Вищий навчальний заклад

«Університет економіки та права «КРОК»
Фаховий коледж

Циклова комісія з інформаційних технологій

Кваліфікаційна робота фахового молодшого бакалавра

на тему Створення вебзастосунку на spring boot для торгівлі криптовалютою ”Kupyrtin”

Виконав _____
(Підпис)

___Радченко Дмитро Олександрович___
(прізвище, ім’я, по батькові)

_____

Науковий керівник
_____Добришин Юрій Євгенович_____
(прізвище, ім’я, по батькові)

_____

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: “До захисту в екзаменаційній комісії ”)

Голова циклової комісії _____ _____ _____
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2025 року

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

Фаховий коледж

Циклова комісія з інформаційних технологій

Спеціальність 121 інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Голова циклової комісії _____ Леонід УВАРОВ

(підпис)

«____» _____ 2025 року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач освіти Радченко Дмитро Олександрович

Тема роботи Створення вебзастосунку на spring boot для торгівлі криптовалютою ”Kupyrtin” «___» _____ 202__ р. № _____

2. Термін здачі закінченої роботи «30» травня 2025 року

3. Вихідні дані до роботи:

1) функціональність – керування торговими ботами для торгівлю криптовалютою, отримання аналітики акаунту користувача.
2) цільова аудиторія – люди, що хочуть заробляти та не знаються на аналізі або погано розуміють потреби крипто-ринку;
3) технічні вимоги – веб-браузер, як платформа для бота, мова програмування, використання фреймворку, база даних та використання API-сервісів;
4) Бот інтегрується з біржою Binance за допомогою API-ключів.
5) Існуючі торгові боти для криптовалютою, кращі практики у сфері створення подібних веб застосунків.

4. Зміст пояснювальної записки

1) Розділ 1 Теоретична частина. Аналіз існуючих рішень та кращих практик у сфері створення веб-застосунків для торгівлею криптовалютою, обґрунтування вибору використаних технологій в даному веб-застосунку.

2) Розділ 2 Проектування та розробка. Створення простого та інтуїтивно зрозумілого інтерфейсу для користувачів. Розробка архітектури застосунку за яким буде працювати проект. Розробка бази даних для кращого доступу до даних та їх ефективнішому використанні.
3) Розділ 3 Експериментальна частина. Перевірка бота на різних торгових парах. Виявлення та виправлення помилок, що були допущені під час створення веб-застосунку. Чи дійсно бот відіграє свою роль у торгівлі криптовалютою, інструкція користувачам, щоб краще зрозуміти як користуватись даним застосунком.

5. Перелік графічного матеріалу:

Скріншоти інтерфейсу продукту

Дата видачі завдання «12» лютого 2025 року

Науковий керівник _____ ___Добришин Ю. Є.

(підпис) (прізвище, ім'я, по батькові)

Завдання прийняла до виконання _____ ___Радченко Д. О.___

(підпис) (прізвище, ім'я, по батькові)

Реферат

Пояснювальна записка:

Об'єкт дослідження – об'єктом дослідження є веб-застосунки, що дозволяють автоматизовану торгівлю криптовалютою, зокрема їх архітектура, функціональні можливості, алгоритмічна логіка та засоби безпечної інтеграції з API криптовалютних біржою.

Мета роботи – створити систему, яка дасть можливість користувачам створювати профілі, створювати та керувати ботами.

У кваліфікаційній роботі проведено аналіз існуючих сучасних рішень у сфері автоматизованої торгівлі криптовалютою, визначено кращі практики та обрано засоби та інструменти які допоможуть реалізувати даний функціонал для веб-застосунку.

У роботі детально описано структуру бази даних, алгоритми взаємодії з біржою, логіку обробки запитів користувачів, алгоритм аналізу даних для відкривання торгових позицій та їх закривання.

Результати розробки. Можуть бути використанні у трейдингових стартапах, особистих інструментах для трейдерів або як навчальний приклад для впровадження систем алгоритмічної торгівлі.

Ключові слова: Spring Framework, Java, Thyamleaf, API, PostgreSQL, трейдинг, криптовалюта, база даних, алгоритми торгівлі, ордери, конфіденційність, інтеграція, автоматизація.

Abstract

Explanatory note.

The object of research is web applications that enable automated cryptocurrency trading, including their architecture, functionality, algorithmic logic, and means of secure integration with the API of cryptocurrency exchanges.

The aim of the work is to create a system that will allow users to create profiles, create and manage bots.

The qualification work analyzes existing modern solutions in the field of automated cryptocurrency trading, identifies best practices, and selects tools and instruments that will help implement this functionality for a web application.

The paper describes in detail the structure of the database, algorithms for interacting with the exchange, the logic for processing user requests, and the algorithm for analyzing data for opening and closing trading positions.

Development results. The application can be used in trading startups, personal tools for traders, or as a training example for implementing algorithmic trading systems.

Keywords: Spring Framework, Java, Thymeleaf, API, PostgreSQL, trading, cryptocurrency, database, trading algorithms, orders, confidentiality, integration, automation.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Код – програмний скрипт який виконує певну дію в застосунку.

Тестування – перевірка функціональності та коректної роботи веб-застосунку.

Інтеграція – об’єднання функціоналу з іншими компонентами в застосунку або з зовнішніми інструментами.

Дані користувача – це дані про зареєстрованого користувача який хочу користуватись певними функціями та інструментами.

API – це ключі, видані тим ресурсом до якого треба звертатись через програмний код для отримання певної інформації або надсилання запитів які дозволяють змінювати інформацію.

Реєстрація – це процес створення користувачем облікових даних, що дають програмі певну інформацію про користувача.

Шифрування – процес закодовування та розкодовування певної інформації.

Конфіденційність – це захист, персональної інформації користувача, для збереження санкціонованого доступу до даних.

HTML – це мова гіпертекстової розмітки для веб-сторінок або інших компонентів інтерфейсу які її використовують.

Spring Framework – це, набір компонентів, що спрощує розробку для веб-застосунків.

Java – мова об’єктно орієнтована мова програмування, використовується для написання великих та гнучких застосунків.

Thyamleaf – це шаблонізатор, що використовується для відображення певної інформацій.

PostgreSQL – це реляційна база даних.

SQL – це мова, що дозволяє керувати даними в базі за допомогою скриптів.

Веб-браузер – це програмне забезпечення, що дозволяє користувачам переглядати веб-сторінки та взаємодіяти з ними в інтернеті.

Веб-застосунок – це розподілений застосунок, в якому браузер виступає в якості клієнта, а веб-сервер бере на себе іншу частину роботи.

Веб-сервер – це сервер що приймає HTTP-запити від клієнтів та обробляє їх.

HTTP – це протокол передавання даних між сервером та клієнтом.

Клієнт – це програмний або апаратний компонент обчислювальної системи що приймає та надсилає запити.

ВСТУП

Актуальність теми. Автоматизована торгівля криптовалютою сьогодні стає стратегічним інструментом для трейдерів та інвесторів, що дозволяє уникати нудної рутини, та бути більш ефективним у сфері торгівлі, тому що з’являється більше часу для страгічного мислення. Процес автоматизації дозволяє ефективніше реагувати на волатильність ринку.

Веб-застосунок із вбудованими торговими алгоритмами здатен аналізувати аналітичні дані та вчасно відкривати або закривати позиції. Реалізація такого рішення на базі перевірених фреймворків і технологій сприятиме підвищенню ефективності торгових операцій, зниженню ризиків і розширенню інструментарію сучасного трейдера. Отже такий веб-застосунок здатен не тільки автоматизувати рутину, а й вести торгівлю цілодобово.

Завдання роботи. Завданням цієї кваліфікаційної роботи:

- Проведення аналізу існуючих веб-застосунків для алгоритмічної торгівлі криптовалютою, виокремити їхні переваги та недоліки.
- Спроектувати структуру бази даних у PostgreSQL для збереження облікових даних користувачів, торгових пар, налаштувань ботів, та журналу ордерів.
- Розробити механізм реєстрації та автентифікації користувачів із підтримкою API ключів біржі.
- Реалізувати функціонал керування торговими ботами, за обраними торговими парами та інтервалами.
- Забезпечити інтерфейс відображення аналітичних даних, (баланси, історія угод, графіки) за допомогою шаблонів Thymeleaf.
- Провести тестування веб-застосунку на працездатність під навантаження та виправити виявленні помилки.

Об’єкт дослідження.Об’єктом дослідження є веб-застосунки, що дозволяють автоматизовану торгівлю криптовалютою, зокрема їх архітектура,

функціональні можливості, алгоритмічна логіка та засоби безпечної інтеграції з API криптовалютних біржою. У роботі вивчаються програмні рішення, що забезпечують повноцінну взаємодію користувача з торговими інструментами через веб-інтерфейс, а також методи забезпечення конфіденційності даних.

Предмет дослідження. Предметом дослідження виступають архітектурні рішення, інструменти та технології (Spring Framework, Java, Thymeleaf, PostgreSQL), алгоритми обробки ринкових даних і виконання торгових операцій у контексті створення веб-застосунку.

Методи дослідження. Під час виконання роботи використовувались такі методи:

- Аналіз і порівняльна оцінка існуючих торгових веб-застосунків
- Проектування бази даних.
- Розробка бекенду на Spring Framework і реалізація інтерфейсу за допомогою шаблонізатора.
- Розробка алгоритмів, що аналізують аналітичні дані для отримання торгових сигналів.
- Тестування для забезпечення роботи системи.

Практичне значення одержаних результатів. Результати діяльності як базовий матеріал у трейдингових стартапах, особистих інструментах для трейдерів або як навчальний приклад для впровадження систем алгоритмічної торгівлі та створення подібного веб-застосунку.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

1.1 Аналіз існуючих рішень

У сучасному світі, де технологій набувають популярності з’являється можливість автоматизувати ті процеси, які раніше технологій не дозволяли. Для кожної галузі з’являються свої застосунки які вирішую ті чи інші проблеми, або автоматизують процеси, які раніше були нудною рутиною, так і в випадку з торгівлею активами.

На ринку вже існує досить велика кількість рішень, які дозволяють ефективно торгувати та налаштовувати торгових ботів, мають інтеграцію з різними біржами та іншими ресурсами, що забезпечують доступ до ринкових даних для їх аналізу. Я наведу кілька прикладів та опишу коротко цих ботів, для розуміння загальної картини.

Одним з найпопулярніших є 3Commas – це платформа, що дозволяє користувачам створювати та керувати власними ботами, налаштування ботів робиться за допомогою шаблонів або з нуля за власними потребами. Ключовою перевагою цього веб-застосунку є інтеграція з популярними біржами та підтримка різних стратегій. Попри, те, що цей сервіс дуже хорошим інструментом в руках знавця, для новачків він досить складний та тяжко зрозумілий.

Інший не менш популярний є Cryptohopper – хмарний торговий бот, він дозволяє автоматизувати торгівлю без необхідності постійного онлайн підключення. Підтримує використання сигналів від сторонніх аналітиків та надає можливість користувачам створювати власні алгоритми за допомогою вбудованого редактора. В числі недоліків – обмежена кастомізація в безкоштовному плані, яка не дає змоги протестувати придумані ідеї.

Також для людей які розуміються в програмуванні та торгівлі існує Gekko – з відкритим кодом, написаний на JavaScript. Він має можливість повністю змінювати алгоритми для аналізу ринку, це потребує досить хорошого рівня знань,

тому для новачків такий інструмент підходить погано. Також він не підтримує торгівлю в реальному часі на деяких біржах та не оновлюється з 2019 року.

Висновок

Проаналізувавши деякі популярні та не дуже рішення можна дійти до висновку, що вони мають як переваги, так і недоліки. Така тенденція створює нішу яка відкрита для нових рішень та форматів, які можуть бути як простими, що використовуються новачками, так і більш складними, які дозволять гнучко налаштовуватися та керуватися.

1.2 Практики розробки

У будь-якого програмного застосунку є архітектура. Розумна архітектура дозволяє розробляти більш складні застосунки, які витримують більші обсяги даних та не ламаються під великим тиском користувачів. А для побудови правильної архітектури потрібно детально спланувати, яку роботу, буде виконувати кожен компонент програми та які функції він буде підтримувати.

Отож в цьому розділі ми поговоримо проте як будувати архітектуру такого застосунку. Він охоплює як архітектурні принципи бекенду, так і проектування інтерфейсу, безпеку та моніторинг, що в сукупності сформують фундамент для надійної й масштабованої системи. Прошу сильно мене не критикувати оскільки це мій перший досвід у створенні такого веб-застосунку для торгівлею криптовалютою.

Архітектурна побудова.

Якщо нам потрібно скласти програму яка буде формувати собою велику монолітну систему. То чи буде вона працювати без зайвих збоїв? Чи буде вона легко-масштабованою? Чи буде вона гнучкою?

Відповідь на всі передні питання – ні, окрім першого, працювати буде, але ні один розробник не захоче підтримувати такий код.

Тож щоб такого не було варто розподілити архітектуру на елементи що будуть інтегруватись один з одним та дозволять ефективніше розділювати

відповідальність між компонентами програми. Нам потрібно розділити відповідальність за допомогою сервісів(для простішого аналізу деяких даних, та доступу до бази даних), контролерів(для роботи з шаблонами), менеджерів(для керування клієнтами та сервісами), клієнтів(для отримання даних з API), та інших утилітних класів(для забезпечення допоміжних компонентів веб-застосунку).

Алгоритмічна модель торгового ядра

Звісно нам потрібен функціонал, що забезпечить аналіз та прийняття рішень для створення ордерів. Отже тут нам потрібні такі функції –

- Збір ринкових даних – це відбувається за рахунок регулярного опитування біржі для отримування даних які потрібні для аналізу.
- Аналіз ринкових даних – за використання бібліотеки **Ta4j.**
- Прийняття рішень на основі попереднього аналізу
- Формування і надсилання ордерів на основі сигналів

Принципи проектування користувацького інтерфейсу.

1. **Патерн MVC (Model-View-Controller) –** це доволі зручний метод оскільки не потребує сильних знань в області фронтенді для відображення веб-сторінок
2. **Простий та зрозумілий інтерфейс –** що дозволить користувачам інтуїтивно розуміти куди потрібно тицяти кнопки.

Механізми забезпечення безпеки

Для конфіденційності та безпеки користувачів потрібно тримати всі важливі дані в зашифрованому вигляді. AES для шифрування API-ключів, шифрування паролів. Захист від атак за допомогою CSRF-токенів та SQL injection.

Висновок

Запропонований підхід охоплює всі ключові аспекти створення сучасного та надійного, масштабованого веб-застосунку для торгівлі криптовалютою. Гармонійне поєднання архітектури, алгоритмів та різних компонентів програми що правильно ділять між собою відповідальність добре підходять для масштабування.

1.3 Вибір технологій для створення веб-застосунку

Для створення веб-застосунку для торгівлі криптовалютою були обрані такі технології.

- **Java** – об’єктно орієнтована мова програмування, яка спрощує написання великих програм, в десятки разів. Підхід об’єктно орієнтованої мови програмування дозволяє просто та гнучко розширювати систему.
- **Spring Framework** функціонує як основна платформа для розробки веб-застосунків на мові програмування Java, що забезпечує високу продуктивність, масштабованість і модульність. Однією з головних переваг є можливість чіткого розподілу відповідальностей і проста реалізація інверсії керування, що дозволяють будувати чисту та зручну для підтримки архітектуру. Крім того Spring пропонує широку екосистему модулів, які значно пришвидшують розробку. Серед них є кілька ключових.
- **Spring web** – відповідає за створення Restful API, обробку HTTP-запитів, маршрутизацію та інтеграцію з шаблонізаторами, такими як Thymeleaf. Це дозволяє зручно створювати контролери, сервіси, та уявлення згідно з патерном MVC.
- **Spring Data JPA** – надає абстракцію над Java Persistence API (JPA) і спрощує роботу з базами даних. Завдяки цьому модулю CRUD-операції виконуються з мінімальними зусиллями через автоматичну генерацію запитів, підтримку репозиторіїв, складних фільтрацій та пагінації. Ідеально підходить для PostgreSQL, забезпечуючи гнучкість і потужну типізовану взаємодію з даними.
- **Spring Security** – модуль, який відповідає за автентифікацію та авторизацію користувача. Підтримує як класичну модель(логін\пароль), так і сучасні рішення(jwt, oAuth2). Дозволяє швидко налаштувати захист для API, ролі користувачів, та шифрування конфіденційної інформації. Включає такі вбудовані засоби від атак типу CSRF, XSS тощо.

- **PostgreSQL** є потужною, стабільною та масштабованою системою управління реляційними базами даних, яка ідеально підходить для розробки сучасних веб-застосунків. Її головними перевагами є підтримка складних типів даних(JSON, масиви, часові інтервали тощо), розширена система індексації, висока надійність завдяки підтримці ACID-транзакцій, активна спільнота розробників з відкритим кодом, а головне суворе дотримання SQL стандартів.
- **Thymeleaf** – це сучасний Java-шаблонізатор, що дозволяє створювати динамічні веб-сторінки.
- Lombok – це Java-бібліотека, яка дозволяє значно зменшити кількість рутинного коду за рахунок генерації конструкторів, геттерів, сеттерів, та інших схожих процесів.
- TA4J – це спеціалізована бібліотека для технічного аналізу фінансових даних. Вона містить набір готових індикаторів.
- JSON – це залежність, що дозволяє використовувати Jackson або Gson, які дозволяють швидко сереліазовувати та десеералізовувати дані.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Опис розробленої програми

В цьому підрозділі ми поговоримо технічну складову веб-застосунку для торгівлі криптовалютою з висоти пташиного польоту. Розберемо детальніше такі пункти.

Короткий опис основного функціоналу

Основний функціонал веб-застосунку передбачає повноцінну взаємодію з API криптовалютної біржі, що дає змогу користувачеві здійснювати торгові операції в режимі реального часу. Застосунок забезпечує авторизацію за допомогою імені користувача та його пароля, отримання та оновлення актуальних ринкових даних, перегляд балансу акаунта, створення та моніторинг ботів, перегляд історії операції застосунку, перегляд загальної інформації про акаунт користувача, та перегляду балансів активів та їх стану. Інтерфейс реалізований таким чином, щоб максимально спростити навігацію користувача по ресурсу. Вся інформація оновлюється динамічно, завдяки оновленню даних через API. Застосунок орієнтований на зручність, та безпечну взаємодію з біржою.

Авторизація та аутентифікація користувача в системі.

Процес аутентифікації та авторизації є критично важливою складовою, безпечного функціонування системи, тому що самі він гарантує конфіденційність акаунту користувача.

Коли користувач вперше реєструється в системі, то він заповнює всі необхідні поля, зокрема – ім’я користувача, пошта, пароль а також API-ключі криптовалютної біржі. Всі необхідні дані шифруються та зберігаються до бази даних PostgreSQL.

Пароль зберігається в зашифрованому вигляді, за допомогою хешування BCrypt, що унеможливлює зворотне його отримання навіть у випадку доступу до бази даних.

Після реєстрації користувач може авторизуватись. Цей процес перевіряє відповідність логіну та паролю, а також генерує токен, який буде використовуватись при кожному подальшому запиті.

Шифрування API-ключів також наявне. При зберіганні API-ключів вони шифруються за допомогою технологій шифрування AES, та розшифровуються назад коли їх потрібно використовувати для створення запитів до біржі.

Таким чино забезпечується конфіденційність акаунту користувача та його API-ключів для зменшення ймовірних витоків даних.

Створення ботів та керування ними.

Однією з ключових функцій веб-застосунку є можливість створення та керуванням торговими ботами, що взаємодіють з криптовалютою біржою в режимі реального часу та дозволяють оперативно відкривати та закривати позиції. Ця функціональність реалізована для новачків, які ще не мають особливого досвіду в цьому напрямку та їм потрібен мінімальний інтерфейс, щоб розібратись.

Процес створення бота починається з зрозумілої форми, де користувач вказує ключові, параметри: вибір торгової пари, вибір інтервалу та виділення коштів для торгівлі торговим ботом. Кожен з параметрів зберігається в базі даних, та асоціюється з конкретним користувачем.

Після створення бот переходить у активний стан та починає моніторити ринок, використовуючи Rest API для отримання даних з біржі. Це дозволяє оперативно реагувати на коливання ціни та приймати рішення для продажів та покупок.

Окрім того веб-застосунок дозволяє в режимі реального часу керувати ботом, тобто можливо зупиняти та відновлювати роботу бота. Це дозволяє більш гнучко використовувати цю платформу як інструмент для торгівлі.

Загалом керування ботом забезпечує гнучку та прозору автоматизацію торгівлі. Значно знижуючи проблему в постійному моніторингу про яку я говорив в першому розділі.

Взаємодія з API

Найбільш важливим функціоналом систем є інтеграція з API криптовалютної біржі, яка дозволяє здійснювати повноцінну взаємодію, з фінансовим ринком у режимі реального часу. Саме через API відбувається основний обмін інформацією веб-застосунку з біржою для отримання інформації, передача торгових наказів, оновлення балансу користувача, та контроль активів переданих боту.

З технічної точки зору, інтеграція API реалізована через REST-запити для стандартних операцій.

Після того, як користувач проходить авторизацію в веб-застосунку та вказує свої API-ключі криптовалютної біржі, система проводить шифрування даних та зберігає їх до бази даних. Усі подальші дії – наприклад створення ордеру на купівлю чи продаж, отримання аналітичних даних чи котирувань відбувається через заздалегідь прописані ендпоінти API, відповідно до технічної документації біржі.

Особливу увагу приділено обробці помилок та стабільності підключень. У випадку втрати з’єднання чи отримання помилки від API система намагається автоматично повторити запит а у випадку критичних збоїв – призупиняє роботу веб-застосунку.

У результаті взаємодії з API криптовалютної біржі є не лише функціональною необхідністю, а й одним з головних факторів, що забезпечують глибоку інтеграцію користувача в сучасний ринок цифрової торгівлі.

Інтерфейс користувача

Створення інтерфейсу користувача вимагає дотримання кількох важливих пунктів, які будуть мати вагоме значення для користування системою. Важливо розуміти які саме процеси будуть здійснюватися на стороні бекенду, щоб розуміти як саме їх можна описувати. Звісно в продакшин проектах спочатку вказує що він хоче бачити в веб-застосунку візуально, але лише потім він описує які в нього є

вимоги до функціоналу, та можливо, рідко вказує технології на якими він хоче бачити розроблений продукт.

Інтерфейс користувача – це, по суті обличчя системи, через яке відбувається взаємодія користувача з внутрішньою логікою. У контексті веб-застосунку для торгівлі криптовалютою особливо актуально є ***інтуїтивність***, ***надійність*** та ***простота*** у взаємодії. Адже користувач працює з реальними коштами і не повинен бути збитий з пантелику інтерфейсом.

1. Інтуїтивність

Це серце нашого дизайну. Кожна кнопка, кожне меню побудовані за принципом одного кліку. Коли користувач дивиться на контекстну панель він вже інтуїтивно розуміє, що я отримаю натиснувши на ту чи іншу кнопку.

Значну роль в таку поданні інформації відграє навбар він як швидка навігаційна панель, через яку можна потрапити в будь яку основну точку програми без не обидності довго шукати, потрібні сторінки.

Візуальна логіка побудована на контрастних відтінках темних кольорів, що дозволяє зосередитись саме на потрібному завданні для якого користувач серфить по платформі.

2. Надійність

Надійність інтерфейсу забезпечується чітким розмежуванням елементів. Кожен потрібний елемент відображається чіткою та структурованою інформацією яка визначена саме для отримання потрібних та структурованих даних.

Усі критичні дії супроводжуються модальними вікнами підтвердження, що не дозволяє випадково здійснювати не бажані дії.

Тобто надійність забезпечується за рахунок відображення структурованої та коректної інформації, яка не втрачає актуальності через постійне оновлення.

3. Простота

Простота інтерфейсу проявляється в кожному елементі, що відображається на сторінці. Немає ніяких зайвих розділів, які можуть заплутати користувача, або

відлякати його складністю інтерфейсу. Таке подання забезпечує в користувча певний рівень спокою стосовно платформи та дозволяє розслабитись та сконцентруватись на потрібній роботі.

Узагальнюючи, інтерфейс миттєво реагує на події, та відображає тільки актуальну інформацію що надходить від бази даних або API біржі.

Адаптивний дизайн забезпечує ідеальне відображення сторінки та дозволяє менше переживати незручності. Хочу наголосити ще разок на навбарі який досить зручно та ефективно забезпечує перехід між основними компонентами та дозволяє швидко взаємодіяти з веб-застосунком.

2.2 Опис архітектури, функціональності та реалізації розробленого рішення

Опис основної бізнес-логіки застосунку

Бізнес-логіка є центром системи, що забезпечує інтеграцію всіх функцій програми, зокрема інтеграції з API, управлінням торгових ботів, обробку запитів користувача, та за роботу з даними загалом. Уся логіка побудована за чітким розподілом відповідальності між шарами.(UI, Controller, Manager, Service, Repository).

Опис процесу створення акаунту користувача та логін користувача.

Процес реєстрації користувача

Процес створення акаунту користувача є багато рівневим і включає як валідацію введених користувачем даних, так і подальшу інтеграцію з зовнішнім API-біржі.

На ***Рис.1.1*** зображено процес створення акаунту користувача.

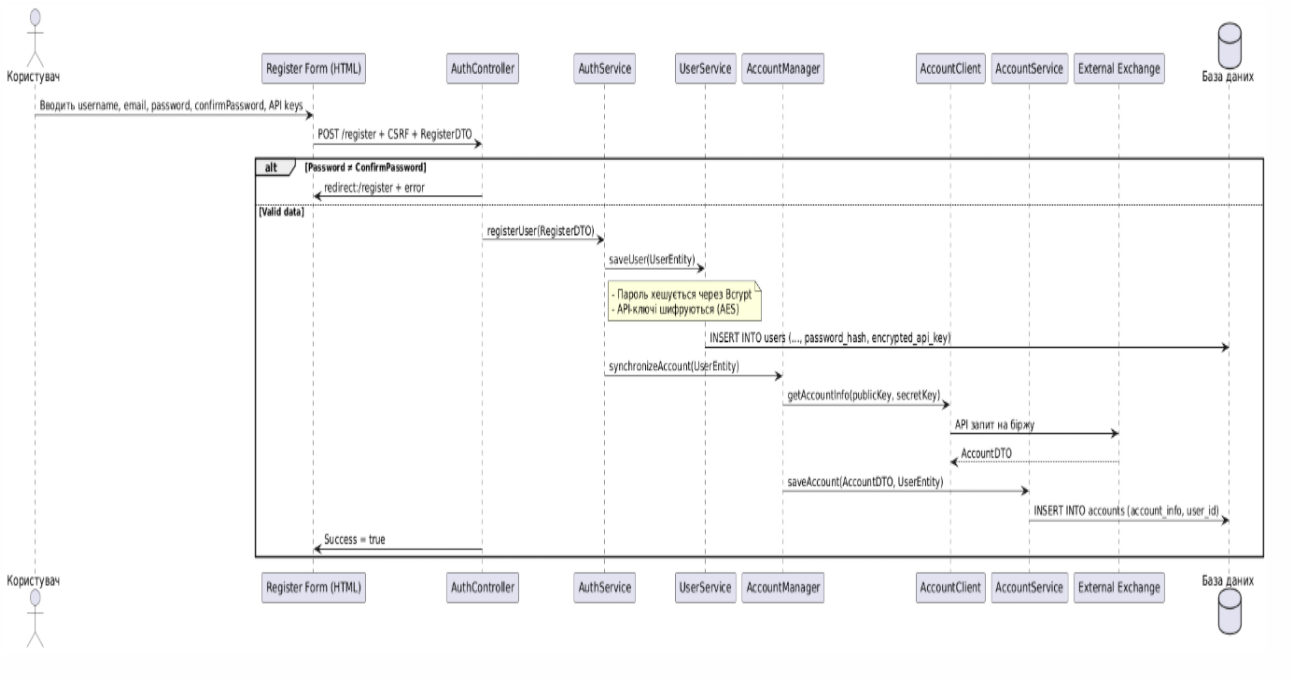

Рис.1.1

Пояснення

Взаємодія систем з користувачем відбувається за допомогою MVC архітектури. Коли користувач вводить дані обробка починається з Controller та делегується всім іншим сервісам, що відповідають за валідацію, шифрування, збереження до бази даних та інтеграція з зовнішніми системами. Розподілення відповідальностей вже описував.

Ключовими аспектами реалізації є

1. Обробка помилок на рівні контролера за зворотнім зв’язком.
2. Валідація даних, введених користувачем.
3. Шифрування паролів та API-ключів для забезпечення відповідальності.
4. Інтеграція з біржою через зовнішні API і збереження інформації.
5. Збереження даних користувача до бази даних.

Процес логіну користувача

Процес аутентифікації користувача в системі реалізований за допомогою стандартних механізмів Spring Security. Його основна ціль надати захищений

доступ до системи для користувачів зі владними обліковими даними. Схематично зображено на ***Рис. 1.2***

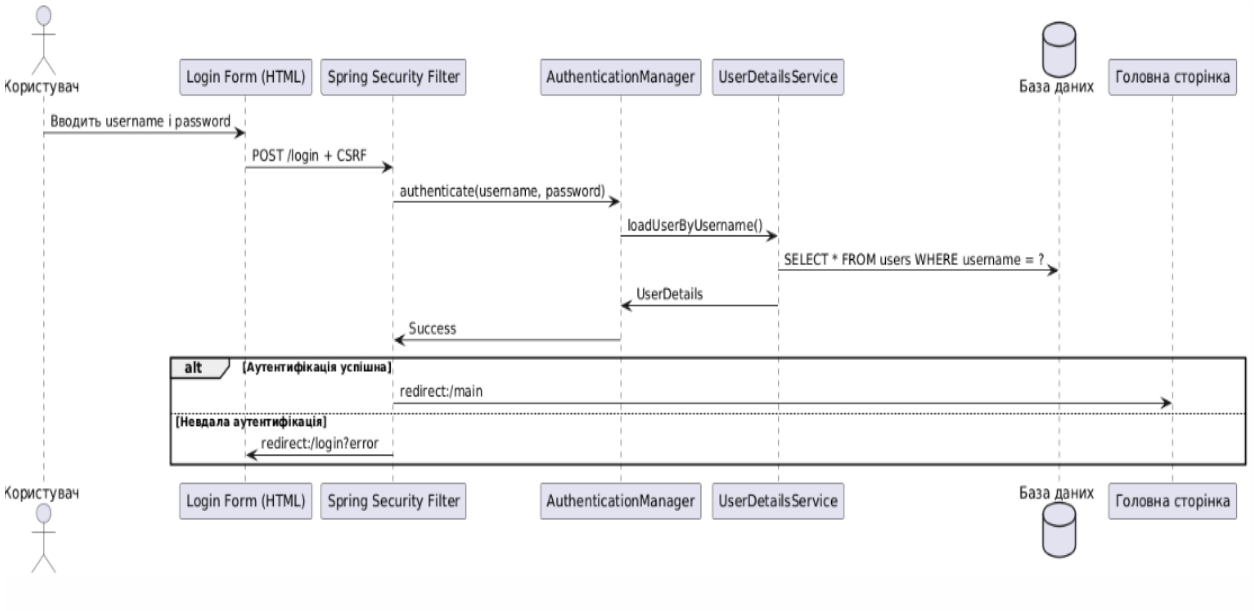

Рис. 1.2

Пояснення

Схема показує типовий процес логіну користувача. Після того як користувач вводить всі дані в формі логіну та натискає кнопку «Увійти», запит автоматично потрапляє до *Spring Security Filter Chain*, без необхідності ручного втручання у Controller.

Компонент *UsernamePasswordAuthetificationFilter* перехоплює запит та передає його в *AuthtificationManager*, який викликає реалізацію *UserDetailsService* для пошуку користувача в базі даних. Якщо користувача знаходить, тоді відбувається звірка паролів.

У разі успішного процесу користувача перекине на головну сторінку.

Опис аналітичних сторінок

Опис головної сторінки

Коли користувач переходить на головну сторінку, система збирає всі необхідні дані, щоб передати їх до головної сторінки користувача. На ***Рис. 1.3*** зображено процес завантаження головної сторінки.

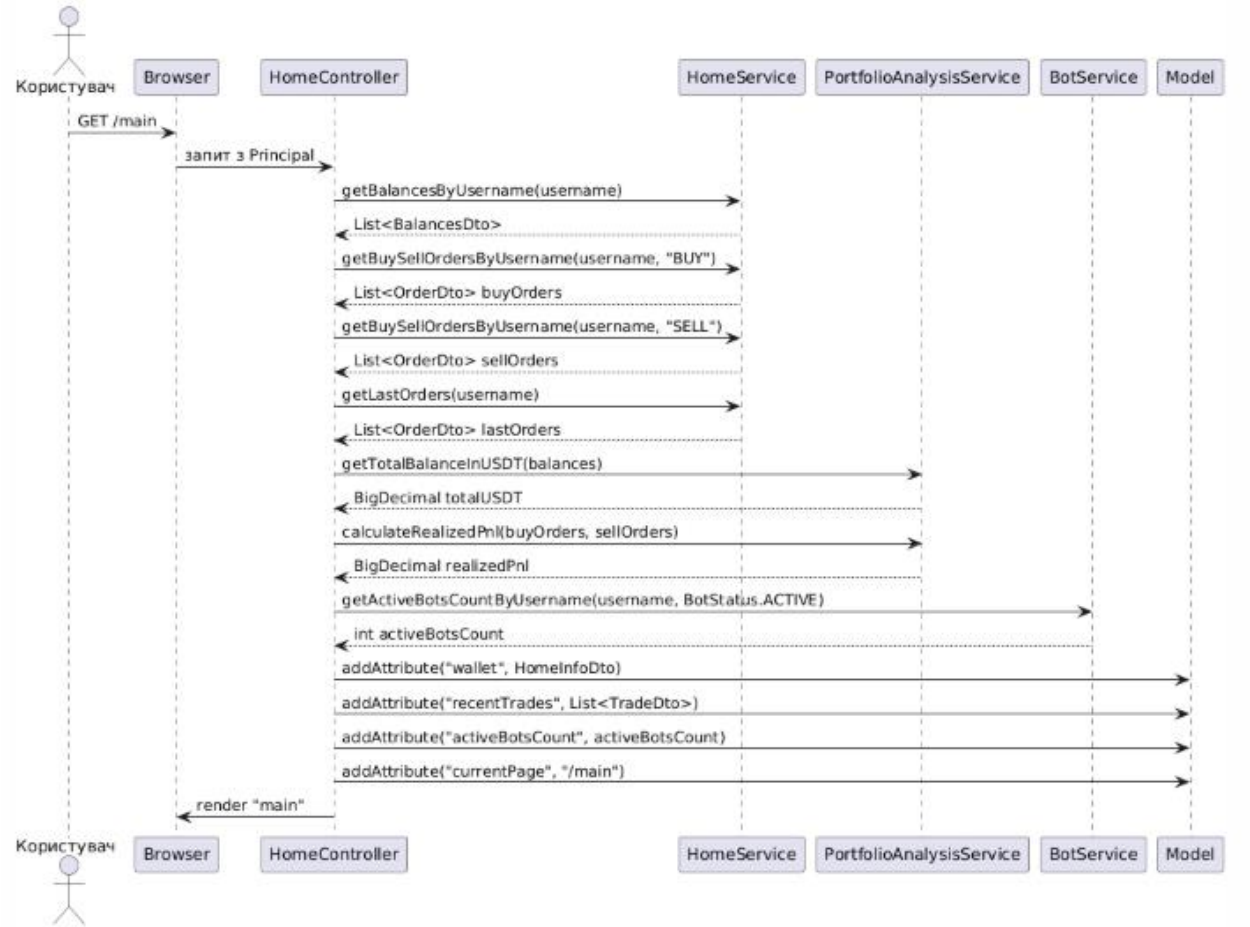

Рис. 1.3

Пояснення

Коли користувач надсилає запит до маршруту ***"/main"***, щоб перейти до головної сторінки, спочатку запит надходить до браузеру, який потім передає його до ***Home Controller,*** разом зі об'єктом Principal, що містить інформацію про поточного користувача. Контролер делегує отримання даних сервісу Home Service. Спочатку витягуються баланси акаунта користувача через UserService та конвертуються в BalancesDto, потім отримуються куплена та продані ордери через фільтрацію списку. Пізніше контролер запитує OrderPersistanceService останні угоди користувача для їх відображення у шаблоні. Далі PortfolioAnalysisService

конвертує всі активи в еквівалент USDT через взаємодію з зовнішніми API, та обчислює реалізований PNL за допомогою алгоритму, який розраховує результат на основі BUY та SELL ордерів. Та визнається кількість активних ботів в користувача. Усі зібрані дані передаються через модель до шаблону та там відображаються.

Сторінка гаманця

Після того як користувач переходить по маршруту ***"/wallet",*** веб-застосунок починає завантажувати всі потрібні дані для відображення. Схематично зображено на ***Рис. 1.4***

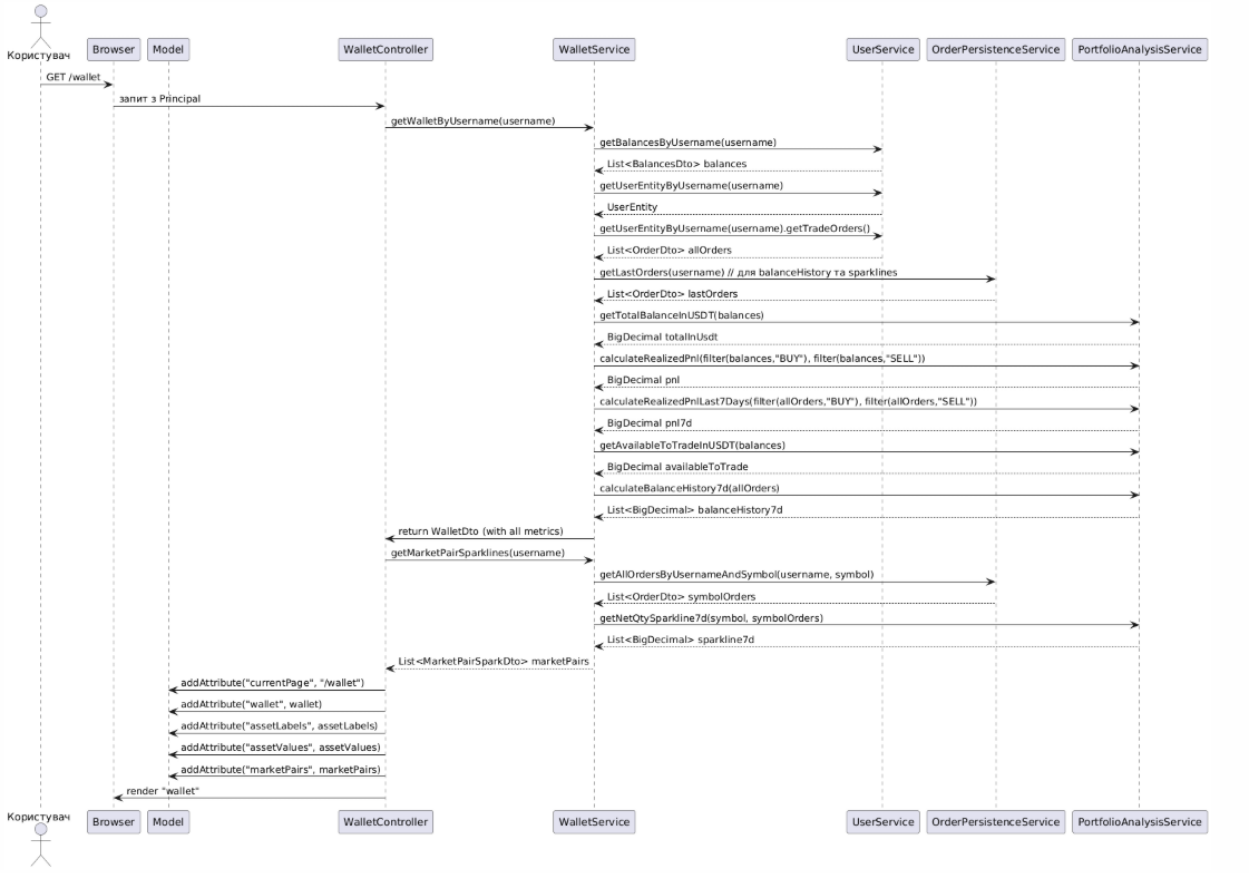

Рис. 1.4

Пояснення

На цій схемі відбувається процес отримання аналітичних даних для сторінки гаманця. Wallet Controller приймає цей запит за допомогою об'єкту Principal та

делегує подальшу обробку до Wallet Service. Спочатку сервіс за допомогою UserService отримує інформацію про списки та формує вільні та заблоковані кошти для кожного активу. Далі отримані дані розраховується та будується історія балансу користувача, паралельно запитує у OrderPersistanceService останні десять угод для спарклайнів та статистичних віджетів. На основі зібраних даних виконується низка аналітичних розрахунків через PortfolioAnalysisService та конвертує всі активи в USDT, обчислює PNL, визначає обсяг доступних коштів, а також формує історію зміни балансу за останні 7 днів. Всі отримані дані формуються в один об’єкт та передаються до шаблону для відображення користувачу.

Сторінка профілю користувача

Коли користувач відкриває маршрут «/profile», система завантажує всі дані по користувачу та його акаунту. ***Рис.1.5***

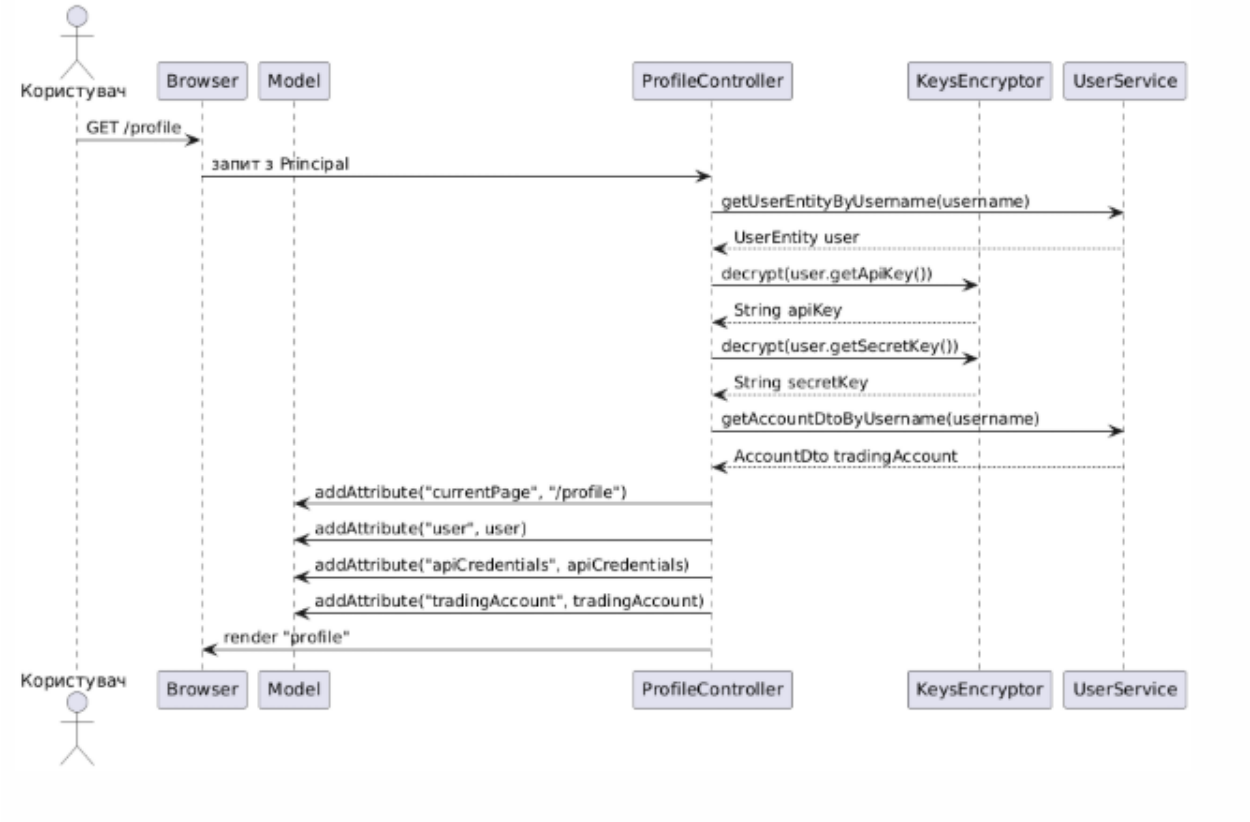

Рис. 1.5

Пояснення

Спочатку запит обробляється за допомогою браузера далі він делегується ProfileController разом із об’єктом Principal, що містить ім’я аутентифікованого користувача. Далі контролер викликає метод що повертає користувача по його username, щоб завантажити повну сутність з бази даних. Наступним кроком контролер отримує розшифровані API-ключі через KeysEncryptor. Після цього ProfileController звертається, щоб отримати інформація про стан акаунту. Потім контролер упаковує всі отримані дані в відповідні Dto для їх відображення в шаблоні.

Опис логіки роботи з ботами.

<u>Процес створення та запуску нового торгового бота</u>

Після заповнення та відправлення форми створення бота, BotController приймає запит та делегує його збереження параметрів BotService, а також створює об’єкт TradingBotManager. Менеджер перевіряє, чи є вже активний бот з таким символом та інтервалом, якщо ні то створює нового при потребі, синхронізує історію свічок за допомогою CandleManager. Та формує першу торгову угоду через TradeManager. ***Рис. 1.6***

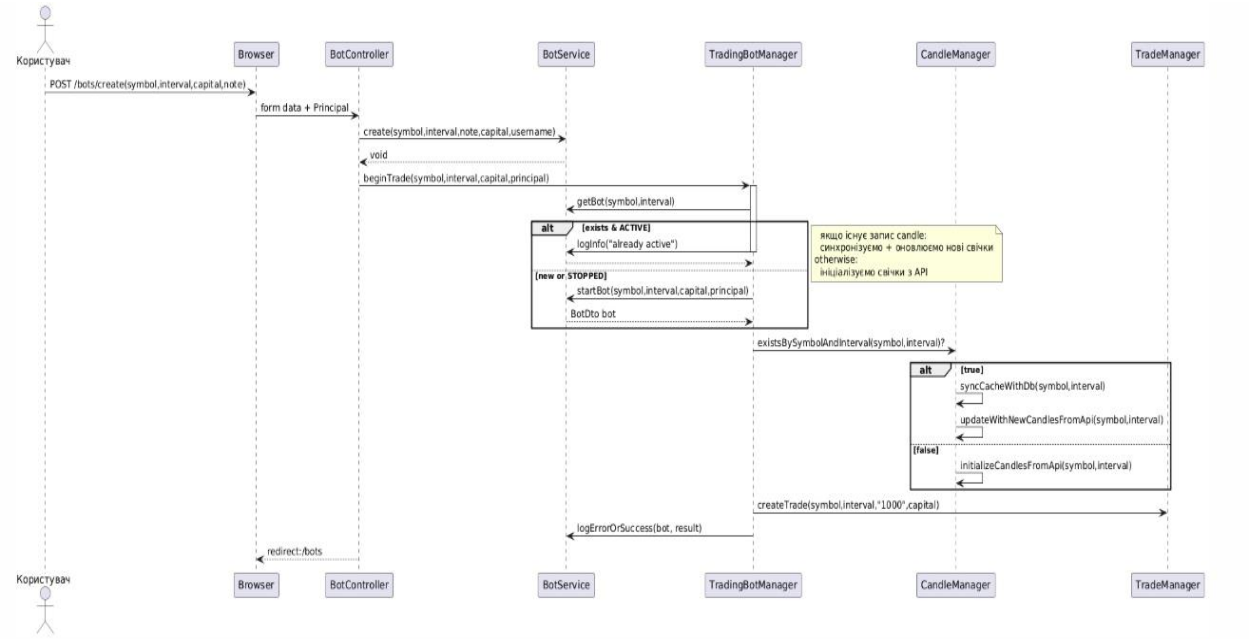

Рис. 1.6

Пояснення

Після того як користувач надсилає запит для створення бота до BotController логіка делегується до BotService через метод create та передає керування TradingBotManager. Менеджер перевіряє статус існуючого бота та створює нового за потреби. Далі він синхронізує дані свічок для аналізу ринку та прийняття рішення для відкриття або закриття позиції.

Оновлення свічок та аналіз ринку

Згідно з розкладом планувальника TradingBotManager регулярно завантажує список активних ботів, отримує останню свічку через CandleManager, оновлює потрібні дані та перевіряє цілісність та стан API через AlereService. Після аналізує ринок за допомогою отриманих даних в TradingAnalysis і, якщо отримано сигнал то створюється відповідний ордер через TradeManager. Зображення на ***Рис. 1.7***

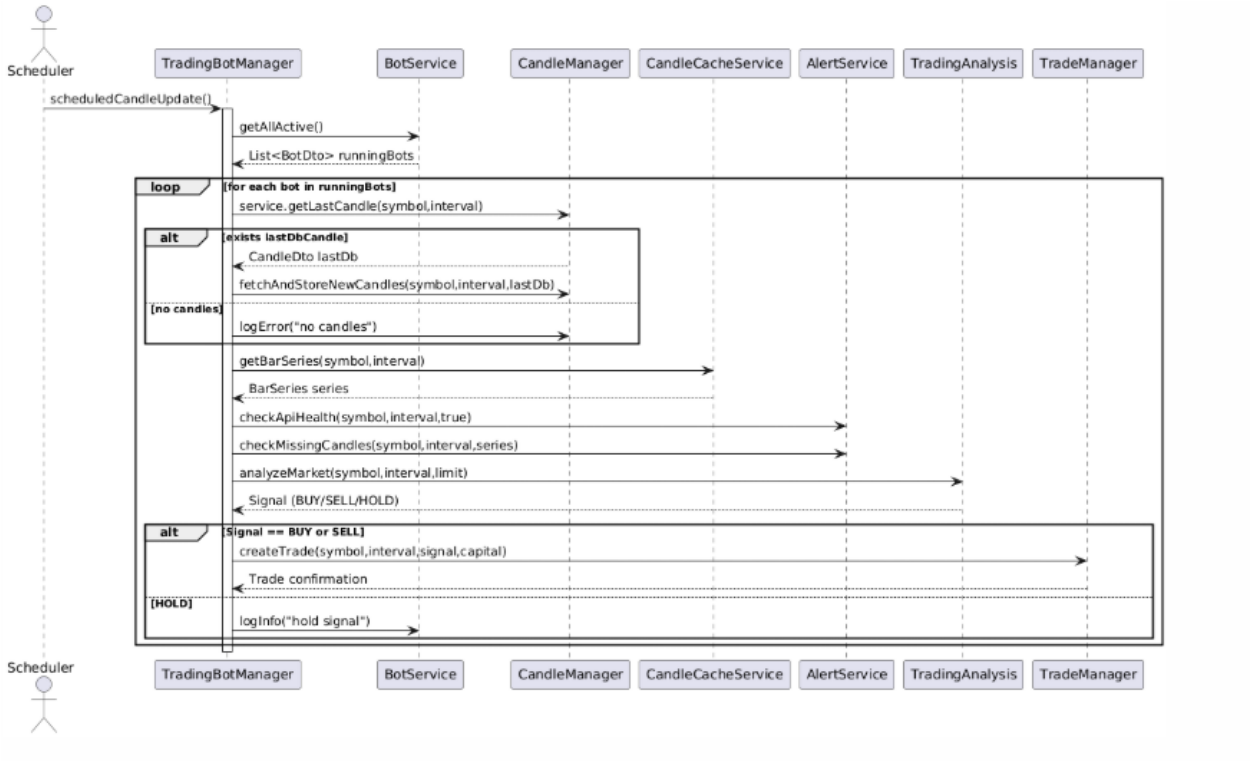

Рис. 1.7

Пояснення

Менеджер запускається за розкладом, отримує всі активні боти, оновлює всі потрібні дані. Конвертує потрібні дані до барів та аналізує їх. Далі на основі сигналу створює ордер.

Опис бази даних

Опис сутностей та зв’язків між ними.

Сутність у контексті програмування – це об’єкт, що має певні поля та пов’язаний з таблицею в базі даних. Використовуються, щоб простіше взаємодіяти з базою даних. Нижче будуть представленні всі сутності, що наявні в даному веб-застосунку та їх опис.

UserEntity ***Рис. 1.8***

```
@Entity
@Data
@Builder
@RequiredArgsConstructor
@AllArgsConstructor
@Table(name = "users")
public class UserEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String apiKey;
```

```
    @Column(nullable = false)
    private String secretKey;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(nullable = false)
    private String role;  // 'admin', 'user', etc.

    private LocalDateTime createdAt;

    private LocalDateTime updatedAt;

    // Relationships
    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    private AccountEntity account;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval =
true)
    private List<OrderEntity> tradeOrders;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<BotEntity> bots;

}
```

Рис. 1.8

Пояснення

Це сутність для користувача яка пов’язана з таблицею “users” в базі даних. Та містить такі поля.

- **Long id** – це ідентифікатор користувача.
- **String username** – ім’я користувача, за яким відбувається низка процесів.
- **String password** – пароль користувача, який потрібен для безпеки користувача.
- **String apiKey** – публічний API-ключ, що забезпечує конект з біржою та акаунтом користувача.
- **String secretKey** – секретний ключ, який потрібен для підписів POST-запитів до біржі.
- **String email** – email користувача.
- **String role –** роль користувача в системі, яка визначає його права.
- **LocalDateTime createdAt –** дата створення користувача.
- **LocalDateTime updatedAt –** дата останнього оновлення користувача.
- **AccountEntity account –** посилання на акаунт для користувача який має зв'язок типу OneToOne з сутністю AccountEntity, це означає, зо один користувач може мати лише один акаунт.
- **List<OrderEntity> tradeOrders –** посилання на список сутностей OrderEntity, які містять інформацію про ордери користувача, та має зав’язок типу OneToMany, що означає що один користувача може мати багато ордерів.
- **List<BotEntity> bots –** посилання на список ботів користувача, поле має зв'язок типу OneToMany з BotEntity, що означає що один користувач може мати багато ботів.

AccountEntity ***Рис.*** **1.9**

```
@Entity
@Table(name = "accounts")
@Data
```

```
public class AccountEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private BigDecimal makerCommission;
    private BigDecimal takerCommission;
    private BigDecimal buyerCommission;
    private BigDecimal sellerCommission;

    private boolean canTrade;
    private boolean canWithdraw;
    private boolean canDeposit;

    private LocalDateTime updateTime;
    private String accountType;

    // Relationships
    @OneToOne
    @JoinColumn(name = "user_id", nullable = false)
    private UserEntity user;

    @OneToMany(mappedBy = "account", cascade = CascadeType.ALL, orphanRemoval
= true)
    private List<BalancesEntity> balances;
}
```

Рис.1.9

Пояснення

AccountEntity – це сутність, що пов’язана з таблицею «accounts». Вона має такі поля.

- **Long id** - ідентифікатор акаунту.
- **BigDecimal makerCommission –** тобто того, хто створює ліміт-ордер (запит на купівлю/продаж), але не одразу виконує його. Зазвичай менша комісія.
- **BigDecimal takerCommission –** комісія того, хто виконує ордер миттєво по ринку. Як правило, трохи більша комісія.
- **BigDecimal buyerCommission -** комісія при купівлі активів. Може дублювати taker/maker, але виділена окремо для гнучкості.
- **BigDecimal sellerCommission -** комісія при продажі активів.
- **boolean canTrade –** поле яке говорить чи має користувач доступ до торгівлі.
- **boolean canWithdraw –** поле яке каже повідомляє користувачу чи може він виводити кошти з акаунта.
- **boolean canDeposit –** це поле яке каже чи може користувач вносити кошти на акаунт.
- **LocalDateTime updateTime – дата останнього оновлення акаунту.**
- **String accountType –** тип акаунту, **SPOT, MARGIN, FUTURES.**
- **UserEntity user** – посилання на користувача якому належить акаунт. Зв'язок типу OneToOne, та мапиться за user_id.
- **List<BalancesEntity> balances** – список балансів які доступні на акаунті. Зв'язок типу OneToOne, мапиться за акаунтом.

BalancesEntity ***Рис. 1.9***

```
@Entity
@Table(name = "balances")
@Data
public class BalancesEntity {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String asset;
    private BigDecimal free;
    private BigDecimal locked;

    // Relationships
    @ManyToOne
    @JoinColumn(name = "account_id", nullable = false)
    private AccountEntity account;
}
```

Рис.1.9

Пояснення

BalancesEntity – це сутність, що пов’язана з таблицею «balances» в базі даних. Вона має такі поля

- **Long id** - ідентифікатор балансу.
- **String asset** – тип валюти.
- **BigDecimal free** – вільні кошти в валюті
- **BigDecimal locked** – заблоковані кошти в валюті
- **AccountEntity account** – посилання на сутність користувача. Зв'язок ManyToOne, значить що багато балансів можуть мати лише одного користувача.

OrderEntity ***Рис.2.1***

```
@Entity
@Table(name = "trade_orders")
@Data
public class OrderEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private Long orderId;
    private String symbol;
    private String side;
    private String type;
    private String status;
    private BigDecimal quantity;
    private BigDecimal price;
    private BigDecimal executedQuantity;
    private BigDecimal cummulativeQuoteQuantity;

    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    private String errorCode;
    private String errorMessage;

    // Relationships
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
```

```
    private                    UserEntity                    user;
}
```

Рис.2.2

Пояснення

Сутність OrderEntity- це сутність що пов’язана з таблицею «trade_orders» в базі даних та має такі поля.

- Long id – ідентифікатор запису в базі даних.
- Long orderId – ідентифікатор ордеру.
- String symbol – тип валюти.
- String side – Buy, Sell це поле містить дані про те чи це ордер на покупку чи на продаж.
- String type – це тип ордеру, тобто LIMIT, MARKET.
- String status – це поле повідомляє про статус ордеру чи він вже виконався, не виконався або в процесі.
- BigDecimal quantity – кількість монети з якою намагаються зробити операцію.
- BigDecimal price – ціна за якої була куплена або продана валюта.
- BigDecimal executedQuantity – виконана кількість, тобто кількість що вже обробилась.
- BigDecimal cummulativeQuoteQuantity –
- LocalDateTime createdAt – дата створення ордеру
- LocalDateTime updatedAt – дата останнього оновлення ордеру.
- String errorCode – поле помилки, яке повідомляє код помилки.
- String errorMessage – поле помилки яке повідомляє message.
- UserEntity user – посилання на сутність UserEntity та має зв'язок типу ManyToOne, що означає що багато ордерів можуть мати лише один акаунт.

CandleEntity ***Рис.2.3***

```
@Data
@Entity
@Table(name = "candles")
public class CandleEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String symbol;
    private String interval;
    private LocalDateTime openTime;

    private BigDecimal open;
    private BigDecimal high;
    private BigDecimal low;
    private BigDecimal close;
    private BigDecimal volume;

    private LocalDateTime closeTime;
    private BigDecimal quoteAssetVolume;
    private Long numberOfTrades;
    private BigDecimal takerBuyBaseAssetVolume;
    private BigDecimal takerBuyQuoteAssetVolume;


}
```

Рис.2.3

Пояснення

CandleEntity – це сутність, яка пов’язана з таблицею «candles» та має такі поля.

- Long id - ідентифікатор сутності.
- String symbol – валюта сутності.
- String interval – інтервал.
- LocalDateTime openTime – час відкриття свічки.
- BigDecimal open – ціна відкриття.
- BigDecimal high – максимальна ціна.
- BigDecimal low – мінімальна ціна.
- BigDecimal close – ціна закриття.
- BigDecimal volume – обсяг торгів активу за період.
- LocalDateTime closeTime – час закриття свічки.
- BigDecimal quoteAssetVolume – обсяг торів у валюті котирування.
- Long numberOfTrades – кількість угод, що були виконані за період часу.
- BigDecimal takerBuyBaseAssetVolume – скільки базового активу було викуплено за маркет-ордерами.
- BigDecimal takerBuyQuoteAssetVolume – сума в валюті котирування, на яку було куплено активів через маркет-ордера.

Опис таблиць в базі даних PostgreSQL

Архітектурний огляд системи

Архітектура бази даних побудована за принципом централізованого управління користувачами, де основною сутністю виступає таблиця users, від якої відходять всі інші компоненти системи. Загальна структура включає шість основних сутностей, що забезпечують комплексне управління торговими операціями.

Детальний опис структур даних

Таблиця users - Користувачі системи

Центральна таблиця системи, що містить основну інформацію про користувачів платформи.

Структура полів –

- id (serial) - первинний ключ, унікальний ідентифікатор користувача
- username (varchar(255)) - унікальне ім'я користувача в системі
- password (varchar(255)) - хешований пароль для аутентифікації
- api_key (varchar(255)) - ключ для доступу до API торгової біржі
- secret_key (varchar(255)) - секретний ключ для підпису API-запитів
- email (varchar(255)) - електронна адреса користувача
- role (varchar(255)) - роль користувача в системі (адміністратор, звичайний користувач)
- created_at (timestamp) - дата та час створення облікового запису
- updated_at (timestamp) - дата та час останнього оновлення даних

Зв'язки: один-до-багатьох з таблицями accounts, bots, trade_orders

Таблиця accounts - Торгові рахунки

Зберігає параметри торгових рахунків користувачів на криптовалютних біржах.

Структура полів –

- id (serial) - первинний ключ рахунку
- maker_commission (numeric) - розмір комісії за створення ліквідності
- taker_commission (numeric) - розмір комісії за споживання ліквідності
- buyer_commission (numeric) - комісія, що стягується з покупця
- seller_commission (numeric) - комісія, що стягується з продавця
- can_trade (boolean) - дозвіл на здійснення торгових операцій
- can_withdraw (boolean) - дозвіл на виведення коштів
- can_deposit (boolean) - дозвіл на поповнення рахунку
- update_time (timestamp) - час останнього оновлення параметрів рахунку
- account_type (varchar(255)) - тип торгового рахунку

- user_id (bigint) - зовнішній ключ, посилання на власника рахунку

Зв'язки: багато-до-одного з users, один-до-багатьох з balances

Таблиця balances - Баланси активів

Відображає поточні баланси криптовалютних активів на торгових рахунках.

Структура полів –

- id (serial) - первинний ключ запису балансу
- asset (varchar(255)) - символ криптовалютного активу (BTC, ETH, USDT тощо)
- free (numeric) - доступний для торгівлі баланс
- locked (numeric) - заблокований баланс (у відкритих ордерах)
- account_id (bigint) - зовнішній ключ, посилання на торговий рахунок

Зв'язки: багато-до-одного з accounts

Таблиця trade_orders - Торгові ордери

Містить інформацію про торгові ордери користувачів, їх параметри та стан виконання.

Структура полів –

- id (serial) - первинний ключ ордеру в системі
- order_id (bigint) - ідентифікатор ордеру на біржі
- symbol (varchar(255)) - торгова пара (наприклад, BTCUSDT)
- side (varchar(255)) - напрямок торгівлі (BUY/SELL)
- type (varchar(255)) - тип ордеру (MARKET, LIMIT тощо)
- status (varchar(255)) - поточний статус ордеру
- quantity (numeric) - загальна кількість активу в ордері
- price (numeric) - ціна виконання ордеру
- executed_quantity (numeric) - фактично виконана кількість
- cumulative_quote_quantity (numeric) - загальний обсяг у котируваній валюті
- created_at (timestamp) - час створення ордеру

- updated_at (timestamp) - час останнього оновлення статусу
- error_code (varchar(255)) - код помилки у разі невдалого виконання
- error_message (text) - детальний опис помилки
- user_id (bigint) - зовнішній ключ, посилання на користувача

Зв'язки - багато-до-одного з users

Таблиця bots - Торгові боти

Зберігає конфігурацію та параметри автоматизованих торгових ботів.

Структура полів –

- id (serial) - первинний ключ бота
- symbol (varchar(255)) - торгова пара для роботи бота
- interval (varchar(255)) - часовий інтервал аналізу (1m, 5m, 1h тощо)
- status (varchar(255)) - поточний стан бота (активний/неактивний)
- start_time (timestamp) - час початку роботи бота
- stop_time (timestamp) - час зупинки бота
- capital (numeric) - виділений капітал для торгівлі
- last_updated_time (timestamp) - час останнього оновлення
- error_message (text) - опис останньої помилки
- is_scheduled (boolean) - прапорець запланованого запуску
- note (text) - додаткові примітки до конфігурації
- user_id (bigint) - зовнішній ключ, посилання на власника бота

Зв'язки: багато-до-одного з users, один-до-багатьох з candles

Таблиця candles - Свічкові дані

Містить історичні ринкові дані у форматі японських свічок для аналізу торговими ботами.

Структура полів –

- id (serial) - первинний ключ запису
- symbol (varchar(255)) - торгова пара

- interval (varchar(255)) - часовий інтервал свічки
- open_time (timestamp) - час відкриття свічки
- close_time (timestamp) - час закриття свічки
- open (numeric) - ціна відкриття періоду
- high (numeric) - максимальна ціна за період
- low (numeric) - мінімальна ціна за період
- close (numeric) - ціна закриття періоду
- volume (numeric) - обсяг торгів у базовій валюті
- quote_asset_volume (numeric) - обсяг торгів у котируваній валюті
- number_of_trades (bigint) - кількість укладених угод
- taker_buy_base_asset_volume (numeric) - обсяг покупок у базовій валюті
- taker_buy_quote_asset_volume (numeric) - обсяг покупок у котируваній валюті

Зв'язки: логічний зв'язок з bots через поля symbol та interval

Схема взаємозв'язків

Структура бази даних демонструє ієрархічну модель з центральною сутністю users, від якої розгалужуються всі інші компоненти системи:

users (користувачі)
├── accounts (торгові рахунки)
│ └── balances (баланси активів)
├── bots (торгові боти)
└── trade_orders (торгові ордери)

candles (ринкові дані)

РОЗДІЛ 3 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТИ

3.1 Інструкція для користувачів

У цьому розділі буде покрокова інформація про те, як користуватись веб-застосунком з детальними поясненнями та візуальними підказками. Кожен крок супроводжується скріншотом інтерфейсу.

Мета цього розділу – забезпечити швидке орієнтування навіть для користувачів без досвіду.

Форма реєстрації та логіну

Реєстрація користувача ***Рис.2.4***

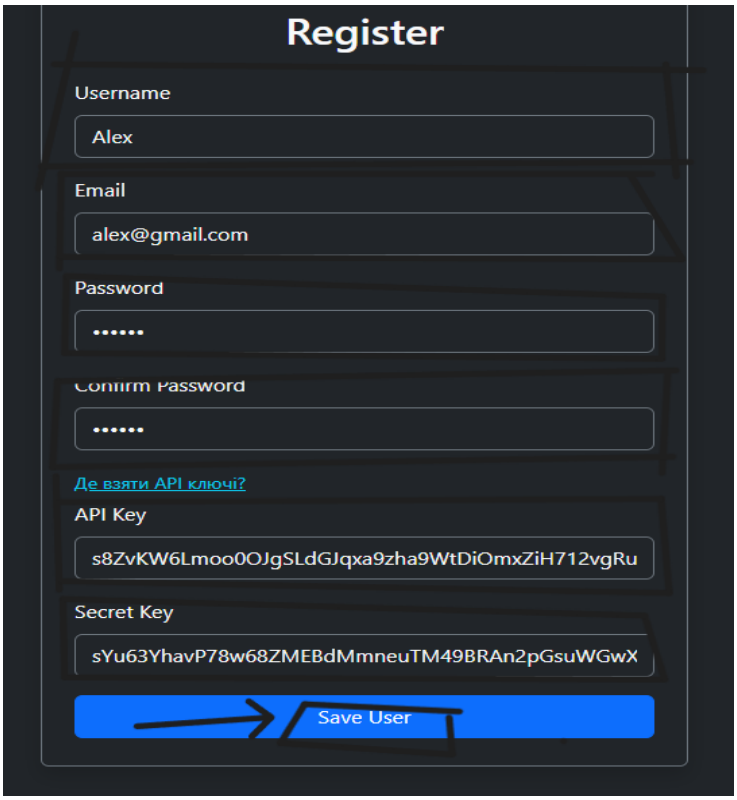

Рис.2.4

Для успішної реєстрації користувача потрібно заповнити всі виділенні на зображенні поля та натиснути кнопку Save User. Після чого ми отримаємо повідомлення зверху див. ***Рис.2.5***

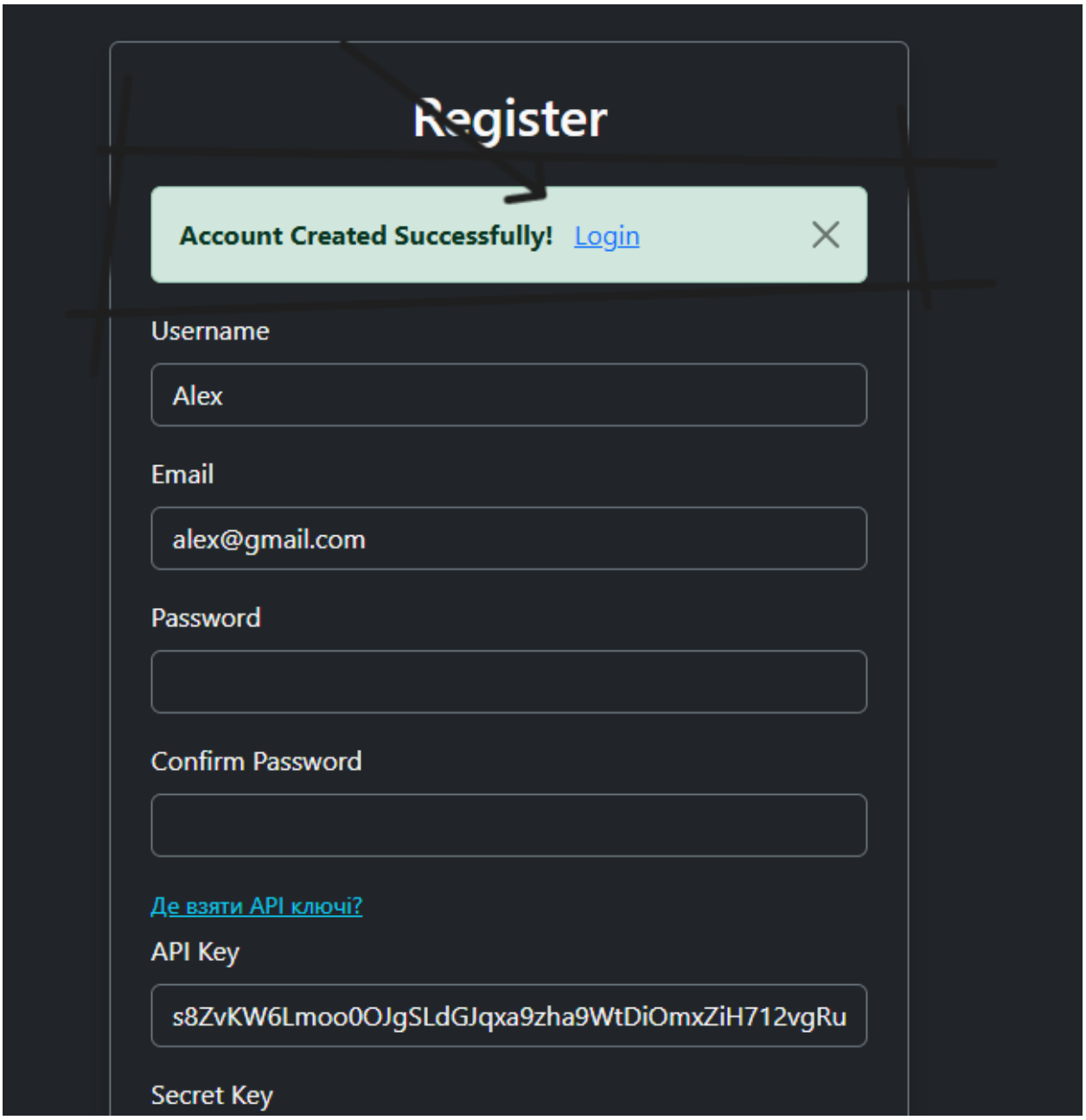

Рис.2.5

Після натискання на кнопку login користувача перекине на сторінку для логіну на ***Рис.2.6***

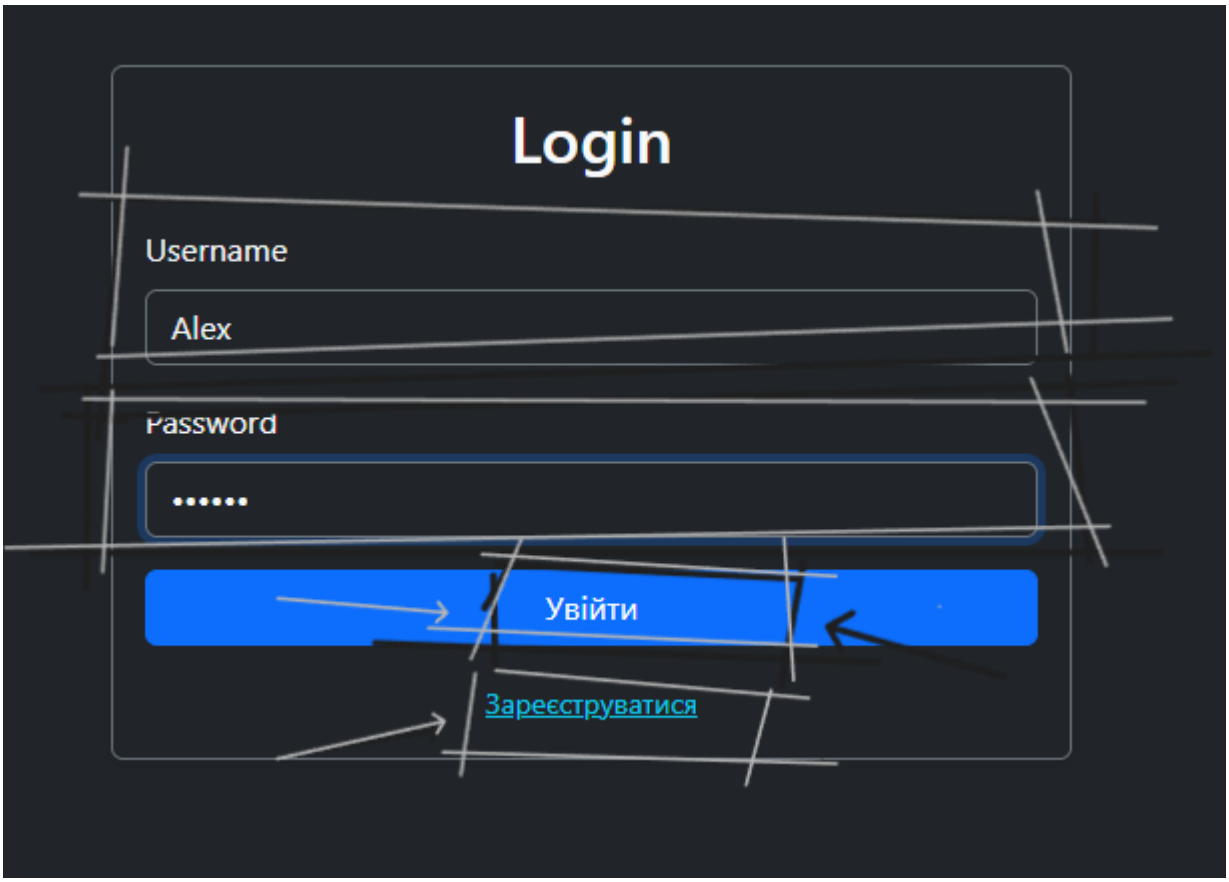

Рис.2.6

Це форма логіну користувача де йому потрібно ввести username та password та натиснути кнопку “Увійти” Після успішного логіну користувача перекине на головну сторінку.

Головна сторінка

Після успішної авторизації користувача перекине на головну сторінку ***Рис.2.7***

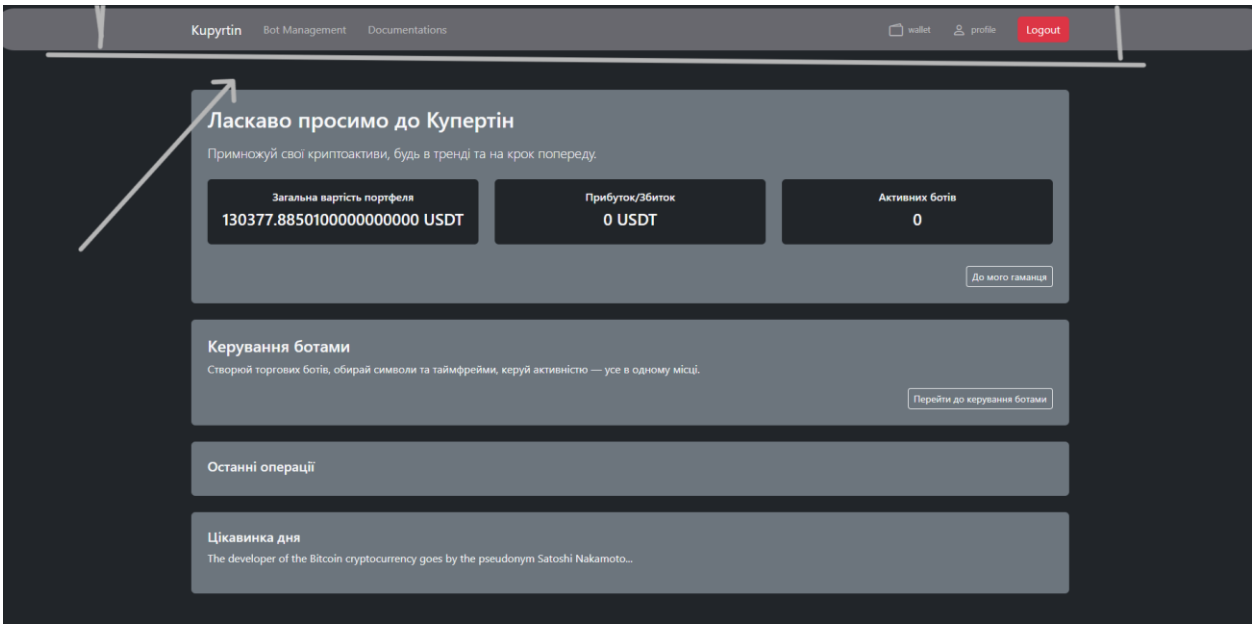

Рис.2.7

На цій сторінці відображається коротка аналітична інформація користувача. Також зверху видно розхвалений навбар. Для того, щоб перемістити з сторінку на сторінку достатньо лише натиснути на текст або на кнопки.

Профіль користувача

В профілі користувача є інформація про акаунт його стан API-ключі та інша інформація про користувача ***Рис.2.8***

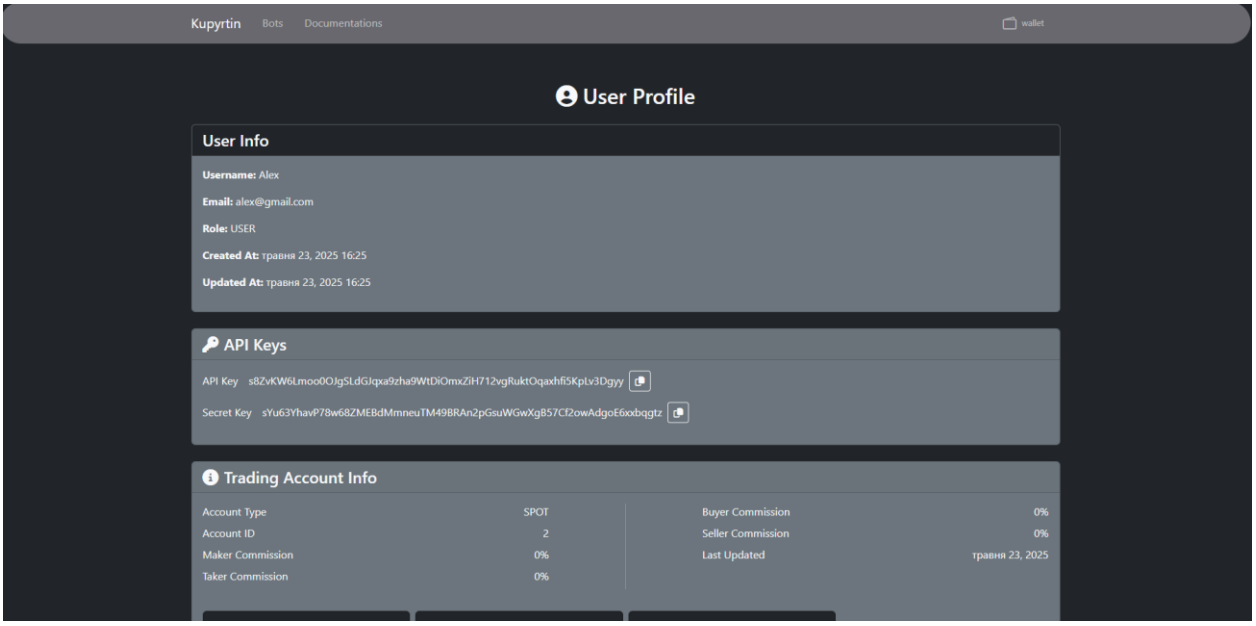

Рис.2.8

Гаманець користувача.

На цій сторінці відображається інформація про активи користувачів, про кошти які доступні та PNL. А також деякі аналітичні дані. ***Рис.2.9 Рис.3.1***

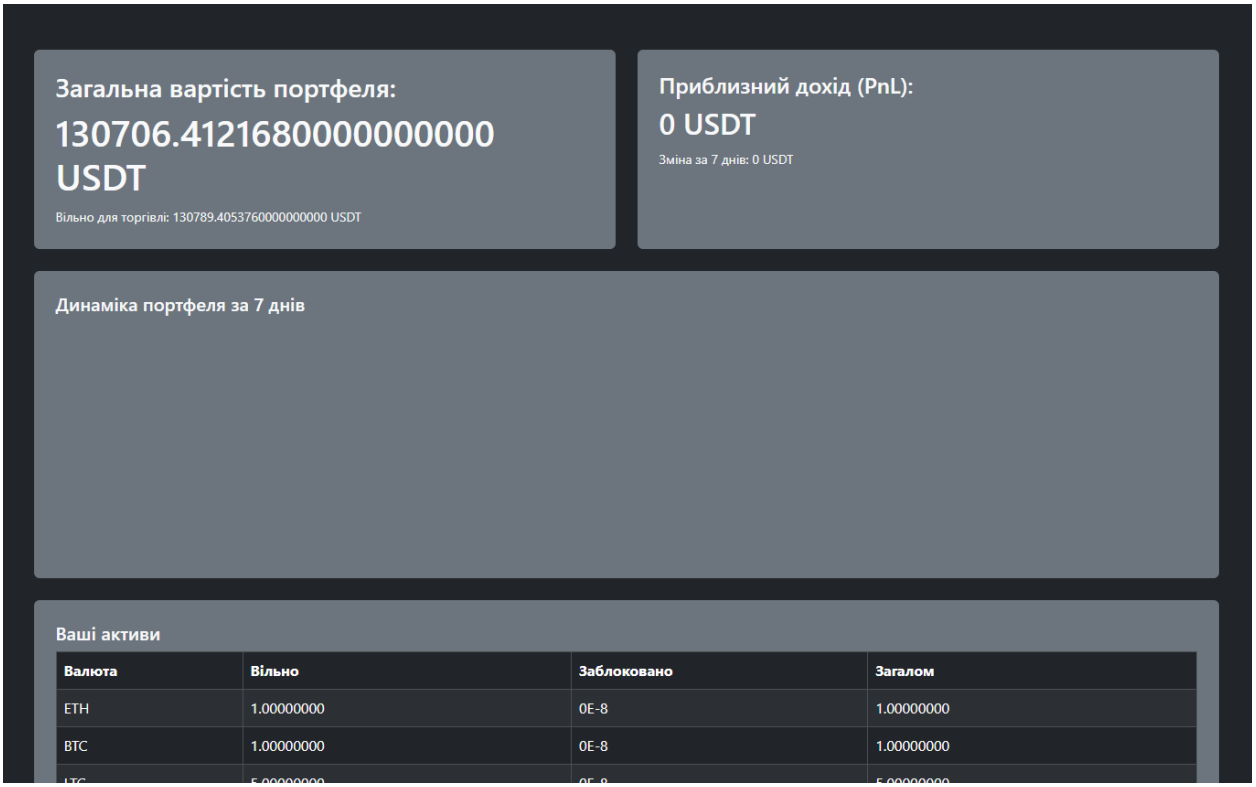

Рис.2.9

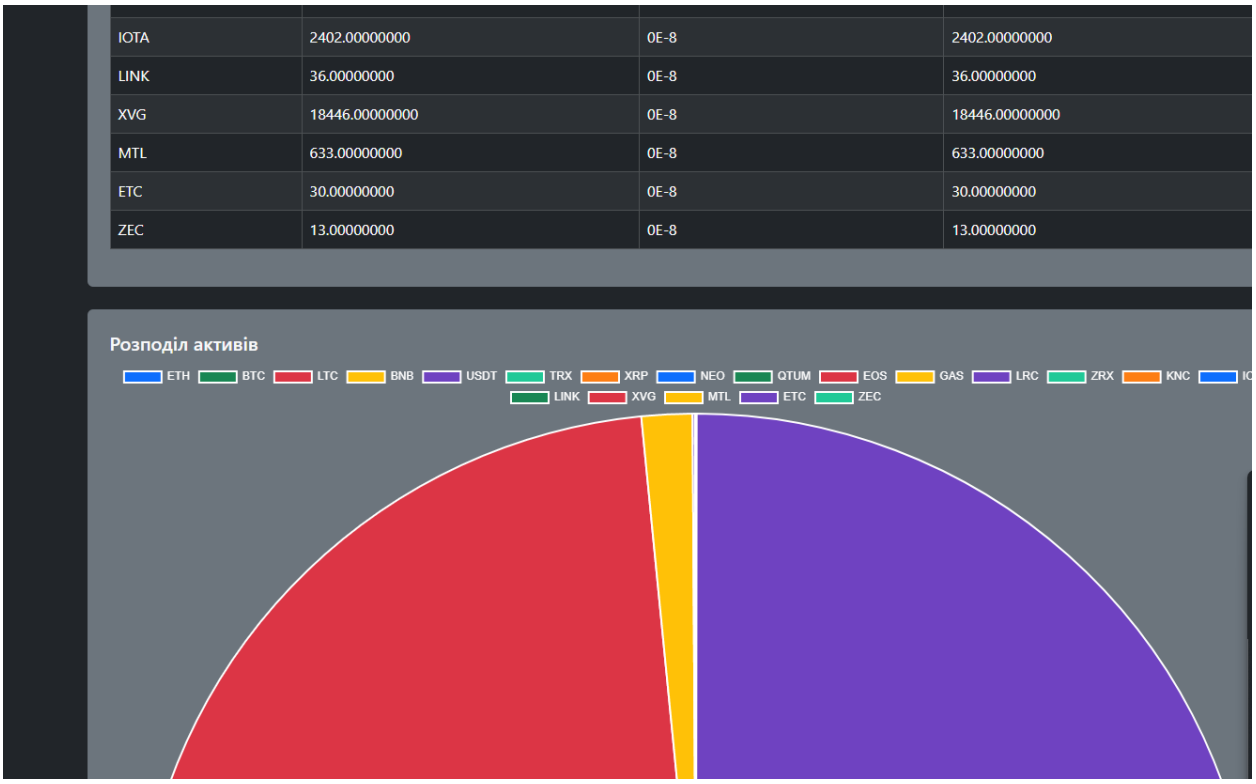
IOTA 2402.00000000 0E-8 2402.00000000
LINK 36.00000000 0E-8 36.00000000
XVG 18446.00000000 0E-8 18446.00000000
MTL 633.00000000 0E-8 633.00000000
ETC 30.00000000 0E-8 30.00000000
ZEC 13.00000000 0E-8 13.00000000
Розподіл активів
ETH BTC LTC BNB USDT TRX XRP NEO QTUM EOS GAS LRC ZRX KNC
LINK XVG MTL ETC ZEC

Рис.3.1

Сторінка з документацією

На даній сторінці відображається вся інформація про застосунок. Це місце де користувач може детально зрозуміти логіку роботи веб-застосунку. ***Рис.3.2***

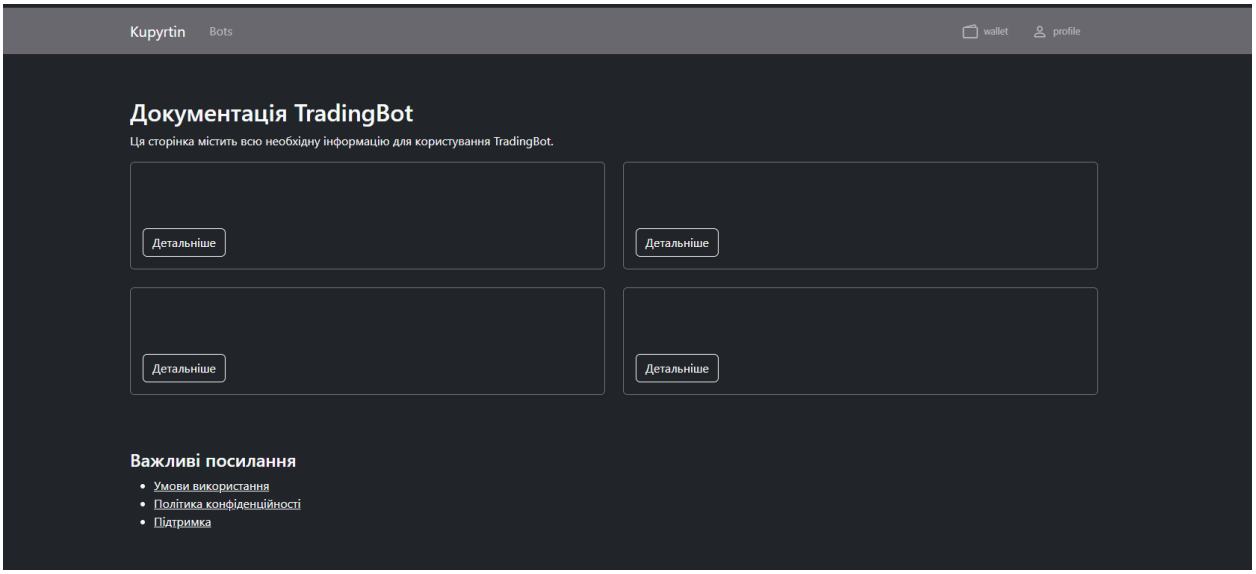

Рис.3.2

Сторінка керування ботами

Щоб потрапити на сторінку переходим через навбар «BotManagment». Далі тиснемо кнопку Create як на ***Рис.3.3***

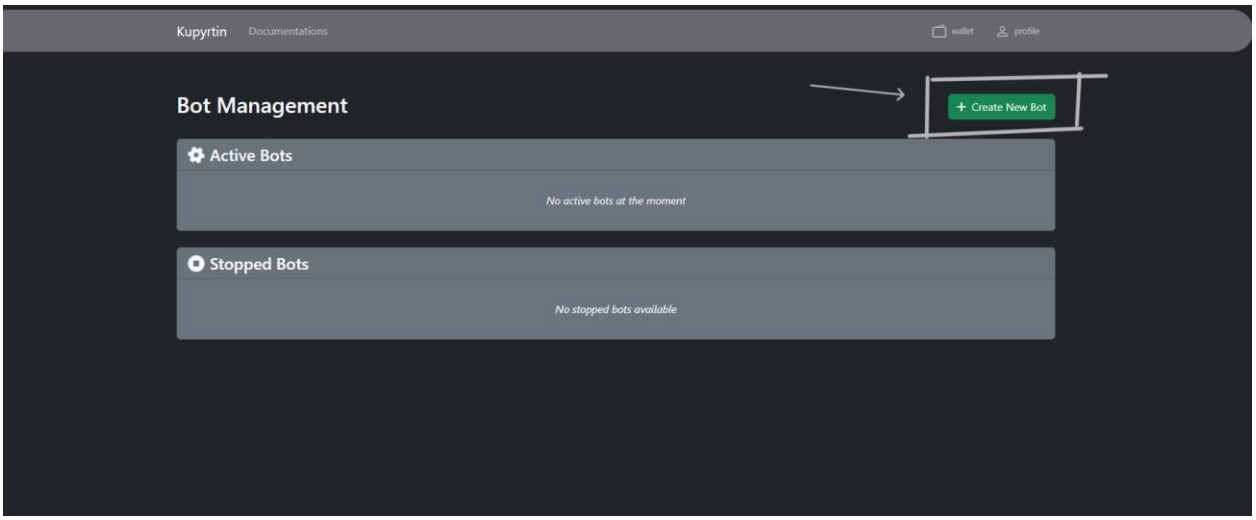

Рис.3.3

Далі з’явиться випадна форма де потрібно буде обрати з випадного списку криптовалюту та інтервал, потім в інших двох полях з права зверху ввести суму з якою буде працювати бот, та якісь можливо помітки або нотатки для боту. Та натиснути старт бот. ***Рис.3.4*** та ***Рис.3.5***

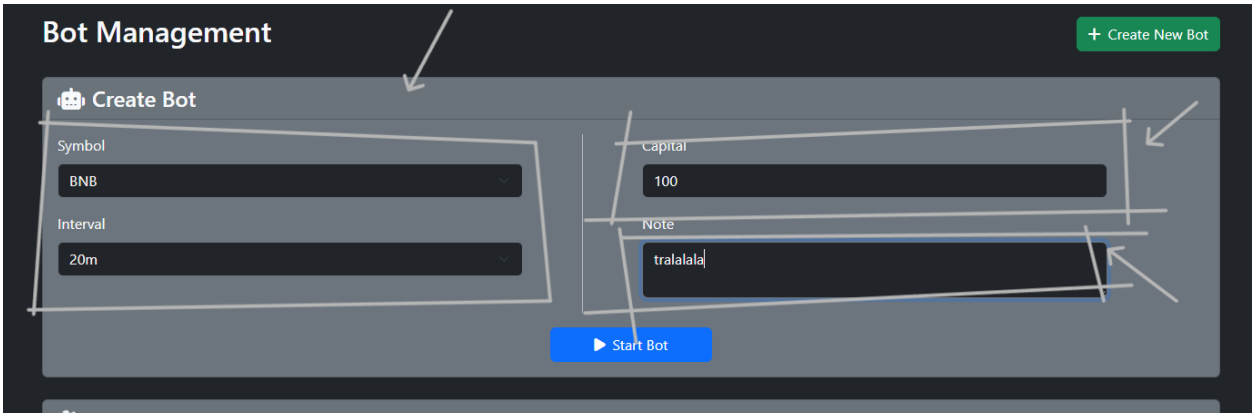

Рис.3.4

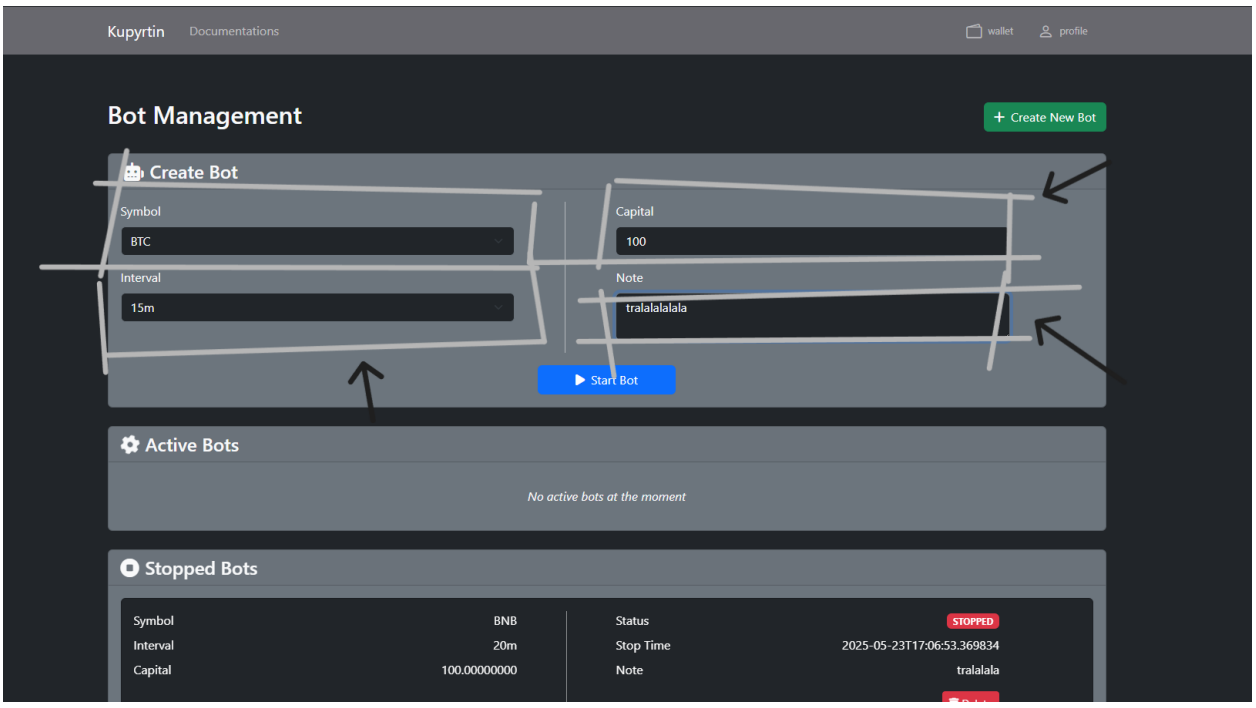

Рис.3.5

Після успішного створення боту користувач побачить ось таку сторінку ***Рис.3.6***

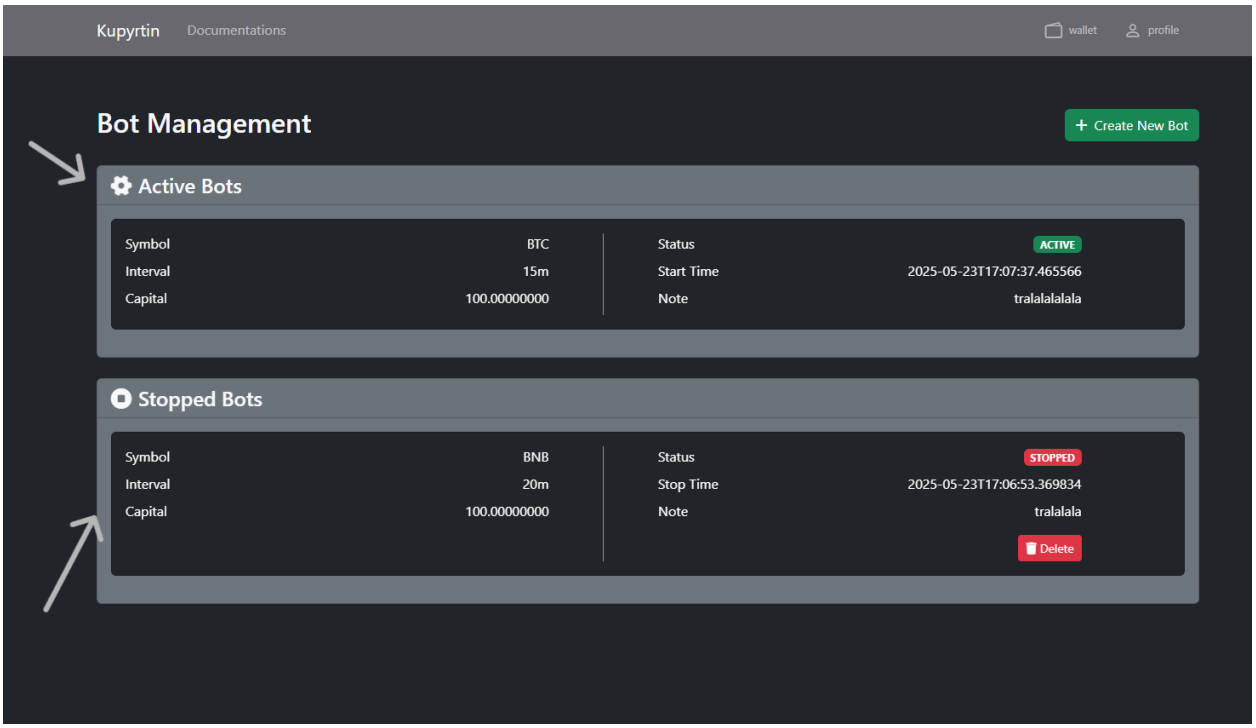

Рис.3.6

Та якщо користувач захоче зупинити якогось бота йому потрібно клацнути на нього, після чого його перекине на сторінку самого бота, де користувач має змогу зупинити та поновити бота. ***Рис.3.7***

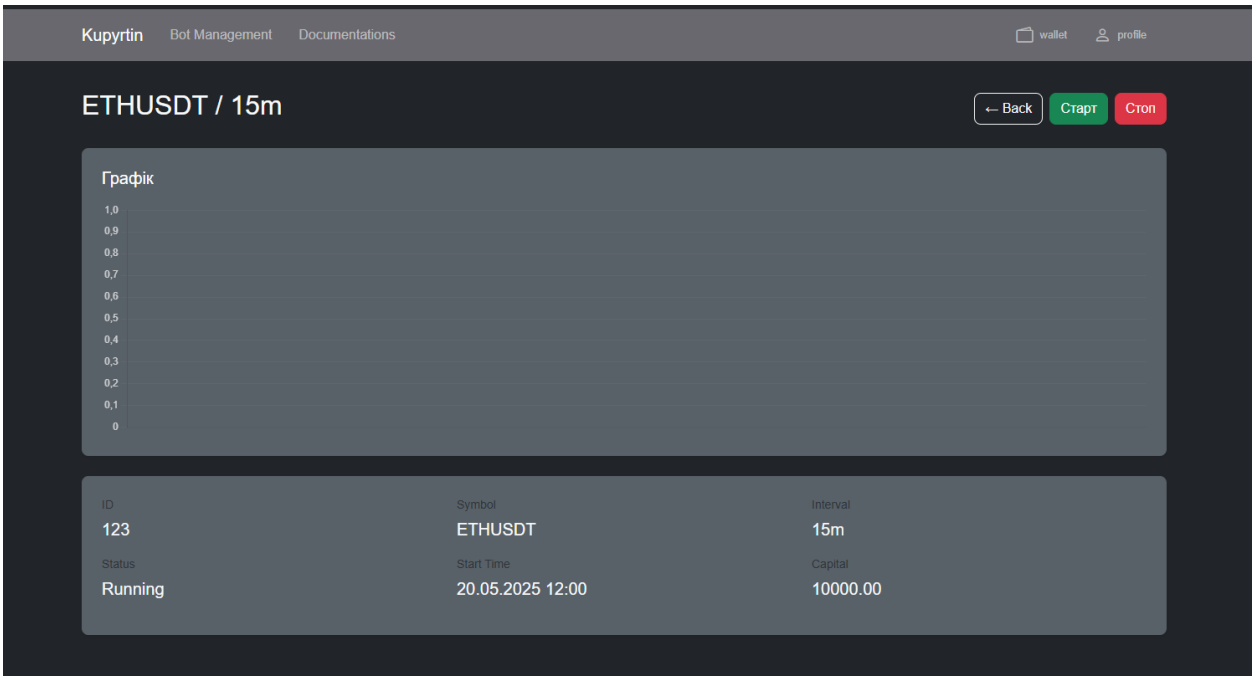

Рис.3.7

3.2 **Результати експериментів і тестування**

У рамках проведеного тестування та експериментів було проведено кілька наступних перевірок, щоб визначити правильність роботи системи.

- Стабільність запуску ботів. Проведено кілька ручних тестів щоб зрозуміти чи правильно працюють боти. Під час тестування було кілька дрібних нюансів які згодом виправив.
- Точність аналізу даних. Також була проведена перевірка з метою визначення того чи правильно аналізуються дані. Після проведення такої перевірки виявилось, що всі дані опрацьовуються коректно.
- Продуктивність оновлення свічок. З цим аспектом є кілька важливих нюансів. Для швидкого отримання даних потрібно використовувати WebSokets а не звичайні Rest Api тому поки нюанс залишиться це не те щоб критично просто можна зробити швидше.

ВИСНОВКИ

Підсумки

Протягом розробки веб-застосунку для торгівлі криптовалютою я здобув перший досвід в розробці застосунку такого типу та розробці та реалізації алгоритмів, що ним керують. Поглибив знання в різних напрямках, але виділю кілька основних.

Java – я потворив, здобуті раніше знання на мові програмування Java. Після чого почив трохи краще розуміти мову та тонкощі її застосування.

Spring framework – я значно поглибив знання, в області розробки веб-застосунків на Spring. Почав добре розуміти як функціонує Security частина фреймворку та загалом почав розуміти деякі глибинні процеси, які відбуваються під капотом. А також почав роздуми над створенням стійкої архітектури для подібних застосунків.

TA4J – здобув перший досвід в роботі з даною бібліотекою, яка дозволяє легко та ефективно аналізувати ринкові дані.

Я поліпшив свої знання в використанні **Json** парсерів, для роботи з API.

Thymeleaf – я черпнув нових знань в розробці шаблонізаторів, та почав краще розуміти Frontend частину (звісно банальний шаблонізатор не охоплює всієї величини фронтенду, але лише складає розуміння як потрібно будувати вміст веб-сторінок.)

PostgreSQL – звісно я дізнався багато нових фічів, які є в цієї бази даних, а також освіжив знання в розробці таблиць та написання SQL-запитів на вибірку та зміну даних в базі.

Отож розробляючи цей застосунок я оптимізував старі та набув нові знання в області розробки аналітичних та алгоритмічних веб-застосунків. Це дозволяє мені поглянути на картину розробки подібного софту під новим для мене кутом та переосмислити багато набутих моїх знання в даній області програмування.

Перспективи розвитку застосунку

Поточна версія веб-застосунку стала чудовою відправною точкою на шляху до створення повноцінного рішення для торгівлі на криптовалютному ринку. Зважаючи на те, що це був мій перший веб-застосунок, який орієнтований на самостійну торгівельну діяльність, я задоволений результатом. Веб-застосунок показує всі базові функції та реалізує початкові стратегії. Але для того, щоб перетворити його на гнучкий та надійний інструмент, я здав низку напрямків, які допоможуть зробити суттєве вдосконалення застосунку.

Архітектура бота.

Поточна логіка бота виконує свої завдання, але вона досить обмежена у плані гнучкості та обробки великих обсягів інформації. В наступній версії хотілося б приділити більше уваги та реорганізувати архітектуру за принципами модульності та розширюваності. Це дасть змогу легко інтегрувати нові функції та логіку без руйнування старих. Підвищити рівень надійності під високі навантаження. Реалізувати незалежні обчислювальні блоки для кращого аналізу даних.

Модернізація бази даних.

Поточна база даних виконує базові функції та її структура не пристосована до масштабування. Отож додавання більш потужних інструментів дуже допомогло б. Впровадження схем для логування та кешування, індексація подій – це все дозволить ефективніше вести журнал змін, логувати дії користувача, а також і мабуть саме головне це інтегрувати систему з іншими інструментами.

Перехід до повноцінного фронтенду

Використання шаблонізатора це добре, але коли потрібно повноцінно працювати з даними, що передаються з веб-сервера то вони тільки будуть обмежувати інтерфейс та динамічність веб-сторінок. Перехід до справжнього фронтенду вирішить ці доволі серйозні питання. Це дозволить реалізувати більш інтерактивний та зручний інтерфейс, підвищити продуктивність клієнту та винести

частину логіки на фронтенд, що розвантажить контролери на сервісній частині веб-застосунку.

Вирок

Поточний стан системи – лише фундамент. Цей проект вже можна вважати MVP(Minimum Viable Product), який показав життєздатність ідеї. Далі потрібно побудувати більш гнучку систему на основі тої яка вже є. Систему, яка дозволить не просто торгувати, а робити це ефективно та автоматизовано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шилдт Г. Java. Повне керівництво. 10-те вид. – Київ: Діалектика, 2018. – 1488 с.

2. Еккель Б. Філософія Java. 4-те повне вид. – Львів: Питер, 2019. – 1168 с.

3. Хорстманн К. Java. Бібліотека професіонала. Том 1. Основи. 11-те вид. – Київ: Вільямс, 2019. – 864 с.

4. Блох Дж. Java. Ефективне програмування. 3-тє вид. – Київ: Діалектика, 2019. – 464 с.

5. Сьєрра К., Бейтс Б. Вивчаємо Java. 2-ге вид. – Київ: Ексмо, 2016. – 720 с.

6. Седжвік Р., Вейн К. Computer Science: основи програмування на Java,

7. ООП, алгоритми та структури даних. – Львів: Питер, 2018. – 1072 с.

8. Гослінг Дж., Джой Б., Стіл Г. Мова програмування Java SE 8. Детальний опис. 5-те вид. – Київ: Вільямс, 2015. – 672 с.

9. Філліпс Б., Стюарт К., Марсікано К. Android. Програмування для професіоналів. 3-тє вид. – Львів: Питер, 2017. – 688 с.

10. Дейтел П., Дейтел Х., Уолд А. Android для розробників. 3-тє вид. – Львів: Питер, 2016. – 512 с.

11. Меднікс З., Дорнін Л., Мік Б., Накамура М. Програмування під Android. 2-ге вид. – Львів: Питер, 2013. – 560 с.

12. Claudi[Електронний ресурс]. – Режим доступу: https://claude.ai/chat/30db652a-d59c-4361-98f5-57e029c645ce – Дата звернення: 01.06.2025.

13. Spring Boot[Електронний ресурс]. – Режим доступу: https://youtu.be/9SGDpanrc8U?si=YoLLcN_Gqcwgyyym – Дата звернення: 01.06.2025.

14. Spring Security[Електронний ресурс]. – Режим доступу: https://youtu.be/If3YIcLZ7sc?si=tWnBA4DkZ3ck_q3u – Дата звернення: 01.06.2025.

15. Spring MVC[Електронний ресурс]. – Режим доступу: https://youtu.be/oIAv6rC7770?si=xdhDLV-3HdF5Mmwp – Дата звернення: 01.06.2025.

16. Spring Data JPA[Електронний ресурс]. – Режим доступу: https://youtu.be/aJulegjqWEw?si=cGrbwSMHMvzgSXBR – Дата звернення: 01.06.2025.

17. Thymeleaf [Електронний ресурс]. – Режим доступу: https://youtu.be/510O21xeelY?si=kxkos8XDwEk8yMy0 – Дата звернення: 01.06.2025.

18. Rest API [Електронний ресурс]. – Режим доступу: https://youtu.be/EaFr0wYaxxM?si=xZ-J5n30hWpFFTet – Дата звернення: 01.06.2025.

19. Rest API та WebSockets [Електронний ресурс]. – Режим доступу: https://youtu.be/XaTwnKLQi4A?si=CgBvso6MYtB-DVgP – Дата звернення: 01.06.2025.

20. Telusko [Електронний ресурс]. – Режим доступу: https://youtube.com/@telusko?si=RlFNbSVEfxCkFFvC – Дата звернення: 01.06.2025.