КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Програмна система з торгівлі на біржі криптовалют»

Виконав ______________________________
(Підпис)
Мурашко Сергій Сергійович
(прізвище, ім'я, по батькові)

Керівник професор Сайко Володимир Григорович
(прізвище, ім'я, по батькові)
____
(Резолюція «До захисту»)

Унікальність тексту 94%
Автор______Мурашко С.С.
(Підпис) (Прізвище, ініціали)

Попередній захист:

_____***23.05.2022***______________________________________
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри _______ Плескач В.Л.
(Підпис) (Прізвище, ініціали)
(Дата)

Київ – 2022 року

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	заява
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	виконано
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	виконано
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2022	виконано
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2022	виконано
9.	Подання роботи у першому варіанті	28.04.2022	виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**23.05.2022**	виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	виконано
14.	Захист кваліфікаційної роботи бакалавра	22.06.2022	виконано

Здобувач вищої освіти ____ ____

(підпис)

Керівник ____ ____

(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	2
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	20
2	15
3	14
Висновки	2
Перелік використаних джерел	2
Додатки	17

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП XXXX 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розробн.</td><td>Мурашко С.С.</td><td></td><td></td><td rowspan="4">Відомість дипломної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Сайко В.Г.</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Базилюк А.М.</td><td></td><td></td><td colspan="2" rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td><td></td></tr>
</table>

АНОТАЦІЯ

Дипломна робота: 62 с., 32 рис., 1 табл., 23 джерел.

Ця дипломна робота присвячена розробці CLI застосунку для торгівлі на криптовалютній біржі Binance.

Метою кваліфікаційної роботи є розроблення CLI застосунку для швидкого доступу та використання на криптовалютній біржі.

Завдання дослідження:

- Дослідити сучасні методології розроблення застосунків.
- Проаналізувати архітектурні рішення, технології та інструменти для реалізації застосунку.
- Реалізувати застосунок для торгівлі на криптовалютній біржі Binance.

Об'єктом дослідження є процеси ведення торгівлі криптовалютою.

Предметом дослідження є програмно-технічні засади щодо побудови CLI застосунку торгівлі криптовалютами для підвищення ефективності доступу та використання на криптовалютній біржі.

Методи дослідження. Теорія управління для дослідження теоретичних аспектів побудови застосунків, емпіричний аналіз і синтез систем, що застосовувався при вивченні прикладів сучасних методів побудови застосунків торгівлі криптовалютами, метод порівняння.

Ключові слова: криптовалюта, Binance, торгівля, застосунок.

ABSTRACT

Thesis: 62 pages, 32 figures, 1 table, 23 sources.

This thesis is devoted to the development of CLI programs for trading and cryptocurrency exchange Binance.

The purpose of the thesis is to develop a CLI application for quick access and use on the cryptocurrency exchange. To achieve this goal you need to solve the following **tasks**:

- Explore modern application development methodologies.
- Analyze architectural solutions, technologies and tools for application implementation.
- Implement an application for trading on the Binance cryptocurrency exchange.

Object of study.

Processes of cryptocurrency trading.

Subject of study.

Software and technical principles for building a CLI application for cryptocurrency trading to improve access and use on the cryptocurrency exchange.

Research methods.

Management theory for the research of theoretical aspects of application building, empirical analysis and synthesis of systems used in the study of examples of modern methods of building applications of cryptocurrency trading, the method of comparison.

Key words: cryptocurrency, binance, trading, cli application.

Зміст

Зміст 6

Перелік умовних позначень та скорочень 7

Вступ 8

Розділ 1. Аналіз методологій та підходів розробки застосунків 10

1.1 CLI за стосунок 10

1.2 Аналіз методологій розробки 10

1.3 Існуючі рішення 25

1.4 Аналіз програмного модуля 28

1.5 Постановка задачі 28

Висновок до першого розділу 29

Розділ 2. Проектування архітектури CLI застосунку 30

2.1 Архітектура програмного модуля 30

2.2 Узагальнена архітектура клієнт-сервер 36

2.3 Опис класів та методів 38

2.4 Вибір середовища та інструментів реалізації 43

Висновок до другого розділу 44

Розділ 3. Технологічні особливості реалізації програмного модуля 45

3.1 Налаштування Binance API 45

3.2 Реалізація за стосунку 47

3.3 Інструктивний матеріал з експлуатації 50

Висновок 59

Список використаних джерел 61

Перелік умовних позначень та скорочень

HTTP – HyperText Transfer Protocol.

HTML – HyperText Markup Language.

API – Application Programming Interface.

Баг – помилка у програмі.

CLI – Command Line Interface.

Беклог – список робочих задач у порядку їх важливості.

Бібліотека – набір класів, методів тощо, які пов’язані між собою вирішенням проблеми однієї тематики.

JSON – JavaScript Object Notation, текстовий формат даних, який представляє собою об’єкт мови JavaScript.

Вступ

Починаючи з далекого 2009 року, коли лише був згенеровані перші біткоіни і вони коштували менше одного долару, почався досить повільний, але односпрямований рух та розвиток цілої індустрії, яка за майже 13 років переросла в багатомільярдну сферу з цілими екосистемами.

Актуальність теми зумовлена тим, що на протязі всіх років криптовалютна індустрія продовжує зростати дуже швидкими темпами, з'являються нові проекти та рішення, які використовують технологію блокчейну та швидко розвиваються. На цьому фоні досвідчені трейдери можуть досить швидко заробляти, а для цього необхідні швидкодіючі застосунки.

Метою кваліфікаційної роботи є розроблення CLI застосунку для швидкого доступу та використання на криптовалютній біржі.

Завдання дослідження:

- Дослідити сучасні методології розроблення застосунків.
- Проаналізувати архітектурні рішення, технології та інструменти для реалізації застосунку.
- Реалізувати застосунок для торгівлі на криптовалютній біржі Binance.

Об'єктом дослідження є процеси ведення торгівлі криптовалютою.

Предметом дослідження є програмно-технічні засади щодо побудови CLI застосунку торгівлі криптовалютами для підвищення ефективності доступу та використання на криптовалютній біржі.

Методи дослідження. Теорія управління для дослідження теоретичних аспектів побудови застосунків, емпіричний аналіз і синтез систем, що застосовувався при вивченні прикладів сучасних методів побудови застосунків торгівлі криптовалютами, метод порівняння.

Ключові слова: криптовалюта, Binance, торгівля, застосунок

Практичне значення одержаних результатів полягає у тому, що розроблений застосунок може стати корисним, швидким та зручним рішенням для торгівлі на криптовалютній біржі.

Структура кваліфікаційної роботи:

Кваліфікаційна робота складається із вступу, трьох розділів та списку джерел.

Розділ 1. Аналіз методологій та підходів розробки застосунків

1.1 CLI застосунок

CLI застосунок обробляє команди для комп’ютерної програми у вигляді рядків тексту. Програма, яка обробляє інтерфейс, називається інтерпретатором командного рядка або процесором командного рядка. Операційні системи реалізують інтерфейс командного рядка в оболонці для інтерактивного доступу до функцій або служб операційної системи.

Сьогодні багато користувачів покладаються на графічні інтерфейси користувача та взаємодію, керовану меню. Однак деякі завдання програмування та обслуговування можуть не мати графічного інтерфейсу користувача і все одно можуть використовувати командний рядок. Інтерфейси командного рядка зазвичай реалізуються в термінальних пристроях, які також можуть використовувати екранно-орієнтований текстовий інтерфейс користувача, який використовує курсорну адресацію для розміщення символів на дисплеї. Програми з інтерфейсом командного рядка часто легше автоматизувати за допомогою скриптів. Багато програмних систем реалізують інтерфейс командного рядка для адміністрування та роботи. До них належать середовища програмування та утиліти.

1.2 Аналіз методологій розробки

1.2.1 Каскадна модель (Waterfall)

Каскадна модель одна із найстаріших, вона містить у собі 5 основних послідовних етапів, кожен з яких не може початися, поки минулий завершиться. Етапи каскадної моделі зображені на рисунку 1.1.

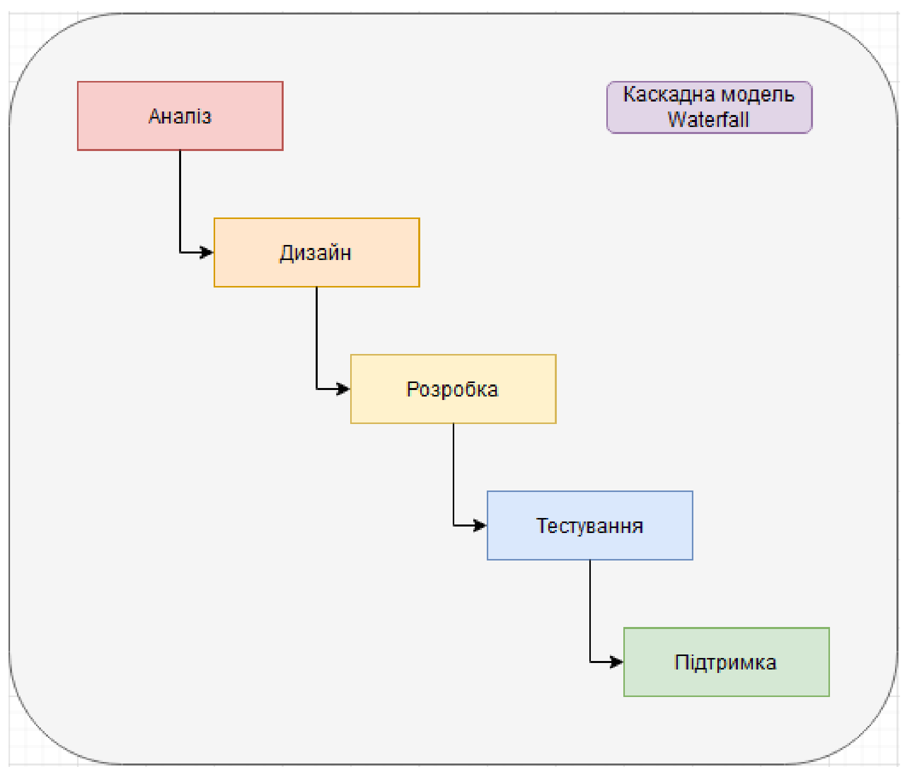

Рисунок 1.1 – Каскадна модель

До етапів розробки відносяться:

- Аналіз - перший і найважливіший етап розробки, під час якого збираються всі вимоги клієнта, розраховуються плани, цілі, бюджети, хід роботи, ризики. Після цього визначаються технічні завдання та інструкції з виконання, від яких не можна відступати. Цей етап є основним і найдорожчим, тому що якщо ви допустили помилку на етапі постановки завдання - змінити її неможливо і ця помилка збережеться.

- Дизайн – коли завдання готове починається розробка архітектури проекту, його прототипу, платформи, а також назначити ролі та відповідальних.

- Розробка – розробка програмного забезпечення, тобто коду продукту, логіки, інтеграції тощо безпосередньо за технічним завданням.

- Тестування – перевірка тестувальниками відповідності розробленого продукту технічним завданням, виправлення помилок та іншої документації. Якщо помилок у випуску продукту занадто багато - весь процес доведеться починати спочатку.

- Підтримка – після введення продукту в експлуатацію, при необхідності, його потрібно модифікувати, замінювати функціональності, додавати, або взагалі видаляти.

До переваг, за виконанням усіх умов, можна віднести:

- Чіткі вимоги до продуктів, середовища та інструментів.

- Фіксована дата завершення проекту

- Фіксований бюджет

До недоліків можна віднести:

- Розробити дійсно якісний продукт відповідно до цієї моделі можливо лише за наявності чітких і продуманих вимог і методів реалізації, техніки та середовища.

- Важко налаштувати вимоги за потреби.

- Тестування відбувається лише на спеціально виділеному етапі, а не впродовж усієї розробки.

- Закритість від замовника. Продукт, який можна побачити буде доступний лише після його завершення.

Незважаючи на всі недоліки сьогодні, ця модель іноді використовується для невеликих проектів, але вважається застарілою.

1.2.2 V-модель

V-модель, як і каскадна модель рухається етап за етапом, проте вона є покращеною версією, адже тестування та контроль якості продукту відбувається на кожному з них. Також для кожного з етапів тестування розробляється свій окремий тест-план, тобто при тестуванні певного рівня для нього обирається певна стратегія, прописуються очікувані результати та критерії початку та завершення тестування.

В основі концепції моделі лежить поняття верифікації та валідації:

- Верифікація – це процес перевірки документів, дизайну, архітектури, коду, тест-кейсів, специфікацій, коду статично без його запуску. Для того, щоб зрозуміти, що це саме верифікація – варто ставити питання “Чи робимо ми продукт правильно?”. Таким чином під час верифікації ми перевіряємо чи робимо ми дійсно якісний продукт, проте це не гарантує, що він буде доцільним та корисним. Також варто зазначити, що верифікація завжди йде до валідації.

- Валідація – це також процес перевірки, проте не зі сторони вимог, а зі сторони користувача, чи відповідає розроблене програмне забезпечення вимогам та очікуванням користувача. Питання, яке варто задати: “Чи робимо ми правильний продукт”. Для валідації необхідний запуск програми.

У V-моделі кожному етапу розробки відповідає свій етап тестування. Процес розробки знаходиться зліва та йде донизу, саме тут відбувається верифікація, а тестування у свою чергу знаходиться зправа та рухається вгору, тут відбувається валідація. Відповідність етапів розробки та тестування відображаються горизонтальними лініями.

Етапи V-моделі зображені на рисунку 1.2.

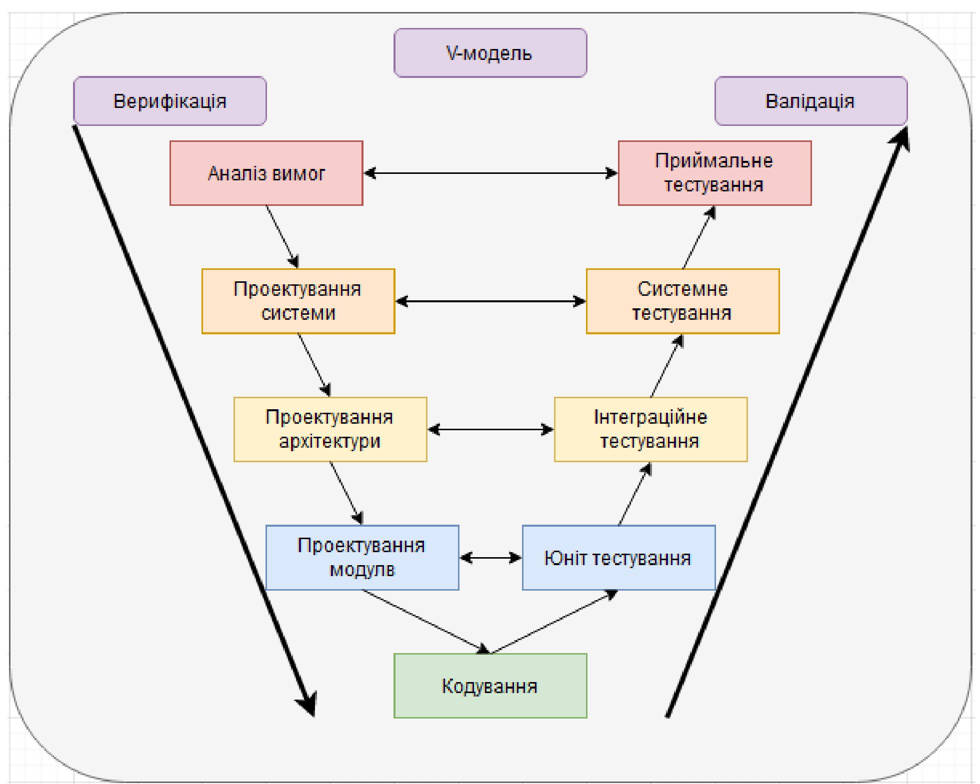

Рисунок 1.2 – V-модель

До переваг можна віднести:

- Модель передбачає планування, верифікацію та валідацію продукту на різних етапах його розробки.

- Верифікація та валідація відбувається на основі всіх отриманих зовнішніх та внутрішніх даних.

- Досить легко дослідити процес розробки, адже завершення кожного етапу є контрольною точкою, що наближає проект до завершення.

- Строге дотримання етапів дозволяє легко розраховувати кінцеві терміни для розробки застосунку та його бюджету.

- Модель проста для управління часом впродовж етапів.

До недоліків можна віднести:

- Модель не передбачає роботу з паралельними подіями.
- Відсутність готовності до динамічних змін на різних етапах життєвого циклу.
- Не має етапу оцінки ризиків.
- Результат можна побачити на різних етапах розробки, проте певна частина продукту буде закритою для замовника до кінця розробки.
- При тестуванні та знаходженні невідповідностей дуже важко не вплинути на графік проекту.

1.2.3 Спіральна модель

Головною особливістю ітеративної моделі є спеціальна увага до ризиків, які можуть вплинути на організацію життєвого циклу та продукт. До таких ризиків зазвичай відносять:

- Дефіцит спеціалістів.
- Нереалістичні очікування щодо бюджету проекту та часу його виконання.
- Реалізація непотрібної функціональності.
- Реалізація неправильного інтерфейсу.
- Непотрібна надзвичайна оптимізація.
- Постійний потік змін.
- Нестача інформації щодо зовнішніх компонентів.
- Недостатня продуктивність системи.

- Невідповідність кваліфікації спеціалістів або великий розрив між спеціалістів.

Модель поділяється на 4 основні сектори:

- Постановка задач, альтернатив та обмежень.
- Оцінка альтернатив, ризиків та способів їх уникнення.
- Розробка та верифікація частини продукту.
- Планування наступних ітерацій.

Вони направлені на відповідні процеси, які у подальшій розробці можуть трохи видозмінюватись. Модель зображена на рисунку 1.3.

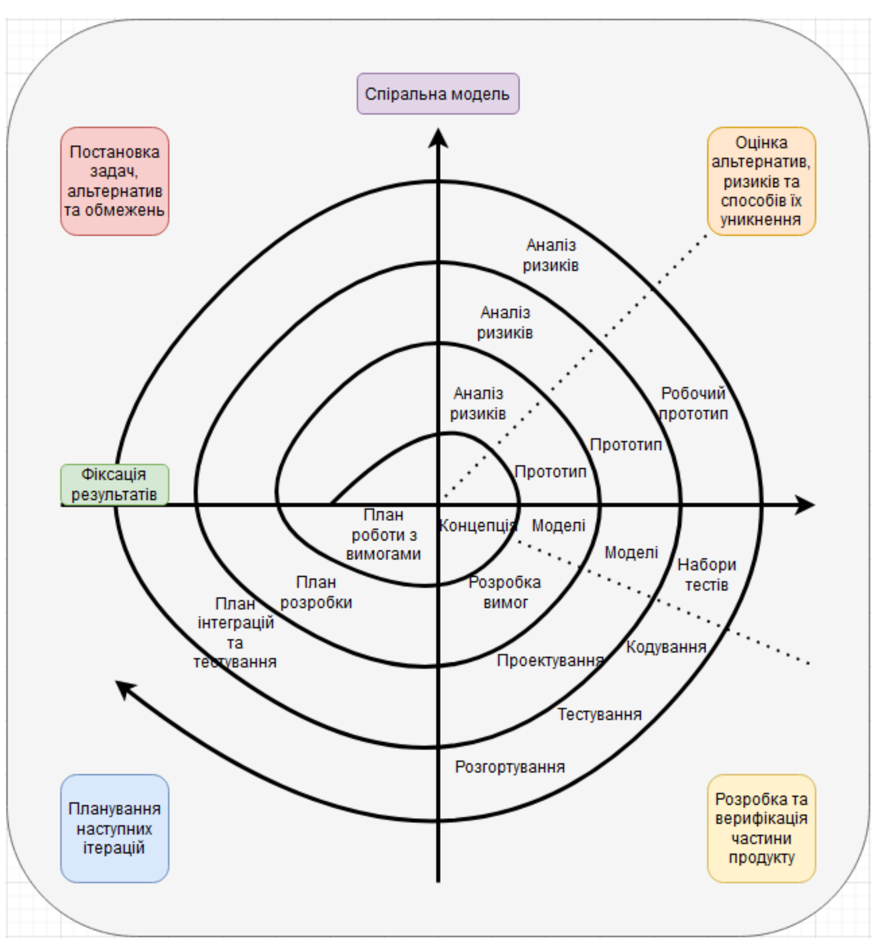

Рисунок 1.3 – Спіральна модель

До переваг можна віднести:

- Глибокий аналіз ризиків на кожній ітерації.
- Гарна документація процесу розробки.
- Можливість додавання нової функціональності навіть на пізніх етапах.
- Розробка прототипів застосунку на ранніх етапах.

До недоліків можна віднести:

- Потребує досить великих коштів і з кожною наступною ітерацією стає більш громіздким.
- Для оцінки ризиків потрібні висококваліфіковані спеціалісти.
- Успіх реалізації напряму залежить від стадії аналізу ризиків.
- Не підходить для малих процесів.

1.2.4 Scrum

Scrum – це мінімальний набір заходів, артефактів, на яких будується процес розробки, що дозволяє за фіксовані невеликі проміжки часу надати користувачу працюючу частину продукту, для яких був визначений найбільший пріоритет.

Спринт - це період часу, достатній для виконання запланованого набору дій, спрямованих на створення інкременту продукту. Час строго встановлений. Спринт триває від 1 до 4 тижнів. Чим коротший спринт, тим гнучкіший процес розробки, чим частіше випускається випуск, тим швидше отримують відгуки споживачів, тим менше часу витрачається на те, щоб рухатися в неправильному напрямку, але багато часу витрачається на мітинги, ретроспективи та планування спринтів.

Можливості реалізації наступного спринту визначаються командою на зустрічі з планування спринту на початку спринту. Відносні оцінки та покер планування найчастіше використовуються для оцінки майбутньої спринтерської роботи. Наприкінці спринту Scrum-команда зустрічається із клієнтом на нараді з огляду спринту та показує йому інкремент продукту, який вона успішно створила за час спринту. Метою спринтерського огляду є отримання зворотного зв’язку з клієнтом щодо того, на чому необхідно звернути увагу в майбутньому та яким має бути наступний інкремент продукту.

До артефактів Scrum відносять:

- Діаграма згоряння задач - Діаграма, що демонструє кількість зробленої роботи, що залишилася, щодо часу на розробку проекту називається діаграмою згоряння. Цю діаграму потрібно оновлювати щодня, щоб показувати в режимі реального часу зміни та витрати на проектну роботу. Діаграма згоряння робіт показує, скільки завдань було виконано в поточному спринті і скільки завдань ще потрібно виконати.

- Журнал побажань проекту містить перелік функціональних вимог, упорядкованих за важливістю та відповідно в порядку виконання. Елементи в цьому журналі називаються історіями користувачів або елементами. Беклог проекту відкрито для редагування для всіх учасників процесу Scrum. Власник продукту SCRUM відповідає за виконання проекту.

- Журнал побажань спринту - містить функції, вибрані власником продукту з елементів, що залишилися в беклозі проекту. Усі функції поділені на завдання, кожне з яких оцінюється. На зустрічі з планування спринту, використовуючи методологію планування покеру, команда оцінює обсяг роботи, яку необхідно виконати, щоб завершити спринт.

- Scrum-дошка – це інструмент для публічного представлення поточного стану Scrum-команди. Scrum-дошка складається з трьох колонок: "зробити", "у процесі", "зроблено". На Scrum-дошці розміщується весь об'єм спринт беклогу,

який команда обрала на плануванні спринту для подальшої реалізації. Зазвичай картки бізнес-завдань розташовуються на дошці зверху вниз у порядку зменшення . Хорошою практикою є декомпозиція бізнес-завдань на конкретні роботи, які необхідно виконати команді, для реалізації бізнес-завдання. Бізнес-завдання та конкретні робочі картки переміщуються від стовпця до стовпця залежно від того, як команда виконує їх для виконання та виконання. Для забезпечення видимості прогресу роботи команди «зменшення роботи» днями відображається на діаграмі згорання задач. У міру виконання робіт за результатами нарад команда фізично переміщає стікери з колонки в колонку.

- Ціль спринту – це короткий опис ділової мети спринту. Як артефакт, цілі спринту допомагають команді приймати зважені бізнес-рішення. Цей артефакт необхідний команді проекту для прийняття власних рішень у визначенні альтернативних шляхів вирішення бізнес-проблеми.

- Інкремент продукту - це готові до використання частини продукту, які необхідно реалізувати до завершення спринту. Під час огляду спринта команда представляє інкременти продукту скрам-майстру, власнику продукту, клієнту та іншим зацікавленим сторонам, отримує від них зворотний зв’язок і вирішує подальший напрямок розвитку продукту.

- Історія користувача - бажана бізнес-функція, додана до відставання проекту, часто називається історією користувача. Найчастіше історія має таку структуру: «Як користувач <тип користувача> я хочу зробити <дія>, щоб отримати <результат>». Така структура зручна тим, що зрозуміла як розробникам, і замовникам.

- Критерії готовності - критерії, що визначають рівень готовності елемента з беклогу.

- Швидкість команди - загальна кількість очок, набраних Scrum командою за попередній спринт. Ця метрика допомагає команді зрозуміти, скільки історій вона може зробити за один спринт.

Також у Scrum виділяють основні наради, які відбуваються регулярно і завдяки яким відбувається контроль розробки та мінімізація інших нарад, які не входять до Scrum, а саме:

- Зустріч з планування спринту – проводиться на початку кожного спринту. На цій зустрічі заплановано повний обсяг роботи, яку необхідно виконати під час спринту. План повинен бути результатом роботи всіх членів Scrum команди. Тривалість зустрічі визначається тривалістю спринту, командним досвідом тощо, але не повинна перевищувати 8 годин. Scrum майстер контролює виконання цього терміну. Зустріч із планування спринту відповідає на такі запитання: Яка мета цього спринту або що можна зробити для цього спринту? Як саме досягаються цілі спринту, щоб отримати додатковий продукт? Першу проблему потрібно вирішувати спільно. Власник продукту обговорює, що необхідно досягти для кожного спринту, включаючи відставання беклогу продукту, попередні інкременти продукту тощо, додаючи нову історію користувача до беклогу та видаляючи завершену історію користувача. Команди розробників намагаються передбачити, які функції вони можуть розробити під час спринту. З другим питанням працює лише команда розробників. Оскільки мету спринту вже визначено, команді розробників необхідно зрозуміти, як її можна досягти. Вони вирішують, яким чином реалізовуватимуть плановану функціональність, щоб отримати новий готовий інкремент продукту за спринт.

- Щоденні Scrum-зустрічі – ці зустрічі проводяться командою розробників за можливою участю Власника продукту та керівника SCRUM щодня в одному місці та в той самий час тривалістю не більше 15 хвилин. Під час цих зустрічей команда розробників планує роботу на день. Такі зустрічі оптимізують командну роботу та продуктивність, переглядаючи роботу, виконану після останньої зустрічі, та плануючи майбутню роботу. Ці щоденні зустрічі допомагають зрозуміти, як робота просувається до цілей спринту. Вони підвищують ймовірність того, що команда розробників впорається з поставленими цілями. Під час зустрічі команда розробників повинна зрозуміти,

як вони повинні працювати разом, щоб досягти цілей спринту та реалізувати заплановані кроки. Структура таких нарад визначається командою розробників, при необхідності, і коли це доцільно структура нарад може бути змінена, при цьому основна увага має приділятися досягненню мети спринту, однак є обов'язкові правила:

1. Зустріч має тривати рівно 15 хвилин.
2. Зустріч проводиться кожен робочий день.
3. Пропуск зустрічі не дозволяється;

Обов'язкові питання, що розглядаються на зустрічі:

1. Що я зробив учора, що допомогло команді розробників досягти мети спринту?
2. Що я зроблю сьогодні, щоб допомогти команді розробників досягти мети спринту?
3. Я бачу будь-які перешкоди, що заважають мені чи команді розробників досягти мети спринту?

- Огляд підсумків спринту - проводиться в кінці спринту, щоб перевірити інкремент продукту та, при необхідності, адаптувати беклог. Scrum команда та всі зацікавлені сторони взяли участь в огляді результатів спринту. Це неформальна зустріч та презентація інкременту призначена для отримання зворотного зв'язку та розвитку співпраці. Огляд підсумків спринту включає такі елементи:

1. Власник продукту показує, які елементи беклогу було виконано та ні.
2. Команда розробників обговорює, що пройшло добре, які виникли труднощі та як їх вирішили.
3. Команда розробників демонструє роботу, яку вона виконала за спринт і який інкремент продукту отримала.

4. Власник продукту обговорює беклог у поточному стані. Також він прогнозує можливі цілі та терміни виконання на підставі досягнутого результату.

5. Відбувається обговорення того, що найцінніше робити далі на підставі аналізу ринку та/або потенційної аудиторії.

6. Огляд термінів, бюджету, можливостей та ринку для доповнення беклогу.

7. Результатом є оновлений белог, який визначає цілі для наступних спринтів. Беклог можна відкоригувати загалом задоволення нових можливостей.

- Ретроспективна наради – зустріч, метою якої є розробка рекомендацій щодо вдосконалення процесу реалізації проекту. В результаті ретроспективного аналізу минулих спринтів формуються плани покращення процесу реалізації проекту на наступний спринт. Зустріч проводиться після ознайомлення з результатами спринту перед плануванням наступного спринту і триває не більше 3 годин. Основними цілями наради є:

1. Аналіз останнього завершеного спринту у частині задіяних людей, процесів та інструментів.

2. Виявлення основних ефективних рішень щодо покращення процесу реалізації проекту прийнятих для минулого спринту та пошук шляхів їх подальшого вдосконалення.

3. Формування плану запровадження, вдосконалення процесу реалізації проекту Scrum командою.

4. Scrum майстер заохочує команду вносити пропозиції щодо підвищення ефективності процесу розробки. Під час кожної ретроспективи спринту команда має шукати та пропонувати шляхи та засоби покращення робочого процесу.

До кінця ретроспективного аналізу спринту, команда має визначити пропозиції щодо покращення для впровадження у наступному спринті. Хоча такі пропозиції можуть бути реалізовані будь-коли, ретроспектива спринту дає

можливість зосередитися на аналізі взаємодій команди та її адаптації для поточних умов.

Основні процеси та артефакти зображені на рисунку 1.4.

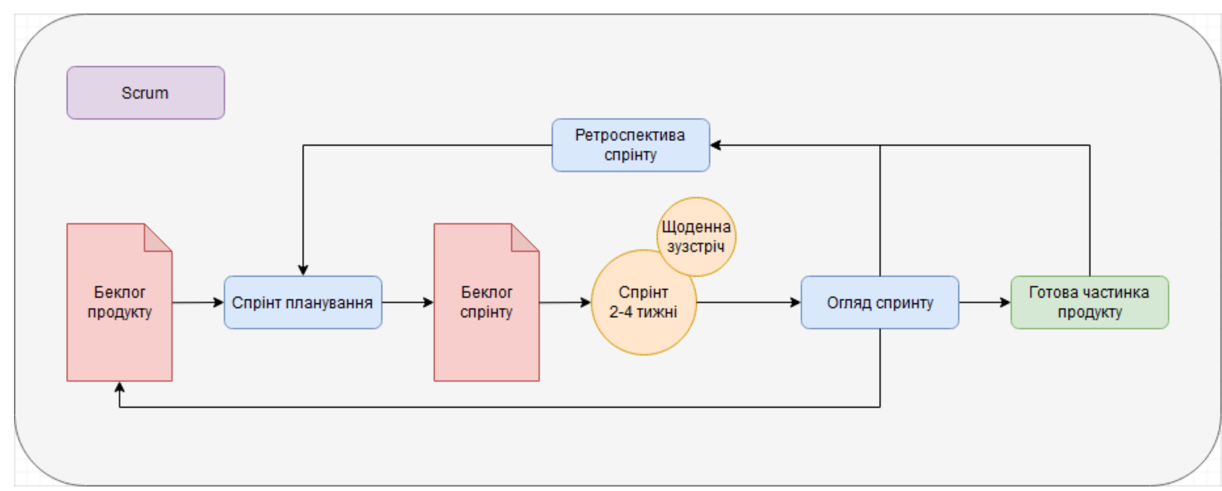

Рисунок 1.4 – Scrum

До переваг можна віднести:

- Команда працює короткими етапами, на кожному з яких визначає цілі та шляхи їх досягнення, що прискорює процес роботи;
- Команда працює над різними завданнями проекту одночасно, що дозволяє швидше досягти бажаної мети;
- Великі завдання поділяють на дрібні, тому внести коригування у процесі роботи набагато простіше, ніж у каскадному підході;
- Скорочується час на пошук помилок та пояснення проблем;
- Мінімізація фінансових ризиків завдяки оперативній реакції на зміни та усунення помилок;
- Кожен член команди чітко знає своє завдання, отже, підвищується рівень відповідальності за роботу;

- Є відкритий обмін інформацією, що робить процес роботи максимально прозорим;
- Підтримання високого рівня мотивації у команді завдяки щоденній видимості досягнень.

До недоліків можна віднести:

- Успіх проекту багато в чому залежить від скрам-майстра, кваліфікації команди та їхньої прихильності до своєї справи;
- Далеко не завжди можна адаптувати метод скрам під сферу діяльності, оскільки є проекти, що вимагають виключно планового підходу у роботі;
- Вимагає регулярної комунікації із замовником, що часом гальмує процес через неможливість отримання зворотного зв'язку.
- Складність впровадження у масштабних та складних проектах, тому що більше підходить для малих та середніх.

1.3 Існуючі рішення

Загалом, додатків CLI, що реалізують торговий термінал, не так багато через велику кількість функцій, які користувачеві важко встановити в консоль, і зазвичай компанії розробляють лише комп’ютерні та веб-версії. Оскільки консольні програми не широко використовуються серед звичайних користувачів, а не розробників чи людей, тісно пов’язаних з технологіями, їх майже не існує, хоча якщо робити вузьконаправлену програму, вона може досить швидко обробляти команди та надсилати запити, що є головним фактором. Веб, або настільний додаток має певну кількість додаткових процесів, які необхідно обробляти перед кожним запитом, а також, якщо у застосунку присутні візуальні елементи, які мають оновлюватись у реальному часі – це призведе до зниження швидкості роботи та затримок, а при надзвичайних навантаженнях застосунок взагалі може зламатись. Також досить значною перевагою CLI застосунків над іншими – це можливість розгортання на серверах без візуальної оболонки, якщо це необхідно.

1.3.1 Binance CLI

Незважаючи на непопулярність рішення розробки CLI застосунків, компанія Binance створила свій офіційний інтерфейс з усіма головними функціями для торгівлі на спотовому ринку, а також отримання інформації щодо активів. Проте більший акцент зроблений саме на API, де вже існує 3 версії, які покривають усі функціональності біржі, а CLI застосунок вийшов досить обмеженим та рідко оновлюється.

Головною проблемою цього застосунку, як було зазначено раніше, є досить обмежена функціональність, а також велика кількість обов’язкових атрибутів, кожен з яких необхідно прописувати окремо у відповідний параметр, що є досить проблемною частиною. Єдина перевага такого методу – це гнучкість написання

команд, через не обов'язковість порядку введення аргументів. Приклад команд застосунку зображені на рисунку 1.4.

buy

```
# place a limit buy order on BNBUSDT with price=350 and qty=0.05
binance-cli buy -s BNBUSDT -t LIMIT -q 0.05 -p 350 -f GTC
```

sell

```
# place a limit sell order on BNBUSDT with price=500 and qty=0.03
binance-cli sell -s bnbusdt -t limit -q 0.03 -p 500 -f GTC
```

Рисунок 1.4 – Приклад використання офіційного Binance CLI застосунку

1.3.2 GeastWood BinanceAPICLI

На відміну від офіційного застосунку, розробка GeastWood має гарний дизайн та візуалізацію даних (1.5), а також інтерактивне меню, де можна вибрати опції після введення команди. Серед операцій, які необхідні для вільної торгівлі присутні усі, проте розроблені також у занадто перегруженій манері.

```
feiliu in binance on master [!] is v0.1.7 via v9.10.1
$ atcb symbol --stat  --openOrder

Pair/Interval        LINKBTC     MDABTC      REQBTC      ENJBTC      BCPTBTC     TNBBTC      WINGSBTC    INSBTC      ONTBTC      STORMBTC
10 min (1*10) - A    0.00006160  0.00012363  0.00002807  0.00001676  0.00005914  0.00000630  0.00006126  0.00018806  0.00097143  0.00000592
30 mins (5*6) - B    0.00006163  0.00012343  0.00002799  0.00001675  0.00005916  0.00000630  0.00006110  0.00018807  0.00097133  0.00000593
1 hour (15*4) - C    0.00006172  0.00012332  0.00002786  0.00001674  0.00005915  0.00000632  0.00006102  0.00018795  0.00097163  0.00000594
2 hours (30*4) - D   0.00006187  0.00012316  0.00002774  0.00001675  0.00005921  0.00000629  0.00006141  0.00018777  0.00097270  0.00000596
4 hours (1*4) - E    0.00006188  0.00012319  0.00002767  0.00001676  0.00005913  0.00000628  0.00006118  0.00018720  0.00097285  0.00000594
8 hours (2*4) - F    0.00006164  0.00012324  0.00002759  0.00001669  0.00005904  0.00000629  0.00006101  0.00018718  0.00097402  0.00000592
12 hours (4*3) - G   0.00006157  0.00012330  0.00002764  0.00001667  0.00005909  0.00000631  0.00006098  0.00018767  0.00097327  0.00000592
16 hours (8*2) - H   0.00006141  0.00012331  0.00002766  0.00001666  0.00005903  0.00000630  0.00006096  0.00018770  0.00097485  0.00000590
36 hours (12*3) - I  0.00006170  0.00012431  0.00002797  0.00001664  0.00005939  0.00000631  0.00006133  0.00018937  0.00098097  0.00000593
2 days (1*2) - J     0.00006145  0.00012482  0.00002795  0.00001675  0.00005982  0.00000633  0.00006146  0.00019080  0.00098945  0.00000595
2 weeks (1*2) - K    0.00005880  0.00012637  0.00002802  0.00001643  0.00005944  0.00000655  0.00006240  0.00019320  0.00097030  0.00000602
Chart
Legend               ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK ABCDEFGHIJK
```

Рисунок 1.5 – Відображення замовленнь у застосунку GeastWood BinanceAPICLI

У цьому випадку необхідно додатково прописувати занадто велику кількість додаткових символів, а також у командах відсутні псевдоніми для скорочених назв, коли у той самий час деякі з атрибутів з самого початку прописані у скороченому форматі, що може збити з пантелику користувача.

Sell

- `atcb sell --pair=MDABTC --price="130%" --qty="100%"` Place a limit order with all your MDA, with 130% of current market price

Рисунок 1.6 – Використання команд у застосунку GeastWood BinanceAPICLI

Подібні недоліки при активній розробці можна досить швидко виправити та розробити дійсно гарний застосунок. Саме з такими намірами розробник написав список майбутніх функціональностей застосунку, які повинні були б з'явитись, проте більша частина з них зараз знаходиться на стадії розробки, а сам проект не оновлювався з 2019 року.

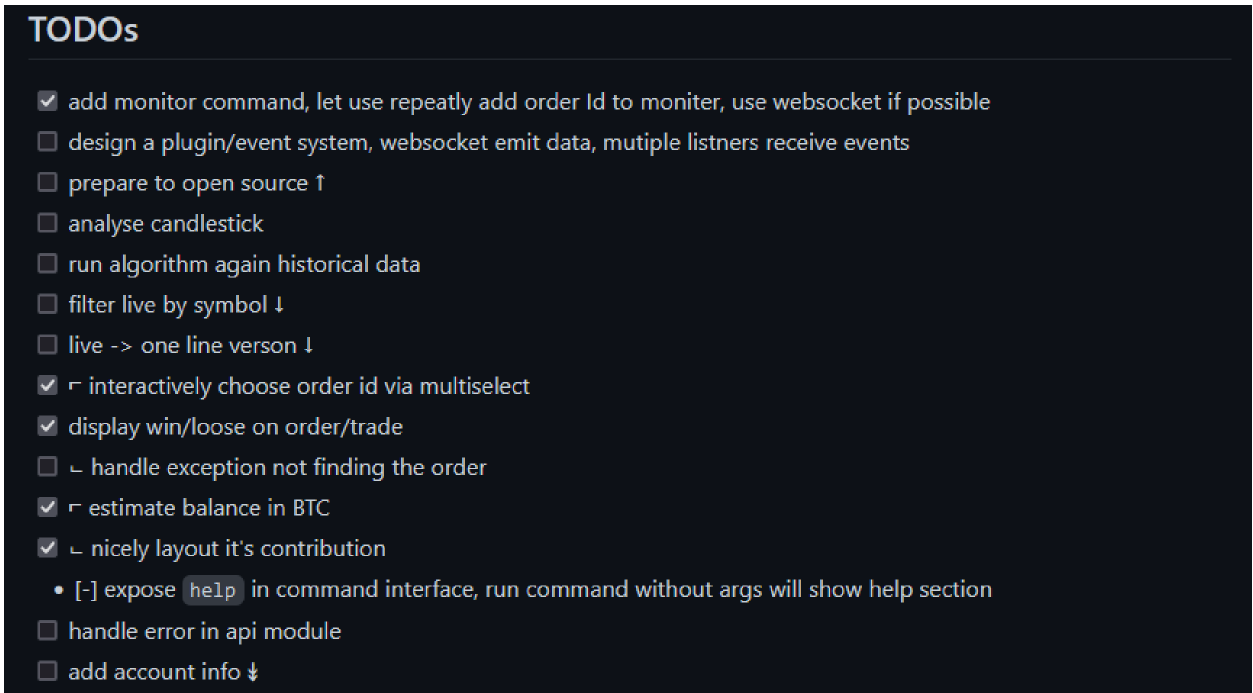

Рисунок 1.7 – Прогрес виконання проекту GeastWood BinanceAPICLI

В результаті отримуємо, що існує мала кількість CLI додатків з такою специфікою, а розробляється або просто підтримується лише офіційний застосунок, тому розробка нового програмного забезпечення, яке має більше функціональностей, а також більш зручний для використання є досить актуальною проблемою.

1.4 Аналіз програмного модуля

Для розробки даного застосунку, необхідно розробити:

- Налаштування та підключення програмою до Binance за допомогою Binance API та ключів доступу до нього.
- Отримання поточної інформації з клієнту користувача.
- Можливість виконувати базові операції купівлі та продажу.
- Створення, видалення, відображення та редагування шаблонів замовлень для прискорення роботи
- Відображення отриманої інформації від API у зручному для читання варіанті для людини
- Обробка помилок та їх вивід у зручному для читання варіанті

1.5 Постановка задачі

У процесі розробки буде створений зручний CLI застосунок для здійснення швидких операцій на криптовалютній біржі Binance використовуючи офіційний API компанії.

Функціональні вимоги:

- Застосунок має підключатись до криптовалютної біржі Binance за допомогою API ключів.
- Застосунок має надавати доступ до балансу користувача, а також інформації щодо ціни та комісії активів.
- Застосунок має відкривати, закривати та передивлятись активні замовлення.
- Застосунок має створювати, видаляти, редагувати, зчитувати шаблони для замовлень.
- Застосунок має виконувати шаблонні замовленя та перевіряти їх цілісність.
- Нефункціональні вимоги:
- Продуктивність – застосунок має виконувати будь-яку команду або виводити помилку менше ніж за 2 секунди.
- Безпека – застосунок не має зберігати жодної інформації щодо користувача, окрім необхідних API ключів та власноруч створених шаблонів.

Висновок до першого розділу

Отже, у ході виконання першого розділу був проведений аналітичний огляд кваліфікаційної роботи. Проаналізовані методології розроблення програмного забезпечення такі як: каскадна модель, v-модель, спіральна модель та Scrum, а також були відокремлені їх недоліки, переваги та особливості у внутрішніх процесів. Проаналізовані та розглянуті вже створені CLI застосунки для торгівлі на криптовалютній біржі Binance, враховуючи офіційний.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ CLI ЗАСТОСУНКУ

2.1 Архітектура програмного модуля

2.1.1 Функції програмних модулів

Застосунок складається з 3 взаємопов'язаних модулів:

- Модуль клієнту

1. Ініціалізація клієнту – підключення до офіційного клієнту Binance за допомогою API ключів.
2. Отримання комісій – отримання інформації щодо комісій активу.
3. Отримання поточного балансу користувача.
4. Отримання поточної ціни активу до всіх або визначених валют.

- Модуль маніпулювання замовленнями

1. Створення замовлення.
2. Відображення активних замовлень та їх детальна інформація.
3. Закриття усіх замовлень певного активу або за допомогою ідентифікаційного номеру.

- Модуль маніпулювання шаблонами

1. Створення шаблону.
2. Відображення інформації про шаблону у консолі.
3. Часткове редагування шаблону за допомогою атрибутів.
4. Видалення шаблону по його назві.
5. Виконання шаблону без змін та з ними за допомогою атрибутів без фізичної зміни.

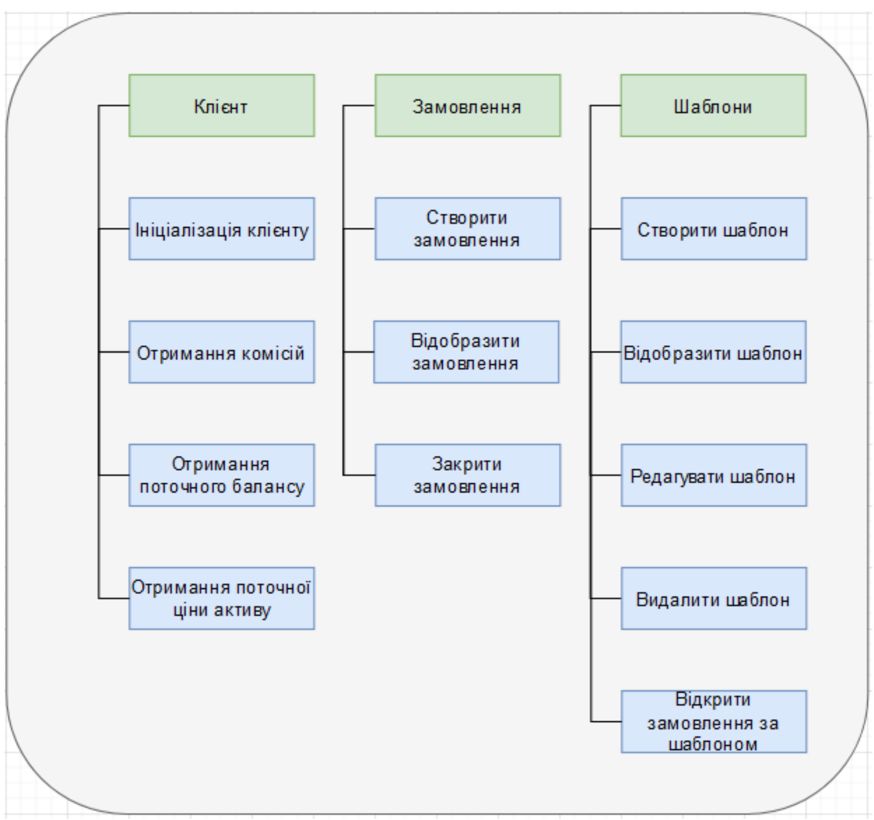

Рисунок 2.1 – Дерево функцій

2.1.2 Архітектура інформаційної системи

Система складається з 4 модулів:

- Підсистема клієнту відповідає за підключення CLI додатку до клієнту Binance за допомогою API ключів, які мають бути записані у файл config.json. Таким чином при виконанні команди ми будемо створювати об’єкт клієнту у якому з реалізацією патерну Singleton, тобто одночасно може бути створений лише 1 об’єкт клієнту, а також класс має функції отримання інформації з клієнту.

- Підсистема команд містить у собі описи функцій, які доступні у застосунку, параметри та посилається на їх реалізацію, а саме функції: отримання інформації балансу, комісій, ціни активів, відкриття, закриття та

відображення відкритих замовлень, створення, видалення, відображення та редагування шаблонів замовлень.

- Підсистема відображення включає у собі набір функцій для відображення колекцій даних, які надходять з API у різних форматах або потребують специфічного прогону елементів, тому що сам інтерфейс надає дані не в уніфікованому виді. Також тут знаходяться стилі, які застосовуються для стилізації консолі. Ця підсистема є допоміжною.

- Файлова підсистема містить у собі функції для зчитування, редагування інформації у шаблонах, а також їх видалення та виконання.

2.1.3 Життєвий цикл

При запуску команди створюється нова робоча сесія та далі відбувається валідація вхідних параметрів команди, введених після команди, якщо деякі параметри відсутні – застосунок виводить відповідну помилку. Далі необхідно ініціалізувати клієнт та підключитись до Binance за допомогою API ключів, які мають бути записані у файл config.json. Після успішної ініціалізації з клієнту на сервер йде запит в залежності від введеної команди, з яких може бути:

- Отримання інформації – надсилається GET запит на сервер, який у результаті повертає JSON файл з необхідною інформацією або помилкою, якщо дані були введені неправильно.

- Створення замовлення – надсилається POST запит з необхідною інформацією для створення замовлення. До обов’язкових атрибутів відносять: назву активу, операцію, кількість, ціну.

- Закриття замовлення – надписається DELETE запит з ідентифікаційним номером замовлення, або назвою активу. При введені лише назви активу будуть закриті усі замовлення за цією назвою

- Створення, редагування, видалення та зчитування шаблонів – ці команди не потребують ініціалізації та підключення до клієнту, тому що вони працюють локально на комп’ютері з файловою системою відповідно до назв команд, проте для команди виконання шаблону ці умови також необхідні.

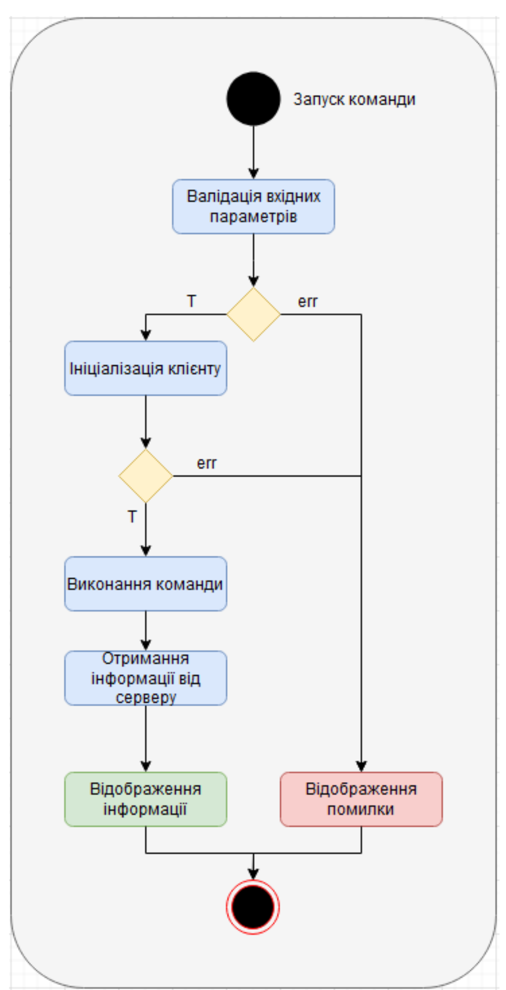

Рисунок 2.2 – Життєвий цикл застосунку

2.1.4 Фізична структура

Файлова структура та опис файлів наведені у таблиці 2.1.

Таблиця 2.1 – Файлова структура проекту

№	Назва	Опис
1	root	Корнева папка проекту, в якій знаходяться усі інші файли та папки проекту
2	index.js	Початковий файл застосунку
3	config.json	JSON-файл з API ключами, необхідними для підключення до клієнту Binance
4	package.json	Конфігураційний файл, у якому описуються усі залежності проекту, які необхідно встановити для коректної роботи застосунку
5	.gitignore	Git-файл, який вказує ті файли та папки, які не потрібно додавати до репозиторію
6	bin	Папка у якій знаходяться файли з реалізацією застосунку
7	Binance.js	Класс, який ініціалізує клієнт та реалізує функції отримання відфільтрованих даних
8	commands	Папка з файлами реалізацій команд
9	commands.js	Файл-контроллер у якому міститься інформація про команди, їх аргументи, параметри, опис та налаштування відображення у консолі
10	binanceInfo	Папка, у якій знаходяться файли з реалізацією команд для отримання інформації з клієнту
12	balance.js	Файл з реалізацією функції отримання балансів по текстовому патерну
12	fee.js	Файл з реалізацією функції отримання комісій по текстовому патерну
13	price.js	Файл з реалізацією функції отримання цін активів по текстовому патерну
14	order	Папка, у якій знаходяться файли з реалізацією команд для маніпулювання замовленнями
15	makeOrder.js	Файл з реалізацією функції створення замовлення
16	displayOrder.js	Файл з реалізацією функції відображення відкритих замовлень
17	closeOrder.js	Файл з реалізацією функції закриття замовлення по ідентифікаційному номеру або усіх

Таблиця 2.1 – Файлова структура проекту

18	template	Папка, у якій знаходяться файли з реалізацією команд для маніпулювання шаблонами замовлень
19	createTemplate.js	Файл з реалізацією функції створення шаблону
20	readTemplate.js	Файл з реалізацією функції відображення параметрів шаблону у консолі
21	updateTemplate.js	Файл з реалізацією функції редагування шаблону
22	deleteTemplate.js	Файл з реалізацією функції видалення шаблону
23	executeTemplate.js	Файл з реалізацією функції виконання шаблонного замовлення
24	helpers	Папка з додатковими утилітами, які не відносяться безпосередньо до роботи з клієнтом
25	FileUtils.js	Файл з реалізацією функцій зчитування та запису файлів
26	BuffedDisplayer.js	Файл з реалізацією функцій стилізованого відображення у консолі, а також налаштування кольорів для елементів таблиць
27	Options.js	Файл з константами
28	templates	Папка у якій зберігаються створенні шаблони замовлень у форматі

Візуалізацію файлової структури можна побачити на рисунку 2.3.

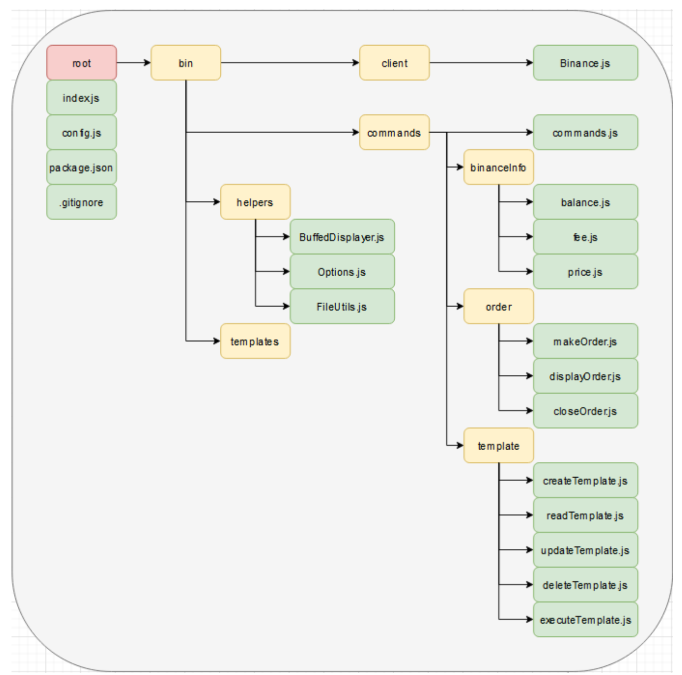

Рисунок 2.3 – Файлова структура застосунку

2.2 Узагальнена архітектура клієнт-сервер

Клієнт-серверна архітектура передбачає те, що є клієнт та сервер, які спілкуються між собою за допомогою запитів від клієнта та відповідей від серверу. Клієнтом, зазвичай, виступає додаток, який використовує графічний інтерфейс, таким клієнтом може бути як браузер, так і настільний додаток. Клієнт надсилає запит на сервер, який у свою чергу намагається його обробити та надіслати відповідь назад. Проте CLI застосунок не має графічного інтерфейсу і для відправлення таких запитів потрібно прописувати ці запити власноруч на кінцеві точки API серверу. У інтерфейсу мають бути описані чіткі правила яку інформацію треба надіслати та за допомогою якого методу, щоб отримати очікуваний результат. До стандартних методів HTTP протоколу відносять:

- GET – запит для отримання інформації з серверу. Результатом запиту може бути будь-який текст у будь-якому текстовому форматі, проте у більшості випадках використовується формат JSON.

- POST – запит для створення інформації на сервері. Для операції створення необхідно надати усю необхідну інформацію, що потребує API до тіла запиту.

- PUT – запит для заміни існуючого елементу на новий. Для заміни даних необхідно вказати ідентифікаційний параметр, який може бути як номером, так і назвою активу.

- DELETE – запит для видалення інформації з серверу. Для видалення, як і для заміни, необхідно вказати ідентифікаційний параметр.

- PATCH – запит для часткової зміни існуючого елементу. Навідміну від PUT запиту, PATCH не замінює об’єкт, а саме зберігає його минулий стан та частково змінює поля, які були зазначені у тілі запиту.

Існує два типи клієнт-серверної архітектури: дворівнева та трирівнева:

- Дворівнева архітектура передбачає, що інтерфейс користувача працює на клієнті, а база даних зберігається на сервері. Логіка додатків,

- Трирівнева архітектура ж розробляється як незалежні модулі: верхній, середній та нижній рівень, де інтерфейс користувача та бізнес-логіка розробляються окремо. До верхнього рівню відносять візуальне відображення застосунку, тобто графічний інтерфейс. Середній рівень – це сервер, який відповідає за процеси моніторингу діяльності, ресурсами тощо.Нижній рівень, у свою чергу, забезпечує можливість управління базами даних.

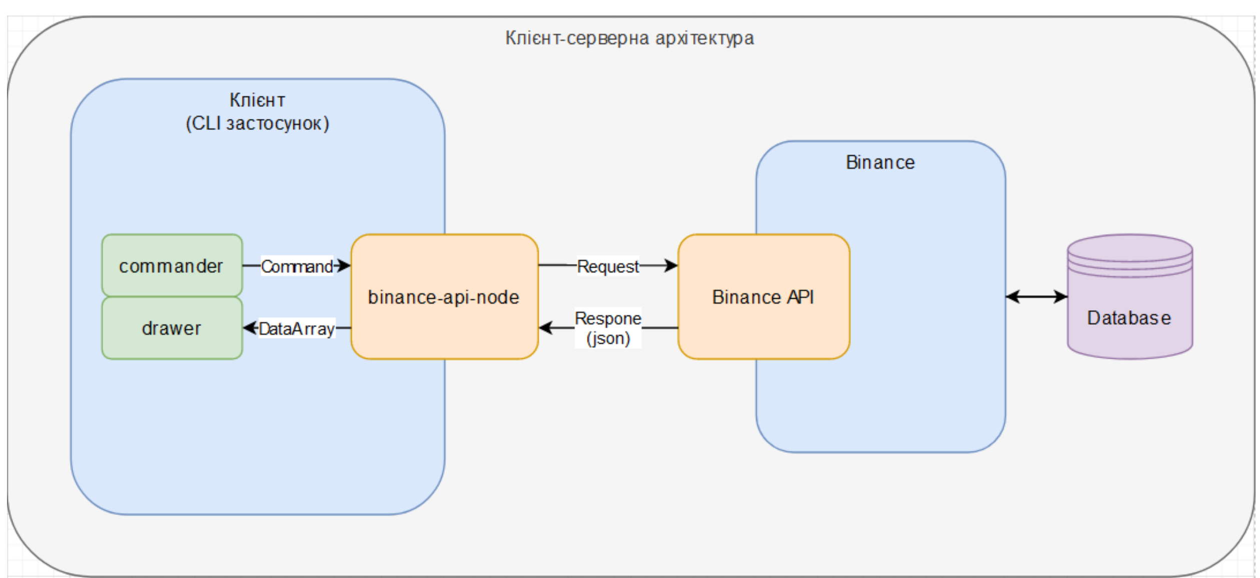

Рисунок 2.4 Клієнт-серверна архітектура

Таким чином, можемо спостерігати, що застосунок знаходиться у трирівневій клієнт-серверній архітектурі.

2.3 Опис класів та методів

index.js – головний файл застосунку в якому міститься “#!/usr/bin/env node”, яка робить наш файл виконуваним, а також імпорт усіх команд для можливості їх використання.

commands.js – містить у собі опис усіх можливих команд, їх параметри, аргументи, описи та реалізації. Файл також є виконуваним як і index.js.

Binance.js – містить у собі розширення класу BinanceProfile, тобто має усі функції, що надаються binance-node-api, а також додатково має реалізацію фільтрації даних. Клас має конструктор, у якому зчитує API ключі з config.json файлу та ініціалізує клієнт з яким далі можна буде працювати. Також є функції:

- getBalance – асинхронна функція, що отримує від сервера інформацію про поточний стан активів користувача та фільтрує їх за ім’ям та флагом. Приймає 2 параметри:

1. symbol – повна або часткова назва активу.
2. notNullBalance – флаг, який вказує на те, що необхідно виводити лише ті активи, сумма яких більше 0.

- getPrice – асинхронна функція, що отримує від сервера інформацію про ціну активу та фільтрує їх за назвою та флагом. Примає 2 параметри:

1. symbol – повна або часткова назва активу.
2. option – об’єкт у якому може бути 3 опції по яким буде виконуватись пошук до певної пари: -u – USDT, -b – BTC, -e – ETH.

- getFee – асинхронна функція, яка отримує рівень комісій активів. Примає 1 параметр:

1. symbol - повна або часткова назва активу.

- getPrescision – асинхронна функція, що отримує точність округлення для певного активу, тобто його мінімальний шаг. Приймає 1 параметр:

1. symbol – повна назва активу.

makeOrder.js – функція для створення замовлення, має 3 обов’язкових параметри:

1. symbol – повна назва активу.
2. operation – операція (BUY або SELL).
3. amount – кількість монет або сума у доларах.

Також команда декілька параметрів, які вказувати не обов’язково:

1. price – ціна, за яку буде куплятись актив. Якщо параметр відсутній, тоді до замовлення буде доданий параметр type = “MARKET”, тобто замовлення виконується відразу по ціні маркету.

2. options – може мати 2 параметри: market – купівля по ціні маркету, та usd – конвертує amount з кількості монет до суми у доларах та округлює його до максимальної точності.

displayOrder.js – функція для відображення активних замовлень, має 1 обов’язковий параметр:

1. symbol – повна назва активу.

closeOrder.js – функція для закриття активних замовлень, має 1 обов’язковий параметр:

1. symbol – повна назва активу.

А також має 1 не обов’язковий параметр:

1. orderId – ідентифікаційний номер замовлення, якщо цього номеру нема, тоді команда закриє усі замовлення обраного активу.

createTemplate.js – функція для створення шаблону замовлення. Приймає ті ж самі параметри, що і makeOrder, а також назву шаблону, проте не виконує замовлення, а зберігає його як json файл до папки templates.

deleteTemplate.js – функція для видалення шаблону замовлення. Приймає лише 1 параметр:

1. templateName – назва шаблону без його розширення.

readTemplate.js – функція для зчитування інформації про певний шаблон. Приймає 1 параметр:

1. templateName - назва шаблону без його розширення.

updateTemplate.js – функція для редагування шаблону приймає 2 обов’язкових параметри:

1. templateName – назва шаблону без його розширення.
2. options – об’єкт, який має наступні параметри:

2.1 symbol – назва активу.

2.2 side – операція (BUY або SELL).

2.3 quantity – кількість активу.

2.4 usd – флаг, при значенні true конвертує кількість монет до доларового еквіваленту.

2.5 price – ціна по якій буде купуватись актив.

executeTemplate.js – функція для створення замовлення на базі шаблону, має 1 обов'язковий параметр:

1. templateName - назва шаблону без його розширення.

А також має такі ж самі опції як і updateTemplate для зміни виконуваного шаблону без його фізичної зміни. Таким чином можна досить гнучко їх використовувати та швидко змінювати за необхідності.

balance.js – функція для відображення поточного балансу користувача. Має 2 не обов'язкових параметри:

1. symbol – часткова назва активу, якщо параметр відсутній, відображатиметься весь список.

2. notNullBalance – якщо значення true –відображає лише ті активи, кількість яких більше 0.

fee.js – функція для відображення комісій активів. Має 1 не обов'язковий параметр:

1. symbol – часткова назва активу, якщо параметр відсутній, тоді відображає повний список.

price.js – функція для відображення ціни активів. Має 2 необов'язкових параметри:

1. symbol – часткова назва активу, якщо параметр відсутній, тоді відображає повний список.

2. options – об’єкт, який містить 3 опції для знаходження ціни для найпоширеніших валютних пар: -u – USDT (Tether), -e – ETH (Ethereum), b – BTC (Bitcoin).

BufferedDisplayer.js – файл з набором функцій, які відповідають за стилізоване відображення таблиць та помилок. Містить у собі наступні функції:

- displayFees – приймає масив комісій, який по черзі відображає у консолі.
- displayPrices – приймає масив Nx2, який по черзі відображає у консолі.
- displayBalances – приймає масив активів користувача та відображає їх у консолі.
- displayBinanceError –приймає помилку від серверу Binance та відображає текст помилки та її код без трасування.
- displayNodeError – приймає помилку від Node.js та відображає текст помилки та її код без трасування.
- displayError – приймає текст кастомної помилки.
- displayWarning – приймає текст для відображення попередження.
- displaySuccess – приймає текст для відображення успішної операції.
- displayOrders – приймає масив замовлень та відображає їх у вигляді таблиці.

FileUtils.js – файл з набором функцій, які відповідають за операції з файлами шаблонів замовлень. Містить у собі наступні функції:

- readTemplateFile – примає назву шаблону. Якщо файл існує – повертає зміст файлу як JavaScript об’єкт.
- deleteTemplateFile – приймає назву шаблону. Якщо файл існує – видаляє його та повертає булеве значення true, якщо ні – false.

- updateTemplateFile – приймає назву шаблону та новий об’єкт. Якщо запис пройшов успішно – повертає булеве значення true, якщо ні – false.

2.4 Вибір середовища та інструментів реалізації

Для реалізації CLI застосунків можна використовувати будь-які мови програмування, такі як Java, C#, C++, Ruby, проте їх використовують для великих проектів. Також при код при розробці консольних застосунків має досить громістку структуру, яку досить важко зрозуміти, тому була обрана мова JavaScript та фреймворк NodeJS. За допомогою пакетного менеджеру можна гнучко налаштовувати залежності проекту та мінімізувати його розмір.

Для розробки команд, їх опису та реалізації функцій була використана бібліотека commander, яка містить у собі клас program поверх якого потрібно лише описати назви команд, їх атрибути, параметри за замовчуванням та псевдоніми.

Для доступу до клієнту використовувалась бібліотека binance-api-node, яка містить у собі певний набір команд та реалізацію надсилання запитів до серверу Binance.

Після отримання інформації з серверу та приведення до певного формату необхідно відобразити її у консоль у читаємому виді для користувача. Для налаштування візуальних стилів використовувалось 2 бібліотеки:

1. chalk – має набір команд для редагування кольорів та стилізації тексту, курсиву, жирного, підкресленого тексту тощо.
2. easy-table – за допомогою класу Table формує таблицю та автоматично розширює її в залежності від розміру контенту кожної з колонок. Функції розроблені таким чином, що усі елементи, що будуть відображатись можна

стилізувати за допомогою chalk, таким чином можна керувати структурою таблиці та її дизайном.

Для шаблонів була використана вбудована в NodeJS бібліотека fs, яка має функції для маніпулюванням файлами, з яких: зчитування, запис та перевірка чи існує файл.

Для того щоб використовувати ці бібліотеки також необхідно мати на комп'ютері встановлений Node.js, а також node package manager, тобто npm – це пакетний менеджер, який використовується для налаштування залежностями, встановленням бібліотек, а також керуванням версій цих бібліотек, їх оновленням тощо.

Висновок до другого розділу

У ході виконання другого розділу дипломної роботи було проведено функціональний аналіз та побудовано дерево функцій програмних модулів. Розглянутий життєвий цикл застосунку, а також файлова структура проекту. Описанана клієнт-серверна взаємодія CLI застосунку з сервером Binance, класи та методи, які використовуються у проекті, а також аналіз та опис середовища та інструментів для реалізації.

РОЗДІЛ 3. ТЕХНОЛОГІЧНІ ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО МОДУЛЯ

3.1 Налаштування Binance API

Для того щоб користуватись додатком та налаштувати доступ до клієнту спочатку потрібно його створити, для цього необхідно зареєструватись на офіційному сайті Binance, підтвердити свій акаунт. Після цього необхідно створити API ключі для того, щоб надати нашому застосунку доступ до клієнту. Для цього потрібно перейти до вкладки “Управління API”, як зазначено на рисунку 3.1.

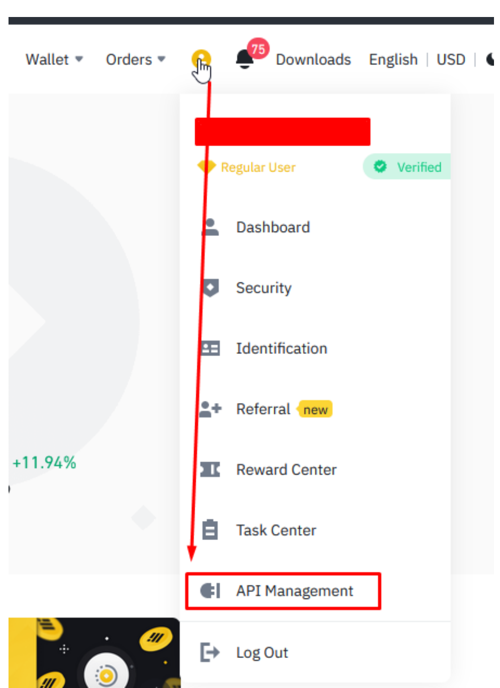

Рисунок 3.1 – Управління Binance API

Після цього необхідно створити новий API та задати йому унікальне ім’я у відповідному полі. Після заповнення форми (3.2) можна натиснути на кнопку “Далі”.

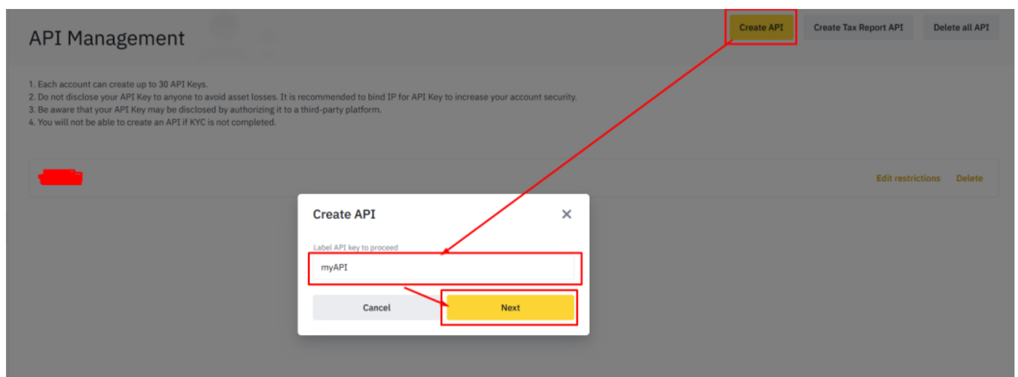

Рисунок 3.2 – Створення Binance API

Сторінка, що з'явилась далі відповідає за налаштування доступу API до клієнту Binance зі сторонніх ресурсів. За допомогою цього можна створити декілька копій, які будуть відповідати за різні модулі торгівлі. Обов’язковим параметром для коректної роботи є “Дозволити спотову та маржинальну торгівлю”. Коли налаштування завершені потрібно зберігати їх та скопіювати ключі і додати їх до config.js файлу для подальшої ініціалізації клієнту у CLI застосунку.

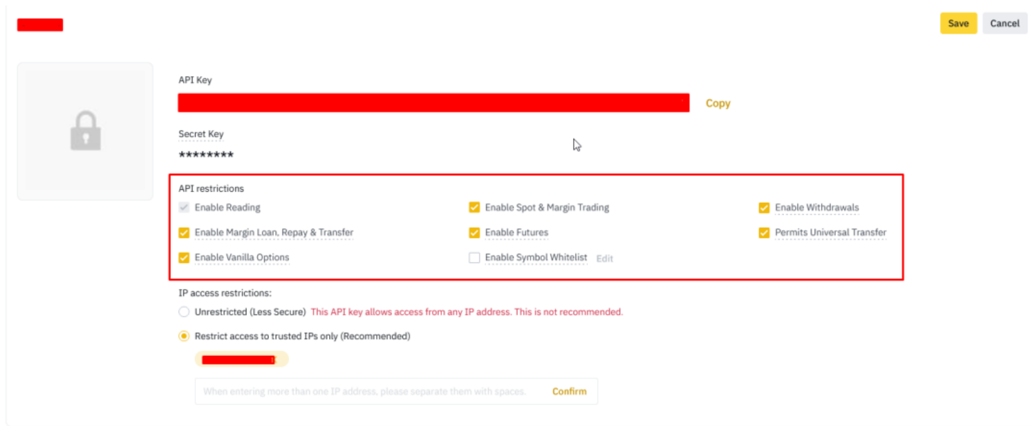

Рисунок 3.3 – Налаштування Binance API

JSON файл з ключами буде мати наступний вигляд (3.4). На місці червоних прямокутників необхідно записати свої ключі отримані на сайті Binance. Також варто відмітити, що назви атрибутів мають бути написані саме так, як це зазначено на рисунку 3.4.

```
{} config.json > ...
1  {
2      "publicKey":"",
3      "privateKey":""
4  }
```

Рисунок 3.4 – Налаштування файлу config.json

3.2 Реалізація застосунку

У першу чергу був розроблений клас клієнта, завдяки якому виконувалась його ініціалізація за допомогою API ключів, а також розробка додаткових функцій для отримання інформації щодо активів, тобто команди “tt balance”, “tt price”, “tt fee”. Якщо застосунок не виводить повідомлення про помилку, тоді налаштування програми пройшло успішно і користувач має доступ до свого акаунту через CLI.

Для того щоб отримати інформацію щодо усіх команд, які є у застосунку необхідно виконати команду “tt --help”. Результат виконання команди наведено на рисунку 3.5.

```
C:\WINDOWS\system32\cmd.exe

:\Users\simpl>tt --help
sage: index tt <command> <parameters> [options]

ptions:
 -h, --help                                                                display help for command

ommands:
 balance|b [options] [symbol]                                              Get balance information
 price|p [options] [symbol]                                                Get asset price
 fee|f [symbol]                                                            Get asset fee
 make-order|mo [options] <symbol> <operation> <amount> [price]             Open orders
 display-order|do <symbol>                                                 Shows all opened orders by asset name
 close-order|co <symbol> [orderId]                                         Close order
 create-template|ct [options] <templateName> <symbol> <operation> <amount> [price]  Create template
 read-template|rt <templateName>                                           Read template
 update-template|ut [options] <templateName>                               Update template
 delete-template|dt <templateName>                                         Delete template
 execute-template|et [options] <templateName>                              Execute template
 help [command]                                                            display help for command

:\Users\simpl>_
```

Рисунок 3.5 – Команди CLI застосунку

На наступному етапі було необхідно зробити базові функціональності маніпулювання замовленнями: створення, закриття та відображення замовлення за допомогою наступних команд: “tt make-order”, “tt close-order” та “tt display-order”. Таким чином ми вже можемо робити усі необхідні операції над замовленнями, а якщо користувач введе не всі атрибути до команди, то отримає відповідну помилку і зможе швидко її дописати. Результат виконання команд зазначено на рисунку 3.6.

```
C:\WINDOWS\system32\cmd.exe

C:\Users\simpl>tt mo
Error: missing required argument 'symbol'

C:\Users\simpl>tt mo MIRUSDT
Error: missing required argument 'operation'

C:\Users\simpl>tt mo MIRUSDT BUY
Error: missing required argument 'amount'

C:\Users\simpl>tt mo MIRUSDT BUY 1000
Error: Account has insufficient balance for requested action. Code #-2010

C:\Users\simpl>_
```

Рисунок 3.6 – Використання команди створення замовлення

Проте кожного разу вводити команди знов і знов – це не найкращій вихід, адже у такій справі секунди можуть відіграти дуже важливу роль, тому необхідно було рішення цієї проблеми. Потрібно було зберігати подібні команди для подальшого використання у скороченій формі, тобто щоб їх можна було використовувати ще швидше.

Рішенням цієї проблеми було створення шаблонів замовлень, де будуть зберігатись усі атрибути та опції у JSON форматі, які у свою чергу будуть просто зберігатись у папці проекту та виконуватись за допомогою команди “tt execute-template”. Таким чином замість введення усіх параметрів раз за разом користувачу необхідно лише ввести скорочену команду “tt et” та додати назву виконуваного файлу. Також для більшої гнучкості до команди була додана можливість виконувати цей шаблон зі зміною атрибутів через їх передачу у консолі без зміни самого файлу. Це дає можливість налаштувати декілька шаблонів та використовувати їх замінюючи 1-2 необхідних параметри зі зберіганням початкового стану шаблону. Параметри, які можна замінити можна побачити на рисунку 3.7.

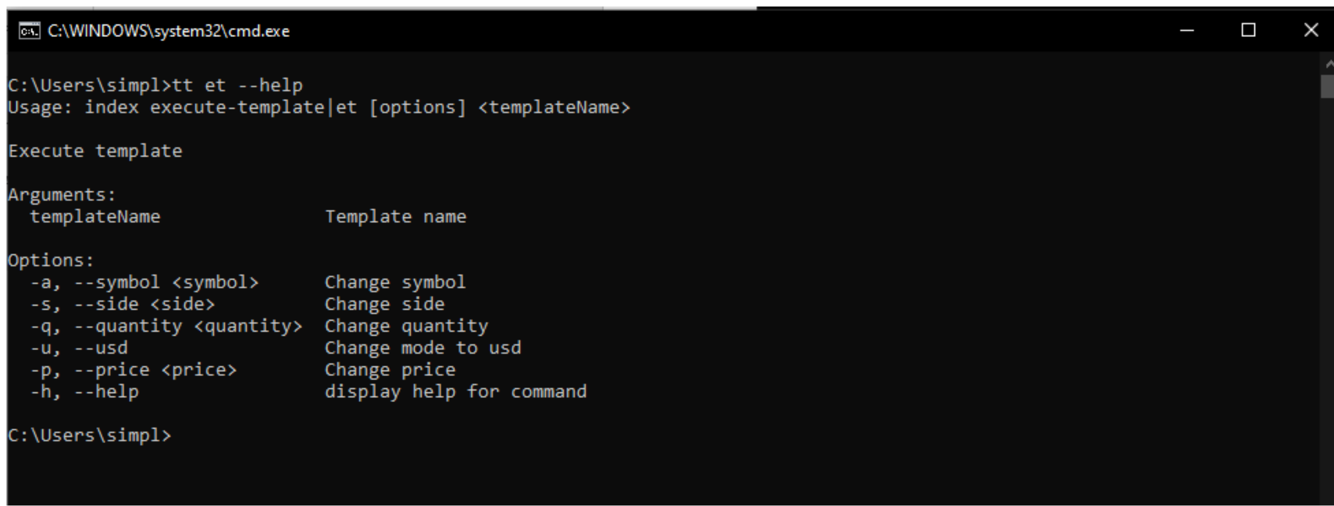

Рисунок 3.7 – Параметри команди execute-template

3.3 Інструктивний матеріал з експлуатації

3.1.1 Експлуатація модулю клієнту

Після того як налаштування завершені можна скористатись перевіркою балансу для цього необхідно ввести команду “tt balance MIR” для того, щоб перевірити кількість активів на гаманці. Результат виконання команди можна побачити на рисунку 3.8.

```
C:\WINDOWS\system32\cmd.exe

C:\Users\simpl>tt b MIR
Asset  Free         Locked      Total
-----  -----------  ----------  -----------
MIR    0.00000000   0.00000000  0
LDMIR  29.02782209  0.00000000  29.02782209


C:\Users\simpl>
```

Рисунок 3.7 – Перевірка балансу користувача

За допомогою команди “tt fee MIR” можна побачити комісії до інших валют на рисунку 3.8. Також варто відмітити, що комаду можна використовувати без назви активу і застосунок відобразить всі існуючі на даний момент.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\simpl>tt fee MIR
Sybmol   Maker Commission  Taker Commission
-------  ----------------  ----------------
MIRBTC   0.001             0.001
MIRBUSD  0                 0.001
MIRUSDT  0.001             0.001


C:\Users\simpl>
```

Рисунок 3.8 – Перевірка комісій активів

За допомогою команди “tt price MIR” можна побачити ціну активу у даний момент на рисунку 3.9. Так як ціна відображається саме до валютної пари, для уточнення варто використовувати або повну назву цієї пари.

```
C:\WINDOWS\system32\cmd.exe

C:\Users\simpl>tt price MIR
Asset    Price
-------  ----------
MIRBTC   0.00000790
MIRBUSD  0.22682000
MIRUSDT  0.22667000


C:\Users\simpl>
```

Рисунок 3.9 – Перевірка ціни активів

3.1.2 Модуль операцій над ордерами

Цей модуль є найважливішим, адже це є головною функціональністю застосунку. Тому необхідно приділити багато уваги на коректній роботі всіх функцій.

Для створення замовлення необхідно прописати команду “tt make-order USDTUAH BUY 10 30”. Приклад виконання команди зображено на рисунку 3.10.

```
C:\WINDOWS\system32\cmd.exe

D:\_Main\University\4v2\Diploma\Project\diploma>tt make-order USDTUAH BUY 10 30
Success: Order has been successfully placed!
{ symbol: 'USDTUAH', side: 'BUY', price: '30', quantity: '10' }

D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.10 – Створення замовлення

Для того щоб перевірити, що замовлення дійсно утворилося використаємо функцію “tt display-order USDTUAH”. А також, для наглядності, можна створити ще 1 замовлення. Приклад виконання команди зображено на рисунку 3.11.

```
C:\WINDOWS\system32\cmd.exe

D:\_Main\University\4v2\Diploma\Project\diploma>tt display-order USDTUAH
Order Id  Symbol   Side  Price        Quantity     Filled      Type
--------  -------  ----  -----------  -----------  ----------  -----
26569484  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT


D:\_Main\University\4v2\Diploma\Project\diploma>tt make-order USDTUAH BUY 10 30
Success: Order has been successfully placed!
{ symbol: 'USDTUAH', side: 'BUY', price: '30', quantity: '10' }

D:\_Main\University\4v2\Diploma\Project\diploma>tt display-order USDTUAH
Order Id  Symbol   Side  Price        Quantity     Filled      Type
--------  -------  ----  -----------  -----------  ----------  -----
26569484  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT
26569694  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT


D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.11 – Відображення активних замовлень

Як можна побачити – всі замовлення відображаються та активні. Через те що була вказана ціна – тип операції був автоматично змінений на LIMIT, що означає, що замовлення на купівлю або продаж буде саме за цією ціною, якщо ціна при замовленні не вказується, тоді тип змінюється на MARKET і замовлення закривається за поточною ціною.

Для того, щоб закрити замовлення пропишемо команду “tt close-order”, тут варто бути досить уважним з використанням даної функції, тому що при введені лише назви активу – будуть закриті усі активні замовлення, а для видалення одного необхідно у наступному агрументі передати ідентифікаційний номер. Отже закриємо усі замовлення за допомогою команди “tt close-order USDTUAH”. Приклад виконання команди зображено на рисунку 3.12.

```
C:\WINDOWS\system32\cmd.exe

D:\_Main\University\4v2\Diploma\Project\diploma>tt close-order USDTUAH
Success: order(s) has been closed!
Order Id  Symbol   Side  Price        Quantity     Filled      Type
--------  -------  ----  -----------  -----------  ----------  -----
26569484  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT
26569694  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT


D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.12 – Закриття замовлень

Після виконання команди бачимо детальну інформацію щодо закритих замовлень. Якщо замовлення за цей час було б частково виконано – кількість купленого або проданого активу з'явиться б у колонці Filled.

3.1.3 Модуль операцій над шаблонами

Для того щоб створити шаблон необхідно прописати команду “tt ct myTemplate MIRUSDT BUY 10”. Приклад виконання команди зображено на рисунку 3.13.

```
C:\WINDOWS\system32\cmd.exe

D:\_Main\University\4v2\Diploma\Project\diploma>tt ct myTemplate MIRUSDT BUY 10
Warning: price is not given, order type is set to MARKET
Success: template has been created

D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.13 – Створення шаблону

Через те що не була вказана ціна тип замовлення мав стати MARKET, для того щоб перевірити, що шаблон утворився та має всі необхідні дані потрібно перейти до папки проекту у bin/templates. Зміст та місце знаходження шаблону зображено на рисунку 3.14.

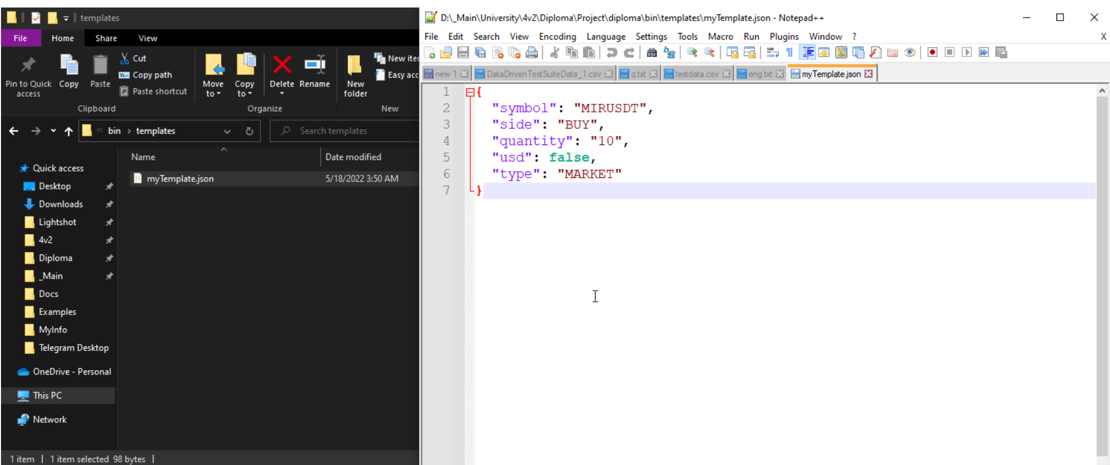

Рисунок 3.14 – Зміст та місце знаходження шаблону

Для того щоб подивитись зміст файлу у консолі необхідно ввести наступну команду “tt rt myTemplate”. Приклад виконання команди зображено на рисунку 3.15.

```
C:\WINDOWS\system32\cmd.exe

D:\_Main\University\4v2\Diploma\Project\diploma>tt rt myTemplate
Success: file has been found
myTemplate.json:
{
  symbol: 'MIRUSDT',
  side: 'BUY',
  quantity: '10',
  usd: false,
  type: 'MARKET'
}

D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.15 – Зчитування шаблону

Для того щоб відредагувати шаблон використовуємо наступну команду “tt ut myTemplate -q 100 -s SELL”. Приклад виконання команди зображено на рисунку 3.16. Варто відмітити, що змінювати можна усі атрибути замовлення, а детальну інформацію можна побачити за допомогою команди “tt ut --help”.

```
C:\WINDOWS\system32\cmd.exe

D:\_Main\University\4v2\Diploma\Project\diploma>tt ut myTemplate -q 100 -s SELL
Success: template has been updated
{
  symbol: 'MIRUSDT',
  side: 'SELL',
  quantity: '100',
  usd: false,
  type: 'MARKET'
}

D:\_Main\University\4v2\Diploma\Project\diploma>tt rt myTemplate
Success: file has been found
myTemplate.json:
{
  symbol: 'MIRUSDT',
  side: 'SELL',
  quantity: '100',
  usd: false,
  type: 'MARKET'
}
```

Рисунок 3.16 – Редагування шаблону

Для того щоб видалити шаблон необхідно прописати команду “tt dt myTemplate”. Для того щоб переконатись, що файл дійсно видалений можна скористатись командою “tt rt myTemplate” та отримати помилку, що такий файл не знайдений, або власноруч зайти до папки з шаблонами. Приклад виконання команди зображено на рисунку 3.17.

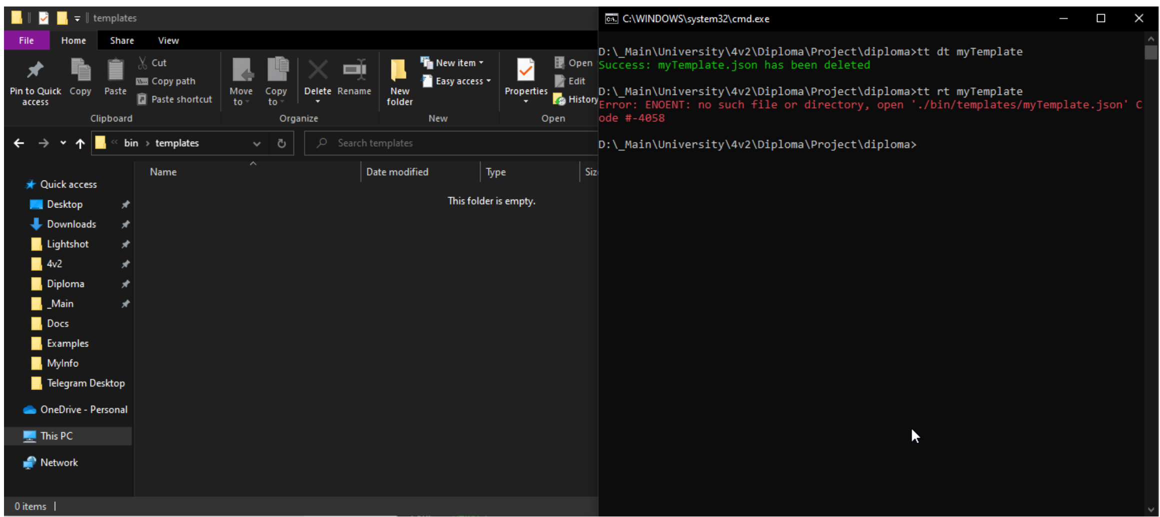

Рисунок 3.17 – Видалення шаблону

Для того щоб виконати шаблон використовується команда “tt execute-template назваШаблону”. Створемо шаблон, який буде конвертувати гривні у долари командою “tt create-template convertUAH USDTUAH BUY 10 30”. Після цього потрібно впевнитись, що шаблон був дійсно створений за допомогою команди “tt read-template convertUAH”, а також що перевірити наявність відкритих замовлень за допомогою “tt display-order USDTUAH”. Тепер можна виконати шаблон командою “tt execute-template convertUAH” та звону перевірити відкриті замовлення командою “tt display-order USDTUAH”. Приклад виконання команди зображено на рисунку 3.18.

```
D:\_Main\University\4v2\Diploma\Project\diploma>tt ct convertUAH USDTUAH BUY 10 30
Success: template has been created

D:\_Main\University\4v2\Diploma\Project\diploma>tt rt convertUAH
Success: file has been found
convertUAH.json:
{
  symbol: 'USDTUAH',
  side: 'BUY',
  quantity: '10',
  usd: false,
  type: 'LIMIT',
  price: '30'
}

D:\_Main\University\4v2\Diploma\Project\diploma>tt do USDTUAH
Warning: no active orders for USDTUAH

D:\_Main\University\4v2\Diploma\Project\diploma>tt et convertUAH
{
  symbol: 'USDTUAH',
  side: 'BUY',
  quantity: '10',
  type: 'LIMIT',
  price: '30'
}
Success: template executed, order has been created

D:\_Main\University\4v2\Diploma\Project\diploma>tt do USDTUAH
Order Id  Symbol   Side  Price        Quantity     Filled      Type
--------  -------  ----  -----------  -----------  ----------  -----
26574482  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT


D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.18 – Виконання шаблону та відображення результатів

Шаблон був успішно виконаний, тепер за допомогою цього шаблону можна відкривати замовлення змінюючи його параметри, наприклад, змінити ціну з 30 до 29 за допомогою параметру -p. Команда після змін матиме наступний вигляд: “tt execute-template convertUAH –p 29”. Перевіримо відкриті замовлення командою “tt display-order USDTUAH”. Приклад виконання команди зображено на рисунку 3.19.

```
D:\_Main\University\4v2\Diploma\Project\diploma>tt et convertUAH -p 29
Success: 'price' option has been changed from 30 to 29
{
  symbol: 'USDTUAH',
  side: 'BUY',
  quantity: '10',
  type: 'LIMIT',
  price: '29'
}
Success: template executed, order has been created

D:\_Main\University\4v2\Diploma\Project\diploma>tt do USDTUAH
Order Id  Symbol   Side  Price        Quantity     Filled      Type
--------  -------  ----  -----------  -----------  ----------  -----
26574482  USDTUAH  BUY   30.00000000  10.00000000  0.00000000  LIMIT
26575972  USDTUAH  BUY   29.00000000  10.00000000  0.00000000  LIMIT


D:\_Main\University\4v2\Diploma\Project\diploma>
```

Рисунок 3.19 – Виконання шаблону зі зміною параметру

Висновки

У ході виконання бакалаврської кваліфікаційної роботи було спроектовано та розроблено застосунок для торгівлі на криптовалютній біржі Binance, а також виконані наступні завдання:

- Досліджені сучасні методології розроблення застосунків. Особливу увагу виділено на гнучку методологію, як одну з найпоширеніших.
- Проаналізовані архітектурні рішення, технології та інструменти реалізації застосунків.
- Програмно реалізований CLI застосунок для торгівлі на криптовалютній біржі Binance.

Були проаналізовані вже існуючі реалізації CLI застосунків торгівлі на криптовалютній біржі Binace, а також були виділені їх переваги та недоліки.

Для проектування та розробки застосунку були використані технології на мові програмування JavaScript до яких входять:

- Commander, за допомогою якого був розроблений інтерфейс, опис команд та їх структурування.
- Binance-node-api, який був використаний для проектування модулів клієнту та модулю, відповідаючого за маніпулювання замовленнями.
- Chalk та easy-table, використовувався для візуального оформлення застосунку та приведення його у зручний для користувача вид.
- Fs, бібліотека, яка надає доступ до файлової системи. За її допомогою була розроблена система маніпулювання шаблонами.

Розробка застосунку виконувалась в IDE Visual Studio Code від компанії Microsoft.

В результаті був розроблений CLI застосунок для торгівлі на криптовалютній біржі Binance, який забезпечує швидкий та зручний доступ до усіх необхідних операцій.

Головними особливостями застосунку є постійний доступ до клієнту за допомогою Binance API, можливість виконувати базові операції для торгівлі, а також збереження, редагування та виконання замовлень за допомогою збереження у файли-шаблони, що значно прискорює подальшу роботу.

Список використаних джерел

1. David Flanagan. JavaScript: The Definitive Guide: Підручник / David Flanagan – JavaScript: The Definitive Guide. 7-ма редакція, 2020 – 689 с.
2. Robert Kowalski. The CLI Book: Writing Successful Command Line Interfaces with Node.js: Підручник / Robert Kowalski – The CLI Book: Writing Successful Command Line Interfaces with Node.js. 1-ша редакція, 2017 – 109 с.
3. Документація NodeJS [Електронний ресурс] // NodeJS – Режим доступу до ресурсу: https://nodejs.org/uk/docs/
4. Документація commander [Електронний ресурс] // Npmjs – Режим доступу до ресурсу: https://www.npmjs.com/package/commander
5. Документація fs [Електронний ресурс] // Npmjs – Режим доступу до ресурсу: https://www.npmjs.com/package/fs
6. Документація chalk [Електронний ресурс] // Npmjs – Режим доступу до ресурсу: https://www.npmjs.com/package/chalk
7. Документація easy-table [Електронний ресурс] // Npmjs – Режим доступу до ресурсу: https://www.npmjs.com/package/easy-table
8. Документація binance-api-node [Електронний ресурс] // Npmjs – Режим доступу до ресурсу: https://www.npmjs.com/package/binance-api-node
9. Документація Binance API [Електронний ресурс] // Binance – Режим доступу до ресурсу: https://www.binance.com/ru/support/faq/c-6
10. SDLC – Waterfall Model [Електронний ресурс] // Tutorial Spoint – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
11. SDLC – V-model [Електронний ресурс] // Tutorial Spoint – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm
12. SDLC – Spiral model [Електронний ресурс] // Tutorial Spoint – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm
13. SDLC – Agile Model [Електронний ресурс] // Tutorial Spoint – Режим доступу до ресурсу: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

14. What is client-server architecture? [Електронний ресурс] // Intellipaat – Режим доступу до ресурсу: https://intellipaat.com/blog/what-is-client-server-architecture/

15. Навчальні матеріали по JavaScript [Електронний ресурс] // Mozilla – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/JavaScript

16. Command Line Interface Guidelines [Електронний ресурс] // Clig – Режим доступу до ресурсу: https://clig.dev/

17. Asynchronous JavaScript [Електронний ресурс] // W3Schools – Режим доступу до ресурсу: https://www.w3schools.com/js/js_asynchronous.asp

18. Binance CLI [Електронний ресурс] // Gthub – Режим доступу до ресурсу: https://github.com/binance/binance-cli

19. Документація Binance API [Електронний ресурс] // Binance Docs – Режим доступу до ресурсу: https://binance-docs.github.io/apidocs/spot/en/#change-log

20. Powershell execution policies [Електронний ресурс] // Microsoft Docs – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7.2

21. How run NodeJS program as an Executable [Електронний ресурс] // GeeksForGeeks – Режим доступу до ресурсу: https://www.geeksforgeeks.org/how-to-run-node-js-program-as-an-executable/

22. JSON Parsing [Електронний ресурс] // Digital Ocean – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/js-json-parse-stringify-ru

23. Докуметнація git [Електронний ресурс] // Git-Scm – Режим доступу до ресурсу: https://git-scm.com/doc

ДОДАТКИ

Додаток А

Лістинг класу BinanceProfile

```
const fs = require("fs")
const Binance = require("binance-api-node").default
const {dirname} = require("path")
const appDir = dirname(require.main.filename)

const BinanceProfile = class BinanceProfile
{
    constructor()
    {
        if(typeof BinanceProfile.instance === 'object')
            return BinanceProfile.instance;

        BinanceProfile.instance = this;
        console.log(appDir);

        var data = JSON.parse(fs.readFileSync(appDir + "\\config.json",{encoding:"utf-8"}))
        this.client = Binance({
            apiKey:data.publicKey,
            apiSecret:data.privateKey,
        })

        return this;
    }

    getBalance = async (assetName, notNullBalance) =>
                    (await BinanceProfile.instance.client.accountInfo()).balances
                    .filter(balance =>  assetName?balance.asset.includes(assetName.toUpperCase()):

notNullBalance===true?(parseFloat(balance.free)>0||parseFloat(balance.locked)>0):true);


    getPrice = async (assetName, option) =>
                    Object.entries(await BinanceProfile.instance.client.prices())
                    .sort()
                    .filter(key => assetName?key[0].includes(assetName.toUpperCase()):true)
                    .filter(key => option && option.bitcoin?key[0].includes("BTC"):
                            option && option.ethereum?key[0].includes("ETH"):
                            option && option.tether?key[0].includes("USDT"):true)

    getFee = async(assetName) =>
    {
        let fees = await BinanceProfile.instance.client.tradeFee()
        let res=[]
        if(!assetName)
```

```
            return fees;

        for(let fee of fees)
            if(fee.symbol.includes(assetName.toUpperCase()))
                res.push(fee);

        return res;
    }

    getPrescision = async (assetName) => {
        return (await this.getExchangeInfo(assetName)).filters.filter(item
=>item.filterType==="LOT_SIZE")[0].stepSize
    }

    getExchangeInfo = async (assetName) => {
        return await ((await
BinanceProfile.instance.client.exchangeInfo({symbol:assetName})).symbols[0])
    }
}
module.exports = BinanceProfile;
```

Додаток Б

Лістинг commands.js

```
#!/usr/bin/env node
const program = require("commander").program
const commander = require("commander")
const {errorStyle} = require("../helpers/BuffedDisplayer.js")
const options = require("../helpers/Options.js")

program.configureOutput({
	outputError:(str,write)=>write(errorStyle(str.replace('error',"Error")))
})

program.usage("tt <command> <parameters> [options]")

program.command("balance")
	.description("Get balance information")
	.argument("[symbol]", "Enter any asset name to see single one")
	.option("-n, --notNullBalance","Shows only assets with amount bigger than zero", false)
	.action(require("./balance.js"))
	.alias("b")

program.command("price")
	.description("Get asset price")
	.argument("[symbol]", "Enter any asset name to see single one")
	.option("-b, --bitcoin", "Show pairs only to bitcoin", false)
	.option("-u, --tether", "Show pairs only to tether", false)
	.option("-e, --ethereum", "Show pairs only to ethereum", false)
	.action(require("./price.js"))
	.alias("p")

program.command("fee")
	.description("Get asset fee")
	.argument("[symbol]", "Enter any aset name to see its fee")
	.action(require("./fee.js"))
	.alias("f")

program.command("make-order")
	.description("Open orders")
	.argument("<symbol>","Asset name")
	.addArgument(new commander.Argument("<operation>", "Enter any aset name to see its fee").choices(options.orderSides))
	.argument("<amount>","Amount of coins you want to buy")
	.argument("[price]","Asset price, use -m to buy with market price")
	.option("-u, --usd", "Use usd amount instead of coin", false)
	.option("-m, --market", "Fill order with market price ", false)
	.action(require("./order/makeOrder.js"))
	.alias("mo")
```

```
program.command("display-order")
    .description("Shows all opened orders by asset name")
    .argument("<symbol>","Asset name")
    .action(require("./order/displayOrder.js"))
    .alias("do")

program.command("close-order")
    .description("Close order")
    .argument("<symbol>","")
    .argument("[orderId]","")
    .action(require("./order/closeOrder.js"))
    .alias("co")

program.command("create-template")
    .description("Create template")
    .argument("<templateName>","Template name")
    .argument("<symbol>","Asset name")
    .addArgument(new commander.Argument("<operation>", "Enter any aset name to see its
fee").choices(options.orderSides))
    .argument("<amount>","Amount of coins you want to buy")
    .argument("[price]","Asset price, use -m to buy with market price")
    .option("-u, --usd", "Use usd amount instead of coin", false)
    .option("-m, --market", "Fill order with market price ", false)
    .action(require("./template/createTemplate.js"))
    .alias("ct")

program.command("read-template")
    .description("Read template")
    .argument("<templateName>","Template name")
    .action(require("./template/readTemplate.js"))
    .alias("rt")

program.command("update-template")
    .description("Update template")
    .argument("<templateName>","Template name")
    .option("-a, --symbol <symbol>","Change symbol")
    .option("-s, --side <side>","Change side")
    .option("-q, --quantity <quantity>","Change quantity")
    .option("-u, --usd","Change mode to usd")
    .option("-p, --price <price>","Change price")
    .action(require("./template/updateTemplate.js"))
    .alias("ut")

program.command("delete-template")
    .description("Delete template")
    .argument("<templateName>","Template name")
    .action(require("./template/deleteTemplate.js"))
    .alias("dt")

program.command("execute-template")
    .description("Execute template")
    .argument("<templateName>","Template name")
```

```
    .option("-a, --symbol <symbol>","Change symbol")
    .option("-s, --side <side>","Change side")
    .option("-q, --quantity <quantity>","Change quantity")
    .option("-u, --usd","Change mode to usd")
    .option("-p, --price <price>","Change price")
    .action(require("./template/executeTemplate.js"))
    .alias("et")

program.parse(program.argv)

module.exports = program;
```

Додаток В

Лістинг реалізацій команд

closeOrder.js

```
const BinanceProfile = require("../../client/Binance.js");
const { displaySuccess,displayOrders,displayError} = require("../../helpers/BuffedDisplayer.js");

const closeOrder = async(symbol, orderId) =>
{
  let binance = new BinanceProfile();
  let closedOrders;

  let assets = (await binance.getPrice()).filter(x=>x[0] === symbol)

  if(assets.length<1){
    displayError(`cannot find such symbol ${symbol}`)
    return
  }

  if(orderId)
    closedOrders = await binance.client.cancelOrder({
        symbol: symbol.toUpperCase(),
        orderId: orderId
      })
  else
    closedOrders = await binance.client.cancelOpenOrders({
        symbol: symbol.toUpperCase()
      })
  displaySuccess("order(s) has been closed!")
  displayOrders(closedOrders)
}

module.exports = closeOrder
```

displayOrder.js

```
const BinanceProfile = require("../../client/Binance.js");
const { displayOrders, displayError, displayWarning } = require("../../helpers/BuffedDisplayer.js");

const displayOrder = async(symbol) => {
  let binance = new BinanceProfile();
  let assets = (await binance.getPrice()).filter(x=>x[0] === symbol)

  if(assets.length<1){
    displayError(`cannot find such symbol ${symbol}`)
```

```
    return
  }

  let orders = await binance.client.openOrders({symbol: assets[0][0]})

  if(orders.length<1)
    displayWarning(`no active orders for ${symbol}`)
  else
    displayOrders(orders)
}

module.exports = displayOrder
```

makeOrder.js

```
const BinanceProfile = require("../../client/Binance.js")
const {displayBinanceError, displayWarning, displaySuccess} = require("../../helpers/BuffedDisplayer.js")

const order = async(symbol, operation, amount, price, options) =>{
    let binance = new BinanceProfile();
    let order = {
        symbol:symbol.toUpperCase(),
        side:operation.toUpperCase(),
    }
    if(options.market || !price){
        price = (await binance.getPrice(symbol.toUpperCase()))[0][1]
        order.type = "MARKET"
    }
    else
        order.price = price
    if(options.usd){
        let prescision = await binance.getPrescision(symbol)
        order.quantity = (amount/price).toFixed(prescision.indexOf("1")-1)
        displayWarning(`Using -u function might decrease amount of asset bought and depends on its prescision!\nCurrent amount of after calculation is ${order.quantity}`)
    }
    else
     order.quantity = amount
    try{
        await binance.client.order(order)
        displaySuccess(`Order has been successfully placed!`)
        console.log(order)
    }catch(e){
        displayBinanceError(e)
    }
}
module.exports = order
```

createTemplate.js

```
const fs = require("fs")
const {displayError, displaySuccess, displayWarning} = require("../../helpers/BuffedDisplayer.js")
const {dirname} = require("path")
const appDir = dirname(require.main.filename)

const createTemplate = (templateName, symbol, operation, amount, price, options) =>
{
    try{
        let path = appDir + `/bin/templates/${templateName}.json`;
        let fileExists = fs.existsSync(path);

        if(!price)
            displayWarning("price is not given, order type is set to MARKET")

        if(fileExists)
            displayWarning("file with the same name is already exists. Fill be updated")

        let template = {
            symbol:symbol.toUpperCase(),
            side:operation.toUpperCase(),
            quantity:amount,
            usd:options.usd,
            type: options.market || !price? "MARKET":"LIMIT",
            price:price
        }

        fs.writeFileSync(path, JSON.stringify(template,null,2),(err)=>{console.log(err)})
        displaySuccess(`template has been ${fileExists?"updated":"created"}`)
    }catch(e){
        displayError(e)
    }

}

module.exports = createTemplate
```

deleteTemplate.js

```
const {displaySuccess} = require("../../helpers/BuffedDisplayer")
const {deleteTemplateFile} = require("../../helpers/Utils.js")

const deleteTemplate = (templateName) =>{
     if(deleteTemplateFile(templateName))
        displaySuccess(`${templateName}.json has been deleted`)
```

```
}

module.exports = deleteTemplate
```

executeTemplate.js

```
const BinanceProfile = require("../../client/Binance.js");
const { displaySuccess, displayBinanceError, displayNodeError } =
require("../../helpers/BuffedDisplayer.js");
const {readTemplateFile} = require("../../helpers/Utils.js")

const executeTemplate = async(templateName, options) => {
   let template;
   try{
      template = readTemplateFile(templateName);
   }catch(e){
      displayNodeError(e)
   }

   if(template)
     for(let option of Object.keys(options))
       if(template[option]!==undefined)
       {
          displaySuccess(`\'${option}\' option has been changed from ${template[option]} to
${options[option]}`)
          template[option] = options[option]
       }
   if(template.usd !== undefined)
      delete template.usd
   console.log(template)

   try{
      await new BinanceProfile().client.order(template)
      displaySuccess("template executed, order has been created")
   }catch(e){
      displayBinanceError(e)
   }
}

module.exports = executeTemplate
```

readTemplate.js

```
const { displaySuccess } = require("../../helpers/BuffedDisplayer");
const { readTemplateFile } = require("../../helpers/Utils");
```

```
const readTemplate = (templateName) => {
  const template = readTemplateFile(templateName);
  if(template){
    displaySuccess(`file has been found\n${templateName}.json:`)
    console.log(template)
  }
}

module.exports = readTemplate
```

updateTemplate.js

```
const { displayWarning, displaySuccess } = require("../../helpers/BuffedDisplayer");
const { readTemplateFile,updateTemplateFile } = require("../../helpers/Utils.js")

const updateTemplate = (templateName, options) => {
  if(Object.keys(options).length===0) {
    displayWarning("no arguments provided to update")
    return
  }

  const template = readTemplateFile(templateName);
  if(template)
    for(let option of Object.keys(options))
      if(template[option]!==undefined)
        template[option] = options[option]



  if(updateTemplateFile(templateName, template)){
    displaySuccess("template has been updated")
    console.log(template)
  }
}

module.exports = updateTemplate
```

balance.js

```
const BinanceProfile = require("../client/Binance.js")
const {displayBalances,displayError} = require("../helpers/BuffedDisplayer.js")

const getBalance = async(symbol, options) => {

   let assets = await new BinanceProfile().getBalance(symbol, options.notNullBalance)
```

```
    if(assets.length<1){
        displayError(`cannot find such symbol ${symbol}`)
        return
    }

    displayBalances(assets)
}

module.exports = getBalance
```

fee.js

```
const BinanceProfile = require("../client/Binance.js")
const {displayFees,displayError} = require("../helpers/BuffedDisplayer.js")

const getFee = async (symbol) => {
    let assets = await new BinanceProfile().getFee(symbol);

    if(assets.length<1){
        displayError(`cannot find such symbol ${symbol}`)
        return
    }

    displayFees(assets)
}

module.exports = getFee
```

price.js

```
const BinanceProfile = require("../client/Binance.js")
const {displayPrices,displayError} = require("../helpers/BuffedDisplayer.js")

const getPrice = async(symbol, option) =>
{
    let assets = await new BinanceProfile().getPrice(symbol,option);

    if(assets.length<1){
        displayError(`cannot find such symbol ${symbol}`)
        return
    }

    displayPrices(assets)
}

module.exports = getPrice
```

Додаток Г

Лістинг допоміжних функцій

BuffedDisplayer.js

```
const Table = require("easy-table")
const chalk = require("chalk")

const log = console.log

const successStyle = chalk.bold.italic.greenBright;
const warningStyle = chalk.bold.italic.yellow;
const errorStyle = chalk.bold.italic.red;

const headerStyle = chalk.bold.blue
const contentStyle = chalk.whiteBright
const assetStyle = chalk.bold.green;


const displayFees = (fees) => {
   const table = new Table();

   for(let fee of fees){
      table.cell(headerStyle("Symbol"),assetStyle(fee.symbol))
      table.cell(headerStyle("Maker Commission"), contentStyle(fee.makerCommission))
      table.cell(headerStyle("Taker Commission"), contentStyle(fee.takerCommission))
      table.newRow();
   }

   log(table.toString())
}

const displayPrices = (prices) => {
   const table = new Table();

   for(let i = 0; i<prices.length;i++){
      table.cell(headerStyle("Symbol"),assetStyle(prices[i][0]))
      table.cell(headerStyle("Price"),contentStyle(prices[i][1]))
      table.newRow();
   }

   log(table.toString());
}

const displayBalances = (balances) => {
   const table = new Table();

   for(let balance of balances){
      table.cell(headerStyle("Symbol"),assetStyle(balance.asset))
```

```
        table.cell(headerStyle("Free"),contentStyle(balance.free))
        table.cell(headerStyle("Locked"),contentStyle(balance.locked))

table.cell(headerStyle("Total"),contentStyle((parseFloat(balance.free)+parseFloat(balance.locked)))
)
        table.newRow()
    }

    log(table.toString())
}

const displayBinanceError = (error) => {
    log(errorStyle(`${error.toString()} Code #${error.code}`))
}

const displayNodeError = (error) => {
    log(errorStyle(`${error.toString()} Code #${error.errno}`))
}

const displayError = (text) => {
    log(errorStyle(`Error: ${text}`))
}

const displayWarning = (text) => {
    log(warningStyle(`Warning: ${text}`))
}

const displaySuccess = (text) => {
    log(successStyle(`Success: ${text}`))
}

const displayOrders = (orders) =>{
    const table = new Table();

    for(let order of orders){
        table.cell(headerStyle("Order Id"),assetStyle(order.orderId))
        table.cell(headerStyle("Symbol"),assetStyle(order.symbol))
        table.cell(headerStyle("Side"),contentStyle(order.side))
        table.cell(headerStyle("Price"),contentStyle(order.price))
        table.cell(headerStyle("Quantity"),contentStyle(order.origQty))
        table.cell(headerStyle("Filled"),contentStyle(order.executedQty))
        table.cell(headerStyle("Type"),contentStyle(order.type))
        table.newRow()
    }

    log(table.toString())
}

module.exports = {
    successStyle,
    warningStyle,
    errorStyle,
```

```
   displayBalances,
   displayPrices,
   displayFees,
   displayError,
   displayWarning,
   displaySuccess,
   displayBinanceError,
   displayNodeError,
   displayOrders
}
```

Options.js

```
const orderSides = ["BUY","SELL"]

module.exports = {
   orderSides
}
```

Utils.js

```
const {displayNodeError} = require("../helpers/BuffedDisplayer.js")
const fs = require('fs')
const {dirname} = require("path")
const appDir = dirname(require.main.filename)

const readTemplateFile = (templateName) => {
   try{
      return JSON.parse(fs.readFileSync(appDir +
`/bin/templates/${templateName}.json`,{encoding:"utf-8"}));
   }catch(e){
      displayNodeError(e)
      return false
   }
}

const deleteTemplateFile = (templateName) => {
   try{
      fs.unlinkSync(appDir + `/bin/templates/${templateName}.json`);
      return true
    }catch(e){
      displayNodeError(e)
      return false
    }
}
```

```
const updateTemplateFile = (templateName, data) => {
   try{
      fs.writeFileSync(appDir + `/bin/templates/${templateName}.json`, JSON.stringify(data,null,2))
      return true
   }catch(e){
      displayNodeError(e)
      return false
   }
}


module.exports = {
   readTemplateFile,
   deleteTemplateFile,
   updateTemplateFile
}
```

Додаток Ґ

Лістинг конфігураційних файлів

.gitignore

```
/node_modules
package-lock.json
config.json
/dev
```

config.json

```
{
   "publicKey":"",
   "privateKey":""
}
```

index.js

```
#!/usr/bin/env node
require("./bin/commands/commands.js");
```

package.json

```
{
  "name": "diploma",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "bin": {
    "tt": "./index.js"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/SimpleSquirrel/diploma.git"
  },
  "author": "SimpleSquirrel",
  "license": "ISC",
  "bugs": {
```

```
    "url": "https://github.com/SimpleSquirrel/diploma/issues"
  },
  "homepage": "https://github.com/SimpleSquirrel/diploma#readme",
  "dependencies": {
    "axios": "^0.26.0",
    "binance-api-node": "^0.11.31",
    "chalk": "^4.1.2",
    "commander": "^9.0.0",
    "conf": "^10.1.1",
    "easy-table": "^1.2.0",
    "jimp": "^0.16.1",
    "rimraf": "^3.0.2"
  }
}
```