

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
УПРАВЛІННЯ БІРЖОВИМИ ГАМАНЦІЯМИ З МОЖЛИВІСТЮ
ПОКУПКИ І ПРОДАЖУ АКТИВІВ. SOFTWARE DEVELOPMENT
FOR MANAGING EXCHANGE WALLETS WITH THE OPTION OF
ASSETS SALE AND PURCHASE

Виконав: студент 4 курсу, групи КНТ-127
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Будюк В.А.

(прізвище та ініціали)

Керівник Федорончак Т.В.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я

НА ДИПЛОМНИЙ РОБОТУ СТУДЕНТА

Будюка Владислава Андрійовича
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення для управління біржовими гаманцями з можливістю покупки і продажу активів. Software Development for Managing Exchange Wallets with the Option of Assets Sale and Purchase,
 керівник роботи Федорончак Тетяна Василівна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103
2. Строк подання студентом проекту (роботи) 31 травня 2020 року
3. Вихідні дані до роботи: рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	Прийняв виконане завдання
1-5 Основна частина	Федорончак Т.В., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “28” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис) Будюк В.А.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Федорончак Т.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
94 с., 34 рис., 3 дод., 10 джерел.

PYTHON, API, PANDAS, PYQT, SQLITE3, SCHEMA.

Об'єкт дослідження - процес реалізації програмного забезпечення для управління біржовими гаманцями з можливістю покупки і продажу активів.

Предмет дослідження - методи розробки програмного забезпечення з використанням Python, PyQt5, Sqlite3, Schema.

Мета роботи - розробка терміналу для торгівлі з реалізацією архітектури "Clean Architecture".

Матеріали, методи та технічні засоби: Python, sqlite3, pandas, pyqt5, schema.

Висновок: розроблено термінал для управління біржовими гаманцями з можливістю продажу та купівлі активів, слідкуванням за статусом заявок.

Галузь використання – розробка терміналу для торгівлі на біржі криптовалют.

ABSTRACT

Explanatory note to the diploma qualification work of the bachelor: 94 pp., 34 figs., 3 appendices, 10 sources.

PYTHON, API, PANDAS, PYQT, SQLITE3, SCHEMA.

The object of research is the process of implementation of software for managing exchange wallets with the ability to buy and sell assets.

The subject of research - methods of software development using Python, Pyqt5, Sqlite3, Schema.

The purpose of the work is to develop a terminal for trade with the implementation of the architecture "Clean Architecture".

Materials, methods and technical means: Python, sqlite3, pandas, pyqt5, schema.

Conclusion: developed a terminal to manage exchange wallets with the ability to sell and buy assets, monitoring the status of applications.

Field of use - development of a terminal for trading on cryptocurrency exchanges.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Загальні відомості.....	10
1.2 Криптовалюта	12
1.3 Криптовалютна біржа.....	14
1.4. Основні компоненти бірж.....	20
1.5. Rest API.....	24
1.6 Висновки за розділом 1	27
2 ОПИС ПРОГРАМИ	29
2.1. Вибір мови програмування та технологій.....	29
2.2 Архітектура програми	31
2.3 Висновки за 2 розділом.....	36
3 РОЗРОБКА ПРОГРАМИ	38
3.1 Опис вимог	38
3.2 Опис архітектури	38
3.3 Опис заявок	39
3.4 Опис варіантів використання	41
3.5 Визначення схеми бази даних	43
3.6 Висновки за розділом 3	45
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ПРОГРАМИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ.....	46

4.1 Опис застосування.....	46
4.2 Робота з додатком.....	46
4.3 Таблиця заявок.....	47
4.4 Інтерфейс створення заявок.....	47
4.5 Додавання та видалення акаунтів	49
4.6 Поле для системних повідомлень	50
ВИСНОВКИ	51
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	52
ДОДАТОК А Технічне завдання.....	53
ДОДАТОК Б Текст програми	58
ДОДАТОК В Слайди презентації.....	87

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Inteface;

БД – База даних.

ВСТУП

Біржі функціонують по всій планеті. Залежно від того, з чим проводяться торги, відбувається їх класифікація. Вони бувають різні, але, не дивлячись на велику кількість відповідей на питання "як працює біржа", один з відповідей залишається незмінним - на них можна торгувати і заробляти.

Те, що відбувається на біржах, сильно впливає на економіку всіх країн.

Власне, біржа - це майданчик, місце, де укладаються торгові угоди. Перша біржа організувалася в місті Брюгге в 15 столітті, вона була торговою. Операції проводилися на площі біля будинку багатого громадянина ван дер Бюрс. Від цього імені і пішла назва Борса - біржа.

Біржі поділяються на валютні, товарні та фондові. Завдяки прогресу і розвитку світової економіки, до них приєдналися ще два види: криптовалюта і Форекс біржа.

З розвитком технологій біржі перейшли в Інтернет і почали надавати доступ до торгівлі через API. Одразу з цим трейдери почали розробляти автоматизовані системи торгівлі. Вони допомагають у торгівлі, незамінні при швидкісному багаточисельному формуванні заявок, стопів, у процесі контролювання ризиків, здійснення швидких обчислень, зберігають час, обмежують збитки.

Біржі дозволяє будь-якій людині інвестувати гроші у вигідні проекти по всьому світу, не змінюючи звичного способу життя і не витрачаючи свій особистий час на самостійне вивчення ринку. Такий спосіб інвестування називають довірче управління. Даний спосіб передбачає передачу трейдеру акаунта з грошовими коштами. Саме проблему управління переліком акаунтів вирішує моє програмне забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні відомості

Біржа - організаційна форма оптової, міжнародної торгівлі масовими товарами, що мають стійкі та чіткі якісні параметри або систематичний операцій з купівлі-продажу інших паперів, золота, валюти, криптовалюти. Як тільки біржова торгівля перейшла у комп'ютерну еру так відразу з'явилися різні види автоматизації торгівлі, які спочатку були доступні тільки тим хто мав кошти на комп'ютері, але з розвитком технологій поріг входу знижувався. Розвиток Інтернету та поява API зробила торгівлю та розробку програмного забезпечення доступною кожному [1].

З часом виділилися декілька видів автоматизацій:

- торгові радники, які надають певну інформацію і не роблять ніяких операцій на ринку [2];
- автоматизовані торгові системи, які повноцінно торгують на ринку [3];
- термінали, які надають доступ до ринку.

Все біржі можливо класифікувати в залежності від цінностей, які використовуються при здійсненні біржових операцій. Види сучасних бірж ґрунтуються на видах цінностей. Як об'єкти торгових операцій можуть виступати:

- корисні ресурси - газ, нафта, руди;
- цінні папери та акції різних підприємств і банків;
- банківські чеки і валюта різних держав;
- продукція повсякденного споживчого попиту - сировина, харчові продукти;
- інформаційні технології;
- електронні грошові кошти (криптовалюта);
- об'єкти інтелектуальної власності;
- дорогоцінні метали і дорогоцінне каміння (золото, алмази, платина);
- грошові боргові зобов'язання (облігації).

На валютних біржах проводяться операції по вільному продажу і купівлі валют різних держав на основі котирувань - курсових співвідношень. Валютна біржа відноситься до категорії найбільш ліквідних бірж. Співвідношення курсів валют на валютній біржі багато в чому залежать від економічного становища в країнах-емітентах і є відображенням купівельної спроможності валют. На валютних біржах відбувається процес встановлення реального ринкового курсу тієї чи іншої національної та іноземної валюти. Всі валютні біржі є юридичними особами і важливими елементами інфраструктури валютного ринку.

Товарні біржі являють собою ринки, що функціонують на регулярній основі, на яких відбувається оптовий продаж товарів з певними характеристиками. На товарній біржі діє принцип чистої конкуренції, а усі товари є легкозамінними в силу своєї однорідності. На даний момент існує близько 100 видів біржових товарів, які відносяться до різних груп - сировина (енергетичне або промислове), метали, текстильна сировина, харчові продукти та інше. Залежно від реалізованого товару товарні біржі можуть бути спеціалізованими (вузька товарна спрямованість) або універсальними.

На фондових біржах здійснюється торгівля цінними паперами, тому вони є найважливішим фінансовим інститутом, що підтримує ринок цінних паперів. Фондові біржі надають місце як для первинного продажу цінних паперів, так і для їх подальшого перепродажу. За рахунок торгів на фондових біржах відбувається природне регулювання ринку цінних паперів і забезпечення ліквідності фінансових вкладень. Списки найбільших фондових бірж регулярно оновлюються залежно від їх ринкової капіталізації. У торгах на фондовій біржі можуть брати участь і здійснювати операції як дилери, так і брокери.

Біржа криптовалюта - веб-служба, яка надає клієнтам послуги з обміну цифрових валют на інші активи, включаючи національні валюти та інші цифрові валюти. Подібні служби зазвичай надають виключно онлайн-сервіси, здійснюючи всі операції тільки в електронній формі. Але можуть пропонуватися і традиційні методи обміну валют. Як правило, за операції стягуються комісійні.

Біржі криптовалют відрізняються від усіх інших низким порогом входу. Для торгівлі на цьому типі бірж не потрібно надавати ніяких документів чи отримувати дозвіл, доступ до торгівлі надається одразу після реєстрації, яка виглядає як будь-яка інша. Також спрощена система доступу до автоматизації торгівлі, потрібно тільки створити API-ключ і ви можете приступати до автоматизації своїх задач.

Біржа праці являє собою державну посередницьку структуру між найманою робочою силою та роботодавцями. Біржа праці є механізмом, що діє на ринку праці і регулює певні його процеси. За допомогою біржі праці держава впливає на ситуацію на ринку праці безробіття і сприяє зайнятості осіб, які опинилися тимчасово безробітними. Важливим елементом в діяльності біржі є вивчення попиту на робочу силу та її структури, а також аналіз наявного на ринку пропозиції. Також до функцій біржі праці відноситься ведення статистики і виплата допомоги по безробіттю.

1.2 Криптовалюта

Для більш глибокого розуміння чим є криптовалютна біржа потрібно мати загальне уявлення про криптовалюту. Отже криптовалюта - різновид цифрової валюти, облік внутрішніх розрахункових одиниць якої забезпечує децентралізована платіжна система (немає внутрішнього і зовнішнього адміністратора або будь-якого його аналога), що працює в повністю автоматичному режимі [4]. Сама по собі криптовалюта не має якоїсь особливої матеріальної або електронної форми - це просто число, що позначає кількість даних розрахункових одиниць, яке записується у відповідній позиції інформаційного пакета протоколу передачі даних і часто навіть не піддається шифруванню, як і вся інша інформація про транзакції між адресами системи.

Криптографічні методи задіяні в механізмах генерації адреси та перевірки повноважень на операції з ним (цифровий підпис на основі системи з відкритим ключем, розпорядження є виключно власникові відповідного даною адресою

секретного ключа), а також формування пакета транзакцій і його взаємозв'язок з іншими пакетами (послідовне хешування, яке унеможливорює зміну інформації про кількість криптовалюти). При цьому в системі немає ніякої інформації про власників адрес або про факт створення адреси (адресу можна генерувати повністю автономно, навіть не підключаючись до мережі і нічого не повідомляючи в мережу в подальшому) - тобто немає механізму переконатися, що адреса одержувача дійсно існує або що ключ доступу до нього не втрачено. Відсутність інформації про власника є основою (але не обмежується тільки цим) анонімності учасників транзакцій. За своїм економічним умовам і наслідків платежі криптовалюта більш схожі на платежі готівкою, ніж на варіанти безготівкових платежів, хоча криптовалюта розробляються в першу чергу для дистанційних покупок (наприклад, через Інтернет).

Платіж (передача криптовалюти між адресами) відбувається без посередників і незворотній - немає механізму скасування підтвердженої операції, включаючи випадки, коли платіж був відправлений на помилковий або неіснуючу адресу, або коли транзакцію зробили сторонні особи, яким став відомий закритий ключ. Криптовалюта ні на конкретну адресу, ні в цілому ніхто не може заблокувати (заарештувати), навіть тимчасово, вона завжди в розпорядженні власника закритого ключа до даною адресою. Правда, технологія мульти підпису дозволяє добровільно залучити третю сторону (арбітра) і реалізувати «оборотні транзакції», які можуть відбуватися проти волі однієї зі сторін. Більш складні умови (розумні контракти) реалізуються за допомогою спеціальних сценарних мов. Властива електронних платежів проблема подвійного витрачання вирішується використанням таких технологій як блокчейн, спрямований ациклічний граф, консенсусний реєстр (ledger) і ін. Інформація про транзакції зазвичай не шифрується і доступна у відкритому вигляді без реєстрації в системі.

Правила формування нової кількості криптовалюта (емісія) спочатку встановлюються протоколом. Зазвичай вони носять лотерейний характер, з впливом на ймовірність виграшу різних факторів - швидкості рішення

встановленої задачі (Майнінг) або розміру володіння встановленим ресурсом (форжинг). У ряді випадків частина або весь заявлений обсяг криптовалюта спочатку формується і розподіляється організаторами по підписці (ICO). Зазвичай використовується тільки одна технологія, але в деяких криптовалюта використовують комбінації з них. Про економічну сутність і юридичного статусу криптовалюта ведуться дискусії. У різних країнах криптовалюта розглядаються як платіжний засіб, специфічний товар, електронний актив, можуть мати обмеження в обороті (наприклад, заборона операцій з ними для банківських установ).

Узагалі криптовалюта створювалась як альтернативна банківській системі, у якій людина не має повного контролю над своїми грошовими одиницями. Для того щоб перевести великий об'єм грошей у традиційній банківській системі потрібно багато часу якого не потребує криптовалюта, час на переведення криптовалюти залежить від протоколу криптовалюти у якій здійснюється переведення для Bitcoin це від двадцяти до шестидесяти хвилин, а для Dogecoin це одна хвилина і цей час не залежить від кількості. Людина може втратити свою криптовалюту тільки якщо забуде чи втратить дані для доступу до свого криптовалютного гаманця і їй ніхто не зможе допомогти. Децентралізованість є основною відмінністю криптовалюти від традиційної банківської системи.

1.3 Криптовалютна біржа

Після випуску офіційного документа біткойнов в жовтні 2008 року і подальшого Genesis Block в січні 2009 року, крім Майнінг, єдиним способом придбання біткоінів в той час була торгівля на форумах або IRC, що вимагає великої довіри з боку іншої сторони.

В результаті всього лише трохи більше року по тому в березні 2010 року була запущена перша криптовалютная біржа. Нині неіснуюча і маловідома платформа отримала назву bitcoinmarket.com відповідно до пропозиції

«dwdollar» на форумі Bitcointalk про створення перший реальний ринок, на якому люди купують і продають біткойни один з одним. Ранні користувачі біткойнов, такі як NLS, запропонували корисну систему ціноутворення в якості відправної точки - на основі потреби в енергії для Майнінг, з котируваннями близько 1500 BTC за 1 долар в той час. Bitcoinmarket.com(рисунок 1.1), однак, надав набагато більший загальний консенсус щодо вартості біткойнов, представивши ціну при запуску близько 0,003 долара США, яка піднялася до 0,05 долара США влітку того ж року, що приблизно відповідає тому, до чого може повернутися будь-якої цінової графік.

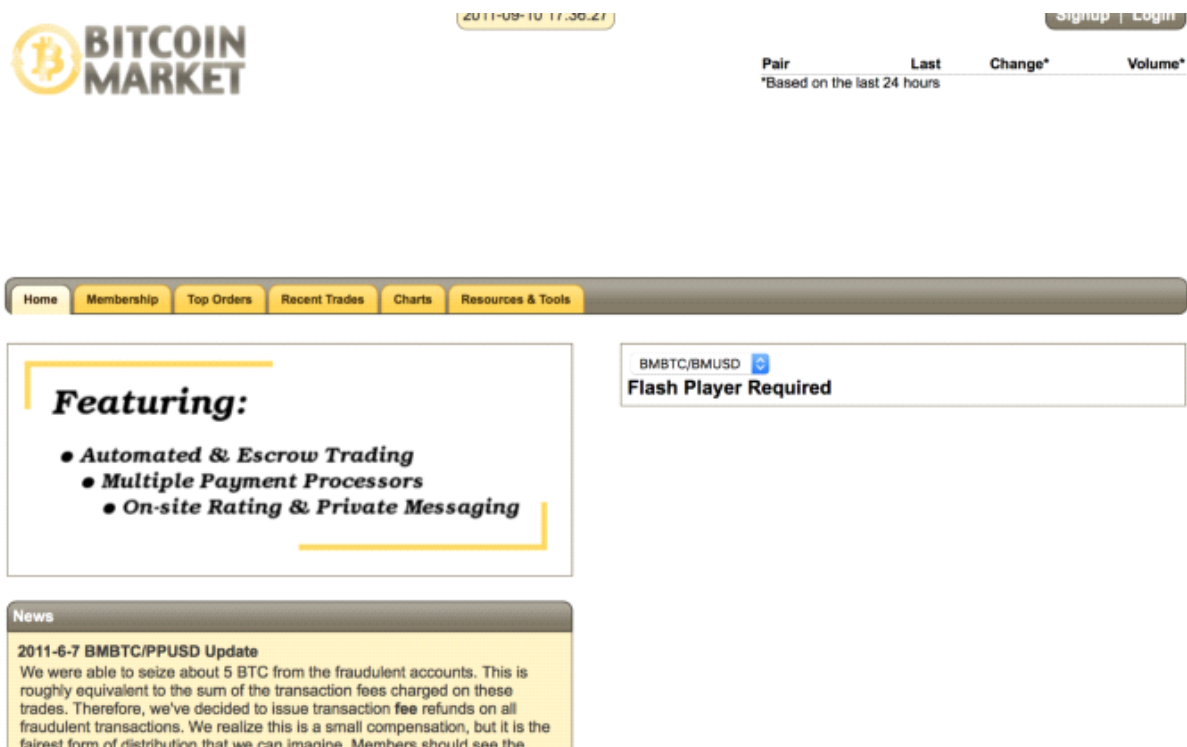


Рисунок 1.1 - Головна сторінка сайту Bitcoin Market

Платформа спочатку використовувала PayPal для обміну фіатних грошей, але була видалена через збільшення числа шахрайських операцій в червні 2011 року. До цього часу ринок біткойнов набрав популярність, і тепер ціна становить \$ 23,99. Bitcoinmarket.com виконав своє завдання і став удосконаленням в порівнянні з обміном на форумах, але тепер щосили намагався

впоратися зі зростаючим числом біткойнов, і його незабаром витіснив сумно відомий бренд.

У 2010 році Mt. Gox (рисунок 1.2) розроблявся Джейдом МакКалєб (зараз працює в Stellar Development Foundation), і за кілька коротких років 70% всіх угод з біткойнов здійснювалося через його платформу.



Рисунок 1.2 - Головна сторінка сайту Mt.Gox

Спочатку доменне ім'я mtgox.com (Magic: The Gathering Online eXchange) було придбано в 2007 році МакКалєб для торгового майданчика Magic: The Gathering cards (цифровий колекційної карткової гри). Маючи досвід програмування і інтерес до біткойнов, МакКалєб пізніше перепрофілював сайт, щоб забезпечити функціональність обміну біткоіну на долари США, яка потім стала популярною, оскільки вона завжди була надійно онлайн, автоматизована і швидше. Однак у МакКалєб були інтереси в іншому місці, і він не хотів керувати сайтом день у день. В результаті всього за кілька місяців він продав бізнес Марку

Карпелесу (він же Magical Tux), хоча залишався міноритарним власником.

Хоча зломи завжди були проблемою, МакКалєб посміявся над загрозою, і вплив користувачів показав, що вони готові піти на ризик. Скарги на погане управління і кошти, яких бракує, подавалися у Mt.Gox на протязом більшої частини життя платформи. Дійсно, коли МакКалєб продав Карпелесу платформу, затверджується, що 80 000 біткоїнів вже зникли. Потім, в червні 2011 року, стався перший злом, в результаті якого було видалено 2650 BTC. В результаті було посилено заходи безпеки, і, незважаючи на зникнення біткоїнів і зломи, Mt. Gox зарекомендувала себе як найбільша в світі біткойн-біржа. Однак за лаштунками ситуація була зовсім іншою. Передбачуване відсутність компетентної організації з боку керівництва (мається на увазі Карпелес), проблеми з кодуванням веб-сайтів і слабкі заходи безпеки - ось основні недоліки, про які згадували колишні співробітники. Приблизно в цей же час почалися судові суперечки, після чого Міністерство внутрішньої безпеки США звинуватило їх (через дочірню компанію) в діяльності в якості незареєстрованого перекладача грошей. В результаті уряд США отримав понад 5 мільйонів доларів з банківських рахунків компанії, що згодом призвело до припинення зняття коштів в доларах і тривалих затримок. Це було погано, але насправді проблема була набагато гірше.

На початку лютого 2014 року всі операції по зняттю коштів були зупинені, щоб «отримати чітке технічне уявлення про валютне процесі». Кілька тижнів потому, незважаючи на попередні запевнення Mt. Gox і впливової особи, такої як Роджер Вер, біржа призупинила всі торги, веб-сайт відключився, а компанія подала заяву про захист від банкрутства. Виявляється, вони були жертвою злomu триваючого протязом багатьох років, починаючи з кінця 2011 року і в результаті чого з їх гарячих гаманців було викрадено в цілому 844 408 біткоїннов.

Користувачі були шоковані, хоча б тому, що керівництво Mt. Gox, очевидно, не знало, що така величезна кількість пропала безвісти протязом стількох років. Хоча Карпелес пізніше був звинувачений в шахрайстві і розтраті

(хоча і не був безпосередньо пов'язаний з крадіжкою), наслідки цього злому і наступний за ним судовий позов все ще тривають, через роки. Хоча деяка віддача для користувачів імовірна, можливо, навіть в розмірі вартості долара, втраченого в той час, з огляду на зростання біткойнов в проміжний період, немає ніяких шансів, що користувачі повернуть всі втрачені суми BTC. Це послужило болючим уроком, тепер вбудованим в психіку ранніх біткойн холдерів і проповідує новим учасникам, що, ви не володієте ключами до гаманців, монети є вашими тільки на папері.

BitInstant(рисуюнок 1.3), заснований Гаретом Нельсоном і Чарлі Шрем, був ще однією помітною альтернативою в ті ж терміни, що фінансується братами Вінкельвосс і Еріком Вурхизом (який зараз очолює ShapeShift) в якості директора з маркетингу. Його спіткала така ж доля, знову в 2014 році, але за інших обставин після звинувачень у відмиванні грошей і подальшого вироку генеральному директору Чарлі Шрем.

BitInstant
providing instant transfers for the bitcoin economy

Partnership with **TRADEHILL**

Home | About us | What is it | How it works | Security | Fees and limits | API | Downloads

BitInstant is a new service that allows you to quickly and easily transfer funds into your online exchange account without delay.

How BitInstant works:

Deposit: You use the payment method of your choice.

BitInstant: We charge a small fee to transfer your funds.

Exchange: Funds are deposited INSTANTLY!

Liberty Reserve, MTGOX, paxum, DWOLLA, BitInstant, TRADEHILL

Our current rates and limits

Payment method	External exchange	Min fee	Max fee
MTGox Coupon	TradeHill	0.3900%	1.2900%
Liberty Reserve	TradeHill	2.1900%	2.3900%
Paxum	TradeHill	1.9900%	2.0900%
Dwolla	TradeHill	1.9900%	2.0900%

Payment method	External exchange	Min order	Max order	Daily volume limit
MTGox Coupon	TradeHill	\$20	\$1.00	\$4000
Liberty Reserve	TradeHill	\$20	\$1.00	\$4000
Paxum	TradeHill	\$20	\$1.00	\$4000
Dwolla	TradeHill	\$20	\$1.00	\$4000

See our [Fees page](#) for more information.
Quoted limits and fees are correct as of Tue Nov 8 11:40:50 2011.
However please note that we may change limits and fees or update them from time to time.
All prices quoted in USD

Get started!

You can fund your exchange account instantly using this form. No complex signup or account is needed - just enter your details and the funds will be deposited into your account within minutes - it's even easier than ordering a pizza!

Payment method Choose how to pay for this transaction: MTGox Coupon

Destination exchange Which online exchange do you wish to top up your account at? TradeHill

Exchange account Your username or account ID on the exchange
Your TradeHill email address - not your bitcoin address

Amount to send \$0.0
Amount to send including fees

Our commission 0.0%
The % fee we will charge

Amount credited \$0.0
The amount that will arrive in your account after fees

Send funds

Please also note that limits apply both to single transactions and to overall daily volume.
At present the daily volume limit per user is 4000USD.
Attempts to bypass these limits will result in a ban from our service.
Payment providers may impose external limits and debit/credit cards may impose fees separately from us.
It is your responsibility to check this with your card provider.
* Unfortunately not as easy as pizza in most cases.
Funds are not edible and for external use only.

Рисуюнок 1.3 - Головна сторінка сайту BitInstant

Зараз ситуація здається сприятливою для ринку криптовалюти і, зокрема, для бірж в більшості країн світу. Якщо ми подивимося на CoinMarketCap, ми побачимо, що на цьому веб-сайті вказано більше 13000 бірж, і їх кількість значно збільшується з плином часу.

Країни починають регулювати і встановлювати особливі правила роботи бірж. Південна Корея і Японія є найбільш прогресивними і відкритими для цих платформ, якщо вони дотримуються певних правил, спрямовані на прозорість і запобігання відмиванню грошей.

Швейцарія, Естонія і Мальта надають найбільше можливостей для створення бірж і інших проектів, заснованих на криптовалюта, тому що у них є чіткі закони, які необхідно дотримуватися і які спрямовані на захист користувачів. Зі свого боку, Сполучені Штати підтримують і контролюють біржі, хоча у них немає чітких правил на користь або проти, і позиція досить неоднозначна, незважаючи на те, що їм дозволено працювати.

Біржі Європи:

- Bitsamp: Це може бути перша біржа, котра розроблена і базується в Європі, зокрема, вона розташована в Люксембурзі і була заснована Нейц Кодрічем в 2011 році. Вона високо цінується і працює в усьому світі;

- LocalBitcoins: Ця біржа сьогодні є найвідомішою і однією з найбільш грошових рухів на європейському рівні. Він був заснований братами Ніколасом і Ієреміасом Кангас в Гельсінкі, Фінляндія, в 2012 році;

- CEX.io: Лондонське обмінне бюро в Великобританії, засноване в 2013 році Олександром Луцкевичем. У них бездоганний послужний список, і вони дійсно вимогливі, коли справа доходить до додавання нових криптовалют.

Біржі США:

- Kraken: Базується у Сан-Франциско, США, перша біржа, котра створена на Американській землі. Вона була заснована 28 липня 2011 Джессі Пауеллом. Біржа має авторитет та один з найбільших оберт BTC у день;

- Coinbase: Біржа в Сан-Франциско, США, заснована в червні 2012 року Брайаном Армстронгом, дозволяє легко купувати біткойни, Ethereum і інші

криптовалюта;

- Bittrex: Базована в Сіетлі, США, була заснована Біллом Шихару в 2013 році. Нещодавно компанія отримала велике оновлення інтерфейсу і додала пари опцій з доларами США;

- Poloniex: Розташований в Делавері, США і заснований Тристаном Д'Агосто в січні 2014 року, він є одним з найбільш вимогливих до додавання криптовалют і пропонує високі заходи безпеки для захисту користувачів.

Біржі Азії:

- Bitfinex: Базована в Гонконгу, вона була заснована в 2012 році Рафаелем Ніколлен і Джанкарло Девасіні і відрізняється своєю надійністю і безпекою і однією з перших вийшла на ринок. Це другий за обсягом рухів біткойнов;

- OKEx: Заснована в 2013 році Стар Сюй з Пекіна, Китай. Біржа є третьою за обсягом обміну BTC і першої, серед іншого, за Ethereum і EOS. Одна з найважливіших бірж на сьогоднішній день;

- Bitflyer: Це найважливіша біржа в Японії, заснована 9 січня 2014 року. Вона відповідає японським нормам щодо прозорості та безпеки;

- Binance: Вона народилася в 2017 році з рук Чанпен Чжао і базується в Шанхаї, Китай. На сьогоднішній день це найбільший обсяг біткойнов, який з кожним днем набуває все більшого значення.

Усі вищеперелічені біржі надають доступ до торгівлі табору статистики через своє API.

1.4. Основні компоненти бірж

Будь-яка біржа має кілька основних компонентів:

- графік;
- заявки на купівлю чи продаж;
- історію заявок;
- обсяг торгів.

Графік(рисунок 1.4) на біржах показує зміну ціни на криптовалюту в

плині часу. По суті графік(рисунок 1.4) є графічним представленням зміни спреда. Спред - різниця між кращими цінами ордерів на продаж і на покупку в один і той же момент часу. Зазвичай сам графік представлений у вигляді японських свічок. Подібні графіки потрібно вміти аналізувати.



Рисунок 1.4 - Графік біржі Binance

Японська свічка - це спосіб представлення руху ціни за певний період часу. Японські свічки дають корисну інформацію, наприклад, про настрої ринку або можливих розворотах на ринку, представляючи рух ціни особливим графічним чином. 1 свічка відображає рух ціни (вгору або вниз) за певний проміжок часу (на деяких біржах його можна налаштовувати), зазвичай це 10, 15 або 30 хвилин. Отже для формування 1 свічки потрібно 10, 15 або 30 хвилин. Потім починає формуватися наступна свічка. І так далі. Японська свічка складається з декількох компонентів:

- тіло свічки. Широка частина свічки називається тілом, воно являє діапазон цін між цінами відкриття торгів і цінами закриття за проміжок часу. Якщо тіло свічки чорного (або червоного) кольору, то це означає, що ціна закриття була менша за ціну відкриття і свічка, в такому випадку називається

ведмежою (тобто показує падіння ціни). Якщо тіло свічки білого (або зеленого) кольору, значить ціна закриття більше ціни відкриття, така свічка називається бичачою (тобто зростання ціни);

- лінії над тілом і під тілом свічки називаються тінями. Вони відображають максимальну і мінімальну ціну за певний проміжок часу. Якщо у свічки немає верхньої тіні, кажуть, що у неї зрізана вершина; якщо відсутня нижня тінь - зрізано підставу. Японські свічки з малим розміром тіла називаються "вовчками", з відсутнім тілом (ціна закриття дорівнює ціні відкриття або майже дорівнює їй) - називаються "доджі".

Це таблиці, де показуються найближчі продавці і покупці. Зазвичай це називають склянка(рисунок 1.5). Тут вказані лише виставлені заявки на покупку / продаж, які можуть спрацювати, якщо хтось вирішить купити / продати за такою ціною. Таблиця ділиться на 3 стовпці - ціна, за яку готові купити / продати, сума всіх кількості кріптомонет, які люди готові купити / продати за цією ціною і сума в доларах дорівнює кількості проданих / купуються монет по даній ціні.

Price(USDT)	Amount(BTC)	Total
62200	0.000263	16
51000	18.609018	949,060
45000	149.437371	6,724,682
44800	0.006412	287
41900	0.001566	66
41100	0.011153	458
39000	95.052545	3,707,049
38100	0.003397	129
38000	203.463609	7,731,617
37900	77.839917	2,950,133
37800	0.003000	113
37700	0.333801	12,566
37600	0.005327	200
37500	1.489258	55,754
37400	13.091408	488,946
37300	37.662639	1,402,451
37200	41.696439	1,548,662
37100	3.845490	142,658
37,093.00 ↑ \$37,093.00		More 
37000	83.943272	3,109,254
36900	18.434080	681,094
36800	38.375556	1,413,586
36700	76.163316	2,800,159
36600	3.079779	112,723
36500	84.788441	3,094,811
36200	0.000321	12
36000	31.458150	1,132,494
35900	0.010000	360
35700	9.949581	355,200
35600	0.000528	19
35500	0.003969	141
35300	0.000581	21
34700	0.000305	11
34500	52.265882	1,803,173
34400	1.547830	53,311
34000	132.740812	4,513,188

Рисунок 1.5 - Складання заявок біржі Binance

Складання(рисунок 1.5) заявок формується таким чином: щоб здійснити операцію на біржі, необхідно через форму купівлі / продажу(рисунок 1.6) відправити заявку, в якій вказуються основні параметри майбутньої угоди - ціна, кількість. Після того, як заявка надходить в обробку на біржу, починається

пошук зустрічній заявки і якщо така заявка знаходиться, то угода відбувається (наприклад, для продажу шукаються заявки на покупку з ціною рівною або більшою, ніж зазначено у вашій заявці). Якщо зустрічна заявка не знаходиться, ваша заявка переноситься в склянку заявок і записується або окремо (якщо за такою ціною більше ніхто не хоче купити / продати), або в заявку з такою ж ціною.

The image shows a screenshot of the Binance trading interface. It features two main sections: 'Buy BTC' on the left and 'Sell BTC' on the right. Both sections have a 'Price' field set to 37061.36 USDT, an 'Amount' field set to BTC, and a 'Total' field set to USDT. The 'Buy BTC' button is green, and the 'Sell BTC' button is red. The interface also includes a 'Limit' tab, a 'Market' tab, and a 'Stop-limit' dropdown menu. The 'Buy with' dropdown is set to EUR.

Рисунок 1.6 - Інтерфейс створення заявок біржі Binance

1.5. Rest API

У написанні API використовується методологія REST, яка зарекомендувала себе. Одже REST це архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST є узгодженим набором обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи. У певних випадках (інтернет-магазини, пошукові системи, інші системи, засновані на даних) це призводить до підвищення продуктивності і спрощення архітектури.

У широкому сенсі компоненти в REST взаємодіють на зразок взаємодії клієнтів і серверів у Всесвітній павутині. REST є альтернативою RPC. API біржі Binance створена за REST методологією.

В мережі Інтернет виклик віддаленої процедури може являти собою звичайний HTTP-запит (зазвичай GET або POST; такий запит називають «REST-запит»), а необхідні дані передаються в якості параметрів запиту.

Для веб-служб, побудованих з урахуванням REST (тобто не порушують накладаються їм обмежень), застосовують термін «RESTful».

На відміну від веб-сервісів (веб-служб) на основі SOAP, не існує «офіційного» стандарту для RESTful веб-API. Справа в тому, що REST є архітектурним стилем, в той час як SOAP є протоколом. Незважаючи на те, що REST не є стандартом сам по собі, більшість RESTful-реалізацій використовують такі стандарти, як HTTP, URL, JSON і, рідше, XML.

Властивості архітектури:

- продуктивність - взаємодія компонентів системи може бути домінуючим фактором продуктивності і ефективності мережі з точки зору користувача;
- масштабованість для забезпечення великої кількості компонентів і взаємодій компонентів;
- простота уніфікованого інтерфейсу;
- відкритість компонентів до можливих змін для задоволення мінливих потреб навіть при працюючому додатку;
- прозорість зв'язків між компонентами системи для сервісних служб;
- переносимість компонентів системи шляхом переміщення програмного коду разом з даними;
- надійність, що виражається в стійкості до відмов на рівні системи при наявності відмов окремих компонентів, з'єднань або даних.

Існує шість обов'язкових обмежень для побудови розподілених REST-додатків по Філдингу.

Виконання цих обмежень обов'язково для REST-систем. Накладаються обмеження визначають роботу сервера в тому, як він може обробляти і відповідати на запити клієнтів. Діючи в рамках цих обмежень, система набуває такі бажані властивості як продуктивність, масштабованість, простота, здатність до змін, переносимість, відстеження та надійність.

Якщо сервіс-додаток порушує будь-яке з цих обмежувальних умов, дану систему можна вважати REST-системою.

Обов'язковими умовами-обмеженнями є:

1. модель клієнт-сервер. Першим обмеженням, які можуть застосовуватися до гібридної моделі, є приведення архітектури до моделі клієнт-сервер. Розмежування потреб є принципом, що лежить в основі даного накладається обмеження. Відділення потреби інтерфейсу клієнта від потреб сервера, що зберігає дані, підвищує переносимість коду клієнтського інтерфейсу на інші платформи, а спрощення серверної частини покращує масштабованість. Найбільше ж вплив на всесвітню павутину, мабуть, має саме розмежування, яке дозволяє окремим частинам розвиватися незалежно один від одного, підтримуючи потреби в розвитку інтернету з боку різних організацій;

2. відсутність стану. Протокол взаємодії між клієнтом і сервером вимагає дотримання наступного умови: в період між запитами клієнта ніяка інформація про стан клієнта на сервері не зберігається (Stateless protocol або «протокол без збереження стану»). Всі запити від клієнта повинні бути складені так, щоб сервер отримав всю необхідну інформацію для виконання запиту. Стан сесії при цьому зберігається на стороні клієнта. Інформація про стан сесії може бути передана сервером будь-якому іншому сервісу (наприклад, в службу бази даних) для підтримки стійкого стану, наприклад, на період встановлення аутентифікації. Клієнт ініціює відправку запитів, коли він готовий (виникає необхідність) перейти в новий стан. Під час обробки клієнтських запитів вважається, що клієнт знаходиться в перехідному стані. Кожний окремий стан додатку представлено зв'язками, які можуть бути задіяні при наступному зверненні клієнта;

3. кешування. Як і у Всесвітній павутині, клієнти, а також проміжні вузли, можуть виконувати кешування відповідей сервера. Відповіді сервера, в свою чергу, повинні мати явне або неявне позначення як кешувальні або некешірльні з метою запобігання отримання клієнтами застарілих або невірних даних у відповідь на наступні запити. Правильне використання кешування здатне

частково або повністю усунути деякі клієнт-серверні взаємодії, ще більше підвищуючи продуктивність і масштабованість системи;

4. одноманітність інтерфейсу. Наявність уніфікованого інтерфейсу є фундаментальним вимогою дизайну REST-сервісів. Уніфіковані інтерфейси дозволяють кожному з сервісів розвиватися незалежно. До уніфікованим інтерфейсів пред'являються наступні обмежувальних умови:

1. ідентифікація ресурсів. Всі ресурси ідентифікуються в запитах, наприклад, з використанням URI в інтернет-системах. Ресурси концептуально відокремлені від уявлень, які повертаються клієнтам. Наприклад, сервер може надсилати дані з бази даних у вигляді HTML, XML або JSON, жоден з яких не є типом зберігання всередині сервера;

2. маніпуляція ресурсами через уявлення. Якщо клієнт зберігає уявлення ресурсу, включаючи метадані - він володіє достатньою інформацією для модифікації або видалення ресурсу;

3. «самоописуванні» повідомлення. Кожне повідомлення містить достатньо інформації, щоб зрозуміти, яким чином його обробляти. Наприклад, обробник повідомлення, необхідний для отримання даних, може бути зазначений у списку MIME-типів.

5. шари. Клієнт зазвичай не здатний точно визначити, взаємодіє він безпосередньо з сервером або ж з проміжним вузлом, в зв'язку з ієрархічною структурою мереж (маючи на увазі, що така структура утворює шари). Застосування проміжних серверів здатне підвищити масштабованість за рахунок балансування навантаження і розподіленого кешування. Проміжні вузли також можуть підкорятися політиці безпеки з метою забезпечення конфіденційності інформації.

1.6 Висновки за розділом 1

Розвиток технологій покращував і спрощував життя людей в усьому світі від початку часів і до сьогодення. Так само технології змінили ринкові

відносини, перенісши торгівлю з ринку спочатку на телеграф, потім на телефон, а після і в Інтернет. Зараз ми спостерігаємо як технології, а саме Інтернет та блокчейн змінює ринкові відносини і економіку в черговий раз. Сатоши Накамото творець Біткоіну своїм дітищем створив альтернативу, яка нерегульована зовнішніми силами, не має представників, які своїми діями можуть впливати на економіку, він дав людям екосистему, в якій гроші належать самим людям, а не банкам, в якій немає обмежень на переказ між країнами, в якій ніхто не може заблокувати рахунок [5]. Цього б не сталося без нинішніх технологій, без Інтернету, комп'ютерів, який не так давно були доступні далеко не всім. Тепер я і будь-яка інша людина може отримати доступ до ринку, торгувати, автоматизувати торгові стратегії, створювати свої торгові термінали.

2 ОПИС ПРОГРАМИ

2.1. Вибір мови програмування та технологій

Для реалізації проекту було обрано мову програмування Python, фреймворк PyQt для реалізації графічного інтерфейсу, бібліотека python-binance для підключення до біржі Binance, бібліотека sqlite3 для підключення до бази даних.

Для розробки програми була вибрана мова програмування Python через те що, дана мова є повністю об'єктно-орієнтованою з динамічною і суворою типізацією, що полегшує написання програми тим що змінні можуть мати у собі посилання на значення різних типів та не потрібно заздалегідь визначати тип змінної, але така особливість потребує уважності тому що можливо передати змінну не з тим типом даних цю проблему вирішує type hinting/анотація типів, котра була реалізована за допомогою бібліотеки typing а потім додана у версії 3.9. Ця мова програмування має мінімалістичний синтаксис та особливість виділення блоків коду пробілами. Такі особливості роблять Python дуже легким для сприйняття. Також є деякі недоліки найбільш помітним серед них є швидкість виконання програм, я прийшов до Python після C++ і у початку моєї роботи з Python це було для мене проблемою. Проблема швидкості Python полягає у тому що він є інтерпретована, але це плата за динамічну типізацією. Мова програмування Python робота з пам'яттю автоматично.

Та тут виходить один з найбільших плюсів Python – це його бібліотеки, які у вирішують більшість проблем python. Стандартна бібліотека включає великий набір корисних функцій, починаючи від функціоналу для роботи з текстом і закінчуючи засобами для написання мережових додатків. Додаткові можливості, такі як математичне моделювання, робота з обладнанням, написання веб-додатків або розробка ігор, можуть реалізовуватися за допомогою великого кількості сторонніх бібліотек, а також інтеграцією бібліотек, написаних на C або C ++, при цьому і сам інтерпретатор Python може інтегруватися в проекти, написані на цих мовах(PyQt одна з них). Існує і спеціалізований

репозиторій програмного забезпечення, написаного на Python, - PyPI. Даний репозиторій надає кошти для простої установки пакетів в операційну систему і став стандартом де-факто для Python. Станом на 2019 рік у ньому міститься понад 175 тисяч пакетів. Використовуючи Python я впевнений, що у мене є інструментарій для рішення будь-якої проблеми. Я писав парсери даних, які отримують дані через API, використовуючи бібліотеки requests, json, schema, або через html сторінку, використовуючи beautiful soup, selenium, писав аналітичні програми, у котрих використовував pandas та numpy для очищення, агрегації та пошуку у великих датафреймах, а отримані дані виводив у виді графіків і гістограм за допомогою matplotlib, створював інтерфейси за допомогою PyQt у які безперешкодно інтегрував графіки з matplotlib.

Через це я його і обрав та не тільки я притримуюсь такої думки. Python став однією з найпопулярніших мов програмування, він використовується в аналізі даних, машинному навчанні, DevOps і веб-розробки, а також в інших сферах, включаючи розробку ігор. За рахунок читабельності, простого синтаксису і відсутності необхідності в компіляції мову добре підходить для навчання програмуванню, дозволяючи сконцентруватися на вивченні алгоритмів, концептів і парадигм. Налагодження ж і експериментування в значній мірі полегшуються тим фактом, що мова є інтерпретується. Застосовується мову багатьма великими компаніями, такими як Google або Facebook. Станом на квітень 2021 року Python займає третє місце в рейтингу TIOBE популярності мов програмування з показником 11,03%. «Мовою року» за версією TIOBE Python оголошувався в 2007, 2010, 2018 і 2020 року.

Вибір бази даних був обґрунтований тим що мій додаток має бути Desktop додатком . Ця база даних має просте підключення та не потребує підключення через ip і port. Sqlite3 є реляційною базою даних. Sqlite3 на python надає не так багато типів даних як інші бази даних, але їх достатньо.

PyQt - набір розширень графічного фреймворка Qt для мови програмування Python, виконаний у вигляді розширення Python. PyQt підтримує

увесь функціонал бібліотеки Qt. PyQt також має QtDesigner, котрий значно прискорює розробку графічного інтерфейсу.

PyQt є подієво-орієнтованим. Події викликають головним чином користувачем додатку. Однак, вони можуть бути викликані іншими засобами, наприклад підключенням до Інтернету, диспетчером вікон або таймером [6].

Schema - це бібліотека для перевірки структур даних Python, таких як ті, що отримані з конфігураційних файлів, форм, зовнішніх служб або аналізу командного рядка, перетворені з JSON або чогось іншого в типи даних Python.

Pandas - це бібліотека Python, що забезпечує швидкі, гнучкі та виразні структури даних, розроблені для полегшення та інтуїтивно зрозумілої роботи з „реляційними” або „позначеними” даними [7].

У роботі я використовував DataFrame які є двомірною структурою даних, яка мінлива за розміром та має потенційно неоднорідні табличні дані. Найкраще у ній це підтримка векторних операцій.

2.2 Архітектура програми

Вибираючи архітектуру керувався наступними критеріями та власним досвідом:

- простота та швидкість додавання нового функціоналу;
- простота та швидкість зміни існуючого функціоналу;
- масштабованість;
- простота підтримки;
- простота тестування.

На основі мого невеликого досвіду найкраще цим критеріям підходить чиста архітектура, автором якої є Роберт Мартін. Роберт Мартін має великий досвід у програмуванні, свій перший рядок коду він написав у 1964 році, а комерційного досвіду більше 40 років. Цю архітектуру він представив у книзі «Clean Architecture»(рисунок 2.1) вона є збірником його досвіду за багаторічну кар’єру [8].

Ціль будь-якої архітектури програмного забезпечення – зменшити трудовитрати на створення і підтримку програмного забезпечення системи. У архітектурі, яка представлена Робертом Мартіном (рисунк 2.1) економія часу досягається за рахунок чітко прописаних правил, дотримуючись яких можливо написати просту і зрозумілу архітектуру.

Чиста архітектура розбита на рівні. Між рівнями є кордони. Зовнішній рівень знає все про нижній рівень, нижній рівень не знає нічого про верхній рівень.

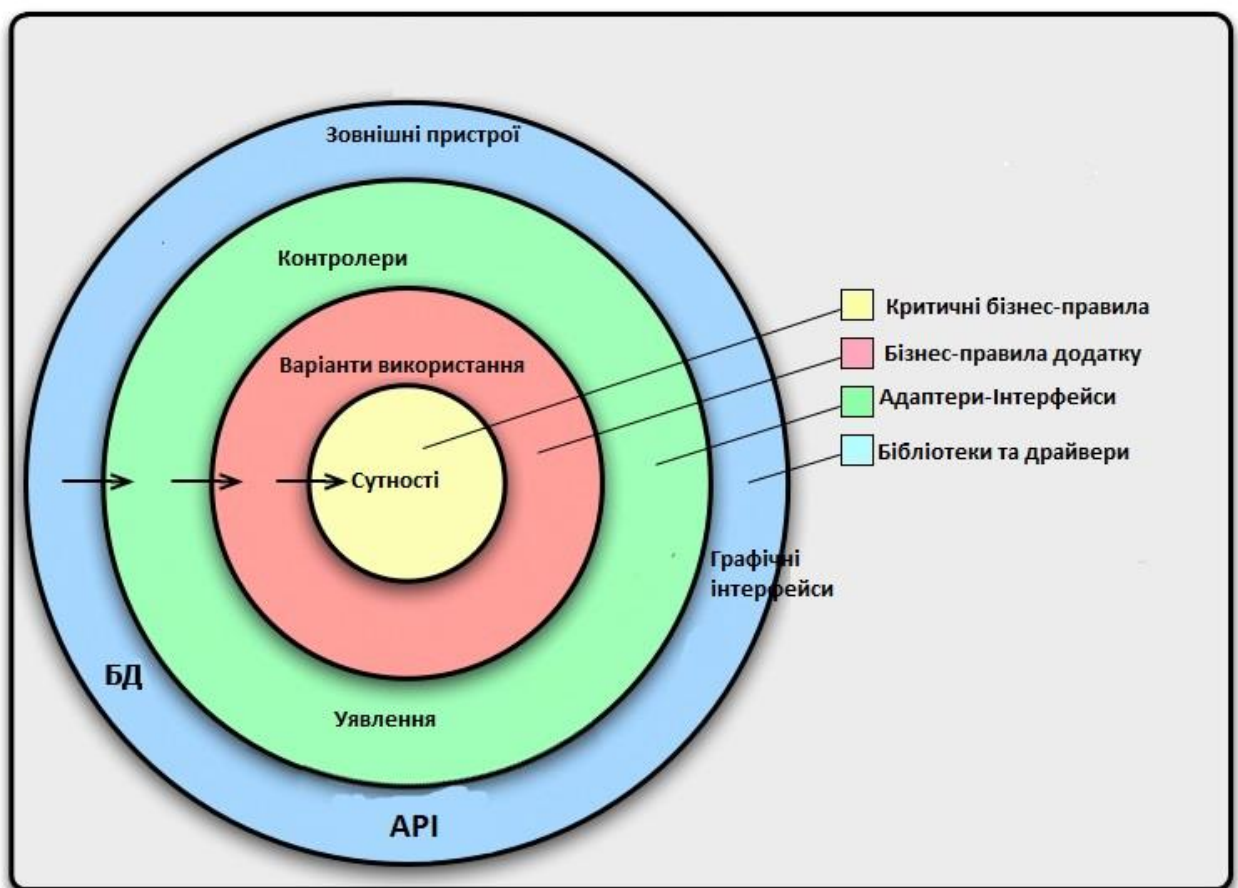


Рисунок 2.1 - Схема "Clean Architecture"

Основою будь-якої програми є бізнес-правила. Бізнес-правила - це правила та процедури, що роблять або економлять гроші незалежно від наявності або відсутності їх реалізації на комп'ютері. Вони роблять або економлять гроші навіть коли виконуються вручну [9]. Або бізнес-логіка - це реалізація предметної

області в інформаційній системі. Бізнес-логіці потрібні дані для роботи. Правила і дані нерозривно пов'язані один з одним і є відмінними кандидати на об'єднання в об'єкт, який називається сутність [9].

Бізнес-логіка є найціннішим що є в системі так як вона безпосередньо пов'язана з грошима. Тому вона повинна бути екранована від більшості змін в інфраструктурі додатку. У моєму додатку сутностями є:

- заявки, які реалізують біржові заявки;
- монети, які зберігають в собі вільну і заблоковану кількість монети;
- символи, які реалізують налаштування зміни ціни і кількості монети;
- акаунти, які зберігають в собі всі заявки і монети;
- менеджер акаунтів, який зберігає в собі всі акаунти і керує ними.

Вони екрановані від зовнішніх змін. Вони не знають який тип бази даних використовується або як і куди виводиться інформація або звідки вона приходить. Вони тільки відповідають предметній області і поведуться так як того вимагає предметна область.

Бізнес-логіка є ядром системи, яким керують бізнес-правила додатка. Бізнес-правила додатка - це бізнес-правила, які не мають сенсу без автоматизації. Варіант використання описує як використовується додаток. Наприклад, при створенні заявки потрібно виконати наступну послідовність дій:

- перевірити чи вистачає на Акаунті монети;
- якщо не вистачає, то відхилити створення Заявки;
- інакше перевірити правильність переданих параметрів Заявки;
- якщо хоч один параметр в неправильний, то відхилити створення Заявки.
- інакше створити заявку;
- записати її в базу даних;
- створити заявку на біржі;
- вивести заявку.

Варіанти використання визначають як і коли викликаються критичні бізнес-правила сутності. Варіанти використання керують діями суті [9]. Варіанти

використання не описують призначений для користувача інтерфейс, вони тільки визначають вхідні і вихідні дані. За варіанту використання неможливо визначити вид інтерфейсу [10].

Варіанти використання знають все про сутності, але суті не знають нічого про варіанти використання, які контролюють їх. Відповідно варіанти використання залежать від сутностей. Якщо будь-яка зміна буде в сутності, вона вплине на варіанти використання. Сутності взаємодіють з варіантами використання за допомогою принципу інверсії залежностей, який має кілька визначень:

- модулі верхніх рівнів не повинні залежати від модулів нижніх рівнів. Обидва типи модулів повинні залежати від абстракцій;
- абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

В даному випадку модулем верхньому рівня є бізнес-логіка так як вона далі від введення виведення. Дотримання принципу інверсії залежностей «змушує» реалізовувати високорівневі компоненти без вбудовування залежностей від конкретних низькорівневих класів, що, наприклад, сильно спрощує заміну використовуваних залежностей як по бізнес-вимогам, так і для цілей тестування.

Є два варіанти кордонів:

- це кордон, який перетинають в напрямку від нижнього рівня до верхнього;
- межа, яку перетинають в напрямку від верхнього рівня до нижнього.

Різниця між цими межами полягає в тому що при перетині кордону від нижнього рівня до верхнього, модуль нижнього рівня звертається безпосередньо до модуля верхнього рівня, а при перетині кордону від верхнього рівня до нижнього, модуль верхнього рівня звертається до модуля нижнього рівня через інтерфейс. У моїй програмі варіанти використання (модулі нижнього рівня) звертаються до сутностей (модулів верхнього рівня), а сутності до варіантів використання не звертаються(рисунок 2.2).

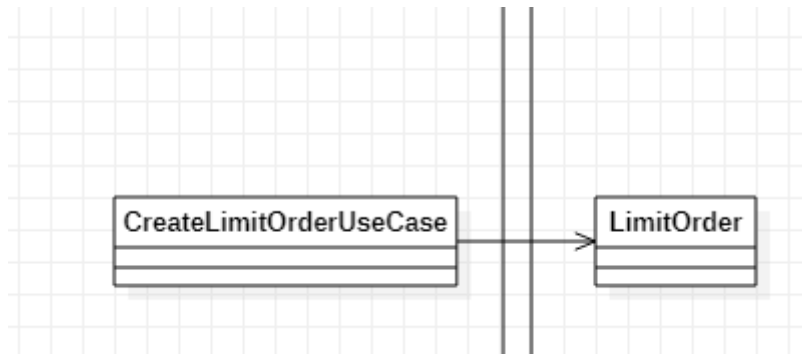


Рисунок 2.2 - Перетин кордону від верхнього рівня до нижнього

Приклад звернення модуля верхнього рівня до модуля нижнього рівня можна подивитися при зверненні варіанту використання, який використовує базу даних або API біржі. У варіанті використання "CreateLimitOrderUseCase", варіант використання звертається до об'єкта підключення до бази даних, який має свій інтерфейс, а саму конкретну реалізацію підключення варіант використання отримує зовні (рисунок 2.3). Такий спосіб підключення дозволяє змінювати варіанти підключення до бази даних без шкоди для варіанту використання. Підключення до API реалізовано через інтерфейс як і до графічного інтерфейсу.



Рисунок 2.3 - Перетин кордону від нижнього рівня до верхнього

Також такий спосіб підключення не зобов'язує використовувати якийсь конкретний спосіб підключення до біржі. Можна використовувати звичайні GET, POST запити або підключення через WebSocket. Для варіанту

використання це не матиме ніякого значення. Якщо реалізувати підключення до бази даних через інтерфейс, то він таке підключення дає можливість використовувати будь-який вид бази даних і змінювати їх в процесі розвитку програмного забезпечення без шкоди для варіантів використання все що потрібно буде змінити це реалізацію підключення. якщо реалізувати підключення до інтерфейсу через інтерфейс, то можливо використовувати різні види інтерфейсів і поміняти їх в процесі розвитку програмного забезпечення.

Єдине що залишилось без уваги це потік управління, який йде від контролера до варіанту використання і якщо є що виводити, то йде в презентер, який виводить інформацію. Таким чином контролер відповідає тільки за отримання сигналів, а презентера тільки за виведення інформації що дає можливість використовувати різні види інтерфейсів. Наприклад під час роботи програмного забезпечення можна виводити одну інформацію як у графічний інтерфейс так і в файл(рисунок 2.4).

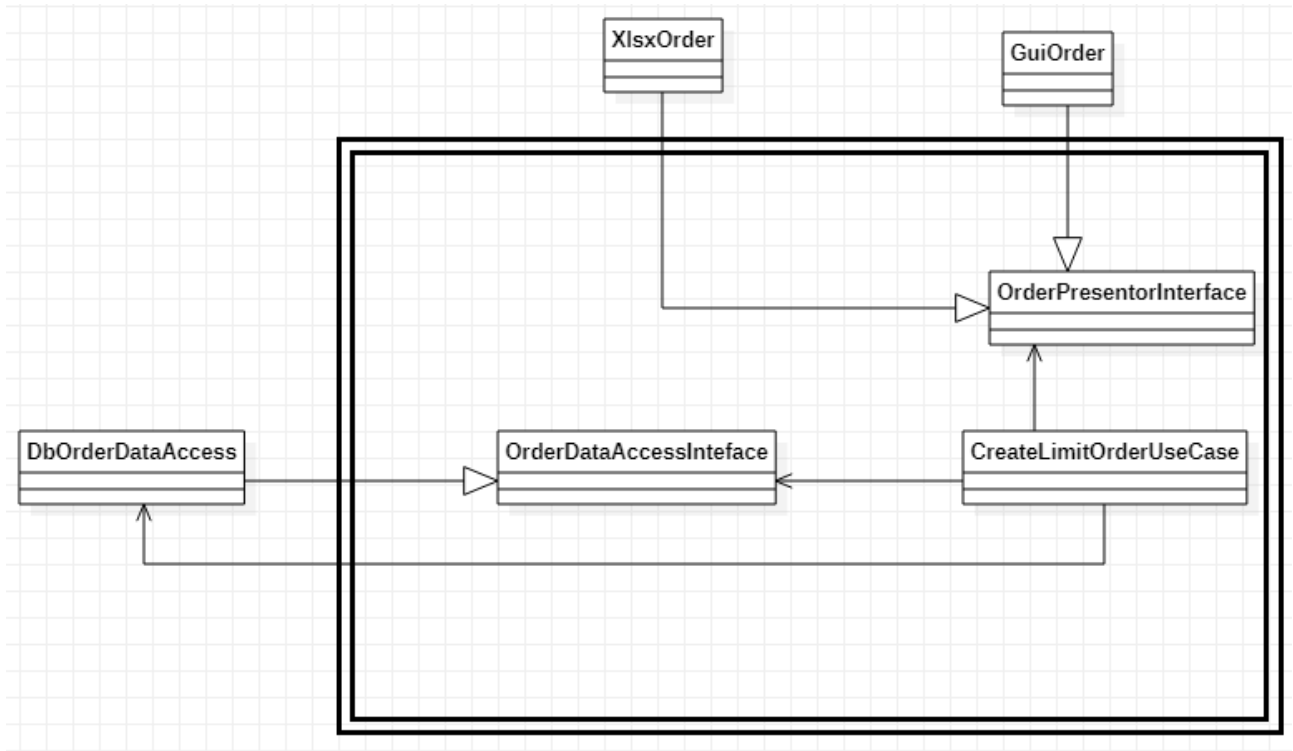


Рисунок 2.4 - Приклад перетину кордонів

2.3 Висновки за 2 розділом

У цьому розділі було описано основні рішення щодо архітектури. Описано чим є бізнес-правила, бізнес-правила додатку, приклад перетину кордонів в додатку, від чого не повинні залежати бізнес-правила, області видимості шарів, який шар архітектури, що знає про інший шар.

3 РОЗРОБКА ПРОГРАМИ

3.1 Опис вимог

На першому етапі створення програми визначається мета проекту: спростити одночасного управління декількома гаманцями біржі.

Після формулювання мети створюється технічне завдання:

- спрощення одночасного управління декількома гаманцями біржі;
- можливість використовувати наступні заявки:
 - маркет заявка;
 - стоп-ліміт-заявка;
 - стоп-маркет-заявка;
 - лімітна заявка;
- можливість додавання і видалення нових акаунтів;
- слідкування за станом заявок;
- виведення заявок на біржу Binance.

3.2 Опис архітектури

Після отримання технічного завдання починається процес написання додатку, які починається з формування архітектури. Так як я використовую чисту архітектуру я почну розробку додатку з ядра системи тобто сутностей чи критичних бізнес правил.

Критичні бізнес правила повинні знаходитись у сутностях, які я поділив наступним чином:

- заявки, які утримують у собі правила їх створення;
- торгові пари, які утримують у собі правила зміни цін та кількості монети, найбільш можливу і найменшу можливу кількість;
- аккаунт, який групує у собі всі заявки та торгові пари з якими працює чи працював той аккаунт та вирішує можливо чи ні торгівля;

- менеджер аккаунтів, який є контейнером для аккаунтів;
- менеджер торгових пар, який є контейнером для них;
- монета, яка утримує у собі кількість самої монети, кількість заблокованої і не заблокованої монети.

3.3 Опис заявок

Найбільш цікавим з точки зору програмування є заявки, тому що вони є частинами один одного та мають спільні методи. Заявки написані з використанням паттерна Стан. Стан - це поведінковий патерн проектування, який дозволяє об'єктам змінювати поведінку залежно від свого стану. Основна ідея в тому, що програма може знаходитися в одному з декількох станів, які весь час змінюють один одного. Набір цих станів, а також переходів між ними, зумовлений і кінцевий. Перебуваючи в різних станах, програма може по-різному реагувати на одні і ті ж події, які відбуваються з нею.

В даному випадку Ліміт-заявка має декілька станів: заповнений, новий, частково заповнений, скасований. Переходи між станами наступні:

- з нового до частково заповненого до заповненого;
- з нового до частково заповненого до скасованого;
- з нового до скасованого.

Доступ до полів ліміт-заявки суворо регульований. Якщо зовнішнім клієнтам не треба мати доступ до внутрішньої інформації вони її не матимуть. Також у кожному сетері своя поведінка, котра змінюється і залежить від стану заявки. Для нової ліміт-заявки неможливо зміни значення `executed_qty`, тому що новою заявка вважається тільки якщо `executed_qty` дорівнює нулю, якщо `executed_qty` більше нуля, то заявка вважається частково заповненою. Це якраз є одне з критичних бізнес-правил. Теж саме з `cumulative_quote_qty`. Доступ до переходу у будь-який інших стан це теж бізнес-правило. Такі правила будуть дійсними для будь-якої біржі і при зміні біржі їх не треба буде змінювати.

Заявкам немає ніякої різниці з якою біржею працює програма, у якій базі даних вони зберігаються.

Перехід з нового стану ордеру можливий у будь-який інший.

Стан ліміт-заявки написан на основі власного інтерфейсу тому що не всі заявки мають такі ж самі стани як і ліміт-заявка і по тій ж самій причині базовий стан для всіх ордерів не має абстрактних гетерів і сетерів.

На прикладі заповненого ордеру можливо подивитися як реалізована заборона на зміну полів і переходів до інших станів. Заборона реалізована визовом помилки `ValueError`. Також поле `executed_qty` неможливо змінити на менше ніж нуль, чи менше від його попереднього значення та неможливо щоб значення `executed_qty` було більше значення `qty`. Важливо зазначити, що я для кожного стану написав свою властивість тому що я хотів відійти від строкових значень таких як `FILLED`, `NEW` до простого так чи ні, так це потребує написання більшої кількості властивостей, але це звільняє мене від використання у коді строкових констант.

Також на основі ліміт заявки написана стоп-ліміт-заявка. Єдина різниця між заявками у тому що стоп-ліміт-заявка має стоп-ціну. Також саме на основі маркет заявки написана стоп-маркет-заявка.

Основою ордерів є клас, який утримує у собі дані для заявок та у якому реалізовані спільні для всіх методи здебільшого це сеттери, геттери. Він утримує дані в собі у вигляді класу даних, який написаний за допомогою бібліотеки `dataclass`. Ця бібліотека спрощує написання класів даних, використовуючи її немає необхідності писати метод `__init__` та перевірку типів. Так само основою станів є клас, який частково абстрактний. Абстрактними є методи, які є у кожному стані ордера, а саме це гетери та сетери.

Усі ордери разом утворюють компонент, тому що в них є спільні причини для зміни та вони не мають спільних причин для змін з іншими класами. Наприклад кожна заявка має використовувати тип даних `Decimal`, який не має проблем з числами з плаваючою комою. Якщо такий тип зберігання буде

вирішено змінити на роздільне зберігання цілої та дробної частини, то потрібно буде змінити її у всіх заявках. Також заявки мають спільний клас.

3.4 Опис варіантів використання

У додатку є наступні варіанти використання системи:

- створити маркет заявку на купівлю монети;
- створити маркет заявки на продаж монети;
- створити лімітну заявку на купівлю монети;
- створити лімітну заявку на продаж монети;
- створити стоп-ліміт заявку на купівлю монети;
- створити стоп-ліміт заявку на продаж монети;
- створити стоп-маркет заявку на купівлю монети;
- створити стоп-маркет заявку на продаж монети;
- додати аккуант;
- видалити аккаунт;
- скласти звіт про акаунт;
- оновити інформацію про заявки;
- оновити інформацію про аккаунт;
- ініціалізувати аккаунт.

Написання варіантів використання я пропоную продемонструвати на варіанті використання “Ініціалізувати аккаунт”. Тому що всі варіанти використання мають одну мету - управляти бізнес-логікою. Вони як бар’єр між інфраструктурою і бізнес-логікою. Розглянемо написання варіантів використання на прикладі ініціалізації аккаунта.

Цей варіант використання отримує дані про акаунти з бази даних через інтерфейс `DbOrderAccessI`, тим самим захищаючи себе від змін у цьому компоненті, який реалізує підключення до бази даних. Також тип бази даних не має ніякого значення і через цей інтерфейс можливо підключити будь-який вид

бази даних навіть замість бази даних можливо використовувати звичайні файли і для варіанту використання не буде ніякої різниці. Так само і з інтерфейсом за інтерфейсом `AddingOrdersI` може бути будь-який вид інтерфейсу можливо виводити дані у веб-браузер чи у консоль. Важливо зазначити що дані передаються між шарами за допомогою структур даних.

Чому не передаються за допомогою звичайних типів даних, наприклад `dict`, тому що у звичайних структурах даних неможливо задокументувати і контролювати дані. У `dict` можуть бути які завгодно ключі та значення будь якого типу, а використовуючи структури даних я можу бути впевненим які дані будуть передані.

Доступ до API біржі Binance за допомогою патерна Абстрактна фабрика - це патерн проектування, який дозволяє створювати сімейства пов'язаних об'єктів, не прив'язуючись до конкретних класів створюваних об'єктів. За допомогою абстрактної фабрики можливо передавати об'єкту, який її використовує різні сімейства об'єктів, які мають однаковий інтерфейс. Тобто об'єкт, який використовує фабрику не розуміє які конкретно об'єкти від використовує так як вони мають однаковий інтерфейс(рисунк 3.1).

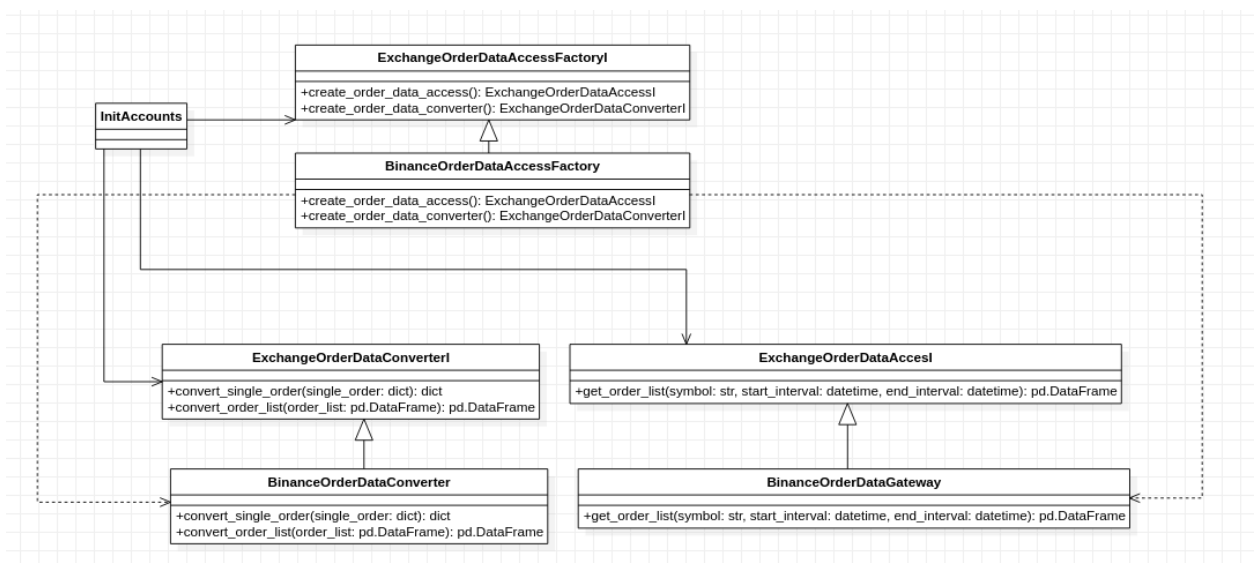


Рисунок 3.1 - Реалізація патерну абстрактна фабрика

Підключення до Binance API не відрізняється нічим особливим. Пара ключів доступу і все готово. У API лімітована кількість викликів. Підтриманням правильного інтервалу між викликами займається клас `BinanceBase`(рисунок 3.2). Після отримання даних вони проходять валідацію за допомогою бібліотеки `schema`, котра надає легкий та зрозумілий механізм перевірки.

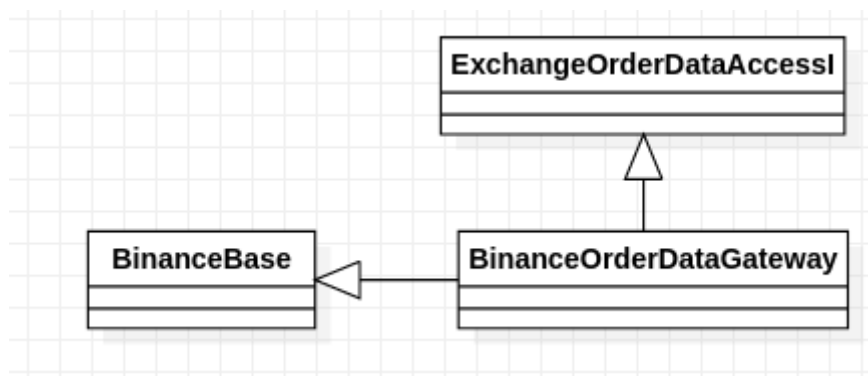


Рисунок 3.2 - Підключення до API

У класі, через який надається доступ до Binance API, не фільтрує поля, тому що для кожної програми потрібен свій перелік полів, тому фільтрація проходить у іншому класі під назвою `BinanceOrderDataConverter`. Таке розмежування допомагає захистити клас `BinanceOrderDataGateway` від змін при використанні його у декількох компонентах чи навіть програмах. Цей клас без змін я використовую вже один рік у різних додатках та компонентах різної величини та різного напрямку дії.

Таке розмежування зберігає час на розробку. Все що для цього потрібно з самого початку правильно написати клас, у якому не буде декількох осей зміни.

3.5 Визначення схеми бази даних

Таблиця `Order` включає поля(рисунок 3.3):

- `order_id` (ідентифікатор заявки);
- `symbol` (символ торгової пари);

- account_id (ідентифікатор аккаунта);
- price (ціна заявки);
- stop_price (стоп-ціна заявки);
- qty (кількість монети у заявці);
- executed_qty (виконана кількість);
- cumulative_quote_qty (заповнену кількість в кватній монеті);
- status (стан заявки);
- side (сторона заявки);
- type (тип заявки).

Таблиця Account включає наступні поля(рисунок 3.3):

- account_secret_key (секретний API ключ);
- account_public_key (публічний API ключ);
- coin_symbol (символ монети).

Таблиця Coin включає наступні поля(рисунок 3.3):

- coin_symbol (символ монети);
- free_coin (вільна кількість монети);
- locked_coin (заблокована кількість монети).

Таблиця Symbol включає наступні поля(рисунок 3.3):

- symbol (символ торгової пари);
- max_amount (максимальна кількість монети);
- min_amount (мінімальна кількість монети);
- amount_step (крок кількості);
- min_price (мінімальна ціна);
- max_price (максимальна ціна);
- price_step (крок ціни).



Рисунок 3.3 - Схема БД

3.6 Висновки за розділом 3

У цьому розділі було описано основні рішення щодо розробки. Описано принцип розробки кожного з шарів архітектури на прикладі додатку та надані приклади реалізації шарів, кордонів, структур даних. Схема бази даних.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ПРОГРАМИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування

Розроблений додаток представляє собою простий торговельний термінал. Який дозволяє управляти декількома біржовими гаманцями з одного терміналу а саме створювати ордери та слідкувати за їх станом.

4.2 Робота з додатком

Після запуску додатку користувач побаче головне вікно програми(рисунок 4.1). Яке має у собі таблицю заявок, кнопку додання аккаунта, кнопку видалення аккаунта, інтерфейс створення заявок, поле для системних повідомлень.

The screenshot shows the main interface of the trading application. It includes a table of orders, buttons for account management, order type selection, and a log of system messages.

symbol	order_id	account	qty	price	executed_qty	side	status	type
BTCUSDT	1424211	txpPyJ	0.08032	37 323	0.06454	BUY	PARTIALLY...	Limit
BTCUSDT	1324211	txpWRs	1.18032	37 023	1.02300	BUY	PARTIALLY...	Limit
BTCUSDT	1233311	sdrWRs	0.18032	36 543	0.18032	SELL	FILLED	Market

Buttons: Del Account, Add account, Limit, Market, Stop-Limit

Buy BTC 4 123 USDT Sell BTC 1.45093 BTC

Price in USD Price in USD

Amount in BTC Amount in BTC

BUY SELL

01.05.21 11:53 the 1424211 order now has the PARTIALLY_FILLED status.
 01.05.21 11:33 BTCUSDT order is placed on Binance.
 01.05.21 11:33 Price: 37 323
 01.05.21 11:33 Qty: 0.08032
 01.05.21 11:33 Side: BUY
 01.05.21 11:33 Type: Limit

Рисунок 4.1 - Головне вікно додатку

4.3 Таблиця заявок

Таблиця заявок(рисунок 4.2) має у собі дані про заявки: символ торгової пари, ідентифікатор заявки, ідентифікатор аккаунта, кількість у заявці, ціна заявки, виконана кількість заявки, сторона заявки, статус заявки та тип заявки.

symbol	order_id	account	qty	price	executed_qty	side	status	type
BTCUSDT	1424211	txpPyJ	0.08032	37 323	0.06454	BUY	PARTIALLY...	Limit
BTCUSDT	1324211	txpWRs	1.18032	37 023	1.02300	BUY	PARTIALLY...	Limit
BTCUSDT	1233311	sdrWRs	0.18032	36 543	0.18032	SELL	FILLED	Market

Рисунок 4.2 - Таблиця заявок

4.4 Інтерфейс створення заявок

Інтерфейс створення(рисунок 4.3) заявок має у собі поля для вводу ціни та кількості монети, інтерфейс вибору заявки у залежності від вибору інтерфейс створення заявки змінюється, кнопки створення заявки у напрямку продажі чи покупки.

Інтерфейс створення лімітної заявки(рисунок 4.3) має у собі поле ціни та кількості.

☐ Limit	☐ Market	☐ Stop-Limit
Buy BTC 4 123 USDT	Sell BTC 1.45093 BTC	
Price in USDT	Price in USDT	
Amount in BTC	Amount in BTC	
BUY	SELL	

Рисунок 4.3 - Інтерфейс створення ліміт-заявки

Інтерфейс створення стоп-ліміт заявки(рисунок 4.4) має у собі поле ціни, стоп-ціни та кількості.

☐ Limit	☐ Market	☒ Stop-Limit
Buy BTC 4 123 USDT	Sell BTC 1.45093 BTC	
Price in USDT	Price in USDT	
Stop price in USDT	Stop price in USDT	
Amount in BTC	Amount in BTC	
BUY	SELL	

Рисунок 4.4 - Інтерфейс створення стоп-ліміт-заявки

Інтерфейс створення ринкової заявки(рисунок 4.5) має у собі тільки поле кількості.

Рисунок 4.5 - Інтерфейс створення ринкової-заявки

4.5 Додавання та видалення акаунтів

Додати акаунт можливо після натискання на кнопку Add account після натискання з'явиться вікно додавання акаунта(рисунок 4.6). Для додання акаунту потрібно написати ключі доступу та натиснути на кнопку Add account. Після цього акаунт буде доданий.

Рисунок 4.6 - Інтерфейс додання акаунту

Видалення акаунта можливо після натискання на кнопку Del account. Після натискання з'явиться вікно з таблицею(рисунок 4.7), в якій будуть всі акаунти, які зараз підключенні до системи. Щоб видалити акаунт потрібно натиснути на кнопку Del поряд з ним. Все після цього акаунт буде видалений.

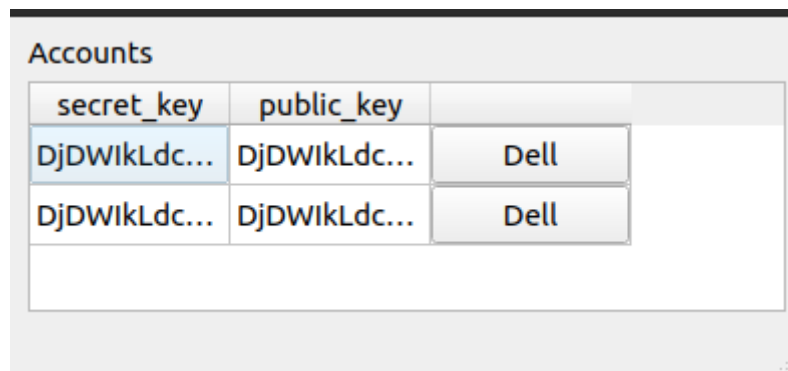


Рисунок 4.7 - Інтерфейс видалення аккаунту

4.6 Поле для системних повідомлень

У це поле виводяться(рисунок 4.8) усі дії користувача та зміни стану заявок. Він виступає бортовим журналом додатку.

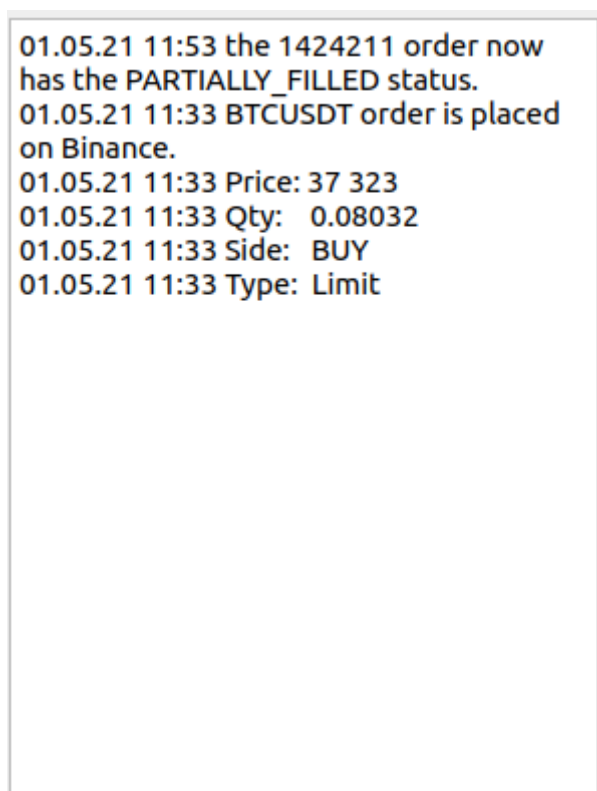


Рисунок 4.8 - Інтерфейс вікна системних повідомлень

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було створено торговий термінал, який дозволяє управляти декількома акаунтами з одного місця, а саме створювати заявки та слідкувати за станом їх виконання. Було розв'язано такі завдання:

- аналіз предметної області;
- розробка технічного завдання;
- опис обраних технологій;
- опис обраної архітектури;
- застосування обраної архітектури під час написання додатку;
- розробка додатку;
- тестування додатку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Как играть и выигрывать на бирже: Психология. Технический анализ. Контроль над капиталом / Александр Элдер. – 8-е изд. – М.: Альпина Паблишер, 2011.
2. Algorithmic trading: winning strategies and their rationale / Ernest P. Chan. Chan, Ernest P., 1966.
3. Trading and Exchanges: Market Microstructure for Practitioners / Larry Harris. - Oxford University Press, 2002 – 656с.
4. Башир И. Б33 Блокчейн: архитектура, криптовалюты, инструменты разработки, смартконтракты / пер. с англ. М. А. Райтмана. – М.: ДМК Пресс, 2019. – 538 с.
5. Vigna P. The Age of Cryptocurrency: How Bitcoin and Digital Money Are Challenging the Global Economic Order / P. Vigna, M. Casey. – New-York: Pan Books Limited, 2017. – 432 p.
6. Python 3 и PyQt 5. Разработка приложений / Н. А. Прохоренко, В. А. Дронов. – СПб.: БХВ-Петербург, 2016 – 832с.
7. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython / Wes McKinney. – Sebastopol: O`Reilly Media, 2013 – 433с.
8. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. - Prentice Hall, 2018 – 432с.
9. The Clean Code Blog of the Robert C. Martin. [Electronic resource]. – Access mode: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>.
10. Dzone website [Electronic resource]. – Access mode: <https://dzone.com/articles/solid-principles-basic-building-blocks-of-a-clean>.

ДОДАТОК А
Технічне завдання

Вступ

Розроблений мною термінал можна вільно використовувати фахівцями, з метою полегшення розробки.

A.1 Підстави для розробки

Підставою для розробки програмного забезпечення є завдання на дипломну кваліфікаційну роботу бакалавра на тему: «Розробка програмного забезпечення для управління біржовими гаманцями з можливістю покупки і продажу активів», що затверджене наказом № 103 від 30 березня 2021 р. по Національному університету «Запорізька політехніка».

A.2 Призначення розробки

Функціональне призначення: додаток дозволяє керувати декількома біржовими гаманцями, слідкувати за станом заявок, створювати заявки, додавати та видаляти нові акаунти.

A.3 Вимоги до програми чи програмному виробу

A.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати:

- виведення списку заявок;
- створення заявок;
- додавання нових акаунтів;
- видалення акаунтів;
- відображати стан заявок.

A.3.2 Вимоги до надійності

Надійне функціонування додатку має забезпечуватися наступними факторами:

- джерело безперебійного живлення апаратно-технічних засобів;
- необхідний рівень підготовки обслуговуючого персоналу.

A.3.3 Умови експлуатації

Для забезпечення експлуатації програмного забезпечення необхідно встановити наступні програмні застосунки:

- Python;
- sqlite3;
- ОС: Linux;

- pandas;
- matplotlib;
- schema;
- pyqt5.

A.3.4 Вимоги до складу й параметрів технічних засобів

Мінімальні системні вимоги:

- ОС Linux 16.04.1 або новіші;
- ОЗУ 1ГБ;
- доступ до інтернету;
- процесор приблизно 3 GHz.

A.3.5 Вимоги маркування й упакування

Вимоги до транспортування та зберігання схожі на носій, на якому зберігається програма, тобто оптичний, дискета або жорсткий диск.

A.3.6 Вимоги транспортування й збереження

Вимоги щодо доставки та зберігання точно такі ж, як вимоги щодо доставки та зберігання цифрових багатоцільових дисків.

A.4 Стадії та етапи розробки

Для роботи над дипломною кваліфікаційною роботою бакалавра заплановано виконання наступних етапів:

- аналіз проблеми;
- проєктування програмного забезпечення;
- розробка програмного забезпечення;
- тестування і налаштування програмного забезпечення;
- оформлення документації.

A.5 Порядок контролю й приймання

Дипломна кваліфікаційна робота бакалавра повинна бути виконана у відповідності з календарним планом. Дипломна кваліфікаційна робота допускається до захисту керівником, нормоконтролером, затверджується завідувачем кафедри і захищається в присутності членів екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б.1 Текст файла `limit_order.py`

```

from __future__ import annotations
from decimal import Decimal
from abc import abstractmethod

from models.spot_orders.order_enums import *
from models.trading_pair import TradingPair
from models.spot_orders.base_order import BaseOrder, BaseOrderData,
BaseOrderState

```

```

class LimitOrderData(BaseOrderData):

```

```

    """

```

Order fields have been moved to a separate class, since the limit order and the state of the limit order need access to them.

Поля ордера вынесены в отдельный класс так как к ним нужен доступ классу лимит ордера и его состояниям.

```

    """

```

```

    price: Decimal

```

```

    time_in_force: str

```

```

class LimitOrderState(BaseOrderState):

```

```

    """

```

The base class of the state of a limit order. The behavior of an order that can change is placed in this class.

Базовый класс состояния ордера. В данный класс вынесена логика, которая меняется в зависимости от состояния.

```

    """

```

```
@abstractmethod
def change_order_status_to_canceled(self) -> None:
    pass
```

```
@abstractmethod
def change_order_status_to_filled(self) -> None:
    pass
```

```
@abstractmethod
def change_order_status_to_partially_filled(self) -> None:
    pass
```

```
@property
@abstractmethod
def is_new(self) -> bool:
    pass
```

```
@property
@abstractmethod
def is_filled(self) -> bool:
    pass
```

```
@property
@abstractmethod
def is_canceled(self) -> bool:
    pass
```

```
@property
@abstractmethod
def is_partially_filled(self) -> bool:
```

pass

```
class NewLimitOrder(LimitOrderState):
```

```
    """
```

Класс состояния ордера NEW. В данное состояние можно перейти только при создании ордера. Из данного состояния

возможно перейти в любое другое состояние.

The NEW-state class of the order. A limit order can enter this state only when an object is created. A limit

order can enter to any state from this state.

```
    """
```

```
    @property
```

```
    def executed_qty(self) -> Decimal:
```

```
    return self.base_order_data.executed_qty
```

```
    @executed_qty.setter
```

```
    def executed_qty(self, executed_qty: Decimal) -> None:
```

```
    error_msg = 'Impossible to change executed_qty because the order has the status {new}!'.format(
```

```
        new=ORDER_STATUS_NEW)
```

```
    raise ValueError(error_msg)
```

```
    @property
```

```
    def cummulative_quote_qty(self) -> Decimal:
```

```
    return self.base_order_data.cummulative_quote_qty
```

```
    @cummulative_quote_qty.setter
```

```
    def cummulative_quote_qty(self, cummulative_quote_qty: Decimal) -> None:
```

```

        error_msg = 'It is impossible to set cummulative_quote_qty because the order
has the status {new}!'.format(
    new=ORDER_STATUS_NEW)
    raise ValueError(error_msg)

```

```

def change_order_status_to_filled(self) -> None:
    self.base_order_data.status = ORDER_STATUS_FILLED

```

```

    self.context.transition_to(state=FilledLimitOrder(order_data=self.base_order_d
ata))

```

```

def change_order_status_to_canceled(self) -> None:
    self.base_order_data.status = ORDER_STATUS_CANCELED

```

```

    self.context.transition_to(state=CanceledLimitOrder(order_data=self.base_ord
er_data))

```

```

def change_order_status_to_partially_filled(self) -> None:
    self.base_order_data.status = ORDER_STATUS_PARTIALLY_FILLED

```

```

    self.context.transition_to(state=PartiallyFilledLimitOrder(order_data=self.base
_order_data))

```

```

@property
def is_new(self) -> bool:
    return True

```

```

@property
def is_filled(self) -> bool:
    return False

```

```

@property
def is_canceled(self) -> bool:
    return False

```

```

@property
def is_partially_filled(self) -> bool:
    return False

```

```

class CanceledLimitOrder(LimitOrderState):

```

```

    """

```

Класс состояния ордера CANCELED. В данное состояние можно перейти из любого состояния. Данное состояние является

одним из концов цепочки состояний.

The CANCELED-state class of the order. A limit order can enter this state from any state. This is one of the

end states.

```

    """

```

```

@property
def executed_qty(self) -> Decimal:
    return self.base_order_data.executed_qty

```

```

@executed_qty.setter
def executed_qty(self, executed_qty: Decimal) -> None:
    if self.base_order_data.executed_qty == Decimal('0.0'):
    if executed_qty < Decimal('0.0'):
        error_msg = 'The executed_qty value cannot be less than 0!'
        raise ValueError(error_msg)

```

```

elif executed_qty > self.base_order_data.qty:
    error_msg = 'The executed_qty value cannot be greater than the qty!'
    raise ValueError(error_msg)
else:
    self.base_order_data.executed_qty = executed_qty
else:
    error_msg = 'Impossible to change executed_qty because the order has the status
{canceled}!'.format(
        canceled=ORDER_STATUS_CANCELED)
    raise ValueError(error_msg)

@property
def cummulative_quote_qty(self) -> Decimal:
    return self.base_order_data.cummulative_quote_qty

@cummulative_quote_qty.setter
def cummulative_quote_qty(self, cummulative_quote_qty: Decimal) -> None:
    if self.base_order_data.cummulative_quote_qty == Decimal('0.0'):
    if cummulative_quote_qty < Decimal('0.0'):
        error_msg = 'The cummulative_quote_qty value cannot be less than 0!'
        raise ValueError(error_msg)
    elif cummulative_quote_qty > self.base_order_data.qty *
self.base_order_data.price:
        error_msg = 'The cummulative_quote_qty value cannot be greater than
the qty * price!'
        raise ValueError(error_msg)
    else:
        self.base_order_data.cummulative_quote_qty = cummulative_quote_qty
    else:

```

```

    error_msg = 'Impossible to change cummulative_quote_qty because the order
has the status {canceled}!'.format(
        canceled=ORDER_STATUS_CANCELED)
    raise ValueError(error_msg)

```

```

def change_order_status_to_canceled(self) -> None:
    error_msg = 'The order already has the status
{canceled}!'.format(canceled=ORDER_STATUS_CANCELED)
    raise ValueError(error_msg)

```

```

def change_order_status_to_filled(self) -> None:
    error_msg = 'It is impossible to change the status of the order to {filled}, because
the order has the status {canceled}!'.format(
        filled=ORDER_STATUS_FILLED,
        canceled=ORDER_STATUS_CANCELED)
    raise ValueError(error_msg)

```

```

def change_order_status_to_partially_filled(self) -> None:
    error_msg = 'It is impossible to change the status of the order to
{partially_filled}, because the order has the status {canceled}!'.format(
        partially_filled=ORDER_STATUS_PARTIALLY_FILLED,
        canceled=ORDER_STATUS_CANCELED)
    raise ValueError(error_msg)

```

```
@property
```

```

def is_new(self) -> bool:
    return False

```

```
@property
```

```

def is_canceled(self) -> bool:

```

```
return True
```

```
@property
```

```
def is_filled(self) -> bool:
```

```
return False
```

```
@property
```

```
def is_partially_filled(self) -> bool:
```

```
return False
```

```
class FilledLimitOrder(LimitOrderState):
```

```
    """
```

Класс состояния ордера FILLED. В данное состояние можно перейти из состояния NEW, PARTIALLY_FILLED. Данное состояние является

одним из концов цепочки состояний.

The FILLED-state class of the order. A limit order can enter this state from NEW, PARTIALLY_FILLED. This is one of the

end states.

```
    """
```

```
@property
```

```
def executed_qty(self) -> Decimal:
```

```
return self.base_order_data.executed_qty
```

```
@executed_qty.setter
```

```
def executed_qty(self, executed_qty: Decimal) -> None:
```

```
if executed_qty < Decimal('0.0'):
```

```
    error_msg = 'The new executed_qty value cannot be less than 0!'
```

```
    raise ValueError(error_msg)
```

```
if executed_qty < self.base_order_data.executed_qty:
```

```

error_msg = 'The new executed_qty value cannot be less than the previous one!'
raise ValueError(error_msg)
if executed_qty > self.base_order_data.qty:
error_msg = 'The new executed_qty value cannot be greater than the qty!'
raise ValueError(error_msg)
self.base_order_data.executed_qty = executed_qty

@property
def cummulative_quote_qty(self) -> Decimal:
return self.base_order_data.cummulative_quote_qty

@cummulative_quote_qty.setter
def cummulative_quote_qty(self, cummulative_quote_qty: Decimal) -> None:
if cummulative_quote_qty < Decimal('0.0'):
error_msg = 'The new cummulative_quote_qty value cannot be less than 0!'
raise ValueError(error_msg)
if cummulative_quote_qty < self.base_order_data.cummulative_quote_qty:
error_msg = 'The new cummulative_quote_qty value cannot be less than the
previous one!'
raise ValueError(error_msg)
if cummulative_quote_qty > self.base_order_data.qty *
self.base_order_data.price:
error_msg = 'The new cummulative_quote_qty value cannot be greater than the
qty * price!'
raise ValueError(error_msg)
self.base_order_data.cummulative_quote_qty = cummulative_quote_qty

def change_order_status_to_canceled(self) -> None:
error_msg = 'It is impossible to change the status of the order to {canceled},
because the order has the status {filled}!'.format(

```

```

        filled=ORDER_STATUS_FILLED,
canceled=ORDER_STATUS_CANCELED)
        raise ValueError(error_msg)

    def change_order_status_to_filled(self) -> None:
        error_msg = 'The order already has the status
{filled}!'.format(filled=ORDER_STATUS_FILLED)
        raise ValueError(error_msg)

    def change_order_status_to_partially_filled(self) -> None:
        error_msg = 'It is impossible to change the status of the order to
{partially_filled}, because the order has the status {filled}!'.format(
        partially_filled=ORDER_STATUS_PARTIALLY_FILLED,
filled=ORDER_STATUS_FILLED)
        raise ValueError(error_msg)

    @property
    def is_new(self) -> bool:
        return False

    @property
    def is_canceled(self) -> bool:
        return False

    @property
    def is_filled(self) -> bool:
        return True

    @property
    def is_partially_filled(self) -> bool:

```

```
return False
```

```
class PartiallyFilledLimitOrder(LimitOrderState):
```

```
    """
```

Класс состояния ордера `PARTIALLY_FILLED`. В данное состояние можно перейти из состояния `NEW`. Из данного состояния

можно перейти в состояние `FILLED`.

The `PARTIALLY_FILLED`-state class of the order. A limit order can enter this state from `NEW`. A limit order can

enter to `FILLED`-state from this state.

```
    """
```

```
    @property
```

```
    def executed_qty(self) -> Decimal:
```

```
        return self.base_order_data.executed_qty
```

```
    @executed_qty.setter
```

```
    def executed_qty(self, executed_qty: Decimal) -> None:
```

```
        if executed_qty < Decimal('0.0'):
```

```
            error_msg = 'The new executed_qty value cannot be less than 0!'
```

```
            raise ValueError(error_msg)
```

```
        if executed_qty < self.base_order_data.executed_qty:
```

```
            error_msg = 'The new executed_qty value cannot be less than the previous one!'
```

```
            raise ValueError(error_msg)
```

```
        if executed_qty > self.base_order_data.qty:
```

```
            error_msg = 'The new executed_qty value cannot be greater than the qty!'
```

```
            raise ValueError(error_msg)
```

```
        self.base_order_data.executed_qty = executed_qty
```

```
    @property
```

```

def cummulative_quote_qty(self) -> Decimal:
    return self.base_order_data.cummulative_quote_qty

@cummulative_quote_qty.setter
def cummulative_quote_qty(self, cummulative_quote_qty: Decimal) -> None:
    if cummulative_quote_qty < Decimal('0.0'):
        error_msg = 'The new cummulative_quote_qty value cannot be less than 0!'
        raise ValueError(error_msg)
    if cummulative_quote_qty < self.base_order_data.cummulative_quote_qty:
        error_msg = 'The new cummulative_quote_qty value cannot be less than the
previous one!'
        raise ValueError(error_msg)
    if      cummulative_quote_qty      >      self.base_order_data.qty      *
self.base_order_data.price:
        error_msg = 'The new cummulative_quote_qty value cannot be greater than the
qty * price!'
        raise ValueError(error_msg)
    self.base_order_data.cummulative_quote_qty = cummulative_quote_qty

def change_order_status_to_canceled(self) -> None:
    self.base_order_data.status = ORDER_STATUS_CANCELED

    self.context.transition_to(state=CanceledLimitOrder(order_data=self.base_ord
er_data))

def change_order_status_to_filled(self) -> None:
    self.base_order_data.status = ORDER_STATUS_FILLED

    self.context.transition_to(state=FilledLimitOrder(order_data=self.base_order_d
ata))

```

```

def change_order_status_to_partially_filled(self) -> None:
    error_msg = 'The order already has the status {partially_filled}!'.format(
        partially_filled=ORDER_STATUS_PARTIALLY_FILLED)
    raise ValueError(error_msg)

```

```

@property
def is_new(self) -> bool:
    return False

```

```

@property
def is_canceled(self) -> bool:
    return False

```

```

@property
def is_filled(self) -> bool:
    return False

```

```

@property
def is_partially_filled(self) -> bool:
    return True

```

```

class LimitOrderI(BaseOrder):

```

```

    @staticmethod
    def check_time_in_force(time_in_force: str) -> None:
        if time_in_force not in TIME_IN_FORCE_LIST:
            raise ValueError('Wrong time_in_force of the order.\n
The order time_in_force
must be one of them: {}'.format(

```

```
        ', '.join(TIME_IN_FORCE_LIST)))
```

```
@staticmethod
```

```
def check_side(side: str) -> None:
```

```
    if side not in (SIDE_BUY, SIDE_SELL):
```

```
        raise ValueError('Wrong side of the order.\n'
                           'The order side must be SELL or '
                           'BUY!')
```

```
@staticmethod
```

```
def check_status(status: str) -> None:
```

```
    if status not in ORDER_STATUS_LIST:
```

```
        raise ValueError('Wrong status of the order.\n'
                           'The order status must be one of '
                           'them: {}'.format(
                               ', '.join(ORDER_STATUS_LIST)))
```

```
@staticmethod
```

```
def check_qty(qty: Decimal) -> None:
```

```
    if qty < Decimal('0.0'):
```

```
        raise ValueError('The qty value cannot be less than 0!')
```

```
    elif qty == Decimal('0.0'):
```

```
        raise ValueError('The qty value cannot be 0!')
```

```
@staticmethod
```

```
def check_price(price: Decimal) -> None:
```

```
    if price < Decimal('0.0'):
```

```
        raise ValueError('The price value cannot be less than 0!')
```

```
    elif price == Decimal('0.0'):
```

```
        raise ValueError('The price value cannot be 0!')
```

```
@staticmethod
```

```

def check_order_id(order_id: int) -> None:
    if order_id < 0:
        raise ValueError('The order_id cannot be less than 0!')
    elif order_id == 0:
        raise ValueError('The order_id value cannot be 0!')

```

```

@property
@abstractmethod
def price(self) -> Decimal:
    pass

```

```

@property
@abstractmethod
def is_new(self) -> bool:
    pass

```

```

@property
@abstractmethod
def is_filled(self) -> bool:
    pass

```

```

@property
@abstractmethod
def is_canceled(self) -> bool:
    pass

```

```

@property
@abstractmethod
def is_partially_filled(self) -> bool:
    pass

```

```

@abstractmethod
def change_order_status_to_filled(self) -> None:
    pass

```

```

@abstractmethod
def change_order_status_to_canceled(self) -> None:
    pass

```

```

@abstractmethod
def change_order_status_to_partially_filled(self) -> None:
    pass

```

```

class LimitOrder(LimitOrderI):

```

```

    def __init__(
        self,
        side: str,
        status: str,
        qty: Decimal,
        order_id: int,
        price: Decimal,
        time_in_force: str,
        symbol: TradingPair,
        executed_qty: Decimal = Decimal('0.0'),
        cumulative_quote_qty: Decimal = Decimal('0.0')
    ) -> None:
        self.check_status(status=status)
        self.check_side(side=side)

```

```

self.check_time_in_force(time_in_force=time_in_force)
self.check_order_id(order_id=order_id)
self.check_qty(qty=qty)
self.check_price(price=price)
price, qty = symbol.formatting_order_parameters(price=price, qty=qty)
self.__order_data = LimitOrderData(
    qty=qty,
    side=side,
    price=price,
    status=status,
    symbol=symbol,
    order_id=order_id,
    time_in_force=time_in_force
)
if status is ORDER_STATUS_NEW:
    self.state = NewLimitOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_FILLED:
    self.state = FilledLimitOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_CANCELED:
    self.state = CanceledLimitOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_PARTIALLY_FILLED:
    self.state = PartiallyFilledLimitOrder(order_data=self.__order_data)
else:
    raise ValueError('Unsupported state!')
self.state.context = self
if executed_qty > 0:
    self.state.executed_qty = executed_qty
if cummulative_quote_qty > 0:
    self.state.cummulative_quote_qty = cummulative_quote_qty

```

```

super().__init__(base_order_data=self.__order_data, state=self.state)

@property
def price(self) -> Decimal:
    return self.__order_data.price

@property
def is_new(self) -> bool:
    return self.state.is_new

@property
def is_filled(self) -> bool:
    return self.state.is_filled

@property
def is_canceled(self) -> bool:
    return self.state.is_canceled

@property
def is_partially_filled(self) -> bool:
    return self.state.is_partially_filled

def change_order_status_to_filled(self) -> None:
    self.state.change_order_status_to_filled()

def change_order_status_to_canceled(self) -> None:
    self.state.change_order_status_to_canceled()

def change_order_status_to_partially_filled(self) -> None:
    self.state.change_order_status_to_partially_filled()

```

Б.2 Текст файла market_order.py

```

from __future__ import annotations
from abc import abstractmethod
from decimal import Decimal
from dataclasses import dataclass

from models.spot_orders.order_enums import *
from models.spot_orders.base_order import BaseOrder, BaseOrderState,
BaseOrderData
from models.trading_pair import TradingPair

@dataclass
class MarketOrderData(BaseOrderData):
    average_price: Decimal = Decimal('0.0')

class MarketOrderState(BaseOrderState):
    @property
    @abstractmethod
    def average_price(self) -> Decimal:
        pass

    @property
    @abstractmethod
    def is_filled(self) -> bool:
        pass

    @property

```

```

@abstractmethod
def is_expired(self) -> bool:
    pass

```

```

class FilledMarketOrder(MarketOrderState):

```

```

    def __init__(self, order_data: BaseOrderData) -> None:
        super().__init__(order_data=order_data)
        self.__cummulative_quote_qty_changed = False
        self.__executed_qty_changed = False

```

```

    @property
    def cummulative_quote_qty(self) -> Decimal:
        return self.base_order_data.cummulative_quote_qty

```

```

    @cummulative_quote_qty.setter
    def cummulative_quote_qty(self, cummulative_quote_qty: Decimal) -> None:
        if not self.__cummulative_quote_qty_changed:
            if cummulative_quote_qty < Decimal('0.0'):
                error_msg = 'The new cummulative_quote_qty value cannot be less than

```

```

0!'

```

```

                raise ValueError(error_msg)
        self.base_order_data.cummulative_quote_qty = cummulative_quote_qty
        self.__update_average_price()
        self.__cummulative_quote_qty_changed = True
    else:
        raise ValueError('The cummulative_quote_qty value cannot be changed!')

```

```

    @property
    def executed_qty(self) -> Decimal:

```

```
return self.base_order_data.executed_qty
```

```
@executed_qty.setter
```

```
def executed_qty(self, executed_qty: Decimal) -> None:
```

```
if not self.__executed_qty_changed:
```

```
if executed_qty < Decimal('0.0'):
```

```
    error_msg = 'The new executed_qty value cannot be less than 0!'
```

```
    raise ValueError(error_msg)
```

```
if executed_qty > self.base_order_data.qty:
```

```
    error_msg = 'The new executed_qty value cannot be greater than the qty!'
```

```
    raise ValueError(error_msg)
```

```
self.base_order_data.executed_qty = executed_qty
```

```
self.__update_average_price()
```

```
self.__executed_qty_changed = True
```

```
else:
```

```
    raise ValueError("The executed_qty value cannot be changed!")
```

```
def __update_average_price(self) -> None:
```

```
    if self.base_order_data.cummulative_quote_qty > Decimal('0.0') and
self.base_order_data.executed_qty > Decimal(
    '0.0'):
```

```
        self.base_order_data.average_price =
```

```
self.base_order_data.cummulative_quote_qty / self.base_order_data.executed_qty
```

```
@property
```

```
def average_price(self) -> Decimal:
```

```
    if self.base_order_data.cummulative_quote_qty > Decimal('0.0') and
self.base_order_data.executed_qty > Decimal('0.0'):
```

```
        self.__update_average_price()
```

```
    return self.base_order_data.average_price
```

```

@property
def is_filled(self) -> bool:
    return True

```

```

@property
def is_expired(self) -> bool:
    return False

```

```

class ExpiredMarketOrder(MarketOrderState):

```

```

    @property
    def cummulative_quote_qty(self) -> Decimal:
        return self.base_order_data.cummulative_quote_qty

```

```

    @cummulative_quote_qty.setter
    def cummulative_quote_qty(self, cummulative_quote_qty: Decimal) -> None:
        error_msg = f'It is impossible to set cummulative_quote_qty because the order
has the status {ORDER_STATUS_EXPIRED}'
        raise ValueError(error_msg)

```

```

    @property
    def executed_qty(self) -> Decimal:
        return self.base_order_data.executed_qty

```

```

    @executed_qty.setter
    def executed_qty(self, executed_qty: Decimal) -> None:
        error_msg = f'It is impossible to set executed_qty because the order has the status
{ORDER_STATUS_EXPIRED}'
        raise ValueError(error_msg)

```

```

@property
def average_price(self) -> Decimal:
    return self.base_order_data.average_price

```

```

@property
def is_filled(self) -> bool:
    return False

```

```

@property
def is_expired(self) -> bool:
    return True

```

```

class MarketOrder(BaseOrder):

```

```

    def __init__(
        self,
        side: str,
        status: str,
        qty: Decimal,
        order_id: int,
        symbol: TradingPair,
        executed_qty: Decimal = Decimal('0.0'),
        cumulative_quote_qty: Decimal = Decimal('0.0')
    ) -> None:
        if status not in (ORDER_STATUS_EXPIRED, ORDER_STATUS_FILLED):
            raise ValueError('Wrong status of the order.\n'
                              'The order status must be one of'
                              'them: {}'.format(
                                  ', '.join((ORDER_STATUS_EXPIRED, ORDER_STATUS_FILLED))))

```

```

if side not in (SIDE_BUY, SIDE_SELL):
    raise ValueError(f'Wrong side of the order.\nThe order side must be
{SIDE_SELL} or {SIDE_BUY}!')
if order_id < 0:
    raise ValueError('The order_id cannot be less than 0!')
elif order_id == 0:
    raise ValueError('The order_id value cannot be 0!')
if qty < Decimal('0.0'):
    raise ValueError('The qty value cannot be less than 0!')
elif qty == Decimal('0.0'):
    raise ValueError('The qty value cannot be 0!')
self.__order_data = MarketOrderData(
    qty=qty,
    side=side,
    status=status,
    symbol=symbol,
    order_id=order_id
)
if status is ORDER_STATUS_FILLED:
    self.__state = FilledMarketOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_EXPIRED:
    self.__state = ExpiredMarketOrder(order_data=self.__order_data)
else:
    raise ValueError('Unsupported state!')
self.__state.context = self
if executed_qty > 0:
    self.__state.executed_qty = executed_qty
if cummulative_quote_qty > 0:
    self.__state.cummulative_quote_qty = cummulative_quote_qty

```

```
super().__init__(base_order_data=self.__order_data, state=self.__state)
```

```
@property
```

```
def average_price(self) -> Decimal:
```

```
return self.__state.average_price
```

```
@property
```

```
def is_filled(self) -> bool:
```

```
return self.__state.is_filled
```

```
@property
```

```
def is_expired(self) -> bool:
```

```
return self.__state.is_expired
```

Б.3 Текст файла stop_limit_order.py

```
from decimal import Decimal
```

```
from models.spot_orders.order_enums import *
```

```
from models.spot_orders.limit_order import LimitOrderI, LimitOrderData,  
FilledLimitOrder, PartiallyFilledLimitOrder, CanceledLimitOrder, NewLimitOrder
```

```
from models.trading_pair import TradingPair
```

```
class StopLimitOrder(LimitOrderI):
```

```
    def __init__(
```

```
        self,
```

```
        side: str,
```

```
        status: str,
```

```
        qty: Decimal,
```

```

order_id: int,
price: Decimal,
time_in_force: str,
stop_price: Decimal,
symbol: TradingPair,
executed_qty: Decimal = Decimal('0.0'),
cumulative_quote_qty: Decimal = Decimal('0.0')
) -> None:
self.check_status(status=status)
self.check_side(side=side)
self.check_time_in_force(time_in_force=time_in_force)
self.check_order_id(order_id=order_id)
self.check_qty(qty=qty)
self.check_price(price=price)
if stop_price == Decimal('0.0'):
raise ValueError("The stop_price value cannot be equal to 0!")
elif stop_price < Decimal('0.0'):
raise ValueError("The stop_price value cannot be less than 0!")
price, qty = symbol.formatting_order_parameters(price=price, qty=qty)
self.__order_data = LimitOrderData(
qty=qty,
side=side,
price=price,
status=status,
symbol=symbol,
order_id=order_id,
stop_price=stop_price,
time_in_force=time_in_force
)
if status is ORDER_STATUS_NEW:

```

```

self.state = NewLimitOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_FILLED:
self.state = FilledLimitOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_CANCELED:
self.state = CanceledLimitOrder(order_data=self.__order_data)
elif status is ORDER_STATUS_PARTIALLY_FILLED:
self.state = PartiallyFilledLimitOrder(order_data=self.__order_data)
else:
raise ValueError('Unsupported state!')
self.state.context = self
if executed_qty > 0:
self.state.executed_qty = executed_qty
if cummulative_quote_qty > 0:
self.state.cummulative_quote_qty = cummulative_quote_qty

super().__init__(base_order_data=self.__order_data, state=self.state)

@property
def price(self) -> Decimal:
return self.__order_data.price

@property
def is_new(self) -> bool:
return self.state.is_new

@property
def is_filled(self) -> bool:
return self.state.is_filled

@property

```

```
def is_canceled(self) -> bool:
    return self.state.is_canceled
```

```
@property
def is_partially_filled(self) -> bool:
    return self.state.is_partially_filled
```

```
def change_order_status_to_filled(self) -> None:
    self.state.change_order_status_to_filled()
```

```
def change_order_status_to_canceled(self) -> None:
    self.state.change_order_status_to_canceled()
```

```
def change_order_status_to_partially_filled(self) -> None:
    self.state.change_order_status_to_partially_filled()
```

```
def is_activated(self) -> bool:
    if not self.state.is_new and not self.state.is_canceled:
        return True
    return False
```

```
@property
def stop_price(self) -> Decimal:
    return self.__order_data.stop_price
```

ДОДАТОК В
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЗАПОРІЗЬКА
ПОЛІТЕХНІКА" КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота бакалавра за темою "РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ БІРЖОВИМИ ГАМАНЦЯМИ З МОЖЛИВІСТЮ ПОКУПКИ І ПРОДАЖУ АКТИВІВ"

Підготував: Будюк Владислав Андрійович, ст. гр. КНТ-127
Керівник: Тетяна Василівна Федорончак к.т.н. доцент НУ"ЗП"

2021

1

Рисунок В.1 - Слайд 1

Об'єкт, предмет дослідження, та мета роботи

Об'єкт дослідження - процес реалізації програмного забезпечення для управління біржовими гаманцями з можливістю покупки і продажу активів.

Предмет дослідження - методи розробки програмного забезпечення з використанням Python, PyQt5, Sqlite3, Schema.

Мета роботи - розробка терміналу для торгівлі з реалізацією архітектури "Clean Architecture".

< 2 > ⋮

2

Рисунок В.2 - Слайд 2



Завдання до роботи

В процесі виконання роботи були поставлені наступні задачі:

- аналіз предметної області;
- розробка технічного завдання;
- опис обраних технологій;
- визначення архітектури додатку;
- розробка додатку;
- тестування розробленого додатку.

Рисунок В.3 - Слайд 3



Вибір мови програмування

Python - високорівнева мова програмування загального призначення з динамічною строгою типізацією і автоматичним управлінням пам'яттю, орієнтований на підвищення продуктивності розробника, читання коду і його якості, а також на забезпечення переносимості написаних на ньому програм. Мова програмування є повністю об'єктно-орієнтованим - все є об'єктами. Незвичайною особливістю мови є виділення блоків коду пробільними відступами. Python є інтерпретованою мовою. Через це він не такий швидкий як компільовані мови такі як С та С++. У Python є один важливий плюс це його співтовариство та бібліотеки, за допомогою яких можливо вирішити будь-яку задачу в будь-якій сфері діяльності.

Рисунок В.4 - Слайд 4

Використанні технології при розробці

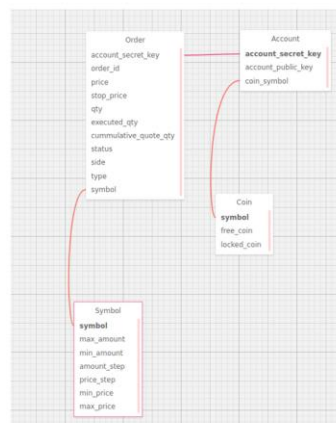
Для реалізації проекту були використані наступні технології:

- pyqt5;
- pandas;
- schema;
- sqlite3.

5

Рисунок В.5 - Слайд 5

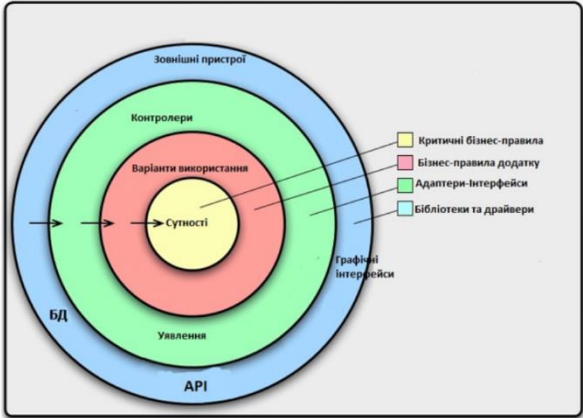
Схема бази даних



6

Рисунок В.6 - Слайд 6

Архітектура проекту



7

Рисунок В.7 - Слайд 7

Головне вікно терміналу

Orders 3

Del Account

Add account

Limit

Market

Stop-Limit

symbol	order_id	account	qty	price	executed_qty	side	status	type
BTCUSDT	1424211	txpPyJ	0.08032	37 323	0.06454	BUY	PARTIALLY...	Limit
BTCUSDT	1324211	txpWRs	1.18032	37 023	1.02300	BUY	PARTIALLY...	Limit
BTCUSDT	1233311	sdrWRs	0.18032	36 543	0.18032	SELL	FILLED	Market

Buy BTC 4.123 USDT

Sell BTC 1.45093 BTC

Price in USDT

Price in USDT

Amount in BTC

Amount in BTC

BUY

SELL

01.05.21 11:53 the 1424211 order now has the PARTIALLY_FILLED status.
01.05.21 11:33 BTCUSDT order is placed on Binance.
01.05.21 11:33 Price: 37 323
01.05.21 11:33 Qty: 0.08032
01.05.21 11:33 Side: BUY
01.05.21 11:33 Type: Limit

8

Рисунок В.8 - Слайд 8

Рисунок В.9 - Слайд 9

☐ Limit
 ☐ Market
 ☐ Stop-Limit

Buy BTC 4 123 USDT
 Sell BTC 1.45093 BTC

Price in USD
 Price in USD

Amount in BTC
 Amount in BTC

BUY
 SELL

☐ Limit
 ☐ Market
 ☒ Stop-Limit

Buy BTC 4 123 USDT
 Sell BTC 1.45093 BTC

Price in USD
 Price in USD

Stop price in USD
 Stop price in USD

Amount in BTC
 Amount in BTC

BUY
 SELL

☐ Limit
 ☒ Market
 ☐ Stop-Limit

Buy BTC 4 123 USDT
 Sell BTC 1.45093 BTC

Amount in BTC
 Amount in BTC

BUY
 SELL

Рисунок В.10 - Слайд 10



Додавання і видалення акаунтів

Secret API key

Public API key

Add account

Accounts

secret_key	public_key	
DjDWikLdc...	DjDWikLdc...	Dell
DjDWikLdc...	DjDWikLdc...	Dell

11

Рисунок В.11 - Слайд 11



“Бортовий журнал” додатку

01.05.21 11:53 the 1424211 order now has the PARTIALLY_FILLED status.
01.05.21 11:33 BTCUSDT order is placed on Binance.
01.05.21 11:33 Price: 37 323
01.05.21 11:33 Qty: 0.08032
01.05.21 11:33 Side: BUY
01.05.21 11:33 Type: Limit

12

Рисунок В.12 - Слайд 12



Висновки

Під час виконання кваліфікаційної роботи бакалавра було створено торговий термінал, який дозволяє управляти декількома акаунтами з одного місця, а саме створювати заявки та слідкувати за станом їх виконання. Було розв'язано такі завдання:

- аналіз предметної області;
- розробка технічного завдання;
- опис обраних технологій;
- опис обраної архітектури;
- застосування обраної архітектури під час написання додатку;
- розробка додатку;
- тестування додатку.