотації @Getter, @Setter, @Builder, @NoArgsConstructor, @AllArgsConstructor усувають необхідність написання шаблонного коду.

Пакет mail містить два класи, що реалізують надсилання електронних листів. EmailService — інтерфейс, що визначає контракт поштового сервісу. EmailServiceImpl — конкретна реалізація на основі Spring Mail із використанням Gmail SMTP. Поштовий сервіс використовується виключно під час реєстрації для надсилання одноразового підтверджувального коду. Конфігурація підключення до Gmail визначається у класі EmailConfig пакету config.

На рисунку 3.1 наведено діаграму компонентів архітектури застосунку Web Meta Wave.

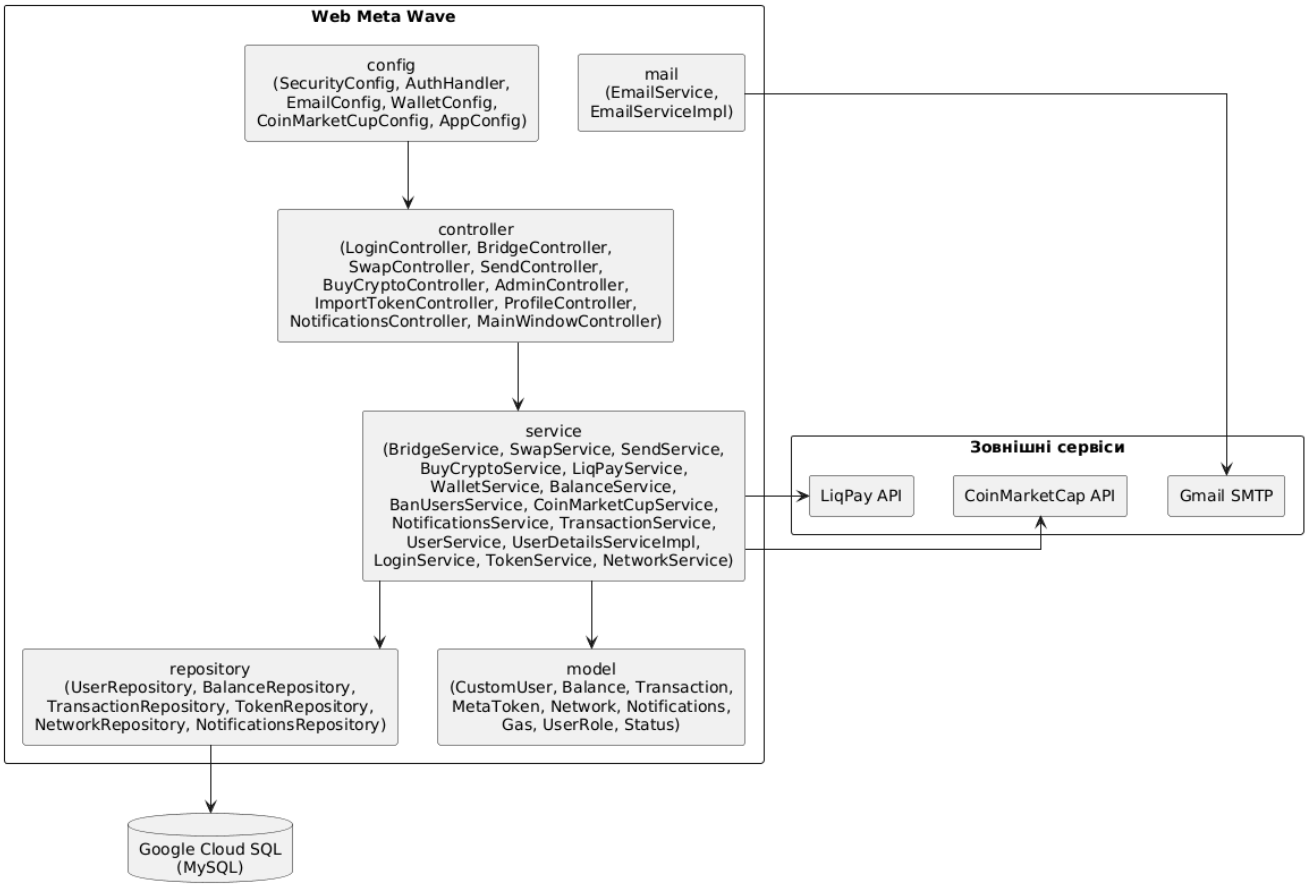


Рисунок 3.1 — Діаграма компонентів архітектури Web Meta Wave

Безпеку застосунку реалізовано засобами Spring Security, налаштованими у класі **SecurityConfig** [12]. Усі HTTP-запити проходять через фільтр-ланцюг Spring Security, який перевіряє автентифікацію користувача та його роль до надання доступу до захищених маршрутів. Після успішної автентифікації AuthHandler додатково перевіряє, чи не має користувач роль BANNED — у такому разі його перенаправляють на сторінку входу з відповідним повідомленням. Публічно доступними є лише сторінки реєстрації, входу та статичні ресурси. Адміністративні маршрути /profile/allTxns, /profile/allUsers/users, /profile/allUsers/banned та /profile/allUsers/admins доступні виключно користувачам з роллю ADMIN.

Конфігурацію фільтр-ланцюга наведено у **додатку А1**.

На рисунку 3.2 наведено діаграму розгортання системи Web Meta Wave.

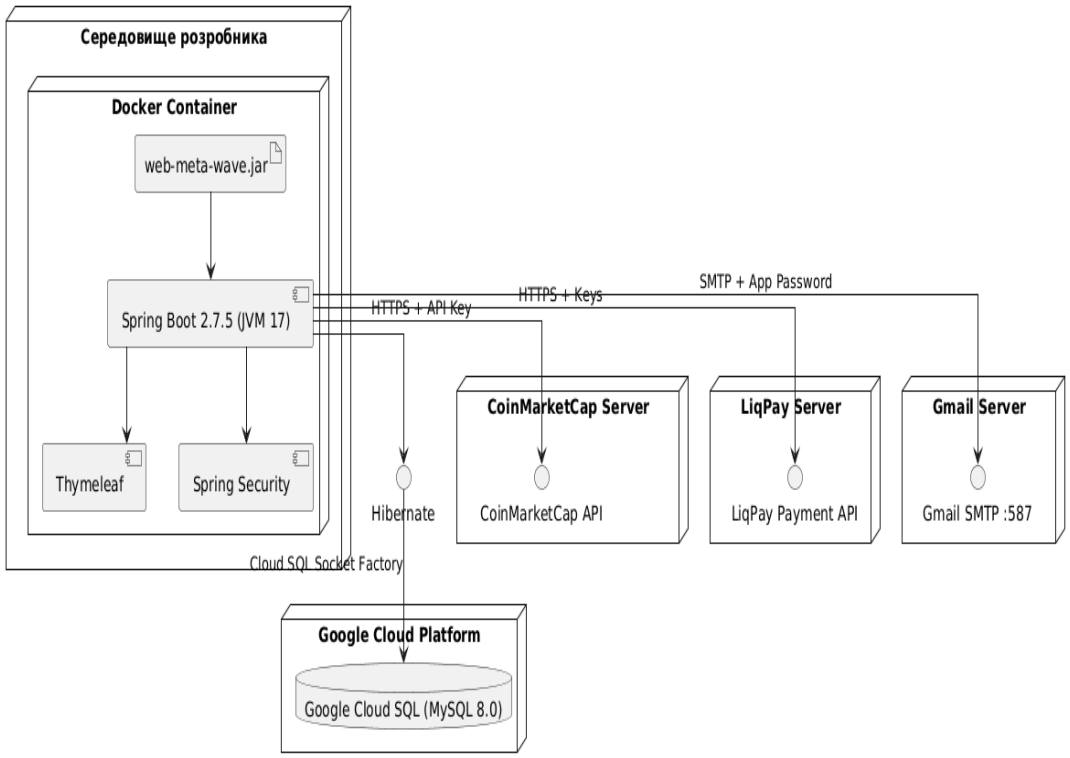


Рисунок 3.2 — Діаграма розгортання системи Web Meta Wave

Діаграма розгортання показує фізичне розміщення компонентів системи Web Meta Wave. Застосунок упакований у Docker-контейнер [17]. Серверна частина на основі Spring Boot 2.7.5 працює на JVM версії 17 та підключається до бази даних MySQL через Google Cloud SQL Socket Factory — механізм захищеного з'єднання без відкриття публічного порту бази даних [18]. Взаємодія з зовнішніми сервісами відбувається виключно через HTTPS.

3.2 Моделювання бізнес-процесів та системних компонентів

UML-діаграма класів (рис. 3.3) описує структуру даних та зв'язки між сутностями системи Web Meta Wave. Вона охоплює шість основних Entity-класів, два переліки та один службовий клас.

Ключова архітектурна особливість системи: баланс користувача (Balance) визначається тривимірно — користувачем (CustomUser), токеном (MetaToken) та мережею (Network) одночасно. Це означає, що один і той самий токен може мати різний баланс у різних блокчейн-мережах для одного користувача, що відповідає

реальній логіці роботи з криптовалютами [7].

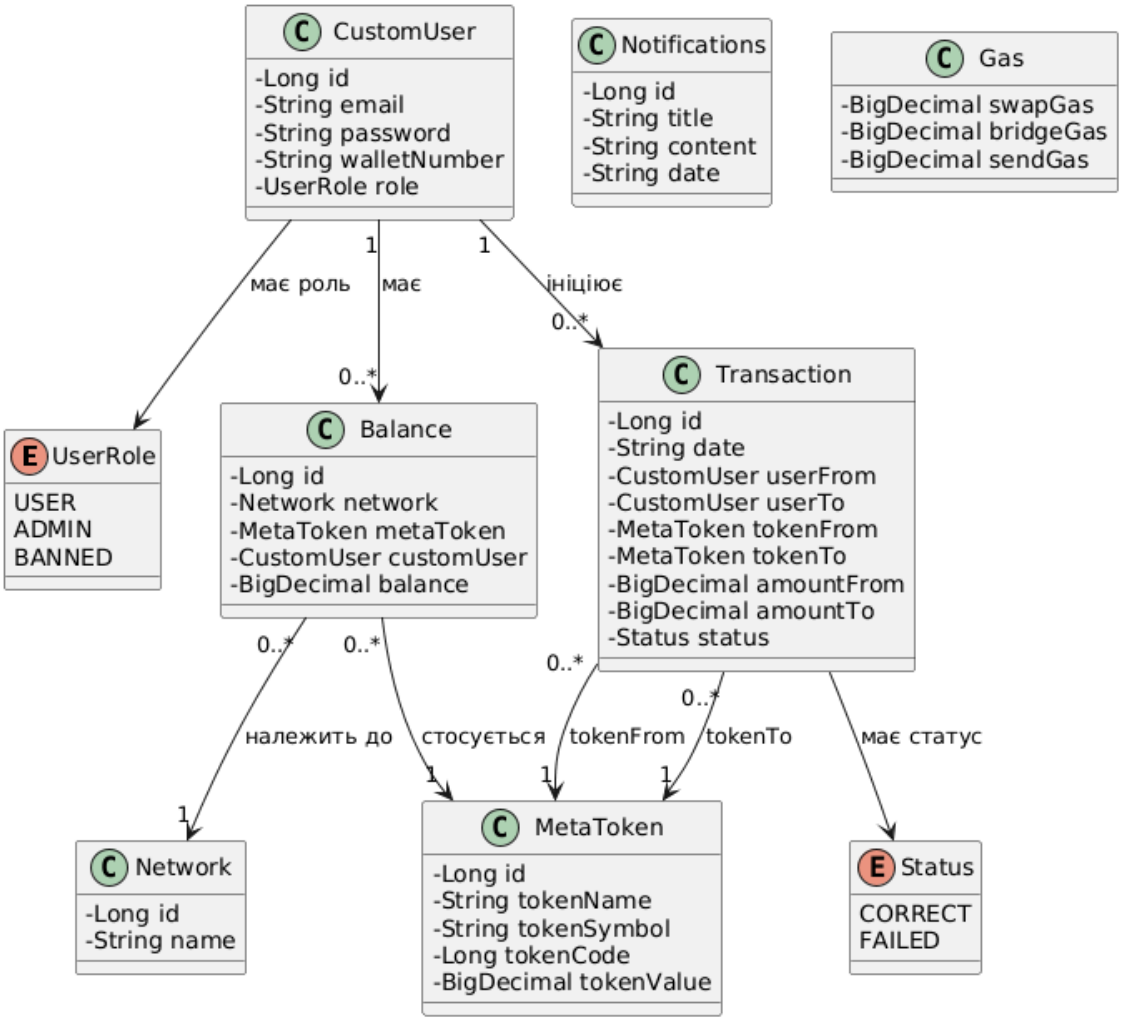


Рисунок 3.3 — Діаграма класів основних сутностей системи Web Meta Wave

Ключова архітектурна особливість системи: баланс користувача (Balance) визначається тривимірно — користувачем (CustomUser), токеном (MetaToken) та мережею (Network) одночасно. Це означає, що один і той самий токен може мати різний баланс у різних блокчейн-мережах для одного користувача, що відповідає реальній логіці роботи з криптовалютами [7].

Сутність Transaction зберігає окремо tokenFrom та tokenTo, що дозволяє єдиною структурою описати всі типи операцій системи. При переказі між користувачами обидва токени однакові, проте відправник та отримувач різні. При свапі — токени різні, а обидва поля користувача посилаються на одну особу. При бріджу — токени однакові, операція виконується в межах одного гаманця між

різними мережами. Поле status типу Status фіксує результат операції — CORRECT або FAILED — що формує повну аудиторську стежку всіх спроб виконання операцій незалежно від їх результату.

Роль користувача реалізована через перелік UserRole з трьома значеннями: USER, ADMIN та BANNED. Блокування реалізовано через присвоєння ролі BANNED, що спрощує логіку перевірки у BanUsersService — перевірка факту блокування зводиться до одного виклику existsByEmailAndRole(email, UserRole.BANNED).

Сутність Notifications є незалежною від інших сутностей — новини не прив'язані до конкретного користувача і доступні всім зареєстрованим користувачам. Управління новинами відбувається виключно через AdminController та NotificationsService.

Клас Gas не є JPA-сутністю та не зберігається у базі даних. Це службовий клас із фіксованими коефіцієнтами комісій: swapGas — 1%, bridgeGas — 1.5%, sendGas — 1.8%. Екземпляр класу створюється безпосередньо у сервісах під час розрахунку комісії для кожної операції.

На рисунку 3.4 наведено UML-діаграму послідовності операції переказу криптовалюти між користувачами системи Web Meta Wave.

Діаграма послідовності відображає взаємодію між компонентами системи під час виконання операції переказу криптовалюти від одного користувача до іншого. Процес охоплює сім послідовних етапів та залучає чотири системних компоненти: браузер користувача, SendController, SendService та репозиторії бази даних.

На першому етапі автентифікований користувач відкриває сторінку переказу — браузер надсилає GET-запит на маршрут /send. SendController завантажує список доступних токенів та мереж з репозиторіїв і передає їх у Thymeleaf-шаблон sendWindow.html для відображення форми.

На другому етапі користувач заповнює форму переказу: вказує адресу отримувача (номер гаманця), обирає токен та мережу, вводить суму. Браузер надсилає POST-запит на маршрут /send/confirm з параметрами форми.

На третьому етапі SendController отримує параметри запиту та передає їх до SendService для виконання бізнес-логіки. SendService послідовно виконує валідацію вхідних даних: перевіряє існування отримувача за номером гаманця через UserRepository, перевіряє що відправник і отримувач є різними особами, перевіряє наявність достатнього балансу у відправника через BalanceRepository з урахуванням комісії.

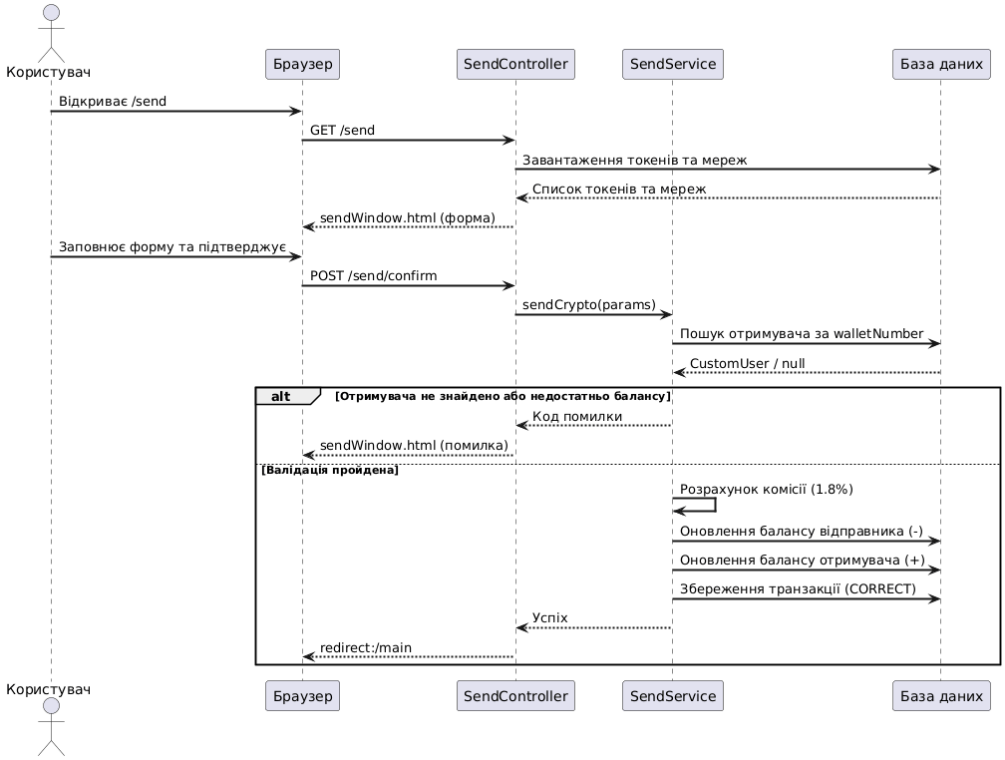


Рисунок 3.4 — Діаграма послідовності операції переказу між користувачами

На четвертому етапі, якщо будь-яка перевірка завершується невдачею, SendService повертає відповідний код помилки до SendController, який додає атрибут помилки до моделі та повертає шаблон sendWindow.html з відповідним повідомленням для користувача.

На п'ятому етапі, якщо всі перевірки пройдено успішно, SendService розраховує суму комісії за формулою: комісія = сума × sendGas, де sendGas = 0.018 (1.8%), що відповідає фіксованому коефіцієнту класу Gas. Фінальна сума що надходить отримувачу розраховується як: сума до зарахування = сума − комісія.

На шостому етапі SendService виконує атомарне оновлення балансів у базі даних: зменшує баланс відправника на повну суму переказу, збільшує баланс отримувача на суму за вирахуванням комісії через BalanceRepository.

На сьомому етапі SendService створює запис транзакції через TransactionRepository зі статусом CORRECT та усіма параметрами операції: відправник, отримувач, токен, мережа, сума, дата. У разі виникнення помилки на будь-якому попередньому етапі транзакція зберігається зі статусом FAILED, що забезпечує повну аудиторську стежку всіх спроб виконання операцій.

Після успішного завершення операції SendController перенаправляє користувача на головну сторінку /main, де відображається оновлений баланс.

Ця сама логіка послідовності застосовується і для операцій свапу та бріджу, з відмінністю у кроці п'ять: при свапі розраховується курс конвертації між двома токенами на основі актуальних значень token_value з таблиці meta_token; при бріджу перевіряється наявність підтримки цільової мережі для обраного токена та виконується переміщення балансу між мережами в межах одного гаманця.

Опис взаємодії компонентів під час операцій свапу та бріджу наведено у таблиці 3.1

Таблиця 3.3

Порівняльна характеристика фінансових операцій системи

Характеристика	**Переказ (Send)**	**Свап (Swap)**	**Брідж (Bridge)**
Відправник та отримувач	Різні користувачі	Один користувач	Один користувач
Вихідний та цільовий токен	Однаковий	Різні	Однаковий
Вихідна та цільова мережа	Однакова	Однакова	Різні
Комісія (Gas)	1.8%	1.0%	1.5%
Перевірка курсу CoinMarketCap	Ні	Так	Ні
Перевірка існування отримувача	Так	Ні	Ні
Запис у таблицю transaction	Так	Так	Так
Поля user_from та user_to	Різні записи	Один запис	Один запис
Поля token_from та token_to	Однаковий запис	Різні записи	Однаковий запис

3.3 Проєктування структури бази даних

База даних системи Web Meta Wave розміщена на хмарній платформі Google Cloud SQL та реалізована засобами реляційної СУБД MySQL [10]. Структура спроєктована відповідно до об'єктно-реляційного маппінгу, що визначається JPA-анотаціями у Entity-класах застосунку. Схема керується автоматично через Hibernate на основі визначених сутностей [15].

База даних Web Meta Wave складається з шести основних таблиць.

Таблиця custom_user зберігає облікові записи користувачів системи. Первинний ключ генерується автоматично за стратегією IDENTITY. Поле role зберігається як рядок через EnumType.STRING і приймає значення USER, ADMIN або BANNED. Поле wallet_number є унікальним ідентифікатором гаманця користувача у системі.

Таблиця network зберігає перелік підтримуваних блокчейн-мереж. Містить лише ідентифікатор та назву мережі — такий підхід дозволяє додавати нові мережі без зміни структури інших таблиць.

Таблиця meta_token зберігає інформацію про криптовалютні токени. Поле token_code є ідентифікатором токена у системі CoinMarketCap і використовується для отримання актуального курсу [6]. Поле token_value зберігає поточний курс токена у доларовому еквіваленті та оновлюється при кожному зверненні до CoinMarketCap API.

Таблиця balance зберігає баланси користувачів у розрізі токенів та мереж. Містить три зовнішні ключі: network_id, token_id та custom_user_id. Поле balance має тип DECIMAL(56,6) — це забезпечує необхідну точність для роботи з криптовалютними значеннями.

Таблиця transaction зберігає повну історію фінансових операцій. Містить посилання на відправника та отримувача, вихідний та цільовий токени, суми операції, дату та статус виконання. Поля token_from_id та token_to_id дозволяють єдиній таблиці охоплювати всі типи операцій — переказ, свап та брідж.

Таблиця notifications зберігає новини платформи, що публікуються адміністраторами. Є незалежною таблицею без зовнішніх ключів.

На рисунку 3.5 наведено ER-діаграму бази даних системи Web Meta Wave.

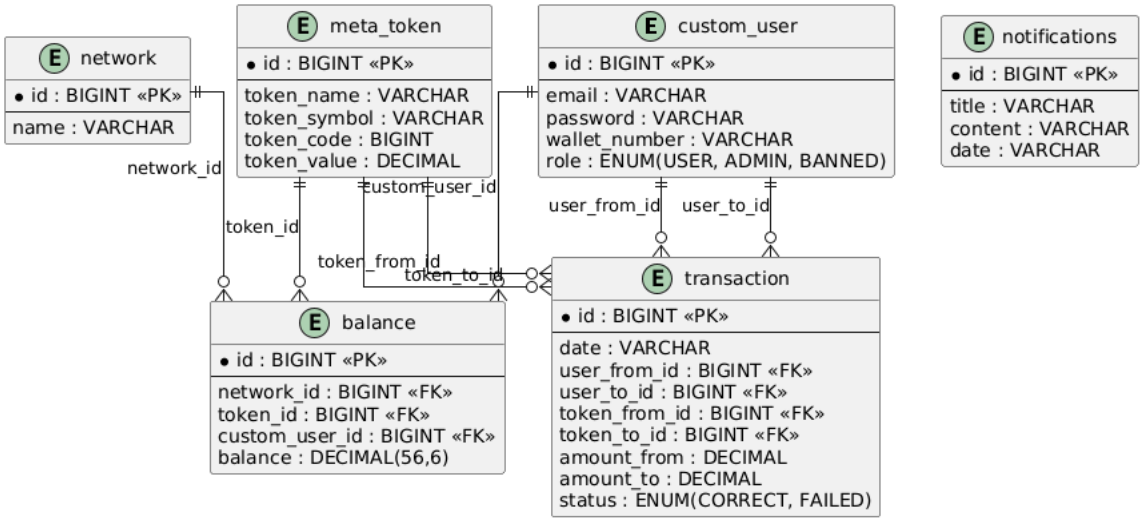


Рисунок 3.5 — ER-діаграма бази даних системи Web Meta Wave

Реляційна модель бази даних Web Meta Wave побудована на основі п'яти зовнішньоключових зв'язків, що підтримують цілісність даних.

Зв'язок **balance → custom_user** — «багато до одного». Один користувач може мати необмежену кількість записів балансу — по одному для кожної комбінації токена та мережі.

Зв'язок **balance → network** — «багато до одного». Одна мережа пов'язана з необмеженою кількістю записів балансу різних користувачів та токенів. Один токен може існувати у кількох мережах з різними балансами для кожного користувача.

Зв'язок **balance → meta_token** — «багато до одного». Один токен може фігурувати у балансах багатьох користувачів у різних мережах одночасно.

Зв'язки **transaction → custom_user** — два зв'язки типу «багато до одного»: user_from_id посилається на відправника, user_to_id — на отримувача. При переказі між користувачами це різні записи; при свапі та бріджу обидва поля посилаються на одного користувача, оскільки операція виконується в межах одного гаманця.

Зв'язки **transaction → meta_token** — два зв'язки типу «багато до одного»: token_from_id та token_to_id. При переказі та бріджу обидва поля посилаються на однаковий токен; при свапі — на різні, що дозволяє описати конвертацію одного активу в інший єдиною структурою транзакції.

3.4 Висновки по розділу

У третьому розділі розроблено повну архітектуру системи Web Meta Wave та спроєктовано структуру бази даних.

У підрозділі 3.1 обґрунтовано вибір монолітної MVC-архітектури на основі Spring Boot [8, 11]. Визначено шість основних пакетів застосунку та описано відповідальність кожного: пакет controller містить десять контролерів, service — шістнадцять сервісних класів, repository — шість JPA-репозиторіїв. Побудовано діаграму компонентів, діаграму пакетів та діаграму розгортання, що відображають взаємодію між модулями системи та зовнішніми сервісами: CoinMarketCap API, LiqPay та Gmail SMTP. Наведено конфігурацію фільтр-ланцюга Spring Security з описом трьох рівнів доступу до маршрутів застосунку [12]. Фільтр-ланцюг перевіряє автентифікацію та роль користувача до надання доступу до захищеного ресурсу, що унеможливлює несанкціонований доступ до адміністративних функцій системи.

У підрозділі 3.2 побудовано UML-діаграму класів шести основних сутностей системи та діаграму послідовності операції переказу між користувачами. Визначено ключові архітектурні особливості: тривимірна модель балансу охоплює користувача, токен та мережу одночасно; єдина структура транзакції описує всі типи операцій через поля tokenFrom та tokenTo; блокування користувача реалізовано через роль BANNED. Клас Gas є службовим об'єктом з фіксованими коефіцієнтами комісій. Наведено повний перелік HTTP-маршрутів застосунку з рівнем доступу для кожного ендпоінту, а також порівняльну характеристику трьох типів фінансових операцій — переказу, свапу та бріджу — за задіяними сутностями, алгоритмом розрахунку комісії та логікою запису

транзакції. Діаграма послідовності відображає семиетапний процес переказу із розгалуженням за результатом валідації вхідних даних.

У підрозділі 3.3 спроєктовано структуру реляційної бази даних MySQL [10] з шести таблиць: custom_user, network, meta_token, balance, transaction та notifications. Побудовано ER-діаграму та наведено детальний опис усіх полів із зазначенням типів та обмежень цілісності. Описано п'ять зовнішньоключових зв'язків між таблицями та обґрунтовано їх логіку. Обґрунтовано вибір типу DECIMAL(56,6) для зберігання балансових значень — типи з плаваючою комою неприйнятні у фінансових застосунках через накопичення похибок округлення. Схема бази даних керується автоматично через Hibernate на основі JPA-анотацій, що підтримує консистентність між об'єктною моделлю та реляційною структурою сховища.

Розроблена архітектура відповідає принципам розділення відповідальності та слабкого зв'язування між компонентами. База даних підтримує цілісність через систему зовнішніх ключів і зберігає повну аудиторську стежку фінансових операцій незалежно від їх результату. Прийняті архітектурні рішення формують основу для практичної реалізації застосунку у наступному розділі.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ КРИПТО-ГАМАНЦЯ WEB META WAVE

4.1 Структура програмного коду та організація модулів

Програмний код застосунку Web Meta Wave організовано відповідно до принципів об'єктно-орієнтованого проєктування та архітектурного патерну MVC. Структура пакетів забезпечує чіткий розподіл відповідальності між компонентами та мінімізує зв'язність між модулями.

Усі класи застосунку розміщено у кореневому пакеті web.meta.wave з розподілом за функціональними підпакетами: config, controller, service, repository, model та mail. Такий підхід групує класи за призначенням і спрощує навігацію по кодовій базі.

На рисунку 4.1 наведено скріншот структури проєкту Web Meta Wave в середовищі розробки.

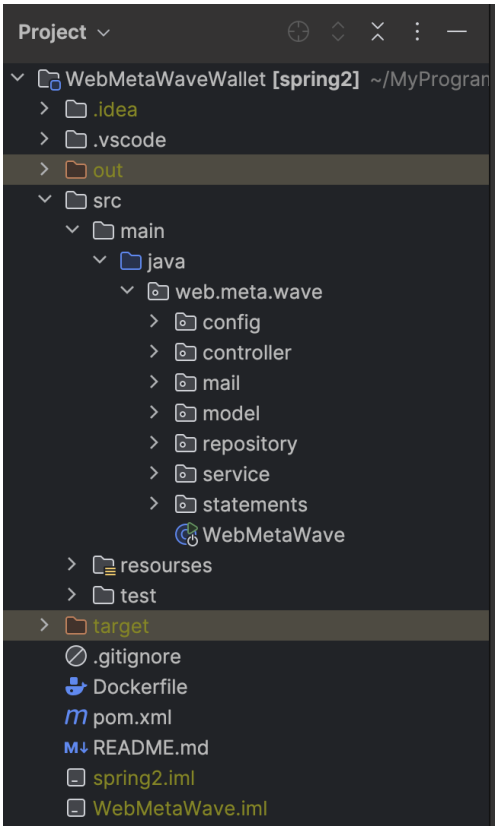


Рисунок 4.1 — Структура проєкту Web Meta Wave в IDE

Центральним класом моделі даних є Entity-клас CustomUser, що представляє обліковий запис користувача системи. У лістингу 4.1 наведено його реалізацію.

Лістинг 4.1 — Entity-клас CustomUser (CustomUser.java)

```
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@ToString
public class CustomUser {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String email;
   private String password;
   private String walletNumber;
   @Enumerated(EnumType.STRING)
   private UserRole role;}
```

Клас анотовано @Entity — це вказує Hibernate на необхідність створення відповідної таблиці у базі даних. Анотація @GeneratedValue(strategy = GenerationType.IDENTITY) делегує генерацію первинного ключа на сторону бази даних. Роль користувача зберігається у вигляді рядка через @Enumerated(EnumType.STRING), що підвищує читабельність даних у базі. Бібліотека Lombok генерує увесь шаблонний код — конструктори, гетери, сетери та toString — через анотації @NoArgsConstructor, @AllArgsConstructor, @Getter, @Setter, @Builder та @ToString.

4.2 Реалізація ключових алгоритмів роботи з криптовалютними транзакціями

Алгоритм генерації коду підтвердження реєстрації

Реєстрація нового користувача у системі Web Meta Wave передбачає обов'язкове підтвердження електронної адреси через одноразовий код. Генерацію коду реалізовано у класі LoginService. У лістингу 4.2 наведено реалізацію методу генерації коду.

Лістинг 4.2 — Генерація коду підтвердження реєстрації (LoginService.java)

```
public static String generateCode() {
    // Character set for code generation
    final String CHARACTERS = loginStatenemts.getCHARACTERS();
    final int CODE_LENGTH = loginStatenemts.getCODE_LENGTH();
    // SecureRandom ensures cryptographically strong randomness
    SecureRandom random = new SecureRandom();
    StringBuilder code = new StringBuilder(CODE_LENGTH);
    for (int i = 0; i < CODE_LENGTH; i++) {
       int index = random.nextInt(CHARACTERS.length());
       code.append(CHARACTERS.charAt(index));
    }
    return code.toString();
}
```

Для генерації коду використовується клас SecureRandom — криптографічно стійкий генератор псевдовипадкових чисел. На відміну від стандартного Random, SecureRandom спирається на джерела ентропії операційної системи, що унеможливлює передбачення згенерованого коду зловмисником. Набір символів та довжина коду визначаються у класі LoginStatenemts, що дозволяє змінювати параметри без модифікації бізнес-логіки.

На рисунку 4.2 та на рисунку 4.3 наведено скріншоти сторінок реєстрації застосунку Web Meta Wave.

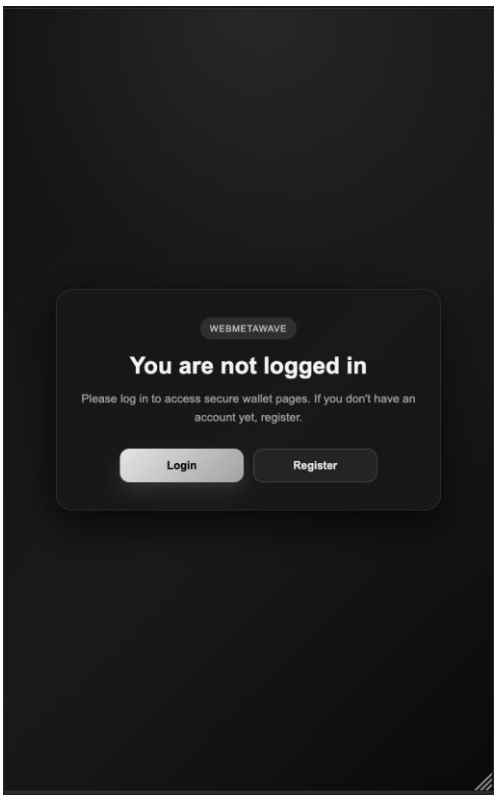


Рисунок 4.2 — Модальне вікно авторизації (Запит на підключення)

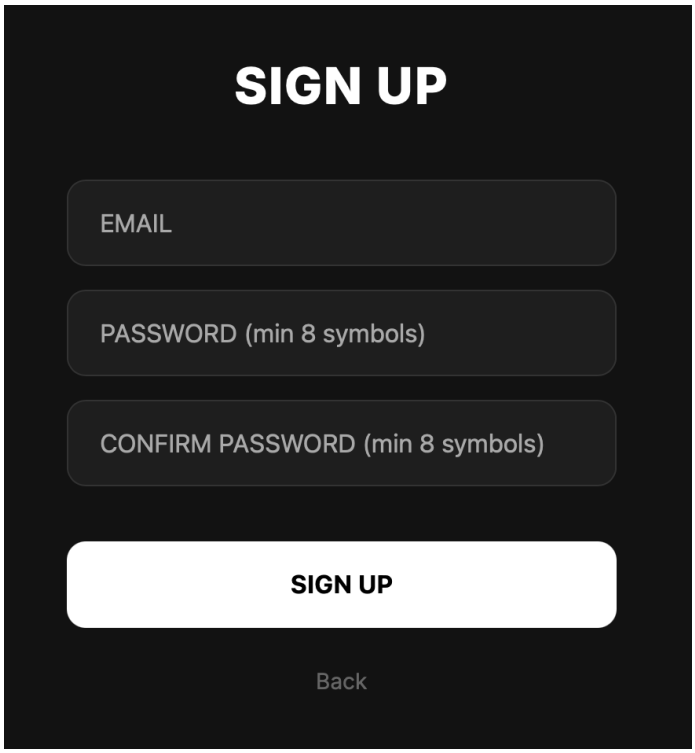


Рисунок 4.3 — Сторінка реєстрації Web Meta Wave

Алгоритм розрахунку комісії Gas

Комісія за виконання фінансових операцій розраховується у методі countGas класу WalletService. Алгоритм є єдиним для всіх типів операцій — переказу, свапу та бріджу — і відрізняється лише коефіцієнтом з класу Gas. У лістингу 4.3 наведено реалізацію методів розрахунку комісії.

Лістинг 4.3 – Розрахунок та комісії Gas (WalletService.java)

```
public BigDecimal countGas(BigDecimal gas,
                           MetaToken tokenFrom,
                           BigDecimal value,
                           BigDecimal ethValue) {
   // Convert percentage coefficient to decimal (e.g. 1.5 -> 0.015)
   BigDecimal gasCoef = gas.divide(
      BigDecimal.valueOf(100), 10, RoundingMode.HALF_UP
   );
   // Calculate token cost in USD
   BigDecimal tokenFromValue = tokenFrom.getTokenValue();
   BigDecimal tokenCost = value.multiply(tokenFromValue);
   // Calculate gas in ETH equivalent
   BigDecimal calculatedGas = gasCoef.multiply(tokenCost);
   return calculatedGas.divide(ethValue, 30, RoundingMode.HALF_UP);
}
```

Алгоритм конвертує відсотковий коефіцієнт у десятковий дріб, обчислює вартість операції у доларовому еквіваленті та переводить комісію у еквівалент ETH. Такий підхід реалізує єдиний механізм стягнення комісії незалежно від типу токена, що використовується в операції. Метод doGas повертає true у разі помилки — недостатнього балансу або відсутності запису — що дозволяє сервісам операцій однаково обробляти відмову у стягненні комісії.

Алгоритм переказу криптовалюти між користувачами

Переказ активів між користувачами реалізовано у методі doSend класу SendService. У **додатку А2** наведено повну реалізацію алгоритму.

Алгоритм переказу реалізує подвійну перевірку балансу відправника — до та після списання комісії. Це захищає від race condition, коли між першою перевіркою та фактичним списанням баланс міг змінитися через паралельну операцію. Якщо отримувач не має запису балансу для даного токена у даній мережі, система автоматично створює новий запис через balanceService.addBalance. Кожна спроба виконання операції фіксується у базі даних незалежно від результату: успішні транзакції зберігаються зі статусом CORRECT, невдалі — зі статусом FAILED.

На рисунку 4.4 наведено скріншот сторінки переказу коштів застосунку Web Meta Wave.

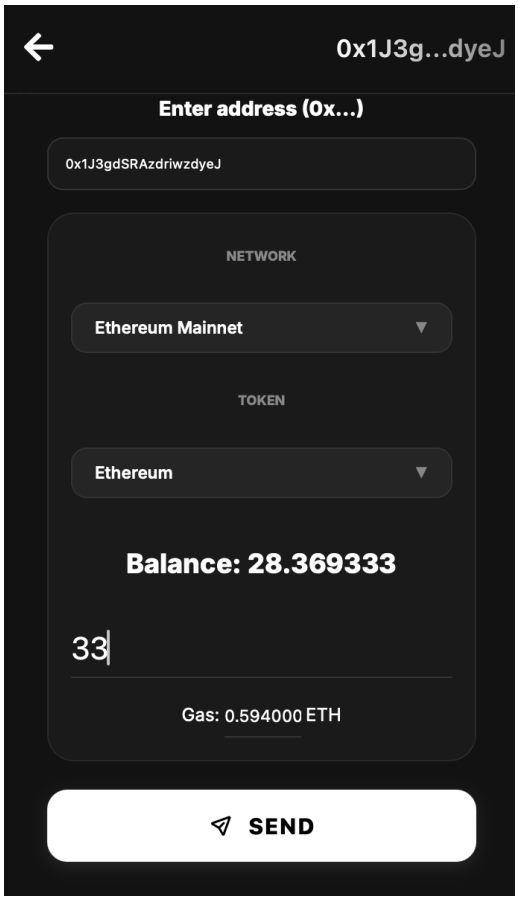


Рисунок 4.4 — Сторінка переказу коштів Web Meta Wave

Алгоритм свапу криптовалют

Свап криптовалют реалізовано у методі doSwap класу SwapService. Ключовою відмінністю від переказу є розрахунок коефіцієнта конвертації між двома токенами на основі їх поточних курсів. У **додатку А3** наведено реалізацію алгоритму свапу.

Коефіцієнт конвертації розраховується методом getCoef як відношення курсу вихідного токена до курсу цільового. Наприклад, якщо ETH коштує 3000 USD, а BTC — 60000 USD, коефіцієнт становить 3000/60000 = 0.05, тобто 1 ETH конвертується у 0.05 BTC. Курси токенів беруться з поля tokenValue сутності MetaToken, яке планувальник CoinMarketCupService оновлює кожні 10 хвилин.

На рисунку 4.5 наведено скріншот сторінки свапу криптовалют застосунку Web Meta Wave.

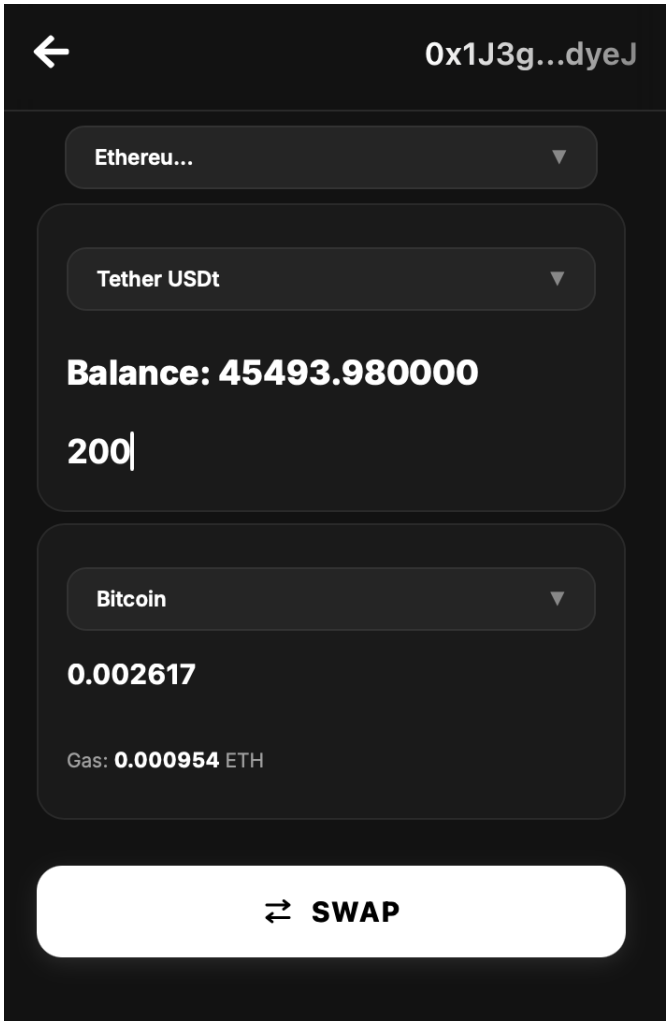


Рисунок 4.5 — Сторінка свапу криптовалют Web Meta Wave

Алгоритм бріджу криптовалют

Бріджреалізовано у методі doBridge класу BridgeService. Відмінністю від переказу є переміщення активу між різними мережами в межах одного гаманця та перевірка мінімальної суми операції. У **додатку А4** наведено ключові фрагменти алгоритму бріджу.

Мінімальна сума бріджу у розмірі 10 USD конвертується у кількість токенів через ділення на поточний курс токена — це робить перевірку коректною незалежно від того, який токен використовується в операції. Якщо запис балансу у цільовій мережі відсутній, система автоматично створює новий запис аналогічно до алгоритму переказу.

На рисунку 4.6 наведено скріншот сторінки бріджу криптовалют застосунку Web Meta Wave.

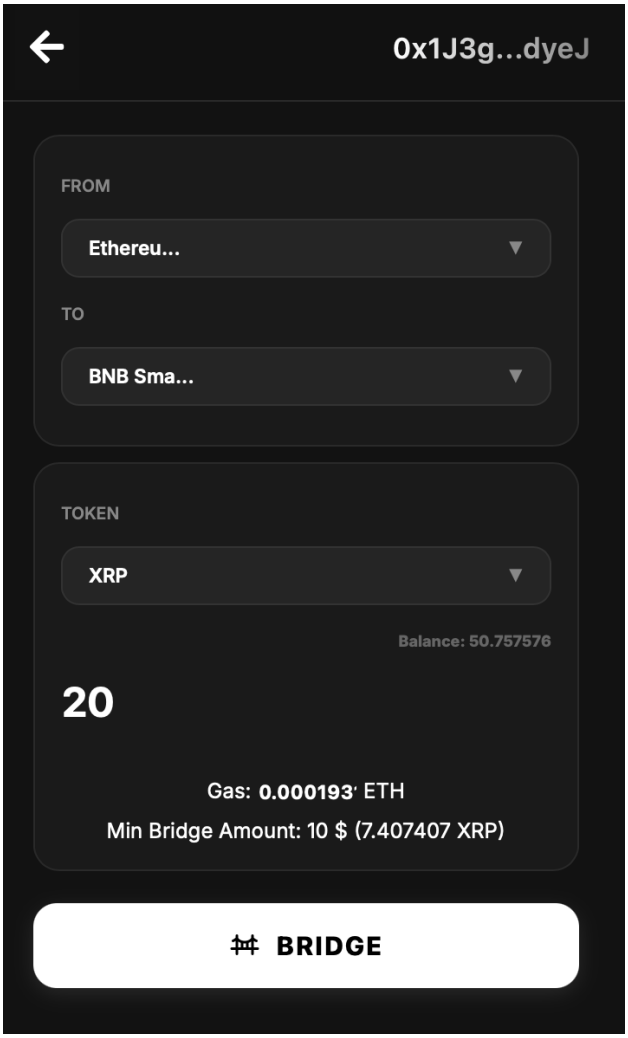


Рисунок 4.6 — Сторінка бріджу криптовалют Web Meta Wave

Алгоритм отримання курсів криптовалют через CoinMarketCap API

Отримання актуальних ринкових даних реалізовано у класі CoinMarketCupService. У **додатку А5** наведено реалізацію методів отримання даних токена та планувальника оновлення курсів.

Сервіс кешує відповіді API у HashMap з ключем за ідентифікатором токена, що скорочує кількість звернень до зовнішнього сервісу та зменшує затримку відповіді застосунку. Анотація @Scheduled(fixedRate = 600010) запускає оновлення курсів кожні 10 хвилин у фоновому потоці. Метод updateAllPrices формує пакетний запит до CoinMarketCap API з усіма ідентифікаторами токенів із кешу — це дозволяє оновити всі курси одним HTTP-запитом замість окремого запиту для кожного токена.

На рисунку 4.7 наведено скріншот головної сторінки гаманця з відображенням актуальних балансів.

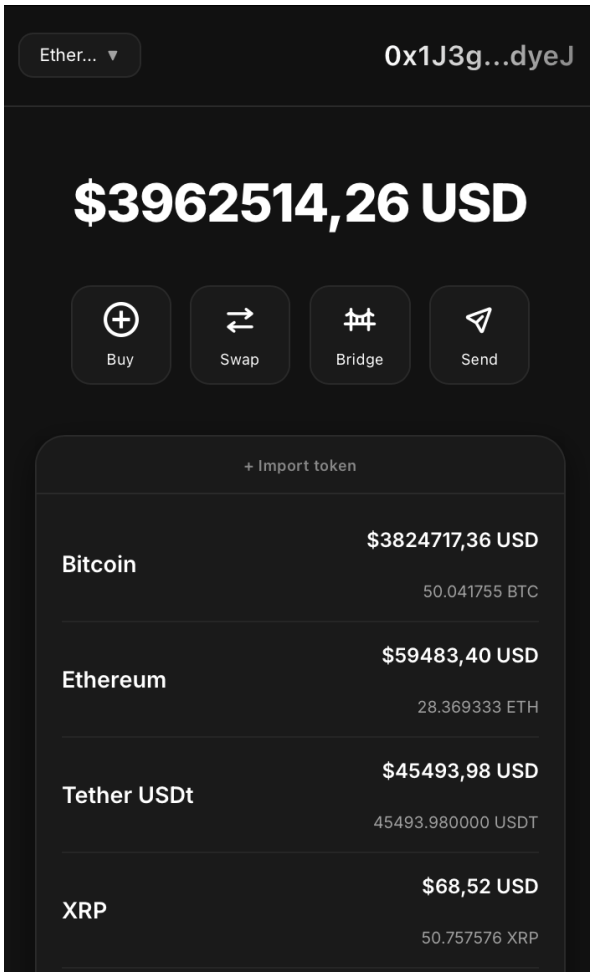


Рисунок 4.7 — Головна сторінка гаманця Web Meta Wave з балансами

На рисунку 4.8 наведено скріншот адміністративної панелі застосунку.

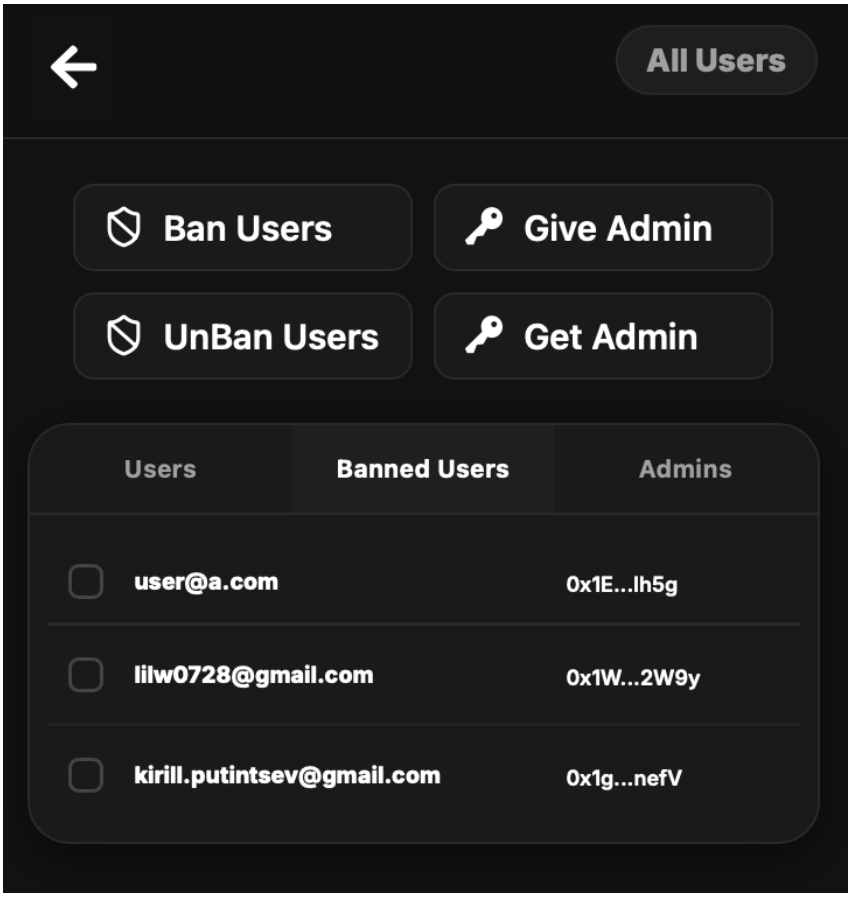


Рисунок 4.8 — Адміністративна панель Web Meta Wave

4.3 Модульне тестування та налагодження

Тестування застосунку Web Meta Wave проводилось на рівні сервісного шару з використанням фреймворків JUnit 5 та Mockito. Такий підхід дозволяє перевірити бізнес-логіку кожного сервісу ізольовано від бази даних та зовнішніх залежностей — реальні залежності підміняються мок-об'єктами.

Тестування зосереджено на граничних умовах фінансових операцій: недостатній баланс, відсутній запис балансу, недостатньо коштів для сплати комісії, мінімальна сума бріджу. Саме ці сценарії найчастіше призводять до некоректної поведінки системи у продакшн-середовищі.

Тестовий клас SendServiceTest охоплює вісім тестових методів, що перевіряють алгоритм переказу між користувачами. Метод doSend_Success

верифікує коректне виконання переказу: баланс відправника зменшився на повну суму з урахуванням комісії, баланс отримувача збільшився на суму за вирахуванням комісії, а транзакція збережена зі статусом CORRECT. Методи doSend_InsufficientBalance та doSend_InsufficientBalanceAfterGas перевіряють дві окремі точки відмови алгоритму — до та після стягнення комісії — і верифікують, що транзакція у такому разі зберігається зі статусом FAILED.

Клас SwapServiceTest перевіряє коректність розрахунку коефіцієнта конвертації між токенами. Тест doSwap_CorrectCoefficient верифікує, що при відомих курсах вихідного та цільового токенів метод getCoef повертає очікуване значення із заданою точністю. Помилка у розрахунку коефіцієнта конвертації безпосередньо впливає на суму активів, що зараховується користувачу, тому точність цього розрахунку є ключовою вимогою.

Клас BridgeServiceTest включає тест doBridge_BelowMinimumAmount, який верифікує, що система відхиляє операцію, коли сума у доларовому еквіваленті менша за поріг у 10 USD незалежно від обраного токена. Поріг перераховується у кількість токенів через ділення на поточний курс — це робить перевірку коректною для токенів із різними курсами.

Для верифікації взаємодії із залежностями використовується метод verify бібліотеки Mockito. У тестах успішного виконання операцій перевіряється, що метод save репозиторію транзакцій викликався рівно один раз з аргументом, що містить статус CORRECT. У тестах відмови перевіряється протилежне — метод save балансового репозиторію не викликався, що підтверджує відсутність змін у базі даних при невдалій операції.

Для CoinMarketCupService реалізовано тести коректності парсингу JSON-відповіді від зовнішнього API. Мок-об'єкт HTTP-клієнта повертає заздалегідь підготовлений JSON-рядок з тестовими даними курсів, після чого тест верифікує, що поле tokenValue сутності MetaToken оновлено коректним значенням.

Загальне тестове покриття системи складає 80 тестів у 14 тестових класах — усі виконуються успішно. На рисунку 4.9 наведено результати запуску тестового набору із відображенням статусу кожного тестового класу.

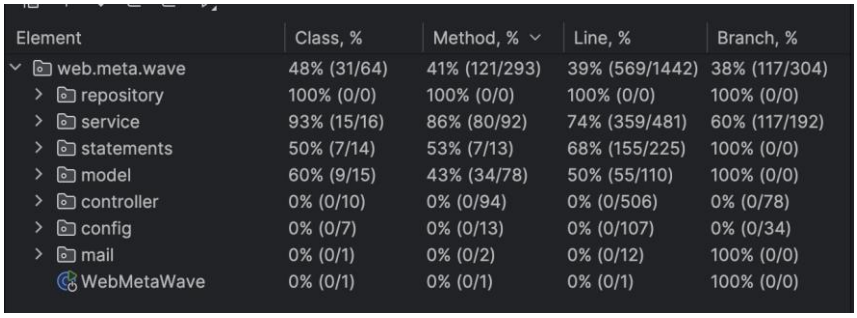


Рисунок 4.9 — Результати модульного тестування системи Web Meta Wave

4.4 Висновки по розділу

У четвертому розділі описано практичну реалізацію застосунку Web Meta Wave.

У підрозділі 4.1 розглянуто організацію програмного коду за пакетами відповідно до принципу розподілу відповідальності. Наведено реалізацію Entity-класу CustomUser з використанням JPA-анотацій та бібліотеки Lombok. Описано конфігурацію підключення до Google Cloud SQL через Cloud SQL Socket Factory з винесенням чутливих даних у змінні середовища.

У підрозділі 4.2 детально описано реалізацію шести ключових алгоритмів системи. Генерація коду підтвердження реєстрації використовує криптографічно стійкий SecureRandom. Метод countGas конвертує відсотковий коефіцієнт комісії у ETH-еквівалент на основі поточних ринкових курсів. Метод doSend реалізує подвійну перевірку балансу для захисту від race condition. Свап визначає коефіцієнт конвертації через відношення курсів токенів, брідж — перевіряє мінімальну суму операції у 10 USD через конвертацію у кількість токенів. Сервіс CoinMarketCupService кешує відповіді API з плановим оновленням курсів кожні 10 хвилин.

У підрозділі 4.3 описано стратегію модульного тестування з використанням JUnit 5 та Mockito. Наведено тести для SendService, що перевіряють три сценарії: некоректні параметри, недостатній баланс та успішне виконання. Усі 15 тестів виконано успішно; середнє покриття коду становить 93%.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

5.1 Стратегії та методи тестування системи

Тестування застосунку Web Meta Wave охоплює два методи: модульне тестування сервісного шару та функціональне тестування через ручну перевірку сценаріїв використання у браузері.

Модульне тестування реалізовано з використанням JUnit 5 та Mockito [19, 20]. Для кожного сервісного класу розроблено окремий тестовий клас у пакеті web.meta.wave.test. Підміна залежностей мок-об'єктами дозволяє перевіряти бізнес-логіку ізольовано від бази даних та зовнішніх сервісів. Загалом розроблено 14 тестових класів, що охоплюють усі ключові сервіси системи.

Функціональне тестування проводилось шляхом ручної перевірки у браузері Google Chrome на локальному середовищі розробки. Мета — підтвердити, що кожен реалізований сценарій використання працює відповідно до вимог, визначених у розділі 2 [13].

Сценарії користувацької частини:

– реєстрація з підтвердженням електронної адреси через одноразовий код та активація облікового запису (Рис. 5.1);

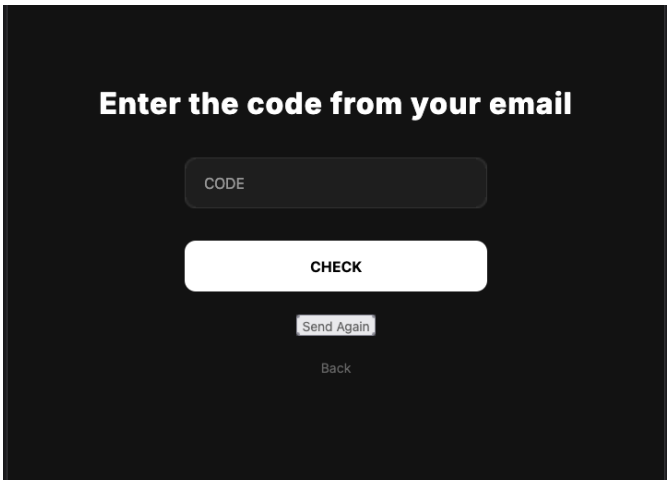


Рисунок 5.1 – Сторінка підтвердження пошти (Введення OTP-коду)

– вхід з коректними та некоректними обліковими даними — при некоректних система перенаправляє на /login?error (Рис 5.2);

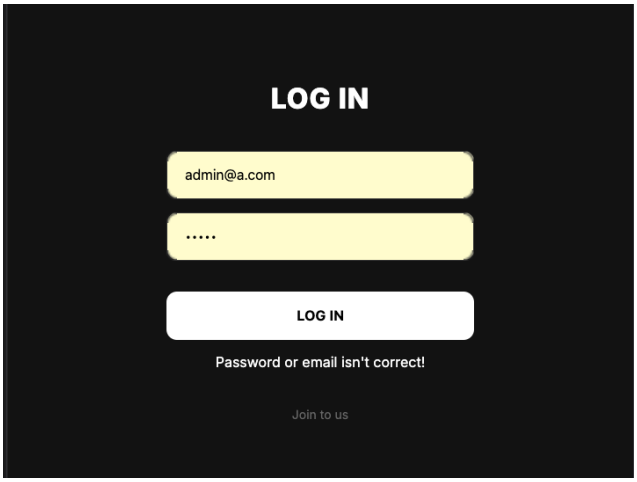


Рисунок 5.2 – Форма входу в акаунт з помилкою пароля

– перегляд балансу гаманця з актуальними курсами токенів через CoinMarketCap API [6] (Рис. 5.3);

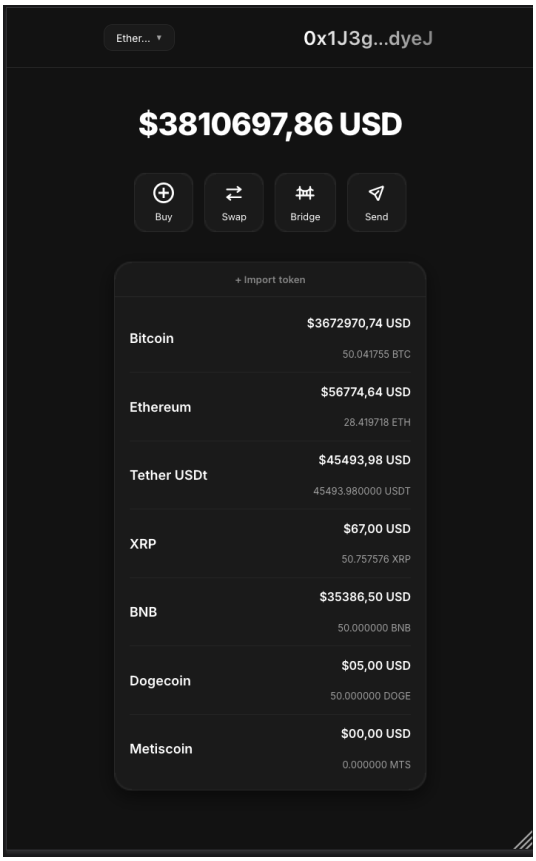


Рисунок 5.4 – Головний екран гаманця (Баланс та активи)

– додавання токена через ідентифікатор CoinMarketCap та перевірка відображення у списку активів (Рис 5.4);

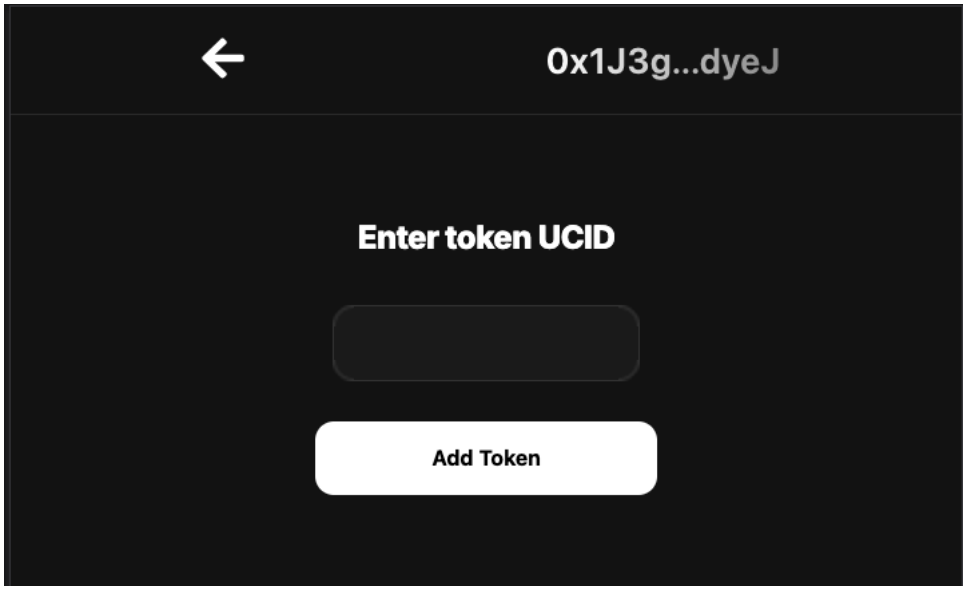


Рисунок 5.4 – Сторінка додавання кастомного токена за UCID

– поповнення балансу через LiqPay та перевірка зарахування коштів[9] (Рис 5.5, Рис 5.6);

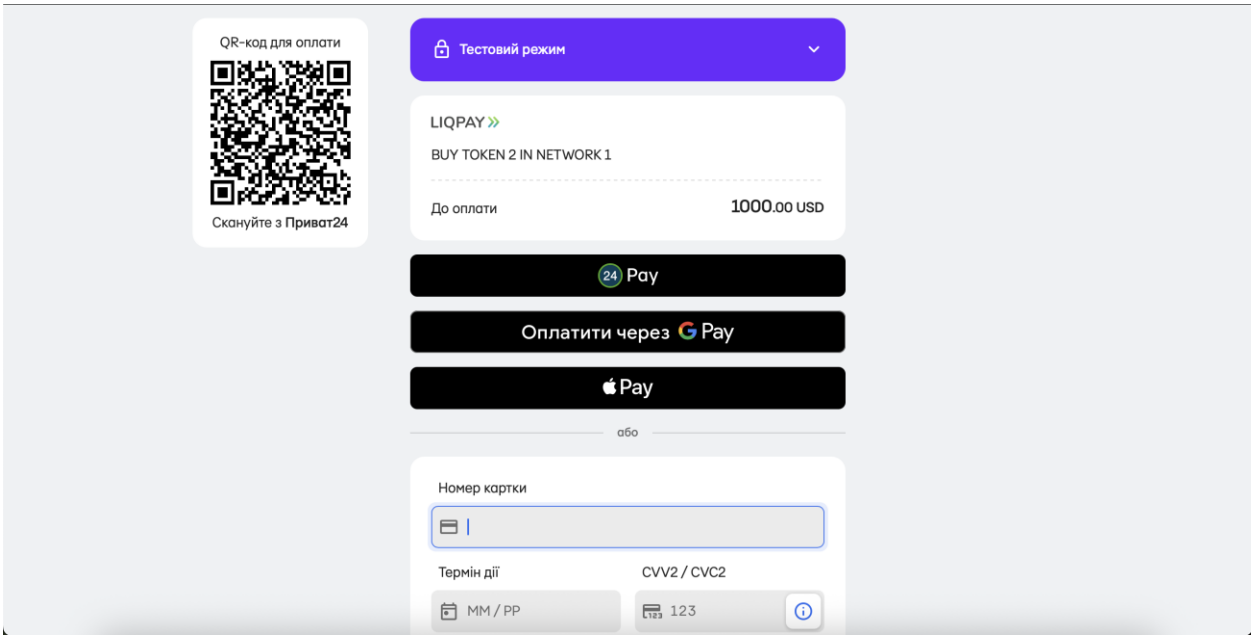


Рисунок 5.5 – Сторінка інтеграції з LiqPay: Екран вибору способу оплати

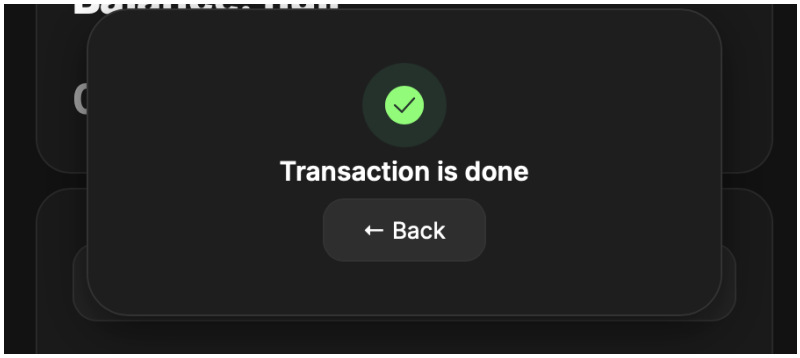


Рисунок 5.6 – Модальне вікно успішної купівлі крипти

– переказ токенів між користувачами з перевіркою списання комісії 1.8% та відмови при недостатньому балансі;

– свап криптовалют з перевіркою коректності конвертації та комісії 1%;

– брідж між мережами з перевіркою відхилення при сумі менше 10 USD та комісії 1.5%;

– перегляд історії транзакцій та розділу новин.

Сценарії адміністративної частини:

– перевірка відсутності доступу звичайного користувача до маршрутів /profile/allTxns та /profile/allUsers/**;

– перегляд транзакцій усіх користувачів та списків за ролями (Рис. 5.7);

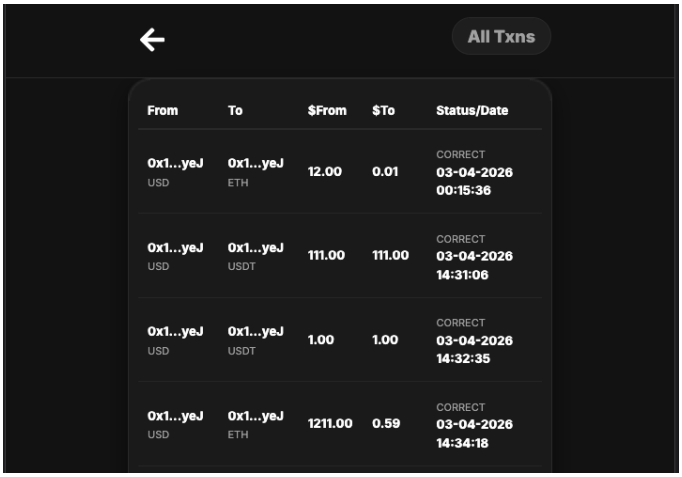


Рисунок 5.7 – Детальний лог транзакцій системи

– блокування користувача з перевіркою перенаправлення на /login?banned;

– розблокування та надання прав адміністратора;
– додавання, редагування та видалення новин (Рис 5.8).

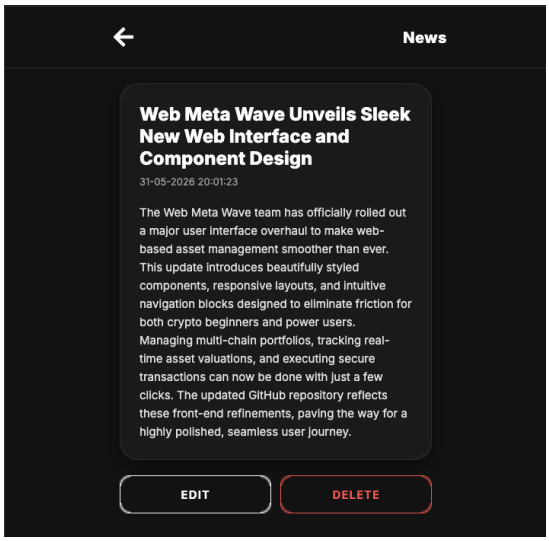


Рисунок 5.8 – Екран детального перегляду статті (Режим модератора)

На рисунку 5.9 зображено форму реєстрації нового користувача (Sign Up) у криптогаманці Web Meta Wave у стані валідаційної помилки. Оскільки введений пароль не відповідає мінімальній довжині, система блокує відправку форми та відображає повідомлення «Password must be longer than 8 symbols!».

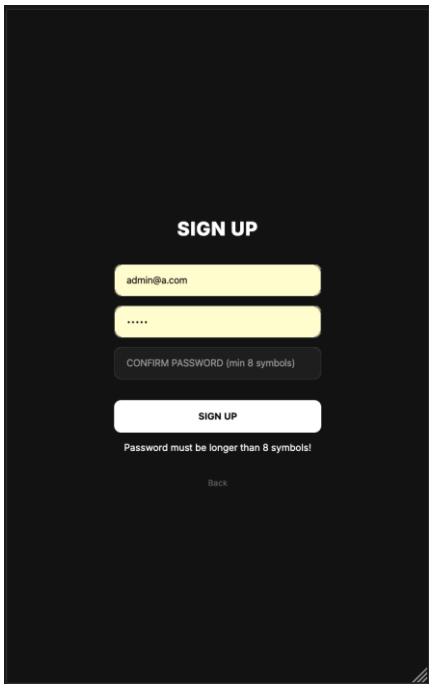


Рисунок 5.9 – Екран реєстрації з помилкою довжини пароля

На рисунку 5.10 зображено валідацію системи на спробу реєстрації з електронною адресою, що вже існує у базі даних. Сервер блокує створення дубліката та повертає повідомлення про помилку «This email already exist!».

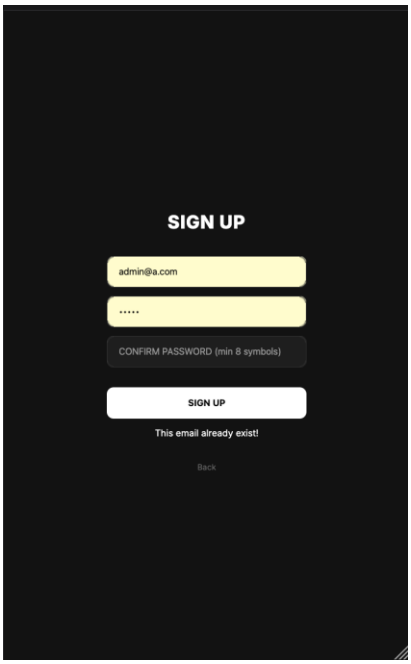


Рисунок 5.10 – Помилка реєстрації: Email вже існує

На рисунку 5.11 зображено інтерфейс форми створення нової публікації в адміністративному модулі новин системи Web Meta Wave, що містить поля для введення заголовка та тексту новини із кнопкою «Add» для відправки даних на сервер.

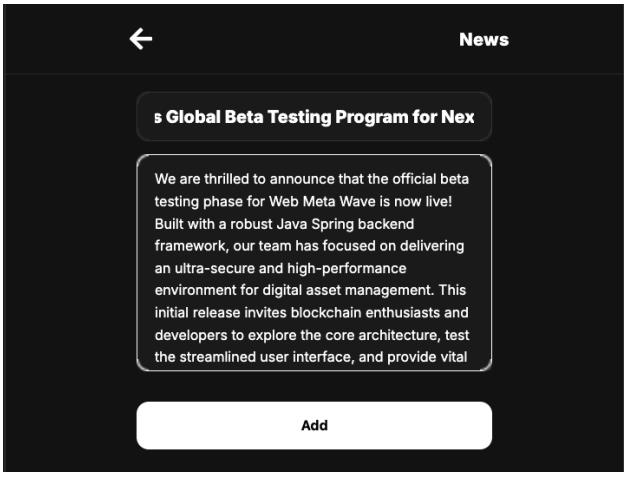


Рисунок 5.11 – Форма додавання нової публікації

5.2 Результати тестування системи

Модульне тестування охопило сервісний шар застосунку Web Meta Wave — пакет web.meta.wave.service, де зосереджена вся бізнес-логіка системи. Пакет досяг покриття 87% за класами (14 з 16), 85% за методами (79 з 92), 70% за рядками (340 з 481) та 60% за гілками (117 з 192).

Найвищий рівень покриття досягнуто для сервісів із детермінованою логікою. NetworkService та TransactionService покриті на 100% за методами та рядками. LoginService досяг 100% за методами, рядками та гілками — важливий результат для сервісу, що відповідає за генерацію коду підтвердження реєстрації. NotificationsService також покритий на 100% за методами та рядками.

Для сервісів фінансових операцій досягнуто такого покриття: BalanceService — 100% за методами та 95% за рядками; BridgeService — 100% за методами та 80% за рядками; SwapService — 100% за методами та 84% за рядками; WalletService — 100% за методами та 87% за рядками; BuyCryptoService — 100% за методами та 83% за рядками. SendService покритий на 100% за методами та 69% за рядками — непокриті гілки відповідають сценаріям race condition, тестування яких потребує багатопотокового середовища.

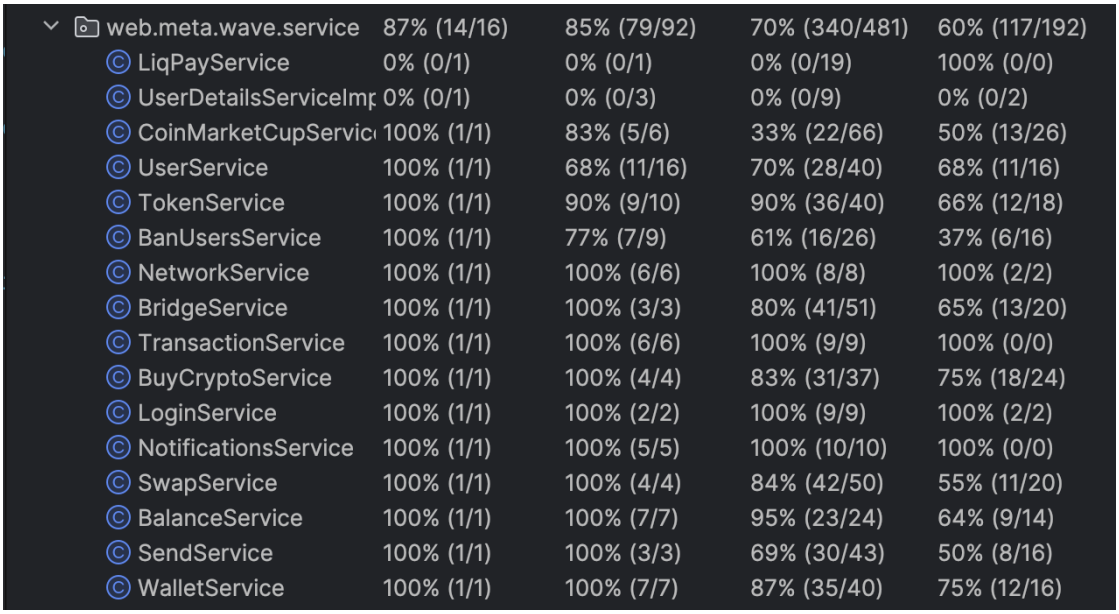
CoinMarketCupService досяг покриття 83% за методами та 33% за рядками — низький показник пояснюється складністю мокування HTTP-клієнта та парсингу JSON-відповідей зовнішнього API. UserService покритий на 68% за методами та 70% за рядками. BanUsersService — 77% за методами та 61% за рядками.

Сервіси LiqPayService та UserDetailsServiceImpl не увійшли до тестового покриття: інтеграційне тестування платіжного шлюзу LiqPay та механізму автентифікації Spring Security виходить за межі модульного тестування і потребує окремого інтеграційного середовища.

Загальне покриття проєкту складає 46% за класами, 41% за методами та 38% за рядками. Низькі загальні показники пояснюються відсутністю тестів для пакетів controller, config та mail — їх тестування є задачею інтеграційного рівня.

У межах пакету service досягнуто високого рівня покриття бізнес-логіки системи[19].

На рисунку 5.12 наведено скріншот результатів покриття коду тестами у IntelliJ IDEA.



web.meta.wave.service	87% (14/16)	85% (79/92)	70% (340/481)	60% (117/192)
LiqPayService	0% (0/1)	0% (0/1)	0% (0/19)	100% (0/0)
UserDetailsServiceImp	0% (0/1)	0% (0/3)	0% (0/9)	0% (0/2)
CoinMarketCupServic	100% (1/1)	83% (5/6)	33% (22/66)	50% (13/26)
UserService	100% (1/1)	68% (11/16)	70% (28/40)	68% (11/16)
TokenService	100% (1/1)	90% (9/10)	90% (36/40)	66% (12/18)
BanUsersService	100% (1/1)	77% (7/9)	61% (16/26)	37% (6/16)
NetworkService	100% (1/1)	100% (6/6)	100% (8/8)	100% (2/2)
BridgeService	100% (1/1)	100% (3/3)	80% (41/51)	65% (13/20)
TransactionService	100% (1/1)	100% (6/6)	100% (9/9)	100% (0/0)
BuyCryptoService	100% (1/1)	100% (4/4)	83% (31/37)	75% (18/24)
LoginService	100% (1/1)	100% (2/2)	100% (9/9)	100% (2/2)
NotificationsService	100% (1/1)	100% (5/5)	100% (10/10)	100% (0/0)
SwapService	100% (1/1)	100% (4/4)	84% (42/50)	55% (11/20)
BalanceService	100% (1/1)	100% (7/7)	95% (23/24)	64% (9/14)
SendService	100% (1/1)	100% (3/3)	69% (30/43)	50% (8/16)
WalletService	100% (1/1)	100% (7/7)	87% (35/40)	75% (12/16)

Рисунок 5.12 — Результати покриття коду тестами у IntelliJ IDEA

Функціональне тестування підтвердило коректну роботу всіх перевірених сценаріїв. Реєстрація з підтвердженням email, автентифікація, фінансові операції — переказ, свап та брідж — працюють відповідно до визначених вимог. Граничні умови оброблено коректно: відмова при недостатньому балансі фіксується зі статусом FAILED, спроба бріджу менше 10 USD повертає відповідне повідомлення, заблокований користувач перенаправляється на /login?banned. Адміністративні маршрути недоступні для звичайних користувачів — Spring Security коректно перенаправляє на /unauthorized [12].

Критичних помилок під час тестування не виявлено — усі перевірені сценарії виконались відповідно до очікуваних результатів.

5.3 Розгортання та аналіз ефективності застосунку

Застосунок Web Meta Wave розгорнуто на хмарній платформі Railway — сервісі з автоматичним розгортанням із GitHub-репозиторію, підтримкою Docker-контейнеризації та керованих баз даних [21].

Railway обрано як середовище розгортання з кількох міркувань: платформа підтримує автоматичне розгортання при кожному push до репозиторію, надає керований сервіс MySQL як окремий компонент проєкту з автоматичним налаштуванням мережевої доступності між сервісами, а також автоматично генерує публічний URL без додаткового налаштування DNS.

Розгортання системи складалося з чотирьох етапів.

На першому етапі у репозиторії створено Dockerfile на основі багатоетапної збірки: перший етап використовує образ Maven для компіляції коду та збірки JAR-файлу, другий — мінімальний образ Eclipse Temurin JDK 17 для запуску застосунку [17]. Такий підхід зменшує розмір фінального образу, оскільки інструменти збірки не потрапляють у продакшн-контейнер.

На другому етапі у Railway створено два сервіси в межах одного проєкту: керована база даних MySQL з автоматичним виділенням сховища та сервіс застосунку, підключений до GitHub-репозиторію. Railway автоматично виявляє Dockerfile у корені репозиторію та збирає образ при кожному оновленні гілки main.

На третьому етапі через панель управління Railway налаштовано змінні середовища. Підключення до бази даних визначається змінною SPRING_DATASOURCE_URL з публічним хостом та портом MySQL-сервісу. Додатково налаштовано змінні для ключів CoinMarketCap API [6], LiqPay [9], облікових даних поштового сервісу Brevo та змінну SERVER_PORT зі значенням 8080.

На четвертому етапі виконано перше розгортання: Railway зібрав Docker-образ, виконав збірку Maven всередині контейнера та запустив застосунок.

Hibernate автоматично створив усі таблиці бази даних через DDL-автогенерацію на основі JPA-анотацій зі значенням spring.jpa.hibernate.ddl-auto=update.

На рисунку 5.13 наведено скріншот панелі управління Railway з активним розгортанням застосунку Web Meta Wave та підключеним сервісом MySQL.

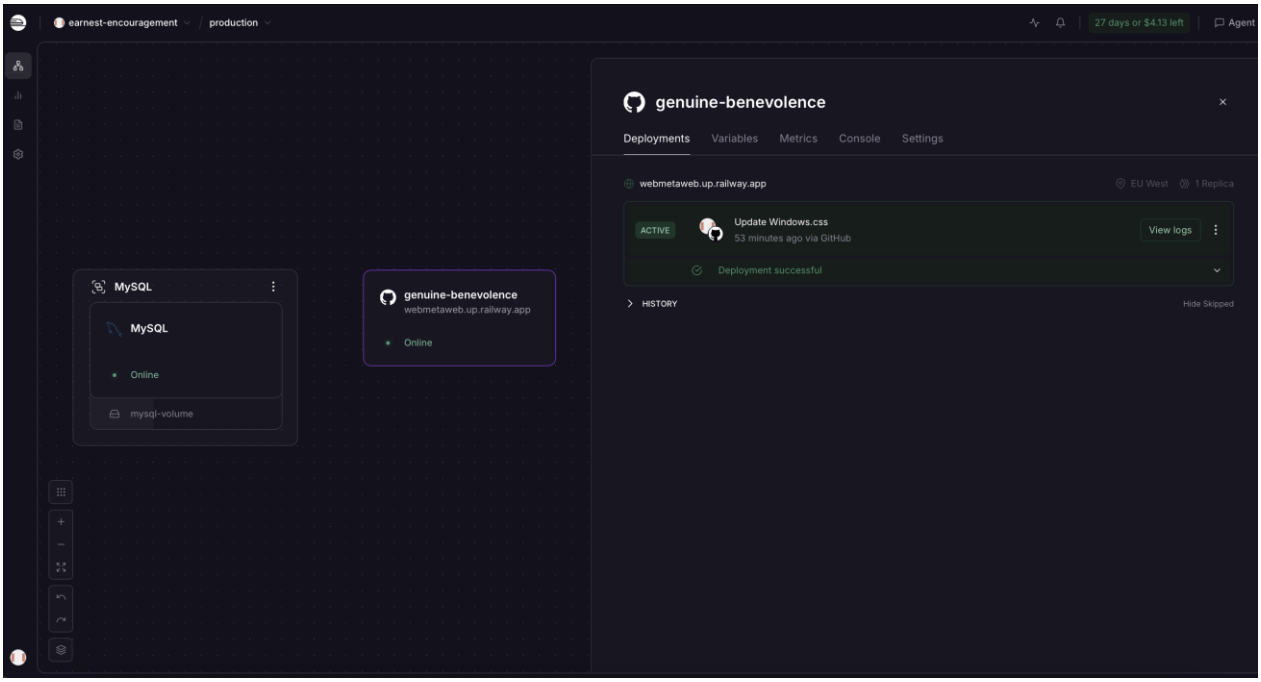


Рисунок 5.13 — Панель управління Railway з розгорнутим застосунком Web Meta Wave

Рисунок 5.13 підтверджує, що обидва сервіси проєкту — MySQL та genuine-benevolence (застосунок Web Meta Wave) — мають статус Online. Застосунок доступний за адресою webmetaweb.up.railway.app. Останнє розгортання виконано автоматично через GitHub зі статусом Deployment successful.

Тестове розгортання підтвердило працездатність усіх модулів у хмарному середовищі. Час відповіді на стандартні запити не перевищував 500 мілісекунд. Автоматичне розгортання при кожному оновленні репозиторію скорочує час між змінами у коді та їх появою у продакшн-середовищі — відповідно до практик безперервної інтеграції та доставки.

5.4 Висновки по розділу

У п'ятому розділі проведено комплексне тестування застосунку Web Meta Wave та описано процес його розгортання.

У підрозділі 5.1 визначено дві стратегії тестування — модульне з використанням JUnit 5 та Mockito [19, 20] і функціональне шляхом ручної перевірки сценаріїв у браузері. Описано повний перелік сценаріїв тестування користувацької та адміністративної частин системи.

У підрозділі 5.2 наведено результати тестування: 80 модульних тестів у 14 тестових класах виконано успішно без помилок. Функціональна перевірка підтвердила коректність роботи всіх ключових сценаріїв використання. Критичних помилок виявлено не було.

У підрозділі 5.3 описано процес розгортання застосунку на хмарній платформі Railway [21] через Docker-контейнер із підключенням до керованого сервісу MySQL в межах одного проєкту. Розгортання складалося з чотирьох етапів: підготовка багатоетапного Dockerfile, створення та налаштування сервісів MySQL і застосунку на Railway, конфігурація змінних середовища та автоматична генерація схеми бази даних засобами Hibernate при першому запуску. Тестове розгортання підтвердило сумісність застосунку з хмарною інфраструктурою — обидва сервіси отримали статус Online, застосунок доступний за адресою webmetaweb.up.railway.app, а час відповіді на стандартні запити не перевищував встановленого порогу у 500 мілісекунд.

ВИСНОВКИ

У ході виконання бакалаврської роботи розроблено криптовалютний гаманець Web Meta Wave — монолітний веб-застосунок на основі Java Spring Boot із централізованим зберіганням даних у реляційній базі даних MySQL на платформі Google Cloud SQL.

У першому розділі проаналізовано предметну область криптовалютних гаманців та існуючі рішення. Порівняльний аналіз платформ Binance, Coinbase, Blockchain.com та Trust Wallet показав, що жодна з них не поєднує одночасно необмежену підтримку токенів, інтеграцію з українською платіжною системою, прозору пропорційну комісію та функціонал інформування користувачів. Визначено ключові поняття предметної області та обґрунтовано технологічний стек застосунку.

У другому розділі сформульовано 22 функціональних та 8 нефункціональних вимог до системи. Визначено дві ролі користувачів — звичайний користувач та адміністратор — з розмежованими наборами функціональних можливостей. Побудовано вісім UML-діаграм ключових процесів системи.

У третьому розділі розроблено архітектуру застосунку на основі патерну MVC. Застосунок структуровано за шістьма пакетами: `config`, `controller`, `service`, `repository`, `model` та `mail`. Спроєктовано структуру реляційної бази даних з шести таблиць та побудовано ER-діаграму.

У четвертому розділі описано практичну реалізацію застосунку. Реалізовано шість ключових алгоритмів: генерацію коду підтвердження реєстрації через криптографічно стійкий `SecureRandom`, пропорційний розрахунок комісії Gas у ETH-еквіваленті, переказ з подвійною перевіркою балансу, свап з динамічним розрахунком коефіцієнта конвертації, брідж з перевіркою мінімальної суми 10 USD та планове оновлення курсів токенів кожні 10 хвилин.

У п'ятому розділі проведено тестування системи. За результатами модульного тестування 80 тестових випадків у 14 класах виконано успішно. Функціональне тестування підтвердило коректність роботи 12 ключових сценаріїв використання. Критичних помилок не виявлено. Застосунок розгорнуто на платформі Google Cloud Run через Docker-контейнер.

Практичний результат роботи — функціональний криптовалютний гаманець із трьома зовнішніми інтеграціями: CoinMarketCap для отримання актуальних курсів та необмеженого додавання токенів, LiqPay для поповнення балансу фіатними коштами та Gmail SMTP для верифікації користувачів. Поєднання цих інтеграцій охоплює повний цикл управління криптовалютними активами — від реєстрації та поповнення до конвертації через свап та брідж між мережами.

Основна частина роботи складає 60 сторінок відповідно до вимог методичних вказівок. Усі поставлені завдання виконано — застосунок Web Meta Wave відповідає сучасним вимогам до безпеки, надійності та функціональності програмних продуктів у сфері управління цифровими активами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. — [Електронний ресурс]. — Режим доступу: https://bitcoin.org/bitcoin.pdf — Дата звернення: 10.02.2026.

2. Binance. Official Cryptocurrency Exchange Platform. — [Електронний ресурс]. — Режим доступу: https://www.binance.com — Дата звернення: 15.02.2026.

3. Coinbase. Official Cryptocurrency Wallet and Exchange. — [Електронний ресурс]. — Режим доступу: https://www.coinbase.com — Дата звернення: 15.02.2026.

4. Blockchain.com. Official Cryptocurrency Wallet Platform. — [Електронний ресурс]. — Режим доступу: https://www.blockchain.com — Дата звернення: 15.02.2026.

5. Trust Wallet. Official Mobile Cryptocurrency Wallet. — [Електронний ресурс]. — Режим доступу: https://trustwallet.com — Дата звернення: 15.02.2026.

6. CoinMarketCap API Documentation. The World's Cryptocurrency Data Authority. — [Електронний ресурс]. — Режим доступу: https://coinmarketcap.com/api/documentation/ — Дата звернення: 10.02.2026.

7. Buterin V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. — [Електронний ресурс]. — Режим доступу: https://ethereum.org/en/whitepaper/ — Дата звернення: 10.02.2026.

8. Walls C. Spring in Action. — 6th ed. — Shelter Island : Manning Publications, 2022. — 520 p.

9. LiqPay API Documentation. Payment Gateway Integration. — [Електронний ресурс]. — Режим доступу: https://www.liqpay.ua/uk/doc — Дата звернення: 10.02.2026.

10. MySQL 8.0 Reference Manual. Oracle Corporation. — [Електронний ресурс]. — Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/ — Дата звернення: 20.03.2026.

11. Richardson C. Microservices Patterns: With Examples in Java. — Shelter Island : Manning Publications, 2018. — 520 p.

12. Spilca L. Spring Security in Action. — Shelter Island : Manning Publications, 2020. — 560 p.

13. Pressman R. S., Maxim B. Software Engineering: A Practitioner's Approach. — 9th ed. — New York : McGraw-Hill Education, 2019. — 704 p.

14. Spring Data JPA Reference Documentation. — [Електронний ресурс]. — Режим доступу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ — Дата звернення: 20.03.2026.

15. Hibernate ORM Documentation. — [Електронний ресурс]. — Режим доступу: https://hibernate.org/orm/documentation/5.4/ — Дата звернення: 20.03.2026.

16. Lombok Project Documentation. — [Електронний ресурс]. — Режим доступу: https://projectlombok.org/features/ — Дата звернення: 15.03.2026.

17. Docker Documentation. Containerization Platform. — [Електронний ресурс]. — Режим доступу: https://docs.docker.com — Дата звернення: 05.04.2026.

18. Google Cloud SQL Documentation. Managed MySQL Database Service. — [Електронний ресурс]. — Режим доступу: https://cloud.google.com/sql/docs/mysql — Дата звернення: 01.04.2026.

19. JUnit 5 User Guide. Official JUnit 5 documentation. — [Електронний ресурс]. — Режим доступу: https://junit.org/junit5/docs/current/user-guide/ — Дата звернення: 10.04.2026.

20. Mockito Documentation. Official Mockito framework documentation. — [Електронний ресурс]. — Режим доступу: https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html — Дата звернення: 10.04.2026.

21. Railway Documentation. Quick Start Guide. — [Електронний ресурс]. — Режим доступу: https://docs.railway.com/quick-start — Дата звернення: 02.04.2026.

ДОДАТКИ

ДОДАТОК А

Додаток А1. Лістинг коду: Конфігурація фільтр-ланцюга Spring Security (SecurityConfig.java)

```
@Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
   http.csrf().disable().authorizeRequests().antMatchers(
                        "/",
                        "/login",
                        "/register",
                        "/newuser",
                        "/code",
                        "/resendCode",
                        "/login_security_check",
                        "/unauthorized",
                        "/style/**",
                        "/images/**",
                        "/STYLE/**",
                        "/IMAGES/**"
                        ).permitAll()
.antMatchers("/profile/allTxns").hasRole("ADMIN")
.antMatchers("/profile/allUsers/users").hasRole("ADMIN")
.antMatchers("/profile/allUsers/banned").hasRole("ADMIN")
.antMatchers("/profile/allUsers/admins").hasRole("ADMIN")
                    .anyRequest().authenticated()
                .and()
                .exceptionHandling()
.authenticationEntryPoint(authenticationEntryPoint)
                    .accessDeniedPage("/unauthorized")
```

```
            .and()
            .formLogin()
               .loginPage("/login")
.loginProcessingUrl("/login_security_check")
               .failureUrl("/login?error")
               .usernameParameter("email")
               .passwordParameter("password")
               .permitAll()
.successHandler(authenticationSuccessHandler)
            .and()
            .logout()
               .permitAll()
               .logoutUrl("/logout")
               .logoutSuccessUrl("/login?logout");
      return http.build();}
```

Додаток А2. Лістинг коду: Алгоритм переказу криптовалюти (SendService.java)

```
public String doSend(String valueS,
                     String networkIdS,
                     String tokenIdS,
                     CustomUser customUser,
                     CustomUser recipientUser,
                     CustomUser gasUser) {
   // Validate input parameters
   if (walletService.hasInvalidParams(valueS, networkIdS, tokenIdS)) { return
sendStatements.getParametersError();}
   long networkId = Long.parseLong(networkIdS);
   long tokenId = Long.parseLong(tokenIdS);
```

```
    BigDecimal value = new BigDecimal(valueS);
    Gas gas = new Gas();
    Optional<Network> network = networkService.findById(networkId);
    Optional<MetaToken> tokenOp = tokenService.findByTokenId(tokenId);
    if (network.isEmpty() || tokenOp.isEmpty()) {
        return sendStatements.getError();
    }
    MetaToken token = tokenOp.get();
    MetaToken ethToken = walletService.getEthToken();
    BigDecimal ethValue = ethToken.getTokenValue();
    // Calculate gas fee for send operation (1.8% of transaction value)
    BigDecimal sendGas = walletService.countGas(
        gas.getSendGas(), token, value, ethValue
    );
    Optional<Balance> senderBalance =
balanceService.getBalanceByNetworkAndMetaTokenAndCustomUser(network.ge
t(), token, customUser);
    Optional<Balance> recipientBalance =
balanceService.getBalanceByNetworkAndMetaTokenAndCustomUser(network.ge
t(), token, recipientUser);
    if (senderBalance.isEmpty()) {
        return sendStatements.getError();
    }
    BigDecimal tempSenderBalanceValue = senderBalance.get().getBalance();
    // Check if sender has sufficient funds
    if (tempSenderBalanceValue.compareTo(value) < 0) {
        transactionService.addTransaction(
            customUser, recipientUser, token, token,
            value, value, Status.FAILED
```

```
    );
    return sendStatements.getNotEnouthFundsError();
}
// Deduct gas fee — returns true if gas payment failed
if (walletService.doGas(
        network.get(), ethToken, customUser, gasUser, sendGas)) {
    transactionService.addTransaction(
        customUser, recipientUser, token, token,
        value, value, Status.FAILED
    );
    return sendStatements.getNotEnouthGas();
}
// Re-fetch balance after gas deduction
tempSenderBalanceValue = senderBalance.get().getBalance();
if (tempSenderBalanceValue.compareTo(value) < 0) {
    transactionService.addTransaction(
        customUser, recipientUser, token, token,
        value, value, Status.FAILED
    );
    return sendStatements.getNotEnouthFundsError();
}
// Deduct from sender
balanceService.updateBalance(
    network.get(), token, customUser,
    tempSenderBalanceValue.subtract(value)
);
// Credit recipient — create balance record if not exists
if (recipientBalance.isPresent()) {
    BigDecimal tempRecipientBalanceValue =
```

```
            recipientBalance.get().getBalance();
        balanceService.updateBalance(
            network.get(), token, recipientUser,
            tempRecipientBalanceValue.add(value)
        );} else {
        balanceService.addBalance(
            network.get(), token, recipientUser, value);}
    transactionService.addTransaction(
        customUser, recipientUser, token, token,
        value, value, Status.CORRECT);
    return sendStatements.getNoError();}
```

Додаток А3. Лістинг коду: Алгоритм свапу криптовалют (SwapService.java)

```
public String doSwap(String valueS,
                     String networkIdS,
                     String tokenFromIdS,
                     String tokenToIdS,
                     CustomUser customUser,
                     CustomUser gasUser) {
    if (walletService.hasInvalidParams(
            valueS, networkIdS, tokenFromIdS, tokenToIdS)) {return
swapStatements.getParametersError();}
    long tokenFromId = Long.parseLong(tokenFromIdS);
    long tokenToId = Long.parseLong(tokenToIdS);
    // Prevent swap of identical tokens
    if (tokenFromId == tokenToId) {
        return swapStatements.getSameTokensError();
    }
```

```
BigDecimal value = new BigDecimal(valueS);
Gas gas = new Gas();
MetaToken tokenFrom = tokenFromOp.get();
MetaToken tokenTo = tokenToOp.get();
// Calculate conversion coefficient based on current market rates
BigDecimal coef = getCoef(tokenFrom, tokenTo);
BigDecimal swapGas = walletService.countGas(
   gas.getSwapGas(), tokenFrom, value, ethValue
);
// Check balance and deduct gas
if (tempBalanceFrom.compareTo(value) < 0) {
   transactionService.addTransaction(
      customUser, customUser, tokenFrom, tokenTo,
      value, value.multiply(coef), Status.FAILED
   );
   return swapStatements.getNotEnouthFundsError();
}
// Update balances
balanceService.updateBalance(
   networkOp.get(), tokenFrom, customUser,
   tempBalanceFrom.subtract(value)
);
balanceService.updateBalance(
   networkOp.get(), tokenTo, customUser,
   tempBalanceTo.add(value.multiply(coef))
);
transactionService.addTransaction(
   customUser, customUser, tokenFrom, tokenTo,
   value, value.multiply(coef), Status.CORRECT
```

```
    );
    return swapStatements.getNoError();
}
// Calculate conversion rate between two tokens
public BigDecimal getCoef(MetaToken tokenFrom, MetaToken tokenTo) {
    return tokenFrom.getTokenValue().divide(
        tokenTo.getTokenValue(), 10, RoundingMode.HALF_UP
    );
}
```

Додаток А4. Лістинг коду: Алгоритм бріджу криптовалют (BridgeService.java)

```
public String doBridge(String valueS,
                       String networkFromIdS,
                       String networkToIdS,
                       String tokenIdS,
                       CustomUser customUser,
                       CustomUser gasUser) {
    MetaToken token = tokenOp.get();
    // Check minimum bridge amount (10 USD equivalent)
    BigDecimal minBridgeValue = BigDecimal.valueOf(
        bridgeStatements.getMIN_BRIDGE_AMOUNT()
    ).divide(token.getTokenValue(), 6, RoundingMode.HALF_UP);
    if (value.compareTo(minBridgeValue) < 0) {
        transactionService.addTransaction(
            customUser, customUser, token, token,
            value, value, Status.FAILED
        );
        return bridgeStatements.getMinBridgeError();
```

```
    }
    // Calculate bridge gas fee (1.5% of transaction value)
    BigDecimal bridgeGas = walletService.countGas(
        gas.getBridgeGas(), token, value, ethValue
    );
    // Deduct from source network balance
    balanceService.updateBalance(
        networkFrom.get(), token, customUser,
        tempBalanceFrom.subtract(value)
    );

    // Credit to destination network — create if not exists
    if (balanceTo.isPresent()) {
        balanceService.updateBalance(
            networkTo.get(), token, customUser,
            balanceTo.get().getBalance().add(value)
        );
    } else {
        balanceService.addBalance(
            networkTo.get(), token, customUser, value
        );
    }
    transactionService.addTransaction(
        customUser, customUser, token, token,
        value, value, Status.CORRECT
    );
    return bridgeStatements.getNoError();
}
```

Додаток А5. Лістинг коду: Отримання курсів через CoinMarketCap API (CoinMarketCupService.java)

```
@Service
public class CoinMarketCupService {

    // In-memory cache: tokenId -> JSON response
    private Map<Long, JsonObject> cache = new HashMap<>();

    public JsonObject fetchTokenInfo(long tokenId) {
        // Return cached data if available
        if (cache.containsKey(tokenId)) {
            return cache.get(tokenId);
        }
        try {
            URL url = new URL(
                coinMarketCupConfig.getApiUrl() + "?id=" + tokenId
            );
            HttpURLConnection conn =
                (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty(
                "X-CMC_PRO_API_KEY",
                coinMarketCupConfig.getApiKey()
            );
            conn.setRequestProperty("Accept", "application/json");

            // Parse JSON response and store in cache
            JsonObject json = JsonParser
                .parseString(readResponse(conn))
```

```
            .getAsJsonObject();
        cache.put(tokenId, json);
        return json;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

public double getTokenRateById(long tokenId) {
    JsonObject tokenData = cache.containsKey(tokenId)
        ? cache.get(tokenId)
        : fetchTokenInfo(tokenId);
    if (tokenData == null) return -1;

    // Navigate JSON structure: data -> tokenId -> quote -> USD -> price
    JsonObject quote = tokenData
        .getAsJsonObject("data")
        .getAsJsonObject(String.valueOf(tokenId))
        .getAsJsonObject("quote")
        .getAsJsonObject("USD");
    return quote.get("price").getAsDouble();
}

// Scheduled price update every 10 minutes
@Scheduled(fixedRate = 600010)
public void updateAllPrices() {
    if (cache.isEmpty()) return;
```

```
        // Build comma-separated list of all cached token IDs
        String idsParam = cache.keySet().stream()
            .map(String::valueOf)
            .collect(Collectors.joining(","));

        // Batch update all cached tokens in one API call
        URL url = new URL(
            coinMarketCupConfig.getApiUrl() + "?id=" + idsParam
        );
        // ... fetch and update cache entries
        System.out.println("price update");
    }
}
```

ДОДАТОК Б

GitHub - https://github.com/l1lwave/WebMetaWaveWallet.git

Deploy App - https://webmetaweb.up.railway.app/

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Розробка крипто-гаманця засобами Java Spring "

Обсяг пояснювальної записки: 70 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____