Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

АВТОМАТИЗОВАНА СИСТЕМА ТОРГІВЛІ НА КРИПТОВАЛЮТНИХ БІРЖАХ НА ОСНОВІ NODE.JS ТА БІБЛІОТЕКИ REACT

AUTOMATED TRADING SYSTEM ON CRYPTOCURRENCY EXCHANGES BASED ON NODE.JS AND THE REACT LIBRARY

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Баранчук Сергій Анатолійович

(підпис)

Керівник:
к.т.н., доц.
Бортник Катерина Яківна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Баранчуку Сергію Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Автоматизована система торгівлі на криптовалютних біржах на основі Node.js та бібліотеки React*

Керівник роботи *к.т.н., доцент Бортник Катерина Яківна*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Загальні відомості про криптовалютні системи
Огляд структури об'єкту проектування
Розробка автоматизованої криптовалютної веб-системи
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Рисунки та схеми: схеми, що демонструють архітектуру криптовалют; макет веб-системи; інтерфейс IDE WebStrong, лістинг коду, моделі БД, інтерфейс розробленої веб-системи
Використані технології: HTML, CSS, JavaScript, React, Node.js, Express.js, MongoDB, IDE WebStrong, Git, GitHub, Figma, OS Ubuntu.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Загальні відомості про криптовалютні схеми	*Бортник К.Я.*		
Огляд структури об'єкту проектування	*Бортник К.Я.*		
Розробка автоматизованої криптовалютної веб-системи	*Бортник К.Я.*		
Висновки	*Бортник К.Я.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	10.11.2022 р.	Виконано
2.	*Дослідження літератури*	20.11.2022 р.	Виконано
3.	*Аналіз предметної області*	14.01.2023 р.	Виконано
4.	*Вибір засобів розробки*	16.01.2023 р.	Виконано
5.	*Створення структури об'єкту проектування*	25.02.2023 р.	Виконано
6.	*Розробка та тестування веб-системи*	24.03.2023 р.	Виконано
7.	*Оформлення матеріалів роботи*	05.04.2023 р.	Виконано
8.	*Нормоконтроль*	20.05.2023 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	29.05.2023 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	07.06.2023 р.	Виконано

Здобувач вищої освіти _____ (Баранчук С.А.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Бортник К.Я.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Баранчук С.А. Автоматизована система торгівлі на криптовалютних біржах на основі Node.js та бібліотеки React. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, трьох додатків.

У першому розділі здійснено огляд предметної області, розглядаються загальні відомості про криптовалютні системи, основні поняття про криптовалютну галузь, криптовалютний ринок, криптовалютні біржі, автоматизовану (скриптову) торгівлю та блокчейн-технології. Також в цьому розділі були описані переваги розробки власної системи в порівнянні з вже існуючими аналогами та здійснено вибір засобів розробки. Обрано наступні засоби: база даних MongoDB, Backend-фреймворк Express.js, бібліотека React для створення UI, платформа Node.JS, мова програмування JavaScript, HTML, CSS, IDE JetBrains WebStorm та GitHub в якості хмарного сховища вихідного коду, що використовує Git в якості системи для контролю версій проекту.

У другому розділі було обґрунтовано обрані засоби розробки, та проведено огляд структури об'єкту розробки: описана функціонально-структурна схема, принципи роботи веб-системи, здійснено опис необхідних класів та створена модель бази даних.

Третій розділ присвячено практичній розробці та тестуванню автоматизованої веб-системи для криптовалютної торгівлі. Було здійснено опис архітектури та принципів взаємодії усіх складових системи між собою, розроблено Frontend та Backend складові, забезпечено сумісність об'єкту розробки з різними пристроями, описаний принцип роботи системи авторизації, проведено тестування системи з метою пошуку несправностей та продемонстровано працездатність створеної веб-системи.

Об'єкт дослідження – технології розробки веб-систем та веб-застосунків для автоматизації процесу торгівлі криптовалютами.

Предмет дослідження – автоматизована веб-система для торгівлі криптовалютами на криптовалютних біржах, з використанням технологій Node.js та бібліотеки React.

Метою роботи є розробка автоматизованої веб-системи для торгівлі на криптовалютних біржах, що буде в автоматичному режимі знаходити можливості для заробітку, забезпечення коректної взаємодії усіх її компонентів, сумісності системи з різними пристроями та проведення тестування для виявлення та усунення несправностей.

Ключові слова: криптовалюти, веб-системи, автоматизована торгівля, скрипти, криптовалютні біржі, Node.js, React, стек MERN, арбітраж.

ANNOTATION

Baranchuk S.A. Automated trading system on cryptocurrency exchanges based on Node.js and React library. Manuscript.

Bachelor's qualification work of the "Computer Engineering" educational program, specialization 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

The qualification work consists of an introduction, three sections, conclusions, a list of references, and three appendices.

The first section provides an overview of the subject area, including general information about cryptocurrency systems, key concepts in the cryptocurrency industry, the cryptocurrency market, cryptocurrency exchanges, automated (scripted) trading, and blockchain technologies. This chapter also describes the advantages of developing a proprietary system compared to existing analogs and the development tools selected. The following tools have been chosen: MongoDB database, Express.js backend framework, React library for UI development, Node.JS platform, JavaScript programming language, HTML, CSS, JetBrains WebStorm IDE, and GitHub as a cloud storage for source code using Git as the version control system.

The second section justifies the chosen development tools and provides an overview of the structure of the development object, including the description of the functional-structural scheme, the principles of the web system's operation, the description of necessary classes, and the creation of a database model.

The third section focuses on the practical development and testing of an automated web system for cryptocurrency trading. It includes the description of the architecture and interaction principles of all system components, the development of frontend and backend components, ensuring compatibility of the development object with different devices, the description of the authentication system's operation principle, testing the system to identify malfunctions, and demonstrating the functionality of the created web system.

Object of research – web development technologies and web applications for automating cryptocurrency trading processes.

Subject of research – automated web system for cryptocurrency trading on cryptocurrency exchanges, using Node.js and React technologies.

The aim of the work is to develop an automated web system for trading on cryptocurrency exchanges that will automatically identify profit opportunities, ensure proper interaction of all its components, compatibility with various devices, and conduct testing to detect and resolve malfunctions.

Keywords: cryptocurrencies, web systems, automated trading, scripts, cryptocurrency exchanges, Node.js, React, MERN stack, arbitrage.

ЗМІСТ

ВСТУП

Актуальність теми. За останні роки криптовалютні біржі стали одними з найбільш перспективних фінансових платформ, привертаючи увагу як досвідчених трейдерів, так і новачків. Зростаючий інтерес до криптовалют та їх потенційно високі ризики й доходи спонукають багатьох інвесторів до участі в цьому ринку. Завдяки постійному зростанню капіталізації ринку криптовалют і появі нових торгових інструментів, автоматизація торгівлі на криптовалютних біржах стає все більш актуальною, так як надає користувачеві неабияку перевагу на криптовалютному ринку.

Метою роботи є розробка автоматизованої веб-системи для торгівлі на криптовалютних біржах, що забезпечуватиме пошук можливостей для заробітку в автоматичному режимі. Вона стане функціональним і надійним інструментом, який допоможе трейдерам здійснювати прибуткову торгівлю.

Об'єкт дослідження – технології розробки сучасних веб-систем та веб-застосунків.

Предмет дослідження – веб-система для автоматизації процесу торгівлі криптовалютами.

Завдання, які необхідно виконати:

– реалізувати функціонал системи, включаючи пошук можливості здійснення торгів на вибраних криптовалютних біржах, перегляду поточного стану ринку, отримання статистичних даних тощо.

– забезпечити високу продуктивність системи, що дозволить отримувати необхідну інформацію про ринок у реальному часі.

– забезпечити зручний та інтуїтивно зрозумілий інтерфейс користувача.

– провести тестування розробленої системи з метою підтвердження відповідності вимогам, а також виявлення та виправлення можливих помилок та недоліків.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ ПРО КРИПТОВАЛЮТНІ СИСТЕМИ

1.1 Аналіз криптовалютної галузі

Криптовалютна галузь є однією з найбільш динамічно зростаючих галузей за останні кілька років. За даними CoinMarketCap (найпопулярніший у світі крипто-сайт для відстеження цін, а також великий постачальник криптографічних даних), станом на 2023 на ринку існує понад 9 тис. криптовалют та більше 400 крипто-бірж [1]. Все це забезпечує велике різноманіття та високу конкуренцію на ринку.

Автоматизована система для торгівлі на криптовалютних біржах повинна обробляти різні типи даних, включаючи дані про курси, статистичні дані про ринок, дані про замовлення на купівлю та продаж криптовалют. Обсяг та формат інформаційних потоків будуть залежати від обсягу торгів на конкретній біржі та від вимог до системи торгівлі.

Функціональний опис вихідного об'єкту повинен включати такі функції, як наприклад, автоматичне виконання замовлень на певних умовах, аналіз даних для прийняття рішень про покупку чи продаж криптовалют, моніторинг ринку для виявлення нових можливостей для торгівлі. Важливо також визначити, як буде відбуватися взаємодія з біржами та як будуть оброблятися дані з них.

Створення такої системи є доцільним з точки зору автоматизації процесу торгівлі та збільшення ефективності прийняття рішень. Застосування високопродуктивних технологій та підходів, таких як Node.js та React, може допомогти забезпечити швидке та надійне функціонування системи торгівлі.

Такий засіб може бути ефективним інструментом для торгів на криптовалютних ринках, якщо будуть враховані всі необхідні функціональні вимоги до системи.

Основна методика торгівлі, на яку буде орієнтуватися запланована система – це арбітраж. Криптовалютний арбітраж – це процес купівлі та продажу криптовалют з метою отримання прибутку в результаті різниці у їхньому

ціновому розриві на різних біржах. Арбітражист шукає можливості для покупки активу на одному ринку за низькою ціною, а потім його продаж на іншому ринку за вищою ціною, отримуючи прибуток в результаті.

Криптовалютні біржі можна розділити на два типи: централізовані (CEX) та децентралізовані (DEX).

Централізована криптобіржа (CEX) – це платформа, яка зберігає кошти користувачів та контролює процес обміну. Такий тип бірж майже завжди потребує верифікації особистості, що дає можливість використовувати функції, такі як купівля/продаж активів за готівку або іншими способами оплати.

Децентралізована криптобіржа (DEX), навпаки, не зберігає кошти своїх користувачів і не контролює процес обміну. Все виконується через блокчейн-технології без посередників, а користувачі керують своїм портфелем самостійно.

Графічно, різниця між біржами зображена на рисунках 1.1 та 1.2.

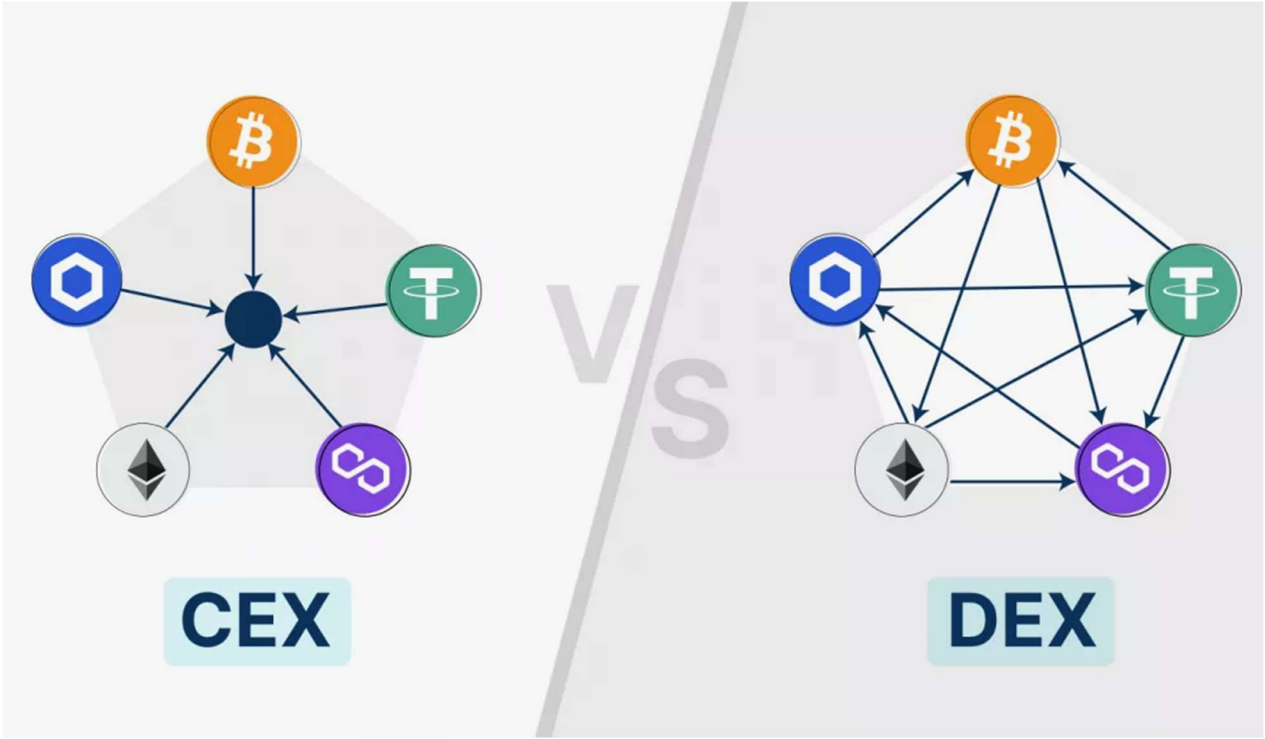

Рисунок 1.1 – Схема архітектурної різниці між різними типами криптовалютних бірж

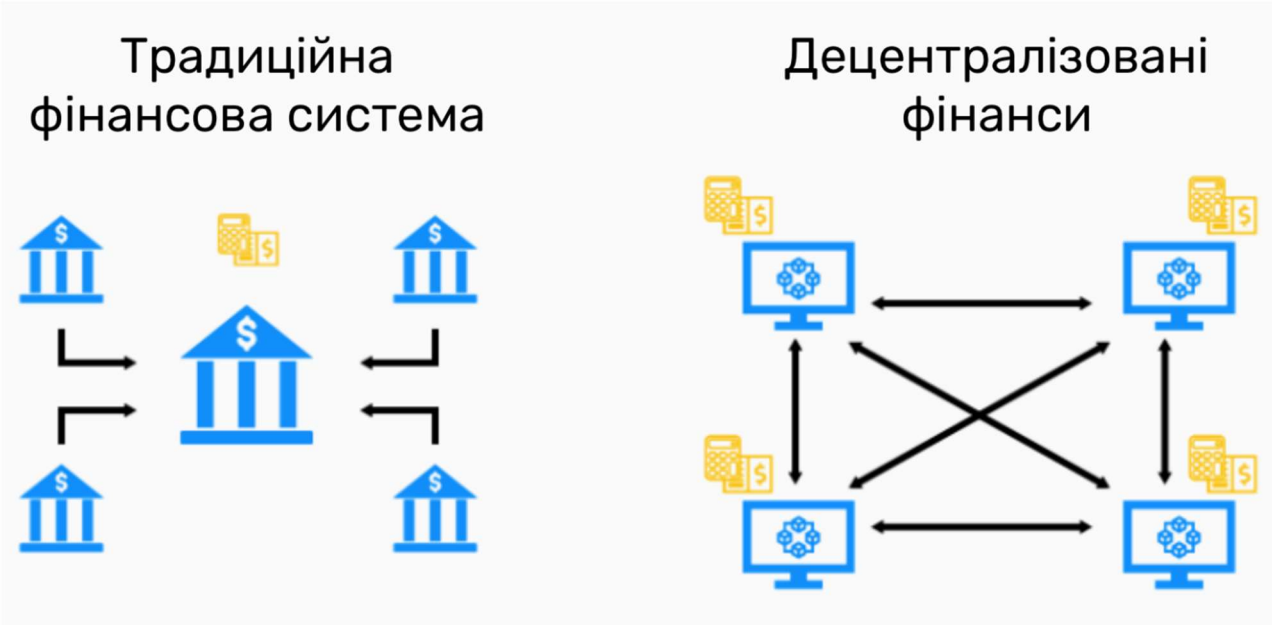

Рисунок 1.2 – Загальна схема різниці між різними типами криптовалютних бірж

Обидва види бірж мають свої переваги та недоліки. CEX-біржі часто пропонують ширший вибір активів та вищу швидкість обміну, але мають більший ризик взлому та крадіжки коштів. DEX-біржі надають користувачам повний контроль над їх активами, але можуть мати менший обсяг ліквідності та складний інтерфейс для новачків.

Ліквідність у криптовалютах відноситься до того, наскільки швидко та ефективно можна купити або продати криптовалюту на ринку без значних втрат. Зазвичай, чим більше кількість торговельних площадок, на яких доступна криптовалюта, та чим більше кількість трейдерів, які торгують цією криптовалютою, тим вище її ліквідність.

Звичайні біржі (CEX) працюють за моделлю книги замовлень. Вона працює добре, якщо на ринку достатньо покупців та продавців. Але якщо ситуація інша, то токенам не вистачає ліквідності через низький обсяг. Через це стає важко вести торги. Такі неліквідні токени можуть викликати непередбачувані коливання ціни при здійсненні окремих великих угод.

Децентралізовані біржі (DEX) для вирішення цієї проблеми використовують пул ліквідності. Пул ліквідності – це ніщо інше, як автоматизований маркет-мейкер, який забезпечує ліквідність, щоб запобігти

значним коливанням цін на активи. Детальне порівняння централізованих та децентралізованих бірж представлене у таблиці 1.1.

Таблиця 1.1 – Порівняння централізованих та децентралізованих бірж

	Централізовані криптобіржі (CEX)	Децентралізовані криптобіржі (DEX)
Зберігання коштів	Біржа зберігає кошти користувачів	Користувачі мають повний контроль над своїми коштами
Контроль процесу обміну	Біржа контролює процес обміну	Обмін виконується без посередництва біржі
Верифікація особистості	Потрібна верифікація особистості для використання	Не потрібна верифікація особистості для використання
Швидкість обміну	Вища швидкість обміну активів	Нижча швидкість обміну активів
Вибір активів	Ширший вибір активів	Обмежений вибір активів
Ризик злому та крадіжки	Високий ризик злому та крадіжки коштів	Низький ризик злому та крадіжки коштів
Ліквідність активів	Висока ліквідність активів	Низька ліквідність активів
Інтерфейс користувача	Простий та зрозумілий інтерфейс користувача	Складний та технічний інтерфейс користувача

Варто зауважити, що часто, централізовані біржі в автоматичному режимі виявляють арбітражні можливості на власних ринках, і самостійно займаються внутрішнім криптовалютним арбітражем. Саме тому, при розробці системи для торгівлі, в пріорітеті буде орієнтація саме на міжбіржевий арбітраж (між різними біржами, та навіть між різними типами бірж).

На відміну від централізованих бірж, децентралізовані біржі практично не займаються самостійним арбітражем, тому варто також окремо розглянути цей ринок. Як вже було зазначено вище, процес торгів на DEX-біржах відбувається

за допомогою звичайних блокчейн-транзакцій. Блокчейн (рис. 1.3) – це основа всієї криптовалютної індустрії [2].

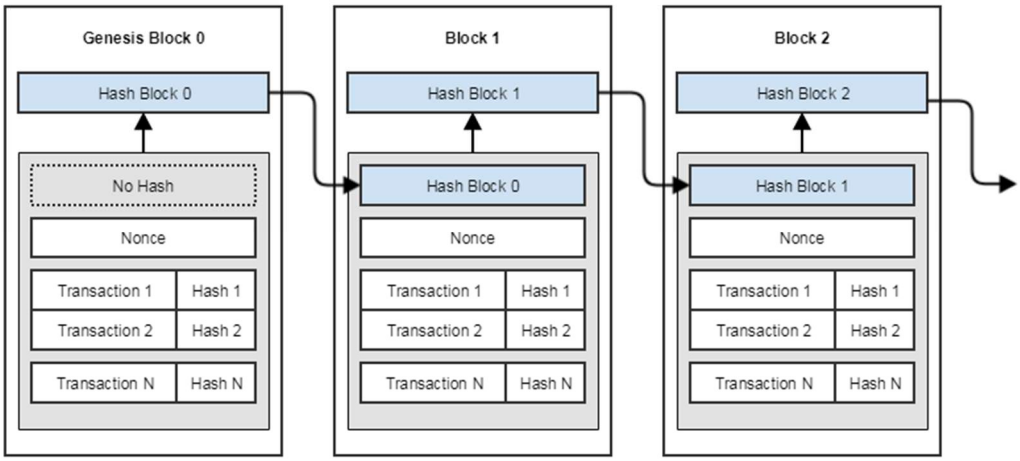

Рисунок 1.3 – Загальна схема архітектури блокчейну [2]

По своїй технології – це розподілена база даних, яка зберігається на багатьох комп'ютерах і містить записи про транзакції. Кожен блок у ланцюжку блоків містить хеш попереднього блока, що забезпечує безпеку та надійність системи. Заключення кількох транзакцій в один блок можливе завдяки функціоналу «batching», коли користувач може об'єднати купу окремих транзакцій в один блок, для подальшої обробки. При цьому комісія сплачується лише за сам блок, а не за кожну окрему операцію. При виконанні криптовалютного арбітражу, можна скористатися таким функціоналом для ефективності операцій: замість виконання ряду окремих операцій (що потребує комісії за кожну транзакцію), користувач може об'єднати всі свої операції до одного блоку та відправити його в блокчейн, що зменшить загальну комісію.

1.2 Порівняльний аналіз існуючих систем для автоматизації торгівлі

Існують різні програми та рішення для автоматизації торгів на криптовалютних біржах, такі як CryptoTrader, HaasBot, Gunbot, 3Commas та інші. Деякі з цих програм мають можливість автоматичного виконання замовлень на основі різних стратегій, таких як трейлінг-стоп, ліміт-ордери, розширені

замовлення та інші. Інші програми пропонують засоби для аналізу ринку та прийняття рішень, такі як показники технічного аналізу, індикатори, засоби для відстеження новин та інші.

Одним з переваг розробки власної системи подібного плану є можливість забезпечити швидкий та надійний функціонал, та впровадити лише необіхдні функції. Крім того, стек MERN (MongoDB, Express.js, React.js, Node.js) є високопродуктивним та широко використовується в сфері веб-розробки [3].

До інших переваг системи власної розробки можна віднести можливість розширення та модифікації системи, враховуючи специфіку кожної криптовалютної біржі. Крім того, використання Node.js дозволяє створювати потужні додатки та забезпечувати високу швидкість обробки даних.

Однак, запропоноване рішення не є бездоганним. Наприклад, необхідно забезпечити високу надійність та безпеку системи, оскільки торгівля криптовалютами пов'язана з високим ризиком. Також, необхідно враховувати велику кількість криптовалютних бірж та їх особливості, а це може вимагати значних зусиль при розробці та тестуванні системи.

При порівнянні з іншими програмами та рішеннями для автоматизації торгів на криптовалютних біржах, запропоноване вище рішення має свої переваги та недоліки. Наприклад, сторонні програми можуть мати більш розширені можливості для аналізу ринку, використовуючи різні алгоритми та індикатори. Однак, розробка власної системи може забезпечити високу продуктивність, надійність роботи системи та гнучкий підхід в плані реалізації необхідного функціоналу.

1.3 Огляд та вибір засобів розробки

Для створення автоматизованої системи торгівлі на криптовалютних біржах був проведений аналіз існуючих інструментів об’єкта проектування.

Для розробки програмного забезпечення була обрана мова програмування JavaScript. Вибір цієї мови обґрунтований тим, що вона є однією з

найпопулярніших мов програмування для веб-розробки, має велику кількість бібліотек та фреймворків, що значно полегшує розробку програмного забезпечення [4]. Порівняння JavaScript з іншими мовами наведено в таблиці 1.2.

Таблиця 1.2 – Порівняння мов для веб-розробки

	Переваги	Недоліки
JavaScript	Висока швидкість, можливість використовувати єдину мову на фронтенді та бекенді, широкий вибір бібліотек та фреймворків, активна спільнота розробників.	Відсутність компіляції може призвести до незвичайного поводження програм, немає стандартів на рівні мови, існують певні архітектурні обмеження.
Python	Легка читабельність коду, велика кількість бібліотек, можливість використовувати у різних областях, включаючи машинне навчання та науку даних.	Менша швидкість виконання порівняно з іншими мовами, менший вибір фреймворків порівняно з JavaScript.
Java	Велика швидкість виконання, сильна типізація, висока безпека, розширюваність.	Високий рівень складності, більш складне налаштування середовища розробки.
Ruby	Легкість написання коду, велика кількість бібліотек та фреймворків, приємна синтаксична структура.	Нижча швидкість виконання порівняно з іншими мовами, менший вибір рішень для великих підприємств.
PHP	Великий вибір фреймворків, підтримка великої кількості серверів та баз даних, легкість використання.	Нижча швидкість виконання порівняно з іншими мовами, невеликий вибір бібліотек порівняно з іншими мовами.

Отже, на основі проведеного аналізу було вирішено використовувати JavaScript та платформу Node.js [5]. Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою

JavaScript. Він буде використаний для серверної частини системи, оскільки він є досить ефективним та забезпечує зручний доступ до потрібних API бірж.

Для клієнтської частини проекту була використана бібліотека React.js, яка забезпечує зручну роботу зі створенням користувацьких інтерфейсів та дозволяє легко взаємодіяти з серверною частиною.

В якості бази даних буде використана MongoDB, яка є популярною документо-орієнтованою системою управління базами даних [6]. MongoDB забезпечує зручний доступ до даних та можливість простого масштабування [7]. Для забезпечення безпеки зберігання даних було вирішено використовувати шифрування та автентифікацію.

Для попереднього проектування користувацького інтерфейсу було використано Figma (рис. 1.4). Figma – це векторний графічний редактор та інструмент для дизайну інтерфейсів, що дозволяє створювати високоякісні макети та прототипи додатків та веб-сайтів.

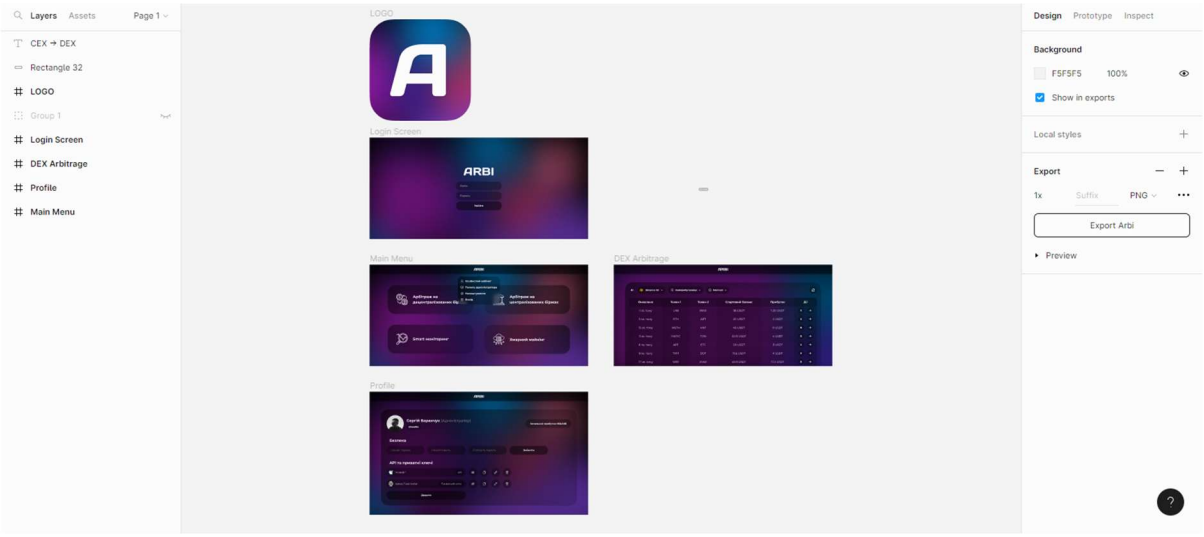

Рисунок 1.4 – Макет інтерфейсу системи

Для безпосередньої розробки програмного забезпечення буде використовуватися Webstorm (рис. 1.5). Webstorm – це інтегроване середовище розробки (IDE), що надає інструменти для розробки на JavaScript, TypeScript, Node.js, HTML, CSS та інших мовах, орієнтованих на веб-розробку. Завдяки своїм функціональним можливостям, Webstorm забезпечує зручну і продуктивну

розробку програмного забезпечення, дозволяючи швидко писати та налагоджувати код, використовуючи різні інструменти, такі як автодоповнення коду, перевірка налагодження, інтеграція з Git та багато іншого.

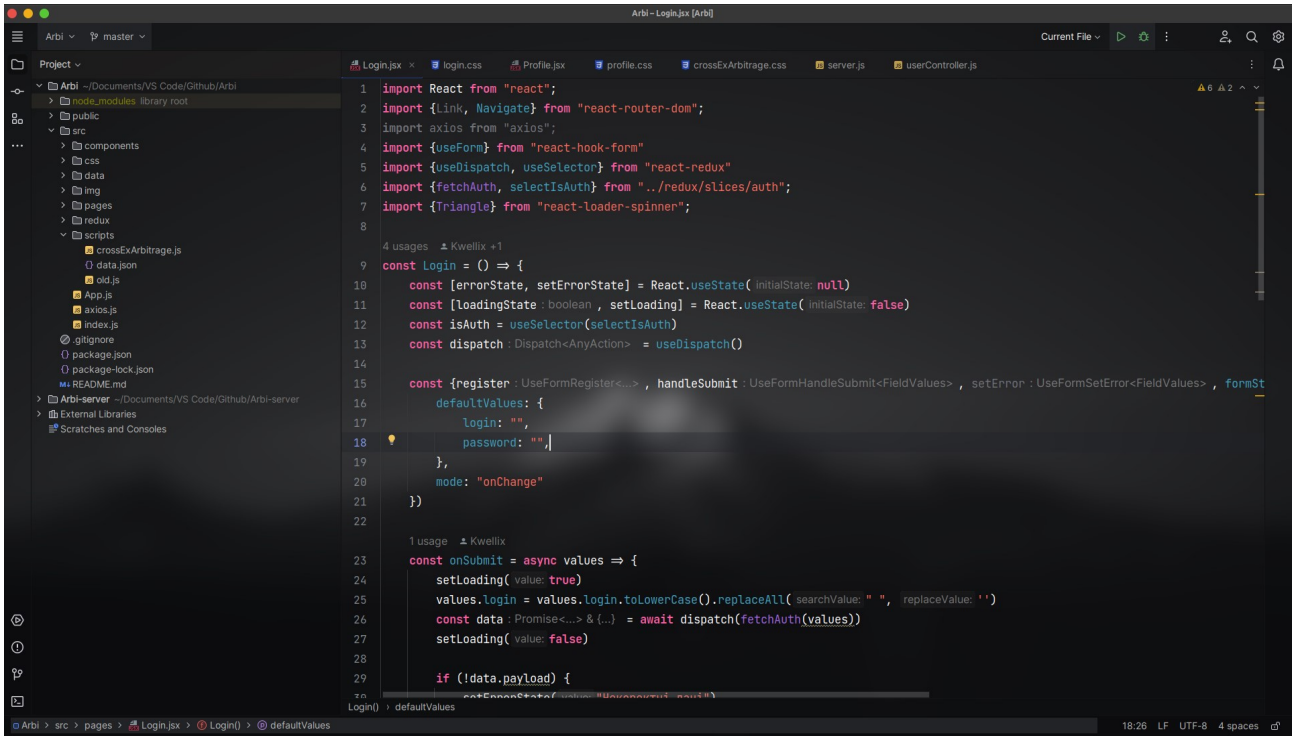

Рисунок 1.5 – JetBrains WebStorm

Для контролю версій веб-додатку під час розробки буде використовуватися Git. Git є одним з найпоширеніших та найефективніших інструментів для контролю версій програмного забезпечення. Він дозволяє зберігати історію змін файлів і проектів, здійснювати колаборацію між розробниками, керувати варіантами програмного коду та дізнаватись, хто вніс певні зміни. Git дозволяє легко повернутись до попередньої версії проекту, якщо потрібно відкотити зміни, а також зручно працювати з гілками (branches) та злиттями (merges) коду.

В якості сховища коду буде використаний GitHub, що дозволяє зберігати резервні копії проекту у хмарному сервісі.

Доступ до нього буде здійснюватися через офіційний десктопний клієнт під назвою GitHub Desktop (рис. 1.6)

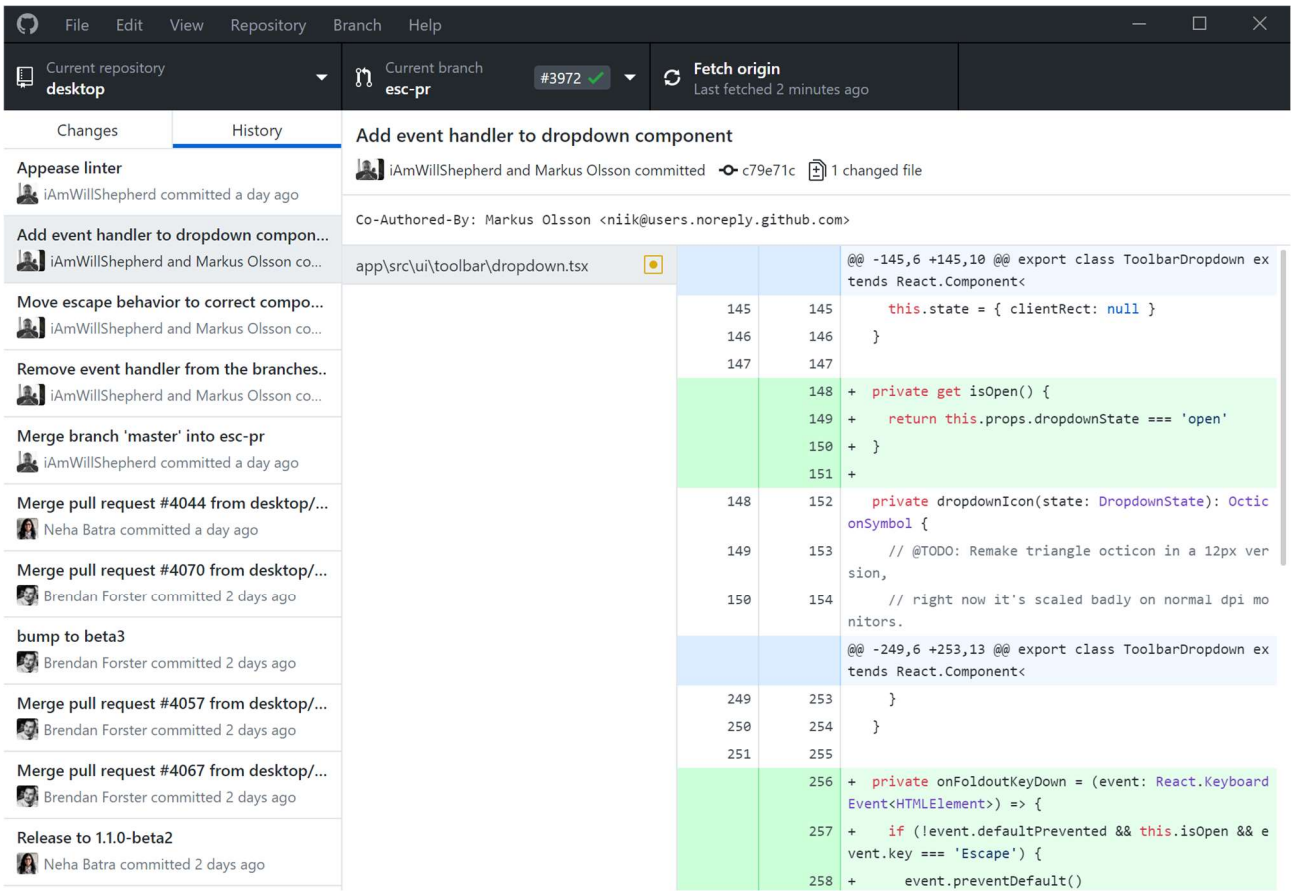

Рисунок 1.6 – Інтерфейс GitHub Desktop

Як наслідок, система Git використовується як засіб контролю версій проекту. Для інтеграції Git у проект, будуть використані інструменти, які дозволяють легко виконувати команди Git, наприклад, командний рядок та вбудовані у Webstorm інструменти.

РОЗДІЛ 2. ОГЛЯД СТРУКТУРИ ОБ'ЄКТУ ПРОЕКТУВАННЯ

2.1 Обґрунтування технологій і засобів вирішення поставленого завдання

Для розробки веб-додатку для автоматизованої торгівлі на криптовалютних біржах було обрано такі інструменти як React.js та платформу Node.js. Ці технології входять до складу стеку MERN, який є дуже сучасним та технологічним рішенням на сьогоднішній день, і дозволяє розробити веб-сайт або веб-додаток практично будь-якої складності [8].

Стек MERN – це стек технологій, який включає чотири основних компоненти: MongoDB (документ-орієнтована база даних), Express.js (фреймворк для розробки серверної частини застосунку), React.js (бібліотека створення користувацьх інтерфейсів (UI)), Node.js (платформа для розробки серверних додатків на JavaScript, що базується на двигуні V8 від Google) [9].

Разом ця комбінація дає можливості розробити повнофункціональний веб-застосунок з базою даних, API та клієнтською частиною на React, який дозволяє зручно та швидко розробляти клієнтську частину веб-додатку з використанням компонентного підходу [10]. Крім того, React дозволяє ефективно керувати станом додатку та оптимізувати його продуктивність [11].

Для забезпечення роботи серверної частини було вирішено використовувати Node.js, що дозволяє виконувати JavaScript на стороні серверу. Використання Node.js дозволяє зручно розробляти серверну логіку та забезпечувати взаємодію клієнтської та серверної частини.

Окрім того, для зберігання даних та забезпечення швидкого доступу до них було вирішено використовувати базу даних MongoDB. MongoDB є нереляційною (NoSQL) базою даних [12], що дозволяє зберігати дані у вигляді JSON-документів та швидко виконувати запити до цих даних [13].

2.2 Функціонально-структурна схема роботи системи

Для взаємодії з криптовалютними біржами, система буде використовувати засоби REST API. REST API (Representational State Transfer Application Programming Interface) – це архітектурний стиль для побудови веб-сервісів, який базується на протоколах HTTP і URI. REST API дозволяє здійснювати комунікацію між клієнтом та сервером за допомогою запитів HTTP методами GET, POST, PUT, DELETE та іншими. Кожен ресурс у REST API представлений у вигляді URI адреси, до якої можна звернутися для отримання або модифікації його стану. У результаті такої комунікації користувач може отримати необхідні дані безпосередньо через URL-адресу веб-сайту. Приклади таких запитів зображені на рисунках 2.1 та 2.2.

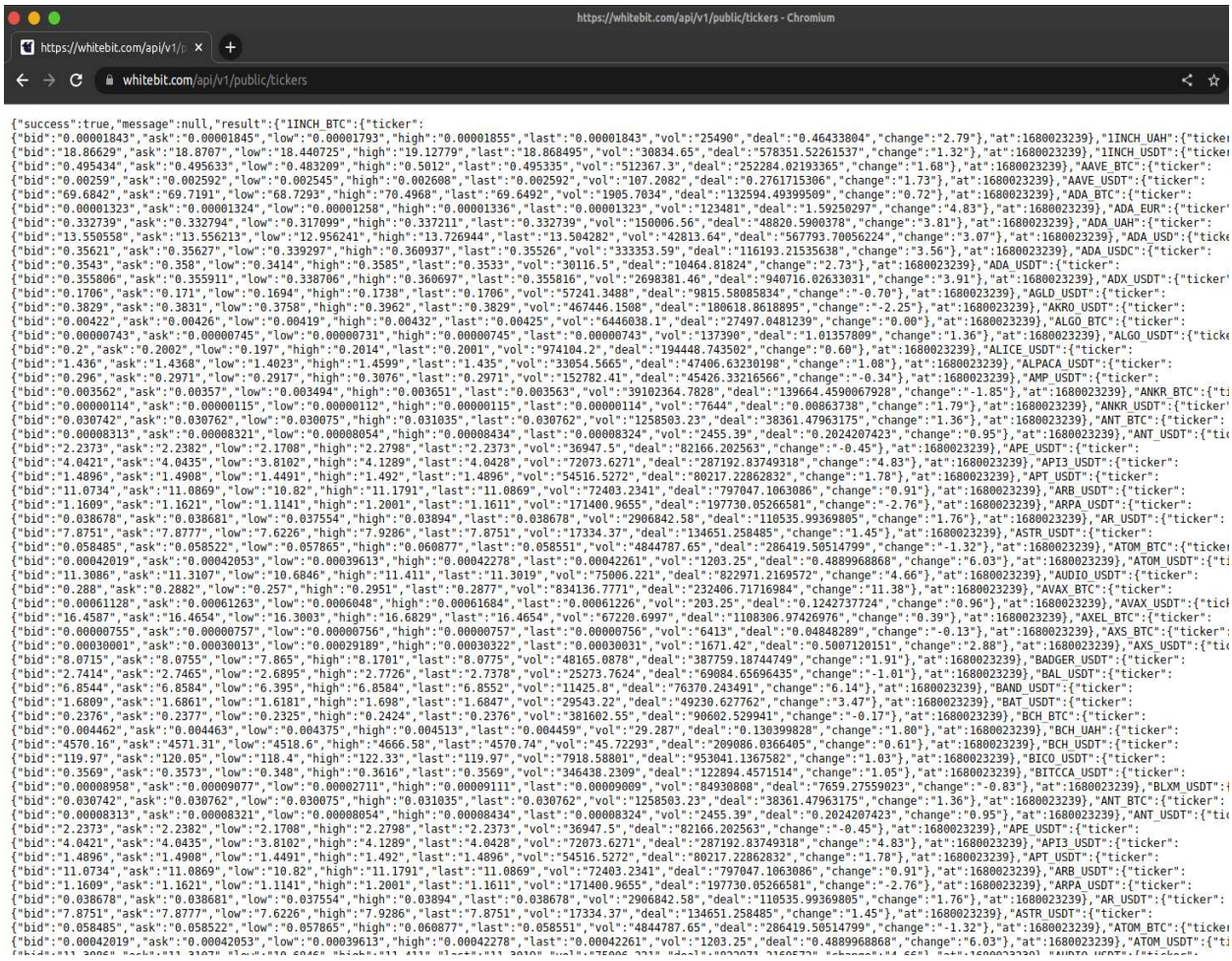

Рисунок 2.1 – HTTP запит до криптобіржі WhiteBIT для отримання ринкової інформації про ціни

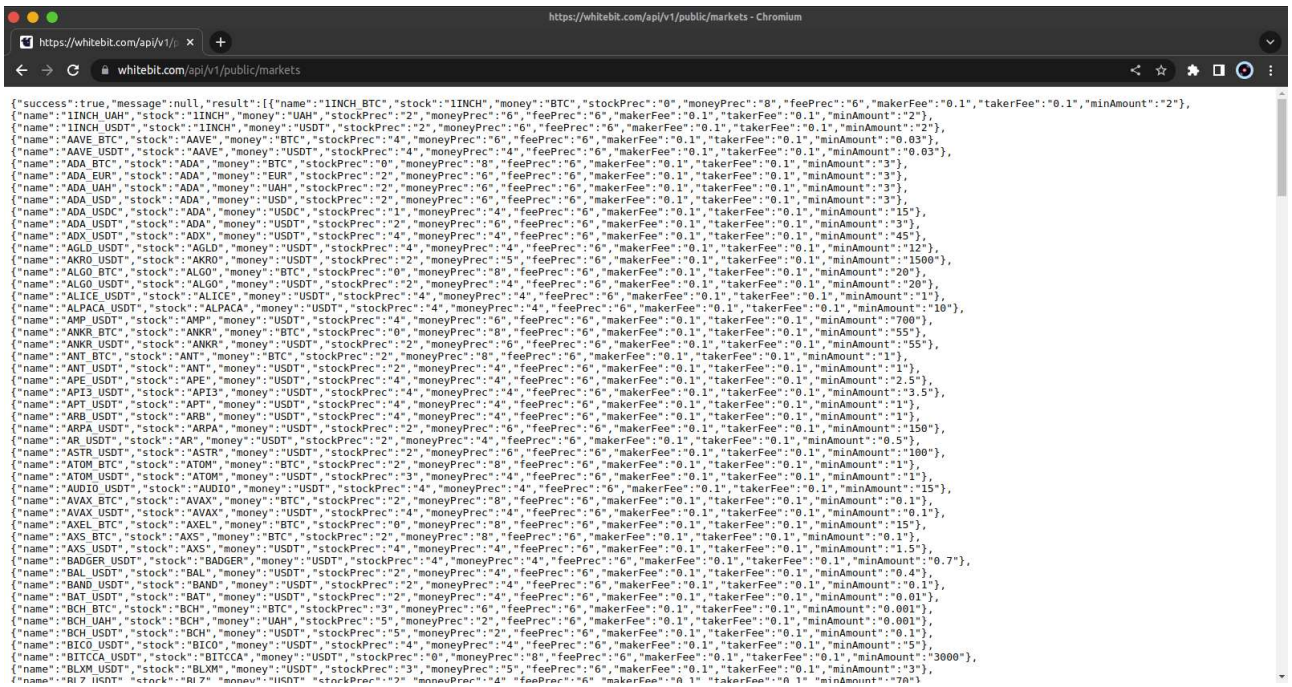

Рисунок 2.2 – HTTP запит до криптобіржі WhiteBIT для отримання списку ринків

Для виконання HTTP засобів використана бібліотека Axios. Axios – це бібліотека JavaScript, яка дозволяє виконувати HTTP-запити з браузера або з середовища Node.js [14]. Вона базується на промісах (асинхронне програмування) і надає зручний і простий інтерфейс для виконання запитів до веб-сервера [15].

Axios дозволяє виконувати різні типи запитів, включаючи GET, POST, PUT, DELETE, PATCH та інші. Вона також дозволяє використовувати різні типи даних, такі як JSON, XML та інші.

Axios дозволяє використовувати інтерсептори для перехоплення і зміни запитів та відповідей перед їх відправленням або після отримання. Вона також підтримує скачування та відвантаження файлів.

Axios є досить популярною бібліотекою в JavaScript-спільноті і використовується для розробки клієнтських і серверних додатків, які потребують взаємодії з веб-сервером.

Асинхронне програмування – це підхід до розробки програм, в якому виконання завдань не блокує потік виконання основної програми. У JavaScript

асинхронність зазвичай досягається за допомогою колбек-функцій, промісів та async/await. Колбек-функції передаються як параметри у функцію і викликаються пізніше, коли операція буде завершена. Проміси є об'єктами, яким можна призначити два стани: успіх (resolve) або помилку (reject), і на них можна повісити обробники .then() та .catch(). Async/await – це синтаксичний цукор для промісів, що дозволяє писати код з очікуванням результату асинхронної операції без глибокого вкладення колбек-функцій.

Promise.all – це метод в JavaScript, який приймає масив промісів і повертає новий проміс. Цей новий проміс буде успішним тоді, коли всі передані в Promise.all промиси будуть успішно виконано. Якщо хоча б один з переданих промисів завершиться помилкою, то повернеться помилковий результат. Результат Promise.all буде масив з результатами кожного переданого промису у порядку їх передачі до методу. Використовуючи Promise.all можна ефективно обробляти багато асинхронних операцій паралельно та чекати на їх завершення без блокування потоку виконання основної програми.

Під час розробки, Promise.all був використаний у блоці коду, що представлений на рисунках 2.3 та 2.4. Цей код одночасно надсилає HTTP запити до обраних бірж.

```
export async function getAllTickersObject(selectedExchanges) {
    let dataObject = {}

    const CEXesList = {
        WhiteBIT_: WhiteBIT,
        Huobi_: Huobi,
        Bitrue_: Bitrue,
        Binance_: Binance,
        OKX_: OKX,
        Gate_io_: Gate_io,
        Bybit_: Bybit,
        Kucoin_: Kucoin
    }

    //Створюємо масив класів обраних бірж
    // eslint-disable-next-line
    let exchangeObjects = Object.entries(CEXesList).map(exchange => {
```

Рисунок 2.3 – Лістинг коду, що відповідає за розсилку HTTP запитів (1/2)

```
        let exchangeName = exchange[0].slice(0, -1)
        if (selectedExchanges[exchangeName] === true) {
            return new (CEXesList[exchange[0]])()
        }
    }).filter(exchange => {
        return exchange !== undefined;
    });

    //Надсилаємо запити .getData() для кожного класу біржі
    //та одразу записуємо отримані дані в об'єкт dataObject
    await Promise.all(exchangeObjects.map(exchangeObj => {
        return exchangeObj.getData()
    })).then(() => {
        exchangeObjects.forEach(exchange => {
            dataObject[exchange.cexName] = exchange.tickers
        })
    })

    return dataObject
}
```

Рисунок 2.4 – Лістинг коду, що відповідає за розсилку HTTP запитів (2/2)

Даний код є функцією getAllTickersObject, яка є асинхронною і повертає об'єкт з даними про цінові дані усіх криптовалют, що розміщені на обраних криптобіржах. Передається параметр selectedExchanges, який містить об'єкт з власне, обраними біржами.

У функції створюється порожній об'єкт dataObject [16]. Далі створюється об'єкт CEXesList, який містить класи бірж. Шаблон класу біржі представлений на рисунках 2.5 та 2.6.

```
class WhiteBIT {
    constructor() {
        this.symbols = []
        this.tickers = []
        this.cexName = "WhiteBIT"
    }

    async getData() {
        try {
            const [marketsData, tickersData] = await Promise.all([
                axios.get("https://whitebit.com/api/v1/public/markets"),
```

Рисунок 2.5 – Лістинг коду з шаблоном класу біржі (1/2)

```
                axios.get("https://whitebit.com/api/v1/public/tickers")
            ])

            this.symbols = this.getMarkets(marketsData.data)
            this.tickers = this.getTickers(tickersData.data)
            console.log("WHITEBIT ✓")
        } catch (err) {
        }
    }

    getMarkets(data) {
        let market_symbols = []
        for (let el of data["result"]) {
            market_symbols.push(el["name"])
        }
        return market_symbols
    }

    getTickers(dataRaw) {
        let data = dataRaw["result"]
        let tickersDict = {}
        for (const [key, value] of Object.entries(data)) {
            tickersDict[key] = {
                "ask": +value["ticker"]["ask"],
                "bid": +value["ticker"]["bid"]
            }
        }
        return tickersDict
    }
}
```

Рисунок 2.6 – Лістинг коду з шаблоном класу біржі (2/2)

Далі створюється масив exchangeObjects, який містить об'єкти бірж, обрані користувачем. Використовується метод Object.entries() для отримання масиву пар ключ-значення CEXesList, який далі обрізається за допомогою методу slice() із ціллю відокремити назву біржі від підкресленого символу.

Далі виконується фільтрування за допомогою методу filter(), який видаляє undefined значення, якщо біржа не була обрана користувачем.

Потім використовується метод Promise.all(), який одночасно надсилає запити на отримання даних з кожної біржі за допомогою методу getData() (рис. 2.7). Результатом методу Promise.all() є масив результатів запитів з кожної біржі.

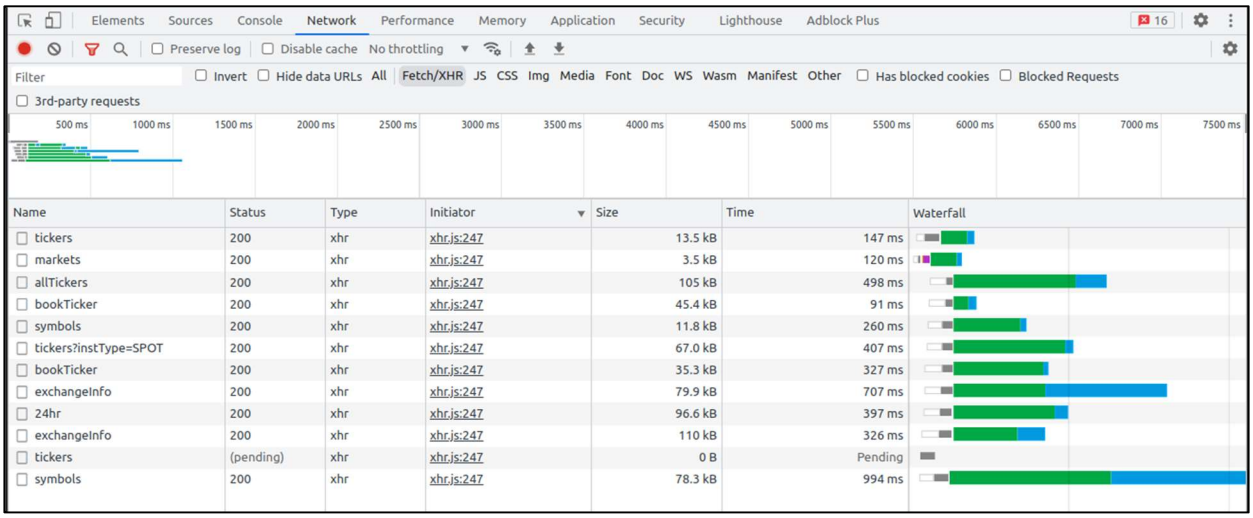

Рисунок 2.7 – Монітор HTTP запитів

Далі виконується метод forEach(), який записує дані з кожної біржі до об'єкту dataObject. Нарешті, повертається об'єкт dataObject з даними про повну ринкову інформацію на обраних біржах. У цілому, код використовує класи бірж та Promise для асинхронного виконання запитів на отримання даних з кожної біржі та записує результати до об'єкту dataObject.

2.3 Створення моделі бази даних

В якості бази даних використовується MongoDB. MongoDB – це документ-орієнтована база даних, яка зберігає дані у вигляді документів, що нагадують об'єкти JavaScript. MongoDB є одним з найпопулярніших рішень NoSQL-баз даних [17].

MongoDB зберігає дані у вигляді JSON-подібних документів з динамічною схемою [18]. У MongoDB використовуються колекції замість таблиць і документи замість рядків записів. MongoDB дозволяє розподіляти дані на кілька серверів та виконувати операції на кластері з декількох машин [19].

У JavaScript робота з цією БД відбувається за допомогою Mongoose. Mongoose – це бібліотека для Node.js, яка дозволяє простіше взаємодіяти з базою даних MongoDB [20]. Вона надає більш високорівневий інтерфейс для взаємодії з MongoDB, який полегшує створення, збереження та отримання даних з бази

даних. Mongoose надає можливість описувати моделі даних зі специфікацією їх структури та валідацією даних, підключатися до бази даних з використанням різних протоколів, таких як MongoDB, MongoDB Atlas, mLab, або Heroku, та багато інших можливостей [21].

Основні переваги використання Mongoose [22]:

– Структурованість: можна визначити модель даних, що дозволяє зберігати та валідувати дані за відповідною схемою.

– Валідація даних: можна визначити властивості обов'язкових та допустимих значень полів, обмеження довжини та формату.

– Легкість взаємодії з MongoDB: Mongoose дозволяє легко виконувати запити до бази даних MongoDB за допомогою Mongoose API.

– Спрощення роботи з базою даних: Mongoose використовується для збереження, оновлення, видалення та пошуку даних в базі даних MongoDB.

Код, який створює модель користувача у базі даних, представлений на рисунку 2.8.

```
const UserSchema = new mongoose.Schema({
    fullName: {
        type: String,
        required: true,
    },
    login: {
        type: String,
        required: true,
        unique: true
    },
    passwordHash: {
        type: String,
        required: true
    },
    userPhoto: String,
    role: {
        type: String,
        required: true
    },

}, {
    timestamps: true,
})
```

Рисунок 2.8 – Лістинг коду для створення моделі користувача в БД

База даних, в якій знаходитиметься дана модель, міститиме одну таблицю «users», яка зберігатиме дані про користувачів системи. Ця таблиця буде містити наступні поля:

– id (тип: ObjectId): унікальний ідентифікатор користувача, який автоматично генерується системою.

– fullName (тип: String): повне ім'я користувача.

– login (тип: String): логін користувача, який використовується для входу в систему.

– passwordHash (тип: String): хеш пароля користувача.

– userPhoto (тип: String): файл base64 з зображенням користувача.

– role (тип: String): роль користувача в системі.

За замовчуванням Mongoose додає поле «id» до всіх моделей даних.

Фізична (рис. 2.9), логічна (рис. 2.10) та інформаційна моделі бази даних будуть мати таблицю "users", яка міститиме стовпці «id», «fullName», «login», «passwordHash», «userPhoto» і «role». Поле «id» буде ключем таблиці. Поле «login» буде встановлено як унікальне поле, щоб запобігти дублюванню логінів користувачів.

Також будуть присутні поля createdAt та updatedAt, які генеруються автоматично за допомогою параметра { timestamps: true } у схемі моделі.

Рисунок 2.9 – Фізична таблиця БД

Рисунок 2.10 – Логічна схема БД

Також, в елементі БД присутній рядок __v. Це ще один автоматично генерований Mongoose рядок, в якому зазначається versionKey. «versionKey» – це властивість, яка задається для кожного документа під час його створення за допомогою Mongoose. Значення цього ключа містить внутрішню ревізію документа. Назва цієї властивості документу може бути налаштована. За замовчуванням вона має назву __v.

На даний момент, БД використовується лише для забезпечення процесу авторизації користувачів. Але, завдяки особливостям MongoDB, модель користувача може бути легко масштабована та доповнена, наприклад, якщо буде необхідно зберігати дані користувача про транзакції, кількість зароблених грошей та ін. Приклад потенційно доповненої моделі БД зображено на рисунку 2.11.

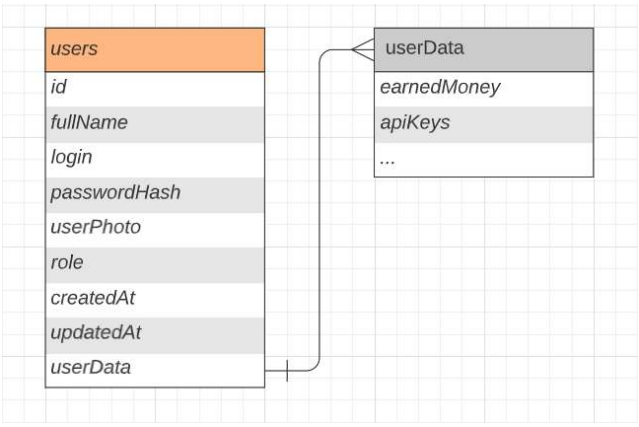

Рисунок 2.11 – Доповнена логічна модель БД

РОЗДІЛ 3. РОЗРОБКА АВТОМАТИЗОВАНОЇ КРИПТОВАЛЮТНОЇ ВЕБ-СИСТЕМИ

3.1 Практична реалізація об’єкта розробки. Створення інтерфейсу користувача

Перед повноцінною розробкою необхідно комплексно оцінити список доступних для цього засобів. Для цього, на рисунку 3.1 наведена контекстна діаграма процесу створення автоматизованої системи торгівлі.

Контекстна діаграма – це вид діаграми, який використовується для моделювання системи з точки зору її взаємодії з іншими системами або окремими користувачами. Вона показує основні компоненти системи та їх взаємозв'язки з іншими сутностями у контексті більш широкого середовища.

Рисунок 3.1 – Контекстна діаграма процесу створення автоматизованої системи торгівлі

Основна архітектура створеної системи включає в себе 3 шари:

– шар подання даних в клієнтській частині;

– сервер додатку;

– шар, що керує ресурсами у базі даних.

Така архітектура може збільшуватися до багатошарової, якщо до неї додати ще додаткові сервери або інші об’єкти взаємодії. На рисунку 3.2 зображена трьох-шарова архітектура веб–додатку, з додатковим шаром, в який входять зовнішні джерела, з якими він взаємодіє.

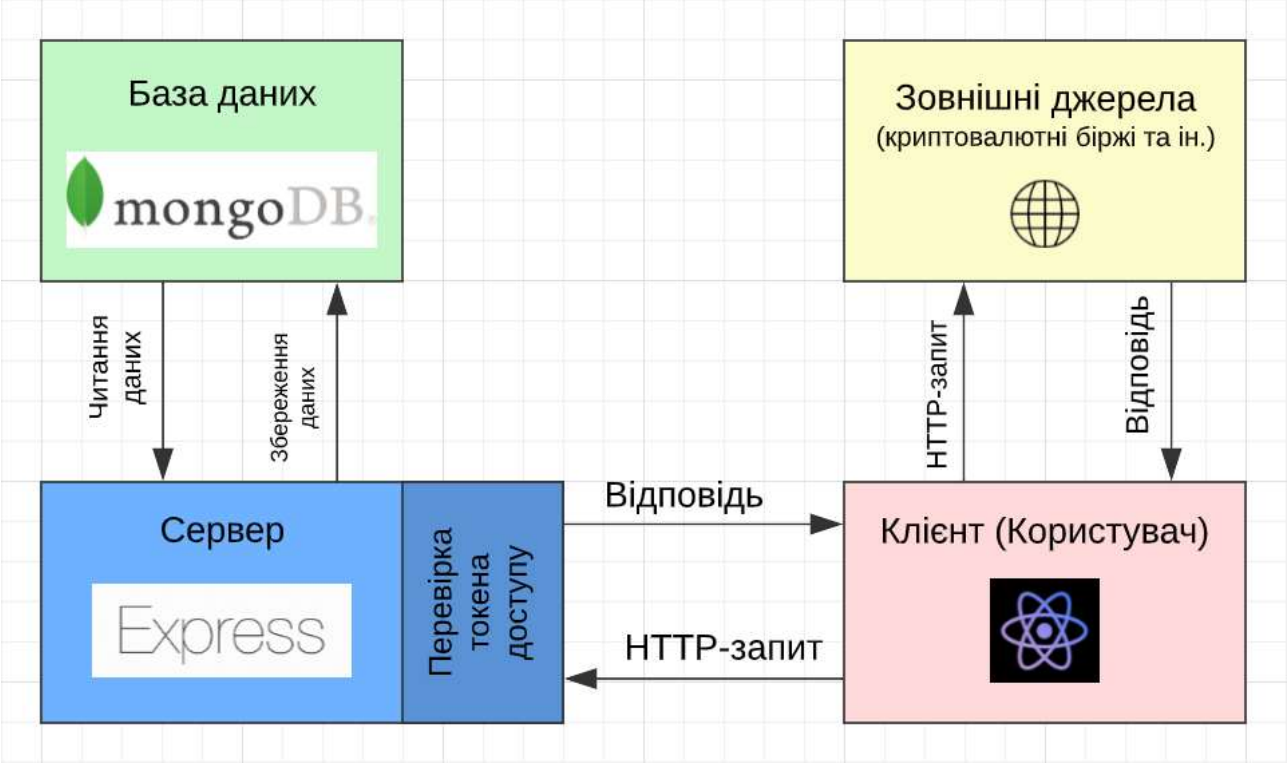

Рисунок 3.2 – Архітектура веб-додатку

Така архітектура може збільшуватися до багатошарової, якщо до неї додати ще додаткові сервери або інші об’єкти взаємодії. На рисунку 3.2 зображена трьох-шарова архітектура веб–додатку, з додатковим шаром, в який входять зовнішні джерела, з якими він взаємодіє.

Під час того, як система отримує дані з бірж за допомогою коду, який описаний у розділі 2 підрозділі 2.2, отримані дані конвертуються, що привести їх до єдиного вигляду. Методи перетворень викликаються у методі getData(),

після отримання результату HTTP запиту. Алгоритми перетворень створені індивідуально для кожної біржі, але всі вони в результаті приводять дані про список ринків до єдиного вигляду, що зображений на рисунку 3.3, а про ціни на криптовалюти вигляду, що зображений на рисунку 3.4.

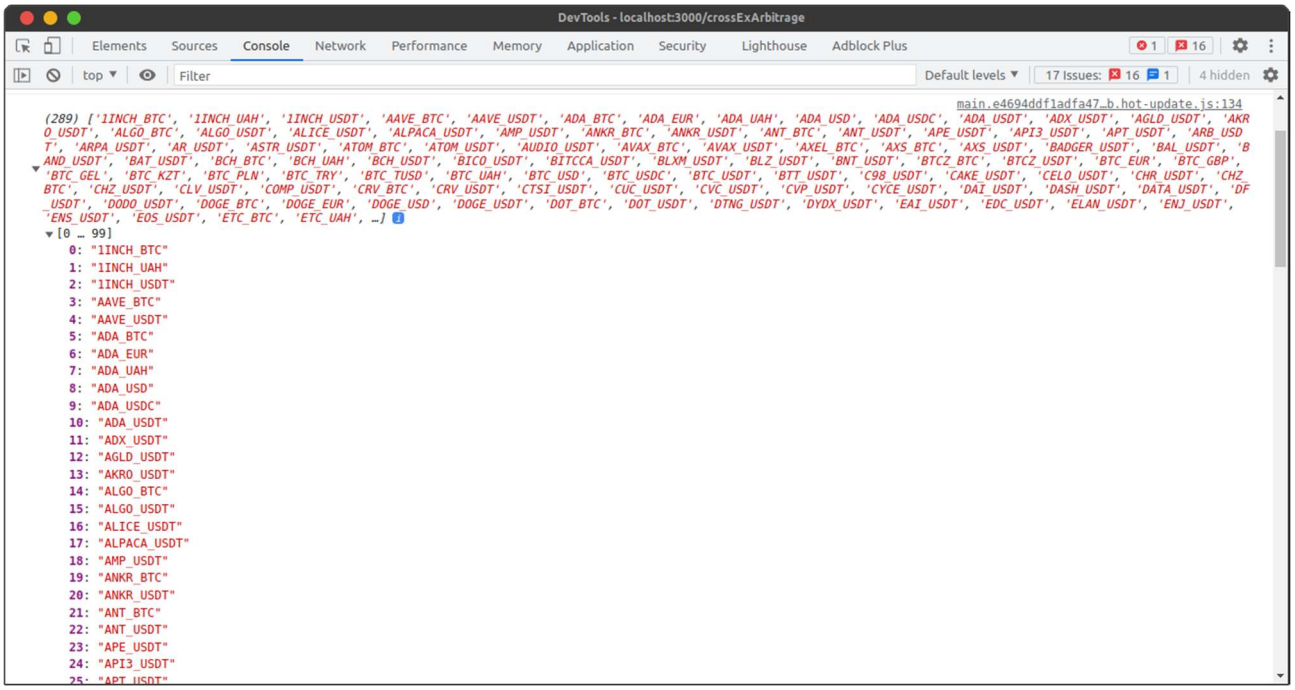

Рисунок 3.3 – Формат даних про список доступних ринків

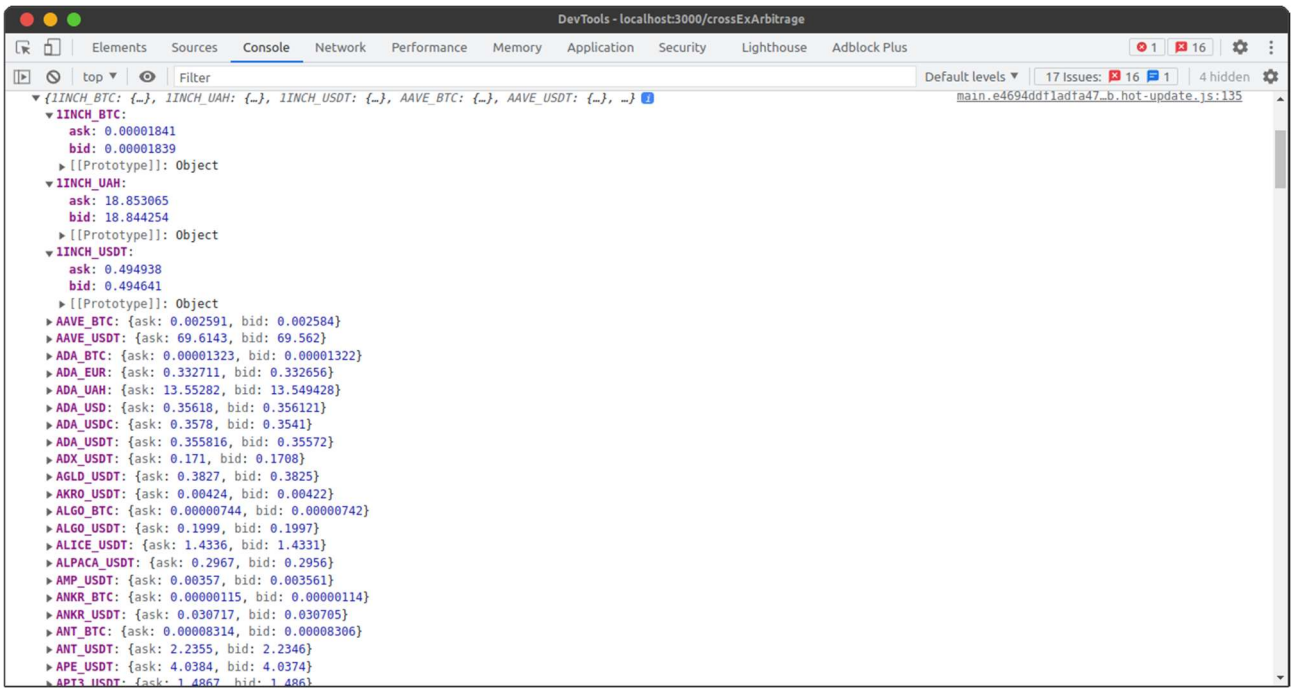

Рисунок 3.4 – Формат даних про ціни на криптовалюти

Ціни на криптовалюти отримуються у вигляді пари Ask і Bid. Ask і Bid – це терміни, які використовуються на фіатових ринках для позначення цін купівлі та продажу активу. Bid (бід) – це максимальна ціна, за яку хтось готовий купити актив. Ask (аск) – це мінімальна ціна, за яку хтось готовий продати актив. Розрив між ask і bid називається спредом і є одним з основних показників легкості обміну на ринку.

Після того, як усі дані було отримано та оброблено, вони надсилаються у функцію getChains(data, deadFilter), яка власне шукає і створює арбітражні зв’язки. Вона приймає два аргументи – об'єкт data та об'єкт deadFilter. Функція шукає можливі арбітражні можливості між криптобіржами на основі даних з об'єкту data та використовує об'єкт deadFilter для фільтрації некоректних даних, які інколи надсилають криптобіржі.

Для кожного ринку зі списку marketsList функція пробігається по всіх криптобіржах з об'єкту data та знаходить найкращу пропозицію на купівлю та продаж для поточного активу. Якщо найкраща ціна на продаж вища за найкращу ціну на купівлю, то функція додає дані про можливу арбітражну угоду до масиву arbitrageData. Дані про угоду складаються з назви ринку, назв криптобірж, найкращих цін на купівлю та продаж, різниці між цінами та потенційного прибутку від угоди. На виході функція повертає масив arbitrageData, який містить дані про всі знайдені можливості для арбітражу.

Для створення компонентів, у React використовується JSX. JSX – це розширення синтаксису JavaScript, яке дозволяє використовувати теги HTML в JavaScript коді для створення інтерфейсів користувача [23]. Це дозволяє розробникам використовувати знайомий синтаксис HTML для створення елементів інтерфейсу, які можуть містити вбудований JavaScript код для динамічних функцій та обробки подій.

JSX є частиною бібліотеки React, яка використовує її для створення інтерфейсів користувача веб-додатків. JSX код компілюється в звичайний JavaScript код, який виконується в браузері або на сервері [24].

JSX-структура головного компоненту App, який відповідає за відображення розробленого React-проекту, представлена на рисунку 3.5.

```
function App() {
    const dispatch = useDispatch()
    const isAuth = useSelector(selectIsAuth)

    React.useEffect(() => {
        dispatch(fetchAuthMe())
    }, [])

    if (!isAuth) {
        return (
            <div className="app">
                <Routes>
                    <Route path="/" element={<HomePage/>}/>
                    <Route path="/login" element={<Login/>}/>
                </Routes>
            </div>
        )
    }

    return (
        <div className="app">
            <Routes>
                <Route path="/" element={<HomePage/>}/>
                <Route path="/login" element={<Login/>}/>
                <Route path="/profile" element={<Profile/>}/>
                <Route path="/adminpanel" element={<AdminPanel/>}/>
                <Route path="/settings" element={<Settings/>}/>
                <Route path="/cexArbitrage" element={<CEXArbitrage/>}/>
                <Route path="/crossExArbitrage" element={<CrossExArbitragePage/>}/>
            </Routes>
        </div>
    )
}

export default App;
```

Рисунок 3.5 – JSX- код компоненту App

Цей код – це функціональний компонент React з використанням хуків useDispatch та useSelector. Функція useDispatch дозволяє компоненту відправляти дії до Redux store, а функція useSelector дозволяє компоненту витягувати дані з Redux store. Компонент також використовує хук useEffect, щоб відправити запит на сервер при завантаженні компонента.

Перш за все, в компоненті App встановлюється зв'язок між компонентом і Redux store через використання хука useSelector. Змінна isAuth містить значення true, якщо користувач автентифікувався, і false в іншому випадку.

У компоненті App використовується хук useEffect з порожнім масивом залежностей, тому що ми хочемо відправити запит на сервер лише один раз, коли компонент завантажується. Це досягається за допомогою функції dispatch, яка відправляє дію fetchAuthMe до Redux store. Ця дія запитує інформацію про автентифікацію користувача на сервері.

Якщо значення isAuth дорівнює false, то компонент повертає JSX код з двома маршрутами: / і /login. Якщо значення isAuth дорівнює true, то компонент повертає JSX код з декількома маршрутами: /, /login, /profile, /adminpanel, /settings, /cexArbitrage та /crossExArbitrage.

У цьому коді використовується також компонент Routes з бібліотеки react-router-dom, який дозволяє відображати різний вміст на основі шляху URL. Компонент Route визначає маршрути, які пов'язані з певним вмістом, що відображається на сторінці.

Додатково, для зручності використання, необхідно забезпечити коректне використання веб-додатку на будь-якому пристрої. Для цього необхідно забезпечити адаптивний веб-інтерфейс.

Існують наступні основні способи створення адаптивного веб-інтерфейсу за допомогою CSS:

– використання медіа-запитів (media queries) для зміни стилів на різних екранах.

– використання властивостей, які дозволяють пропорційно масштабувати елементи (наприклад, width: 100%; height: auto;).

– використання флексбоксів та грід систем для автоматичного розташування елементів на сторінці.

– використання rem одиниць виміру замість px для задавання розміру шрифту та інших параметрів, що дає можливість браузерам коректно масштабуватися при змінах розміру екрана.

Цими методами можна досягти адаптивності сайту до будь-яких типів пристроїв та екранів будь-яких роздільних здатностей. Під час розробки було передбачено адаптивність сайту для десктопних та мобільних пристроїв [25].

3.2 Принципи безпеки у розробленій системі

Основними способами захисту даних у створеній системі виступають авторизація та шифрування даних. Принцип авторизації у MERN веб-додатках полягає в тому, щоб перевірити, чи має користувач право доступу до певної сторінки або ресурсу. Для цього зазвичай використовуються токени доступу (access tokens), які генеруються при успішному вході користувача на сайт.

При кожному запиті до сервера, клієнт надсилає свій access token разом із запитом. Сервер перевіряє цей токен і дозволяє або забороняє доступ до ресурсу залежно від його допустимості.

Для реалізації авторизації у MERN стеку можна скористатись багатьма бекенд фреймворками, такими як Passport.js або JSON Web Token (JWT) (використовується у даній системі).

Для JWT необхідно підписати на сервері дані ключем і передавати кліентам отриманий токен. Кліент повинен буде надсилати цей токен при кожному запиті на сервер для отримання потрібних даних.

Перевірка наявності доступу у користувача, що надсилає HTTP запит, відбувається на сервері, за допомогою Middleware (додаток А).

Middleware – це функція, яка оброблює запити до сервера перед тим, як вони дістаються до маршрутів (routes) або інших обробників запиту. Middleware може виконувати різноманітні завдання, такі як перевірка авторизації користувача, логування запитів або зміна формату даних.

Middleware можуть бути глобальними для всього додатку або специфічними для окремих маршрутів. В MERN стеку Middleware часто використовуються для перевірки доступності ресурсу певного користувача за його токеном доступу. Приклад застосування Middleware представлений на рисунку 3.6.

```
app.use('/uploads', express.static('uploads'));
app.post("/upload", checkAuth, upload.single('image'), (req, res) => {
    res.json({
        url: `/uploads/${req.file.originalname}`,
    })
})

app.post("/auth/login", userController.login)
app.post("/auth/register", registerValidation, userController.register)
app.get("/auth/me", checkAuth, userController.authMe)
app.post("/update/userData", checkAuth, userController.updateData)
```

Рисунок 3.6 – Основні backend-методи

У даному коді, в якості Middleware виступає функція checkAuth(), яка перед обробкою запиту перевіряє токен доступу користувача, що надсилає запит.

Як альтернативний варіант, у Express.js Middleware можна створювати також за допомогою методу use(). Приклад представлений на рисунку 3.7.

```
app.use((req, res, next) => {
  // код middleware
  next();
});
```

Рисунок 3.7 – Middleware, створений за допомогою методу use()

У даному прикладі app.use() встановлює новий Middleware на всьому додатку. Функція-обробник отримає параметри req (запит), res (відповідь) та next, що дозволить продовжити ланцюг обробки запитів.

3.3 Тестування та відлагодження об’єкта під час розробки

Існує декілька підходів до тестування React додатків, але основні методи тестування включають:

– unit-тестування компонентів: це тестування окремих компонентів без залежностей від інших компонентів або залежностей від зовнішніх сервісів. Для цього використовуються тестувальні фреймворки, такі як Jest або Mocha, які дозволяють писати тестові сценарії на JavaScript.

– integration-тестування компонентів: це тестування взаємодії компонентів між собою та з іншими частинами додатку, такими як API або база даних. Для цього можна використовувати фреймворки, такі як Enzyme або React Testing Library, які дозволяють робити запити до віртуального DOM та перевіряти поведінку компонентів.

– end-to-end тестування: це означає тестування додатку в цілому, включаючи взаємодію з реальними користувачами та залежностями, такими як база даних та сервер. Для цього використовуються фреймворки, такі як Cypress або Selenium, які дозволяють створювати автоматизовані тестові сценарії та запускати їх у браузері.

– snapshot-тестування: це означає збереження стану віртуального DOM компонентів та порівняння його з попередньо збереженим станом для перевірки наявності змін. Для цього використовується фреймворк Jest, який дозволяє зберігати та перевіряти знімки віртуального DOM.

Крім цих методів, також можуть бути використані інші техніки тестування, такі як тестування функціональності за допомогою моків, фікстур, тестування відповідності до стандартів доступності, тестування швидкості та продуктивності додатку, тестування взаємодії з різними пристроями та браузерами тощо.

При тестуванні React додатків важливо враховувати специфіку фреймворку та особливості взаємодії компонентів між собою та з іншими частинами додатку. Для ефективного тестування необхідно використовувати тестові фреймворки та бібліотеки, які забезпечують швидке та точне тестування, а також допомагають забезпечити покриття коду тестами.

Тестування React додатків допомагає забезпечити якість та стабільність додатку, зменшити кількість помилок та забезпечити легкість розробки та підтримки.

Загалом, будь яке тестування або відлагодження починається з запуску додатку на локальному сервері. Конкретно в цьому випадку, необхідно окремо

запустити Frontend (React) частину (рис. 3.8), та Backend (Node.js/Express) частину (рис. 3.9).

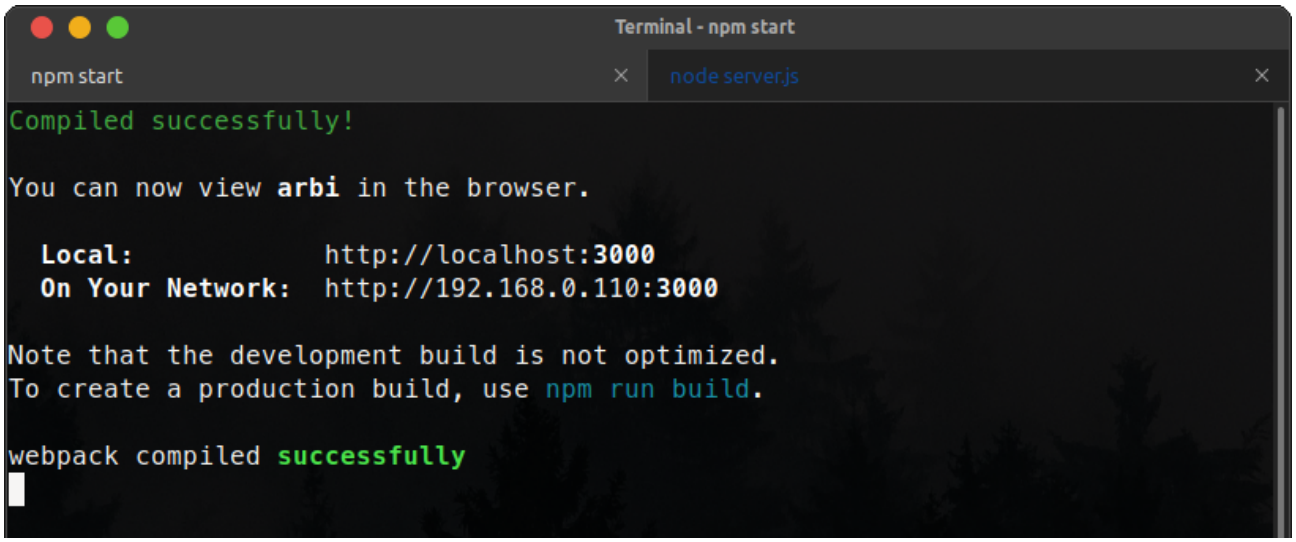

Рисунок 3.8 – Запуск Frontend складової веб-додатку, за допомогою команди «npm start»

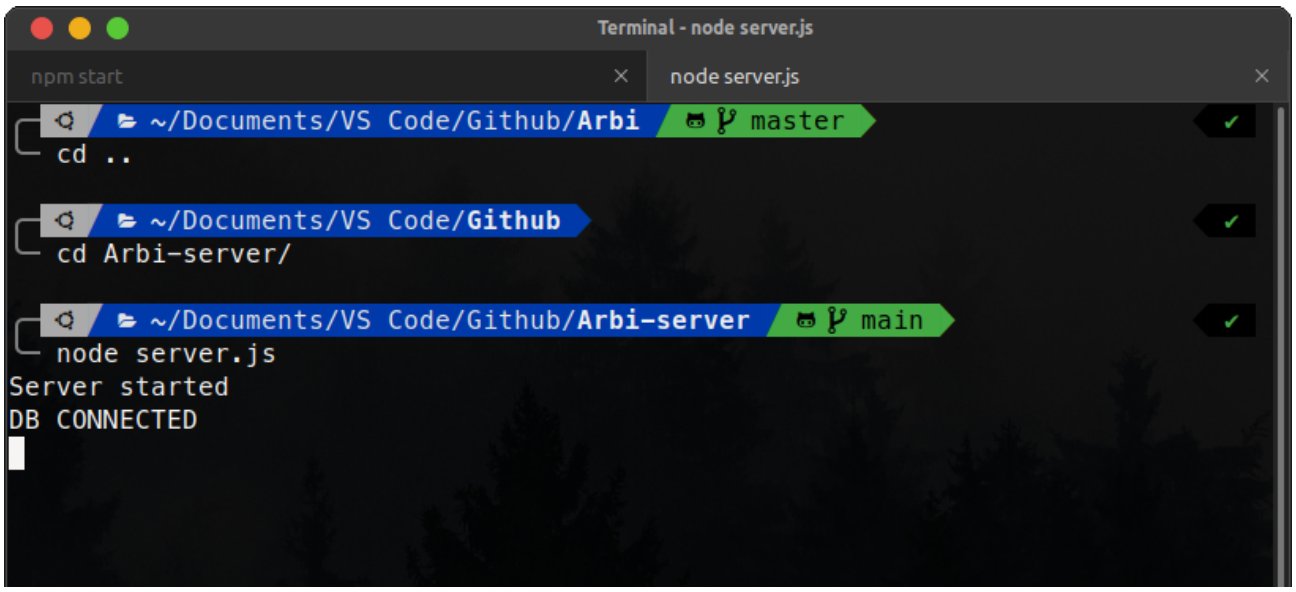

Рисунок 3.9 – Запуск Backend складової веб-додатку, за допомогою команди «node server.js»

Після даних дій, проекти запустяться на локальному сервері розробника, на попередньо заданих, або автоматично визначених портах. Ще дасть можливість вести розробку та налагодження в режимі реально часу, де при

внесенні змін у код, зміст сторінки автоматично оновлюватиметься в режимі реального часу.

3.4 Використання розробленої веб-системи для автоматизації торгівлі

Так як розроблена система передбачає обмежений доступ (лише для визначеного кругу осіб), то традиційна система реєстрації тут відсутня, а доступ надається обраним користувачам вручну.

При відкритті сайту, користувача зустріне вікно авторизації (лістинг коду цієї сторінки представлений у додатку Б) (рис. 3.10).

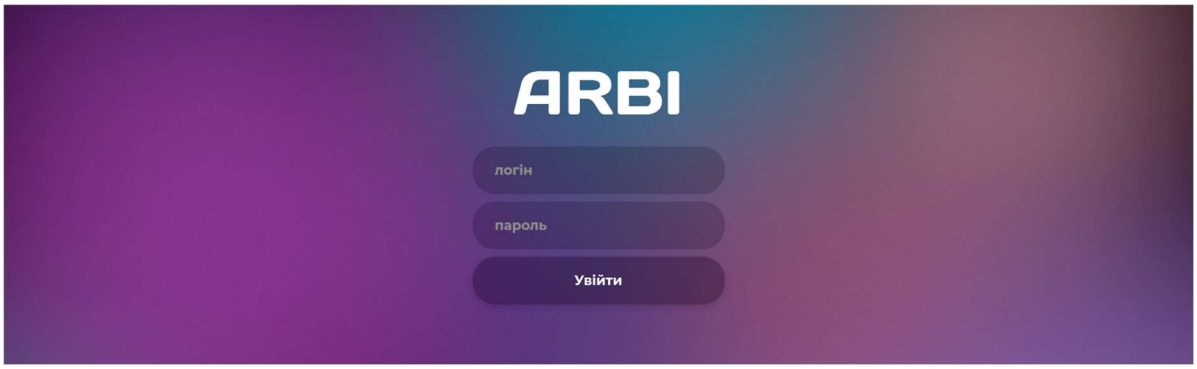

Рисунок 3.10 – Вікно авторизації

Після авторизації, користувач попадає у головне меню (рис. 3.11).

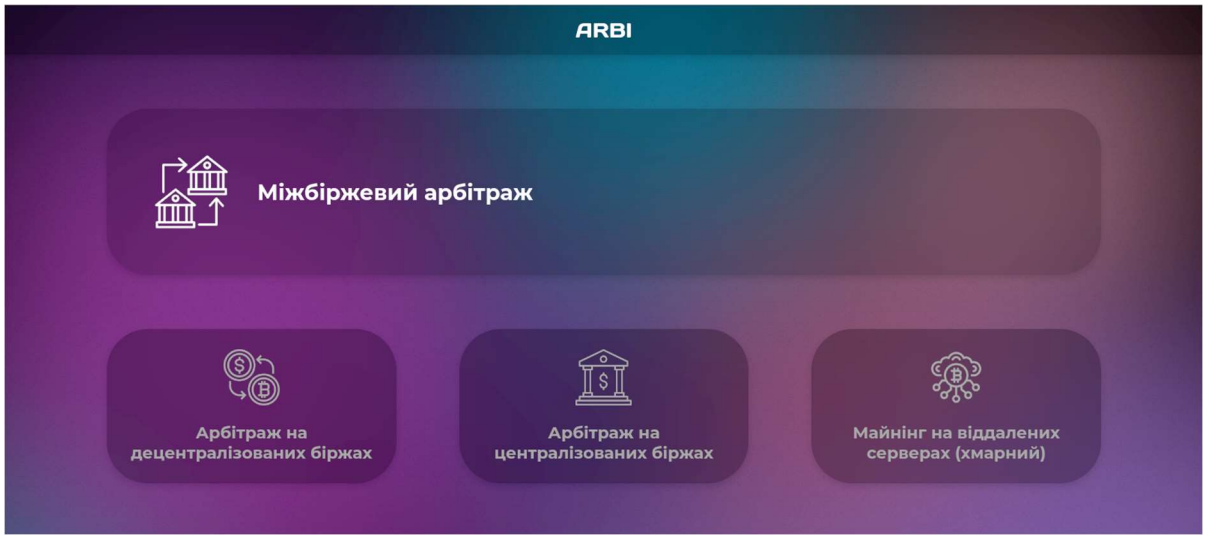

Рисунок 3.11 – Головне меню

При наведенні на логотип «ARBI», користувач бачить контекстне меню для навігації по веб-додатку (рис. 3.12). З нього є можливість перейти в особистий кабінет (рис. 3.13), звідки можна налаштувати власні дані (Ім’я, прізвище, фото профілу), встановити/змінити пароль і т.д.

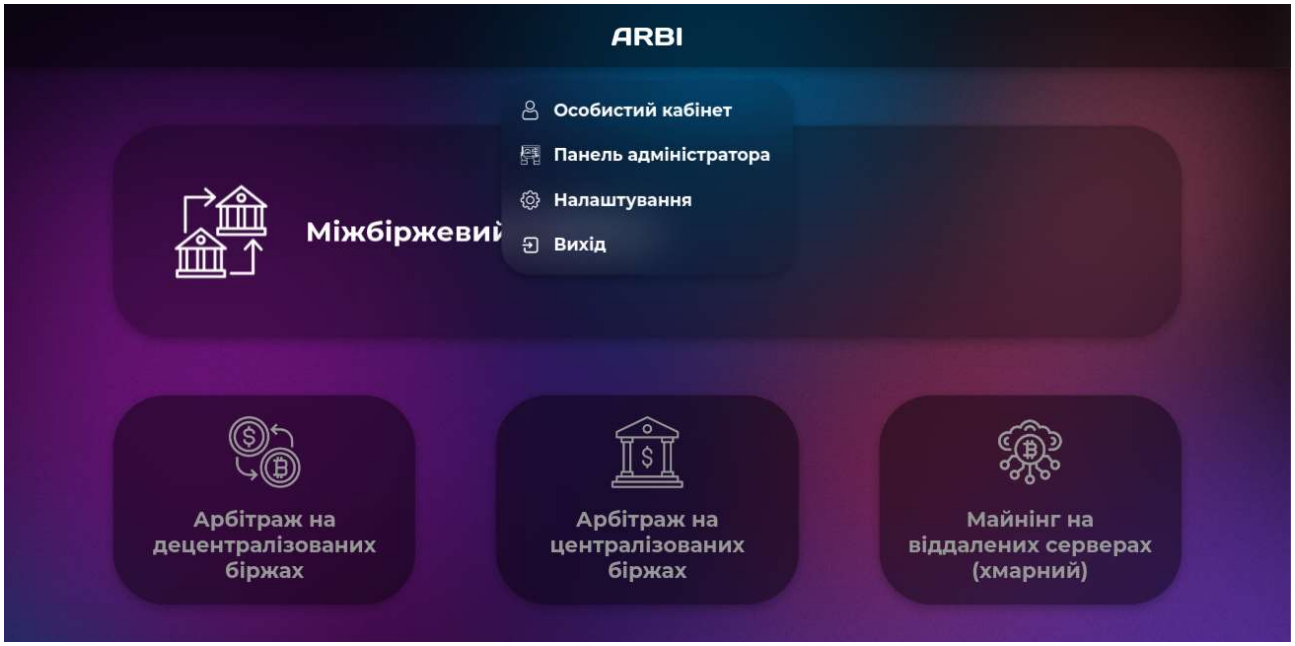

Рисунок 3.12 – Контекстне меню для навігації

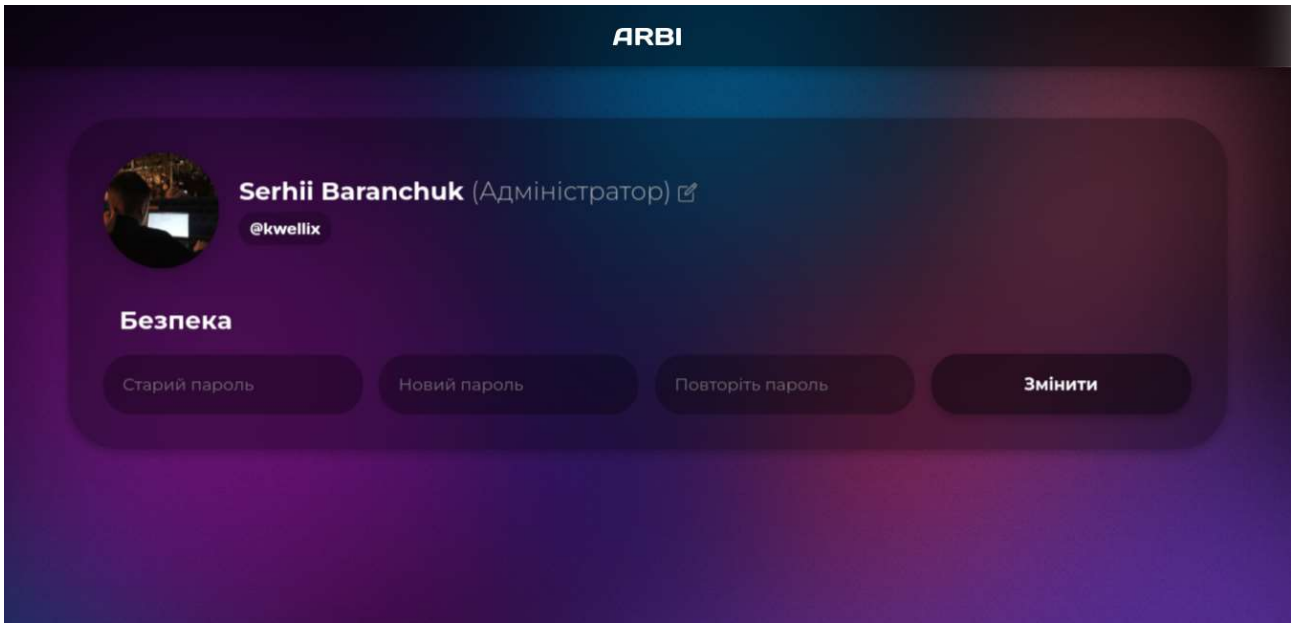

Рисунок 3.13 – Особистий кабінет користувача

З головного меню користувач має можливість перейти в основні режими роботи веб-додатку. На разі, нас найбільше цікавить пункт «Міжбіржевий арбітраж» (додаток В) (рис. 3.14).

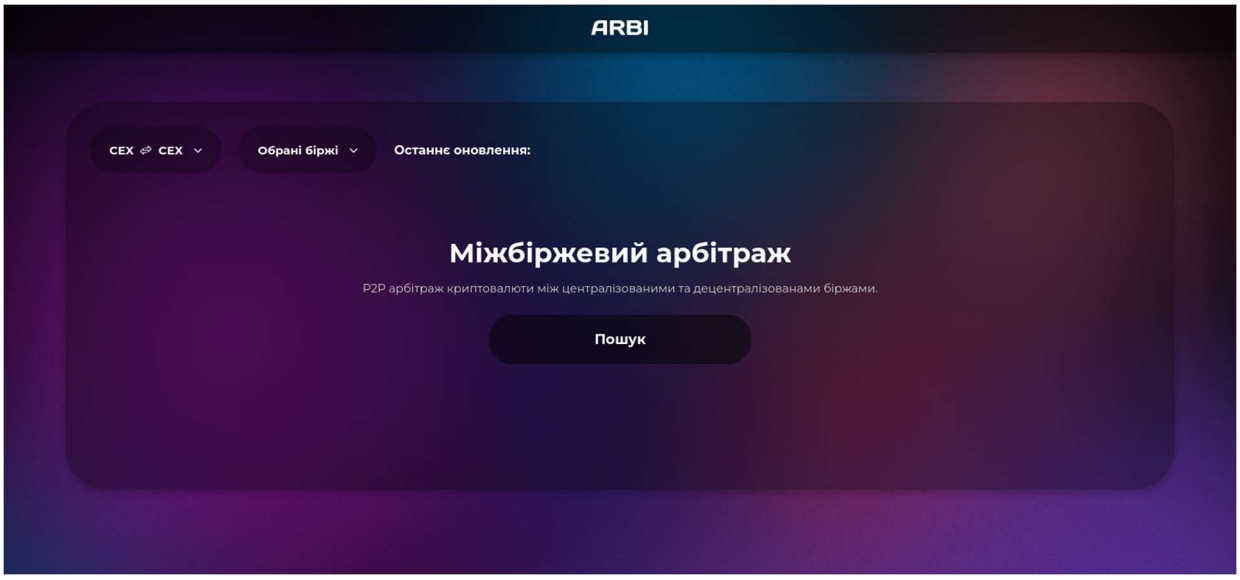

Рисунок 3.14 – Меню «Міжбіржевий арбітраж»

Даний режим представляє собою описаний на початку роботи пошук арбітражних ланцюгів між різними біржами та типами бірж. Після вибору необхідних бірж та натискання кнопки «Пошук», користувач побачить перед собою список усіх можливостей (рис. 3.15).

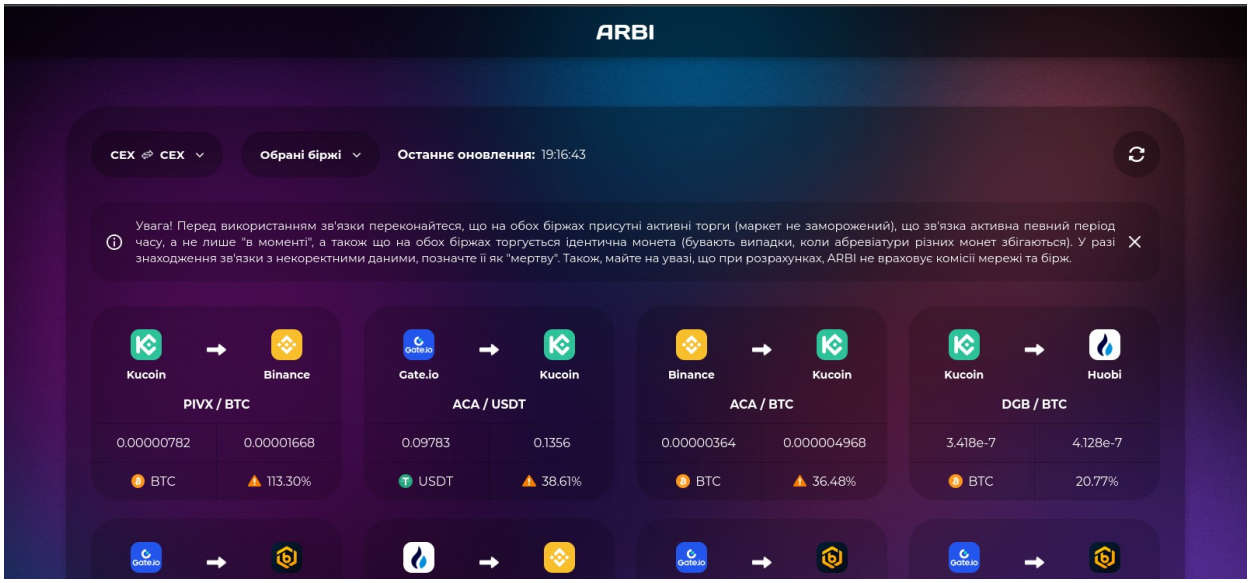

Рисунок 3.15 – Список знайдених арбітражних ланцюгів

У системі також передбачена перевірка на підозрілі арбітражні можливості, так як біржі можуть інколи надавати некоректні дані. Такі зв’язки позначаються помаранчевим знаком оклику (рис. 3.16).

Рисунок 3.16 – Попередження про можливі некоректні дані

Наступним кроком, користувач повинен обрати арбітражний ланцюг, який його зацікавив, та вже працювати напряму з біржами, на яких він був знайдений. Для прикладу, на рисунку 3.17 була знайдений арбітражний ланцюг для валютної пари MDX/BTC, в якій «базовою» валютою виступає Bitcoin. Простими словами, користувач може купити певну кількість валюти MDX на біржі Huobi по ціні 0.00000252 BTC, і продати її на біржі Binance по ціні 0.00000299 BTC. Різниця складає 18.65%. Для конкретного аналізу можливого потенційного об’єму покупки, необхідно відкрити кожну біржу (рис. 3.18, рис. 1.19) натиснувши на їх логотипи, та провести аналіз по торговій книзі замовлень.

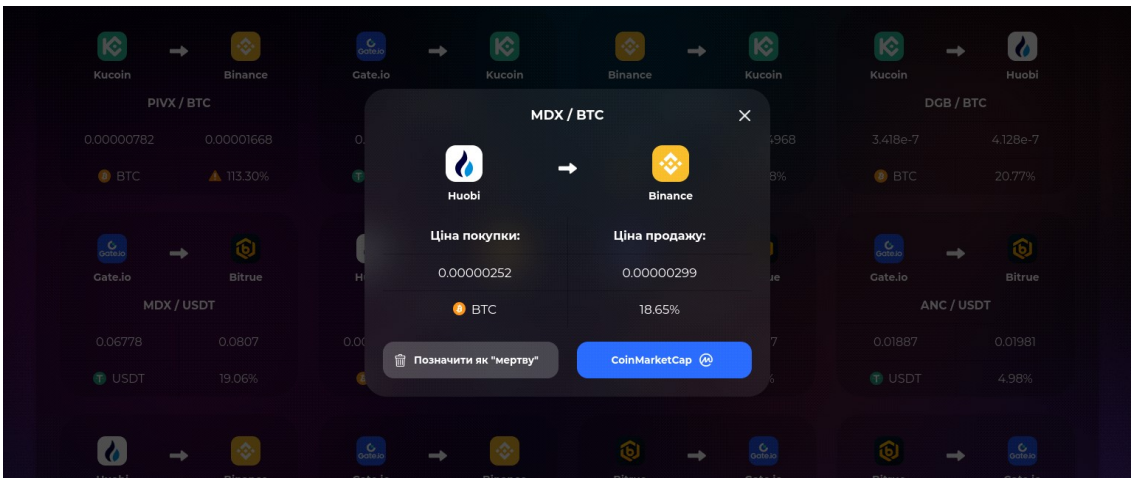

Рисунок 3.17 – Арбітражний ланцюг, базований на торговій парі MDX/BTC

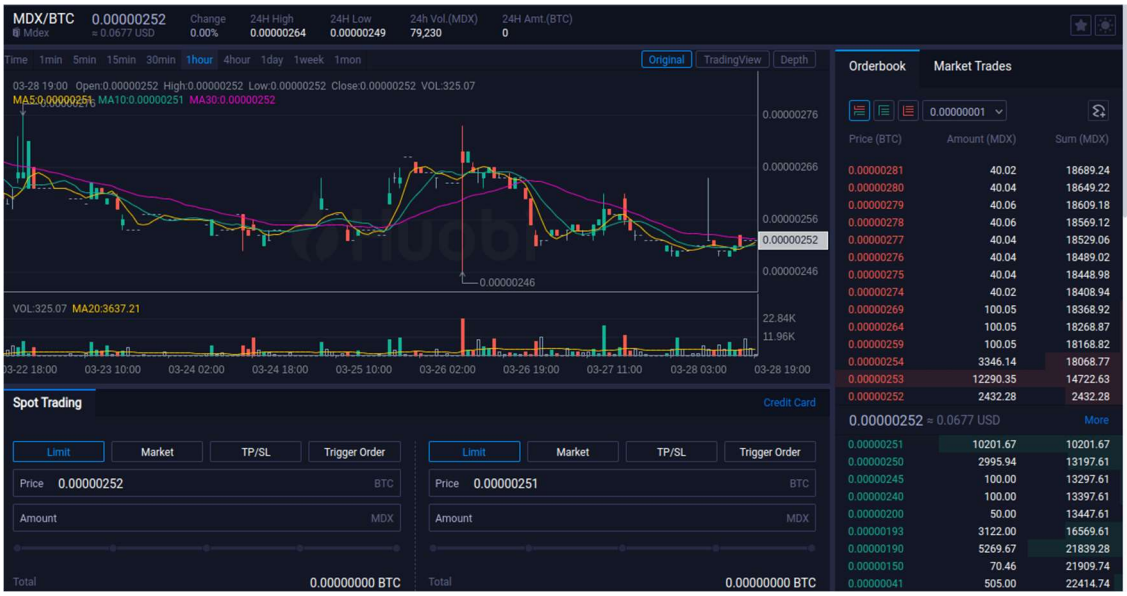

Рисунок 3.18 –Торгова пара MDX/BTC на біржі Huobi

Рисунок 3.19 –Торгова пара MDX/BTC на біржі Binance

Додатково, частина функціоналу розробленої системи представлена у вигляді «скриптів» командного рядка терміналу. Даний функціонал дозволяє проводити внутрішні торги на біржі, напряму, взаємодіючі з торговим API. Даний функціонал дозволяє проводити торгові операції зі швидкістю, недосяжною для звичайної людини (на рисунку 3.20 присутній приклад використання, на якому видно, що на повний «торговий круг» програма витратила всього 3 секунди).

Відкриті ордери (0) | Баланси | Історія ордерів | Історія торгів

Дата	Закрито	Пара	Сторона	Тип	Ціна
01.10.2022 20:26:46	01.10.2022 20:26:46	WBT/USDT	Продаж	Limit	10.829 USDT
01.10.2022 20:26:44	01.10.2022 20:26:44	WBT/BTC	Купівля	Limit	0.0005613 BTC
01.10.2022 20:26:43	01.10.2022 20:26:43	BTC/USDT	Купівля	Limit	19,268.48 USDT

Рисунок 3.20 – Результат «скриптової» торгівлі

Додатково, користувач може використовувати веб-додаток з будь якого пристрою, так як було передбачено адаптивний інтерфейс. Мобільний вигляд сайту зображений на рисунках 3.21, 3.22, 3.23.

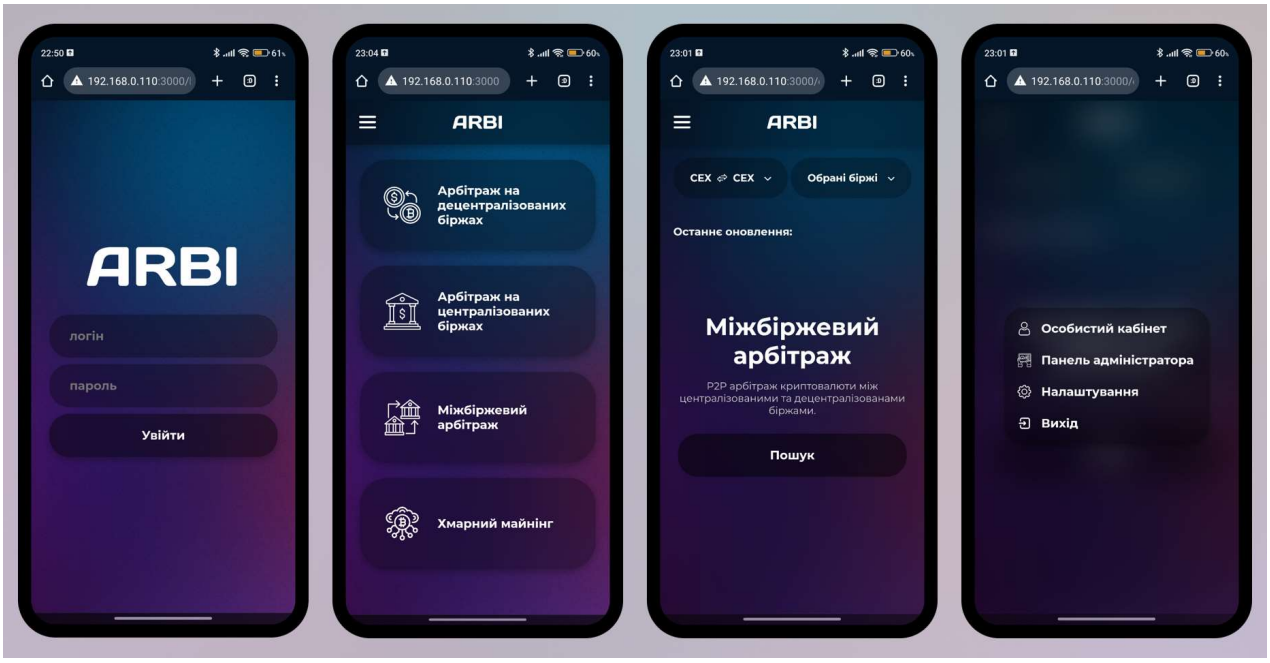

Рисунок 3.21 – Мобільна версія розробленої системи

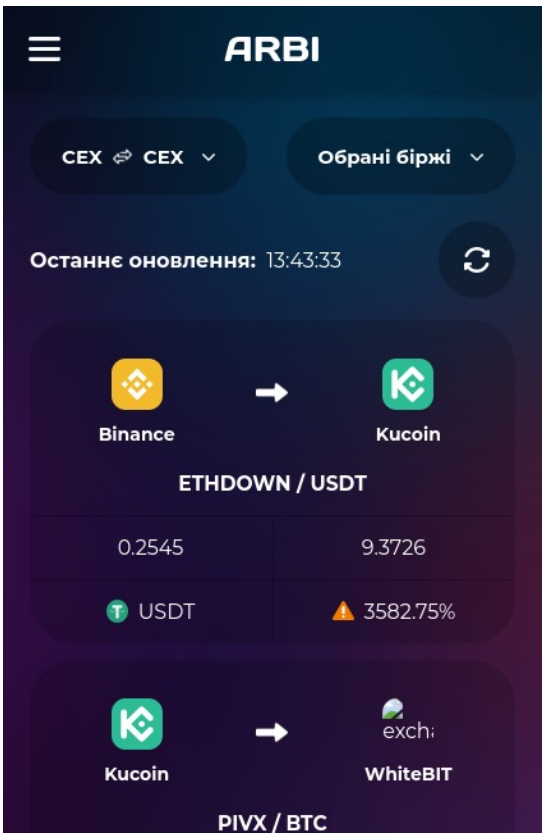

Рисунок 3.22 – Мобільна версія меню «Міжбіржевий арбітраж» зі знайденими арбітражними можливостями

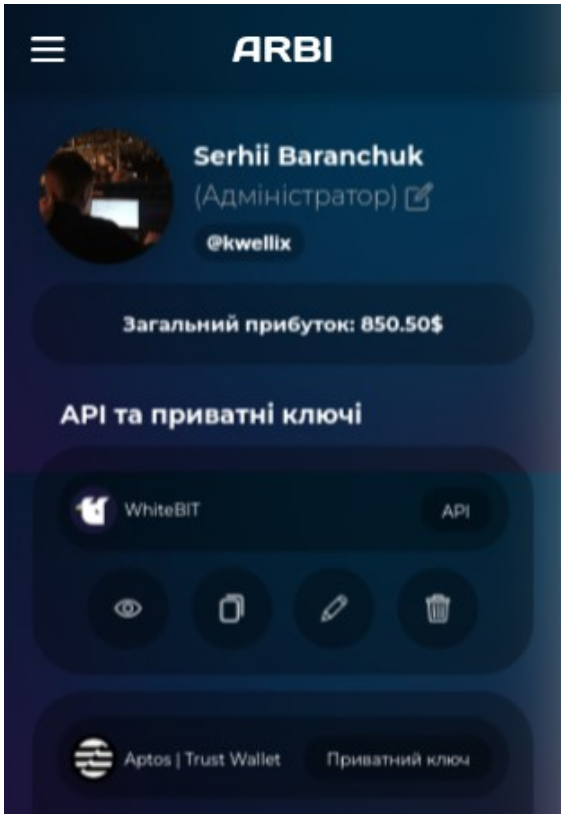

Рисунок 3.23 – Мобільна версія особистого кабінету користувача

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи була проведена розробка та реалізація автоматизованої веб-системи для торгівлі криптовалютами на криптовалютних біржах. Метою роботи було створення функціонального і надійного інструменту, що допомагає трейдерам ефективно використовувати можливості ринку криптовалют для заробітку.

В ході дослідження було розглянуто загальні відомості про криптовалютні системи, криптовалютний ринок, криптовалютні біржі та блокчейн-технології. Була обґрунтована актуальність автоматизації торгівлі на криптовалютних біржах та переваги розробки власної системи.

У процесі реалізації була описана функціонально-структурна схема системи, розроблені необхідні алгоритми, класи, створена модель бази даних, забезпечено високу швидкодію та продуктивність усіх елементів системи, та створено зручний та зрозумілий інтерфейс користувача. Також, було проведено аналіз та опис архітектури системи, принципів взаємодії між її складовими та розроблені усі необхідні компоненти. Важливим етапом також було забезпечення сумісності системи з різними пристроями.

Після реалізації було проведено тестування системи з метою пошуку та усунення несправностей. В результаті тестування системи було виявлено, що розроблена веб-система працює стабільно та безперебійно, забезпечуючи широкий функціонал для торгівлі криптовалютами на криптовалютних біржах.

Отже, можна зробити висновок, що розроблена веб-система є ефективним інструментом для автоматизованої торгівлі криптовалютами та може бути використана трейдерами для досягнення успіху на криптовалютних ринках. Результати дослідження показали, що використання сучасних технологій розробки дозволяє створювати високоякісні веб-системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Топ-100 криптовалют за ринковою капіталізацією. CoinMarketCap. URL: https://coinmarketcap.com/ (дата звернення: 20.02.2023).

2. Чернявський Ю. С. Можливість застосування технології блокчейн. Підприємництво та інновації. 2022. № 24. С. 118–122. URL: https://doi.org/10.32782/2415-3583/24.20 (дата звернення: 22.02.2023).

3. Розробка MERN-додатку крок за кроком. Частина 1. Dev.to. URL: https://dev.to/osiroski/guide-to-develop-mern-application-step-by-step-part-1-2h3l/ (дата звернення: 03.03.2023).

4. Документація JavaScript. Developer.mozilla.org. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript. (дата звернення: 13.03.2023).

5. Болож О. В. Розробка веб-сайту «Artise» засобами React, Node.js та MongoDB : кваліфікаційна робота освітнього рівня «Бакалавр». Тернопіль: ТНТУ, 2022. 45 с.

6. Devecosystem databases 2022. JetBrains. URL: https://www.jetbrains.com/lp/devecosystem-2022/databases/ (дата звернення: 14.03.2023).

7. Документація MongoDB. Itglobal. URL: https://itglobal.com/ru-ru/company/glossary/mongodb/ (дата звернення: 14.03.2023).

8. Христинець Н., Лавренчук С., Свиридюк К., Скригунець В. Розробка масштабованих веб-додатків з використанням фреймворку React та бази даних MongoDB. Computer-integrated technologies: education, science, production. 2021. № 45. С. 97–102. URL: https://doi.org/10.36910/6775-2524-0560-2021-45-14 (дата звернення: 18.03.2023).

9. Why MERN Stack? H. Sharma et al. International Journal for Research in Applied Science and Engineering Technology. 2022. Vol. 10, no. 12. P. 521–527. URL: https://doi.org/10.22214/ijraset.2022.47927 (дата звернення: 20.03.2023).

10. Бібе А. Є. Розробка веб-додатків з використанням JavaScript фреймворків Vue та React: дипломна робота на здобуття кваліфікаційного ступеня бакалавра. Дніпро: УДУНТ, 2022. 43 с.

11. Трипутін В. С. Дослідження впливу React на фреймворки розробки Web-застосунків: дипломна робота на здобуття кваліфікаційного ступеня магістра. Дніпро: УДУНТ, 2022. 154 с.

12. NoSQL. Wikipedia. URL: https://uk.wikipedia.org/wiki/NoSQL (дата звернення: 20.03.2023).

13. Швець М.Ю., Заруба Д.С., Хохлов Ю.В. Порівняння SQL та NOSQL баз даних. Інформатика, обчислювальна техніка та автоматизація. Київ: КПІ, 2018. С. 21-25.

14. Simplifying JavaScript. Network Security. Vol. 2018, no. 5. 2018. P. 5. URL: https://doi.org/10.1016/s1353-4858(18)30041-2 (дата звернення: 22.03.2023).

15. DiPierro M. The Rise of JavaScript. Computing in Science & Engineering. Vol. 20, no. 1. 2018. P. 9–10. URL: https://doi.org/10.1109/mcse.2018.011111120 (дата звернення: 23.03.2023).

16. Малохвій Е.Е, Бугай В.С., Молчанов Г.І., Черних О.П. Аналітичний огляд та порівняння сучасних Javascript рішень для розробки веб-додатків. Системи управління, навігації та зв'язку. Збірник наукових праць. Полтава: ПНТУ, 2021. Т. 4 (66). С. 55-58.

17. Стецик О., Теренчук С. Порівняльний аналіз архітектур нереляційних баз даних. Управління розвитком складних систем. Київ: КНУБД, 2021. С. 78–82.

18. Як краще використовувати NoSQL бази даних. IBM. URL: https://www.ibm.com/cloud/blog/how-to-choose-a-database-on-ibm-cloud (дата звернення: 12.04.2023).

19. Ісаков А. В., Ліміна В. Ю., Романюк О. В. Сфери застосування нереляційних баз даних. Матеріали XLIX науково-технічної конференції підрозділів ВНТУ, Вінниця, 27-28 квітня 2020 р. Вінниця: ВНТУ. 2020. URL:

https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9416 (дата звернення: 15.04.2023).

20. Іванівна Т., Захарченко С. Комбінований метод масштабування баз даних. Grail of Science. 2022. № 14-15. С. 320–326. URL: https://doi.org/10.36074/grail-of-science.27.05.2022.056 (дата звернення: 17.04.2023).

21. Клачко Л. Ю. Методи та засоби міграції з реляційних баз даних в нереляційні : автореф. Thesis Abstract. 2018. URL: http://elartu.tntu.edu.ua/handle /lib/23563 (дата звернення: 21.04.2023).

22. Переваги та недоліки нереляційних баз даних. Quality-assurance-group. URL: https://www.quality-assurance-group.com/nosql-perevagy-ta-nedoliky-nerelyatsijnyh-baz-danyh (дата звернення: 22.04.2023).

23. Інструменти для Frontend-розробки, які вам точно допоможуть. Itstep. URL: https://msk.itstep.org/blog/frontend (дата звернення: 24.04.2023).

24. Фролов О., Білодід О. Дослідження та аналіз інструментів Front-end розробки Web-додатків. Тези доповідей Міжнародної науково-практичної конференції «Інформаційні технології та системи» 14 - 15 квітня 2022 р. Харків: ХНЕУ, 2022. С. 48-57

25. Бойко І.В., Куніц В.В. Аналіз особливостей технологій Frontend розробки. У: Матеріали IX Міжнародної науково-технічної конференції молодих учених та студентів. Актуальні задачі сучасних технологій – Тернопіль 25-26 листопада 2020. Тернопіль: ТНТУ, 2020. С. 34-35

ДОДАТКИ

Додаток А

Лістинг коду Backend’у веб-системи

```
import express from "express"
import {registerValidation} from "./validations/auth.js";
import mongoose from "mongoose"
import checkAuth from "./utils/checkAuth.js";
import * as userController from "./controllers/userController.js"
import multer from "multer"
import cors from "cors"

mongoose.connect('***')
    .then(() => {
        console.log("DB CONNECTED")
    })
    .catch(err => console.log("DB ERROR", err))

const app = express()

app.use(express.json({limit: '10mb'}));
app.use(cors())
const storage = multer.diskStorage({
    destination: (_, __, cb) => {
        cb(null, 'uploads');
    },
    filename: (_, file, cb) => {
        cb(null, file.originalname);
    },
});

const upload = multer({storage})

app.use('/uploads', express.static('uploads'));
app.post("/upload", checkAuth, upload.single('image'), (req, res) => {
    res.json({
        url: `/uploads/${req.file.originalname}`,
    })
})

app.post("/auth/login", userController.login)
app.post("/auth/register", registerValidation, userController.register)
app.get("/auth/me", checkAuth, userController.authMe)
app.post("/update/userData", checkAuth, userController.updateData)

app.listen(4444, (err) => {
    if (err) return console.log(err)

    console.log("Server started")
})

app.get("/", (req, res) => {
    res.json({answer: "Welcome to server"})
})
```

```
            <main>
                <Header />
                <div className="CEA-contentCard">
                    <div className="CEA-contentCard-header">
                        <div className="CEA-dropDowns">
                            <div className="CEA-dropDownMenu">
                                <CrossWayDropdown crossWay={crossWay}
setCrossWay={setCrossWay} />
                            </div>
                            <div className="CEA-dropDownMenu">
                                <ExchangesListDropdown
selectedExchanges={selectedExchanges} selectExchange={selectExchange} />
                            </div>
                        </div>
                        <div className="CEA-updatesBlock">
                            <div className="CEA-lastUpdateTime">
                                <p style={{ fontWeight: 'bold', whiteSpace:
'nowrap' }}>Останнє оновлення:</p><p style={{ whiteSpace: 'nowrap' }}> 
{lastUpdateTime}</p>
                            </div>
                            {isDataVisible && (
                                <div className="CEA-refreshBtn">
                                    <div onClick={getData} className="CEA-
refreshBtn-bg">
                                        <img className="CEA-refreshBtn-icon"
src="https://cdn-icons-png.flaticon.com/512/3031/3031712.png" alt="refresh" />
                                    </div>
                                </div>
                            )}
                        </div>
                    </div>
                    {!isDataVisible && (
                        <div>
                            {isDataLoading && (
                                <div className="CEA-PageTitleAndButton">
                                    <Triangle
                                        height="125"
                                        width="125"
                                        color="white"
                                        ariaLabel="triangle-loading"
                                        wrapperStyle={{ marginTop: '3vh' }}
                                        wrapperClassName=""
                                        visible={true}
                                    />
                                </div>
                            )}
                            {!isDataLoading && (
                                <div className="CEA-PageTitleAndButton">
                                    <p style={{ fontSize: 35, fontWeight:
'bold', textAlign: 'center' }}>Міжбіржевий арбітраж</p>
                                    <div style={{ color: 'lightgray', marginTop:
15, fontSize: 15, textAlign: 'center' }}>
                                        <p>P2P арбітраж криптовалюти між
централізованими та децентралізованами біржами.</p>
                                    </div>
                                    <button onClick={getData} type="button"
className="CEA-findArbitageBtn">Пошук</button>
                                </div>
                            )}
                        </div>
                    )}
                    {isDataVisible && (
                        <div>
```

Додаток Б

Лістинг коду сторінки авторизації

```
import React from "react";
import {Link, Navigate} from "react-router-dom";
import axios from "axios";
import {useForm} from "react-hook-form"
import {useDispatch, useSelector} from "react-redux"
import {fetchAuth, selectIsAuth} from "../redux/slices/auth";
import {Triangle} from "react-loader-spinner";

const Login = () => {
    const [errorState, setErrorState] = React.useState(null)
    const [loadingState, setLoading] = React.useState(false)
    const isAuth = useSelector(selectIsAuth)
    const dispatch = useDispatch()

    const {register, handleSubmit, setError, formState: {errors, isValid}} =
useForm({
        defaultValues: {
            login: "",
            password: "",
        },
        mode: "onChange"
    })

    const onSubmit = async values => {
        setLoading(true)
        values.login = values.login.toLowerCase().replaceAll(" ", '')
        const data = await dispatch(fetchAuth(values))
        setLoading(false)

        if (!data.payload) {
            setErrorState("Некоректні дані")
            return 0
        }

        if (`token` in data.payload) {
            window.localStorage.setItem('token', data.payload.token)
        }
    }

    if (isAuth) {
        return <Navigate to="/"/>
    }

    return (
        <div className="loginForm">
            <form className="loginFormInputs" onSubmit={handleSubmit(onSubmit)}
autocomplete="off" onChange={() => setErrorState(null)}>
                <p className="loginLogo">ARBI</p>
                <input
                    className="loginInput"
                    placeholder="Логін"
                    type="login"
                    {...register("login", {required: "Введіть логін"})}
                />
                <input
                    className="passwordInput"
                    placeholder="пароль"
                    type="password"
                    {...register("password", {required: "Введіть пароль"})}
```

```
                />
                {loadingState && (
                    <div style={{width: "100%", display: "flex", justifyContent:
"center"}}>
                        <Triangle
                            height="100"
                            width="100"
                            color="white"
                            ariaLabel="triangle-loading"
                            wrapperStyle={{marginTop: '3vh'}}
                            wrapperClassName=""
                            visible={true}
                        />
                    </div>
                )}
                {!loadingState &&
                    <button type="submit" className="loginButton">
                        Увійти
                    </button>
                }
                {errorState !== null && <p
className="loginFormError">{errorState}</p>}
            </form>
        </div>
    );
};

export default Login;
```

Додаток В

Лістинг коду сторінки з міжбіржевим арбітражем

```
import React from "react"
import Header from '../components/general/Header.jsx'
import CrossExArbCard from "../components/crossExArb/CrossExArbCard.jsx"
import CrossWayDropdown from "../components/crossExArb/CrossWayDropdown.jsx"
import ExchangesListDropdown from
"../components/crossExArb/CrossExListDropdown.jsx"
import exchanges from "../data/exchanges.js"
import { Triangle } from 'react-loader-spinner'
import { getChains, getAllTickersObject } from "../scripts/crossExArbitrage.js"

const CrossExArbitrage = () => {
    document.title = "Міжбіржевий арбітраж | ARBI"
    const [isDataVisible, setVisible] = React.useState(false)
    const [isDataLoading, setLoading] = React.useState(false)
    const [isAttentionVisible, setAttentionVisible] = React.useState(true)
    const [lastUpdateTime, setLastUpdateTime] = React.useState("")
    const [arbitrageData, setArbitrageData] = React.useState([])
    const [crossWay, setCrossWay] = React.useState("CEX")
    const [selectedExchanges, selectExchange] = React.useState({ "Binance":
true, "WhiteBIT": true, "Huobi": true, "Bitrue": true, "OKX": true, "Gate_io":
true, "Bybit": true, "Kucoin": true })
    const [deadFilter, addDeadFilter] = React.useState({ "Binance": {},
"WhiteBIT": {}, "Huobi": {}, "Bitrue": {}, "OKX": {}, "Gate_io": {}, "Bybit":
{}, "Kucoin": {} })

    const getData = () => {
        setLoading(true)
        setVisible(false)
        console.clear()

        getAllTickersObject(selectedExchanges).then(answer => {
            setLoading(false)
            setVisible(true)
            let data = getChains(answer, deadFilter).sort((x, y) => {
                x = x["Спред (%)"]
                y = y["Спред (%)"]
                return y - x
            })
            setArbitrageData(data)

            let now = new Date()
            let curTime = now.getHours() + ':' + now.getMinutes() + ':' +
now.getSeconds();
            setLastUpdateTime(curTime)
        })

    }

    const filterDeadMarket = (exchange, token1, token2) => {
        let newValue = deadFilter[exchange]
        let pair = token1 + "_" + token2
        newValue[pair] = "dead"
    }
    let id = 0
    console.log(arbitrageData)
    //console.log(deadFilter)

    return (
        <div>
```

```
                            {isAttentionVisible && (<div style={{ marginTop:
'calc(100vw * 0.02)', display: 'flex', alignItems: 'center', padding: 20,
backgroundColor: "rgba(0, 0, 0, 0.15)", borderRadius: 25, }}>
                                <img src="https://cdn-icons-
png.flaticon.com/512/157/157933.png" style={{ height: 20, width: 20,
marginRight: 17.5, filter: "invert(100%)" }} alt="info" />
                                <p style={{ fontSize: 15, textAlign: 'justify',
lineHeight: 1.4 }}>Увага! Перед використанням зв'язки переконайтеся, що на обох
біржах присутні активні торги (маркет не заморожений), що зв'язка активна певний
період часу, а не лише "в моменті", а також що на обох біржах торгується
ідентична монета (бувають випадки, коли абревіатури різних монет збігаються). У
разі знаходження зв'язки з некоректними даними, позначте її як "мертву". Також,
майте на увазі, що при розрахунках, ARBI не враховує комісії мережі та бірж.</p>
                                <img onClick={() => { setAttentionVisible(false)
}} src="https://cdn-icons-png.flaticon.com/512/1828/1828778.png" style={{
cursor: 'pointer', height: 15, width: 15, paddingLeft: 17.5, marginRight: 5,
filter: "invert(100%)" }} alt="info" />
                            </div>
                            )}
                            <div className="CEA-contentCard-content">
                                {// eslint-disable-next-line
                                    arbitrageData.map((item) => {
                                        id++;
                                        let arbWay = item["Шлях"].split(" -> ")
                                        let market = item["Торгова
пара"].split("_")
                                        let baseCurrencyIcon =
market[1].toLowerCase()
                                        if ((item["Спред (%)"] < 10000) &&
(item["Спред (%)"] > 0.5)) {
                                            return (
                                                <div key={id}>
                                                    <CrossExArbCard
                                                        market={market[0] + " /
" + market[1]}

askCexName={exchanges[arbWay[0]].title}

askCexIcon={exchanges[arbWay[0]].icon}

askCexUrl={exchanges[arbWay[0]].url}

askCexType={exchanges[arbWay[0]].type}

bidCexName={exchanges[arbWay[1]].title}

bidCexIcon={exchanges[arbWay[1]].icon}

bidCexUrl={exchanges[arbWay[1]].url}

bidCexType={exchanges[arbWay[1]].type}

baseCurrency={item["Валюта"]}

baseCurrencyIcon={baseCurrencyIcon}
                                                        lowestAsk={item["Ціна
покупки"]}
                                                        highestBid={item["Ціна
продажу"]}
                                                        profit={item["Спред
(%)"]}
                                                        crossWay={crossWay}

filterDeadMarket={filterDeadMarket}
```

```
                                                        />
                                                    </div>
                                                )
                                            }
                                        })}
                                    </div>
                                </div>
                            )}
                        </div>
                    </main >
                </div >
            )
        }

        export default CrossExArbitrage
```