Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

(підпис)

«___» _____ 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

освітньо-професійного ступеня «фаховий молодший бакалавр»

зі спеціальності 122 «Комп’ютерні науки»

на тему: «Реалізація автоматизованої системи торгівлі на фінансовому ринку»

Студент групи КН-41	Сергій ОСТАП’ЮК	_____ (підпис)
Керівник роботи	Наталія СИРОТЮК	_____ (підпис)
Консультанти: з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Наталія КУЛЬЧИНСЬКА	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення комп’ютерних технологій

Наталія СТЕФУРАК /_____/

(підпис)

«___» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студенту Остап’юку Сергію Сергійовичу

(прізвище, ім’я та по-батькові студента)

1. Тема роботи Реалізація автоматизованої системи торгівлі на фінансовому ринку

затверджена наказом по коледжу від “25” листопада 2024 р., № 253а-н

2. Термін здачі студентом завершеної роботи “___” _____202_ р.

3. Вихідні дані до роботи Аналіз існуючих торгових платформ та симуляторів, історичні котирування фінансових активів, реалізована система

4. Перелік питань, які повинні бути розроблені:

а) основна частина аналіз предметної області та існуючих рішень, постановка завдання, проєктування, реалізація і тестування системи.

б) техніко-економічне обгрунтування аналіз ринку збуту продукту, розрахункова частина та обґрунтування необхідності розробки

5. Перелік графічного матеріалу UML-діаграма, схема архітектури системи, діаграма послідовності взаємодій, схема бази даних, макети сторінок

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
З техніко-економічного обґрунтування	Меленчук Л. І. (вчена ступінь, звання П.І.Б. _____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1	Вибір теми кваліфікаційної роботи, ознайомлення з вимогами до кваліфікаційної роботи	10.11.2024	24.12.2024
2	Аналіз існуючих рішень і написання відповідного розділу	25.12.2024	27.02.2025
3	Аналіз та вивчення технологій реалізації.	28.02.2025	04.03.2025
4	Робота над стурктурою програмного продукту, розробка функціональних вимог. Написання відповідного розділу	05.03.2025	01.04.2025
5	Налаштування середовища реалізації	02.04.2025	04.04.2025
6	Встановлення та налаштування середовища розробки.	05.04.2025	22.04.2025
7	Проєктування системи	23.04.2025	28.04.2025
8	Реалізація та тестування системи	29.04.2025	18.05.2025
9	Опрацювання економічної частини. Написання відповідного розділу.	19.05.2025	28.05.2025
10	Оформлення пояснювальної записки.	29.05.2025	15.06.2025
11	Попередній захист кваліфікаційної роботи.	13.06.2025	13.06.2025
12	Підготовка до захисту кваліфікаційної роботи.	14.06.2025	24.06.2025
13	Захист кваліфікаційної роботи.	25.06.2025	25.06.2025

Дата видачі “25” листопада 2024 р. Керівник _____/ Сиротюк Н.С.

Завдання прийняла до виконання _____/ Остап’юк С.С.

Реферат

Реалізація автоматизованої системи торгівлі на фінансовому ринку. Кваліфікаційна робота. Остап’юк Сергій Сергійович. Галицький фаховий коледж імені В’ячеслава Чорновола, відділення комп’ютерних технологій. Спеціальність 122 «Комп’ютерні науки», 2025. Сторінок – 60, рисунків – 18, додатків – 6, джерел – 8.

Об’єктом проєктування є автоматизована система для моделювання й тестування торгових стратегій.

Метою проєкту є розробка вебзастосунку, що дозволяє користувачам без ризику оцінювати ефективність підходів до торгівлі на основі історичних фінансових даних.

У роботі повинні застосовуватись методи аналізу часових рядів, моделювання торгових стратегій, обробки даних, а також візуалізації результатів.

Новизна полягає в поєднанні візуального інтерфейсу з можливістю тестування стратегій у симульованому середовищі.

У реалізації використано PHP, HTML/CSS, JavaScript та реляційну базу даних MariaDB з таблицями для користувачів, котирувань, стратегій і результатів тестування.

Платформа рекомендована для трейдерів-початківців та дослідників у сфері фінансів. Подальший розвиток передбачає інтеграцію реальних біржових API та розширення інструментів технічного аналізу.

АВТОМАТИЗОВАНА ТОРГІВЛЯ, ФІНАНСОВИЙ РИНОК, ТОРГОВІ СТРАТЕГІЇ, СИМУЛЯЦІЯ, PHP, БАЗА ДАНИХ, КОТИРУВАННЯ, ТЕСТУВАННЯ СТРАТЕГІЙ.

Abstract

Implementation of an automated trading system in the financial market. Qualification work. Ostapyuk Serhii Serhiiovych. Vyacheslav Chornovil Halytsky College, Department of Computer Technologies. Speciality 122 «Computer Science», 2025. Pages - 60, figures - 18, appendices - 6, sources - 8.

The object of the project is an automated system for modelling and testing trading strategies.

The aim of the project is to develop a web application that allows users to risk-free evaluate the effectiveness of trading approaches based on historical financial data.

The work should use time series analysis, trading strategy modelling, data processing, and results visualisation methods.

The novelty lies in the combination of a visual interface with the ability to test strategies in a simulated environment.

The implementation uses PHP, HTML/CSS, JavaScript, and a MariaDB relational database with tables for users, quotes, strategies, and test results.

The platform is recommended for beginner traders and researchers in the field of finance. Further development includes the integration of real exchange APIs and the expansion of technical analysis tools.

AUTOMATED TRADING, FINANCIAL MARKET, TRADING STRATEGIES, SIMULATION, PHP, DATABASE, QUOTES, STRATEGY TESTING.

ЗМІСТ

Вступ 7

1 Аналіз предметної області та постановка завдань 8

1.1 Опис предметної області 8

1.2 Аналіз наявних рішень 9

1.3 Аналіз вимог до програмного засобу та постановка завдання 15

2 Проєктування системи 18

2.1 Проєктування системи та бази даних 18

2.2 Моделі та алгоритми 24

2.3 Аналіз засобів реалізації 25

2.4 Проєктування інтерфейсу системи 27

3 Реалізація та тестування системи 30

3.1 Розробка системи 30

3.2 Тестування системи 47

4 Техніко-економічне обґрунтування 51

4.1 Аналіз ринку збуту продукту 51

4.2 Розрахункова частина 52

4.3 Обґрунтування необхідності розробки 57

Висновки 59

Перелік джерел посилання 60

Додатки 61

СКОРОЧЕННЯ І УМОВНІ ПОЗНАКИ

API – Application Programming Interface

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

JS – JavaScript

PHP – Hypertext Preprocessor

SQL – Structured Query Language

UML –Unified Modeling Language

UI – User Interface

XAMPP – Cross-Platform (X), Apache, MariaDB, PHP, Perl

ВСТУП

Тема «Реалізація автоматизованої системи торгівлі на фінансовому ринку» є надзвичайно актуальною в умовах стрімкого розвитку цифрових технологій та зростаючого інтересу до фінансових інструментів з боку широкого кола користувачів. На сучасному етапі фінансові ринки демонструють динамічність, глобальність та складність, що потребує впровадження високотехнологічних рішень для ефективної роботи трейдерів і інвесторів. Автоматизовані торгові системи дозволяють підвищити точність, швидкість та ефективність ухвалення рішень, мінімізувати вплив людського чинника та оптимізувати торгові стратегії на основі аналізу великих обсягів даних у реальному часі.

Значущість теми також зумовлена постійним розширенням ринку фінансових послуг, появою нових цифрових платформ, активним використанням алгоритмічної торгівлі та необхідністю забезпечення стабільної роботи таких систем у режимі 24/7.

Метою кваліфікаційної роботи є розробка і впровадження автоматизованої системи торгівлі на фінансовому ринку, яка забезпечуватиме безперервний аналіз ринкових даних, прийняття торгових рішень на основі заданих стратегій та виконання операцій у реальному часі.

Об'єктом дослідження є процеси автоматизованої торгівлі на фінансовому ринку. Предметом є програмне забезпечення, алгоритми прийняття рішень та технічні засоби реалізації автоматизованих торгових стратегій.

Отримані результати можуть бути використані у сфері фінансових технологій, зокрема в компаніях, що займаються трейдингом, інвестуванням, розробкою програмного забезпечення для фінансових ринків, а також у наукових дослідженнях, пов’язаних з алгоритмічною торгівлею та машинним навчанням у фінансовому секторі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Опис предметної області

Предметна область, що розглядається в межах цієї кваліфікаційної роботи, охоплює діяльність на фінансовому ринку, зокрема процеси торгівлі цінними паперами, валютами, деривативами та іншими фінансовими інструментами. Фінансовий ринок є складною динамічною системою, в якій щодня здійснюється велика кількість операцій купівлі-продажу, приймаються стратегічні рішення на основі аналізу ринкових даних, економічних показників та новин. Сучасні умови вимагають від учасників ринку високої швидкості реагування, аналітичної гнучкості та здатності до прогнозування − саме тому автоматизація процесів торгівлі є ключовим напрямком удосконалення [1].

Традиційна торгівля, що здійснюється вручну, має низку обмежень, серед яких – людський фактор, емоційність, повільність ухвалення рішень та нездатність оперативно обробляти великі обсяги ринкових даних. Унаслідок цього підвищується ймовірність фінансових втрат, зменшується ефективність роботи трейдера і зростає потреба у впровадженні автоматизованих рішень. З розвитком обчислювальної техніки, алгоритмів машинного навчання, нейронних мереж і хмарних технологій з’явилися передумови для створення складних автоматизованих систем, здатних проводити фінансові операції без прямого втручання людини.

Розробка автоматизованої системи торгівлі дозволяє підвищити точність і швидкість виконання біржових угод, зменшити ризики, пов’язані з людськими помилками, а також забезпечити безперервний моніторинг ринку. Така система здатна аналізувати історичні та поточні ринкові дані, визначати потенційно прибуткові моменти для входу або виходу з позиції, оптимізувати розміщення капіталу та здійснювати угоди за заздалегідь визначеними алгоритмами. У зв’язку з цим автоматизовані торгові системи набувають дедалі більшого поширення серед приватних трейдерів, фінансових компаній та інвестиційних фондів.

Аналіз предметної області виявив низку актуальних проблем, серед яких – необхідність у створенні доступного, гнучкого та надійного інструменту, який би поєднував у собі функції збору, обробки, аналізу ринкових даних та безпосередньої торгівлі. У центрі уваги − підвищення ефективності торгівлі та автоматизація рутинних операцій. Також актуальним є забезпечення прозорості, відстежуваності виконаних операцій та захисту від зовнішніх загроз. У межах даної кваліфікаційної роботи буде запропоновано рішення, яке дозволяє вирішити вказані проблеми за допомогою сучасного програмного забезпечення, орієнтованого на стабільну роботу в режимі реального часу, адаптивну торгівельну логіку та підтримку масштабованості системи.

1.2 Аналіз наявних рішень

У межах аналізу предметної області автоматизованої торгівлі на фінансовому ринку було досліджено низку існуючих рішень, які вже активно використовуються трейдерами, інвестиційними компаніями та фінансовими аналітиками по всьому світу. Основною метою цього аналізу є вивчення можливостей, архітектурних підходів, функціональних особливостей та недоліків найпопулярніших програмних засобів, що дозволяє врахувати найкращі практики та уникнути типових помилок під час реалізації власної автоматизованої системи торгівлі.

Серед найбільш відомих рішень у цій галузі можна виокремити MetaTrader 5, NinjaTrader, cTrader та QuantConnect. Кожна з цих платформ має власні особливості й орієнтована на певну цільову аудиторію – від приватних трейдерів до професійних алгоритмічних розробників.

На Рисунку 1.1 зображено головне вікно торгового термінала MetaTrader 5. Як видно, інтерфейс програми орієнтований на зручність користувача: графіки, індикатори, панель ордерів і список активів винесені на основний екран [2]. MetaTrader 5, розроблений компанією MetaQuotes, є одним з найпоширеніших інструментів у світі роздрібної торгівлі. Система підтримує написання автоматичних стратегій за допомогою мови MQL5, що є

спеціалізованою мовою для трейдингових скриптів. Незважаючи на зручність і розповсюдженість, платформа має обмежену гнучкість у створенні складних алгоритмічних стратегій.

Рисунок 1.1 – Інтерфейс MetaTrader 5 із відкритим графіком активу та вікном торгівлі

MetaTrader 5 також має мобільну версію, доступну для Android та iOS, що дозволяє трейдерам здійснювати торгівлю, переглядати графіки та новини, але не дає змоги створювати або редагувати складні торгові стратегії.

На Рисунку 1.2 представлено середовище розробки стратегій у платформі NinjaTrader. Дана платформа, розроблена NinjaTrader LLC, надає розширені можливості для створення користувацьких індикаторів та роботів на мові C#, а також глибокий технічний аналіз даних [3]. Професійно орієнтований інтерфейс дозволяє точне налаштування параметрів стратегії, але може бути складним для новачків. Крім того, повна функціональність доступна лише у платній версії.

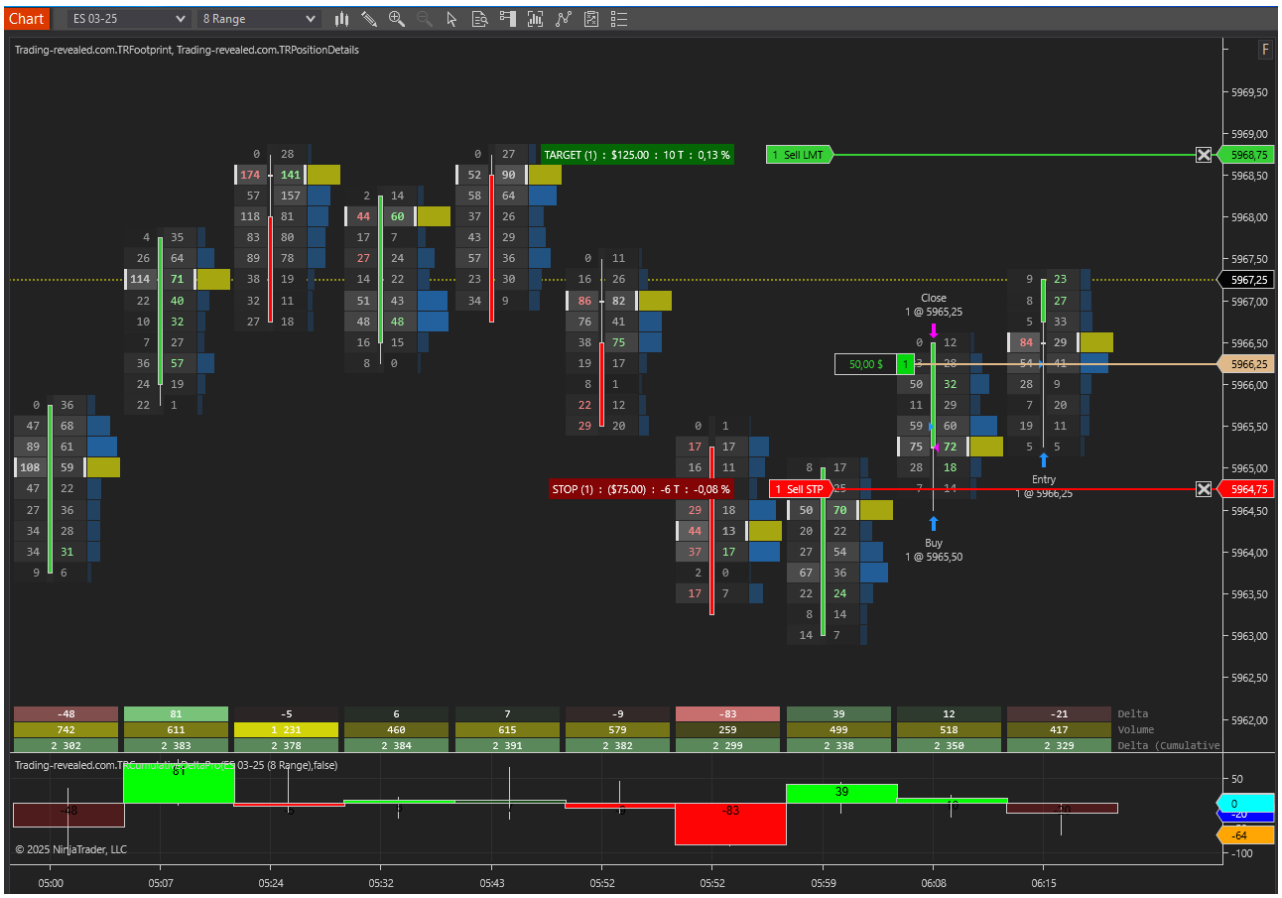


Рисунок 1.2 – Розробка стратегії в NinjaTrader

NinjaTrader не має офіційної мобільної версії і орієнтований виключно на десктопну платформу для професійних трейдерів, що може бути значним недоліком для багатьох користувачів.

Рисунок 1.3 демонструє торгове середовище cTrader, створене компанією Spotware Systems Ltd. Платформа орієнтована на інтерактивну торгівлю та автоматизацію за допомогою мови C# та інструмента cAlgo. Вона має сучасний інтерфейс, зручну навігацію та підтримує велику кількість вбудованих індикаторів, однак обмежена у гнучкості створення складних алгоритмічних логік, які виходять за межі стандартних шаблонів.

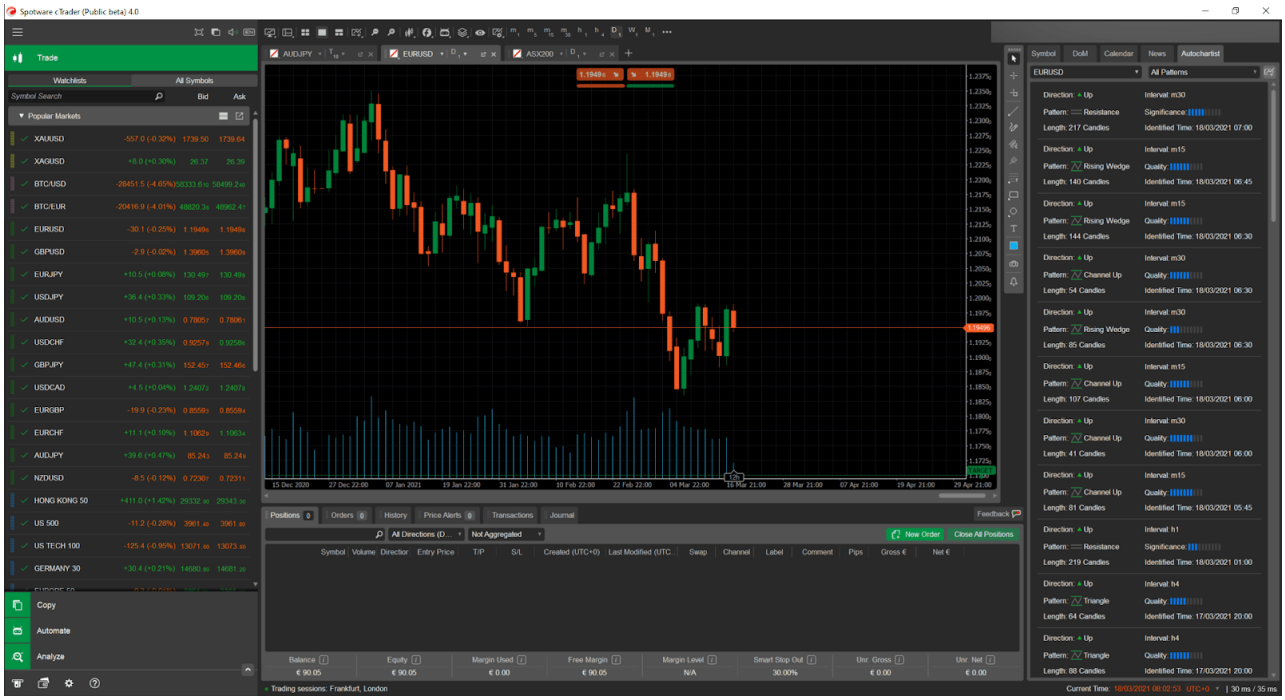


Рисунок 1.3 – Головне вікно cTrader із візуалізацією графіка та панеллю стратегії

cTrader має мобільну версію для Android та iOS, що дозволяє здійснювати торгівлю та переглядати графіки, хоча редагування автоматичних стратегій можливо тільки на десктопі.

QuantConnect – це хмарна платформа для алгоритмічної торгівлі, яка використовує фреймворк Lean та підтримує розробку торгових стратегій на мовах програмування C# та Python [4]. Платформа дозволяє трейдерам і розробникам створювати, тестувати та розгортати алгоритмічні стратегії на основі великих обсягів історичних даних, проводячи ефективні бек-тести. Зокрема, QuantConnect підтримує хмарне розгортання, що дозволяє запускати стратегії на віддалених серверах, зменшуючи навантаження на локальні ресурси. Крім того, платформа має потужне API, яке забезпечує інтеграцію з різними джерелами фінансових даних, брокерами та іншими інструментами для автоматизованої торгівлі.

На рисунку 1.4 подано приклад інтерфейсу QuantConnect IDE з відкртою вкладкою стратегій. Інтерфейс користувача цієї платформи є досить мінімалістичним і орієнтованим на технічних спеціалістів. Це означає, що для

ефективної роботи з платформою необхідно мати високий рівень технічної підготовки. Інтерфейс не надає інтуїтивно зрозумілих візуальних інструментів для створення стратегій, а весь процес відбувається через написання коду, що може бути складним для новачків. Платформа орієнтована, переважно, на професіоналів, які мають досвід програмування та хочуть працювати з великими обсягами фінансових даних і складними алгоритмами.

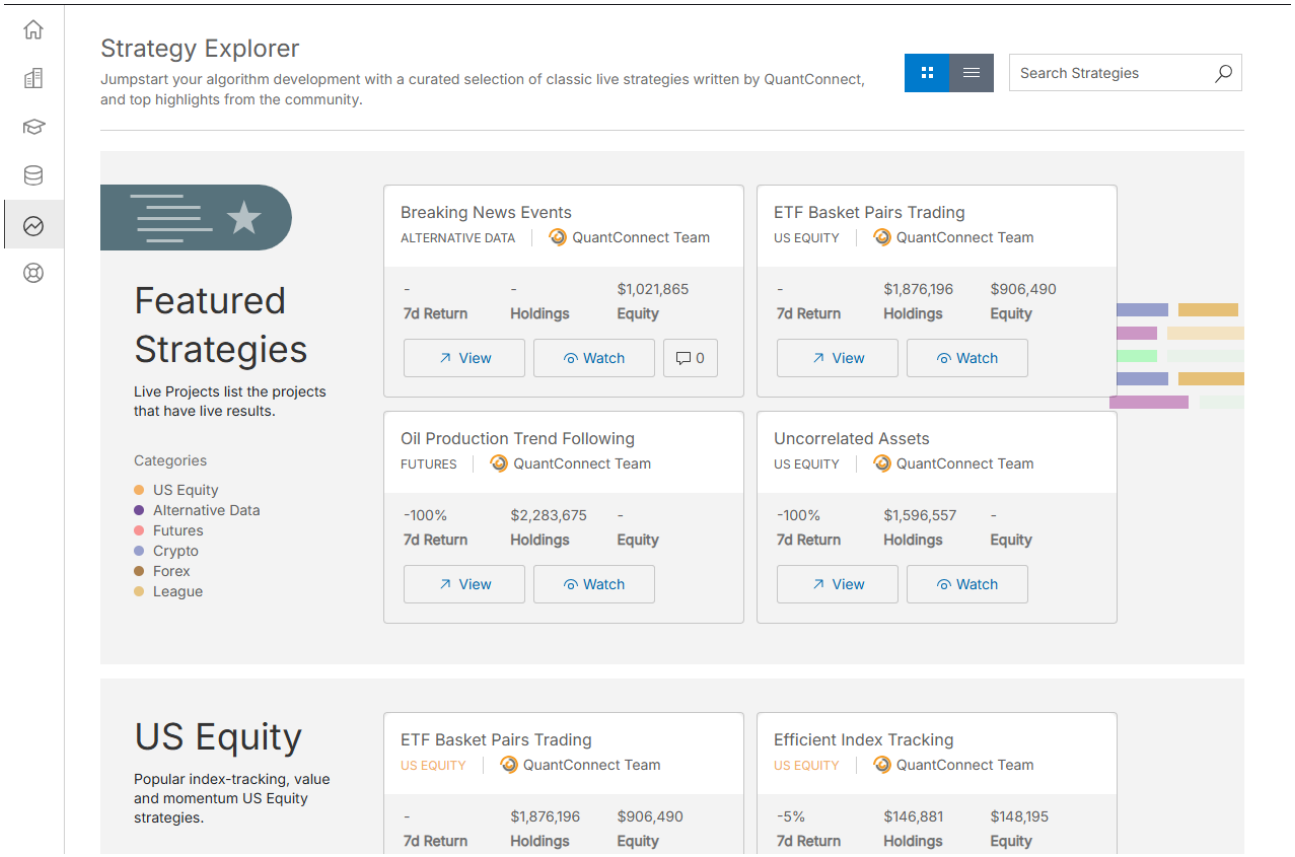


Рисунок 1.4 – Приклад створення стратегії в QuantConnect

Важливо зазначити, що QuantConnect не має офіційної мобільної версії для жодних пристроїв. Всі операції з платформою здійснюються через веб-інтерфейс, що значно обмежує можливості для використання з мобільних пристроїв. Це може бути мінусом для тих користувачів, які потребують доступу до платформи на ходу, оскільки мобільні браузери не завжди забезпечують комфортну роботу з такими складними інструментами, як QuantConnect.

Загалом, дизайн платформи QuantConnect є функціональним, але його мінімалізм може здатися неприязним для користувачів, які шукають більш інтуїтивно зрозуміле середовище. Водночас для досвідчених користувачів та професіоналів така структура є перевагою, оскільки вона дає повний контроль над алгоритмами і стратегіями без обмежень, які можуть виникати в більш «пластичних» рішеннях.

Для наочності переваги та недоліки основних аналогів подано в таблиці 1.1.

Таблиця 1.1 – Порівняння наявних рішень

Назва платформи	Розробник	Переваги	Недоліки
MetaTrader 5	MetaQuotes Ltd.	Простота використання, поширеність, підтримка ботів, візуалізація	Обмежена гнучкість, орієнтація на роздрібного трейдера
NinjaTrader	NinjaTrader LLC	Гнучкість через C#, аналітика, інтеграція з брокерами	Високий поріг входу, частина функцій доступна лише у платній версії
cTrader	Spotware Systems	Сучасний інтерфейс, висока швидкодія, підтримка автоматизації через C#	Менша кастомізація складних логік, потреба у сторонніх сервісах
QuantConnect	QuantConnect LLC	Можливість моделювання, Python/C#, доступ до великих історичних даних	Складність у використанні, слабка підтримка GUI

Проаналізувавши подані рішення, можна дійти висновку, що жодне з них не забезпечує повністю відкритого, доступного, водночас гнучкого середовища для розробки та використання торгових алгоритмів, яке можна адаптувати під специфічні потреби користувача без комерційних обмежень. Саме тому доцільним є створення власного програмного засобу, що врахує найкращі риси згаданих платформ – зручний інтерфейс, гнучкість у налаштуваннях, інтеграцію з біржовими API, можливості тестування на історичних даних, адаптивність торгової логіки – і при цьому буде відкритим до вдосконалення.

1.3 Аналіз вимог до програмного засобу та постановка завдання

Аналіз предметної області автоматизованої торгівлі на фінансовому ринку дав змогу сформувати основні вимоги до програмного засобу, який розробляється для забезпечення ефективної, гнучкої та надійної роботи з фінансовими активами в режимі реального часу.

Система має бути орієнтована на користувачів, які мають базові знання в галузі фінансової торгівлі, але не обов’язково є досвідченими програмістами. Водночас вона має дозволяти більш просунутим користувачам реалізовувати власні алгоритмічні моделі. Основними категоріями користувачів є:

– Приватні трейдери – користувачі, що хочуть запустити власну стратегію або скористатися типовими шаблонами.

– Аналітики/розробники стратегій – користувачі, які створюють, тестують і вдосконалюють алгоритми.

– Адміністратори системи – відповідальні за налаштування інфраструктури, керування правами доступу, моніторинг стабільності.

Функціональні вимоги до програмного засобу з реалізації автоматизованої торгівлі на фінансовому ринку передбачають забезпечення можливості виконання торгових стратегій у режимі реального часу з автоматичним формуванням та надсиланням торгових ордерів на біржові платформи. Система повинна підтримувати інтеграцію з популярними біржами через API, що

дозволяє отримувати актуальні ринкові котирування, здійснювати операції купівлі-продажу та отримувати історичні дані. Користувач повинен мати змогу створювати, редагувати та тестувати власні стратегії за допомогою візуального конструктора або через середовище написання коду, а також запускати їх із заданими параметрами.

Розроблюваний програмний засіб повинен забезпечувати можливість тестування стратегій на історичних даних з подальшим аналізом результатів та візуалізацією змін у портфелі. Передбачено ведення журналу подій, де фіксуються всі дії системи, зокрема виконані ордери, спрацьовування умов, виникнення помилок. Інтерфейс користувача має бути інтуїтивно зрозумілим, адаптивним до різних пристроїв і забезпечувати зручний доступ до ключових показників: балансу рахунку, поточних позицій, історії торгів і загальної ефективності стратегії. Також передбачається система сповіщень про важливі події або збої в роботі стратегії.

Користувачі повинні проходити автентифікацію, після чого система, з огляду на їхню роль, відкриває відповідний набір функцій. Загалом система має забезпечувати стабільну, надійну та безпечну роботу з мінімальним втручанням з боку людини, що є критично важливим для алгоритмічної торгівлі.

За результатами аналізу предметної області автоматизованої торгівлі на фінансовому ринку та дослідження наявних рішень можна зробити наступні висновки.

Традиційна ручна торгівля на фінансових ринках має суттєві обмеження, пов'язані з людським фактором, серед яких емоційність при прийнятті рішень, повільність обробки інформації, неможливість безперервного моніторингу ринку та оперативного реагування на зміни. Автоматизація торгових процесів є об'єктивною необхідністю для підвищення ефективності та зменшення ризиків.

Проведений аналіз основних торгових платформ виявив, що кожна з них має як переваги, так і значні недоліки. MetaTrader 5 відзначається простотою використання та поширеністю, але має обмежену гнучкість для складних алгоритмічних стратегій. NinjaTrader забезпечує високу гнучкість через C#, але

характеризується високим порогом входу та частковою платністю функцій. cTrader має сучасний інтерфейс та високу швидкодію, проте обмежений у кастомізації складних логік. QuantConnect надає потужні можливості моделювання, але є складним у використанні та має слабку підтримку графічного інтерфейсу.

Аналіз показав відсутність на ринку повністю відкритого, доступного та водночас гнучкого середовища для розробки торгових алгоритмів, яке поєднувало б зручний та інтуїтивно зрозумілий інтерфейс, гнучкість у налаштуваннях та створенні стратегій, відкритість до вдосконалення без комерційних обмежень та підтримку різних категорій користувачів від новачків до професіоналів.

На основі проведеного аналізу сформульовано комплексні вимоги до розроблюваного програмного засобу. Функціональні вимоги передбачають виконання торгових стратегій у режимі реального часу, інтеграцію з біржовими API для отримання котирувань та здійснення операцій, створення та редагування стратегій через візуальний конструктор або код, тестування на історичних даних з аналізом результатів, ведення журналу подій та системи сповіщень. Користувацькі вимоги включають підтримку різних категорій користувачів, інтуїтивно зрозумілий та адаптивний інтерфейс, систему автентифікації та розмежування прав доступу.

Результати аналізу обґрунтовують доцільність створення власного програмного засобу, що поєднає найкращі характеристики існуючих платформ та усуне їх недоліки. Розроблювана система має стати універсальним інструментом для автоматизованої торгівлі, доступним широкому колу користувачів незалежно від їх технічної підготовки. Отримані висновки формують надійну основу для переходу до наступних етапів розробки системи автоматизованої торгівлі на фінансовому ринку.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування системи та бази даних

На етапі проєктування майбутньої системи важливо визначити її загальну структуру. Архітектура системи задає основні компоненти, спосіб їх взаємодії, а також напрямки для подальшої розробки. У цьому проєкті планується створення простої навчальної системи з базовим функціоналом, яка реалізуватиметься за принципом клієнт-серверної архітектури.

На діаграмі варіантів використання зображено ключові дії, які доступні різним учасникам системи «Автоматизована система торгівлі на фінансовому ринку». Основними акторами виступають трейдер та адміністратор (рис. 2.1).

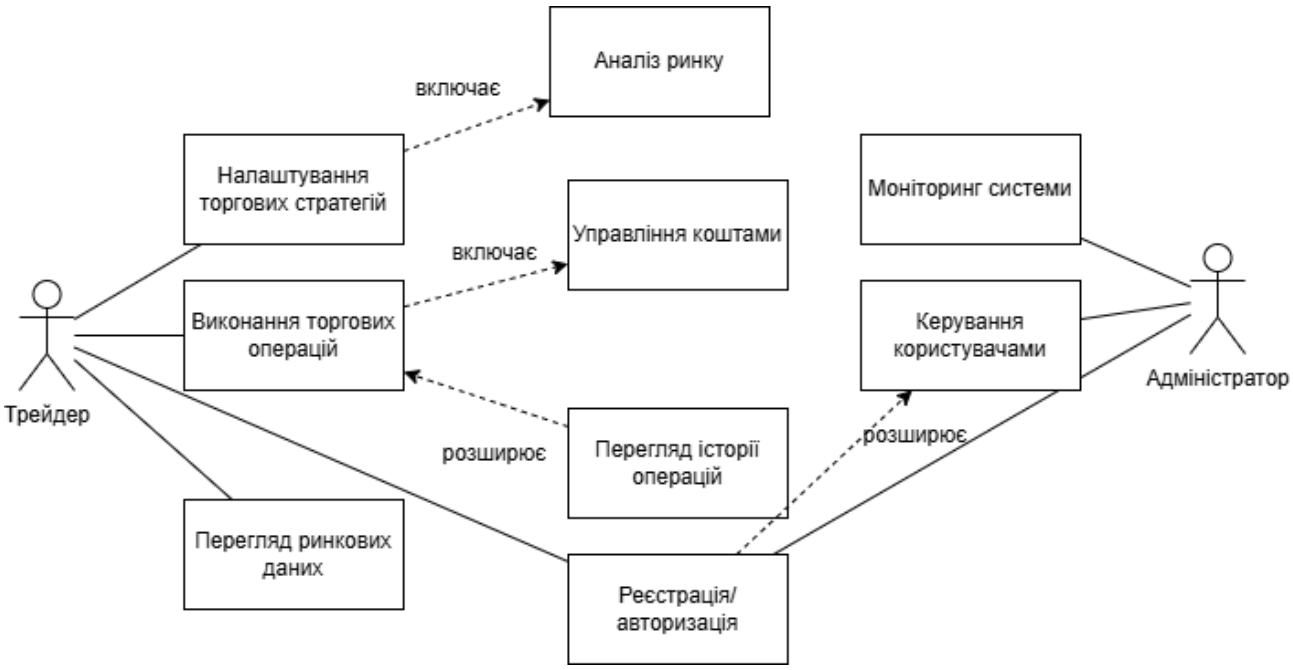


Рисунок 2.1 – UML-діграма варіантів використання

Трейдер має змогу проходити реєстрацію та авторизацію, переглядати ринкові дані, виконувати торгові операції, налаштовувати торгові стратегії, аналізувати ринок, переглядати історію своїх операцій та управляти коштами.

Адміністратор також може авторизуватись, а також керувати користувачами й здійснювати моніторинг системи.

Взаємозв’язки між варіантами використання вказують на залежності, наприклад: виконання торгової операції пов’язано з управлінням коштами та переглядом історії операцій, а налаштування стратегій передбачає попередній аналіз ринку.

Клієнтська частина матиме мінімалістичний дизайн і дозволятиме користувачу виконувати основні дії: переглядати інформацію про торгові стратегії, запускати їх на історичних даних та переглядати результати таких тестів. Наголос робиться не на візуальній складності, а на доступності й зрозумілості інтерфейсу.

Серверна частина системи буде відповідати за обробку запитів від клієнта. У межах проєкту планується створити серверну логіку, яка дозволить завантажити або обрати одну із заздалегідь підготовлених стратегій, запустити її на тестових історичних даних (наприклад, на простій таблиці з цінами), і передати результати користувачу.

Логіка роботи з базою даних буде спрощеною: при кожному новому тестуванні створюватиметься новий запис, у якому фіксуватимуться основні параметри (дата, назва стратегії, прибуток, кількість операцій тощо).

У процесі проєктування було сформовано загальну архітектурну модель системи, що охоплює клієнтську частину, сервер та базу даних [5]. На рисунку 2.2 зображено взаємозв’язки між основними компонентами.

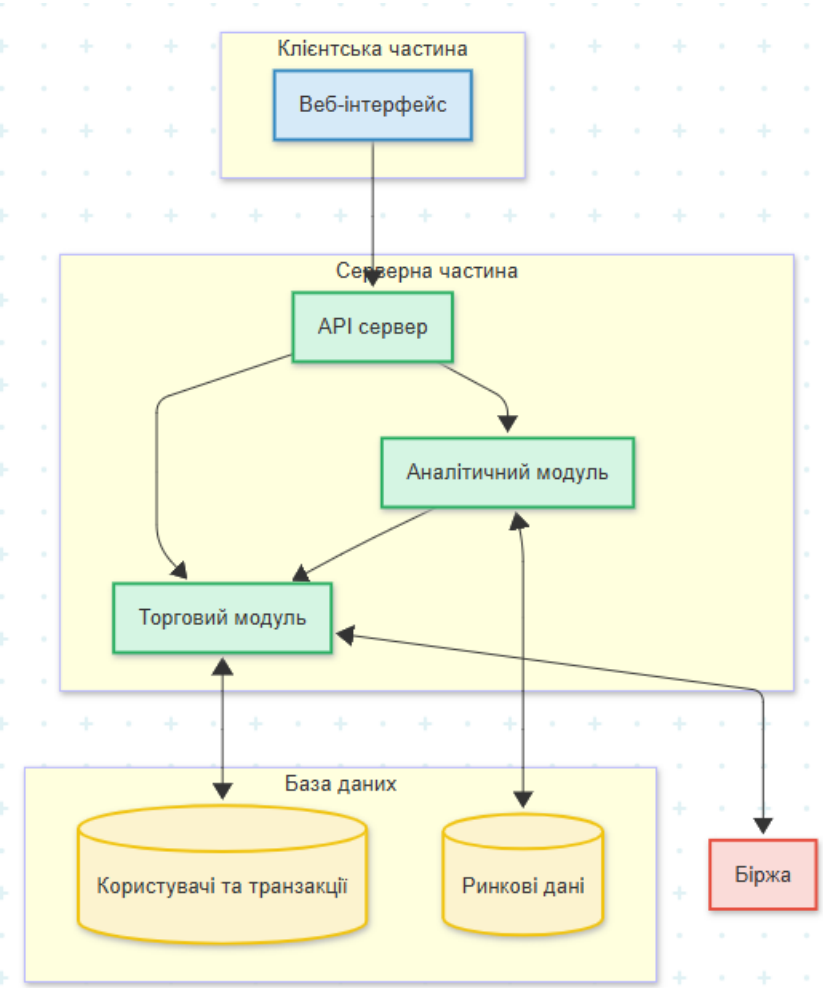


Рисунок 2.2 – Загальна схема архітектури системи

Клієнтська частина представлена вебінтерфейсом, через який користувач взаємодіє із системою. Вона надсилає запити на сервер, який складається з API-сервера, торгового модуля та аналітичного модуля. API-сервер обробляє вхідні запити й передає їх до відповідних функціональних частин. Торговий модуль здійснює обчислення на основі ринкових даних, взаємодіє з базою даних (де зберігаються дані користувачів і транзакцій), а також з біржею – умовним зовнішнім джерелом ринкової інформації. Аналітичний модуль використовує історичні ринкові дані для аналізу ефективності стратегій, передає результати в торговий модуль або безпосередньо на клієнт. База даних розділена логічно на дві частини: одна зберігає дані про користувачів і виконані операції, інша – історичні ринкові котирування. Така структура дозволяє забезпечити чітке

розмежування обов’язків між компонентами, спрощує підтримку та масштабування системи.

У межах проєктування також було змодельовано загальний сценарій взаємодії користувача із системою, що представлено на рисунку 2.3.

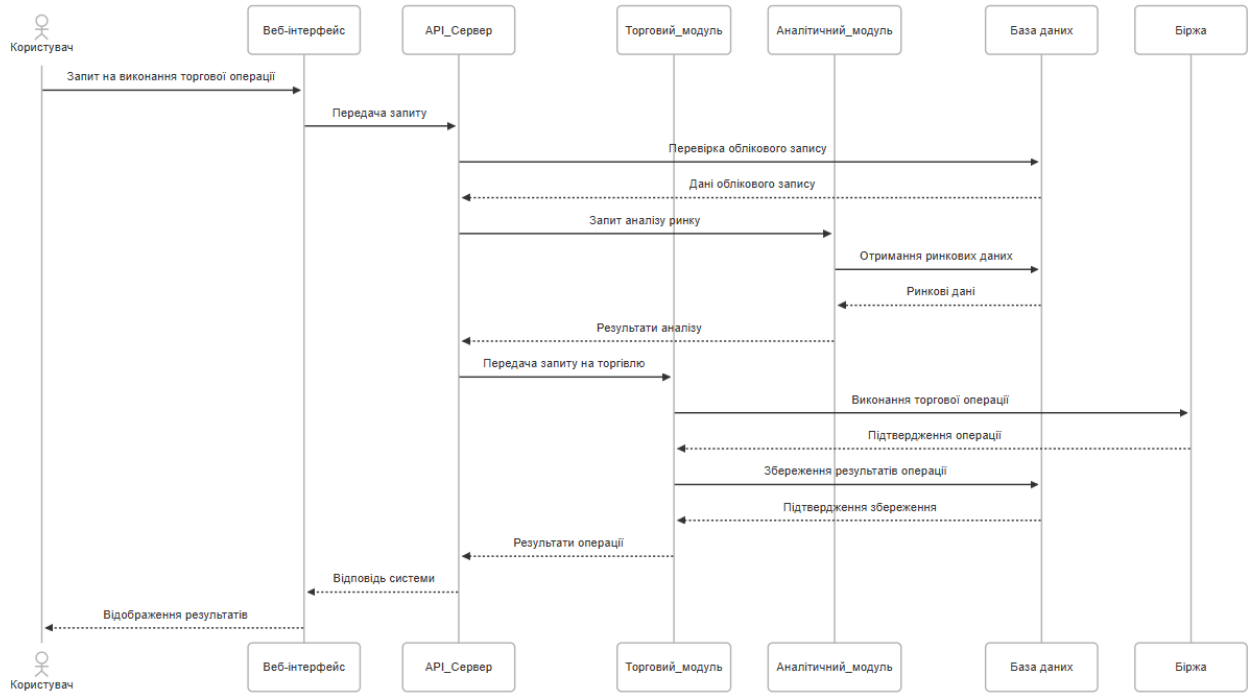


Рисунок 2.3 – Послідовність взаємодії між компонентами системи під час торгової операції

Сценарій починається з дії користувача, який надсилає запит на виконання торгової операції через вебінтерфейс. Інтерфейс передає цей запит на сервер, де API-сервер проводить перевірку облікового запису, звертаючись до бази даних. Після успішної перевірки API-сервер ініціює запит до аналітичного модуля з метою отримання актуального аналізу ринку. Аналітичний модуль витягує потрібні ринкові дані з бази, проводить розрахунки та передає результати назад. На основі отриманої інформації API-сервер формує запит до торгового модуля, який виконує операцію через біржу. Після виконання, результати фіксуються в базі даних. Нарешті, користувач отримує зворотній зв’язок через вебінтерфейс. Такий ланцюг дій демонструє типовий хід роботи майбутньої системи,

підкреслюючи її логічну послідовність та взаємозв'язки між основними компонентами.

Для реалізації системи було спроектовано базу даних, яка включає кілька ключових компонентів, що взаємодіють між собою для забезпечення необхідного функціоналу системи (рис. 2.4). База даних містить таблиці для зберігання даних про користувачів, торгові операції, стратегії, ринкові котирування, а також результати тестування стратегій. Ці таблиці взаємодіють між собою, забезпечуючи повний цикл роботи системи – від реєстрації користувача до виконання торгових операцій та аналізу результатів.

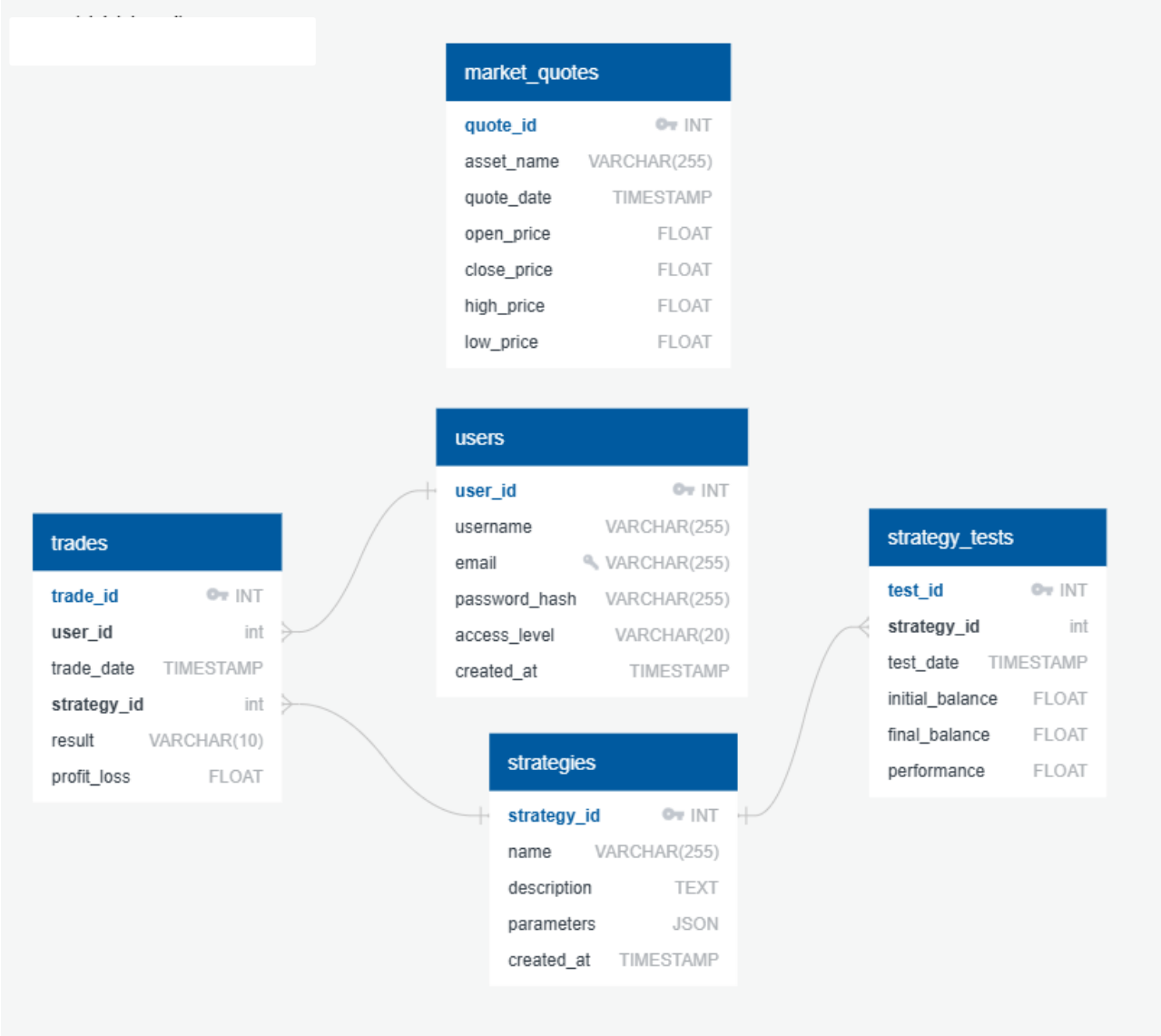


Рисунок 2.4 – Схема бази даних

Однією з основних таблиць є таблиця користувачів, яка зберігає інформацію про реєстрацію користувачів, їхні особисті дані, рівень доступу, а також дані для авторизації. Вона пов’язана з іншими таблицями через ідентифікатори користувачів, що дає змогу відслідковувати торгові операції та тестування стратегій кожного окремого користувача.

Таблиця торгових операцій зберігає всі операції, що виконуються в системі, включаючи дату та час проведення операції, кількість транзакцій, їхній результат, а також прибуток або збитки. Вона тісно пов'язана з таблицею користувачів, що дозволяє ідентифікувати, які саме користувачі здійснили ці операції.

Таблиця ринкових котирувань містить історичні дані про ціни на активи, які використовуються для тестування торгових стратегій. Ці котирування є основою для аналітичних модулів, які оцінюють ефективність стратегій за допомогою цих даних.

Таблиця стратегій зберігає дані про всі налаштовані стратегії, їхні параметри, а також результати їхнього тестування. Кожна стратегія має унікальний ідентифікатор, що дозволяє її пов’язувати з результатами тестування і виконаними операціями.

Ці таблиці бази даних організовані таким чином, що забезпечують ефективну взаємодію між основними компонентами системи, такими як користувачі, торгові операції, стратегії та ринкові дані. Це дозволяє не лише зберігати необхідні дані, а й виконувати операції на основі актуальної інформації, надаючи користувачеві точні та своєчасні результати.

Всі ці дані зберігаються в реляційній базі даних, що дозволяє ефективно керувати ними за допомогою SQL-запитів. База даних організована таким чином, що кожен компонент має чітко визначену роль і відповідальність, а також зручний доступ до необхідних даних для подальшого використання в системі.

Таким чином, уже на етапі проєктування визначено, що архітектура системи буде логічно структурованою та орієнтованою на ефективну взаємодію між основними її компонентами – клієнтом, сервером і базою даних. Така

побудова дозволяє забезпечити чітке розмежування функціональності, спростити підтримку й розвиток системи, а також створити надійне середовище для реалізації закладених алгоритмів і обробки даних.

2.2 Моделі та алгоритми

На етапі проєктування автоматизованої системи торгівлі важливим завданням є визначення набору моделей та алгоритмів, які в майбутньому забезпечать ефективну роботу системи в умовах фінансового ринку. Система має орієнтуватися на здатність обробляти великі обсяги динамічних даних, виявляти закономірності у зміні цінових показників, а також приймати торгові рішення на основі аналітики.

Планується впровадження алгоритмів технічного аналізу, які дозволяють аналізувати історичні дані з графіків цін і обсягів торгів з метою виявлення трендів, точок розвороту, зон підтримки та опору. До таких алгоритмів можуть входити індикатори ковзних середніх, осцилятори, смуги Боллінджера, індекси сили ринку та інші класичні методи. Їх застосування дасть змогу формувати перші торгові сигнали без використання складних прогнозних моделей.

Окрім технічного аналізу, розглядається можливість використання математичних моделей часових рядів для передбачення короткострокових змін на ринку. Моделі авторегресії, ковзного середнього та комбіновані ARIMA-моделі можуть бути використані для прогнозування майбутніх цін на основі попередніх значень. Такий підхід є особливо актуальним на початкових етапах, коли ще не накопичено достатньо даних для навчання більш складних моделей.

На наступному етапі розвитку системи доцільно впроваджувати алгоритми машинного навчання, які здатні адаптуватися до нових ринкових умов і вдосконалювати свої прогнози на основі накопичених даних. До потенційно ефективних підходів належать моделі класифікації та регресії, зокрема логістична регресія, дерева рішень, ансамблеві методи (градієнтний бустинг, Random Forest) та нейронні мережі. Залежно від обсягу доступної інформації та

поставлених цілей, ці алгоритми можуть бути використані як для прогнозування цін, так і для виявлення сигналів купівлі або продажу.

Додатково до алгоритмів прийняття торгових рішень планується реалізація елементів рекомендаційної логіки, яка допоможе користувачам обирати оптимальні стратегії або фінансові інструменти відповідно до їхніх інтересів і поведінкових шаблонів. Такі системи можуть бути побудовані на основі фільтрації за контентом, колаборативної фільтрації або гібридних моделей.

Оскільки система ще не реалізована, всі алгоритми розглядаються на теоретичному рівні, з урахуванням специфіки фінансового середовища, вимог до продуктивності та можливостей масштабування.

2.3 Аналіз засобів реалізації

Під час проєктування системи розглядається низка варіантів засобів реалізації, які можуть бути застосовані для створення кожного компонента – клієнтської частини, серверної логіки, бази даних, аналітичних модулів та торгового інтерфейсу з біржею. Основними критеріями відбору є зручність розробки, продуктивність, масштабованість, доступність бібліотек для фінансового аналізу та інтеграції з зовнішніми сервісами, а також досвід команди розробки.

У сфері розробки клієнтської частини аналізуються варіанти створення веб-інтерфейсу за допомогою HTML, CSS і JavaScript у поєднанні з фреймворками React, Angular або Vue.js. Всі ці фреймворки підтримують створення інтерактивних SPA-застосунків, з високою продуктивністю та адаптивністю. Перевагу попередньо надається React через широку спільноту, гнучкість та зручність у реалізації компонентного підходу.

Для серверної частини розглядаються кілька популярних мов програмування: Python, JavaScript (Node.js), Java та C#. Python вирізняється потужною екосистемою для обробки даних і реалізації аналітичних обчислень, тому є перспективним для модулів, що виконують обчислення та аналіз ринку. Node.js дозволяє будувати високонавантажені системи з асинхронною обробкою

запитів, що особливо корисно для реального часу. Java і C# пропонують високу стабільність і типізацію, що може бути важливо при роботі з фінансовими транзакціями [7].

Для реалізації API зручно використовувати REST або GraphQL, де REST залишається класичним і добре підтримується більшістю фреймворків. У разі вибору Python як основної мови для сервера, доцільно застосовувати Flask або FastAPI для швидкої розробки REST-інтерфейсів. Якщо буде обрано Node.js, варто розглянути Express або NestJS.

У контексті зберігання даних розглядаються як реляційні, так і нереляційні СУБД. Реляційні бази даних, такі як PostgreSQL або MariaDB, є надійними для зберігання структурованої інформації про користувачів, транзакції, історію торгів тощо [6]. Нереляційні бази даних, такі як MongoDB, можуть бути корисними для гнучкого зберігання великих обсягів аналітичних або неструктурованих даних. Остаточний вибір схиляється до PostgreSQL через її стабільність, підтримку складних запитів, транзакцій і розширень для роботи з часовими рядами.

Для обробки ринкових даних, реалізації аналітики та створення моделей аналізуються можливості використання бібліотек Pandas, NumPy, scikit-learn, TensorFlow, Prophet (від Meta) та інших інструментів машинного навчання. Вони дозволяють будувати як класичні статистичні моделі, так і нейромережеві прогнози. У якості середовища розробки таких моделей буде використовуватися Python завдяки великому набору бібліотек і прикладів у фінансовій сфері.

Для комунікації з біржею розглядаються біржові API, зокрема REST і WebSocket API провідних бірж (наприклад, Binance, Coinbase або інтеграція із симуляторами). Важливим фактором тут є підтримка документації, стабільність, швидкість обміну даними та безпечна аутентифікація.

Після аналізу можливих варіантів реалізації було прийнято рішення щодо використання наступного стеку технологій, який найкраще відповідає вимогам функціональності, гнучкості, розширюваності та зручності підтримки в межах даного проєкту.

Клієнтську частину системи вирішено реалізовувати як вебзастосунок з використанням HTML, CSS та фреймворку Bootstrap [8]. Такий підхід забезпечує швидку розробку адаптивного, зручного та візуально привабливого інтерфейсу без потреби в складних JS-фреймворках. Bootstrap дозволяє легко реалізовувати сіткову систему, типові компоненти інтерфейсу, а також гарантує коректне відображення на різних пристроях. Завдяки великій спільноті та готовим рішенням, його можна швидко адаптувати до потреб фінансової платформи.

В якості системи управління базами даних обрано MariaDB – реляційну СУБД з відкритим вихідним кодом, яка забезпечує високу продуктивність, надійність та сумісність з MySQL. MariaDB підтримує транзакції, індекси, зовнішні ключі та складні SQL-запити, що дозволяє ефективно працювати з великою кількістю структурованих даних. Для адміністрування бази даних використовується phpMyAdmin – зручний веб-інтерфейс, який спрощує взаємодію з базою даних, дозволяє переглядати таблиці, виконувати SQL-запити, створювати резервні копії та керувати користувачами.

Загальна архітектура системи передбачає чіткий поділ на клієнтську, серверну та аналітичну частини з урахуванням майбутньої можливості масштабування або винесення окремих компонентів на інші сервери. Обрані інструменти дозволяють гнучко й ефективно реалізувати ключовий функціонал, забезпечити зручну підтримку й подальший розвиток системи без необхідності повної перебудови її структури.

2.4 Проєктування інтерфейсу системи

Проєктування інтерфейсу є важливим етапом розробки, адже саме через нього користувач взаємодіє із системою, отримує інформацію та виконує цільові дії. У межах цієї системи розроблено набір шаблонів, які демонструють основні вікна користувацького інтерфейсу, їхню структуру, логіку розміщення елементів і загальні принципи навігації.

Перш за все, макет головної сторінки надає користувачеві узагальнений огляд ринку, показники останніх змін та ключові інструменти для торгівлі.

Панель навігації дозволяє швидко перемикатися між розділами, такими як аналітика, історія операцій, налаштування стратегії чи управління коштами. Інформація подається у зрозумілому та лаконічному вигляді, з акцентами на ключових даних, які потребують швидкої реакції. На рисунку 2.5 зображено приклад головної панелі трейдера з доступом до основного функціоналу.

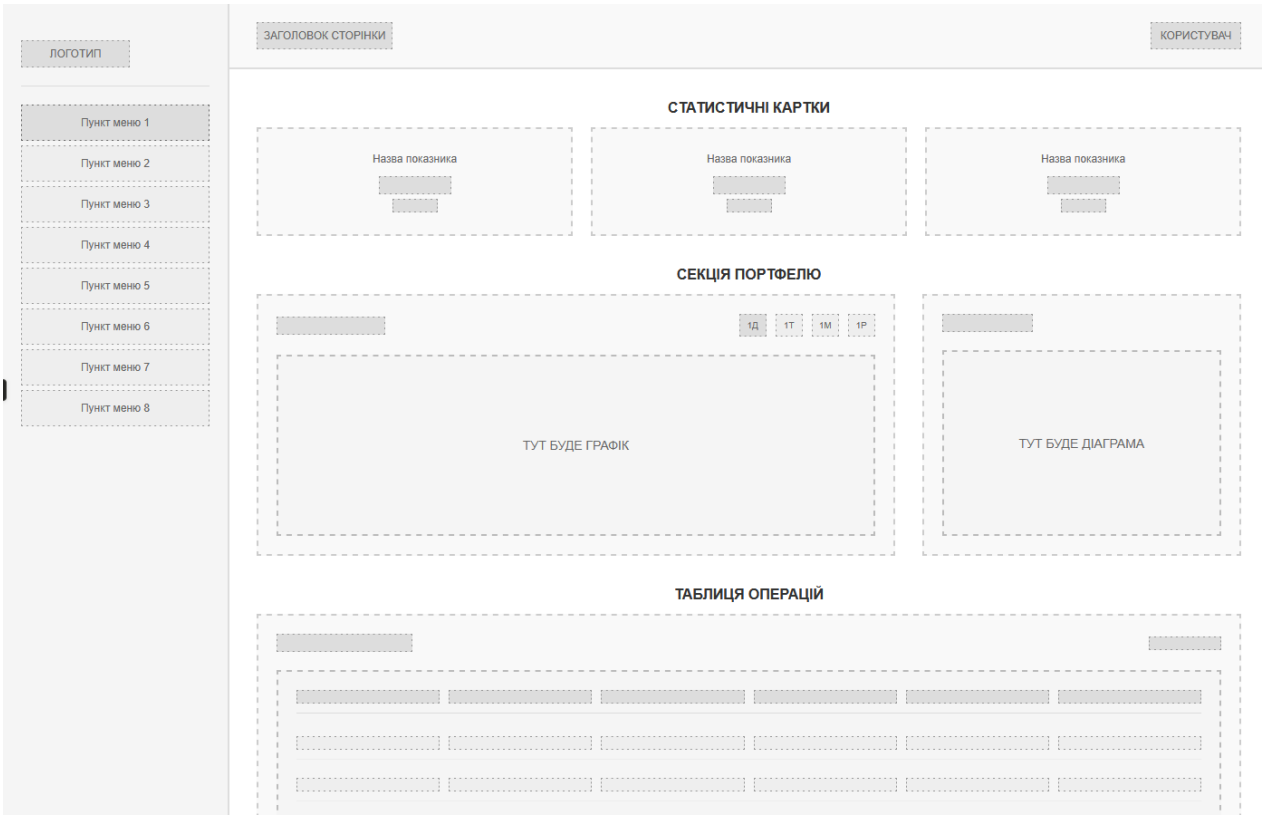


Рисунок 2.5 – Макет головної сторінки

Для розділу аналітики передбачено окреме вікно з графіками, динамічними таблицями та фільтрами, що дають змогу аналізувати ринок за різними критеріями. Передбачено використання кольорових індикаторів, що візуально підкреслюють важливі зміни та сигнали, які можуть вплинути на прийняття торгових рішень. Рисунок 2.6 демонструє структуру сторінки аналітики з прикладом інтерактивного графіка та панелі налаштувань.

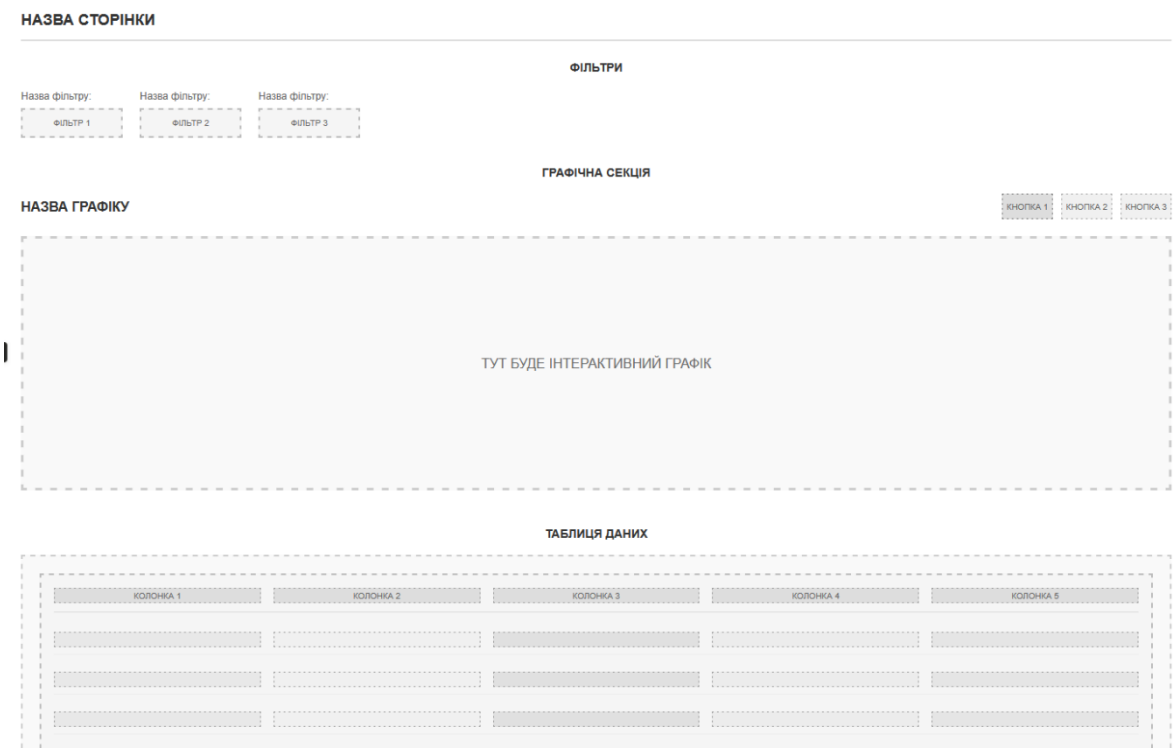


Рисунок 2.6 – Макет сторінки аналітики

Також розроблено шаблон інтерфейсу виконання торгових операцій, який має бути максимально простим і захищеним від помилкових дій. У ньому користувач може ввести параметри угоди, перевірити поточний баланс, обрати актив і миттєво надіслати команду на виконання. Зовнішній вигляд цього розділу фокусується на зручності введення та перевірки даних перед фінальним підтвердженням.

Інтерфейс адміністратора відрізняється доступом до розширених функцій, зокрема контролю за користувачами, перегляду логів системи та управління загальними налаштуваннями. Макет цієї частини містить більше службових блоків, однак зберігає загальний стиль і принципи взаємодії, що застосовано в інтерфейсі трейдера.

Таким чином, розроблені шаблони інтерфейсу демонструють зручну і послідовну взаємодію користувача з усіма компонентами системи. Це сприяє позитивному користувацькому досвіду та дозволяє ефективно виконувати поставлені завдання.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Розробка системи

Розробка автоматизованої системи торгівлі на фінансовому ринку розпочалась з формування основних функціональних компонентів, що забезпечують повний цикл роботи: від отримання ринкових даних до прийняття торгових рішень та їх виконання. Важливим аспектом стало створення модуля збору даних, який відповідає за отримання актуальної інформації з біржових API в режимі реального часу. Наступним кроком був розвиток аналітичного модуля, що дозволяє обробляти як історичні, так і поточні ринкові дані, формувати технічні індикатори та генерувати торгові сигнали на основі алгоритмів машинного навчання. Серверна частина системи забезпечує координацію взаємодії між користувачем, аналітикою та біржею, а також відповідає за збереження даних і передачу команд на виконання торгових операцій. Для користувачів передбачено інтерфейс, що дозволяє відстежувати ринкову ситуацію, переглядати результати аналітики та керувати торговими стратегіями.

Для розробки вебпроекту було створено нову папку в директорії xampp/htdocs, яка є кореневою для вебсерверу Apache у складі XAMPP. У цій папці розміщуватимуться всі файли і ресурси проекту, що дозволить запускати його локально через браузер за адресою http://localhost/назва_папки.

На рисунку 3.1 наведено скріншот провідника, на якому видно стуктуру папки xampp.

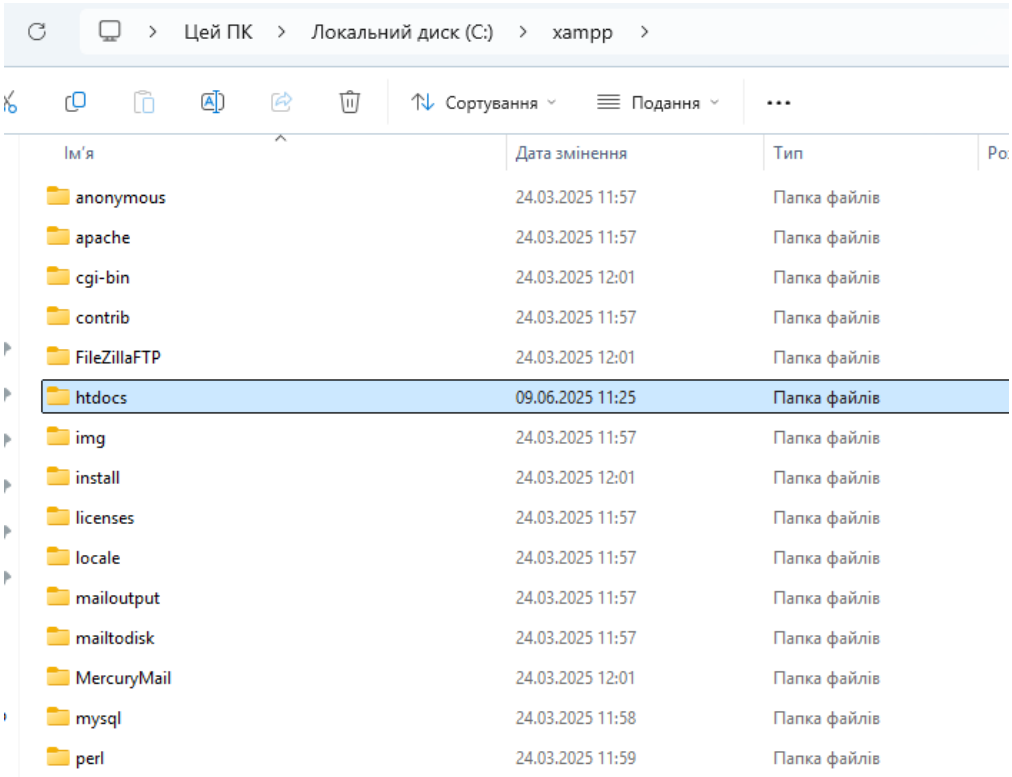


Рисунок 3.1 – Папка проекту у директорії xampp

Після створення папки проекту наступним кроком було її відкриття у редакторі коду Visual Studio Code. Використання VS Code дозволяє зручно організувати роботу з проектом, оскільки редактор надає можливість одночасно переглядати всі файли та папки, що входять до складу проекту. У середовищі VS Code можна легко редагувати код, запускати інтегрований термінал для виконання команд. Завдяки цьому забезпечується ефективна і зручна розробка програмного забезпечення.

На рисунку 3.2 наведено скріншот інтерфейсу Visual Studio Code з відкритою папкою проекту.

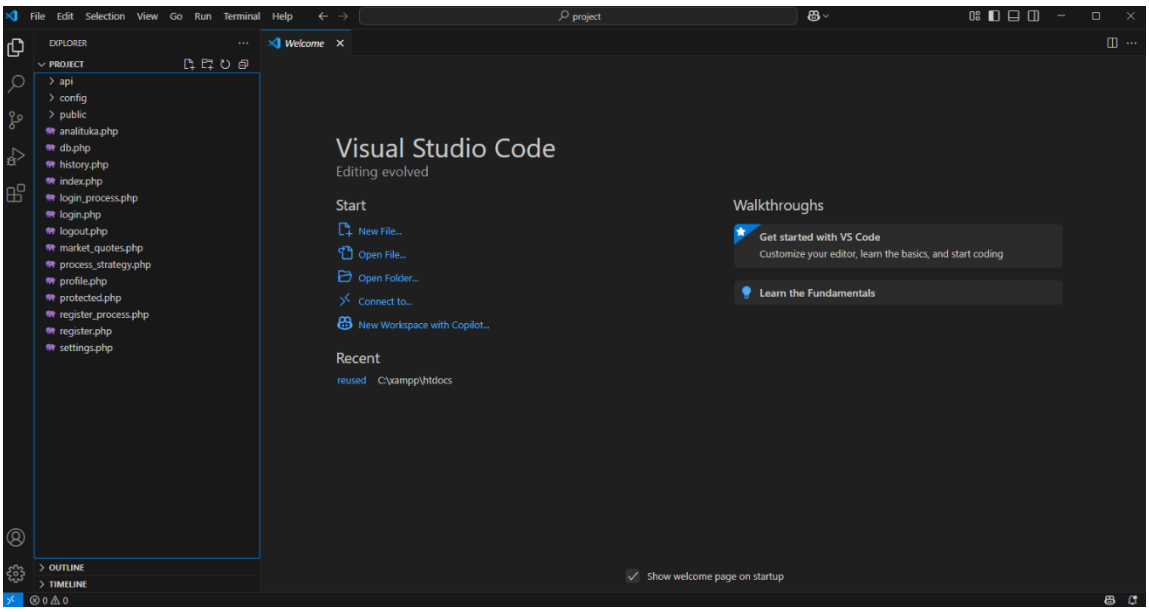


Рисунок 3.2 – Відкриття папки проекту у Visual Studio Code

Лістинг 3.1 демонструє реалізацію процесу автентифікації користувача у веб-додатку з використанням PHP та підготовлених SQL-запитів. Код починається з виконання параметризованого запиту до бази даних для пошуку користувача за ім'ям, що забезпечує захист від SQL-ін'єкцій завдяки використанню заповнювачів замість прямої конкатенації рядків.

Лістинг 3.1 – Авторизація користувачів

```
<?php
session_start();
require 'db.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = trim($_POST['username']);
    $password = $_POST['password'];
    $stmt = $pdo->prepare("SELECT * FROM users WHERE username
= ?");
    $stmt->execute([$username]);
    $user = $stmt->fetch();
    if ($user && password_verify($password,
$user['password_hash'])) {
        $_SESSION['user_id'] = $user['user_id'];
        $_SESSION['username'] = $user['username'];
        $_SESSION['access_level'] = $user['access_level'];
        header('Location: index.php');
        exit();
    } else {
        header('Location: login.php?error=' .
urlencode('Невірне ім’я користувача або пароль'));
        exit();
    }}
```

Після отримання даних користувача з бази даних виконується перевірка правильності введеного пароля за допомогою функції password_verify(), яка порівнює надісланий пароль з хешованим паролем, збереженим у базі даних. Така реалізація забезпечує безпечне зберігання паролів користувачів, оскільки паролі не зберігаються у відкритому вигляді.

У випадку успішної автентифікації система створює користувацьку сесію, записуючи до глобального масиву $_SESSION необхідні дані користувача, включаючи його ідентифікатор, ім'я користувача та рівень доступу. Після встановлення сесії відбувається перенаправлення користувача на головну сторінку панелі управління за допомогою HTTP-заголовка Location.

Якщо автентифікація не вдається через неправильні облікові дані, система перенаправляє користувача назад на сторінку входу login.php з параметром помилки, який містить повідомлення про невірне ім'я користувача або пароль. Використання функції urlencode() забезпечує правильне кодування повідомлення для передачі через URL-параметри.

Код завершується викликом функції exit() в обох гілках виконання, що гарантує припинення подальшого виконання скрипта після перенаправлення та запобігає можливим проблемам безпеки, пов'язаним з продовженням обробки після відправлення заголовків перенаправлення.

Представлений в лістингу 3.2 код реалізує функціонал реєстрації нових користувачів у веб-додатку з використанням PHP та PDO для роботи з базою даних. Скрипт починається з ініціалізації сесії та підключення до бази даних через файл db.php, після чого перевіряє, чи запит надійшов методом POST, що є стандартною практикою для обробки форм реєстрації.

Лістинг 3.2 – Реєстрація користуввачів

```
<?php
session_start();
require_once 'db.php'; if ($_SERVER['REQUEST_METHOD'] ===
'POST') {$username = trim($_POST['username']);
    $email = trim($_POST['email']);
$password = $_POST['password'];
    if (empty($username) || empty($email) ||
empty($password)) {
header('Location: register.php?error=Заповніть всі поля');
```

```
            exit();}
         try {$stmt = $pdo->prepare("SELECT * FROM users WHERE
username = :username OR email = :email");
$stmt->execute(['username' => $username, 'email' => $email]);
            if ($stmt->rowCount() > 0) {
                header('Location: register.php?error=Користувач з
таким ім’ям або email вже існує');exit();}
        $hashedPassword = password_hash($password,
PASSWORD_DEFAULT);
            $stmt = $pdo->prepare("INSERT INTO users (username,
email, password_hash, access_level) VALUES (:username, :email,
:password, 'user')");
            $stmt->execute([
                'username' => $username,
                'email' => $email,
                'password' => $hashedPassword]);
            $user_id = $pdo->lastInsertId();
        $_SESSION['user_id'] = $user_id;
            $_SESSION['username'] = $username;
            header('Location: index.php');
            exit();
         } catch (PDOException $e) {
            header('Location: register.php?error=Помилка: ' .
urlencode($e->getMessage()));
            exit();}} else {
         header('Location: register.php');
         exit();}?>
```

Першим кроком обробки POST-запиту є отримання та очищення вхідних даних користувача, включаючи ім'я користувача, електронну пошту та пароль. Функція trim() використовується для видалення зайвих пробілів з початку та кінця рядків, що допомагає уникнути помилок валідації через випадкові пробіли.

Система виконує базову валідацію отриманих даних, перевіряючи, чи всі обов'язкові поля заповнені користувачем. У випадку незаповнення будь-якого з полів відбувається перенаправлення назад на сторінку реєстрації з відповідним повідомленням про помилку.

Перед створенням нового облікового запису система перевіряє унікальність ім'я користувача та електронної пошти шляхом виконання SQL-запиту з використанням логічного оператора OR. Це запобігає створенню дублікатів облікових записів і забезпечує унікальність ключових ідентифікаторів користувачів у системі.

Якщо перевірка на унікальність пройшла успішно, пароль користувача хешується за допомогою функції password_hash() з алгоритмом PASSWORD_DEFAULT, що забезпечує безпечне зберігання паролів у базі даних. Використання сучасних алгоритмів хешування є критично важливим для захисту користувацьких даних.

Новий користувач додається до бази даних з рівнем доступу «user» за замовчуванням, після чого система отримує ідентифікатор щойно створеного запису за допомогою методу lastInsertId(). Автоматичне призначення базового рівня доступу спрощує процес реєстрації та забезпечує консистентність даних.

Після успішного створення облікового запису система автоматично авторизує користувача, створюючи сесію з необхідними параметрами та перенаправляючи на головну сторінку додатку. Така реалізація покращує користувацький досвід, оскільки не вимагає додаткового входу після реєстрації.

Весь код обробки даних обгорнутий у блок try-catch для перехоплення можливих помилок бази даних, що забезпечує коректну обробку винятків та інформування користувача про проблеми у зручному форматі через параметри URL.

Представлений в додатку А, код реалізує комплексну систему управління профілем користувача у веб-додатку, що включає функціонал зміни пароля, оновлення персональних даних та завантаження аватару. Скрипт починається з перевірки авторизації користувача шляхом верифікації наявності ідентифікатора у сесії, що забезпечує доступ до функцій профілю лише для автентифікованих користувачів.

Система обробки POST-запитів розділена на три основні блоки функціональності, кожен з яких відповідає за окремий аспект управління профілем. Перший блок реалізує функціонал зміни пароля з багаторівневою валідацією, включаючи перевірку правильності поточного пароля через порівняння з хешованим значенням у базі даних, валідацію співпадіння нового пароля з підтвердженням та перевірку мінімальної довжини нового пароля.

Другий функціональний блок відповідає за оновлення особистих даних користувача, включаючи електронну пошту, ім'я, прізвище та номер телефону. Система використовує вбудовану PHP-функцію filter_var() з фільтром FILTER_VALIDATE_EMAIL для валідації коректності електронної адреси, що забезпечує відповідність введених даних стандартним форматам електронної пошти.

Третій блок реалізує функціонал завантаження аватару користувача з комплексною системою безпеки та валідації файлів. Система перевіряє тип завантажуваного файлу, обмежуючи дозволені формати до JPEG, PNG та GIF, що запобігає завантаженню потенційно небезпечних файлів. Код автоматично створює директорію для зберігання аватарів, якщо вона не існує, та генерує унікальні імена файлів на основі ідентифікатора користувача.

Обробка помилок реалізована через масив errors, що дозволяє накопичувати та відображати множинні повідомлення про помилки валідації одночасно. Успішні операції відзначаються через змінну success, що забезпечує зворотний зв'язок з користувачем про результати виконання операцій.

Система завантаження файлів включає перевірку статусу завантаження через константу UPLOAD_ERR_OK та використовує функцію move_uploaded_file() для безпечного переміщення завантаженого файлу з тимчасової директорії до постійного сховища. Шлях до аватару зберігається у базі даних для подальшого відображення у інтерфейсі користувача.

Завершальна частина коду відповідає за отримання поточних даних користувача з бази даних для попереднього заповнення форм редагування профілю, що покращує користувацький досвід та зменшує кількість необхідних дій для оновлення інформації.

Лістинг 3.3 демонструє реалізацію системи отримання та відображення інформації про поточного користувача з використанням безпечних методів роботи з базою даних. Код починається з ініціалізації сесії та підключення до

бази даних, після чого виконує обов'язкову перевірку авторизації користувача шляхом верифікації наявності ідентифікатора у глобальному масиві $_SESSION.

Лістинг 3.3 – Відображення інформації про користувача

```
<?php
session_start();
require_once 'db.php';
if (!isset($_SESSION['user_id'])) {
    header('Location: login.php');
    exit();
}
$user_id = $_SESSION['user_id'];
try {
    $stmt = $pdo->prepare("SELECT username, email,
access_level FROM users WHERE user_id = :id");
    $stmt->execute(['id' => $user_id]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$user) {
        echo "Користувача не знайдено.";
        exit();
    }
} catch (PDOException $e) {
    echo "Помилка бази даних: " . htmlspecialchars($e-
>getMessage());
    exit();
}
?>
```

Також написано реалізацію функціоналу виходу користувача з системи через повну очистку сесійних даних та перенаправлення на головну сторінку додатку в лістингу 3.4.

Лістинг 3.4 – Вихід з акаунту

```
<?php
session_start();
session_unset();
session_destroy();
header('Location: index.php');
?>
```

Лістинг 3.5 демонструє реалізацію системи отримання історії торгових операцій з використанням SQL-з'єднання таблиць для агрегації інформації з різних джерел даних. Код виконує запит, що об'єднує таблиці торгових операцій та користувачів через внутрішнє з'єднання за ідентифікатором користувача, що дозволяє отримати не лише дані про операції, але й відповідні імена користувачів для зручного відображення.

Лістинг 3.5 – Отримання історії торгових операцій

```
<?php
require_once 'db.php';
try {
    $stmt = $pdo->query("
        SELECT t.trade_id, u.username, t.trade_date
        FROM trades t
        JOIN users u ON t.user_id = u.user_id
        ORDER BY t.trade_date DESC
        LIMIT 50
    ");
    $trades = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    die("Помилка в отриманні історії: " . $e->getMessage());
}
?>
```

Запит включає сортування результатів за датою торгівлі у спадному порядку, що забезпечує відображення найсвіжіших операцій на початку списку. Обмеження результатів до 50 записів через оператор LIMIT запобігає надмірному навантаженню на сервер та забезпечує оптимальну продуктивність при роботі з великими обсягами історичних даних.

Лістинг 3.6 реалізує функціонал отримання актуальних котирувань фінансових активів з використанням складного підзапиту для фільтрації найсвіжіших даних по кожному активу. Основний запит використовує корельований підзапит у умові для вибору лише тих записів, дата яких відповідає максимальній даті для конкретного активу, що забезпечує отримання найактуальніших котирувань для кожного фінансового інструмента.

Лістинг 3.6 – Вибірка останніх котирувань активів з відсотковою зміною ціни

```
<?php
require_once 'db.php';
$stmt = $pdo->query("
    SELECT asset_name, close_price,
        ROUND(((close_price - open_price) / open_price) *
100, 2) AS percent_change,
        quote_date
    FROM market_quotes
    WHERE quote_date = (
        SELECT MAX(quote_date) FROM market_quotes AS mq2
WHERE mq2.asset_name = market_quotes.asset_name
    )
    ORDER BY asset_name
```

```
        LIMIT 3
    ");
    $quotes = $stmt->fetchAll(PDO::FETCH_ASSOC);
    ?>
```

Запит включає обчислення відсоткової зміни ціни активу через математичну формулу, що порівнює ціну закриття з ціною відкриття та округлює результат до двох знаків після коми за допомогою функції ROUND(). Така реалізація забезпечує автоматичне обчислення важливих фінансових показників на рівні бази даних, що підвищує ефективність обробки даних.

Обмеження результатів до трьох записів через LIMIT та сортування за назвою активу забезпечує контрольоване відображення ключових фінансових інструментів у компактному форматі, що є оптимальним для оглядових інтерфейсів.

В лістингу 3.7 представлений програмний код, який демонструє реалізацію системи аналітичної звітності для торгової платформи з використанням SQL-агрегатних функцій та статистичних запитів. Перша секція коду отримує базові метрики системи через прямі запити підрахунку записів у ключових таблицях, включаючи загальну кількість зареєстрованих користувачів, створених торгових стратегій та виконаних угод.

Лістинг 3.7 – Система аналітики

```
    <?php
    $users = $pdo->query("SELECT COUNT(*) FROM users")-
>fetchColumn();
    $strategies = $pdo->query("SELECT COUNT(*) FROM strategies")-
>fetchColumn();
    $trades = $pdo->query("SELECT COUNT(*) FROM trades")-
>fetchColumn();
    $avgProfit = $pdo->query("SELECT AVG(profit_loss) FROM
strategy_tests")->fetchColumn();
    echo "<ul>
      <li>Користувачів: $users</li>
      <li>Стратегій: $strategies</li>
      <li>Угод: $trades</li>
      <li>Середній прибуток стратегій: " . round($avgProfit, 2) .
"</li></ul>";
    $totalTests = $pdo->query("SELECT COUNT(*) FROM
strategy_tests")->fetchColumn();
    $totalUsersActiveLastMonth = $pdo->query("
        SELECT COUNT(DISTINCT user_id) FROM trades
```

```
        WHERE trade_date >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
    ")->fetchColumn();
    echo "<ul>
      <li>Загальна кількість тестів стратегій: $totalTests</li>
      <li>Активних користувачів за останній місяць:
$totalUsersActiveLastMonth</li>
    </ul>";
    $topUsers = $pdo->query("
        SELECT u.username, COUNT(t.trade_id) AS trades_count
        FROM users u
        JOIN trades t ON u.user_id = t.user_id
        GROUP BY u.user_id
        ORDER BY trades_count DESC
        LIMIT 5
    ")->fetchAll(PDO::FETCH_ASSOC);?>
```

Обчислення середнього прибутку стратегій реалізовано через агрегатну функцію AVG(), що автоматично вираховує середнє арифметичне значення з усіх записів тестування стратегій у базі даних. Результат округлюється до двох знаків після коми за допомогою функції round(), що забезпечує зручне відображення фінансових показників у стандартному форматі.

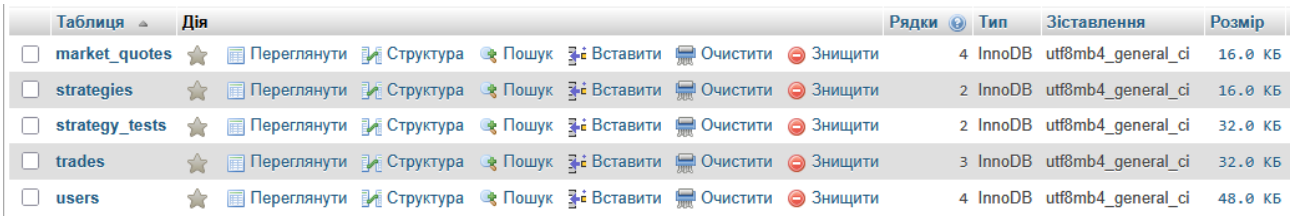
Виведення базових статистичних даних здійснюється через HTML-список з прямим вбудовуванням PHP-змінних у розмітку, що є простим та ефективним способом інтеграції серверних даних з презентаційним шаром. Такий підхід забезпечує швидке відображення ключових показників без додаткової обробки або форматування.

Друга секція розширює аналітику додатковими метриками, включаючи загальну кількість проведених тестів стратегій та кількість активних користувачів за останній місяць. Запит для визначення активних користувачів використовує функцію DATE_SUB() для обчислення дати місяць тому від поточної дати та оператор DISTINCT для підрахунку унікальних користувачів.

Завершальна частина коду реалізує рейтинг найактивніших користувачів через запит з об'єднанням таблиць користувачів та торгових операцій. Використання операторів GROUP BY та ORDER BY дозволяє згрупувати дані за користувачами, підрахувати кількість їхніх операцій та відсортувати результати за спаданням активності.

Обмеження результатів рейтингу до п'яти користувачів через LIMIT забезпечує компактне відображення топ-учасників платформи, що є оптимальним для дашбордів адміністрування. Код отримує результати у вигляді асоціативного масиву через fetchAll(), що дозволяє легко обробляти та відображати структуровані дані у подальших частинах інтерфейсу.

Після встановлення та запуску локального сервера XAMPP з активними службами Apache та MariaDB було відкрито phpMyAdmin за адресою http://localhost/phpmyadmin/. Тут була створена реляційна база даних, яка містить таблиці для зберігання інформації про ринкові котирування, користувачів, торги, торгові стратегії та результати їх тестування (рисунок 3.3).



	Таблиця	Дія						Рядки	Тип	Зіставлення	Розмір
☐	market_quotes	☆	Переглянути	Структура	Пошук	Вставити	Очистити	Знищити	4	InnoDB	utf8mb4_general_ci	16.0 КБ
☐	strategies	☆	Переглянути	Структура	Пошук	Вставити	Очистити	Знищити	2	InnoDB	utf8mb4_general_ci	16.0 КБ
☐	strategy_tests	☆	Переглянути	Структура	Пошук	Вставити	Очистити	Знищити	2	InnoDB	utf8mb4_general_ci	32.0 КБ
☐	trades	☆	Переглянути	Структура	Пошук	Вставити	Очистити	Знищити	3	InnoDB	utf8mb4_general_ci	32.0 КБ
☐	users	☆	Переглянути	Структура	Пошук	Вставити	Очистити	Знищити	4	InnoDB	utf8mb4_general_ci	48.0 КБ

Рисунок 3.3 – Створена база даних

Структура бази даних забезпечує цілісність даних завдяки використанню зв’язків між таблицями, а також дає змогу ефективно опрацьовувати інформацію під час роботи системи. Код створення цієї бази даних наведено у додатку Б.

Таким чином, база даних готова для подальшого наповнення тестовими даними та інтеграції з серверною логікою проекту. В подальшому планується реалізація API на PHP для роботи з цією базою, а також впровадження заходів безпеки для захисту інформації.

Реалізація інтерфейсу системи TradeSys представляє собою вебсайт з використанням технології PHP та Bootstrap для створення адаптивного користувацького інтерфейсу.

Головна структура інтерфейсу організована у вигляді навігаційної панелі з центрально розташованими вкладками та правою секцією для відображення інформації про користувача. Навігаційна панель містить логотип системи

«TradeSys» зліва, набір вкладок у центрі та область привітання користувача з іконкою акаунту справа. Центральна навігація включає п'ять основних розділів: Панель, Стратегії, Ринкові дані, Операції та Налаштування (рисунок 3.4).

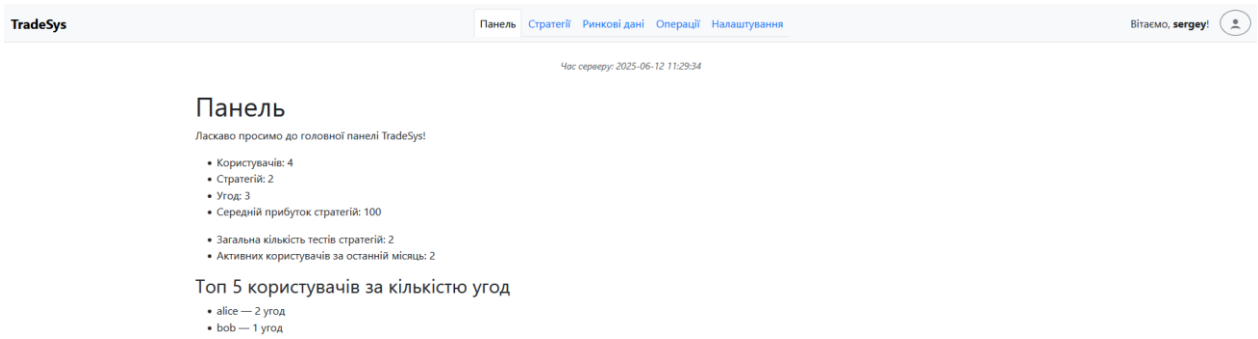


Рисунок 3.4 – Головна сторінка

Розділ «Панель» служить головною сторінкою системи та відображає загальну статистику функціонування платформи. Він включає відомості про загальну кількість користувачів, створених стратегій, проведених угод та середній показник прибутковості стратегій. Додатково представлена інформація про загальну кількість тестів стратегій, активних користувачів за останній місяць та рейтинг п'яти найактивніших користувачів за кількістю угод.

Вкладка «Стратегії» містить табличне представлення всіх торгових стратегій з детальною інформацією про кожну з них. Таблиця включає назву стратегії, її опис, кількість проведених тестів, середній показник прибутковості та дату створення. Дані отримуються через SQL-запит з об'єднанням таблиць стратегій та результатів їх тестування.

Розділ «Ринкові дані» відображає останні котирування фінансових інструментів у табличному форматі. Таблиця містить інформацію про назву активу, дату котирування, ціни відкриття, закриття, максимальну та мінімальну ціни за певний період. Система показує десять останніх записів котирувань, отриманих з бази даних.

Вкладка «Операції» має обмежений доступ лише для адміністраторів системи. Вона відображає останні торгові операції користувачів з вказанням імені користувача та дати проведення операції. Для користувачів без адміністративних прав показується повідомлення про обмеження доступу.

Інтерфейс використовує Bootstrap для забезпечення адаптивного дизайну та стилізації компонентів. Застосовуються стандартні класи Bootstrap для створення таблиць, кнопок, навігаційних елементів та системи сітки. Додатково використовуються власні CSS-стилі для налаштування позиціонування центральної навігації та візуального оформлення елементів.

Система інтегрована з Chart.js для можливості створення графічних візуалізацій даних, хоча в поточній реалізації графіки ще не використовуються. Весь контент відображається українською мовою з правильним налаштуванням кодування UTF-8 та мета-тегів для адаптивності на мобільних пристроях.

Програмний код головної сторінки наведений в додатку В.

Сторінка реєстрації реалізована як окремий PHP-файл з чистим та мінімалістичним інтерфейсом для створення нових облікових записів користувачів, що наведено в додатку Г. Система спочатку перевіряє, чи користувач вже авторизований через перевірку сесійної змінної user_id, і в разі позитивного результату автоматично перенаправляє на головну панель dashboard.php, запобігаючи повторній реєстрації вже авторизованих користувачів.

Форма реєстрації містить три обов'язкових поля для введення даних користувача. Перше поле призначене для введення імені користувача з типом «text» та обов'язковим атрибутом required для валідації на стороні браузера. Друге поле призначене для електронної пошти з відповідним типом «email», що забезпечує автоматичну валідацію формату email-адреси. Третє поле для пароля має тип «password», що приховує введені символи для забезпечення безпеки.

Система обробки помилок реалізована через GET-параметри URL. Якщо під час процесу реєстрації виникають помилки, вони передаються назад на сторінку реєстрації через параметр error і відображаються користувачу у вигляді

червоного попереджувального повідомлення з використанням класу alert-danger від Bootstrap. Це забезпечує зворотний зв'язок з користувачем про будь-які проблеми під час створення акаунту.

Кнопка подання форми стилізована як зелена кнопка успіху, що займає повну ширину контейнера форми та візуально підкреслює дію реєстрації. Форма налаштована на відправку даних методом POST до файлу register_process.php, який відповідає за серверну обробку реєстраційних даних.

У нижній частині форми розміщена навігаційна секція для користувачів, які вже мають облікові записи. Ця секція містить текст «Вже маєте акаунт?» з посиланням на сторінку входу login.php, оформленим як кнопка-посилання. Така організація забезпечує зручну навігацію між формами авторизації та реєстрації без необхідності використання браузерної навігації.

Сторінка входу в систему реалізована як спрощена форма авторизації з мінімалістичним дизайном та інтуїтивним користувацьким інтерфейсом. На початку виконання скрипта, який наведений в додатку Ґ, проводиться перевірка активної сесії користувача через змінну user_id, і якщо користувач вже авторизований, система автоматично перенаправляє його на головну сторінку index.php, уникаючи необхідності повторного входу.

Форма входу містить два основних поля для введення облікових даних користувача. Перше поле призначене для імені користувача з типом text та має атрибут autofocus, який автоматично встановлює фокус на це поле при завантаженні сторінки, покращуючи користувацький досвід. Друге поле призначене для пароля з відповідним типом password, що забезпечує приховування введених символів для захисту конфіденційної інформації. Обидва поля мають обов'язковий атрибут required для клієнтської валідації.

Система обробки та відображення помилок авторизації реалізована через GET-параметри URL. Якщо під час процесу входу виникають помилки аутентифікації або валідації, вони передаються назад на сторінку входу через параметр error та відображаються користувачу у формі червоного попереджувального блоку з використанням Bootstrap-класу alert-danger. Це

забезпечує негайний зворотний зв'язок про проблеми з обліковими даними або технічні помилки системи.

Кнопка подання форми оформлена як основна синя кнопка Bootstrap, що займає повну ширину контейнера форми та візуально підкреслює основну дію входу в систему. Форма налаштована на відправку даних методом POST до обробника login_process.php, який відповідає за серверну валідацію облікових даних та створення користувацької сесії.

У нижній секції форми розміщена додаткова навігація для нових користувачів системи. Ця область містить текстове запитання «Немає акаунту?» з відповідним посиланням на сторінку реєстрації register.php, оформленим як кнопка-посилання Bootstrap. Така організація забезпечує плавний перехід між процесами входу та реєстрації без порушення користувацького досвіду.

Сторінка профілю користувача реалізована як інформаційна панель для відображення персональних даних авторизованого користувача. Інтерфейс побудований на основі Bootstrap з використанням світлого фону та центрованого розташування основного контенту через контейнер з верхнім відступом.

Структура сторінки організована у вигляді картки з тінню, що створює візуальну глибину та фокусує увагу на інформації профілю. Картка обмежена максимальною шириною 500 пікселів та центрована горизонтально через клас mx-auto, забезпечуючи оптимальну читабельність на різних розмірах екранів. Заголовок картки містить назву «Профіль користувача» та виділений через окремий блок card-header з центрованим текстом (рисунок 3.5).

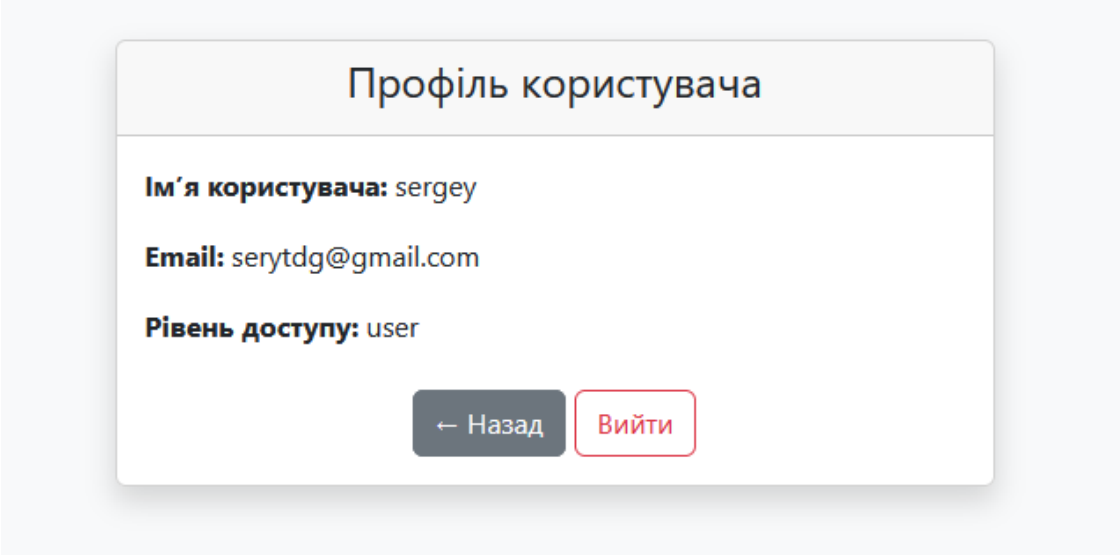


Рисунок 3.5 – Профіль користувача

Основний контент профілю розміщений у тілі картки та містить три ключових елементи інформації про користувача. Перший елемент відображає ім'я користувача, отримане з серверної змінної. Другий елемент показує електронну адресу користувача з аналогічним захистом від небезпечного коду. Третій елемент демонструє рівень доступу користувача в системі, що може включати різні ролі або права доступу.

Навігаційна секція розташована в нижній частині картки з центрованим вирівнюванням та верхнім відступом для візуального розділення від основної інформації. Ця секція містить дві функціональні кнопки для взаємодії користувача з системою. Перша кнопка «Назад» забезпечує повернення на головну сторінку index.php, дозволяючи користувачу легко повернутися до основного інтерфейсу системи.

Друга кнопка «Вийти» має стиль outline-danger, що візуально підкреслює її функцію завершення сесії користувача. Ця кнопка перенаправляє на обробник logout.php, який відповідає за коректне завершення користувацької сесії та очищення відповідних даних. Використання червоного кольору для кнопки виходу є стандартною практикою інтерфейсного дизайну, що інтуїтивно сигналізує про завершення роботи.

Лістинг програмного коду цієї сторінки наведено в додатку Д.

3.2 Тестування системи

Проведено комплексне тестування веб-системи TradeSys, яка призначена для управління торговими стратегіями та аналізу ринкових даних. Тестування охоплювало функціональні, інтерфейсні та безпекові аспекти системи з метою виявлення можливих дефектів та оцінки загальної якості програмного продукту.

Система складається з декількох основних модулів, включаючи головну панель управління, систему авторизації та реєстрації, профіль користувача та різні функціональні розділи для роботи зі стратегіями, ринковими даними та операціями.

Модуль авторизації показав стабільну роботу при введенні коректних облікових даних користувача. Система правильно перевіряє наявність активної сесії та перенаправляє авторизованих користувачів на відповідні сторінки. Функція автоматичного фокусування на поле введення імені користувача працює коректно та покращує користувацький досвід.

Процес реєстрації нових користувачів функціонує належним чином з валідацією обов'язкових полів на клієнтській стороні. Система коректно обробляє помилки реєстрації через GET-параметри та відображає їх користувачу у зрозумілому форматі. Перевірка унікальності електронної пошти та імені користувача працює стабільно (рисунок 3.6).

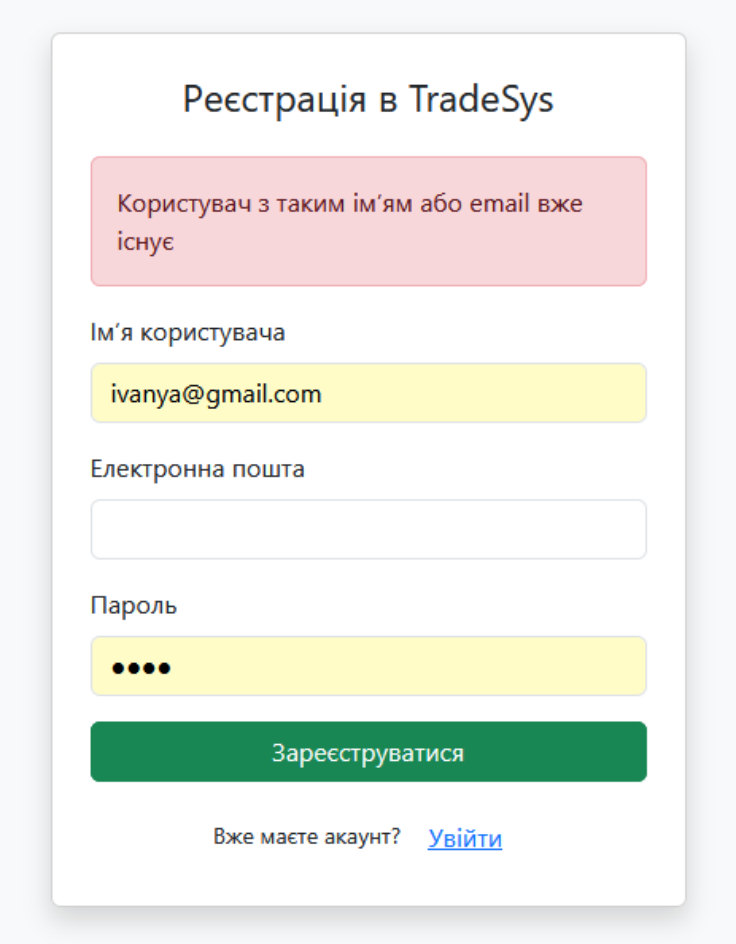


Рисунок 3.6 – Тестування реєстрації

Виявлено, що система не має додаткових заходів безпеки, таких як обмеження кількості спроб входу, що може створювати уразливість для брут-форс атак. Також відсутня функція відновлення пароля, що може ускладнити процес відновлення доступу для користувачів.

Головна панель демонструє коректне відображення статистичної інформації про систему, включаючи кількість користувачів, стратегій, угод та середній показник прибутковості. SQL-запити для отримання статистики виконуються ефективно та повертають актуальні дані з бази даних.

Навігаційна система з вкладками працює стабільно завдяки використанню Bootstrap JavaScript компонентів. Перемикання між розділами відбувається плавно без перезавантаження сторінки (рисунок 3.7).

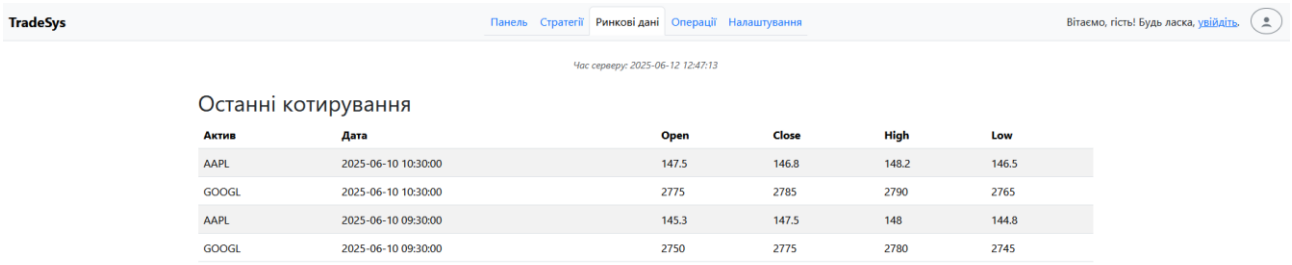


Рисунок 3.7 – Тестування навігаційної панелі

Розмежування доступу для адміністраторів функціонує правильно, приховуючи або показуючи відповідні розділи залежно від ролі користувача (рисунок 3.8).

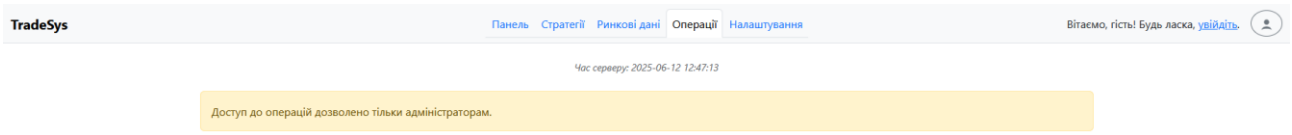


Рисунок 3.8 – Тестування розмежування доступу

Розділ стратегій коректно відображає табличну інформацію з об'єднанням даних з кількох таблиць бази даних. Статистичні показники, такі як кількість тестів та середній прибуток, розраховуються правильно та відображаються у зрозумілому форматі.

Секція ринкових даних стабільно показує останні котирування фінансових інструментів з правильним сортуванням за датою. Відображення цінових показників відбувається коректно з належним форматуванням числових значень.

Розділ операцій функціонує згідно з налаштуваннями доступу, показуючи інформацію лише адміністраторам та відображаючи попереджувальне повідомлення для звичайних користувачів. Це підтверджує правильну реалізацію системи безпеки та розмежування прав.

Система демонструє базовий рівень безпеки через використання сесійної авторизації та захист від XSS-атак. Перевірка ролей користувачів працює коректно та запобігає несанкціонованому доступу до адміністративних функцій.

Однак виявлено потенційні уразливості у відсутності додаткових заходів безпеки, таких як CSRF-токени для форм, обмеження швидкості запитів та додаткова валідація на серверній стороні. Система також не має механізмів логування безпекових подій.

Основними проблемами, виявленими під час тестування, є відсутність розширених заходів безпеки та обмежена функціональність системи відновлення доступу.

Також варто розглянути можливість додавання клієнтської та серверної валідації даних, покращення системи логування та впровадження механізмів резервного копіювання даних. Інтеграція з Chart.js потребує активного використання для повноцінної візуалізації торгових даних.

Система TradeSys демонструє стабільну базову функціональність з коректною реалізацією основних модулів авторизації, відображення даних та управління користувачами. Інтерфейс користувача інтуїтивний та адаптивний, забезпечуючи гарний користувацький досвід.

Основні функціональні вимоги виконані задовільно, але система потребує покращення в аспектах безпеки та розширення функціональних можливостей. Рекомендується продовжити розробку з акцентом на підвищення рівня безпеки та додавання нових аналітичних інструментів для повноцінної роботи з торговими стратегіями.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку збуту продукту

Автоматизована система торгівлі на фінансовому ринку є сучасним програмним продуктом, що забезпечує автоматичне виконання торгових операцій на підставі заданих алгоритмів. Вона характеризується високою швидкістю обробки ринкових даних, стабільністю в умовах навантаження, широкою функціональністю для роботи з різними фінансовими інструментами та інтеграцією з алгоритмами штучного інтелекту. Її інтерфейс орієнтований на зручність користувача, а архітектура дозволяє масштабування під великі обсяги операцій.

Пропонований виріб не є цілковито новим, адже на ринку вже існують подібні рішення. Втім, він виступає як удосконалена модифікація з розширеними можливостями, адаптованими до сучасних технологій і потреб фінансових гравців. Його функціонал перевищує типові пропозиції за рахунок машинного навчання, гнучких налаштувань та підвищеного рівня безпеки.

Цільовими покупцями виступають професійні трейдери, інвестиційні компанії, приватні інвестори, а також фінансові організації, які зацікавлені у зниженні впливу людського фактора на процес торгівлі. Ринок збуту охоплює як внутрішні, так і міжнародні ринки фінансових послуг, особливо у країнах з розвиненою біржовою інфраструктурою. Очікується, що попит на такі системи зростатиме, адже автоматизація стає ключовим чинником конкурентоспроможності в торгівлі.

Продаж продукту доцільно здійснювати через інтернет – за допомогою власного сайту, демонстраційних версій, маркетингових кампаній та партнерства з брокерськими платформами. Важливою складовою комерційного успіху є наявність якісного сервісного обслуговування: допродажні консультації, впровадження системи під потреби клієнта та технічна підтримка після продажу, що включає оновлення, усунення неполадок та модернізацію функцій.

Обсяги реалізації залежать від ефективності маркетингової стратегії та якості продукту, але при правильному підході вони можуть досягати значних масштабів, особливо у середовищі професійної біржової торгівлі. Конкурентами виступають такі продукти, як MetaTrader, NinjaTrader та TradingView. Їх програмні рішення мають високий рівень технічної реалізації, широкий набір функцій та визнану надійність. Проте вони не завжди забезпечують належну гнучкість у кастомізації або інтеграцію з новітніми методами аналізу даних.

Продукція конкурентів відрізняється за дизайном, структурою, рівнем технічної підтримки, а також ціною: від безкоштовного базового функціоналу до ліцензій з підпискою. У цьому контексті конкурентними перевагами нового продукту є висока якість і надійність, доступність професійного сервісу, вигідна система знижок, персоналізація продукту під потреби клієнтів.

Життєвий цикл системи охоплює кілька етапів: впровадження, ріст використання, оновлення функціоналу та, зрештою, модернізація чи заміна внаслідок змін на ринку. Враховуючи стрімкий розвиток фінансових технологій, очікується, що продукт залишатиметься актуальним упродовж щонайменше 5–7 років із можливістю подальшого оновлення.

Такий комплексний аналіз свідчить про доцільність розробки та виведення на ринок нової автоматизованої системи торгівлі, яка здатна зайняти гідне місце серед сучасних фінансових рішень.

4.2 Розрахункова частина

У розробці проєкту брали участь три фахівці: розробник програмного забезпечення, тестувальник та UI/UX дизайнер.

Розробник програмного забезпечення здійснив основну технічну реалізацію системи: проектування архітектури, написання коду, налагодження взаємодії з біржовими API та реалізацію логіки автоматизованої торгівлі. Тривалість його участі у проєкті склала 160 годин, погодинна оплата становила 80 гривень. Нарахована заробітна плата розраховується як $160 \cdot 80 = 12\,800$ гривень. Єдиний соціальний внесок (22%) від цієї суми становить $12\,800 \cdot 0.22 =$

2 816 гривень. Загальні витрати на оплату праці розробника з урахуванням ЄСВ склали 12 800 + 2 816 = 15 616 гривень.

Тестувальник відповідав за перевірку функціональної коректності системи, виявлення та документування помилок, а також перевірку надійності алгоритмів торгівлі в умовах різних ринкових сценаріїв. Він працював протягом 60 годин, з погодинною оплатою 80 гривень. Заробітна плата становила 60 · 80 = 4 800 гривень. ЄСВ: 4 800 · 0.22 = 1056 гривень. Повна сума витрат на оплату праці тестувальника: 4 800 + 1056 = 5 856 гривень.

UI/UX дизайнер створив логіку взаємодії користувача з системою, розробив інтерфейс торгової платформи, візуальні елементи та адаптивну структуру. Загальна тривалість його роботи склала 60 годин. Погодинна оплата – 80 гривень. Заробітна плата розраховується як 60 · 80 = 4 800 гривень. ЄСВ: 4 800 · 0.22 = 1056 гривень. Загальні витрати на оплату праці дизайнера становлять 4 000 + 1056 = 5 856 гривень.

Таким чином, сумарні витрати на оплату праці всієї команди з урахуванням ЄСВ становлять:

15 616 (розробник) + 5 856 (тестувальник) + 5 856 (дизайнер) = 27 328 гривні.

Для розрахунку витрат на електроенергію беремо один робочий комп’ютер, на якому проводилася розробка, тестування та дизайн. Потужність цього ПК становить 200 Вт. Для переведення у кіловати:

P = 200 / 1000 = 0.2 кВт

Тариф за електроенергію для фізичних осіб на 2025 рік – 6.0 грн/кВт·год.

Для наочності в таблиці 4.1 наведено розрахунок витрат на електроенергію окремо для кожного працівника

Таблиця 4.1 – Розрахунок витрат на електроенергію

№	Посада	Години роботи	Потужність (кВт)	Тариф (грн/кВт·год)	Витрати
1	Розробник	160	0.2	6.0	192

Продовження таблиці 4.1

2	Тестувальник	60	0.2	6.0	72
3	UI/UX дизайнер	50	0.2	6.0	60

Таким чином, витрати на електроенергію під час реалізації проєкту становлять 324 гривні.

Ці витрати хоч і незначні порівняно з оплатою праці, але додають точності до загального кошторису.

У процесі реалізації автоматизованої системи торгівлі на фінансовому ринку виникає необхідність у витратах на забезпечення ефективного зв'язку між учасниками команди розробки. Такі витрати охоплюють мобільний зв'язок, доступ до мережі Інтернет, а за потреби – й інші засоби комунікації, що використовуються для обговорення, координації дій та віддаленої взаємодії працівників.

Загальна сума витрат на зв'язок розраховується за формулою:

$С_{зв'яз} = С_{моб} + С_{інтернет} + С_{інші}$,

де:

$С_{зв'яз}$ – загальні витрати на зв'язок;

$С_{моб}$ – витрати на мобільний зв'язок;

$С_{інтернет}$ – витрати на інтернет;

$С_{інші}$ – інші витрати на зв'язок (не використовувалися у цьому проєкті, тому становлять 0 грн).

Для розробника передбачено 150 грн на мобільний зв'язок і 300 грн на інтернет, оскільки він активно працював над основною логікою системи, використовував хмарні ресурси та брав участь у технічних онлайн-нарадах. Загальна сума витрат на зв'язок для розробника становить 450 грн.

Тестувальник витратив менше ресурсів на зв'язок, оскільки його завдання не потребували постійного з'єднання з віддаленими сервісами. Витрати на мобільний зв'язок становили 100 грн, на інтернет – 200 грн, загалом – 300 грн.

UI/UX дизайнер також використовував онлайн-зв’язок для узгодження макетів та обговорення інтерфейсних рішень. Його витрати склали 100 грн на мобільний зв’язок і 200 грн на інтернет. Загальні витрати – 300 грн.

Таким чином, загальна сума витрат на зв’язок для всіх учасників проєкту становить 1050 грн. Ці витрати є обґрунтованими, оскільки забезпечують необхідний рівень комунікації, що є критично важливим під час командної роботи над ІТ-проєктом.

Загальна сума витрат на розробку проєкту складається з наступних компонентів:

$С_{зп} + С_{вн} = 5\,856 + 5\,856 + 15\,616 = 27\,328$ грн

$С_{ел} = 192 + 72 + 60 = 324$ грн

$С_{зв'яз} = 450 + 300 + 300 = 1{,}050$ грн

$С_{інш} = 0$ грн

Загальні витрати на розробку проєкту склали:

$С_{розр} = 27\,328 + 6\,160 + 324 + 1\,050 + 0 = 34\,862$ грн

Отже, загальні витрати на розробку автоматизованої системи торгівлі на фінансовому ринку становлять 34 862 грн.

Доречно провести розрахунок економічного ефекту, спираючись на ключові переваги автоматизації саме у сфері біржової торгівлі. Основними напрямами, у яких проявляється економічна вигода, є зростання прибутку завдяки підвищенню продуктивності та точності, зменшення кількості збиткових угод, зниження витрат на оплату праці трейдерів, а також можливість цілодобової роботи системи без залучення додаткових ресурсів.

Найбільший економічний ефект досягається за рахунок того, що автоматизована система здатна реагувати на ринкові зміни швидше за людину, працювати без перерв і емоційного впливу. Це дозволяє значно збільшити кількість вигідних угод і зменшити ймовірність помилок.

До впровадження автоматизованої системи щоденно вручну укладалося в середньому 25 угод, з яких прибутковими були 60 %, тобто 15 угод. Середній

прибуток з однієї вигідної угоди становив 200 грн, а середній збиток – 150 грн. За місяць (20 торгових днів) загальний прибуток складав:

$(15 \times 200 - 10 \times 150) \times 20 = (3000 - 1500) \times 20 = 30\,000$ грн.

Після впровадження автоматизованої системи середня кількість угод зросла до 60 на день, при цьому частка прибуткових угод зросла до 75 %, тобто 45 угод. Середній прибуток з прибуткової угоди залишився 200 грн, а збиток – 150 грн. За місяць прибуток склав:

$(45 \times 200 - 15 \times 150) \times 20 = (9000 - 2250) \times 20 = 135\,000$ грн.

Таким чином, приріст місячного прибутку після автоматизації становить $135\,000 - 30\,000 = 105\,000$ грн, або 1 260 000 грн на рік. Це і є основний економічний ефект від підвищення продуктивності.

Щодо скорочення витрат на персонал: до автоматизації для ведення торгів працювали два трейдери з заробітною платою по 25 000 грн на місяць. Після впровадження системи торгівлю супроводжує лише один фахівець-аналітик із зарплатою 30 000 грн. Таким чином, щомісячна економія становить

$(25\,000 \times 2 - 30\,000) = 20\,000$ грн, або 240 000 грн на рік.

До автоматизації фіксувалося до 12 критичних помилок на рік через людський фактор. Середній збиток від кожної помилки – 3 000 грн. Після впровадження системи кількість помилок зменшилася до 2 на рік. Отже, економічний ефект за цим напрямом становить $(12 - 2) \times 3\,000 = 30\,000$ грн на рік.

Сумарний річний економічний ефект від впровадження автоматизованої системи становить:

1 260 000 грн (прибуток) + 240 000 грн (зарплата) + 30 000 грн (зменшення помилок) = 1 530 000 грн.

Це свідчить про високу ефективність системи та її окупність упродовж короткого терміну.

Окупність проєкту визначається як період, за який витрати на розробку та впровадження системи будуть повністю компенсовані за рахунок отриманого економічного ефекту. У випадку реалізації автоматизованої системи торгівлі на

фінансовому ринку витрати на розробку склали 180 000 грн, включаючи оплату праці розробників, витрати на інфраструктуру та супутні витрати.

Загальний річний економічний ефект від впровадження системи, розрахований на основі підвищення прибутковості торгів, зменшення витрат на персонал та скорочення кількості критичних помилок, становить 1 530 000 грн.

Термін окупності розраховується за формулою:

$T_{окуп} = C_{розр} / E$

$T_{окуп} = 34\ 862 \ / 1\ 530\ 000 \approx 0.022$ року, тобто приблизно тиждень.

Також можна розрахувати коефіцієнт економічної ефективності:

$K_{еф} = E / C_{розр} = 1\ 530\ 000 / 34\ 862 \approx 43{,}887$

Це означає, що система забезпечує економічний ефект, який у 43,887 разів перевищує витрати на її розробку. Отже, проєкт є економічно доцільним і високоефективним. Крім того, у порівнянні з аналогічними рішеннями на ринку, розроблена система має нижчу вартість впровадження, кращу адаптацію до конкретних потреб замовника та розширений набір функцій, що забезпечує її конкурентоспроможність.

4.3 Обґрунтування необхідності розробки

Розробка автоматизованої системи торгівлі на фінансовому ринку зумовлена сучасними вимогами до швидкості, точності та ефективності здійснення біржових операцій. У реаліях висококонкурентного середовища, де навіть незначні коливання цін можуть призвести до значних фінансових наслідків, використання автоматизованих інструментів стає необхідністю для забезпечення стабільного доходу та зменшення втрат. Традиційні підходи, що базуються виключно на людському аналізі й прийнятті рішень, не здатні забезпечити потрібну швидкість реакції на зміну ринкових умов. Програмні рішення дозволяють проводити моніторинг фінансових інструментів у режимі реального часу, здійснювати технічний аналіз та оперативно відкривати або закривати позиції на основі заздалегідь визначених алгоритмів.

Пропонований програмний продукт задовольняє потреби трейдерів, інвесторів, фінансових компаній і банків у потужному, адаптивному та надійному інструменті, який виконує дії без участі людини, знижуючи рівень помилок, викликаних суб'єктивними рішеннями чи стресовими ситуаціями. Система дозволяє встановлювати індивідуальні стратегії, наприклад, агресивну торгівлю з високим ризиком або консервативну з акцентом на збереження капіталу. Така гнучкість відкриває можливості для широкого кола користувачів, незалежно від рівня досвіду та знань.

Впровадження автоматизованої системи забезпечує економічний ефект через зменшення витрат на оплату праці аналітиків та трейдерів, скорочення часу, необхідного для прийняття рішень, та мінімізацію фінансових втрат, які можуть виникати через упущені можливості або неправильні дії оператора. Завдяки тому, що система працює безперервно, у тому числі вночі або під час значних коливань ринку, вона здатна реагувати на зміни миттєво, що суттєво збільшує потенційну прибутковість.

Окрім цього, автоматизація сприяє підвищенню масштабованості бізнесу: один програмний модуль може обробляти сотні одночасних операцій, тоді як ручне керування потребувало би десятків співробітників. Це дозволяє фінансовим компаніям розширювати свою діяльність без пропорційного збільшення витрат.

У довгостроковій перспективі використання такої системи позитивно впливає на загальні економічні показники підприємства: зростають доходи, зменшується кількість збиткових операцій, підвищується рентабельність інвестицій. Таким чином, розробка та впровадження автоматизованої системи торгівлі не лише відповідає актуальним вимогам ринку, але й забезпечує конкурентну перевагу та стале фінансове зростання.

ВИСНОВКИ

В рамках кваліфікаційної роботи було успішно розроблено веб-систему TradeSys для управління торговими стратегіями та аналізу фінансових даних. Проєкт повністю відповідає поставленій меті створення функціональної платформи для автоматизації процесів торгівлі на фінансових ринках та забезпечення ефективного управління торговими операціями.

Розроблена система демонструє повну інтеграцію всіх заявлених компонентів, включаючи модуль управління користувачами, систему авторизації з розмежуванням доступу, інтерфейс для роботи з торговими стратегіями, аналітичну панель для відображення ринкових даних та операційну систему для моніторингу торгових операцій. Архітектура системи побудована на сучасних веб-технологіях з використанням PHP, MySQL та Bootstrap, що забезпечує стабільність роботи та можливість масштабування.

Рекомендується в подальшому впровадити додаткові заходи безпеки, включаючи CSRF-токени для всіх форм, обмеження швидкості запитів для запобігання DDoS-атакам, двофакторну аутентифікацію для підвищення безпеки облікових записів та систему логування всіх безпекових подій. Також необхідно додати функцію відновлення пароля та блокування облікових записів після невдалих спроб входу.

Перспективними напрямками розвитку є інтеграція з реальними торговими API для отримання актуальних ринкових даних, впровадження алгоритмів машинного навчання для прогнозування ринкових тенденцій, додавання інструментів для створення та бектестингу власних торгових стратегій, реалізація системи сповіщень про важливі ринкові події.

Виконана кваліфікаційна робота демонструє успішне застосування сучасних технологій веб-розробки для створення спеціалізованої системи управління торговими операціями. Розроблене рішення має практичну цінність та може служити основою для подальшого розвитку більш складних торгових систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Why trade futures with NinjaTrader?. *Ninjatrader*. URL: https://ninjatrader.com/ (дата звернення: 02.03.2025).
2. MetaTrader 5 Trading Platform for Forex, Stocks, Futures. *MetaTrader 5 Trading Platform for Forex, Stocks, Futures*. URL: https://www.metatrader5.com/en (дата звернення: 02.03.2025).
3. Why trade futures with NinjaTrader?. *Ninjatrader*. URL: https://ninjatrader.com/ (дата звернення: 02.03.2025).
4. QuantConnect - Open Source Algorithmic Trading Platform. *Open Source Algorithmic Trading Platform. - QuantConnect.com*. URL: https://www.quantconnect.com/ (дата звернення: 02.03.2025).
5. Online FlowChart & Diagrams Editor - Mermaid Live Editor. *Online FlowChart & Diagrams Editor - Mermaid Live Editor*. URL: https://mermaid.live/ (дата звернення: 21.03.2025).
6. MariaDB Foundation - MariaDB.org. *MariaDB.org*. URL: https://mariadb.org/ (дата звернення: 12.04.2025).
7. Bootstrap. *Bootstrap · The most popular HTML, CSS, and JS library in the world.* URL: https://getbootstrap.com/ (дата звернення: 28.04.2025).
8. Welcome to Flask – Flask Documentation (3.1.x). *Welcome to Flask – Flask Documentation (3.1.x)*. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 28.04.2025).

ДОДАТКИ

Додаток А

Система управління профілем користувача

```
<?php
session_start();
require 'db.php';
if (!isset($_SESSION['user_id'])) {
    header('Location: login.php');
    exit;
}

$user_id = $_SESSION['user_id'];
$errors = [];
$success = '';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['change_password'])) {
        $current_password = $_POST['current_password'] ?? '';
        $new_password = $_POST['new_password'] ?? '';
        $confirm_password = $_POST['confirm_password'] ?? '';
        $stmt = $pdo->prepare("SELECT password FROM users
WHERE user_id = ?");
        $stmt->execute([$user_id]);
        $user = $stmt->fetch(PDO::FETCH_ASSOC);

        if (!$user || !password_verify($current_password,
$user['password'])) {
            $errors[] = 'Поточний пароль введено невірно.';
        } elseif ($new_password !== $confirm_password) {
            $errors[] = 'Новий пароль і підтвердження не
співпадають.';
        } elseif (strlen($new_password) < 6) {
            $errors[] = 'Новий пароль має бути не менше 6
символів.';
        } else {
           $new_hashed = password_hash($new_password,
PASSWORD_DEFAULT);
            $stmt = $pdo->prepare("UPDATE users SET password
= ? WHERE user_id = ?");
            $stmt->execute([$new_hashed, $user_id]);
            $success = 'Пароль успішно змінено.';
        }
    }
    if (isset($_POST['update_profile'])) {
        $email = filter_var($_POST['email'] ?? '',
FILTER_VALIDATE_EMAIL);
        $first_name = trim($_POST['first_name'] ?? '');
        $last_name = trim($_POST['last_name'] ?? '');
        $phone = trim($_POST['phone'] ?? '');

        if (!$email) {
```

```
                $errors[] = 'Введіть коректну електронну пошту.';
            }

            if (empty($errors)) {
                $stmt = $pdo->prepare("UPDATE users SET email =
?, first_name = ?, last_name = ?, phone = ? WHERE user_id = ?");
                $stmt->execute([$email, $first_name, $last_name,
$phone, $user_id]);
                $success = 'Профіль оновлено.';
            }
        }
        if (isset($_POST['upload_avatar']) &&
isset($_FILES['avatar'])) {
            $file = $_FILES['avatar'];
            if ($file['error'] === UPLOAD_ERR_OK) {
                $allowed = ['image/jpeg', 'image/png',
'image/gif'];
                if (in_array($file['type'], $allowed)) {
                    $upload_dir = __DIR__ . '/avatars/';
                    if (!is_dir($upload_dir)) {
                        mkdir($upload_dir, 0755, true);
                    }

                    $ext = pathinfo($file['name'],
PATHINFO_EXTENSION);
                    $new_filename = $user_id . '.' . $ext;
                    $filepath = $upload_dir . $new_filename;

                    if (move_uploaded_file($file['tmp_name'],
$filepath)) {
                        // Зберігаємо шлях в базі
                        $stmt = $pdo->prepare("UPDATE users SET
avatar = ? WHERE user_id = ?");
                        $stmt->execute(['avatars/' .
$new_filename, $user_id]);
                        $success = 'Аватар завантажено успішно.';
                    } else {
                        $errors[] = 'Помилка при завантаженні
файлу.';
                    }
                } else {
                    $errors[] = 'Дозволені лише файли JPG, PNG,
GIF.';
                }
            } else {
                $errors[] = 'Помилка при завантаженні файлу.';
            }
        }
    }
    $stmt = $pdo->prepare("SELECT email, username FROM users
WHERE user_id = ?");
    $stmt->execute([$user_id]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);?>
```

Додаток Б

SQL-запити створення таблиць

```
CREATE TABLE market_quotes (
  quote_id INT AUTO_INCREMENT PRIMARY KEY,
  asset_name VARCHAR(255) NOT NULL,
  quote_date TIMESTAMP NOT NULL,
  open_price FLOAT NOT NULL,
  close_price FLOAT NOT NULL,
  high_price FLOAT NOT NULL,
  low_price FLOAT NOT NULL
);

CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  access_level VARCHAR(20) NOT NULL
);

CREATE TABLE trades (
  trade_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  trade_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE
CASCADE
);

CREATE TABLE strategies (
  strategy_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  parameters JSON,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE strategy_tests (
  test_id INT AUTO_INCREMENT PRIMARY KEY,
  strategy_id INT NOT NULL,
  test_date TIMESTAMP NOT NULL,
  initial_balance FLOAT NOT NULL,
  final_balance FLOAT NOT NULL,
  performance FLOAT NOT NULL,
  result VARCHAR(10),
  profit_loss FLOAT,
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (strategy_id) REFERENCES
strategies(strategy_id) ON DELETE CASCADE
);
```

Додаток В

Програмний код головної сторінки

```
<?php
if (session_status() == PHP_SESSION_NONE) {
    session_start();
}

require 'db.php';

if (!isset($_SESSION['user_id']) || ($_SESSION['role'] ?? '')
!== 'admin') {
    $isAdmin = false;
} else {
    $isAdmin = true;
}

// Визначення посилання на акаунт
$accountLink = isset($_SESSION['user_id']) ? 'profile.php' :
'login.php';
?>
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1" />
  <title>TradeSys — Панель</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootst
rap.min.css" rel="stylesheet" />
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstra
p.bundle.min.js"></script>
  <style>
    .nav-link {
      font-weight: 500;
    }
    .navbar-nav-center {
      position: absolute;
      left: 50%;
      transform: translateX(-50%);
    }
  </style>
  <!-- Додай у <head> -->
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>
<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light
border-bottom">
  <div class="container-fluid">
```

```
        <a class="navbar-brand fw-bold" href="#">TradeSys</a>
        <div class="navbar-nav navbar-nav-center d-none d-lg-
flex">
          <ul class="nav nav-tabs" id="navTabs" role="tablist">
            <li class="nav-item" role="presentation">
              <button class="nav-link active" id="panel-tab"
data-bs-toggle="tab" data-bs-target="#panel" type="button"
role="tab" aria-controls="panel" aria-
selected="true">Панель</button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="strategies-tab" data-
bs-toggle="tab" data-bs-target="#strategies" type="button"
role="tab" aria-controls="strategies" aria-
selected="false">Стратегії</button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="market-tab" data-bs-
toggle="tab" data-bs-target="#market" type="button" role="tab"
aria-controls="market" aria-selected="false">Ринкові дані</button>
            </li>
            <li class="nav-item" role="presentation">
              <button class="nav-link" id="operations-tab" data-
bs-toggle="tab" data-bs-target="#operations" type="button"
role="tab" aria-controls="operations" aria-
selected="false">Операції</button>
            </li>
            <li class="nav-item" role="presentation">
    <a href="settings.php" class="nav-link" id="settings-tab"
role="tab" aria-controls="settings" aria-
selected="false">Налаштування</a>
            </li>
          </ul>
        </div>
    <div class="d-flex ms-auto align-items-center gap-3">
      <?php if (isset($_SESSION['username'])): ?>
        <p class="mb-0">Вітаємо, <strong><?=
htmlspecialchars($_SESSION['username']) ?></strong>!</p>
      <?php else: ?>
        <p class="mb-0">Вітаємо, гість! Будь ласка, <a
href="login.php">увійдіть</a>.</p>
      <?php endif; ?>
      <a href="<?= htmlspecialchars($accountLink) ?>" class="btn
btn-outline-secondary rounded-circle" title="Акаунт">
        <svg xmlns="http://www.w3.org/2000/svg" width="20"
height="20" fill="currentColor" class="bi bi-person" viewBox="0 0
16 16">
          <path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6z"/>
          <path d="M14 14s-1-1.5-6-1.5S2 14 2 14s1-4 6-4 6 4 6
4z"/>
        </svg>
      </a>
```

```
      </div>
        </div>
      </nav>
      <div class="container my-4">
        <p class="text-center text-muted fst-italic small">
          Час серверу: <?= date('Y-m-d H:i:s') ?>
        </p>
      </div>
      <div class="container mt-4">
        <div class="tab-content" id="navTabsContent">
          <div class="tab-pane fade show active" id="panel"
role="tabpanel" aria-labelledby="panel-tab">
            <h1>Панель</h1>
            <p>Ласкаво просимо до головної панелі TradeSys!</p>
      <?php
      $users = $pdo->query("SELECT COUNT(*) FROM users")-
>fetchColumn();
      $strategies = $pdo->query("SELECT COUNT(*) FROM strategies")-
>fetchColumn();
      $trades = $pdo->query("SELECT COUNT(*) FROM trades")-
>fetchColumn();
      $avgProfit = $pdo->query("SELECT AVG(profit_loss) FROM
strategy_tests")->fetchColumn();

      echo "<ul>
        <li>Користувачів: $users</li>
        <li>Стратегій: $strategies</li>
        <li>Угод: $trades</li>
        <li>Середній прибуток стратегій: " . round($avgProfit, 2) .
"</li>
      </ul>";
$totalTests = $pdo->query("SELECT COUNT(*) FROM strategy_tests")-
>fetchColumn();
      $totalUsersActiveLastMonth = $pdo->query("
          SELECT COUNT(DISTINCT user_id) FROM trades
          WHERE trade_date >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
      ")->fetchColumn();

      echo "<ul>
        <li>Загальна кількість тестів стратегій: $totalTests</li>
        <li>Активних користувачів за останній місяць:
$totalUsersActiveLastMonth</li>
      </ul>";
      $topUsers = $pdo->query("
          SELECT u.username, COUNT(t.trade_id) AS trades_count
          FROM users u
          JOIN trades t ON u.user_id = t.user_id
          GROUP BY u.user_id
          ORDER BY trades_count DESC
          LIMIT 5
      ")->fetchAll(PDO::FETCH_ASSOC);
      ?>
      <h3>Топ 5 користувачів за кількістю угод</h3>
```

```
    <ul>
    <?php foreach ($topUsers as $user): ?>
        <li><?= htmlspecialchars($user['username']) ?> — <?=
$user['trades_count'] ?> угод</li>
    <?php endforeach; ?>
    </ul>

        </div>

        <div class="tab-pane fade" id="strategies"
role="tabpanel" aria-labelledby="strategies-tab">
    <?php
    // Стратегії
    $stmt = $pdo->query("
        SELECT s.name, s.description, s.created_at,
               COUNT(st.test_id) AS tests,
               ROUND(AVG(st.profit_loss), 2) AS avg_profit
        FROM strategies s
        LEFT JOIN strategy_tests st ON s.strategy_id =
st.strategy_id
        GROUP BY s.strategy_id
    ");
    echo "<h2>Стратегії</h2><table class='table table-bordered'>
    <thead><tr><th>Назва</th><th>Опис</th><th>Тестів</th><th>Сере
дній прибуток</th><th>Дата</th></tr></thead><tbody>";

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        echo "<tr>
            <td>" . htmlspecialchars($row['name']) . "</td>
            <td>" . htmlspecialchars($row['description']) .
"</td>
            <td>" . (int)$row['tests'] . "</td>
            <td>" . htmlspecialchars($row['avg_profit']) . "</td>
            <td>" . htmlspecialchars($row['created_at']) . "</td>
        </tr>";
    }

    echo "</tbody></table>";
    ?>
        </div>

        <div class="tab-pane fade" id="market" role="tabpanel"
aria-labelledby="market-tab">
    <?php
    $stmt = $pdo->query("
        SELECT * FROM market_quotes
        ORDER BY quote_date DESC
        LIMIT 10
    ");
    echo "<h2>Останні котирування</h2><table class='table table-
striped'>
    <thead><tr>
```

```
     <th>Актив</th><th>Дата</th><th>Open</th><th>Close</th><th>Hig
h</th><th>Low</th>
     </tr></thead><tbody>";

     while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
         echo "<tr>
           <td>" . htmlspecialchars($row['asset_name']) . "</td>
           <td>" . htmlspecialchars($row['quote_date']) . "</td>
           <td>" . htmlspecialchars($row['open_price']) . "</td>
           <td>" . htmlspecialchars($row['close_price']) . "</td>
           <td>" . htmlspecialchars($row['high_price']) . "</td>
           <td>" . htmlspecialchars($row['low_price']) . "</td>
         </tr>";
     }
     echo "</tbody></table>";
     ?>
         </div>
         <div class="tab-pane fade" id="operations"
role="tabpanel" aria-labelledby="operations-tab">
           <?php if ($isAdmin): ?>
               <?php
               $stmt = $pdo->query("
                   SELECT t.trade_date, u.username
                   FROM trades t
                   JOIN users u ON t.user_id = u.user_id
                   ORDER BY t.trade_date DESC
                   LIMIT 10
               ");
               echo "<h2>Останні операції</h2><table
class='table'>

<thead><tr><th>Користувач</th><th>Дата</th></tr></thead><tbody>";

               while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
                   echo "<tr><td>" .
htmlspecialchars($row['username']) . "</td><td>" .
htmlspecialchars($row['trade_date']) . "</td></tr>";
               }

               echo "</tbody></table>";
               ?>
           <?php else: ?>
               <div class="alert alert-warning">Доступ до операцій
дозволено тільки адміністраторам.</div>
           <?php endif; ?>
         </div>


       </div>
     </div>
     </body>
     </html>
```

Додаток Г

Реєстраційна форма

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Реєстрація — TradeSys</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootst
rap.min.css" rel="stylesheet">
</head>
<body class="bg-light">
    <div class="container d-flex justify-content-center
align-items-center vh-100">
        <div class="card p-4 shadow" style="width: 100%; max-
width: 400px;">
            <h4 class="text-center mb-4">Реєстрація в
TradeSys</h4>
            <?php if (isset($_GET['error'])): ?>
                <div class="alert alert-danger"><?=
htmlspecialchars($_GET['error']) ?></div>
            <?php endif; ?>
            <form action="register_process.php"
method="post">
                <div class="mb-3">
                    <label for="username" class="form-
label">Ім’я користувача</label>
                    <input type="text" name="username"
class="form-control" required>
                </div>
                <div class="mb-3">
                    <label for="email" class="form-
label">Електронна пошта</label>
                    <input type="email" name="email"
class="form-control" required>
                </div>
                <div class="mb-3">
                    <label for="password" class="form-
label">Пароль</label>
<input type="password" name="password" class="form-control"
required>
                </div>
                <button type="submit" class="btn btn-success
w-100">Зареєструватися</button></form>
            <div class="text-center mt-3">
                <small>Вже маєте акаунт?</small>
                <a href="login.php" class="btn btn-
link">Увійти</a></div></div>
    </div></body></html>
```

Додаток Ґ

Форма входу

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Вхід — TradeSys</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootst
rap.min.css" rel="stylesheet">
</head>
<body class="bg-light">
    <div class="container d-flex justify-content-center
align-items-center vh-100">
        <div class="card p-4 shadow" style="width: 100%; max-
width: 400px;">
            <h4 class="text-center mb-4">Вхід у TradeSys</h4>
            <?php if (isset($_GET['error'])): ?>
                <div class="alert alert-danger"><?=
htmlspecialchars($_GET['error']) ?></div>
            <?php endif; ?>
            <form action="login_process.php" method="post">
                <div class="mb-3">
                    <label for="username" class="form-
label">Ім’я користувача</label>
                    <input type="text" name="username"
class="form-control" required autofocus>
                </div>
                <div class="mb-3">
                    <label for="password" class="form-
label">Пароль</label>
                    <input type="password" name="password"
class="form-control" required>
                </div>
                <button type="submit" class="btn btn-primary
w-100">Увійти</button>
            </form>
            <div class="text-center mt-3">
    <small>Немає акаунту?</small>
    <a href="register.php" class="btn btn-
link">Зареєструватися</a>
</div>

        </div>

    </div>

</body>
</html>
```

Додаток Д

Сторінка акаунту користувача

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Профіль — TradeSys</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootst
rap.min.css" rel="stylesheet">
</head>
<body class="bg-light">
    <div class="container mt-5">
        <div class="card shadow mx-auto" style="max-width:
500px;">
            <div class="card-header text-center">
                <h4>Профіль користувача</h4>
            </div>
            <div class="card-body">
                <p><strong>Ім’я користувача:</strong> <?=
htmlspecialchars($user['username']) ?></p>
                <p><strong>Email:</strong> <?=
htmlspecialchars($user['email']) ?></p>
                <p><strong>Рівень доступу:</strong> <?=
htmlspecialchars($user['access_level']) ?></p>
                <div class="mt-4 text-center">
                    <a href="index.php" class="btn btn-
secondary">← Назад</a>
                    <a href="logout.php" class="btn btn-
outline-danger">Вийти</a>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```