

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Веб-додаток з обміну цифрових активів по принципу
SWAP»**

Виконав студент групи ІПЗ-20-3
ПАТЮК Артем Іванович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПАТЮК Артем Іванович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Вебдодаток з обміну цифрових активів по принципу SWAP»**,
затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74/с**

Вихідні дані для роботи: **використання сучасних технологій для
проектування, розробки та реалізації вебдодатку з обміну цифрових
активів по принципу SWAP**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	27.02.2024р.	27.02.2024р.
2	Морозов А.В.	27.02.2024р.	27.02.2024р.
3	Морозов А.В.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку з обміну цифрових активів по принципу SWAP та пояснювальної записки. Пояснювальна записка до випускної роботи містить 64 сторінок, 42 ілюстрацій та 5 таблиць.

Метою роботи є розробка веб-додатку з обміну цифрових активів, який призначений для забезпечення можливості користувачам безпечно та зручно обмінювати свої цифрові активи без необхідності зберігання їх на платформі.

У роботі визначено основні завдання на розробку системи, проведено аналіз вимог та функціональності додатку, також був проведений аналіз аналогів створеного програмного продукту. Було здійснено проектування бази даних та обрано набір сучасних технологій розробки, використання яких забезпечує високу продуктивність додатку.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, КРИПТОВАЛЮТА, КРИПТОБІРЖА, ОБМІН КРИПТОВАЛЮТ, NODE.JS, REACT.JS, REDUX.

					ІПЗ.КР.Б-121-24-ІПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Патюк А.І.			Вебдодаток з обміну цифрових активів по принципу SWAP	Літ.	Арк.	Аркушів
Керівник		Морозов А.В.					3	
Керівник						Житомирська політехніка, група ІПЗ-20-3		
Н. контр.		Тичина О.П.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of a web application for the exchange of digital assets according to the SWAP principle and an explanatory note. The explanatory note to the final work contains 64 pages, 42 illustrations and 5 tables.

The aim of the work is to develop a web application for the exchange of digital assets, which is designed to enable users to securely and conveniently exchange their digital assets without the need to store them on the platform.

The paper defines the main tasks for developing the system, analyzed the requirements and functionality of the application, and also analyzed the analogues of the created software product. The database was designed and a set of modern development technologies was chosen, the use of which ensures high application performance.

KEY WORDS: WEB APPLICATION, CRYPTOCURRENCY, CRYPTO EXCHANGE, CRYPTOCURRENCY EXCHANGE, NODE.JS, REACT.JS, REDUX.

ЗМІСТ

РЕФЕРАТ	3
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ВЕБДОДАТКУ З ОБМІНУ ЦИФРОВИХ АКТИВІВ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	16
Висновки до першого розділу.....	18
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБДОДАТКУ З ОБМІНУ ЦИФРОВИХ АКТИВІВ ПО ПРИНЦИПУ SWAP	19
2.1 Визначення варіантів використання та структури системи	19
2.2 Розробка бази даних системи	24
2.3 Проектування та реалізація алгоритмів роботи системи	26
2.4 Реалізація вебдодатку з обміну цифрових активів	27
Висновки до другого розділу	37
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ДОДАТКУ З ОБМІНУ ЦИФРОВИХ АКТИВІВ ПО ПРИНЦИПУ SWAP	39
3.1 Порядок встановлення та налаштування параметрів системи.....	39
3.2 Структура інтерфейсу веб-додатку та порядок роботи	40
3.3 Тестування роботи вебдодатку	50
Висновки до третього розділу	51
ВИСНОВКИ.....	53
СПИСОК ЛІТЕРАТУРИ.....	54
ДОДАТКИ.....	57

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

WEBSOCKET – протокол зв'язку між браузером та сервером

БД – база даних

СУБД – система управління базами даних

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						6

ВСТУП

Актуальність теми. Світ стрімко розвивається, і разом з ним еволюціонують й фінансові системи. На зміну традиційним валютам, що контролюються банківськими установами, приходять нові, децентралізовані альтернативи – криптовалюти. За останнє десятиліття криптовалюта стала суттєвим фактором у глобальній економіці, впливаючи на розвиток суспільства.

Виникнення та прогрес біткоїна та інших активів сприяли народженню альтернативної фінансової системи, яка не підпорядкована централізованим органам та державним регуляторам. Розвиток криптоплатформ, бірж та веб-сайтів в останні роки відображає стрімке зростання інтересу до криптовалют та блокчейн-технологій. Ці платформи стають все більш важливими для торгівлі, інвестування та обміну цифровими активами, а також для надання різноманітних фінансових послуг, від управління портфелем до позик та страхування.

Метою кваліфікаційної роботи є розробка та проектування веб-додатку з обміну цифрових активів, який призначений для забезпечення можливості користувачам безпечно та зручно обмінювати свої цифрові активи без необхідності зберігання їх на платформі.

Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз вимог та функціональності додатку, а також аналіз аналогів існуючих рішень.
2. Визначити архітектуру системи та спроектувати базу даних для зберігання даних.
3. Розробити вебдодаток з обміну цифрових активів.

Об'єктом дослідження є проектування та реалізація веб-додатку з обміну цифрових активів по принципу SWAP з використанням програмного інтерфейсу криптобіржі для здійснення безпечних транзакцій.

Предметом дослідження є використання сучасних технологій та інтеграції децентралізованих платформ для здійснення автоматизованих переводів активів.

Розроблений програмний продукт можна використовувати як сайт підприємств або компаній що займаються обміном цифрових активів. Окрім поточних можливостей, веб-додаток має значний потенціал для подальшого розвитку та розширення функціоналу, що дозволить йому задовольняти зростаючі потреби ринку.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ВЕБДОДАТКУ З ОБМІНУ ЦИФРОВИХ АКТИВІВ

1.1 Постановка задачі

Зараз кожен, хто має криптовалюту, може відчувати потребу у зручному та безпечному способі обміну її на інші види криптовалют. З цією метою створення веб-додатку для обміну цифровими активами по принципу SWAP стає ключовим завданням. Враховуючи швидке розширення ринку цифрових активів та зростаючий інтерес до них, такий додаток може стати важливим інструментом для криптоінвесторів та користувачів які володіють криптовалютою.

Основною метою цього додатку є надання користувачам можливості обмінювати свої цифрові активи один на одного за принципом SWAP без необхідності зберігання їх на платформі. У цій кваліфікаційній роботі ми ставимо за мету розробку зручного та безпечного веб-додатку, який надасть користувачам можливість обмінювати свої криптовалютні активи швидко та ефективно.

Для досягнення цієї мети необхідно розробити наступний функціонал та основні етапи:

1. Аналіз вимог та функціональності додатку;
2. Проектування та створення архітектури програмного продукту;
 - а). Розробка серверної та клієнтської частин додатку з врахуванням безпеки та швидкодії;
 - б). Застосування сучасних технологій та підходів для забезпечення масштабованості та надійності системи;
3. Проектування дизайну додатку;
 - а). Розробка інтуїтивно зрозумілого та привабливого інтерфейсу;
 - б). Дизайн, що адаптується під різні розміри екранів для забезпечення зручного користування на будь-яких пристроях;

4. Створення компонентів, реалізація сторінок та функціоналу додатку;
 - a). Реалізація функціоналу реєстрації та авторизації користувачів;
 - b). Розробка інтерфейсу вибору цифрових активів та введення необхідних даних для обміну;
 - c). Інтеграція системи сповіщень для інформування користувачів про статус їхніх обмінів;
 - d). Реалізація інтерфейсу адміністратора для перегляду та підтвердження обмінів;
 - e). Розробка системи забезпечення безпеки та конфіденційності даних користувачів;

5. Тестування програмного продукту

Результатом реалізації поставленого завдання є вебдодаток з обміну цифровими активами за принципом SWAP, який надає зручний та безпечний механізм обміну криптовалют. Додаток надає інформацію користувачам про доступні активи, їх обмінний курс та можливість створити заявку з вибраними активами для обміну, а також надає можливість здійснювати перекази адміністратором безпосередньо в самому додатку.

1.2 Аналіз аналогів програмного продукту

В ході дослідження ринку онлайн-платформ для обміну цифрових активів було проведено ретельний аналіз аналогів програмного продукту та визначено їх недоліки та переваги. Користувачі завжди відчували потребу в зручному та багатофункціональному сервісі для обміну цифрових активів. Ця платформа має поєднувати в собі зручність, надійність, а також швидкий та безпечний обмін.

Одним з таких додатків, який є аналогом розробленого додатку, є «changelly.com»[5]. Даний додаток є відомою платформою для обміну

криптовалют, яка відзначається своєю простотою та швидкістю процесу обміну (рис. 1.1).

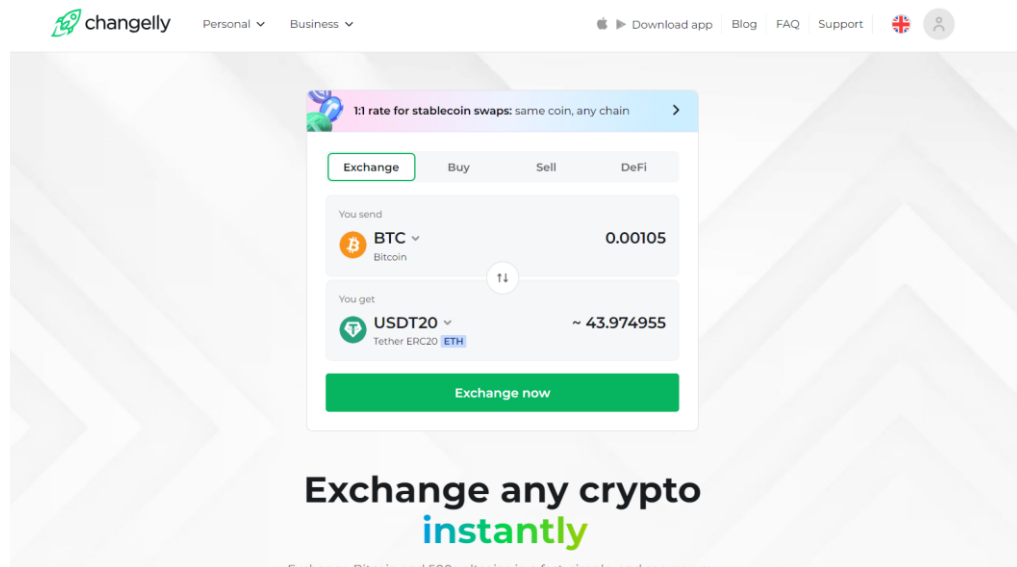


Рисунок 1.1 – Головна сторінка у веб-додатку «changelly.com».

Однією з його головних переваг є широкий вибір криптовалют для обміну, що робить його досить затребуваним для тих, хто володіє різними альткоїнами (рис. 1.2). До того ж, додаток дозволяє обмінювати свої активи без реєстрації, що дозволяє економити час для користувачів.

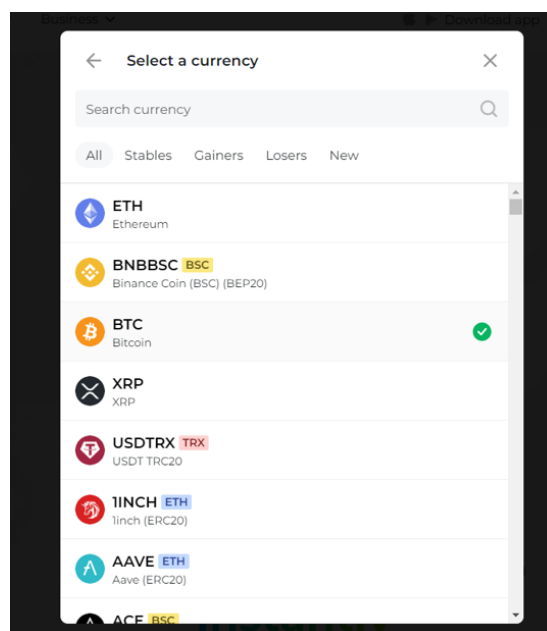


Рисунок 1.2 – Список доступних криптовалют для обміну у веб-додатку «changelly.com»

Сайт містить багато різної інформації, починаючи від актуальних курсів криптовалют, закінчуючи коротким посібником як обміняти криптовалюту

Є можливість отримати актуальні курси обміну для кожної валюти, що допомагає зробити правильне фінансове рішення. Сайт надає користувачам можливість вибрати зручний спосіб оплати з врахуванням комісій, а також повну і детальну інструкцію як правильно заповнити заявку та як перевести і отримати кошти. Серед недоліків хочу додати те, що на сайті відсутні багато популярних криповалютних пар, що робить його менш зацікавленим серед інших додатків з обміну цифрових активів.

Додаток має досить простий та зрозумілий інтерфейс, що дає можливість легко користуватись сайтом. Однак, слід зауважити, що інтерфейс може здатися трохи застарілим порівняно з сучасними обмінниками. Також додаток має зручні графіки з цінами на популярні криптовалюти (рис. 1.5).

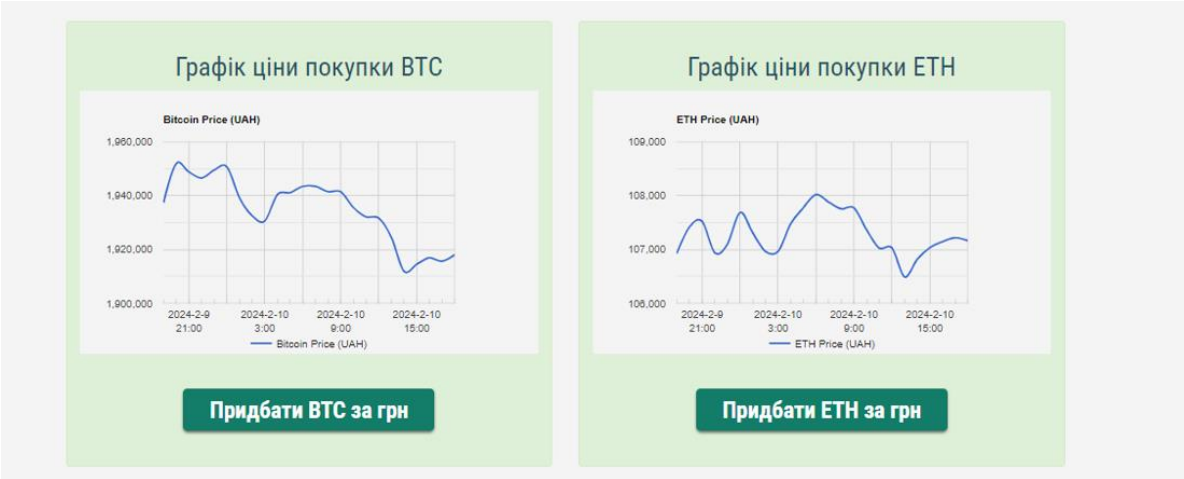


Рисунок 1.5 – Графіки актуальних цін у веб-додатку «bitcoin24.com.ua»

Останнім аналогом є український веб-додаток «100btc.kiev.ua»[7], який також призначений для обміну цифрових активів (рис. 1.6). Основним його недоліком є те, що він дозволяє обмінювати тільки фіатні гроші на криптовалюту, обмін криптовалют між собою не передбачено.

Валютна підтримка	150+ криптовалюта	BTC, ETH, USDT	50+ криптовалюта
Необхідність реєстрації	Необхідна для обміну на суму понад 150 EUR	Не потрібна	Не потрібна
Фіатні валюти	Відсутні	UAH	UAH, USD, EUR
Інструкція для користування	Присутня	Присутня	Присутня
Швидкість обміну	5-40 хвилин	5-30 хвилин	5-120 хвилин

Аналізуючи (табл. 1.1), можна зробити висновок, що кожен з цих сервісів вирішує лише певний аспект завдань, які були поставлені перед розробленим додатком. Додаток «changelly.com» має велику кількість криптовалют для обміну, а також обмін за принципом SWAP. «bitcoin24.com.ua» є гарним додатком для обміну криптовалюти на фіатну валюту, також він має досить простий і зрозумілий інтерфейс, але в ньому відсутня можливість обмінювати пари активів. Додаток «100btc.kiev.ua», навпаки, надає можливість для обміну різних криптовалют між собою, а також отримання і поповнення фіатними активами з використанням карток українських та зарубіжних банків.

Згідно з результатів порівняння аналогів, було виявлено всі переваги та недоліки задля того, щоб правильно спроектувати веб-додаток з обміну цифрових активів по принципу SWAP.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для розробки програмного продукту було вирішено використовувати клієнт-серверний підхід розробки. Даний підхід дозволяє розділити клієнтську (фронтенд) та серверну (бекенд) частину на окремі компоненти, що дозволяє спростити логіку роботи додатку, а також легко та зручно виконувати тестування.

Для розробки серверної сторони програмного продукту обрано середовище виконання Node.js[1] у поєднанні з фреймворком Express.js.[2] Клієнтську частину було вирішено розробляти за допомогою бібліотеки React[3] у поєднанні з TypeScript.

Node.js – середовище виконання JavaScript[4], яке дозволяє виконувати код поза браузером. Завдяки використанню неблокуючої моделі вводу-виводу(Event Loop) дозволяє обробляти велику кількість запитів одночасно без затримок.

Express.js - фреймворк для Node.js, що надає набір функцій для створення веб- і мобільних додатків. Express.js заснований на стандартних API Node.js, що робить його простим у вивченні та використанні.

Переваги використання Node.js з Express:

Швидкість. Використання двигуна V8 забезпечує високу швидкість обробки та виконання JavaScript коду.

Масштабованість. Завдяки своїй асинхронній природі та можливості створення кластерів Node.js легко масштабується під великі навантаження.

Розширюваність та функціональність. Express надає широкий вибір функціональності та велику кількість допоміжних HTTP-методів та проміжних обробників, що дає змогу легко створювати надійні API.

React - бібліотека для створення користувацьких інтерфейсів, яка дозволяє створювати багаторазово використовувані компоненти. Використання TypeScript у поєднанні з React забезпечує більш надійний та підтримуваний код завдяки статичній типізації.

Переваги React:

Компонентний підхід. Бібліотека надає можливість створювати багаторазово використовувані компоненти, що дозволяє спростити розробку.

Продуктивність. Завдяки віртуальному DOM, React забезпечує високу швидкість оновлення інтерфейсу.

Статична типізація. TypeScript додає типізацію до JavaScript коду, що дозволяє виявляти помилки на етапі компіляції, підвищуючи надійність коду.

Як сховище даних було вирішено використовувати базу даних Sqlite[8] з використанням ORM (Object-Relational Mapping) системи Prisma[9].

SQLite – компактна вбудована реляційна база даних з відкритим кодом. Вона ідеально підходить для невеликих додатків, оскільки не вимагає складної конфігурації.

Переваги SQLite:

Простота та вбудованість. SQLite не є окремим сервером баз даних, як, наприклад, MySQL або PostgreSQL, натомість її можна вбудувати безпосередньо в додаток, що робить її ідеальним вибором для невеликих додатків.

Швидкість роботи. Вона забезпечує високу швидкість роботи з даними завдяки своїй простоті структури, це особливо важливо коли швидкість доступу до даних є критично важливою

Зручне управління даними. Всі дані зберігаються в одному файлі бази даних, що робить процес резервного копіювання, перенесення та розгортання дуже простим та зручним.

Безсерверна архітектура. Однією з особливостей SQLite є відсутність необхідності в окремому сервері бази даних, це дає змогу використовувати дані як на клієнтській, так і на серверній стороні, спрощуючи синхронізацію даних між ними.

Для отримання даних про криптовалютні ринки, ціни торгових пар та автоматичної відправки активів було вирішено інтегрувати роботу з

платформою Binance, використовуючи свій обліковий запис та їх публічний набір методів для взаємодії з платформою **Binance API** [10].

RTK Query [12] - це потужний інструмент отримання та кешування даних. Його розроблено, щоб спростити звичайні випадки завантаження даних у вебдодатку.

Переваги RTK Query:

Автоматичне кешування. RTK Query автоматично кешує дані, отримані з API, це дозволяє зменшити кількість запитів до сервера та позитивно впливає на продуктивність роботи додатку.

Інтеграція з Redux. Бібліотека може синхронізувати кешовані дані з Redux store, забезпечуючи узгодженість даних у всьому додатку.

Висновки до першого розділу

У першому розділі було проведено аналіз предметної області та вимог до розробки програмного забезпечення. В результаті було сформовано основні завдання для створення вебдодатку, визначивши ключові функціональні можливості та етапи їх реалізації.

Згідно з результатів порівняння аналогів, було виявлено всі переваги та недоліки задля того, щоб правильно спроектувати веб-додаток, а також функції та особливості які необхідно включити у власному додатку, щоб він задовольняв потреби користувачів.

Результати дослідження інструментальних засобів допомогли сформувати набір технологій, які будуть використовуватись для розробки додатку. Обраний стек технологій ідеально підходить для досягнення поставлених завдань.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБДОДАТКУ З ОБМІНУ ЦИФРОВИХ АКТИВІВ ПО ПРИНЦИПУ SWAP

2.1 Визначення варіантів використання та структури системи

Для забезпечення неперервного виконання необхідних функцій і відповідності встановленим стандартам та специфікаціям, наведено високорівневі вимоги, які повинен задовольняти вебдодаток з обміну цифрових активів по принципу SWAP. Ці вимоги допомагають забезпечити безперебійну роботу системи та виконання всіх необхідних стандартів і специфікацій.

Бізнес вимоги

1. Основні цілі: проект створюється для забезпечення швидкого та безпечного обміну активів та автоматизації цього процесу.
2. Представлення проекту: проект буде реалізовано у вигляді вебдодатку з інтуїтивно зрозумілим інтерфейсом.

Вимоги користувачів

1. Можливість переглядати список доступних цифрових активів для обміну.
2. Можливість вибрати потрібну пару активів та ознайомитись з її поточний курсом.
3. Можливість створити заявку на обмін.
4. Можливість переглядати список створених заявок та їх статус.
5. Можливість створювати відгуки та переглядати відгуки інших користувачів.

Вимоги персоналу вебдодатку

1. Можливість додавання активів з автоматичним отриманням їх адреси.
2. Можливість встановлювати комісії.
3. Можливість відстежувати та управляти поточними обмінами.

Характеристика об'єкта комп'ютеризації

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						19

Користувачі системи можуть зручно та швидко створювати заявки на обмін з вибраними цифровими активами, переглядати доступні обмінні курси та іншу інформацію за допомогою інтуїтивного та зручного інтерфейсу.

Функціональні вимоги

1. Авторизація у системі: система повинна надавати можливість реєстрації та авторизації користувачів.
2. Обмін цифрових активів: користувач повинен мати можливість вибирати потрібні активи, отримати їх поточний курс та можливість вказати потрібну інформацію для створення заявки.
3. Управління обмінами: адміністратор повинен мати можливість переглядати та управляти поточними обмінами.

Нефункціональні вимоги

1. Сприйняття

- час, необхідний для освоєння звичайним користувачам – 20 хвилин, а для освоєння просунутим користувачам – 5 хвилин;
- час відгуку для типових задач – не більше 5 секунд, для складних завдань – не більше 10 секунд.

2. Надійність

- доступність – час, що витрачається на обслуговування системи не повинно перевищувати 5% від загального часу роботи.
- середній час безвідмовної роботи – 30 робочих днів.
- максимальна норма помилок та дефектів в роботі системи – 1 помилка на десять тисяч рядків коду

3. Продуктивність

- система повинна підтримувати мінімум 30 одночасно працюючих користувачів, пов'язаних із загальною базою даних.

Аналіз вимог користувачів дав підстави для визначення варіантів використання додатку обміну цифрових активів. На (рис. 2.1) наведено діаграму використання програмної системи.

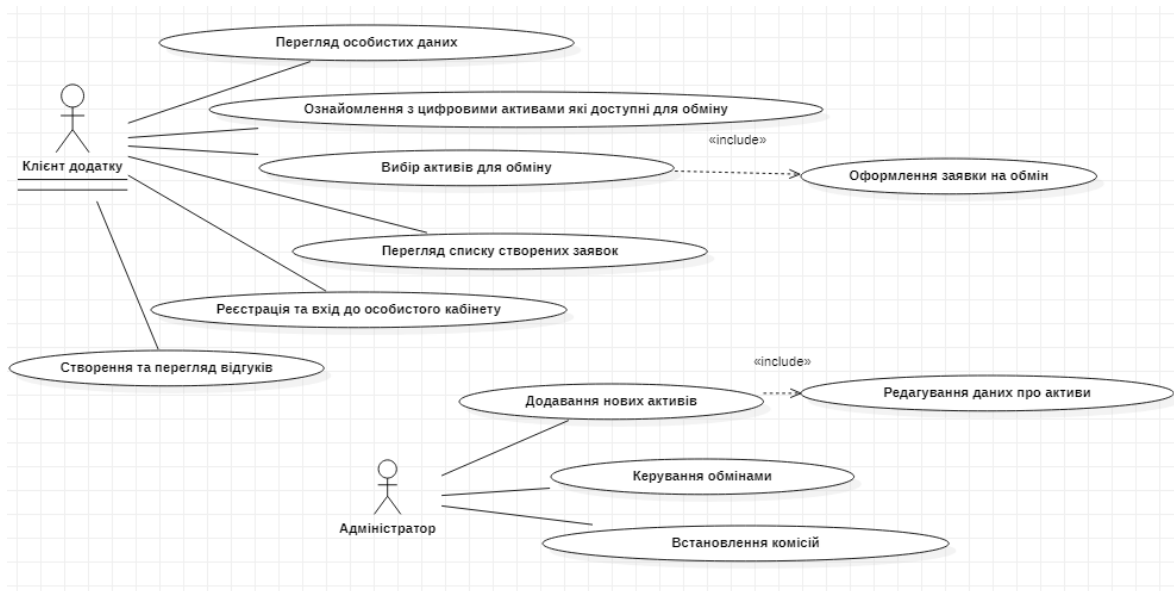


Рис. 2.1 Діаграма варіантів використання

Ретельний аналіз функціональних вимог допоміг визначити основні сутності, необхідні для створення додатку. У результаті було створено діаграму сутностей (рис. 2.2), яка відображає моделі даних, діаграму компонентів системи (рис. 2.3), на якій було визначено структуру та логіку взаємодії компонентів системи

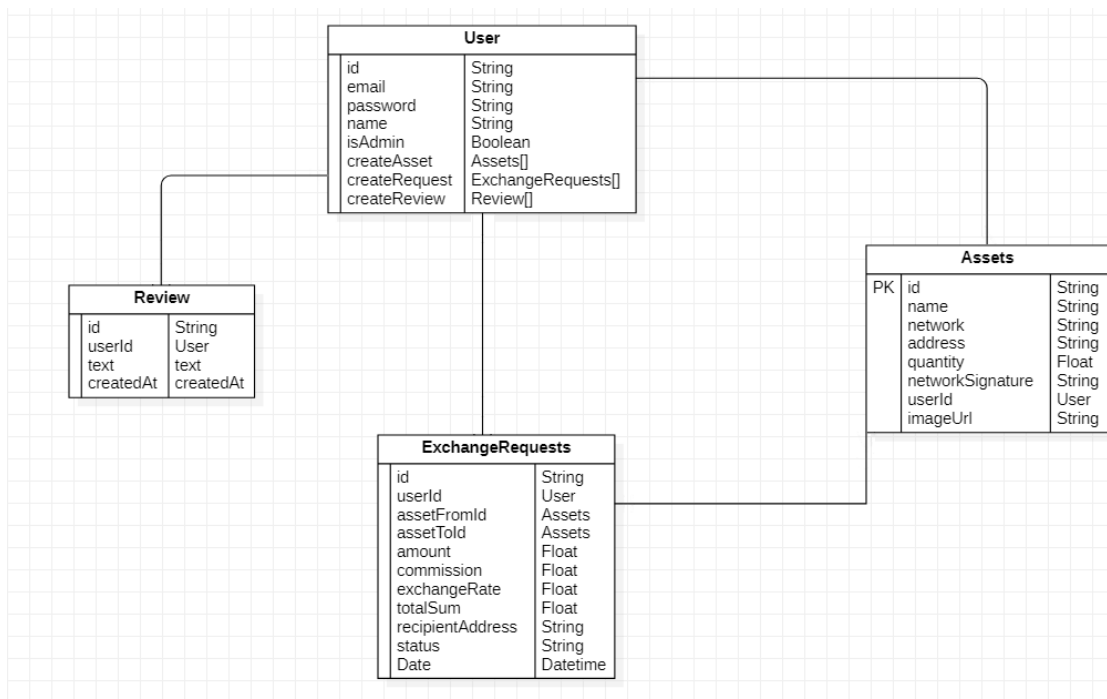


Рис. 2.2 Діаграма сутностей вебдодатку з обміну цифрових активів по принципу SWAP

- **Controllers:** Вони обробляють вхідні HTTP-запити і надсилають відповіді. Кожен контролер відповідає певному маршруту.
- **Routes:** Визначають кінцеві точки API. Кожен маршрут пов'язаний з контролером, який обробляє запити до цього маршруту.
- **Middlewares:** Це функції, які мають доступ до об'єктів запиту і відповіді, а також до наступної функції проміжного програмного забезпечення в циклі запиту-відповіді програми. Вони можуть виконувати будь-який код, вносити зміни в об'єкти запиту та відповіді, завершувати цикл запиту-відповіді та викликати наступну функцію проміжного програмного забезпечення в стеку.
- **Error handling:** Це місце, де обробляються помилки, що виникають у додатку.

Persistence – це компонент системи, що відповідає за зберігання та пошук даних. Він включає в себе:

1. **Repositories:** Використовуються для взаємодії з базою даних. Вони містять методи для отримання, створення, оновлення та видалення записів у базі даних.

2. **Models:** Вони представляють дані у додатку. Кожна модель відповідає таблиці в базі даних.

Configuration (Web) – це компонент системи, який включає конфігураційні файли типу `.env`, які зберігають змінні оточення та іншу конфігураційну інформацію для клієнтської частини додатку.

Configuration (Application) – це компонент системи, який включає конфігураційні файли типу `.env`, які зберігають змінні оточення та іншу конфігураційну інформацію для серверної частини програми. Сюди також входять:

3. **Database configuration:** Тут налаштовується з'єднання з базою даних, включаючи хост бази даних, порт, ім'я користувача, пароль тощо.

4. **API configuration:** Тут налаштовується API, включаючи будь-які ключі API, кінцеві точки тощо.

Database – Це місце, де зберігаються всі дані додатку. Кожній моделі на сервері відповідає таблиця в базі даних.

2.2 Розробка бази даних системи

Для керування та зберігання даних використовується СУБД SQLite, яка вбудовується безпосередньо в додаток, що надає зручність легко отримувати дані.

База даних складається з чотирьох таблиць:

1. Users – таблиця, яка представляє користувачів системи.
2. Assets – таблиця, що містить інформацію про активи, які доступні для обміну.
3. ExchangeRequests – таблиця, яка містить в собі список створених заявок на обмін.
4. Review – таблиця, яка містить список відгуків користувачів.

Опис кожної таблиці наведений у таблицях 2.1 – 2.4.

Таблиця 2.1

Опис таблиці Users

Поле	Тип даних	Опис
id	string	Ідентифікатор користувача.
email	string	Пошта користувача
password	string	Пароль користувача
name	string	Ім'я користувача
isAdmin	boolean	Визначає, чи є користувач адміністратором
createAsset	assets []	Список активів, створених адміністратором
createRequest	exchangeRequests []	Список запитів на обмін, створених користувачем
createReview	review []	Список відгуків, створених користувачем

Таблиця 2.2

Опис таблиці Assets

Поле	Тип даних	Опис
name	string	Назва активу
network	string	Мережа, до якої належить актив
address	string	Адреса активу
quantity	float	Кількість активу
networkSignature	string	Підпис мережі
user	user	Ідентифікатор користувача, який створив актив
imageUrl	string	URL зображення активу

Таблиця 2.3

Опис таблиці ExchangeRequests

Поле	Тип даних	Опис
id	string	Ідентифікатор заявки
user	user	Користувач, який створив заявку
assetFrom	assets	Ідентифікатор активу, який відправляється
assetTo	assets	Ідентифікатор активу, який отримується
amount	float	Кількість активів що відправляється
commission	float	Комісія за обмін
exchangeRate	float	Обмінний курс
totalSum	float	Сума яку отримує користувач в вибраному активі
recipientAddress	string	Адреса активу отримувача
status	string	Статус заявки
date	datetime	Дата створення заявки

Опис таблиці Review

Поле	Тип даних	Опис
id	string	Ідентифікатор відгуку
user	user	Користувач, який створив відгук
text	string	Текст відгуку
createdAt	datetime	Дата створення відгуку

З даних таблиць було сформовано діаграму бази даних (рис. 2.4):

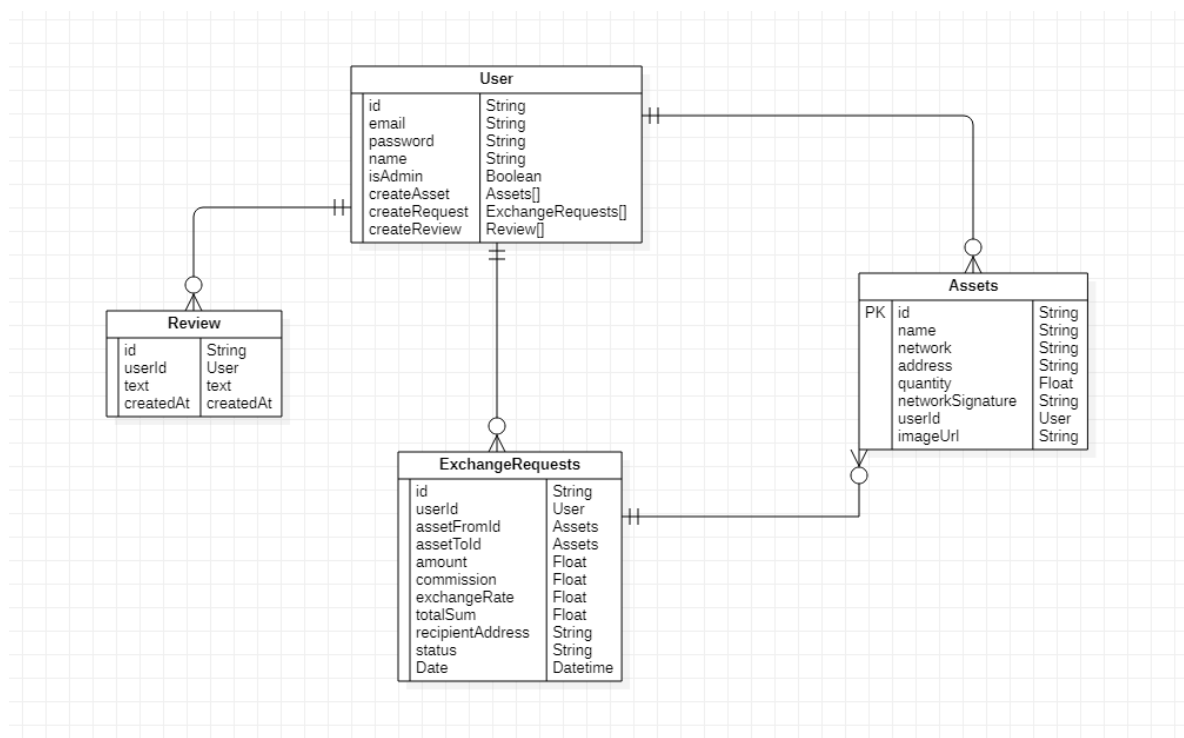


Рис. 2.4 Діаграма бази даних

2.3 Проектування та реалізація алгоритмів роботи системи

Веб-додаток дозволяє користувачам зареєструватись та авторизуватись у своїх облікових записах. Система надає можливість кожному зареєстрованому користувачу створювати заявки на обмін активів. Після заповнення відповідної форми, запит на обмін додається в базу даних, після чого адміністратор може управляти заявкою, приймаючи або відхиляючи її. Кількість активу який отримає користувач розраховується на формі, у випадку

успішного обміну вибрана користувачем криптовалюта відправляється на вказану адресу (рис. 2.5).

Адміністратор також має можливість додавати, редагувати та видаляти активи в додатку.

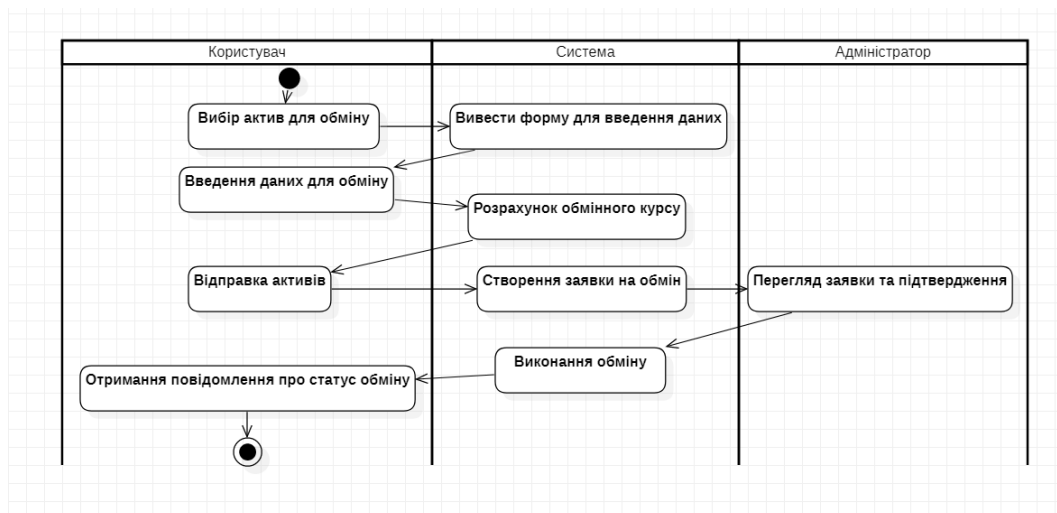


Рис. 2.5 Діаграма діяльності здійснення успішного обміну

2.4 Реалізація вебдодатку з обміну цифрових активів

Серверна частина додатку реалізована за допомогою Node.js та Express.js. Основні компоненти серверної частини включають контролери, маршрути, проміжне програмне забезпечення (middlewares) та обробку помилок.

Розглянемо етап створення заявки на обмін активів та її погодження. Щоб мати можливість створювати заявки, користувачу необхідно зареєструватись в системі. Метод для створення заявки показаний на (рис. 2.6).

```

const create = async (req,res) =>{
  try{
    const data = req.body;

    if (!data) {return res.status(400).json({ message: "Всі поля обов'язкові!" })}
    const exchange = await prisma.exchangeRequests.create({
      data: data
    })
    return res.status(201).json(exchange)
  }catch(err) {
    console.log(err)
    res.status(500).json({ message: "Не вдалось створити заявку" });
  }
}

```

Рис. 2.6 Створення нової заявки

Функція додавання нового активу до списків активів, що можуть застосовуватись для обмінів. Щоб додати новий актив застосовуються чотири методи. Перший метод є проміжним, він здійснює перевірку активу в мережі та повертає підпис мережі, мінімальну кількість для депозиту, комісію, яка знімається за переказ в мережі та мінімальну суму для відправки активу з облікового запису(рис. 2.7).

Рис. 2.7 Отримання підпису мережі активу

```
const getDepositAddress = async (coin, network) => {
  const options = {};
  if (network) {
    options.network = network;
  }
  const addressResponse = await client.depositAddress(coin, options);
  const coinConfig = await getCoinConfig();
  const networkInfo = getNetworkInfo(coinConfig, coin, network);
  return {
    ...addressResponse.data,
    networkInfo
  };
};
```

Рис. 2.8 Збирання інформації про актив

Далі функція приймає дані з клієнтської частини про актив та намагається отримати інформацію про нього надіславши запит до платформи (рис 2.9).

```
const getDepositAddressRoute = async (req, res) => {
  const { coin } = req.params;
  const { network } = req.query;

  try {
    const addressData = await getDepositAddress(coin, network);
    res.status(200).json(addressData);
  } catch (error) {
    console.error('Error fetching deposit address:', error);
    res.status(500).json({ message: 'Не вдалось отримати адресу для депозиту' });
  }
};
```

Рис. 2.9 Отримання даних з клієнтської сторони

Основний метод створення активу приймає дані введені у відповідній формі адміністратором через візуальний інтерфейс і викликає попередній метод та у випадку, якщо він не вказав мережу, метод підставить мережу по замовчуванню та створить новий актив у додатку (рис. 2.10).

```
const add = async (req, res) => {
  try {
    const data = req.body;
    if (!data.name) {return res.status(400).json({ message: "Всі поля обов'язкові!" });}
    if (!req.user.isAdmin) {
      return res.status(403).json({ message: "Доступ заборонено. Тільки адміністратор може додавати активи." });
    }
    const addressData = await getDepositAddress(data.name, data.network);
    const asset = await prisma.assets.create({
      data: {
        name: data.name,
        network: data.network || addressData.network,
        networkSignature: addressData.networkInfo.networkSignature,
        withdrawFee: parseFloat(addressData.networkInfo.withdrawFee),
        withdrawMin: parseFloat(addressData.networkInfo.withdrawMin),
        depositDust: parseFloat(addressData.networkInfo.depositDust) || 0.00000001,
        address: addressData.address,
        userId: req.user.id,
        imageUrl: data.imageUrl
      }
    });
    return res.status(201).json(asset);
  } catch (err) {
    res.status(500).json({ message: "Не вдалось створити новий актив" });
  }
};
```

Рис. 2.10 Створення нового активу

Клієнтська частина реалізована таким чином, що поділяє усіх користувачів на три групи і кожній групі доступний свій набір маршрутів. Файл “paths.ts” містить визначення шляхів для різних сторінок вебдодатку, він

є важливою частиною навігаційної логіки, оскільки зберігає всі маршрути додатку:

```
export const Paths = {
  home: "/",
  review: "/reviews",
  status: "/status",
  contacts: "/contacts",

  login: "/login",
  register: "/register",
  cabinet: "/cabinet",
  admin: "/admin",

  assetsList: "/admin/assets",
  assetProfile: "/admin/assets/profile",
  assetsAdd: "/admin/assets/add",
  assetsEdit: "/admin/assets/edit",

  exchangesList: "/admin/exchanges",
  exchangeProfile: "/admin/exchanges/profile",
  exchangeCreate: "/admin/exchanges/create",
  exchangeEdit: "/admin/exchanges/edit"
} as const;
```

Надалі можна передавати маршрути через змінну “Paths” та застосовувати виконання компонентів при спробі перейти на відповідний шлях. У файлі “routers.tsx” відбувається поділ користувачів за допомогою використання бібліотеки “react-router-dom”[13] на групи та список сторінок які доступні для них:

```
export const router = createBrowserRouter([
  {
    element: <LayoutPage />,
    children: [
      {
        path: Paths.home,
        element: <Home />
      },
      {
        path: `${Paths.review}`,
        element: <Reviews />
      },
      {
        path: `${Paths.login}`,
        element: <Login />
      },
      {
        path: `${Paths.register}`,

```

```

        element: <Register />
      },
      {
        path: `${Paths.contacts}`,
        element: <ContactPage />
      }
    ]
  },
  {
    element: <RegisteredUser />,
    children: [
      {
        path: `${Paths.cabinet}`,
        element: <Cabinet />
      },
    ]
  },
  {
    element: <AdminOutlet />,
    children: [
      {
        path: `${Paths.admin}`,
        element: <Admin />
      },
      {
        path: `${Paths.assetsList}`,
        element: <ListAssets />
      },
      {
        path: `${Paths.assetsAdd}`,
        element: <AddAssets />
      },
      {
        path: `${Paths.assetProfile}/${id}`,
        element: <ProfileAsset />
      },
      {
        path: `${Paths.assetsEdit}/${id}`,
        element: <EditAssets />
      },
      {
        path: `${Paths.exchangesList}`,
        element: <ListExchanges />
      },
      {
        path: `${Paths.exchangeProfile}/${id}`,
        element: <ProfileExchange />
      },
    ]
  },
],
},

```

```

    {
      element: <LayoutPage />,
      children: [
        {
          path: '*/',
          element: <Home />
        }
      ]
    }
  ]
});

```

Не зареєстровані користувачі можуть переглядати всі сторінки що знаходяться в групі “LayoutPage”, для авторизованих користувачів є доступ до компоненту “RegisteredUser”, який містить всі компоненти попередньої групи та можливість перейти на сторінку особистого кабінету. Після чого ми додаємо цей компонент до головного файл “index.tsx” та обгортаємо наш додаток “App” (рис. 2.11). Цей підхід забезпечує гнучке та масштабоване управління маршрутами у додатку, дозволяючи легко додавати нові маршрути або змінювати існуючі, а також зручно корегувати доступом до сторінок для користувачів з різними правами.

```

import './index.css';

const container = document.getElementById('root')!;
const root = createRoot(container);

root.render(
  <Provider store={store}>
    <ConfigProvider
      theme={{
        algorithm: theme.darkAlgorithm,
        components: {
          Radio: { ...
        },
        List :{ ...
      }
    }
  }
  >
    <App notification={{ placement: 'bottomRight' }}>
      <Auth>
        <RouterProvider router={router} />
      </Auth>
    </App>
  </ConfigProvider>
</Provider>
);

```

Рис. 2.11 Використання маршрутизації на сторінках додатку

Використання СУБД SQLite з використанням Prisma надає зручний набір методів для взаємодії з базою даних, запити виконуються з суворою типізацією, що значно зменшує кількість помилок. Розглянемо застосування Prisma на прикладі створення користувачем відгуків (рис. 2.12).

```
const add = async (req, res) => {
  try {
    const data = req.body;
    if (!data.text) {
      return res.status(400).json({ message: "Потрібно залишити текст відгуку" });
    }
    const review = await prisma.review.create({
      data: {
        text: data.text,
        userId: req.user.id,
      },
      include: {
        user: {
          select: {
            name: true,
          },
        },
      },
    });
    return res.status(201).json(review);
  } catch (err) {
    res.status(500).json({ message: "Не вдалось створити відгук" });
  }
};
```

Рис. 2.12 Створення нового відгуку в контролері

Щоб виконати запит потрібно звернутись до змінної модуля “prisma-client”, вибрати потрібну таблицю та написати метод запиту до бази даних. Для взаємодії з API використовується Redux Toolkit Query, що дозволяє зручно виконувати запити до серверу та зберігати стан додатку :

```
import { Review } from "@prisma/client"

import { api } from "../api"

export interface User {
  name: string;
}

export interface Reviews {
  id: string;
  userId: string;
  text: string;
  createdAt: Date;
  user?: User;
}

export const reviewsApi = api.injectEndpoints({
  endpoints: (builder) => ({
```

```

    getAllReviews: builder.query<Reviews[], void>({
      query: () => ({
        url: "/reviews",
        method: "GET",
      })
    }),
    addReview: builder.mutation<Review, Review>({
      query: (review) => ({
        url: "/reviews/add",
        method: "POST",
        body: review
      })
    }),
    removeReview: builder.mutation<string, string>({
      query: (id) => ({
        url: `/reviews/remove/${id}`,
        method: "POST",
        body: { id }
      })
    }),
  })
}
)
export const {
  useGetAllReviewsQuery, useAddReviewMutation, useRemoveReviewMutation
} = reviewsApi;

```

В цьому прикладі використовуються типи, згенеровані Prisma, для типізації даних відгуків. Це дозволяє гарантувати, що дані, які відправляються і отримуються від API, відповідають схемі бази даних.

Також є можливість використовувати типи безпосередньо при обробці форми для створення відгуку:

```

import { Review } from "@prisma/client";
type Props<T> = {
  onFinish: (value: T) => void;
  btnText: string;
  title: string;
  errors?: ErrorValidator[];
  review?: T;
}
export const ReviewForm = ({
  onFinish,
  btnText,
  errors,
  review: reviews
}: Props<Review>) => {
  const user = useSelector(selectUser);

```

```

return (
  <Card style={{ width: "40rem" }}>
    <Form name="review-form" onFinish={onFinish} layout="vertical"
      requiredMark="optional" initialValues={{ ...reviews, user: user?.email }}>
      <Form.Item name="user" label="Ваша пошта">
        <CustomInput name="user" readOnly prefix={<UserOutlined />} />
      </Form.Item>
      <Form.Item name="text" label="Введіть текст відгуку" rules={[{ required:
true, message: 'Будь ласка, введіть текст відгуку' }]}>
        <TextArea autoSize placeholder="Поділіться з нами своїми враженнями
про користування нашим додатком ..." maxLength={200} />
      </Form.Item>
      <Space style={{ justifyContent: "center", alignItems: "center" }}>
        <ErrorMessage errors={errors} />
        <Button htmlType="submit" style={{ marginTop: "10px" }}>
          {btnText}
        </Button>
      </Space>
    </Form>
  </Card>
)
}

```

Це надає змогу відразу перевіряти чи відповідають введені дані полям описаним в моделі бази даних.

Основна функція додатку – переказ активів на гаманець користувача. Для реалізації даної функціональності було вирішено використовувати бібліотеку “binance/connector” та ключі власного облікового запису на платформі Binance. Активи надсилаються відразу з гаманця, у разі якщо баланс буде менше ніж кількість активу, що відправляється, система сповістить про відсутність балансу (рис. 2.13).

```

const sendCrypto = async (asset, amount, address, network) => {
  try {
    const response = await client.withdraw(asset, address, amount, {
      network: network,
    });
    return response.data;
  } catch (error) {
    console.error('Error sending crypto:', error);
    throw new Error('Failed to send crypto');
  }
};

```

Рис. 2.13 Запит на здійснення виводу активу

Даний метод приймає такі параметри:

1. Asset – актив який буде відправлятися.
2. Amount – кількість активу.
3. Address – адреса гаманця отримувача.
4. Network – мережа в якій виконується переказ.

Наступним кроком буде отримання необхідних даних заявки за допомогою ендпоінту, в якому адміністратор може переглянути повну інформацію вибраної заявки:

```
...
getExchangeById: builder.query<ExchangeRequests, string>({
  query: (id) => ({
    url: `/exchanges/${id}`,
    method: "GET",
  })
}),
...
```

Отримані дані передаються в метод на сервері, виклик якого здійснює переказ активів на вказану користувачем адресу:

```
const sendExchangeFunds = async (req, res) => {
  const { id } = req.params;
  if (!req.user.isAdmin) {
    return res.status(403).json({ message: "Доступ заборонено. Тільки адміністратор може виконувати цю дію." });
  }
  try {
    const exchange = await prisma.exchangeRequests.findUnique({
      where: { id },
      include: { assetFrom: true, assetTo: true, user: true, });
    if (!exchange) { return res.status(404).json({ message: "Заявка не знайдена." }); }
    const asset = exchange.assetTo.name;
    const amount = exchange.totalSum;
    const address = exchange.recipientAddress;
    const network = exchange.assetTo.network;

    const result = await sendCrypto(asset, amount, address, network);

    // Оновлення статусу заявки
    await prisma.exchangeRequests.update({
      where: { id },
      data: {
        result,
        status: 'completed',
      }
    })
  }
}
```

```

});

res.status(200).json({
  message: 'Кошти успішно відправлено!',
});

} catch (error) {
  console.error('Error sending funds:', error);
  res.status(500).json({ message: 'Не вдалось відправити кошти.' });
}
};

```

Щоб здійснити виклик методу для здійснення відправлення коштів, був написаний додатковий ендпоінт:

```

sendCrypto: builder.mutation<string, string>({
  query: (id) => ({
    url: `/exchanges/send/${id}`,
    method: "POST",
    body: { id }
  })
}),

```

На основі цього ендпоінта був створений хук “useSendCryptoMutation”, який використовується для відправки активів. Цей хук був закріплений на кнопці, щоб при її натисканні виконувати відповідний запит (рис. 2.14).

```

const handleSendCrypto = async () => {
  hideSendModal();
  try {
    await sendCrypto(data.id).unwrap();
    localStorage.setItem('SendSuccessExchanges', 'Кошти успішно надіслані');
    navigate(`${Paths.admin}/exchanges`);
  } catch (error) {
    setError(getErrors(error));
  }
}

```

Рис. 2.14 Налаштування обробника подій для відправки активів на користувацькому інтерфейсі

Висновки до другого розділу

Під час проектування вебдодатку з обміну цифрових активів по принципу SWAP було визначено варіанти використання та проаналізовано всі можливі сценарії взаємодії користувачів із системою, що дозволило створити модель, яка чітко відображає взаємодію між різними компонентами системи.

Основні цілі проекту включали забезпечення швидкого та безпечного обміну активів та автоматизацію цього процесу. Проект був представлений як вебдодаток з інтуїтивно зрозумілим інтерфейсом, що полегшує користувачам взаємодію з системою.

Було спроектовано структуру бази даних, що забезпечує ефективне зберігання та обробку даних про заявки на обмін, користувачів та активи. На основі розроблених сутностей були створені таблиці у базі даних, а також діаграма бази даних для їх візуалізації.

Основні етапи проектування та реалізації системи включали аналіз функціональних вимог, розробку алгоритмів створення активів, обробки заявок на обмін активів, виконання переказів та оновлення статусів заявок. Було розглянуто всі необхідні функції системи, що забезпечують безперебійну роботу та відповідність встановленим стандартам

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ДОДАТКУ З ОБМІНУ ЦИФРОВИХ АКТИВІВ ПО ПРИНЦИПУ SWAP

3.1 Порядок встановлення та налаштування параметрів системи

Вебдодаток з обміну цифрових активів по принципу SWAP являє собою декілька компонентів, які взаємодіють між собою. Їх встановлення та налаштування вимагає розгортання кожного елементу. Щоб відобразити архітектуру додатку було створено діаграму розгортання (рис. 3.1).

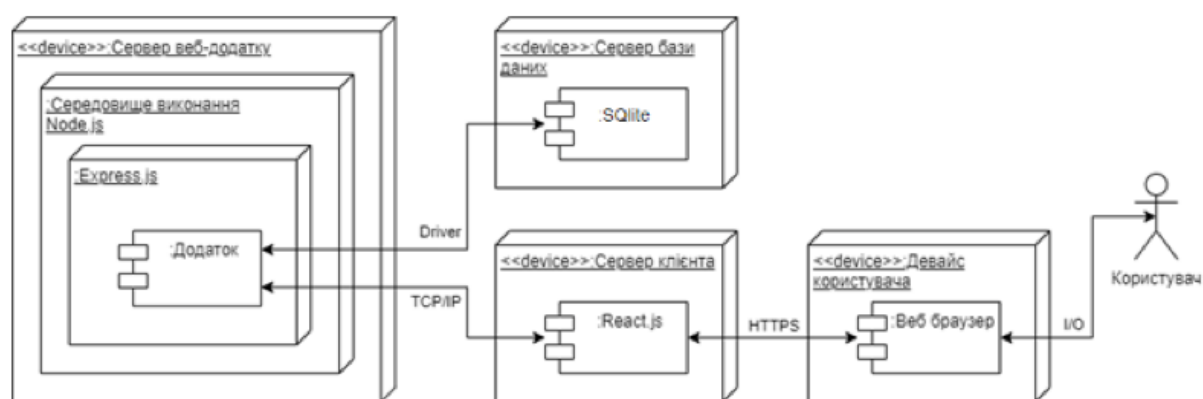


Рис. 3.1 Діаграма розгортання вебдодатку з обміну цифрових активів

Девайс користувача — Це апаратне забезпечення, за допомогою якого кінцевий користувач взаємодіє з додатком. Він включає в себе веб-браузер — програмне забезпечення, яке використовується для доступу до інтернету. Користувач взаємодіє з React.js-клієнтом через веб-браузер. Веб-браузер зв'язується з клієнтом React.js через HTTP (Hypertext Transfer Protocol).

Сервер клієнта — Це сервер, на якому розміщується клієнтський додаток React.js. Він включає в себе клієнт React.js — фронтенд-додаток. Він надсилає запити до сервера та оновлює інтерфейс на основі отриманих відповідей. Клієнт React.js взаємодіє з додатком на веб-сервері за допомогою TCP/IP (Transmission Control Protocol/Internet Protocol).

Веб-сервер — Це сервер, на якому розміщується серверний додаток Express.js. Він включає в себе:

- Середовище Node.js: Це середовище виконання, яке дозволяє запускати JavaScript поза браузером.
- Express.js: Це фреймворк, який використовується для створення серверного додатку.
- Додаток: Він обробляє запити від клієнта, взаємодіє з базою даних і надсилає відповіді назад клієнту. Додаток взаємодіє з базою даних SQLite через спеціальний драйвер для цієї бази даних.

Сервер бази даних: Це сервер, на якому розміщується ваша база даних SQLite. Він включає в себе базу даних SQLite — місце, де зберігаються всі дані додатку. Додаток взаємодіє з базою даних через драйвер.

3.2 Структура інтерфейсу веб-додатку та порядок роботи

При відкритті вебдодатку користувач переходить на головну сторінку де він відразу може побачити форму для створення нової заявки, елементи навігації які йому доступні (рис. 3.2).

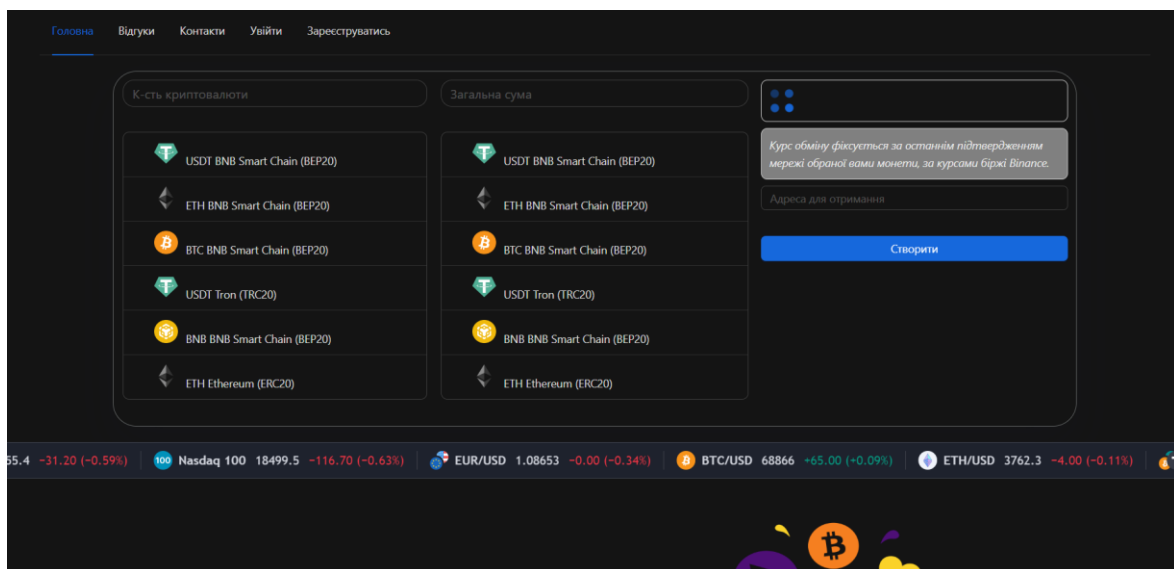


Рис. 3.2 Головна сторінка

Користувач також має можливість прокрутити нижче та ознайомитись з інформацією про додаток (рис. 3.3).

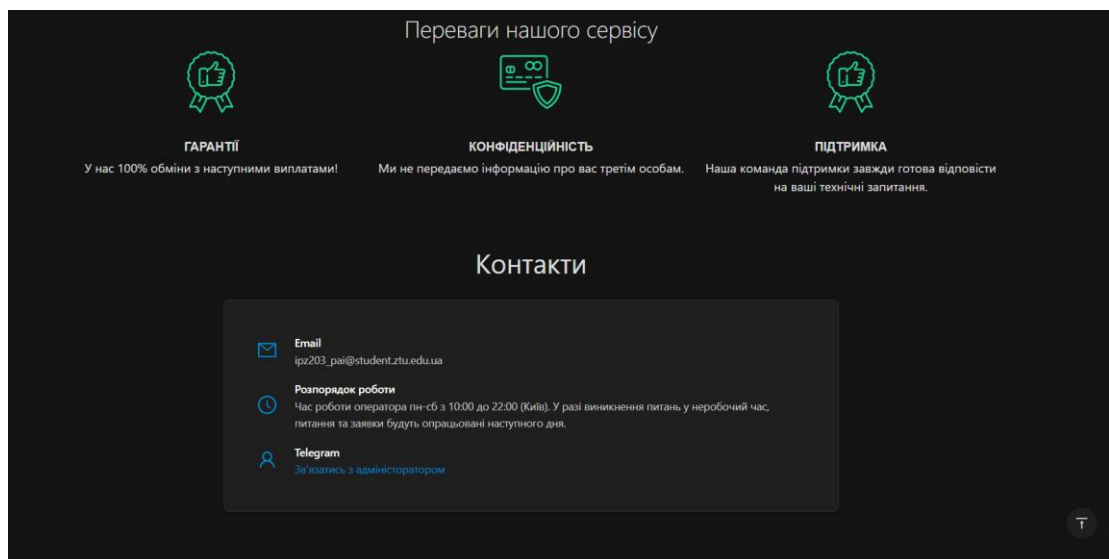


Рис. 3.3 Додаткова інформація на головній сторінці

Розглянемо роль користувача який вирішив створити заявку на обмін. Щоб отримати деталі обміну користувач вибирає потрібну пару активів, перший актив це криптовалюта, яку він бажає обміняти, а друга, яку бажає отримати (рис. 3.4).

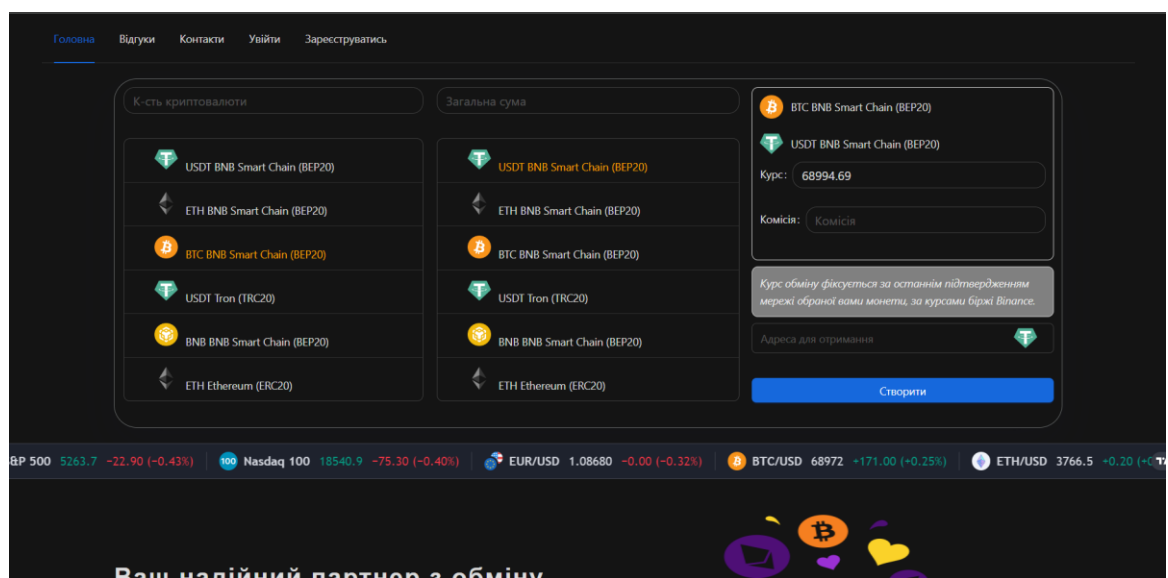


Рис. 3.4 Отримання обмінного курсу на формі створення заявки

Після чого він може вказати кількість обмінюваного активу і форма розрахує суму бажаного активу з врахуванням комісії (рис. 3.5).

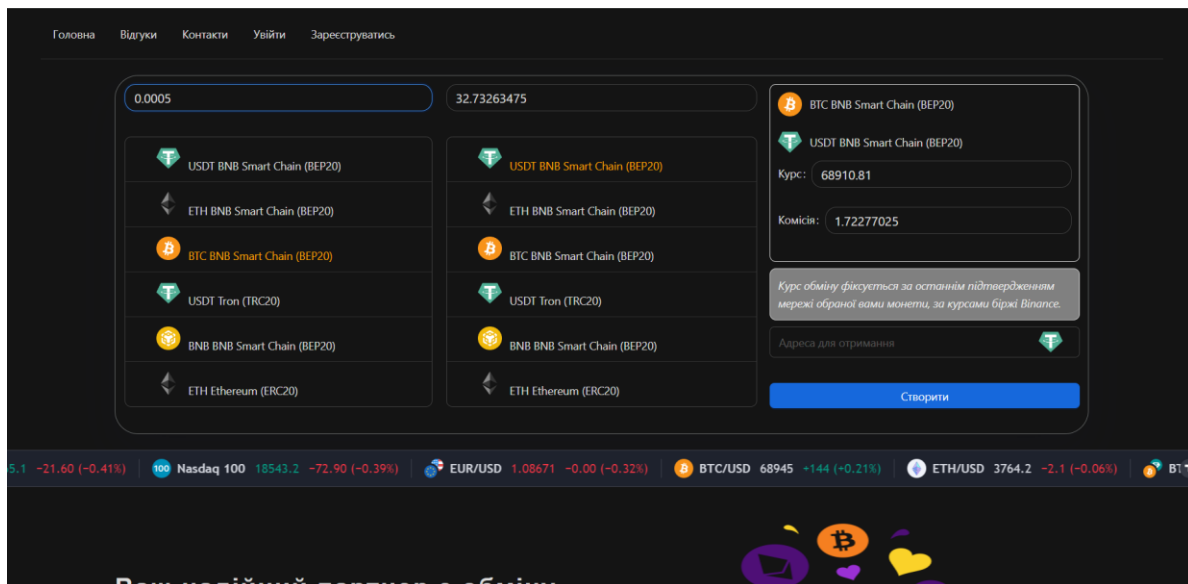


Рис. 3.5 Розрахунок суми отримання бажаного активу

Далі користувач вводить адресу криптовалюти і нажимає кнопку “Створити”. Система перевіряє чи користувач авторизувався в системі, якщо ні – то інформує його відповідним повідомленням на екрані (рис. 3.6).

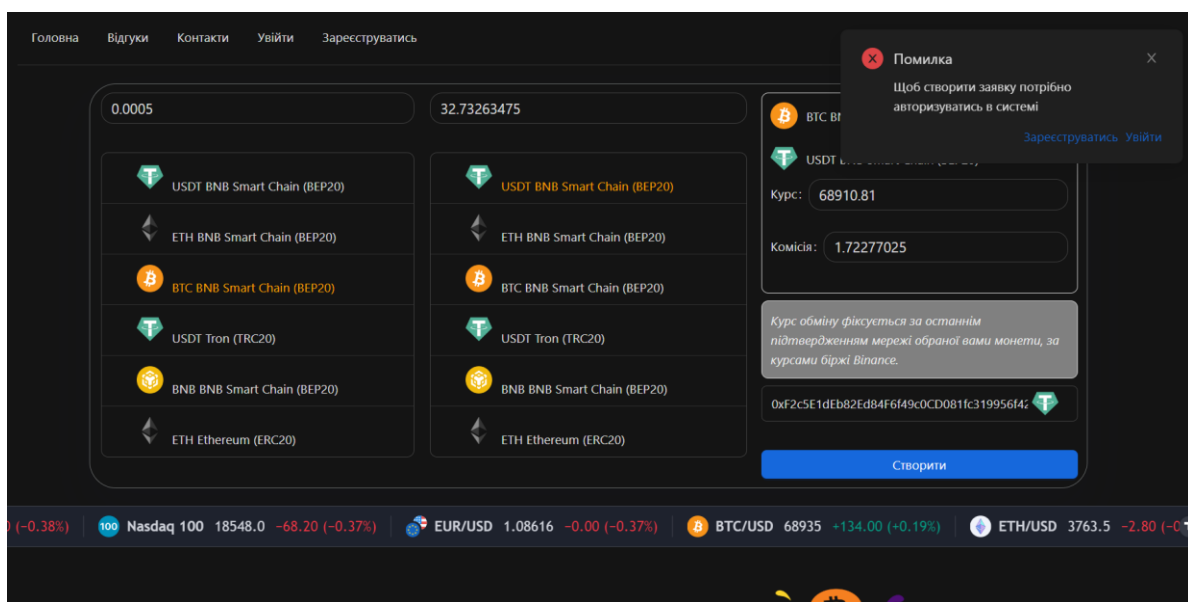


Рис. 3.6 Відображення повідомлення про помилку

Спробуємо зареєструвати користувача в системі, для цього потрібно перейти на сторінку “Зареєструватись” (рис. 3.7).

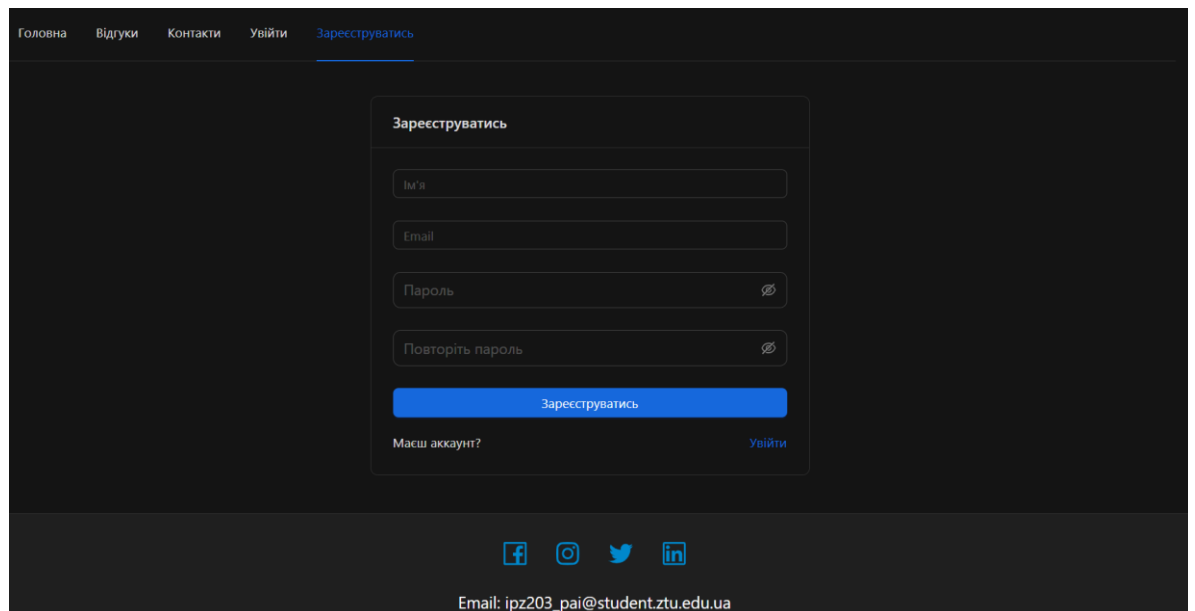


Рис. 3.7 Сторінка реєстрації облікового запису

На формі присутня валідація введених даних і, у разі помилки, відповідне поле буде підкреслюватись та з'явиться підказка, які дані потрібно ввести (рис. 3.8). У випадку правильно заповненої форми помилки не відображаються та у полях паролю відображається значок успіху (рис. 3.9).

Рис. 3.8 Приклад невдалої ваділації форми

Рис. 3.9 Приклад правильно заповненої форми

Після реєстрації система авторизує користувача в системі, про це свідчить новий елемент в групі навігаційних елементів (рис. 3.10).

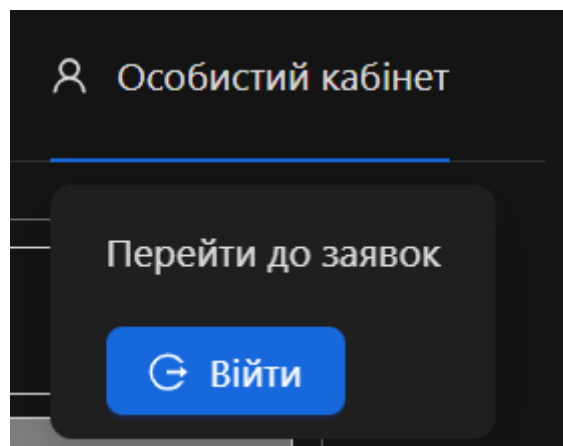


Рис. 3.10 Інформування користувача про успішну авторизацію

Виконаємо повторне створення заявки користувачем (див. рис. 3.6), який авторизований в системі. При натисканні кнопки «Створити» з'являється модальне вікно, де він може переглянути детальніше введені значення в форму, ознайомитись з правилами обміну та отримати адресу на яку потрібно надіслати обмінюваний актив (рис. 3.11).

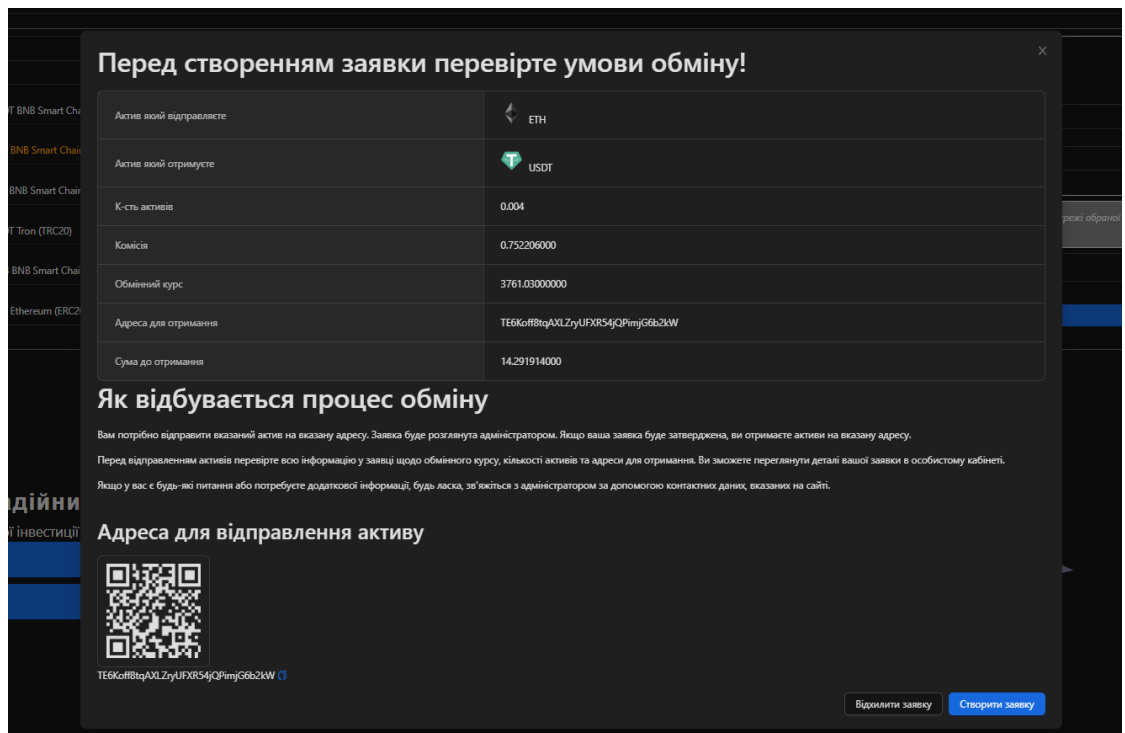


Рис. 3.11 Відображення модального вікна з деталями заявки

Після успішного створення заявки, система сповістить користувача (рис. 3.12) та у особистому кабінеті буде відображатись в таблиці (рис. 3.13).

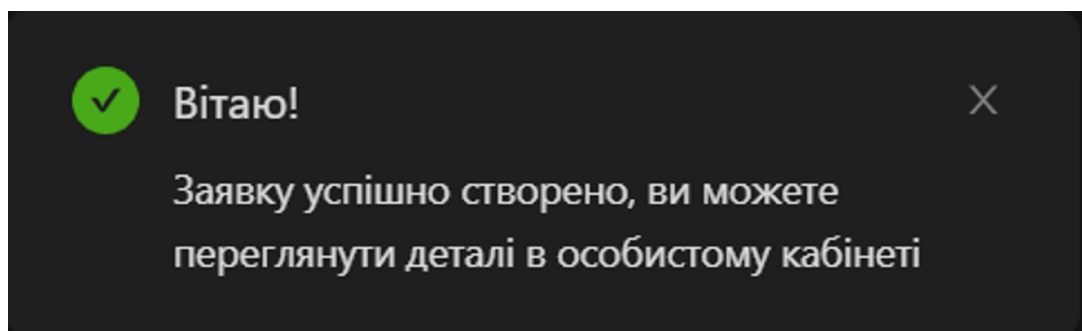


Рис. 3.12 Повідомлення про успішне створення заявки

Ваші заявки								
Віддасте	Отримуєте	К-сть активів	Комісія	Обмінний курс	Ви отримаєте	Адреса для отримання	Статус	Дата
ETH	USDT	0.004	0.752206	3761.03	14.291914	TE6Koff8tqAXLZryUFXR54jQPimjG6b2kW	pending	2024-06-04 15:51:20

Рис. 3.13 Відображення створеної заявки в особистому кабінеті

Розглянемо можливості адміністратора на прикладі створення нового активу в додатку та управління заявками користувачів. При авторизації

адміністратора в додатку, йому відкривається доступ до всіх сторінок додатку, а також сторінки пов'язані з адмініструванням (рис. 3.14).

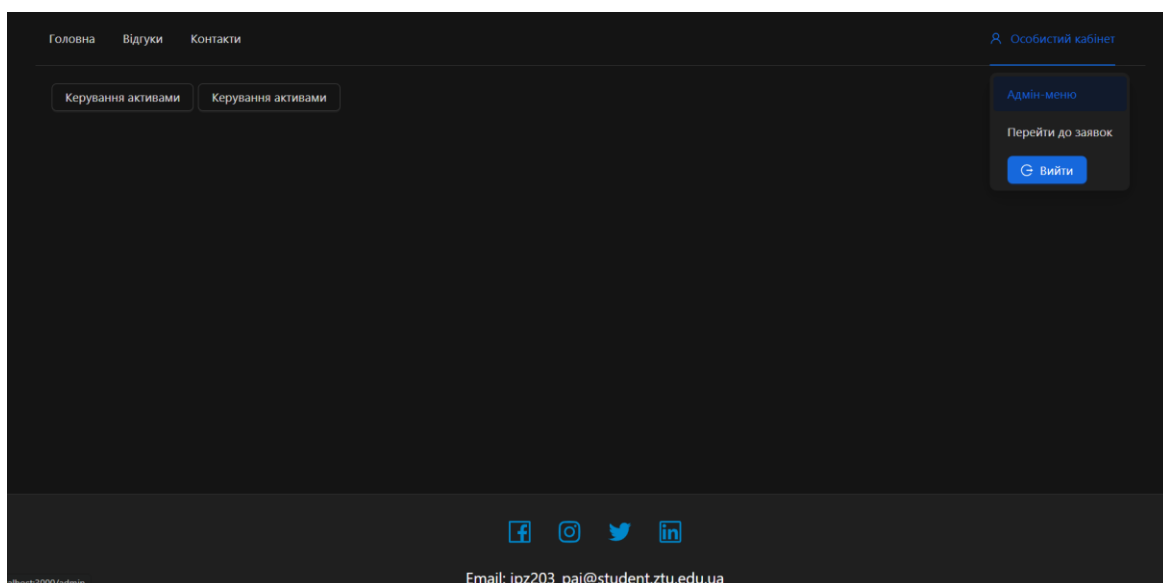


Рис. 3.14 Сторінка адміністратора

Сторінка «Керування активами» містить в собі таблицю з існуючими активами та кнопку для створення нового активу (рис. 3.15).

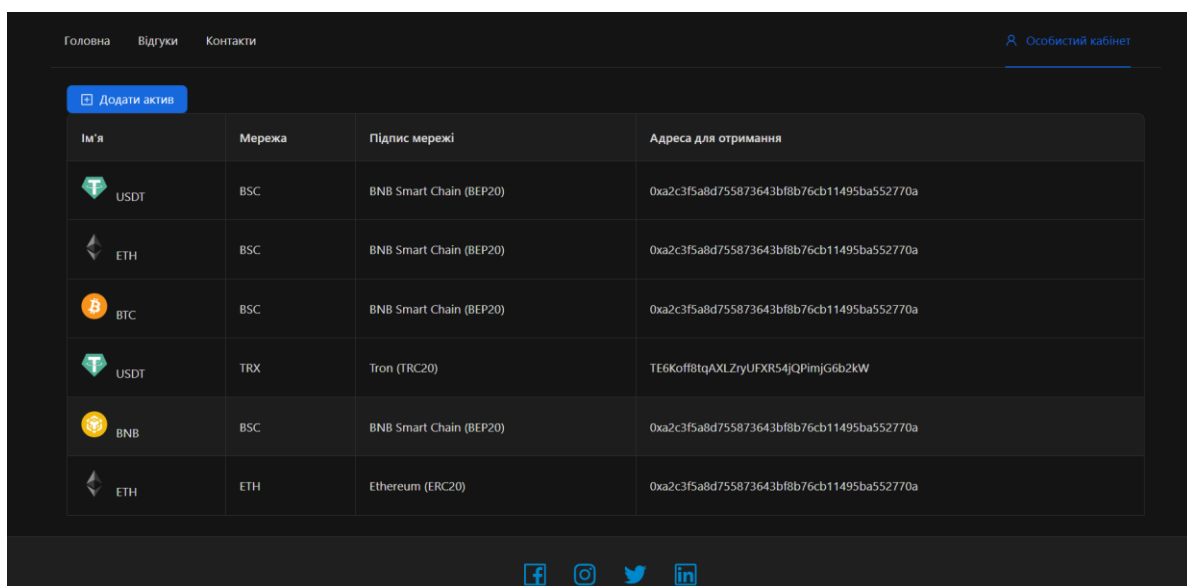


Рис. 3.15 Сторінка керування активами

Кожний рядок таблиці є інтерактивним та веде до профілю активу (рис. 3.16), на якій можна внести зміни в існуючий актив (рис. 3.17) або видалити його (рис. 3.18). Кнопка «Додати актив» направляє адміністратора на сторінку, де відображена форма для створення активу (рис. 3.19).

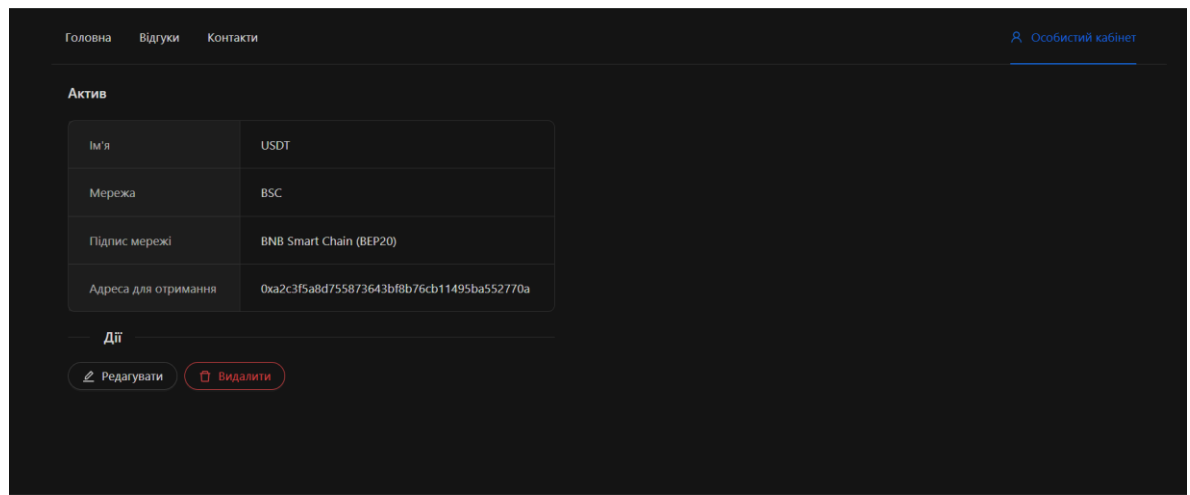


Рис. 3.16 Профіль створеного активу

Редагувати актив

USDT

BSC

<https://cryptologos.cc/logos/tether-usdt-logo.png?v=032>

<https://cryptologos.cc/>

Редагувати

Рис. 3.17 Форма редагування активу

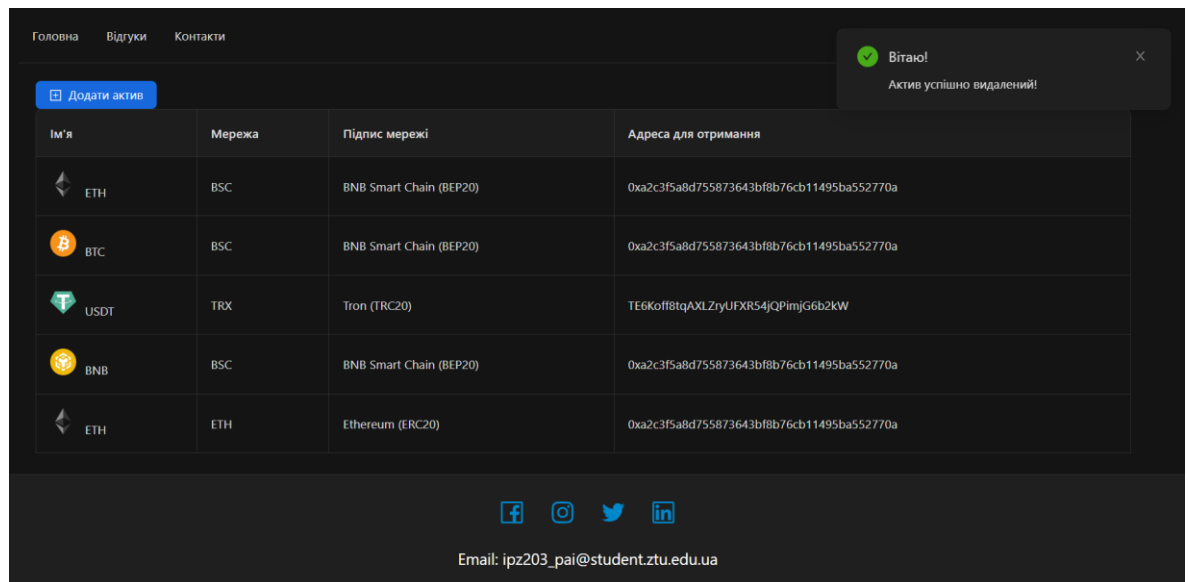


Рис. 3.18 Повідомлення про успішне видалення

The screenshot shows a form titled "Додати новий актив". It contains four input fields: the first contains "BNB", the second contains "BSC", the third contains the URL "https://cryptologos.cc/logos/bnb-bnb-logo.png?v=032", and the fourth contains "https://cryptologos.cc/". Below the input fields is a button labeled "Додати".

Рис. 3.19 Форма для додавання нового активу

Розглянемо процес створення та відображення відгуків. Неавторизований користувач може переглядати сторінку відгуків, але завжди буде отримувати повідомлення, що додавати відгуки можуть тільки зареєстровані користувачі (рис. 3.20).

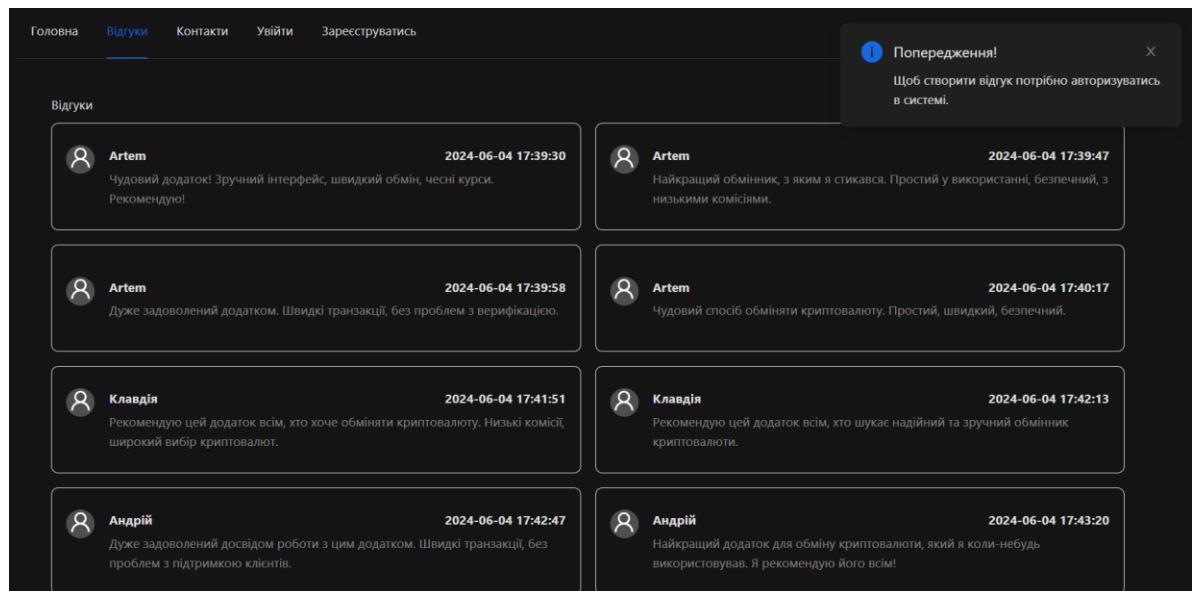


Рис. 3.20 Сторінка відгуків разом з повідомленням

У випадку, якщо користувач авторизований, після списку відгуків у нього буде відображатись форма для додавання власного відгуку (рис. 3.21). Після створення він відразу потрапляє у список і користувач отримує відповідне повідомлення (рис. 3.22).

The screenshot shows the review creation form. At the top, there are two review cards from the user Андрій, dated 2024-06-04 17:42:47 and 2024-06-04 17:43:20. Below the cards is a form with the following fields: "Ваша пошта (опціонально)" with the email ipz203_pai@student.ztu.edu.ua, "Введіть текст відгуку" with the text "Дякую за швидкий сервіс!", and a "Створити" button. At the bottom of the form, there are social media icons for Facebook, Instagram, Twitter, and LinkedIn, and the email address Email: ipz203_pai@student.ztu.edu.ua.

Рис. 3.21 Форма створення відгуку

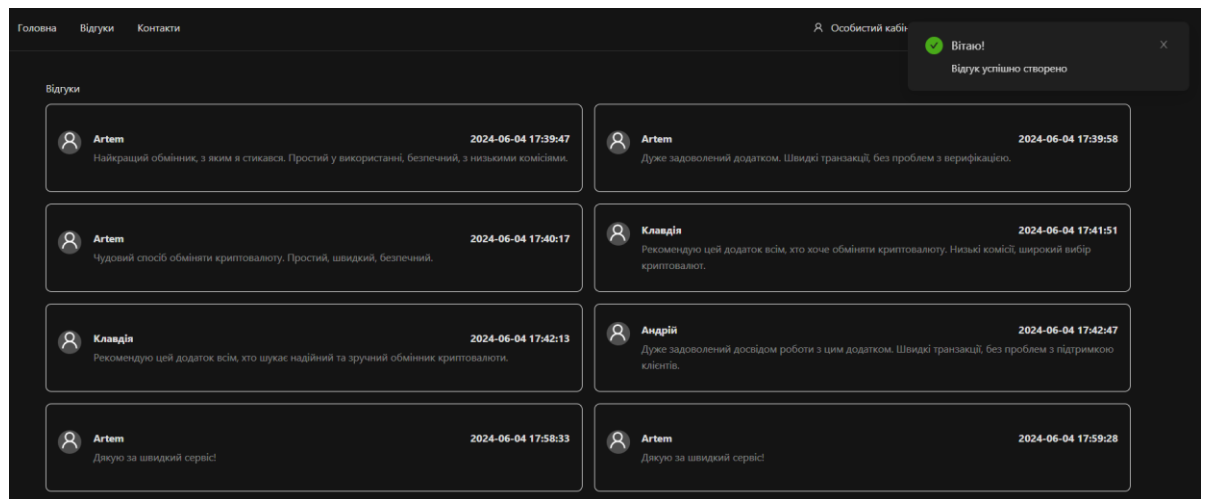


Рис. 3.22 Доданий відгук в списку

3.3 Тестування роботи вебдодатку

Тестування програмного забезпечення є невід’ємною частиною розробки будь-якого проекту. Це процес, який допомагає виявити помилки і дефекти, поліпшити якість продукту і впевнено випустити його на ринок. Цілі тестування полягають у забезпеченні надійності, функціональності та безпеки додатків, щоб користувачі могли повноцінно скористатися всіма його можливостями.

Для забезпечення стійкості та надійності під час розробки використовувались різноманітні методи тестування, такі як “причина-наслідок”, “аналіз граничних значень” та “передбачення помилок”.

Причина-наслідок застосовувався для перевірки чи правильно додаток реагує на дії користувачів системи, наприклад створення користувачем заявки або додавання нового активу адміністратором. Аналіз граничних значень застосовувався для перевірки як додаток реагує на введення граничних значень у формах та введення мінімальних кількостей кожного активу. Застосування техніки “передбачення помилок” допомогло вирішити велику кількість помилок під час розробки додатку.

Розглянемо приклад використання вищезазначених методів для тестування певних функцій додатку.

1. Отримання обмінного курсу для вибраної пари активів з платформи Binance.

Завдання: Перевірити, чи правильно додаток отримує та відображає обмінний курс для певної пари активів.

Кроки:

1. Користувач авторизується та переходить на головну сторінку.
2. Вибирає пару активів (наприклад, BTC/USDT).
3. Користувач змінює пару активів (наприклад, ETH/USDT).
4. Користувач змінює кількість обмінюваного активу.

Очікувані результати:

1. Додаток отримує правильні обмінні курси для різних пар активів.
2. Курс обміну динамічно змінюється при зміні пар активів.

Фактичні результати:

1. Додаток отримує актуальний обмінний курс для вибраної пари активів, у випадку вибору інших активів обмінний курс динамічно змінюється та відображається для нової пари вибраних активів.

Для тестування даної функціональності застосовувались методи “причина-наслідок” та “аналіз граничних значень”, що дало змогу відразу вирішити проблеми з отриманням обмінного курсу.

Використання методу “передбачення помилок” дало змогу протестувати весь користувацький інтерфейс, це допомогло вирішити проблеми та дефекти взаємодії користувачів з компонентами, а також протестувати перехід користувачів на не існуючі адреси додатку. Даний метод використовувався практично в кожній складовій розробки, починаючи від розробки контролерів на сервері та закінчуючи використанням стороннього API.

Висновки до третього розділу

У даному розділі було розглянуто процес встановлення та налаштування параметрів системи. Описано порядок розгортання системи та взаємодію її компонентів, що дозволило чітко визначити архітектуру рішення. Була

наведена діаграма розгортання, яка ілюструє, як компоненти системи співпрацюють між собою для забезпечення належної роботи додатку.

Також була представлена структура інтерфейсу додатку з обміну цифрових активів. Описано основні функції користувачів, такі як реєстрація, створення заявок на обмін та додавання відгуків, а також можливості адміністратора, включаючи додавання, редагування та видалення активів і управління обмінами користувачів.

Було висвітлено процес тестування додатку, наведено конкретні приклади застосування методів тестування.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано вебдодаток з обміну цифрових активів по принципу SWAP. На початкових етапах було проведено детальний аналіз предметної області та порівняння різних аналогів програмного продукту, що дозволило визначити основні вимоги до вебдодатку. На основі цього аналізу були сформовані основні завдання для створення вебдодатку, визначені ключові функціональні можливості та етапи їх реалізації.

Було виконано проектування вебдодатку, що включало аналіз варіантів використання та проаналізовано всі можливі сценарії взаємодії користувачів із системою.

Спроектовано структуру бази даних, що забезпечує ефективне зберігання та обробку даних про заявки на обмін, користувачів, відгуків та активи. На основі розроблених сутностей були створені таблиці у базі даних, а також діаграма бази даних для їх візуалізації.

Також була представлена структура інтерфейсу додатку з обміну цифрових активів. Описано основні функції користувачів, такі як реєстрація, створення заявок на обмін та додавання відгуків, а також можливості адміністратора, включаючи додавання, редагування та видалення активів і управління обмінами користувачів. Користувачі можуть легко та зручно обмінювати свої активи без необхідності зберігати їх в додатку.

СПИСОК ЛІТЕРАТУРИ

1. Node.js [Електронний ресурс]. Режим доступу до ресурсу: <https://nodejs.org/uk/>
2. Express.js [Електронний ресурс]. Режим доступу до ресурсу: <https://foxminded.ua/express-js/>
3. React.js [Електронний ресурс]. Режим доступу до ресурсу: <https://uk.reactjs.org/docs/getting-started.html>
4. JavaScript [Електронний ресурс]. Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
5. Changelly [Електронний ресурс]. Режим доступу до ресурсу: <https://changelly.com/decentralized-exchange>
6. Bitcoin24 [Електронний ресурс]. Режим доступу до ресурсу: <https://bitcoin24.com.ua/>
7. 100BTC.KIEV [Електронний ресурс]. Режим доступу до ресурсу: <https://100btc.kiev.ua/>
8. SQLite [Електронний ресурс]. Режим доступу до ресурсу: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/>
9. Prisma [Електронний ресурс]. Режим доступу до ресурсу: <https://www.prisma.io/docs/orm/overview/databases/sqlite>
10. Binance API [Електронний ресурс]. Режим доступу до ресурсу: <https://binance-docs.github.io/apidocs/spot/en/>
11. Binance API [Електронний ресурс]. Режим доступу до ресурсу: <https://binance.github.io/binance-connector-node/module-Wallet.html>
12. RTK Query [Електронний ресурс]. Режим доступу до ресурсу: <https://redux-toolkit.js.org/rtk-query/overview>
13. react-router-dom [Електронний ресурс]. Режим доступу до ресурсу: <https://reactrouter.com/en/main>
14. TradingView Widget [Електронний ресурс]. Режим доступу до ресурсу: <https://www.tradingview.com/widget/advanced-chart/>

15. JSON Web Token [Электронный ресурс]. Режим доступа до ресурсу: <https://jwt.io/introduction>
16. Moment.js [Электронный ресурс]. Режим доступа до ресурсу: <https://momentjs.com/docs/>
17. Cookie-parser [Электронный ресурс]. Режим доступа до ресурсу: <https://www.npmjs.com/package/cookie-parser>
18. Express-validator [Электронный ресурс]. Режим доступа до ресурсу: <https://express-validator.github.io/docs>
19. ESLint [Электронный ресурс]. Режим доступа до ресурсу: <https://eslint.org/docs/latest/>
20. Ant Design Icons [Электронный ресурс]. Режим доступа до ресурсу: <https://ant.design/components/icon/>
21. Ant Design Components [Электронный ресурс]. Режим доступа до ресурсу: <https://ant.design/components/overview/>
22. Redux Toolkit [Электронный ресурс]. Режим доступа до ресурсу: <https://redux-toolkit.js.org/>
23. Crypto Logos [Электронный ресурс]. Режим доступа до ресурсу: <https://cryptologos.cc/>
24. React Redux [Электронный ресурс]. Режим доступа до ресурсу: <https://react-redux.js.org/>
25. React useState [Электронный ресурс]. Режим доступа до ресурсу: <https://react.dev/reference/react/useState>
26. React useEffect [Электронный ресурс]. Режим доступа до ресурсу: <https://react.dev/reference/react/useEffect>
27. Types of testing [Электронный ресурс]. Режим доступа до ресурсу: <https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/>
28. StarUML documentation [Электронный ресурс]. Режим доступа до ресурсу: <https://docs.staruml.io/>
29. Bcrypt [Электронный ресурс]. Режим доступа до ресурсу: <https://www.npmjs.com/package/bcrypt>

30. Concurrently [Электронный ресурс]. Режим доступа до ресурсу:
<https://www.npmjs.com/package/concurrently>

ДОДАТКИ

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						57

```

/**
 * @route GET /api/assets/
 * @desc Отримання всіх активів
 * @access Public
 */
const all = async (req, res) => {
  try {
    const assets = await prisma.assets.findMany();

    res.status(200).json(assets);
  } catch {
    res.status(500).json({ message: "Не вдалось отримати список активів" });
  }
}

/**
 * @route POST /api/assets/add
 * @desc Додавання нового актива
 * @access Private
 */
const add = async (req, res) => {
  try {
    const data = req.body;

    if (!data.name) {
      return res.status(400).json({ message: "Всі поля обов'язкові!" });
    }

    const addressData = await getDepositAddress(data.name, data.network);
    const asset = await prisma.assets.create({
      data: {
        name: data.name,
        network: data.network || addressData.network,
        networkSignature: addressData.networkSignature,
        address: addressData.address,
        userId: req.user.id,
        imageUrl: data.imageUrl
      }
    });

    return res.status(201).json(asset);
  } catch (err) {
    res.status(500).json({ message: "Не вдалось створити новий актив" });
  }
};

/**
 * @route POST /api/assets/remove/:id
 * @param {string} req.params.id
 */
const remove = async (req, res) => {
  const { id } = req.params;

```

```

    try {
      await prisma.assets.delete({
        where: { id }
      });

      res.status(204).json({ message: "Актив успішно видалено!" });
    } catch (error) {
      return res.status(500).json({ message: "Не вдалось видалити актив" });
    }
  }
  /**
   * Change asset data.
   * @route PUT /api/assets/edit/:id
   * @param {string} req.params.id
   */
  const edit = async (req, res) => {
    const data = req.body;
    const { id } = req.params;

    try {
      await prisma.assets.update({
        where: { id },
        data,
      });

      res.status(204).json({ message: "Актив успішно змінено!" });
    } catch {
      return res.status(500).json({ message: "Не вдалось змінити актив" });
    }
  }
  /**
   * @route GET /api/assets/:id
   * @param {string} req.params.id
   */
  const assetById = async (req, res) => {
    const { id } = req.params;

    try {
      const asset = await prisma.assets.findUnique({
        where: {
          id
        }
      });
      res.status(200).json(asset)

    } catch {
      return res.status(500).json({ message: "Не вдалось отримати актив" });
    }
  }

  module.exports = {
    all,

```

```

    add,
    remove,
    edit,
    assetById,
  }
}
/**
 * @route GET /api/exchanges/
 * @desc Отримання всіх заявок(адміністратор)
 * @access Admin
 */
const all = async (req, res) => {
  try {

    const exchanges = await prisma.exchangeRequests.findMany({
      include: {
        assetFrom: true,
        assetTo: true,
        user: true
      }
    });
    res.status(200).json(exchanges);
  } catch {
    res.status(500).json({ message: "Не вдалось отримати список заявок" });
  }
}
/**
 * @route GET /api/exchanges/:id
 * @desc Отримання однієї заявки(адміністратор)
 * @access Admin
 */
const exchangeById = async (req, res) => {
  const { id } = req.params;

  try {
    const exchange = await prisma.exchangeRequests.findUnique({
      where: {
        id
      },
      include: {
        assetFrom: true,
        assetTo: true,
        user: true,
      }
    });
    const assetFromName = exchange.assetFrom.name;
    const assetToName = exchange.assetTo.name;

    const exchangeWithNames = { ...exchange, assetFromId: assetFromName, assetToId:
assetToName };

    res.status(200).json(exchangeWithNames);
  }
}

```

```

    } catch {
      return res.status(500).json({ message: "Не вдалось отримати актив" });
    }
  }
}
/**
 * @route GET /api/exchanges/userexchanges/
 * @desc Отримання всіх заявок користувача(user)
 * @access User
 */
const getUserExchanges = async (req, res) => {
  try {
    const userId = req.user.id;

    const exchanges = await prisma.exchangeRequests.findMany({
      where: {
        userId,
      },
      include: {
        assetFrom: true,
        assetTo: true,
        user: true,
      },
    });

    res.status(200).json(exchanges);
  } catch {
    res.status(500).json({ message: "Не вдалось отримати список заявок" });
  }
}

const create = async (req, res) => {
  try {
    const data = req.body;

    if (!data.assetFromId || !data.assetToId || !data.amount ||
      !data.commission || !data.exchangeRate || !data.totalSum ||
      !data.recipientAddress
    ) {
      return res.status(400).json({ message: "Всі поля обов'язкові!" })
    }

    const parsedData = {
      assetFromId: data.assetFromId,
      assetToId: data.assetToId,
      amount: parseFloat(data.amount),
      commission: parseFloat(data.commission),
      exchangeRate: parseFloat(data.exchangeRate),
      totalSum: parseFloat(data.totalSum),
      recipientAddress: data.recipientAddress,
      Date: new Date(),
      userId: req.user.id
    };

    const exchange = await prisma.exchangeRequests.create({
      data: parsedData
    });
  }
}

```

```

    })
    console.log(exchange)
    return res.status(201).json(exchange)

  } catch (err) {
    console.log(err)
    res.status(500).json({ message: "Не вдалось створити заявку" });
  }
}

/**
 * @route POST /api/exchanges/:id
 * * @desc Видалення заявки
 * * @access Admin
 * @param {string} req.params.id
 */
const remove = async (req, res) => {
  const { id } = req.params;

  try {
    await prisma.exchangeRequests.delete({
      where: { id }
    });

    res.status(204).json({ message: "Заявку успішно видалено!" });
  } catch (error) {
    console.log(error);
    return res.status(500).json({ message: "Не вдалось видалити Заявку" });
  }
}

/**
 * @route PUT /api/exchanges/edit/:id
 * @param {string} req.params.id
 */
const edit = async (req, res) => {
  const data = req.body;
  const { id } = req.params;

  try {
    const assetFrom = await prisma.assets.findUnique({
      where: { name: data.assetFromName },
    });
    const assetTo = await prisma.assets.findUnique({
      where: { name: data.assetToName },
    });

    if (!assetFrom || !assetTo) {
      return res.status(400).json({ message: "Некоректні імена активів." });
    }

    const parsedData = {
      assetFromId: assetFrom.id,
      assetToId: assetTo.id,
    }

```

```

        amount: parseFloat(data.amount),
        commission: parseFloat(data.commission),
        exchangeRate: parseFloat(data.exchangeRate),
        totalSum: parseFloat(data.totalSum),
        recipientAddress: data.recipientAddress,
        Date: new Date(data.Date),
        status: data.status,
    };

    await prisma.exchangeRequests.update({
        where: { id },
        parsedData,
    });

    res.status(204).json({ message: "Заявку успішно змінено!" });
} catch {
    return res.status(500).json({ message: "Не вдалось змінити заявку" });
}
}
/**
 * @route POST /api/exchanges/send/:id
 * @desc Відправка коштів по заявці
 * @access Admin
 */
const sendExchangeFunds = async (req, res) => {
    const { id } = req.params;

    try {
        const exchange = await prisma.exchangeRequests.findUnique({
            where: { id },
            include: {
                assetFrom: true,
                assetTo: true,
                user: true,
            }
        });

        if (!exchange) {
            return res.status(404).json({ message: "Заявка не знайдена." });
        }

        // Дані для відправки
        const asset = exchange.assetTo.name;
        const amount = exchange.totalSum;
        const address = exchange.recipientAddress;
        const network = exchange.assetTo.network;

        // Відправка
        const result = await sendCrypto(asset, amount, address, network);

        // Оновлення статусу заявки

```

```

await prisma.exchangeRequests.update({
  where: { id },
  data: {
    status: 'completed',
  }
});

res.status(200).json({
  message: 'Кошти успішно відправлено!',
});

} catch (error) {
  console.error('Error sending funds:', error);
  res.status(500).json({ message: 'Не вдалось відправити кошти.' });
}
};

module.exports = {
  all,
  getUserExchanges,
  exchangeById,
  create,
  remove,
  edit,
  sendExchangeFunds
}

```