

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАММНА РЕАЛІЗАЦІЯ СИМУЛЯТОРА БІНАРНОГО
ОПЦІОНУ РЕАЛЬНОГО ЧАСУ
SOFTWARE IMPLEMENTATION OF A REAL TIME BINARY
OPTION SIMULATOR

Виконав: студент(ка) 4 курсу, групи КНТ-128
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Лоцман Б.А.

(прізвище та ініціали)

Керівник Зеленьова І.Я.

(прізвище та ініціали)

Рецензент Касьян М.М.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Лоцман Богдана Анатолійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація симулятора бінарного опціону реального часу. Software Implementation of a Real Time Binary Option Simulator.

керівник проєкту (роботи) Зеленьова Ірина Яківна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Зеленьова І.Я., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання « 04 » квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Лоцман Б.А.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Зеленьова І.Я.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 72с., 38 рис., 13 табл., 2 дод., 13 джерел.

СИМУЛЯТОР, ВЕБ-ЗАСТОСУНОК, БІНАРНИЙ ОПЦІОН, ФРЕЙМВОРК, КРИПТОВАЛЮТА.

Об'єкт дослідження – симулятор бінарного опціону реального часу.

Предмет дослідження – програмне забезпечення симулятора бінарного опціону.

Мета роботи – створення веб-застосунку симулятора бінарного опціону вгору/вниз (Call/Put), для ознайомлення з роботою бінарних опціонів.

Для реалізації програмного продукту використовувалися наступні програмні засоби: фреймворк NestJS для розробки бек-серверу та бібліотеку React для розробки фронт-часини з використанням каскадних таблиць стилів CSS3 та мови розмітки HTML5, вся робота було виконана в текстовому редакторі Visual Studio Code.

У кваліфікаційній роботі було розроблено веб-застосунок симулятор бінарного опціону реального часу. Ця програма може використовуватися у освітній сфері, тому що користування нею дає базові навички роботи з бінарними опціонами.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 72 pages, 38 figures, 13 tables, 2 appendixes, 13 sources.

SIMULATOR, WEB APPLICATION, BINARY OPTION, FRAMEWORK, CRYPTOCURRENCY.

The object of study - real-time binary option simulator.

The subject of research - binary option simulator software.

The purpose of the work - create a web application of the binary option Call/Put simulator, to get acquainted with the work with binary options.

The following software tools were used to implement the software product: NestJS back-server development framework and React library for front-end part, using CSS3 cascading style sheets and HTML5 markup language, all work was done in Visual Studio Code text editor.

In the qualifying work, a real-time binary option simulator web application was developed. This program can be used in the field of education, because its use provides basic skills in working with binary options.

ЗМІСТ

Перелік скорочень та умовних познач	8
1 Аналіз предметної області	10
1.2 Класифікації бінарних опціонів	11
1.3 Аналіз сучасних брокерів	12
1.3.1 Огляд брокера «IQ Option»	12
1.3.2 Огляд брокера «Pocket Option»	14
1.3.3 Огляд брокера «Binary»	15
1.4 Постановка завдання роботи	17
1.5 Висновки за розділом 1	17
2 Аналіз технологій розробки	18
2.1 React	18
2.2 TypeScript	19
2.3 Redux-saga	20
2.3 Node.js	21
2.4 NestJS	22
2.5 Висновки за розділом 2	23
3 Основні рішення з реалізації системи	24
3.1 Розробка бази даних	24
3.2 Розробка серверної частини	27
3.3 Розробка клієнтської частини	31
3.4 Висновки за розділом 3	38
4 Експлуатація та експериментальне дослідження програми	38
4.1 Характеристики програми	38
4.2 Запуск застосування	38
4.3 Використання програми	41
4.4 Висновки за розділом 4	477
Висновки	488
Перелік джерел посилання	49
Додаток А Текст програми	51

Додаток Б Слайди презентації	67
------------------------------------	----

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
SMA	– Simple Moving Avarage;
IDE	– Integrated Development Environment,
MT5	– MetaTrader5;
VSC	– Visual Studio Code;
MVC	– Model View Controller;
JS	– Java Script;
DOM	– Document Object Model;
HTTP	– Hypertext Transfer Protocol;
CLI	– Common Language Interface;
AJAX	– Asynchronus Javascript and XML;
БД	– База даних;
СКБД	– Система керування базами даних;

ВСТУП

Біржові операції є привабливими для багатьох людей зі всього світу своїм масштабним потенціалом. Окрім бажання, знань, трейдинг потребує певних витрат. З цієї причини затребуваними стали такі фінансові інструменти, що не передбачають великого капіталовкладення. Але не все так просто. Одночасно такі операції пов'язані з великим ризиком.

Бінарні опціони – на валютному ринку явище нове. І як і будь-яке нове явище, вони викликають у широкого загалу як недовіру, так і щирий інтерес. І перше, і друге – прояви цікавості. Але варто бути обережним і уважно вивчити ринок, перш ніж поспішати задовольнити цю цікавість.

Велика кількість людей втрачає гроші через свою не освіченість та брак досвіду на ринку бінарних опціонів. Для початківців які приходять на ринок бінарних опціонів дуже важливо спочатку здобути досвід та знання на симуляторі бінарного опціону, щоб уникнути великих збитків.

Тому зараз розробка веб-застосунку віртуальної біржі є дуже актуальною задля здобуття необхідних навичок роботи з біржовими інструментами. Метою цієї роботи є створення симулятора бінарного опціону щоб особа яка зацікавлена в темі інвестицій та заробітку грошей на ринку «Форекс» уникла великих збитків та здобула базовий досвід на початковому етапі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття бінарного опціону

Бінарний опціон - це цифровий контракт, предметом якого є передбачення руху ціни активу в певний проміжок часу. Своєю назву цей інструмент отримав через те що є тільки два можливі наслідки угоди — трейдер або отримує прибуток, або нічого не отримує. У будь-якому разі він не може втратити більше грошей, ніж вклав. Це одна з основних переваг опціону. Якщо трейдер зробив аналіз активу, створив угоду та ціна збіглася з його прогнозом, він отримує справжній прибуток. В іншому випадку ставка програє.

Бінарні опціони пропонують трейдерам не складний спосіб заробітку, у якому прибуток отримують через вміння правильно передбачити рух ціни. Трейдеру для цього не треба проводити багато часу на ринку, щоб слідкувати за станом курсу.

Прибутковість у роботі з цими договорами не визначається змінами ціни активу з моменту відкриття угоди і до її закриття. Заробіток залежить від вартості купівлі контракту. Розмір відсоткових значень для різних угод відрізнятиметься. У залежності від ризикованості та ліквідності вона може бути 10–80%, для незахищених транзакцій — понад 100% [1].

Хоча користування бінарними опціонами і дозволяє отримати фіксований прибуток, треба завжди пам'ятати про те, що існує ризик втратити свої інвестиції. Тому людям які тільки почали цим займатись потрібно обирати надійніший вид опціонів, наприклад «Вгору/вниз». Якщо неправильно передбачити рух ціни активу, ставка програє. Тому не треба довіряти ненадійним джерелам торгових сигналів, вони можуть бути збитковими.

1.2 Класифікації бінарних опціонів

Сьогодні брокери пропонують багато різноманітних торгових інструментів. Вони постійно змінюються та модернізуються. Серед опціонів, які використовуються трейдерами найчастіше, слід виділити:

- діапазон;
- один дотик (One touch);
- вгору/вниз (Call/Put);
- спред.

Діапазон. Щоб отримати прибуток працюючи з цим інструментом, треба спрогнозувати ціну активу так, щоб вона за певний проміжок часу не вийшла за меж вказаного інвестором цінового діапазону.

Один дотик (One touch). Сутність цього виду контракту у тому, щоб передбачити рівень, якого торкнеться чи не торкнеться ціна активу під час коливань своєї вартості. Незалежно від напрямку руху ціни, після досягнення рівня вказаного інвестором, він отримає дохід. Якщо працювати з іншим видом опціонів — якщо він не передбачає торкання до певної позначки — треба дочекатися закінчення виконання угоди. Якщо трейдер правильно спрогнозував рівень, трейдер отримає дохід.

Вгору/вниз (Call/Put) - це один з найпопулярніших інструментів. При його використанні важливо спрогнозувати підвищення чи зниження ціни у вказаний час. Попередньо потрібно вибрати одну з дій. Наприклад, якщо інвестор обрав «вгору» і після завершення виконання опціону вартість зросла, він отримає свій прибуток.

Спред - більш ризикований вид опціону. При прогнозі як визначається зниження чи зростання ціни, а й вказується її конкретне значення. Наприклад, може бути прогнозовано, що вартість активу за півгодини

підвищиться на 8 пунктів. Зробити такий прогноз набагато складніше, що визначає високий ризик такого контракту. Але брокери під час укладання таких угод дають вищий відсоток [2].

1.3 Аналіз сучасних брокерів

1.3.1 Огляд брокера «IQ Option»

Повна назва цього бінарного брокера - IQ Option LTD, головний офіс компанії знаходиться на Сент-Вінсент та Гренадіна. Цей брокер бінарних опціонів працює вже 7 років, регулюється зокрема кіпрською комісією CySEC. Зарекомендував себе надійним партнером, який завжди виконує зобов'язання перед трейдерами. Є власний компенсаційний фонд.

Комісії IQ Option - Традиційно для топу найкращих бінарних брокерів 2022 IQ Option не бере комісій за типові дії (тільки за виведення коштів)

Торгові інструменти IQ Option - Бінарний брокер пропонує 53 інструменти для торгівлі опціонами. Інструменти об'єднані за чотирма напрямками: валютні пари, дорогоцінні метали, акції та індекси [3].

Торгова платформа IQ Option - Брокер пропонує власний торговельний термінал, завантажити його можна на офіційному сайті компанії. Термінал мультимовний, тож працювати з ним можуть резиденти різних країн. Незважаючи на колосальну кількість можливостей та налаштувань, інтерфейс платформи є максимально інтуїтивним. Ось її основні характеристики:

- 18 технічних індикаторів, кілька десятків вбудованих ліній (наприклад, лінії Фібоначчі);
- терміни експірації – від 1 хвилини до 30 днів, мультиплікатор до 1:300;

- з платформи є доступ до календарів прибутковості компаній, а також календарів форекс та крипто-подій;
- функція створення власного трейд-портфоліо;
- є версія для десктопу, веб та мобільних гаджетів під iOS та Android, функціонал версій ідентичний;
- абсолютні можливості щодо кастомізації графіків - вибір таймфреймів, типу відображення котирувань, масштабування, історія ринку [4].

Відкриття рахунку у цього бінарного брокера здійснюється через офіційний сайт, де трейдеру необхідно зареєструватися та верифікуватися, після цього він отримує доступ до Особистого кабінету. У 2022 році брокер пропонує два типи рахунків - стандартні та VIP. По обох рахунках мінімальний депозит 10 \$, мінімальна сума на виведення також 10 \$. Інтерфейс брокера можна побачити на рис. 1.1.

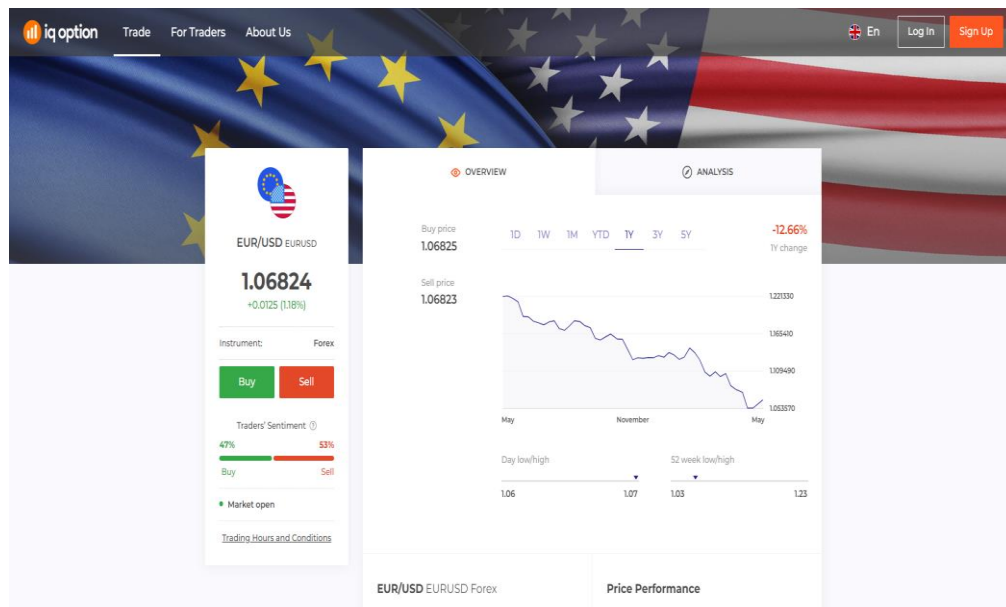


Рисунок 1.1 - Інтерфейс опціону «IQ Option»

1.3.2 Огляд брокера «Pocket Option»

Pocket Option був заснований професійною командою, яка має досвід як трейдингу, так і розвитку Fintech проєктів. Тому і торговий термінал цього брокера власної розробки. На Pocket Option доступні 3 версії торгових платформ:

- Web;
- Desktop;
- Mobile.

На платформі є таймфрейми від 5 секунд до 1 дня. Трейдери можуть користуватися 30 індикаторами технічного аналізу. Також є 10 графічних інструментів. Є функція "Мультиграфік", завдяки чому користувачі можуть налаштувати на одному екрані до 4 графіків. Це дозволяє зручно відстежувати кілька торгових інструментів одночасно.

Брокер Pocket Option пропонує клієнтам 7 типів рахунків. Вони відрізняються за розміром мінімального депозиту та за привілеями для трейдерів. Pocket Option має кілька особливостей. Наприклад, це «кристали» – спеціальні бонуси, які дозволяють скасувати збиткову угоду. Клієнти можуть брати участь у лотереї «кристалів», а також займатися їх майнінгом. Також трейдерам доступний безкоштовний демо-рахунок, торгувати на демо можна навіть без реєстрації.

Цей брокер пропонує вигідні умови поповнення рахунку та виведення коштів. Компанія не стягує комісії за поповнення за транзакції через усі платіжні системи, крім Perfect Money. Всі операції здійснюються через особистий кабінет на сайті. Інтерфейс брокера можна побачити на рис. 1.2.



Рисунок 1.2 - Інтерфейс опціону «Pocket Option»

1.3.3 Огляд брокера «Binary»

Binary є дочірнім підприємством Regen Markets Group, яке засноване понад 20 років тому. Binary – найстаріший бінарний брокер на ринку Форекс, діяльність якого контролюється п'ятьма міжнародними регуляторами.

На 2022 рік Binary проводить понад 1 мільйон транзакцій на день, компанія широко відома найвищою швидкістю проведення всіх типів операцій.

Як і багато кращих бінарні брокери, Binary не бере з трейдера комісії за більшість операцій. Але зверніть увагу – точна величина комісії з угоди в кожному випадку визначається індивідуально, і розрахувати її неможливо.

Брокер пропонує кілька десятків торгових інструментів, включаючи 25 валютних пар, акції, індекси та сировинні товари.

Серед найкращих бінарних брокерів за 2022 рік Бінарі – єдиний брокер, який пропонує одразу три торгові платформи:

- SmartTrader. Відкривається у вкладці браузера, має наочний функціонал і широкий спектр аналітичних інструментів;

– Webtrader. Технічно виступає функціональним аналогом SmartTrader, але орієнтований на трейдерів, що працюють одночасно з великою кількістю інструментів. Гнучкіший у плані налаштувань інтерфейсу, полегшує відстеження відразу за декількома інструментами;

– MetaTrader 5 (MT5). Особливість роботи через цей термінал полягає в тому, що трейдер працює не безпосередньо з MT5, а через допоміжний радник, який завантажується на офіційному сайті брокера. В іншому це стандартна версія популярної платформи;

Що стосується відкриття рахунків Бінарі унікальний тим, що пропонує особливий тип рахунку - Індеси волатильності. Торгівля цьому рахунку складає синтетичному активі, створеному самої компанією. Робота ведеться тільки у вихідні, можливе поповнення в EUR. Також є демонстраційний рахунок із віртуальним депозитом [5].

Серед найкращих бінарних брокерів з топу за 2022 рік Бінарі пропонує найбільшу кількість варіантів для введення коштів на рахунок та виведення з рахунку. Всі операції здійснюються на кілька кліків через Особистий кабінет трейдера на сайті. Інтерфейс брокера можна побачити на рис. 1.3.

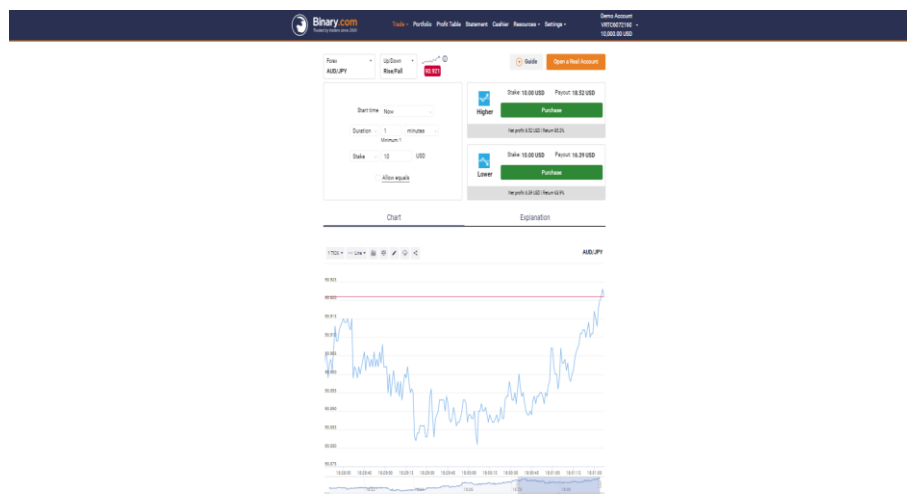


Рисунок 1.3 - Інтерфейс опціону «Binary»

1.4 Постановка завдання роботи

Веб-застосунок має містити в собі:

- можливість робити ставки Long/Short;
- перегляд історії ставок;
- наявність обрання валютної пари;
- присутність графіку що показує ціну валютної пари в реальному часі;
- перегляд історії ціни валютної пари у вигляді графіку;
- можливість передчасно та безризиково закрити ставку;
- поповнення умовного рахунку;
- реєструватися або авторизуватись на сайті.

1.5 Висновки за розділом 1

У першому розділі було розглянуто поняття бінарного опціону його актуальність та надано їх класифікації. Також було проаналізовано трьох бінарних брокерів з їх особливостями та інтерфейсом. Був наданий перелік складових частин які має містити розроблюваний веб-застосунок.

Метою виконання випускної кваліфікаційної роботи бакалавра є програмна реалізація симулятору бінарного опціону використовуючи технології ReactJS + NestJS, який дозволить отримати базовий досвід роботи з валютними парами та аналіз графіку

2 АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 React

React - відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

На даний час React використовують Khan Academy, Netflix, Yahoo, Airbnb, Sony, Atlassian та інші.

Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript.

JSX - розширення мови JavaScript. Його рекомендують використовувати коли треба пояснити React, як повинен виглядати UI. JSX нагадує мову шаблонів, яка наділена силою JavaScript [6], [7].

Однією з ключових особливостей React є те, що він може використовувати в якості мови програмування як JavaScript, так і TypeScript.

Як ми бачимо на з рис. 2.1, бібліотека React є дуже популярною серед фронт-енд розробників.

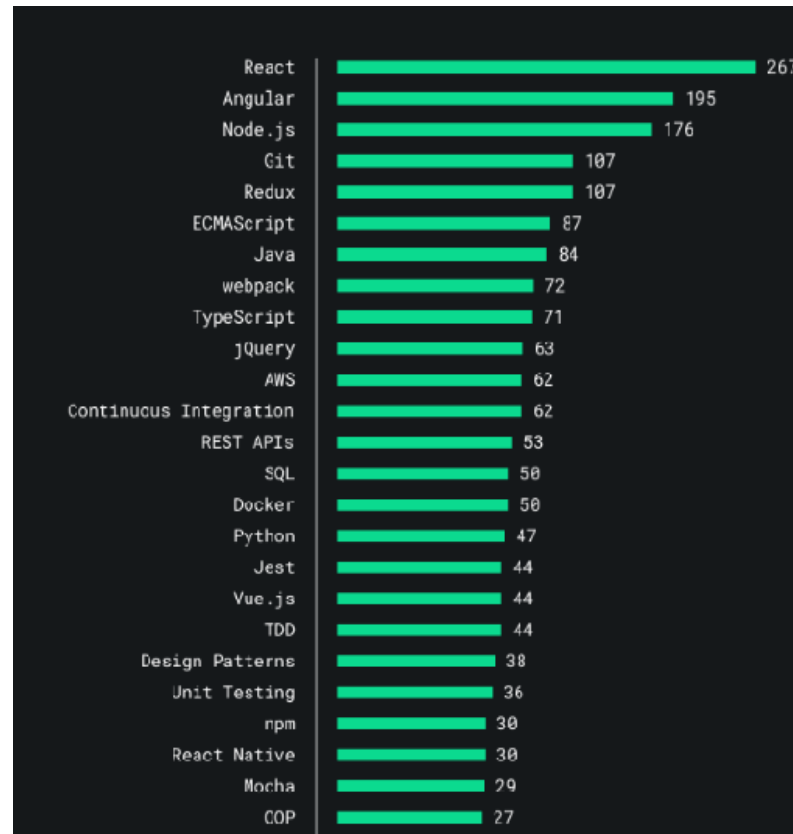


Рисунок 2.1 – Популярність React

Це, робить React найкращим вибором серед доступних фронт-енд технологій.

2.2 TypeScript

TypeScript — мова програмування, розроблена Microsoft у 2012 році і позиціонується як засіб розробки веб-додатків, для розширення можливостей JavaScript.

TypeScript є сумісним з JavaScript і компілюється в останній. Фактично, після компіляції програму на TypeScript можна використовувати у

будь-якому сучасному браузері або використовувати разом із серверною платформою Node.js. Код експериментального компілятора, який трансліює TypeScript JavaScript, поширюється під ліцензією Apache. Його технологія застосовується у громадському репозиторії через обслуговування GitHub.

TypeScript відрізняється від JavaScript наявністю статичного призначення типів, підтримкою використання повноцінних класів (як у традиційних об'єктно-орієнтованих мовах), а також підтримкою підключення модулів, що покликане підвищити швидкість розробки, полегшити читання, рефакторинг та повторне використання коду, він допомагає здійснювати пошук помилок на етапі розробки та компіляції і може, прискорити виконання програм.

Планується, що з повної зворотної сумісності адаптація існуючих додатків до нової мови програмування може відбуватися поетапно, шляхом поступового визначення типів.

TypeScript виник через певні недоліки JavaScript у великих додатках як компанії Microsoft, так і в інших користувачів JavaScript. Проблеми з розробкою складних програм на JavaScript призвели до полегшення розробки компонентів мови [8], [9].

2.3 Redux-saga

Redux-saga це бібліотека яка створена для того щоб спростити та покращити побічні ефекти (асинхронні операції, завантаження даних, або доступ до кешу браузера) зробити легшим у тестуванні та краще впоратися з помилками.

Можна уявити це так, що saga — це як окремий потік у вашому додатку, який відповідає за побічні ефекти. redux-saga — це мідлвар redux. Це означає, що цей потік може запускатися, зупинятися та відмінятися з основного додатку за допомогою звичайних дій redux, він має доступ до

повного стану `redux` додатку та може диспачити дії `redux` [10].

2.3 Node.js

Node.js є серверної технологією, яка заснована на розробленому компанією Google JavaScript-движку V8. Це прекрасно масштабується система, підтримують не програмні потоки або окремі процеси, а асинхронний ввід-висновок, керований подіями. Вона ідеально підходить для веб-додатків, які не виконують складних обчислень, але до яких відбуваються часті звернення. За цілями використання Node схожий з фреймворками Twisted на мові Python і Eve.

Node.js не окрема мова програмування, а платформа для використання JavaScript на стороні сервера. Якщо говорити про мову, то як для фронтенда, так і для бекенд використовується один і той же JavaScript. Різниця тільки в наборі API, які використовують фронтендери і бекендери.

Браузерні JavaScript використовує Web API, які забезпечують доступ до DOM і призначеного для користувача інтерфейсу сторінок і веб-додатків. Серверний JavaScript використовує API, що забезпечують доступ до файлової системи додатків, http-запитів, потокам. Тобто Node.js - це технологія для використання JS на бекенд.

Node.js поступається PHP в популярності. Досить згадати, що на PHP написані найпопулярніші системи управління контентом, включаючи WordPress, Joomla !, Drupal. Тільки на WordPress працює близько третини всіх сайтів в інтернеті.

Інфраструктура PHP швидше перевершує інфраструктуру Node.js. Наприклад, розробники на PHP можуть вибирати між кількома гідними веб-фреймворками.

Python вважається одним з кращих мов для вивчення програмування. Але головна різниця між «Пайтон» і Node.js полягає в сферах застосування.

Python більш універсальний: його використовують як в веб-розробці, так і в Data Science, Machine Learning та інших сферах. Node.js швидше «заточений» під веб-розробку. Завдяки Electron ця технологія також застосовується для розробки десктопних додатків.

Проте інфраструктура Node.js зріла, і у розробників є якісні і зручні інструменти. Наприклад, вище згадувалися пакетний менеджер npm і фреймворк Electron для створення десктопних додатків. Також варто згадати фреймворки Express і Nest.js, призначені для розробки веб-додатків [11].

2.4 NestJS

Nest (NestJS) - це платформа для створення ефективних масштабованих серверних додатків Node.js. Він використовує прогресивний JavaScript, побудований на TypeScript і повністю підтримує його (але при цьому дозволяє розробникам кодувати на чистому JavaScript) і поєднує в собі елементи ООП (об'єктно-орієнтоване програмування), FP (функціональне програмування) і FRP (функціональне реактивне програмування).

Під капотом Nest використовує надійні фреймворки HTTP-серверів, такі як Express (за замовчуванням), і, при бажанні, також може бути налаштований на використання Fastify!

Nest забезпечує рівень абстракції вище цих загальних фреймворків Node.js (Express / Fastify), але також надає їх API безпосередньо розробнику. Це дає розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи [12].

NestJS дозволяє з легкістю вирішувати такі завдання:

- логування і обробка помилок;
- безпека і шифрування;
- реєстрація і авторизація;

- міграції;
- сідування;
- хешування.
- зберігання та обробка даних;
- роутінг;
- локалізація;
- пагінація;
- сесії;
- валідація.

Причини використовувати фреймворк замість нативної мови:

- архітектура додатків “з коробки”;
- легкість у створюванні тестованих та масштабованих додатків високої якості;
- генерація додатків Nest.js за допомогою Nest CLI.

2.5 Висновки за розділом 2

Вибір технологій розробки є дуже важливим. Деякі технології можуть просто не підходити для розробки того чи іншого застосунку. Тому необхідно обирати технології які дозволять найефективнішим шляхом розробити цей додаток.

У процесі роботи була використана бібліотека React тому що це найпопулярніша фронтенд технологія. Були використані найпопулярніші фронтенд бібліотеки такі як `redux-saga` та `react-redux`.

У розробці бек-енд частини був використаний фреймворк NestJS тому що він дуже зручний та підходящий для розробки даного проєкту, та використовує типізовану мову TypeScript.

3 ОСНОВНІ РІШЕННЯ З РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Розробка бази даних

– Проаналізувавши бінарний опціон вгору/вниз (Call/Put) можна побачити що розроблювана БД має містити в собі інформацію про користувача, історію укладених їм контрактів та кількість коштів які він має на своєму рахунку. Тобто ми можемо виділити наступні сутності, які будуть зберігатися у БД:

- access_token;
- user;
- trade;
- wallet.

Таблиця user (табл. 3.1) буде містити в собі головну інформацію про користувача.

Таблиця 3.1 – Атрибути таблиці user

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
id	varchar	not null	45	Первинний ключ
email	varchar	not null	100	
name	varchar	not null	50	
password	varchar	not null	255	
created_at	datetime	-	-	
updated_at	datetime	-	-	
deleted_at	datetime	-	-	

Інформація про історію укладених користувачем контрактів буде міститися у таблиці trade (табл. 3.2).

Таблиця 3.2 – Атрибути таблиці trade

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
id	varchar	not null	45	Первинний ключ
user_id	varchar	not null	100	Зовнішній ключ
wallet_id	varchar	not null	50	Зовнішній ключ
trade_price	float	-	-	
direction	tinyint	-	-	
state	varchar	-	255	
price_on_close	double	-	-	
result	varchar	-	255	
price_on_open	double	-	-	
time	varchar	not null	255	
end_time	varchar	not null	255	
currency	varchar	not null	255	
created_at	datetime	-	-	
updated_at	datetime	-	-	
deleted_at	datetime	-	-	

Таблиця wallet (табл. 3.3) призначена для зберігання кількості ресурсів на рахунку у користувача.

Таблиця 3.3 – Атрибути таблиці wallet

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
id	varchar	not null	255	Первинний ключ
user_id	varchar	not null	255	Зовнішній ключ
amount_of_money	varchar	not null	50	
created_at	datetime	-	-	
updated_at	datetime	-	-	
deleted_at	datetime	-	-	

А таблиця access_token (табл. 3.4) буде містити в собі токени, які потрібні для аутентифікації авторизованого користувача.

Таблиця 3.4 – Атрибути таблиці access_token

Атрибут	Тип даних	Обов'язковість	Припустиме значення	Ключовий атрибут
user_id	varchar	not null	255	Зовнішній ключ
access_token	varchar	not null	255	

Розроблювана БД була спроектована за допомогою СКБД MySQL. Макет спроектованої БД можна побачити на рис. 3.1.

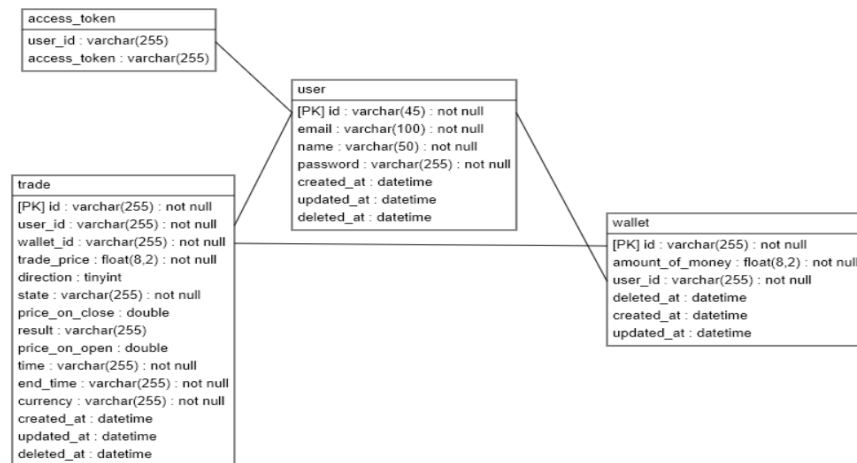


Рисунок 3.1 – Макет створеної БД

3.2 Розробка серверної частини

Шаблон серверної частини проєкту був створений за допомогою NestCLI. Для того щоб зробити структуру проєкту зрозумілою, було створено папку `modules`, яка зберігає в собі основні модулі серверної частини.

Серед основних модулів системи можна виділити наступні:

- `auth`;
- `trade`;
- `wallet`;
- `history`;
- `socket`.

Майже кожен з цих модулів має свої контролери та сервіси. Контролер – частина патерну проектування MVC, яка відповідає за прийняття та обробку даних з фронт-частини системи, а також за надсилання даних до фронт-частини шляхом AJAX запитів з компонентів React. Сервіси потрібні для обробки інформації, що приймає контролер. Вони були

відокремлені від контролера щоб забезпечити читаність програмного коду і для створення легкої для розуміння структури файлів. Сервіси були створені у вигляді класів. Вони містять у собі методи, які оброблюють інформацію, що приймає контролер.

Модуль auth містить у собі сервіси AuthService (табл. 3.5) та RegisterService (табл. 3.6). Сервіс AuthService відповідає за авторизацію користувача, та створення токена, який буде використаний для аутентифікації користувача. Для реєстрації користувача у системі є сервіс RegisterService.

Таблиця 3.5 – Методи сервісу AuthService

Метод	Опис
login	Метод який виконується під час авторизації. Отримує логін та пароль користувача і перевіряє їх правильність та чи існує користувач із зазначеним логіном. У випадку успіху вертає JWT та сутність користувача. В іншому випадку вертає відповідне помилці повідомлення.
logout	Метод, який викликається при виході з програми. Вилучає токен користувача з бази даних.
getAuthUser	Метод, який викликається при перезавантаженні сторінки. Якщо у localStorage зберігається валідний токен то метод відправляє на фронтенд частину дані користувача.
refreshToken	Метод, який викликається коли час валідності існуючого токена вичерпано. Він оновлює токен у БД.

Таблиця 3.6 – Методи сервісу RegisterService

Метод	Опис
registerUser	Метод який виконується під час реєстрації користувача. Він перевіряє пароль на валідність та викликає метод.
createUser	Метод, який викликається у методі registerUser. Він створює запис користувача у БД.

Сервіс модуля trade відповідає за все що пов'язано з купівлею/продажом активу (табл. 3.7). У функції що стартує сервер є змінна об'єкту бібліотеки stop, яка кожну секунду перевіряє усі відкриті угоди. Якщо час угоди закінчився, працює функція яка перевіряє результат угоди, та записує його у БД.

Таблиця 3.7 – Методи сервісу TradeService

Метод	Опис
findByUser	Метод який повертає усі угоди здійснені користувачем. Це потрібно для відображення історії укладених угод та відображення не закінчених, на клієнтській стороні.
closeTrade	Метод, який дозволяє безризиково та передчасно зупинити угоду.
insert	Метод, який працює під час укладення угоди. Він записує її у БД
update	Метод, який оновлює інформацію про угоду у БД.
checkTrades	Метод, який кожну секунду перевіряє чи закінчився час угоди, і з яким результатом ця угода закінчилась.

Сервіс модуля wallet дозволяє створювати гаманець користувача, та поповнювати його (табл. 3.8).

Таблиця 3.8 – Методи сервісу WalletService

Метод	Опис
getWallet	Метод який дозволяє отримати інформацію стосовно залишку коштів на рахунку користувача.
createWallet	Метод, який створює гаманець користувача. Ця функція спрацьовує коли користувач зареєструвався.
refillWallet	Метод, який дозволяє поповнити гаманець користувача.

Сервіс модуля history дозволяє отримати історію ціни активу, потрібну для графіку зі стороннього арі (табл. 3.9).

Таблиця 3.9 – Методи сервісу HistoryService

Метод	Опис
getHistory	Метод який дозволяє отримати історію ціни активу.

Модуль socket не такий як інші. Він не має контролера. Усе тому що цей модуль є адаптером. Адаптер потрібен для того щоб створити websocket сервер у nestjs проєкті. Цей сервер підключається до websocket сервера біржі, яка у режимі реального часу надсилає поточну ціну активу. Під час отримання даних розроблюваний сервер записує їх у змінні та відправляє на клієнтську частину проєкту. Далі значення ціни активу використовується для визначення результату угоди коли час угоди підійшов до свого кінця.

3.3 Розробка клієнтської частини

Шаблон клієнтської частини проєкту був створений за допомогою ReactCLI. Як і у серверній частині, для оптимізації структури проєкту було створено папку `modules`, яка зберігає в собі основні модулі клієнтської частини.

Серед основних модулів системи можна виділити наступні:

- `Auth`;
- `Chart`;
- `MainPage`;
- `Trade`;
- `User`.

Майже усі ці модулі складаються з компонентів, сервісів та сховища даних. React-компоненти – це маленькі, повторно використовувані частини коду, що вертають React-елементи для відображення їх на сторінці. Кожен компонент бере дані зі стейту проєкту та працює з ними. Для зміни стейту у React-компоненті викликаються екшени, які у свою чергу можуть викликати саги для запитів на сервер. Кожна сага використовує певний сервіс для запиту на сервер.

Модуль `Auth` містить у собі компоненти які пов'язані з реєстрацією та авторизацією.

Компонент авторизації є формою, яка перевіряє усі введені дані на валідність (рис. 3.2). При натисканні кнопки `Login` дані з форми відправляються на серверну частину для перевірки на предмет існування користувача та на правильність введеного паролю. Якщо авторизація успішна, сервер надсилає до клієнтської частини сутність користувача, та його токен.

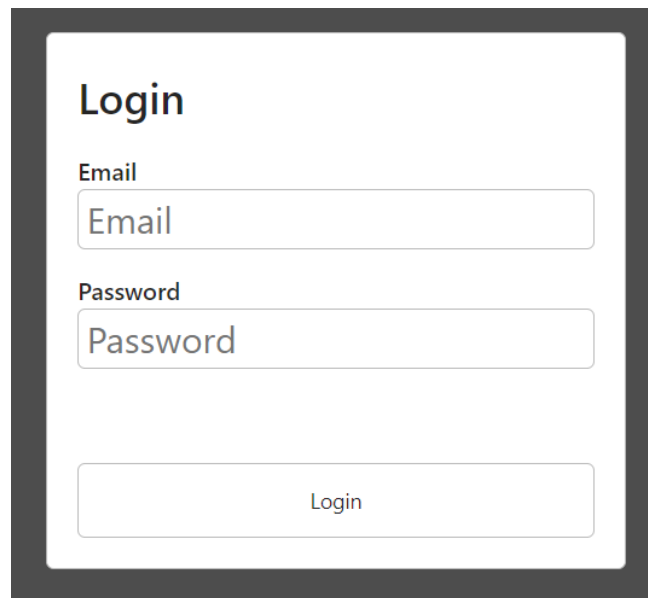
A login form component with a dark gray border. It features a title "Login" at the top. Below the title are two input fields: "Email" and "Password". Each field has a label above it and a placeholder text inside the input box. At the bottom of the form is a "Login" button.

Рисунок 3.2 – Компонент авторизації

Компонент реєстрації є також формою, яка перевіряє введені дані на валідність (рис. 3.3). При натисканні кнопки Register дані з форми відправляються на серверну частину для перевірки. Якщо перевірка пройшла успішно, дані користувача записуються у БД. Після цього він може авторизуватися та користуватися додатком. Цей модуль має сагу AuthSaga (табл. 3.10).

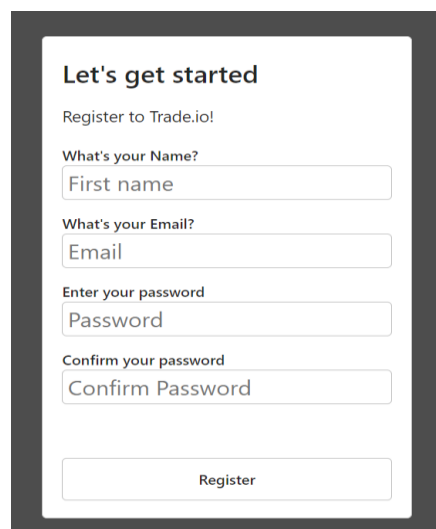
A registration form component with a dark gray border. It features a title "Let's get started" at the top. Below the title is a subtitle "Register to Trade.io!". There are four input fields: "What's your Name?" (with placeholder "First name"), "What's your Email?" (with placeholder "Email"), "Enter your password" (with placeholder "Password"), and "Confirm your password" (with placeholder "Confirm Password"). At the bottom of the form is a "Register" button.

Рисунок 3.3 – Компонент реєстрації

Таблиця 3.10 – Методи саги AuthSaga

Метод	Опис
logining	Сага яка викликає сервіс на бек стороні, що відповідає за вхід користувача у систему. Після отримання відповіді від сервера записує у стейт дані про користувача та статус входу систему(вдала спроба чи помилка). Якщо вхід успішний, завантажує оголошення у стейт.
logout	Сага яка спрацьовує при виході з системи. Вона надсилає запит на сервер який виконує вихід з системи.
registration	Сага яка спрацьовує при реєстрації користувача. Вона викликає сервіс на бек стороні, що відповідає за реєстрацію користувача у системі.

Модуль Chart відповідає за відображення графіку ціни активу (рис. 3.4). У ньому створюється з'єднання з серверною частиною через websocket, для отримання поточної ціни активу. Цей модуль має сагу ChartSaga (табл. 3.11).



Рисунок 3.4 – Компонент графіку ціни активу

Графік ціни активу зроблений за допомогою бібліотеки `lightweight-charts`. Він бере дані історії вартості активу з серверної частини застосунку. На рисунку 3.4 можна побачити три лінії зображені на графіку. Це лінії індикатори SMA. Індикатор SMA показує середній рівень руху ціни за вибраний період часу. Виглядає індикатор як лінія, яка на певному відхиленні слідує за графіком ціни. Насправді індикатор SMA відображає середнє значення цін за останні кілька періодів, що є дуже важливим показником, оскільки за однією з теорій технічного та фундаментального аналізу ціна завжди прагне повернення на попередні середні значення [13].

Індикатор SMA визначається за наступною формулою (3.1):

$$SMA = \text{SUMM} (I (\text{close}), N)/N , \quad (3.1)$$

де SUMM – підсумовування;

$I (\text{close})$ – рівень закриття періоду;

N – кількість періодів.

Таблиця 3.11 – Методи саги ChartSaga

Метод	Опис
<code>gettingChartHistory</code>	Сага яка викликає сервіс на бек стороні, що відповідає за отримання історії ціни активу зі стороннього арі. Після отримання відповіді від сервера записує у стейт історію активу. Цією історією заповнюється графік та розраховуються індикатори SMA.

Модуль `MainPage` містить у собі компонент, який збирає до купи усі компоненти які відображені на головній сторінці. Він не має сервісів та саг.

Також у ньому реалізований компонент який відповідає за обрання поточного активу (рис. 3.5), історія якого відображається на графіку. Коли поточний актив змінюється, на сервер йде запит на отримання історії ціни обраного активу. Отриманні дані записуються у графік.

На клієнтську частину за допомогою сокету в режимі реального часу приходять ціни усіх активів які є у розроблюваному застосунку. Серед них необхідно обрати один актив, ціну якого треба відображати у реальному часі. Саме для цього під час обрання активу також записується його ім'я у стейт, щоб обрати ціну потрібного активу.

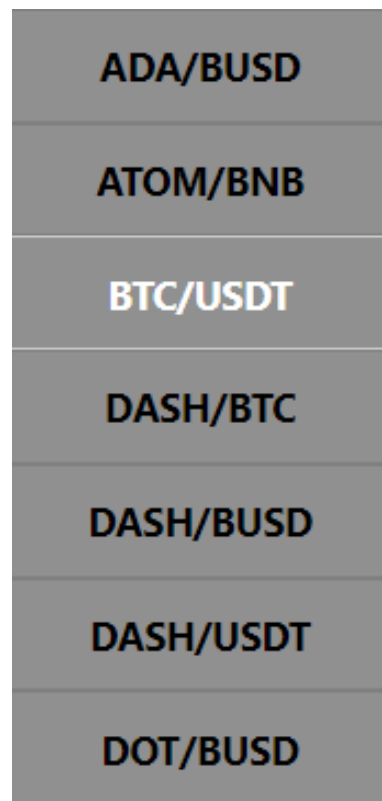


Рисунок 3.5 – Компонент переліку існуючих валютних пар

Модуль Trade має декілька компонентів. Вони пов'язані з інформацією стосовно угод які уклав користувач (рис. 3.6, та рис. 3.7) та з функціоналом їх реалізації (рис. 3.8).

CLOSED							
Currency	Bet	Bet direction	Time	Result	Price on open	Price on close	Closed at
DASH/USDT	10\$	SELL/SHORT	1min.	LOSE	58.27	58.3000	June 3rd 2022, 18:51:46
BTC/USDT	10\$	SELL/SHORT	1min.	WIN	29528.01	29519.1500	June 3rd 2022, 18:50:05
BTC/USDT	10\$	BUY/LONG	1min.	WIN	29504.76	29511.4000	June 3rd 2022, 18:48:30
BTC/USDT	10\$	BUY/LONG	1min.	WIN	29504.76	29511.4000	June 3rd 2022, 18:48:30

Рисунок 3.6 – Компонент історії укладених угод

OPENED						
Currency	Bet	Bet direction	Time	Price	Time left	
BTC/USDT	10\$	BUY/LONG	1min.	29548.67\$	00:00:45	Close
BTC/USDT	10\$	SELL/SHORT	1min.	29548.62\$	00:00:46	Close
BTC/USDT	10\$	BUY/LONG	1min.	29551.62\$	00:00:41	Close
BTC/USDT	10\$	SELL/SHORT	1min.	29551.62\$	00:00:42	Close

Рисунок 3.7 – Компонент поточних угод

Під час завантаження сторінки, на сервер йде запит на отримання даних стосовно історії угод. У разі успішного запиту, сервер повертає масив інформації стосовно укладених угод. Далі ця інформація відображається у відповідних компонентах. Модуль Trade містить у собі сагу TradeSaga (табл. 3.12).

Рисунок 3.8 – Компонент заключення договору

Таблиця 3.12 – Методи саги TradeSaga

Метод	Опис
gettingTrades	Сага яка робить запит до сервісу на отримання закритих та поточних угод. У разі успішного запиту, у стейт записується масив угод які далі розподіляються на закінчені та поточні.
closingTrade	Сага яка виконується коли користувач хоче передчасно та без ризику втратити гроші закрити угоду.
creatingTrade	Сага яка спрацьовує при створенні угоди. Вона відправляє інформацію стосовно створеної угоди на сервер. Той у свою чергу записує дані у БД.

Модуль User не має компонентів, але він має сервіси та саги, які відповідають за отримання з сервера інформації про стан гаманця, та особисті дані. Цей модуль має сагу UserSaga (табл. 3.13).

Таблиця 3.13 – Методи саги UserSaga

Метод	Опис
gettingAuthUser	Сага яка робить запит до сервісу на отримання даних про користувача. Вона викликається при перезавантаженні сторінки, якщо користувач авторизований. Сервер знаходить його інформацію завдяки токєну.
getWallet	Сага яка робить запит на сервер задля отримання інформації про кількість коштів на рахунку користувача.
refillWallet	Сага яка робить запит на сервер, щоб поповнити гаманець.

Також існує компонент який не є частиною жодного модуля. Цей компонент Header. Він може змінюватися в залежності від того чи авторизований користувач (рис 3.9 та рис. 3.10).



Рисунок 3.9 – Header до авторизації



Рисунок 3.10 – Header після авторизації

3.4 Висновки за розділом 3

У процесі розробки даного проекту були створені всі необхідні компоненти та контролери для роботи розроблюваного веб-застосунку. Спроектвана база даних, розроблена клієнтська частин сайту яка забезпечує легку та зрозумілу взаємодію між користувачем та сервісом. Була розроблена серверна частина яка оброблює дані що приходять з клієнтської частини застосунка та зі сторонніх арі.

4 ЕКСПЛУАТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Характеристики програми

У розробленому веб-застосунку було реалізовано усі потрібні складові для коректної роботи сервісу. Основні серед них є наступні:

- реєстрація та авторизація користувачів;
- можливість укладання фінансової угоди ;
- перегляд історії угод;
- перегляд поточних угод;
- можливість безризиково та передчасно закрити угоду;
- поповнення гаманця;
- можливість зміни поточного активу;
- наявність індикатора для торгівлі;

4.2 Запуск застосування

Приклад запуску застосунку буде здійснено використовуючи локальний ресурс комп'ютера. Спершу треба встановити на комп'ютері MySQL server. Після цього за допомогою командного рядка створюємо БД для розробленого застосунку. Це можна зробити за допомогою команди на рис. 4.1.



```
mysql> create database trade_db;
```

Рисунок 4.1 – Команда створення БД

Далі треба заповнити БД таблицями. Для цього необхідно застосувати міграції, які знаходяться у корені папки серверної частини проєкту. Щоб зробити це треба відкрити папку сервера за допомогою IDE VSC, відкрити у ньому командний рядок, і прописати команду `knex migrate:latest` (рис. 4.2).

```
PS C:\Users\Богдан\Desktop\trade-io> knex migrate:latest
```

Рисунок 4.2 – Команда запуску міграцій

Коли БД заповнилась можна запускати серверну частину застосунку (рис. 4.3).

```
PS C:\Users\Богдан\Desktop\trade-io> nest start --watch
```

Рисунок 4.3 – Команда запуску сервера NestJS

```
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RoutesResolver] AppController {/}: +6ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RoutesResolver] RegisterController {/register}: +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/register, POST} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RoutesResolver] AuthController {/auth}: +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/auth/login, POST} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/auth/logout, GET} route +2ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/auth, GET} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/auth/password/:id, PATCH} route +2ms
16:58:28 LOG [RoutesResolver] CustomerController {/wallet}: +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/wallet/:id, GET} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/wallet/:id, PUT} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RoutesResolver] TradeController {/trade}: +2ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/trade/close/:id, GET} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/trade/user/:id, GET} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/trade, POST} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/trade/:id, PATCH} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/trade/:id, DELETE} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RoutesResolver] HistoryController {/history}: +2ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [RouterExplorer] Mapped {/history, GET} route +1ms
[Nest] 14540 - 2022-06-08 16:58:28 LOG [NestApplication] Nest application successfully started +10ms
```

Рисунок 4.4 – Успішно збудований сервер NestJS

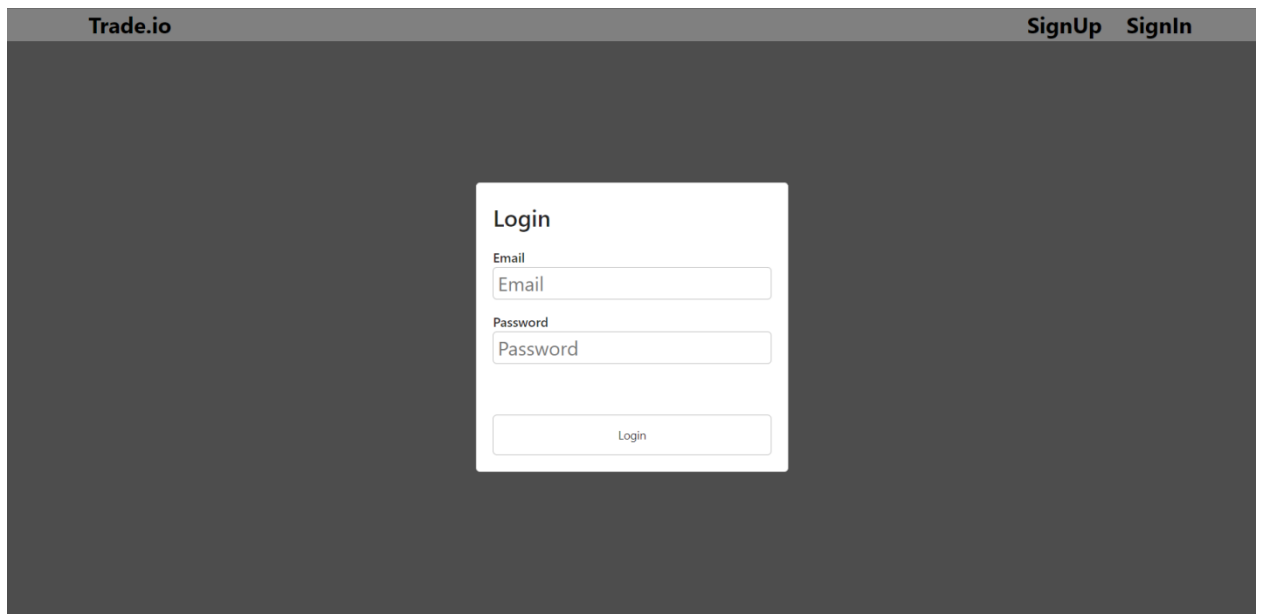
Після успішного старту, контролери сервера доступні за адресом `http://localhost:5000`. Далі необхідно запустити клієнтську частину. Щоб зробити це треба відкрити папку з клієнтською частиною за допомогою IDE VSC. Відкрити у ньому командний рядок, і прописати команду `npm run start` (рис. 4.5).

```
PS C:\Users\Богдан\Desktop\trade-front> npm run start
```

Рисунок 4.5 – Команда запуску React JS застосунку

4.3 Використання програми

Для того щоб почати використовувати веб-застосунок локально, необхідно у пошуковому рядку ввести посилання `http://localhost:3000`. Якщо користувач не авторизований, його перенесе на сторінку авторизації, яка зображена на рис. 4.6.



Trade.io

SignUp SignIn

Login

Email

Password

Login

Рисунок 4.6 – Сторінка авторизації

Для входу у систему необхідно ввести свій логін, пароль та натиснути кнопку Login. У випадку якщо користувач не має свого профілю, він повинен його створити. Для цього треба натиснути на кнопку SignUp у заголовку сторінки. Натискання на цю кнопку перенесе його на сторінку реєстрації, яка зображена на рис. 4.7.

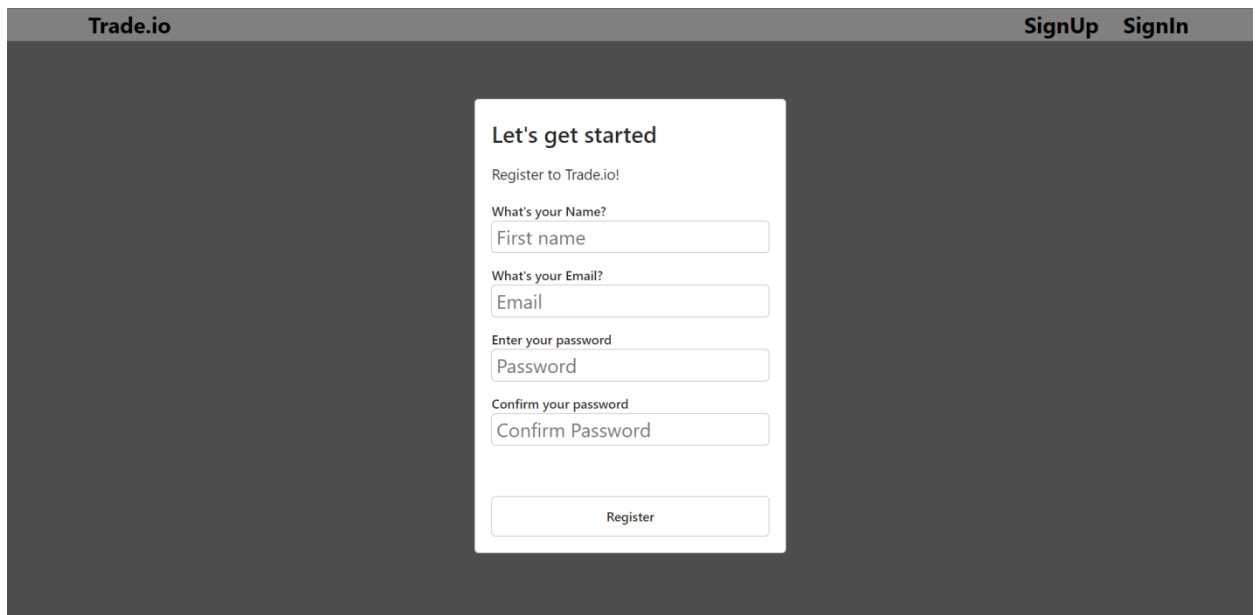


Рисунок 4.7 – Сторінка реєстрації

Для того щоб зареєструватися, треба ввести наступні дані:

- ім'я;
- електронну пошту;
- пароль;
- підтвердження паролю.

У разі введення некоректної інформації будуть відображені відповідні повідомлення (рис. 4.8).

What's your Email?

CCC

Email must be a valid email

Enter your password

...

Password must be at least 8 characters

Confirm your password

..

Passwords don't match

Рисунок 4.8 – Повідомлення про помилки вводу

Коли користувач зареєструвався та авторизувався, він потрапляє на головну сторінку, яка зображена на рис. 4.9.



Рисунок 4.9 – Головна сторінка

На початку роботи з програмою користувачу надається 10000 одиниць віртуальної валюти для початку укладання угод. Якщо гроші на рахунку закінчилися, треба натиснути кнопку Refill. Вона відновлює суму на рахунку користувача.

Перед створенням угоди варто обрати актив для її укладення. Це можна зробити натиснувши на будь-яку валютну пару зі списку у лівій частині вікна. Якщо вона виділена білим кольором, це означає що вона обрана (рис. 4.10).



Рисунок 4.10 – Обраний актив

Щоб створити угоду, необхідно ввести суму угоди та час її тривалості. Це потрібно зробити у формі, яка зображена на рис. 4.11. Існує обмеження на ціну та час. Ціна не має бути вищою за 2000 одиниць, а час не має бути більшим ніж 100 хвилин. У випадку введення некоректних значень, будуть показані відповідні повідомлення (рис. 4.12).

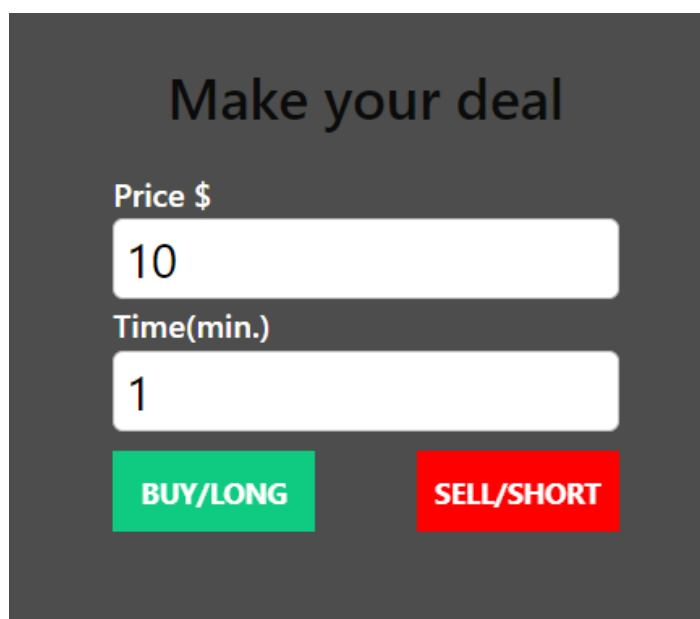


Рисунок 4.11 – Форма створення договору

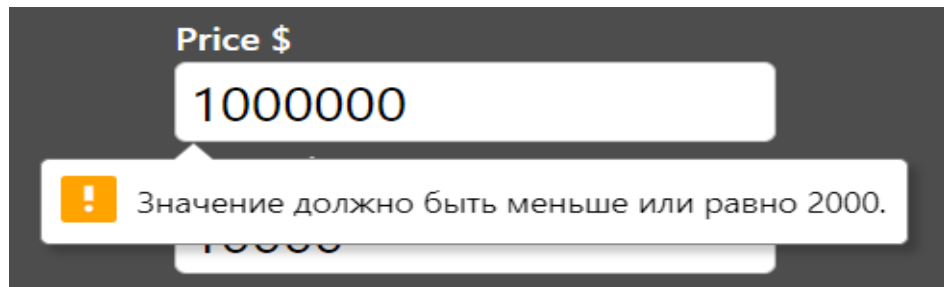


Рисунок 4.12 – Повідомлення про помилку

Якщо користувач вважає що ціна активу буде більшою за поточну, через проміжок часу який він вказав, то він має натиснути зелену кнопку BUY/LONG. Та якщо користувач вважає інакше, він повинен натиснути червону кнопку SELL/SHORT. У випадку якщо користувач був правий, його прибуток складає 85%, тобто якщо він вклав 100 то отримує 185 одиниць умовної валюти. У разі програшу, користувач не отримує нічого.

Після утворення угоди, вона відображається у таблиці Opened (рис. 4.13). Ця таблиця відображає наступні дані:

- currency (відображає назву активу);
- bet (відображає ціну угоди);
- bet direction (напрямок ставки);
- time (відображає тривалість угоди);
- price (відображає ціну активу);
- time left (відображає кількість часу що залишився).

Currency	Bet	Bet direction	Time	Price	Time left	
BTC/USDT	10\$	BUY/LONG	1min.	29548.67\$	00:00:45	Close
BTC/USDT	10\$	SELL/SHORT	1min.	29548.62\$	00:00:46	Close
BTC/USDT	10\$	BUY/LONG	1min.	29551.62\$	00:00:41	Close
BTC/USDT	10\$	SELL/SHORT	1min.	29551.62\$	00:00:42	Close

Рисунок 4.13 – Таблиця Opened

Якщо ваша угода на даний проміжок часу успішна, вона зафарбована зеленим кольором. Якщо ціна активу не співпадає з правилом угоди, то вона фарбується червоним. Кольори укладених угод динамічно змінюються, в залежності від правила угоди. Якщо користувач не впевнений в успішності угоди, він може достроково та безризиково її закрити. Щоб це зробити потрібно натиснути на кнопку Close.

Задля аналізу своїх помилок користувач може дивитися у таблицку з історією укладених угод, яка зображена на рис. 4.14.

Ця таблиця відображає наступні дані:

- currency (відображає назву активу);
- bet (відображає ціну угоди);
- bet direction (напрямок ставки);
- time (відображає тривалість угоди);
- result (результат угоди)
- price on open (ціна на початку угоди);
- price on close (ціна на момент закриття угоди);
- closed at (дата та час закриття угоди).

Currency	Bet	Bet direction	Time	Result	Price on open	Price on close	Closed at
DASH/USDT	10\$	SELL/SHORT	1min.	LOSE	58.27	58.3000	June 3rd 2022, 18:51:46
BTC/USDT	10\$	SELL/SHORT	1min.	WIN	29528.01	29519.1500	June 3rd 2022, 18:50:05
BTC/USDT	10\$	BUY/LONG	1min.	WIN	29504.76	29511.4000	June 3rd 2022, 18:48:30
BTC/USDT	10\$	BUY/LONG	1min.	WIN	29504.76	29511.4000	June 3rd 2022, 18:48:30

Рисунок 4.14 – Таблиця закритих угод

Щоб закінчити роботу з застосунком треба вийти зі свого облікового запису натиснувши кнопку Logout у верхньому правому кутку.

4.4 Висновки за розділом 4

У даному розділі було розглянуто характеристики розроблюваного проекту, та рекомендації щодо його запуску на локальному ресурсі. Також був пояснений процес взаємодії користувача з програмою. Даний бінарний опціон добре захищений від зламу системи. Усі паролі у БД хешуються алгоритмом хешування bcrypt, тому дані користувача надійно захищені.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра був розроблений веб-застосунок симулятор бінарного опціону. Під час розробки була проаналізована предметна область, та існуючі аналоги. Створено технічне завдання для розробки застосування. Перед проектуванням веб-застосунку були проаналізовані сучасні технології. Для розробки сервісу були обрані технології TypeScript, React, Node.js, NestJS. Завдяки аналізу аналогів системи, були визначені та збудовані основні сутності БД та зв'язки між ними. Були розроблені необхідні для роботи з клієнтською частиною контролери.

Веб-застосунок дозволяє оволодіти базовим навичкам роботи з бінарними опціонами. Система дуже легка у використанні, та має ергономічний інтерфейс. Саме завдяки цьому у нового користувача не буде проблем з розумінням роботи даного сервісу.

У результаті розробки симулятора бінарного опціону були удосконалені навички у роботі з мовою програмування TypeScript, бібліотекою React та фреймворком NestJS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Натенберг, Ш. Опціони. Волатильність і оцінка вартості. Стратегії та методи опціонної торгівлі / Ш. Натенберг – М.: Нове видання. 2007. – 540 с.
2. Бінарні опціони [Електрон. ресурс]. – Режим доступу: <https://ffin.ua/ru/blog/articles/investopediia/post/binarni-opcioni-shho-take-ta-yak-pracyuyut>.
3. Огляд IQ Option [Електрон. ресурс]. – Режим доступу: <https://uk.tech-sensor.com/10020992-iq-option-review>.
4. IQ Option – детальний огляд і відгуки про брокера [Електрон. ресурс]. – Режим доступу: <https://moneytotem.com/ukr/iq-option-ohlyad/>.
5. Обзор брокера Binary и отзывы реальных клиентов [Электрон. ресурс]. – Режим доступа: https://rating-market.com/binarnye-opciony/obzor-brokera-binary-i-otzyvy-realnyh-klientov#__Binary.
6. Тиленс, Т. М. React в действии / Т. М. Тиленс – Санкт-Петербург: Питер, 2019. – 368с.
7. Вступ до JSX [Електрон. ресурс]. – Режим доступу: <https://uk.reactjs.org/docs/introducing-jsx.html>.
8. Введение в TypeScript [Электрон. ресурс]. – Режим доступа: <https://metanit.com/web/typescript/1.1.php>
9. Файн, Я. TypeScript быстро / Я. Файн, А. К. Моисеев. – Санкт-Петербург: Питер Пресс , 2021. – 528с.
10. Redux-Saga Basics Concepts [Electronic resource]. – Access mode: <https://redux-saga.js.org/docs/basics/DeclarativeEffects>.
11. Пауэрс, Ш. Изучаем Node.js / Ш. Пауэрс. – Санкт-Петербург: Питер Пресс, 2014. – 400с.

12. NestJS documentation [Electronic resource]. – Access mode: <https://docs.nestjs.com/>.

13. Индикатор SMA [Электрон. ресурс]. – Режим доступа: <https://academyfx.ru/indikator-sma>.

ДОДАТОК А
Текст програми

```

import {
  BadRequestException,
  Body,
  Controller,
  Get,
  Param,
  Patch,
  Post,
  Req,
} from '@nestjs/common';
import { ILoginUser, IPasswordChange } from '../types';
import { loginSchema } from '../../lib/validator/auth/login.validator';
import { YupOptions } from '../../lib/validator/yup.validator';
import { AuthService } from '../services/auth.service';
import { HashService } from '../../lib/services/hash.service';
import { changePasswordSchema } from 'src/lib/validator/auth/change-
password.validator';
import { UserService } from 'src/modules/user/services/user.service';
import { WalletService } from 'src/modules/wallet/services/wallet.service';

@Controller('auth')
export class AuthController {
  constructor(
    private userService: UserService,
    private authService: AuthService,
    private walletService: WalletService,
  ) {}

  @Post('login')
  async login(@Req() request, @Body() body: ILoginUser) {
    const payload = loginSchema.validateSync(body, YupOptions);
    const user = await this.userService.findByEmail(payload.email);
    const wallet = await this.walletService.getWallet(user.id);
    if (!user) {
      return {
        data: {
          message: new BadRequestException('User not found'),
        },
      };
    }
  }

  const check = await HashService.check(body.password, user.password);

  if (!check) {
    return {

```

```

        data: {
            message: new BadRequestException('Password invalid'),
        },
    };
}
const token = await this.authService.login(user.id);

return {
    data: {
        token,
        user,
        wallet,
    },
};
}

@Get('logout')
async logout(@Req() request) {
    try {
        await this.authService.logout(request['access_token']);
        return { message: 'Logout Success' };
    } catch (e) {
        console.log(e);
        return { message: 'Logout error' };
    }
}

@Get('/')
async getAuthUser(@Req() request) {
    try {
        const user = await
this.authService.getAuthUser(request['access_token']);
        const wallet = await this.walletService.getWallet(user.id);
        return {
            data: {
                user,
                wallet
            },
        };
    } catch (e) {
        console.log(e);
        return { message: 'Get Auth User error' };
    }
}

```

```

    @Patch('password/:id')
    async password(@Param('id') id: string, @Body() body:
IPasswordChange) {
        let data: any;

        try {
            const payload = changePasswordSchema.validateSync(body,
YupOptions);

            const user = await this.userService.findById(id);

            const check = await HashService.check(
                this.authService.salt(payload.old_password),
                user.password,
            );

            if (!check) {
                data = { message: 'Password invalid' };
            } else {
                data = { message: 'Success' };
            }

            await this.userService.updateUser(user.id, {
                password: this.authService.hash(body.password),
            });

            return { data };
        } catch (e) {
            data = e;

            return { data };
        }
    }
}

import { HttpException, Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UserModel } from '../user/models/user.model';
import { HashService } from '../lib/services/hash.service';
import { TokenService } from '../lib/services/token.service';
import _ from 'underscore';
import { UserService } from 'src/modules/user/services/user.service';

@Injectable()
export class AuthService {

```

```

constructor(
  protected readonly jwtService: JwtService,
  private tokenService: TokenService,
  private readonly userService: UserService,
) {}

async logout(token) {
  await this.tokenService.destroy(token);
}

async getAuthUser(access_token: string) {
  const userData = await this.tokenService.findByToken(access_token);
  const user: UserModel = await
this.userService.findById(userData.user_id);

  return user;
}

async refreshToken(refresh_token: string) {
  const data = this.jwtService.decode(refresh_token);
  const metadata = data['metadata'] ?? {};
  await this.jwtService.verifyAsync(refresh_token);
  const user: UserModel = await this.userService.findById(data['sub']);

  if (
    await this.tokenService.isTokenBlacklisted(data['access_token'],
user.id)
  ) {
    throw new HttpException(
      {
        statusCode: 400,
        message: ['Token invalid'],
        error: 'Bad Request',
      },
      400,
    );
  }

  await this.tokenService.blacklistToken(user.id, data['access_token']);
  return this.login(user.id, metadata);
}

async isTokenExists(token: string, userId: string): Promise<boolean> {
  return (await this.tokenService.find(token, userId)) !== void 0;
}

```

```

async login(id: string, metadata = {}) {
  const access_token = HashService.rand();

  const payload = {
    sub: id,
    access_verify: HashService.base64_encode(HashService.rand(24)),
    access_token,
    metadata: metadata,
  };

  await this.tokenService.create({
    user_id: id,
    access_token,
  });

  const jwt_access_token = this.jwtService.sign(payload, {
    expiresIn: ['test', 'development'].includes(process.env.NODE_ENV)
      ? '1y'
      : '1h',
  });

  const jwt_refresh_token = this.jwtService.sign(
    {
      sub: id,
      access_token: HashService.rand(),
    },
    {
      expiresIn: '1d',
    },
  );

  return {
    access_token: this.metadata(jwt_access_token, Object.keys(payload)),
    refresh_token: this.metadata(jwt_refresh_token, Object.keys(payload)),
  };
}

metadata(token: string, exclude: string[]) {
  return {
    token,
    ..._.omit(this.jwtService.decode(token), exclude),
  };
}

```

```

hash(password: string) {
  return HashService.hash(this.salt(password));
}

salt(password: string) {
  return HashService.salt(password);
}
}

import { Injectable } from '@nestjs/common';
import { ITrade } from '../types';
import { TradeModel } from '../models/trade.model';
import { WalletModel } from 'src/modules/wallet/models/wallet.model';
import * as moment from 'moment';
import { currPrice } from 'src/modules/common/curPrice';

@Injectable()
export class TradeService {
  static _instance: TradeService;

  API_ROUTE = process.env.REACT_APP_API_HOST;

  static getInstance(): TradeService {
    if (!TradeService._instance) {
      TradeService._instance = new TradeService();
    }
    return TradeService._instance;
  }

  async findById(id: string): Promise<TradeModel> {
    return TradeModel.query().select().where('id', id).first();
  }

  async findByUser(id: string): Promise<TradeModel[]> {
    return TradeModel.query()
      .select()
      .where('user_id', id)
      .orderBy('end_time', 'DESC');
  }

  async closeTrade(id: string): Promise<any> {
    const trade = await TradeModel.query().select().where('id', id).first();
    const amount = await WalletModel.query()
      .select()
      .where('user_id', trade.user_id)
      .first();
  }

```

```

        amount.amount_of_money = amount.amount_of_money +
trade.trade_price;
        await WalletModel.query().update(amount).where('user_id',
trade.user_id);

const coeff = Boolean(trade.direction) ? 1 : -1;
let curr = trade.currency.replace('/', '');
if ((currPrice[curr] - trade.price_on_open) * coeff > 0) {
    await TradeModel.query()
        .update({
            state: 'CLOSED',
            price_on_close: currPrice[curr],
            result: 'WIN',
            end_time: moment().format(),
        })
        .where('id', trade.id);
} else {
    await TradeModel.query()
        .update({
            state: 'CLOSED',
            price_on_close: currPrice[curr],
            result: 'LOSE',
            end_time: moment().format(),
        })
        .where('id', trade.id);
}

const newTrades = await TradeModel.query()
    .select()
    .where('user_id', trade.user_id)
    .orderBy('end_time', 'DESC');
const currWallet = await WalletModel.query().select().where('user_id',
trade.user_id).first();
return {trades: newTrades, wallet: currWallet };
}

async insert(data: TradeModel): Promise<TradeModel> {
    const trade = await TradeModel.query().insert(data);
    const amount = await WalletModel.query()
        .select()
        .where('user_id', data.user_id)
        .first();
    amount.amount_of_money = amount.amount_of_money -
data.trade_price;

```

```

        await WalletModel.query().update(amount).where('user_id',
data.user_id);
        return TradeModel.query().select().where('id', trade.id).first();
    }

    async update(id: string, data: ITrade): Promise<TradeModel> {
        await TradeModel.query().update(data).where('id', id);
        return TradeModel.query().select().where('id', id).first();
    }

    async delete(id) {
        return TradeModel.query().delete().where('id', id);
    }

    async chaeckTrades() {
        const openedTrades = await TradeModel.query()
            .select()
            .where('state', 'OPENED')
            .where('end_time', '<=', moment().format());
        openedTrades.map(async (item) => {
            let curr = item.currency.replace('/', '');
            var currentAmount = await WalletModel.query()
                .select()
                .where('user_id', item.user_id);
            const coeff = Boolean(item.direction) ? 1 : -1;

            if ((currPrice[curr] - item.price_on_open) * coeff > 0) {
                await TradeModel.query()
                    .update({
                        state: 'CLOSED',
                        price_on_close: currPrice[curr],
                        result: 'WIN',
                    })
                    .where('id', item.id);
                await WalletModel.query()
                    .update({
                        amount_of_money:
                            currentAmount[0].amount_of_money +
                            (item.trade_price / 100) * 82 +
                            item.trade_price,
                    })
                    .where('user_id', item.user_id);
            } else {
                await TradeModel.query()
                    .update({

```

```

        state: 'CLOSED',
        price_on_close: currPrice[curr],
        result: 'LOSE',
    })
    .where('id', item.id);
    }
    });
}
}

import {
    Body,
    Controller,
    Delete,
    Get,
    Param,
    Patch,
    Post,
    Put,
    Query,
    Req,
    UploadedFile,
    UseInterceptors,
} from '@nestjs/common';
import { ITrade } from '../types';
import { TradeService } from '../services/trade.service';

@Controller('trade')
export class TradeController {
    constructor(protected tradeService: TradeService) {}

    @Get('/close/:id')
    async closeTrade(@Param('id') id: string) {
        const data = await this.tradeService.closeTrade(id);
        return { data };
    }

    @Get('/user/:id')
    async byUser(@Param('id') id: string) {
        const data = await this.tradeService.findByUser(id);
        return { data };
    }

    @Post('/')
    async add(@Body() trade: any) {

```

```

    const data = await this.tradeService.insert(trade);
    return { data };
  }

  @Patch('/:id')
  async update(@Param('id') id: string, @Body() data: any) {
    const dataobj = JSON.parse(data.payload);
    const trade = await this.tradeService.update(id, dataobj);
    return { trade };
  }

  @Delete('/:id')
  async delete(@Param('id') id: string) {
    const data = await this.tradeService.delete(id);
    return { data };
  }
}

import { HttpException, Injectable } from '@nestjs/common';
import { UserModel } from '../models/user.model';

@Injectable()
export class UserService {
  async findById(id): Promise<UserModel> {
    return UserModel.query().findById(id);
  }

  async findByEmail(email): Promise<UserModel> {
    return UserModel.query().where('email', email).limit(1).first();
  }

  async updateUser(id: string, userData: any) {
    await UserModel.query().update(userData).where(`id`, id);

    const data = await UserModel.query().select().where('id', id);

    return { data };
  }

  async deleteUser(userID: string): Promise<number> {
    return UserModel.query().delete().where('id', userID);
  }
}

import {
  MessageBody,

```

```

    OnGatewayConnection,
    SubscribeMessage,
    WebSocketGateway,
    WebSocketServer,
  } from '@nestjs/websockets';
import { Socket } from 'dgram';
import { IsJsonString } from '../lib/services/isJSON';
import socketAddEvents from '../lib/services/socketEvents';
import { currPrice } from '../common/curPrice';
import { Server } from 'socket.io';
import { currencies, getConnectionsPool } from
'src/lib/enums/connectionPool';

@WebSocketGateway(7000)
export class SocketGateway implements OnGatewayConnection {
  @WebSocketServer() server: Server;

  handleConnection(client: Socket, ...args: any[]) {
    console.log('connected');

    for (let i = 0; i < currencies.length; i++) {
      socketAddEvents(getConnectionsPool()[currencies[i]], this.server);
    }
    client.on('message', (event: Buffer) => {
      if (IsJsonString(event.toString())) {
        const tradeObj = JSON.parse(event.toString());
        const currency = tradeObj.currency.replace('/', '');

        setTimeout(() => {
          tradeObj.direction == 'UP'
            ? currPrice[currency] >= tradeObj.close
            ? client.send('win')
            : client.send('lose')
            : currPrice[currency] <= tradeObj.close
            ? client.send('win')
            : client.send('lose');

          console.log('Sended');
        }, tradeObj.time * 60000);
      }
    });
  }
  @SubscribeMessage('message')
  handleMessage(@MessageBody() message: any): void {}
}

```

```

import * as WebSocket from 'ws';
import { WebSocketAdapter, INestApplicationContext } from
'@nestjs/common';
import { MessageMappingProperties } from '@nestjs/websockets';
import { Observable, fromEvent, EMPTY } from 'rxjs';
import { mergeMap, filter } from 'rxjs/operators';
import { IsJsonString } from 'src/lib/services/isJSON';

export class WsAdapter implements WebSocketAdapter {
  constructor(private app: INestApplicationContext) {}

  create(port: number, options: any = {}): any {
    return new WebSocket.Server({ port, ...options });
  }

  bindClientConnect(server, callback: Function) {
    server.on('connection', callback);
  }

  bindMessageHandlers(
    client: WebSocket,
    handlers: MessageMappingProperties[],
    process: (data: any) => Observable<any>,
  ) {
    fromEvent(client, 'message')
      .pipe(
        mergeMap((data) => this.bindMessageHandler(data, handlers,
process)),
        filter((result) => result),
      )
      .subscribe((response) => client.send(JSON.stringify(response)));
  }

  bindMessageHandler(
    buffer,
    handlers: MessageMappingProperties[],
    process: (data: any) => Observable<any>,
  ): Observable<any> {
    const message = IsJsonString(buffer.data)
      ? JSON.parse(buffer.data)
      : buffer.data;

    const messageHandler = handlers.find(
      (handler) => handler.message === message.event,
    );
  }

```

```

    if (!messageHandler) {
      return EMPTY;
    }
    return process(messageHandler.callback(message.data));
  }

```

```

close(server) {
  server.close();
}

```

```

import { Injectable } from '@nestjs/common';
import axios from 'axios';

```

```

@Injectable()
export class HistoryService {
  async getHistory(currfsym: string, currtsym: string) {
    try {
      let array = await axios.get(
        `https://min-
api.cryptocompare.com/data/v2/histominute?fsym=${currfsym}&tsym=${currtsy
m}&limit=2000&api_key=d3fc97332d795f68cd956f3d80ba98bb5b3d5599293859
0dbd8a1c865d00b1f5`,
      );

      return array.data.Data.Data.map((item) => {
        return {
          time: item.time,
          open: parseFloat(item.open),
          high: parseFloat(item.high),
          low: parseFloat(item.low),
          close: parseFloat(item.close),
          volumefrom: parseFloat(item.volumefrom),
        };
      });
    }
    catch (e) {
      return e
    }
  }
}

import {
  Controller,
  Get,

```

```

    Param,
    Query,
    Req,
  } from '@nestjs/common';
import { HistoryService } from '../service/history.service';

@Controller('history')
export class HistoryController {
  constructor(
    private historyService: HistoryService,
  ) {}

  @Get('/')
  async getHistory(@Query('currency') currency: string) {
    try {
      const currArray = currency.split('/');
      const history = await this.historyService.getHistory(currArray[0],
currArray[1]);
      return {
        data: {
          history,
        },
      };
    } catch (e) {
      console.log(e);
      return { message: 'Get history error' };
    }
  }
}

import { HttpException, Injectable } from '@nestjs/common';
import { WalletModel } from '../models/wallet.model';

@Injectable()
export class WalletService {

  async getWallet(id: string): Promise<WalletModel> {
    return WalletModel.query().where('user_id', id).first();
  }

  async createWallet(id: string): Promise<WalletModel> {
    const wallet = {
      user_id : id,
      amount_of_money: 10000,
    }
  }
}

```

```

    return WalletModel.query().insert(wallet);
  }

  async refillWallet(id: string): Promise<WalletModel> {
    await WalletModel.query().update({amount_of_money:
10000}).where('id', id);
    return WalletModel.query().select().where('id', id).first();
  }
}

import * as dotenv from 'dotenv';
dotenv.config();

import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { NestExpressApplication } from '@nestjs/platform-express';
import * as path from 'path';
import { CronJob } from 'cron';
import { WsAdapter } from './modules/sockets/socket.adapter';
import { TradeService } from './modules/trade/services/trade.service';

async function bootstrap() {
  const app = await
NestFactory.create<NestExpressApplication>(AppModule, {
  cors: true,
});
  const tradeService = TradeService.getInstance();
  app.useStaticAssets(path.join(__dirname, '../static'));
  var job = new CronJob('* * * * *',
function(){tradeService.chaeckTrades()}, null, true);
  app.useWebSocketAdapter(new WsAdapter(app));
  job.start();
  await app.listen(process.env.PORT || 5000);
}

bootstrap();

```

ДОДАТОК Б
Слайди презентації

Програмна реалізація симулятора бінарного опціону реального часу

Виконав студент групи КНТ-128
Лоцман Б.А.

1

Рисунок Б.1 – Слайд №1.

Мета роботи

- ❖ Об'єкт дослідження - симулятор бінарного опціону реального часу.
- ❖ Предмет дослідження - програмне забезпечення симулятора бінарного опціону.
- ❖ Мета роботи - створення вебзастосунку симулятора бінарного опціону вгору/вниз (Call/Put), для ознайомлення з роботою бінарних опціонів.

2

Рисунок Б.2 – Слайд №2.

Технології використані для розробки

- ❖ Серверна частина – фреймворк NestJS на базі мови JavaScript.
- ❖ Клієнтська частина – бібліотека React на базі мови JavaScript.
- ❖ Середовище розробки – IDE Visual Studio Code.

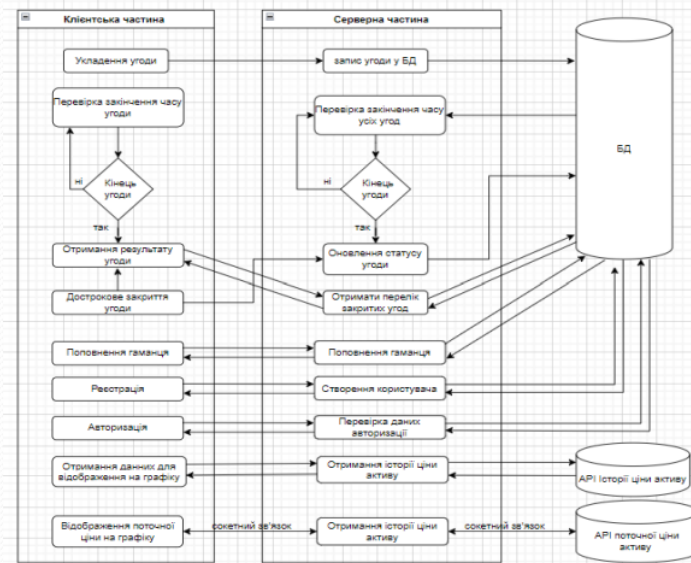
Таблиця порівняння фронт-енд технологій

Фронт-енд технологія	Angular	React.js	Vue.js
Рівень складності	високий	середній	низький
Мова програмування	TypeScript	JavaScript, TypeScript	JavaScript
Розмір файлу	500кб	100кб	80кб
Середні результати Google Trend	50	90	45
Upwork вакансії	726	816	257
Рік запуску	2010	2013	2014

3

Рисунок Б.3 – Слайд №3.

Алгоритм роботи розробленої програми



4

Рисунок Б.4 – Слайд №4

Структура серверної частини проєкту

Основні модулі:

auth – авторизація та реєстрація;

history – завантаження історії ціни активу;

trade – створення та відображення угод;

user – редагування профілю користувача;

wallet – відображення та поповнення гаманця.



5

Рисунок Б.5 – Слайд №5

Структура клієнтської частини проєкту

Основні модулі:

Auth – авторизація та реєстрація;

Chart – графік ціни активу;

Trade – реалізація та відображення угод;

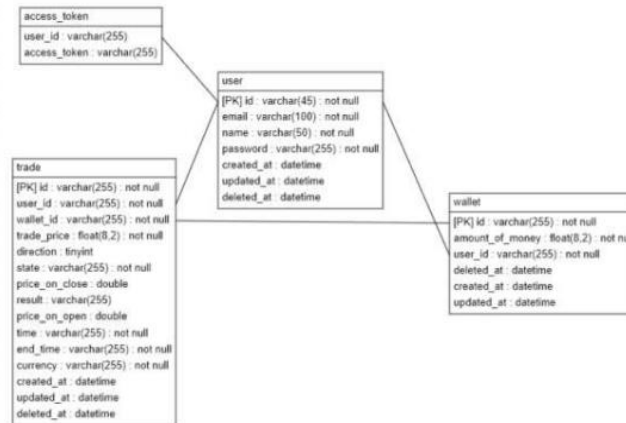
User – інформація про користувача.



6

Рисунок Б.6 – Слайд №6

Розроблена схема бази даних



Програма працює з реляційною базою даних MySQL

7

Рисунок Б.7 – Слайд №7

Функціонал розробленої програми

- ❖ реєстрація та авторизація користувачів;
- ❖ можливість укладання фінансової угоди ;
- ❖ перегляд історії угод;
- ❖ перегляд поточних угод;
- ❖ можливість безризиково та передчасно закрити угоду;
- ❖ поповнення гаманця;
- ❖ можливість зміни поточного активу;
- ❖ наявність індикатора для торгівлі;

8

Рисунок Б.8 – Слайд №8

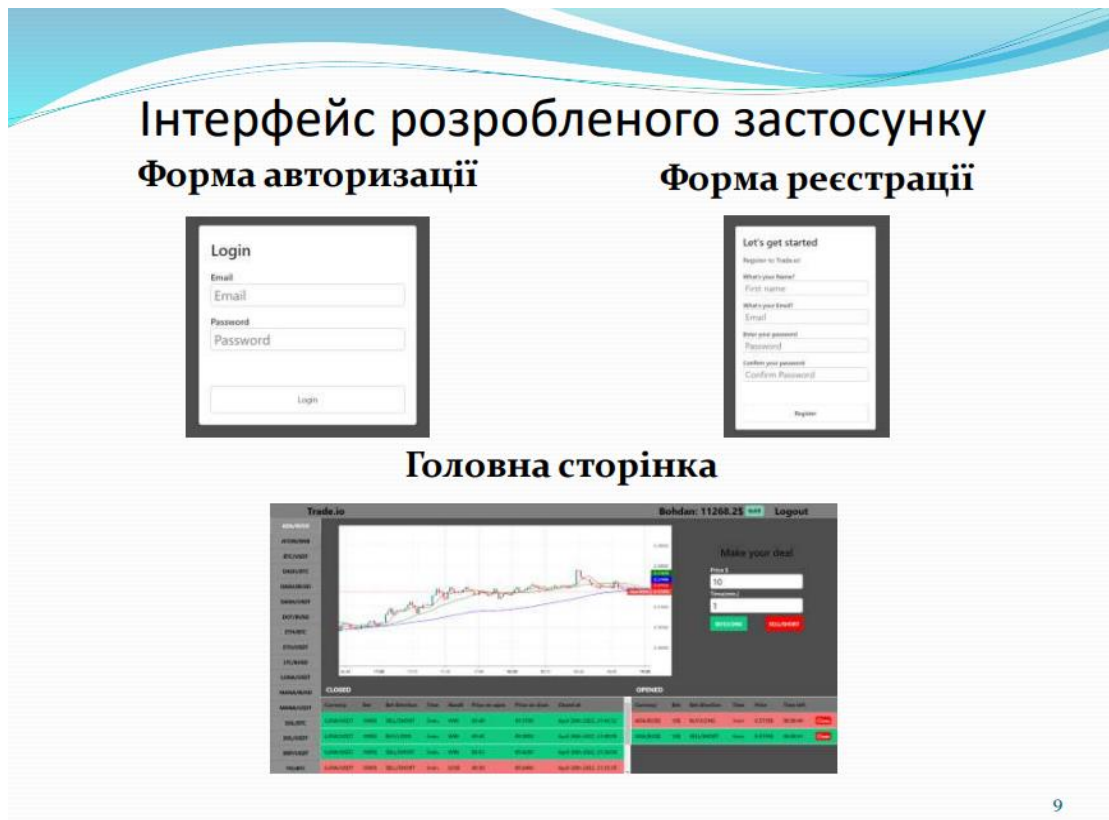


Рисунок Б.9 – Слайд №9

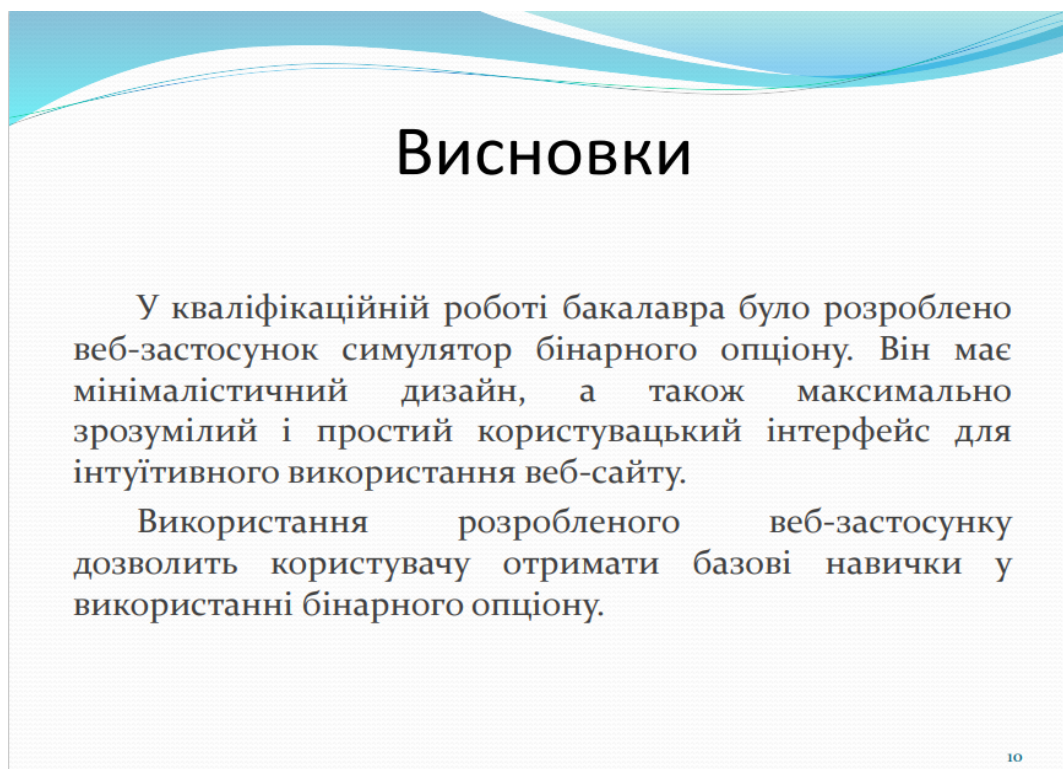


Рисунок Б.10 – Слайд №10