Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
## Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Машинний алгоритм підтримки прийняття ринкових рішень
(тема)

Виконав:

студент II курсу, групи СПм-22-4

Гайдай Я. А.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Бовчалюк С. Я.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А. А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
**Зав.**
**кафедри** _____
**(підпис)**

**"____" _____ 20__ р.**

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Гайдаю Ярославу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Машинний алгоритм підтримки прийняття ринкових рішень

затверджена наказом по університету від " 01 " квітня 2024 р. № 257 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи 1) технології антиципації ринкових рухів
2) нейронні мережі зворотного поширення
3) інструменти програмної інтеграції з біржами
4) хмарні технології

4. Перелік питань, що потрібно опрацювати у роботі
1) Аналіз актуальності проблеми автоматизації ринкових рішень
2) Аналіз ринкових рухів та впливових факторів
3) Аналіз існуючих рішень антиципації ринкових рухів
4) Огляд сучасних інструментів розробки і технологій для інтеграції з валютними
6) Розробка архітектури програмного алгоритму
7) Розробка веб-інтерфейсу користувача
8) Оформлення документації
9) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз актуальності проблеми автоматизації ринкових рішень | 07.01.24-09.01.24 | |
| 2 | Аналіз ринкових рухів та впливових факторів | 10.01.24-11.01.24 | |
| 3 | Аналіз існуючих рішень антиципації ринкових рухів | 12.01.24-14.01.24 | |
| 4 | Огляд сучасних інструментів розробки і технологій для інтеграції з валютними біржами | 15.01.24-16.01.24 | |
| 5 | Огляд хмарних обчислювальних сервісів | 17.01.24-18.01.24 | |
| 6 | Розробка архітектури програмного алгоритму | 01.08.23-22.04.24 | |
| 7 | Розробка веб-інтерфейсу користувача | 01.08.23-22.04.24 | |
| 8 | Розробка API для адміністрування платформи | 01.08.23-22.04.24 | |
| 9 | Оформлення документації | 06.05.24-10.05.24 | |
| 10 | Подання кваліфікаційної роботи керівникові та її попередній захист | 05.06.24-07.06.24 | |
| 11 | Подання кваліфікаційної роботи на | 08.06.24-12.06.24 | |
| | рецензування | | |

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Бовчалюк С. Я.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 104 с., 22 рис., 6 табл., 2 дод., 32 джерела.

РИНКОВІ РУХИ, КРИПТОВАЛЮТА, ХМАРНІ ОБЧИСЛЕННЯ, ТОРГІВЛЯ, ТОРГОВІ СТРАТЕГІЇ, АЛГОРИТМИ

Метою кваліфікаційної роботи є дослідження сучасних методів і інструментів антиципації ринкових рухів, визначення їхніх переваг і недоліків, та розвиток шляхом розробки програмного алгоритму підтримки ринкових рішень.

У ході виконання кваліфікаційної було розглянуто актуальність потреби завдання розробки алгоритму підтримки ринкових рішень, проведено аналіз технологій передбачення ринкових рухів із впровадженням нейронних моделей зворотного поширення помилки. У роботі наведено огляд сучасних інструментів розробки і технологій для інтеграції з валютними біржами. Виконано розробку архітектури програмного алгоритму та надано опис його технічної реалізації. Наведено деталі технічної реалізації платформи для адміністрування застосунку. Зроблено опис інтерфейсу користувача основного продукту та веб-інтерфейсу, показано процес використання застосунку та веб-сайту.

# ABSTRACT

Master’s thesis: 104 pages, 22 figures, 6 tables, 2 appendices, 32 sources.

MARKET TRENDS, CRYPTOCURRENCY, CLOUD COMPUTING, TRADING, TRADING STRATEGIES, ALGORITHMS

The purpose of qualification work is investigation of current methods and tools for anticipating market trends, identification of their advantages and disadvantages, and development of software algorithm for market decisions support.

During the qualification work made a detailed analysis of the relevance of the need to develop an algorithm for supporting market decisions and provided an analysis of the backpropagation neural model technologies for market trends prediction. Reviewed modern development tools and programmatical technologies for currency exchanges integration. Developed architecture of the software algorithm and provided the description of its technical implementation. Provided details of the technical implementation of the platform for product administration. Compiled description of the user interface of the main product and the web interface. Shown the process of the application and the website setup.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – програмний інтерфейс застосунку (англ., Application Programming Interface)

ARIMA – авторегресійна інтегрована ковзна середня (англ., AutoRegressive Integrated Moving Average)

AWS – веб сервіси Амазон (англ., Amazon Web Services)

BPNN – нейронні мережі зворотного поширення (англ., Back Propagation Neural Network)

CRM – управління взаєминами з клієнтами (англ., Customer Relationship Management)

DTO – об'єкт передачі даних (англ., Data Transfer Object)

ESVM – еволюційна машина опорних векторів (англ., Evolutionary Support Vector Machine)

GA – генетичний алгоритм (англ., Genetic Algorithm)

HTRBF – важка порогова радіальна базисна функція (англ., Hard Thresholding Radial Basis Function)

MACD – рухома ковзна середня розходження (англ., Moving Average Convergence Divergence)

MVP – мінімально життєздатний продукт (англ., Minimum Viable Product)

RDP – відсоток відносної різниці (англ., Relative Difference Percentage)

RSI – індекс відносної сили (англ., Relative Strength Index)

SDK – набір засобів розробки програмного забезпечення (англ., Software Development Kit)

SEO – оптимізація для пошукових систем (англ., Search Engine Optimization)

SVM – машина опорних векторів (англ., Support Vector Machine)

# ВСТУП

Безпека, надійність і швидкість – головний запит сучасного світу, що є майже неможливим поєднанням у реальному житті, оскільки велика швидкість у будь-якому розумінні часто призводить до помилок. Швидкий розвиток та впровадження нових технологій часто випереджає здатність суспільства адаптуватись до них. Якщо раніше для освоєння спеціальностей та певних галузей людина витрачала роки, або навіть десятки років для досягнення професіоналізму, то на сьогоднішній день темп розвитку технологій вимагає значно інтенсивніших підходів до їх вивчення. Невідповідність між інтенсивністю розвитку галузей та спеціалістів щоденно генерує тисячі некомпетентних кадрів, або призводить до професійної деградації існуючих.

Поява криптовалют та розвиток їх технологічних основ, таких як блокчейн надало чималий вплив на економіку та суспільство загалом. Криптовалюти перевернули традиційні підходи до фінансових послуг, посприяли розвитку розвитку нових продуктів і послуг, таких як цифрові платежі, міжнародні перекази, децентралізовані фінансові системи тощо. Вони надали можливість широкому колу людей по усьому світу здійснювати інвестиції у цифрові активи. Навіть ті, хто раніше не був знайомий з традиційними фінансовими ринками, тепер мають можливість за декілька хвилин інвестувати свої гроші у цифровий актив. Це посприяло демократизації інвестицій і розширило доступність до онлайн фінансових послуг. Однак легка доступність та стрімкий розвиток несуть за собою чимало викликів та ризиків, що можуть призвести до фінансових втрат окремих людей, або навіть сприяти кримінальному використанню цих технологій.

Ризики які несе крипто-ринок в основному напряму пов’язані з неправильним використанням та недосвідченістю користувачів. Часом навіть досвідчені трейдери, що освоїли фінансовий та технічний аналіз, або прийшли з традиційних ринків активів, виявляються недостатньо компетентними у

питанні інвестицій на крипто-ринку. Велика кількість новачків у цій сфері обирає на допомогу неефективні інструменти аналізу та антиципації ринкових рухів, і в результаті стикаються зі значними фінансовими втратами.

Потужний ривок в індустрії також зробили технології пов’язані саме з аналізом крипто-ринку. Розробники так само, як і трейдери намагаються встигати за стрімким розвитком, одночасно вивчаючи технічну і економічну складову криптовалют, а також фактори впливу на ціни та об’єми продажів. Слід зауважити, що існує великий попит на інструменти передбачення ринкових трендів. Це в першу чергу пов’язано з бажанням людей отримувати прибуток від трейдингу без значних часових затрат. В свою чергу ринок подібних продуктів на сьогоднішній день не задовольняє усі потреби клієнтів. Більшість інструментів потребує значного вдосконалення та подальшого розвитку.

Впровадження інновацій у методики аналізу ринкових рухів та розробка інструментів підтримки ринкових рішень на сьогоднішній день є актуальною проблемою. Розвиток сучасних технологій дозволяє розробникам швидко адаптуватись під ринкові тенденції та створювати насправді корисні та конкурентоспроможні продукти. Таким чином розробка алгоритму підтримки ринкових рішень є відповіддю на запит суспільства, зацікавленого у безпечних фінансових інвестиціях.

# 1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

## 1.1 Аналіз актуальності потреби автоматизації ринкових рішень

Антиципація (що означає передбачення чи здогад) рухів на фондовому ринку є ключовим викликом для фінансових аналітиків і інвесторів, оскільки успіх в інвестуванні часто залежить від здатності передбачати майбутні тренди та зміни цін на акції. Ця задача виявляється особливо складною через величезну кількість змінних, які впливають на ринкові ціни, включаючи економічні індикатори, фінансові звіти компаній, політичні події, а також непередбачувані глобальні події. Розуміння і прогнозування цих динамічних і взаємопов'язаних процесів є надзвичайно важливим для розробки ефективних стратегій інвестування та управління портфелем, з метою забезпечення стабільного зростання капіталу та мінімізації потенційних ризиків.

Використання лише історичних даних для аналізу тенденцій може бути недостатнім через швидкі зміни в глобальній економіці. Тому експерти шукають інноваційні підходи, щоб покращити точність прогнозів, використовуючи складні алгоритми і новітні технології аналізу даних. Значна увага приділяється розробці та впровадженню моделей машинного навчання та штучного інтелекту, які можуть ефективно обробляти великі масиви інформації, виявляти складні закономірності та надавати більш точні прогнози цін на акції, ніж традиційні методи аналізу.

З огляду на складність і взаємозв'язок між різними факторами, що впливають на фондовий ринок, такими як економічні індикатори, політичні події, громадські настрої та навіть природні катастрофи, інвестори та аналітики використовують розширений арсенал інструментів для аналізу та прогнозування. Це включає комплексні алгоритми, які можуть аналізувати новинні повідомлення в реальному часі, оцінювати громадські настрої в

соціальних мережах та інтернеті, а також використовувати передові статистичні методи для розробки прогностичних моделей [1].

Однак, з появою машинного навчання та глибокого навчання, фокус досліджень перемістився на розробку більш складних і адаптивних моделей, таких як штучні нейронні мережі та мережі зворотного поширення, які здатні автоматично виявляти високо-рівневі абстракції в даних.

## 1.2 Аналіз ринкових рухів та впливових факторів

Аналіз ринкових рухів та впливових факторів включає в себе оцінку і розуміння того, які чинники впливають на ринок та його динаміку. Економічні показники грають ключову роль у розумінні ринкових умов. Це можуть бути дані про ВВП, безробіття, індекси споживчих цін, обсяги виробництва та споживання. Ці показники вказують на стан економіки та можуть впливати на ринкову активність.

Політичні рішення також мають значний вплив на ринок. Законодавчі зміни, рішення уряду та геополітичні події можуть створювати нестабільність та невизначеність на ринку, що може вплинути на ціни та торгівельну активність. Соціальні та культурні чинники можуть включати зміни в споживчих уподобаннях, демографічні тенденції та інші соціокультурні аспекти, які можуть впливати на ринок через зміни в попиті та поведінці споживачів. Психологічні фактори важливі для розуміння психології ринку та поведінки трейдерів. Страх або ейфорія може впливати на ринкову поведінку та створювати або посилювати тренди.

Технічний аналіз включає дослідження цінових графіків та інших технічних показників для прогнозування напрямку цін та ринкових трендів. Це допомагає інвесторам зрозуміти психологію ринку та зробити кращі рішення. Фундаментальний аналіз використовується для оцінки фундаментальної цінності активів. Він включає аналіз фінансових показників компаній, балансів виробництва та попиту на сировину.

У прагненні досягнути вищої точності прогнозування, було розроблено численні методики, зокрема, методи, що базуються на технічному аналізі, такі як індикатор MACD, який допомагає визначати моменти купівлі або продажу акцій за допомогою аналізу різниці між двома ковзними середніми [2]; ARIMA, яка дозволяє моделювати та прогнозувати часові ряди, враховуючи аспекти авторегресії, інтеграції та ковзного середнього; та індекс відносної сили RSI, що допомагає оцінити, чи перекуплена або перепродана певна акція в даний момент [3].

## 1.3 Аналіз існуючих рішень антиципації ринкових рухів

Моделі антиципації ринкових рухів на основі машинного навчання виявилися особливо ефективними у прогнозуванні нелінійних часових рядів, які є характерними для фондового ринку, де цінові патерни та тренди часто змінюються непередбачувано і мають складну структуру.

Серед машинного навчання, метод опорних векторів (SVM) виділяється своєю унікальною здатністю ефективно працювати з великими обсягами та різноманітними типами даних, забезпечуючи при цьому високу точність прогнозування [4]. Метод опорних векторів ефективно використовує принцип мінімізації структурного ризику, на відміну від інших алгоритмів, які зосереджені на мінімізації емпіричного ризику [5]. Цей підхід забезпечує вищу загальну точність і кращу узагальнюючу здатність, особливо при роботі з обмеженими наборами даних. Порівняно з іншими підходами, SVM менш схильний до перенавчання, оскільки алгоритм зосереджений на знаходженні найбільш значущих опорних векторів, що представляють критичні точки даних, ігноруючи менш значущі спостереження.

Використання методу опорних векторів у фінансовому прогнозуванні стало предметом численних досліджень, які підтверджують його ефективність у порівнянні з традиційними стратегіями прийняття рішень на ринку. Дослідники виявили, що SVM здатний забезпечити високу точність у прогнозуванні

цінових рухів, навіть у найбільш непередбачуваних і волатильних ринкових умовах. Ця здатність робить SVM особливо цінним для розробки стратегій інвестування, які можуть адаптуватися до різноманітних ринкових сценаріїв і забезпечити стабільний прибуток навіть у фазах високої невизначеності.

Практичне застосування SVM у фінансовому аналізі та інвестуванні показує, що цей метод може бути ефективно використаний не тільки для прогнозування цін на акції, але й для аналізу ринкових трендів, визначення оптимальних моментів для купівлі або продажу цінних паперів, а також для ризик-менеджменту. Завдяки своїй високій точності та універсальності, SVM знайшов застосування в різних сферах фінансового аналізу, включаючи кредитний скоринг, виявлення шахрайства та оптимізацію портфелів.

У детальному дослідженні, проведеному Шомом Прасад Дасом і Сударсаном Падхі [6], акцент було зроблено на аналізі ефективності машинного навчання в прогнозуванні ринкових цін, де застосовувались два передових методи: метод опорних векторів (SVM рисунок 1.1) та нейронні мережі зі зворотним поширенням помилок (BPNN рисунок 1.2).

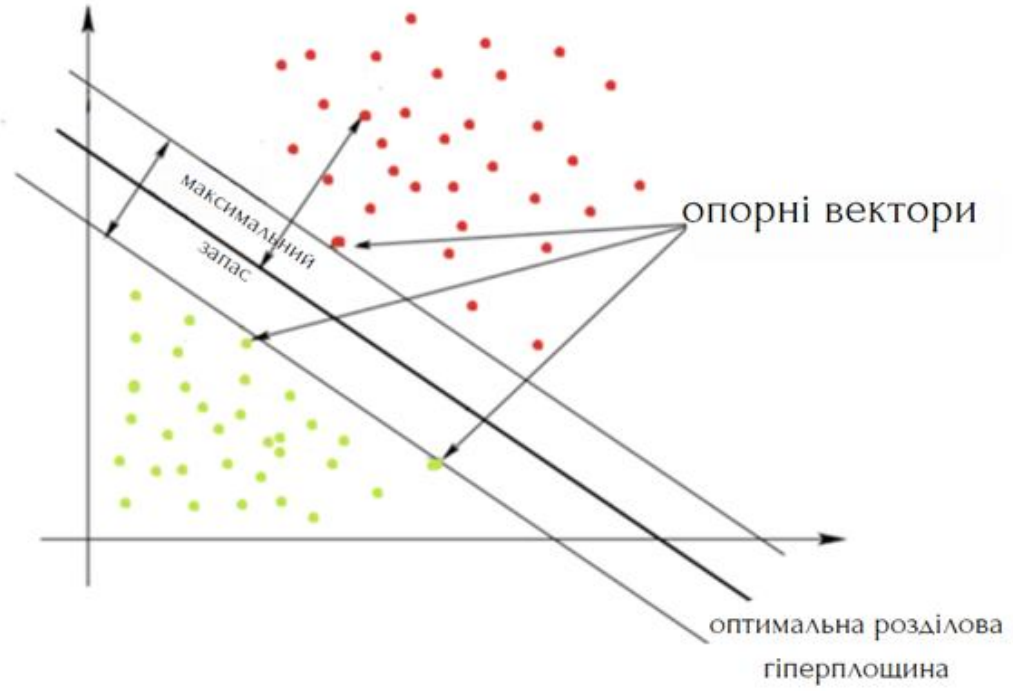


Рисунок 1.1 – Схема роботи методу опорних векторів

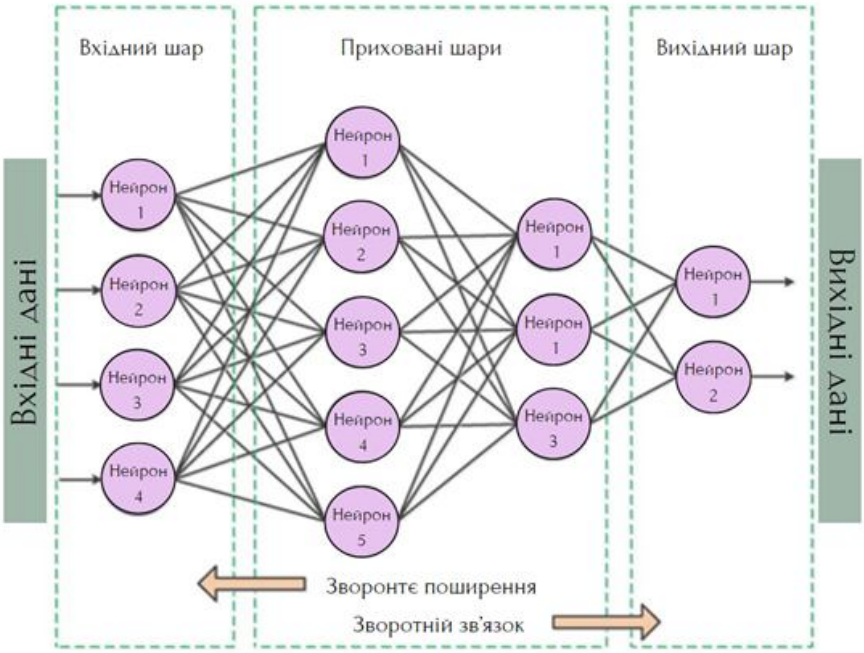


Рисунок 1.2 – Схема роботи нейронної мережі зі зворотним поширенням помилки

Дослідження було спрямоване на розробку професійної прогностичної моделі, яка б могла ефективно аналізувати і передбачати майбутні тренди цін на ринку на основі історичних даних. Для цього було використано великий набір даних, який включав денні ціни відкриття та закриття, а також максимальні та мінімальні ціни, обсяги та вартості угод за період більше ніж сім років. Такий глибокий аналіз історичних даних забезпечує міцну основу для побудови надійних прогностичних моделей.

В ході дослідження було приділено значну увагу визначенню вхідних параметрів для обох моделей. Використання чотирьох значень відносної відсоткової різниці (RDP) у п'ятиденному проміжку дозволило детально аналізувати динаміку змін цін, враховуючи короткострокові коливання. Окрім того, було застосовано експоненційну середню рухому змінну, яка сприяла збереженню інформації про фактичні ціни закриття. Цей підхід дозволяє врахувати важливу інформацію, яка може бути втрачена при простому використанні відсоткових різниць, надаючи моделі більш точне і повне

уявлення про ринкові умови.

Аналіз показав, що точність прогнозування нейронних мереж суттєво зростає, коли в модель вводиться згладжування як залежна змінна. Таким чином, застосування триденної експоненційної середньої рухомої змінної як вихідного значення для відносної відсоткової різниці дозволило підвищити прогностичну здатність моделі, забезпечуючи вищу адаптацію до реальних ринкових умов і змін. Це підтверджує важливість інтеграції гнучких аналітичних інструментів та вдосконалених статистичних методів для підвищення точності і надійності фінансових прогнозів.

Для досягнення максимальної ефективності, Дас і Падхі застосували ретельний підхід до вибору і оптимізації параметрів для кожної з моделей. У випадку з методом опорних векторів SVM, велика увага була приділена вибору та налаштуванню ядерної функції, що є ключовим фактором у визначенні здатності моделі коректно розділяти різні класи даних на основі складних нелінійних взаємозв'язків. Таке налаштування дозволяє SVM ефективно працювати з великим об'ємом даних, забезпечуючи високу точність прогнозування.

З іншого боку, при роботі з нейронними мережами зі зворотним поширенням помилок BPNN, фокус був зроблений на архітектурі мережі та процесі тренування. Адаптація структури мережі, включаючи кількість шарів та нейронів в кожному з них, а також вибір функції активації, є критичною для забезпечення здатності моделі до ефективного навчання та узагальнення на основі вхідних даних. Точне налаштування цих параметрів допомагає уникнути проблеми перенавчання та підвищує здатність моделі до прогнозування майбутніх тенденцій.

Результати дослідження виявили, що обидва методи мають свої переваги та недоліки, проте комбінація SVM і BPNN може пропонувати сильні сторони обох підходів, підвищуючи загальну прогностичну здатність моделі. Це підкреслює важливість гібридних моделей у фінансовому прогнозуванні, де різноманітність підходів може допомогти краще адаптуватися до складності та

непередбачуваності ринкових даних.

Загалом, дослідження Шома Прасада Даса і Сударсана Падхі підкреслює величезний потенціал машинного навчання в області фінансових прогнозів. Воно ілюструє, як через ретельний вибір методологій, оптимізацію моделей та інноваційний підхід до аналізу даних можна значно покращити точність і надійність прогнозів ринкових трендів. Експериментальні результати демонструють, що інтеграція різних технік машинного навчання може допомогти виявити глибші закономірності в даних, які не завжди очевидні при використанні традиційних статистичних методів.

Одним із ключових висновків цього дослідження є те, що успіх у прогнозуванні фінансових ринків не обмежується лише вибором правильного алгоритму, але також залежить від якості та обробки даних, а також від розуміння специфіки ринку. Правильне використання передових алгоритмів, таких як SVM і BPNN, вимагає глибокого аналізу вхідних даних і адаптації моделі під конкретні задачі прогнозування.

В дослідженні також наголошується на важливості використання комплексного підходу до вибору функцій і параметрів моделі. Автори демонструють, як тонке налаштування параметрів може суттєво вплинути на результати, дозволяючи моделям більш точно вловлювати складні залежності між різними ринковими індикаторами. Такий підхід дозволяє не тільки покращити точність прогнозів, але й зрозуміти механізми, що лежать в основі ринкових рухів, що може мати значний вплив на стратегії інвестування.

У підсумку, дослідження Даса і Падхі стає цінним внеском у літературу з фінансового аналізу та прогнозування, нагадуючи про складність і динамічність фінансових ринків та про величезний потенціал машинного навчання у розкритті цих складнощів. Воно також вказує на необхідність глибшого занурення в дані, не лише з точки зору кількості, але й якості, аналізуючи їх через призму різноманітних аналітичних методів і моделей. Підхід, який використовується в цьому дослідженні, підкреслює важливість вибору відповідних методів обробки даних і аналізу, які можуть адекватно відображати

реальні ринкові умови та динаміку, дозволяючи розробляти ефективні стратегії для прогнозування ринкових рухів.

Автори дослідження визнають, що попри значні успіхи в прогнозуванні фінансових ринків завдяки застосуванню SVM та BPNN, існує безліч викликів, зокрема, пов'язаних з волатильністю ринку та змінами у зовнішньому середовищі, які можуть впливати на точність прогнозів. Це вимагає від дослідників бути готовими до неперервного оновлення своїх моделей та методів аналізу, а також до розробки нових підходів, які можуть ефективно адаптуватися до швидко змінюваних умов.

У своєму дослідженні Каратанасопулос та співавтори [1] представили еволюційний метод опорних векторів ESVM [7], як модель для прогнозування поведінки акцій. ESVM (рисунок 1.3) поєднує в собі метод опорних векторів (SVM) та генетичні алгоритми (GA). В той час як SVM зазвичай використовуються для аналізу даних та розпізнавання патернів у завданнях класифікації, генетичний алгоритм використовується в цій моделі для оптимізації параметрів SVM та вибору найбільш значущих ознак. Цей метод спрямований на покращення торгівельної та статистичної ефективності, зменшуючи ризик перенавчання та підвищуючи точність прогнозів.

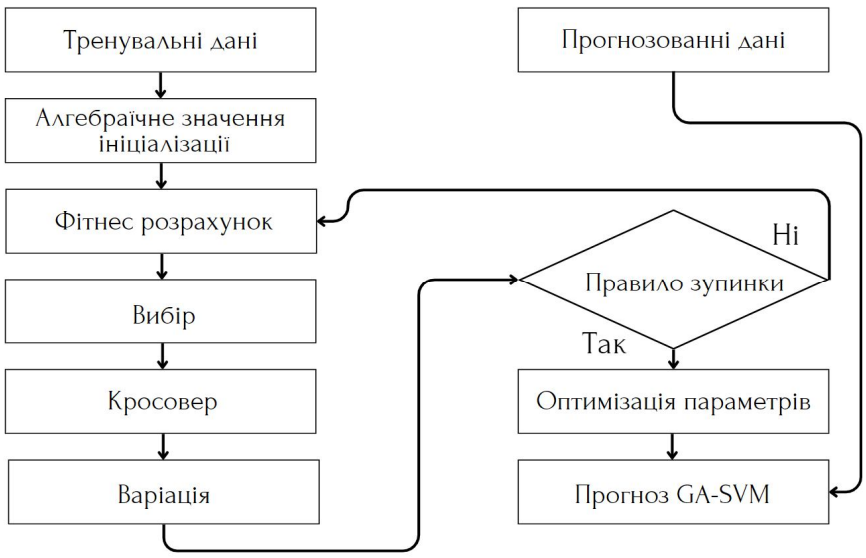


Рисунок 1.3 – Схема роботи еволюційного методу опорних векторів

Застосування ESVM дозволило досягти значних успіхів у прогнозуванні поведінки акцій, завдяки чому модель вирізняється серед інших традиційних методів аналізу ринку. Ефективність ESVM особливо вражаюча у відсотковому вираженні річних доходів, демонструючи високу прибутковість у тестовому середовищі та на реальних ринкових даних. Комбінація SVM та генетичних алгоритмів у ESVM надає унікальну здатність адаптуватися до різних ринкових умов і ефективно виявляти прибуткові торгівельні стратегії.

Особливістю ESVM є її висока адаптивність та здатність до роботи у складних багатовимірних просторах даних, що разом із поліпшеними можливостями інтеграції виходить за рамки традиційних методів прогнозування. Важливим результатом дослідження стало виявлення того, що ESVM перевершує п'ять традиційних методів прогнозування, демонструючи вражаючі результати як у торгівельній, так і в статистичній ефективності. Зокрема, відзначено його надзвичайну торгівельну продуктивність, що виражається через високий відсоток річних доходів, що є значущим досягненням у контексті інвестиційних стратегій.

Це дослідження підкреслює важливість застосування гібридних моделей у фінансовому аналізі та прогнозуванні, де поєднання традиційних методів з інноваційними підходами машинного навчання відкриває нові горизонти для виявлення ринкових можливостей. Подібні моделі, як ESVM, забезпечують не тільки глибше розуміння динаміки ринку, але й дозволяють розробляти більш точні та адаптивні стратегії інвестування. Використання генетичних алгоритмів для оптимізації параметрів та вибору вхідних даних підвищує гнучкість моделі та її спроможність до автономного налаштування відповідно до змінних умов ринку, мінімізуючи тим самим ризик перенавчання та підвищуючи загальну продуктивність інвестиційних стратегій.

Ключовим аспектом успіху ESVM є її здатність ефективно інтегрувати різноманітні джерела даних та аналітичні інструменти в єдину когерентну систему, що дозволяє інвесторам отримати більш повне й глибоке розуміння ринкових тенденцій. Однак, для ефективного застосування таких складних

моделей, як ESVM, інвесторам та аналітикам необхідно мати не лише технічні знання та досвід у галузі машинного навчання, але й глибоке розуміння фінансових ринків. Це включає здатність до критичного аналізу та інтерпретації модельних прогнозів у контексті ширшого ринкового середовища. Важливим є також вміння швидко адаптуватися до змін і навчатися на основі нових даних та ринкових розладнань.

В підсумку, інтеграція еволюційних методів, таких як ESVM, у практику фінансового аналізу та інвестування, відкриває перед фінансовими професіоналами широкі перспективи для підвищення ефективності та досягнення високих інвестиційних результатів. Інвесторам та аналітикам необхідно витримувати крок зі швидким розвитком технологій, водночас залишаючись вірними перевіреним фінансовим принципам та стратегіям. Важливість розуміння того, як машинне навчання та інші передові аналітичні інструменти можуть бути інтегровані в традиційні фінансові моделі, набуває нового значення у світлі потреби оптимізації інвестиційних стратегій для досягнення більшої віддачі та меншого ризику.

Таким чином, дослідження Каратанасопулоса та співавторів стає підтвердженням того, що майбутнє фінансового аналізу та інвестування буде тісно пов'язане з застосуванням комплексних, багаторівневих підходів, що поєднують глибоке розуміння ринку з новітніми технологічними досягненнями. Цей підхід дозволяє не просто адаптуватися до постійно змінних умов ринку, але й випереджати ці зміни, успішно ідентифікуючи та реалізуючи прибуткові інвестиційні можливості. У цьому контексті, постійне самовдосконалення, відкритість до нових ідей та готовність експериментувати стають вирішальними якостями для кожного, хто прагне досягнути успіху в сучасному динамічному світі фінансів.

Дослідження, проведене Росілло Гінером та Де Ла Фуенте [3], підкреслює інноваційний підхід до аналізу ринку акцій, використовуючи метод опорних векторів (SVM) з відносним індексом сили (RSI) та індексом MACD як ключовими індикаторами для аналізу. Цей підхід демонструє гнучкість SVM у

роботі з різними видами фінансових інструментів, адаптуючи RSI для акцій з великим капіталом і MACD для акцій з малим капіталом. Ця спеціалізація індикаторів дозволяє більш точно аналізувати ринкові умови, відповідно до специфіки активів. Використання SVM як класифікатора забезпечує перевагу в кількісному прийнятті рішень, враховуючи його здатність ефективно розрізняти складні шаблони в даних, що є особливо важливим для інвесторів, які прагнуть оптимізувати свої інвестиційні стратегії.

Протягом тестування на періодах у 200, 250, 300 та 500 днів було виявлено, що період в 250 днів пропонує найкращий баланс між тривалістю аналізу та точністю прогнозу, що підкреслює важливість вибору оптимального часового проміжку для аналізу. Це відкриває нові можливості для інвесторів у плануванні стратегій купівлі та продажу, забезпечуючи краще розуміння ринкових тенденцій та поведінки акцій на короткостроковому горизонті. SVM категоризація даних на класи купівлі та продажу дозволяє інвесторам мати чітке розуміння потенційних точок входу та виходу з ринку, забезпечуючи їм стратегічну перевагу.

Використання функції ядра важкої радіальної базисної (HTRBF) у SVM забезпечує високу точність у розрізненні класів, підкреслюючи спроможність методу оптимально адаптуватися до складності та непередбачуваності ринкових даних. Це дозволяє інвесторам використовувати складні аналітичні моделі для ефективного управління портфелем, мінімізуючи ризики та максимізуючи потенційний прибуток.

Представлені результати у таблиці 1.1 і таблиці 1.2 демонструють порівняльні значення отримані під час симуляцій на бичачому і ведмежому ринках, а результати таблиці 1.3 і таблиці 1.4, показують значення отримані під час симуляцій на моделі з високою і низькою волатильністю відповідно. У таблиці порівнюються метод опорних векторів (SVM) з методом купівлі і утримання (BH) та зі наївною стратегією (N). Метод купівлі і утримання полягає у придбанні інвестицій і зберіганні їх протягом за-даного часу, у цьому дослідженні мінімальний період утримання складав 5 днів. Наївна стратегія в

свою чергу приймає зміни за останній ринковий період, як основу для подальшого прогнозування. Результати представлені у вигляді умовних одиниць, які демонструють продуктивність прийнятих ринкових рішень, та не мають валютного еквіваленту.

Таблиця 1.1 – Результат симуляції на бичачому ринку

| | SVM | BH | N |
|---|---|---|---|
| Симуляція_1 | 337.0 | 800.1 | -425.5 |
| Симуляція_2 | -303.3 | 1147.3 | -592.5 |
| Симуляція_3 | 148.8 | 377.1 | -193.8 |
| Середнє значення | 60.8 | 774.8 | -403.9 |

Таблиця 1.2 – Результат симуляції на ведмежому ринку

| | SVM | BH | N |
|---|---|---|---|
| Симуляція_1 | 153.1 | -17.0 | -425.5 |
| Симуляція_2 | 104.8 | -389.9 | -592.5 |
| Симуляція_3 | -9.0 | 377.1 | -193.8 |
| Середнє значення | 83.0 | -216.1 | -187.6 |

Таблиця 1.3 – Результат симуляції з високою волатильністю

| | SVM | BH | N |
|---|---|---|---|
| Симуляція_1 | -284.8 | 655.8 | 297.0 |
| Симуляція_2 | 2599.4 | 22.8 | 789.5 |
| Симуляція_3 | -288.9 | -755.5 | -458.1 |
| Середнє значення | 867.8 | -25.6 | 209.5 |

Таблиця 1.4 – Результат симуляції з низькою волатильністю

| | **SVM** | **BH** | **N** |
|---|---|---|---|
| Симуляція_1 | 145.2 | -43.9 | -84.4 |
| Симуляція_2 | 9.1 | -181.8 | -3.8 |
| Симуляція_3 | -111.8 | 143.1 | 33.8 |
| Середнє значення | 14.2 | -27.5 | -18.2 |

Порівняльний аналіз з базовими стратегіями торгівлі показав, що метод опорних векторів демонструє кращий результат на моделях ринку як з високою волатильністю, так і низькою. Також, застосування методу опорних векторів на моделі ведмежого ринку продемонструвало єдиний позитивний результат серед інших стратегій, що є безумовною перевагою даного алгоритму. Однак, на бичачому ринку стратегія купівлі і утримання показала найкращий результат, що є закономірною ситуацією, з урахуванням специфікації та призначення цього методу аналізу ринку. Під час дослідження метод опорних векторів продемонстрував загальну позитивну продуктивність на кожній ринковій мотелі. Було доведено перевагу SVM та його здатність знизити максимальні збитки і річне стандартне відхилення.

Метод опорних векторів має високу ефективність у порівнянні зі стандартними стратегіями прийняття ринкових рішень. Використання цього методу рекомендовано на ринках з високою волатильністю та ринках зі спадаючим трендом. Використання SVM є найкращим вибором стратегії під час фінансової нестабільності та може зберегти інвестиції під час тривалих ведмежих трендів, або навіть зафіксувати прибутки під час кризових ситуацій. Однак метод не є достатньо чутливим до маленьких цінових змін, що робить його не рекомендованим до використання на ринках з низькою волатильністю.

Для покращення роботи алгоритму рекомендується використовувати індикатор тренду, як одне із значень вхідних даних, для вчасного визначення переходу ринку у стан високої або низької волатильності.

Об’єднання методу опорних векторів з експертною системою оцінки предметної області, гарантовано знизить ризики втрати капіталу під час

ринкових операцій [8]. У ролі експертної системи можуть виступати уже існуючі рішення, такі як Neural Forex Network, Algo Trader тощо. Більшість таких систем також базується на роботі штучного інтелекту. Поєднання підвищить надійність алгоритму шляхом надання вхідних значень відповідних до специфічної предметної області. Серед цих значень можуть бути такі, що впливають на волатильність ринку, наприклад політичні події, економічні фактори, рішення центрального банку тощо. Алгоритм може використовувати ці дані для прийняття більш обґрунтованих рішень та оптимізації торгівельних стратегій, що в свою чергу покращить ефективність прогнозування.

Для підвищення ефективності навчання нейронної моделі рекомендовано використовувати відносний індекс сили та значення збігу і розходження на входах для SVM. Також продуктивність нейронної мережі можливо підвищити використовуючи залежну змінну для згладжування, наприклад експоненційну рухому змінну.

## 1.4 Постановка завдання дослідження

Аналіз рішень антиципації ринкових рухів показав, що розвиток сучасних технологій надихнув трейдерів та розробників, які цікавляться економікою та її механікою, на впровадження нових підходів в індустрію ринкової торгівлі. Хвиля розвитку штучного інтелекту захопила чимало індустрій, зокрема пов’язаних з аналітикою великих об’ємів даних. Сфера економіки та криптовалютної торгівлі не стала виключенням. Швидкого поширення набуває розробка складних і адаптивних моделей, таких як штучні нейронні мережі та мережі зворотного поширення призначених для вирішення проблеми підтримки ринкових рішень. Ці моделі виявилися особливо ефективними у прогнозуванні нелінійних часових рядів, які є характерними для фондового ринку, де цінові патерни та тренди часто змінюються непередбачувано і мають складну структуру.

Однак штучний інтелект потребує великої кількості даних для

ефективного навчання. Неякісні дані та їхня неповнота або неточність можуть призвести до помилкових прогнозів і втрат. Чималою проблемою є недостатня адаптивність під динамічні ринкові умови, які піддаються не тільки технічному та економічному аналізу, а й залежать від психо-емоційного стану суспільства. Нестабільні політичні, економічні або технічні події можуть впливати на криптовалютний ринок, і штучний інтелект може не завжди враховувати ці фактори або реагувати на них ефективно.

Для найбільш безпечної стратегії передбачення ринкових рухів, необхідно використовувати штучний інтелект як допоміжний інструмент, а не як повноцінне рішення. Принаймні на перших етапах розвитку нейронних моделей. Такий підхід наддасть трейдерам гнучкіші можливості та більшу впевненість під час управління власними активами.

Враховуючи вищевказане, метою кваліфікаційної роботи є дослідження сучасних методів і інструментів антиципації ринкових рухів, визначення їхніх переваг і недоліків, та розвиток шляхом розробки програмного алгоритму підтримки ринкових рішень.

Відповідно до поставленої мети необхідно розв’язати наступні часткові задачі:

- провести аналіз існуючих алгоритмів антиципації ринкових рухів;
- виконати огляд сучасних інструментів розробки і технологій для інтеграціх з валютними біржами;
- спроектувати та розробити конкурентоспроможний продукт, що передбачає алгоритм підтримки ринкових рішень;
- розробити інструкцію користування інтерфейсом продукту.

# 2 ОГЛЯД СУЧАСНИХ ІНСТРУМЕНТІВ РОЗРОБКИ

## 2.1 Огляд сучасних інструментів розробки і технологій для інтеграції з валютними біржами

Сучасній індустрії розробки програмного забезпечення не характерні такі проблеми, як дефіцит інструментів чи середовищ програмування, аналізу та розгортання рішень. Інструментарій сучасного розробника дозволяє самостійно імплементувати масштабні проекти та підтримувати їх, витрачаючи якомога менше зусиль та часу. Так само можна визначити безліч інструментів для координації та полегшення командної розробки, які є одним з найважливіших аспектів сучасних комерційних проектів, оскільки мають на меті паралелізацію та прискорення розвитку проекту від планування до релізу.

Великий вибір інструментів розробки може збентежити менеджерів та розробників на початковій стадії планування, оскільки вибір правильного інструментарію безпосередньо впливає на швидкість та якість розробки. Більш того невірний вибір технологій та сервісів може призвести до додаткових витрат, як під час розробки програмного забезпечення так і на етапі продуктової стадії (production). Кожен окремий сервіс створений для поліпшення та оптимізації процесів, як менеджменту так і програмування, що у свою чергу знижує вірогідність помилок та ризиків пов’язаними з ними, а отже має свою ціну за надані послуги та інструменти.

Дуже важливо при плануванні проекту визначити основні компоненти майбутнього продукту, критичні модулі, зону відповідальності кожного члена команди, або окремих команд розробників. На основі відокремлених модулів необхідно обдумати потенційні ризики та підібрати специфічні інструменти для вирішення цих проблем. Важливо провести детальний аналіз у сегменті кожного інструментарію, та дослідити особливості, які пропонує той чи інший сервіс. На основі аналізу ринку технологій та виділених критичних секцій, вже

набагато легше робити вибір інструментів та розраховувати бюджет необхідний для розробки та релізу продукту.

Почнемо з планування майбутнього проекту та виділення окремих модулів для підбору інструментів та сервісів необхідних на відповідних етапах розробки. Майже кожне сучасне програмне забезпечення, яке працює через мережу Інтернет складається з трьох частин – інтерфейс користувача, серверна частина, або головна логіка застосунку, та сховище даних. Ці ж компоненти притаманні і нашій розробці, за винятком декількох додаткових модулів, що будуть розглянуті окремо. У даному аналізі буде наведений огляд інструментів необхідних саме для розробки та розгортання проектів, інструменти для тестування, аналізу та менеджменту наведені не будуть.

Перший і найважливіший для користувача модуль – це інтерфейс користувача, обличчя програмного застосунку та головний посередник у спілкуванні між клієнтом та машинним кодом. Інтерфейс користувача – це та частина програмного забезпечення або пристрою, яка дозволяє взаємодіяти користувачеві з системою. Це включає в себе всі елементи, які користувач може бачити, натискати або взаємодіяти з ними, такі як кнопки, меню, поля для введення даних, вікна та інші елементи керування. Інтерфейс користувача відіграє критичну роль у забезпеченні зручності та ефективності користування програмою чи пристроєм, і відповідальний за перетворення складних технічних можливостей у доступні та зрозумілі операції для кінцевого користувача.

У нашому випадку, як і у більшості, інтерфейс користувача має бути простим, тобто зрозумілим для людини, і зручним у користуванні. Також серед ключових вимог є легкість доступу до продукту. Необхідно визначити цільову аудиторію і найпоширеніший спосіб користування подібними сервісами серед цієї аудиторії. Вибір зазвичай полягає між мобільним застосунком, де користувач має завантажити програму собі на мобільний девайс, веб-інтерфейсом (вебсайтом), де клієнт зможе використовувати браузер для доступу до продукту, та десктоп застосунок, тобто програма призначена для використання за допомогою комп’ютера. Також останнім часом набув

поширення четвертий метод комунікації користувачів з сервером – боти у соціальних мережах та месенджерах, що собою являє перевтілення командних інтерфейсів, але у більш зрозумілій для користувача формі з допоміжними візуальними елементами, шаблонами та інструментами для покращення досвіду використання. Додатково розробники подібних платформ пропонують зручні API для побудови логіки та командного інтерфейсу, що в свою чергу полегшує процес розробки.

Для охвату більшої аудиторії зазвичай розробляється одразу декілька інтерфейсів користувача під різні платформи та варіативні групи людей, які звикли користуватись різними пристроями та операційними системами. У нашому випадку така опція майже недоступна, вочевидь через брак бюджету необхідного на фінансування розробки та підтримки мультиплатформенного рішення. Натомість розробка базується на створенні MVP Product, для спрощення усього процесу і зниження кількості ресурсів необхідних для досягнення мети. Мінімально життєздатний продукт (MVP) – це стратегія розробки програмного забезпечення, яка полягає у створенні продукту з мінімальним набором функціоналу, необхідним для того, щоб задовольнити потреби перших користувачів чи вирішити основну проблему. MVP дозволяє швидко випустити продукт на ринок з мінімальними витратами і часом, отримати зворотний зв'язок від користувачів і використовувати його для подальшого вдосконалення та розвитку. Цей підхід допомагає зменшити ризики та витрати на розробку, а також дозволяє ефективно визначити напрямок розвитку продукту, зосереджуючись на його основному функціоналі та важливих потребах користувачів. Отже з огляду на вищевказані фактори, фокус зміщається у бік більш простих для розробки рішень, і в свою чергу доступних для потенційних користувачів. Інтерфейс інтегрований у соціальну мережу, або месенджер якомога найкраще підходить під визначені вимоги, оскільки є легким у розробці і доступний для усіх користувачів обраної мережі, а також може повністю задовольнити потреби у користуванні ранніми версіями продукту.

Оскільки чат-боти набули великої популярності і зайняли відповідне місце у сфері бізнесу та некомерційному використанні, більшість популярних соціальних платформ має власні інструменти для розробки і інтеграції чат-ботів. Також свою популярність вони отримали і серед розробників, відрізняючись складністю інтерфейсів і запропонованими можливостями. Розглянемо найпопулярніші платформи для чат-ботів на сьогодні:

- WhatsApp Business API;
- Viber Chatbot API;
- Facebook Messenger Platform;
- Telegram Bot API.

Наразі WhatsApp надає можливість розробки чат-ботів через свій WhatsApp Business API. Для створення чат-ботів на WhatsApp необхідно зареєструвати бізнес-акаунт на WhatsApp Business API та отримати доступ до його можливостей. Для цього потрібно зв'язатися з партнером WhatsApp або пройти процес реєстрації на офіційному веб-сайті WhatsApp Business API. Чат-боти на WhatsApp можуть обробляти текстові повідомлення від користувачів, розуміти їх запити та взаємодіяти з ними природним мовою. Платформа надає можливість налаштувати автоматичні відповіді на певні запити або запити, що відповідають певним шаблонам. Це може бути корисно для надання базової інформації або підтримки клієнтів. Також можливо інтегрувати чат-бота на WhatsApp з іншими системами, такими як CRM, e-commerce платформи чи інші зовнішні сервіси, щоб забезпечити більш широкий функціонал. Вартість розробки чат-бота на платформі WhatsApp може залежати від кількості функціоналу та складності проекту. WhatsApp Business API підтримує декілька мов програмування, для яких має відповідні SDK:

- Node.js;
- Python;
- Java.

Viber також є можливість розробки чат-ботів через їхню платформу Viber Chatbot API, яка дозволяє розробникам створювати чат-ботів для Viber.

Розробник можете зареєструвати свого бота на Viber і отримати доступ до необхідних інструментів для розробки та управління ботом. Чат-боти на Viber можуть обробляти текстові повідомлення, зображення, відео та інші медіафайли від користувачів. Також є можливість налаштувати автоматичні відповіді на певні запити користувачів або певні події. Це може бути корисно для надання інформації про продукти, послуги або відповіді на загальні запитання. Viber має декілька SDK для таких мов програмування як Python та Node.js. Вартість використання Viber Chatbot API також може варіюватись в залежності від необхідного функціоналу, але має місячну комісію на підтримку чат-боту у розмірі 100 євро. Нижче наведено вартість кожного ініційованого повідомлення у чат-боті в залежності від регіону (таблиця 2.1).

Таблиця 2.1 – Вартість ініційованого повідомлення до Viber чат-боту в залежності від регіону

| Country | Region | Price |
|---|---|---|
| Sri Lanka | SEA | €0.0023 |
| Switzerland | WE | €0.0173 |
| Ukraine | CEE | €0.0196 |
| Vietnam | SEA | €0.0104 |
| Rest of the world | ROW | €0.0137 |

Facebook Messenger Platform – це набір інструментів і сервісів, які дозволяють розробникам створювати чат-ботів та інтегрувати їх з Facebook Messenger. Facebook пропонує використовувати Messenger API – основний інтерфейс програмування застосунків, який дозволяє розробникам взаємодіяти з Messenger. Цей API можна використовувати для створення чат-ботіву тому числі, а також обробки повідомлень від користувачів, відправлення повідомлень та багато іншого. Також серед інструментів на платформі Facebook можна визначити Messenger Plugins – набір віджетів та інструментів, які можна вбудувати на веб-сайти або в мобільні додатки. Наприклад, кнопки "Надіслати

повідомлення" або "Поділитися" дозволяють користувачам легко почати чат з вашим ботом. Messenger Codes і Links – унікальні QR-коди та посилання, які користувачі можуть сканувати або клікнути, щоб відкрити чат з ботом у Messenger. Додатково пропонується функція AI та інтеграція з іншими сервісами: розробник може використовувати штучний інтелект, щоб забезпечити більш розумні та контекстуальні відповіді від бота. Щоб розпочати роботу з Messenger Platform, вам потрібно зареєструвати свого бота на платформі Facebook Developers та розпочати розробку відповідно до їхньої документації та рекомендацій.

Telegram Bot API – це інтерфейс програмування застосунків [9], який надається Telegram для розробників, щоб створювати свої власні чат-боти та інтегрувати їх з платформою Telegram. Розробник може реєструвати свого бота на Telegram і обробляти повідомлення від користувачів. Це можуть бути текстові повідомлення, зображення, відео, аудіозаписи та інші медіафайли. Бот може надсилати відповіді користувачам, які можуть бути текстовими, медіафайлами або навіть структурованими повідомленнями з кнопками. Розробник може програмувати бота так, щоб він розумів певні команди, вказані користувачами, і виконував відповідні дії. Telegram Bot API підтримує створення кнопок, що дозволяє створювати інтерактивні інтерфейси для взаємодії з користувачами. Telegram має вбудовану підтримку шифрування для захисту конфіденційності повідомлень та даних користувачів. Telegram постійно оновлює свої API та надає нові функції для розробників. Створення та розгортання чат-бота на платформі Telegram може бути досить простим завданням, особливо завдяки зручному API та документації, яка доступна на сайті Telegram для розробників.

Telegram має у своїй документації перелік багатьох бібліотек, що доступні для безпечної інтеграції з API і пропонують додаткові можливості взаємодії. На сьогоднішній день такі бібліотеки створені на багатьох мовах програмування:

- PHP;

- Go;
- Python;
- Rust;
- .NET;
- Kotlin;
- C++;
- Ruby.

Головною особливістю розробки під платформу Telegram є абсолютна безкоштовність використання API.

Важливим аспектом розроблюваного рішення є відокремлення модулю реєстрації та авторизації користувачів від модулю доступу до самого продукту. Тож беручи це до уваги, можна зробити висновок, що фінальний продукт буде складатися з двох окремих застосунків, що не мають прямого зв’язку один з одним. Авторизаційний застосунок може виконувати додаткові функції, такі як адміністрування користувачів, керування їхнім доступом до продукту, а також надання основної інформації про сам продукт. Таке рішення потребує більш складного інтерфейсу ніж інтерфейс для використання продукту, оскільки дозволяє управління великим об'ємом даних, їх перегляд та редагування. Для застосунку такого масштабу більш підійде платформа загального доступу, яку можна буде використовувати за допомогою різних девайсів, як статичних так і портативних. Звичайно мова йде про веб-інтерфейс, що доступний для використання за допомогою веб-браузера. Отже задача полягає у виборі інструментів для розробки сучасного і багатофункціонального сайту. Абстрагуючись від використання чистого HTML та CSS, розглянемо найсучасніші інструменти та фремворки, призначені для frontend розробки, серед яких можна виділити наступні: React.js, Angular, Flutter.

React.js – це бібліотека JavaScript, яка використовується для створення інтерактивних користувацьких інтерфейсів. Вона була розроблена компанією Facebook і випущена в 2013 році. Одним із головних розробників React.js є Facebook, а також активна спільнота розробників, яка постійно вносить внески

в розвиток та підтримку цієї бібліотеки. React.js є однією з найпопулярніших технологій для розробки фронтенду у сучасному веб-розробництві. Вона використовується компаніями різного масштабу, від стартапів до великих корпорацій, для створення веб-додатків та інтерфейсів. Деякі з найвідоміших компаній, що використовують React.js, включають Facebook, Instagram, Netflix, Airbnb, Dropbox та багато інших. React має безліч корисних інструментів для розробки веб-сайтів, найкорисніші включають:

- React Developer Tools. Розширення для браузера, яке дозволяє розробникам візуально інспектувати та відлагоджувати React компоненти на стороні клієнта;
- Redux. Бібліотека управління станом, яка широко використовується в React-додатках для керування складними даними та станом додатку;
- Material-UI. Бібліотека компонентів інтерфейсу користувача на основі Material Design, яка спрощує створення красивих та функціональних компонентів в React-додатках;
- Styled-components. Бібліотека для стилізації компонентів за допомогою CSS в JS підходу, що дозволяє забезпечити розділення стилів на рівні компонентів.

Angular – повноцінний фреймворк для розробки веб-додатків, розроблений і підтримуваний компанією Google. Angular розробляється і підтримується командою інженерів від Google, а також активною спільнотою розробників. Він був випущений в 2016 році як переробка попереднього AngularJS, і з тих пір став одним із найпопулярніших фреймворків для веб-розробки. Angular постійно оновлюється і покращується. Кожні шість місяців випускається нова версія фреймворку, що включає в себе нові функції, покращення та виправлення помилок. Angular залишається актуальним і конкурентоспроможним у сфері веб-розробки. Angular використовується компаніями різних розмірів та галузей для розробки веб-додатків. Від стартапів до великих корпорацій, Angular використовується для створення різноманітних проектів, включаючи управління контентом, адміністративні панелі,

електронну комерцію та багато іншого. Серед особливих інструментів фреймворку можна виділити наступні:

- Angular Universal. Це платформа для серверного рендерингу Angular додатків, яка дозволяє покращити SEO та швидкодію веб-застосунків;

- NgRx. Це бібліотека управління станом, яка дозволяє розробникам ефективно керувати станом додатку та його даними;

- Angular Material. Це набір готових до використання компонентів інтерфейсу користувача на основі Material Design, що спрощує створення стильних та функціональних інтерфейсів;

- RxJS. Це бібліотека для роботи з асинхронним програмуванням в Angular додатках, що дозволяє керувати потоками даних та подій.

Flutter – це відкритий фреймворк [10] розробки користувацьких інтерфейсів (UI). Flutter розробляється та підтримується Google. Він був випущений у вересні 2017 року і з тих пір став одним з інструментів, для розробки мобільних та веб-додатків, що найшвидше розвиваються. Flutter є актуальним і популярним фреймворком, який швидко набирає популярність серед розробників. Він має широку підтримку з боку Google та активну спільноту розробників, які постійно вносять внески у розвиток та підтримку цього фреймворка. Flutter використовується для розробки мобільних додатків для платформ Android та iOS, а також веб-додатків та десктопних застосунків. Він є відмінним вибором для розробників, які хочуть створювати крос-платформені додатки з високоякісним інтерфейсом та продуктивністю. Flutter використовує мову програмування Dart для написання додатків. Dart – це модерна, ефективна мова програмування, яка дозволяє створювати швидкі та надійні додатки. Flutter має наступні інструменти для полегшення процесу розробки та аналізу кодової бази:

- Flutter DevTools. Це набір інструментів для розробників Flutter, який надає розширені можливості для відлагодження, аналізу та профілювання додатків;

- Flutter Inspector. Інструмент для відлагодження та аналізу Flutter

додатків, який дозволяє вам вивчати структуру та поведінку вашого додатка в реальному часі;

- Flutter Packages. Flutter має велику екосистему пакетів, які розширюють його функціональність та дозволяють швидше створювати додатки. Ви можете використовувати ці пакети для додавання нових функцій та взаємодії зі сторонніми сервісами.

Наступним модулем програмної розробки, що потребує аналізу, є серверна частина застосунку, а отже найголовнішим, оскільки містить усю логіку, алгоритми і є ядром усього рішення. У нашому випадку так само, як і з користувацьким інтерфейсом, цей модуль розділяється на дві частини: рішення що містить логіку авторизаційного застосунку, та рішення що містить логіку головного продукту. Обидва рішення потребують зручні інструменти backend розробки та підтримку масштабування проектів за використання мінімуму ресурсів. Найпопулярнішими фоеймворками для створення API на сьогодні є:

- Django;
- Express.js;
- Flask;
- Laravel;
- Ruby on Rails;
- Spring;
- ASP.NET.

Джанго – це високорівневий веб-фреймворк, розроблений на мові програмування Python. Основною його метою є полегшення та прискорення процесу розробки веб-додатків шляхом надання розробникам готових інструментів і шаблонів. Однією з ключових особливостей Django є його вбудований адміністративний інтерфейс, який дозволяє розробникам легко створювати та управляти базами даних та моделями. Також Django має широкі можливості управління користувачами та аутентифікацією, що робить його популярним вибором для розробки веб-додатків, які потребують системи управління користувачами. Фреймворк також надає потужні засоби для роботи

з URL-адресами та маршрутизацією, що спрощує реалізацію складних веб-додатків з різними сторінками та маршрутами. Що стосується безпеки, Django має вбудовані механізми захисту від різних типів атак, таких як SQL-ін'єкції, міжсайтовий скриптінг та інші. Це дозволяє розробникам створювати безпечні веб-додатки без необхідності вручну реалізовувати захист. Django має велику та активну спільноту розробників, яка надає підтримку, допомогу та розвиток фреймворку. Велика кількість розширень та сторонніх бібліотек також сприяє розширенню функціональності Django та розробці високоякісних веб-додатків.

Express.js є легким та гнучким веб-фреймворком для Node.js, який дозволяє розробникам швидко створювати веб-додатки та API. Він базується на JavaScript, що робить його привабливим вибором для багатьох розробників, оскільки JavaScript є однією з найпопулярніших мов програмування. Основні особливості Express.js включають маршрутизацію, middleware для обробки запитів, підтримку різних двигунів шаблонів для генерації HTML-коду, зручне обслуговування статичних файлів, розширення функціональності за допомогою сторонніх пакетів та активну спільноту розробників. Express.js дозволяє швидко створювати високоякісні веб-додатки та API, забезпечуючи при цьому простоту використання та широкі можливості розширення функціональності.

Flask – легкий та гнучкий мікрофреймворк для розробки веб-додатків на мові програмування Python. Він надає простий синтаксис та мінімальний набір інструментів, що дозволяє швидко створювати веб-додатки та API без зайвого навантаження. Flask не накладає обмежень на структуру додатку, тому розробники можуть використовувати лише необхідні компоненти та бібліотеки. Основні особливості фреймворку включають простоту використання, зручну маршрутизацію для обробки HTTP-запитів, підтримку шаблонів Jinja2 для генерації HTML-коду, розширення для додаткової функціональності та активну спільноту розробників, яка надає підтримку та допомогу в розвитку. Flask є ідеальним вибором для початківців у веб-розробці та для швидкої розробки простих веб-додатків на Python.

Laravel – це високорівневий фреймворк для розробки веб-додатків на мові

програмування PHP. Він відомий своєю простотою використання та широким набором функцій, що дозволяють розробникам швидко створювати потужні та масштабовані додатки. Фреймворк пропонує конвенції над конфігураціями, що дозволяє розробникам швидко створювати додатки за стандартами MVC (Model-View-Controller). Він має вбудовану підтримку для роботи з базами даних, автентифікації, маршрутизації, сесій та інших типових завдань. Основні особливості Laravel включають в себе шаблонний механізм Blade, який дозволяє створювати красивий та ефективний HTML-код, а також міграції баз даних, що спрощують управління схемою бази даних та версіонування. Laravel також має вбудований механізм маршрутизації, який дозволяє легко визначати URL-адреси та обробляти HTTP-запити. Що стосується збагачення функціоналу та інструментів, Laravel має широкий вибір сторонніх пакетів та розширень, які розширюють його функціональність. Він також має велику та активну спільноту розробників, яка надає підтримку та допомогу в розвитку. Laravel є ідеальним вибором для розробки веб-додатків на PHP завдяки своїй простоті використання та багатому функціоналу.

Ruby on Rails – це високорівневий веб-фреймворк для швидкої розробки веб-додатків на мові програмування Ruby. Rails пропонує концепцію "Конвенція перед конфігурацією", що означає, що він має стандартизований спосіб організації коду та структуру проекту, що дозволяє розробникам швидко розібратися в проекті та працювати над ним. Він використовує архітектурний шаблон MVC (Model-View-Controller), що дозволяє розділити бізнес-логіку, представлення та обробку запитів. Основні особливості Ruby on Rails включають в себе широкий набір вбудованих функцій, таких як генерація коду, міграції баз даних, автоматична маршрутизація, шаблонний механізм для представлення даних та багато іншого. Що стосується спільноти, Ruby on Rails має велику та активну групу розробників, яка надає підтримку, допомогу та документацію. Rails є популярним фреймворком у спільноті Ruby, і він широко використовується для створення різноманітних веб-застосунків, від простих блогів до складних веб-платформ.

Spring – це потужний фреймворк для розробки Java-застосунків. Він дозволяє розробникам швидко створювати різноманітні застосунки, від простих веб-додатків до складних корпоративних систем. Фреймворк базується на інверсії керування та контейнерізації об'єктів, що дозволяє розробникам зосередитися на бізнес-логіці, а не на деталях інфраструктури. Основні особливості Spring включають в себе підтримку різних модулів, таких як Spring MVC для розробки веб-застосунків, Spring Boot для автоматичного налаштування застосунків, Spring Data для роботи з базами даних та Spring Security для забезпечення безпеки. Spring також має вбудовану підтримку для тестування, що дозволяє розробникам створювати надійне та стабільне програмне забезпечення. Spring є популярним вибором для розробки Java-додатків завдяки своїй простоті використання, широким можливостям та високій продуктивності.

ASP.NET – фреймворк для веб-розробки, розроблений компанією Microsoft, який дозволяє розробникам створювати потужні та масштабовані веб-застосунки та сервіси. ASP.NET пропонує ряд інструментів та технологій для швидкої розробки програмного забезпечення. Його основу складає технологія .NET, яка дозволяє розробникам використовувати різні мови програмування, такі як C#, VB.NET, F#, для створення веб-застосунків. Основні особливості ASP.NET включають в себе ASP.NET MVC для створення веб-застосунків за шаблоном MVC (Model-View-Controller), ASP.NET Web Forms для швидкої розробки веб-форм та ASP.NET Web API для створення веб-сервісів. ASP.NET також має вбудовану підтримку для роботи з базами даних, включаючи Entity Framework для роботи з реляційними базами даних та ASP.NET Identity для реалізації системи автентифікації та авторизації.

Аналіз вищезазначених фреймворків та технологій дозволяє зробити висновок, що для розробки можна застосовувати різноманітні мови програмування. Відповідно до наведених мов і фреймворків слід підібрати відповідні середовища розробки. Основні вимоги полягають у зручності використання, середовище має покращувати досвід написання коду та

пришвидшувати процес, а також низькій вартості придбання програмного забезпечення. Далі наведені популярні середовища розробки та відповідні мови програмування, які підтримуються:

- Visual Studio (C++, C#, F#);
- Visual Studio Code (безліч мов, залежить від встановлених плагінів);
- Rider (C#);
- PyCharm (Python);
- InteliJ Idea (Java);
- Eclipse (Java);
- PhpStorm (PHP).

Наступним і не менш важливим модулем розроблюваного рішення є рівень зберігання даних. Будь яке кліент-серверне рішення потребує сховище даних, хоча б для зберігання власного стану. Більшість рішень збирає та зберігає інформацію своїх користувачів, дані домену, інфраструктури тощо. Велика увага приділяється безпеці під час зберігання даних, особливо якщо зберігаються чуттєві дані, такі як персональна інформація користувачів, банківські дані, приватні ключі та авторизаційні дані. У деяких випадках окремі типи даних потребують додаткового шифрування, що в свою чергу створює потребу у додатковому сховищі для ключів. Кожне сховище повинно мати систему контролю доступу до зберігаємої інформації, а також аутентифікацію та авторизацію користувачів.

Не менш важливим аспектом зберігання даних є надійність та стійкість до помилок. Втрата бази даних з великою вірогідністю призведе до закриття проекту, з технічних та репутаційних причин. На сьогоднішній день таких проблем не має існувати, оскільки усі сучасні бази даних та двигуни, що забезпечують їх роботу пропонують достатньо засобів уникнення подібних ситуацій.

База даних повинна підтримувати транзакційність для забезпечення цілісності даних. Транзакції реалізують нероздільність операції та повний її відкат у випадку помилки. Дуже важливо регулярно створювати резервні копії

усієї бази даних для відновлення у випадку втрати. Ситуації втрати цілого серверу разом з усіма даними цілком реальні в тому числі через регіональні та геокатастрофи. Тож слід розміщувати резервні копії у різних регіонах, або навіть на різних континентах. База даних повинна надавати зручні механізми реплікації даних та автоматичного відновлення у разі відмов. Також пов'язаною з реплікацією функціональністю є розділення навантажень між різними серверами. Така можливість наддасть змогу долати великі навантаження, наприклад у часи активного напливу користувачів, що в свою чергу знизить затримки та підвищить швидкість вирахування складних запитів. Виходячи з цього, важливим аспектом у виборі бази даних є швидкодія, її здатність забезпечувати продуктивність на необхідному рівні.

Додаткову увагу при виборі бази даних слід приділити інструментам моніторингу та управлінню. Наявність таких інструментів дозволяє швидко виявляти вразливі місця та проблеми. Іноді подібні механізми автоматично пропонують шляхи вирішення проблем та покращення продуктивності бази, що важливо при роботі з великими об'ємами даних. Також інструменти моніторингу зазвичай включають аналіз використання ресурсів, перегляд логів та моніторинг загального стану бази.

Одним з важливих аспектів при виборі бази даних також є структура самих даних. Слід визначити вимоги проекту до структури даних, а також їх обсягу. Загалом бази даних можна розділити на реляційні та нереляційні [11]. Реляційна база даних (RDBMS) – це тип бази даних, який зберігає дані в структурованих таблицях з відносинами (реляціями) між ними. Кожна таблиця складається зі стовпців (поля) та рядків (записів), а відношення між таблицями виражається за допомогою зовнішніх ключів. Реляційні бази даних використовують мову структурованого запиту (SQL) для взаємодії з даними, включаючи запити, вставку, оновлення та видалення. Нереляційна база даних (NoSQL) – це тип бази даних, який використовує нетрадиційні моделі зберігання даних, не засновані на реляційній моделі. Вони можуть включати ключ-значення, документ-орієнтовані, колонкові та графові бази даних. NoSQL

бази даних часто використовуються для зберігання неструктурованих або поліструктурованих даних, таких як тексти, зображення, відео або дані великих обсягів, які не легко моделювати за допомогою традиційної реляційної моделі.

Отже для визначення типу бази даних, необхідно спроектувати моделі да об’єкти даних, щоб розуміти їх структуру та вірогідний формат зберігання. Якщо ж дані структурованого формату з чіткою схемою та явними зв’язками, то слід віднести їх до реляційного типу. У іншому випадку, якщо дані неструктуровані, такі як текстові документи, зображення, великі документи без статичної схеми, тоді слід обрати нереляційний тип бази.

Також серед особливостей нереляційних баз даних варто виділити здатність ефективно масштабуватись та оптимізувати роботу з великими обсягами інформації. Деякі NoSQL бази даних, такі як Cassandra або Hadoop, спеціалізуються на роботі з великими обсягами даних. Зазвичай нереляційні бази даних показують більшу швидкість зчитування. Якщо дані часто змінюються або мають високу динаміку, то важливо обрати базу даних, яка підтримує швидкі зміни та має ефективні механізми синхронізації даних. Деякі NoSQL бази даних, такі як Redis або Apache Kafka, спеціалізуються на роботі з високою динамікою даних.

Розглянемо найпопулярніші бази даних обох типів, враховуючи вещеперелічені аспекти та вимоги. Слід зазначити, що для повноцінного функціонування продукту необхідно зберігати як реляційні так і не реляційні дані.

Microsoft SQL Server – це одна з найпопулярніших відкритих реляційних баз даних [12], яка використовується для зберігання та управління даними. Вона підтримує широкий спектр функцій, включаючи транзакції, індексацію, кешування, аутентифікацію та авторизацію, реплікацію та резервне копіювання. SQL Server використовує мову SQL для взаємодії з даними та має широку підтримку з боку спільноти та розробників. SQL Server відомий своєю швидкодією та надійністю, що робить його популярним вибором для веб-додатків, систем керування контентом, електронної комерції та багатьох інших

веб-проектів.

PostgreSQL – це ще одна відкрита реляційна база даних [13], яка відрізняється високою рівнем функціональності та розширюваності. Вона підтримує транзакції, тригери, збережені процедури, реплікацію, керування правами доступу та багато іншого. PostgreSQL також використовує мову SQL та має широкий набір функцій для аналізу та оптимізації запитів. PostgreSQL відомий своєю масштабованістю та гнучкістю, що робить його популярним вибором для великих підприємств та складних систем. Він має потужні інструменти для адміністрування та моніторингу.

MongoDB – це популярна документ-орієнтована NoSQL база даних [14], яка використовує JSON-подібні документи для зберігання даних, а саме бінарний формат BSON (Binary JSON). Вона не вимагає строго визначеної схеми, що робить її гнучкою та простою в застосуванні. MongoDB підтримує розподілені архітектури, високу доступність та масштабованість. MongoDB часто використовується для зберігання великих обсягів неструктурованих даних, таких як тексти, зображення та відео. Вона надає гнучкість у роботі з даними та швидкість розвитку, що робить її популярним вибором для розробників веб-додатків та інших проектів з великими обсягами інформації.

Elasticsearch – це потужна пошукова та аналітична NoSQL база даних, яка спеціалізується на обробці та аналізі великих обсягів неструктурованих даних. Вона використовує гнучку модель документів та широкий набір функцій для пошуку, агрегації та аналізу даних. Elasticsearch широко використовується для пошуку та аналізу великих обсягів даних у реальному часі. Вона часто використовується для створення розподілених пошукових двигунів, систем моніторингу та аналітики, а також для розробки пошукових інтерфейсів у веб-додатках.

Firestore – хмарна база даних [15], яка розроблена Google і призначена для зберігання та синхронізації даних між різними пристроями та платформами. Firestore використовує документи та колекції для організації даних. Це дозволяє створювати ієрархічні структури даних та легко отримувати доступ до них.

Однією з головних переваг Firestore є можливість отримувати реальний час оновлення даних. Це дозволяє розробникам створювати додатки, які миттєво реагують на зміни в базі даних та синхронізуються між різними пристроями. irestore автоматично масштабується відносно навантаження, що дозволяє обробляти великі обсяги даних та високі навантаження без необхідності вручну масштабувати інфраструктуру. Firestore підтримує різні типи даних, включаючи рядки, числа, масиви, об'єкти та булеві значення. Він також надає можливість створювати складні запити та фільтри для отримання потрібних даних. Firestore забезпечує високий рівень безпеки даних, використовуючи різноманітні механізми автентифікації та авторизації, а також забезпечуючи шифрування даних в спокої.

Фінальним об’єктом аналізу залишається спосіб інтеграції з валютними біржами. Варто зазначити що існує безліч валютних бірж, на яких буде можливо використовувати розроблювану технологію, оскільки інструменти аналітики і прогнозування рухів однакові, як на традиційних так і на крипто-ринках. Однак крипто-валютні біржі зазвичай є більш модерними і мають сучасніші способи інтеграції та актуальнішу документацію. Аналогічно з користувацьким інтерфейсом, розробка додаткових інтеграцій може бути виконана на пізніших етапах проекту для охвату більшої аудиторії. На ранніх етапах необхідно розглядати найпопулярніші і найпростіші рішення, знову ж таки притримуючись MVP концепції.

Більшість криптовалютних бірж сьогодні мають зручні способи інтеграції, зазвичай це HTTP API. Такий спосіб дозволяє комунікувати з біржою завдяки будь-якій кодовій базі, що значно розширює перелік доступних для вибору технологій.

Розглянемо що пропонує розробнику одна з найпопулярніших крипто-бірж Binance, яка є дуже популярною серед трейдерів та розробників, оскільки надає широкий функціонал та добре документований інтерфейс для розробки програм, які взаємодіють з біржею. Binance API [16] – це набір програмних інтерфейсів, які надають доступ до функціоналу криптовалютної біржі Binance.

Цей API дозволяє розробникам створювати програми, які автоматизують торгівлю криптовалютами, отримувати ринкові дані, керувати замовленнями та виконувати інші операції з рахунками користувачів. Binance API має кілька рівнів доступу, включаючи публічний доступ до ринкових даних, таких як ціни торгів, глибина ринку та історичні дані. Крім того, є також приватний доступ, який дозволяє користувачам маніпулювати своїми замовленнями, переводити кошти та виконувати інші дії на їхніх рахунках. Основні функції Binance API включають:

- розміщення, скасування та перегляд замовлень на ринку або лімітовані замовлення, виконання торгових стратегій автоматично;
- отримання актуальних цін, обсягів торгів та інших ринкових даних для різних криптовалютних пар;
- переведення коштів між рахунками, перегляд історії торгів та операцій;
- керування ключами API, обмеження доступу та інші заходи безпеки.

Як альтернативу проаналізуємо не менш зручний програмний інтерфейс Kraken API, який надає доступ до функціоналу криптовалютної біржі Kraken. Цей API дозволяє розробникам створювати програми для торгівлі криптовалютами, отримання ринкових даних, керування замовленнями та інші операції з рахунками користувачів. Основні можливості Kraken API:

- розміщення, скасування та перегляд замовлень на ринку або лімітовані замовлення, виконання торгових стратегій автоматично;
- отримання актуальних цін, обсягів торгів та інших ринкових даних для різних криптовалютних пар;
- переведення коштів між рахунками, перегляд історії торгів та операцій;
- керування ключами API, обмеження доступу до ресурсів, захист особистої інформації та фінансових операцій;
- доступ до платформи для проведення аналізу ринку, отримання фінансових звітів та інші додаткові можливості.

Kraken API широко використовується серед трейдерів та розробників через його потужний функціонал, стабільність та безпеку. Біржа також надає

детальну документацію та приклади коду для полегшення розробки програм, які взаємодіють з API.

На цих двох біржах перелік зручних програмних інтерфейсів не завершується і його можна продовжувати тривалий час. Наскільки би зручними пропоновані інтерфейси не були, при реальній роботі з біржами розробники стикаються з купою додаткових проблем і задач, які потребують вирішення для повноцінного і безпомилкового функціонування коду. До цих проблем, як правило, відносяться питання делегування і менеджменту секретів користувачів, валідація вхідних даних, залучення сторонніх сервісів та допоміжних функцій. Усе це потребує величезної кількості ресурсів та часу на створення додаткового шару, або ж цілої інфраструктури, яка стоїть між кінцевими точками інтерфейсу біржі і користувацьким інтерфейсом. Мати широкий стек технологій і вміння аналізувати існуючі рішення є надважливим навиком. Тож розглянемо, які існуючі бібліотеки і інструменти розробки і інтеграції з крипто-валютними біржами наявні у вільному доступі сьогодні.

Перша бібліотека, яка підходить під наші вимоги, написана на Python. Freqtrade – це безкоштовна бібліотека [17] для криптовалютної торгівлі з відкритим кодом. Вона розроблений для підтримки всіх основних бірж і керування через Telegram або веб UI. Freqtrade містить інструменти ретроспективного тестування, побудови та управління грошима, а також оптимізацію стратегії за допомогою машинного навчання. Серед основних можливостей Freqtrade варто виділити:

- можливість створювати власні стратегії, використовуючи Pandas – швидкий та гнучкий інструмент для аналізу відкритих даних та маніпуляцій з ними, написаний на Python;

- завантажувати історичні дані з крипто-валютних ринків для подальшого аналізу та тестування стратегій;

- пошук найкращих параметрів для своєї стратегії за допомогою гіпероптимізації, яка використовує методи машинного навчання. Розробник може оптимізувати параметри купівлі, продажу, тейк-профіту (ROI), стоп-лосс і

трейлінг-стоп-лосс для своєї стратегії;

- тестування стратегій з віртуальними грошима;
- подальший аналіз на основі даних зворотного тестування або історії торгівлі Freqtrade (база даних SQL), включаючи автоматизовані стандартні графіки та методи завантаження даних в інтерактивне середовище.

Gekko – платформа з відкритим кодом для автоматизованої торгівлі криптовалютою [18], в основному реалізована на JavaScript. Цей вибір мови забезпечує гнучкість і доступність, враховуючи широке використання JavaScript і його легкість. Gekko взаємодіє з криптовалютними біржами через їхні відповідні API, що дозволяє користувачам отримувати доступ до ринкових даних, здійснювати операції та впроваджувати торгові стратегії. Архітектура Gekko базується на модульних компонентах, що дозволяє користувачам налаштовувати та розширювати функціональні можливості відповідно до своїх конкретних потреб. Ядро Gekko включає модулі для збору даних, виконання стратегії та управління торгівлею. Основні функціональні можливості Gekko:

- Gekko дозволяє користувачам реалізовувати торгові стратегії за допомогою JavaScript. Трейдери можуть визначати свої стратегії на основі технічних індикаторів, цінових моделей та інших критеріїв. Модульна архітектура Gekko дозволяє інтегрувати власні стратегії або використовувати готові стратегії, доступні в екосистемі;
- після визначення торгівельної стратегії Gekko здійснює угоди від імені користувача відповідно до заданих параметрів. Це включає розміщення ордерів на купівлю та продаж, моніторинг виконання ордерів та управління позиціями портфеля. Можливості управління торгівлею Gekko спрямовані на оптимізацію ефективності торгівлі та мінімізацію ризиків;
- Gekko збирає ринкові дані в реальному часі з криптовалютних бірж через їхні API. Ці дані включають котирування цін, інформацію про книгу замовлень, обсяг торгів та інші відповідні показники. Постійно збираючи ці дані, Gekko надає користувачам актуальну інформацію про ринкові тенденції та умови.

## 2.2 Вибір та обґрунтування обраних технологій

Розглянувши найпопулярніші інструменти і технології, що можуть бути обрані для імплементації нашого рішення, а також провівши поверхневий аналіз кожної із них, повернемось до планування архітектури нашого рішення. Детальне планування архітектури проекту на початкових стадіях має вагомий вплив на подальшу розробку. Неправильне планування деталей проекту може призвести до помилок під час розробки та додаткової витрати часу. Аби запобігти подібним сценаріям, слід ретельно підбирати технології та інструменти під кожен окремий модуль проекту та етап розробки. Як було зазначено раніше платформа буде складатися з двох окремих застосунків, або модулів. Перший являє собою основний продукт доступний для користувача, а саме програмний алгоритм підтримки ринкових рішень із користувацьким інтерфейсом, представленим у вигляді бота у месенджері. Другий модуль має клієнт-серверну архітектуру і призначений для реєстрації і менеджменту користувачів.

Під кожен застосунок важливо підібрати найбільш гармонійні технології розробки. Під час вибору важливо враховувати індивідуальні особливості тої чи іншої мови програмування, фреймворку, бібліотеки тощо. Отже визначимо ключові потреби кожного модуля платформи.

Першочерговою потребою проектів з комерційним потенціалом є охоплення великої аудиторії. Для розвитку і масштабування потрібна широка і активна база користувачів. Під час планування слід визначити напрям та потенційну аудиторію продукту, аби спиратись на це при виборі технологій. У нашому випадку необхідно визначити цільову аудиторію, серед холдерів та тредерів криптовалюти.

Проаналізуємо відкриті джерела, що підрахували статистику використання криптовалюти, як в Україні так і в інших державах, серед звичайних громадян. За даними глобальної сервісної IT-компанії Triple-A [19], доволі неочікувано Україна посідає перше місце за рейтингом країн, де є

найбільше власників криптовалют стосовно усього населення. Так, в Україні мають криптовалюту 12,7% населення – 5,6 млн осіб (рисунок 2.1). Також за 2023 рік загальний відсоток українців, які володіють цифровими активами зріс на 2%.


Рисунок 2.1 – Кількість власників криптовалюти у країнах світу

За даними міністерства фінансів [20] України власників криптовалют чоловічої статі нараховується 93,8%, жінок – 6,2% (рисунок 2.2). З них 42% – віком від 35 до 44 років. Друга за чисельністю група – люди віком від 45 до 54 років. 6,6% опитаних розповіли, що їм більше ніж 55 років.

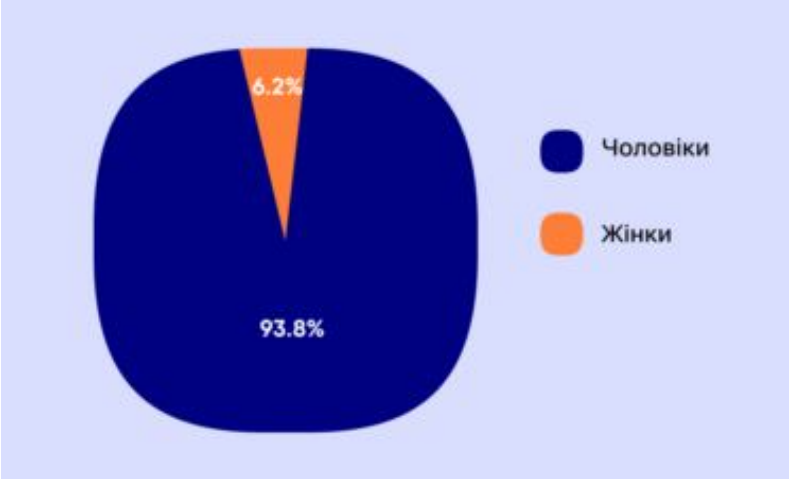


Рисунок 2.2 – Співвідношення чоловічої та жіночої крипто-аудиторії

Тож найбільшою аудиторією в Україні, що володіє криптовалютою, згідно соціальних досліджень є чоловіки віком від 35 до 44 років. Тепер необхідно визначити якою платформою зручніше користуватись для більшості з них. Раніше ми вже згадували, що для розробки першої версії продукту, більш зваженим та потенційним варіантом є рішення доступне одразу на багатьох платформах. Таким рішенням на сьогоднішній день виступають численні месенджери, що пропонують своїм користувачам доступ з мобільних девайсів, комп'ютерних застосунків та браузерів.

За даними соціального опитування [21] 2021 року, яке проводилось методом телефонних інтерв'ю з використанням комп'ютера. Всього опитали 2002 особи на основі випадкової вибірки мобільних телефонних номерів. Опитування проводилося серед мешканців віком 18 років і старше у всіх регіонах України крім окупованих територій. Статистична похибка вибірки не перевищує 2,4% для показників, близьких до 50%.

Найбільш популярним серед українців засобом для комунікації виявився Viber, яким користуються 73,6% громадян. Про це йдеться в результатах опитування Київського міжнародного інституту соціології. На другому місці за популярністю виявився Messenger від Facebook. Ним користуються 42,7% опитаних.

Ще 31,6% громадян використовують Telegram, а 25,3% – WhatsApp. Згідно з динамікою використання мобільних додатків, українці почали більше використовувати Telegram – популярність підвищилась з 24,2% до 31,6%. За статевим розподілом, більш суттєва різниця, ніж в інших додатків, виявилися в Telegram, яким користуються 34,9% чоловіків та 28,9% ніж жінок. Якщо говорити про вікові групи (таблиця 2.2), то Viber є найпопулярнішим у кожній з категорій. Водночас серед українців 18-29 років, на другому і третьому місці за популярністю знаходиться Telegram (64,3%) та Instagram (61,0%).

Таблиця 2.2 – Відсоток користувачів мобільними застосунками соціальних мереж та месенджерів в залежності від віку

| Застосунок | 18-26 | 30-39 | 40-49 | 50-59 | 60-69 | 70 + |
|---|---|---|---|---|---|---|
| Viber | 85,5 | 89,1 | 82,6 | 72,2 | 63,2 | 35,1 |
| WhatsApp | 27,7 | 33,7 | 27,4 | 22,5 | 23,7 | 11,5 |
| Фейсбук | 51,6 | 48,9 | 41,9 | 39,0 | 29,9 | 18,5 |
| Телеграм | 64,3 | 46,0 | 26,0 | 22,2 | 12,8 | 7,7 |
| Твіттер | 8,9 | 8,4 | 3,1 | 1,4 | 1,2 | 0,3 |
| Сигнал | 7,8 | 5,4 | 3,1 | 2,6 | 1,2 | 0,8 |
| Скайп | 14,0 | 12,8 | 11,4 | 10,6 | 12,8 | 8,5 |
| Інстаграм | 61,0 | 38,4 | 21,1 | 10,6 | 3,1 | 0,6 |
| Tik Tok | 20,7 | 10,9 | 8,0 | 7,3 | 3,3 | 2,3 |
| Інше | 3,5 | 3,8 | 3,7 | 2,4 | 1,7 | 1,1 |
| Нічим | 1,4 | 2,7 | 9,7 | 10,3 | 12,8 | 27,2 |
| Немає відповіді | 1,2 | 1,0 | 0,0 | 1,2 | 0,3 | 0,8 |

При цьому додатки Telegram та WhatsApp дещо більше використовує міське населення. Так, Telegram користуються 37,1% українців серед міського населення та 20,3% серед сільського. WhatsApp же використовують 29,7% містян та 16,3% серед сільського населення. Водночас серед громадян, які

живуть у селі, вищий рівень тих, хто не використовує жоден з перерахованих застосунків – 24,7%. Серед містян таких українців 15,8%.

Аналіз статистичних даних показує, що найпопулярнішим месенджером серед вікової категорії потенційних користувачів є Viber. Однак на другому місці зі значенням 46 – 64% серед населення віком від 18 до 39 років розмістився Telegram. Також опитування продемонструвало, що більша частка респондентів, що надає перевагу месенджеру Telegram є чоловічої статі. Отже зважаючи на соціологічне опитування та стрімкий ріст популярності серед українців Telegram є одним з найпріоритетніших кандидатів, поступаючись лише Viber. Однак головною особливістю платформи Telegram є безкоштовність та зручність Telegram API, на відміну від інтерфейсу запропонованого Viber. Додатково величезний набір бібліотек, створених для розширення можливостей побудови ботів у месенджері Telegram, а також бездоганна документація роблять цей месенджер найбільш підходящим вибором для розробки нашого застосунку.

Для розробки головного алгоритму пріоритет надано бібліотеці Freqtrade, написаній на Python, зо в свою чергу гарантує швидку та просту інтеграцію бібліотеки в загальний проект. Також інструмент дозволяє будувати користувацькі стратегії, і оптимізувати їх за допомогою FreqAI, програмного забезпечення, призначеного для автоматизації різноманітних завдань, пов’язаних із навчанням моделі прогнозного машинного навчання для створення прогнозів ринку за набору вхідних сигналів. Додатково користувач зможе тестувати свої стратегії на історичних даних. Freqtrade пропонує великий перелік крипто-бірж з якими можна інтегруватись, а також простий на надійний інструмент делегування секретів користувача. Ці та багато інших переваг роблять бібліотеку Freqtrade ідеальним кандидатом для розробки алгоритму підтримки ринкових рішень.

Основну логіку застосунку і інтеграцію з бібліотекою Freqtrade буде написано мовою програмування Python. Оскільки Python має простий і лаконічний синтаксис, що робить код більш зрозумілим та легким для читання,

а також підтримується на різних операційних системах, включаючи Windows, macOS та різні дистрибутиви Linux. Це робить його універсальним і доступним для розробників на різних платформах.

Застосунок буде зберігати інформацію про аккаунт користувача у месенджері Telegram, його секрети для інтеграції з акаунтами на крипто-біржах, дані про власні стратегії, а також дані сесії. Оскільки один користувач може мати декілька різних стратегій та варіативний набір інтеграцій з біржами, дані не є структурованими та більш відносяться до нереляційних, оскільки можуть зберігатись єдиним документом вільного формату. Сховище даних має гарантувати синхронізацію між різними застосунками, з огляду на те, що частина даних користувача буде заноситися авторизаційним модулем. З огляду на те, що сховище має зберігати чутливі дані воно має бути безпечним та надійним, а дані сесії мають завжди мати актуальний стан і бути цілісними. Згідно аналізу усім цим вимогам відповідає база даних, розроблена компанією Google, і призначена для швидкої взаємодії між пристроями і платформами – Firestore.

Модуль авторизації та менеджменту користувачів в свою чергу ставить задачу обробки, сортування та фільтрації великого об’єму інформації, оскільки збирає реєстраційні дані користувачів. Дані такого роду, як логін, пароль, імейл, номери телефонів є чутливими і потребують додаткового захисту під час зберігання, можливості відновлення у разі втрати. Окрім даних користувачів, цей застосунок оперує довідковою інформацією для користувачів, надає доступ до базових стратегій, створених платформою і дозволяє закріпити їх за своїм акаунтом. Структура та взаємодія усіх даних створює явну реляційну схему побудовану на таблицях, що зберігають скалярні типи даних. Для цих цілей добре підходить Microsoft SQL Server.

Для обробки даних перевага надається також продукту Microsoft, який добре взаємодіє з SQL Server, має зручні бібліотеки для менеджменту і обробки великих обсягів даних. Цим продуктом звісно є ASP.NET фреймворк, він пропонує зручні інструменти авторизації та аутентифікації користувачів,

обробки веб запитів, аналітичні та статистичні інструменти. Цей фреймворк дозволить побудувати потужний та легко масштабований програмний інтерфейс.

Для розробки веб-інтерфейсу модуля найбільш підходить Flutter завдяки можливості використовувати одну кодову базу, як для веб сайту так і для мобільних застосунків. Така перевага робить потенційне масштабування проекту на різні платформи більш легкою задачею. Гнучкість дизайну надає широкі можливості для створення інтерактивних веб-інтерфейсів. А власний механізм розроблений для ефективного рендерингу віджетів та сторінок дозволяє створювати рішення з високою швидкодією і плавною анімацією.

## 2.3 Огляд хмарних обчислювальних сервісів

Сьогодні важко уявити стартап, чи молоду компанію, яка на старті свого бізнесу або проекту вирішує орендувати приміщення для розміщення там серверів для хостингу власного програмного забезпечення. Звісно такий варіант може бути обраний з метою забезпечення повної приватності даних, що зберігаються на серверах. Але з часом підтримка власних серверів стає все менш популярною практикою серед продуктових компаній, через свою непрактичність, високу вартість та ненадійність. Все більшої популярності набирають хмарні обчислювальні сервіси, що зазвичай є більш лаконічними у своєму налаштуванні, не потребують підтримки технічного та програмного забезпечення, а головне є більш надійними.

Провайдери хмарних сервісів повністю підтримують роботу власних серверів, вчасно адаптують їх під стрімкий розвиток технологій і розростання об’єму інформації, що транспортується мережею. Зазвичай великі провайдери мають сервери по усьому світу і пропонують інструменти налаштування вашого програмного забезпечення для розподілення по різним регіонам, або навіть континентам. Такий підхід забезпечує стійкість до різноманітних катастроф, оскільки трафік у будь-який момент може буде переправлений на

інший сервер. Інструменти розподілення трафіку також можуть стати у нагоді при врегулюванні підвищеної активності на платформі. Провайдери додатково роблять бекапи та резервні копії баз даних та стану застосунків.

Головним аспектом при виборі хмарних сервісів для обчислень, зберігання даних чи розгортання програмного забезпечення, є відносна дешевизна у порівнянні з ціною підтримки власного серверу. Хмарні обчислювальні сервіси є привабливою пропозицією як для маленьких стартапів так і для великих корпорацій. Розглянемо найбільші та найнадійніші провайдери хмарних обчислювальних сервісів.

Amazon Web Services (AWS) [22] один з провідних хмарних обчислювальних сервісів у світі, який пропонує широкий спектр послуг для різних потреб, включаючи хостинг застосунків, зберігання даних та підвищення надійності. Amazon Elastic Compute Cloud (EC2) дозволяє запускати віртуальні сервери у хмарі, що надає гнучкість та масштабованість для хостингу веб-застосунків. Клієнт може вибрати розмір, тип та конфігурацію серверів відповідно до власної потреби. Хмарні сервіси зазвичай мають вбудовані механізми для забезпечення високої доступності застосунків. AWS, наприклад, надає можливості реплікації та регіональної розподіленості, що дозволяє зменшити ризик від відмов. Хмарні обчислювальні сервіси зазвичай мають вбудовані механізми безпеки, такі як мережеві брандмауери, шифрування даних та автентифікація користувачів. AWS пропонує різні інструменти безпеки, такі як AWS Identity and Access Management (IAM), що дозволяє керувати доступом користувачів до ресурсів. AWS надає Amazon Elastic Kubernetes Service (EKS), що дозволяє легко розгортати, керувати та масштабувати кластери Kubernetes [23] у хмарному середовищі. EKS автоматизує такі операції, як розгортання, масштабування та керування кластерами Kubernetes, дозволяючи розробникам зосередитися на розробці застосунків, а не на управлінні інфраструктурою. Це робить інтеграцію Kubernetes з AWS привабливим вибором для тих, хто шукає сучасні та ефективні засоби розгортання та управління контейнеризованими

застосунками.

Microsoft Azure [24], подібно до Amazon Web Services, надає розгалужені можливості для хостингу та управління інфраструктурою. Azure пропонує широкий набір послуг хмарного обчислення, включаючи обчислювальні ресурси, зберігання даних, мережеві послуги, інструменти розгортання та моніторингу. Azure пропонує розподілення по зонам (Availability Zones), що дозволяє розгортати додатки у різних фізичних областях для забезпечення високої доступності та стійкості. Це дозволяє уникнути відмов у разі відмови в одній зоні та забезпечує безперервну роботу додатків. Зокрема, Azure Kubernetes Service (AKS) дозволяє легко створювати, керувати та масштабувати кластери Kubernetes в хмарному середовищі Azure. Також важливою послугою є Azure DevOps, яка надає інструменти для автоматизації процесів розробки, тестування та розгортання додатків у хмарному середовищі. Azure DevOps підтримує інтеграцію з Kubernetes, що дозволяє створювати CI/CD конвеєри для розгортання змін у кластери Kubernetes.

Обидві платформи пропонують схожі рішення як для розгортання проектів, так і для моніторингу масштабування. Як зазначено вище також і Azure, і Amazon надають інструменти роботи з Kubernetes Service, що є дуже важливим аспектом у нашому проекті, оскільки архітектура алгоритму підтримки ринкових рішень буде побудована на оркестрації контейнеризованих програм за допомогою Kubernetes.

Платформа Azure розроблена компанією Microsoft є нативною для застосунків побудованих на .Net фреймворку та SQL Server. Сервіси запропоновані Azure дозволять не тільки розгорнути базу даних та API, а й підтримувати і масштабувати. Потужні моніторингові сервіси дозволять вчасно реагувати на відмову сервісів та вчасно відновлювати їх роботу.

Amazon Web Services пропонує більш легкий інтерфейс менеджменту такого складного сервісу як Kubernetes. Наявні на платформі інструменти достатні для забезпечення стабільної та контрольованої роботи усієї інфраструктури алгоритму підтримки ринкових рішень.

# 3 РОЗРОБКА АЛГОРИТМУ ПІДТРИМКИ РИНКОВИХ РІШЕНЬ

Алгоритм підтримки ринкових рішень є комплексним рішенням для побудови і оптимізації торгівельних стратегій, шляхом аналізу даних та зворотного тестування. Фінальний продукт має дозволити користувачу за допомогою інтерфейсу чат-боту Telegram швидко налаштувати оптимальну стратегію торгівлі, та взаємодіяти з криптобіржою виключно за допомогою команд у чаті. Мікросервісна архітектура, розроблена для підтримки одночасного функціонування багатьох екземплярів програм-контейнерів, забезпечить відмово-стійкість та надійність системи.

Додатково створено веб-застосунок, першочергово для розповсюдження інформації про продукт та залучення нових користувачів. За допомогою веб-сайту нові користувачі зможуть реєструватись у системі та фіксувати за своїм акаунтом стратегії, пропоновані платформою. До застосунку також додана реферальна система для залучення користувачів. Сайт надає широкий спектр адміністративних функцій, таких як керування доступ до торгівельних стратегій, менеджмент користувачів та їх можливостей, ведення блогу тощо.

Разом ці два застосунки утворюють конкурентно-спроможну платформу з великим потенціалом до росту та розвитку з залученням нових технологій.

## 3.1 Розробка архітектури програмного алгоритму

Основний алгоритм має функціонувати наступним чином: отримувати вхідні сигнали з інтерфейсу користувача та виконувати відповідні функції за допомогою бібліотеки FreqTrade. Алгоритм має забезпечувати ізольованість даних кожного користувача. Інформація стосовно транзакцій, угод, ордерів та портфелю користувача має залишатись приватною і не зберігатись на стороні системи. Користувач повинен мати можливість у будь-який час зупинити виконання алгоритму, або остаточно відмовитись від послуг платформи. Так

само адміністратор платформи повинен мати змогу обмежити доступ користувача до алгоритму або зупинити його роботу. Першочерговою метою при побудові архітектури застосунку є забезпечення необхідної відмовостійкості та продуктивності системи для подолання високих навантажень. Архітектура застосунку будується на роботі з представленою бібліотекою FreqTrade. Деякі обмеження можуть накладатися самою бібліотекою, тож розглянемо детальніше її основний функціонал.

Freqtrade (далі бот) має багато функціональностей та можливостей. За замовчуванням увесь функціонал налаштовується через конфігураційний файл. Під час роботи бот використовує набір параметрів конфігурації, які всі разом відповідають конфігурації бота. Зазвичай він зчитує свою конфігурацію з файлу (файл конфігурації Freqtrade). За замовчуванням бот завантажує конфігурацію з файлу config.json, розташованого в поточному робочому каталозі. Також можливо вказати інший файл конфігурації, який використовуватиме бот, за допомогою параметра командного рядка -c/--config.

Якщо використовується метод швидкого запуску для встановлення бота, сценарій інсталяції автоматично створює файл конфігурації за замовчуванням (config.json). Якщо файл конфігурації за замовчуванням не створено, рекомендується використовувати freqtrade new-config --config user_data/config.json для створення базового файлу конфігурації. Файл конфігурації Freqtrade має бути написаний у форматі JSON. Бот перевіряє синтаксис конфігураційного файлу під час запуску та попереджатиме, якщо були допущені помилки під час його редагування, вказуючи на проблемні рядки.

Встановлені параметри в конфігурації Freqtrade за допомогою змінних середовища мають пріоритет над відповідним значенням у конфігурації чи стратегії. Змінні середовища повинні мати префікс FREQTRADE__ (лістинг 3.1) для завантаження в конфігурацію. Символ «__» служить роздільником рівнів, тому використовуваний формат має відповідати FREQTRADE__{section}__{key}. Таким чином, змінна середовища, визначена

як export FREQTRADE__STAKE_AMOUNT=200, призведе до {stake_amount: 200}. Більш складним прикладом може бути експорт FREQTRADE__EXCHANGE__KEY=<yourExchangeKey>, щоб зберегти ключ обміну в секреті. Це перемістить значення в розділ конфігурації exchange.key. За допомогою цієї схеми всі налаштування конфігурації також будуть доступні як змінні середовища.

Лістинг 3.1 – Приклад конфігурації за допомогою змінних середовища

```
FREQTRADE__TELEGRAM__CHAT_ID=<telegramchatid>
FREQTRADE__TELEGRAM__TOKEN=<telegramToken>
FREQTRADE__EXCHANGE__KEY=<yourExchangeKey>
FREQTRADE__EXCHANGE__SECRET=<yourExchangeSecret>
```

Деякі параметри можуть бути встановлені у файлі стратегії. Варто зауважити що параметри встановленні на рівні середовища переважають параметри конфігураційного файлу, а він в свою чергу переважає над параметрами файлу стратегії. Настільні параметри можливо встановити як через конфігураційний файл так і через стратегію:

- minimal_roi;
- timeframe;
- stoploss;
- max_open_trades;
- trailing_stop;
- trailing_stop_positive;
- trailing_stop_positive_offset.

Бот надає можливість створювати власні кастомізовані стратегії для покращення досвіду торгівлі на біржі. Щоб створити пустий файл стратегії достатньо викликати команду freqtrade new-strategy --strategy <StrategyName>. Розгнлянемо з яких основних компонентів складається файл стратегії:

- індикатори ;
- правила стратегії входу;

- правила стратегії виходу;
- рекомендована мінімальна рентабельність інвестицій;
- правила стоп-лосс.

Крім того, існує атрибут INTERFACE_VERSION, який визначає версію інтерфейсу стратегії, яку має використовувати бот. Поточна версія – 3, яка також є стандартною, якщо її не встановлено явно в стратегії.

Freqtrade використовує pandas для зберігання/надання свічкових даних (OHLCV). Pandas – бібліотека, розроблена для обробки великих обсягів даних. Кожен рядок у кадрі даних відповідає одній свічці на діаграмі, причому остання свічка завжди є останньою у кадрі даних (лістинг 3.2). Pandas надає швидкі способи обчислення показників. Щоб отримати вигоду від такої швидкості, рекомендується не використовувати цикли, а натомість використовувати векторизовані методи. Векторизовані операції виконують обчислення в усьому діапазоні даних і тому, порівняно з циклічним проходженням кожного рядка, набагато швидше під час обчислення показників.

Лістинг 3.2 – Метрики що зберігаються у pandas

```
> dataframe.head()
            date      open      high       low     close     volume
0 2021-11-09 23:25:00 67279.67 67321.84 67255.01 67300.97   44.622
1 2021-11-09 23:30:00 67300.97 67301.34 67183.03 67187.01   61.380
2 2021-11-09 23:35:00 67187.02 67187.21 67031.93 67123.81   113.42
3 2021-11-09 23:40:00 67123.80 67222.40 67080.33 67160.48   78.96
4 2021-11-09 23:45:00 67160.48 67160.48 66901.26 66943.37   111.392
```

Більшість індикаторів мають нестабільний період запуску, в якому вони або недоступні (NaN), або розрахунок невірний. Це може призвести до невідповідностей, оскільки Freqtrade не знає, як довго має тривати цей нестабільний період. Щоб врахувати це, стратегії можна призначити атрибут startup_candle_count. Це повинно бути встановлено на максимальну кількість свічок, яку вимагає стратегія для розрахунку стабільних показників. Значення – це максимальний період (у свічках), який необхідний будь-якому з

інформаційних таймфреймів для обчислення стабільних індикаторів. Гарною практикою є використання рекурсивного аналізу, щоб перевірити та знайти правильний параметр startup_candle_count.

Хоча основні функції стратегії (populate_indicators(), populate_entry_trend(), populate_exit_trend()) слід використовувати у векторизованому вигляді та викликати лише один раз під час ретестування, зворотні виклики викликаються «за потреби». Таким чином, слід уникати важких обчислень у зворотних викликах, щоб уникнути затримок під час операцій. Залежно від використаного зворотного виклику (лістинг 3.3), вони можуть бути викликані під час входу/виходу з угоди або протягом угоди.

Лістинг 3.3 – Простий зворотний виклик, який викликається один раз під час завантаження стратегії

```
import requests

class AwesomeStrategy(IStrategy):
    def bot_start(self, **kwargs) -> None:
        if self.config['runmode'].value in ('live', 'dry_run'):
            self.custom_remote_data =
requests.get('https://some_remote_source.example.com')
```

Обробники списку пар визначають список пар, якими бот має торгувати. Вони налаштовуються в розділі парних списків параметрів конфігурації. У конфігурації можливо використовувати статичний список пар (визначений обробником списку пар StaticPairList) і динамічний список пар (визначений обробником списку пар VolumePairList). Крім того, AgeFilter, PrecisionFilter, PriceFilter, ShuffleFilter, SpreadFilter і VolatilityFilter діють як фільтри списку пар, видаляючи певні пари та/або переміщуючи їхні позиції в списку пар. Якщо використовується кілька обробників парних списків, вони об’єднуються в ланцюжок, а комбінація всіх парних обробників утворює кінцевий парний список, який бот використовує для торгівлі та ретестування. Обробники списків пар виконуються в послідовності, на яку вони налаштовані. Додатково є

можливість визначити StaticPairList, VolumePairList, ProducerPairList, RemotePairList або MarketCapPairList як початковий обробник списку пар. Неактивні ринки завжди видаляються з отриманого списку пар. Явно занесені в чорний список пари (ті, що в налаштуваннях конфігурації pair_blacklist) також завжди видаляються з отриманого списку пар.

Чорний список пар (налаштовується через exchange.pair_blacklist у конфігурації) забороняє певним парам торгувати. Це може бути так само просто, як виключення DOGE/BTC, що видалить саме цю пару. Чорний список пар також підтримує символи підстановки (у стилі регулярних виразів), тому BNB/.* виключатиме BCI пари, які починаються з BNB.

Бот пропонує зручний інструмент для бектестінгу, або ж тестування стратегії на історичних даних. Після конфігурації стратегій входу та виходу необхідно завантажити деякі історичні дані бажаної валютної пари. Під час тестування за умовчанням використовуватимуться криптовалюти (пари) із конфігураційного файлу та історичні дані свічки (OHLCV) з user_data/data/<exchange>. Якщо дані для комбінації біржі/пари/таймфрейму недоступні, тестування зупиниться та повідомить про необхідність спочатку завантажити їх за допомогою даних завантаження freqtrade. Результат тестування підтвердить, чи стратегія має кращі шанси на отримання прибутку, ніж на збиток. Усі розрахунки прибутку включають комісію, і бот використовуватиме для розрахунку комісію біржі за замовчуванням. Для ретестування знадобиться початковий баланс, який можна надати як аргумент командного рядка --dry-run-wallet <balance> або --starting-balance <balance> або через налаштування конфігурації dry_run_wallet. Ця сума має бути вищою за stake_amount, інакше бот не зможе симулювати жодну торгівлю.

Бібліотека FreqTrade пропонує сучасний інструмент оснований на роботі штучного інтелекту для розширення можливостей бота. FreqAI – це програмне забезпечення, призначене для автоматизації різноманітних завдань, пов’язаних із навчанням моделі прогнозного машинного навчання для створення прогнозів ринку за набору вхідних сигналів. FreqAI потребує набір спеціальних базових

індикаторів (так само, як і в типовій стратегії Freqtrade), а також цільові значення (мітки). Для кожної пари в білому списку FreqAI навчає модель для прогнозування цільових значень на основі введення користувацьких індикаторів. Потім моделі послідовно перенавчаються із заздалегідь визначеною періодичністю, щоб адаптуватися до ринкових умов. FreqAI пропонує можливість як тестувати стратегії (імітація реальності з періодичним перенавчанням на історичних даних), так і розгортати живі запуски. У реальних умовах FreqAI можна налаштувати на постійне перенавчання у фоновому потоці, щоб підтримувати моделі якомога актуальнішими. FreqAI налаштовується за допомогою типового конфігураційного файлу Freqtrade і стандартної стратегії Freqtrade.

Основні обмеження, які накладає бібліотека напряму пов’язані зі специфікою ринкових рухів. Розробники рекомендують уникати важких обчислень під час зворотних викликів у виконанні стратегії. Оскільки іноді ринкові рухи можуть бути доволі динамічними, час відведений на обробку інформації та розрахунки має бути зведений до мінімуму. Головним обмеженням є саме конфігурація бота, оскільки базовий алгоритм спирається на конфігураційні файли у директорії екземпляру, або ж на змінні середовища. Таку проблему доволі легко вирішити розділенням основного потоку у якому працює алгоритм на декілька, що працюють з індивідуальними конфігураційними файлами. Але варто зазначити, що робота стратегій, є доволі трудомістким процесом, навіть без урахування функції FreqAI. В залежності від потужності апаратного забезпечення, один екземпляр застосунку зможе обробляти від декількох одиниць до декількох десятків користувачів без значної втрати продуктивності. Активація FreqAI значно знижує швидкість обробки стратегії та майже унеможливлює обробку декількох стратегій одночасно.

На меті проекту також є захист секретів та приватних даних користувачів. Під час проектування було прийнято рішення відмовитись від зберігання особистих даних користувачів, що стосуються роботи з валютами та біржами.

За замовчуванням усі секрети що надає користувач та дані, які алгоритм отримує від бірж зберігаються в оперативній пам’яті разом з іншими змінними екземпляру бота. Тривалість життя подібних секретів дорівнює тривалості життя бота. При такому рішенні постає нова проблема, також пов’язана з мультикористувацьким характером алгоритму. Алгоритм не може завершити роботу, доки останній активний користувач не наддасть боту команду зупинки. Такий варіант не задовольняє першочергову постановку задачі.

Альтернативним рішенням обох проблем можна обрати архітектуру при якій один екземпляр боту буде належати лише одному користувачу. Це знижує навантаження на кожен окремий екземпляр, а також дозволяє ізолювати персональні конфігурації користувачів та їхні секрети у межах одного екземпляру, водночас обмежити тривалість життя. Один ізольований бот зможе забезпечити значно швидшу взаємодію з біржою і убезпечити чутливі дані.

При такій роздрібненій архітектурі виникають нові проблеми пов’язані з управлінням великою кількістю окремих екземплярів та масштабуванням. При кожному запиті на старт бота, система має згенерувати новий файл конфігурації, загрузити відповідну стратегію і запустити новий екземпляр програми. Коли ж користувач адресує запити до бота для повернення певної інформації, або ж зміни конфігурації, система має визначити правильний екземпляр програми, який дійсно належить користувачу, і виконати певні інструкції для досягнення мети. Відповідно при команді на зупинку алгоритму, система має безпомилково зупинити вірний екземпляр бота.

Кожна окрема програма має обмежуватися необхідною кількістю оперативної пам’яті і обчислювальних ресурсів процесору, а також мати доступ до мережі інтернет. Для такої задачі чудово підходить контейнеризація програми, що сприятиме розподіленню та обмеженню ресурсів, споживаних алгоритмом. При контейнерному підході кожен окремий екземпляр бота буде належати до власного контейнеру і мати тривалість життя пов’язану з існуванням контейнеру. Проблема менеджменту контейнерів та маршрутизації користувача до необхідного екземпляру залишається відкритою.

Для подолання проблематики управління контейнерами та їхнім життєвим циклом буде використаний Kubernetes – відкрита платформа для автоматизації розгортання, масштабування та управління контейнеризованими застосунками. Kubernetes сприяє використанню контейнерної технології, такої як Docker, для упаковки програмного коду та всіх його залежностей в контейнери. Це дозволяє додаткам працювати незалежно від середовища та забезпечує їх консистентність в будь-якому середовищі. Для вирішення питання менеджменту ресурсів для контейнерів, Kubernetes провадить автоматичне розміщення та управління контейнерами на вузлах кластера. Він розподіляє ресурси, такі як CPU та пам'ять, між контейнерами для оптимального використання ресурсів. Таким чином контейнеризований екземпляр алгоритму отримує переваги оркестрації Kubernetes у вигляді підтримки необхідної для функціонування програми ресурсної бази. Додатково при великих навантаженнях доступна функція автоматичного масштабування. Kubernetes дозволяє автоматично масштабувати додатки вгору або вниз в залежності від навантаження. Він може виявити аномалії у роботі та навантаженні контейнеру, коли потрібно запустити додаткові копії контейнерів або зупинити непотрібні, для забезпечення стабільної роботи програми. Також під час розробки використовується інструмент Kubernetes, що надає можливість створювати мережеві сервіси для забезпечення комунікації між різними контейнерами та компонентами системи. Він також дозволяє налаштовувати правила маршрутизації та балансування навантаження.

Kube API надає доступ до великого переліку функцій Kubernetes і покриває усі потреби нашої системи, пов’язані з менеджментом одразу багатьох контейнерів. Інтерфейс дозволяє доволі у простій формі надсилати команди до механізму Kubernetes. Наприклад, Kube API дозволяє створювати, оновлювати та видаляти різні ресурси Kubernetes, такі як контейнери, розгортки, сервіси, конфігурації мережі тощо. Додатково за допомогою Kube API можна отримувати дані про стан та роботу кластера Kubernetes, а також журнали подій для відслідковування проблем та відлагодження.

Для менеджменту користувацьких контейнерів буде реалізовано керуючий модуль на мові програмування Python. Цей модуль буде поєднувати запити з клієнтської частини у логічні конструкції та перенаправляти їх, за допомогою Kube API, до Kubernetes Engine. Надалі цей модуль буде мати назву адміністративного. Модуль буде контейнеризований як окрема програма та не буде мати безпосереднього доступу до користувацьких контейнерів. Уся система буде мати чітку послідовність обробки запитів, як показано на рисунку 3.1. Зображений кластер об'єднує адміністративний модуль та усі контейнери користувачів.

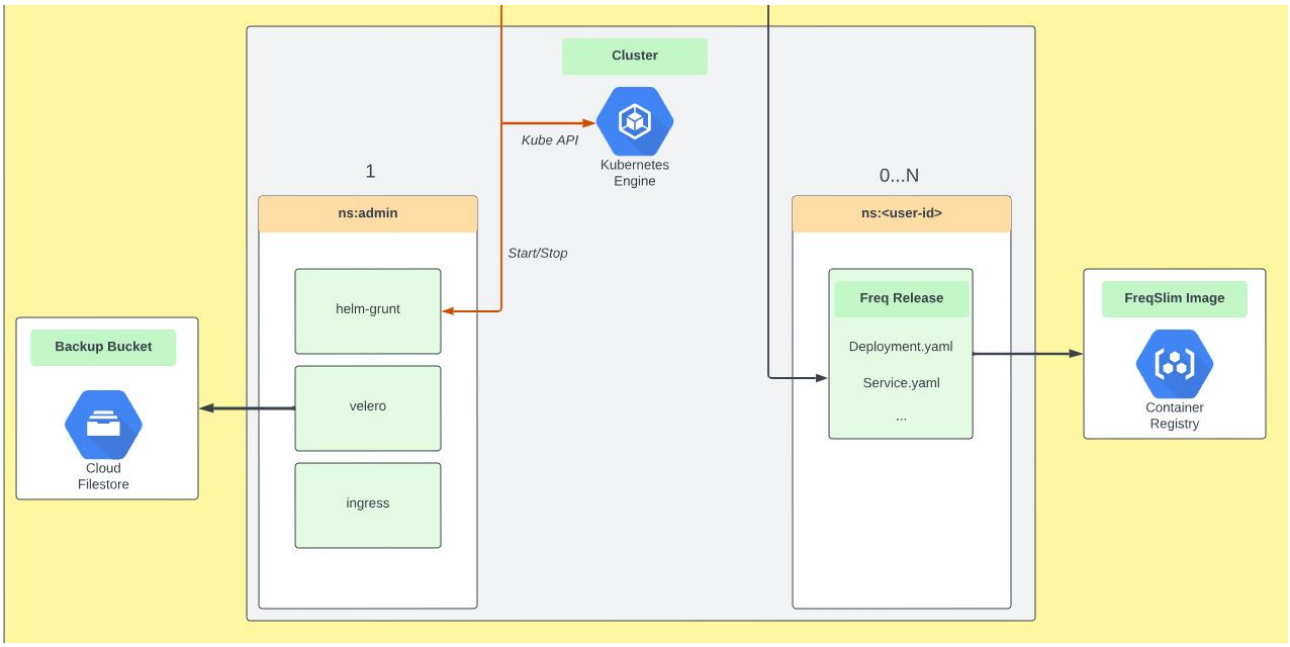


Рисунок 3.1 – Kubernetes кластер для управління контейнерами застосунку

Адміністративний модуль переважно має завдання менеджменту життєвого циклу користувацьких контейнерів, виконувати умовні команди старту та стопу алгоритму. Відповідні команди ініціюють формування чарту, пакету, який містить всі компоненти та ресурси, необхідні для розгортання контейнеру в кластері. Чарт мітить всередині yaml файл, що представляє набір інструкцій для розгортання нового екземпляру програми. Модуль збирає усі

необхідні конфігурації в залежності від запиту, що надійшов від клієнта. Таким чином у чарті розмішується файл стратегії, який користувач бажає використовувати для торгівлі на біржі, а також загальний конфігураційний файл що мітить налаштування алгоритму. Для конфігурації облікових даних користувача формується скрипт для запису їх в зміні середовища. Сформований чарт використовує Helm [25] – пакетний менеджер для Kubernetes, який допомагає спростити розгортання, управління та оновлення додатків у кластері Kubernetes. Він використовує концепцію «чартів» (charts), які описують конфігурацію та ресурси, необхідні для розгортання додатку в Kubernetes. Оскільки бот кожного користувача відрізняється лише загальною конфігурацією, інструмент шаблонізації Helm чудово підходить для створення та редагування контейнерів усередині кластеру.

Початкові вимоги до реалізації також зосереджуються на відмовостійкості алгоритму. У разі будь яких помилок на рівні контейнеру, кластеру, або навіть серверу, система повинна мати можливість відновити свою роботу з останньої зафіксованої позиції і запобігти втраті прогресу. Для реалізації вищезазначеного функціоналу до адміністративного модуля входить сервіс Velero, інструмент для резервного копіювання та відновлення додатків та об'єктів Kubernetes. Він дозволяє зручно створювати резервні копії об'єктів Kubernetes. Velero підтримує різні джерела сховищ для зберігання резервних копій, включаючи AWS S3, Google Cloud Storage, Azure Blob Storage, а також локальне сховище. Резервна копія кожного контейнеру буде створюватись раз на годину та зберігатись у хмарному сховищі Firestore bucket. За необхідності частоту копіювання можна збільшити або зменшити. Також задля запобігання розростанню об’єму сховища, застосовані інструменти управління життєвим циклом копій. Файли, що відстають більш ніж на дві версії та тривалий час не зчитуються і не модифікуються будуть автоматично видалятись зі сховища.

Поєднання кластеру алгоритму з клієнтським середовищем є не менш важливою задачею для побудови архітектури. Верхній рівень системи має являти собою прошарок між користувацьким інтерфейсом і раніше описаним

кластером. Зазвичай модулі такого типу проектують абстрактним для можливості зміни реалізації, та спрощенні залежностей між модулями. Тобто нижні рівні застосунку не мають залежати від верхніх рівнім і навпаки. Це ж стосується і найвищого рівня нашої системи – клієнтського інтерфейсу, Telegram боту. За абстрактної реалізації транспортного рівня архітектури, у майбутньому зовсім не буде проблемою змінити, або додати нові користувацькі інтерфейси, на кшталт веб-сайту чи мобільного застосунку. Поточною реалізацією абстрактного транспортного рівня виступає Executor (рисунок 3.2).

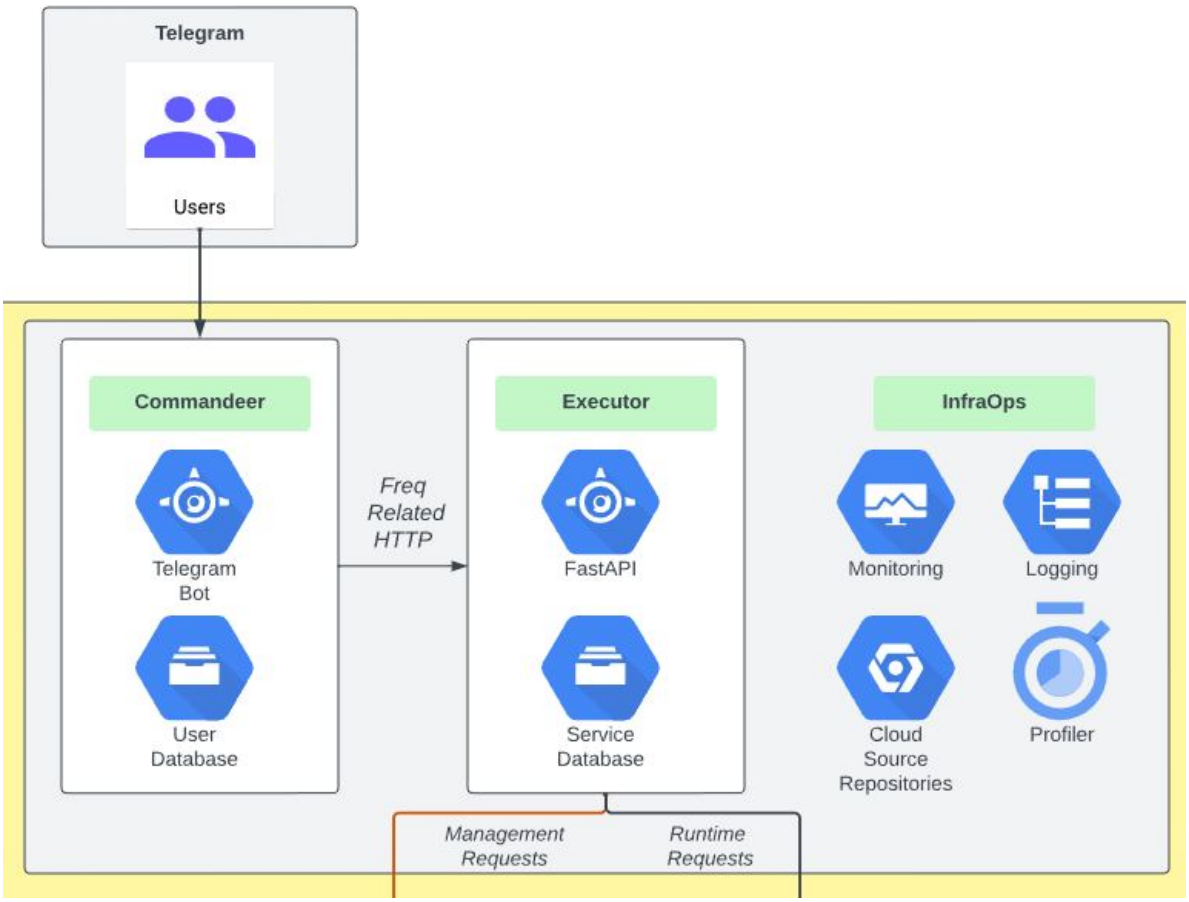


Рисунок 3.2 – Схема роботи транспортного рівня застосунку

Представлена схема є продовженням схеми з рисунку 3.2, вона демонструє архітектуру компонентів транспортного рівня застосунку. Програма контейнеру Executor відповідає за комунікацію між екземпляром Telegram боту, що розміщений у контейнері Commandeer. Обидва контейнери під’єднані до

хмарного сховища Firestore, де зберігають користувацькі дані та конфігурації. Executor отримує HTTP запити від контейнеру Telegram боту. Наступним кроком логіка застосунку має виділити ідентифікаційні дані користувача, що робить запит, і отримати з бази даних інформацію про стратегію закріплену за його акаунтом і інші конфігураційні дані. Далі отримана інформація конвертується у HTTP запит, який в свою чергу надсилається до кластеру алгоритму. Запити від Executor контейнеру розділяються на менеджмент і рантайм типи. Запити з менеджменту відповідають за конфігурацію користувацьких контейнерів, їх створення або зупинку. Рантайм запити надсилаються одразу до контейнеру користувача, та полягають у взаємодії з алгоритмом і отриманні інформативної відповіді.

Важливою задачею Executor є маршрутизація запиту користувача до відповідного контейнеру. Ця задача вирішується дуже легко, оскільки кожен контейнер алгоритму розташований у своєму просторі імен. Кожен namespace іменований унікальним ідентифікатором користувача. При отриманні запиту, модуль перевіряє наявність особистого ідентифікатора користувача у базі даних і обирає визначений маршрут. Таким чином користувач має доступ тільки до свого контейнера у системі, уся маршрутизація виконується автоматично.

## 3.2 Розробка веб-інтерфейсу користувача

Під час планування платформи було прийнято рішення розробити додатковий сервіс, який би дозволив поширювати доступ до продукту та надавати можливості адміністрування. Ключовим фактором при виборі технології та платформи для розробки стала загальна доступність для великої кількості користувачів, незалежно від девайсів та операційних систем, які вони використовують. Доступ до мережі Інтернет через браузер дозволяє переглядати веб сторінки як зі смартфонів, так і з персональних комп’ютерів, що гарантує легку доступність ресурсу і широке охоплення. Отже сервіс побудовано як веб-сайт за допомогою фреймворку Flutter. Така зв’язка

дозволить легко перенести вже готовий інтерфейс користувача на інші платформи і операційні системи.

Серед цільових призначень сайту, одним із головних є просування продукту у мережі Інтернет, можливість залучення реклами і рефералів з різних медіа. Веб-сайт містить загальну інформацію про продукт, FAQ, або ж часті запитання. За допомогою цього сервісу клієнт зможе контактувати зі службою підтримки і отримувати додаткову інформацію та відповіді на запитання. На сайті реалізована реферальна система, що дозволяє активним користувачам за запрошення клієнтів до сервісі отримувати внутрішні бонуси, які можна витрачати на додатковий функціонал торгівельного алгоритму, такий як підтримка нейронних мереж, або спеціально розроблені стратегії.

Клієнт має створити акаунт на сайті, щоб отримати доступ до торгівельного алгоритму та розроблених стратегій. Після реєстрації створюється відповідний запис у базі даних. Усі наступні дії користувача будуть зберігатися у базі даних, а його купівлі та бонуси будуть прив’язані до облікового запису.

Адміністратор платформи зможе керувати доступом користувачів до торгівельного боту, через адміністративну панель сайту. Також адміністратор матиме можливість слідкувати за активністю користувачів, нараховувати їм бонуси та контактувати у разі необхідності. Серед адміністративних функцій доступне ведення блогу, який доступний для усіх гостей сайту. Незареєстровані гості сайту зможуть переглядати спеціальні пропозиції та новини пов’язані з продуктом.

Основа функціоналу клієнтської частини застосунку, або ж frontend, побудована на спілкуванні з backend API, створеною за допомогою фреймворку ASP.NET. Спілкування між компонентами відбувається завдяки обміну HTTP повідомленнями через ApiClient (лістинг 3.4) – клас, що займається надсиланням запитів до серверу. ApiClient наслідує базовий інтерфейс BaseClient, де реалізовані класичні REST запити: GET, POST, PUT, DELTE тощо.

Лістинг 3.4 – Декларація класу ApiClient

```
class ApiClient implements BaseClient {
  final String baseUrl;
  final Dio dio = Dio();
  final Logger _logger = Logger();
  ApiClient({required this.baseUrl}) {
    dio.options.validateStatus = (int? status) {
      return status != null &&
          status < 500;
    };
    dio.interceptors.add(InterceptorsWrapper(
      onRequest: (RequestOptions options, handler) {
        _logger.d('Request body: ${options.data}');
        return handler.next(options);
      },
      onResponse: (Response response, handler) {
        _logger.d('Response text: ${response.statusMessage}');
        return handler.next(response);
      },
      onError: (DioError e, handler) {
        _logger.e('Error: ${e.message}');
        return handler.next(e);
      },
    ));
  }
```

ApiClient розширює деякі можливості базового клієнту і додає логування та обробку запитів і відповідей. Під потреби застосунку були адаптовані методи надсилання запитів. Перед відправленням контент валідується, а його тип форматується в залежності від формату даних. До midleware застосунку додаються обробники помилок, а також AuthInterceptor – клас, який модифікує вихідні запити до серверу, додаючи до заголовків, авторизаційний заголовок з токеном користувача.

AuthInterceptor взаємодіє зі TokenStorage, клас що викликається в рантаймі та займається менеджментом авторизаційних токенів користувачів. Інспектор підтримує тривалість життя токенів і надає за запитом клієнта актуальний токен за ключем. Після отримання актуального токену клієнт форматує його відповідно до правил авторизації, визначених сервером.

У клієнтській частині застосунку також визначені DTO класи призначені для обміну даними із серверу, та транспортуванню домених моделей між програмами. Такі моделі також застосовуються запису даних отриманих з форм та інших вводів веб інтерфейсу. До класів DTO моделей інтегрована первісна валідація, наприклад яка дозволяє встановити обов'язкові поля (лістинг 3.5), а також плавила конвертації з/до формату JSON (лістинг 3.6).

Лістинг 3.5 – Приклад DTO моделі

```
class ReferalLevel {
  final int id;
  final int level;
  final double reward;
  final int requiredReferralsCount;

  ReferalLevel({
    required this.id,
    required this.level,
    required this.reward,
    required this.requiredReferralsCount,
  });
```

Лістинг 3.6 – Приклад згенерованого коду для налаштування конвертації з/до формату JSON

```
ReferalLevel _$ReferalLevelFromJson(Map<String, dynamic> json) =>
ReferalLevel(
      id: json['id'] as int,
      level: json['level'] as int,
      reward: (json['reward'] as num).toDouble(),
      requiredReferralsCount: json['requiredReferralsCount'] as
int);

Map<String, dynamic> _$ReferalLevelToJson(ReferalLevel instance)
=>
    <String, dynamic>{
      'id': instance.id,
      'level': instance.level,
      'reward': instance.reward,
      'requiredReferralsCount': instance.requiredReferralsCount};
```

Програма оброблює відповіді від серверу, трансформує їх у транспортні

моделі, та за необхідності підготовлює до виведення на інтерфейс користувача. UI також розробляється на мові програмування dart. У більшості випадків використовуються базові віджети та візуальні елементи пакету material.dart. Застосунок використовує сховище статичних об’єктів на сервері, таких як зображення, статичні сторінки та тексти. Така методика пришвидшує завантаження сторінок і сприяє покращенню досвіду користування сервісом.

## 3.3 Розробка API для адміністрування платформи

Для розробки API веб-застосунку обрану фреймворк ASP.NET та мову програмування C#. Кодова база, побудована за використання актуального фреймфорку .Net 8, забезпечує високу продуктивність і широкий спектр інструментів для побудови бізнес логіки застосунку. Особлива увага приділяється безпеці, захисту сервісу від шкідливих атак і потенційних загроз. API пропонує зручні механізми авторизації та аутентифікації користувачів, безпечний транспорт та зберігання даних.

При розробці використаний сучасний підхід REST API – архітектурний стиль розробки веб-сервісів, що базується на принципах простоти, легкості, використання, розширюваності та безпеки. REST API підтримує широкий набір стандартів безпеки, таких як HTTPS, OAuth, JWT, що дозволяє забезпечувати безпеку даних та аутентифікацію користувачів. REST API підтримує кешування, що дозволяє зберігати копії ресурсів на клієнтській стороні для покращення продуктивності та зменшення навантаження на сервер.

Застосунок має класичну тришарову архітектуру [26], що складається з шару доступу до даних, шару бізнес-логіки та репрезентаційного шару, безпосередньо самого API. Такий підхід є зручним для розділення компонентів програми, що впроваджує модульність та розширюваність. Розділення дозволяє розробляти кожен модуль незалежно від інших, це спрощує модифікацію коду та внесення змін у всю систему. Такий підхід дозволяє використовувати одну кодову базу для різних мікро сервісів, сприяє чистоті коду і уникненню

взаємозалежності між компонентами системи. Також при тришаровій архітектурі дозволяє впровадити модульне тестування та виявлення помилок логіки застосунку на ранніх етапах. Розділення бізнес-логіки та доступу до даних дозволяє керувати доступом до ресурсів та забезпечувати безпеку даних шляхом встановлення правил доступу на кожному рівні.

Взаємодія різних модулів реалізована завдяки інструменту керування залежностями Dependency Injection (DI). Реалізовані інтерфейси для опису контрактів сервісів програми, які завдяки механізму впровадження залежностей передаються до конструкторів конкретних реалізацій. Контейнери інверсії керування залежностями дозволяють налаштувати життєвий цикл реалізацій, що усуває необхідність у додатковому менеджменті використання ресурсів серверу. Завдяки DI можна легко змінювати реалізації залежностей або додавати нові, що полегшує розвиток та підтримку програми.

Транспортування даних також є важливим аспектом у взаємодії модулів. При тришаровій архітектурі кожен модуль має власні моделі уявлення даних. Для забезпечення конвертацій з типів між модулями використовується бібліотека AutoMapper. Вона забезпечує автоматичне перенесення та трансформацію полів від одного представлення до іншого. Інструмент підтримує роботу зі складними багаторівневими вкладеними об’єктами та колекціями.

Доступ до даних (DAL) реалізований за допомогою бібліотеки Entity Framework Core (EF Core) – об'єктно-реляційна маппінгова (ORM) технологія, що надає зручний спосіб взаємодії з базами даних у .NET застосунках. EF Core дозволяє працювати з даними в базі даних у вигляді об'єктів, замість прямої роботи з SQL запитами. У якості системи управління базами даних (СУБД) обраний SQL Server. База даних побудована за допомогою Code First підходу, коли усі доменні моделі та зв’язки між ними декларовані класами C# та описані у DbContext, який представляє собою взаємодію з базою даних та включає в себе набір DbSet властивостей для доступу до різних сутностей. Code First полегшує управління структурою даних шляхом впровадження міграцій, що

дозволяє швидко розгортати та змінювати схеми бази даних. При старті застосунку контекст ініціалізується та, під час під'єднання до SQL серверу, порівнюється зі схемою бази даних. Якщо база даних не існує, або відрізняється від останньої актуальної міграції, програма автоматично створює задекларовану базу даних та запускає останній скрипт міграції. Це гарантує актуальність структури даних та відповідність до кодової бази.

Для взаємодії з сутностями, таблицями у базі даних, реалізований загальний репозиторій. Репозиторій дозволяє виконувати численні операції над об'єктами різних типів. Оскільки EF Core підтримує Language Integrated Query (LINQ), що дозволяє писати структуровані запити до бази даних без прямого використання SQL. Це полегшує написання запитів та забезпечує безпеку типів. При запиті на вибірку чи модифікацію об'єкту бази даних, репозиторій взаємодії з контекстом даних, дістає типизований DbSet, та виконує необхідну операцію (лістинг 3.7).

Лістинг 3.7 – Методи загального репозиторію

```
public Task<List<T>> GetListAsync(Expression<Func<T, bool>>?
expression = default,
     Func<IQueryable<T>, IQueryable<T>>? include = default,
CancellationToken cancellation = default)
 {
     IQueryable<T> query = _dbSet;

     if (include != null)
         query = include(query);

     if (expression != null)
         query = query.Where(expression);

     return query.ToListAsync(cancellation);
 }
public IQueryable<T> GetList() => _dbSet;
```

Усі запити до бази даних є асинхронними та потокобезпечними, це знижує навантаження на програму та гарантує безпеку у роботі з даними. EF Core надає підтримку транзакцій та безпеки, що дозволяє виконувати атомарні

операції з даними та забезпечувати консистентність даних. На базі цих інструментів EF Core реалізовано паттерн Unit Of Work, для забезпечення забезпечення цілісності даних та атомарності операцій. Unit of Work дозволяє групувати декілька операцій з базою даних у межах однієї транзакції. Це дозволяє забезпечити консистентність даних та уникнути ситуацій, коли певні зміни в базі даних виконуються напівзавершено. Паттерн дозволяє ізолювати операції з базою даних від решти програми, також зберігає стан об'єктів даних у відповідності зі станом бази даних. Це дозволяє уникнути ситуацій, коли дані у пам'яті програми не відображають дійсний стан бази даних. Методи інтерфейсу IUnitOfWork (лістинг 3.8) дозволяють розпочинати та підтверджувати транзакції, впроваджувати зміни, або ж відміняти їх у разі помилок.

Лістинг 3.8– Методи інтерфейсу IUnitOfWork

```
Task SaveAsync(CancellationToken cancellationToken = default);
 Task BeginTransactionAsync(CancellationToken cancellationToken =
default);
 Task RollbackTransactionAsync(CancellationToken cancellationToken
= default);
 Task CommitTransactionAsync(CancellationToken cancellationToken =
default);
```

Реалізовано окремий репозиторій для роботи з нереляційною базою даних Firestore. Цей репозиторій використовується для запису унікальних ключів доступу користувачів до торгівельного алгоритму на платформі Telegram. Коли користувач закріплює за своїм акаунтом стратегію підтримки ринкових рішень, він отримує унікальний ключ доступу. Цей ключ у зашифрованому вигляді додається до документу, що додатково містить час активації, та термін дії цього доступу. Документ зберігається у сховищі, звідки може бути зчитаний модулем торгівельного алгоритму.

Рівень бізнес-логіки розгалужений відповідно до схеми доменних сутностей. Кожна схема має власні сервіси що містять логіку роботи з цією схемою. Наприклад, схема Account містить наступні сервіси:

- AuthenticationManager;
- ReferralLevelService;
- UserService.

Розглянемо складову User сервісу. Він відповідає за виконання задач пов’язаних з об’єктами, які представляють сутність користувача сайту. Сервіс виконує завдання по маніпуляціям і транспорту даних зі схеми User, викликає методи репозиторію на вибірку користувачів із бази даних за різними параметрами, впроваджує сортування і фільтрацію (лістинг 3.9). Також взаємодіє з UserManager сервісом, інструментом менеджменту користувачів бібліотеки ASP.NET Core Identity. UserManager надає механізми для аутентифікації користувачів за допомогою різних схем, можливості створення, редагування, видалення користувачів, встановлення та зміни їхніх ролей та інших атрибутів, а також зберігання інформації про користувачів, таку як ім'я, адреса електронної пошти, пароль (зашифрований), ролі тощо.

Лістинг 3.9 – Методи інтерфейсу IUserService

```
    Task<IdentityResult> DeleteAsync(string id);
    Task<IdentityResult> HardDeleteAsync(string id);
    Task<IEnumerable<UserDto>> GetListAsync(string keyword,
CancellationToken cancellation = default);
    Task<IEnumerable<UserDto>> GetAdminsAsync();
    Task<UserDto> GetAsync(string id, CancellationToken
cancellation = default);
    Task<IdentityResult> ChangeBalanceAsync(string id, double
sum);
    Task<IEnumerable<UserDto>> GetReferralsAsync(string id,
CancellationToken cancellation = default);
```

Модуль бізнес-логіки містить імплементації мейл-сервісів. Платформа потребує механізм зв’язку з користувачами, зокрема можливість надсилати електронні листи їм на пошту. В першу чергу такий функціонал вирішує проблему верифікації користувачів, активації акаунту та зміни паролю. За необхідності надає інструмент виконання двохфакторної аутентифікації. Імплементовано два сервіси MailService та NotificationService. MailService являє

собою абстракцію для підключення різних провайдерів електронної пошти. Завдяки абстрактній реалізації сервіс не залежить від обранного провайдера і дозволяє змінювати налаштування, навіть у рантаймі, необхідно лише змінити конфігурацію. Цей сервіс містить лише один метод SendEmailAsync, який приймає адресу отримувача, тему та контент листа. NotificationService займається побудовою об’єктів електронних повідомлень. Сервіс дістає зі статичного сховища шаблони повідомлень та наповнює їх відповідною інформацією. Наразі цей сервіс використовується для відправлення коду верифікації для зміни паролю, а також для надсилання ключа доступу до торгівельного алгоритму користувачу (лістинг 3.10).

Лістинг 3.10 – Код методу відправки ключа доступу до продукту

```
var template = Template.Parse(notificationTemplate!.Template);
 var resources = _configuration.GetSection("SiteResources");
 var support = resources.GetSection("SupportEmail").Value;
var data = new
 {
     KeyCode = key.ToString(),
     BotUrl = botUrl,
     Support = support,
     Date = DateTime.Today.Year,
     Product = productTitle,
     EndDate = subscriptionEndDate.ToString("yyyy-MM-dd")
 };
 var message = template.Render(Hash.FromAnonymousObject(data));
 await _mailService.SendEmailAsync(receiver,
notificationTemplate.Title, message, true, cancellation);
```

Аутентифікація користувачів основана на JwtBearer, middleware для ASP.NET Core, який надає підтримку для аутентифікації користувачів за допомогою токенів JWT (JSON Web Tokens). Цей middleware дозволяє захищати застосунки ASP.NET шляхом перевірки та перевірки цифрового підпису токенів JWT, що передаються в HTTP заголовку або в запиті. JwtBearer використовує цифровий підпис для перевірки цілісності та автентичності токенів JWT, а також для розкодування їх для отримання інформації про клейми. Токен генерується через симетричний шифр та ключ заданий у

конфігурації. Після аутентифікації створюються клейми користувача, які включають ролі, імейл, логін та унікальний ідентифікатор.

Застосунок має декілька рівнів доступу до платформи. Веб-інтерфейс також розділений на користувацьку та адміністративну частини, доступ до кожної надається в залежності від ролі користувача. Наразі у системі передбачено три ролі:

- SuperAdmin;
- Admin;
- User.

Авторизація виконується інструментами бібліотеки AspNetCore.Identity. Кожен контролер, або його методи можуть мати авторизаційний атрибут, що декларує ролі необхідні для доступу до контенту. При кожному запиті, компонент авторизації перевіряє клейми користувача на відповідність до вимог запитуваного ресурсу.

Система має ряд інструментів аналітики, моніторингу та відмовостійкості. Таким чином в middleware було додано оброблювач помилок, який маршрутизує необроблені помилки системи до Error контролеру. У цьому контролері помилки логуються та аналізуються, після чого повертається відповідний HTTP статус код, якщо такий передбачений, або статус 500, у випадку коли помилка невідома. Для перевірки загального стану системи додано middleware HealthChecks. Цей механізм дозволяє звернутись за до ендпоінту /health, який повертає статус код 200 у разі справності компонентів системи.

Логування реалізовано за використання бібліотеки Serilog. Він дозволяє легко створювати структуровані логи за допомогою розширеного синтаксису логування. Serilog підтримує різноманітні назначення логів, включаючи консоль, файли, бази даних, поштові сервіси та багато інших. Це дозволяє легко налаштовувати логування згідно з потребами застосунку. Реалізовано два види логування: файл і консоль. Логування у консоль дозволяє полегшити процес виявлення помилок під час розробки і дебагу.

Задля поглибленого аналізу і моніторингу поведінки системи та її користувачів додано Application Insights Telemetry [27] – компонент платформи Azure, який дозволяє збирати, аналізувати та візуалізувати дані про продуктивність та поведінку застосунків в реальному часі. Application Insights надає широкі можливості для моніторингу продуктивності застосунку, включаючи швидкість завантаження сторінок, час відгуку сервера, використання ресурсів та інші метрики. Компонент автоматично відстежує помилки та винятки, які виникають у застосунку, та надає детальну інформацію про них, що дозволяє швидко виявляти та виправляти проблеми. Завдяки цьому інструменту з'являється можливість відстежувати користувацькі сесії та взаємодію з веб-сайтом, що дозволяє аналізувати поведінку користувачів та вдосконалювати дизайн та функціональність застосунку.

У якості документації API виступає Swagger інтерфейс [28]. Інструменти Swagger дозволяють автоматично створювати документацію для API на основі анотацій коду. Swagger надає веб-інтерфейс (рисунок 3.3), який дозволяє користувачам взаємодіяти з API безпосередньо з браузера, без необхідності реалізації клієнт програми. Це дозволяє швидко та зручно тестувати та розуміти функціонал API. Також Swagger надає можливість аутентифікації за допомогою токену і пропонує користувачу інструкції на веб-інтерфейсі.

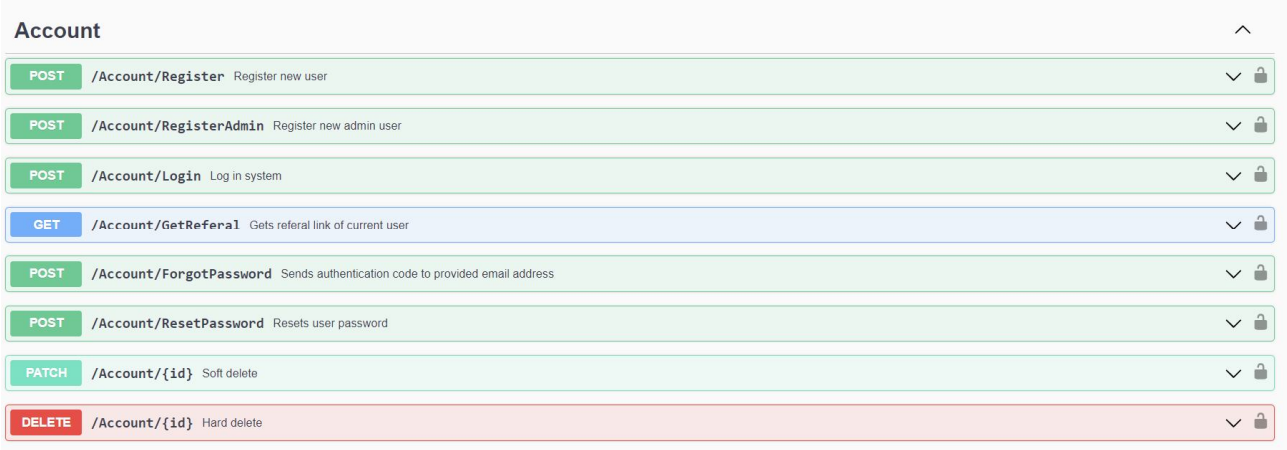


Рисунок 3.3 – Інтерфейс Swagger на прикладі Account контролеру

# 4 ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

## 4.1 Реєстрація користувача на платформі та отримання доступу до сервісу підтримки ринкових рішень

Для початку використання розроблених та протестованих торгових стратегій користувач має створити акаунт та авторизуватись на веб-сайті продукту. Для реєстрації на платформі необхідно натиснути кнопку Account, як показано на рисунку 4.1.

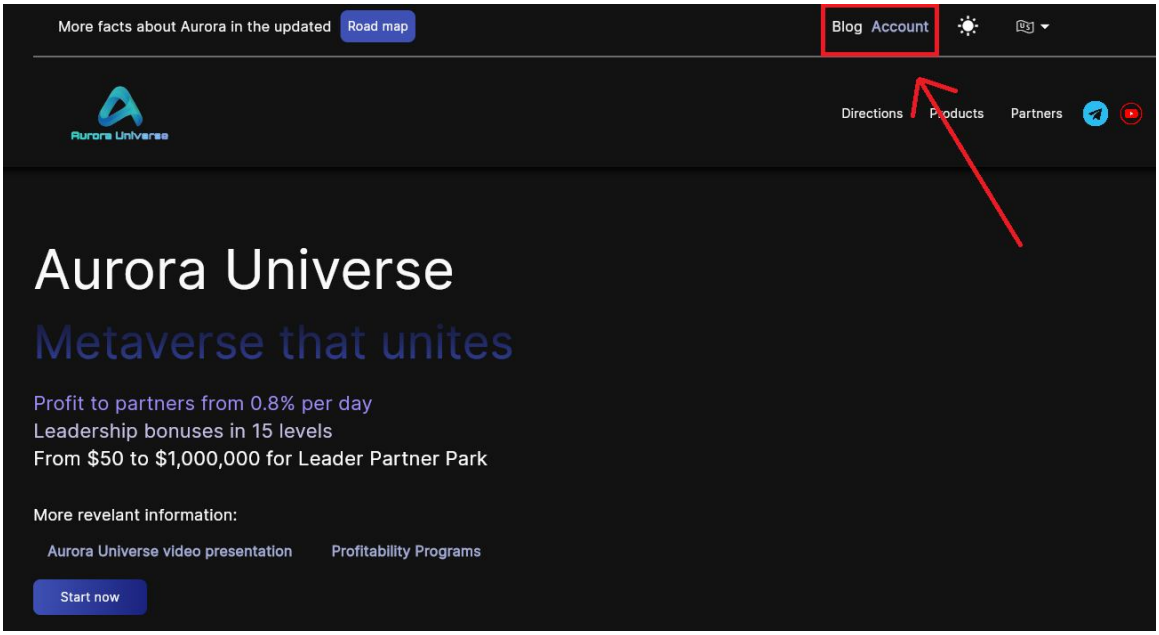


Рисунок 4.1 – Домашня сторінка веб-сайту

Далі користувач потрапляє на сторінку входу у систему. Користувачу пропонується заповнити форму авторизації, якщо він вже має акаунт, або створити новий натиснувши кнопку Sign Up у лівому нижньому куті форми (рисунок 4.2). Після натискання, відкривається форма реєстрації на платформі. Користувачеві пропонується внести реєстраційні дані до форми (рисунок 4.3). Логін може складатися з латинських символів та цифр.

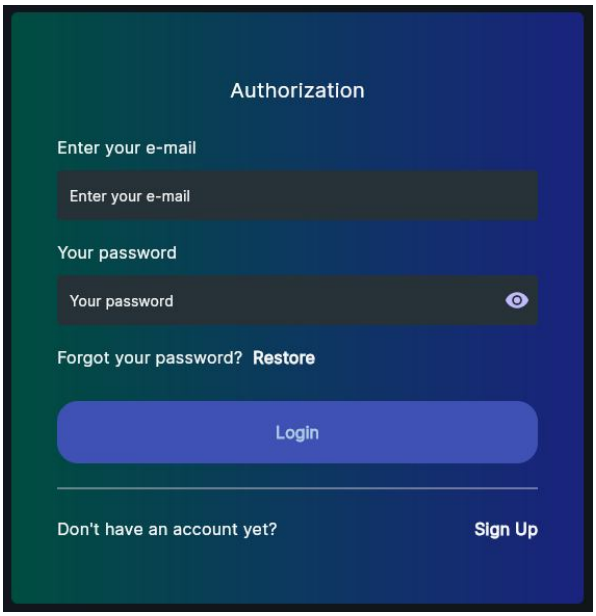


Рисунок 4.2 – Форма авторизації

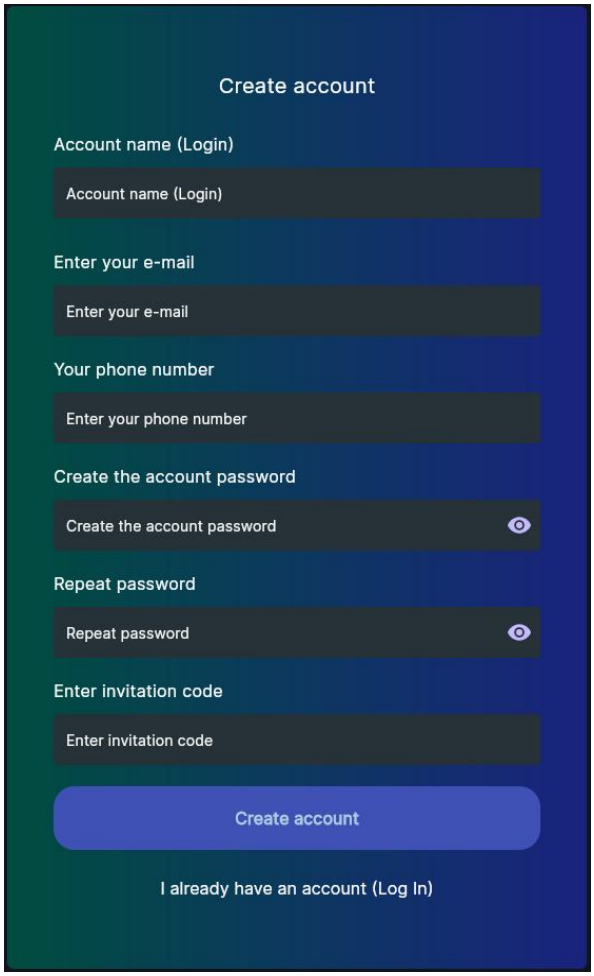


Рисунок 4.3 – Форма реєстрації

Після заповнення форми необхідно натиснути кнопку Create account, далі на екрані має з’явитись каптча для валідаціїї користувача (рисунок 4.4). Необхідно, використовуючи слайдер, скласти пазл у цілісне зображення.


Рисунок 4.4 – Captcha

Виконавши вхід, користувач потрапляє на домашню сторінку (рисунок 4.5), де може переглядати інформацію стосовно свого партнерського рівня і доступних бонусів.

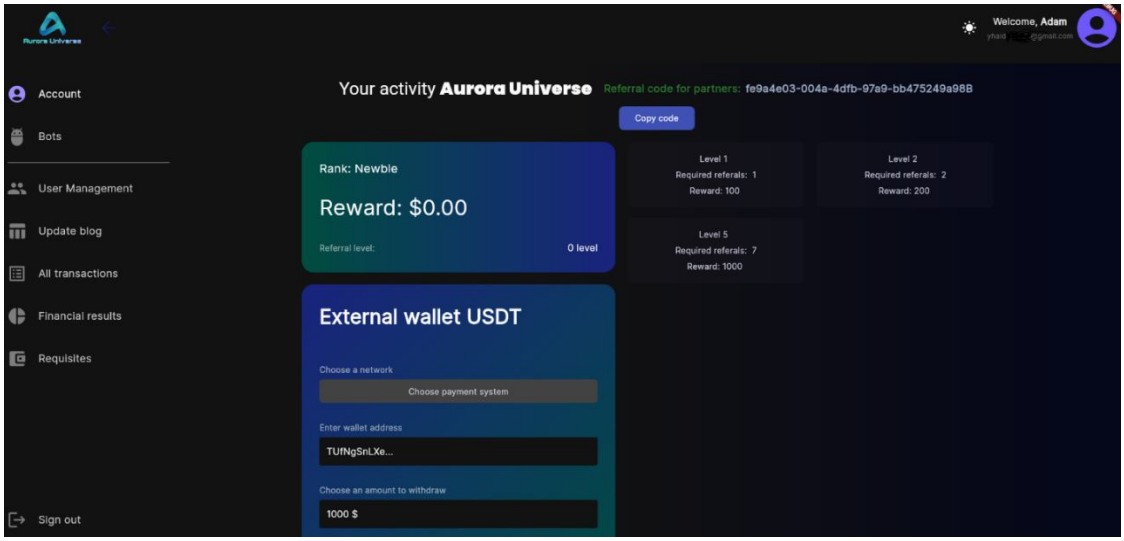


Рисунок 4.5 – Домашня сторінка акаунту користувача

Для перегляду доступних стратегій необхідно обрати розділ під назвою Bots на панелі навігації праворуч (рисунок 4.5). Сайт навігує на сторінку перегляду доступних ботів (стратегій). На сторінці можна переглянути опис кожної стратегії та таблицю порівняння (рисунок 4.6).

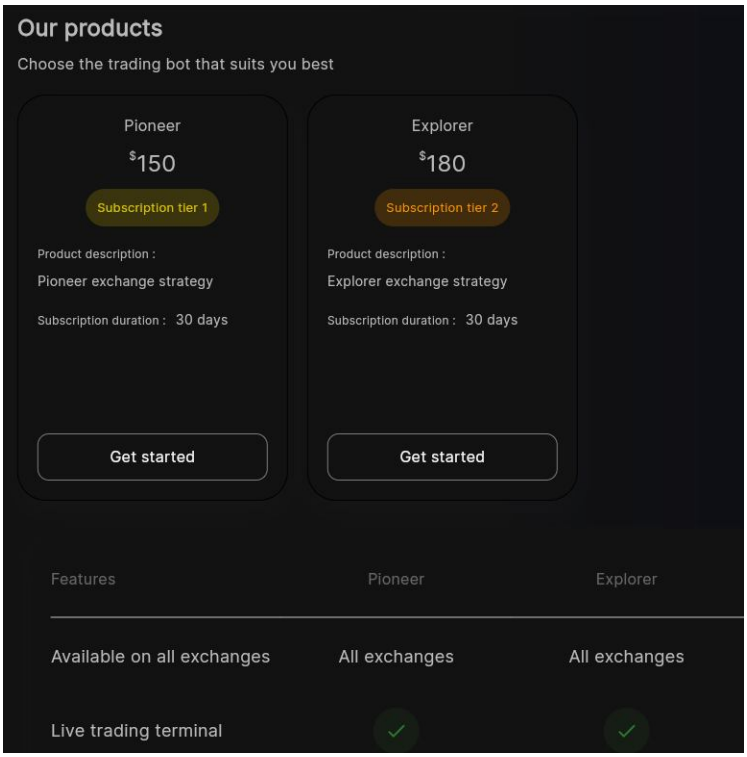


Рисунок 4.6 – Список торгових стратегій

Ознайомившись з перевагами кожної стратегії, користувач має натиснути кнопку Get Started під обраною щоб перейти на форму вибору платіжної системи (рисунок 4.7). На цій формі необхідно обрати одну з пропонованих систем. Після вибору натиснути кнопку Confirm, що відкриє форму активації підписки (рисунок 4.8).

Процес активації підписки може зайняти від 1 до 5 хвилин після переказу коштів. Коли підписку активовано, користувач має отримати електронного листа з ключем доступу до продукту (рисунок 4.9). Цей ключ знадобиться вже на наступному етапі активації боту у телеграмі.

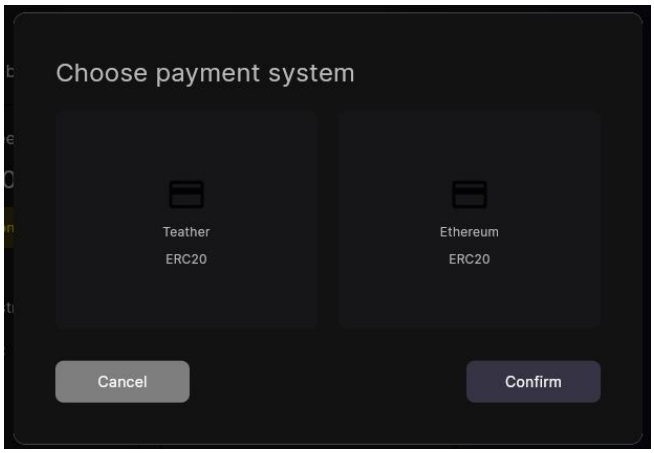


Рисунок 4.7 – Форма вибору платіжної системи

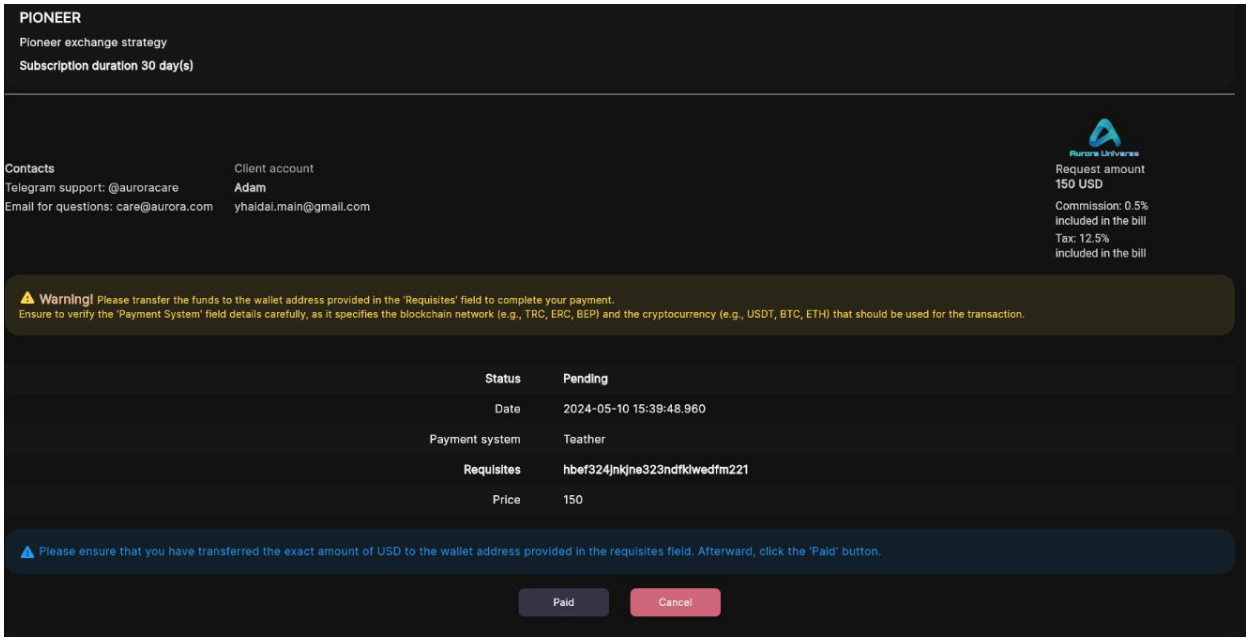


Рисунок 4.8 – Форма активації підписки

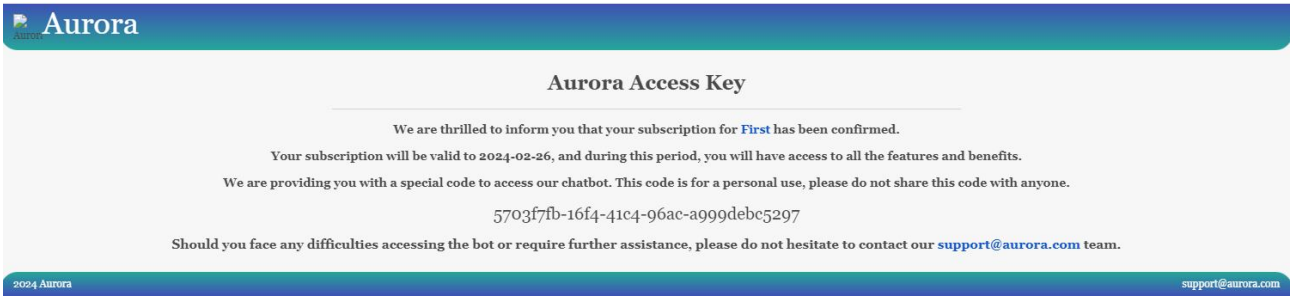


Рисунок 4.9 – Електронний лист з ключем активації продукту

## 4.2 Налаштування та користування сервісом через інтерфейс користувача

Використовуючи, посилання з електронного листа, користувач має можливість одразу перейти до Telegram боту. Для старту боту можна скористатись інтерактивним меню, або надіслати команду /start. Далі буде запитано ключ доступу до продукту (рисунок 4.10), який теж знаходиться в електронному листі.

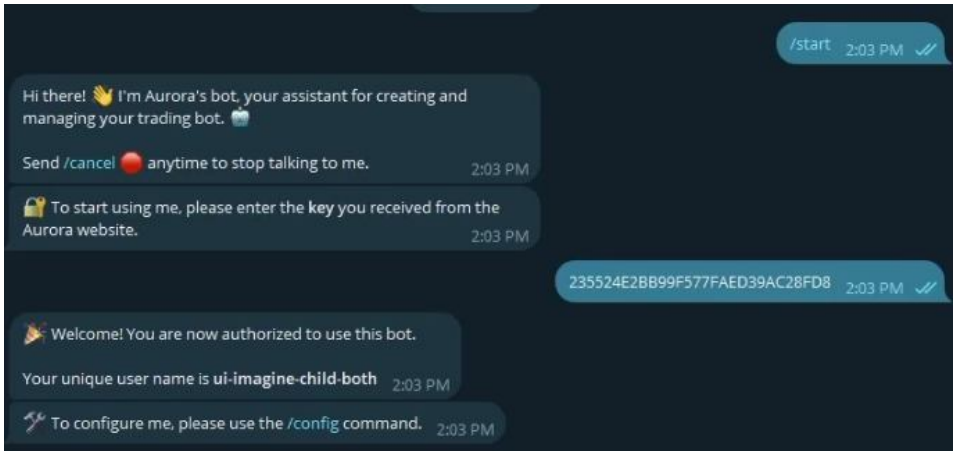


Рисунок 4.10 – Старт роботи з торгівельним ботом

Наступним кроком буде налаштування алгоритму, для цього необхідно скористатись інтерактивним меню, або надіслати команду /config. Першим етапом налаштування є вибір біржи для інтеграції (рисунок 4.11).


Рисунок 4.11 – Вибір біржи для роботи алгоритму

Після вибору біржі достатньо послідовно виконувати інструкції налаштування, на деяких етапах буде запропоновано перейти за посиланням для отримання додаткових інструкцій (рисунок 4.12).

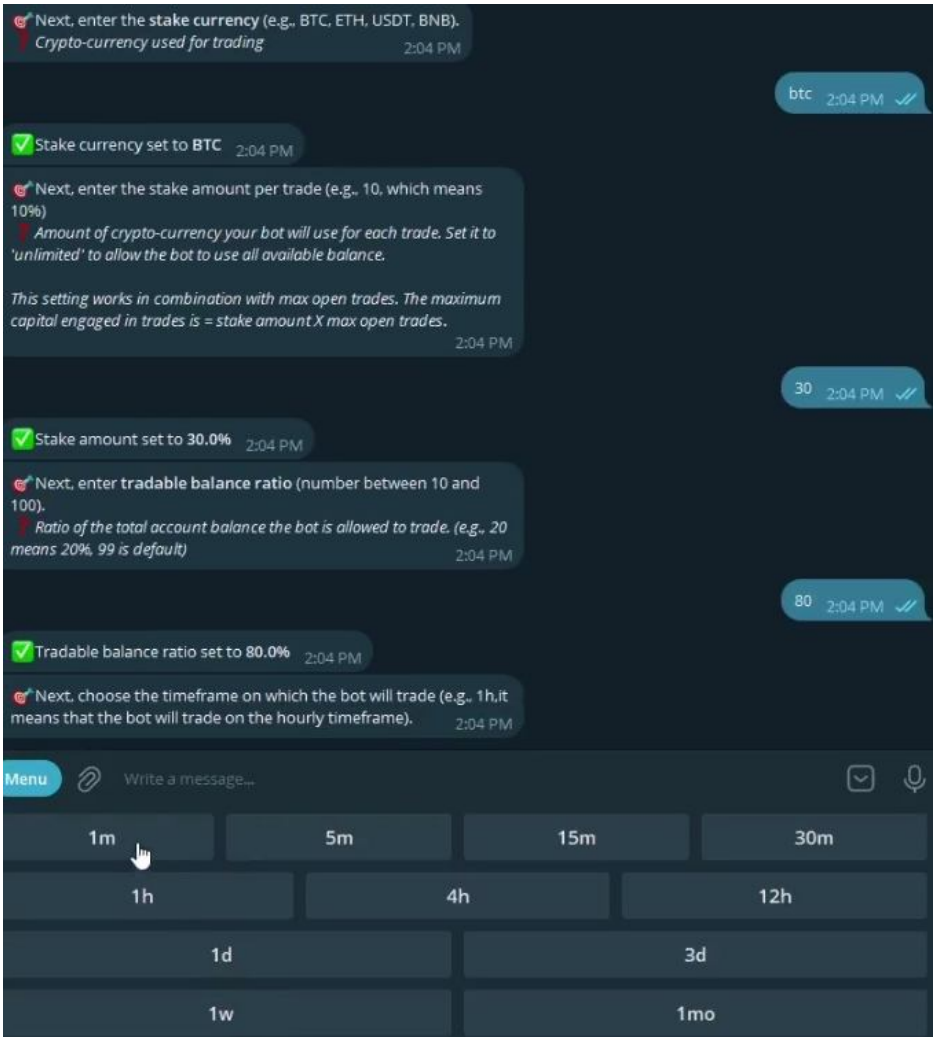


Рисунок 4.12 – Процес налаштування алгоритму

По завершенні налаштування на екран буде виведено загальну інформацію за підсумками конфігурації. Запуск алгоритму може зайняти до однієї хвилини. Бот повідомить про готовність до роботи відповідним повідомленням у чаті (рисунок 4.13).

Для взаємодії з ботом надано інтерактивне меню (рисунок 4.14). Там можна ознайомитись з прогресом, зупинити алгоритмом, переглянути історію та змінити конфігурації.

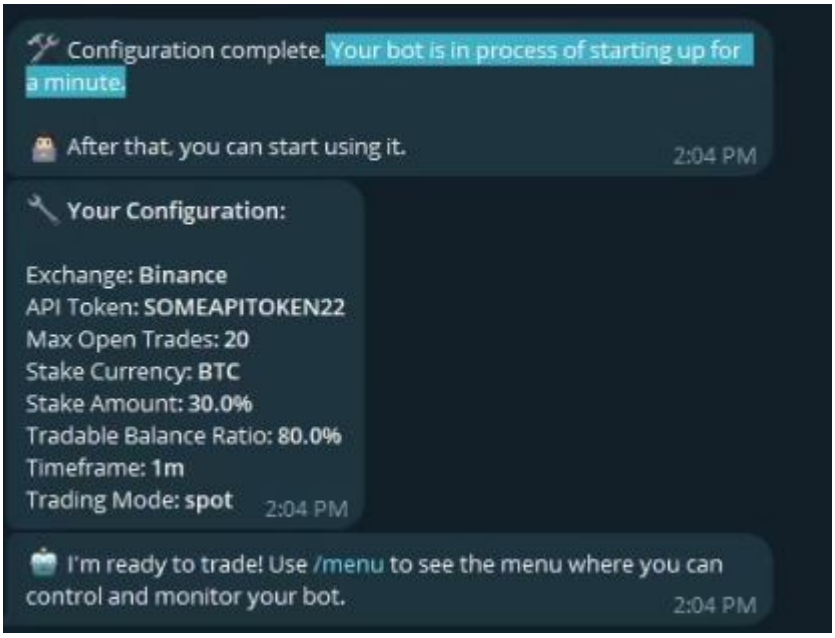


Рисунок 4.13 – Повідомлення про завершення конфігурації

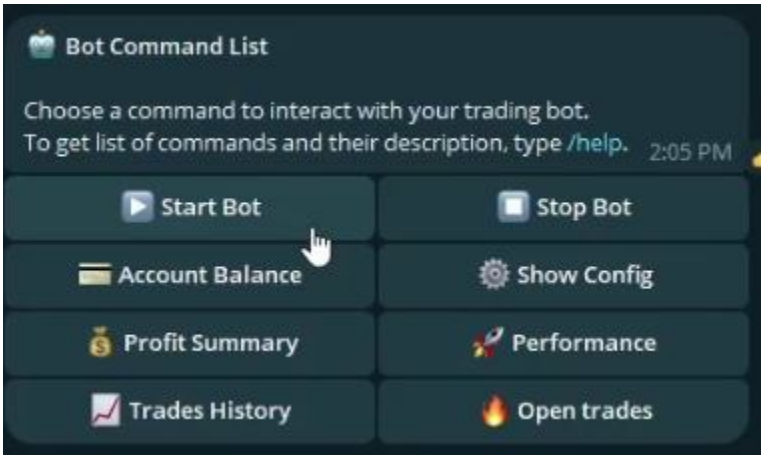


Рисунок 4.14 – Інтерактивне меню бота

Для зупинки алгоритму необхідно натиснути кнопку Stop Bot. Це призведе до закриття усых відкритих замовлень та позицій по ринковій вартості. Користувач може переглянути відкриті позиції натиснувши кнопку Open trades.

# ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розглянуто актуальність потреби завдання розробки алгоритму підтримки ринкових рішень, розглянуті існуючі методи антиципації ринкових рухів та готові програмні рішення. Зроблено аналіз технологій передбачення ринкових рухів із впровадженням нейронних моделей зворотного поширення помилки.

У роботі наведено огляд сучасних інструментів розробки і технологій для інтеграції з валютними біржами. Також було обґрунтовано обрані технології для розробки кінцевого продукту.

Виконано розробку архітектури програмного алгоритму та надано опис його технічної реалізації. Наведено деталі технічної реалізації платформи для адміністрування застосунку. Зроблено опис інтерфейсу користувача основного продукту та веб-інтерфейсу, показано процес використання застосунку та веб-сайту.

Розроблений програмний продукт, що дозволяє створювати стратегії валютної торгівлі для підтримки ринкових рішень. Для збільшення ефективності застосунок дозволяє впровадження аналітичних нейронних мереж. Для адміністрування доступу користувачів до продукту та його просування розроблено веб-сайт, де користувачі можуть обирати перевірені та протестовані торгівельні стратегії. Впроваджені сервіси хмарних обчислень для поглибленої аналітики та моніторингу стану і використання застосунку.

Використання розробленого продукту показало задовільні результати у роботі з популярними крипто-біржами. Фінальний результат цілком задовольняє потреби користувача та надає зручний інструмент підтримки ринкових рішень.

Результати досліджень кваліфікаційної роботи опубліковано у збірнику наукових праць [29].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Karathanasopoulos (2016). “Stock market anticipation utilizing evolutionary support vector machines: an application of the ASE20 index”, The European Journal of Finance

2. M.T. Leung et al. “Forecasting stock indicesa comparison of classification and level estimation models”, International Journal of Forecasting (2000)

3. Rosillo (2014), “Stock market simulation utilizing support vector machines. Journal of Forecasting”

4. F.E.H. Tay et al. “Application of support vector machines in financial time series forecasting”, Omega (2001)

5. Studies, E. (2000). A Stock Market Prediction Method Based on Support Vector Machines and Independent Component Analysis

6. Prasad Das (2012). Support Vector Machines at Prediction of Futures Costs in Indian Stock Market. International Journal of Computer Applications.

7. Y.S. Abu-Mostafa et al. “Introduction to financial forecasting”, Applied Intelligence (1996)

8. Ichinose, K., & Shimada, K. (2016), “Stock market prediction from news on the web and a new evaluation approach in trading. Proceedings” 2016 5th IIAI International Congress on Advanced Applied Informatics

9. Telegram Bot API / Telegram – Режим доступу до ресурсу: https://core.telegram.org/bots/api

10. Flutter documentation / Flutter – Режим доступу до ресурсу: https://docs.flutter.dev/

11. Relational vs. Non-Relational Databases / MongoDB – Режим доступу до ресурсу: https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases

12. What is Microsoft SQL Server? / TechTarget – Режим доступу до ресурсу: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server

13. What is PostgreSQL? / PostgreSQL – Режим доступу до ресурсу: https://www.postgresql.org/about/

14. NoSQL Databases / TrustRadius – Режим доступу до ресурсу: https://www.trustradius.com/nosql-databases

15. Firestore documentation / Google Cloud – Режим доступу до ресурсу: https://cloud.google.com/firestore/docs

16. Binance API Product Page Guidance / Binance – Режим доступу до ресурсу: https://www.binance.com/en/support/faq/binance-api-product-page-guidance-865f0fe3cb6a4d73a21609b3b7326f31

17. Freqtrade library documentation /Freqtrade – Режим доступу до ресурсу: https://www.freqtrade.io/en/stable/

18. About Gekko / Gekko – Режим доступу до ресурсу: https://gekko.wizb.it/docs/introduction/about_gekko.html

19. Cryptocurrency Ownership Data / Triple-A – Режим доступу до ресурсу: https://triple-a.io/cryptocurrency-ownership-data/

20. Криптовалютами володіє 6 мільйонів українців — дослідження / Мінфін – Режим доступу до ресурсу: https://minfin.com.ua/ua/2023/02/02/100080832/

21. Які мобільні додатки є найбільш популярними в Україні? / Київський міжнародний інститут соціології – Режим доступу до ресурсу: https://kiis.com.ua/?lang=ukr&cat=reports&id=1027

22. Cloud computing with AWS / Amazon – Режим доступу до ресурсу: https://aws.amazon.com/what-is-aws/?nc2=h_ql_le_int

23. Kubernetes Documentation / Kubernetes – Режим доступу до ресурсу: https://kubernetes.io/docs/home/

24. What is Azure? / Microsoft – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure

25. What is Helm in Kubernetes? / Sysdig – Режим доступу до ресурсу: https://sysdig.com/learn-cloud-native/kubernetes-101/what-is-helm-in-kubernetes/

26. What is Three-Tier Architecture / IBM – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/three-tier-architecture

27. Application Insights overview / Microsoft – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview

28. Swagger documentation / Swagger – Режим доступу до ресурсу: https://swagger.io/docs/

29. Гайдай Я. А. Аналіз методу опорних векторів у порівнянні з традиційними методами передбачення ринкових рухів / Я. А. Гайдай, С. Я. Бовчалюк // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2024. – Випуск 3 (77). – С. 52-55. doi: 10.26906/SUNZ.2024.3.052.