Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система ведення обліку криптобіржевих фінансових операцій
(тема)

Виконав:
студент 4 курсу, групи ПЗПІп-22-2

Борбунюк О.О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Русакова Н.Є.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Борбунюку Олексію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система ведення обліку криптобіржевих фінансових операцій
Затверджена наказом по університету від 17.06. 2024р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 20.07.2024
3. Вихідні дані до роботи Розробити програмну система ведення обліку криптобіржевих фінансових операцій, мовами програмування Java та Kotlin, з використанням мікросервісної архітектури.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2024	*виконано*
2	Створення специфікації ПЗ	22.05.2024	*виконано*
3	Проектування ПЗ	24.05.2024	*виконано*
4	Розробка ПЗ	28.06.2024	*виконано*
5	Тестування ПЗ	30.06.2024	*виконано*
6	Оформлення пояснювальної записки	05.07.2024	*виконано*
7	Підготовка презентації та доповіді	10.07.2024	*виконано*
8	Попередній захист	12.07.2024	*виконано*
9	Нормоконтроль, рецензування	12.07.2024	*виконано*
10	Здача роботи у електронний архів	12.07.2024	*виконано*
11	Допуск до захисту у зав. кафедри	18.07.2024	*виконано*

Дата видачі завдання 06 травня 2024р.

Студент (ка)_____ Борбунюк О.О.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Русакова Н.Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 79 стор., 39 рис., 3 табл., 17 джерел.

КРИПТОБІРЖА, МІКРОСЕРВІСНА АРХІТЕКТУРА, ОБЛІК, ПРОГРАМНА СИСТЕМА, ФІНАНСОВІ ОПЕРАЦІЇ, BINANCE API, JAVA, KOTLIN, POSTGRESQL, RABBITMQ, REST API, SPRING BOOT

Об’єкт розробки – Програмна система ведення обліку криптобіржевих фінансових операцій.

Мета розробки – створення Програмної системи ведення обліку криптобіржевих фінансових операцій з мікросервісною архітектурою.

Метод рішення – середовище розробки IntelliJ IDEA Ultimate, мови програмування Java 21 та Kotlin, Фреймворк: Spring Boot 3.2.5, Компоненти Spring: Spring WEB, Spring Actuator, Spring AMPQ, Spring Data JPA, Spring Cloud Config, Система черг: RabbitMQ, База даних: PostgreSQL, ORM: Hibernate, Архітектура: Мікросервіси, API: REST, Інструмент збірки: Gradle, Коннектор: Binance Connector Java, Публічний API: Binance API.

У результаті розробки створена програмна система ведення обліку криптобіржевих фінансових операцій з мікросервісною архітектурою, яку розгорнуто в контейнерах і мережі Docker з доступом через HTTP проксі сервер NGINX.

CRYPTO EXCHANGE, MICROSERVICE ARCHITECTURE, ACCOUNTING, SOFTWARE SYSTEM, FINANCIAL OPERATIONS, BINANCE API, JAVA, KOTLIN, POSTGRESQL, RABBITMQ, REST API, SPRING BOOT

The object of development is a software system for maintaining records of crypto exchange financial transactions.

The purpose of the development is to create a software system for keeping records of crypto exchange financial transactions with a microservice architecture.

Solution method – IntelliJ IDEA Ultimate development environment, Java 21 and Kotlin programming languages, Framework: Spring Boot 3.2.5, Spring components: Spring WEB, Spring Actuator, Spring AMPQ, Spring Data JPA, Spring Cloud Config, Queuing system: RabbitMQ, Base Database: PostgreSQL, ORM: Hibernate, Architecture: Microservices, API: REST, Build Tool: Gradle, Connector: Binance Connector Java, Public API: Binance API.

As a result of the development, a software system for keeping records of crypto-exchange financial transactions with a microservice architecture was created, which was deployed in containers and the Docker network with access via the NGINX HTTP proxy server.

Я, Борбунюк Олексій Олександрович, студент гр. ПЗПІп-22-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система ведення обліку криптобіржевих фінансових операцій», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

Криптобіржа - онлайн-сервіс, який дозволяє клієнтам обмінювати криптовалюти на інші активи, наприклад, звичайні фіатні гроші або інші цифрові валюти.

ПЗ - сукупність програм системи оброблення інформації та програмних документів, необхідних для їх експлуатації.

Kotlin[1] - статично типізована мова програмування, що працює поверх JVM і розробляється компанією JetBrains.

Spring Boot[2] - фреймворк на основі Java з відкритим вихідним кодом, розроблений компанією Pivotal Software.

RabbitMQ[3] – платформа, що реалізує систему обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP (Advanced Message Queuing Protocol).

PostgreSQL[4] – об'єктно-реляційна система керування базами даних.

API – інтерфейс програмування застосунків (прикладний програмний інтерфейс (англ. application programming interface)).

Binance[5] – сервіс обміну криптовалют.

REST – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів (скор. англ. Representational State Transfer, «передача репрезентативного стану»).

Мікросервіси – шаблон програмування, за яким єдиний застосунок будується як сукупність невеличких сервісів, кожен з яких працює у своєму власному процесі та спілкується з рештою, використовуючи прості та швидкі протоколи передачі даних.

ВСТУП

Криптовалютні біржі стали невід'ємною частиною сучасної фінансової системи, пропонуючи інвесторам та трейдерам нові можливості для здійснення фінансових операцій. Вони виступають як платформи для купівлі, продажу та обміну криптовалют, забезпечуючи швидкі, безпечні та прозорі трансакції. В умовах стрімкого розвитку криптовалютного ринку, програмне забезпечення, що лежить в основі цих бірж, повинно відповідати високим вимогам до продуктивності, масштабованості та безпеки.

У цьому контексті особливу увагу привертає мікросервісна архітектура, яка дозволяє створювати гнучкі та масштабовані програмні рішення. Використання мікросервісів сприяє розподілу функцій між окремими сервісами, що спрощує їх розробку, тестування та підтримку. Крім того, інтеграція з публічними API таких платформ, як Binance, дозволяє ефективно зберігати та обробляти дані про фінансові операції, що є критично важливим для успішної роботи криптобіржі.

Метою роботи є розробка програмної системи, основою якої є мікросервісна архітектура, що використовує REST API та RabbitMQ для обміну та отримання інформації між мікросервісами та клієнтом. Для розгортання додатку на сервері було обрано хмарний дроплет DigitalOcean, систему контейнеризації Docker, HTTP проксі сервер NGINX.

Важливою складовою сучасної програмної інфраструктури є використання контейнеризації, зокрема Docker, яка дозволяє забезпечити ізольоване середовище для кожного мікросервісу. Це не лише сприяє стабільності та передбачуваності роботи системи, але й полегшує її розгортання та масштабування. Контейнеризація дозволяє швидко розгортати нові версії сервісів, виправляти помилки та оновлювати функціонал без впливу на інші компоненти системи.

Для розробки продукту використовувались мови програмування Kotlin та Java 21, фреймворк Spring Boot та його складники, система обміну повідомленнями RabbitMQ та база даних PostgreSQL.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Аналіз предметної галузі включає вивчення ринку криптовалют, його основних компонентів та технологій, що використовуються для обліку фінансових операцій на криптобіржах.

Криптовалюта – це децентралізована цифрова валюта, яка використовує криптографію для забезпечення безпеки. Вона може працювати незалежно від посередників, таких як банки та платіжні системи [6]. Прикладами криптовалют є Bitcoin, Ethereum, Binance Coin та інші.

“Блокчейн або децентралізований цифровий реєстр – це особливий вид бази даних, який підтримується численними комп'ютерами, розміщеними по всьому світу. Дані блокчейну організовані в блоки, які розташовані в хронологічному порядку і захищені криптографією” [7]. Кожна транзакція підтверджується вузлами мережі та додається до ланцюжка блоків.

Криптовалютна біржа — це платформа, де можна купувати, продавати та торгувати криптовалютою. Криптобіржа дає змогу легко обмінювати одну криптовалюту на іншу або державні валюти на криптовалюту. Отже, біржа — це посередник між покупцем та продавцем криптовалюти, який забезпечує безпечність таких транзакцій. [8]. Криптобіржі дозволяють користувачам торгувати криптовалютами за фіатні гроші або інші криптовалюти. Прикладами криптобірж є Binance, Coinbase, Kraken та інші.

Криптовалютні гаманці - це цифрові пристрої зберігання даних, які зберігають коди, необхідні для безпечного доступу до криптоактивів та обміну ними [9]. Гаманці дозволяють користувачам зберігати приватні ключі та здійснювати транзакції. Існують різні типи гаманців, включаючи апаратні, програмні та онлайн-гаманці.

Ринок криптовалют продовжує розвиватися і залучає все більше інвесторів. Обсяги торгів і загальна капіталізація ринку зростають, хоча волатильність залишається високою.

Основними сучасними трендами ринку криптовалют є, зокрема: висока волатильність, швидкі зміни вартості активів та регулярні технологічні інновації. В свою чергу популярність децентралізованих фінансів (DeFi) і невзаємозамінних токенів (NFT) зростає.

Ключовими гравцями на ринку криптовалют є великі криптобіржі, такі як Binance, Coinbase, Kraken та інші, які забезпечують високу ліквідність та широкий спектр послуг для користувачів.

В контексті безпеки захист даних користувачів і транзакцій є критичним аспектом. Необхідно забезпечити надійне шифрування даних, багаторівневу аутентифікацію та регулярні аудити безпеки.

Щодо масштабованості треба зазначити, що зростання кількості користувачів і обсягу транзакцій вимагає масштабованої архітектури, здатної ефективно обробляти великі обсяги даних.

Інтеграція з API потребує забезпечення надійної та швидкої інтеграції з API криптобіржі Binance для отримання актуальних даних про ринок та виконання торгових операцій.

Перспективами розвитку є розширення функціональних можливостей системи, включаючи підтримку нових криптовалют, інтеграцію з іншими біржами та вдосконалення аналітичних інструментів.

Таким чином було розглянуто основні аспекти ринку криптовалют та технологій, що використовуються для створення систем обліку фінансових операцій на криптобіржах.

1.2 Аналіз існуючих аналогів

Для обліку фінансових операцій на криптобіржах існує декілька популярних рішень.

CoinStats [10] — один із найпопулярніших трекерів крипто-портфоліо на ринку з понад мільйоном користувачів! CoinStats дозволяє відстежувати всі ваші активи — незалежно від того, чи зберігаються вони на централізованих біржах чи в кастодіальних гаманцях. Інтерфейс CoinStats наведено на рис.1.

Рисунок 1 – UI CoinStats (за даними [10])

Особливості системи:

– гаманець CoinStats: CoinStats пропонує криптовалютний гаманець, який дозволяє купувати та обмінювати криптовалюту та отримувати дохід від своїх активів;
– партнерство CoinLedger: CoinStats співпрацює з CoinLedger, щоб допомогти вам подавати податки на криптовалюту;
– безкоштовний план: CoinStats безкоштовно до 10 гаманців або 1000 транзакцій;
– ціна: платний план CoinStat починається від 9,99 доларів на місяць.

CoinLedger[11]: криптовалютна податкова платформа та трекер портфоліо, яким довіряють понад 400 000 інвесторів у всьому світі. Платформа дозволяє відстежувати ваші прибутки, збитки та доходи на всіх ваших гаманцях і біржах, а також створювати повний податковий звіт про крипто. Інтерфейс CoinLedger наведено на рис.2.

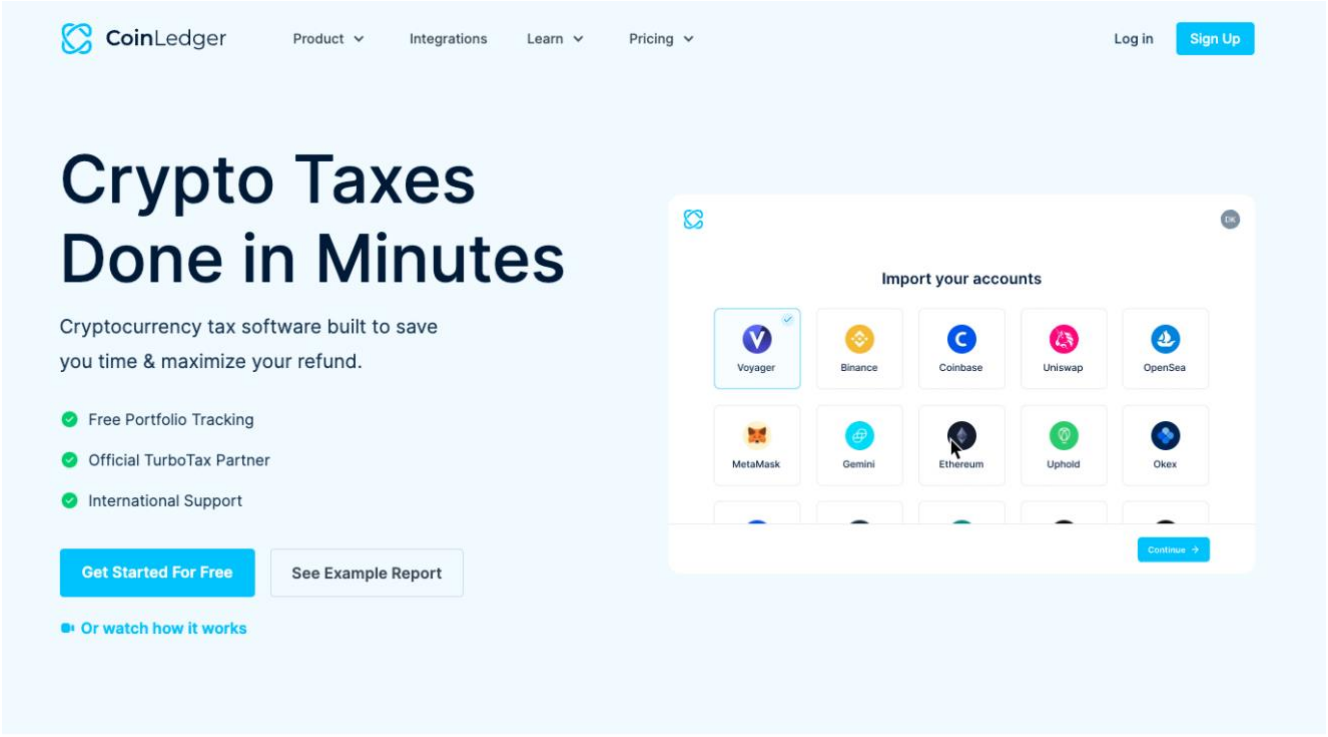


Рисунок 2 – UI CoinLedger (за даними [11])

Особливості системи:

– гаманець може автоматично імпортувати транзакції з бірж, таких як Coinbase, і блокчейнів, таких як Ethereum;
– CoinLedger інтегрується з сотнями блокчейнів і бірж;
– додаток може підкреслити найбільші можливості заощадження податків у вашому портфелі;
– визнано однією з найбільш зручних платформ на ринку. Платформа отримала оцінку 4,8 зірки на основі понад 700 відгуків;
– функції відстеження портфоліо CoinLedger абсолютно безкоштовні. Вам потрібно буде заплатити лише за завантаження податкового звіту CoinLedger.

Інтерфейс генерації форми податкового звіту IRS Form 8949 CoinLedger наведено на рис.3.

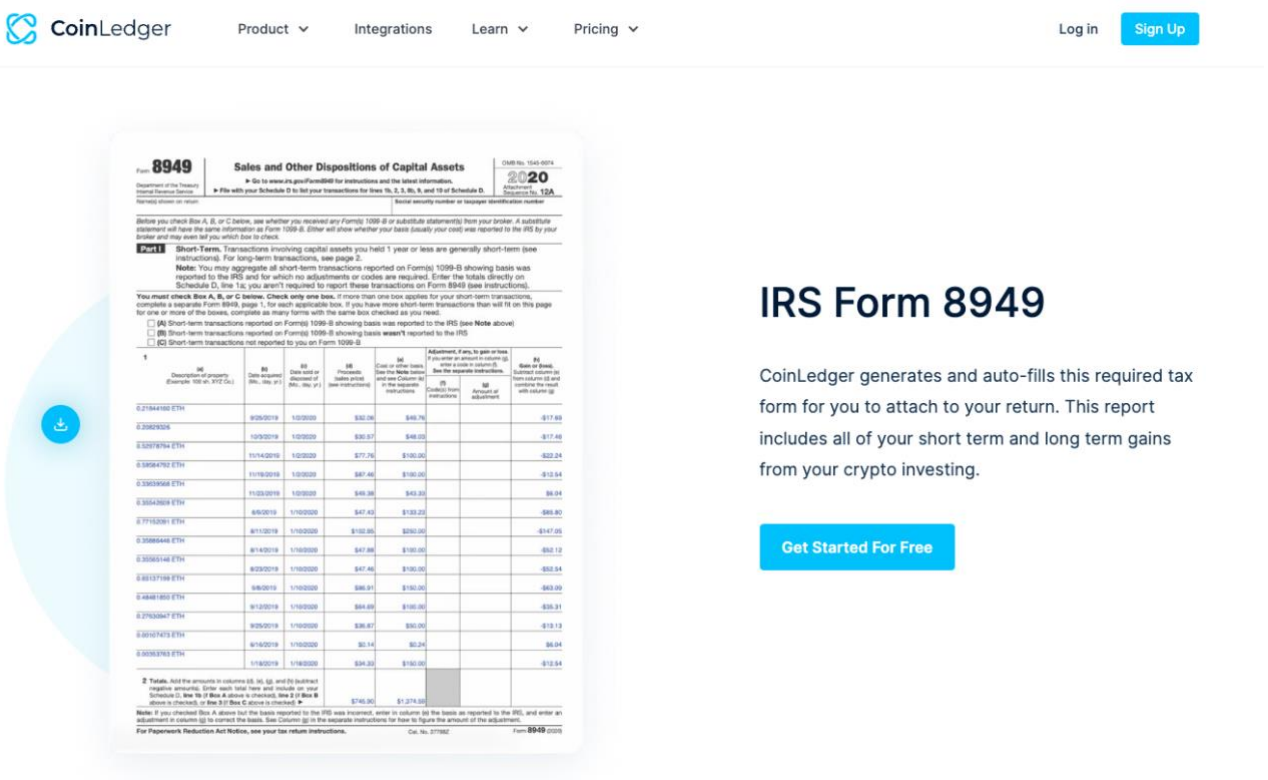


Рисунок 3 – UI форми податкового звіту IRS Form 8949 (за даними [11])

Delta[12]: заснована в 2017 році, є портфельним трекером із сотнями тисяч користувачів. Платформу часто хвалять за те, що вона дозволяє користувачам плану Delta Pro отримувати статистичні дані на основі минулих транзакцій (див.рис.4).

Особливості системи:

– має зручний інтерфейс для комп’ютера та мобільного пристрою;
– надає візуалізацію даних і інформацію про торгівлю для користувачів Delta Pro;
– дозволяє відстежувати інші інвестиції, такі як акції (хоча деякі користувачі сказали, що брокерська інтеграція Delta наразі відсутня);
– пропонує безкоштовний план. Згідно з веб-сайтом Delta, вартість платної версії платформи залежить від регіону та місця розташування.

Coingecko [13] є одним із найбільших у світі агрегаторів даних про криптовалюту. Наразі платформа відстежує ціни понад 10 000 криптовалют (див. рис.5).

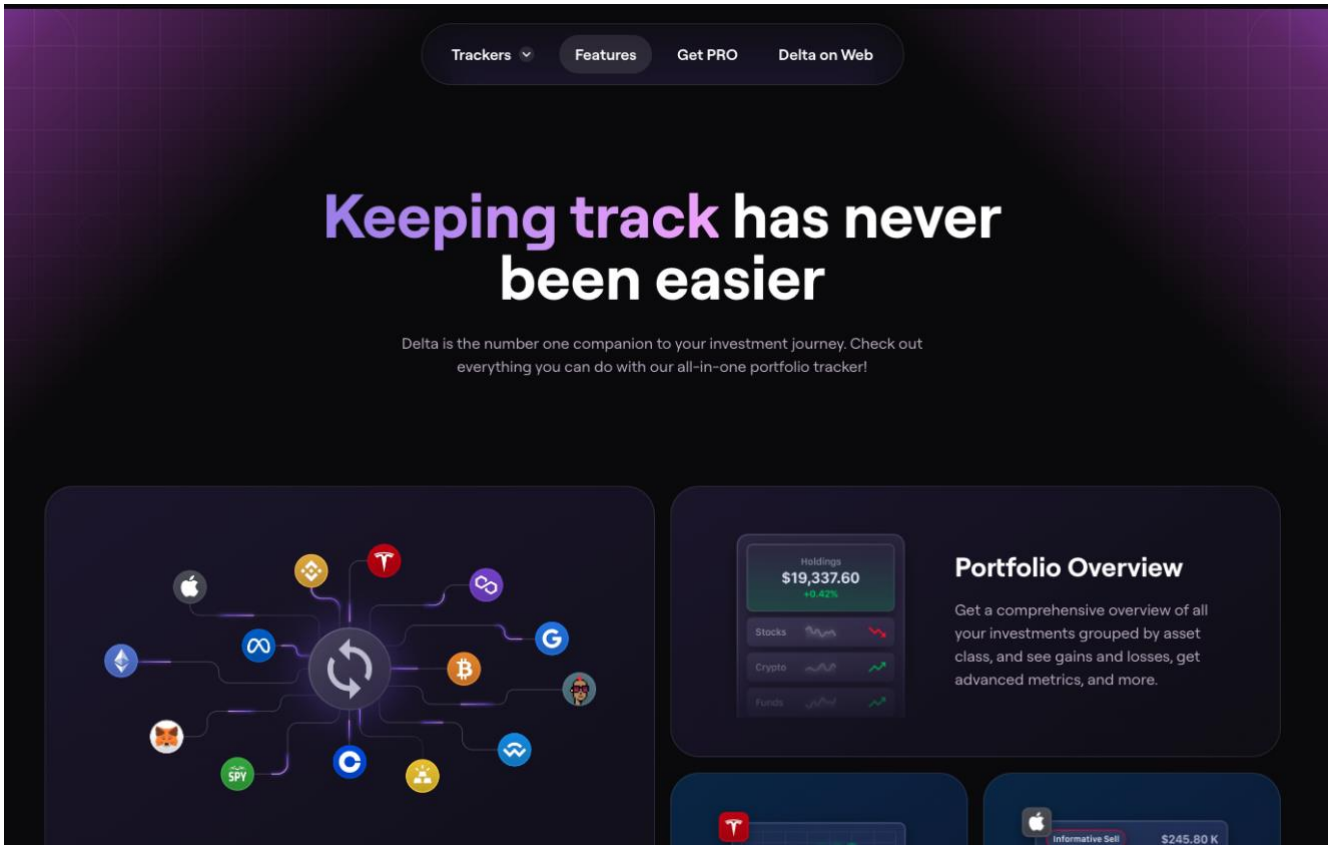


Рисунок 4 – UI Delta (за даними [12])

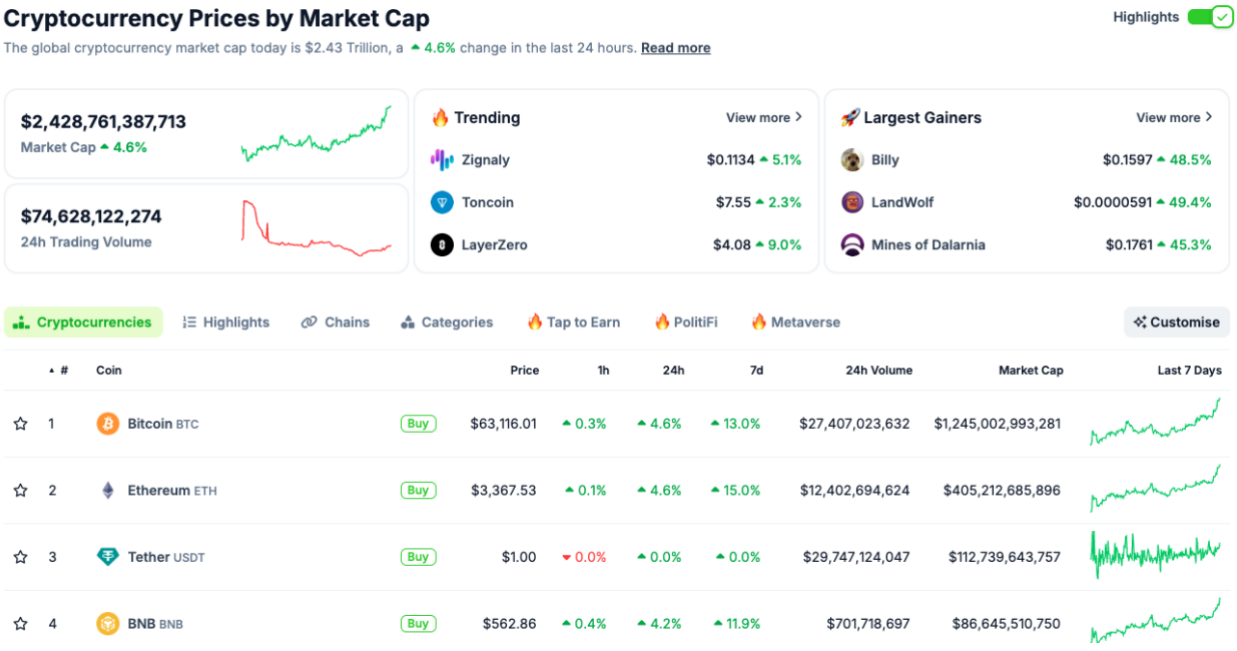


Рисунок 5 – UI Coingecko (за даними [13])

Особливості системи:

– дозволяє відстежувати свій портфель в Інтернеті та мобільному додатку Coingecko.
– хоча Coingecko не дозволяє автоматично імпортувати угоди та транзакції з гаманців і бірж, платформа дозволяє вручну додавати криптовалютні транзакції.
– дозволяє створювати власні списки спостереження для криптовалют, які цікавлять користувача.
– використання Coingecko абсолютно безкоштовно.

Kubera[14]: найкраще підходить для відстеження криптовалютних і некриптових активів. Kubera — це платформа відстеження статків, яка дозволяє відстежувати всі ваші активи — акції, криптовалюту, нерухомість або навіть ціну вашого веб-сайту (див. рис.6).

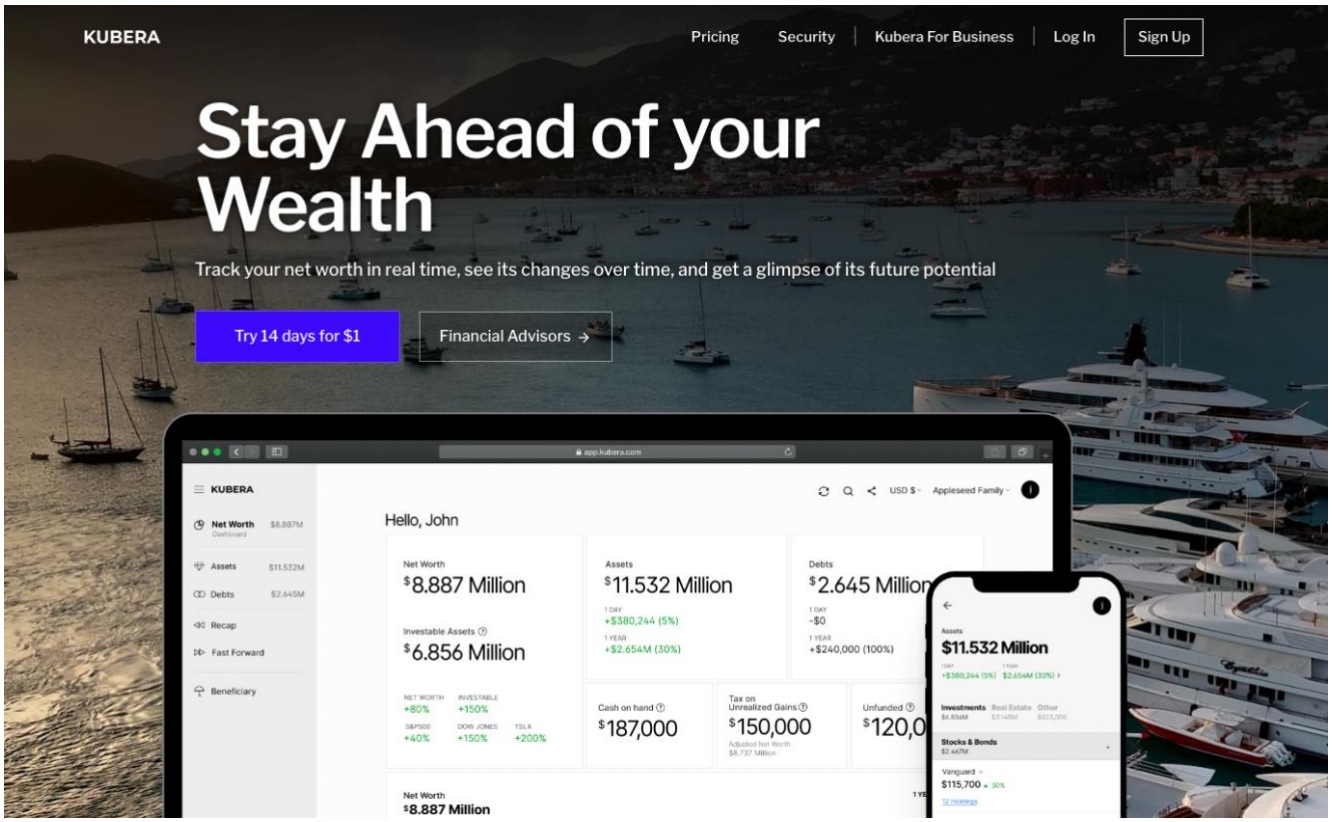


Рисунок 6 – UI Kubera (за даними [14])

Особливості системи:

– хоча Kubera може не мати інтеграції, яку має крипто-спеціальний трекер портфоліо, платформа пропонує можливість відстежувати крипто- та

некрипто-активи;

– дає можливість побачити, який із ваших активів у вашому портфоліо цінується найбільше;

– дозволяє легко ділитися своїм портфоліо з ким завгодно, наприклад з членами сім'ї чи менеджером капіталу;

– Kubera пропонує послугу передплати, яка коштує 150 доларів на рік.

Coinbase[15] вважається однією з найбільш надійних бірж у світі. Сьогодні компанія має понад 110 мільйонів користувачів і понад 80 мільярдів доларів загальних активів на своїй платформі (див. рис.7).

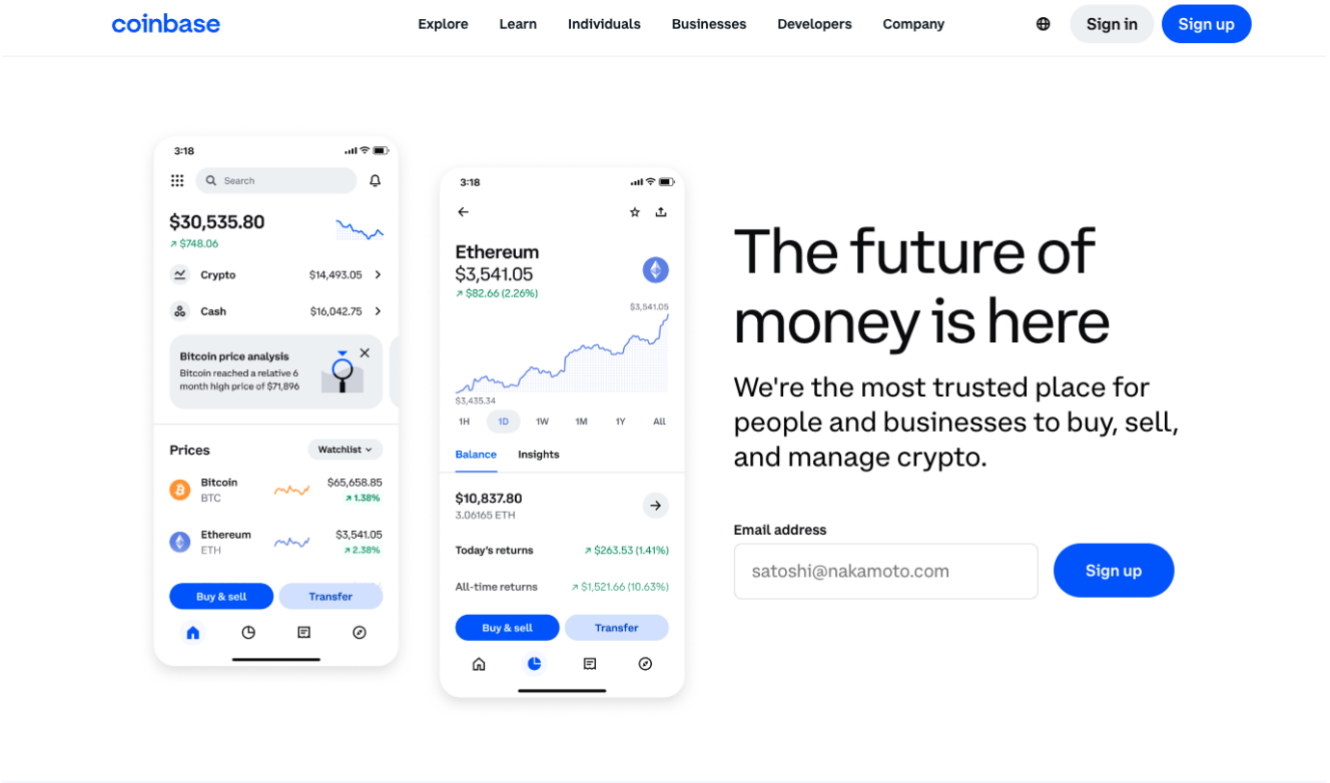


Рисунок 7 – UI Coinbase (за даними [15])

Особливості системи:

– Coinbase дозволяє переглядати короткі модулі про крипто-екосистему та отримувати винагороди;

– вважається однією з найбезпечніших бірж у світі — не зазнавала жодного серйозного зламу з моменту її заснування в 2012 році;

– дозволяє вам заробляти винагороди за ставки та отримувати автоматичні

відсотки на певні стейблкойни, як-от USDC;

– пропонує різні комісії залежно від розміру вашої транзакції. Комісія становить 0,99 дол. США за транзакції в розмірі 10 доларів США або менше, 1,49 дол. США за транзакції від 10 до 25 доларів США, 1,99 дол. США для транзакцій від 25 до 50 доларів США та 2,99 дол. США для транзакцій від 50 до 200 доларів США.

Відповідно до Coinbase, комісія за транзакції понад 200 доларів США «визначається комбінацією факторів, включаючи, але не обмежуючись, ваше місцезнаходження, вибраний спосіб оплати, розмір замовлення та ринкові умови, такі як волатильність і ліквідність.

Компанія Gemini [16] була заснована в 2014 році Кемероном і Тайлером Вінклвоссами. Наразі платформа дозволяє інвесторам купувати, продавати та торгувати криптовалютою у всіх 50 штатах (див. рис.8).

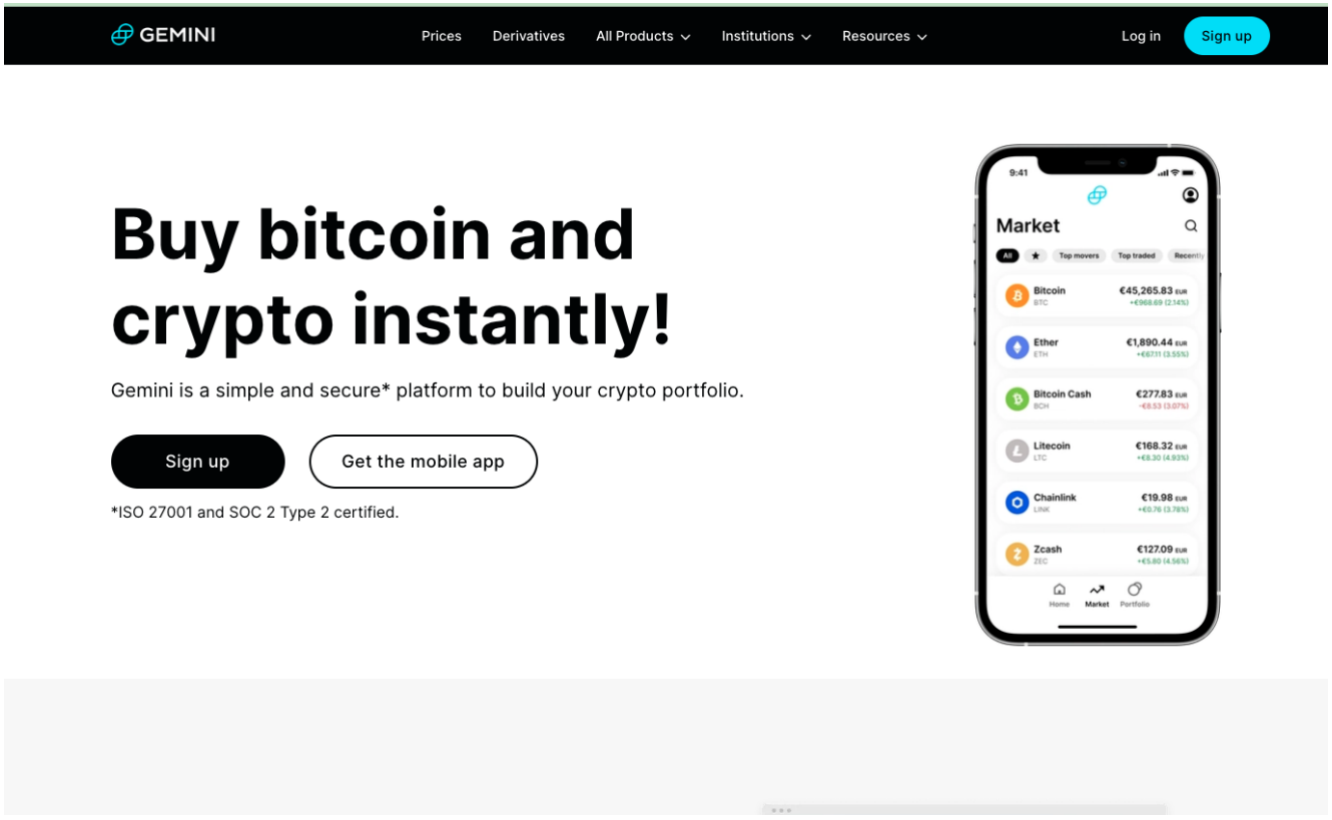


Рисунок 8 – UI Gemini (за даними [16])

Особливості системи:

– криптокредитна картка Gemini пропонує кешбек винагороди до 3%;

– через партнерство з Flexa Gemini дозволяє витрачати свою криптовалюту на щоденні покупки, як-от бензин і продукти;
– Gemini пишається довірою, безпекою та дотриманням нормативних вимог. Оскільки платформа має нью-йоркську ліцензію трасту, компанія підлягає щорічному аудиту фінансової звітності;
– ціна: 1,49% комісії за транзакції понад 200 доларів США, 3,49% комісії за покупки дебетовою та кредитною картками.

Kraken[17]: Найкраще за низьку плату. Заснована в 2011 році, Kraken є однією з найстаріших криптовалютних бірж, яка все ще працює (див. рис.9). Kraken вважається чудовим вибором для досвідчених інвесторів, які шукають низькі комісії.

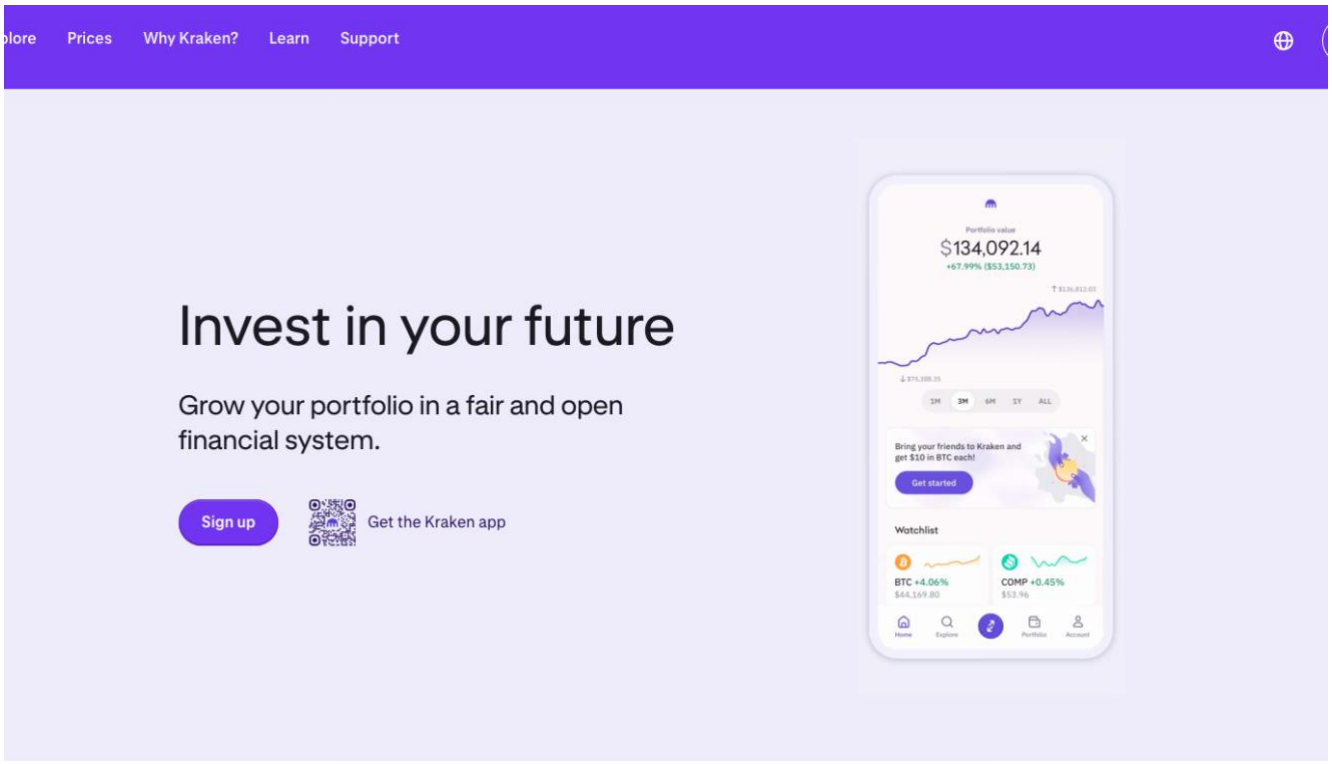


Рисунок 9 – UI Kraken (за даними [17])

Особливості системи:

– пропонує підтримку для маржинальних і ф’ючерсних торгів;
– Kraken співпрацює із зовнішнім аудитором, щоб переконатися, що платформа має підтвердження резервів;
– комісія Kraken Pro коливається від 0 до 0,26% залежно від розміру

транзакції.

Підведемо підсумки. Аналіз існуючих систем обліку фінансових операцій на криптобіржах вказує на наступні кращі практики та недоліки, які варто врахувати при розробці власної системи:

Кращі практики:

– автоматизація відстеження транзакцій та інтеграція з максимальною кількістю криптобірж і гаманців;
– зручний та інтуїтивно зрозумілий інтерфейс користувача;
– підтримка обчислення податків з урахуванням місцевого законодавства;
– реалізація функцій сповіщень про зміну цін у реальному часі.

Основні недоліки:

– висока вартість підписки на преміум-функції;
– обмежена підтримка менш популярних криптовалют та бірж;
– проблеми з точністю та своєчасністю оновлення даних.

Враховуючи ці аспекти, можна розробити систему, яка буде конкурентоспроможною, зручною та надійною для користувачів.

1.3 Виявлення проблем

Враховуючи специфіку роботи криптовалютних систем варто виділити наступні проблеми:

– недостатня масштабованість;
– проблеми з безпекою;
– складність інтеграції з різними криптобіржами;
– проблеми користувачів.

Багато існуючих систем обліку фінансових операцій на криптобіржах мають проблеми з масштабованістю. Коли кількість користувачів та обсяги транзакцій зростають, продуктивність системи може суттєво знизитися, що призводить до затримок в обробці даних та зниження якості обслуговування.

Зниження продуктивності системи при збільшенні кількості користувачів може призвести до затримок в обробці транзакцій, що негативно впливає на досвід

користувачів. Необхідність в ручному масштабуванні інфраструктури може збільшити витрати на технічне обслуговування.

Безпека є критично важливим аспектом для систем, що працюють з фінансовими даними. Поточні рішення можуть мати вразливості, які піддають користувачів ризику втрати активів або конфіденційної інформації через хакерські атаки або зловмисні дії.

Вплив на ефективність роботи: Вразливості в системі можуть призвести до втрати даних користувачів або їх фінансових активів, що може завдати серйозної шкоди репутації компанії. Забезпечення високого рівня безпеки вимагає постійного моніторингу та оновлення системи, що збільшує операційні витрати.

Багато систем стикаються з проблемами інтеграції з різними криптобіржами через відсутність стандартизації API, розбіжності в форматах даних та часті зміни в інтерфейсах бірж. Це ускладнює процес автоматичного відстеження транзакцій та вимагає значних зусиль для підтримки актуальності інтеграцій.

Вплив на ефективність роботи: Проблеми з інтеграцією можуть призвести до неповного або неточного відображення даних про транзакції, що ускладнює ведення обліку та прийняття рішень. Часті зміни в API криптобірж вимагають постійних ресурсів для підтримки актуальності інтеграцій, що збільшує витрати на розробку.

Відгуки та потреби користувачів - аналіз відгуків користувачів виявляє декілька критичних аспектів, які потребують покращення:

Швидкість обробки даних - користувачі очікують миттєвого відображення своїх транзакцій та оновлень портфелів.

Надійність та безпека - користувачі висловлюють занепокоєння щодо безпеки своїх даних та активів, вимагаючи надійних рішень для захисту від хакерських атак.

Простота інтеграції - користувачі хочуть мати можливість легко інтегрувати свої облікові записи з різних криптобірж та гаманців без необхідності складних налаштувань.

Інтерфейс користувача - зручний та інтуїтивно зрозумілий інтерфейс є важливим фактором для залучення та утримання користувачів.

Враховуючи ці аспекти, розробка нової системи повинна включати рішення для масштабування, забезпечення високого рівня безпеки та спрощення процесу інтеграції з різними криптобіржами.

1.4 Постановка задачі

Мета роботи є розробка системи для обліку фінансових операцій на криптобіржі.

Завданнями цієї роботи є: створення системи для обліку фінансових операцій на криптобіржі, яка буде забезпечувати високий рівень безпеки взаємодії з фінансовими даними користувачів, інтеграція з API Binance для отримання інформації про торгові операції та стан рахунків користувачів. Крім того, важливим аспектом розробки масштабованої архітектури системи для забезпечення ефективності та зручності розширення функціональності є дотримання патерну мікросервісної архітектури, а також впровадження механізмів аналізу даних для забезпечення прийняття обґрунтованих інвестиційних рішень та реалізація моніторингу та журналювання фінансових операцій для забезпечення атомарності та консистентності дій користувачів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

В контексті формування вимог до програмної системи необхідно виділити наступні функціональні вимоги:

- функціонал обліку фінансових транзакцій;
- можливість генерація звітів ;
- інтеграція з API Binance;
- окремий модуль аналітики;
- широка взаємодія з користувачами.

Зупинимось детально на кожній функціональній вимозі.

Система повинна забезпечувати детальний облік всіх фінансових транзакцій, що здійснюються на криптобіржі. Це включає такі основні операції, як покупка, продаж та обмін криптовалют. Кожна транзакція повинна бути збережена з усіма необхідними атрибутами, такими як дата та час виконання, тип операції, кількість та тип криптовалюти, ціна, комісії та ідентифікатор користувача, який здійснив операцію. Ця інформація має бути легко доступною для користувачів системи та повинна забезпечувати можливість проведення детального аналізу та аудиту транзакцій.

Система повинна мати функціонал для генерації звітів про фінансові операції та стан рахунків користувачів. Звіти повинні містити інформацію про всі здійснені транзакції, поточний баланс на рахунках, зміни в балансах за певний період, а також інші фінансові показники, які можуть бути корисними для користувачів. Звіти мають бути налаштовані під різні потреби користувачів: від детальних звітів для індивідуальних інвесторів до агрегованих звітів для корпоративних клієнтів та адміністраторів системи. Крім того, система повинна підтримувати можливість експорту звітів у різні формати, такі як PDF, Excel або CSV, для подальшого аналізу або зберігання.

Інтеграція з API Binance є ключовою вимогою для забезпечення актуальної інформації про ринкові дані та виконані торгові операції. Система повинна мати можливість автоматичного отримання даних про поточні ринкові ціни, історичні

дані про торги, а також інформацію про стан рахунків користувачів на платформі Binance. Це включає інформацію про баланси, відкриті ордери, виконані угоди та інші фінансові дані. Інтеграція повинна бути реалізована таким чином, щоб забезпечувати максимальну швидкість та надійність отримання даних, а також можливість безперервного оновлення інформації у режимі реального часу.

Система повинна включати модуль аналітики, який дозволить користувачам проводити детальний аналіз своїх фінансових операцій. Цей модуль повинен надавати різні інструменти для візуалізації даних, такі як графіки, діаграми та інші візуальні елементи, що допомагають користувачам краще зрозуміти свої фінансові показники. Аналітичний модуль також повинен підтримувати можливість створення користувацьких звітів та проведення складних запитів для отримання необхідної інформації.

Інтерфейс системи повинен бути інтуїтивно зрозумілим та зручним у використанні. Користувачі повинні мати можливість легко знаходити необхідну інформацію, здійснювати фінансові операції та генерувати звіти. Для цього інтерфейс повинен бути розроблений з урахуванням принципів ергономіки та юзабіліті, а також підтримувати багатомовність для зручності користувачів з різних регіонів.

Щодо нефункціональних вимог, варто виділити наступні вимоги:

- продуктивність системи;
- надійність системи;
- безпеку та захист даних системи;
- масштабованість та продуктивність системи;
- вимоги до користувацького інтерфейсу системи;
- доступність системи;
- обслуговування та підтримку системи.

Зупинимось детально на кожній вимозі.

Система повинна забезпечувати високу продуктивність, що передбачає обробку та відображення даних з мінімальним часовим затриманням. Зокрема, швидкість відповіді на запити користувачів не повинна перевищувати 200

мілісекунд для стандартних операцій, таких як перегляд балансу або історії транзакцій. Це досягається шляхом оптимізації алгоритмів обробки даних та використання ефективних методів кешування. Серверна частина повинна бути здатна обробляти до 10 000 запитів на хвилину без зниження продуктивності, що особливо важливо в пікові моменти торгової активності.

Система повинна бути надійною та стійкою до відмов, що означає мінімізацію ризиків виникнення системних помилок та забезпечення безперебійної роботи. Це досягається за рахунок використання кластерних рішень для серверної інфраструктури та резервного копіювання даних у режимі реального часу. У випадку відмови одного з компонентів система повинна автоматично переключатися на резервні ресурси без втрати даних і з мінімальним впливом на користувачів. Середній час відновлення після відмови (MTTR) не повинен перевищувати 5 хвилин, а середній час безвідмовної роботи (MTBF) повинен бути не менше 99,9%.

Забезпечення високого рівня безпеки є однією з головних вимог до системи. Всі дані користувачів, включаючи фінансові транзакції, повинні бути надійно захищені від несанкціонованого доступу. Це включає використання сучасних методів шифрування для зберігання та передачі даних, а також реалізацію багаторівневої системи автентифікації та авторизації. Система повинна також підтримувати журналювання всіх дій користувачів для можливості проведення ретроспективного аналізу та аудиту.

Забезпечення високого рівня безпеки фінансових даних та особистої інформації користувачів є критично важливим аспектом системи. Для цього використовуються сучасні методи шифрування, такі як AES-256 для зберігання даних та TLS 1.3 для захисту даних під час їх передачі. Кожен користувач повинен проходити багатофакторну автентифікацію (2FA) при вході в систему, що значно знижує ризики несанкціонованого доступу. Також передбачено регулярні аудити безпеки та тестування на проникнення (penetration testing), які проводяться щоквартально для виявлення та усунення потенційних вразливостей.

Система повинна бути здатною до ефективного масштабування, що дозволяє підтримувати зростаючу кількість користувачів та обсягів даних без зниження продуктивності. Для цього використовується мікросервісна архітектура, яка дозволяє окремо масштабувати різні компоненти системи в залежності від навантаження. Використання контейнеризації (Docker) та оркестрації (Kubernetes) забезпечує гнучкість та швидкість розгортання нових інстанцій сервісів. Система повинна бути здатною обробляти до 1 мільйона активних користувачів одночасно та зберігати більше 100 терабайтів даних з можливістю подальшого розширення.

Користувацький інтерфейс (UI) повинен бути інтуїтивно зрозумілим та зручним для використання. Основні функції системи, такі як додавання транзакцій, перегляд звітів та налаштування профілю, повинні бути легко доступними та зрозумілими навіть для користувачів з мінімальним технічним досвідом. Інтерфейс повинен підтримувати різні типи пристроїв, включаючи комп'ютери, планшети та смартфони, забезпечуючи адаптивний дизайн для комфортного використання на різних роздільних здатностях екрану. Це досягається шляхом використання сучасних фронтенд-технологій, таких як React або Vue.js, та ретельного тестування на різних пристроях.

Система повинна підтримувати багатомовність, що дозволить користувачам з різних країн користуватися програмою на своїй рідній мові. Крім того, інтерфейс має бути оптимізований для людей з обмеженими можливостями, включаючи підтримку екранних читалок та можливість налаштування розміру шрифту та контрастності кольорів.

Доступність системи повинна забезпечуватися на рівні 99,9% протягом року, що передбачає мінімальний час простою. Це досягається за рахунок використання резервних дата-центрів та механізмів автоматичного переключення на резервні сервери у випадку відмови основного. Крім того, система повинна підтримувати роботу у режимі 24/7 з наданням цілодобової технічної підтримки для користувачів.

Система повинна включати інструменти для моніторингу та обслуговування, такі як Prometheus та Grafana, що дозволяють відстежувати стан компонентів

системи, завантаження серверів, продуктивність баз даних та інші критичні параметри. Це дозволяє оперативно виявляти та усувати потенційні проблеми, забезпечуючи стабільну роботу системи.

Впровадження засобів автоматичного тестування, таких як Mockito та Junit5, забезпечує високий рівень якості коду та мінімізує ризик виникнення помилок. Swagger від OpenAPI та Postman використовуються для документування та тестування REST API, що спрощує процес розробки та підтримки системи.

Таким чином, реалізація всіх зазначених нефункціональних вимог дозволить створити високопродуктивну, надійну та безпечну систему, яка задовольнятиме потреби користувачів та забезпечуватиме високу якість обслуговування.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ

3.1 Моделювання бізнес процесів

Проведемо аналіз бізнес-процесів.

Система для обліку фінансових операцій на криптобіржі повинна підтримувати наступні основні бізнес-процеси:

- аутентифікація та авторизація користувача;
- введення та редагування особистих даних користувача;
- перевірка та підтвердження ідентифікації, данних користувача (імейл адреси, платіжних та інш даних);
- створення облікового запису користувача;
- облік фінансових транзакцій;
- введення даних про торгові операції з криптовалютами (купівля, продаж, обмін);
- обробка та збереження інформації про кожну транзакцію;
- автоматичне оновлення стану рахунків користувачів після кожної транзакції;
- генерація фінансових звітів;
- збір даних про фінансові операції за певний період;
- формування звіту про стан рахунків, доходи та витрати користувача;
- відображення аналітичних даних для прийняття управлінських рішень.

Програмна система має мати наступні можливі активності користувача:

а) доступ до додатку: користувач отримує доступ до програми.

б) реєстрація / вхід:

1) якщо користувач не зареєстрований, він може зареєструватися;

2) якщо зареєстрований, виконується вхід в систему;

в) отримання даних ринку: після входу система отримує актуальні ринкові дані;

г) головний екран: користувач потрапляє на головний екран, де доступні такі опції:

1) виконання транзакції;

2) вибір типу транзакції;

3) перевірка балансу в базі даних;

4) збереження транзакції в базі даних;

5) відправка повідомлення через брокер повідомлень;

д) перевірка курсів обміну: отримання курсів обміну з бази даних;

е) перевірка історії транзакцій;

ж) отримання транзакцій з бази даних;

з) завершення сесії: користувач може завершити роботу з системою в будь-який момент.

Ця система забезпечує базовий функціонал для управління криптовалютними операціями, включаючи автентифікацію, виконання транзакцій, перевірку курсів та історії операцій.

Для розуміння конкретного алгоритму дій користувача та реакції програмної системи на його дії у процесі проектування ПЗ було розроблено діаграму активностей. Деталі вказаних активностей показано на загальній діаграмі активностей, яка наведена на рис. 10.

Для побудови системи використаємо наступну архітектуру:

а) клієнтська частина:

1) Client App на пристрої користувача;

2) взаємодія з сервером через HTTP REST виклики на порти 80 та 443;

б) серверна частина (Digital Ocean cloud server instance) з Docker Engine, що містить:

1) Nginx load balancer network:

– Nginx container;

– Certbot container;

– Letsencrypt container;

2) Monitoring Network:

– Grafana container;

– Prometheus container;

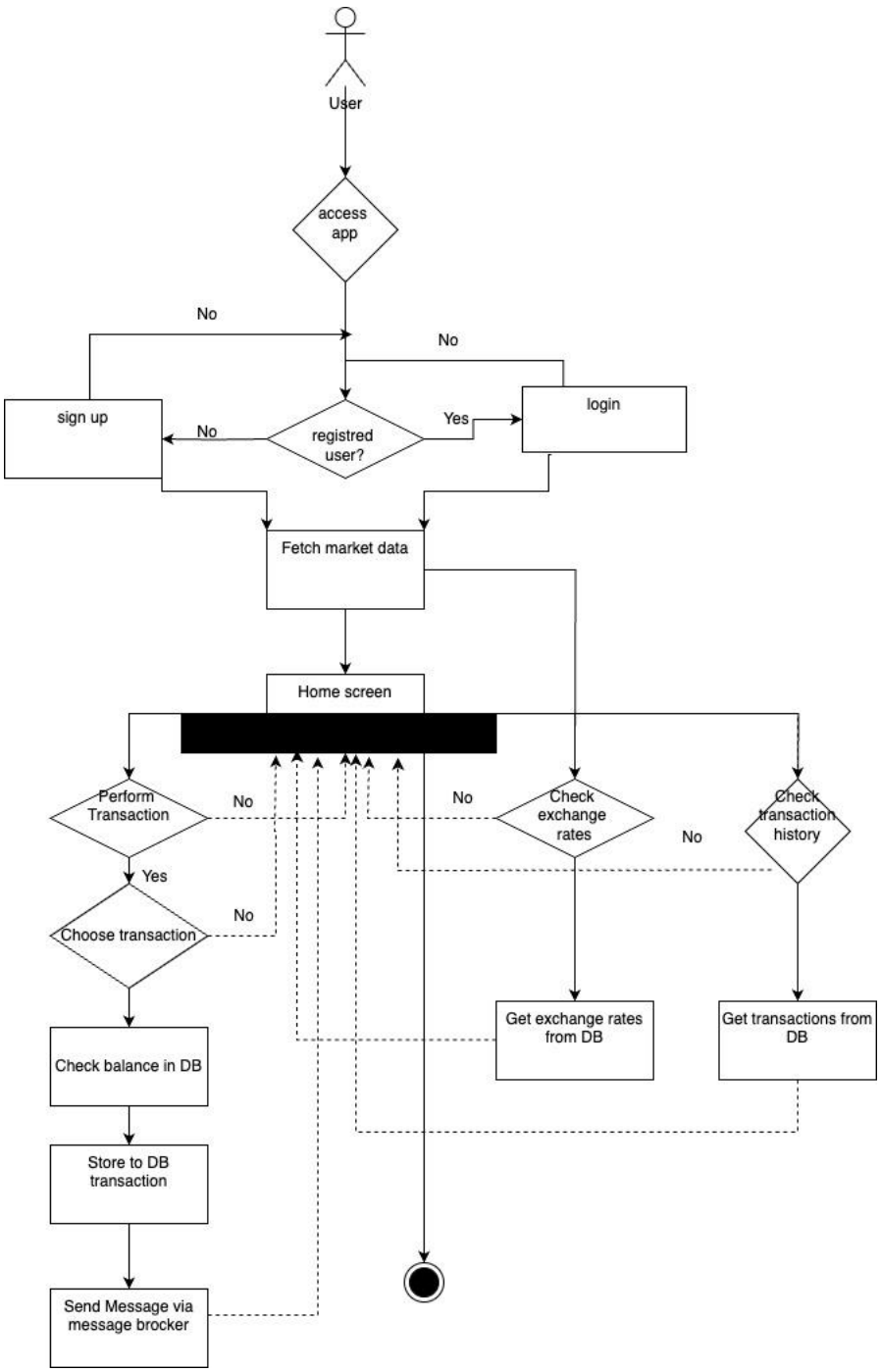


Рисунок 10 – Загальна діаграма активності (рисунок виконаний самостійно)

3) Microservices Network:

– Auth DB Image;

– Auth manager container;

– Binance manager container;

– Binance DB container;

– Payment manager DB;

– Payment manager container;

– RabbitMQ container;

– Wallet DB;

– Wallet manager container;

– Gateway manager container (на порту 8080);

Взаємодія компонентів відбувається наступним чином:

– Nginx balancer взаємодіє з Letsencrypt для SSL/T;LS;
– Certbot взаємодіє з Letsencrypt
– Gateway manager взаємодіє з мікросервісами через localhost:8080;
– мікросервіси взаємодіють між собою та з відповідними базами даних;
– RabbitMQ використовується для обміну повідомленнями між мікросервісами;
– Grafana та Prometheus забезпечують моніторинг системи;

Ця архітектура забезпечує масштабованість, безпеку та надійність системи управління крипто транзакціями, використовуючи мікросервісний підхід, контейнеризацію та хмарне розгортання.

ПЗ працює в режимі серверного API, яке дозволяє подальшу взаємодію як мобільних додатків на різних ОС(Android, IOS та ін.) так і різноманітних WEB клієнтів. Має у собі бізнес-логіку системи, а саме аналіз та обробку фінансових трансакцій крипто біржою. Головним актором виступає користувач - криптотрейдер.

Серверна частина має багатошарову архітектуру та містить наступні шари:

а) шар проксі сервера та HTTPS клієнта у вигляді NGINX в Docker в окремій Docker мережі;
б) шар моніторингу стану сервера та ПЗ за допомогою звязки Grafana та Prometheus в контейнерах Docker в окремій Docker мережі;
в) шар програмного забезпечення розміщеного в контейнерах Docker в окремій Docker мережі (сервіси і відповідні бази даних розміщені в окремих контейнерах, що забезпечує незалежність та стресостійкість ПО

для роботи за концепцією мікросервісної архітектури). Цей шар включає в себе:

1) шар доступу до даних, із забезпеченням коректності доступу;
2) шар сервісів, що містить основну бізнес-логіку програмної системи;
3) шар REST контролерів, за допомогою якого зовнішнім підсистемам надається доступ до сервера;
4) шар брокера повідомлень RabbitMQ для обміну повідомленнями між сервісами всередині мережі.

Клієнт-серверний зв’язок проходить у форматі JSON за протоколом HTTPS та за допомогою брокера повідомлень RabbitMQ.

Більш детально можна прослідкувати вказану архітектуру на діаграмі розгортання(див. рис. 11).

Відповідно до діаграми розгортання, серверна частина має розміщену в мережі Docker систему мікро сервісів, написаних на мові програмування Kotlin та Java, кожний з мікро сервісів розміщений в окремому контейнері Docker, з власною базою даних Postgres, що забезпечує консистентність та стресостійкість кожного окремого сервіса. Система мікросерсервісів включає в себе окремий сервіс аутентифікації та авторизації auth-manager для імплементації Security API, сервіс gateway-manager, який забезпечує реалізацію концепції Gateway API, сервіс конектор з Binance API – binance manager, а також payment-manager wallet manager, що дозволяє імплементувати логіку покупок(підписок) та збереження портфоліо користувача відповідно.

На підставі вказаної архітектури та загальна діаграма активності користувача побудуємо use case діаграму для різних акторів. На основі наданої діаграми Use Case для програмної системи управління крипто транзакціями можна виділити варіанти використання для користувача (User):

- реєстрація та вхід (User Registration and Login);
- пошук (Search), який розширюється на: додавання нової транзакції (Add new Transaction), оновлення транзакції (Update Transaction) та видалення транзакції (Delete Transaction);

– перегляд історії транзакцій (View Transaction History);

– отримання курсів обміну криптовалют (Get crypto exchange rates);

– генерація звітів (Generate reports).

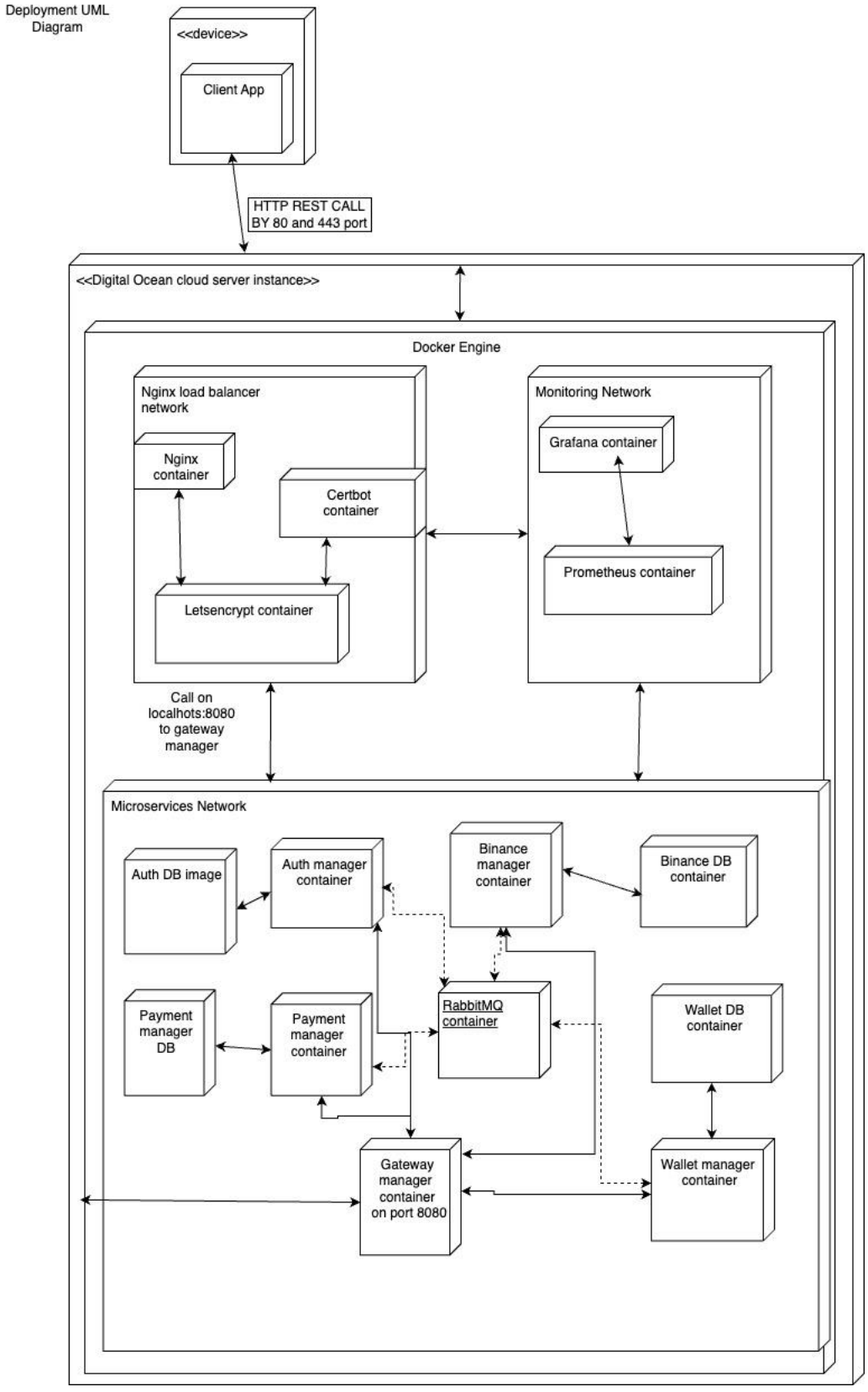


Рисунок 11 – Діаграма розгортання (рисунок виконаний самостійно)

Для адміністратора (Admin):

– генерація звітів (Generate reports);

– управління базою даних (Database Management);

– адміністративне управління (Admin management);

– управління чергою повідомлень (Message Queue Management).

Для системи (System):

– перевірка стану системи (System Health check);

– отримання ринкових даних (Fetch Market Data).

Пошук (Search) є центральним варіантом використання, який розширюється на додавання, оновлення та видалення транзакцій.

Перегляд історії транзакцій також розширює функціонал пошуку.

Адміністратор має додаткові можливості порівняно зі звичайним користувачем, включаючи управління системою та базою даних.

Система автоматично виконує перевірку свого стану та отримує актуальні ринкові дані.

Також у процесі проектування серверної частини програмної системи було створено діаграму варіантів використання (див. рис. 12), за допомогою якої було визначено акторів, що фігурують у програмній системі, визначено відносини між ними, а також можливості та доступні функції.

Додаткові спостереження щодо можливих use case сценаріїв:

Ця діаграма демонструє основні функції та взаємодії в системі управління крипто транзакціями, охоплюючи як користувацькі, так і адміністративні аспекти.

Детально можно розглянути use case діаграму, що наведена на рис.12.

Основне призначення діаграми - опис функціональності і поведінки, що дозволяє замовнику, кінцевому користувачеві і розробнику спільно обговорювати проектовану або існуючу систему. Отже, відповідно до створеної діаграми, у системі наявні такі ролі, як “User”, “System”, “Admin”.

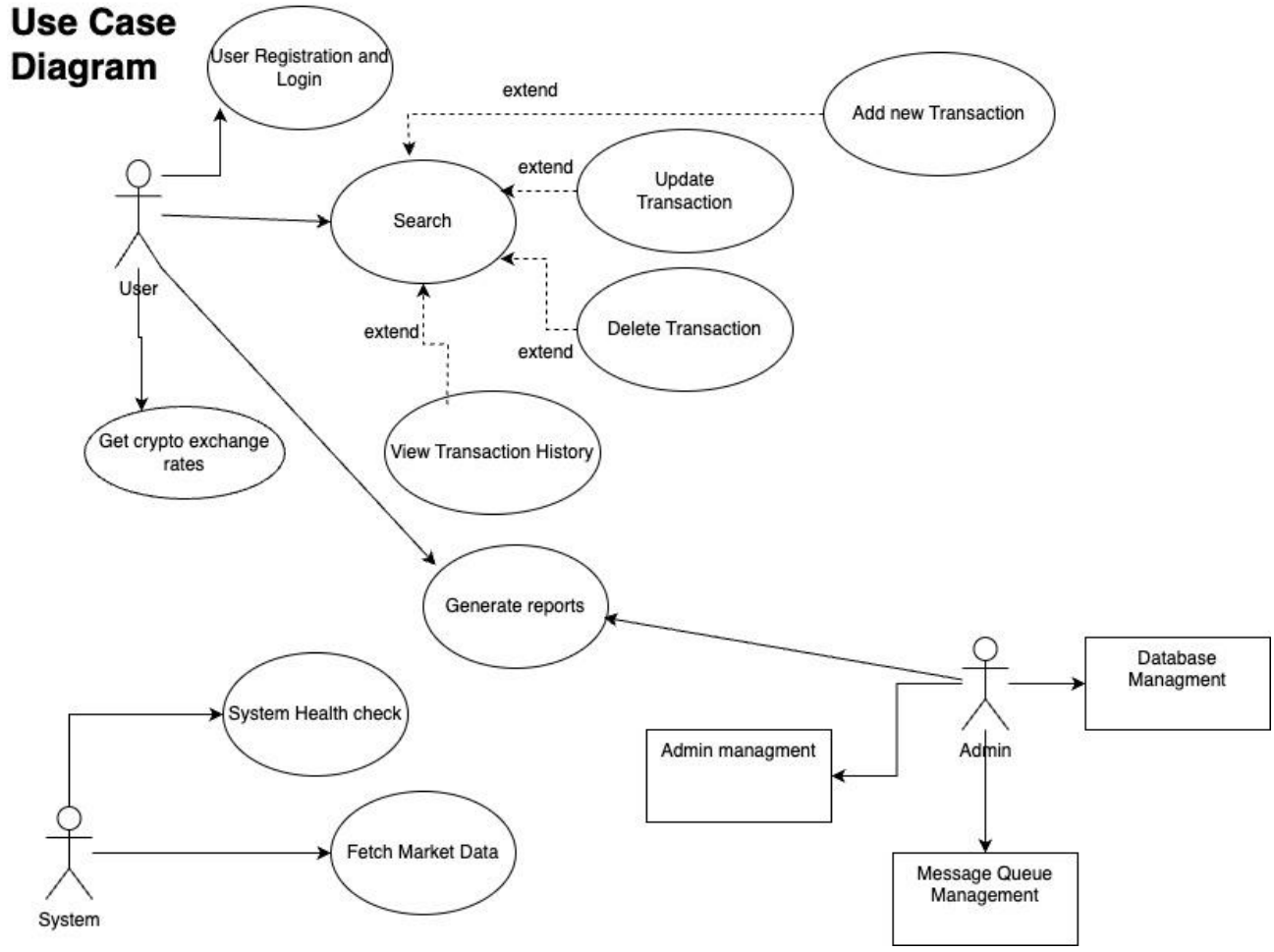


Рисунок 12 – Use-case діаграма (рисунок виконаний самостійно)

Ролі користувачів системи наведені у таблиці 1, а у таблиці 2 наведені повноваження користувачів. На підставі аналізу даних була побудована матриція ролей і повноважень, що відображає доступність тих чи інших можливостей системи в залежності від ролі користувача (див. табл.3).

Таблиця 1 – Ролі користувачів системи (таблиця виконана самостійно)

№	Роль	Опис
1	Admin	Адмін програмної системи, має найбільші привілеї
2	User	Власник та користувач клієнта на якому буде розгортатися ПЗ, що буде взаємодіяти з серверною частиною ПЗ
3	System	Система, яка буде стежити за усіма бізнес-процесами та станом сервера.

Таблиця 2 – Повноваження користувачів (таблиця виконана самостійно)

№	Повноваження	Опис
1	Зареєструватися в системі	Виконати реєстрацію у системі
2	Здійснити управління трансакціями	Пошук, додавання, редагування та видалення трансакцій
3	Здійснити управління користувачами	Додавання, редагування та видалення користувачів
4	Змінити мову інтерфейсу	Змінити мову інтерфейсу на англійську або українську
5	Керувати профілем у системі	Редагувати та видаляти профіль користувача
6	Отримати статистику	Отримання статистики дій користувача та статистичних показників на щодо окремої монети
7	Пошук по назві монет в бд	Пошук по назві монет в бд для додавання в портфоліо трансакції з монетою
8	Зробити резервну копію БД	Створення файлу з копією бд для відновлення
9	Здійснити управління сертифікатами	Додавання сертифікату та продовження його дії

Таблиця 3 - Матриця ролей (таблиця виконана самостійно)

	Роль 1 (Admin)	Роль 2 (User)	Роль 3 (System)
Повноваження 1	X	X	
Повноваження 2	X	X	
Повноваження 3	X	X	
Повноваження 4	X	X	
Повноваження 5	X	X	
Повноваження 6	X	X	X
Повноваження 7	X	X	
Повноваження 8	X		X
Повноваження 9	X		X

3.2 Проектування сховища даних

Визначемо основні сутності системи та їх взаємозв'язки для забезпечення чіткого розуміння структури даних і логіки роботи системи.

User Account є центральною сутністю системи, яка представляє обліковий запис користувача. Вона має такі атрибути: id, account_non_expired,

account_non_locked, created_at, date_of_birth, email, first_name, gender, last_name, picture, status, type, updated_at. Обліковий запис користувача має один до одного відношення з user_login_data та user_login_data_external, і один до багатьох відношення з такими сутностями, як email_validation, reset_password, user_roles, user_session, purchase_user_data та user_receipts.

User Login Data зберігає інформацію про облікові дані користувача, включаючи атрибути id, created_at, updated_at, login_name, password_hash. Ця сутність має багато до одного відношення з user_account та login_type.

User Login Data External зберігає зовнішні облікові дані користувачів, такі як id, external_provider_token. Вона має багато до одного відношення з user_account та external_provider.

Email Validation та Reset Password є сутностями, що відповідають за валідацію електронної пошти та скидання пароля відповідно. Вони мають однакові атрибути: id, created_at, updated_at, status, token та мають багато до одного відношення з user_account.

User Roles визначають ролі користувачів в системі і мають атрибути id, created_at, updated_at. Вони пов'язані багато до одного відношенням з user_account та role.

User Session зберігає інформацію про сесії користувачів, включаючи атрибути id, created_at, updated_at, device_token, device_type, expire_time, ip_address, token. Ця сутність має багато до одного відношення з user_account.

Purchase User Data зберігає інформацію про покупки користувачів, має атрибути id, created_at, updated_at, device_type, purchase_id, status та багато до одного відношення з user_account.

User Receipts є сутністю, що зберігає квитанції користувачів. Вона має атрибути id, created_at, updated_at та багато до одного відношення з user_account, а також один до багатьох відношення з android_receipt та ios_receipt.

Android Receipt та iOS Receipt зберігають квитанції для відповідних платформ. Android квитанція має атрибути id, created_at, updated_at, adam_id, app_item_id, application_version, bundle_id, download_id, environment,

original_application_version, original_purchase_date, original_purchase_date_ms, original_purchase_date_pst, receipt_creation_date, receipt_creation_date_ms, receipt_creation_date_pst, receipt_type, request_date, request_date_ms, request_date_pst, status, version_external_identifier. iOS квитанція має атрибути id, created_at, updated_at, bundle_id, environment, latest_receipt, original_purchase_date, product_id, receipt_creation_date, request_date, status, transaction_id. Обидві сутності мають багато до одного відношення з user_receipts.

Role визначає ролі у системі з атрибутами id, description, name та має один до багатьох відношення з granted_permissions та role_privilege.

Permission визначає дозволи у системі з атрибутами id, description, name та має один до багатьох відношення з granted_permissions та role_privilege.

External Provider визначає зовнішніх провайдерів аутентифікації з атрибутами id, created_at, updated_at, name, status та має один до багатьох відношення з user_login_data_external.

Login Type визначає типи входу в систему з атрибутами id, name, status та має один до багатьох відношення з user_login_data.

Currency визначає валюту, яка використовується в системі, з атрибутами id, created_at, updated_at, name, symbol, exchange_rate та має один до багатьох відношення з wallet та orders.

Wallet представляє гаманець користувача з атрибутами login_account_id, balance, created_at, updated_at, name та має один до багатьох відношення з orders.

Orders представляє замовлення у системі з атрибутами id, amount, created_at, price, updated_at, operation, status та має багато до одного відношення з currency та wallet.

Детальна структура сутностей вказана в ER-діаграмі (див. рис. 13, 14, 15).

Логічна модель бази даних, яка включає таблиці, поля, ключі та індекси, а також взаємозв'язки між таблицями для забезпечення цілісності даних:

Скрипт для створення таблиці external_provider:

```
CREATE TABLE external_provider (
id bigint Primary Key,
```

```
created_at timestamp NOT NULL,
updated_at timestamp NOT NULL,
name varchar(255), unique  NOT NULL,
status varchar(255)).
```

Звернемо увагу, що поле name має бути унікальним, що виконано з використанням обмеження unique, а поле status може приймати значення NULL.

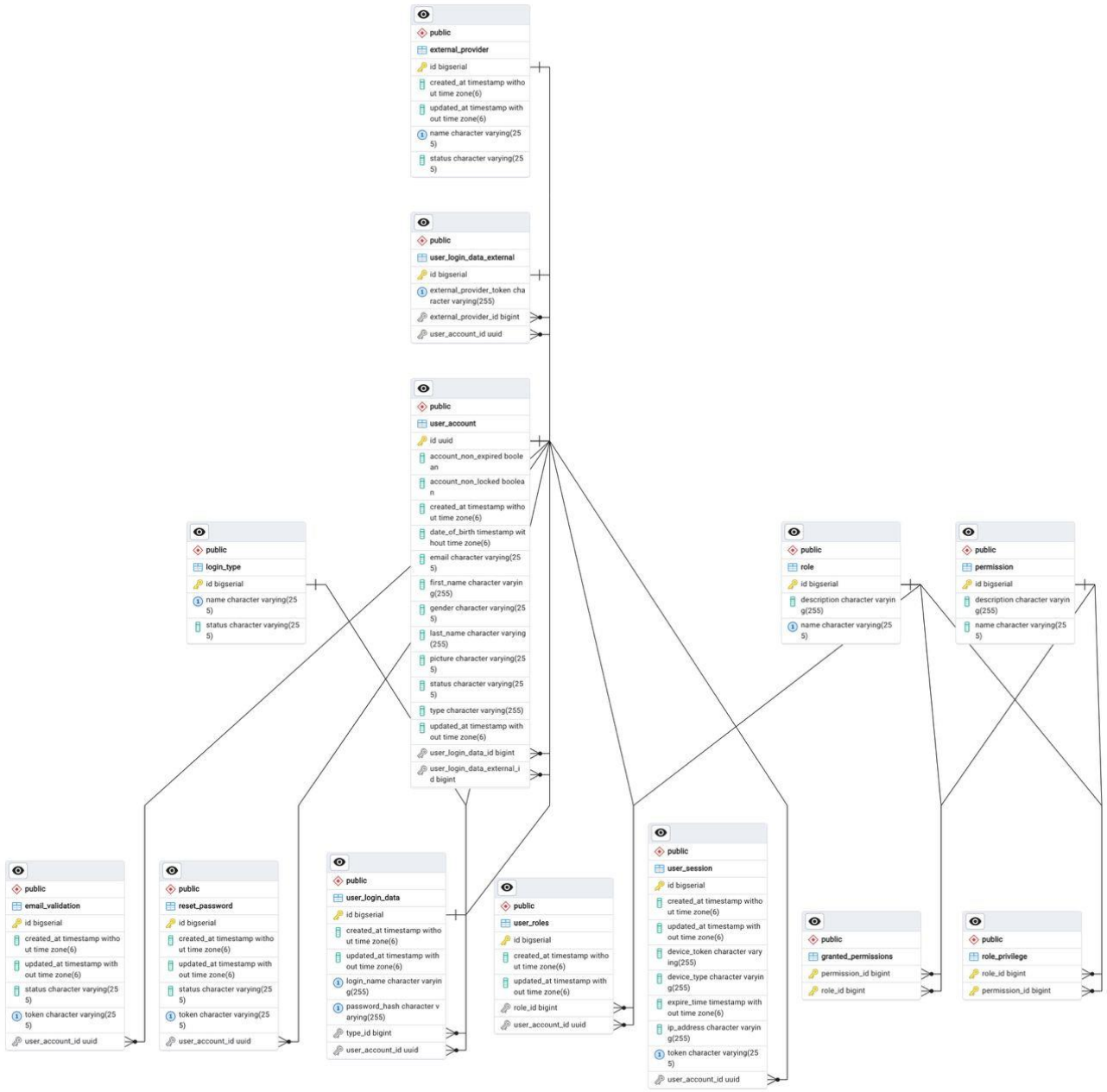


Рисунок 13 – ER діаграма auth_crypto_db (рисунок виконаний самостійно)

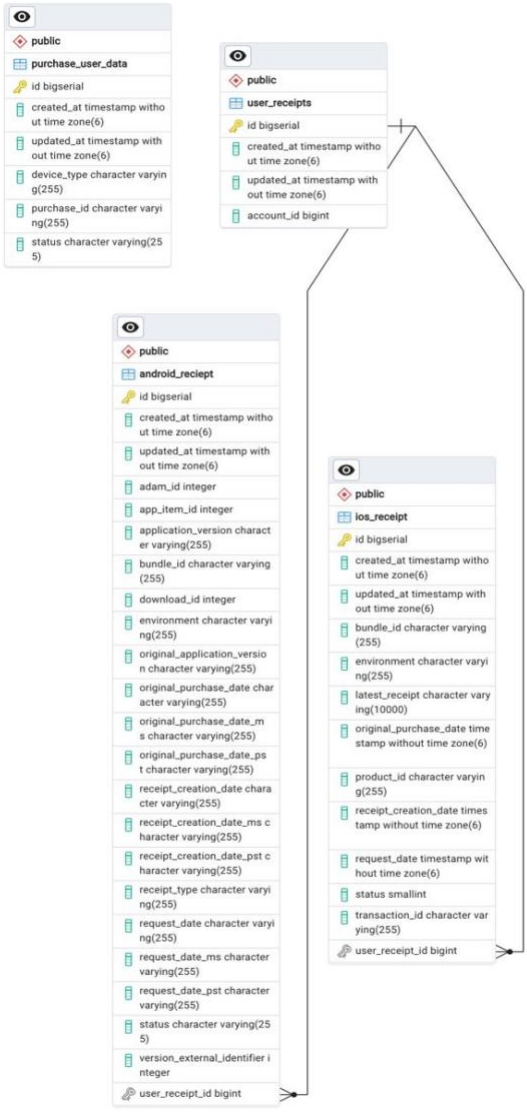


Рисунок 14 – ER діаграма payment_crypto_db (рисунок виконаний самостійно)

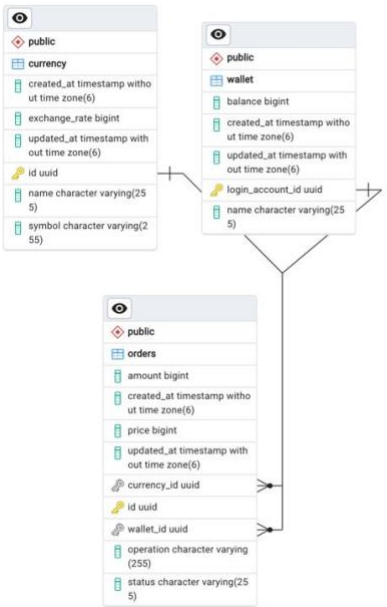


Рисунок 15 – ER діаграма wallet_crypto_db (рисунок виконаний самостійно)

Скрипт для створення таблиці login_type:

```
CREATE TABLE login_type (
    id bigint PRIMARY KEY,
    name varchar(255) UNIQUE NOT NULL,
    status varchar(255)
);
```

Звернемо увагу, що поле name є унікальним ім'ям типу входу користувача, що виконано з використанням обмеження unique, та не може бути NULL.

Скрипт для створення таблиці permission:

```
CREATE TABLE permission (
    id bigint PRIMARY KEY,
    description varchar(255),
    name varchar(255)
);
```

В таблиці permission не має унікальних полів або зовнішніх ключів.

Скрипт для створення таблиці role:

```
CREATE TABLE role (
    id bigint PRIMARY KEY,
    description varchar(255),
    name varchar(255) UNIQUE NOT NULL
);
```

Звернемо увагу, що поле name є унікальним ім'я ролі користувача, що виконано з використанням обмеження unique, та не може бути NULL.

Скрипт для створення таблиці granted_permissions:

```
CREATE TABLE granted_permissions (
    permission_id bigint REFERENCES permission(id),
    role_id bigint REFERENCES role(id),
    PRIMARY KEY (permission_id, role_id)
);
```

Звернемо увагу, що поле permission_id є зовнішнім ключем до permission(id), а поле role_id є зовнішнім ключем до role(id).

Скрипт для створення таблиці role_privilege:

```
CREATE TABLE role_privilege (
    role_id bigint REFERENCES role(id),
```

```
    permission_id bigint REFERENCES permission(id),
    PRIMARY KEY (role_id, permission_id)
);
```

Звернемо увагу, що поле permission_id є зовнішнім ключем до permission(id), а поле role_id є зовнішнім ключем до role(id).

Скрипт для створення таблиці user_account:

```
CREATE TABLE user_account (
    id uuid PRIMARY KEY,
    account_non_expired boolean,
    account_non_locked boolean,
    created_at timestamp,
    date_of_birth timestamp,
    email varchar(255) UNIQUE NOT NULL,
    first_name varchar(255),
    gender varchar(255),
    last_name varchar(255),
    picture varchar(255),
    status varchar(255),
    type varchar(255),
    updated_at timestamp,
    user_login_data_id bigint REFERENCES user_login_data(id),
    user_login_data_external_id bigint REFERENCES
user_login_data_external(id)
);
```

Звернемо увагу, що поле email є унікальним полем для адреси електронної пошти користувача, що виконано з використанням обмеження unique, та не може бути NULL, поле user_login_data_id є зовнішнім ключем до user_login_data(id), поле user_login_data_external_id є зовнішнім ключем до user_login_data_external(id).

Скрипт для створення таблиці email_validation:

```
CREATE TABLE email_validation (
    id bigserial PRIMARY KEY,
    created_at timestamp,
    updated_at timestamp,
    status varchar(255),
    token varchar(255) UNIQUE,
    user_account_id uuid REFERENCES user_account(id)
);
```

Звернемо увагу, що поле token є унікальним полем для токену підтвердження електронної пошти, а поле user_account_id є зовнішній ключем до user_account(id).

Скрипт для створення таблиці reset_password:

```
CREATE TABLE reset_password (
    id bigserial PRIMARY KEY,
    created_at timestamp,
    updated_at timestamp,
    status varchar(255),
    token varchar(255) UNIQUE,
    user_account_id uuid REFERENCES user_account(id)
);
```

Звернемо увагу, що поле token є унікальним полем для токену скидання пароля, а поле user_account_id є зовнішнім ключем до user_account(id).

Скрипт для створення таблиці user_login_data:

```
CREATE TABLE user_login_data (
    id bigserial PRIMARY KEY,
    created_at timestamp,
    updated_at timestamp,
    login_name varchar(255) UNIQUE,
    password_hash varchar(255) UNIQUE,
    type_id bigint REFERENCES login_type(id),
    user_account_id uuid REFERENCES user_account(id)
);
```

Звернемо увагу, що поле login_name є унікальним полем для імені входу користувача, а password_hash є унікальним полем для хешу пароля. Крім того поле type_id є зовнішнім ключем до login_type(id), а поле user_account_id є зовнішнім ключем до user_account(id).

Скрипт для створення таблиці user_login_data_external:

```
CREATE TABLE user_login_data_external (
    id bigserial PRIMARY KEY,
    external_provider_token varchar(255) UNIQUE,
    external_provider_id bigint REFERENCES external_provider(id),
    user_account_id uuid REFERENCES user_account(id)
);
```

Звернемо увагу, що поле external_provider_token є унікальним полем для токену зовнішнього постачальника. В свою чергу поле external_provider_id є зовнішнім ключем до external_provider(id), а поле user_account_id є зовнішнім ключем до user_account(id).

Скрипт для створення таблиці user_roles:

```
CREATE TABLE user_roles (
```

```
    id bigserial PRIMARY KEY,
    created_at timestamp,
    updated_at timestamp,
    role_id bigint REFERENCES role(id),
    user_account_id uuid REFERENCES user_account(id)
);
```

Звернемо увагу, що поле role_id є зовнішнім ключем до role(id), а user_account_id є зовнішнім ключем до user_account(id).

Скрипт для створення таблиці user_session:

```
CREATE TABLE user_session (
    id bigserial PRIMARY KEY,
    created_at timestamp,
    updated_at timestamp,
    device_token varchar(255),
    device_type varchar(255),
    expire_time timestamp,
    ip_address varchar(255),
    token varchar(255) UNIQUE,
    user_account_id uuid REFERENCES user_account(id)
);
```

Звернемо увагу, що поле token є унікальним полем для токену сесії користувача, а поле user_account_id є зовнішнім ключем до user_account(id).

Ця логічна модель відображає структуру бази даних із зазначеними таблицями, полями та взаємозв'язками.

Основна таблиця системи — це user_account, яка пов'язана з іншими таблицями через зовнішні ключі (FK). Вона містить інформацію про користувачів, включаючи їхні облікові дані, статус та особисту інформацію.

Таблиці user_login_data та user_login_data_external зберігають інформацію про облікові дані для внутрішнього та зовнішнього входу відповідно. Ці таблиці пов'язані з user_account через поля user_account_id.

Таблиці email_validation та reset_password зберігають інформацію про верифікацію електронної пошти та скидання паролів. Вони також містять поля user_account_id, що зв'язують їх з основною таблицею user_account.

Таблиці user_roles та user_session зберігають інформацію про ролі користувачів та активні сесії відповідно. Вони також мають зв'язки з user_account через поля `user_account_id

Таблиця purchase_user_data зберігає інформацію про покупки користувачів, включаючи тип пристрою, ідентифікатор покупки та статус операції.

Таблиця user_receipts є центральною для зберігання інформації про квитанції користувачів. Вона зв'язана з таблицями android_receipt та ios_receipt, що містять конкретні дані про купівлі на платформах Android та iOS.

Таблиця currency містить інформацію про валюти, які використовуються в системі. Вона зв'язана з таблицями wallet та orders, що дозволяє ефективно відслідковувати та аналізувати фінансові операції користувачів.

Таблиця wallet зберігає інформацію про гаманці користувачів, включаючи залишок та ім'я гаманця. Вона зв'язана з таблицею orders, що дозволяє користувачам виконувати фінансові операції в системі.

Таблиця orders містить дані про замовлення, включаючи суму, валюту та інші атрибути, які дозволяють системі відстежувати та обробляти фінансові операції користувачів.

Ця логічна модель бази даних відображає складні взаємозв'язки між різними аспектами системи, включаючи управління користувачами, їх доступами, фінансовими операціями та іншими ключовими функціями. Взаємозв'язки через зовнішні ключі дозволяють забезпечити цілісність даних та ефективність обробки інформації в базі даних.

3.3 Приклади методів та алгоритмів

Функціонал кожного мікросервісу розділений на сервісний, контролер, сутності (ентіті), виключення (ексепшин,) модел, конфіг, репозиторій шари.

Кожен з шарів відповідає за відповідний функціонал мікросервісу: сервісний за бізнес процеси, контролер за вибудову REST відповідей та відповідну взаємодію, шар виключень дає можливість кастомізувати логіку відстеження і оброблення помилок, шар сутностей – відповідає за побудову сутностей для бази данних, модел – за побудову моделей для реалізації патерну DTO (data transfer object), репозиторій – взаємодію з базою данних, а конфігураційний - за відповідні конфігурації.

Таким чином забезпечується гнучкість та багаташаровість кожного мікросервісу.

Наведемо приклад контролеру, за допомогою якого отримується інформація з Binance API (див. рис. 16).

```
@RestController("/connector") new *
class ConnectorController(
    private val connectorService: ConnectorService
) {
    @GetMapping("/v1/exchange-info") new *
    fun getExchangeInfo() = connectorService.getExchangeInfo()
}
```

Рисунок 16 – Connector” контролер binance-manager мікросевіса в Intelij IDEA (виконаний самостійно)

Далі представимо сервісний метод отримання інформації з публічного API (див. рис. 17), який викликається в репозиторії представленому вище.

```
@Service new *
class ConnectorServiceImpl(
    private val restTemplate: RestTemplateHttpRestClient,
    @Value("\${binance.api.url}")
    private val binanceUrl: String,
) : ConnectorService {
    private val logger = KotlinLogging.logger {}

    override fun getExchangeInfo(): JsonNode { new *
        logger.info { "Getting exchange info from Binance" }
        return restTemplate.executeRequest(
            headers = HttpHeaders(),
            url = "$binanceUrl/eapi/v1/exchangeInfo",
            httpMethod = HttpMethod.GET,
            requestBody = null,
            responseDtoClass = JsonNode::class.java
        )
    }
}
```

Рисунок 17 – Сервісний шар binance-manager мікросевіса в Intelij IDEA (за даними crypto-pet-project проекта)

На підставі зазначених методів та алгоритмів забезпечується дотримання основних принципів програмування, що дає можливість маштабувати продукт і вносити зміни відповідно до вимог клієнта.

3.4 Опис UI/UX

UI/UX частина додатку представлена HTML-сторінками, призначеними для відображення інформації про котирування криптовалют, портфоліо криптовалют користувача, з можливістю додавання валют, зміни портфоліо, шляхом проведення певних трансакцій і їх логування. Базовою сторінкою є сторінка хоум екрана, на яку користувач потрапляє після логіна або реєстрації. На хоум сторінці доступні кнопки які відповідають за відповідні активності, які імплементовані серверною частиною.

Сторінки побудовані з використанням HTML5. Для стилізації використовуються вбудовані CSS-стилі, які забезпечують приємний візуальний вигляд та адаптивність дизайну.

Основний контент сторінки розміщений у контейнері з обмеженою шириною, що центрується на екрані. Це досягається за допомогою CSS-властивостей max-width та margin: auto. Сторінки містять відповідні заголовки та кнопки для отримання інформації та область для відображення даних.

JavaScript-код, вбудований у сторінки, відповідає за функціональність. Наприклад при кліку на кнопку "Get Exchange Info" виконується асинхронний запит до API '/connector/v1/exchange-info' за допомогою Fetch API. Отримані дані у форматі JSON обробляються та відображаються на сторінці у відформатованому вигляді. У разі виникнення помилки під час запиту, користувачеві показується відповідне повідомлення.

```
<script>
  document.getElementById('fetchInfo').addEventListener('click', () => {
    fetch('/connector/v1/exchange-info')
            .then(response => response.json())
            .then(data => {
              document.getElementById('exchangeInfo').innerHTML =
`<pre>${JSON.stringify(data, null, 2)}</pre>`;
            })
```

```
            .catch(error => {
              document.getElementById('exchangeInfo').innerHTML = `<p>Error
fetching data: ${error}</p>`;
            });
  });
</script>
```

Сторінки розроблені з урахуванням принципів адаптивного дизайну. Мета-тег viewport забезпечує коректне відображення на мобільних пристроях, а обмеження ширини контейнера покращує вигляд на великих екранах.

Загалом, це простий, але ефективний приклад додатку, який демонструє взаємодію з API, динамічне оновлення контенту та базову обробку помилок. Код написаний структуровано, що робить його легким для розуміння та подальшої модифікації.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Розгортання сервісної інфраструктури

Для розгортання сервісної інфраструктури програмного продукту було обрано хмарний хостинг DigitalOcean (див. рис.18).

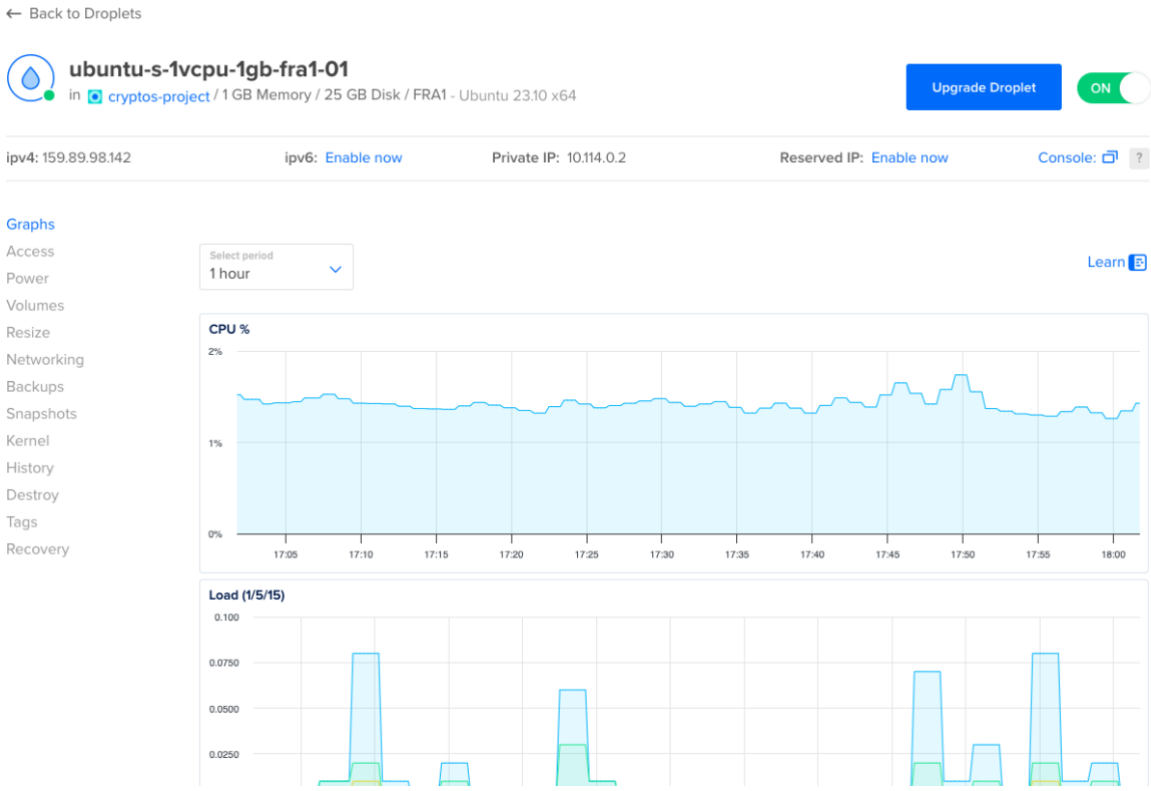


Рисунок 18 – UI хмарного хостингу DigitalOcean (рисунок зроблений самостійно)

DigitalOcean є відмінним вибором завдяки своїм хмарним дроплетам, які надають еластичні ресурси для серверів. Це дозволяє динамічно змінювати обсяги обчислювальної потужності та об'єму пам'яті в залежності від потреб додатку, що забезпечує високу доступність та швидку реакцію на зростання навантаження.

Додаток розгортався на DigitalOcean з використанням Docker для контейнеризації різних частин системи.

Доступ до серверної частини відбувається через ssh запит: «ssh -i "/Users/oleksii/.ssh/cryptos_rsa" root@159.89.98.142» з застосуванням SSH ключа для забезпечення безпеки зєднання (див. рис. 19).

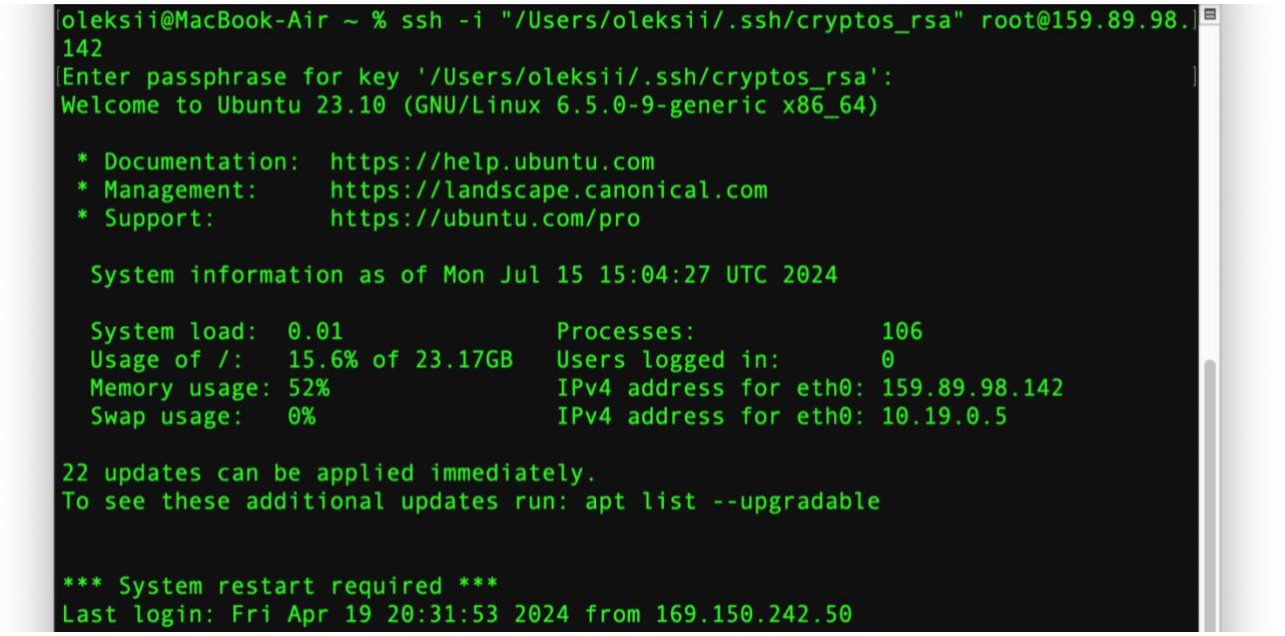


Рисунок 19 – Ssh tunnel to DigitalOcean droplet (рисунок зроблений самостійно)

Docker дозволяє ізолювати додатки в контейнерах, що спрощує їх розгортання та забезпечує однакове середовище для різних етапів розробки та впровадження (див. рис. 20).

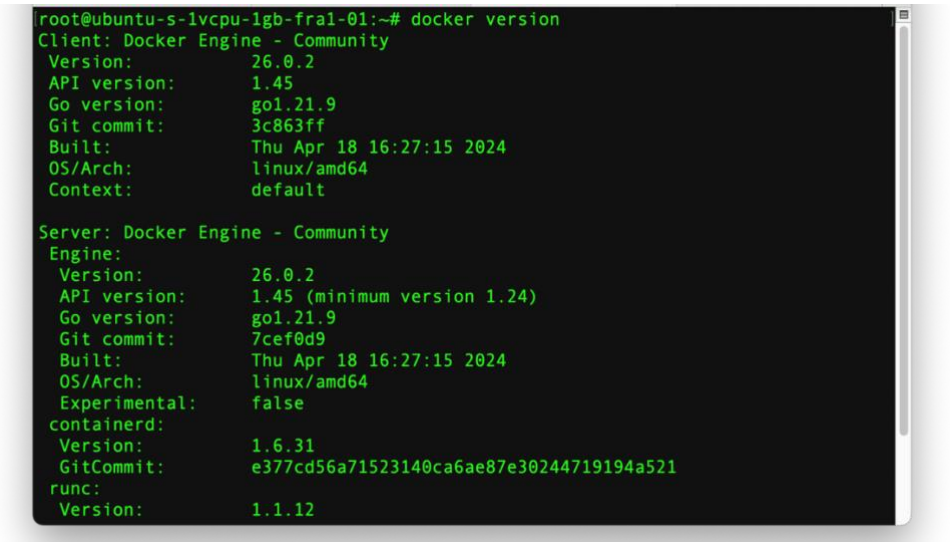


Рисунок 20 – docker --version command performing in droplet (рисунок зроблений самостійно)

NGINX також розгорнуто в контейнері на сервері, що дозволяє легко розгортати та масштабувати проект (див. рис. 21).

Рисунок 21 – docker ps command performing in droplet (рисунок зроблений самостійно)

Налаштування NGINX (див. рис. 22, 23, 24) для реалізації поставлених завдань без 443 порту (так як відсутнє доменне імя та відповідний SSL сертифікат, що може бути змінено в майбутньому після придбання(резервування) доменного імені) та з додатковими налаштуваннями логів для можливості використання сервісу Amplify для моніторингу стану NGINX, запитів та серверного середовища.

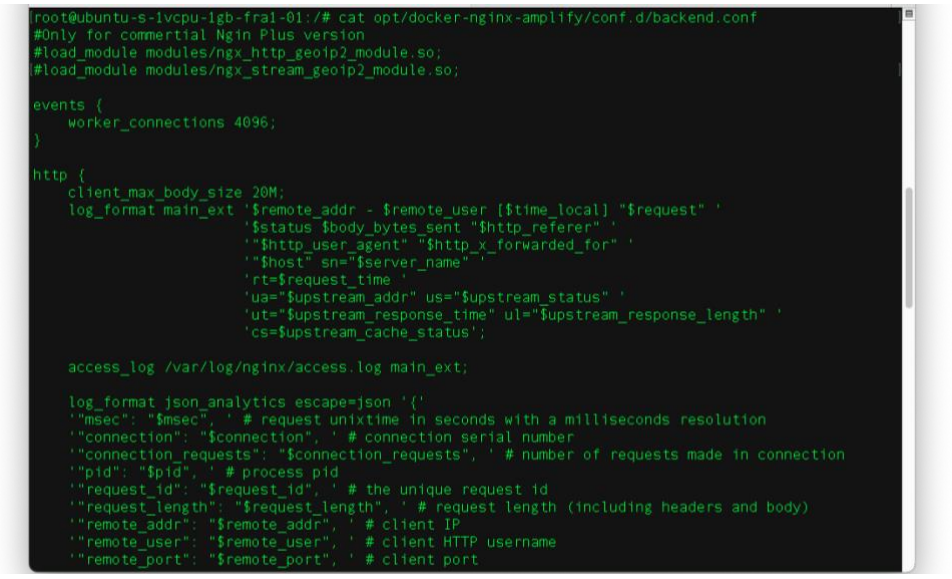


Рисунок 22 – NGINX backend.conf file ч.1 (рисунок зроблений самостійно)

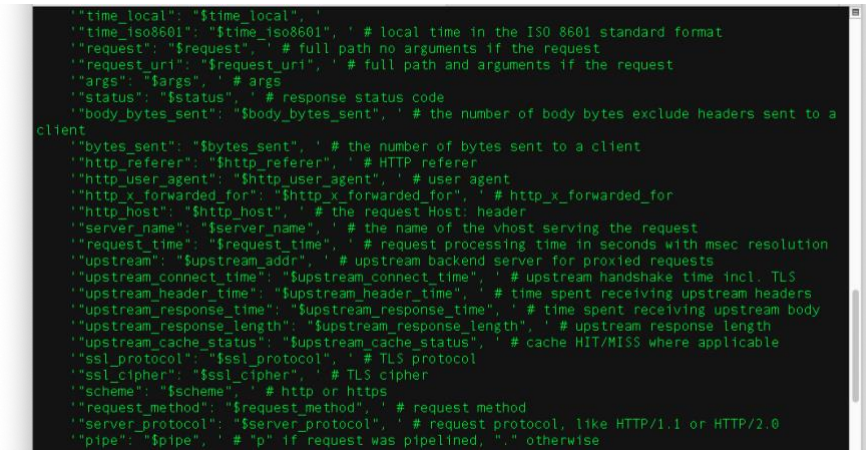
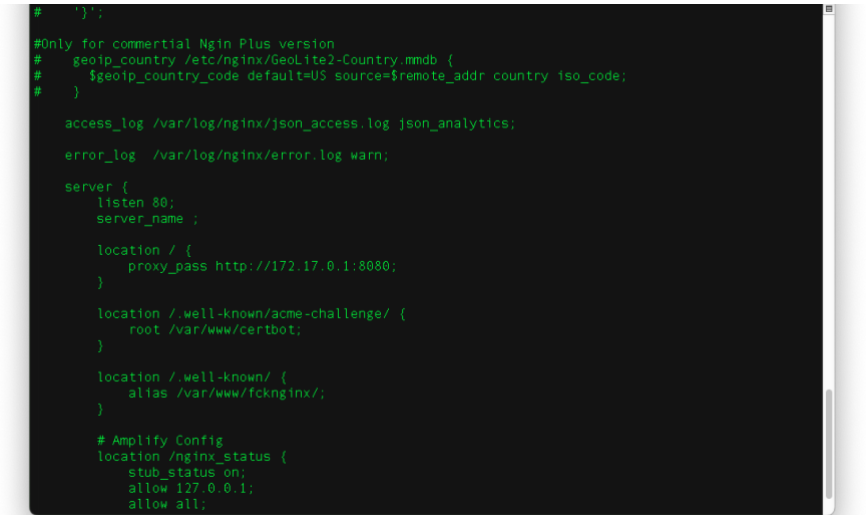


Рисунок 23 – NGINX backend.conf file ч.2 (рисунок зроблений самостійно)



```
#    '}';

#Only for commertial Ngin Plus version
#    geoip_country /etc/nginx/GeoLite2-Country.mmdb {
#      $geoip_country_code default=US source=$remote_addr country iso_code;
#    }

    access_log /var/log/nginx/json_access.log json_analytics;

    error_log  /var/log/nginx/error.log warn;

    server {
        listen 80;
        server_name ;

        location / {
            proxy_pass http://172.17.0.1:8080;
        }

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

        location /.well-known/ {
            alias /var/www/fcknginx/;
        }

        # Amplify Config
        location /nginx_status {
            stub_status on;
            allow 127.0.0.1;
            allow all;
```

Рисунок 24 – NGINX backend.conf file ч.3 (рисунок зроблений самостійно)

Такі налаштування NGINX дають можливість переглянути системні метрики за адресою https://amplify.nginx.com/dashboard/33618 (див. рис. 25) та загальну інформацію по роботі NGINX та сервера за адресою https://amplify.nginx.com/overview/ .

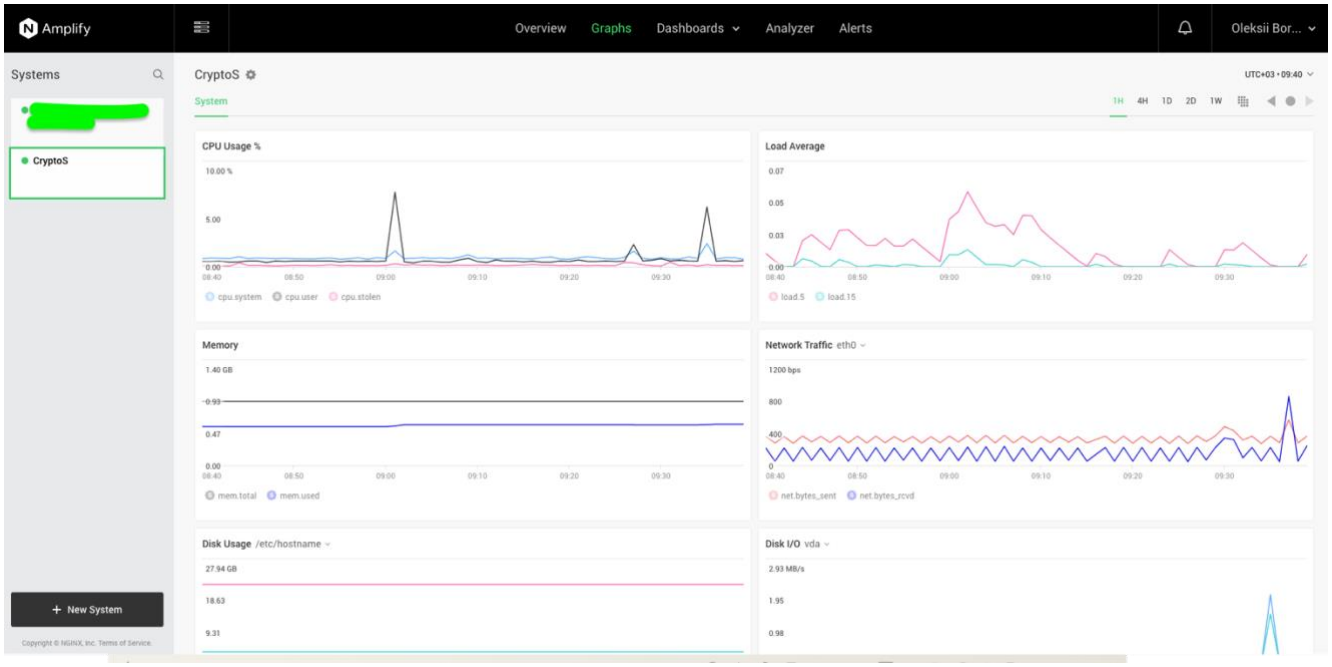


Рисунок 25 – UI Системні метрики NGINX (рисунок зроблений самостійно)

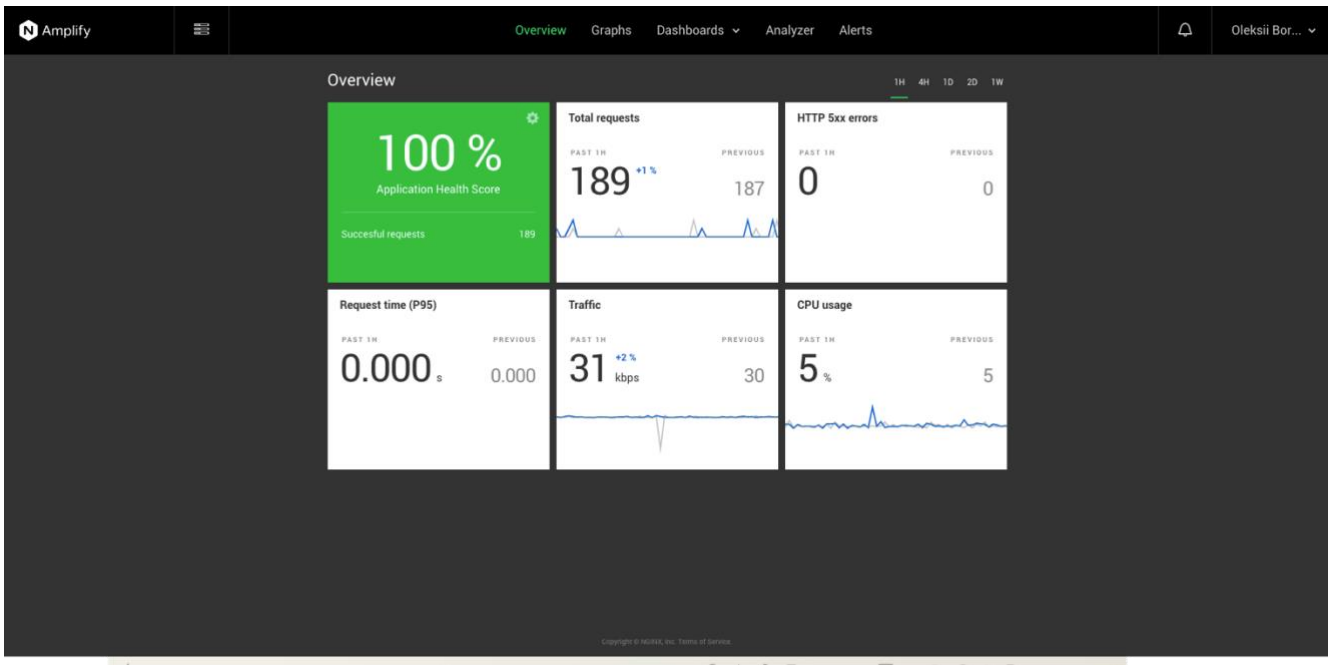


Рисунок 26 – UI загальної інформації сервера та NGINX (рисунок зроблений самостійно)

Крім того, Amplify дозволяє додати alerts які будуть надходити на зазначену електронну пошту, які можливо привязати до відповідних івентів, таких як: зменшення дискового простору, відсутність оновлення статусу клієнта та ін. Такі alerts можливо додати за адресою https://amplify.nginx.com/alerts/ (див. рис. 27).

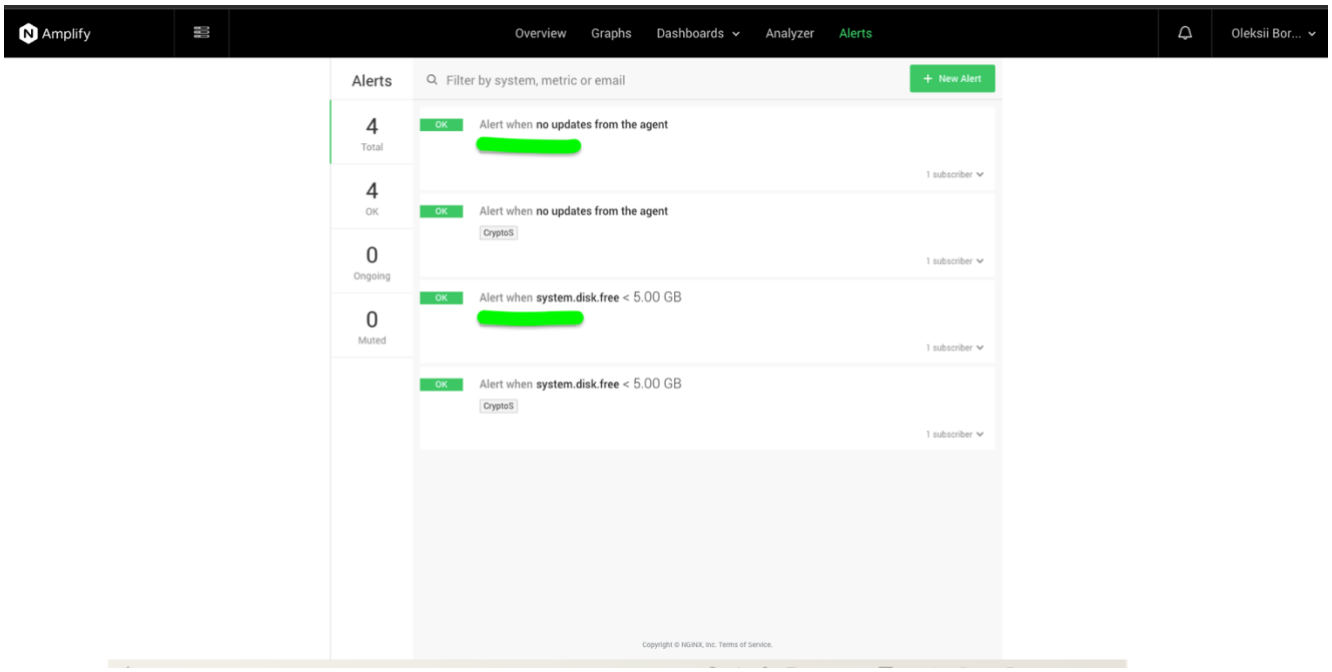


Рисунок 27 – UI загальної інформації сервера та NGINX (рисунок зроблений самостійно)

4.2 Створення мікросервісної архітектури додатку

Основною архітектурною концепцією розробленої системи є мікросервісна архітектура. Кожен мікросервіс відповідає за конкретний аспект функціональності: автентифікація користувачів, управління фінансовими транзакціями, інтеграція з API Binance та інші завдання (див. рис. 28).

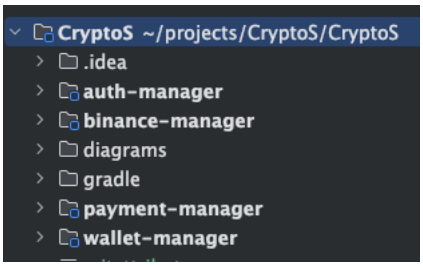


Рисунок 28 – Загальна структура crypto-pet-project(CryptoS) проекта в Intelij IDEA (рисунок зроблений самостійно)

Така розбита на модулі архітектура дозволяє ефективно масштабувати окремі компоненти системи, розвивати їх незалежно один від одного та спрощує супровід системи в цілому.

Проект використовує репозиторій Git та платформу GitHub для віддаленого зберігання програмного застосунку за адресою: https://github.com/Uka0001/crypto-pet-project.

Кожен мікросервіс був реалізований з використанням Spring Boot 3.2.3, що забезпечує швидку розробку та високу ефективність. Кожен сервіс також має власну базу даних, що дозволяє підтримувати велику кількість користувачів та обробляти великий потік даних без значного впливу на загальну продуктивність системи.

Системою автоматичного збирання був обраний Gradle, що є поширеною практикою в проектах написаних на мові програмування Kotlin. Наведемо приклад build.gradle.kts файлу одного з мікросервісів з відповідними залежностями (див. рис. 29):

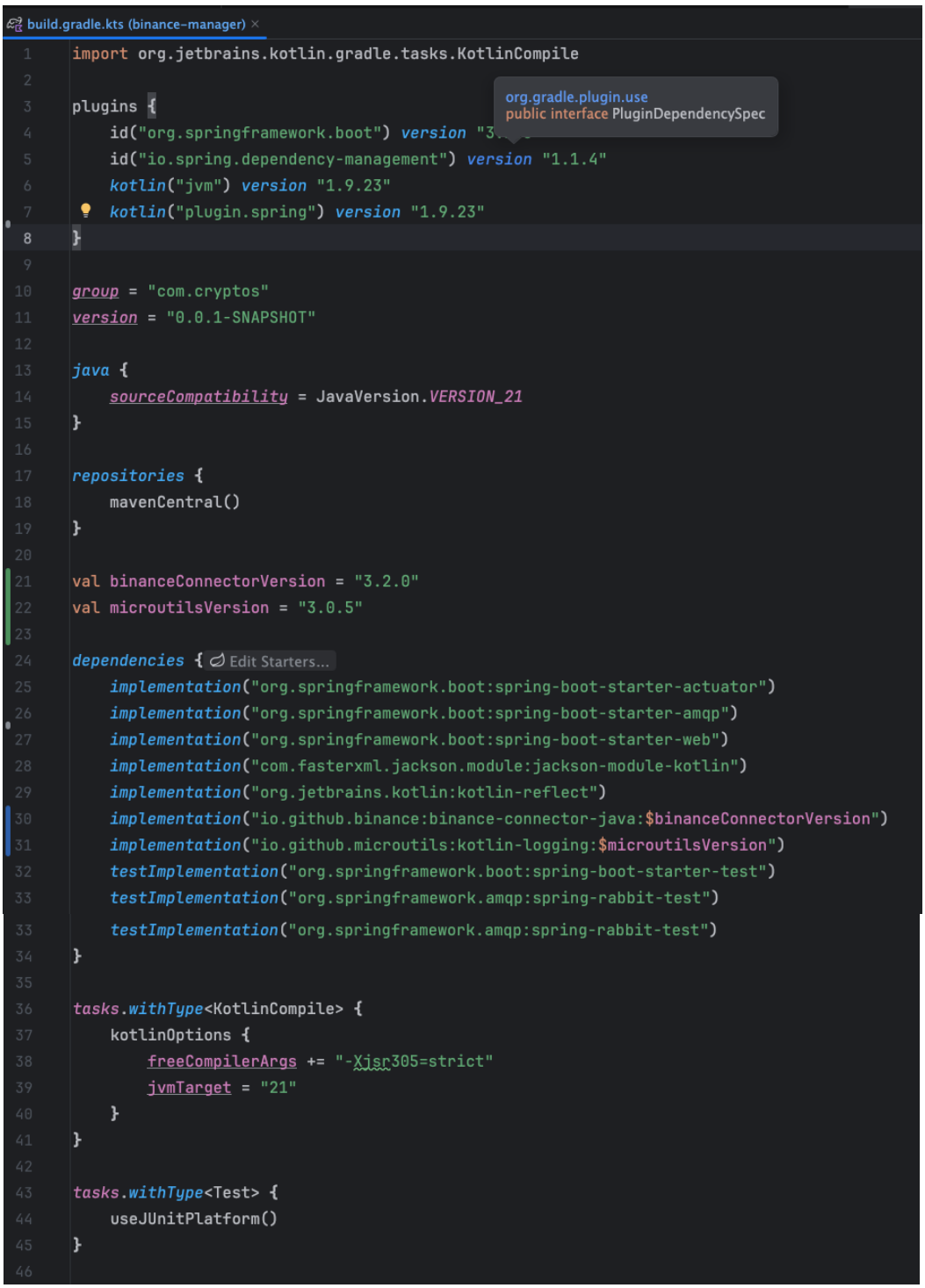


Рисунок 29 – Загальна структура build.gradle.kts файлу в Intelij IDEA (рисунок зроблений самостійно)

4.3 Створення конектора з стороннім публічним API

Для інтеграції з публічним API криптобіржі Binance був розроблений спеціалізований конектор. Цей компонент взаємодіє з API Binance для отримання

актуальних даних про котирування криптовалют, виконані торгові операції та стан рахунків користувачів. Конектор забезпечує стабільну та надійну інтеграцію з зовнішнім API, використовуючи сучасні підходи до управління взаємодією між системами (див. рис. 30).

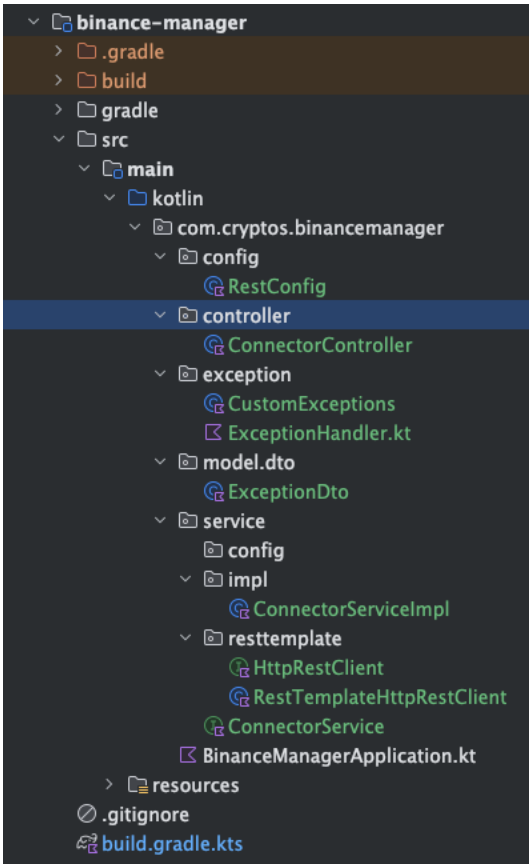


Рисунок 30 – Загальна структура binance-manager мікросевіса в Intelij IDEA (рисунок зроблений самостійно)

Реалізація конектора дозволяє системі автоматично оновлювати дані із зовнішнього ресурсу, що забезпечує користувачам актуальну інформацію та зручність в управлінні фінансовими активами на криптовалютному ринку.

Логування результатів виклику метода та відповідного ендпоінта в консолі (див. рис. 31):

Рисунок 31 – логування результатів роботи binance-manager мікросевіса в консолі Intelij IDEA (рисунок зроблений самостійно)

Приклад JSON результату виклику публічного API (див. рис. 32):

Рисунок 32 – JSON результат виклику get exchange rates ендпоінту binance-manager мікросевіса в консолі Intelij IDEA (рисунок зроблений самостійно)

UI частина мікросервісу реалізована наступним чином: створено HTML файл, index.html. У цьому файлі створено основну структуру, яка містить секцію для відображення інформації про обмін і кнопку, яка буде ініціювати отримання даних та викликати відповідний ендпоінт сервісної частини.

Після цього додано CSS стилі, щоб покращити вигляд користувацького інтерфейсу. Це робить інтерфейс приємнішим для користувачів.

Логіка виклику контролера написана в JavaScript коді, який використовує Fetch API для надсилання запиту до серверного ендпоінту. Отримані відображаються на веб-сторінці.

Приклад UI частини з відповідною кнопкою виклику ендпоінта (див. рис. 33).

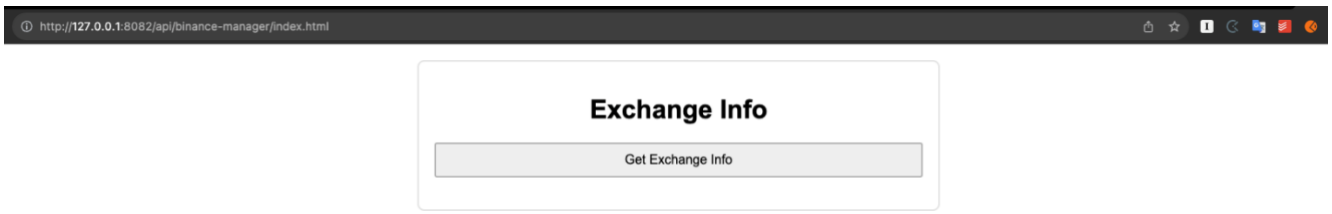


Рисунок 33 – UI частини з відповідною кнопкою виклику ендпоінта binance-manager мікросевіса в браузері (рисунок зроблений самостійно)

По події натиснення на відповідну кнопку відбувається виклик серверного ендпоінта та виводиться інформація в режимі поточного часу з публічного API Binance, далі наводиться приклад UI результатів запиту (див. рис. 34).

Рисунок 34 – UI частини з відповідною кнопкою виклику ендпоінта binance-manager мікросевіса в браузері (рисунок зроблений самостійно)

На підставі впровадження ПС можливо сформувати наступні рекомендації для подальшого покращення ПС.

Опишемо підвищення безпеки системи:

– шифрування даних: використання сучасних алгоритми шифрування для

захисту даних, як під час передачі, так і при зберіганні. Це включає використання HTTPS для всіх комунікацій та шифрування бази даних;

– аутентифікація та авторизація: впровадження багаторівневої системи аутентифікації (наприклад, двофакторну аутентифікацію) для підвищення безпеки облікових записів користувачів;
– регулярні аудити безпеки: проведення регулярних перевірок та аудиту безпеки для виявлення та усунення потенційних вразливостей.

Опишемо підтримку масштабованості:

– мікросервісна архітектура: розподіл системи на мікросервіси дозволяє легко масшт4абувати окремі компоненти системи в залежності від навантаження
– горизонтальне масштабування: використання технології контейнеризації, такі як Docker і Kubernetes, для автоматичного розгортання та масштабування системи;
– кешування: впровадження механізму кешування для зменшення навантаження на базу даних і підвищення продуктивності системи.

Опишемо покращення інтеграції з API Binance:

– регулярні оновлення: слідкування за оновленнями API Binance та своєчасне адаптування системи до нових версій;
– оптимізація запитів: оптимізація запитів до API Binance для зменшення затримок і підвищення ефективності обробки даних;
– моніторинг та логування: впровадження системи моніторингу та логування для відстеження роботи інтеграції з API та швидкого виявлення проблем.

Опишемо покращення користувацького інтерфейсу:

– зручність використання: розробка інтерфейсу з урахуванням принципів UX/UI дизайну, щоб забезпечити зручність і інтуїтивність використання системи;
– адаптивний дизайн: забезпечення адаптивності інтерфейсу для коректного відображення на різних пристроях (настільні ПК, планшети, смартфони);
– регулярне оновлення інтерфейсу: оновлення інтерфейсу з урахуванням

зворотного зв'язку від користувачів для підвищення їх задоволеності та ефективності роботи.

Опишемо забезпечення надійності та продуктивності:

– балансування навантаження: використання балансувальників навантаження для рівномірного розподілу запитів між сервісами;
– моніторинг продуктивності: впровадження системи моніторингу (наприклад, Prometheus, Grafana) для відстеження продуктивності та виявлення вузьких місць;
– резервне копіювання: налаштування регулярного резервного копіювання бази даних для запобігання втраті даних у випадку збоїв.

Опишемо безперервне тестування та вдосконалення:

– автоматизоване тестування: впровадження автоматизованого тестування (юніт-тести, інтеграційні тести) для забезпечення якості коду та швидкого виявлення помилок;
– CI/CD процеси: використання процесів безперервної інтеграції та безперервного розгортання (CI/CD) для швидкого та безпечного розгортання нових версій системи;
– аналіз та оптимізація: постійний аналіз роботи системи, збір зворотнього зв'язку від користувачів та оптимізація системи для покращення її продуктивності та функціональності.

Опишемо навчання користувачів та підтримку:

– документація: створення детальної документації для користувачів та розробників, що включає інструкції з налаштування, використання та вирішення можливих проблем;
– підтримка користувачів: забезпечення ефективної підтримки користувачів через різні канали (чат, електронна пошта, форум) для швидкого вирішення їх запитань та проблем;
– навчальні матеріали: розробка навчальних матеріалів (відеоуроків, вебінарів) для допомоги користувачам у освоєнні системи та максимізації її можливостей.

Ці рекомендації допоможуть забезпечити високий рівень безпеки, масштабованості, інтеграції та зручності використання системи для обліку фінансових операцій на криптобіржі, забезпечуючи таким чином її ефективність та задоволеність користувачів.

Як бачимо така мікросервісіна архітектура та його імплементація дозволяє маштабувати та змінювати основний бізнес функціонал додатку без зміни його оркемих елементів, які виконують поставлені завдання, що дозволяє швидко реагувати на зміну ринку та попит та завдання клієнта (користувача).

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування додатку

Тестування програмного забезпечення є ключовим етапом у забезпеченні його надійності та стабільності. Для тестування серверної частини нашого застосунку ми використовуємо комбінацію інструментів Mockito, Junit5, Swagger від OpenAPI та Postman. Mockito дозволяє створювати mock-об'єкти для моделювання поведінки залежностей, що спрощує процес ізоляції компонентів для тестування. Junit5 надає можливості для написання та виконання модульних тестів, що охоплюють різні аспекти бізнес-логіки додатку.

Процес тестування включає наступні кроки:

- модульне тестування: використовуючи Junit5, ми пишемо тести для кожного окремого мікросервісу, перевіряючи коректність виконання бізнес-логіки. За допомогою Mockito створюються mock-об'єкти для залежностей, що дозволяє перевірити окремі частини коду ізолюючи їх від інших компонентів системи;
- інтеграційне тестування: цей етап передбачає тестування взаємодії між різними мікросервісами та базою даних. Ми перевіряємо, чи коректно мікросервіси обмінюються повідомленнями через RabbitMQ та чи правильно виконуються запити до бази даних PostgreSQL;
- функціональне тестування: перевірка основних функцій системи з точки зору кінцевого користувача. Це включає тестування REST API за допомогою Swagger від OpenAPI, що дозволяє документувати і тестувати API-запити безпосередньо в браузері, сервіси Intelij IDEA, а також за допомогою Postman, який забезпечує зручний інтерфейс для тестування HTTP-запитів і дозволяє створювати колекції запитів для автоматизованого тестування.

Як приклад наведемо тестування запиту вбудованими сервісами Intelij IDEA(див. рис. 35).

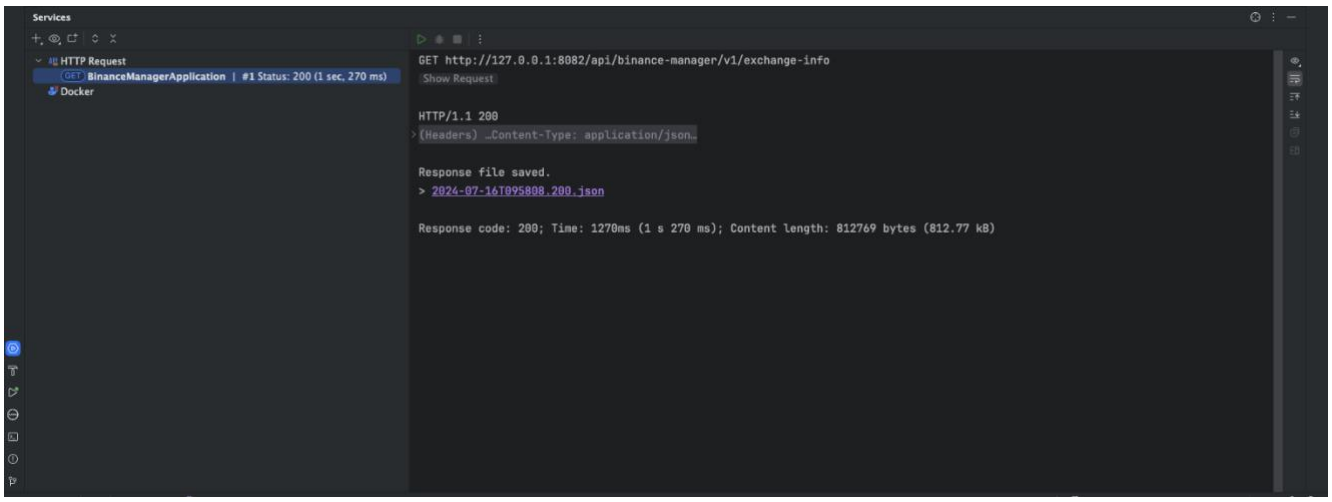


Рисунок 35 – UI тестування запиту вбудованими сервісами Intelij IDEA (рисунок зроблений самостійно)

Приклад перевірки запиту в Postman (див. рис. 36).

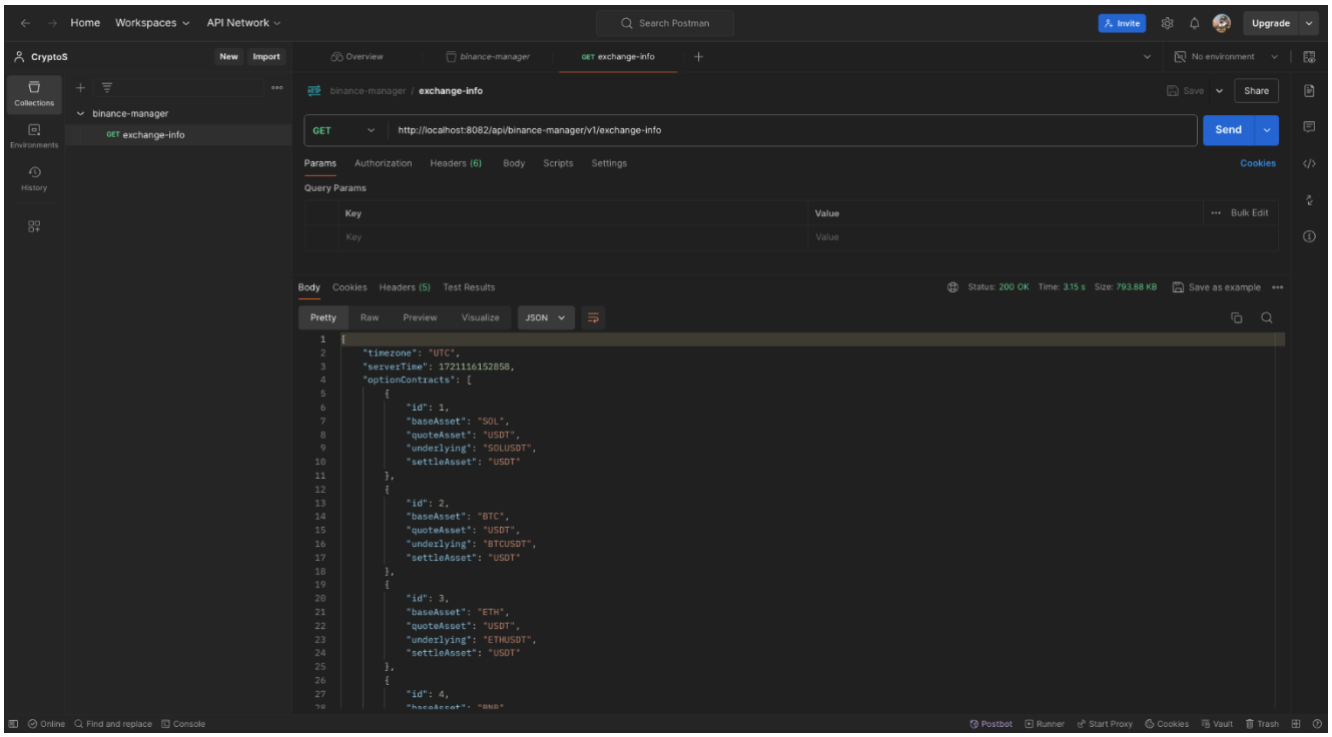


Рисунок 36 – UI тестування запиту Postman (рисунок зроблений самостійно)

- Навантажувальне тестування: Використовуючи Prometheus та Grafana, ми моніторимо продуктивність системи під час високих навантажень. Цей етап дозволяє виявити потенційні "вузькі місця" та забезпечити стабільну роботу системи навіть при великій кількості запитів.

5.2Приклади критичних помилок

Під час тестування програмного забезпечення можуть виникати різні критичні помилки, які необхідно виявити та усунути. Ось декілька прикладів таких помилок:

– помилка з'єднання з базою даних: якщо мікросервіси не можуть встановити з'єднання з PostgreSQL, це може призвести до збою всієї системи. За допомогою інтеграційного тестування ми перевіряємо коректність налаштувань з'єднання та обробку помилок при недоступності бази даних(див. рис. 37);

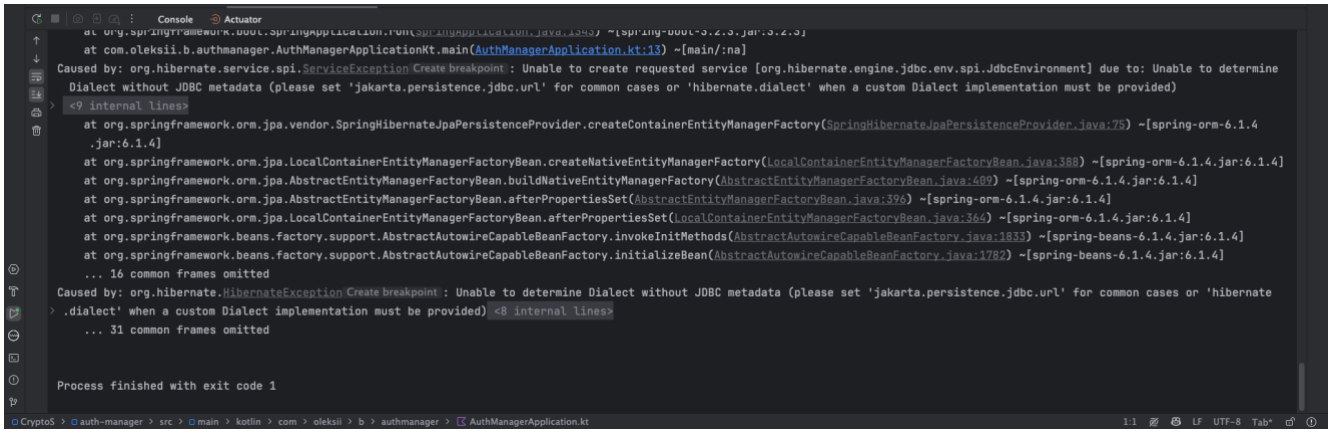


Рисунок 37 – Помилка з'єднання з базою даних в Intelij IDEA (рисунок зроблений самостійно)

– неправильна маршрутизація запитів: якщо NGINX неправильно розподіляє запити між мікросервісами, це може спричинити некоректну роботу додатку. Функціональне тестування REST API за допомогою Swagger та Postman допомагає виявити такі проблеми та перевірити правильність маршрутизації;
– неправильний маппінг данних (див. рис. 38).

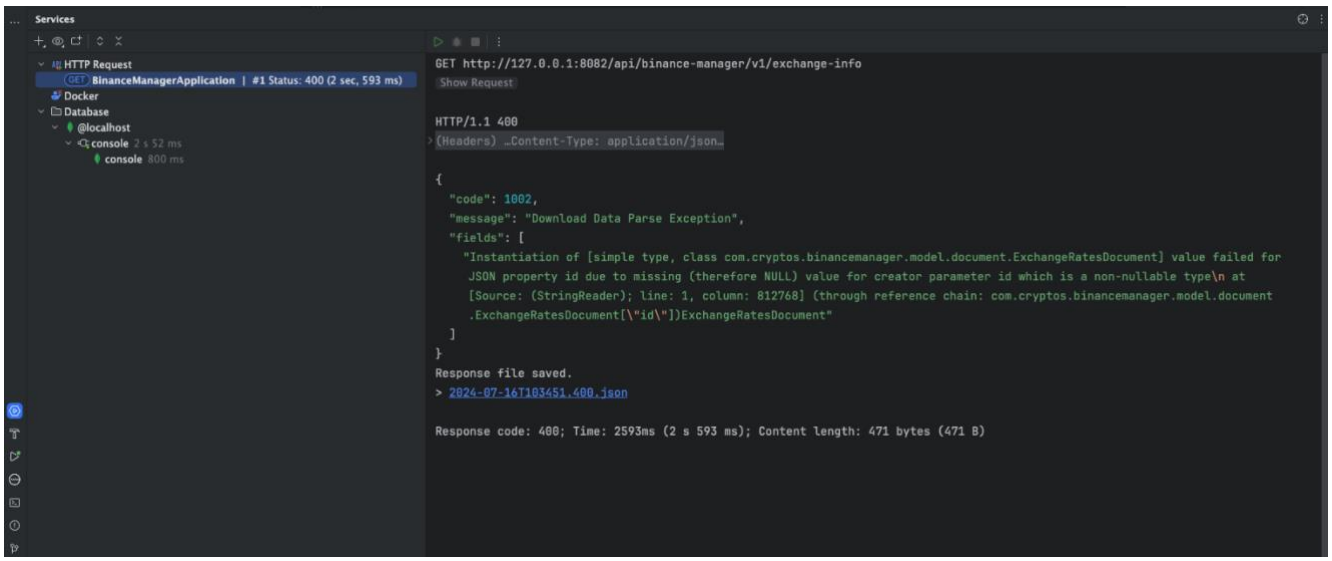


Рисунок 38 – Помилка неправильного маппінгу данних в Intelij IDEA (рисунок зроблений самостійно)

– збої у RabbitMQ: якщо виникають проблеми з передачею повідомлень між мікросервісами через RabbitMQ, це може призвести до втрати або дублювання даних. Інтеграційне тестування включає перевірку надійності передачі повідомлень та обробку помилок, пов'язаних з RabbitMQ (див. рис. 39);

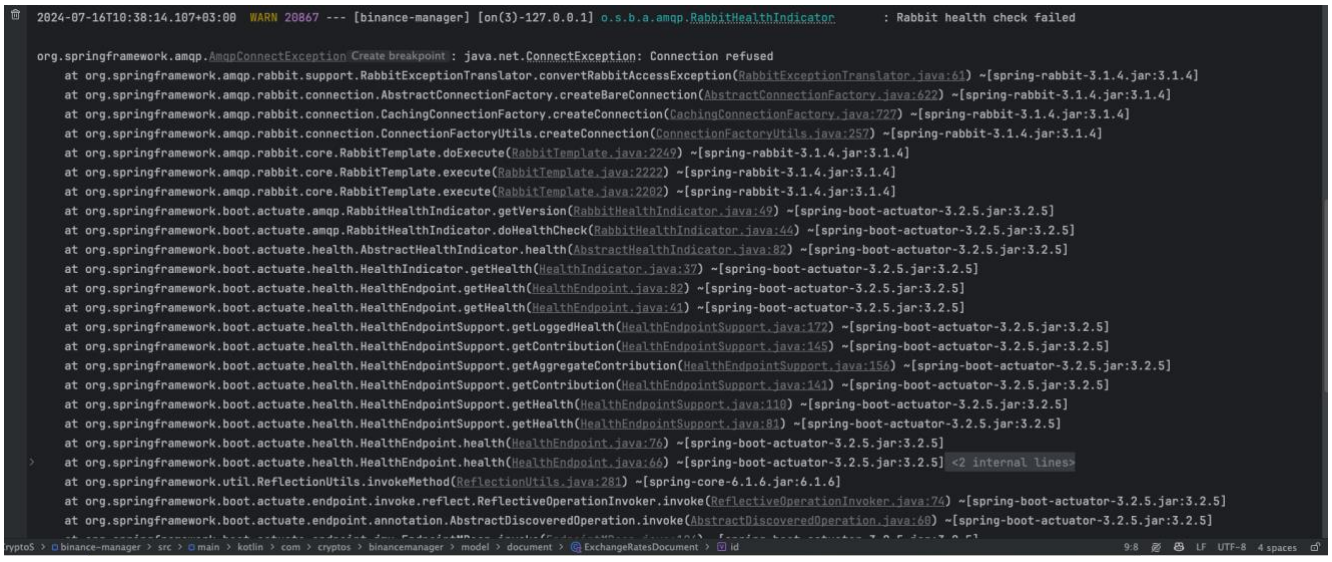


Рисунок 39 – Помилка проблеми з передачею повідомлень між мікросервісами через RabbitMQ в Intelij IDEA (рисунок зроблений самостійно)

- перевантаження системи: під час навантажувального тестування за допомогою Prometheus та Grafana ми виявляємо випадки, коли система не може обробити великий обсяг запитів. Це дозволяє нам оптимізувати продуктивність та налаштувати автоматичне масштабування;
- невідповідність даних: помилки у синхронізації даних між мікросервісами та базою даних можуть призвести до некоректного відображення інформації для користувачів. Модульне та інтеграційне тестування допомагають виявити та усунути такі помилки, забезпечуючи цілісність даних.

Застосування комплексного підходу до тестування з використанням Mockito, Junit5, Swagger від OpenAPI, Postman, Prometheus та Grafana дозволяє забезпечити високу якість і надійність програмного забезпечення, виявити та усунути критичні помилки на ранніх етапах розробки.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для успішного впровадження програмного забезпечення необхідно дотримуватися детально розробленого плану, що включає кілька етапів. Першим етапом є підготовка серверної інфраструктури, яка включає налаштування хмарного дроплету на платформі DigitalOcean. На цьому етапі необхідно створити хмарний сервер, налаштувати мережеві параметри, забезпечити безпеку доступу та встановити систему контейнеризації Docker.

Наступним кроком є налаштування проксі-сервера NGINX для ефективного розподілу навантаження та забезпечення безпечного з'єднання за допомогою HTTPS. Це включає конфігурацію NGINX для роботи в якості зворотного проксі та забезпечення SSL-сертифікатами для шифрування трафіку.

Після налаштування серверної інфраструктури необхідно розгорнути контейнери Docker з основними компонентами системи. Це включає:

- контейнери з мікросервісами, які реалізують бізнес-логіку додатку;
- контейнер з базою даних PostgreSQL для зберігання даних;
- контейнер з брокером повідомлень RabbitMQ для обміну повідомленнями між сервісами;
- контейнери з інструментами моніторингу Grafana та Prometheus для відстеження стану системи.

Далі, проводиться інтеграція з публічним API Binance для отримання даних про фінансові операції. Після налаштування та тестування всіх компонентів, система проходить етап тестування на надійність та продуктивність.

Завершальним етапом є підготовка документації та навчання користувачів, а також забезпечення технічної підтримки на період впровадження та експлуатації системи.

Хоча основний акцент цієї роботи робиться на серверну частину програмного забезпечення, важливо також розглянути можливість впровадження мобільного застосунку в PlayMarket. Це забезпечить зручний доступ користувачам до основних функцій системи через мобільні пристрої.

Впровадження мобільного додатку в PlayMarket передбачає декілька етапів:

- розробка мобільного застосунку: Використання відповідних фреймворків (наприклад, Flutter або React Native) для розробки мобільного клієнта, який буде взаємодіяти з серверною частиною через REST API;
- тестування мобільного застосунку: Перевірка функціональності, продуктивності та безпеки мобільного додатку на різних пристроях та операційних системах;
- підготовка до випуску: Створення облікового запису розробника в PlayMarket, підготовка опису додатку, скріншотів, іконок та іншої необхідної інформації;
- публікація мобільного застосунку: Завантаження додатку до PlayMarket, заповнення усіх необхідних форм та проходження процесу модерації;
- моніторинг та оновлення: Після успішного випуску, необхідно моніторити відгуки користувачів та оперативно випускати оновлення для покращення функціональності та виправлення можливих помилок.

Таким чином, впровадження програмного забезпечення включає не лише налаштування серверної інфраструктури та розгортання мікросервісів, але й створення зручного мобільного клієнта, доступного в PlayMarket, що забезпечує повний спектр можливостей для користувачів.

ВИСНОВКИ

Розробка системи для обліку фінансових операцій на криптобіржі є важливим кроком для забезпечення прозорості, безпеки та ефективності ринку криптовалют. Використання сучасних технологій та архітектурних підходів дозволяє створювати потужні та гнучкі рішення, які відповідають вимогам динамічного ринку криптовалют. Запропонована система на основі мікросервісної архітектури та інтеграції з API Binance дозволяє ефективно зберігати та обробляти дані про фінансові операції, забезпечуючи високий рівень продуктивності та безпеки.

У ході розробки було проведено детальний аналіз предметної галузі, виявлено основні проблеми та сформульовано чіткі вимоги до програмної системи. Це дозволило створити архітектуру, яка відповідає сучасним стандартам розробки та забезпечує необхідну функціональність.

Особлива увага була приділена проектуванню сховища даних та розробці алгоритмів обробки фінансової інформації. Це дозволило створити надійну основу для зберігання та аналізу великих обсягів даних, що є критично важливим для роботи з криптовалютними операціями.

Реалізація мікросервісної архітектури та інтеграція з API Binance демонструють здатність системи взаємодіяти з зовнішніми джерелами даних та забезпечувати масштабованість рішення. Це створює передумови для подальшого розвитку системи та її адаптації до змін на ринку криптовалют.

Проведене тестування підтвердило надійність та ефективність розробленого програмного забезпечення. Виявлені та виправлені критичні помилки свідчать про ретельний підхід до забезпечення якості продукту.

Загалом, розроблена система є важливим інструментом для учасників криптовалютного ринку, який допоможе підвищити ефективність управління фінансовими операціями та сприятиме розвитку прозорої та безпечної екосистеми криптовалют.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kotlin Programming Language. *Kotlin*. URL: https://kotlinlang.org (дата звернення: 14.06.2024).

2. Spring Boot. *Spring Boot*. URL: https://spring.io/projects/spring-boot (дата звернення: 14.06.2024).

3. RabbitMQ: One broker to queue them all | RabbitMQ. *RabbitMQ: One broker to queue them all | RabbitMQ*. URL: https://www.rabbitmq.com (дата звернення: 14.06.2024).

4. PostgreSQL. *PostgreSQL*. URL: https://www.postgresql.org (дата звернення: 14.06.2024).

5. Binance – криптовалютна біржа для Bitcoin, Ethereum та альткоїнів. *Binance*. URL: https://www.binance.com/uk-UA (дата звернення: 14.06.2024).

6. Academy B. Що таке криптовалюта? | Binance Academy. Binance Academy. URL: https://academy.binance.com/uk/articles/what-is-a-cryptocurrency (дата звернення: 01.07.2024).

7. Academy B. Що таке блокчейн і як він працює? | Binance Academy. Binance Academy. URL: https://academy.binance.com/uk/articles/what-is-blockchain-and-how-does-it-work (дата звернення: 01.07.2024).

8. Що таке криптовалютна біржа? Як працює біржа криптовалют | WhiteBIT.Blog. WhiteBIT Blog. URL: https://blog.whitebit.com/uk/what-is-a-cryptocurrency-exchange/ (дата звернення: 01.07.2024).

9. CoinMarketCap. 10 найкращих криптогаманців для початківців | CoinMarketCap. CoinMarketCap Academy. URL: https://coinmarketcap.com/academy/uk/article/10-best-crypto-hot-wallets-for-beginners (дата звернення: 01.07.2024).

10. Crypto Tracker Trusted by 1 Million People Worldwide. *Crypto Tracker Trusted by 1 Million People Worldwide*. URL: https://coinstats.app (дата звернення: 14.06.2024).

11. CoinLedger The #1 Free Crypto Tax Software. *CoinLedger The #1 Free Crypto Tax Software*. URL: https://coinledger.io (дата звернення: 14.06.2024).

12. Delta Investment Tracker | #1 Multi-Asset Portfolio Tracker. *Delta Investment Tracker | #1 Multi-Asset Portfolio Tracker*. URL: https://delta.app/en (дата звернення: 14.06.2024).

13. Cryptocurrency Prices, Charts, and Crypto Market Cap | CoinGecko. *CoinGecko*. URL: https://www.coingecko.com (дата звернення: 14.06.2024).

14. Master Your Net Worth with Kubera. *Master Your Net Worth with Kubera*. URL: https://www.kubera.com (дата звернення: 14.06.2024).

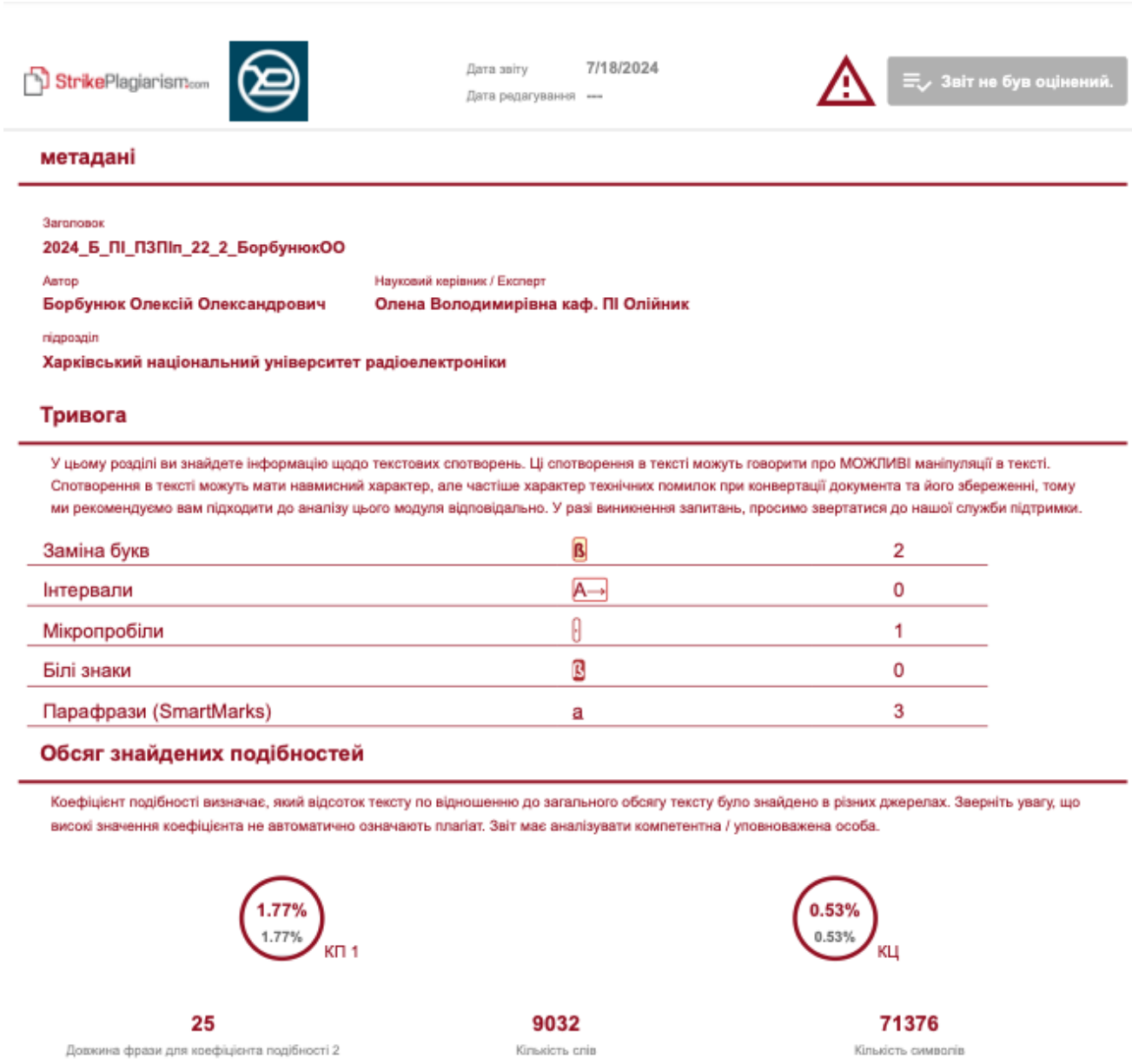
15. Coinbase. URL: *https://www.coinbase.com/* (дата звернення: 14.06.2024).

16. Buy, Sell & Trade Bitcoin & Other Crypto Currencies with Gemini's Platform | Gemini. *Gemini*. URL: https://www.gemini.com/eu (дата звернення: 14.06.2024).

17. Kraken - Learn about blockchain, crypto and NFTs Crypto shouldn't be cryptic. URL: *https://www.kraken.com* (дата звернення: 14.06.2024).

ДОДАТОК А

Звіт з результатів перевірки на плагіат



StrikePlagiarism.com

Дата звіту 7/18/2024
Дата редагування ---

Звіт не був оцінений.

метадані

Заголовок
2024_Б_ПІ_ПЗПІп_22_2_БорбунюкОО

Автор
Борбунюк Олексій Олександрович

Науковий керівник / Експерт
Олена Володимирівна каф. ПІ Олійник

підрозділ
Харківський національний університет радіоелектроніки

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	2
Інтервали	A→	0
Мікропробіли	¶	1
Білі знаки	ß	0
Парафрази (SmartMarks)	a	3

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.77% 1.77% КП 1

0.53% 0.53% КЦ

25
Довжина фрази для коефіцієнта подібності 2

9032
Кількість слів

71376
Кількість символів

ДОДАТОК Б

Слайди презентації

Кваліфікаційна робота бакалавра

Програмна система ведення обліку криптобіржевих фінансових операцій

Студент гр. ПЗПІп-22-2 Борбунюк О.О.
Керівник – доц. Русакова Н.Є.

1

Актуальність обраної теми

- Зростаюча роль криптовалютних бірж у сучасній фінансовій системі.
- Необхідність високопродуктивних, масштабованих та безпечних програмних рішень для криптобірж та відстеження трансакцій.
- Перспективність застосування мікросервісної архітектури для створення гнучких систем.
- Важливість інтеграції з публічними API (наприклад, Binance) для ефективної обробки даних.

2

Аналіз конкурентів та виявлення недоліків

Кращі практики:

- автоматизація відстеження транзакцій та інтеграція з максимальною кількістю криптобірж і гаманців;
- зручний та інтуїтивно зрозумілий інтерфейс користувача;
- підтримка обчислення податків з урахуванням місцевого законодавства;
- реалізація функцій сповіщень про зміну цін у реальному часі.

Основні недоліки:

- висока вартість підписки на преміум-функції;
- обмежена підтримка менш популярних криптовалют та бірж;
- проблеми з точністю та своєчасністю оновлення даних.

3

Постановка задачі

Мета роботи є розробка системи для обліку фінансових операцій на криптобіржі.

Завданнями цієї роботи є:

- створення системи для обліку фінансових операцій на криптобіржі,
- інтеграція з API Binance для отримання інформації про торгові операції та стан рахунків користувачів,
- дотримання патерну мікросервісної архітектури
- впровадження механізмів аналізу даних для забезпечення прийняття обґрунтованих інвестиційних рішень,
- реалізація моніторингу та журналювання фінансових операцій для забезпечення атомарності та консистентності дій користувачів.

4

Загальна діаграма активності

Програмна система має мати наступні можливі активності користувача:

а) доступ до додатку: користувач отримує доступ до програми.

б) реєстрація / вхід:

1) якщо користувач не зареєстрований, він може зареєструватися;

2) якщо зареєстрований, виконується вхід в систему;

в) отримання даних ринку: після входу система отримує актуальні ринкові дані;

г) головний екран: користувач потрапляє на головний екран, де доступні такі опції:

1) виконання транзакції;

2) вибір типу транзакції;

3) перевірка балансу в базі даних;

4) збереження транзакції в базі даних;

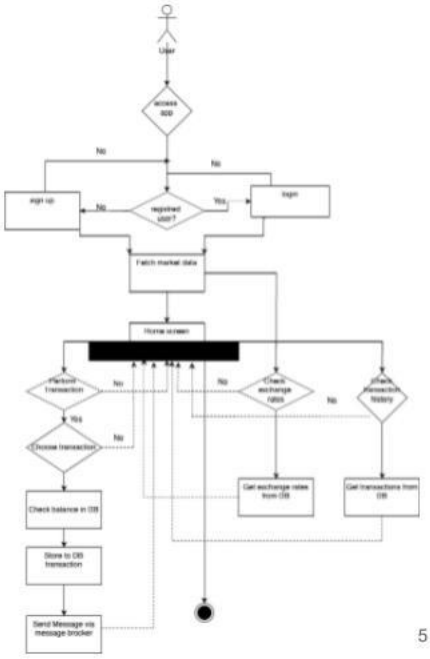
5) відправка повідомлення через брокер повідомлень;

д) перевірка курсів обміну: отримання курсів обміну з бази даних;

е) перевірка історії транзакцій;

ж) отримання транзакцій з бази даних;

з) завершення сесії: користувач може завершити роботу з системою в будь-який момент.



5

Діаграма розгортання

Серверна частина має багатошарову архітектуру та містить наступні шари:

а) шар проксі сервера та HTTPS клієнта у вигляді NGINX в Docker в окремій Docker м

б) шар моніторингу стану сервера та ПЗ за допомогою звязки Grafana та Promet контейнерах Docker в окремій Docker мережі;

в) шар програмного забезпечення розміщеного в контейнерах Docker в окремій мережі (сервіси і відповідні бази даних розміщені в окремих контейнерах, що забе незалежність та стресостійкість ПО для роботи за концепцією мікросе архітектури). Цей шар включає в себе:

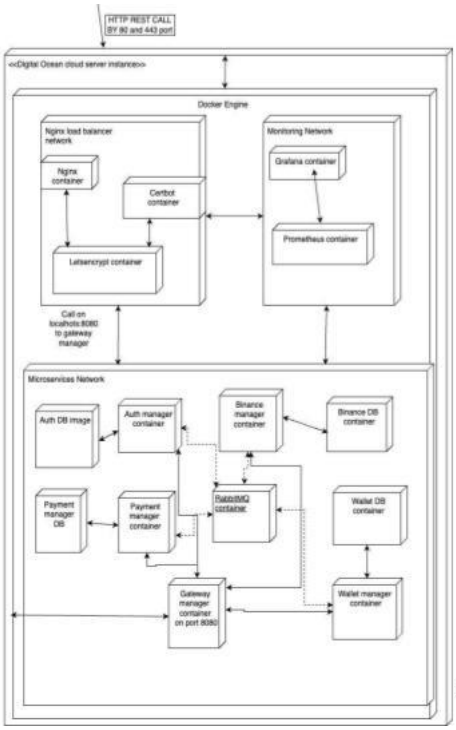
1) шар доступу до даних, із забезпеченням коректності доступу;

2) шар сервісів, що містить основну бізнес-логіку програмної системи;

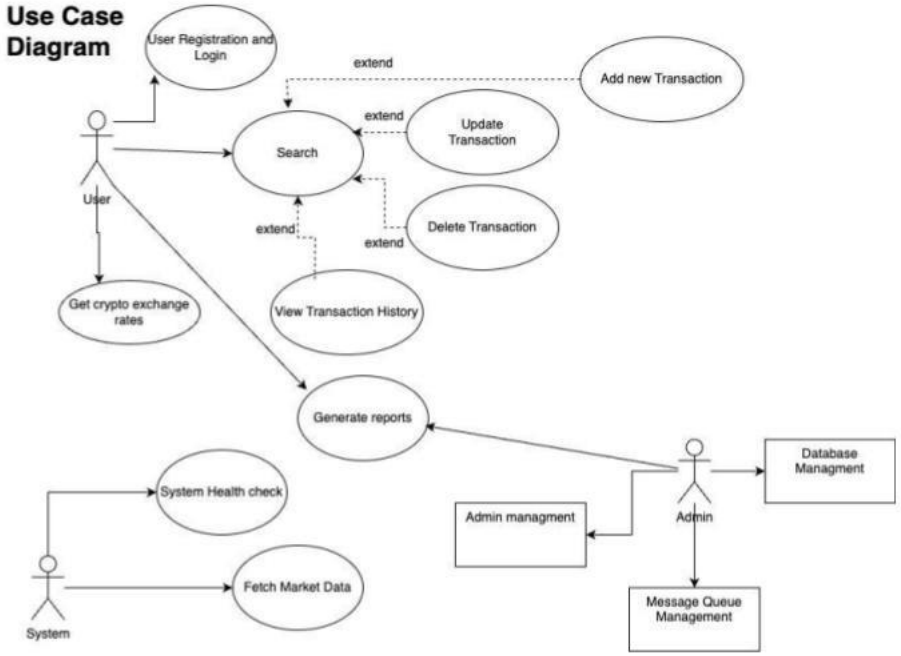
3) шар REST контролерів, за допомогою якого зовнішнім підсистемам над доступ до сервера;

4) шар брокера повідомлень RabbitMQ для обміну повідомленнями між сер всередині мережі.

Клієнт-серверний зв'язок проходить у форматі JSON за протоколом HTTPS та за допом брокера повідомлень RabbitMQ.



6



7

Повноваження користувачів

№	Повноваження	Опис
1	Зареєструватися в системі	Виконати реєстрацію у системі
2	Здійснити управління трансакціями	Пошук, додавання, редагування та видалення трансакцій
3	Здійснити управління користувачами	Додавання, редагування та видалення користувачів
4	Змінити мову інтерфейсу	Змінити мову інтерфейсу на англійську або українську
5	Керувати профілем у системі	Редагувати та видаляти профіль користувача
6	Отримати статистику	Отримання статистики дій користувача та статистичних показників на щодо окремої монети
7	Пошук по назві монет в бд	Пошук по назві монет в бд для додавання в портфоліо трансакції з монетою
8	Зробити резервну копію БД	Створення файлу з копією бд для відновлення
9	Здійснити управління сертифікатами	Додавання сертифікату та продовження його дії

8

Матриця ролей

	Роль 1 (Admin)	Роль 2 (User)	Роль 3 (System)
Повноваження 1	X	X	
Повноваження 2	X	X	
Повноваження 3	X	X	
Повноваження 4	X	X	
Повноваження 5	X	X	
Повноваження 6	X	X	X
Повноваження 7	X	X	
Повноваження 8	X		X
Повноваження 9	X		X

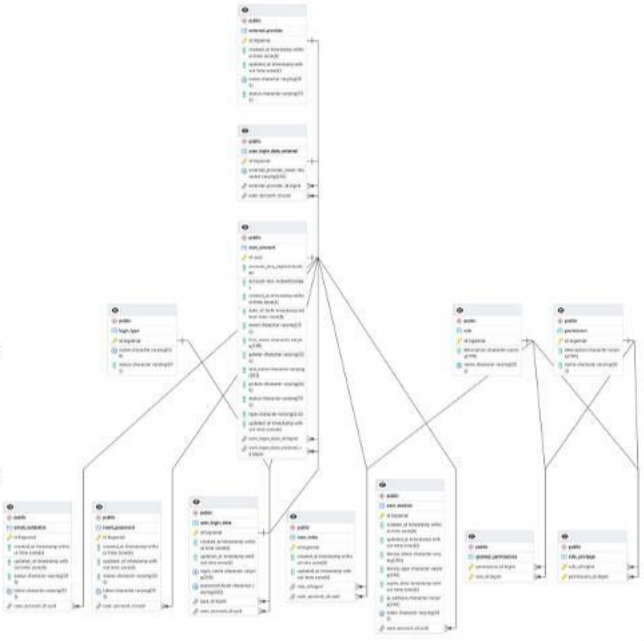
9

Приклад sql скрипту

Скрипт для створення таблиці user_login_data:

```
CREATE TABLE user_login_data (
       id bigserial PRIMARY KEY,
       created_at timestamp,
       updated_at timestamp,
       login_name varchar(255) UNIQUE,
       password_hash varchar(255) UNIQUE,
       type_id bigint REFERENCES login_type(id),
       user_account_id uuid REFERENCES user_account(ic
);
```

Звернемо увагу, що поле login_name
є унікальним полем для імені входу користувача,
а password_hash є унікальним полем для хешу парс
Крім того поле type_id є зовнішнім ключем
до login_type(id), а поле user_account_id
є зовнішнім ключем до user_account(id).



10

Засоби розробки

11

Функціонал додатка - приклад контролеру

```
@RestController("/connector")
class ConnectorController(
    private val connectorService:
ConnectorService
) {
    @GetMapping("/v1/exchange-info")
    fun getExchangeInfo() =
connectorService.getExchangeInfo()
}
```

12

Функціонал додатка - приклад сервісу

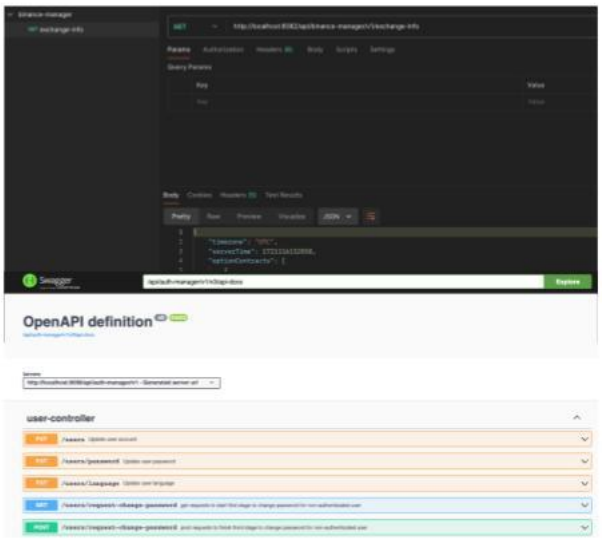
```
class ConnectorServiceImpl(

    override fun getExchangeInfo(): JsonNode {  new *
        logger.info { "Getting exchange info from Binance" }
        val response = restTemplate.executeRequest(
            headers = HttpHeaders(),
            url = "$binanceUrl/sapi/v1/exchangeInfo",
            httpMethod = HttpMethod.GET,
            requestBody = null,
            responseDtoClass = JsonNode::class.java
        )
        logger.info { "Exchange info received successfully from $binanceUrl" }
        logger.info( msg: "Saving exchange info to database")
        val ratesDocument = repository.save(ExchangeRatesDocument( id: null, response))
        logger.info( msg: "Exchange info saved successfully: ${ratesDocument.id}")
        return ratesDocument.body
    }
}
```

13

Тестування програмної системи

- проведено тестування програмними засобами Postman;
- додано документацію та можливість тестування програмним засобом Swagger UI;
- додано функціонал Mockito та JUnit5 для модульного тестування бізнес логіки додатку



14

Висновки

Розроблена система забезпечує ефективний облік та аналіз фінансових операцій на криптобіржі, підвищуючи прозорість та безпеку ринку.

Використання мікросервісної архітектури та інтеграції з API Binance дозволяє досягти високої масштабованості та гнучкості системи.

Ретельне тестування та виправлення критичних помилок гарантують надійність та стабільність роботи програмного забезпечення.

Система створює основу для подальшого розвитку та адаптації до динамічних змін на ринку криптовалют, сприяючи формуванню прозорої фінансової екосистеми.