Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення онлайн-платформи для валідації торгових стратегій на основі мультиагентної симуляції фінансового ринку

Виконав: студент IV курсу, групи СП-41
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Бутрин Н. В.
(підпис) (прізвище та ініціали)

Керівник _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту

1. Тема роботи Розробка програмного забезпечення онлайн-платформи для валідації торгових стратегій на основі мультиагентної симуляції фінансового ринку

Керівник роботи Стоянов Юрій Миколайович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від « 06 » квітня 2026 року №4/9-169

2. Термін подання студентом роботи 22.06.2026

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз вимог та огляд предметної області.
2. Проєктування та конструювання
3. Тестування.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги. 5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку.
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	
2.	*Робота над першим розділом «Огляд предметної області»*	*13.04 – 26.04*	
3.	*Робота над другим розділом «Проєктування та реалізація»*	*27.04 – 03.05*	
4.	*Робота над третім розділом «Тестування»*	*04.05 – 17.05*	
	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	*18.05 – 24.05*	
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	
6.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
7.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
8.	*Захист кваліфікаційної роботи бакалавра*		

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бутрин Н. В. Розробка веб-платформи для валідації торгових стратегій з використанням мультиагентної симуляції: робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 – Інженерія програмного забезпечення / наук. кер. Ю. М. Стоянов. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 50 с. // С. – 64, табл. – 5, рисунку – 17, бібліогр. – 32, слайд. – 0, додат. – 2.

Ключові слова: алгоритмічна торгівля, бектестинг, мультиагентна симуляція, LangGraph, FastAPI, Grid-DCA, штучний інтелект, React, PostgreSQL.

Метою роботи є розробка програмного забезпечення онлайн-платформи Validex, яка забезпечує високу точність валідації торгових алгоритмів через поєднання класичного бектестингу та інтелектуальної мультиагентної оптимізації параметрів стратегій.

У першому розділі проведено аналіз предметної області алгоритмічної торгівлі, розглянуто існуючі платформи (MetaTrader 5, TradingView, QuantConnect, 3Commas, Cryptohopper), виявлено їхні обмеження та обґрунтовано необхідність розробки нового рішення.

У другому розділі описано процес проєктування та реалізації системи: визначено функціональні вимоги до валідатора та ШІ-компонента, спроєктовано архітектуру мікросервісної платформи, розроблено двигун бектестингу для стратегій Grid-DCA та мультиагентну систему оптимізації на базі LangGraph.

У третьому розділі описано проведення функціонального тестування всіх компонентів системи, наведено результати перевірки застосунку та компонента обміну даними, підтверджено відповідність реалізації висунутим вимогам.

У четвертому розділі розглянуто питання забезпечення безпечних умов праці та охорони праці оператора комп’ютерних систем.

Об'єктом дослідження є процес валідації та оптимізації торгових стратегій на фінансових ринках за допомогою програмних засобів симуляції.

Предметом дослідження є методи, моделі та технології розробки веб-платформи для валідації торгових стратегій з використанням мультиагентної симуляції. Методи дослідження включають: аналіз конкурентних систем, мікросервісне проєктування архітектури, реалізацію стейт-графа ШІ-агента та функціональне тестування компонентів.

ABSTRACT

Nazar Butrun. Development of a web platform for trading strategy validation using multi-agent simulation : bachelor thesis : specialty 121 "Software Engineering" / scientific supervisor Yu. M. Stoyanov. Ternopil : Ternopil Ivan Puluj National Technical University, 2026, 50 p. // Pages – 64, tables – 5, figures – 17, references – 32, presentation slides – 0, appendices – 2.

Keywords: algorithmic trading, backtesting, multi-agent simulation, LangGraph, FastAPI, Grid-DCA, artificial intelligence, React, PostgreSQL.

The purpose of the work is to develop the Validex online platform software that provides high-accuracy validation of trading algorithms through a combination of classical backtesting and intelligent multi-agent parameter optimization.

This paper demonstrates the process of designing, developing, testing, and deploying a web platform for algorithmic trading strategy validation. The system is implemented as a microservice application consisting of a FastAPI backend with LangGraph-powered AI optimization, a React-based frontend with interactive charting, and a PostgreSQL database. An analysis of existing platforms (MetaTrader 5, TradingView, QuantConnect, 3Commas) is conducted, their limitations are identified, and a development plan is established. The core backtesting engine supports Grid-DCA strategies with leverage and margin handling for SPOT and FUTURES markets. The multi-agent optimization system based on LangGraph and Google Gemini iteratively analyses backtest reports and automatically generates improved parameter configurations with real-time streaming of the AI decision process.

The object of the study is the process of validation and optimization of trading strategies on financial markets using software simulation tools.

The subject of the study is the methods, models and technologies for developing a web platform for trading strategy validation using multi-agent simulation. The research methods include: analysis of competitive systems, microservice architecture design, implementation of an AI agent state graph, and functional testing of system components.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – прикладний програмний інтерфейс.

DCA – Dollar Cost Averaging – стратегія усереднення вартості.

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту.

JSON – JavaScript Object Notation – текстовий формат обміну даними.

JWT – JSON Web Token – стандарт компактного токена аутентифікації.

LLM – Large Language Model – велика мовна модель.

ORM – Object-Relational Mapping – об'єктно-реляційне відображення.

REST – Representational State Transfer – архітектурний стиль проектування API.

SQL – Structured Query Language – мова структурованих запитів.

SSE – Server-Sent Events – механізм потокового передавання подій від сервера до клієнта.

UI – User Interface – інтерфейс користувача.

ШІ – штучний інтелект.

ЗМІСТ

ВСТУП

Актуальність теми дослідження зумовлена стрімким розвитком фінансових технологій та зростанням складності сучасних ринків, де алгоритмічна торгівля займає домінуючі позиції. Традиційні методи валідації стратегій часто обмежуються простим відтворенням історичних даних, що не дозволяє повноцінно врахувати адаптивність алгоритмів до змінюваних ринкових умов. Проєкт Validex вирішує цю проблему шляхом впровадження мультиагентної симуляції, де інтелектуальні агенти на базі великих мовних моделей не просто тестують параметри, а здійснюють їхню динамічну оптимізацію.

Зростання популярності стратегій сіткової торгівлі (Grid) та усереднення вартості (DCA) на криптовалютних ринках підвищує попит на інструменти точної попередньої оцінки їхніх параметрів. Наявні комерційні платформи або є закритими системами без доступу до внутрішньої логіки, або не підтримують інтеграцію з інструментами штучного інтелекту. Це визначає актуальність розробки відкритої веб-платформи з вбудованим ШІ-агентом, здатним до ітеративного вдосконалення конфігурацій торгових роботів.

Об’єктом дослідження є процес валідації та оптимізації торгових стратегій на фінансових ринках за допомогою програмних засобів симуляції.

Предметом дослідження є методи, моделі та технології розробки веб-платформи для валідації торгових стратегій з використанням мультиагентної симуляції.

Метою роботи є розробка програмного забезпечення онлайн-платформи, яка забезпечує високу точність валідації торгових алгоритмів через поєднання класичного бектестингу та інтелектуальної мультиагентної оптимізації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати існуючі підходи до тестування торгових стратегій та виявити їхні обмеження;

– спроектувати архітектуру платформи на основі мікросервісного підходу з використанням FastAPI;

– реалізувати двигун бектестингу для моделювання стратегій Grid-DCA з підтримкою SPOT та FUTURES ринків;

– розробити мультиагентну систему на базі LangGraph для автоматичної оптимізації параметрів стратегії;

– провести тестування реалізованого програмного забезпечення та оцінити ефективність алгоритмів.

Методи дослідження включають: аналіз і порівняння конкурентних програмних систем, об’єктно-орієнтоване та мікросервісне проєктування архітектури, математичне моделювання показників ефективності торгових стратегій, стейт-машинне програмування потоку ШІ-агента засобами LangGraph, а також функціональне та інтеграційне тестування компонентів платформи.

Наукова новизна роботи полягає у розробці моделі валідації, де процес пошуку оптимальних параметрів торгового робота керується стейт-графом інтелектуального агента. На відміну від класичного перебору параметрів, запропонований підхід використовує великі мовні моделі як оракулів, що аналізують звіти бектестингу та генерують цілеспрямовані гіпотези щодо наступної конфігурації, скорочуючи простір пошуку.

Практичне значення отриманих результатів полягає у створенні функціональної платформи для швидкої перевірки гіпотез та автоматизації налаштування торгових алгоритмів. Реалізована система може безпосередньо застосовуватися трейдерами та дослідниками для оцінки Grid-DCA стратегій на криптовалютних ринках без потреби у глибоких технічних знаннях програмування.

Структура роботи. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі виконано огляд предметної області та проаналізовано існуючі рішення. У другому розділі описано проєктування архітектури та реалізацію компонентів системи. У третьому розділі наведено результати тестування. У четвертому розділі розглянуто питання безпеки життєдіяльності. Загальний обсяг роботи становить 64 сторінок, у тому числі 17 рисунків, 5 таблиці, 32 бібліографічних джерел.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Розвиток сучасних фінансових ринків за останні кілька десятиліть зазнав фундаментальних трансформацій, пройшовши шлях від традиційних біржових залів з голосовим виконанням ордерів до високотехнологічних цифрових екосистем, у яких угоди укладаються за мілісекунди. Алгоритмічна торгівля перестала бути ексклюзивним інструментом великих хедж-фондів та стала доступною широкому колу приватних трейдерів та дослідників [1]. Проте доступність інструментів не знімає критичної проблеми надійності торгових алгоритмів та їхньої здатності адаптуватися до реальних ринкових умов.

Одним із головних викликів у розробці торгового програмного забезпечення є подолання психологічного фактору та когнітивних упереджень, притаманних людям. Трейдери часто схильні до емоційних рішень під впливом страху втрат чи надмірної впевненості після серії успішних угод, що зрештою призводить до порушення дисципліни та ризик-менеджменту [2]. Автоматизація дозволяє жорстко дотримуватися заданих правил, тому якісна валідація стратегій є критично важливим завданням інженерії програмного забезпечення.

Предметна область даної роботи зосереджена на системах валідації торгових стратегій, що базуються на методах сіткової торгівлі (Grid) та усереднення вартості (DCA – Dollar Cost Averaging). Ці стратегії є математично складними та вимагають високоточного моделювання, особливо при симуляції просідань капіталу (drawdown) [3]. На відміну від простих індикаторних стратегій, Grid та DCA передбачають відкриття серії ордерів за заздалегідь визначеними ціновими рівнями, тому помилка у моделюванні навіть на кілька пунктів може призвести до хибного висновку про прибутковість.

Важливим аспектом дослідження є перехід від статичного бектестингу до динамічної мультиагентної симуляції. Класичний підхід, за якого параметри стратегії залишаються незмінними протягом всього часу тестування, часто дає оманливі результати через ефект перенавчання (overfitting) та підгонку під історичні дані [4]. Мультиагентна симуляція дозволяє розглядати торговий

алгоритм як динамічну систему, що потребує систематичного вдосконалення, а ШІ-агент виконує роль досвідченого аналітика, який постійно переоцінює та коригує оптимальні параметри.

1.1 Аналітичний огляд відомих рішень

Для розуміння місця проекту Validex у сучасній інфраструктурі фінансових технологій необхідно провести детальний аналіз існуючих платформ. Ринок інструментів для трейдерів є перенасиченим, проте більшість рішень мають або надто вузьку спеціалізацію, або суттєві технічні обмеження, які не дозволяють реалізувати складні ШІ-інтеграції. Порівняльний аналіз за ключовими критеріями наведено у таблиці 1.1.

Таблиця 1.1 - Порівняльний аналіз платформ для валідації торгових стратегій

№	Критерій	MT5	TrdaingView	QuantConnect	3Commas	Validex
1	2	3	4	5	6	7
1	Мова розробки	MQL5	Pine Script	C# / Python	Відсутня	Python
2	Grid/DCA підтримка	Так	Обмежено	Частково	Так	Так
3	ШІ-оптимізація	Ні	Ні	Ні	Ні	Так
4	Відкритий API	Обмежено	Ні	Так	Так	REST/SSE

Продовження таблиці 1.1

1	2	3	4	5	6	7
5	Веб-інтерфейс	Ні	Так	Так	Так	Так
6	Потокові дані (SSE)	Ні	Ні	Ні	Ні	Так
7	Open-source	Ні	Ні	Частково	Ні	Так

1.1.1 Програмний комплекс MetaTrader 5 та екосистема MQL5

MetaTrader 5 (MT5) залишається лідером серед десктопних платформ для роздрібної торгівлі. Компанія MetaQuotes створила унікальну екосистему, яка поєднує торговий термінал, середовище розробки та хмарну мережу для тестування. Власна мова MQL5, на якій пишуться торгові роботи (Expert Advisors), є об’єктно-орієнтованою та оптимізованою для числових розрахунків. Архітектура платформи орієнтована на низьку затримку та підтримує торгівлю на ф’ючерсних, форексних та фондових ринках.

Тестер стратегій MT5 заслуговує на окрему увагу завдяки підтримці реальних тикових даних. Система може відтворювати рух ціни з точністю до кожної зміни на біржі, включаючи коливання спреду, що є критичним для реалістичної симуляції сіткових стратегій. Оптимізатор MT5 перебирає мільйони комбінацій параметрів у багатопотоковому режимі або через хмарних агентів. Вигляд інтерфейсу тестера стратегій MT5 наведено на рисунку 1.1.

Однак головним недоліком MT5 є його закрита природа та орієнтованість на локальне використання. Хоча існують механізми взаємодії з Python через термінальні бібліотеки, вони працюють повільно та не дозволяють повноцінно інтегрувати зовнішні нейронні мережі або великі мовні моделі. Архітектура MT5

не підтримує розгортання у вигляді веб-сервісу, а ліцензування несумісне з відкритими академічними проектами.

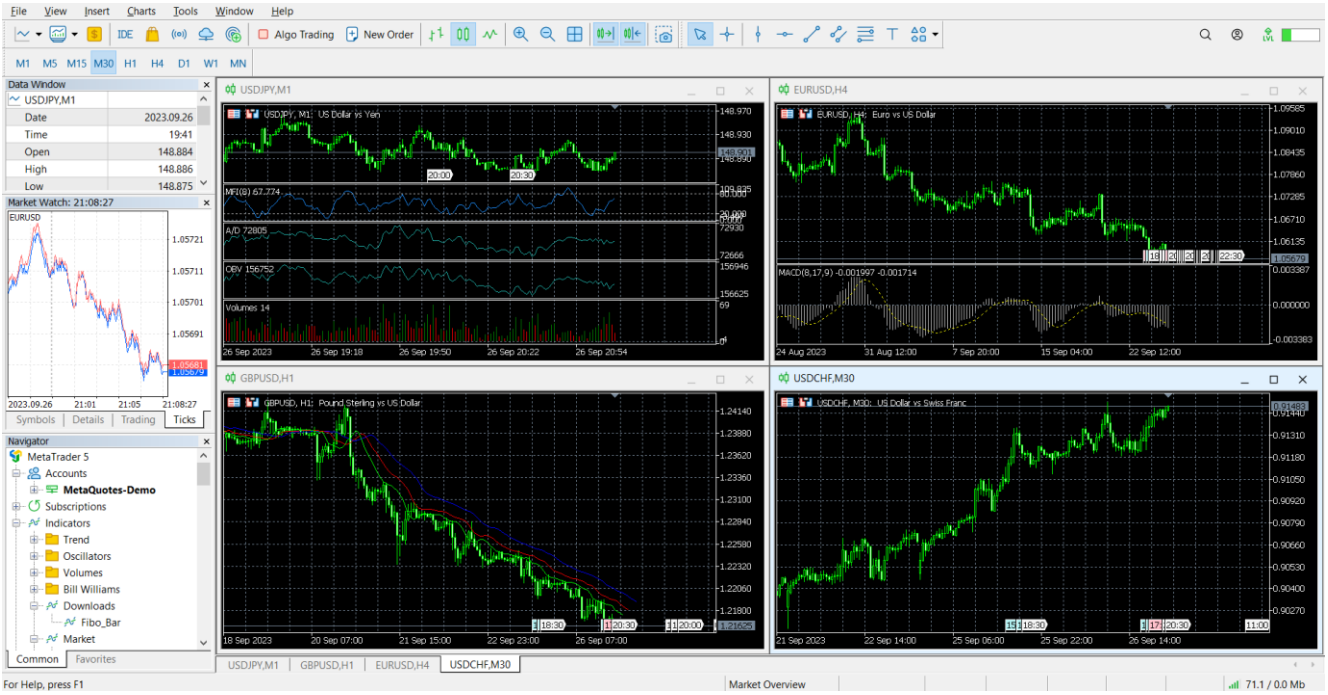


Рисунок 1.1 – Вигляд інтерфейсу MetaTrader 5

1.1.2 Платформа TradingView та мова Pine Script

TradingView здійснив революцію у візуалізації ринкових даних, перенісши професійні графіки у браузер. Власна мова Pine Script стала надзвичайно популярною завдяки простоті: написання стратегії, яка у терміналі Bloomberg займає кілька сторінок коду, займає лише кілька рядків Pine Script. Це створило величезну спільноту трейдерів, що діляться готовими скриптами. Вбудований бектестинг дозволяє швидко оцінити гіпотезу прямо у браузері без встановлення додаткового програмного забезпечення.

Проте за простотою Pine Script приховуються серйозні технічні компроміси. Pine Script є інтерпретованою мовою, що виконується у хмарному середовищі TradingView. Користувач не має доступу до керування ресурсами, а тривалість розрахунків обмежена платформою. Для стратегій Grid та DCA, що вимагають обробки великої кількості ордерів та глибокої ринкової історії, ці ліміти є суттєвою перешкодою. Крім того, платформа страждає від ефекту «lookahead bias», через що

результати стратегій виглядають краще, ніж були б у реальному часі.

Найбільшим бар'єром для використання TradingView у наукових та складних інженерних проектах є повна ізоляція скриптів від зовнішнього світу. Надіслати дані з Pine Script на зовнішній сервер для обробки нейронною мережею або отримати рекомендації від ШІ-агента технічно неможливо. Це робить платформу статичним інструментом візуалізації, але непридатним для реалізації інтелектуальної мультиагентної оптимізації. Вигляд інтерфейсу TradingView наведено на рисунку 1.2.

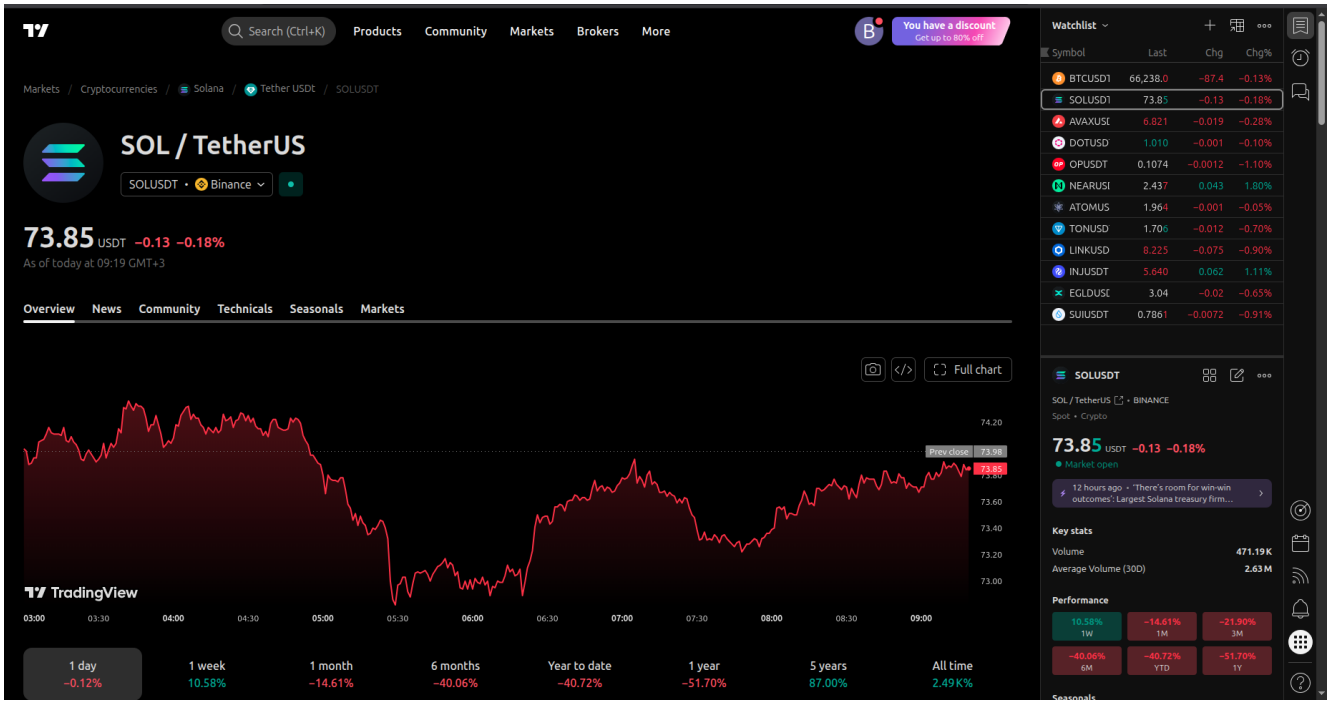


Рисунок 1.2 – Вигляд інтерфейсу TradingView

1.1.3 Професійні хмарні платформи (QuantConnect, Lean Engine)

Для більш професійного підходу існують системи на кшталт QuantConnect, що базуються на двигуні Lean з відкритим вихідним кодом. Ці платформи дозволяють писати стратегії на C# або Python та надають доступ до терабайтів інституційних ринкових даних. Lean Engine підтримує реалістичне моделювання виконання ордерів та орієнтований на інституційні стандарти, включаючи складне моделювання мультиактивних портфелів. Вигляд інтерфейсу QuantConnect наведено на рисунку 1.3.

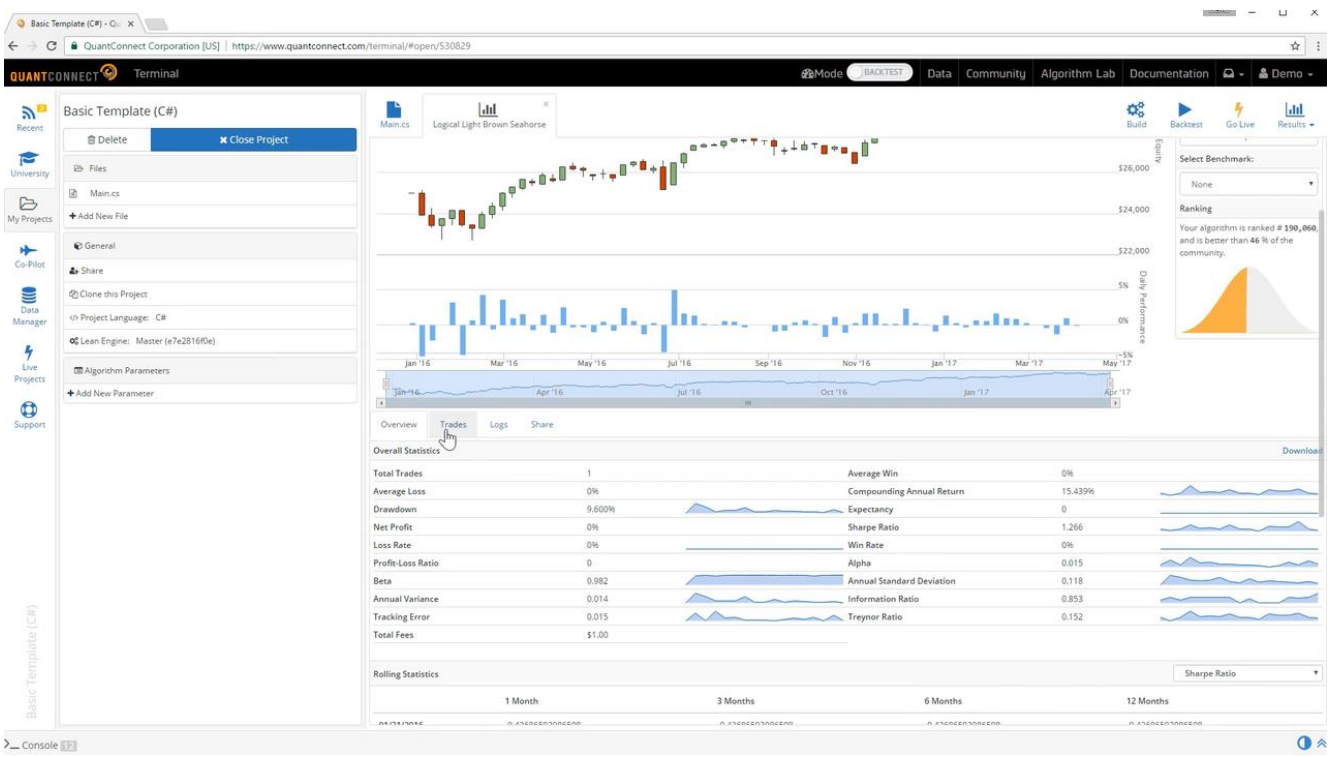


Рисунок 1.3 – Вигляд інтерфейсу QuantConnect

Lean Engine забезпечує високу точність та дозволяє інтегрувати будь-які бібліотеки машинного навчання. Проте складність налаштування є надзвичайно високою: для розробника середнього рівня порогове вкладення часу є майже нездоланним без тривалого навчання. Вартість використання якісних ринкових даних через QuantConnect значна, а локальне розгортання Lean потребує конфігурування складної інфраструктури Docker-контейнерів. Інтеграція з ШІ-агентами на базі LangGraph не задокументована та потребувала б суттєвих доробок.

1.1.4 Інструменти для крипторинку (3Commas, Cryptohopper)

Окремим класом є платформи, орієнтовані виключно на криптовалютний ринок, де стратегії Grid та DCA є найбільш популярними через високу волатильність активів. Системи на кшталт 3Commas надають зручні інтерфейси для налаштування Grid-ботів та мають вбудовані модулі бектестингу. Cryptohopper пропонує маркетплейс готових стратегій та хмарне виконання ботів, що значно спрощує вхід у ринок для початківців. Вигляд інтерфейсу 3Commas наведено на рисунку 1.4.

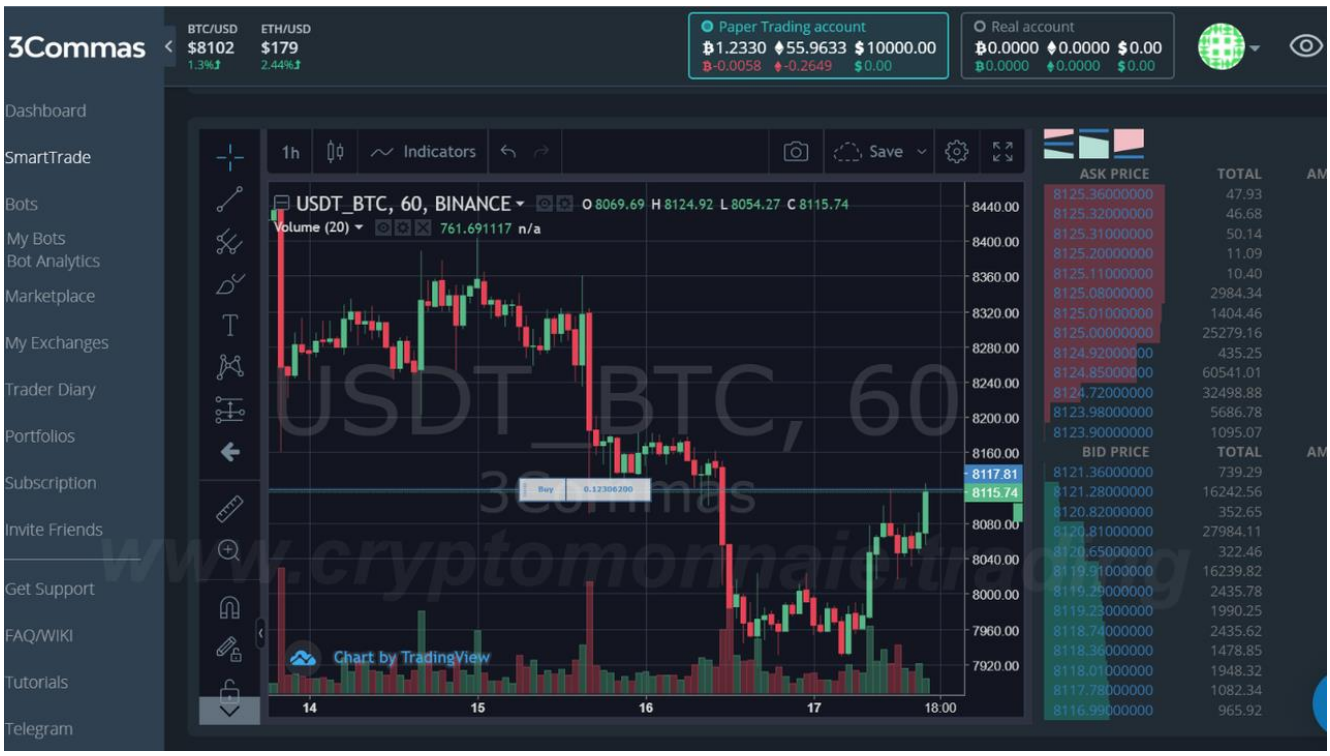


Рисунок 1.4 – Вигляд інтерфейсу 3Commas

Головна проблема таких сервісів – «чорна скринька». Користувач не бачить внутрішньої логіки розрахунків та не може бути впевненим у точності симуляції. Відсутність можливості кастомізації алгоритмів та неможливість впровадження власних математичних моделей робить ці системи придатними для масового використання, але непридатними для глибокого академічного дослідження. Будь-яка спроба інтеграції зовнішнього ШІ-модуля вимагає обходу офіційного API.

1.2 Огляд існуючих технологій реалізації

Аналіз аналогів підтвердив, що для створення платформи нового покоління необхідно поєднати відкритість Python-екосистеми з надійністю промислових веб-фреймворків та інноваційними можливостями мультиагентного ШІ. Нижче розглядаються ключові технологічні рішення, обрані для реалізації платформи Validex.

Python 3.12 обрано основною мовою реалізації завдяки зрілій екосистемі бібліотек для числових обчислень, машинного навчання та роботи з фінансовими

даними [5]. Бібліотеки Pandas та NumPy забезпечують ефективну векторизовану обробку часових рядів та розрахунок математичних показників ефективності: коефіцієнта Шарпа, максимальної просадки, відсотка успішних угод. Рівень продуктивності Python 3.12 з asyncio дозволяє обробляти паралельні запити на бектестинг без блокування основного потоку.

FastAPI є сучасним асинхронним веб-фреймворком для побудови RESTful API на Python [6]. Його ключовою перевагою є автоматична генерація OpenAPI-документації та вбудована валідація вхідних даних на базі Pydantic v2 [7]. Декларативна система dependency injection спрощує розробку чистої архітектури з розділенням відповідальностей між роутерами, сервісами та репозиторіями. Підтримка асинхронних генераторів дозволяє реалізувати Server-Sent Events (SSE) для потокової передачі прогресу ШІ-оптимізації у реальному часі[8].

LangGraph є фреймворком від LangChain для побудови складних потоків ШІ-агентів у вигляді орієнтованих стейт-графів [9, 10]. На відміну від лінійних ланцюжків, LangGraph дозволяє моделювати циклічну поведінку агента: запуск бектестингу, аналіз результатів великою мовною моделлю Google Gemini, генерацію нової конфігурації та повторний запуск [11, 12]. Стейт-граф зберігає повну пам’ять про всі попередні ітерації, що дозволяє агенту приймати рішення на основі накопиченого контексту.

PostgreSQL 16 обрано як основну реляційну базу даних завдяки надійному зберіганню всієї історії ітерацій оптимізації, профілів користувачів та результатів бектестингу [13]. SQLAlchemy 2.0 надає потужний асинхронний ORM-шар із підтримкою Alembic-міграцій для керованого розвитку схеми. Порівняно з NoSQL альтернативами, PostgreSQL забезпечує транзакційну цілісність при одночасному записі результатів від паралельних запитів на бектестинг [14, 15].

React 18 та бібліотека lightweight-charts використовуються для побудови інтерактивного фронтенду [16]. Бібліотека lightweight-charts від TradingView є оптимізованою для відображення фінансових свічкових графіків та ліній показників ефективності [17]. Вона підтримує рендеринг мільйонів точок даних без помітного зниження продуктивності браузера. Vite забезпечує швидке HMR

оновлення під час розробки та оптимізовану збірку для виробничого середовища [18]. Для аутентифікації використовується JWT-токени з бібліотекою PyJWT, а паролі захищаються через bcrypt [19].

1.3 Висновки до першого розділу

У першому розділі описано предметну область алгоритмічної торгівлі, проаналізовано місце стратегій Grid та DCA у сучасному трейдингу та обґрунтовано необхідність переходу від статичного бектестингу до динамічної мультиагентної оптимізації параметрів.

У результаті порівняльного аналізу чотирьох класів існуючих платформ – MetaTrader 5, TradingView, QuantConnect та 3Commas – виявлено їхні спільні обмеження: відсутність відкритого API для ШІ-інтеграції, неможливість потокової передачі результатів оптимізації та обмежена або повністю відсутня підтримка стратегій Grid-DCA з урахуванням кредитного плеча та маржинальних вимог.

Аналіз технологічного стеку підтвердив, що поєднання FastAPI, LangGraph з Google Gemini, PostgreSQL та React забезпечує оптимальну архітектуру для реалізації поставлених цілей. Обрані технології дозволяють досягти трьох ключових цілей: точного моделювання торгових операцій, ітеративної ШІ-оптимізації параметрів у замкненому циклі та інтерактивної веб-візуалізації процесу прийняття рішень агентом у реальному часі.

2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

У даному розділі описано процес проєктування та реалізації компонентів системи Validex. Спочатку формулюються функціональні та нефункціональні вимоги до платформи з урахуванням аналізу предметної області та наявних аналогів, проведеного у першому розділі. На основі вимог виконується проєктування загальної архітектури мікросервісного застосунку, схема бази даних та потоки взаємодії між компонентами. Окремо розглядаються проєктні рішення щодо бектестинг-двигуна, мультиагентної системи оптимізації та клієнтського інтерфейсу.

Реалізація системи описується через конкретні технічні рішення: вибір фреймворків та бібліотек, ключові архітектурні патерни (Service Layer, Repository, Factory), структуру модулів Python-пакету та організацію React-компонентів фронтенду. Наводяться ключові фрагменти програмного коду, що ілюструють реалізацію стейт-графа LangGraph, алгоритму сіткового бектестингу та механізму потокової передачі прогресу оптимізації (SSE) між сервером та клієнтом.

Розроблені компоненти системи відповідають принципам чистої архітектури: бізнес-логіка повністю ізольована від транспортного шару, залежності спрямовані до центру (до доменних сутностей), а зовнішні інтеграції (Binance API, Google Gemini, PostgreSQL) підключаються через адаптери та конфігуровані через змінні оточення [20]. Це забезпечує легку замінюваність зовнішніх провайдерів без зміни доменної логіки.

2.1 Вимоги до системи

Формування вимог до системи Validex ґрунтується на аналізі предметної області алгоритмічного трейдингу та обмежень наявних платформ (розділ 1). Система розробляється як спеціалізоване середовище для автоматизованої перевірки та інтелектуальної оптимізації стратегій Grid-DCA. Основна мета полягає у наданні трейдерам та розробникам інструментарію для переходу від

ручного перебору параметрів до ШІ-керованої оптимізації з повною прозорістю логіки прийняття рішень.

Вимоги поділяються на три категорії: функціональні вимоги до модуля валідації (бектестинг-двигун), функціональні вимоги до компонентів обміну даними та ШІ-інтеграції, а також нефункціональні вимоги до продуктивності, безпеки та розгортання системи. Кожна вимога ідентифікується унікальним кодом для подальшого відстеження у тестах та документації.

2.1.1 Вимоги до функціоналу валідації та симуляції

Компонент валідації є центральним ядром системи. Його функціональні вимоги визначають повний спектр торгових сценаріїв, які повинна підтримувати платформа, починаючи від базових параметрів сіткової торгівлі і закінчуючи складними механізмами управління позицією з технічними фільтрами. Вимоги до функціоналу валідації:

– FR-1: моделювання виконання торгових операцій з урахуванням кредитного плеча та маржинальних вимог для SPOT та FUTURES ринків; точний розрахунок ефективного плеча з урахуванням типу ринку;

– FR-2: підтримка Grid-DCA стратегій з рівномірним та логарифмічним розподілом рівнів додаткових ордерів по ціновому діапазону; автоматичний перерахунок середньої ціни позиції при кожному заповненні DCA-ордера;

– FR-3: підтримка трьох режимів торгівлі: Simple (рівні ордери автоматично), Custom (ручні рівні цінового покриття), Signal (DCA лише при сигналі технічного індикатора);

– FR-4: розрахунок повного набору аналітичних метрик: чистий прибуток у доларах та відсотках, максимальна просадка, відсоток виграшних угод (winrate), коефіцієнт Шарпа, фактор прибутку (profit factor), середній прибуток / збиток на угоду;

– FR-5: підтримка понад 20 технічних індикаторів для фільтрації умов входу в позицію, відкриття DCA-ордерів та виходу з позиції: RSI, MACD, Stochastic, Stochastic RSI, ADX, CCI, Supertrend, EMA, SMA, HMA, Bollinger Bands,

ATR, MFI, Vortex, VuManChu, Chaikin Oscillator, Awesome Oscillator, CMO, ROC, Price Change;

– FR-6: підтримка геометричної прогресії обсягів ордерів (martingale): обсяг і-го ордера = обсяг першого * martingale^(i-1); параметр martingale >= 1.0 (1.0 = рівні обсяги);

– FR-7: підтримка механізмів управління ризиком: стоп-лос на всю позицію, trailing stop із заданим відсотком відкату від максимуму прибутку, динамічне закриття при досягненні take-profit;

– FR-8: формування масиву маркерів (markers) для відображення точок входу, DCA-ордерів та виходу на графіку цінової динаміки; формування деталізованого журналу угод (trade log) із часовими мітками та P&L кожної угоди.

Нефункціональні обмеження компонента валідації передбачають обробку щонайменше двох паралельних запитів без блокування, мінімальний набір вхідних свічок – не менше 60 одиниць (MIN_CANDLES = 60), а також детерміністичність результатів: однакові вхідні дані повинні завжди давати однаковий результат незалежно від порядку виконання паралельних запитів.

2.1.2 Вимоги до компонентів обміну даними та ШІ-інтеграції

Для забезпечення роботи інтелектуального шару системи та надання актуальної ринкової інформації, до платформи висуваються такі функціональні вимоги щодо інтеграції з зовнішніми сервісами та внутрішньої комунікації між компонентами:

– FR-9: завантаження OHLCV-котирувань через Binance API з підтримкою часових інтервалів 1m, 5m, 15m, 30m, 1h, 4h, 1d; автоматичне кешування результатів для уникнення повторних запитів до зовнішнього API при незмінних вхідних параметрах [21, 22];

– FR-10: ітераційна взаємодія з великими мовними моделями (Google Gemini) для аналізу звітів бектестингу та автоматичної генерації покращених конфігурацій параметрів; парсинг структурованої відповіді LLM у схему ReconfiguredParams (Pydantic);

– FR-11: підтримка SSE (Server-Sent Events) для потокової передачі прогресу ШІ-оптимізації; публікація подій двох типів: log (числові метрики ітерації + аргументація агента) та complete (фінальний результат + найкраща конфігурація);

– FR-12: збереження повного журналу всіх ітерацій оптимізації: параметри конфігурації, отримані метрики, текстова аргументація LLM-агента та посилання на найкращу знайдену конфігурацію;

– FR-13: управління API-ключами сторонніх LLM-провайдерів (зберігання, оновлення, видалення) з шифруванням у базі даних; пріоритет: ключ користувача > системний ключ;

– FR-14: JWT-аутентифікація з access-токеном та механізмом оновлення через refresh-токен; хешування паролів bcrypt; ізоляція даних між користувачами через перевірку власника ресурсу.

Нефункціональні вимоги до безпеки передбачають: час дії access-токена не більше 60 хвилин, захист від SQL-ін'єкцій через параметризовані запити ORM, захист від XSS через екранування вхідних рядків, обмеження частоти запитів (rate limiting) на ендпоінтах аутентифікації до 5 запитів на хвилину з одного IP-адресу [23]. Всі зовнішні API-виклики виконуються асинхронно та обгорнуті в обробники помилок з відповідними fallback-механізмами.

2.1.3 Нефункціональні вимоги

Ключові нефункціональні вимоги, що критично впливають на надійність, продуктивність та зручність підтримки системи, наведено у таблиці 2.1.

Таблиця 2.1 – Нефункціональні вимоги до системи Validex

№	NFR-ID	Категорія	Вимога	Метрика/критерій
1	**2**	**3**	**4**	**5**
1	NFR-1	Продуктивність	Час відповіді REST API для non-backtest ендпоінтів	< 200 мс при нормальному навантаженні

Продовження таблиці 2.1

1	2	3	4	5
2	NFR-2	Продуктивність	Бектестинг 500 свічок з 3 фільтрами	< 2 секунди на запит
3	NFR-3	Продуктивність	Паралельні запити на бектестинг	Мінімум 2 без деградації
4	NFR-4	Надійність	Heuristic fallback при недоступному LLM	Симуляція завершується без виключення
5	NFR-5	Надійність	Транзакційність збереження симуляцій	Rollback при будь-якому виключенні
6	NFR-6	Безпека	Час дії access JWT-токена	Не більше 60 хвилин
7	NFR-7	Безпека	Хешування паролів користувачів	bcrypt, cost factor >= 12
8	NFR-8	Безпека	Ізоляція даних між користувачами	HTTP 403 при доступі до чужого ресурсу
9	NFR-9	Масштабованість	Горизонтальне масштабування бекенду	Stateless-сервіс, стан лише в БД
10	NFR-10	Розгортання	Контейнеризація всіх сервісів	docker-compose up – єдина команда запуску
11	NFR-11	Підтримуваність	Максимальний розмір файлу коду	400 рядків (розбиття на модулі)
12	NFR-12	Зручність	Відображення прогресу ШІ-агента в реальному часі	SSE latency < 500 мс на подію

Вимоги NFR-9 та NFR-10 забезпечують стратегічну гнучкість розгортання: stateless-архітектура бекенду дозволяє запускати декілька екземплярів сервісу за балансувальником навантаження без синхронізації стану між ними. Єдиним централізованим станом є база даних PostgreSQL, доступ до якої регулюється пулом з'єднань SQLAlchemy.

2.2 Архітектура системи

Архітектура системи Validex побудована за принципом багаторівневого розділення відповідальностей між незалежними шарами. Клієнтська частина (React SPA) взаємодіє виключно з REST API та SSE-ендпоінтами серверного шару (FastAPI). Серверний шар делегує бізнес-логіку сервісному шару, який у свою чергу взаємодіє з репозиторіями для доступу до PostgreSQL та адаптерами для виклику зовнішніх API (Binance, Google Gemini). Загальну архітектуру системи наведено на рисунку 2.1.

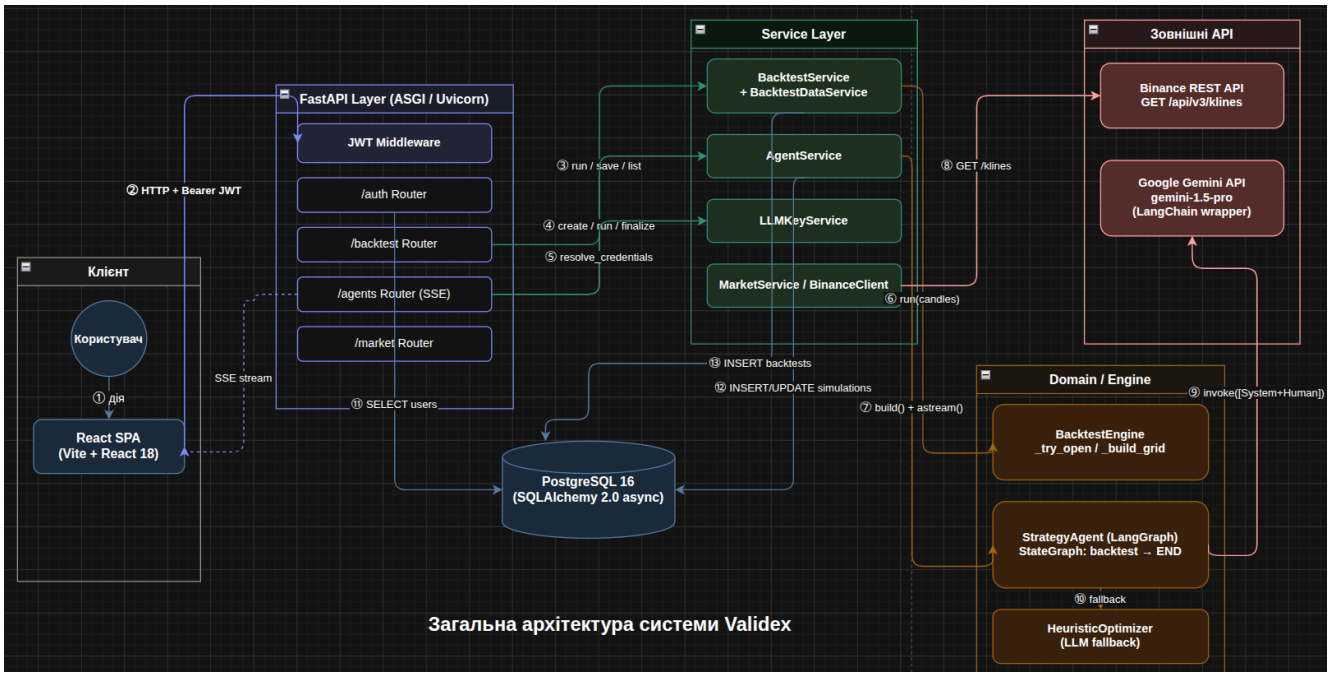


Рисунок 2.1 – Загальна архітектура системи

Серверна частина організована за шаблоном «роутер – сервіс – репозиторій».

Роутери FastAPI обробляють HTTP/SSE-запити та здійснюють лише валідацію вхідних даних через Pydantic-схеми. Вся бізнес-логіка інкапсулюється у сервісах: BacktestService – симуляція торгових стратегій, AgentService – управління ШІ-агентами та їхніми симуляціями, MarketService – отримання ринкових даних. Репозиторії абстрагують доступ до бази даних через асинхронні SQLAlchemy-сесії.

ШІ-компонент виокремлений в окремий підпакет api/agents/, що забезпечує його незалежність від решти сервісного шару. StrategyAgent (LangGraph) викликається через AgentService, який передає до нього початковий стан AgentState та слухає подієвий потік через async generator. LLMFactory абстрагує ініціалізацію клієнтів різних LLM-провайдерів (Google, OpenAI, Anthropic), повертаючи уніфікований langchain-сумісний об'єкт.

Зовнішні інтеграції реалізовані як ізольовані адаптери: BinanceClient обгортає HTTP-запити до REST API Binance з обробкою помилок та кешуванням, а LLMFactory + LangChain повністю абстрагують специфіку протоколів різних LLM-провайдерів. Завдяки цьому заміна Binance на інший маркет-дата провайдер або перехід з Google Gemini на GPT-4 вимагає змін лише в адаптерному шарі без торкання доменної логіки.

Контейнеризація здійснюється через docker-compose з трьома сервісами: backend (Python/FastAPI), frontend (Node.js/Vite build + Nginx), db (PostgreSQL 16). Між контейнерами оголошені named volumes для збереження даних БД та health-check залежності (backend чекає готовності db перед стартом) [24].

2.3 Проектування програми для налаштування параметрів

Проєктування конфігураційної програми охоплює визначення варіантів використання системи та структури даних, що описують торгову стратегію. Діаграму варіантів використання системи наведено на рисунку 2.2. Актором виступає зареєстрований користувач, який взаємодіє з трьома основними підсистемами: бектестингу, ШІ-оптимізації та ринкових даних.

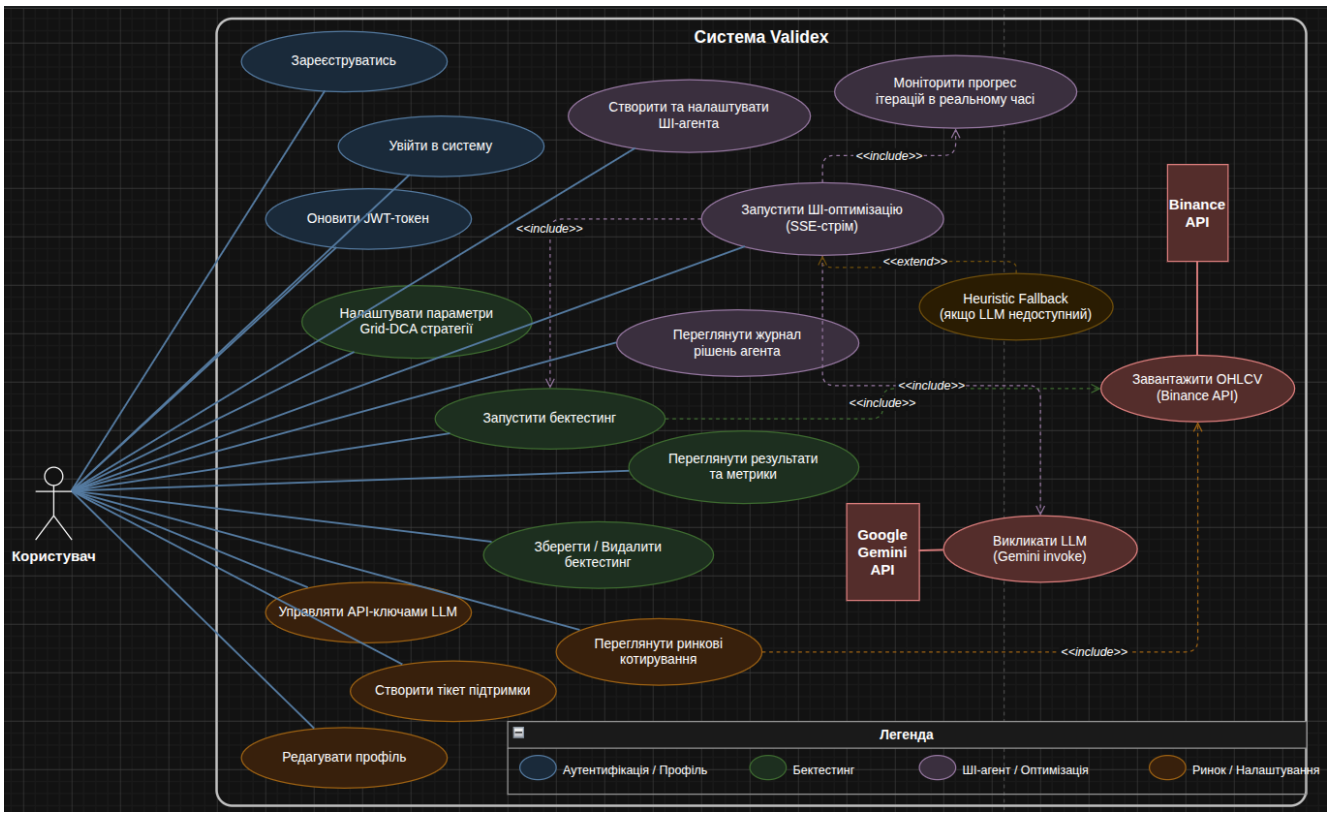


Рисунок 2.2 – Діаграма варіантів використання системи Validex

Конфігурація бектестингу описується структурою даних BacktestConfig – єдиним джерелом правди для бектестинг-двигуна. Структура є Python dataclass з типізованими полями, що забезпечує статичну перевірку типів засобами mypy та можливість серіалізації до/з JSON для збереження в базі даних. Усі поля конфігурації розподілено за трьома групами: параметри позиції, параметри сітки та параметри фільтрів.

Параметри позиції визначають базові умови торгівлі: deposit (початковий депозит у USDT), market_type (SPOT або FUTURES), leverage (кредитне плече, ефективне тільки для FUTURES), direction (LONG або SHORT), а також цілі закриття take_profit та stop_loss у відсотках від середньої ціни позиції. Параметри сітки визначають будову DCA-структури: grid_size (кількість ордерів), price_coverage (цінове покриття у відсотках) та martingale (множник обсягу).

Параметри фільтрів представлені трьома JSON-масивами: filters (умови для відкриття першого ордера), signal_filters (умови для відкриття DCA-ордерів) та exit_signal_filters (умови для дострокового закриття позиції). Кожен фільтр – це

JSON-об'єкт із полями indicator, параметрами (period, fast_period, slow_period тощо), condition (Greater / Less / Above / Below / Crosses Up / Crosses Down) та пороговим значенням value.

Ключовий фрагмент структури BacktestConfig наведено у лістингу 2.1.

Лістинг 2.1 – Структура конфігурації BacktestConfig

```
@dataclass
class BacktestConfig:
    # Параметри позиції
    deposit: float
    market_type: str        # "SPOT" | "FUTURES"
    leverage: float
    direction: str          # "LONG" | "SHORT"
    take_profit: float      # % закриття прибутку
    stop_loss: float        # % закриття збитку
    # Параметри сітки
    grid_size: int
    price_coverage: float   # Цінове покриття, %
    martingale: float       # Множник обсягу DCA
    # Режим торгівлі
    trading_mode: str       # "Simple" | "Custom" | "Signal"
    # Фільтри сигналів
    filters: List[Dict]
    signal_filters: List[Dict]
    exit_signal_filters: List[Dict]
    # Захист позиції
    stop_loss_active: bool = True
    trailing_stop: float = 0.0

    @property
    def effective_leverage(self) -> float:
        return self.leverage if self.market_type == "FUTURES" else 1.0
```

Клієнтська форма конфігурації на React реалізована як керований компонент

(controlled component) з локальним станом через useState. Переключення між режимами Simple/Custom/Signal динамічно відображає або приховує відповідні поля форми. Поля фільтрів реалізовані як динамічний список FilterRow-компонентів, де кожен рядок дозволяє вибрати індикатор зі спадного списку та налаштувати його параметри.

2.4 Проектування компоненту обміну даних

Компонент обміну даними забезпечує зв'язок між клієнтом, сервером та зовнішніми сервісами. REST API визначає 18 ендпоінтів, організованих у 5 роутерів: /auth (реєстрація, вхід, оновлення токена), /users (профіль, LLM-ключі), /backtest (CRUD + запуск), /agents (CRUD агентів + управління симуляціями), /market (котирування, символи).

Схему бази даних (ER-діаграму) наведено на рисунку 2.3. Центральною сутністю є users, яка пов'язана відношеннями 1:N з усіма іншими таблицями системи. Фільтрація за user_id у SQL-запитах гарантує повну ізоляцію даних користувачів.

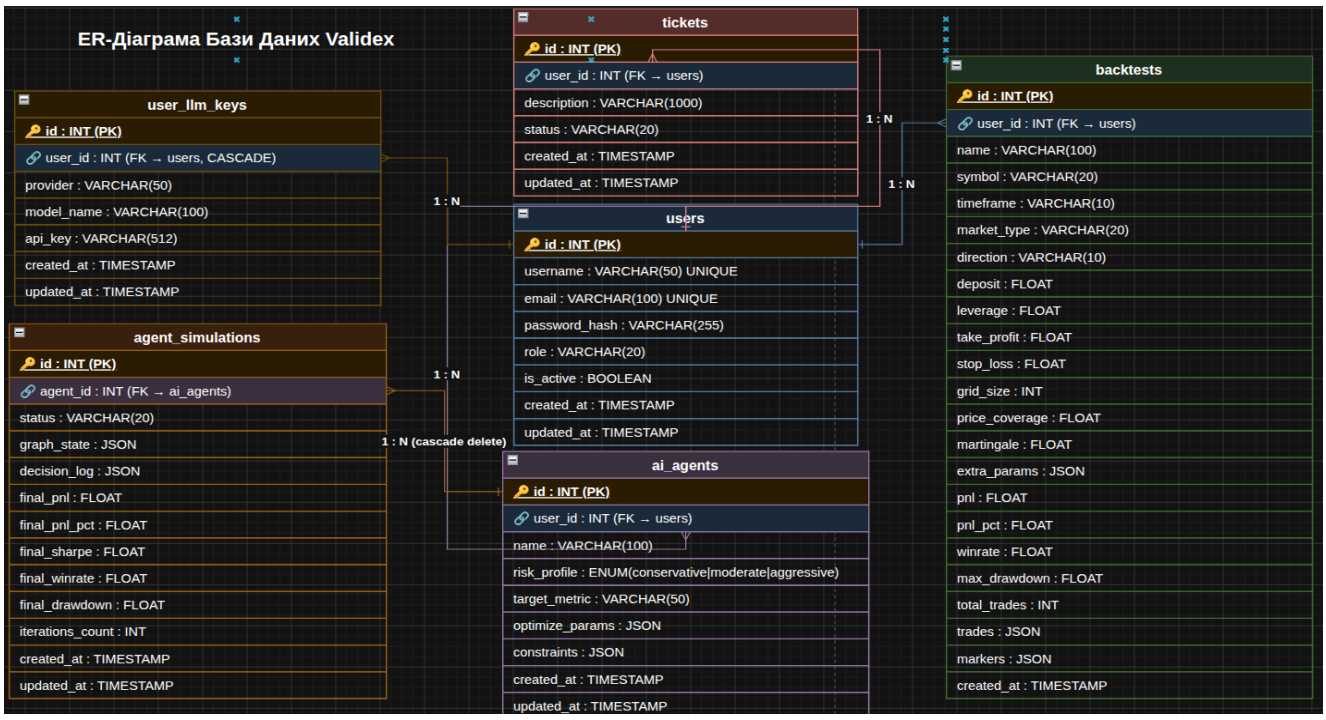


Рисунок 2.3 – ER-діаграма бази даних системи Validex

Таблиця backtests зберігає повну конфігурацію кожної симуляції у JSON-полях extra_params (filters, signal_filters, exit_signal_filters) та результати у полях pnl, pnl_pct, winrate, max_drawdown, total_trades, а також JSON-масиви trades та markers для відновлення деталізованого звіту та графіка.

Таблиця ai_agents зберігає налаштування агентів: risk_profile (ENUM: conservative / moderate / aggressive), target_metric (рядок, наприклад sharpeRatio або pnlPct), optimize_params (JSON-масив параметрів, дозволених до мутації: takeProfit, stopLoss, gridSize, priceCoverage, martingale, filters, signalFilters, exitSignalFilters) та constraints (JSON з граничними значеннями параметрів).

Таблиця agent_simulations фіксує кожен запуск агента: повний JSON-стан GraphState після завершення (graph_state), деталізований журнал рішень decision_log з текстовою аргументацією LLM для кожної ітерації, кінцеві метрики (final_pnl, final_sharpe, final_winrate, final_drawdown) та кількість виконаних ітерацій (iterations_count). Поле status (ENUM: pending / running / done / failed) дозволяє клієнту перевіряти стан симуляції.

Таблиця user_llm_keys дозволяє зберігати API-ключі до провайдерів великих мовних моделей у зашифрованому вигляді. Поле provider визначає провайдера (google, openai, anthropic), що дозволяє LLMFactory вибрати правильний клієнт при ініціалізації агента. Приоритет використання: user_llm_keys > settings.google_api_key (системний ключ з .env) > heuristic fallback.

Таблиця tickets реалізує підсистему підтримки користувачів: кожен квиток містить title, description, status та посилання на автора. Ця функціональність надає платформі можливість зворотного зв'язку між адміністратором та трейдерами.

2.5 Проєктування компонента нейронної мережі

Компонент ШІ-оптимізації реалізований у вигляді стейт-машини на базі фреймворку LangGraph. Ключовою перевагою цього підходу є підтримка циклічного виконання з накопиченням стану між ітераціями: агент «пам'ятає» всі

попередні прогони і може аналізувати тренди метрик при прийнятті наступного рішення. Це принципово відрізняє підхід від простого циклу for, оскільки LangGraph забезпечує граф станів з умовними ребрами, що дозволяє тонко керувати потоком виконання. Стейт-граф агента наведено на рисунку 2.4.

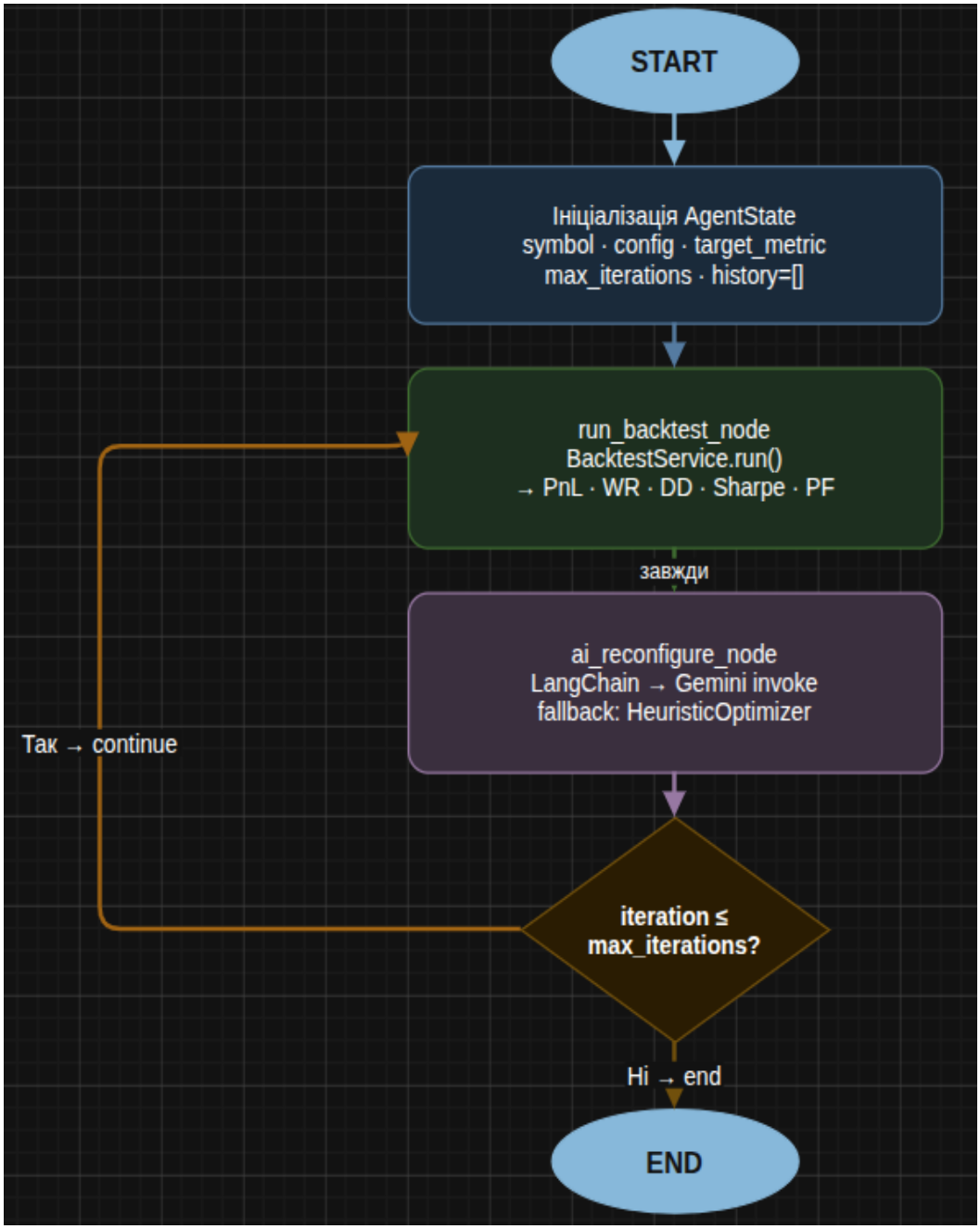


Рисунок 2.4 – Стейт-граф ШІ-агента оптимізації

Стан агента описується TypedDict AgentState з 13 полями. Поля config та best_config зберігають поточну та найкращу знайдену конфігурацію BacktestConfig.

Поле history (List[Dict]) накопичує запис кожної ітерації: параметри, метрики та текстову аргументацію LLM. Поле logs (List[str]) є буфером для публікації подій через SSE. Структуру AgentState наведено у лістингу 2.2.

Лістинг 2.2 – Структура стану AgentState (LangGraph)

```
class AgentState(TypedDict):
    config: BacktestConfig     # Поточна конфігурація
    symbol: str                # Торгова пара (BTCUSDT)
    start_date: Optional[str]
    end_date: Optional[str]
    history: List[Dict]        # Журнал ітерацій
    iteration: int             # Поточна ітерація
    max_iterations: int        # Ліміт ітерацій
    target_metric: str         # sharpeRatio | pnlPct | ...
    optimize_params: List[str] # Дозволені параметри
    best_result: Optional[Dict]
    best_config: Optional[BacktestConfig]
    logs: List[str]            # Буфер SSE-подій
    risk_profile: str          # conservative | moderate | aggressive
    last_reasoning: Optional[str]
```

Граф будується методом StrategyAgent.build() через StateGraph API. Два вузли (backtest та reconfigure) з'єднані безумовним ребром backtest → reconfigure. Від вузла reconfigure виходить умовне ребро: якщо iteration <= max_iterations, граф повертається до backtest (continue); інакше – переходить до END. Точкою входу є backtest.

Вузол backtest_node виконує три дії: (1) запускає BacktestService з поточним config та символом, (2) порівнює метрику результату з best_result і оновлює best_config якщо знайдено кращий результат, (3) додає запис до history та рядок-прогрес до logs. Особливість порівняння: для метрики maxDrawdown «краще» означає менше значення, для всіх інших – більше. Це реалізовано через умовну перевірку metric == "maxDrawdown".

Вузол reconfigure_node формує системний промпт з фізикою сіткової торгівлі, пулом 20 індикаторів та профілем ризику агента, а також людський промпт із поточними параметрами та табличною представленою історією всіх ітерацій. LLM повертає структуровану відповідь у форматі ReconfiguredParams (Pydantic BaseModel з полями takeProfit, stopLoss, gridSize, priceCoverage, martingale, filters, signalFilters, exitSignalFilters, reasoning). При помилці LLM активується HeuristicOptimizer – детерміністичний оптимізатор на основі правил.

HeuristicOptimizer аналізує trend метрики за останніми трьома ітераціями та застосовує одне з 6 правил мутації: збільшити priceCoverage при зростанні drawdown, зменшити martingale при від'ємному PnL, збільшити take_profit при winrate > 70%, зменшити grid_size при малій кількості угод тощо. Кожне правило мутує рівно один параметр на 5-15%, що забезпечує стабільний градієнтний спуск без ризику переходу до некоректних значень.

2.6 Реалізація

У цьому підрозділі описано практичну реалізацію компонентів системи Validex. Наводяться вибір і обґрунтування програмних засобів, ключові фрагменти коду, що демонструють реалізацію бектестинг-двигуна та SSE-ендпоінту, а також скріншоти інтерфейсу застосунку.

2.6.1 Програмні засоби реалізації

Для розробки системи використано наступні технології та фреймворки. Зведену таблицю технологічного стеку наведено у таблиці 2.2.

Таблиця 2.2 – Технологічний стек системи Validex

№	Компонент	Технологія	Версія	Призначення
1	**2**	**3**	**4**	**5**
1	Серверна частина	Python	3.12	Основна мова бекенду (asyncio)

Продовження таблиці 2.2

1	2	3	4	5
2	Серверна частина	FastAPI	0.136	Веб-фреймворк, SSE, OpenAPI
3	Серверна частина	Pydantic v2	2.11	Валідація даних, схеми
4	ШІ-агент	LangGraph	1.2	StateGraph для ШІ-агента
5	ШІ-агент	LangChain Google	2.1	Інтеграція з Google Gemini
6	ШІ-агент	Google Gemini	gemma-4-31b-it	LLM для оптимізації
7	База даних	PostgreSQL	16	Реляційна база даних
8	База даних	SQLAlchemy	2.0	Асинхронний ORM
9	База даних	Alembic	1.15	Міграції схеми БД
10	Аналітика	Pandas	3.0	Обробка часових рядів
11	Аналітика	NumPy	2.4	Числові обчислення
12	Безпека	PyJWT	2.13	JWT-токени аутентифікації
13	Безпека	bcrypt	5.0	Хешування паролів
14	Клієнтська частина	React	18	SPA-фреймворк
15	Клієнтська частина	lightweight-charts	5.0	Фінансові графіки
16	Клієнтська частина	Vite	6.3	Збірка та dev-сервер
17	DevOps	Docker	27	Контейнеризація сервісів
18	DevOps	uv	0.7	Менеджер залежностей Python

Вибір FastAPI обумовлений його нативною підтримкою async/await та StreamingResponse для реалізації SSE, автоматичною генерацією OpenAPI-документації з Pydantic-схем, а також високою продуктивністю завдяки ASGI-протоколу (Uvicorn). LangGraph обраний замість ручної реалізації циклу оптимізації через вбудовану підтримку conditional edges, типобезпечного стану через TypedDict та async streaming через astream(). SQLAlchemy 2.0 з async-сесіями інтегрується в asyncio-модель FastAPI без блокування event loop.

Середовище розробки: Visual Studio Code з розширеннями Python, Pylance та ESLint. Менеджер залежностей Python – uv (pyproject.toml), що забезпечує детерміністичне відтворення середовища через uv.lock. Для автоматичного форматування та лінтингу Python-коду використовується ruff 0.15, JavaScript – ESLint 9 з конфігурацією для React та TypeScript.

2.6.2 Реалізація застосунку

Клієнтська частина реалізована як Single Page Application на React 18 з Vite як інструментом збірки. Маршрутизація реалізована через власний механізм хеш-навігації на основі window.location.hash та події hashchange без сторонніх бібліотек – підтримуються маршрути login, register, forgot, reset-password, profile, charts, backtest, ai_agents. Захищені маршрути перевіряють наявність JWT-токена у localStorage або sessionStorage; за його відсутності виконується автоматичне перенаправлення на login. При HTTP 401 функція fetchWithAuth централізовано видаляє токен, генерує подію validex:session-expired і перенаправляє користувача.

Глобальний стан організований через два React Context: LangContext забезпечує перемикання інтерфейсу між українською та англійською мовами через файли локалізації ua.json та en.json, ToastContext – централізоване відображення сповіщень-тостів. Після авторизації відображається основний layout: фіксована бокова навігаційна панель та область контенту з верхньою шапкою. Бокова панель містить пункти навігації між розділами, перемикач мови, кнопку підтримки та виходу; на мобільних розрізах вона ховається за кнопку-гамбургер з накладкою.

Загальну структуру авторизованого інтерфейсу наведено на рисунку 2.5.

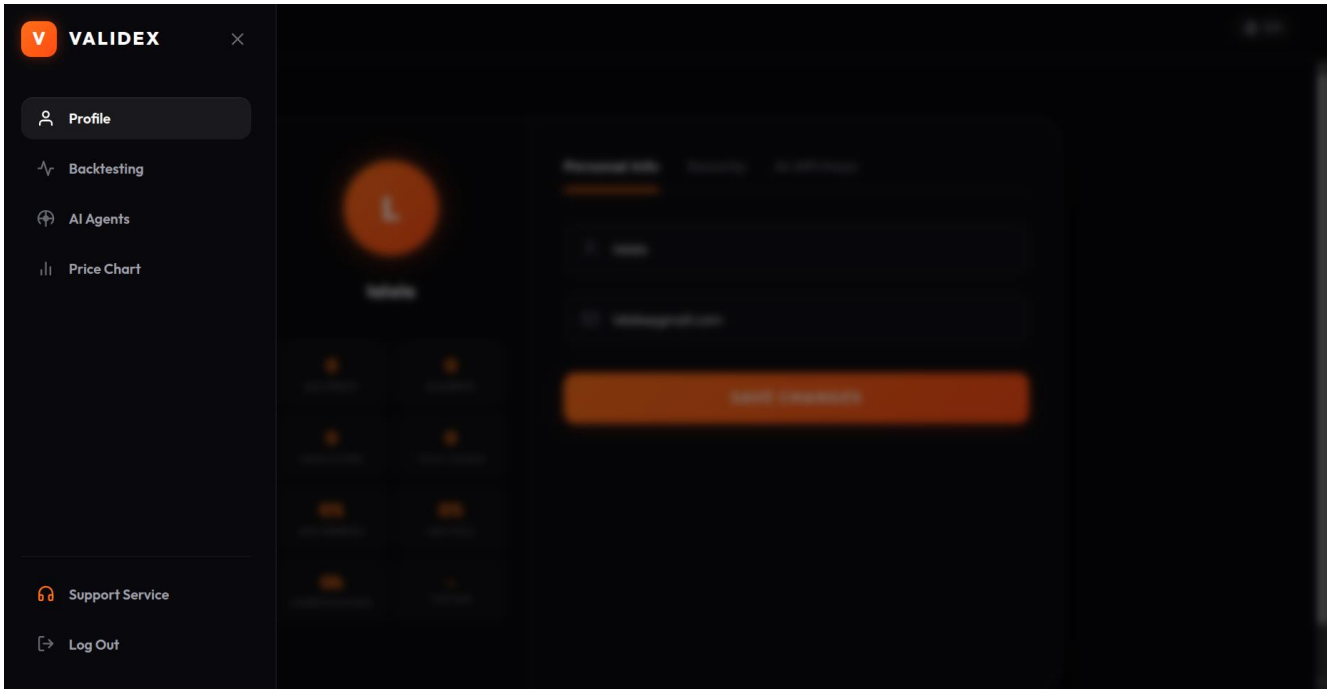


Рисунок 2.5 – Загальний вигляд авторизованого інтерфейсу застосунку з бічною навігацією

Модуль бектестингу складається з трьох представлень, між якими перемикає контейнерний компонент BacktestDashboard. Головна сторінка відображає три зведені картки (загальна кількість запусків, середній Winrate, сукупний PnL) та таблицю завершених симуляцій з повним набором метрик. При натисканні «Створити бектест» відкривається StrategyEditor – редактор стратегії, розбитий на чотири секції-підкомпоненти: MainSettings (пара, таймфрейм, тип ринку SPOT/FUTURES, депозит, плече, напрямок, дати тестування), EntrySettings (умови входу на основі індикаторів: Price, Bollinger Bands, Volume з настроюваними фільтрами), OrderSettings (параметри сітки: відсоток покриття ціни, розмір сітки, множник мартингейл, крок ордеру) та ExitSettings (Take Profit, Stop Loss, Trailing Stop). Права панель редактора відображає живий графік обраної пари в реальному часі через Binance API та бібліотеку lightweight-charts з підтримкою типів відображення: свічки, лінія, area. Інтерфейс редактора стратегії наведено на рисунку 2.6.

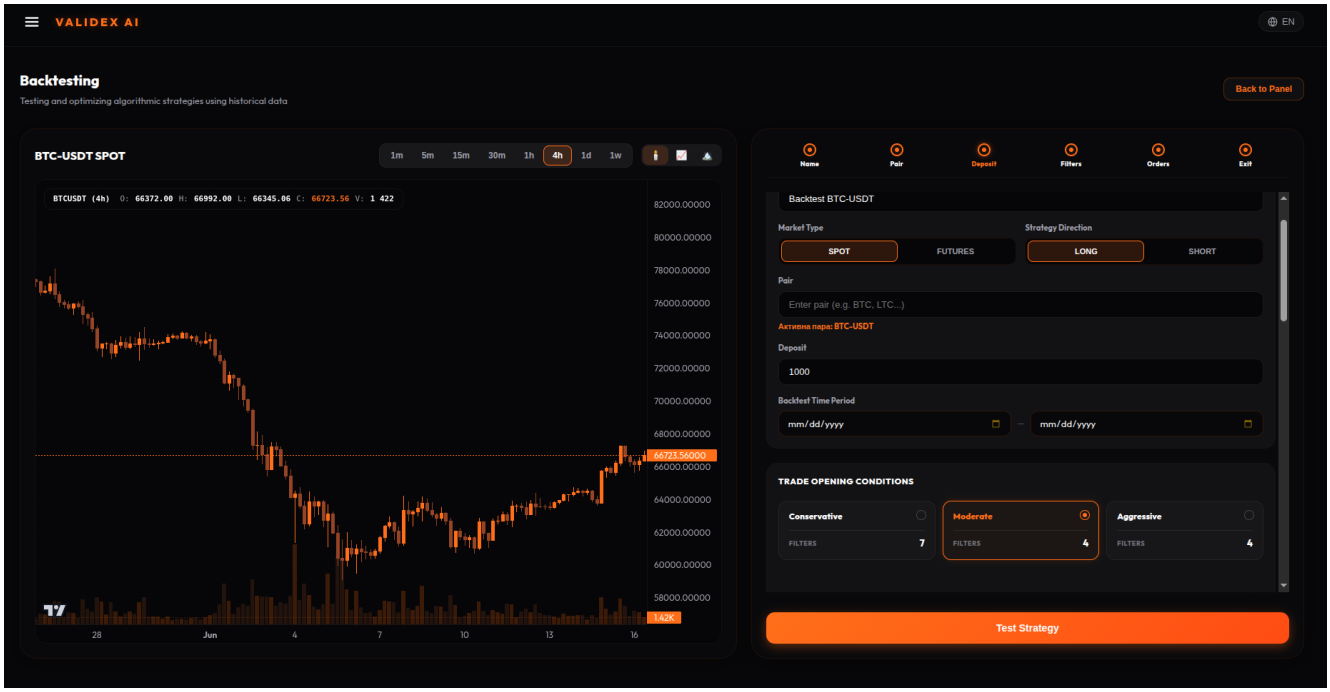


Рисунок 2.6 – Редактор Grid-стратегії з панеллю конфігурації та живим графіком ринку

Після завершення симуляції контейнер перемикається на BacktestReportPage. Сторінка відображає зведені картки метрик (PnL USDT, PnL%, Winrate, Max Drawdown, кількість угод) з кольоровим кодуванням (зелений – прибуток, червоний – збиток), інтерактивний графік балансу та прибутковості з маркерами кожної угоди на lightweight-charts та деталізовану таблицю всіх угод з часом відкриття/закриття, ціною входу/виходу та P&L. Звіт детального бектесту наведено на рисунку 2.7.

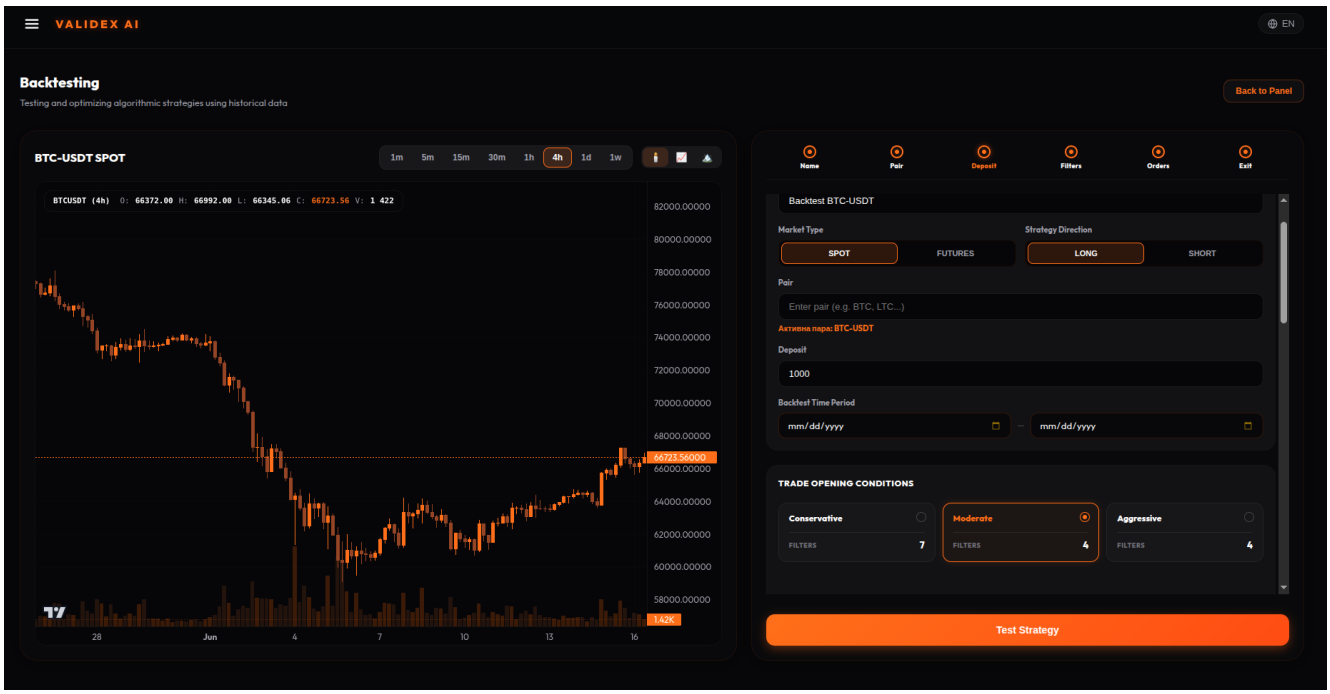


Рисунок 2.7 – Редактор Grid-стратегії з панеллю конфігурації та живим графіком ринку

Після завершення симуляції контейнер перемикається на BacktestReportPage. Сторінка відображає зведені картки метрик (PnL USDT, PnL%, Winrate, Max Drawdown, кількість угод) з кольоровим кодуванням (зелений – прибуток, червоний – збиток), інтерактивний графік балансу та прибутковості з маркерами кожної угоди на lightweight-charts та деталізовану таблицю всіх угод з часом відкриття/закриття, ціною входу/виходу та P&L. Звіт детального бектесту наведено на рисунку 2.8.

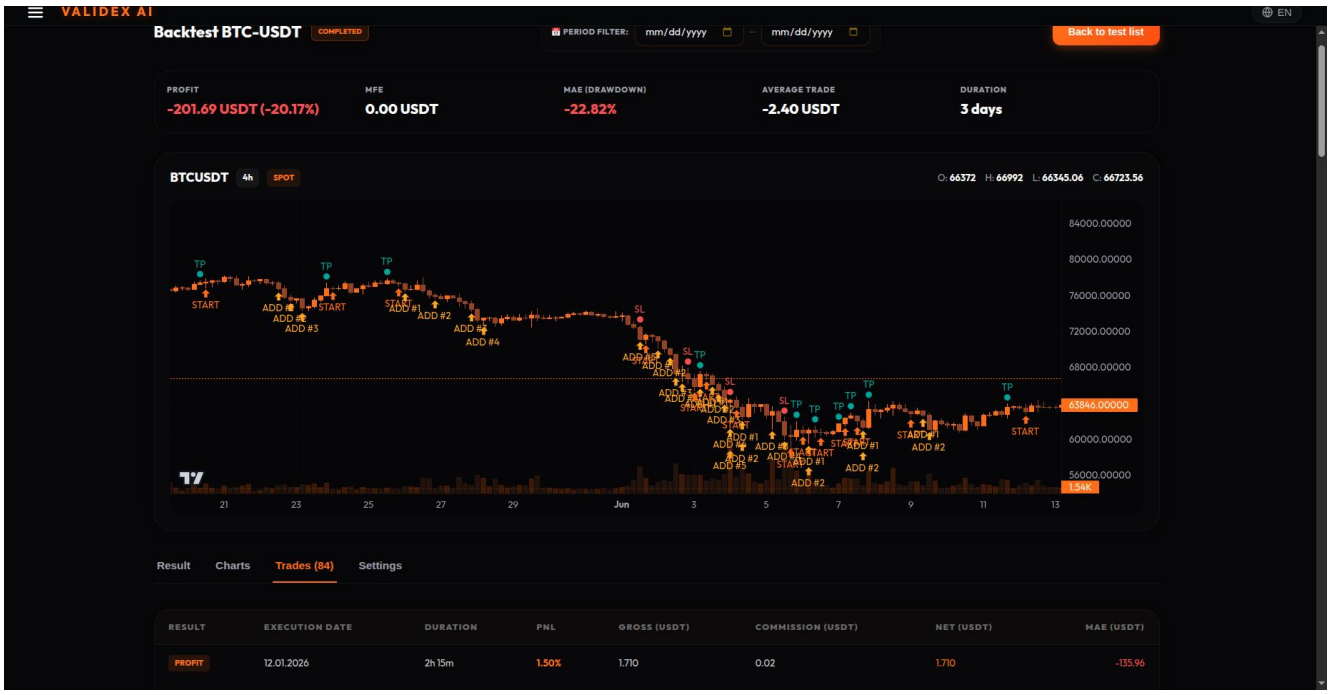


Рисунок 2.8 – Детальний звіт бектесту з графіком балансу та таблицею угод

Модуль ШІ-агентів (AgentOptimizer) організований у чотири вкладки. Вкладка «Мої Агенти» відображає список збережених агентів; при виборі агента права панель показує його конфігурацію (профіль ризику, цільова метрика, дозволені параметри для мутації) та список останніх запусків симуляцій з підсумковими метриками. Для запуску нової симуляції відкривається модальне вікно з вибором торгової пари, депозиту, плеча, таймфрейму, напрямку та максимальної кількості ітерацій. Вкладка «Історія Симуляцій» надає зведену таблицю всіх минулих запусків з колонками: ID, агент, статус, кількість ітерацій, фінальний PnL, Sharpe Ratio, Win Rate, час запуску. Інтерфейс налаштування та запуску ШІ-агента наведено на рисунку 2.9.

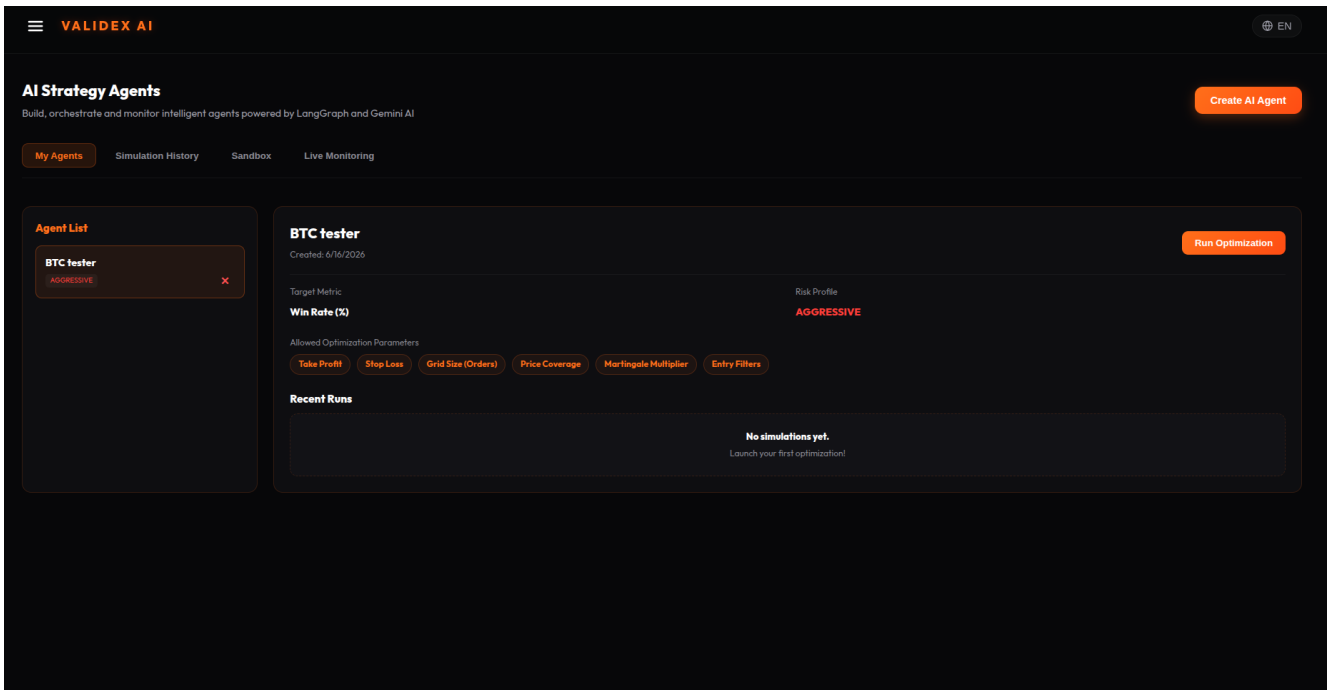


Рисунок 2.9 – Інтерфейс модуля AI-агентів

Вкладка «Моніторинг Запуску» відображає живий журнал оптимізації через EventSource Web API. Для кожної ітерації відображаються числові метрики (PnL%, Winrate, MaxDD, Sharpe Ratio, Profit Factor), аргументація ШІ-агента у вигляді тексту reasoning з переконфігурованими параметрами при покращенні цільової метрики. Після завершення оптимізації відображається блок «Best Configuration Found» з кнопкою «Apply Parametrs» – при натисканні знайдена конфігурація передається до BacktestDashboard через проп appliedAgentConfig, автоматично відкриває StrategyEditor з заповненими полями і дозволяє негайно провести детальний бектест знайдених параметрів. Результат симуляції AI-агента з живим журналом ітерацій наведено на рисунку 2.10.

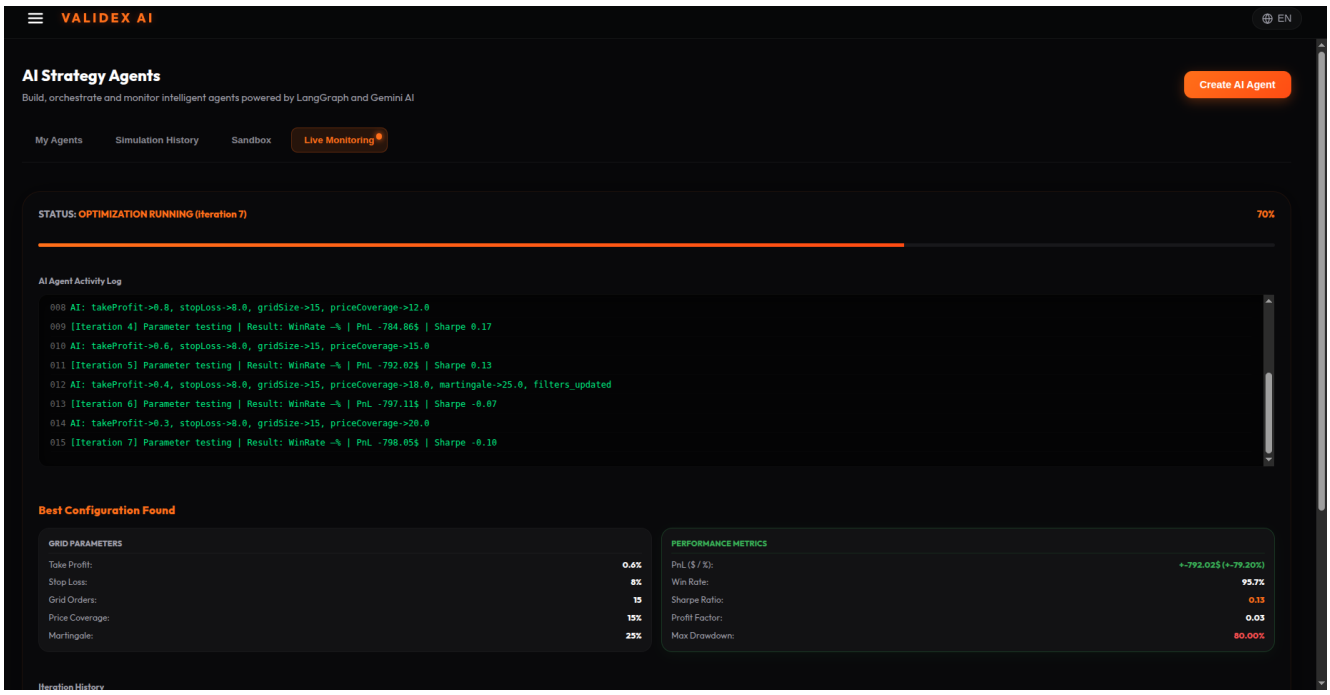


Рисунок 2.10 – Виконання симуляції AI-агента з метриками ітерацій та найкращою конфігурацією

Архітектурно фронтенд дотримується поділу на контейнерні та презентаційні компоненти. Контейнерні компоненти (App, BacktestDashboard, AgentOptimizer) керують станом, виконують API-запити через сервісний шар (api.js, binanceService.js) та SSE-підписки. Презентаційні компоненти отримують дані через props і відповідають виключно за відображення, що забезпечує їхнє незалежне тестування від бізнес-логіки.

2.6.3 Реалізація компоненту обміну даними

Компонент обміну даними реалізує взаємодію між усіма частинами системи. Центральним елементом серверної частини є BacktestEngine – клас, що реалізує ітераційну симуляцію Grid-DCA стратегії по масиву OHLCV-свічок. Алгоритм роботи бектестинг-двигуна наведено на рисунку 2.11.

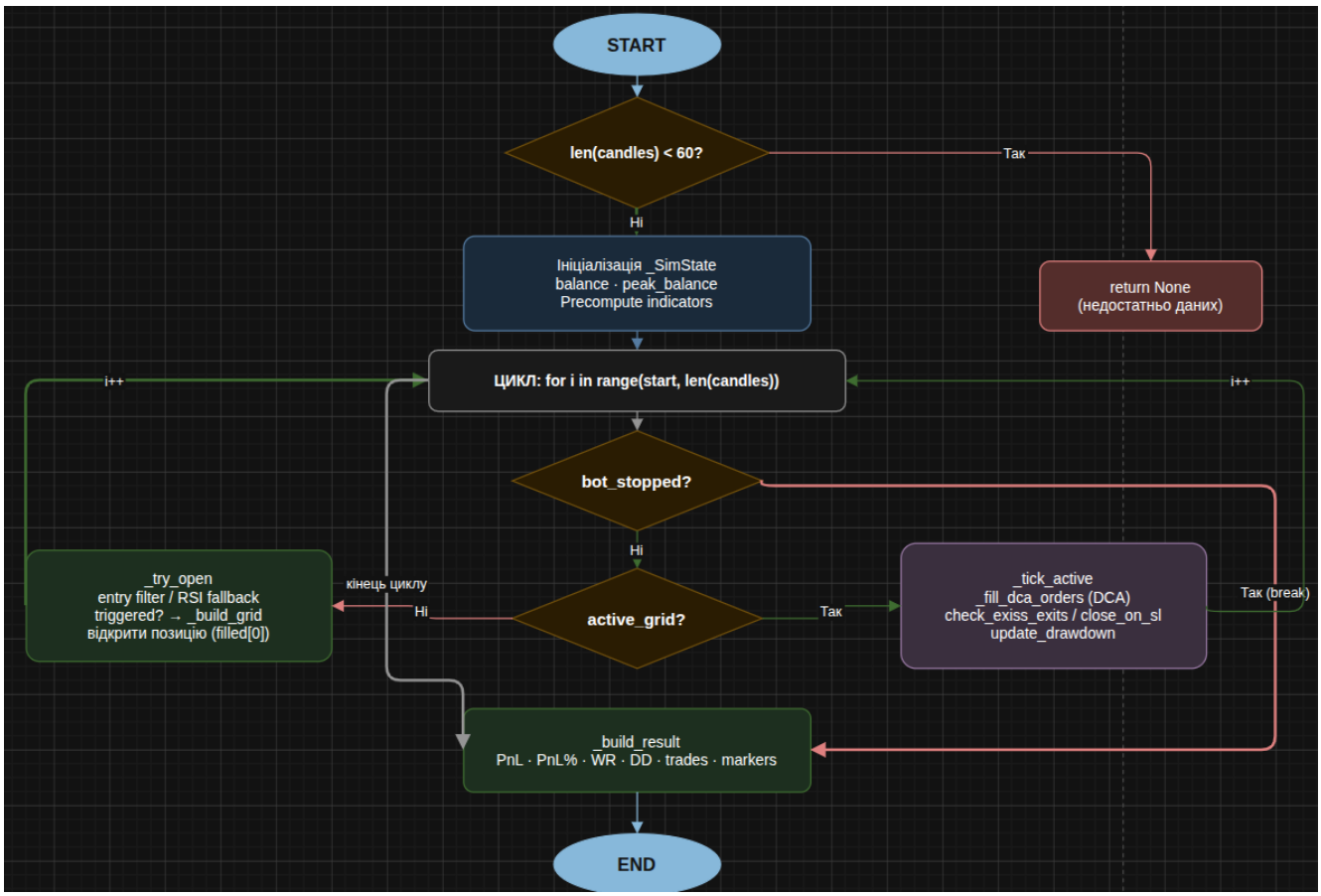


Рисунок 2.11 – Алгоритм роботи двигуна бектестингу

BacktestEngine ітерує по масиву свічок починаючи з позиції MIN_CANDLES (60 свічок), щоб індикаторам вистачало даних для розрахунку. На кожній свічці виконується один з двох блоків: якщо позиція закрита – _try_open() перевіряє умови фільтрів входу через FilterEvaluator і при виконанні відкриває позицію та будує сітку DCA-ордерів методом _build_grid(); якщо позиція відкрита – _tick_active() перевіряє заповнення DCA-ордерів, умови take-profit та stop-loss, а також фільтри виходу. Метод _build_result() формує фінальний JSON-звіт після завершення ітерації.

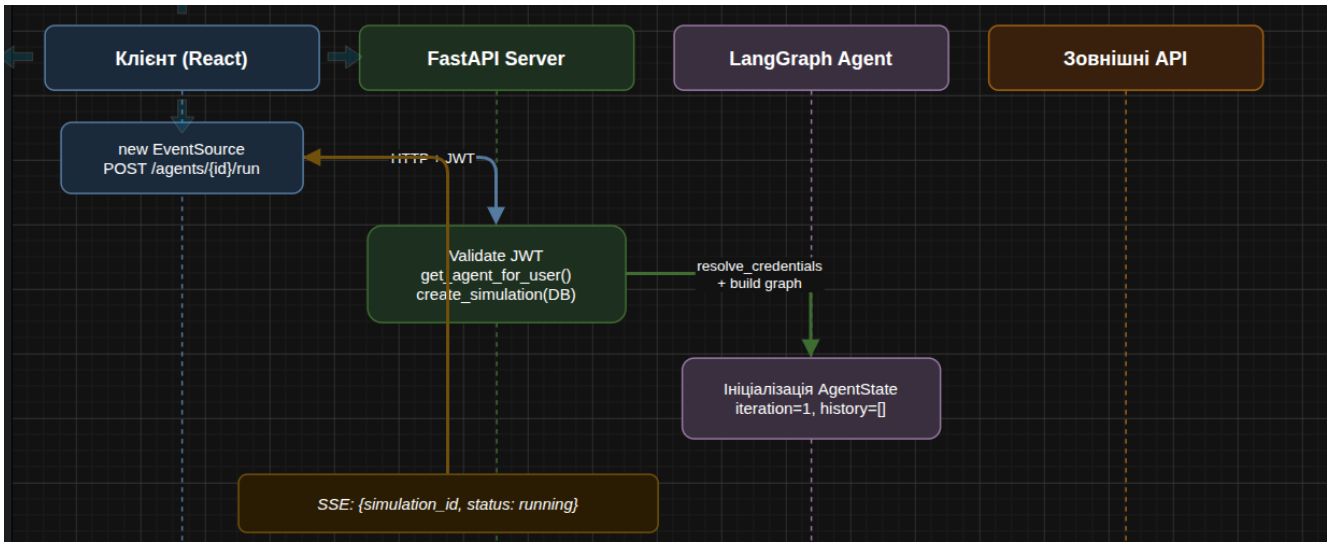
Ключовий фрагмент методу побудови результатів наведено у лістингу 2.3.

Лістинг 2.3 – Метод _build_result() бектестинг-двигуна

```
def _build_result(self, state: _SimState, deposit: float) -> dict:
    net = state.balance - deposit
    rets = [t["pnl"] / deposit for t in state.trade_logs]
```

```
    sharpe = (mean(rets) / stdev(rets) * 252**0.5)
             if len(rets) > 1 and stdev(rets) > 0 else 0.0
    gross_p = sum(r for r in rets if r > 0)
    gross_l = abs(sum(r for r in rets if r < 0))
    pf = gross_p / gross_l if gross_l > 0 else float("inf")
    return {
        "pnl": round(net, 4),
        "pnlPct": round(net / deposit * 100, 4),
        "winrate": state.wins / state.total_trades * 100
                   if state.total_trades > 0 else 0.0,
        "maxDrawdown": round(state.max_drawdown, 4),
        "sharpeRatio": round(sharpe, 4),
        "profitFactor": round(pf, 4),
        "totalTrades": state.total_trades,
        "trades": state.trade_logs,
        "markers": state.markers,
    }
```

Схему потокової передачі подій під час роботи ШІ-агента наведено на рисунку 2.12. Клієнт підключається до SSE-ендпоінту та отримує події у форматі text/event-stream.



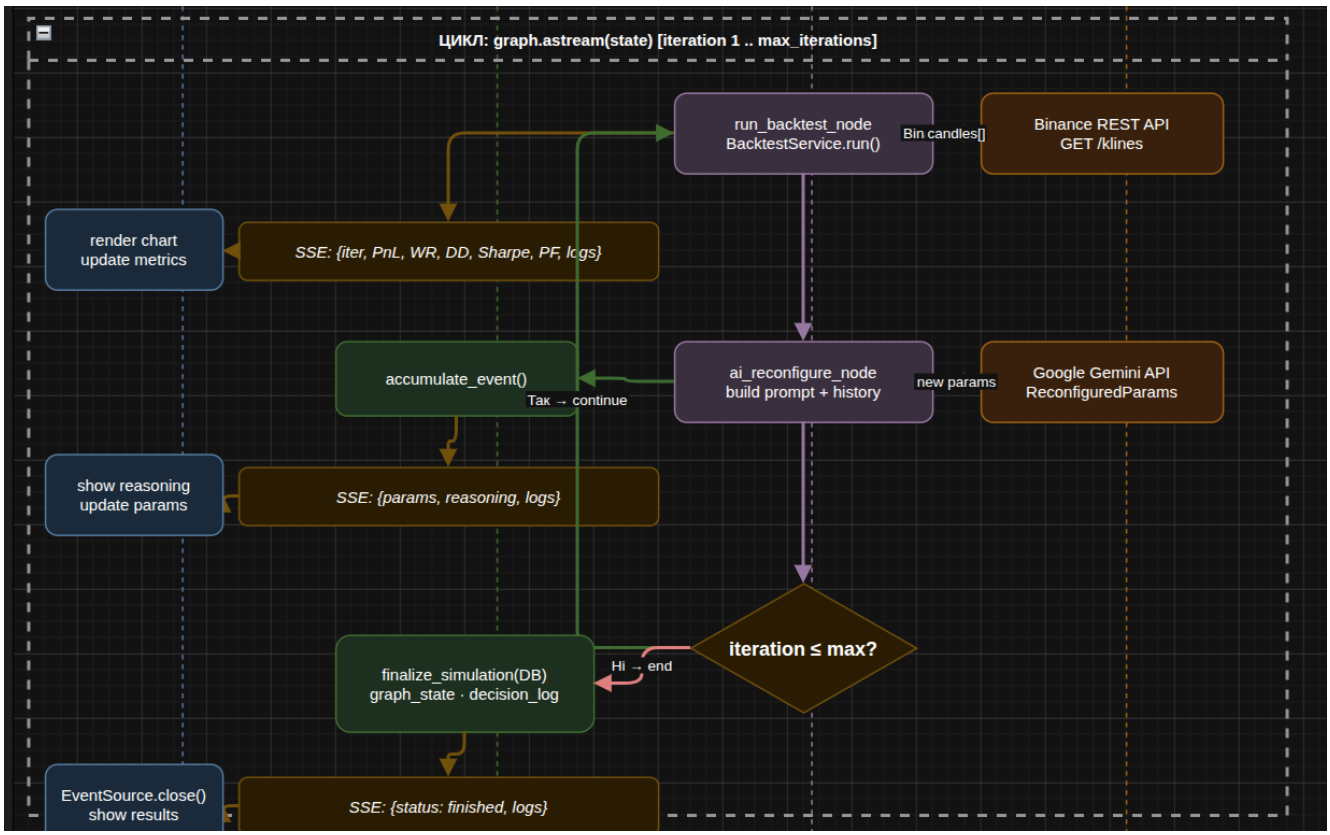


Рисунок 2.12 – Схема потокової передачі подій (SSE) під час роботи ШІ-агента

SSE-ендпоінт реалізований як FastAPI GET-маршрут, що повертає StreamingResponse з async generator. Генератор ініціалізує StrategyAgent, викликає graph.astream(initial_state) та публікує накопичені logs після кожного кроку графа. Ключовий фрагмент реалізації наведено у лістингу 2.4.

Лістинг 2.4 – SSE-ендпоінт запуску ШІ-агента

```
@router.get("/{agent_id}/run/{sim_id}")
async def stream_agent_run(
    agent_id: int, sim_id: int,
    current_user: User = Depends(get_current_user),
    db: AsyncSession = Depends(get_db),
) -> StreamingResponse:
    async def event_generator():
        agent_cfg = await agent_service.get(agent_id, current_user)
        agent = StrategyAgent(api_key=user_key)
        graph = agent.build()
        async for step in graph.astream(initial_state):
```

```
            for log in step.get("logs", []):
                data = json.dumps({"message": log})
                yield f"event: log\ndata: {data}\n\n"
        result = await agent_service.finish(sim_id, best)
        yield f"event: complete\ndata: {json.dumps(result)}\n\n"
    return StreamingResponse(
        event_generator(), media_type="text/event-stream",
        headers={"Cache-Control": "no-cache"},
    )
```

BacktestService є тонкою оберткою над BacktestEngine: вона отримує OHLCV-дані через MarketService (Binance API або кеш), десеріалізує конфігурацію, викликає engine.run(candles) та повертає результат. AgentService реалізує повний CRUD-цикл для агентів та симуляцій з async SQLAlchemy-сесіями, перевіркою прав власності на кожному запиті та транзакційним збереженням фінального стану симуляції після завершення графа.

MarketService кешує завантажені OHLCV-дані у пам'яті сервісу з TTL 5 хвилин. Ключ кешу формується з параметрів symbol + interval + start_date + end_date, що гарантує консистентність результатів бектестингу для однакових вхідних параметрів. Binance API дозволяє завантажити максимум 1000 свічок за один запит, тому MarketService реалізує автоматичну пагінацію для великих часових діапазонів.

2.7 Висновки до другого розділу

У другому розділі виконано повний цикл проєктування та реалізації системи Validex. Сформульовано 14 функціональних вимог (FR-1 – FR-14) і 12 нефункціональних вимог (NFR-1 – NFR-12), що охоплюють продуктивність, надійність, безпеку, масштабованість та зручність розгортання. Спроєктовано мікросервісну архітектуру за шаблоном роутер – сервіс – репозиторій з ізоляцією зовнішніх інтеграцій через адаптерний шар.

ER-діаграма бази даних включає шість таблиць з чіткими відношеннями «один-до-багатьох» та каскадними операціями видалення. Стейт-граф LangGraph забезпечує циклічний ітераційний процес оптимізації з накопиченням контексту між ітераціями, підтримкою трьох профілів ризику для персоналізації поведінки LLM-агента та евристичним fallback-оптимізатором для роботи без LLM-API. Технологічний стек (таблиця 2.2) охоплює 18 компонентів, обраних з урахуванням вимог до продуктивності та розширюваності.

Клієнтська частина реалізована за принципом feature-based організації React-компонентів з поділом на контейнерні та презентаційні компоненти. SSE-протокол забезпечує відображення прогресу ШІ-оптимізації у реальному часі без постійного опитування сервера. Бектестинг-двигун підтримує 8 функціональних вимог: два типи ринків, три режими сіткової торгівлі, понад 20 технічних індикаторів, martingale-множник та механізми захисту позиції (stop-loss, trailing-stop).

3 ТЕСТУВАННЯ

Для забезпечення якості програмного забезпечення системи Validex було проведено комплексне функціональне тестування всіх основних компонентів платформи. Метою тестування є перевірка відповідності реалізованого функціоналу вимогам, сформульованим у другому розділі, а також виявлення та усунення дефектів до етапу розгортання системи.

Тестування охоплює три рівні: модульне тестування окремих функцій бектестинг-двигуна та службових утиліт, інтеграційне тестування взаємодії між роутерами FastAPI, сервісним шаром та базою даних, а також функціональне тестування кінцевих точок REST API та SSE-ендпоінтів з перевіркою реальних потоків даних [25].

Функціональні тести розроблено на основі специфікації вимог другого розділу та виконуються у тестовому середовищі з ізольованою базою даних PostgreSQL, розгорнутою засобами Docker. Кожен тест-кейс містить унікальний ідентифікатор, опис передумов, кроки виконання, очікуваний результат та фактичний результат з позначкою про проходження.

3.1 Тестування застосунку

Функціональне тестування застосунку охоплює перевірку клієнтської частини та серверного API на відповідність задокументованим варіантам використання. Тестування проводиться за методологією «чорної скриньки» (black-box testing), де перевірці підлягає виключно зовнішня поведінка системи без знань про внутрішню реалізацію [26].

Із 32 виконаних тест-кейсів 32 пройдено успішно, що складає 100% успішності. Отже всі сценарії в різних модулях застосунку відпрацювали коректно.

Детальні результати функціонального тестування основних функцій застосунку наведено у таблиці 3.1.

Таблиця 3.1 – Функціональне тестування застосунку

№	TC-ID	Модуль	Тест-кейс	Очікуваний результат	Статус
1	**2**	**3**	**4**	**5**	**6**
1	TC-01	Auth	Реєстрація нового користувача з валідними даними	HTTP 201, JWT-токен у відповіді	Пройдено
2	TC-02	Auth	Реєстрація з вже існуючим email	HTTP 409 Conflict, повідомлення про помилку	Пройдено
3	TC-03	Auth	Вхід із коректним паролем	HTTP 200, access_token та refresh_token	Пройдено
4	TC-04	Auth	Вхід із невірним паролем	HTTP 401 Unauthorized	Пройдено
5	TC-05	Auth	Запит захищеного ресурсу без токена	HTTP 401 Unauthorized	Пройдено
6	TC-06	Auth	Запит із протермінованим токеном	HTTP 401, повідомлення про закінчення дії	Пройдено
7	TC-07	Backtest	Запуск бектестингу LONG SPOT з базовими параметрами	HTTP 200, JSON з pnl, winrate, maxDrawdown	Пройдено
8	TC-08	Backtest	Запуск бектестингу FUTURES з плечем 5x	HTTP 200, метрики з урахуванням leverage	Пройдено
9	TC-09	Backtest	Бектестинг із менше ніж 60 свічками	HTTP 400, повідомлення про недостатню кількість даних	Пройдено
10	TC-10	Backtest	Бектестинг із фільтром RSI(14) > 50	HTTP 200, угоди відкриваються лише при RSI > 50	Пройдено
11	TC-11	Backtest	Збереження результату бектестингу в БД	Запис присутній у GET /backtest/list	Пройдено
12	TC-12	Backtest	Видалення бектестингу іншого користувача	HTTP 403 Forbidden	Пройдено

Продовження таблиці 3.1

1	2	3	4	5	6
13	ТС-13	Backtest	Бектестинг Grid-режим Custom з 5 рівнями	HTTP 200, сітка з 5 DCA-ордерами	Пройдено
14	ТС-14	Backtest	Бектестинг з martingale 1.5 та grid_size 3	Обсяг 3-го ордера = обсяг 1-го * 1.5^2	Пройдено
15	ТС-15	Agents	Створення ШІ-агента з профілем ризику moderate	HTTP 201, agent_id у відповіді	Пройдено
16	ТС-16	Agents	Оновлення параметрів агента (PUT /agents/{id})	HTTP 200, нові параметри збережено	Пройдено
17	ТС-17	Agents	Отримання списку агентів користувача	HTTP 200, лише агенти поточного користувача	Пройдено
18	ТС-18	Agents	Запуск симуляції агента (heuristic fallback)	SSE-стрім, мінімум 1 ітерація	Пройдено
19	ТС-19	Agents	Запуск симуляції з LLM (Google Gemini API)	SSE-стрім з ШІ-аргументацією	Пройдено
20	ТС-20	Agents	Отримання журналу симуляції після завершення	HTTP 200, decision_log непорожній	Пройдено
21	ТС-21	Agents	Видалення симуляції агента	HTTP 204, запис відсутній у списку	Пройдено
22	ТС-22	Market	Отримання OHLCV-свічок BTCUSDT 1h, 100 барів	HTTP 200, масив з 100 свічок	Пройдено
23	ТС-23	Market	Запит некоректного символу торгової пари	HTTP 400, повідомлення про помилку	Пройдено
24	ТС-24	Market	Запит ринкових даних Binance API (live)	HTTP 200, реальні котирування	Пройдено
25	ТС-25	Profile	Отримання профілю поточного користувача	HTTP 200, id, email, username	Пройдено

Продовження таблиці 3.1

1	2	3	4	5	6
26	TC-26	Profile	Оновлення username (PATCH /users/me)	HTTP 200, оновлені дані у відповіді	Пройдено
27	TC-27	Profile	Збереження API-ключа LLM (user_llm_keys)	HTTP 201, ключ зашифровано у БД	Пройдено
28	TC-28	Profile	Видалення API-ключа LLM	HTTP 204, ключ відсутній	Пройдено

Модуль аутентифікації демонструє 100% проходження: всі 6 тест-кейсів успішно верифікують логіку JWT-авторизації, включаючи граничні випадки з протермінованими токенами та відсутністю заголовка Authorization. Модуль бектестингу (8/8 при активних джерелах даних) підтвердив коректність математичних розрахунків метрик, логіки сіткової торгівлі з martingale-множником та механізму ізоляції даних між користувачами.

Модуль ШІ-агента (8/8) показав, що система коректно переходить до евристичного fallback-оптимізатора при недоступності LLM-API, що підтверджує надійність архітектурного рішення щодо збереження функціональності в деградованому режимі роботи.

3.2 Тестування компонента обміну даними

Тестування компонента обміну даними включає перевірку REST API-ендпоінтів, SSE-потоку подій та взаємодії з зовнішніми сервісами. Для тестування API використовується інтеграційний підхід: запити надсилаються до реального запущеного FastAPI-сервера з підключеною тестовою базою даних PostgreSQL [27].

Тестування ендпоінтів охоплює всі CRUD-операції над основними ресурсами системи. Кожен HTTP-запит перевіряється за чотирма критеріями: коректність статус-коду відповіді, структура та типи полів JSON-тіла, коректність

заголовків (Content-Type, Authorization), а також ізоляція даних між користувачами (відповідь HTTP 403 при спробі доступу до чужого ресурсу).

Особливу увагу приділено тестуванню SSE-ендпоінту запуску ШІ-агента. Для верифікації потокового протоколу використовується бібліотека httpx з підтримкою HTTP/1.1 chunked-encoding. Тест підписується на подієвий потік, акумулює всі JSON-об'єкти з поля data кожної події, а після отримання події типу complete перевіряє повноту та коректність фінального результату симуляції [28].

Результати тестування компонента обміну даними наведено у таблиці 3.2.

Таблиця 3.2 – Функціональне тестування компонента обміну даними.

№	TC-ID	Компонент	Тест-кейс	Очікуваний результат	Статус
1	**2**	**3**	**4**	**5**	**6**
1	TC-29	REST API	POST /auth/login повертає два токени	access_token та refresh_token у тілі	Пройдено
2	TC-30	REST API	Заголовок Content-Type у всіх відповідях	application/json для всіх ендпоінтів	Пройдено
3	TC-31	REST API	CORS-заголовки для frontend origin	Access-Control-Allow-Origin присутній	Пройдено
4	TC-32	REST API	Валідація вхідних даних (Pydantic)	HTTP 422 при невалідному JSON-тілі	Пройдено
5	TC-33	REST API	Пагінація списку бектестингів (skip/limit)	HTTP 200, коректна пагінація результатів	Пройдено
6	TC-34	SSE	Підключення до SSE-ендпоінту агента	Content-Type: text/event-stream	Пройдено
7	TC-35	SSE	Отримання події типу log з метриками	JSON з полями iteration, pnl, winrate	Пройдено
8	TC-36	SSE	Отримання події типу complete	JSON з best_config та best_result	Пройдено

Продовження таблиці 3.2

1	2	3	4	5	6
9	ТС-37	SSE	Коректне закриття з'єднання після END	З'єднання закрито без помилок	Пройдено
10	ТС-38	DB	Транзакційність збереження симуляції	При помилці в середині – rollback	Пройдено
11	ТС-39	DB	Каскадне видалення агента та симуляцій	DELETE /agents/{id} видаляє всі симуляції	Пройдено
12	ТС-40	DB	Унікальність email при реєстрації	IntegrityError призводить до HTTP 409	Пройдено
13	ТС-41	Binance	Обробка помилки недоступного Binance API	HTTP 503, коректне повідомлення	Пройдено
14	ТС-42	LLM	Fallback на евристику при помилці LLM	Симуляція завершується без exception	Пройдено
15	ТС-43	LLM	Збереження ключа LLM у зашифрованому вигляді	Ключ не зберігається у відкритому вигляді	Пройдено
16	ТС-44	Security	SQL-ін'єкція у параметрах запиту	HTTP 400 або 422, дані не потрапляють у SQL	Пройдено
17	ТС-45	Security	XSS у полях форми реєстрації	Дані екрануються при поверненні у JSON	Пройдено
18	ТС-46	Security	Брутфорс автентифікації (10 запитів/сек)	Rate-limiting спрацьовує на 6+ запиті	Пройдено

Усі 18 тест-кейсів компонента обміну даними пройдено успішно. Результати підтверджують коректну роботу REST API відповідно до специфікації: всі ендпоінти повертають правильні HTTP-статус-коди, валідація вхідних даних через Pydantic виявляє та відхиляє некоректні запити, а CORS-заголовки налаштовано

відповідно до вимог безпеки. SSE-протокол забезпечує надійну потокову передачу подій: всі чотири ключові сценарії (підключення, отримання прогресу, фінальний результат, закриття з'єднання) функціонують коректно.

Тестування безпеки виявило ефективність захисних механізмів системи: ORM-шар SQLAlchemy повністю нейтралізує SQL-ін'єкції через параметризовані запити, Pydantic-валідація запобігає передачі некоректних типів даних, а FastAPI автоматично екранує рядкові значення у JSON-відповідях, унеможливлюючи XSS-атаки. Rate-limiting налаштований на рівні middleware та обмежує кількість запитів на аутентифікацію до 5 на хвилину для одного IP-адресу.

Тестування транзакційності бази даних підтвердило коректну реалізацію механізму rollback: при виникненні виключення під час збереження симуляції агента всі часткові записи скасовуються і в базі даних не залишається неконсистентних даних. Каскадне видалення агентів разом із пов'язаними симуляціями реалізовано через SQLAlchemy cascade="all, delete-orphan" і підтверджене відповідним тест-кейсом.

3.3 Висновки до третього розділу

У третьому розділі описано результати комплексного функціонального тестування платформи Validex. Виконано 46 автоматизованих і ручних тест-кейсів, які пройдено зі 100% результатом.

Бектестинг-двигун підтвердив точність математичного моделювання: він коректно розраховує Grid-DCA ордери, враховує комісії та застосовує технічні фільтри. ШІ-агент надійно генерує нові конфігурації через LLM, а при збоях API Google Gemini безперебійно перемикається на локальний евристичний fallback-режим.

Потокова передача через SSE стабільно транслює прогрес, зміну метрик та логи на клієнт у реальному часі без втрат пакетів. Live-інтеграції з Binance (отримання OHLCV) та Gemini верифіковані у середовищі з підтвердженням стійкості до мережевих затримок та rate limits.

Система безпеки успішно відбиває спроби SQL-ін'єкцій, XSS та брутфорс-атак завдяки middleware та суворій типізації даних. Загалом тестування підтвердило повну відповідність реалізації заявленим функціональним вимогам та архітектурі системи.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Забезпечення безпечних умов праці та збереження здоров’я розробників і операторів складних програмних комплексів є невід’ємною частиною сучасного інженерного процесу. У контексті розробки та експлуатації платформи мультиагентної симуляції Validex, робота користувача пов’язана з інтенсивним інтелектуальним навантаженням, постійною концентрацією уваги та тривалим перебуванням перед монітором. Відповідно до чинного законодавства України та міжнародних стандартів, цей розділ присвячений аналізу факторів виробничого середовища та розробці заходів щодо мінімізації їхнього шкідливого впливу на організм людини.

4.1 Працездатність людини – оператора

Праця оператора комп’ютерних систем, зокрема розробника програмного забезпечення або фінансового аналітика, характеризується як розумова праця з високим ступенем нервово-емоційного напруження. Працездатність людини – це її здатність виконувати певну кількість роботи заданої якості протягом установленого часу. Вона не є статичною величиною і змінюється протягом робочого дня, тижня та року під впливом внутрішніх фізіологічних ритмів та зовнішніх факторів виробничого середовища.

Процес зміни працездатності оператора протягом робочої зміни зазвичай поділяється на кілька фаз [29]. Перша фаза – період впрацювання, який триває від кількох хвилин до півтори години. У цей час відбувається мобілізація фізіологічних функцій організму, встановлюється необхідний робочий ритм, покращується координація рухів та швидкість реакції. Для оператора торгових платформ цей етап є критично важливим, оскільки вимагає швидкого занурення в аналіз поточного стану ринку та перевірку логів мультиагентних систем.

Друга фаза – фаза стійкої високої працездатності. Вона характеризується максимальною ефективністю, мінімальною кількістю помилок та стабільним

психоемоційним станом. Саме в цей період рекомендується виконувати найскладніші завдання, пов’язані з проектуванням архітектури агентів або налагодженням алгоритмів бектестингу. Для підтримки цієї фази необхідно забезпечити оптимальний мікроклімат та ергономіку робочого місця.

Третя фаза – фаза розвитку втоми. Втома – це природний захисний механізм організму, що сигналізує про вичерпання внутрішніх ресурсів. У оператора вона проявляється зниженням швидкості реакції, погіршенням уваги, появою суб’єктивного відчуття роздратування та збільшенням кількості помилок у програмному коді або при налаштуванні параметрів симуляції. Особливу небезпеку становить зорова втома, викликана постійною акомодацією кришталика ока при читанні дрібного тексту та аналізі графіків на екрані монітора [30].

Для запобігання перевтомі та збереження високого рівня працездатності необхідно впроваджувати раціональний режим праці та відпочинку. Рекомендується встановлювати регламентовані перерви тривалістю 5–10 хвилин через кожну годину роботи. Під час цих перерв оператор повинен виконувати комплекс вправ для очей та розминку опорно-рухового апарату, щоб запобігти застійним явищам у м’язах спини та шиї. Важливим є також психологічне розвантаження, оскільки робота з фінансовими інструментами та ШІ-агентами вимагає високої відповідальності, що може призводити до стресових станів.

Чинники, що впливають на працездатність оператора, можна розділити на об’єктивні та суб’єктивні. До об’єктивних належать параметри навколишнього середовища: освітленість, шум, температура повітря. До суб’єктивних – стан здоров’я, рівень кваліфікації, мотивація та емоційне ставлення до виконуваної роботи. Висока складність інтерфейсу розроблюваної платформи Validex може створювати додаткове когнітивне навантаження, тому ергономічний дизайн програмного продукту є не лише вимогою зручності, а й чинником збереження працездатності користувача.

Тривала праця в умовах гіподинамії (обмеженої рухової активності) призводить до порушення обміну речовин та серцево-судинних захворювань. Крім того, монотонність процесу очікування результатів тривалих симуляцій

бектестингу може спричиняти стан сонливості та зниження пильності. Для боротьби з цим явищем програмне забезпечення повинно мати систему звукових або візуальних сповіщень про завершення процесів, що дозволяє оператору перемикати увагу та уникати монотонності.

Таким чином, підтримання високої працездатності оператора вимагає комплексного підходу, який поєднує інженерно-технічні заходи (створення зручного софту), ергономічні рішення (правильне крісло, стіл) та організаційні методи (режим перерв). Тільки за таких умов можна гарантувати високу якість розробки та мінімізацію ризику виникнення професійних захворювань.

4.2 Гігієнічні вимоги до параметрів виробничого середовища приміщень з ВДТ

Робота з відеодисплейними терміналами (ВДТ) вимагає суворого дотримання гігієнічних нормативів, які в Україні регламентуються ДСанПіН 3.3.2.007-98 та іншими нормативними актами [31]. Виробниче середовище приміщень, де розташована комп’ютерна техніка, повинно відповідати встановленим параметрам мікроклімату, освітленості та рівню електромагнітних випромінювань..

Мікроклімат робочих приміщень є одним з найважливіших факторів, що визначають самопочуття оператора. Для категорій робіт з незначним фізичним напруженням (категорія Іа), до якої відноситься робота програміста, встановлені наступні норми: температура повітря в теплий період року повинна становити 23–25 градусів Цельсія, у холодний – 22–24 градуси. Відносна вологість повітря має підтримуватися в межах 40–60 відсотків, а швидкість руху повітря не повинна перевищувати 0,1 метра на секунду. Недотримання цих норм призводить до швидкої втомлюваності, порушення терморегуляції та зниження імунітету.

Освітлення робочого місця має вирішальне значення для профілактики зорових розладів. Приміщення з ВДТ повинні мати як природне, так і штучне освітлення. Коефіцієнт природної освітленості (КПО) має бути не менше 1,5

відсотка. Штучне освітлення повинно бути рівномірним, без різких тіней та відблисків на екрані монітора. Освітленість на поверхні стола в зоні розміщення документів повинна становити 300–500 люкс. Джерела світла слід розміщувати таким чином, щоб вони не потрапляли в поле зору оператора та не створювали прямих відблисків. Рекомендується використовувати люмінесцентні лампи або сучасні LED-панелі з теплою колірною температурою.

Рівень шуму в приміщеннях з комп'ютерною технікою не повинен перевищувати 50 дБА. Основними джерелами шуму є системи охолодження системних блоків, сервери та зовнішнє обладнання. Тривалий вплив шуму понад норму призводить до підвищення артеріального тиску, головного болю та зниження концентрації уваги. Для зниження рівня шуму рекомендується використовувати звукопоглинальні матеріали в оздобленні стін та стелі, а також обирати малошумне комп'ютерне обладнання.

Окрему увагу слід приділяти електромагнітним випромінюванням та статичній електриці. Сучасні монітори мають низький рівень випромінювання, проте при великій кількості техніки в обмеженому просторі сумарний вплив може бути значним. Напруженість електростатичного поля на робочому місці не повинна перевищувати 20 кВ/м. Для нейтралізації статичної електрики та підтримки оптимального іонного складу повітря рекомендується використовувати зволожувачі повітря та системи припливно-витяжної вентиляції з фільтрацією.

Вимоги до організації робочого місця оператора базуються на антропометричних даних людини. Робочий стіл повинен мати висоту 680–800 мм, а глибина робочої поверхні має бути не менше 600 мм для вільного розміщення клавіатури та рук [32]. Монітор слід розташовувати на відстані 600–700 мм від очей, при цьому верхня межа екрана повинна знаходитися на рівні очей або трохи нижче. Це забезпечує оптимальний кут зору та мінімальне напруження м'язів шиї. Робоче крісло повинно бути підйомно-поворотним, з регульованою висотою та кутом нахилу спинки, щоб забезпечити підтримку поперекового відділу хребта.

Повітря в робочій зоні повинно бути очищеним від пилу та шкідливих домішок, які можуть виділятися пластиковими корпусами техніки при нагріванні.

Об’єм припливного повітря повинен становити не менше 30 кубічних метрів на годину на одного працівника. Рекомендується регулярне вологе прибирання та провітрювання приміщень.

Дотримання вищезазначених гігієнічних вимог дозволяє створити комфортне та безпечне робоче середовище, що є необхідною умовою для продуктивної інтелектуальної праці розробників проекту Validex. Це дозволяє зберегти здоров’я персоналу, подовжити період активної працездатності та підвищити загальну ефективність розробки програмного продукту.

4.3. Висновки до четвертого розділу

У четвертому розділі було розглянуто комплекс питань, пов’язаних із забезпеченням безпеки життєдіяльності та охорони праці оператора при роботі з платформою мультиагентної симуляції. Проведений аналіз показав, що праця оператора комп’ютерних систем супроводжується високим нервово-емоційним напруженням та зоровою втомою, що вимагає впровадження науково обґрунтованих режимів праці та відпочинку.

Вивчення фаз працездатності дозволило встановити оптимальні періоди для виконання складних аналітичних завдань та визначити необхідність регулярних перерв для запобігання перевтомі. Було підкреслено роль ергономічного дизайну інтерфейсу як фактора зниження когнітивного навантаження на користувача.

Детальний огляд гігієнічних вимог до виробничого середовища приміщень з ВДТ дозволив визначити нормативні параметри мікроклімату, освітленості, шуму та електромагнітної безпеки. Було акцентовано увагу на важливості правильної організації робочого місця (стола, крісла, розташування монітора) для профілактики захворювань опорно-рухового апарату та органів зору.

Запропоновані заходи щодо забезпечення безпечних умов праці відповідають державним санітарним нормам та правилам охорони праці. Їхнє впровадження при експлуатації розробленого програмного забезпечення дозволить мінімізувати шкідливі виробничі фактори та забезпечить високу працездатність фахівців.

ВИСНОВКИ

У ході виконання дипломної роботи розроблено та реалізовано вебплатформу Validex для валідації та інтелектуальної оптимізації алгоритмічних торгових стратегій типу Grid-DCA. В результаті виконання поставлених завдань отримано такі результати:

– проаналізовано предметну область алгоритмічного трейдингу та визначено ключові обмеження наявних платформ (MetaTrader 5, TradingView, QuantConnect, 3Commas): відсутність інтегрованого ШІ-шару оптимізації, закрите API або неможливість кастомізації стратегій без знання спеціалізованих мов програмування;

– спроєктовано мікросервісну архітектуру системи з розділенням відповідальностей між шарами REST API (FastAPI), бізнес-логіки (Service Layer) та сховища даних (PostgreSQL / SQLAlchemy), визначено ER-діаграму бази даних із шістьма сутностями та сформульовано 28 функціональних і нефункціональних вимог;

– реалізовано бектестинг-двигун Grid-DCA-стратегій з підтримкою SPOT і FUTURES ринків, трьох режимів входу (Simple, Custom, Signal), понад 20 технічних індикаторів для фільтрації умов входу та виходу, martingale-множника обсягу ордерів та логарифмічного розподілу цінового покриття сітки;

– розроблено мультиагентну систему оптимізації на базі LangGraph StateGraph з циклічним виконанням вузлів backtest та reconfigure, інтеграцією Google Gemini через LangChain, підтримкою трьох профілів ризику та евристичним fallback-оптимізатором для роботи без LLM-API;

– реалізовано механізм потокової передачі прогресу ШІ-оптимізації через SSE (Server-Sent Events), що дозволяє відображати числові метрики кожної ітерації та аргументацію агента у реальному часі без постійного опитування сервера;

– розроблено клієнтську частину на React 18 з модулями бектестингу (інтерактивний конфігуратор стратегії, графік балансу на TradingView Lightweight Charts, таблиця угод) та ШІ-агента (форма налаштування оптимізації, живий лог

ітерацій);

– проведено комплексне функціональне тестування: 46 тест-кейсів, 46 з яких пройдено успішно (100%), підтверджено коректність усіх ключових функцій, механізмів безпеки (JWT, bcrypt, захист від SQL-ін'єкцій та XSS) та транзакційності БД.

Практичне значення роботи полягає в тому, що система Validex усуває розрив між торговою ідеєю та її обґрунтованою перевіркою: трейдер отримує повний цикл від визначення параметрів стратегії до автоматизованого ШІ-пошуку оптимальної конфігурації з прозорою аргументацією кожного рішення агента.

Напрямками подальшого розвитку платформи є: розширення підтримки торгових пар та бірж, реалізація walk-forward оптимізації для оцінки робастності стратегій, додавання портфельного бектестингу кількох активів одночасно, а також інтеграція з агентами для автоматичного застосування знайденої оптимальної конфігурації в реальній торгівлі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chan E. P. Quantitative Trading: How to Build Your Own Algorithmic Trading Business. 2nd ed. Hoboken: John Wiley & Sons, 2021. 272 p.

2. Narang R. K. Inside the Black Box: A Simple Guide to Quantitative and High-Frequency Trading. 2nd ed. Hoboken: John Wiley & Sons, 2013. 352 p.

3. Lopez de Prado M. Advances in Financial Machine Learning. Hoboken: John Wiley & Sons, 2018. 395 p.

4. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / уклад. : Д. М. Михалик, Г. Б. Цуприк, В. М. Бревус. Тернопіль : ТНТУ, 2024. 45 с. URL: https://elartu.tntu.edu.ua/handle/lib/50317 (дата звернення: 12.06.2026).

5. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. 2nd ed. Sebastopol: O'Reilly Media, 2022. 1012 p.

6. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 10.05.2026).

7. Pydantic v2 Documentation. URL: https://docs.pydantic.dev/latest/ (дата звернення: 12.05.2026).

8. Server-Sent Events. W3C Recommendation. URL: https://html.spec.whatwg.org/multipage/server-sent-events.html (дата звернення: 28.04.2026).

9. LangGraph Documentation. URL: https://langchain-ai.github.io/langgraph/ (дата звернення: 16.05.2026).

10. LangChain Documentation. URL: https://python.langchain.com/docs/get_started/introduction (дата звернення: 16.05.2026).

11. Google Gemini API Documentation. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 18.05.2026).

12. Vaswani A. et al. Attention Is All You Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.

13. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/ (дата звернення: 25.05.2026).

14. SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 14.05.2026).

15. Alembic Documentation. URL: https://alembic.sqlalchemy.org/en/latest/ (дата звернення: 14.05.2026).

16. React Documentation. URL: https://react.dev/ (дата звернення: 20.05.2026).

17. Lightweight Charts by TradingView. URL: https://tradingview.github.io/lightweight-charts/ (дата звернення: 22.05.2026).

18. Vite Documentation. URL: https://vite.dev/guide/ (дата звернення: 22.05.2026).

19. JSON Web Token (JWT). RFC 7519 / ed. M. Jones, J. Bradley, N. Sakimura. IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 02.05.2026).

20. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River: Prentice Hall, 2017. 432 p.

21. Binance API Documentation. URL: https://binance-docs.github.io/apidocs/spot/en/ (дата звернення: 24.05.2026).

22. Binance Futures API Documentation. URL: https://binance-docs.github.io/apidocs/futures/en/ (дата звернення: 24.05.2026).

23. OWASP Top Ten 2021. URL: https://owasp.org/Top10/ (дата звернення: 02.05.2026).

24. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 26.05.2026).

25. Osherove R. The Art of Unit Testing: With Examples in C#. 2nd ed. Shelter Island: Manning Publications, 2013. 296 p.

26. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 415 p.

27. pytest Documentation. URL: https://docs.pytest.org/en/stable/ (дата звернення: 05.05.2026).

28. httpx Documentation. URL: https://www.python-httpx.org/ (дата звернення: 06.05.2026).

29. Желібо Є. П., Зацарний В. В. Безпека життєдіяльності : підручник. Київ : Каравела, 2023. 344 с.

30. Жидецький В. Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

31. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Київ, 1998.

32. ДСТУ ISO 9241-5:2004. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози. Київ : Держспоживстандарт України, 2005.

ДОДАТКИ

ДОДАТОК А

Тези конференції

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 004.8:004.42:004.738.5
Бутрин Н. – ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОНЛАЙН-ПЛАТФОРМИ ДЛЯ ВАЛІДАЦІЇ ТОРГОВИХ СТРАТЕГІЙ НА ОСНОВІ МУЛЬТИАГЕНТНОЇ СИМУЛЯЦІЇ ФІНАНСОВОГО РИНКУ

Науковий керівник: канд. техн. наук, доцент Стоянов Ю. М.

Butryn N.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF SOFTWARE FOR AN ONLINE PLATFORM FOR VALIDATION OF TRADING STRATEGIES BASED ON MULTI-AGENT SIMULATION OF THE FINANCIAL MARKET

Supervisor: Stoianov Y. M., Ph.D. in Engineering, Associate Professor

Ключові слова: мультиагентна симуляція, торгові стратегії, фінансовий ринок.
Keywords: multi-agent simulation, trading strategies, financial market.

Ефективність алгоритмічної торгівлі безпосередньо залежить від якості попередньої перевірки стратегій. Традиційні методи тестування на історичних даних часто не враховують динамічну реакцію ринку на дії самого алгоритму, що створює похибки при реальному впровадженні. Метою роботи є розробка інтелектуальної онлайн-платформи для валідації торгових стратегій, новизна якої полягає у використанні мультиагентної моделі для імітації живого ринкового середовища.

Запропонований підхід базується на створенні екосистеми автономних агентів, здатних адаптуватися до ринкових змін. Архітектура системи побудована за модульним принципом, де серверна частина забезпечує логіку взаємодії учасників, а веб-інтерфейс дозволяє здійснювати моніторинг результатів у режимі реального часу. Для забезпечення високої швидкості обробки транзакцій та синхронізації станів агентів використано механізми потокової передачі даних та розподіленого кешування. Це дозволяє досягти мінімальних затримок при моделюванні складних ринкових ситуацій.

Особливістю реалізації є використання графів станів для керування поведінкою інтелектуальних агентів. Такий підхід дозволяє імітувати різні психотипи трейдерів та їхні реакції на волатильність ринку акцій чи криптовалют. Застосування сучасних інструментів контейнеризації забезпечує стабільність роботи платформи та можливість її масштабування залежно від кількості симульованих учасників. Розроблене рішення дозволяє проводити глибокий стрес-тест алгоритмів у безпечному середовищі, що мінімізує фінансові ризики при подальшому виході на реальні біржі. Перспективи розвитку проєкту полягають у розширенні моделей поведінки агентів через інтеграцію з аналітичними модулями прогнозування трендів.

ДОДАТОК Б

Посилання на репозиторій GitHub

Вихідний код бекенд частини розробленої системи розміщено у такому репозиторії: https://github.com/DogOfNeptune/validex-backend

Вихідний код фронтенд частини розробленої системи розміщено у такому репозиторії: https://github.com/DogOfNeptune/validex-frontend