

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:

**«Платформа для демо-трейдингу криптоакти-
вами»**

Виконав студент групи ПЗ-21-5
ВОЙТЕНКО Олег Миколайович

Керівник роботи:
КОНЦИДАЙЛО Андрій Михайлович

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програ-
много забезпечення

Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВОЙТЕНКО Олег Миколайович**

Керівник роботи: **КОНЦИДАЙЛО Андрій Михайлович**

Тема роботи: **«Платформа для демо-трейдингу криптоактивами»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: платформа для демо-трейдингу криптоактивами за-
безпечує симуляцію ринку через API та автоматизацію торгівлі. Додаток під-
тримує навчання трейдингу, надаючи безпечне та прозоре середовище для
відпрацювання навичок трейдингу на тестових даних.

Інструменти розробки: React, Node.js, Express та MongoDB.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розді-
лів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концидайло А.М	27.02.2025р.	27.02.2025р.
2	Концидайло А.М	27.02.2025р.	27.02.2025р.
3	Концидайло А.М	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку платформи для демо-трейдингу криптоактивами та пояснювальної записки. Пояснювальна записка до випускної роботи містить 57 сторінку, 32 ілюстрації та 6 таблиць.

Метою роботи є створення платформи для демо-трейдингу криптоактивами з використанням сучасних технологій та інтеграцією API для симуляції ринкових транзакцій і навчання трейдингу.

Архітектура платформи для демо-трейдингу криптоактивами базується на API, що забезпечує симуляцію ринкових даних і автоматизацію торгових операцій. Платформа створює безпечне та прозоре середовище для відпрацювання навичок трейдингу на тестових даних, без використання блокчейну.

Додаток має зручний інтерфейс, який дозволяє користувачам легко симулювати торгові операції, аналізувати ринкові дані та навчатися трейдингу.

КЛЮЧОВІ СЛОВА: ВЕБСЕРВІС, API, ПЛАТФОРМА, ДЕМО-ТРЕЙДИНГ, КРИПТОАКТИВИ, СИМУЛЯЦІЯ РИНКУ, НАВЧАННЯ ТРЕЙДИНГУ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ					
Змн.	Арк.	№ докум.	Підпис	Дата						
Розроб.		Войтенко О.М.			Платформа для демо-трейдингу криптоактивами		Літ.	Арк.	Аркушів	
Керівник		Концидайло А.М							3	72
Н. контр.		Тичина О.П.					Житомирська політехніка, група ІПЗ-21-5			
Зав. каф.		Вакалюк Т.А.								

ABSTRACT

The bachelor's graduation qualification work consists of a web application of a demo trading platform for crypto assets and an explanatory note. The explanatory note to the graduation work contains 57 pages, 32 illustrations and 6 tables.

The purpose of the work is to create a demo trading platform for crypto assets using modern technologies and API integration for simulating market transactions and learning trading.

The architecture of the demo trading platform for crypto assets is based on an API that provides simulation of market data and automation of trading operations. The platform creates a safe and transparent environment for practicing trading skills on test data, without using the blockchain.

The application has a user-friendly interface that allows users to easily simulate trading operations, analyze market data and learn trading.

KEYWORDS: WEB SERVICE, API, PLATFORM, DEMO TRADING, CRYPTOASSETS, MARKET SIMULATION, TRADING TRAINING.

					ИПЗ.КР.Б – 121 – 25 – ИПЗ	Арк.
	.					4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ ДЕМО ТРЕЙДИНГУ КРИПТОАКТИВАМИ.....	9
1.1 Постановка задачі.....	11
1.2 Аналіз аналогів програмного продукту.....	9
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апарат- ного забезпечення.....	18
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЕКТУВАННЯ ПЛАТФОРМИ ДЛЯ ДЕМО ТРЕЙДИНГУ КРИПТОАКТИВАМИ	21
2.1 Вимоги до системи.....	21
2.2 Розробка бази даних.....	24
2.3 Проектування та реалізація алгоритмів роботи платформи.....	28
2.4 Проектування серверної частини та API	31
2.5 Реалізація функціоналу платформи для демо трейдингу Криптоакти- вами	33
Висновки до другого розділу.....	40
РОЗДІЛ 3. . ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ДЕМОТРЕЙДИНГУ КРИПТОАКТИВАМИ	41
3.1 Порядок встановлення та налаштування параметрів	41
3.2 Структура інтерфейсу. Інтерфейс та порядок роботи з системою.....	42
3.3 Тестування платформи.....	51
Висновки до третього розділу	54
ВИСНОВКИ	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	57
ДОДАТКИ.....	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

SDK – набір для розробки програмного забезпечення (software development kit)

CRUD – основні операції, що використовуються для управління даними (Create , Read , Update , Delete)

JSON – текстовий формат обміну даними (JavaScript Object Notation)

JWT – маркер веб-токена JSON (JSON Web Token)

REST – архітектурний стиль взаємодії між клієнтом і сервером (Representational State Transfer)

SPA – односторінковий застосунок (Single Page Application)

PnL – прибуток і збиток (Profit and Loss)

DOM – модель об'єктів документа (Document Object Model)

ВСТУП

Актуальність теми полягає у зростанні популярності криптоактивів та необхідності створення доступних інструментів для навчання трейдингу. Використання сучасних технологій, зокрема API для отримання реальних ринкових даних, дозволяє створювати платформи демо-трейдингу, які забезпечують безпечне та прозоре середовище для відпрацювання торгових навичок. Демо-трейдинг дає можливість користувачам, особливо новачкам, вивчати ринкові механізми, аналізувати реальні дані та тестувати стратегії шляхом імітації ордерів без фінансових ризиків. Такий підхід сприяє підвищенню фінансової грамотності та підготовці до реальної торгівлі криптоактивами.

Метою випускної кваліфікаційної роботи бакалавра є розробка платформи для демо-трейдингу криптоактивами з використанням API для отримання реальних ринкових даних та імітації ордерів. Для реалізації поставленої мети необхідно виконати низку завдань серед яких проведення огляду функціональності платформ демо-трейдингу методів імітації ордерів на основі реальних ринкових даних отриманих через API а також аналізу статистичних даних. Наступним кроком є вибір засобів проектування та розробки вебдодатку що відповідає вимогам до сучасних систем взаємодії з користувачем у реальному часі. Завершальним етапом є безпосередня розробка самої платформи з інтеграцією API для отримання реальних ринкових котирувань та модулів які забезпечують імітацію торгових операцій що дозволяє користувачам без ризику відпрацьовувати стратегії трейдингу.

Об'єктом дослідження є процес демо-трейдингу криптоактивами, який використовує API для отримання реальних ринкових даних та імітації ордерів для навчання користувачів.

Предметом дослідження є функціональність платформ демо-трейдингу на базі API, їх можливості та переваги у порівнянні з реальними торговими платформами. Дослідження також охоплює аналіз ролі API у забезпеченні доступу до

реальних ринкових даних, можливості автоматизації імітації ордерів та створення зручного інтерфейсу для навчання трейдингу.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ ДЕМО ТРЕЙДИНГУ КРИПТОАКТИВАМИ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення платформи для демо-трейдингу криптоактивами [28, 29], використовуючи сучасні технології, зокрема API для отримання реальних ринкових даних, що дозволить легко використовувати та масштабувати платформу в майбутньому.

Демо-трейдинг – це практика імітації торгових операцій на фінансових ринках без реальних фінансових ризиків, яка дозволяє користувачам навчатися трейдингу, тестувати стратегії та аналізувати ринкові дані. Одним із відомих прикладів демо-трейдингу є платформа Binance Futures Testnet, яка надає користувачам доступ до симуляції торгівлі ф'ючерсами на основі реальних ринкових даних. Ця платформа дозволяє трейдерам-початківцям відпрацьовувати навички, використовуючи віртуальний баланс, та вивчати ринкові механізми.

У 2020 році Binance запустила Testnet для ф'ючерсів, що стало популярним інструментом серед трейдерів. Платформа дозволяла користувачам імітувати ордери на основі реальних ринкових даних, отриманих через API, і тестувати стратегії в умовах, максимально наближених до реальних. За короткий час платформа привернула увагу тисяч користувачів, що підкреслює попит на інструменти демо-трейдингу.

Платформа для демо-трейдингу має складатися з кількох взаємопов'язаних програмних модулів, кожен з яких виконує окрему функцію і забезпечує користувачам повноцінний функціонал для навчання трейдингу. Зокрема, має бути реалізований модуль імітації торгівлі, який дозволяє користувачам створювати віртуальні ордери на основі реальних ринкових даних, отриманих через API. Користувач має можливість встановлювати торгові цілі, виконувати

операції та переглядати результати своєї діяльності. Також платформа повинна містити модуль аналізу даних, який надає інструменти для перегляду графіків, аналізу історичних котирувань та оцінки ефективності обраних стратегій.

Крім того, у складі платформи має бути модуль безпеки та авторизації, який забезпечує захист персональних даних, надійну систему автентифікації та можливість розмежування доступу до функціоналу в залежності від ролі користувача. Окремо має бути реалізований модуль перегляду статистики, який надає зручний інтерфейс для аналізу особистої торгової активності, оцінки відкритих і закритих позицій, а також моніторингу прибутковості створених портфелів.

У результаті має бути створений вебсайт, який дозволяє будь-якому користувачу зареєструватися, увійти до системи, отримати доступ до функцій демо-трейдингу та виконувати симуляцію торгових операцій на основі реальних ринкових даних, що надходять через API.

Процес розробки платформи має включати кілька основних етапів. На першому етапі має відбуватися розробка дизайну вебсайту, зокрема створення концепції зовнішнього вигляду, проєктування UI-компонентів та макетів сторінок. Далі має бути реалізована архітектура клієнтської частини, яка забезпечує логіку взаємодії користувача з додатком, та архітектура серверної частини, яка обробляє запити, керує даними та забезпечує обмін інформацією з базою даних.

Також у межах реалізації функціоналу має бути розроблена система авторизації. Необхідно створити логіку авторизації з перевіркою токенів, реалізувати віддалене зберігання інформації, налаштувати обмін даними між клієнтом і сервером, а також здійснити інтеграцію API для отримання ринкових даних і виконання імітації торгових операцій.

Останнім етапом має бути тестування платформи. Потрібно провести функціональне тестування роботи окремих модулів та інтерфейсів, перевірити зручність користування графічним інтерфейсом та переконатися у стабільності

роботи всієї системи. Завершенням розробки має стати готова до використання платформа, яка дозволяє користувачам у безпечному середовищі відпрацьовувати навички трейдингу, аналізувати результати своєї діяльності та ефективно навчатися роботі з фінансовими інструментами.

1.2. Аналіз аналогів програмного продукту

Для розробки платформи демо-трейдингу криптоактивами необхідно провести аналіз існуючих аналогів, щоб визначити їхні сильні та слабкі сторони, а також врахувати сучасні тенденції у сфері фінансових технологій. У цьому підрозділі розглядаються популярні платформи демо-трейдингу, які використовують реальні ринкові дані через API для імітації торгових операцій. Основними аналогами є Binance Futures Testnet, TradingView Paper Trading та Bit-MEX Testnet.

Одним з таких додатків є Binance Futures Testnet (Рис. 1.1)

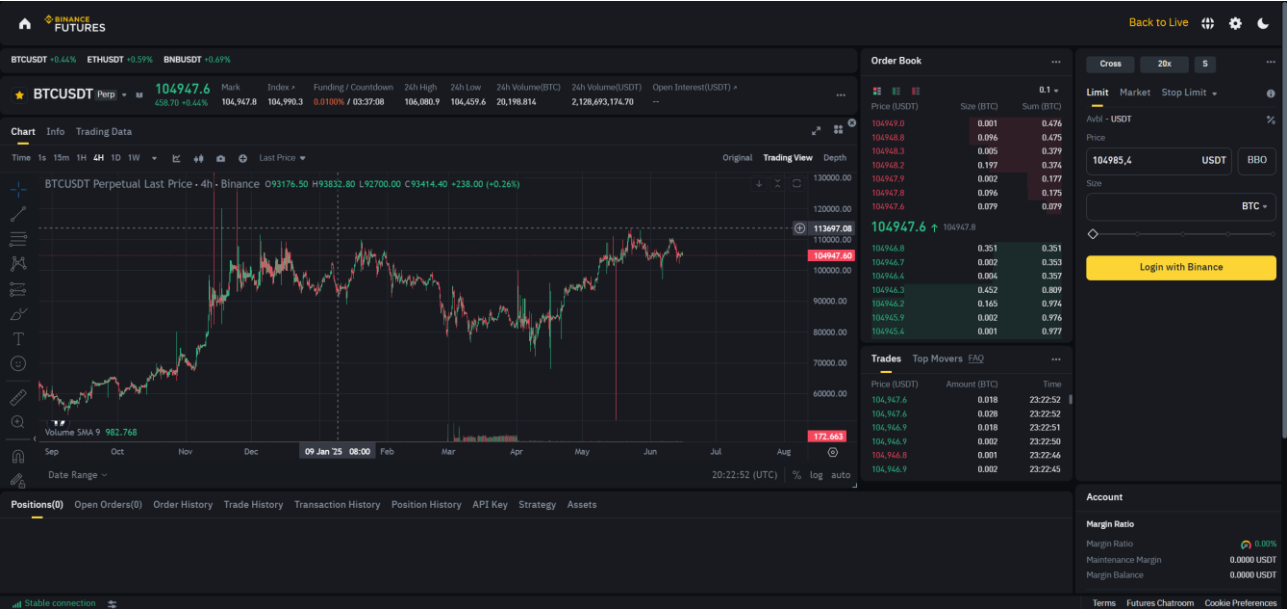


Рис. 1.1. Платформа Binance Futures Testnet[1]

Binance Futures Testnet – це платформа для демо-трейдингу, яка дозволяє користувачам імітувати торгівлю ф’ючерсами, використовуючи реальні ринкові дані через API біржі Binance [9]. Платформа надає віртуальний баланс для

виконання ордерів, що дозволяє користувачам тестувати торгові стратегії без фінансових ризиків.

До переваг даного програмного продукту можна віднести використання реальних ринкових даних, які надходять через API та забезпечують високу точність симуляції торгівлі. Платформа має зручний інтерфейс із підтримкою графіків і базових інструментів технічного аналізу, що дозволяє користувачам комфортно орієнтуватися у ринковій ситуації. Також варто зазначити підтримку різних типів ордерів, зокрема лімітних, ринкових та стоп-лімітних, що дає змогу моделювати різні сценарії торгівлі. Ще однією важливою перевагою є безкоштовний доступ до тестової платформи після проходження простої реєстрації, що робить її доступною для широкого кола користувачів.

Серед недоліків можна виділити обмежену функціональність у порівнянні з основною платформою Binance, що може бути недостатньо для повноцінного тестування складних стратегій. Крім цього, у платформі відсутня інтеграція з зовнішніми аналітичними інструментами, які часто використовуються професійними трейдерами. Також для новачків, які не мають попереднього досвіду роботи з біржами, налаштування середовища та навігація в інтерфейсі можуть виявитися дещо складними.

Наступний з таких додатків є TradingView Paper Trading (Рис. 1.2)

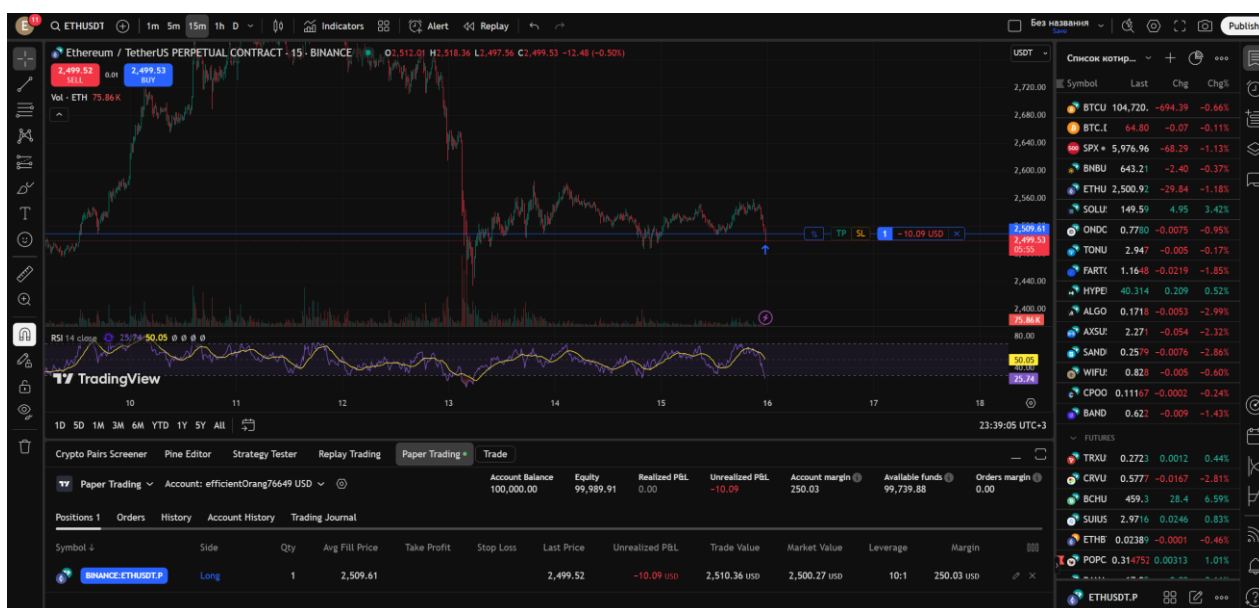


Рис. 1.2. Платформа TradingView Paper Trading [2]

До переваг даного програмного продукту можна віднести інтеграцію з розширеними інструментами технічного аналізу, серед яких графіки, індикатори та можливість використання власних скриптів на мові Pine Script. Користувачі можуть здійснювати демо-торгівлю різними видами активів, зокрема криптовалютами, акціями та валютними парами, що робить платформу універсальним інструментом для тренування. Інтерфейс є інтуїтивно зрозумілим і зручним навіть для новачків, а можливість створення власних стратегій дозволяє адаптувати торговий процес до індивідуальних потреб. Платформа працює на основі реальних ринкових даних, що забезпечує високу достовірність симуляції торгових операцій.

Серед недоліків можна виокремити обмежену підтримку складних типів ордерів, таких як ф'ючерси або маржинальна торгівля, що дещо звужує можливості для відпрацювання професійних стратегій. Також слід зазначити, що частина функціоналу доступна лише за умови платної підписки, що може бути бар'єром для деяких користувачів. Окрім цього, відсутня повноцінна інтеграція з реальними біржами, що обмежує глибину симуляції складних торгових сценаріїв і реального взаємодії з ринком. Наступний з таких додатків є BitMEX Testnet (Рис. 1.3)

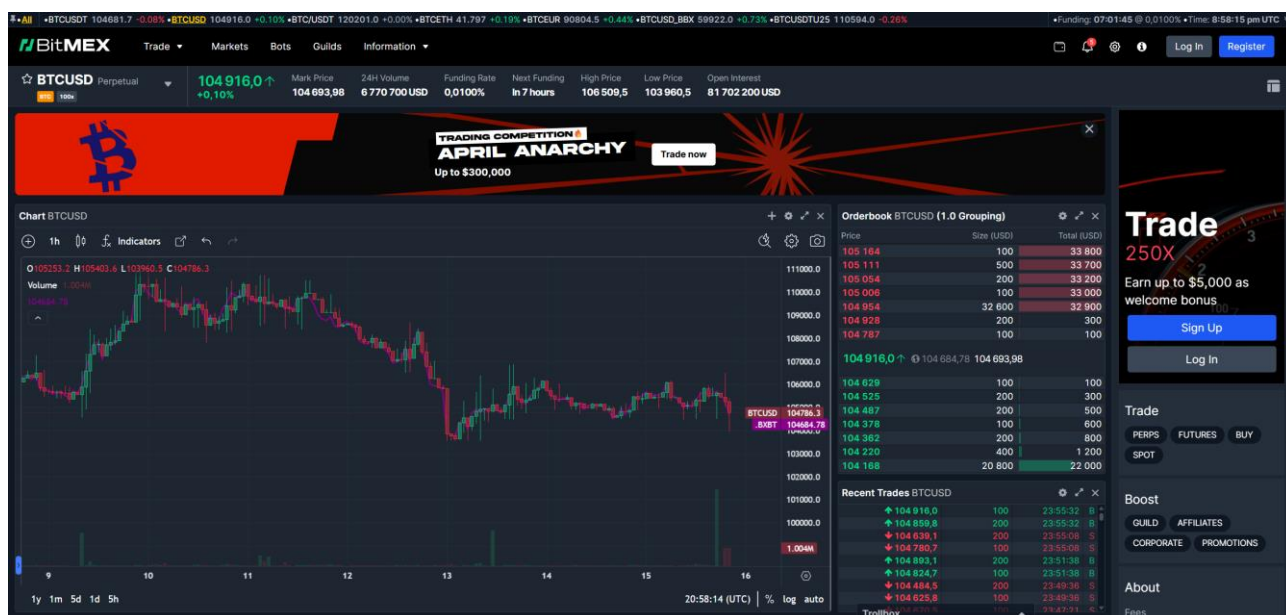


Рис. 1.3. Платформа BitMEX Testnet [3]

BitMEX Testnet – це тестова платформа біржі BitMEX, яка спеціалізується

на торгівлі криптовалютними деривативами. Платформа дозволяє користувачам імітувати ордери, використовуючи реальні ринкові дані через API, та тестувати стратегії на віртуальному балансі.

До переваг даного програмного продукту можна віднести високу точність симуляції торгових операцій, що досягається завдяки використанню реальних ринкових даних. Платформа підтримує маржинальну торгівлю та ф'ючерси, що дозволяє користувачам моделювати складні стратегії з урахуванням кредитного плеча. Також BitMEX Testnet надає безкоштовний доступ до тестового середовища, що робить його зручним для практичного навчання та тестування без фінансових ризиків. Окремою перевагою є можливість підключення та тестування автоматизованих торгових ботів за допомогою API, що корисно для розробників і трейдерів, які використовують алгоритмічні стратегії.

Серед недоліків варто зазначити складність інтерфейсу, який може бути незручним і заплутаним для новачків, що не мають досвіду роботи з подібними платформами. Крім того, вибір криптоактивів на тестнет-версії є обмеженим у порівнянні з реальною платформою, що зменшує можливості для повного тестування різноманітних торгових пар. Також відсутні вбудовані інструменти технічного аналізу, що змушує користувачів звертатися до сторонніх сервісів для аналізу ринку. Наступним прикладом серед подібних платформ є Bybit Testnet (Рис. 1.4)

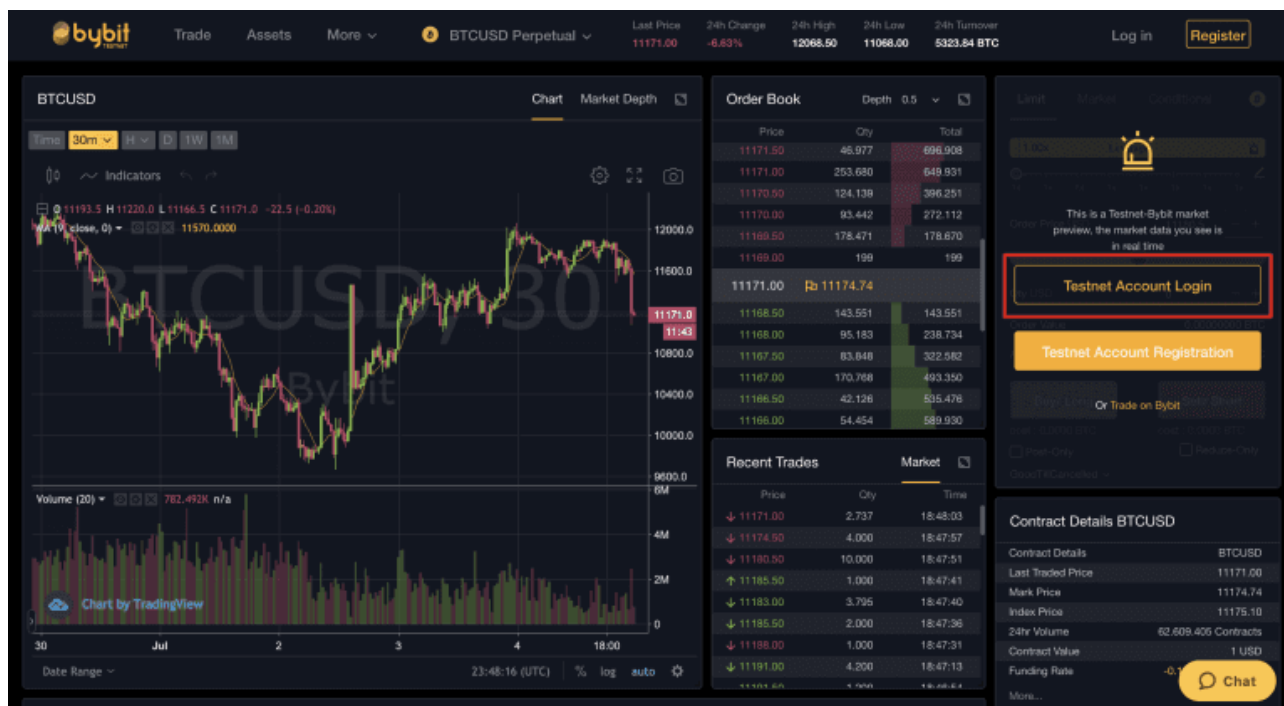
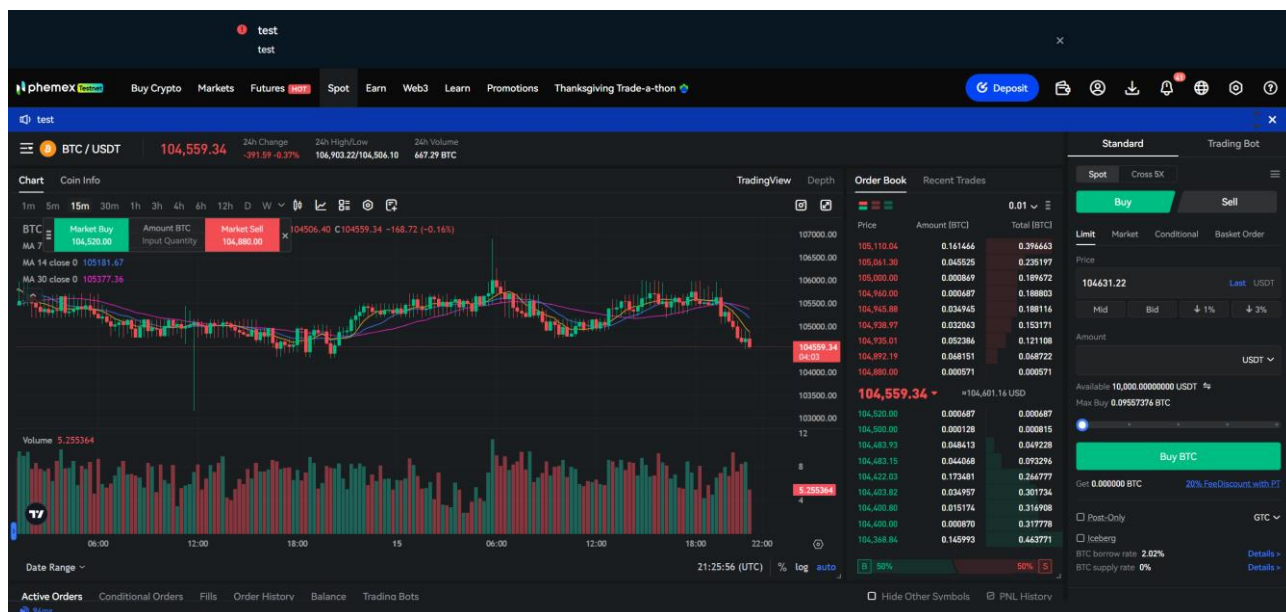


Рис. 1.4. Платформа Bybit Testnet [4]

До переваг даного програмного продукту можна віднести використання реальних ринкових даних, що дозволяє моделювати торгові ситуації з високою точністю. Платформа підтримує як спотову, так і ф'ючерсну торгівлю, що розширює можливості для тестування різноманітних стратегій. Доступ до тестового середовища є безкоштовним, і користувачі отримують віртуальний баланс, зокрема 10 000 доларів США та 1 BTC щодня, що дозволяє відпрацьовувати торгові навички без фінансових ризиків. Процес початку роботи є простим, оскільки достатньо зареєструватися на основній платформі. Крім того, існує можливість тестування автоматизованих стратегій за допомогою API, що є особливо корисним для розробників торгових ботів.

Серед недоліків можна зазначити обмежену підтримку аналітичних інструментів, що ускладнює глибокий технічний аналіз без використання сторонніх ресурсів. Також платформа орієнтована переважно на торгівлю деривативами, що може створити труднощі для новачків. Ще одним обмеженням є менша кількість доступних активів у порівнянні з повноцінною версією платформи, що звужує спектр можливих сценаріїв для тестування.

Наступний з таких додатків є Phemex Testnet (Рис. 1.5)



До переваг даного програмного продукту можна віднести простий і зручний інтерфейс, який особливо підходить для новачків, що лише починають знайомство з трейдингом. Платформа підтримує як спотову, так і ф'ючерсну торгівлю, що дозволяє користувачам тренуватися в різних ринкових умовах. Доступ до тестового середовища є безкоштовним, при цьому кожен користувач отримує віртуальний баланс, що дає змогу практикуватися без ризику втрати реальних коштів. Також важливою перевагою є можливість тестування торгових стратегій із використанням реальних ринкових даних, що дозволяє наблизити досвід до умов реального трейдингу.

Серед недоліків варто згадати обмежену інтеграцію з аналітичними інструментами, що може ускладнити виконання глибокого технічного аналізу без зовнішніх сервісів. Також кількість доступних криптоактивів на платформі є меншою порівняно з провідними біржами, що зменшує варіанти для відпрацювання різноманітних сценаріїв. Крім того, функціонал Phemex Testnet може бути недостатнім для тестування складних стратегій, які потребують гнучких інструментів управління ордерами чи ризиками.

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця порівняння платформ для демо трейдингу

Характеристика	Binance Futures Testnet	TradingView Paper Trading	BitMEX Testnet	Bybit Testnet	Phemex Testnet
Доступність тренування	+	+	+	+	+
Підтримка API	+	+	+	+	+
Типи ордерів	+ (лімітні, ринкові)	+ (лімітні, ринкові)	+ (маржинальні)	+ (ф'ючерсні)	+ (ф'ючерсні)
Аналітичні інструменти	+	+	-	-	-
Зручність для новачків	+	+	-	+	+
Підтримка активів	+ (криптовалюти)	+ (різні активи)	+ (деривативи)	+ (спот, ф'ючерси)	+ (спот, ф'ючерси)

Переглянувши платформи демо-трейдингу, можна виділити основні відмінності: деякі з них надають доступ до реальних ринкових даних через API, інші підтримують широкий спектр типів ордерів, а також відрізняються за зручністю інтерфейсу та орієнтацією на конкретну аудиторію, наприклад, новачків чи досвідчених трейдерів. Деякі платформи обмежені у функціоналі, тоді як інші пропонують розширені аналітичні інструменти, але з меншою підтримкою складних торгових операцій.

Враховуючи результати дослідження, було прийнято рішення створити платформу для демо-трейдингу криптоактивами, яка працює виключно з тестовими даними та не пов'язана з реальними фінансовими операціями. Платформа побудована на сучасних вебтехнологіях із використанням архітектури клієнт-сервер, що забезпечує гнучкість, масштабованість і зручність

обслуговування. Інтерфейс системи є інтуїтивно зрозумілим, а користувач має змогу взаємодіяти з симульованим торговим середовищем, яке відображає актуальні ринкові дані, отримані через API. Реалізовано підтримку тестування базових торгових стратегій у режимі симуляції, включаючи відкриття позицій, аналіз результатів та ведення статистики віртуального портфеля.

Отже, розроблений додаток буде функціонувати в тестовому режимі, де всі торгові операції є імітацією з використанням віртуальних активів. Це дозволяє забезпечити прозорість процесів, базовий рівень безпеки та надати користувачам можливість відпрацьовувати різні торгові стратегії й обирати криптоактиви без фінансових ризиків. Після виконання симульованих угод користувачі одразу отримують відповідні віртуальні кошти.

1.3. Обґрунтування вибору інструментальних засобів та вимоги до-апаратного забезпечення

Для створення додатку демо-торгівлі криптоактивами планується використати сучасну клієнт-серверну архітектуру з застосуванням таких технологій, як React, Node.js, Express, MongoDB та WebSocket. Такий підхід дозволить розділити відповідальність між клієнтською частиною, серверною логікою та системою збереження даних, а також забезпечить можливість обміну інформацією в режимі реального часу, що є критично важливим для торгових платформ.

Клієнтська частина буде реалізована за допомогою бібліотеки React [11], що дасть змогу створити швидкий, інтерактивний та зручний інтерфейс користувача. Завдяки компонентній структурі React буде забезпечено можливість масштабування та повторного використання модулів, таких як форма створення ордера, торговий графік, таблиця позицій тощо. Односторінкова архітектура (SPA) дозволить реагувати на дії користувача без

необхідності перезавантаження сторінки, що значно підвищить зручність взаємодії з системою.

Серверна частина буде побудована на основі Node.js [12] із використанням фреймворку Express [8], що дасть змогу реалізувати RESTful API [24, 25] для обробки запитів клієнта: реєстрація, авторизація, створення віртуальних ордерів, оновлення портфоліо, фіксація дій користувача тощо. Express також забезпечить гнучку маршрутизацію, підтримку middleware та зручну обробку помилок. Сервер взаємодіятиме з базою даних MongoDB для збереження основної інформації.

MongoDB [15] буде використано як головну систему управління даними. Завдяки гнучкому формату BSON, вона дозволить зберігати структуровану інформацію, включаючи профілі користувачів, історію угод, відкриті та закриті позиції, налаштування акаунтів і симульовані ринкові котирування.

Ключову роль у платформі відіграватиме WebSocket-з'єднання, яке буде реалізовано для забезпечення двостороннього обміну даними між клієнтом і сервером у режимі реального часу. Цей механізм дозволить миттєво передавати інформацію про зміну цін, оновлення статусів ордерів або виконання операцій. Наприклад, якщо в симульованому ринку відбудеться зміна котирування, ця інформація одразу буде надіслана клієнту без потреби в ручному оновленні або регулярних запитах.

Таким чином, архітектура системи передбачатиме наступну взаємодію: клієнтська частина на React надсилатиме HTTP-запити до сервера на базі Express через REST API [24, 25], а також встановлюватиме WebSocket-з'єднання [14] для отримання торгових даних у реальному часі. Сервер, у свою чергу, оброблятиме запити, зберігатиме інформацію в MongoDB [15] та ініціюватиме події для оновлення інтерфейсу на стороні клієнта. Така комбінація REST API та WebSocket забезпечить як стабільну обробку запитів, так і швидке оновлення даних без затримок.

Обрана архітектура є сучасною, гнучкою та масштабованою, що робить її оптимальною для реалізації симульованої торгової платформи. У майбутньому

також передбачено можливість розширення функціоналу: інтеграція з реальними біржовими API, впровадження системи аналітики, налаштування повідомлень і навіть реалізація мультикористувацьких симуляцій.

Висновки до першого розділу.

У першому розділі було проведено детальний аналіз предметної області та обґрунтовано доцільність створення програмного забезпечення для демо-торгівлі криптоактивами. Розглянуто ключові аспекти діяльності існуючих торгових платформ та визначено основні відмінності демо-середовищ від реального трейдингу, що дозволяє зосередитися на навчанні та тестуванні стратегій без фінансового ризику.

В результаті аналізу аналогічних рішень на ринку були виокремлені типові функціональні можливості, сильні та слабкі сторони платформ-конкурентів. Це дозволило визначити унікальні риси розроблюваної системи, серед яких: симуляція ринку в реальному часі, створення віртуальних ордерів, відстеження динаміки активів, облік віртуального балансу та миттєві повідомлення через WebSocket.

Обґрунтовано вибір архітектури програмного забезпечення, яка базується на сучасному клієнт-серверному підході з використанням стеку технологій React, Node.js, Express та MongoDB. Також до архітектури інтегровано WebSocket для забезпечення передачі ринкових подій і оновлень в режимі реального часу, що критично важливо для повноцінного відтворення торгового середовища.

У результаті проведеного аналізу сформовано технічне підґрунтя для подальшого проєктування, реалізації та тестування демо-платформи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПЛАТФОРМИ ДЛЯ ДЕМО ТРЕЙДИНГУ КРИПТОАКТИВАМИ

2.1. Визначення варіантів використання та структури системи

Проектована система демо-торгівлі криптовалютами передбачає створення модуля, який надає користувачеві широкий спектр функцій для взаємодії з віртуальним торговим середовищем. Основним користувачем є трейдер (Користувач), який після входу в систему отримує доступ до ряду операцій, пов'язаних із торгівлею, налаштуванням профілю та керуванням портфелем. Також частина операцій автоматизована або ініціюється Системою.

Взаємодія користувача із системою описується за допомогою UML діаграми варіантів використання [26, 27], наведеної на (рис. 2.1.) На ній представлено ключові функціональні сценарії, які доступні після авторизації.

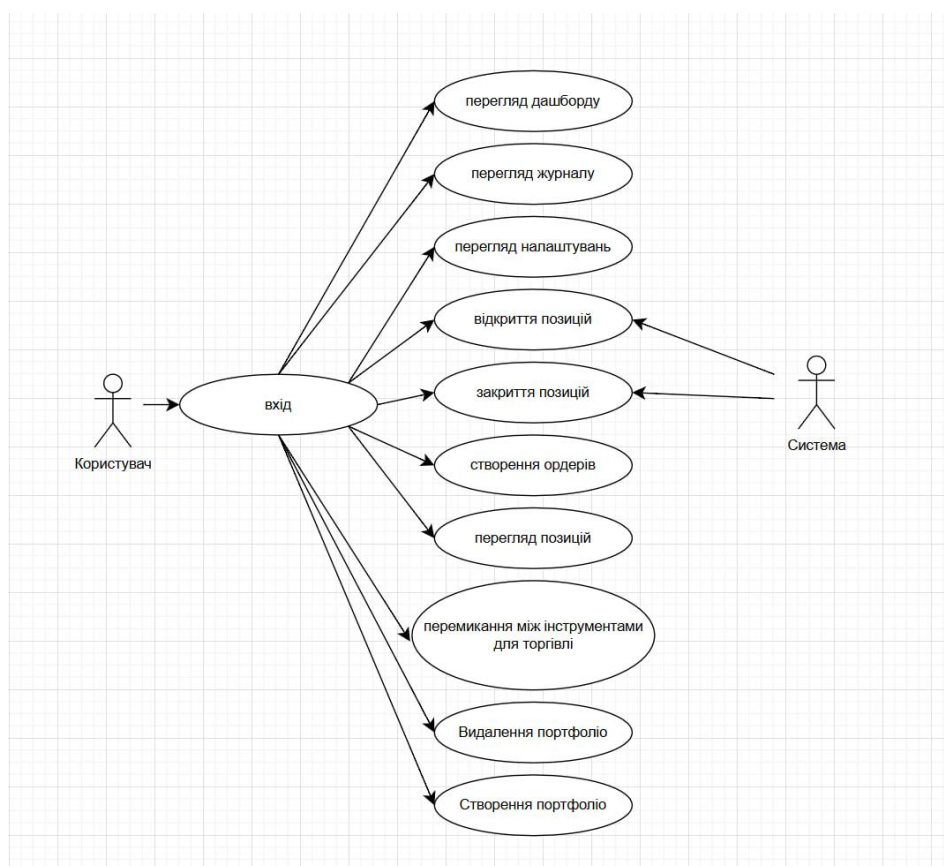


Рис. 2.1 – Діаграма варіантів використання користувача в модулі демо-торгівлі

Як видно з діаграми, після входу до системи користувач отримує доступ до таких можливостей як перегляд дашборду, що дозволяє бачити зведену інформацію про стан ринку, баланс користувача та активні позиції. Також користувач може переглядати журнал, у якому зберігається хронологія дій, торгових операцій та подій. У розділі налаштувань доступна можливість редагувати портфолію, змінювати пароль та обирати кольорову тему інтерфейсу. Крім того, користувач має змогу створювати ордери, подаючи заявки на купівлю або продаж криптоактивів з віртуального портфеля, а також переглядати та управляти відкритими позиціями. Відкриття та закриття позицій відбувається частково в автоматичному режимі залежно від змін ринкової ситуації. Також передбачена можливість перемикання між різними інструментами для торгівлі, що дозволяє обирати активний ринок або криптовалюту в межах торгового терміналу. Окремо реалізована функція створення портфолію, яка дозволяє сформувати початковий склад віртуальних активів, а також функція видалення портфолію для скидання або очищення наявних активів.

Усі згадані дії реалізовано у вигляді окремих функціональних модулів, з якими користувач взаємодіє через інтерфейс, створений за допомогою React [11]. Запити обробляються на стороні сервера за допомогою Node.js у зв'язці з Express, а для збереження стану ордерів, позицій, журналів і портфелів використовується база даних MongoDB [15]. Таким чином, діаграма варіантів використання допомагає чітко окреслити межі функціональності системи, визначити ролі користувачів і надати структуроване уявлення для реалізації кожного зі сценаріїв у програмному коді та загальній логіці застосунку [21].

Функціональні вимоги:

Функціональні вимоги визначають, що саме повинна виконувати система. Для платформи демо-торгівлі криптоактивами основними функціональними вимогами є забезпечення реєстрації нового користувача з використанням електронної пошти та паролю. Користувач має мати можливість увійти до

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						22

системи через форму авторизації. Після входу користувач отримує доступ до особистого кабінету. Система повинна надавати можливість перегляду дашборду, де користувач може бачити зведену інформацію про поточний баланс, відкриті позиції та стан ринку. Користувач повинен мати змогу переглядати ринкові дані щодо різних криптовалют, включаючи ціну, обсяг та зміну. Система має дозволяти перемикання між різними торговими інструментами. Крім того, користувач повинен мати змогу створювати, переглядати і скасовувати ордери, створювати ордер на купівлю або продаж активу з такими параметрами, як тип, сума і ціна. Користувач також повинен мати можливість відкривати позиції згідно з поточними ринковими умовами, а система має дозволяти закривати позиції з урахуванням симульованої ціни. Важливо, щоб користувач міг створити новий віртуальний портфель та видалити існуючий. Передбачено перегляд журналу подій і історії торгів, де користувач матиме доступ до всіх дій і подій, таких як створення ордера, відкриття позиції та зміни балансу. Користувач також повинен мати змогу переглядати та змінювати базові параметри профілю, включаючи пароль, кольорову тему та портфоліо.

Нефункціональні вимоги описують, як повинна працювати система. Встановлено, що система повинна обробляти запити не довше ніж за дві секунди при навантаженні до ста одночасних користувачів. Сервіс має бути доступним безперервно цілодобово і сім днів на тиждень з гарантованим часом безвідмовної роботи не менше дев'яносто дев'яти відсотків. Для безпеки паролі користувачів повинні зберігатися у зашифрованому вигляді, наприклад, із використанням bcrypt, а аутентифікація повинна реалізовуватися через JWT-токени. Доступ до API повинен бути захищеним за допомогою протоколу HTTPS. Архітектура системи має підтримувати горизонтальне масштабування шляхом розгортання окремих сервісів на кількох вузлах. Інтерфейс користувача має бути адаптивним і коректно працювати як на десктопах, так і на мобільних пристроях. Він повинен бути зрозумілим і інтерактивним, реалізованим за допомогою React. Оновлення даних у системі, таких як ціни, статуси ордерів і баланс, повинні відбуватися

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						23

автоматично у режимі реального часу через WebSocket без необхідності оновлювати сторінку вручну. Для забезпечення надійності всі критичні операції, зокрема відкриття позицій і збереження даних, мають супроводжуватися обробкою помилок та логуванням.

Системні вимоги:

Системні вимоги описують апаратне та програмне забезпечення, необхідне для роботи платформи. Програмні вимоги для серверної частини передбачають використання операційної системи Windows, Ubuntu 20.04 або іншої UNIX-подібної операційної системи. Потрібно мати встановлену версію Node.js не нижче 18, а також використовувати Express.js і MongoDB версії 6.0 або вище. Для забезпечення зв'язку в реальному часі застосовується WebSocket через бібліотеку socket.io. Для контролю версій проекту використовується система Git. [23] Програмні вимоги для клієнтської частини включають використання React версії 18 і вище, WebSocket-клієнта, а також REST API, що реалізується через бібліотеки Axios [19] або Fetch. Для коректної роботи платформи необхідно мати сучасний браузер, наприклад Chrome, Firefox або Edge. Щодо апаратних вимог для тестового сервера, то рекомендується мати процесор з двома віртуальними ядрами, оперативну пам'ять об'ємом чотири гігабайти, твердотільний накопичувач з об'ємом не менше сорока гігабайтів, а також стабільне підключення до Інтернету з пропускнуою здатністю не менше десяти мегабіт на секунду.

2.2. Розробка бази даних системи

Одним із ключових етапів розробки платформи для демо-торгівлі криптоактивами є створення ефективної та гнучкої бази даних, яка забезпечує надійне зберігання, обробку та доступ до інформації про користувачів, торгові операції, ринкові позиції та віртуальні активи. Вибір було зупинено на MongoDB [15] —

документоорієнтованій NoSQL-системі керування базами даних, що забезпечує масштабованість, високу продуктивність і зручність роботи з JSON-подібними структурами даних.

Для проєкту було створено базу даних `crypto_db`, яка містить 5 основних колекцій:

1. `users` — інформація про користувачів;
2. `orders` — заявки на купівлю/продаж криптоактивів;
3. `positions` — відкриті торгові позиції користувачів;
4. `trades` — виконані угоди;
5. `portfolios` — віртуальні портфелі користувачів;

Опис кожної з колекцій бази даних наведений у таблицях 2.8 – 2.11.

Таблиця 2.1

Опис колекції Users

Поле	Тип	Опис
<code>_id</code>	ObjectId	Унікальний ідентифікатор користувача
<code>username</code>	string	Ім'я користувача
<code>password</code>	string	Хешований пароль (bcrypt)

Таблиця 2.2

Опис колекції Orders

Поле	Тип	Опис
<code>_id</code>	ObjectId	Унікальний ідентифікатор ордера
<code>side</code>	string	Напрямок ордера: "long" або "short"
<code>leverage</code>	int	Значення кредитного плеча
<code>price</code>	int	Ціна входу в позицію
<code>takeProfit</code>	int / null	Рівень фіксації прибутку
<code>stopLoss</code>	int / null	Рівень обмеження збитків

Таблиця 2.3

Опис колекції Positions

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор позиції
side	string	Напрямок позиції: "long" або "short"
pair	string	Торгова пара, наприклад "BTCUSDT"
leverage	int	Кредитне плече
price	int	Ціна відкриття позиції
marge	int	Сума, виділена під позицію
takeProfit	int / null	Ціна фіксації прибутку
stopLoss	int / null	Ціна обмеження збитків
userId	ObjectId	ID користувача, який відкрив позицію

Таблиця 2.4

Опис колекції Trades

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор угоди
userId	ObjectId	Ідентифікатор користувача, який здійснив угоду
portfolioId	ObjectId	Ідентифікатор портфеля, у межах якого виконано операцію
side	string	Напрямок угоди: "long" або "short"

Продовження таблиці 2.4

pair	string	Торгова пара (наприклад, "BTCUSDT")
openPrice	number	Ціна відкриття позиції
closePrice	number	Ціна закриття позиції
pnl	number	Прибуток або збиток за угодою (Profit & Loss)
fee	number	Комісія за виконання угоди
marge	number	Обсяг маржі (віртуальних коштів), що були виділені на позицію
leverage	number	Значення кредитного плеча, що було використано

Таблиця 2.5

Опис колекції Portfolios

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор портфоліо
name	string	Назва портфеля
initialBalance	int	Початковий баланс
balance	int	Поточний віртуальний баланс
userId	ObjectId	Ідентифікатор власника портфеля

2.3. Проектування та реалізація алгоритмів роботи системи

На основі визначених функціональних вимог та моделювання поведінки користувача було реалізовано ключові алгоритми, що забезпечують роботу платформи для демо-торгівлі криптоактивами. Основна увага під час проектування приділялася логіці переходів між етапами взаємодії, автоматизації бізнес-процесів, обробці даних у реальному часі та дотриманню моделі (рис.2.2)



Рис. 2.2 – Клієнт-серверна модель взаємодії

Для візуалізації логіки взаємодії з інтерфейсом платформи було побудовано діаграму активності (рис. 2.3), яка відображає всі основні переходи між етапами життєвого циклу сесії користувача — від реєстрації до взаємодії з торговими функціями.

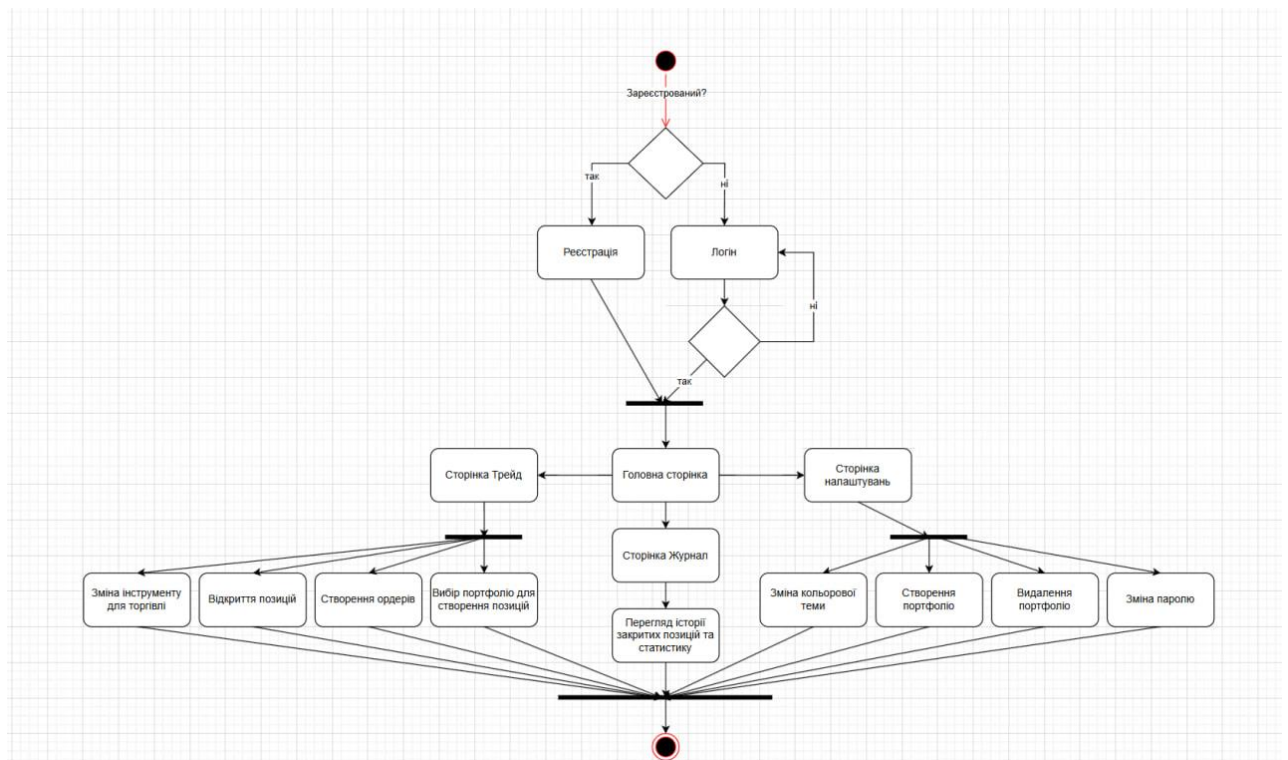


Рис. 2.3 – UML [26, 27] Діаграма активності

Авторизація та вхід у систему

Першим алгоритмічним блоком системи є перевірка стану користувача: чи має він обліковий запис. Якщо ні — відкривається форма реєстрації, яка після заповнення створює нового користувача в колекції users. Якщо користувач вже зареєстрований — система перенаправляє його на форму логіну, де здійснюється перевірка облікових даних. Паролі хешуються з використанням bcrypt [17], а після вдалого входу видається JWT-токен [16] для доступу до особистих функцій.

Навігація по інтерфейсу

Після входу користувач потрапляє на головну сторінку, яка є навігаційним центром системи. З цієї сторінки реалізовано перехід до сторінки Трейд, де користувач взаємодіє з ринковими даними, відкриває позиції та створює ордери. Також доступна сторінка Журнал, яка містить історію закритих позицій, торгову статистику та інформацію про прибуток і збиток. Крім того, є сторінка Налаштувань, що дозволяє редагувати профіль користувача, змінювати тему інтерфейсу та керувати портфелями. Кожен перехід між цими сторінками ініціюється клієнтською частиною, реалізованою на React, яка надсилає відповідні запити до

серверного API.

Алгоритми взаємодії з торговими функціями

Відкриття позиції відбувається у кілька кроків. Спочатку користувач вибирає торгову пару та портфель, з яким він планує працювати. Потім вводить параметри позиції, такі як напрямок — лонг або шорт, кредитне плече та розмір маржі. Після натискання кнопки Відкрити позицію відбувається перевірка правильності введених даних. Якщо валідація пройшла успішно, сервер створює новий запис у колекції `positions`, а баланс відповідного портфеля зменшується на суму маржі. Всі зміни зберігаються, і оновлення миттєво передаються користувачу через `WebSocket` для актуалізації інформації.

Закриття позиції відбувається наступним чином. Користувач натискає кнопку Закрити позицію, після чого система фіксує ціну на момент закриття, імітуючи реальні ринкові умови. Потім розраховується прибуток або збиток, який отримав користувач від цієї угоди. Результати записуються у новий документ у колекції `trades`. Позиція видаляється з колекції `positions`, а баланс портфеля оновлюється відповідно до результатів закриття позиції.

Алгоритми управління портфелем

Створення портфеля відбувається шляхом додавання нового запису до колекції `portfolios` із початковим балансом. Видалення портфеля передбачає видалення відповідного документа з бази даних. Якщо в портфелі є відкриті позиції, користувачу видається попередження про це. Зміна теми або пароля здійснюється через оновлення відповідних полів у колекції `users`.

Перегляд журналу

Користувач переходить на сторінку журналу, де через API отримуються всі записи з колекції `trades`, відфільтровані за `userId`. Дані сортуються за датою, обчислюється `PnL` і виводиться загальна статистика.

Оптимізація та WebSocket

Усі критичні події, які мають оновлюватись в інтерфейсі в режимі реального часу — наприклад, зміна ринкової ціни або відкриття/закриття позиції — реалізовано через `WebSocket`-з'єднання. Це дозволяє уникнути частого

опитування сервера та забезпечити користувачеві інтерактивний досвід, близький до реальної біржі.

2.4. Проектування серверної частини та API

Серверна частина платформи для демо-торгівлі криптоактивами реалізована з використанням Node.js [12] у поєднанні з фреймворком Express.js [13]. Уся логіка обробки запитів, доступу до бази даних та взаємодії з API сторонніх сервісів структурована через REST-маршрути [24, 25] та WebSocket. Доступ до більшості маршрутів обмежено через JWT-аутентифікацію.

Авторизація та реєстрація

Користувач має змогу зареєструвати новий обліковий запис (маршрут /register) або увійти до системи через /login. Паролі зберігаються у хешованому вигляді за допомогою bcrypt [17], а після успішної аутентифікації генерується JWT-токен [16], який клієнт використовує для доступу до приватних маршрутів. Токен додається до кожного запиту у заголовок Authorization.

Захист маршрутів та перевірка токенів

Усі маршрути, які обробляють чутливі дії (отримання позицій, створення ордерів, доступ до портфелів тощо), захищені middleware-функцією auth, яка перевіряє валідність JWT-токена. Після перевірки токена отримується userId, що використовується для персоналізованого доступу до даних у базі MongoDB [15].

Основні маршрути серверної частини

Маршрут positions з методом GET повертає список відкритих позицій користувача. Маршрут position/open з методом POST відкриває нову позицію за переданими параметрами. Маршрут position/close з методом POST закриває відкриту позицію користувача. Маршрут order з методом POST створює лімітний ордер. Маршрут orders з методом GET отримує всі лімітні ордери користувача. Маршрут order/delete з методом POST скасовує лімітний ордер. Маршрут trades

з методом GET повертає історію завершених угод користувача. Маршрут create-portfolio з методом POST створює новий віртуальний портфель. Маршрут portfolio-list з методом GET повертає список портфелів користувача. Маршрут update-portfolio з методом POST змінює назву портфеля. Маршрут delete-portfolio з методом POST видаляє вибраний портфель. Маршрут change-password з методом POST змінює пароль користувача після перевірки чинного пароля. Маршрут chart з методом GET отримує свічки kline з API біржі Bybit для побудови графіків.

Інтеграція з API Bybit

Одним із важливих елементів реалізації є отримання реальних ринкових даних з біржі Bybit. За допомогою REST-запиту /chart сервер надсилає запит до Bybit API [6] для отримання історичних свічок за заданим символом. Дані конвертуються у формат, зручний для побудови графіків на клієнті (час, ціна відкриття, максимум, мінімум, ціна закриття) та повертаються у JSON-форматі.

Передача оновлень через WebSocket

Крім REST API, у проєкті використовується WebSocket [14] для миттєвого надсилання оновлень до клієнтів: зміна ціни, оновлення балансу після закриття позиції, підтвердження створення або виконання ордера. Це забезпечує роботу платформи в режимі реального часу та формує ефект живого ринку для демо-торгівлі.

Обробка помилок

Усі серверні маршрути обгорнуті в конструкції try-catch, а також мають внутрішній catch для промісів, що дозволяє безпечно обробляти неочікувані винятки та повертати статуси помилок (500) із журналюванням у консоль. Таким чином, серверна частина проєкту реалізована як безпечний та масштабований API-рівень, що забезпечує обробку всіх критичних торгових операцій, обмін ринковими даними з Bybit, авторизацію, та роботу з портфелями користувача.

2.5. Реалізація функціоналу платформи для демо трейдингу криптоактивами

Платформа для демо-торгівлі криптоактивами була реалізована як повноцінний вебзастосунок із розділенням на клієнтську та серверну частини. Головною метою реалізації є надання користувачам зручного інтерфейсу для симульованої торгівлі криптовалютами в умовах, максимально наближених до реального біржового середовища, але без фінансового ризику.

З боку клієнта платформа реалізована з використанням бібліотеки React [11], що забезпечує створення динамічного, інтуїтивно зрозумілого та швидкодіючого інтерфейсу. Весь інтерфейс побудований як односторінковий застосунок (SPA), що дозволяє здійснювати навігацію без перезавантаження сторінок.

Користувач після проходження авторизації потрапляє на головну сторінку платформи, звідки отримує доступ до основних функціональних модулів: торгового терміналу, журналу угод, керування портфелем та налаштувань облікового запису. Авторизація реалізована на стороні клієнта та сервера з використанням JWT (JSON Web Token) [16]. Після успішного входу користувачеві видається токен, який зберігається у локальному сховищі браузера й автоматично додається до всіх наступних API-запитів через заголовок Authorization. Це дозволяє забезпечити захищений доступ до персоналізованого функціоналу, а також ізолювати дії кожного користувача в межах його облікового запису. Нижче наведено приклад коду, який реалізує автентифікацію на клієнтській стороні за допомогою React та API-запиту на сервер (рис.2.4; рис.2.5):

```
const signUp = (event: FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  api
    .post("/register", {
      email,
      password,
    })
    .then((response) => {
      localStorage.setItem("token", response.data.token);
      localStorage.setItem("user", JSON.stringify({ email }));
      window.location.href = "/";
    })
    .catch((error) => {
      if (
        error.response.data.error === "Account already exist with this email"
      ) {
        setUserExistErrorMsg(error.response.data.error);
      } else {
        setShowErrorMsg(true);
      }
    });
};
```

Рис. 2.4 – Реєстрація нового користувача

```
const signIn = (event: FormEvent<HTMLFormElement>) => {
  event.preventDefault();
  api
    .post("/login", {
      email,
      password,
    })
    .then((response) => {
      localStorage.setItem("token", response.data.token);
      localStorage.setItem(
        "user",
        JSON.stringify({ email })
      );
      window.location.href = "/";
    })
    .catch((error) => {
      if (error.response.data.error === "User not found") {
        setUserNotFoundMsg(true);
      } else if (error.response.data.error === "Invalid password") {
        setInvalidPasswordMsg(true);
      } else {
        setShowErrorMsg(true);
      }
    });
  console.log(error.response.data.error);
};
```

Рис. 2.5 – Авторизація користувача

Коли користувач потрапляє на головну сторінку платформи (дашборд), він одразу бачить інтерактивні віджети з актуальними ринковими даними, отриманими через публічний API сервісу CoinMarketCap [8]. Ці віджети відображають ключову інформацію про найпопулярніші криптовалюти, зокрема: поточну ціну, зміну за 24 години, добовий обсяг торгів, а також ринкову капіталізацію. Вони

автоматично завантажуються після ініціалізації компонента дашборду, що дає змогу користувачу миттєво оцінити стан ринку ще до переходу до торгового терміналу. Дані представлені у вигляді таблиць та графічних блоків, інтегрованих у загальний інтерфейс платформи. Така реалізація створює ефект роботи з реальним ринком і підвищує зручність та інформативність демо-торгівлі. Приклад реалізації сторінки з віджетами на основі CoinGecko API [7] наведено на (рис 2.6.)

```

25 <gecko-coin-price-marquee-widget
26   locale="en"
27   outlined="true"
28   coin-ids=""
29   initial-currency="usd"
30 />
31 <div className="container mt-4">
32   <div className="row">
33     <div className="col-md-4">
34       { /* @ts-expect-error */ }
35       <gecko-coin-list-widget
36         locale="en"
37         outlined="true"
38         coin-ids=""
39         initial-currency="usd"
40       />
41     </div>
42     <div className="col-md-4">
43       { /* @ts-expect-error */ }
44       <gecko-coin-converter-widget
45         locale="en"
46         outlined="true"
47         initial-currency="usd"
48       />
49     </div>
50   </div>

```

Рис. 2.6 – Віджети на основі CoinGecko API [7]

Окремо варто виділити функціонал торгового терміналу, який дозволяє здійснювати основні дії користувача, пов'язані з торгівлею. На цій сторінці користувач може обрати торговий інструмент (наприклад, BTC/USDT), створити ордер на купівлю або продаж, вказати параметри кредитного плеча, рівні Take Profit і Stop Loss. У випадку створення маркет-ордера, позиція відкривається миттєво. Якщо створюється лімітний ордер, він зберігається до виконання умов. Для збереження станів використовується база даних MongoDB [15], у якій реалізовані колекції orders, positions, trades та portfolios.

У процесі торгівлі система дозволяє відкривати та закривати позиції. При відкритті позиції враховується поточна ринкова ціна, розмір маржі та кредитне плече. Після закриття позиції система розраховує фінансовий результат угоди (PnL — прибуток або збиток), який зберігається в колекції trades. Усі зміни миттєво передаються на клієнт через WebSocket-з'єднання, що забезпечує оновлення інформації в реальному часі без потреби в перезавантаженні сторінки. Приклад реалізації відкриття позиції наведено на (рис 2.7.)

```
const openPosition = (
  side: "long" | "short",
  leverage: number,
  price: number,
  marge: number,
  takeProfit: number | null,
  stopLoss: number | null,
  portfolioId: string
) => {
  api
    .post("/position/open", {
      side,
      pair,
      leverage,
      price,
      marge,
      takeProfit,
      stopLoss,
      portfolioId,
    })
    .then(fetchPositions);
};
```

Рис. 2.7 – Приклад реалізації відкриття позиції

Функціонал портфеля реалізований у вигляді окремого модуля. Кожен користувач може створити один або декілька віртуальних портфелів, які мають свій баланс, історію позицій та прив'язку до торгових ордерів. Це дозволяє тестувати різні стратегії незалежно одна від одної. Система підтримує створення, редагування назви та видалення портфелів. Баланс портфеля оновлюється автоматично після кожної операції — відкриття, закриття позиції або нарахування прибутку/збитку. Приклад реалізації створення торгового портфелю наведено на (рис 2.8.)

```

const createPortfolio = (name: string, balance: number) => {
  api
    .post("/create-portfolio", {
      portfolio: {
        name,
        initialBalance: balance,
        balance,
      },
    })
    .then(() => {
      fetchPortfolioList();
    });
};

```

Рис. 2.8 – Приклад реалізації створення торгового портфелю

Крім торгових операцій, реалізовано модуль "Журнал", у якому користувач може переглядати історію всіх завершених угод, фільтрувати записи за датою, торговою парою або фінансовим результатом. Це дає змогу аналізувати ефективність демо-стратегії та проводити самооцінку. Приклад реалізації показу списку трейдів наведено на (рис 2.9.)

```

{trades.length === 0 && <p className="text-center">No Trades</p>}
<table className="trades table table-bordered">
  <thead>
    <tr>
      <th>№</th>
      <th>Side</th>
      <th>Pair</th>
      <th>Margin</th>
      <th>Open Price</th>
      <th>Close Price</th>
      <th>PNL (USDT)</th>
    </tr>
  </thead>
  <tbody className="table-group-divider">
    {trades.map((trade, index) => (
      <tr key={trade.id}>
        <td>{index + 1}</td>
        <td className={trade.side === "long" ? "positive" : "negative"}>
          {trade.side}
        </td>
        <td>{trade.pair}</td>
        <td>{trade.margin}</td>
        <td>{trade.openPrice}</td>
        <td>{trade.closePrice}</td>
        <td className={trade.pnl > 0 ? "positive" : "negative"}>
          {trade.pnl}
        </td>
      </tr>
    ))}
  </tbody>
</table>
</>
);
}

```

Рис. 2.9 – Приклад реалізації показу списку трейдів

Модуль налаштувань дозволяє змінювати тему інтерфейсу (світла/темна), мову, а також редагувати пароль. Усі ці дії супроводжуються валідацією даних на клієнтському та серверному рівнях. Приклад зміни паролю та зміни кольорової теми застосунку наведено на (рис 2.10.)

```
function Settings() {
  const [theme, setTheme] = useState(localStorage.getItem("theme") || "light");
  const [currentPassword, setCurrentPassword] = useState("");
  const [newPassword, setNewPassword] = useState("");
  const [showErrorMsg, setShowErrorMsg] = useState(false);
  const [showSuccessMsg, setShowSuccessMsg] = useState(false);

  const toggleTheme = () => {
    setTheme((prev) => (prev === "light" ? "dark" : "light"));
  };

  const changePassword = () => {
    api
      .post("/change-password", {
        email: JSON.parse(localStorage.getItem("user") as string).email,
        currentPassword,
        newPassword,
      })
      .then(() => {
        setCurrentPassword("");
        setNewPassword("");
        setShowSuccessMsg(true);
      })
      .catch((error) => {
        setShowErrorMsg(error.response.data.error);
      });
  };

  useEffect(() => {
    document.documentElement.setAttribute("data-bs-theme", theme);
    localStorage.setItem("theme", theme);
  }, [theme]);
}
```

Рис. 2.10 – Приклад зміни паролю та зміни кольорової теми застосунку

У межах проєкту реалізовано симулятор ринку, який забезпечує динамічне оновлення цін криптовалют у режимі реального часу. На відміну від моделей з псевдовипадковою генерацією даних, у даній реалізації використовується реальне ринкове джерело — публічне API біржі Bybit, з якого періодично отримується актуальна інформація про курси популярних криптоактивів (наприклад, BTC/USDT, ETH/USDT). Це дозволяє користувачам взаємодіяти з максимально наближеним до реального ринку торговим середовищем, але у безпечному, демо-форматі.

Приклад отримання цін з Bybit API [6] та передача їх на клієнтську частину за допомогою Websocket наведено на (рис 2.11.)

```
wss.on("connection", (clientWs) => {
  console.log("Клієнт підключився");

  // Єдиний глобальний обробник для всіх оновлень
  bybitSocket.on("update", (data) => clientWs.send(JSON.stringify(data)));

  clientWs.on("message", async (msg) => {
    const data = JSON.parse(msg);

    if (data.type === "subscribe") {
      const symbol = data.symbol;

      // Додаємо нову підписку на символ
      clients.push({ ws: clientWs, symbol });

      if (!subscriptions[symbol]) {
        subscriptions[symbol] = 1;

        bybitSocket.subscribeV5([`kline.5.${symbol}`], "linear");
        console.log(`Підписка на ${symbol}`);
      } else {
        subscriptions[symbol]++;
      }
    }

    if (data.type === "unsubscribe") {
      const symbol = data.symbol;
      clients = clients.filter((c) => c.ws !== clientWs || c.symbol !== symbol);

      console.log(subscriptions);

      if (subscriptions[symbol]) {
        subscriptions[symbol]--;
        if (subscriptions[symbol] === 0) {
          delete subscriptions[symbol];
          await bybitSocket.unsubscribeV5([`kline.5.${symbol}`], "linear");
          console.log(`Відписка від ${symbol}`);
        }
      }
    }
  });
});
```

Рис. 2.11 – Приклад отримання цін з Bybit API за допомогою Websocket

Реалізація цього функціоналу відбувається на стороні сервера. Серверна частина періодично надсилає HTTP-запити до REST API Bybit, обробляє відповідь, формує структуровані дані і транслює їх до всіх підключених клієнтів через WebSocket. Це дає змогу реалізувати push-оновлення без опитування серверу з боку клієнта.

Таким чином, було реалізовано повноцінну інтеграцію з біржею Bybit, яка дозволяє отримувати живі ринкові котирування та передавати їх на клієнтську частину платформи за допомогою WebSocket-протоколу. Це не лише підвищує реалістичність симуляції, а й дозволяє користувачу своєчасно реагувати на зміну ринку при відкритті позицій або створенні ордерів, наближаючи досвід демо-торгівлі до реальних умов криптовалютної біржі.

Висновки до другого розділу

У другому розділі було детально розглянуто процес проєктування та реалізації платформи для демо-торгівлі криптоактивами. Визначено основні функціональні та нефункціональні вимоги до системи, обґрунтовано вибір інструментів і технологій, зокрема стеку React, Node.js, Express та MongoDB. Створено архітектуру системи, розроблено структуру бази даних, а також побудовано UML-діаграми [26, 27], які ілюструють логіку взаємодії користувача з платформою. Особливу увагу було приділено реалізації серверної частини, побудованої за REST-архітектурою з використанням JWT-аутентифікації та WebSocket-з'єднання [14] для передачі ринкових даних у реальному часі. Також реалізовано інтеграцію з API біржі Bybit, що забезпечує високу наближеність до реального торгового середовища. На основі отриманих даних користувач може здійснювати відкриття позицій, створювати ордери, керувати портфелем і переглядати власну торгову статистику.

Результатом розробки став функціональний вебзастосунок, що надає користувачу повноцінне середовище для тренування навичок торгівлі в умовах без ризику, а також містить всі ключові елементи сучасної криптобіржі в симульованому форматі.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ДЕМОТРЕЙДИНГУ КРИПТОАКТИВАМИ

3.1. Порядок встановлення та налаштування параметрів платформи

Розроблена платформа для демо-трейдингу криптовалютами є сучасним веб-додатком, реалізованим за архітектурою клієнт сервер із підтримкою роботи в режимі реального часу. Такий підхід дозволяє ефективно розділити функціональні обов'язки між клієнтською частиною яка відповідає за інтерфейс користувача та серверною яка відповідає за бізнес логіку базу даних і інтеграцію з API біржі Bybit. Уся структура проєкту зберігається в одному репозиторії та складається з двох основних частин. Клієнтська частина розташована у директорії client написана на бібліотеці React.js і відповідає за візуалізацію інтерфейсу та взаємодію з API через HTTP і WebSocket. Серверна частина знаходиться у директорії server реалізована з використанням Node.js Express бібліотеки WebSocket та бази даних MongoDB з інтеграцією до API Bybit для отримання ринкових котирувань і симуляції торгових дій.

Для запуску платформи необхідна мінімальна серверна конфігурація яка включає операційну систему Windows macOS або Linux процесор з двома ядрами частотою не менше двох гігагерц щонайменше чотири гігабайти оперативної пам'яті для оптимальної роботи бажано вісім гігабайт мінімум два гігабайти вільного місця на диску для зберігання даних бази журналів та медіафайлів а також стабільне підключення до інтернету для доступу до API біржі. Додатково необхідно мати встановлене програмне забезпечення Node.js версії не нижче шістнадцятої MongoDB п'ятої версії або вище а також Git.

Для початку роботи слід клонувати репозиторій проєкту за допомогою GitLab [23]. Для цього потрібно виконати команду клонування репозиторію на сервер або локальний комп'ютер. Далі необхідно створити файл із середовищними змінними у кореневій директорії де буде зберігатися конфіденційна

інформація така як секретний ключ JWT адреса підключення до бази даних MongoDB і назва бази даних.

Після цього необхідно встановити усі залежності для обох частин проєкту. Для серверної частини потрібно перейти в папку server і виконати команду встановлення залежностей а для клієнтської перейти в client і виконати аналогічну команду. Далі можна запускати платформу. Серверна частина запускається з каталогу server і піднімає HTTP сервер на порту вказаному у файлі налаштувань за замовчуванням 3000 а також створює WebSocket сервер. Клієнтська частина запускається з каталогу client і зазвичай працює на порту 1573 з автоматичним підключенням до серверного API та WebSocket для отримання оновлень у режимі реального часу.

У разі потреби у запуску в продакшн режимі React додаток можна зібрати за допомогою команди build і передати його Express серверу для обслуговування як статичних файлів. Перед початком використання платформи користувач повинен зареєструвати обліковий запис або увійти до існуючого акаунта після чого створити портфоліо на сторінці налаштувань і обрати його при створенні ордеру або відкритті позиції.

Для стабільної роботи платформи рекомендується використовувати один із сучасних браузерів таких як Google Chrome Mozilla Firefox або Microsoft Edge версій не нижче ста. Після виконання усіх необхідних кроків платформа буде доступна за локальними адресами для клієнта сервера та WebSocket і готова до використання для симуляції торгових операцій з криптоактивами в умовах реального часу.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з системою

Після переходу користувача на головну сторінку застосунку система автоматично перевіряє, чи він авторизований. Якщо користувач не пройшов

авторизацію, система перенаправляє його на сторінку входу (Рис. 3.1).

 Logo

Please sign in

Email address

Password

Sign in

Don't have an account? [Sign Up](#)

Рис. 3.1 – Сторінка авторизації

На сторінці авторизації користувач може ввести свій логін та пароль для входу до особистого кабінету. Якщо ж у нього ще немає облікового запису, він може перейти на сторінку реєстрації та створити новий акаунт. (Рис. 3.2).

 Logo

Please sign up

Email address

Password

Sign up

Already have an account? [Sign in](#)

Рис. 3.2 – Сторінка реєстрації

Після авторизації користувач потрапляє на головну сторінку застосунку(Рис. 3.3), яка є стартовою точкою для роботи з платформою. Інтерфейс головної сторінки побудований таким чином, щоб забезпечити

зручний доступ до ключових розділів системи, а також швидкий огляд актуальної інформації про ринок криптоактивів.

У верхній частині інтерфейсу платформи розташовано навігаційне меню яке забезпечує швидкий доступ до основних сторінок застосунку за допомогою бібліотеки React Router [18]. Користувач може у будь-який момент повернутися на головну сторінку для перегляду дашборду перейти до торгового терміналу де здійснюється відкриття позицій та створення ордерів переглянути журнал з історією своїх угод та аналітикою або перейти до налаштувань де доступні опції зміни теми профілю паролю та портфелів. Також у меню розміщено кнопку виходу з акаунту яка завершує поточну сесію.

У центрі головної сторінки розташовано інтерактивні інформаційні віджети які в режимі реального часу відображають найактуальнішу ринкову інформацію. Один із ключових елементів це індекс страху та жадібності [20] який ілюструє емоційний настрій учасників ринку і може слугувати індикатором для прийняття торгових рішень. Також виводяться поточні курси основних криптовалют таких як Bitcoin Ethereum USDT у вигляді табличок або невеликих графіків а додатково показуються глобальні ринкові показники включно з загальною капіталізацією крипторинку та добовим обсягом торгів.

Усі ці дані динамічно оновлюються за допомогою підключення до зовнішніх API таких як CoinGecko CoinMarketCap або Bybit що забезпечує користувачу оперативний доступ до свіжої інформації без потреби оновлювати сторінку. Крім цього кожен віджет є інтерактивним і виконує функцію переходу на сторонні сервіси для перегляду детальнішої аналітики. Наприклад при натисканні на блок із індексом страху та жадібності відкривається офіційний сайт де можна побачити повну історію його змін а натискання на курс Bitcoin або Ethereum дозволяє відкрити сторінку з графіками в реальному часі на одній із бірж або в аналітичному сервісі.

Такий підхід дозволяє поєднати зручність загального огляду ринку безпосередньо в межах платформи із можливістю отримання розширеної інформації з авторитетних джерел. Уся функціональність головної сторінки

створена з метою надати користувачу повне уявлення про стан ринку ще до початку торгівлі а також дозволити швидко перейти до розділів торгівлі або перегляду власної торгової статистики.

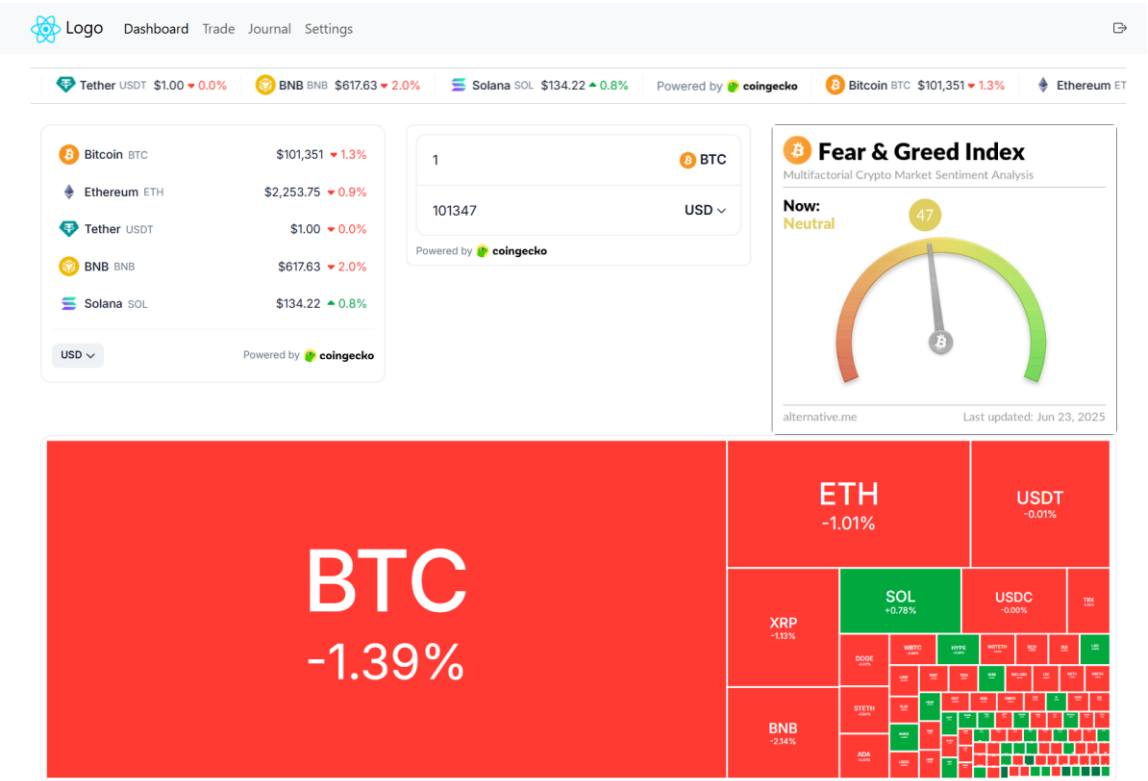


Рис. 3.3 – Головна сторінка застосунку

Користувач має можливість перейти на сторінку налаштувань яка представлена на (рис 3.4) і містить усі необхідні інструменти для персоналізації платформи та керування власними торговими портфелями. У цьому розділі доступна зміна кольорової теми інтерфейсу що дозволяє обрати між світлою та темною темою відповідно до індивідуальних уподобань або умов роботи. Крім того користувач може змінити свій пароль для підвищення безпеки облікового запису ввівши чинний пароль новий пароль та підтвердження нового пароля що мінімізує ризики несанкціонованого доступу. Окрему частину функціоналу сторінки становить менеджер портфоліо який дозволяє створювати нові торгові портфелі що дає змогу розділяти стратегії або групувати активи за тематиками. Користувач може редагувати назви вже створених портфелів щоб легше ідентифікувати їх у списку а також видаляти ті що більше не використовуються

щоб вони не враховувалися під час побудови загальної торгової статистики. Слід зазначити що баланс портфельів не підлягає ручному редагуванню і змінюється виключно в результаті торгових операцій. Такий підхід забезпечує збереження достовірності статистичних даних і дозволяє об'єктивно оцінювати ефективність обраних стратегій. Цей розділ відіграє важливу роль у забезпеченні зручного керування особистим обліковим записом дає можливість гнучко налаштовувати вигляд інтерфейсу та підтримує порядок у віртуальних портфелях що є необхідним для якісного аналізу результатів торгівлі.

Logo Dashboard Trade Journal Settings

Login as admin@admin.com

☐ Dark theme

Change Password

Current password

New password

Change Password

Portfolio list

Name: portfolio 1	Edit	
Balance:		
Initial balance: 1000		

Add new

Рис. 3.4 – Сторінка налаштувань

Користувач може перейти на сторінку "Торгівля" (Рис. 3.5) за допомогою навігаційного меню, розташованого у верхній частині інтерфейсу. Ця сторінка є центральним елементом платформи, де відбувається взаємодія з ринком у режимі демо-торгівлі.

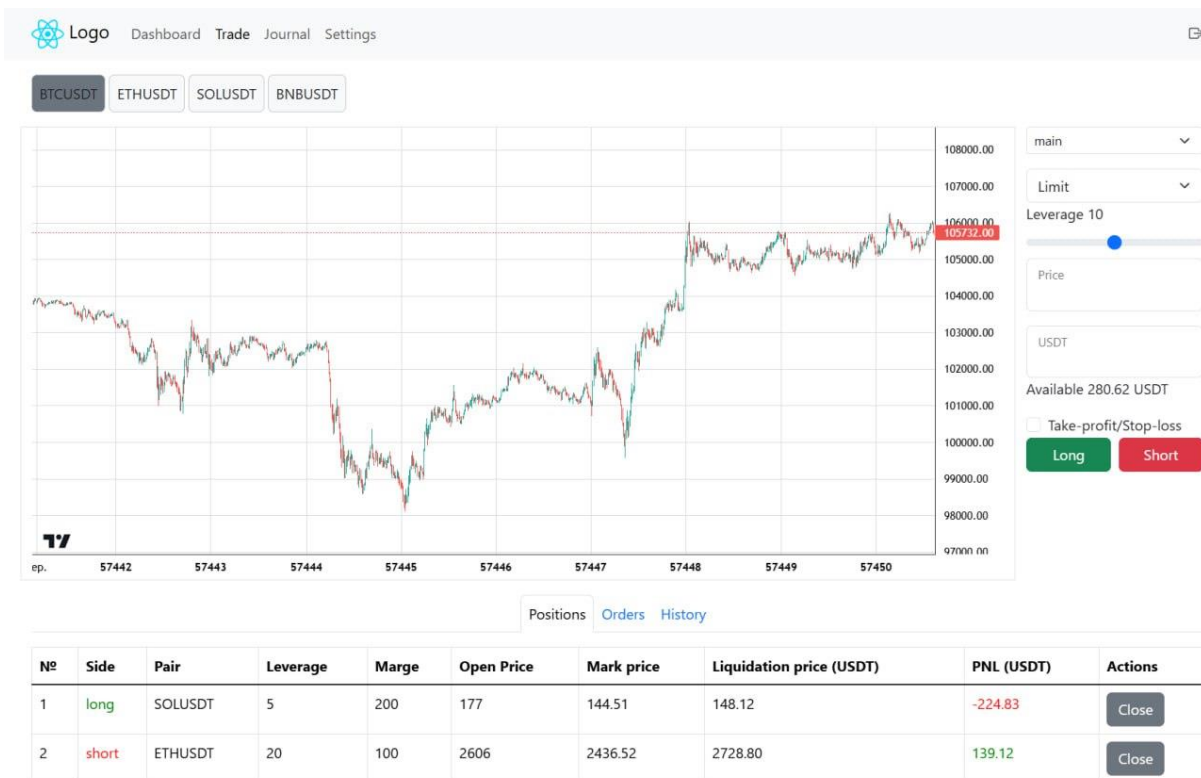


Рис. 3.5 – Сторінка торгівлі

Основним елементом сторінки торгівлі є інтерактивний графік ринку який представлено у верхній частині інтерфейсу як показано на (рис 3.6.) Цей графік реалізовано за допомогою TradingView Chart [10] і він відображає зміну ціни вибраної криптовалютної пари наприклад BTC до USD у режимі реального часу. Завдяки використанню WebSocket-з'єднання з сервером оновлення графіка здійснюється щосекунди що дає змогу користувачеві оперативно відслідковувати навіть найменші ринкові коливання та швидко реагувати на зміни ситуації. Така візуалізація дозволяє приймати обґрунтовані торгові рішення на основі актуальної динаміки цін.



Рис. 3.6 – Графік ринку

Однією з ключових функцій на сторінці торгівлі є можливість вибору торгового інструменту як показано на (рис 3.7.) Над графіком або у бічному меню розташований селектор який дозволяє користувачеві обрати криптовалютну пару з якою він хоче працювати наприклад BTC до USDT ETH до USDT SOL до USDT та інші. Після вибору нового торгового інструменту усі пов'язані дані включно з графіком ринку списком ордерів та позицій автоматично оновлюються що забезпечує зручність і швидкість у роботі з різними активами. Такий механізм дозволяє користувачу легко перемикатись між ринками і адаптувати торгову стратегію залежно від ситуації.

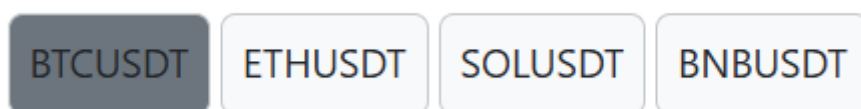


Рис. 3.7 – Вибір торгового інструменту

Торговий термінал як показано на (рис 3.8) розташовується праворуч від графіка і представляє собою панель для створення ордерів на купівлю або продаж криптоактивів. У цьому блоці користувач має змогу обрати один із доступних типів ордерів. Перший тип це ордер ринку який дозволяє миттєво купити або продати актив за поточною ринковою ціною. Другий тип це лімітний ордер за якого користувач самостійно задає бажану ціну за якою буде виконано купівлю або продаж. Для створення ордера користувач обирає тип угоди вводить

необхідну кількість активу та підтверджує операцію натисканням відповідної кнопки. Усі введені дані проходять валідацію і надсилаються на сервер де формується запит до бази та відбувається запис до журналу угод.

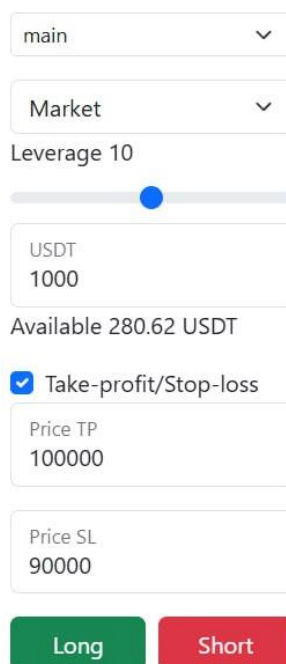


Рис. 3.8 – Торговий термінал

У нижній частині торгової панелі як показано на (рис 3.9) розміщується список активних ордерів і відкритих позицій користувача. У цьому блоці відображається основна інформація щодо кожної торгової операції зокрема назва криптоактиву що бере участь в угоді кількість обрана для купівлі або продажу ціна за якою була відкрито позицію поточна ринкова ціна а також прибуток або збиток який утворився в результаті зміни ціни. Крім того кожен запис містить статус що дозволяє швидко визначити чи є ордер активним чи вже був виконаний або скасований. Цей функціонал надає користувачеві зручний інструмент для відстеження ходу торгів у реальному часі.

Positions Orders History									
№	Side	Pair	Leverage	Marge	Open Price	Mark price	Liquidation price (USDT)	PNL (USDT)	Actions
1	long	SOLUSDT	5	200	177	144.64	148.12	-223.73	Close
2	short	ETHUSDT	20	100	2606	2438.4	2728.80	137.47	Close

Рис. 3.9 – Список активних позицій та ордерів

Цей розділ платформи максимально наближений до реальних торгових терміналів, що дозволяє користувачеві відпрацьовувати навички управління позиціями, прийняття рішень на основі динаміки ринку, а також формування торгових стратегій без фінансового ризику.

Користувач може перейти на сторінку "Журнал" (англ. Journal) (Рис. 3.10) через навігаційне меню для перегляду розширеної статистики своєї демо-торгівлі. Цей розділ платформи виконує функцію історичного аналізу та дозволяє користувачеві оцінити ефективність своєї торгової стратегії на основі завершених угод.

№	Side	Pair	Marge	Open Price	Close Price	PNL(USDT)
1	long	ETHUSD	100	2400	2418.14	6.8
2	short	BTCUSD	200	105000	105230.1	-3.07
3	long	BTCUSD	100	104999.9	104989.9	-0.07
4	long	BTCUSD	200	105176.9	104970	-2.75
5	long	BTCUSD	10	105119.3	105145.6	0.03
6	long	BTCUSD	10	105135.1	105116.2	-0.04
7	short	BTCUSD	100	105135	105149.5	-0.28

Рис. 3.10 – Сторінка "Журнал"

У верхній частині сторінки журналу як показано на (рис 3.11) розміщується таблиця що містить перелік усіх завершених угод користувача. Ця таблиця надає повну інформацію про кожну торгову операцію яка була закрита включаючи назву торгової пари наприклад BTC до USDТ напрямок позиції тобто чи була вона довгою або короткою дату коли позицію було відкрито та закрито ціну за якою здійснено вхід у ринок і ціну виходу з ринку розмір позиції в обраній валюті а також підсумковий фінансовий результат у вигляді прибутку або збитку. Такий формат дозволяє користувачеві швидко оцінити ефективність своїх рішень та зробити висновки щодо доцільності обраної стратегії.

№	Side	Pair	Marge	Open Price	Close Price	PNL(USDT)
1	long	BTCUSDT	100	102500	102547.8	0.47

Рис. 3.11 – Таблиця завершених угод

У верхній частині сторінки як зображено на (рис 3.12) відображається зведена статистика яка узагальнює результати торгівлі користувача. Тут подано кількість прибуткових та збиткових позицій що дозволяє оцінити співвідношення успішних і невдалих угод. Також зазначено кількість коротких позицій які відкривались із розрахунком на зниження ціни та довгих позицій що передбачали зростання ринку. Окремо вказується загальний фінансовий результат позначений як PnL який демонструє сумарний прибуток або збиток користувача. Крім того обчислюється середній прибуток або збиток на одну угоду що дає змогу оцінити ефективність торгівельної стратегії в цілому. Такий формат подачі дозволяє користувачу швидко проаналізувати власні результати та приймати більш обґрунтовані рішення під час наступних торгів.

Long positions: 5
Short positions: 2
Initial balance: 1000
Current balance: 280.62
Total PNL(USDT): -719.38

Рис. 3.12 – Загальна статистика

Сторінка "Журнал" дозволяє користувачеві аналізувати власну торгову поведінку, визначати сильні та слабкі сторони своєї стратегії, а також відслідковувати прогрес у навчанні демо-трейдингу. Вся інформація оновлюється автоматично на основі історії завершених позицій у базі даних.

3.3. Тестування платформи

За допомогою онлайн сервісу було проведено тестування вебдодатку. Для

тестування було використано GTMetrix [22] та Pagespeed [21]. GTMetrix – це інструмент для тестування продуктивності та швидкості завантаження вебсторінок. Він надає звіти та аналіз продуктивності вашого вебсайту на основі різних метрик, таких як час завантаження сторінки, розмір сторінки, кількість запитів до сервера і багато іншого.

Після завершення тестування було отримано звіт, який містить різноманітну інформацію про продуктивність сайту (рис. 3.13).

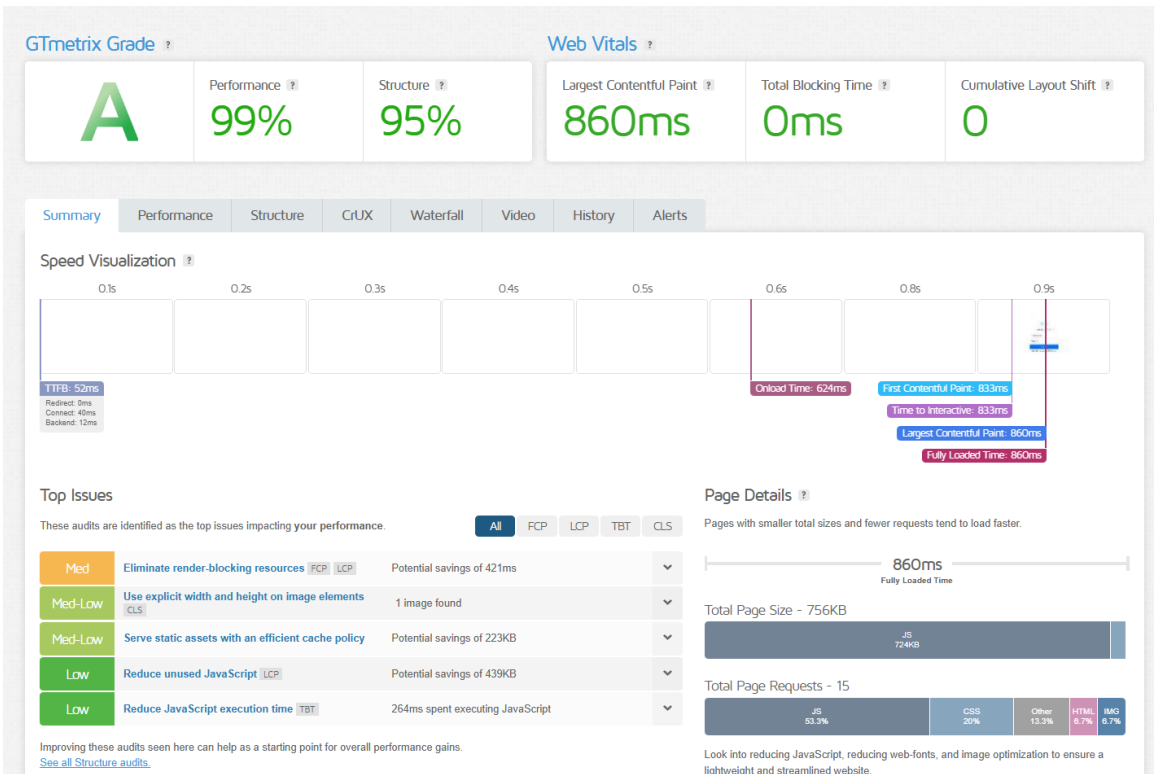


Рис. 3.13 – Тестування вебдодатку за допомогою GTMetrix

Звіти GTMetrix містять такі дані, як загальний час завантаження сторінки, час завантаження вмісту, кількість запитів до сервера, розмір сторінки, швидкість завантаження на різних типах з'єднань і багато іншого. Ця інформація допомагає вам зрозуміти, як ваш вебсайт працює з точки зору продуктивності та швидкості, і дозволяє вам вжити заходів для поліпшення цих показників (рис.3.14).

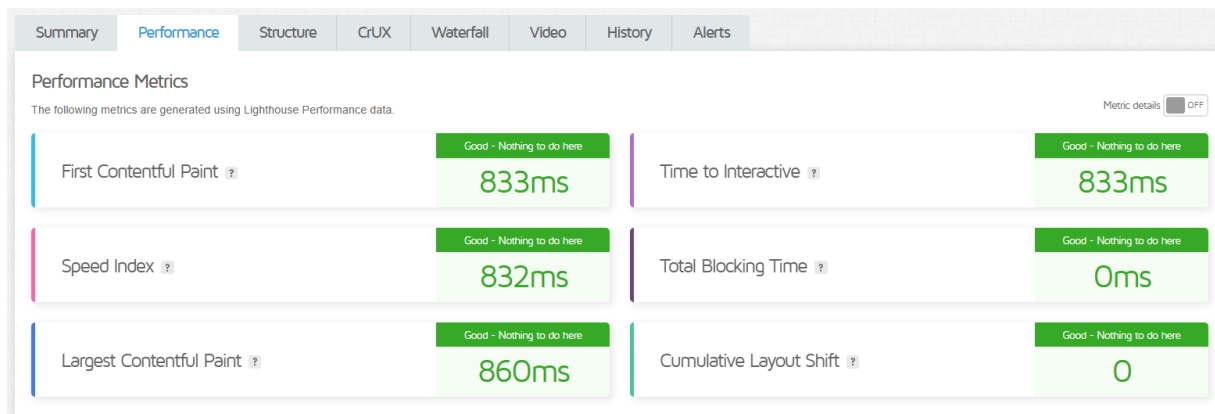


Рис. 3.14 – Тестування швидкості вебдодатку за допомогою GTMetrix [22]

GTMetrix також надає ряд інструментів та функцій для оптимізації продуктивності вебсайту, включаючи можливість зберігання та порівняння звітів, моніторинг продуктивності, сповіщення про проблеми продуктивності та багато іншого. Pagespeed (рис. 3.15) вебсайт, який надає інструменти та рекомендації щодо оптимізації швидкості завантаження сторінок. Цей вебсайт створений командою Google з метою допомогти веброзробникам покращити продуктивність своїх вебсайтів [26].

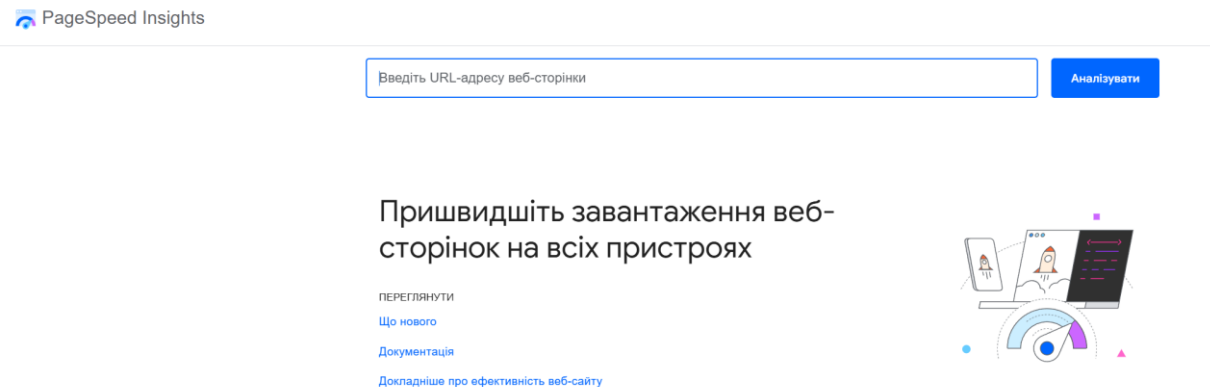


Рис. 3.15 – Веб сайт Pagespeed [21]

Аналіз швидкості сторінок. Вебсайт надає можливість ввести URL-адресу свого вебсайту і отримати докладний аналіз швидкості його завантаження. Інструмент оцінює різні аспекти, включаючи час завантаження, розмір сторінки, кількість запитів до сервера та інші параметри (рис. 3.15).

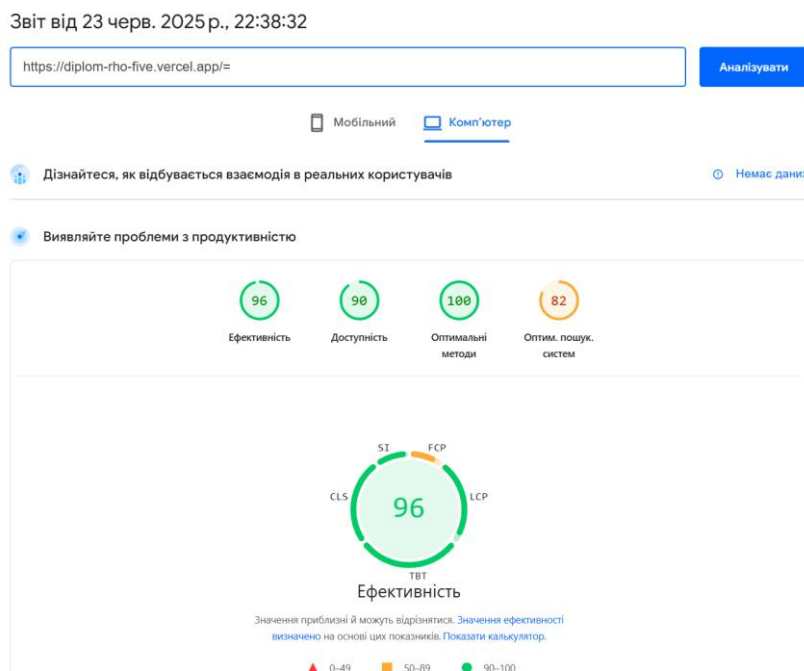


Рис. 3.15 – Тестування вебдодатку за допомогою Pagespeed [21]

Рекомендації щодо оптимізації. Вебсайт надає докладний список рекомендацій та практичні поради щодо того, як покращити швидкість завантаження сторінок. Ці рекомендації базуються на кращих практиках та рекомендаціях Google для оптимізації вебсайтів.

Висновки до третього розділу

У цьому розділі було детально розглянуто процес розгортання платформи, зокрема її архітектуру, а також описано функціональні частини системи та принципи їх взаємодії. Завдяки цьому було отримано цілісне уявлення про структуру застосунку та логіку обміну даними між клієнтською та серверною частинами.

Далі було проведено аналіз інтерфейсу користувача. Розглянуто основні елементи інтерфейсу та сценарії роботи в залежності від ролі користувача. Кожен тип користувача має доступ до специфічного набору функцій, що дозволяє ефективно використовувати можливості платформи відповідно до власних потреб і цілей.

Окрему увагу приділено реалізації модуля статистики. У межах платформи передбачено набір аналітичних інструментів, доступних кожному користувачу для самостійного аналізу власної торгової активності. Зокрема, реалізовано відображення ключових метрик, таких як кількість угод, фінансовий результат, співвідношення прибуткових і збиткових позицій, а також розподіл операцій за напрямком (довгі та короткі). Уся статистика є персоналізованою — користувач отримує аналітичні дані лише щодо власних дій у системі. Такий підхід забезпечує зручність контролю за результатами торгівлі та сприяє формуванню ефективних стратегій на основі зворотного зв'язку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено та реалізовано веб-платформу для демо-трейдингу криптоактивами, яка дозволяє користувачам без фінансових ризиків опановувати базові та просунуті навички торгівлі цифровими активами в реальному часі.

Під час розробки було проаналізовано основні вимоги до систем подібного типу, вивчено наявні рішення на ринку та визначено ключові функціональні компоненти, необхідні для ефективного навчального процесу. Особливу увагу приділено архітектурі системи — клієнт-серверному підходу з використанням сучасного JavaScript-стеку: React на фронтенді та Node.js із WebSocket-підключенням на бекенді. Для зберігання даних використано нереляційну базу MongoDB, а для отримання ринкових котирувань інтегровано API біржі ByBit.

Реалізований інтерфейс є адаптивним, інтуїтивно зрозумілим і орієнтований на досвід користувача. Передбачено авторизацію, систему портфелів, функціонал відкриття позицій та аналізу історії угод. Завдяки інтерактивним віджетам з реальними ринковими даними користувачі можуть практикувати торгівлю в умовах, максимально наближених до реальних. Особлива увага була приділена модулю статистики: кожен користувач має доступ до журналу торгів, зведеної інформації про прибутковість, кількість позицій, співвідношення довгих і коротких угод тощо.

Розроблена система може бути використана як навчальний інструмент у закладах освіти, а також як практична платформа для новачків у сфері криптовалют. Вона створює безпечне середовище для експериментів, допомагає формувати торгові стратегії без ризику втрати реальних коштів і сприяє популяризації фінансової грамотності у сфері цифрових активів.

У результаті проведеної роботи досягнуто поставлену мету, реалізовано всі заплановані функції та підтверджено ефективність запропонованого рішення. Платформа готова до подальшого розвитку зокрема, впровадження системи навчальних модулів, алгоритмічної торгівлі та інтеграції з іншими біржовими API.

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						56

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Binance Futures Testnet [Електронний ресурс] – Режим доступу до ресурсу: <https://testnet.binancefuture.com/en/futures/BTCUSDT>
2. TradingView Paper Trading [Електронний ресурс] – Режим доступу до ресурсу: <https://www.tradingview.com/chart/>
3. BitMEX Testnet [Електронний ресурс] – Режим доступу до ресурсу: <https://testnet.bitmex.com/>
4. Bybit Testnet [Електронний ресурс] – Режим доступу до ресурсу: <https://testnet.bybit.com/>
5. Phemex Testnet [Електронний ресурс] – Режим доступу до ресурсу: <https://testnet.phemex.com/>
6. Bybit API [Електронний ресурс] – Режим доступу до ресурсу: <https://bybit-exchange.github.io/docs/>
7. CoinGecko API [Електронний ресурс] – Режим доступу до ресурсу: <https://www.coingecko.com/en/api>
8. CoinMarketCap API [Електронний ресурс] – Режим доступу до ресурсу: <https://coinmarketcap.com/api/>
9. Binance API [Електронний ресурс] – Режим доступу до ресурсу: <https://binance-docs.github.io/apidocs/>
10. TradingView Charting Library [Електронний ресурс] – Режим доступу до ресурсу: <https://www.tradingview.com/charting-library-docs/>
11. React [Електронний ресурс] – Режим доступу до ресурсу: <https://reactjs.org/>
12. Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en>
13. Express.js [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>
14. WebSocket API [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>

15. MongoDB [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.mongodb.com/>
16. JWT – JSON Web Token [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/>
17. Bcrypt для Node.js [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/bcrypt>
18. React Router [Електронний ресурс] – Режим доступу до ресурсу:
<https://reactrouter.com/>
19. Axios – HTTP клієнт [Електронний ресурс] – Режим доступу до ресурсу:
<https://axios-http.com/>
20. Fear and Greed Index [Електронний ресурс] – Режим доступу до ресурсу:
<https://alternative.me/crypto/fear-and-greed-index/>
21. Google PageSpeed Insights [Електронний ресурс] – Режим доступу до ресурсу: <https://pagespeed.web.dev/>
22. GTmetrix [Електронний ресурс] – Режим доступу до ресурсу:
<https://gtmetrix.com/>
23. GitLab [Електронний ресурс] – Режим доступу до ресурсу:
<https://gitlab.com/>
24. RESTful API Tutorial [Електронний ресурс] – Режим доступу до ресурсу:
<https://restfulapi.net/>
25. REST API Design Rulebook [Електронний ресурс] – Режим доступу до ресурсу: <https://www.oreilly.com/library/view/rest-api-design/9781449317904/>
26. UML Tutorial by Visual Paradigm [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/>
27. UML Diagrams Overview – IBM Developer [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.ibm.com/articles/uml-diagrams-overview/>

28.Chen, Y., Lin, Y., & Xu, W. (2023). Simulation of Cryptocurrency Trading Strategies in Educational Platforms. Journal of Financial Technologies, 12(2), 45–59.
[Электронный ресурс] – Режим доступа до ресурсу:
<https://doi.org/10.1016/j.irfa.2023.102914>

29. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.
[Электронный ресурс] – Режим доступа до ресурсу:
<https://bitcoin.org/bitcoin.pdf>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						60
	.					

```

// bybit-api websocket
const bybitSocket = new WebSocketClient({});

const server = http.createServer(app);
const wss = new WebSocket.Server({ server });

let clients = []; // [{ ws, symbol }]
let subscriptions = {}; // { symbol: count }

wss.on("connection", (clientWs) => {
  console.log("Клієнт підключився");

  // Єдиний глобальний обробник для всіх оновлень
  bybitSocket.on("update", (data) => clientWs.send(JSON.stringify(data)));

  clientWs.on("message", async (msg) => {
    const data = JSON.parse(msg);

    if (data.type === "subscribe") {
      const symbol = data.symbol;

      // Додаємо нову підписку на символ
      clients.push({ ws: clientWs, symbol });

      if (!subscriptions[symbol]) {
        subscriptions[symbol] = 1;

        bybitSocket.subscribeV5([`kline.5.${symbol}`], "linear");
        console.log(`Підписка на ${symbol}`);
      } else {
        subscriptions[symbol]++;
      }
    }

    if (data.type === "unsubscribe") {
      const symbol = data.symbol;
      clients = clients.filter((c) => c.ws !== clientWs || c.symbol !== symbol);

      console.log(subscriptions);

      if (subscriptions[symbol]) {
        subscriptions[symbol]--;
        if (subscriptions[symbol] === 0) {
          delete subscriptions[symbol];
          await bybitSocket.unsubscribeV5([`kline.5.${symbol}`], "linear");

          console.log(`Відписка від ${symbol}`);
        }
      }
    }
  });
});

```

```

    }
  }
});

clientWs.on("close", () => {
  console.log("Клієнт відключився");

  clients = clients.filter((c) => c.ws !== clientWs);

  // При відключенні клієнта відписуємось від всіх його підписок
  Object.keys(subscriptions).forEach((symbol) => {
    if (!clients.some((c) => c.symbol === symbol)) {
      console.log(`Автоматична відписка від ${symbol}`);
      bybitSocket.unsubscribeV5([`kline.5.${symbol}`], "linear");
      subscriptions = {};
    }
  });
});
});
});
});

```

Додаток Б

```

import {
  createChart,
  ColorType,
  CandlestickSeries,
  IChartApi,
} from "lightweight-charts";
import { useEffect, useRef } from "react";
import { useParams } from "react-router-dom";
import { Candle } from "../types";
import api from "../../api";

function Chart({ currentCandle }: { currentCandle: Candle }) {
  const backgroundColor = "white";
  const textColor = "black";

  const { pair } = useParams();

  const chartContainerRef = useRef<IChartApi>(null);
  const seriesRef = useRef(null);

  useEffect(() => {
    seriesRef.current?.update({
      time: Math.round(currentCandle.timestamp / 300000) * 300000,
      open: +currentCandle.open,
      high: +currentCandle.high,
      low: +currentCandle.low,
      close: +currentCandle.close,
    });
  }, [currentCandle]);
}

```

```

useEffect(() => {
  api
    .get("/chart", {
      params: {
        symbol: pair,
      },
    })
    .then((response) => {
      // eslint-disable-next-line @typescript-eslint/ban-ts-comment
      // @ts-expect-error
      seriesRef.current.setData(response.data);
    });
}, [pair]);

useEffect(() => {
  api
    .get("/chart", {
      params: {
        symbol: pair,
      },
    })
    .then((response) => {
      if (chartContainerRef.current) {
        const handleResize = () => {
          // eslint-disable-next-line @typescript-eslint/ban-ts-comment
          // @ts-expect-error
          chartContainerRef.current.applyOptions({
            // eslint-disable-next-line @typescript-eslint/ban-ts-comment
            // @ts-expect-error
            width: chartContainerRef.current.clientWidth,
          });
        };
        // eslint-disable-next-line @typescript-eslint/ban-ts-comment
        // @ts-expect-error
        chartContainerRef.current = createChart(chartContainerRef.current, {
          layout: {
            background: { type: ColorType.Solid, color: backgroundColor },
            textColor,
          },
          // eslint-disable-next-line @typescript-eslint/ban-ts-comment
          // @ts-expect-error
          width: chartContainerRef.current.clientWidth,
          height: 500,
        });
        chartContainerRef.current.timeScale().fitContent();

        // eslint-disable-next-line @typescript-eslint/ban-ts-comment
        // @ts-expect-error
        seriesRef.current = chartContainerRef.current.addSeries(

```

```

        CandlestickSeries,
        {
            upColor: "#26a69a",
            downColor: "#ef5350",
            borderVisible: false,
            wickUpColor: "#26a69a",
            wickDownColor: "#ef5350",
        }
    );
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-expect-error
    seriesRef.current.setData(response.data);

    window.addEventListener("resize", handleResize);

    return () => {
        chartContainerRef.current?.remove();
    };
}
});
}, []);

return (
    <>
        <div className="col-10 border">
            <div
                ref={chartContainerRef as unknown as React.RefObject<HTMLDivElement>}
            />
        </div>
    </>
);
}

export default Chart;

```

Додаток В

```

import Terminal from "./Terminal";
import Chart from "./Chart";
import Sidebar from "./Sidebar";
import { useEffect, useRef, useState } from "react";
import { NavLink, useParams } from "react-router-dom";
import { Position, Order, Candle } from "./types";
import api from "../../api";

function Trade() {
    const [positions, setPositions] = useState<Position[]>([]);
    const [orders, setOrders] = useState<Order[]>([]);
    const [currentCandle, setCurrentCandle] = useState<Candle>({} as Candle);
    const [candlesData, setCandlesData] = useState<{ string: Candle }[]>([]);

```

```

const { pair } = useParams();

const ws = useRef<WebSocket>(null);
const prevSymbol = useRef<string>(null);
const positionsRef = useRef<Position[]>([]);
const chartDataRef = useRef<Record<string, Candle>>({});

const closePosition = (position: Position) => {
  api
    .post("/position/close", {
      position,
      closePrice:
        (candlesData as unknown as Record<string, { close: number }>)[
          position.pair
        ]?.close,
    })
    .then(() => {
      setPositions(positions.filter((p) => p._id !== position._id));
    });
};

const openPosition = (
  side: "long" | "short",
  leverage: number,
  price: number,
  marge: number,
  takeProfit: number | null,
  stopLoss: number | null,
  portfolioId: string
) => {
  api
    .post("/position/open", {
      side,
      pair,
      leverage,
      price,
      marge,
      takeProfit,
      stopLoss,
      portfolioId,
    })
    .then(fetchPositions);
};

const fetchPositions = () => {
  api.get("/positions").then((response) => {
    setPositions(response.data);
    positionsRef.current = response.data;
  });
};

```

```

const fetchOrders = () => {
  api.get("/orders").then((response) => {
    setOrders(response.data);
  });
};

const createLimitOrder = (
  side: "long" | "short",
  leverage: number,
  price: number,
  marge: number,
  takeProfit: number | null,
  stopLoss: number | null,
  portfolioId: string
) => {
  api
    .post("/order", {
      pair,
      side,
      leverage,
      price,
      marge,
      takeProfit,
      stopLoss,
      portfolioId,
    })
    .then(() => {
      console.log("orderCreated");
      fetchOrders();
    });
};

const deleteOrder = (id: string) => {
  api.post("/order/delete", { id }).then((response) => {
    if (response.data.deletedCount > 0) {
      setOrders(orders.filter((order) => order._id !== id));
    }
  });
};

// eslint-disable-next-line @typescript-eslint/ban-ts-comment
// @ts-expect-error
const prepareChartData = (candle) => {
  const chartDate = {} as Record<string, Candle>;
  let isChartDateExist = null;
  positionsRef.current.map((position) => {
    if (candle.topic.includes(position.pair)) {
      chartDate[position.pair] = candle.data[0];
      isChartDateExist = true;
    }
  });
};

```

```

    }
  });

  return isChartDateExist ? chartDate : isChartDateExist;
};

useEffect(() => {
  positions.forEach((position) => {
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-expect-error
    ws.current.send(
      JSON.stringify({ type: "subscribe", symbol: position.pair })
    );
  });
}, [positions]);

useEffect(() => {
  fetchPositions();
  fetchOrders();

  ws.current = new WebSocket(`ws://localhost:3000`);
  ws.current.onopen = () => {
    console.log("WebSocket connected");
    if (pair) {
      // eslint-disable-next-line @typescript-eslint/ban-ts-comment
      // @ts-expect-error
      ws.current.send(
        JSON.stringify({
          type: "subscribe",
          symbol: pair,
        })
      );

      prevSymbol.current = pair;
    }
  };
  ws.current.onmessage = (event) => {
    const candle = JSON.parse(event.data);

    setCandlesData((prev) => ({
      ...prev,
      [candle.topic.split(".")[2]]: candle.data[0],
    }));

    if (prepareChartDate(candle)) {
      chartDataRef.current = prepareChartDate(candle) as Record<
        string,
        Candle
      >;
    }
  }
}

```

```

    if (candle.topic.includes(prevSymbol.current)) {
        setCurrentCandle(JSON.parse(event.data).data[0]);
    }
};
return () => {
    // Відписка при демонтажі
    if (ws.current && ws.current.readyState === WebSocket.OPEN) {
        ws.current.send(
            JSON.stringify({ type: "unsubscribe", symbol: prevSymbol.current })
        );
        ws.current.close();
    }
};
}, []);

useEffect(() => {
    // Коли symbol змінюється (наприклад, ти переходиш з /BTCUSDT на /ETHUSDT)
    if (
        ws.current &&
        ws.current.readyState === WebSocket.OPEN &&
        prevSymbol.current &&
        prevSymbol.current !== pair
    ) {
        // Відписка від старого
        ws.current.send(
            JSON.stringify({ type: "unsubscribe", symbol: prevSymbol.current })
        );

        // Підписка на новий
        ws.current.send(
            JSON.stringify({
                type: "subscribe",
                symbol: pair,
            })
        );

        prevSymbol.current = pair as string;

        // setChartData(null);
    }
}, [pair]);

return (
    <div className="row">
        <div className="d-flex mb-3">
            <NavLink className="navbar-brand nav-link me-1" to="/trade/BTCUSDT">
                <div
                    className={
                        pair === "BTCUSDT"

```

```

        ? "card p-2 bg-secondary border border-secondary-subtle"
        : "card p-2 bg-light border border-secondary-subtle"
    }
    >
    BTCUSDT
  </div>
</NavLink>
<NavLink className="navbar-brand nav-link me-1" to="/trade/ETHUSDT">
  <div
    className={
      pair === "ETHUSDT"
        ? "card p-2 bg-secondary border border-secondary-subtle"
        : "card p-2 bg-light border border-secondary-subtle"
    }
  >
    ETHUSDT
  </div>
</NavLink>
<NavLink className="navbar-brand nav-link me-1" to="/trade/SOLUSDT">
  <div
    className={
      pair === "SOLUSDT"
        ? "card p-2 bg-secondary border border-secondary-subtle"
        : "card p-2 bg-light border border-secondary-subtle"
    }
  >
    SOLUSDT
  </div>
</NavLink>
<NavLink className="navbar-brand nav-link" to="/trade/BNBUSDT">
  <div
    className={
      pair === "BNBUSDT"
        ? "card p-2 bg-secondary border border-secondary-subtle"
        : "card p-2 bg-light border border-secondary-subtle"
    }
  >
    BNBUSDT
  </div>
</NavLink>
</div>

<Chart currentCandle={currentCandle} />

<Terminal
  openPosition={openPosition}
  onCreateLimitOrder={createLimitOrder}
  currentCandle={currentCandle}
/>

```

```

    <Sidebar
      positions={positions}
      closePosition={closePosition}
      candlesData={candlesData}
      orders={orders}
      deleteOrder={deleteOrder}
    />
  </div>
);
}

```

```
export default Trade;
```

Додаток Г

```

const { MongoClient, ObjectId } = require("mongodb");

const connectionUrl = process.env.connectionURI
const dbName = "crypto_db";

let db;

const init = () => {
  MongoClient.connect(connectionUrl, { useNewUrlParser: true }).then(
    (client) => {
      db = client.db(dbName);
    }
  );
};

const getPositions = (userId) => {
  const collection = db.collection("positions");
  return collection.find({ userId }).toArray();
};

const openPosition = (position, userId) => {
  const collection = db.collection("positions");
  return collection.insertOne({ ...position, userId });
};

const closePosition = async ({ position, closePrice }, userId) => {
  const collectionTrades = db.collection("trades");
  const collection = db.collection("positions");

  const positionSize = (position.leverage * position.marge) / position.price;
  const pnl =
    position.side === "long"
      ? (Number(closePrice) - position.price) * positionSize
      : (position.price - Number(closePrice)) * positionSize;

  await collectionTrades.insertOne({

```

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк.
						70

```

    userId,
    side: position.side,
    pair: position.pair,
    pnl: Number(pnl.toFixed(2)),
    fee: 0,
    openPrice: position.price,
    closePrice,
    marge: position.marge,
    leverage: position.leverage,
    portfolioId: position.portfolioId
  });

  const portfolio = await getPortfolioById(position.portfolioId);

  const newBalance = Number(portfolio.balance) + position.marge +
    Number(pnl.toFixed(2));

  console.log(newBalance);

  await updatePortfolioBalance(position.portfolioId, newBalance);

  return await collection.deleteOne({
    _id: ObjectId.createFromHexString(position._id),
  });
};

const createLimitOrder = (order, userId) => {
  const collection = db.collection("orders");
  return collection.insertOne({ ...order, userId });
};

const getLimitOrders = (userId) => {
  const collection = db.collection("orders");
  return collection.find({ userId }).toArray();
};

const deleteOrder = ({ id }) => {
  const collection = db.collection("orders");
  return collection.deleteOne({ _id: ObjectId.createFromHexString(id) });
};

const getTrades = (userId, portfolioId) => {
  const collection = db.collection("trades");
  const filter = {
    userId,
    ...(portfolioId != null && { portfolioId })
  };
  return collection.find(filter).toArray();
};

```

```

const getUser = (email) => {
  const collection = db.collection("users");
  return collection.findOne({ email });
};

const saveNewUser = (user) => {
  const collection = db.collection("users");
  return collection.insertOne(user);
};

const changeUserPassword = (email, password) => {
  const collection = db.collection("users");
  return collection.updateOne(
    { email },
    { $set: { password } }
  );
};

const createPortfolio = (portfolio, userId) => {
  const collection = db.collection("portfolios");
  return collection.insertOne({ ...portfolio, userId });
};

const getPortfolioList = (userId) => {
  const collection = db.collection("portfolios");
  return collection.find({ userId }).toArray();
};

const deletePortfolio = (portfolioId) => {
  const collection = db.collection("portfolios");
  return collection.deleteOne({ _id: ObjectId.createFromHexString(portfolioId) });
};

const updatePortfolio = (portfolioId, name) => {
  const collection = db.collection("portfolios");
  return collection.updateOne(
    { _id: ObjectId.createFromHexString(portfolioId) },
    { $set: { name } }
  );
};

const updatePortfolioBalance = (portfolioId, balance) => {
  const collection = db.collection("portfolios");
  return collection.updateOne(
    { _id: ObjectId.createFromHexString(portfolioId) },
    { $set: { balance } }
  );
};

```

```

const getPortfolioById = (portfolioId) => {
  const collection = db.collection("portfolios");
  return collection.findOne({ _id: ObjectId.createFromHexString(portfolioId) });
};

module.exports = {
  init,
  getPositions,
  openPosition,
  closePosition,
  createLimitOrder,
  getLimitOrders,
  deleteOrder,
  getTrades,
  getUser,
  saveNewUser,
  changeUserPassword,
  createPortfolio,
  getPortfolioList,
  deletePortfolio,
  updatePortfolio,
  updatePortfolioBalance,
  getPortfolioById
};

```