ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики і інформатики

«До захисту допущено»
Завідувачка кафедри ПМІ
_____ Ольга ДМИТРІЄВА
«_7_» ____червня____ 2022 р.

Випускна кваліфікаційна робота
бакалавра
(освітній ступень)

на тему
«Розробка телеграм-боту «Система моніторингу курсів криптовалют»
зі спецчастиною
Розробка програмних модулів та інтерфейсу засобами Python.

Виконав: студент 4 курсу, групи ІПЗ-18

Спеціальності 121 Інженерія програмного забезпечення

Максим ОБОЛОНСЬКИЙ		
(прізвище та ініціали)		(підпис)
Керівник	зав. каф. ПМІ, д.т.н., проф. Ольга ДМИТРІЄВА	
	(посада, науковий ступінь, вчене звання, ім'я та прізвище)	(підпис)
Рецензент	зав. каф. АТ, д.т.н., доц. Іван ЛАКТІОНОВ	
	(посада, науковий ступінь, вчене звання, ім'я та прізвище)	(підпис)

Нормоконтроль:
_____ Ірина НАЗАРОВА
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Дата 3.06.2022

Студент _____
(підпис)

Луцьк – 2022 р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерних наук і технологій
Кафедра прикладної математики та інформатики
Освітньо-кваліфікаційний рівень (освітній ступінь) бакалавр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ
Ольга ДМИТРІЄВА
/_____/
"____" _____2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Максиму Оболонському
1. Тема проєкту (роботи) Розробка телеграм-боту «Система моніторингу курсів криптовалют».
Спецчастина Розробка програмних модулів та інтерфейсу засобами Python.
Керівник проєкту (роботи) Ольга ДМИТРІЄВА, д.т.н, проф., зав. каф. ПМІ
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "06" травня 2022 року № 180
2. Строк подання студентом проєкту (роботи)_____
3. Вихідні дані до проєкту (роботи) результати науково-дослідної роботи, матеріали переддипломної практики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) аналіз предметної області і постановка завдання;
2) проєктування програмної системи;
3) проєктування та розробка інтерфейсу;
4) розробка системи моніторингу даних веб-сайту;
5) розробка телеграм-боту;
6) тестування боту.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1) блок-схема розробленої системи моніторингу даних веб-сайту;
2) діаграма варіантів використання;
3) екранні форми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 6 квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області і постановка завдання	10.04.2022	
2.	Проєктування системи	15.04.2022	
3.	Проєктування та розробка інтерфейсу	23.04.2022	
4.	Розробка системи моніторингу даних веб-сайту	28.04.2022	
5.	Розробка телеграм-боту	15.05.2022	
6.	Тестування боту	26.05.2022	
7.	Оформлення пояснювальної записки	01.06.2022	
8.	Підготовка слайдів презентації	05.06.2022	
9.	Підготовка демонстраційної версії розробленого програмного продукту	10.06.2022	
10.	Написання доповіді	12.06.2022	

Студент _____ Максим Оболонский
(підпис) (ім'я, прізвище)

Керівник роботи _____ Ольга ДМИТРІЄВА
(підпис) (ім'я, прізвище)

АНОТАЦІЯ

Максим ОБОЛОНСЬКИЙ. Телеграм-бот «Система моніторингу курсів криптовалют» / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення ДВНЗ ДонНТУ, Покровськ, Луцьк, 2022.

Об’єктом дослідження є процеси створення системи моніторингу даних, розробки та тестування боту у месенджері Telegram.

Предметом дослідження є методи та алгоритми функціонування системи моніторингу даних з незалежним агрегатором.

Метою роботи є розробка програмної системи моніторингу курсів криптовалют для месенджера Telegram.

Практичне значення розробки телеграм-боту полягає в створенні компактного помічника для роботи з криптовалютами, основними завданнями якого є здійснення моніторингу курсів валют, візуалізація динаміки поведінки, калькуляція конвертування, обґрунтування пропозицій щодо обміну і т.п.

Методи дослідження, які використані при розробці телеграм-боту, базуються на основних положеннях теорії алгоритмів, емпіричних методах програмної інженерії, принципах об’єктно-орієнтованого програмування, проєктування програмного забезпечення.

Ключові слова: телеграм-бот, система моніторингу, курси криптовалют, мова Python, API, CoinGecko, Aiogram.

ANNOTATION

Maxim OBOLONSKY. Telegram-bot "Cryptocurrency rate monitoring System" / Graduate qualification work for bachelor’s degree in 121 Software Engineering of DonNTU, Pokrovsk, Lutsk, 2022.

The object of research is the processes of creating a system of data monitoring, development and testing of the bot in the Telegram messenger.

The subject of the research is the methods and algorithms of functioning of the data monitoring system with an independent aggregator.

The aim of the work is to develop a software system for monitoring cryptocurrency rates for the Telegram messenger.

The practical significance of telegram-bot development is to create a compact assistant for working with cryptocurrencies, the main tasks of which are to monitor exchange rates, visualize the dynamics of behavior, calculate conversion, justify exchange proposals, etc.

The research methods used in the development of the telegram-bot are based on the basic principles of the theory of algorithms, empirical methods of software engineering, the principles of object-oriented programming, software design.

Keywords: telegram bot, monitoring system, cryptocurrency rates, Python language, API, CoinGecko, Aiogram.

ЗМІСТ

ВСТУП

Починаючи з 2021 року Україна має першість серед країн Європи, щодо впровадження криптовалюти [1].

Незважаючи на численні проблеми, українці останнім часом виявили великий інтерес до цифрових активів. У громадян країн, чия економіка виявляє схильність до інфляції та просідання національної валюти, очевидно з’явиться необхідність частих обмінів у більш стабільні валюти та необхідність актуальної ринкової інформації.

Дивлячись на сучасні реалії стає зрозумілий різкий сплеск популярності нових сервісів соціальних мереж та месенджерів. Користувачі намагаються отримувати, якомога більше корисної інформації через улюблені ними системи.

Одним з таких сервісів є бот у месенджері Telegram, який дозволяє автоматично обробляти та відправляти повідомлення. Боти в Telegram являють собою спеціальні програми, які виконують закладені розробником функції та полегшують життя користувачів.

Зважаючи на обґрунтування, темою кваліфікаційної роботи було обрано розробку телеграм-боту «Система моніторингу курсів криптовалют».

Предметом дослідження є мова Python з бібліотекою Aiogram та незалежний агрегатор даних CoinGecko, який відстежує дані про криптовалюти.

Метою роботи є розробка телеграм-боту.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОНІТОРИНГУ КУРСІВ КРИПТОВАЛЮТ

1.1 Актуальність вибраної теми

Згідно з інформацією платіжною платформою Triple A [2] Україна посіла перше місце у списку країн, населення якої володіє цифровими активами. Більше 5.5 мільйонів українців, а це 12.7% від населення, які володіють криптовалютами.

За оцінками аналітичної блок-чейн компанії Chainalysis [3] Україна має найбільший обсяг транзакцій із цифровими активами та займає 4 місце у списку країн, у яких висока популярність криптовалют (рис. 1.1).

Country	Index score	Overall index ranking	Ranking for individual weighted metrics feeding into Global Crypto Adoption Index		
			On-chain value received	On-chain retail value received	P2P exchange trade volume
Vietnam	1.00	1	4	2	3
India	0.37	2	2	3	72
Pakistan	0.36	3	11	12	8
Ukraine	0.29	4	6	5	40
Kenya	0.28	5	41	28	1
Nigeria	0.26	6	15	10	18
Venezuela	0.25	7	29	22	6
United States	0.22	8	3	4	109
Togo	0.19	9	47	42	2
Argentina	0.19	10	14	17	33
Colombia	0.19	11	27	23	12
Thailand	0.17	12	7	11	76
China	0.16	13	1	1	155
Brazil	0.16	14	5	7	113
Philippines	0.16	15	10	9	80
South Africa	0.14	16	18	16	62
Ghana	0.14	17	32	37	10
Russian Federation	0.14	18	8	6	122
Tanzania	0.13	19	60	45	4
Afghanistan	0.13	20	53	38	7

Рисунок 1.1 – Звіт аналітичної блок-чейн компанії Chainalysis взято з [3]

Щорічно через країну проходить криптовалюта на суму близько 8 мільярдів доларів, а українці заробляють близько 400 мільйонів доларів на продажі біткоіна.

Міністерство цифрової трансформації України заявило, що є кілька причин популярності криптовалюти серед українців: велика спільнота розробників блокчейну та технічно підковане населення загалом, обтяжливі правила експортно-імпортних операцій та відсутність фондового ринку в країні. Роздрібні інвестори цікавляться криптовалютою, оскільки інших варіантів заощаджень та пасивного доходу в Україні не так багато. Економіка мала і немає національного фондового ринку. Все це спонукає людей випробувати цифрові активи.

Зважаючи на такі фактори як загальна кількість власників цифрових активів та глобальний індекс прийняття, стає зрозуміло, що країна має велику ідею стати однією з провідних світових юрисдикцій для криптокомпаній.

У зв’язку з бурхливим інтересом українців до світу криптовалют з’являються запити на прості системи, які дозволять швидко та легко зорієнтуватися на ринковому полі.

Люди все більш намагаються відходити від концепції отримання цифрової інформації тільки від персонального комп’ютера та переходять до компактних смартфонів. Через це постає необхідність кросплатформового додатку для майбутньої системи.

Telegram є кросплатформеним, швидким та захищеним додатком для обміну повідомленнями, який стрімко набирає популярність. Месенджер є 10-ю за популярністю соціальною мережею у світі, а за загальною кількістю завантажень Telegram є 7-м за популярністю мобільним додатком для iOS та Android. Згідно з даними Statista [4] на 2022 рік Telegram має 550 мільйонів користувачів та більше 55 мільйонів активних користувачів за добу (рис. 1.2).

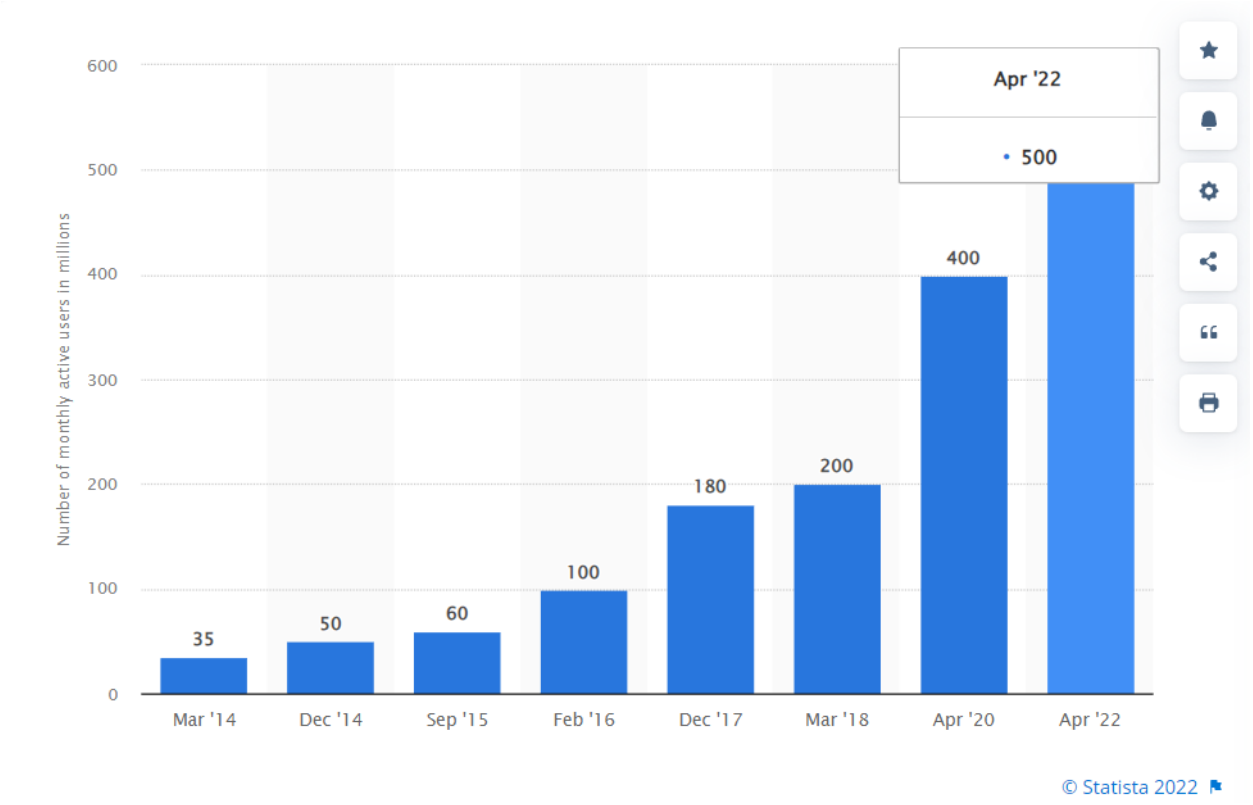


Рисунок 1.2 – Графік користувачів Telegram за даними Statista взято з [4]

Фактично, база користувачів Telegram зростає більш ніж на 40% щороку з моменту його запуску в 2013 році.

Перевагою Telegram для розробки системи моніторингу криптовалют є можливість створення боту – програми, яка автоматично, за командою або заданим розкладом, згідно з зазначеними параметрами виконує певні дії.

1.2 Мета роботи та практичне значення

Метою роботи є розробка програмної системи моніторингу курсів криптовалют для месенджера Telegram.

Практичне значення розробки телеграм-боту полягає в створенні компактного помічника для роботи з криптовалютами, основними завданнями якого є здійснення моніторингу курсів валют, візуалізація динаміки поведінки, калькуляція конвертування, обґрунтування пропозицій щодо обміну і т.п.

1.3 Методи, об’єкт та предмет дослідження

Об’єктом дослідження є процеси створення системи моніторингу даних, розробки та тестування боту у месенджері Telegram.

Предметом дослідження є методи та алгоритми функціонування системи моніторингу даних з незалежним агрегатором.

Методи дослідження, які використані при розробці телеграм-боту базуються на основних положеннях теорії алгоритмів, емпіричних методах програмної інженерії [5-6], принципах об’єктно-орієнтованого програмування [7-9], проєктування програмного забезпечення [10-11].

1.4 Аналіз конкурентних продуктів

З розвитком ботів у месенджері Telegram та зацікавленістю користувачів світом криптовалют, з’явилося чимало різноманітних версій ботів для роботи з цифровими активами.

Функціонал переважної кількісті з них дуже обмежений, через те, що розробкою займається незалежний розробник без фінансування. Такі боти, як правило, мають змогу видавати лише необхідний курс певної монети відносно однієї фіатної валюти.

Також існують боти, які створюються на замовлення для великих фінансових компаній та бірж.

Яскравим прикладом таких компаній є Cryptochan [12] продукт якої має великий функціонал та здатний видавати окрім курсу, волатильність монети, її мінімальну та максимальну ціну за добу, будувати графіки криптовалют та видавати дані цифрових активів з багатьох бірж.

На рисунках 1.3 – 1.4 зображено роботу функцій курс та волатильність.

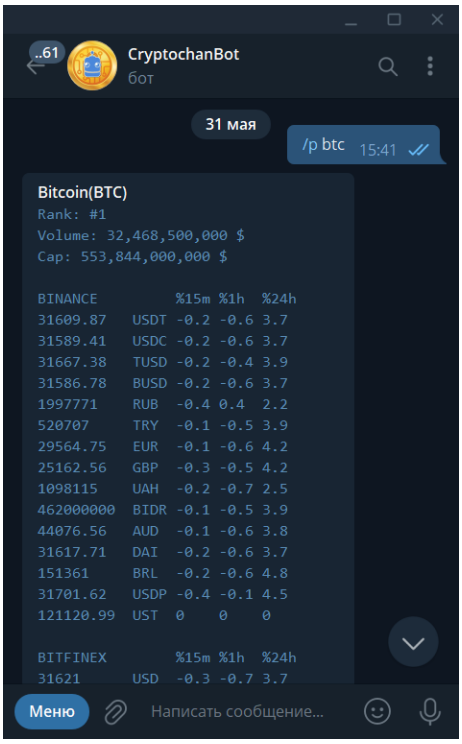


Рисунок 1.3 – Курс монети біткоін у CryptochatBot, взято з [12]

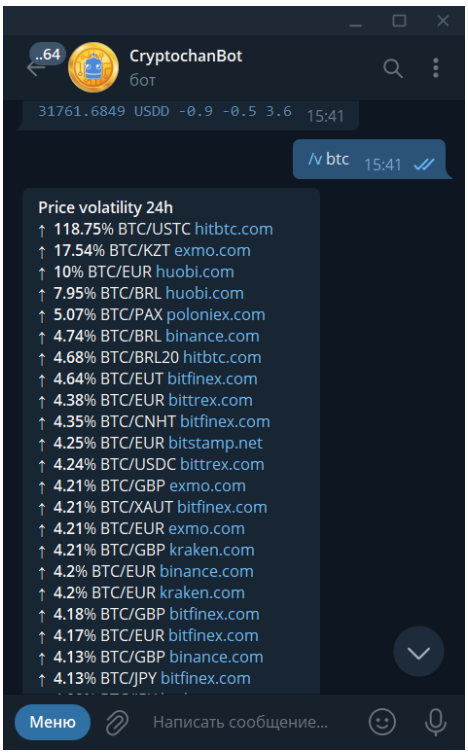


Рисунок 1.4 – Волатильність монети біткоін CryptochatBot, взято з [12]

На рисунках 1.5 – 1.6 продемонстровано роботу команд виводу максимальної та мінімальної вартості монети за день та побудову графіку.

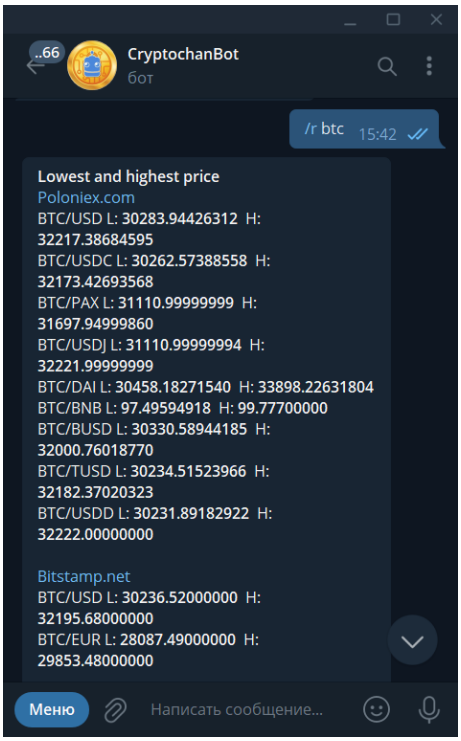


Рисунок 1.5 – Максимальна та мінімальна вартість монети біткоін за добу у CryptochatBot, взято з [12]

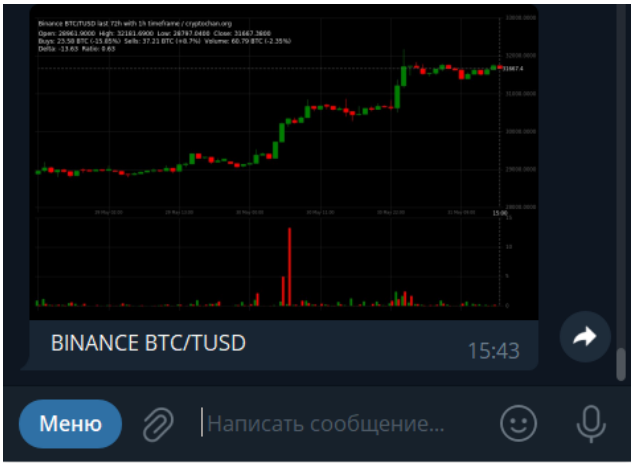


Рисунок 1.6 – Графік ціни у CryptochatBot, взято з [12]

1.5 Аналіз принципів і положень методів дослідження та технологій для розробки системи

Система телеграм-боту передбачає збір та обробку даних про курс криптовалют, а саме ціни, ринки, об’єми торгів, добові зміни курсу та історичні дані монет за певний проміжок часу.

Через це необхідно застосувати прикладний програмний інтерфейс(API) [13-14], який дозволить отримати доступ до необхідних даних певного ресурсу.

Фільтрована інформація про цифрові активи буде оброблятися в залежності від вимог та виводитися через створений інтерфейс користувачеві.

Телеграм-бот після обробки даних з веб-сайту повинен виводити поточний курс криптовалют відносно фіатної валюти, на основі об’єму торгів, ціни та проміжку часу візуалізувати динаміку поведінки монети.

Функціонал системи передбачає пошук та конвертацію криптовалют. Через необхідність запам’ятовувати назву чи кількість цифрових активів потрібно буде застосувати машину зі скінченною кількістю станів.

Візуалізація даних потребує побудови графіку, його подальше збереження і вивід.

Перед розробкою системи необхідно спроєктувати роботу функцій за допомогою UML-діаграм та блок-схем, та створити макети віртуального інтерфейсу.

1.5.1 Telegram-bot

Боти [15] – це невеликі сторонні програми, які можуть вбудовуватись у чати Telegram або загальнодоступні канали та виконувати певну функцію. Взаємодія з ботами відбуваються шляхом надсилання їм повідомлень, команд та запитів. Керування ботами відбувається за допомогою HTTPS-запитів до Telegram Bot API.

Написані для кросплатформеної системи Telegram, вони призначені для виконання різноманітних функцій: від інформування користувачів про ті чи інші події до торгівлі валютами. Крім цього, боти Telegram також можуть забезпечувати підтримку клієнтів або збирати потенційних клієнтів, підключаючи їх до системи управління взаємовідносинами із клієнтами, системи продажу квитків або платформи обміну повідомленнями.

Телеграм-боти діляться на декілька категорій:

– чат-боти;
– боти-інформатори;
– ігрові боти;
– боти-помічники.

Хоча більшість з них не мають чіткої категорії, бо поєднують у собі одночасно декілька механік.

1.5.2 Переваги об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування (ООП) [7-9] – це стиль програмування, що характеризується ідентифікацією класів об'єктів, що тісно пов'язані з методами (функціями). Він також включає ідеї успадкування атрибутів та методів.

Одна з основних переваг методів об'єктно-орієнтованого програмування проти процедурного програмування у тому, що вони дозволяють програмістам створювати модулі, які потрібно змінювати при додаванні нового типу об'єкта. Програміст може просто створити новий об'єкт, який успадковує властивості існуючих об'єктів. Це полегшує модифікацію об'єктно-орієнтованих програм.

Інкапсуляція: процес об'єднання елементів для створення нового об'єкта. Процедура є типом інкапсуляції, оскільки вона поєднує низку комп'ютерних інструкцій.

Для взаємодії з API, інкапсуляція є важливим фактором через необхідність ізолювати та приховувати певні дані компоненту та його деталі від інших компонентів програми (рис. 1.7).

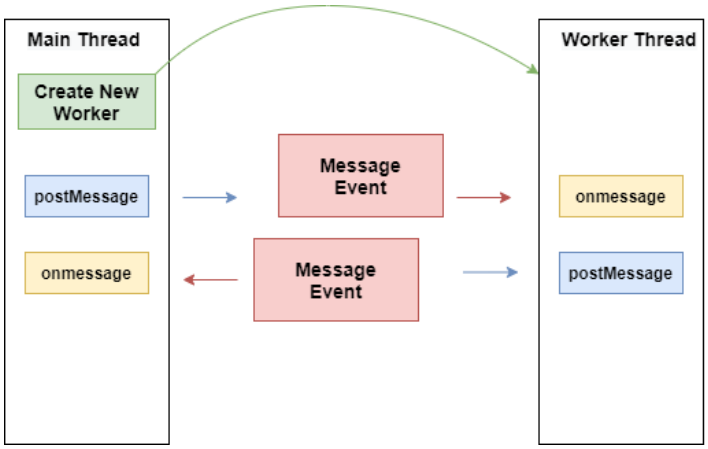


Рисунок 1.7 – Принцип дії інкапсуляції

Клас в об'єктно-орієнтованому програмуванні – категорія об'єктів. Клас визначає всі загальні властивості різних об'єктів, що йому належать (рис. 1.8).

Рисунок 1.9 – Успадкування класів

1.5.3 Інтерфейс прикладного програмування

Інтерфейс прикладного програмування (API) [13-14] являє собою набор програмного коду, який забезпечує доступ до даних програмного додатку, служб або операційних систем. API по суті є посередником для різних програмних платформ, які використовують одні і ті самі дані (рис. 1.10).

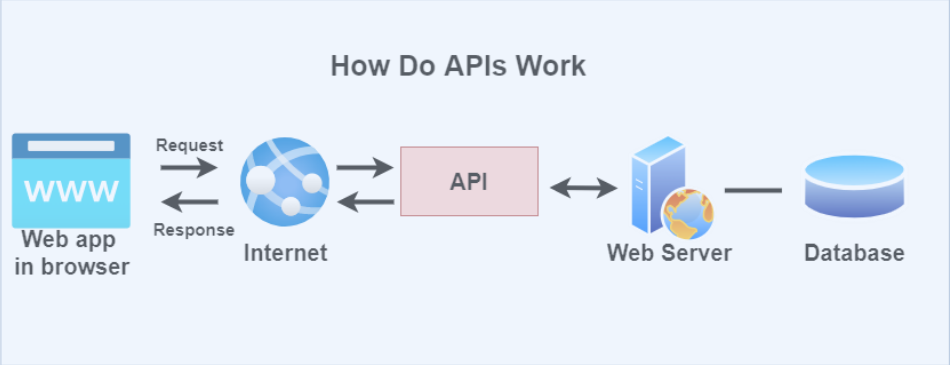


Рисунок 1.10 – Принцип дії API

Споживачами API можуть бути компанії, або окремі особи, які розробляють програмне забезпечення.

Наприклад, торгова біржа може створити власний API, який дозволить окремим розробникам отримувати доступ до їх даних і створити свій продукт. В результаті користувачі API розраховують на власника даних, який може обмежувати деталізацію даних або стягувати за них платню.

1.5.4 Можливості використання машини станів

Кінцевий автомат – це математична абстракція, що використовується для розробки алгоритмів. Кінцевий автомат зчитує набір вхідних даних і переходить на інший стан з урахуванням цих вхідних даних.

Стан – це опис системи, що очікує на виконання переходу. Перехід – це набір дій, які виконуються під час виконання умови або отримання події (рис. 1.1).

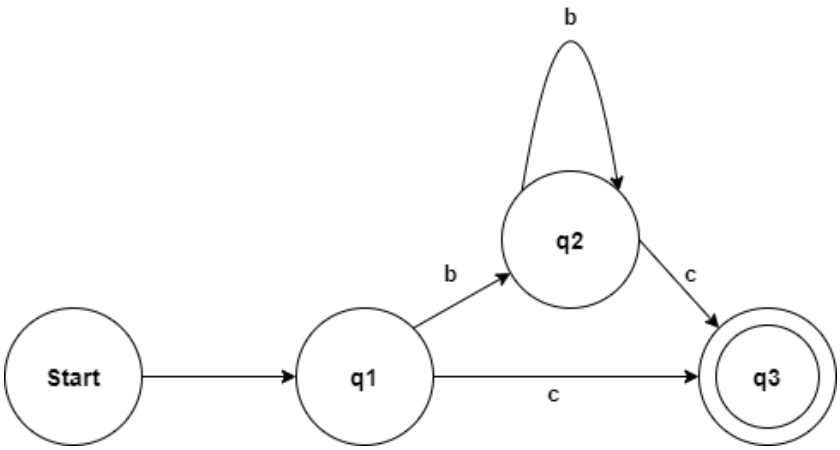


Рисунок 1.10 – Принцип дії кінцевих автоматів

1.5.5 Аналіз підходів до проєктування програмного забезпечення

Проєктування програмного забезпечення [10-11] – це процес перетворення потреб користувача в деяку відповідну форму, яка допомагає програмісту в кодуванні та реалізації програмного забезпечення.

Загалом процес проєктування програмного забезпечення ділиться на три рівні етапи проєктування.

Архітектурне проєктування – це специфікація основних компонентів системи, їх обов'язків, властивостей, інтерфейсів, а також взаємозв'язків та взаємодій між ними. Під час архітектурного проєктування вибирається загальна структура системи, але ігноруються внутрішні деталі основних компонентів.

Проєктування дизайну [16] – це специфікація взаємодії між системою та її оточенням. Ця фаза протікає на високому рівні абстракції стосовно внутрішньої роботи системи, тобто під час проєктування інтерфейсу внутрішня частина системи повністю ігнорується, і система сприймається як чорний ящик. Увага зосереджена на діалозі між цільовою системою та користувачами, пристроями та іншими системами, з якими вона взаємодіє.

Детальне проєктування – це процес, який орієнтований на використання базової структури, що розбивається на проєктування задач, процесів, об’єктів, класів та модулів.

Модульність – це метод поділу програмної системи на кілька дискретних і незалежних частин, які, як очікується, будуть здатні виконувати завдання незалежно один від одного. Ці модулі можуть працювати як базові конструкції для програмного забезпечення.

1.5.6 Застосування емпіричних методів програмного забезпечення

Програмна інженерія – це не лише технічні рішення. Емпіричний метод [5-6] значною мірою також пов'язаний із організаційними питаннями, управлінням проєктами та людською поведінкою. Для такої дисципліни як розробка програмного забезпечення, емпіричні методи мають вирішальне значення, оскільки вони дозволяють включити людську поведінку до дослідницького підходу. Емпіричні методи поширені у багатьох інших дисциплінах.

Існує два основних типи дослідницьких парадигм, які мають різні підходи до емпіричних досліджень.

Якісні дослідження пов'язані з вивченням об'єктів у їх природній середовищі. Якісне дослідження починається з прийняття тези, що існує цілий ряд різних способів інтерпретації. Це стосується виявленню причин, помічених суб'єктами дослідження, і розуміння їх з точки зору розглянутої проблеми.

Кількісні дослідження в основному пов’язані з кількісною оцінкою взаємозв’язків. Кількісні дослідження часто проводяться шляхом встановлення контрольованих експериментів або збору даних за допомогою тематичних досліджень. Дослідження є доречними, коли перевіряється ефект деяких маніпуляцій або діяльності.

1.6 Основні завдання роботи

Для досягнення поставленої мети необхідно виконати наступні завдання роботи:

– провести об’єктний аналіз предметної області, дослідити сучасні технології створення телеграм-ботів;

– розглянути існуючи алгоритм розробки систем моніторингу даних та дослідити особливості систем моніторингу, які застосовуються на криптовалютному ринку, провести порівняльний аналіз розглянутих систем;

– спроєктувати програмну систему моніторингу курсів криптовалют для месенджера Telegram;

– обґрунтувати програмні та апаратні вимоги до розроблюваної системи;

– отримати програмну реалізацію розроблювального додатку на мові Python з бібліотекою Aiogram та з незалежним агрегатором CoinGecko, який відстежує дані про криптовалюту;

– отримати програмну реалізацію модулів інтерфейсу телеграм-боту;

– провести тестування програмної системи для роботи з криптовалютами.

1.7 Висновки за розділом

У цьому розділі було проведено аналіз кваліфікаційної роботи. Обґрунтовано тему та її актуальність, було визначено об’єкт, предмет та методи дослідження.

У розділі проведено аналіз конкурентних продуктів, визначено, що більшість схожих телеграм ботів мають менш обширний функціонал, проте є боти, які були створені з більш професійним підходом та фінансуванням.

Проведено аналіз та обґрунтування принципів та положень методів дослідження та технологій розробки.

Було детально описано принцип роботи інтерфейсу прикладного програмування, принципи дії інкапсуляції та успадкування класів.

Детально розглянуто етапи проєктування програмного забезпечення. Також було описано положення емпіричних методів програмного забезпечення.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Опис архітектури системи

Для створення обраної системи, перш за все необхідно зареєструвати акаунт у месенджері Telegram. Після цього у месенджері потрібно знайти головного бота на ім’я FatherBot і набрати команду /newbot та ввести user name та account name свого боту. Бот згенерує унікальний токен, який знадобиться на наступних етапах. За допомогою команди /setuserpic можна обрати іконку для власного боту.

Для роботи з токеном боту на мові програмування Python [17-18] реалізовано декілька бібліотек, а саме:

– aiogram;
– python-telegram-bot;
– telepot;
– telegram Bot Service;
– telebot;
– twx.botapi;
– pyTelegramBotAPI.

Бібліотека Aiogram, відрізняється швидкістю роботи та асинхронністю. Aiogram потребує інтерпретатор мови Python версії не нижче 3.6.

Для того, щоб скористатися API і отримати потрібні дані існує незалежний агрегатор даних криптовалют Coingecko [19]. Id монет з агрегатора зберігається у окремому файлі CoinGecko, Token API list [20].

Щоб з великою системою було легше працювати, її слід розбити на декілька компонентів – модулів. Необхідно розумно ділити систему на модулі так, щоб кожен модуль вирішував деяке відносно ізольоване завдання. Система боту буде розбита за технічним критерієм.

Для зручнішого користування системою було створено одну команду /start, яка викликає головне меню, створене на базі асинхронних кнопок, які не заважають працювати одна одній.

До окремих модулів потрібно винести процеси створення, запуску боту, а також створення віртуальної клавіатури та кнопок.

Функція моніторингу курсів криптовалют, а саме пошуку бажаної монети та виведення інформації про піковий курс за добу, найнижчий курс за добу, добову зміну курсу для топу криптовалют винесено в окремий модуль.

Для реалізації функції пошуку криптовалюти потрібне використання кінцевих автоматів для запам’ятовування введеної користувачем назви криптовалюти.

При вирішенні складних завдань завчасне планування та моделювання значно спрощують програмування, тому використано мову проєктування UML для створення докладних діаграм для певних процесів.

Нижче на рисунку 2.1 продемонстровано діаграму варіантів використання. Діаграму діяльності показано на рисунку 2.2.

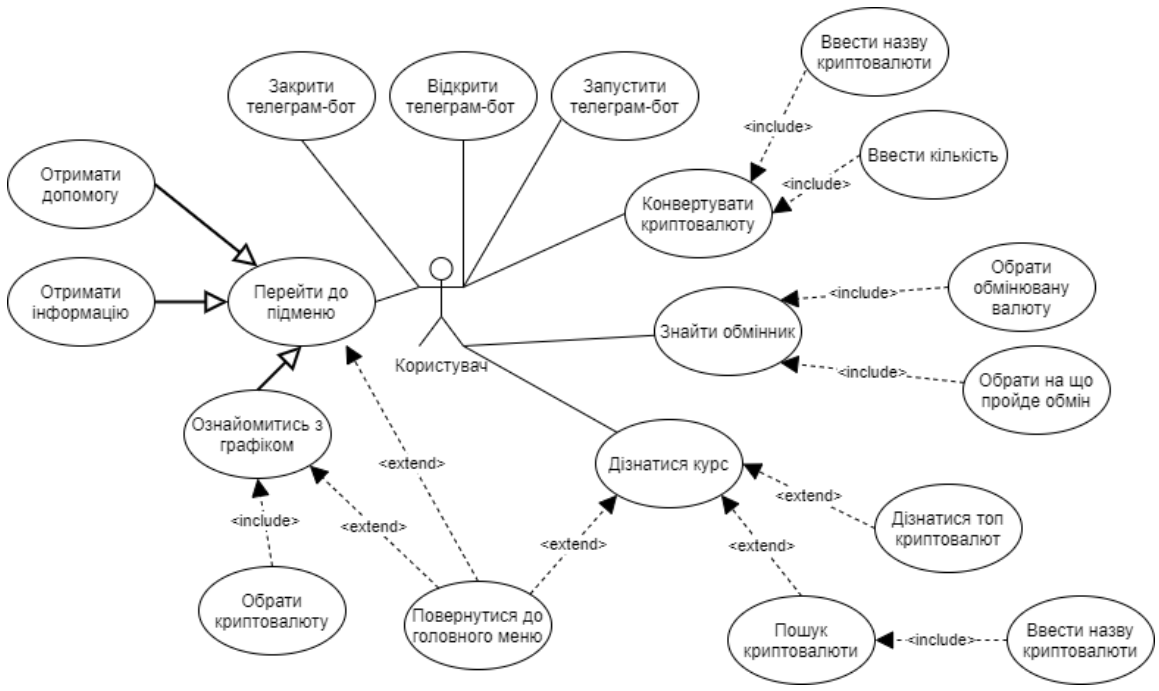


Рисунок 2.1 – Діаграма варіантів використання для актора «Користувач»

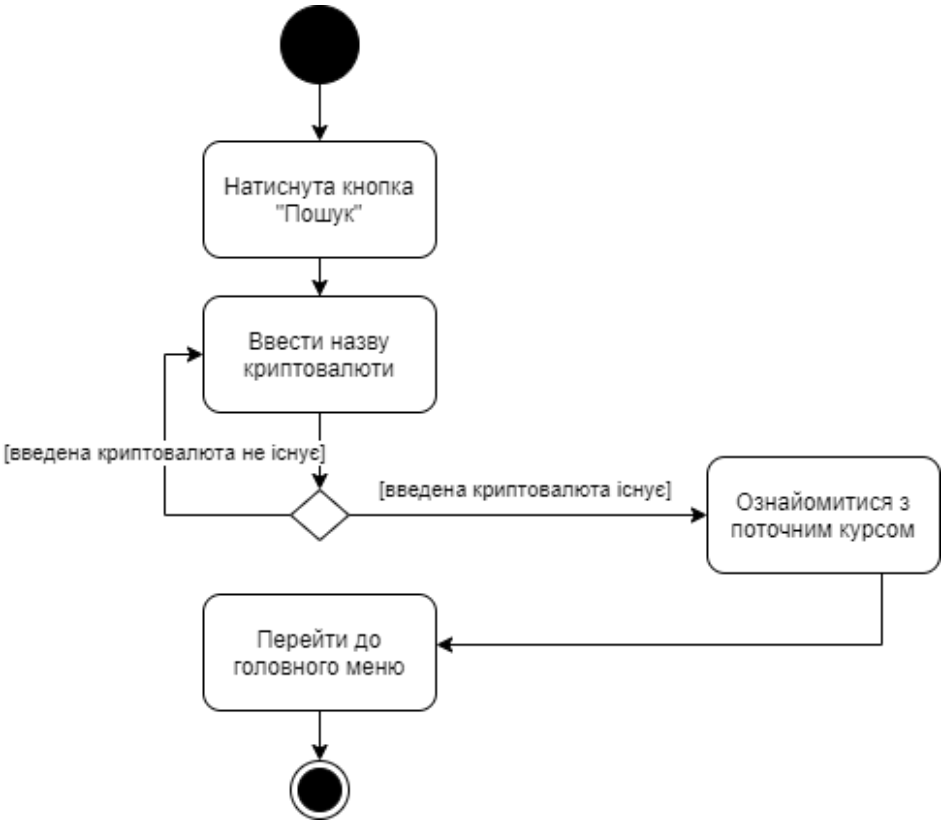


Рисунок 2.2 – Діаграма послідовності для актора «Користувач»

На рисунку 2.3 зображено діаграму кооперації функції «Топ» для актора «Користувач».

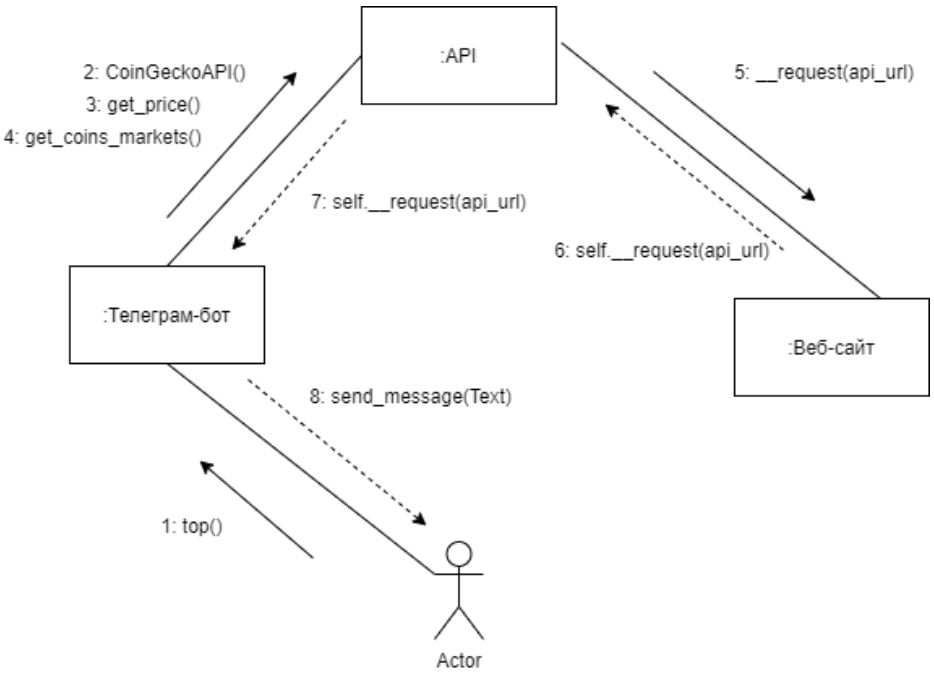


Рисунок 2.3 – Діаграма кооперації для актора «Користувач»

Функція калькуляції конвертування криптовалют була винесена до окремого модуля. Користувач натиснувши кнопку «Конвертер» має вписати назву потрібної монети, після чого необхідно буде вказати кількість. У разі, якщо користувач неправильно ввів назву криптовалюти чи її кількість система запропонує повторну спробу вводу.

Реалізація функції конвертування передбачає використання машини зі скінченною кількістю станів, через необхідність запам’ятовувати введену користувачем назву та кількість криптовалюти.

На рисунку 2.4 представлено діаграму станів. Діаграма послідовності показана на рисунку 2.5.

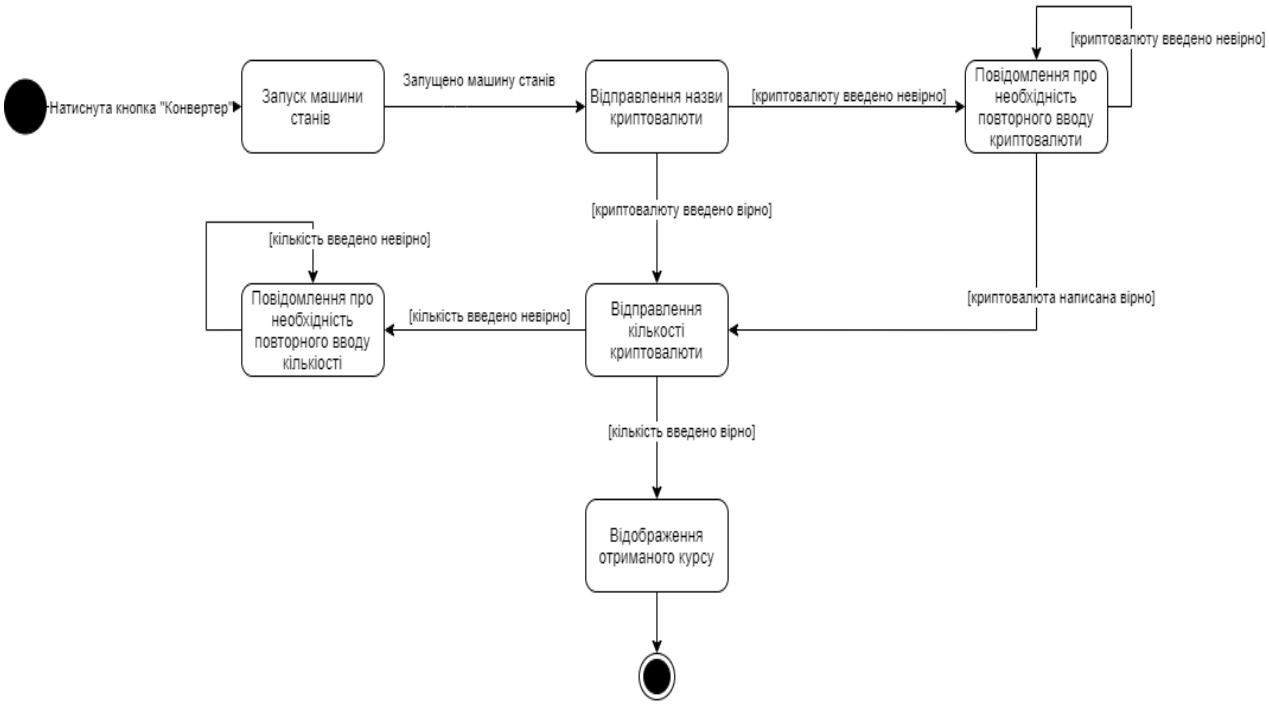


Рисунок 2.4 – Діаграма станів для функції конвертування

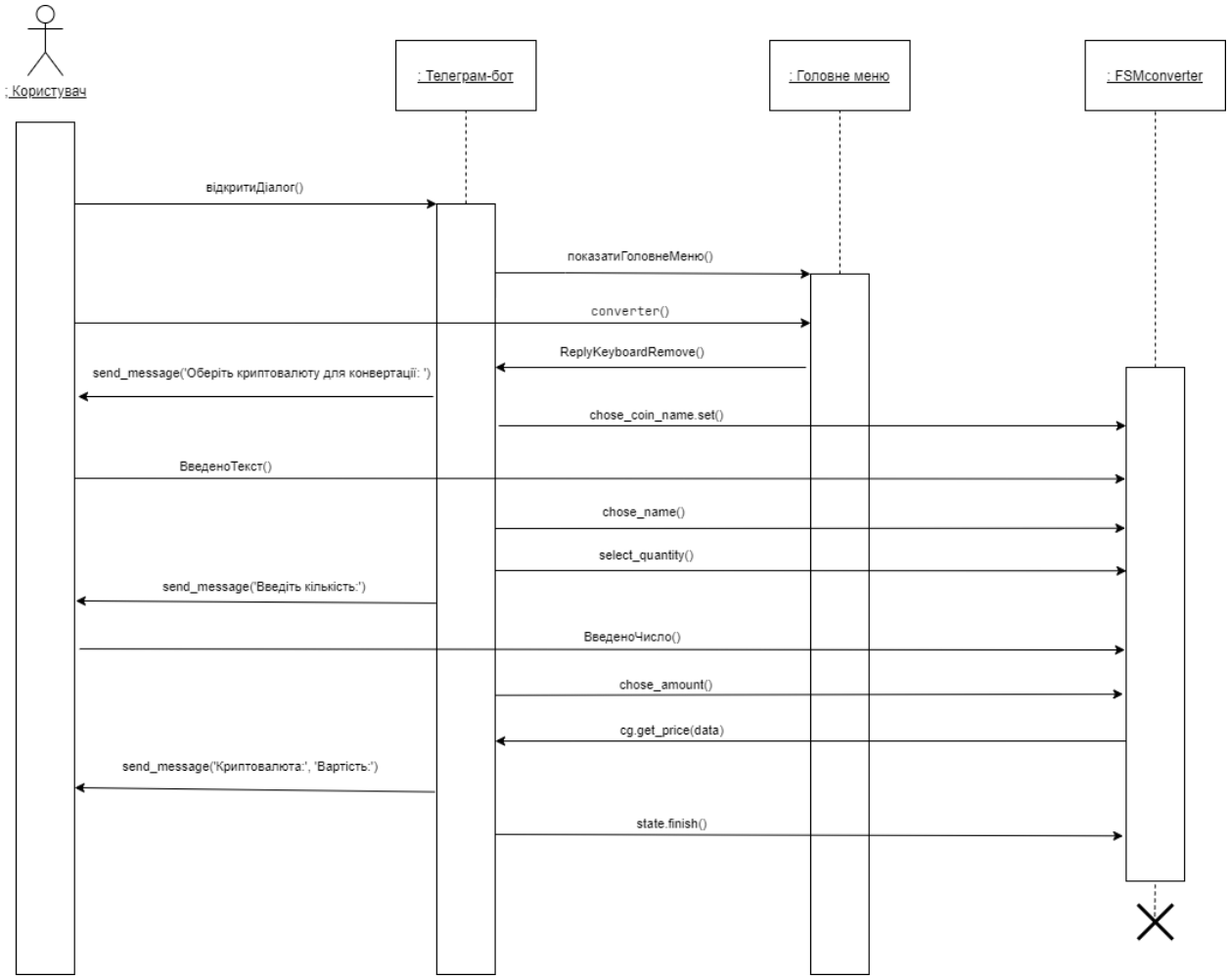


Рисунок 2.5 – Діаграма послідовності функції конвертування

Візуалізацію динаміки поведінки криптовалюти винесено в окремий модуль. Для побудови графіку необхідно обробити дані з ціною, об’ємом торгів за певний проміжок часу. Необхідно використати бібліотеки NumPy [21] та Pandas [22], які відповідають за роботу з даними та наукові обчислення. За візуалізацію та збереження побудованого графіку відповідають бібліотека Altair [23] та програмна платформа Node.js [24].

Система моніторингу криптовалют буде розміщена на локальному сервері, для взаємодії користувачу необхідно буде встановити месенджер Telegram на свій ПК чи смартфон.

Незалежний агрегатор даних криптовалют забезпечить систему необхідними даними за допомогою API.

На рисунку 2.6 побудована діаграма розміщення.

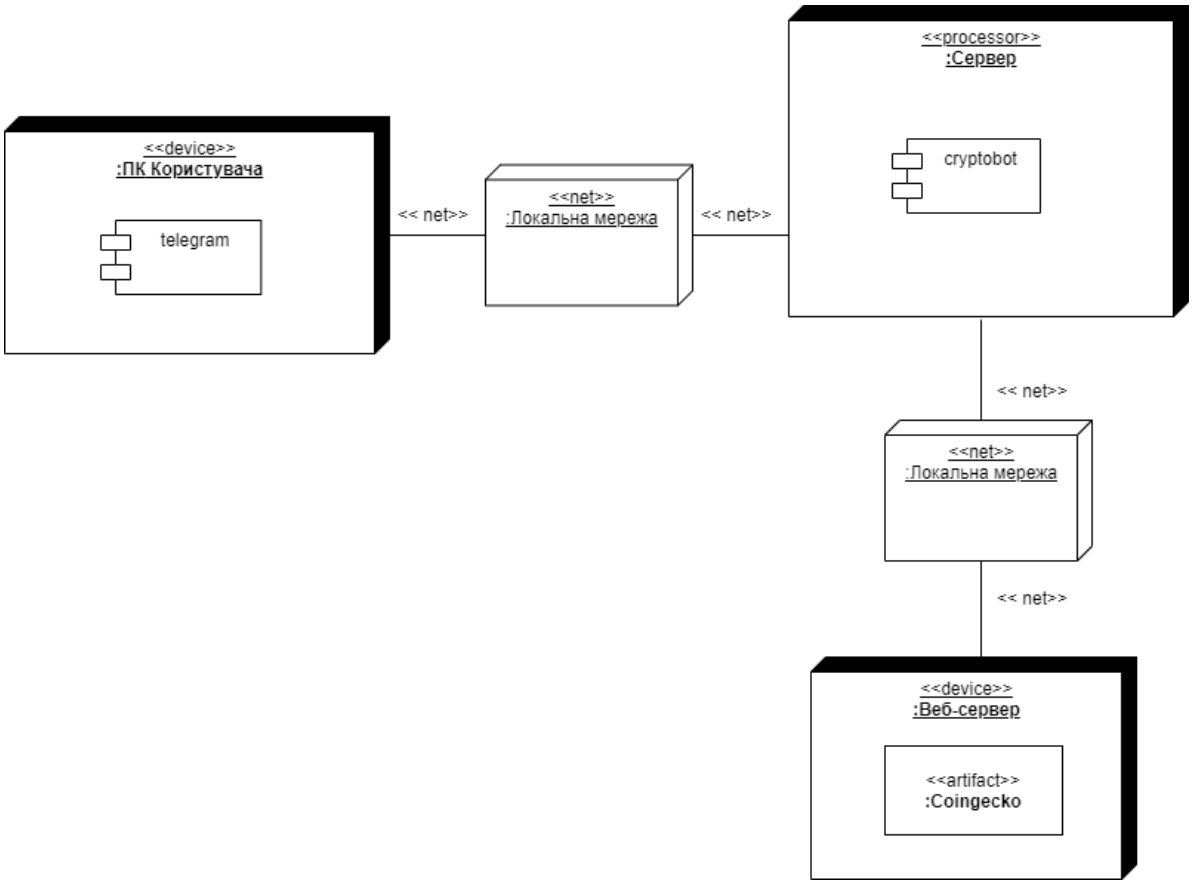


Рисунок 2.6 – Діаграма розміщення

2.1.1 Визначення середовища розробки

PyCharm [25] є інтегрованим середовищем розробки для Python з додатковим набором програмних засобів для оптимізації продуктивності. Середовище розробки має функцію автозаповнення, яка видає пропозиції під час написання коду. Це робить написання коду більш ефективним, швидким і менш схильним до помилок. Завдяки різним колірним схемам для ключових слів, класів і функцій підвищується рівень розуміння та зручності читання коду. Переваги даного середовища можна виділити:

– вікно редактора проєкту для ефективного управління та організації файлів, які необхідні для програми/проєкту;

– перевірка виведення коду, написаного за допомогою вікна виводу;

– пропозиції щодо усунення помилок та попереджень;

– ряд модулів та пакетів, які доступні в одному місці.

На рисунках 2.7 – 2.8 зображено інтерфейс PyCharm та завантажена бібліотека Aiogram відповідно.

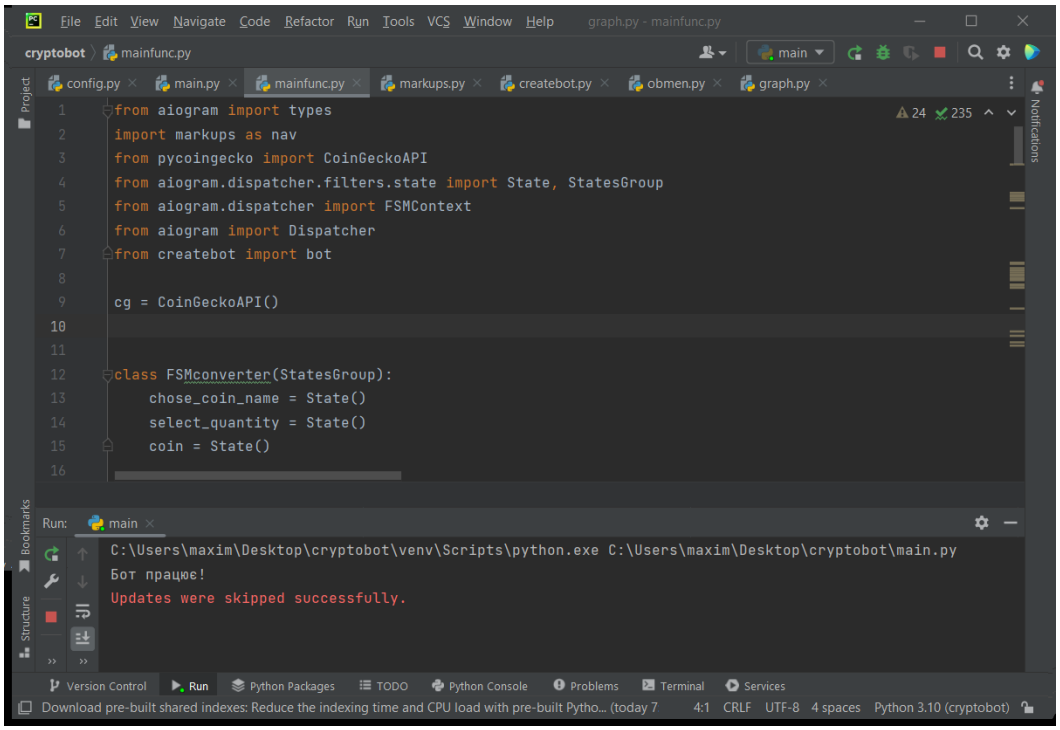


Рисунок 2.7 – Вікно середовища розробки PyCharm

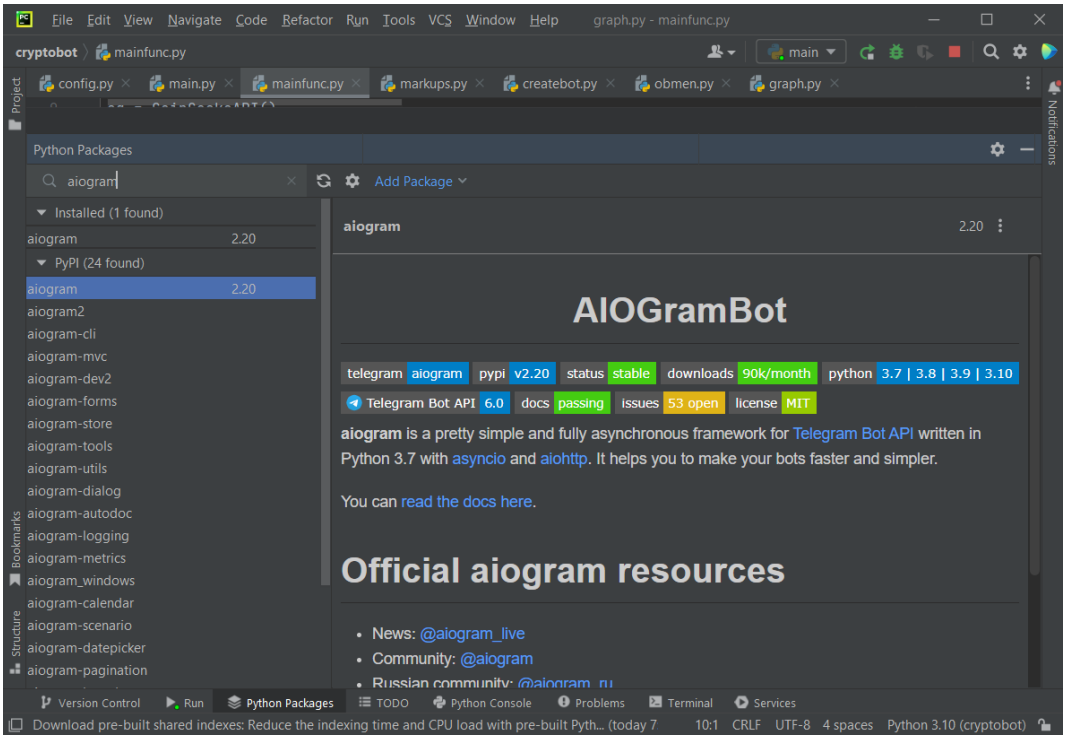


Рисунок 2.8 – Встановлена бібліотека Aiogram

2.1.2 Обґрунтування обрання мови програмування.

Python [17-18] – це високорівнева інтерпретована об'єктно-орієнтована мова програмування загального призначення.

Python – універсальна мова, яку можна використовувати практично для всього. Він застосовується практично в кожній області для різних завдань. Будь-то виконання таких короткострокових завдань, як тестування програмного забезпечення чи довгострокова розробка продукту.

Python спирається на структури, що прості для розуміння, які пізніше перекладаються на мову низького рівня. Вихідний код запускається на центральному процесорі комп'ютера (ЦП).

Ця мова також широко застосовується в науковій галузі для теоретичних досліджень та вирішення прикладних завдань. Мова підтримує модулі та пакети, що сприяє модульності програм та повторному використанню коду.

Переваги мови програмування Python:

- простий синтаксис;
- потужний набір інструментів;
- швидкість розвитку;
- гнучкість;
- портативність.

2.1.3 Візуалізація даних бібліотекою Altair

Altair [23] – це декларативна, статистична бібліотека візуалізації для Python, яка заснована на граматиках візуалізації Vega та Vega-Lite та дозволяє створювати інтерактивні моделі візуалізації даних. Altair має різноманітний функціонал у плані перетворення даних. Бібліотека надає змогу використовувати безліч різних видів перетворень під час створення візуалізації. Це робить бібліотеку ще більш ефективнішою для дослідницького аналізу даних. Основні переваги бібліотеки:

– базовий код однаковий для всіх типів графіків, користувачеві потрібно лише змінити атрибут мітки, щоб отримати різні графіки;

– код коротший і простий у написанні, ніж у інших бібліотеках імперативної візуалізації;

– огранювання та інтерактивність дуже прості у реалізації.

Для того, щоб зберегти створений графік необхідно використати пакет Altair Saver, який дозволяє зберігати діаграми в різні типи даних такі як:

– .json;

– .png;

– .html;

– .svg;

– .pgf;

– .vg.json.

На рисунку 2.9 приведено побудований графік за допомогою бібліотеки Altair.

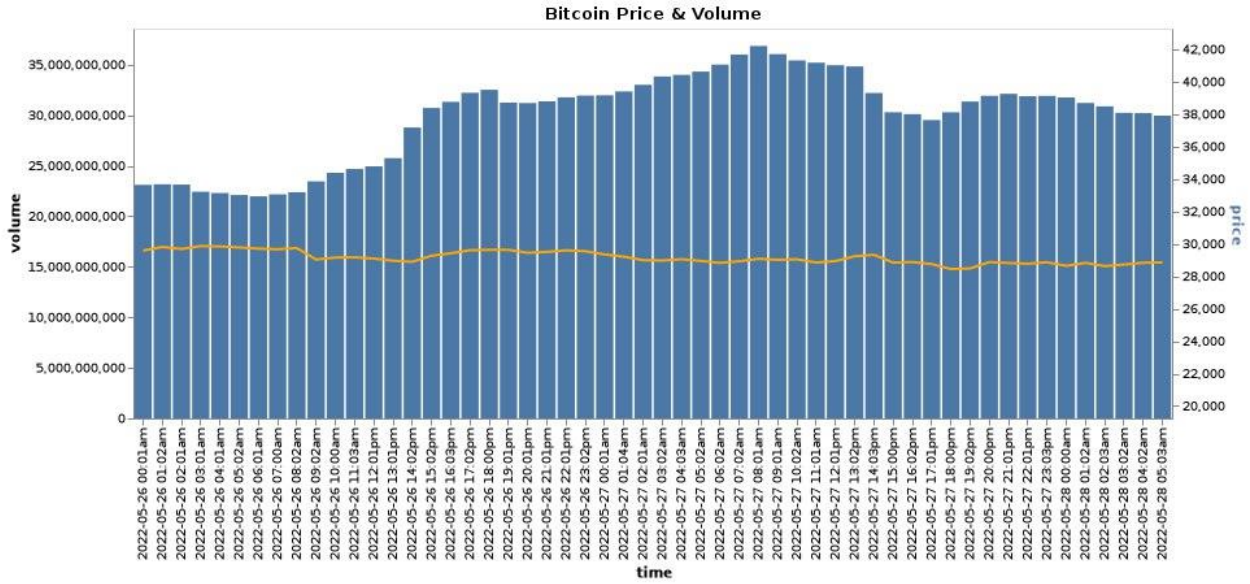


Рисунок 2.9– Графік біткоіна побудований бібліотекою Altair

2.1.4 Застосування бібліотеки NumPy

NumPy [21] – це бібліотека для роботи з масивами Python. Вона додає підтримку багатовимірних масивів та надає інструменти для роботи з цими масивами. Бібліотека прагне надати об'єкт масиву, який працює до 50 разів швидше ніж традиційні списки Python.

Масиви NumPy зберігаються в одному безперервному місці в пам'яті, на відміну від списків, тому процеси можуть ефективно звертатися до них і маніпулювати ними. Це основна причина, через яку NumPy працює швидше, ніж списки. Також бібліотеку оптимізовано для роботи з останніми архітектурами ЦП.

Основні категорії маніпуляцій з масивами:

- атрибути масивів;
- індексування масивів;
- нарізка масивів;
- зміна форми масивів;
- об'єднання та поділ масивів.

2.1.5 Основні приципи роботи з даними бібліотеки Pandas

Для побудови графіку криптовалюти обрано бібліотеку Pandas [22], яка призначена для роботи з даними Python. Бібліотека забезпечує готовність використовувати високопродуктивні структури даних та інструменти аналізу даних.

Набір даних – це перший аргумент, який ви передаєте діаграмі. Дані Altair побудовані на основі Pandas Dataframe, тому кодування стає досить простим, і бібліотека може визначати типи даних, необхідні для кодування. У Pandas можна вивчити набір даних, почистити його, внести зміни, проаналізувати та зробити висновки, побудувати графіки та багато іншого.

В екосистемі Python, Pandas є найбільш просунутою бібліотекою, що швидко розвивається та дозволяє обробляти та аналізувати дані.

2.1.6 Обґрунтування вибору платформи Node.js

Для того, щоб зберегти діаграму у форматі PNG необхідно встановити програмну платформу Node.js [24].

Node.js – це програмна платформа для масштабованих серверних та мережевих програм. Програми Node.js призначені для забезпечення максимальної пропускної здатності та ефективності за рахунок використання неблокуючого введення-виводу та асинхронних подій. Програми Node.js працюють у однопотоковому режимі, хоча Node.js використовує кілька потоків для файлових та мережевих подій. Node.js зазвичай використовується для програм реального часу через його асинхронний характер.

Підтримуються такі вихідні формати:

– .png;

– .svg;

– .pgf;

– .vg.json.

На рисунку 2.10 приведено перевірку роботи Node.js.

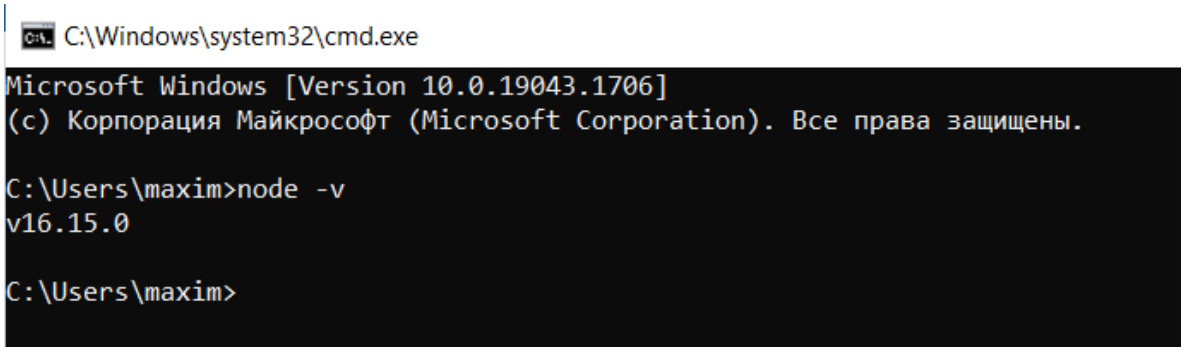


Рисунок 2.10 – Перевірка роботи та актуальної версії Node.js

2.2 Переваги агрегатора CoinGecko

CoinGecko [19] безкоштовний агрегатор даних про криптовалюти, який являє собою набір надійних API-інтерфейсів. Будь-який розробник може використовувати API від CoinGecko не тільки для покращення своїх існуючих

програм та сервісів, але й для створення просунутих програм для крипто-ринку. Основні переваги агрегатора:

– безкоштовний крипто API;
– непотрібність ключів допуску;
– надійність;
– великі ринкові дані.

На рисунка 2.11 зображено запити незалежного агрегатора Coingecko.

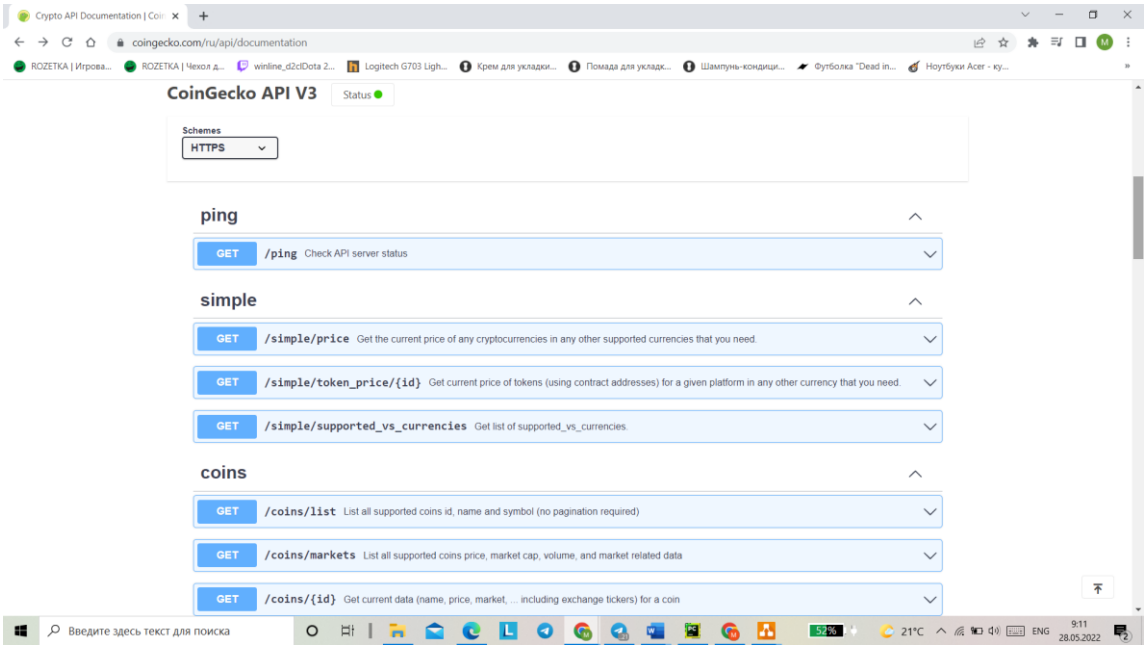


Рисунок 2.11 – Запити незалежного агрегатора Coingecko

2.3 Проєктування інтерфейсу

Проєктування інтерфейсу користувача – це створення тестової версії програми. Це початковий етап розробки інтерфейсу користувача, коли ми розподіляємо функції програми по екранах, визначаємо макет екранів, вміст, елементи управління та їх поведінку.

Макет інтерфейсу – це візуальне статичне уявлення концепції інтерфейсу користувача.

Для побудови макетів інтерфейсу було обрано інструмент Draw.io [26]. Draw.io – інструмент, який дозволяє створювати блок-схеми, мережеві

діаграми, інтелект-карти, відносини сутностей, програмні блоки, UML, макети і т.д. Багата функціональність Draw.io дозволяє користувачам відстежувати зміни, імпортувати і експортувати в PDF, PNG, XML, VSDX, HTML, а також автоматично публікувати і ділитися роботами.

Особливості:

– великий набір шаблонів;
– швидке створення блоків;
– інтуїтивно зрозумілий інтерфейс;
– призначені для користувача бібліотеки;
– діаграми процесів;
– створення організаційної схеми;
– бібліотека форм;
– макети;
– автоматична розкладка;
– відстеження змін і відновлення;
– каркасні моделі;
– багатосторінкові діаграми;
– мережеві діаграми.

Для побудови макетів інтерфейсу, використано концепцію грубого макету.

Це макет низької точності, який отримано в результати проєктування інтерфейсів. Грубі макети мають відображати порядок, структуру, та розташування елементів на екрані. Грубі макети інтерфейсу мають вигляд динамічного прототипу, який можна використовувати для тестування ергономічності або початку розробки програми.

Нижче на рисунках 2.12 – 2.13 приведена демонстрація макетів головного меню та роботи кнопки «Топ».

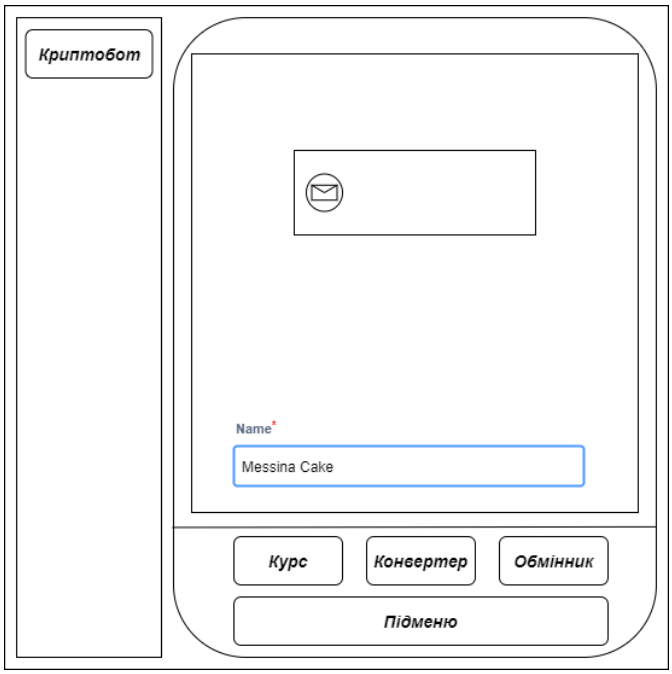


Рисунок 2.12 – Макет головного меню

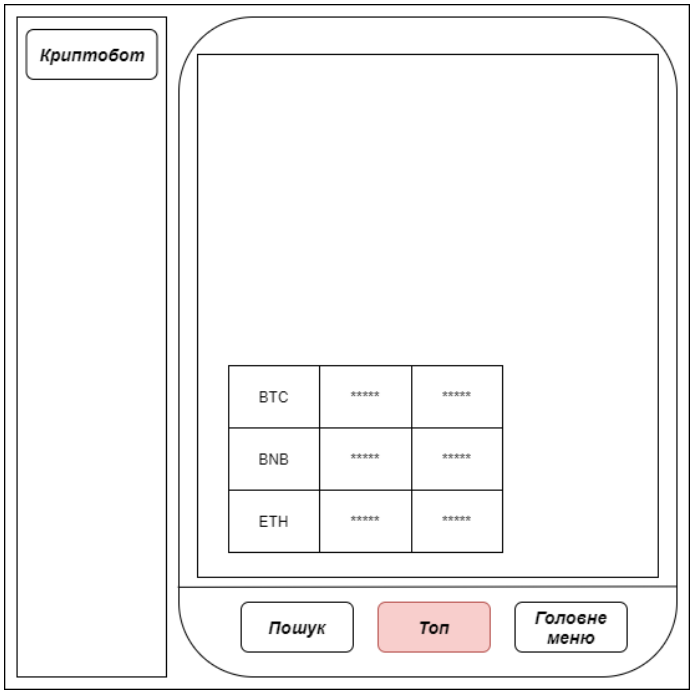


Рисунок 2.13 – Макет меню кнопки «Курс» та робота кнопки «Топ»

На рисунках 2.14 – 2.16 зображено макети підменю та роботи кнопок «Конвертер», «Обмінник» та «Графік».

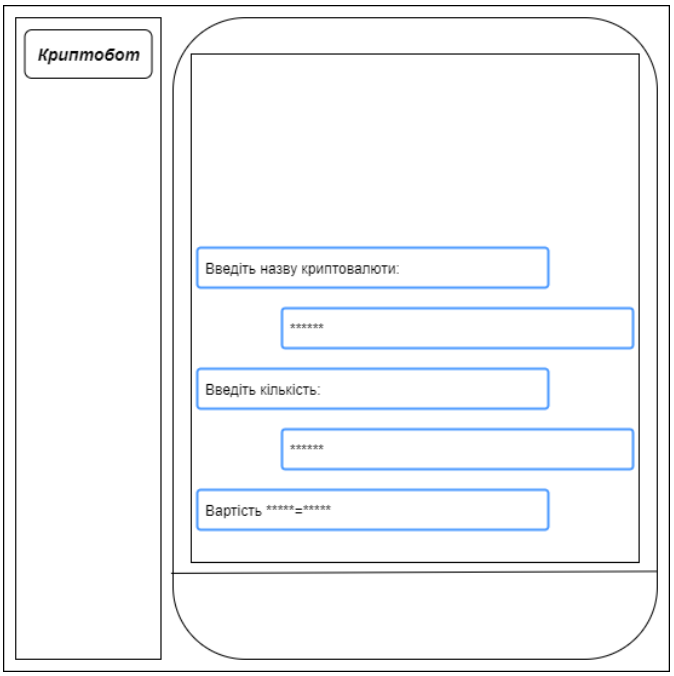


Рисунок 2.14 – Макет роботи кнопки «Конвертер»

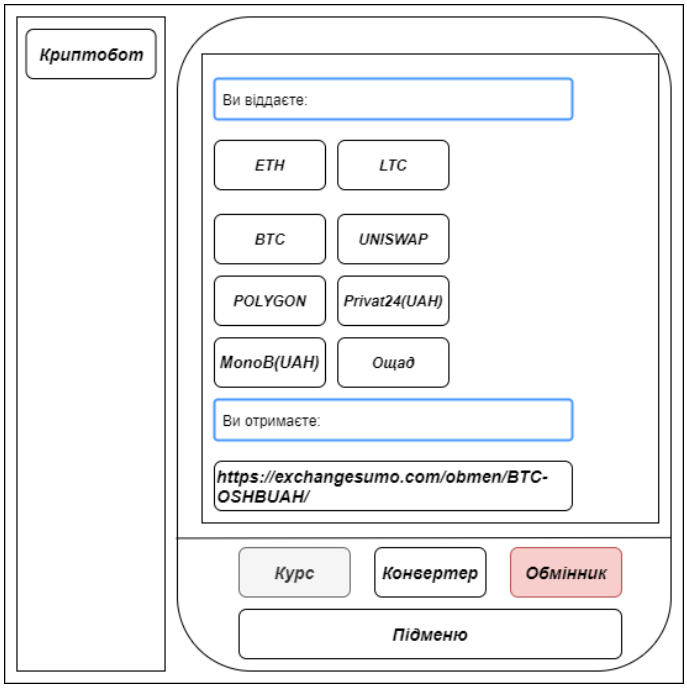


Рисунок 2.15 – Макет головного меню та робота кнопки «Обмінник»

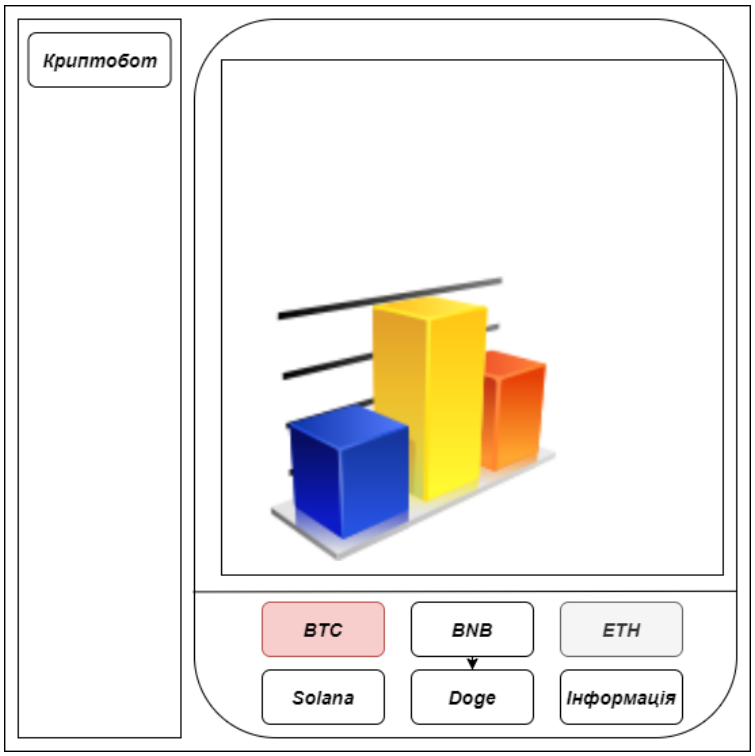


Рисунок 2.16 – Макет меню кнопки «Графік» та робота кнопки «BTC»

2.4 Висновки за розділом

Підводячи підсумки другого розділу, у ньому була описана архітектура системи, основні функції телеграм-боту, його модулі та спроєктовано макети інтерфейсу.

Для створення системи було обрано бібліотеку Aiogram для роботи з телеграм-ботом.

Незалежний агрегатор CoinGecko забезпечить систему ринковими даними про криптовалюту за допомогою API.

За обробку даних для побудови графіку відповідатиме бібліотеки Pandas та Numpy. Для візуалізації і збереження графіка обрано бібліотеку Altair та програмну платформу Node.js.

Вибір мови програмування був зроблений на користь Python, середовищем розробки обрано PyCharm.

Також було створено макети інтерфейсу та UML діаграми за допомогою графічного сервісу DrawIo.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Реєстрація боту у месенджері Telegram

Реєстрація боту у Telegram відбувається за стандартною схемою. Для отримання спеціального токену необхідно скористатися головним ботом Telegram – BotFather. Після запуску, BotFather пришле повідомлення з усіма наявними командами для створення власного боту (рис. 3.1).

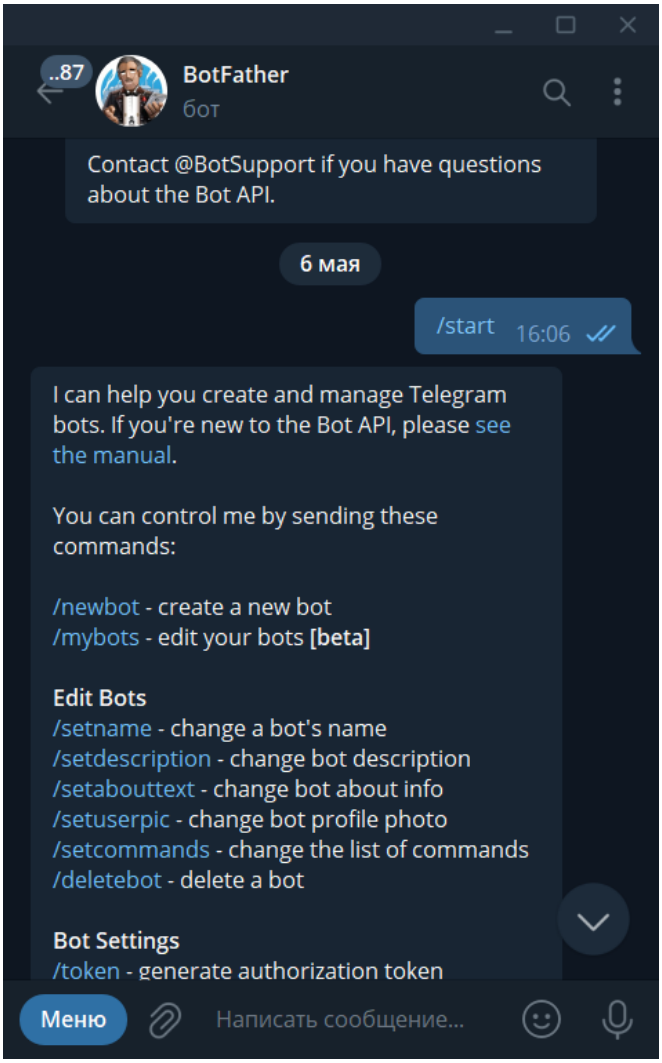


Рисунок 3.1 – Наявні команди BotFather

Скориставшись командою /newbot необхідно обрати ім’я, яке будуть бачити користувачі під час спілкування з ботом. Наступний крок це вибір

нікнейму боту по якому можна буде знайти робота в Telegram. Для вибору нікнейму існує правило він має бути унікальним, не повторювати існуючі в базі та закінчуватися словом «bot».

Після того, як ім’я та нікнейм буде обрано користувач отримаєте повідомлення з посиланням на бота, рекомендації по налаштуванню головного зображення, опису боту та список команд для його налаштування.

Щоб подивитися спеціальний токен до боту потрібно скористатися командою /token (рис. 3.2).

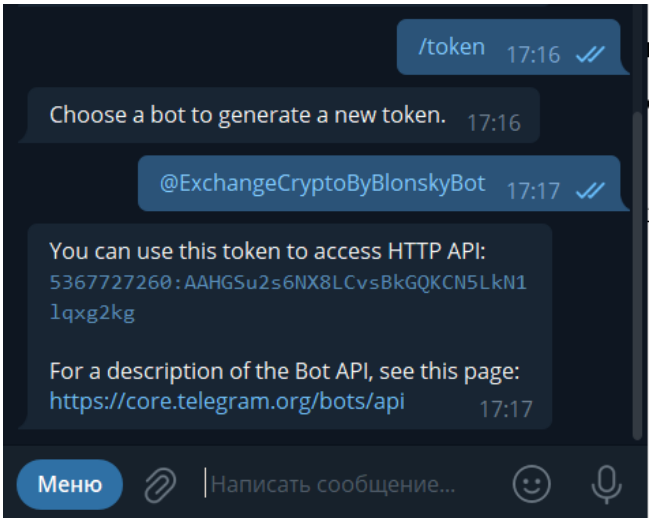


Рисунок 3.2 – Персональний токен боту

Для того, щоб змінити опис, який будуть бачити користувачі при першому запуску боту, потрібна команда /setdescription. Нове головне зображення можна завантажити за допомогою команди /setuserpic (рис. 3.3).

Рисунок 3.3 – Встановлено головне зображення

3.2 Розробка модулів системи

Реалізація програмної системи базується на принципах модульного програмування, яке використовується для забезпечення технологічності програмного забезпечення, що розробляється.

Основними перевагами даного програмування є:

– модульні програми легко складати та налагоджувати;

– модульну програму легше супроводжувати та модифікувати;

– полегшення процесу управління.

Враховуючи вищесказане було створено такі модулі:

– generate;

– config;

– main;

– mainfunc;

– markups;

– exchange;

– graph;

– conv;

– swap.

3.2.1 generate.py

Даний модуль відповідає за створення екземпляру пам’яті, боту та диспетчеру.

Змінна пам’яті є екземпляром класу MemoryStorage та відповідає за зберігання назви та кількості криптовалюти, яку вводить користувач.

Екземпляр класу Dispatcher необхідний для відкривання боту.

Також в цьому модулі розроблено функцію, яка реєструє команду для запуску боту /start.

Лістинг модуля міститься у додатку В.1.

3.2.2 main.py

Модуль відповідає за запуск боту та реєструє обробники з інших модулів.

Для запуску боту використано функцію start_polling, яку містить імпортований модуль executor.

Робота боту базується на технології long polling,

Long polling це технологія, яка використовується у веб-додатках для забезпечення відповіді на повідомлення сервера з малою затримкою без використання ЦП або трафіку при високочастотному опитуванні.

Клієнт робить запит до веб-сервера, але замість негайної відповіді сервер утримує з'єднання (протягом потенційно тривалого часу) та відповідає лише тоді, коли дані доступні.

Функція start_polling має деякі параметри, серед яких – skip_updates, значення якого встановлено як True. Завдяки цьому, коли бот вмикається після

паузи він не обробляю звернення користувачів, які надійшли під час неактивності.

Лістинг модуля міститься у додатку В.2.

3.2.3 mainfunc.py

Даний модуль створює змінну для з'єднання з CoinGecko API. Для зберігання станів необхідно створити клас, що успадковується від класу StatesGroup, всередині нього потрібно створити змінні, надавши їм екземпляри класу State. Реалізовано клас машини станів, яка використовується у модулях exchange та conv. Кінцевий автомат має такі стани:

– chose_coin_name;

– select_quantity;

– coin.

Клас кінцевого автомату наслідується від класу StatesGroup. Діаграму класів приведено на рисунку 3.4.

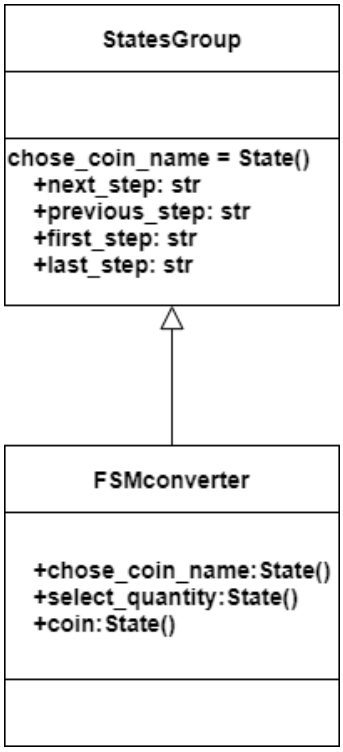


Рисунок 3.4 – Діаграма класів машини станів

Також у цьому модулі реалізовані обробники основних кнопок та команди /start. Ці обробники додаються до системи функцією реєстрації.

У модулю написано обробник, який видаляє будь-які повідомлення від користувача та виводить попереджуючий текст. Обробник реалізовано у кінці модулю для того, щоб він не реагував на натискання кнопок та запуску команди /start.

Лістинг модуля міститься у додатку В.3.

3.2.4 markups.py

Даний модуль створює кнопки та клавіатури для головного меню, підменю, меню кнопки «Графік», кнопки «Курс», кнопки «Обмінник».

Для реалізації клавіатури використано шаблон ReplyKeyboardMarkup. Така клавіатура відображається замість основної і не прив'язана до жодного повідомлення.

У кнопки типу KeyboardButton такої клавіатури не можна закласти жодної інформації.

Для створення кастомної клавіатури було використано шаблон InlineKeyboardMarkup, з його допомогою можна виконувати складніші дії. Клавіатура прив'язується до повідомлення, з яким було відправлено. У кнопки типу InlineKeyboardButton можна закласти будь-який текст розміром від 1 до 64 байт.

Обидві клавіатуру можна редагувати, але у різний спосіб. Перша оновлюється при надсиланні повідомлення з новою клавіатурою ReplyKeyboardMarkup, другу можна редагувати разом з повідомленням, до якого вона прикріплена.

Лістинг модуля міститься у додатку В.4.

3.2.5 exchange.py

У Модуль exchange створено функції пошуку певної монети та виведення топу цікавих криптовалют.

З модулю mainfucn імпортовано змінну API.

Функція виведення топу реалізована за допомогою двох запитів до агрегатора CoinGecko, get_price та get_coins_markets.

Створивши змінні цих запитів до них записується необхідні дані. Запит get_price відповідає за виведення поточної ціни обраної криптовалюти, запит get_coins_markets передає більш детальну інформацію про монету, а саме її ринкову капіталізацію, об’єм та інші ринкові дані.

Дані з CoinGecko приходять у форматі json, тобто у вигляді словників у яких дані зберігається у форматі ключ-значення. Ключів може буде декілька. Наприклад, щоб дізнатися ціну 1 біткоіна у доларах необхідно вказати відповідні ключі (рис. 3.5).

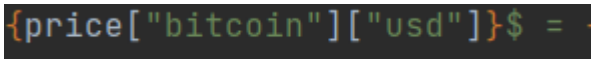


Рисунок 3.5 – Приклад отримання значення зі словника

Отримані результати відправляються користувачеві.

Обробник кнопки пошуку певної криптовалюти встановлює для машини станів стан coin. Це означає, що система очікує введення назви криптовалюти. Кожне наступне повідомлення потрапляє у функцію яка записує у пам’ять отриманий від користувача текст. Система надсилає запит до CoinGecko API, щоб отримати дані про монету з функцій get_price та get_coins_markets.

У випадку якщо користувач увів назву існуючої криптовалюти функція chose_nam надсилає повідомлення з отриманими даними, інакше спрацьовує обробник виключення KeyError і надсилається повідомлення про некоректну назву.

Система очікує наступного повідомлення з назвою монети. Це відбувається доти доки введений текст не буде відповідати існуючій криптовалюти.

Лістинг модуля міститься у додатку В.5.

3.2.6 graph.py

При розробці функції будування графіку було прийнято рішення про вибір періоду побудови 3 дні. Лівою межею є попередні два дні, правою межею слугує наступний день.

Далі за допомогою функцій tomorrow та prev отримуються дані проміжку часу за яким будується графік. Ці функції необхідні для отримання коректних значень попередніх та наступного днів. Обидві функції у якості параметрів отримують поточну дату. Далі викликається функція, яка підраховує кількість днів з початку року. До поточної дати додається/віднімається необхідна кількість днів, після чого алгоритм вираховує нову дату враховуючи високосність року та кількість днів у кожному місяці.

Для моніторингу даних криптовалюти використовується запит get_coin_market_chart_range_by_id, який передає історичні ринкові дані, включно ціну та об’єм на проміжку часу. Розбиття отриманих даних на інтервали відбуваються автоматично виходячи з вказаного періоду.

Побудова графіку відбувається за стандартною схемою, отримані дані заносяться до необхідних змінних які потім виступають у якості параметрів функцій побудови графіку.

Створений функцією alt.layer графік, зберігається в теці проєкту у форматі «PNG» за допомогою функції save. Після чого користувач отримує повідомлення зі створеним графіком.

Для створеної функції моніторингу даних криптовалюти та побудови графіку створено блок-схему на рисунку 3.6.

Лістинг модуля міститься у додатку В.6.

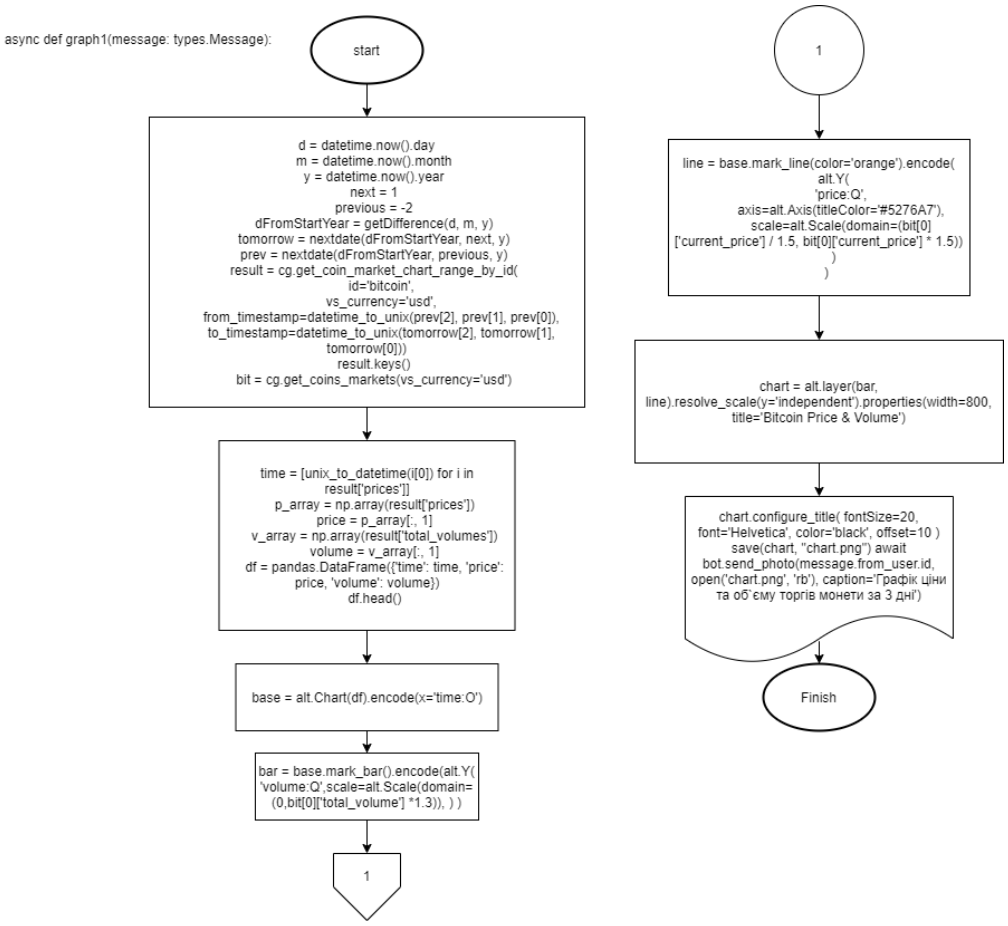


Рисунок 3.6 – Функція моніторингу даних криптовалюти та побудови графіку

3.2.7 conv.py

Модуль conv відповідає за функції калькуляції конвертування. До модуля імпортовано змінну cg, яка реалізую передачу даних з CoinGecko, а також клас машини станів FSMconverter.

Натиснувши кнопку «Конвертер» функція запускає перший стан машини станів chose_coin_name та виводить повідомлення користувачу про необхідність ввести назву криптовалюти.

При неправильному введені назви спрацює обробник виключення KeyError, який видає повідомлення про необхідність повторного введення. Цей процес буде повторюватися доки не буде введена коректна назва.

Після успішного введення назви монети встановлюється другий стан – select_quantity. Система очікує від користувача введення кількості монет. У разі коректного вводу система виконає конвертацію, результати якої надішле

в повідомленні. У випадку, якщо замість кількості введено текст чи від’ємне значення спрацює обробник виключення ValueError та проінформує користувача, що необхідно коректно ввести кількість валюти.

Лістинг модуля міститься у додатку В.7.

3.2.8 swap.py

Модуль swap зберігає обробники кастомної клавіатури InlineKeyboardMarkup, яка використовується при роботі кнопки «Обмінник».

При натисканні кнопки «Обмінник» у діалозі створюються кнопки типу Inline та виводиться повідомлення з пропозицією обрати валюту для обміну. Після того, як користувач вибере валюту для обміну клавіатура видалиться та створиться нова. Обравши валюту яку хоче отримати користувач, клавіатура видаляється та система надсилає повідомлення посилання на сервіс обмінників.

Лістинг модуля міститься у додатку В.8.

3.3 Висновки за розділом

У даному розділі було описано роботу створених модулів. Було описано клас та створено блок-схему моніторингу даних криптовалюти та побудови графіку.

Також наведено принципи роботи алгоритмів розрахунку часових значень для побудови графіків.

Описано реалізацію машини станів та виключень помилок.

4 ОПИС РОЗРОБЛЕНОГО ТЕЛЕГРАМ-БОТУ

4.1 Опис роботи системи

Під час першого запуску телеграм-боту автоматично спрацьовує команда /start, система вітається з користувачем та виводить віртуальну клавіатуру разом с базовим описом боту (рис. 4.1).

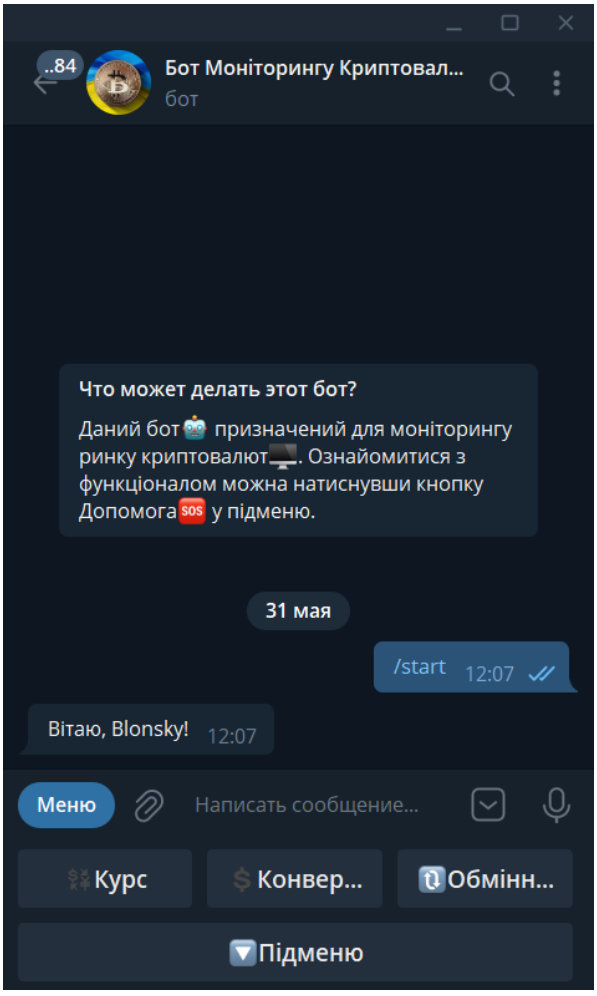


Рисунок 4.1 – Запуск команди старт та виведення головного меню

Телеграм-бот має інтуїтивно зрозумілий інтерфейс для користувачів, які активно цікавляться світом криптовалют, але якщо новий користувач тільки починає вивчати світ цифрових активів для нього інтерфейс і функціонал може бути не зрозумілим.

Тому для цього бот спеціально виводить опис системи при першому запуску та пропонує ознайомитися з функціоналом, натиснувши кнопку «Допомога», яка знаходиться у підменю.

На рисунку 4.2 зображено підменю телеграм-боту.

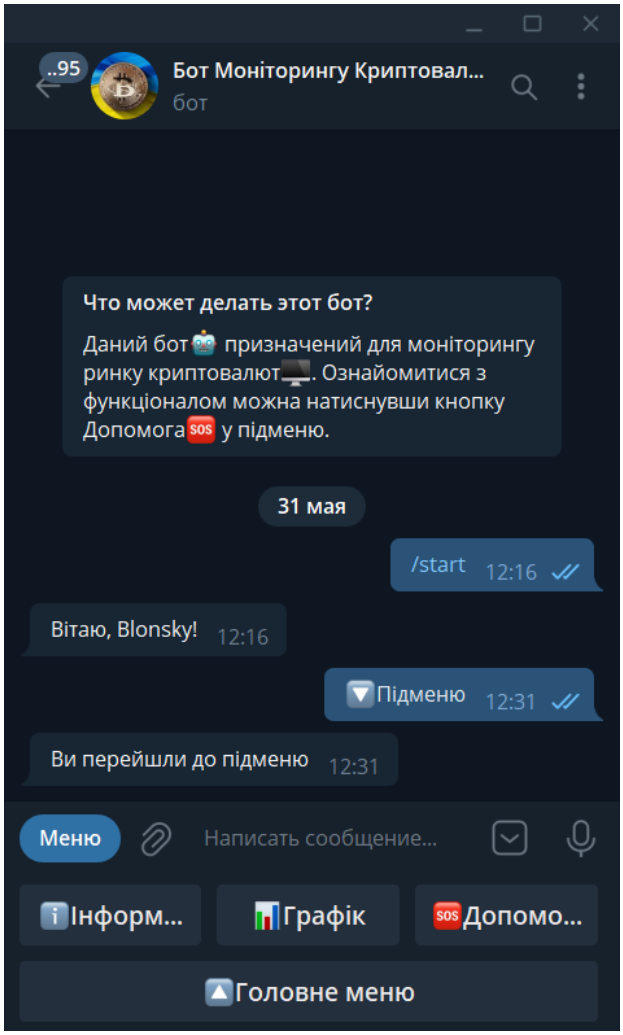


Рисунок 4.2 – Підменю телеграм-боту

Після того, як користувач натиснув кнопку «Допомога», бот надсилає користувачу повідомлення з описом усіх основних функцій системи.

На рисунку 4.3 показано роботу кнопки «Допомога».

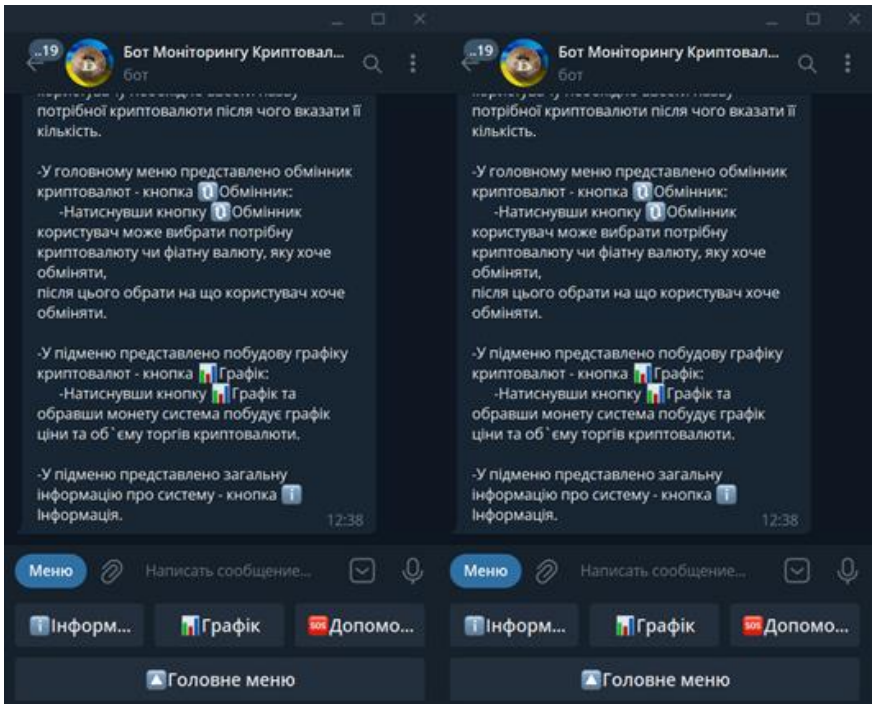


Рисунок 4.3 – Робота кнопки «Допомога»

Натиснувши кнопку «Інформація», яка знаходиться у підменю боту, система надішле повідомлення користувачу, яке містить загальну інформацію про розроблену систему та контакт автора (рис. 4.4).

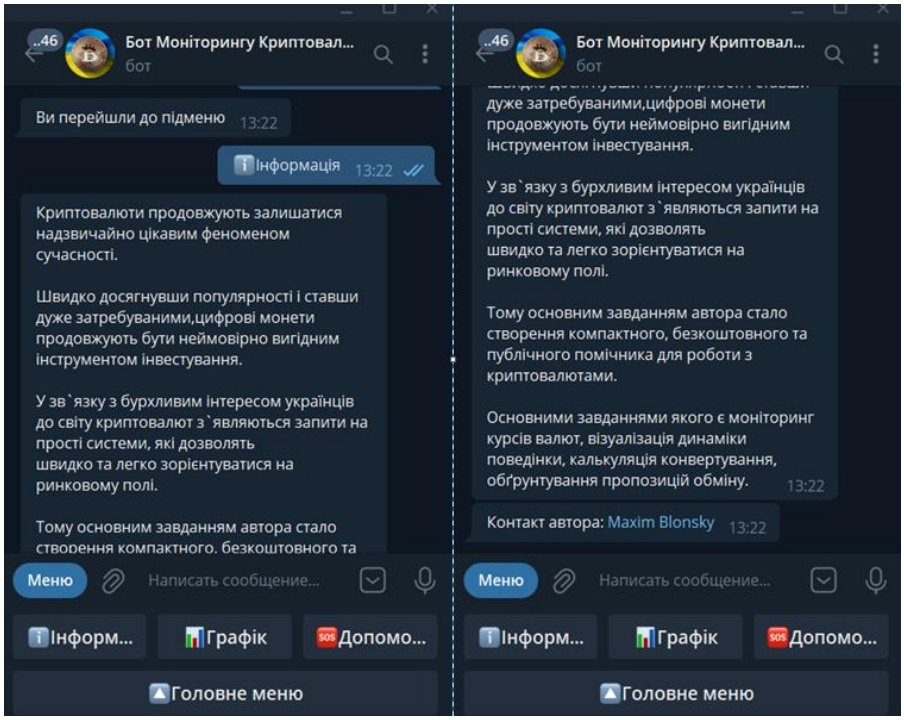


Рисунок 4.4 – Робота кнопки «Інформація»

Робота з ботом передбачена за допомогою віртуальної клавіатури тому, щоб не засмічувати діалог з ботом, система видаляє усі повідомлення користувача та інформує його про це (рис. 4.5).

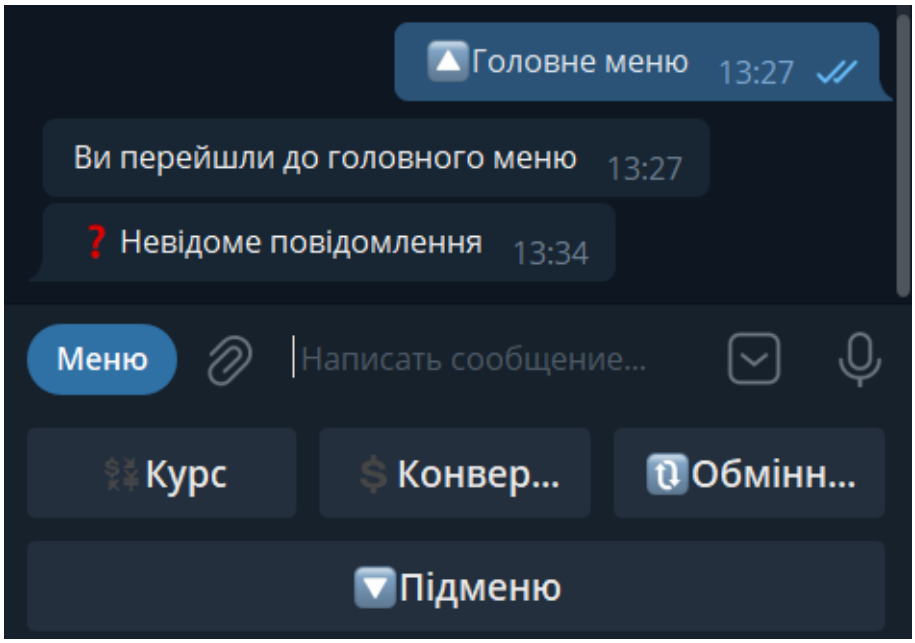


Рисунок 4.5 – Результат роботи обробника невідомих повідомлень

Перейшовши до головного меню та натиснувши кнопку «Курс», користувач переходить до меню цієї кнопки, яке виводить основні функції системи моніторингу криптовалют (рис. 4.6).

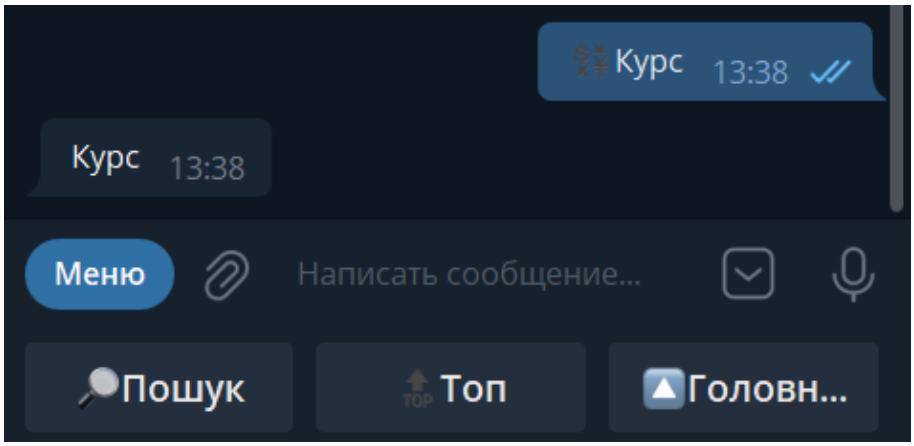


Рисунок 4.6 – Меню кнопки «Курс»

Після натискання на кнопку «Пошук», система запустить машину станів та видалить клавіатуру для більш зручного користування.

Бот сповістить користувача про необхідність введення назви потрібної йому криптовалюти та у разі коректного вводу користувач отримає інформацію про поточний курс монети, піковий і найнижчий курс за добу та добову зміну курсу, після чого система поверне віртуальну клавіатуру та виведе головне меню (рис. 4.7).

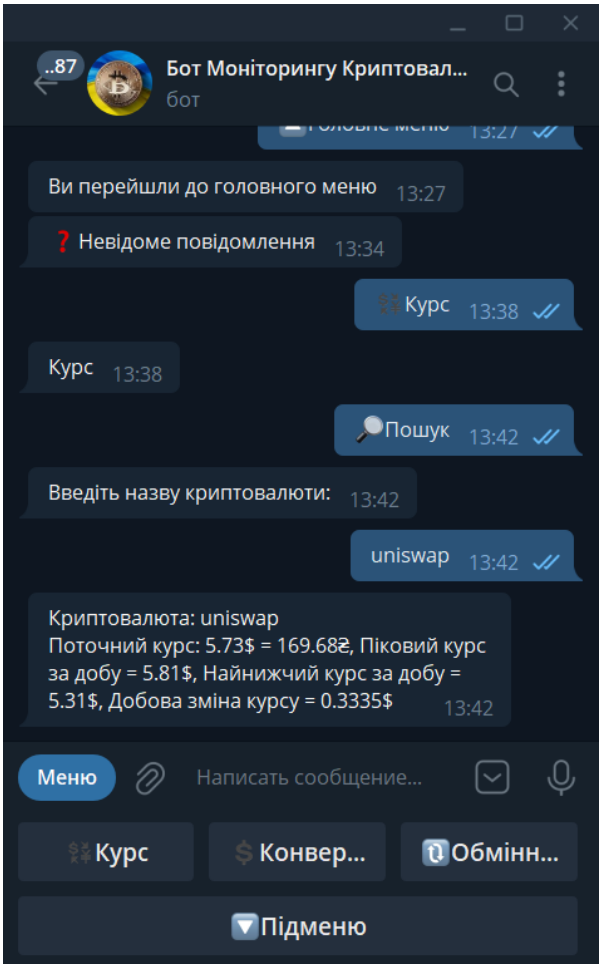


Рисунок 4.7 – Коректна робота кнопки «Пошук»

У разі некоректного введення назви криптовалюти система надішле повідомлення користувачу про необхідність повторити спробу та ввести назву існуючої монети(рис. 4.8).

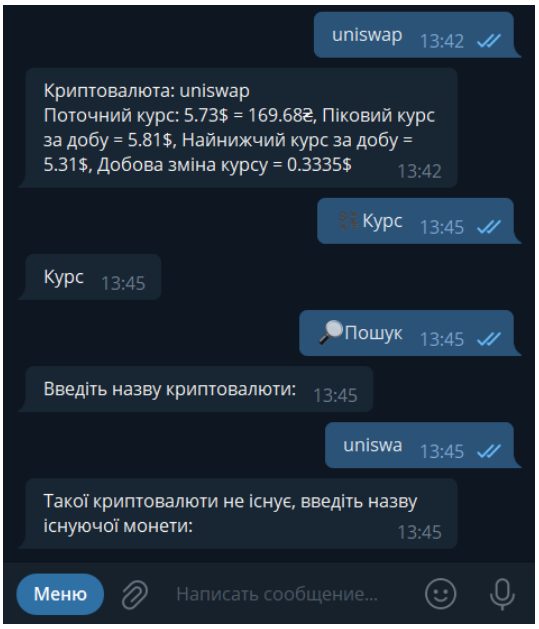


Рисунок 4.8 – Назва монети введено не вірно

Натиснувши кнопку «Топ», яка знаходиться у меню кнопки «Курс» користувач отримає дані про створений топ найбільш цікавих монет, який також виводить ринкові дані криптовалют (рис. 4.9).

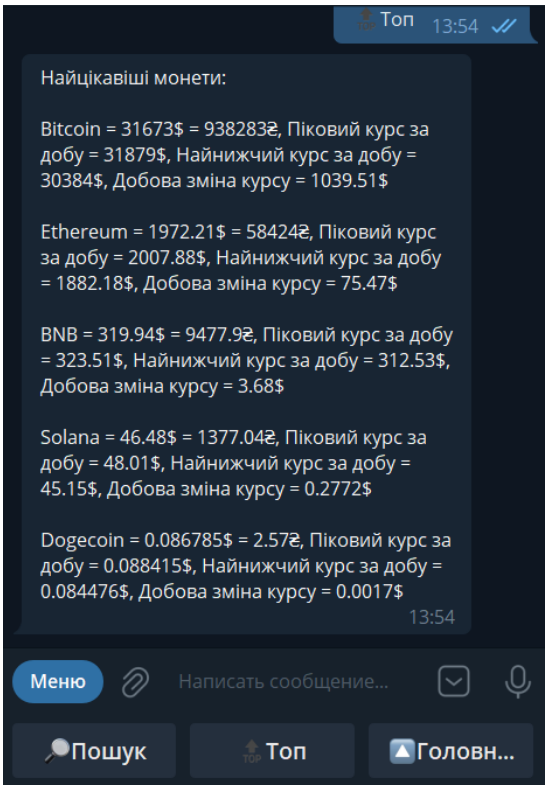


Рисунок 4.9 – Робота кнопки «Топ»

Кнопка «Конвертер», яка знаходиться на головному меню відповідає за виклик функції калькуляції конвертування. Після того, як користувач натиснув її, система видаляє віртуальну клавіатуру та запускає машину станів пропонуючи спочатку ввести назву необхідної монети, а потім її кількість. Провівши конвертацію система повертає віртуальну клавіатуру та виводить дані користувачеві (рис. 4.10).

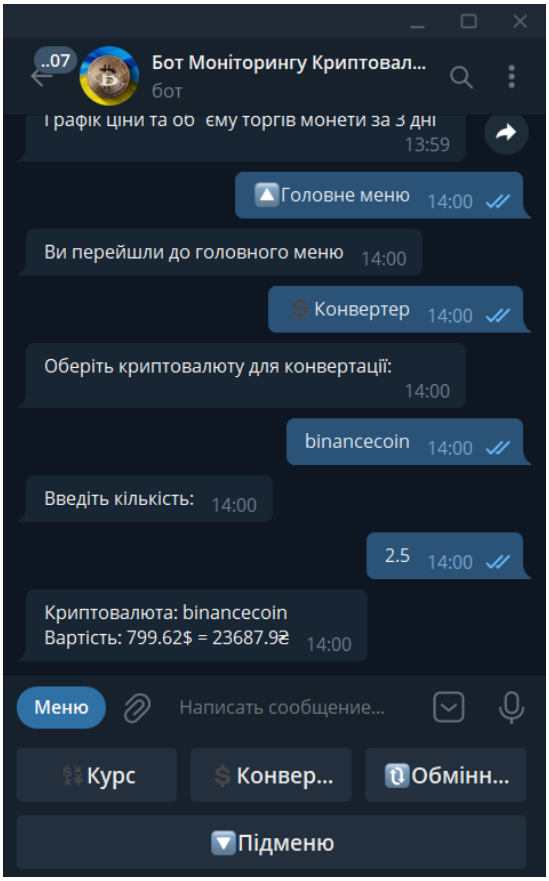


Рисунок 4.10 – Робота кнопки «Конвертер»

У разі неправильного введення назви монети чи її кількості система виведе повідомлення про необхідність коректного вводу назви та кількості необхідної криптовалюти (рис. 4.11).

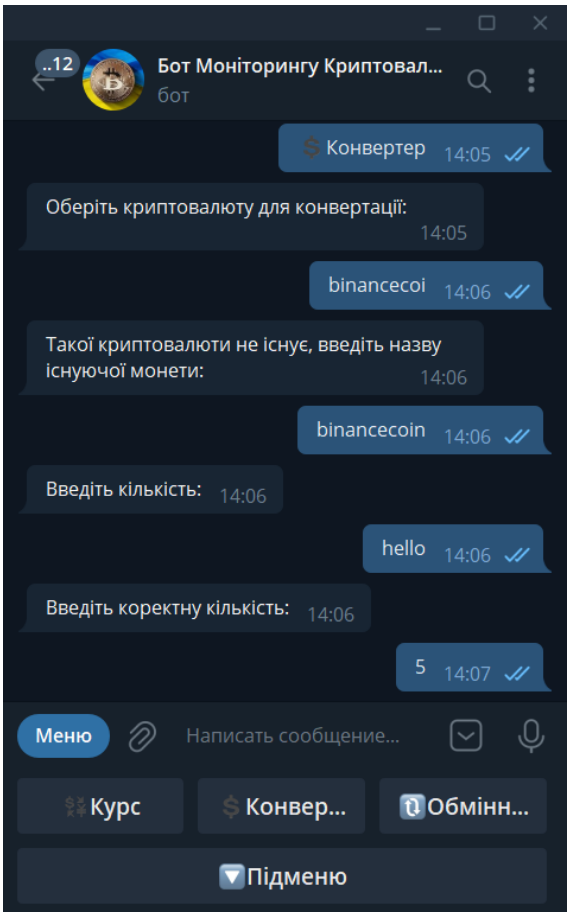


Рисунок 4.11 – Назва монети та її кількість введено не вірно

Пропозиції щодо вигідного обміну криптовалют можна знайти натиснувши кнопку «Обмінник» у головному меню, система створить спеціальні кнопки типу Inline у діалозі(рис. 4.12).

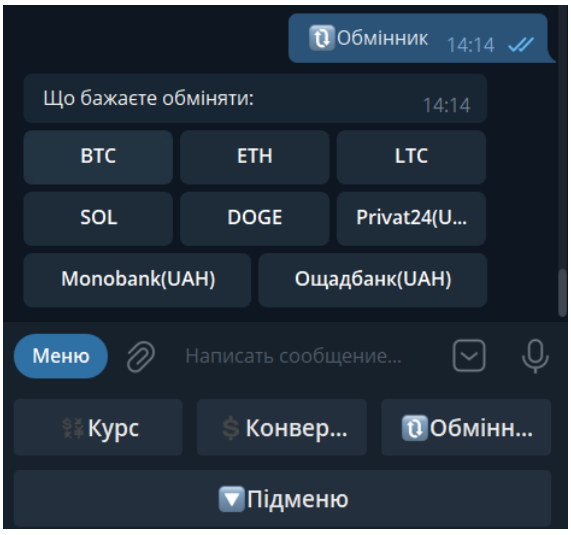


Рисунок 4.12 – Робота кнопки «Обмінник»

Обравши валюту, яку потрібно обміняти система видаляє клавіатуру в діалозі та створює нову, де користувач має обрати валюту, яку хоче отримати, після чого система надсилає повідомлення з посиланням на сервіс з найвигіднішими обмінниками (рис. 4.13 – 4.14).

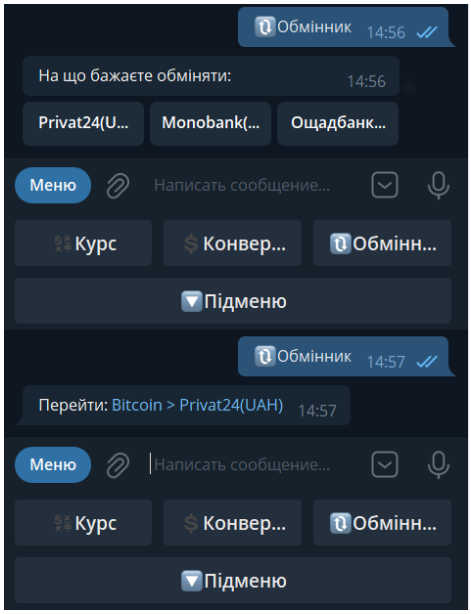


Рисунок 4.13 – Нова клавіатура у діалозі та посилання на сервіс

Обменник	Отдадите	Получите	Резерв	Отзывы
EExchanger	1 Ethereum (ETH) от 0.02875	69 570.53 Приват24 UAH	5 134 056	893
WmStar	1 Ethereum (ETH) от 0.25	69 383.69 Приват24 UAH	1 539 209	289
btckassa	1 Ethereum (ETH) от 0.15	69 242.81 Приват24 UAH	2 363 484	67
BTCSale	1 Ethereum (ETH) от 0.05	69 095.64 Приват24 UAH	3 153 891	1004
Artek-exchanger	1 Ethereum (ETH) от 0.015	68 876.05 Приват24 UAH	1 334 797	154
AvanChange	1 Ethereum (ETH) от 0.036	68 361.27 Приват24 UAH	2 528 927	2251
CoinCat	1 Ethereum (ETH) от 0.0182	68 189.64 Приват24 UAH	12 423 078	955
E-MoneyCC	1 Ethereum (ETH) от 0.2	68 162.92 Приват24 UAH	1 621 336	527
1654	1 Ethereum (ETH) от 0.05	67 920.12 Приват24 UAH	439 086	14
99francs	1 Ethereum (ETH) от 0.071	67 902.05 Приват24 UAH	1 434 598	63
cashout	1 Ethereum (ETH) от 0.015	67 613.46 Приват24 UAH	300 000	202
Cripta	1 Ethereum (ETH) от 0.003	67 364.22 Приват24 UAH	1 498	512
Cash2Pay	1 Ethereum (ETH) от 0.01	67 285.90 Приват24 UAH	1 016 171	44
Papa-Change	1 Ethereum (ETH) от 0.2	66 766.40 Приват24 UAH	22 615	444
Coins black	1 Ethereum (ETH) от 0.0765	66 382.63 Приват24 UAH	93 678	245

Рисунок 4.14 – Сервіс з найвигіднішими обмінниками

Для візуалізації динаміки поведінки криптовалюти користувачу потрібно натиснути кнопку «Графік» у підменю. Після цього користувач перейде до меню кнопки «Графік», де може обрати побудову графіку певної монети (рис. 4.15).

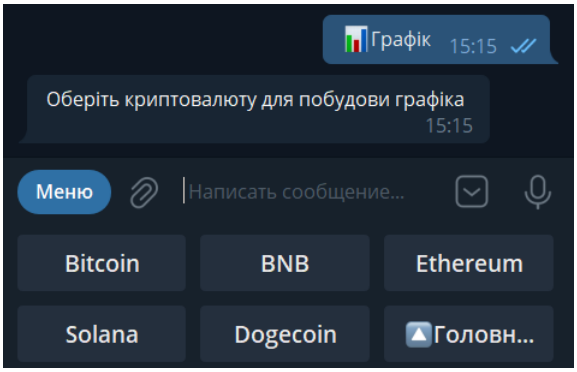


Рисунок 4.15 – Меню кнопки «Графік»

Користувач, обравши монету, викликає функцію, що будує графік на основі отриманих від агрегатора даних: ціни та об’єму торгів за певний проміжок часу. Після чого система надсилає повідомленням створений графік користувачу (4.16 – 4.17).

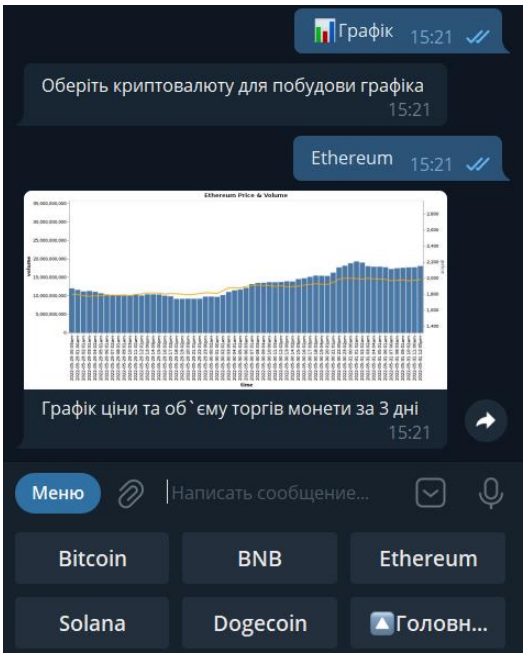


Рисунок 4.16 – Робота кнопки «Графік»

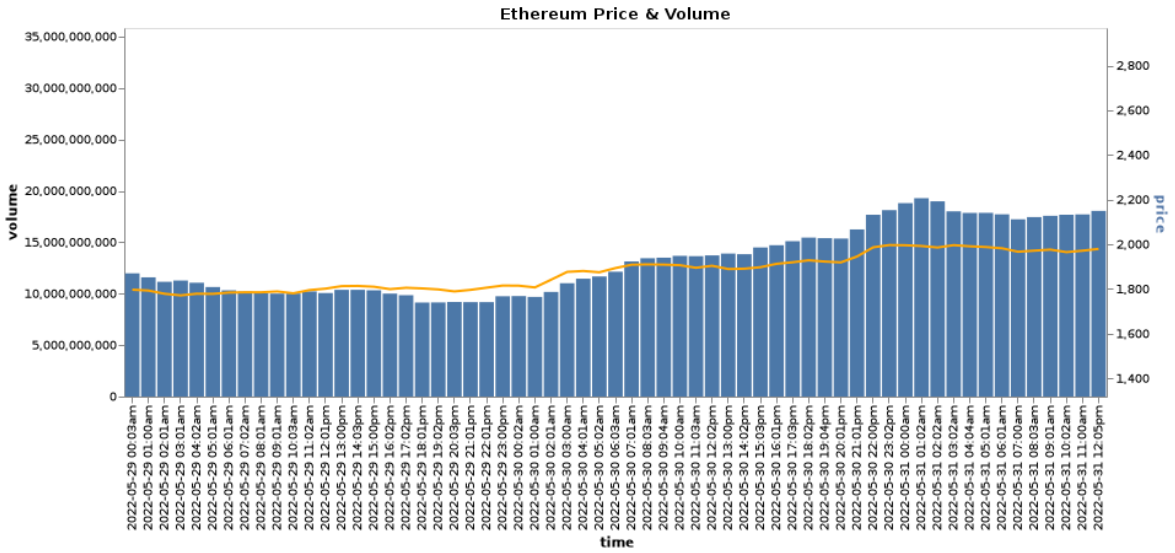


Рисунок 4.17 – Побудований графік монети ethereum

4.2 Висновки за розділом

У даному розділі було розглянуто роботу розробленої системи. Детально описано роботу усіх функцій.

Також було продемонстровано створений інтерфейс, його зовнішній вигляд та взаємодію з користувачем при виклику різних функцій.

Було показано роботу обробників помилок при запуску машини станів коли користувач вводить неправильні дані.

5 ТЕСТУВАННЯ СИСТЕМИ

5.1 API тестування

Тестування API – це практика тестування програмного забезпечення, яка безпосередньо перевіряє API – від їхньої функціональності, надійності, продуктивності до безпеки. Як частина інтеграційного тестування, тестування API ефективно перевіряє логіку архітектури складання за короткий проміжок часу.

Тести API проходять швидко, забезпечують високу рентабельність інвестицій та спрощують перевірку бізнес-логіки, безпеки, відповідності та інших аспектів програми. У випадках, коли API є загальнодоступним, надаючи кінцевим користувачам програмний доступ до нашої програми або служб, тести API фактично стають наскрізними тестами і повинні охоплювати всю історію користувача.

Тестування допомагає переконатися, що програма виконує поставлену перед нею мету та зможе коректно взаємодіяти з іншими програмами. Перевіряти та автоматизувати тести API можна навіть із мінімальною теоретичною базою. Головне – підібрати правильні інструменти.

Основними завданнями функціонального тестування API є:

– переконатися, що реалізація API працює коректно;

– гарантувати, що реалізація API працює відповідно до специфікації вимог;

– запобігти регресії між написаним кодом і релізом

Для тестування створення системи було використане функціональне тестування, яке включає тестування певних функцій кодової бази. Ці функції є представленням конкретних сценаріїв, щоб переконатися, що функції API обробляються відповідно до запланованих параметрів.

5.1.1 Тестування функцій CoinGecko API

Система використовує дані з сервісу CoinGecko за допомогою API, тому для коректної роботи телеграм-боту необхідно перевірити на наявність та статус підключення до серверу. Для цього було використано спеціально розроблені тести, які надає агрегатор CoinGecko (рис. 5.1).

```
@responses.activate
def test_connection_error(self):
    with pytest.raises(requests.exceptions.ConnectionError):
        exchange.cg.ping()

@responses.activate
def test_failed_ping(self):
    # Arrange
    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/ping',
                  status = 404)
    exception = HTTPError("HTTP Error")

    # Act Assert
    with pytest.raises(HTTPError) as HE:
        exchange.cg.ping()

@responses.activate
def test_ping(self):
    # Arrange
    ping_json = {'gecko_says':'(V3) To the Moon!'}
    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/ping',
                  json = ping_json, status = 200)
    # Act
    response = exchange().ping()
```

Рисунок 5.1 – Лістинг тестів функцій Coingecko API

На рисунку 5.2 зображено результати тестування функцій Coingecko API

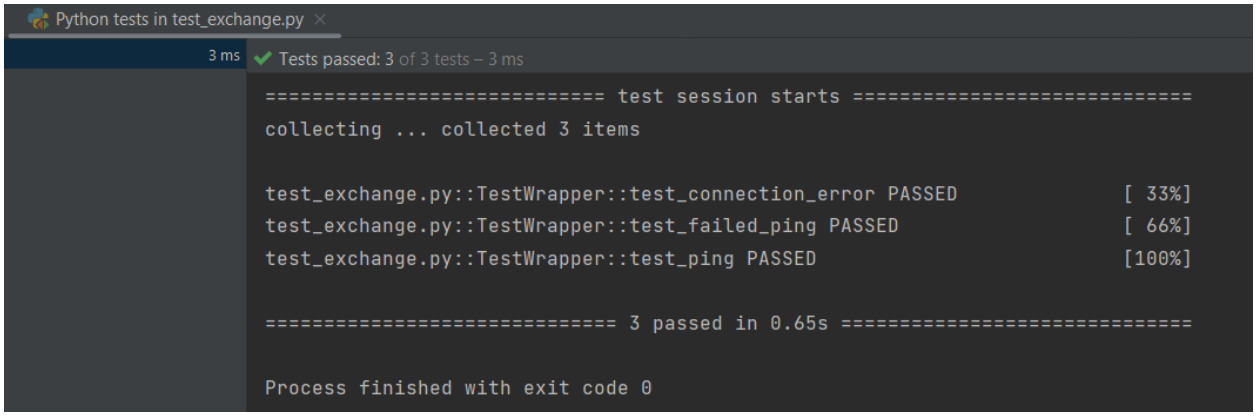


Рисунок 5.2 – Результати тестування функцій Coingecko API

5.1.2 Тестування запитів для моніторингу курсу криптовалют

При виклику функцій пошуку, топу, конвертації криптовалют система за допомогою спеціальних запитів отримує дані про ціну монети, найвищу та найнижчу ціну за день та добову зміну курсу з агрегатора. Для перевірки коректності отриманих даних про монети було використано тести від CoinGecko (рис. 5.3-5.4).

```
@responses.activate
def test_get_price(self):
    # Arrange
    coins_json_sample = {"bitcoin": {"usd": 7984.89}}

    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd',
                  json=coins_json_sample, status=200)

    # Act
    response = exchange.cg.get_price('bitcoin', 'usd')

    ## Assert
    assert response == coins_json_sample

@responses.activate
def test_failed_get_price(self):
    # Arrange
    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd',
                  status=404)
    exception = HTTPError("HTTP Error")

    # Act Assert
    with pytest.raises(HTTPError) as HE:
        exchange.cg.get_price('bitcoin', 'usd')
```

Рисунок 5.3– Лістинг тестів запиту ціни криптовалюти

```python
@responses.activate
def test_get_coins_markets(self):
    # Arrange
    markets_json_sample = [ {"id": "bitcoin", "symbol": "btc", "name": "Bitcoin", "image": "https:

    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd',
                  json = markets_json_sample, status = 200)

    # Act
    response = exchange.cg.get_coins_markets('usd')

    ## Assert
    assert response == markets_json_sample

@responses.activate
def test_failed_get_coins_markets(self):
    # Arrange
    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd',
                  status = 404)
    exception = HTTPError("HTTP Error")

    # Act Assert
    with pytest.raises(HTTPError) as HE:
        exchange.cg.get_coins_markets('usd')
```

Рисунок 5.4 – Лістинг тестів запиту ринків криптовалюти

На рисунку 5.5 зображено результати тестування функцій ціни та ринків криптовалюти.

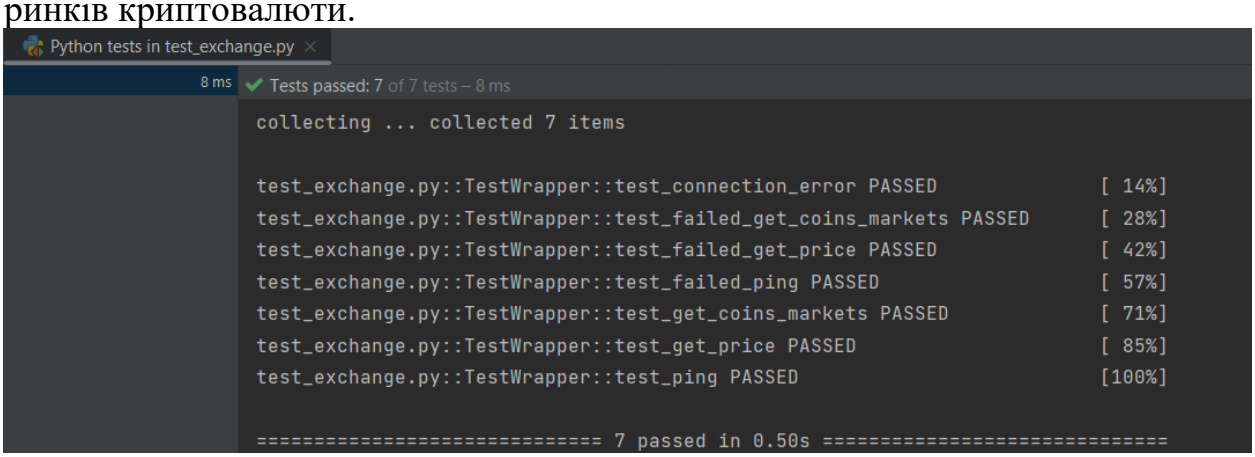


Рисунок 5.5 – Результати тестів запитів ціни та ринків криптовалюти

5.1.3 Тестування запиту для побудови графіку

При виклику функції побудови графіку використовується запит для отримання історичних ринкових даних, включаючи ціну, ринкову капіталізацію та 24-годинний об’єм у певному часовому діапазоні.

Для перевірки точності отриманих даних використано тести від CoinGecko (рис 5.6).

```python
@responses.activate
def test_failed_get_coin_market_chart_by_id(self):
    # Arrange
    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&days=1',
                  status=404)
    exception = HTTPError("HTTP Error")

    # Act Assert
    with pytest.raises(HTTPError) as HE:
        graph.cg.get_coin_market_chart_by_id('bitcoin', 'usd', 1)


@responses.activate
def test_get_coin_market_chart_by_id(self):
    # Arrange
    json_response = {"prices": [[1535373899623, 6756.942910425894], [1535374183927, 6696.894541693875],
                                [1535374496401, 6689.990513793263], [1535374779118, 6668.291007556478],
                                [1535375102688, 6703.7499879964], [1535375384209, 6706.898948451269]]}

    responses.add(responses.GET, 'https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&days=1',
                  json=json_response, status=200)

    # Act
    response = graph.cg.get_coin_market_chart_by_id('bitcoin', 'usd', 1)

    ## Assert
    assert response == json_response
```

Рисунок 5.6 – Лістинг тестів запиту для побудови графіку

На рисунку 5.7 зображено результат роботи тестів запиту для побудови графіку.

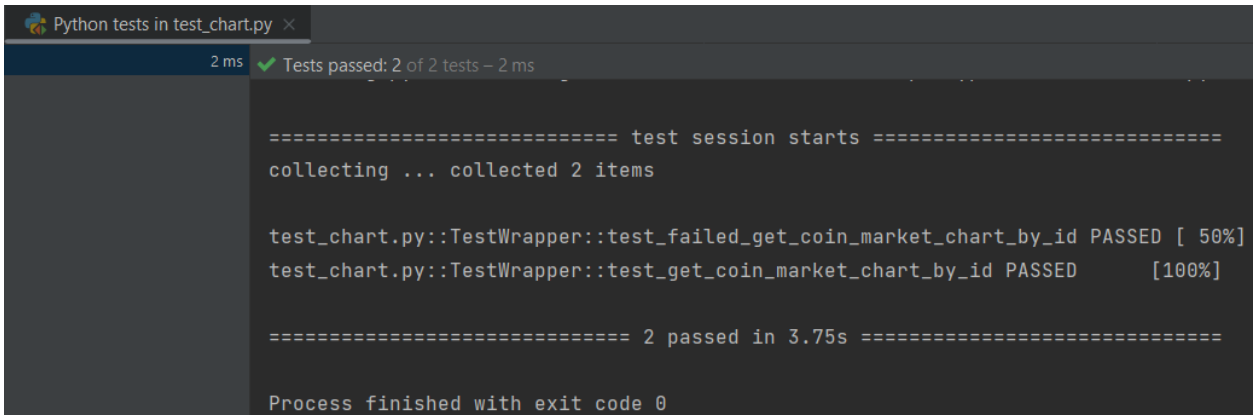


Рисунок 5.7 – Результат роботи тестів запиту для побудови графіку

5.1.4 Тестування обробника, який зчитує кількість валюти

Натискання кнопки Конвертер запускає роботу функції, яка повинна зчитати назву криптовалюти. Після успішного введення назви викликається функція, яка зчитує кількість валюти. Для успішної конвертації, вхідні дані для запиту мають бути коректними. Тобто користувач має ввести позитивне числове значення кількості валюти. Для перевірки роботи конвертеру було написано тест, вхідними даними якого є від'ємне число та нуль.

Лістинг модуля міститься у додатку В.10.

Результати тесту провальні, але механізм обробки виявленої помилки було розроблено до етапу тестування(рис. 5.8).

```
# @dp.message_handler(state= FSMconverter.select_quantity)
async def chose_amount(message: types.Message, state: FSMContext):
    try:
        async with state.proxy() as data:
            data['amount'] = float(message.text)
            result = cg.get_price(ids=data['name'], vs_currencies='usd, uah')
            p = round(data['amount'] * result[data['name']]['usd'], 2)
            p2 = round(data['amount'] * result[data['name']]['uah'], 2)
            await bot.send_message(message.from_user.id,
                                   f"Криптовалюта: {data['name']}\nВартість: {p}$ = {p2}₴", reply_markup=nav.mainMenu)
            await state.finish()
    except ValueError as VE:
        print(VE)
        await bot.send_message(message.from_user.id, 'Введіть коректну кількість:')
        await FSMconverter.select_quantity.set()
```

Рисунок 5.8– Лістинг механізму обробки

На рисунку 5.9 зображено результат тесту обробника.

```
Python tests in tesn_func.py
0 ms   Tests failed: 1 of 1 test – 0 ms
0 ms   tesn_func.py::Test::test_check FAILED                              [100%]
0 ms   tesn_func.py:7 (Test.test_check)
eck 0 ms   self = <tesn_func.Test testMethod=test_check>

           def test_check(self):
>              self.assertRaises(ValueError, chose_amount, -5, state= FSMconverter.select_quantity)
E              AssertionError: ValueError not raised by chose_amount

tesn_func.py:9: AssertionError
```

Рисунок 5.9– Результат тесту обробника.

5.1.5 Висновки за розділом

Підводячи підсумки цього розділу у ньому було визначено метод тестування, який включає тестування певних функцій кодової бази. Це дозволить розробнику переконатися, що функції API обробляються відповідно до запланованих параметрів.

Були перевірені наявність та статус підключення до серверу CoingGecko. Також було перевірено коректність роботи запитів, які відповідають за дані про ціну та інші ринкові дані криптовалют.

Для перевірки коректності роботи запитів було використано спеціально створені тести від агрегатора CoinGecko.

Також було написано тест для обробника, який зчитує кількість валюти. Результати тестування виявилися незадовільними через те, що механізм обробки виявленої помилки було розроблено до етапу тестування.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проаналізовано тему роботи, її актуальність, визначено мету дослідження та практичне значення роботи, описано методи, об’єкт та предмет дослідження.

Розглянуто різноманітні версії систем для роботи з цифровими активами та проведено аналіз конкурентних продуктів.

Було проаналізовано принципи та положення методів дослідження на яких базується робота. Також розглянуто основні технології створення системи, а саме:

– telegram-бот;

– об'єктно-орієнтоване програмування;

– інтерфейс прикладного програмування(api);

– машина станів;

– проєктування програмного забезпечення;

– емпіричні методи програмного забезпечення.

Також визначено основні завдання роботи:

– провести об’єктний аналіз предметної області, дослідити сучасні технології створення телеграм-ботів;

– розглянути існуючий алгоритм розробки систем моніторингу даних та дослідити особливості систем моніторингу, які застосовуються на криптовалютному ринку, провести порівняльний аналіз розглянутих систем;

– спроєктувати програмну систему моніторингу курсів криптовалют для месенджера Telegram;

– обґрунтувати програмні та апаратні вимоги до розроблюваної системи;

– отримати програмну реалізацію розроблювального додатку на мові Python з бібліотекою Aiogram та з незалежним агрегатором CoinGecko, який відстежує дані про криптовалюту;

– отримати програмну реалізацію модулів інтерфейсу телеграм-боту;

– провести тестування програмної системи для роботи з криптовалютами.

При описі архітектури системи було обрано мову програмування, середовище програмування та необхідні інструменти. Також було спроєктовано макети інтерфейсу майбутньої системи та побудовано такі UML-діаграми:

– діаграма варіантів використання для актора «користувач»;

– діаграма послідовності для актора «користувач»;

– діаграма кооперації для актора «користувач»;

– діаграма станів для функції конвертування;

– діаграма послідовності функції конвертування;

– діаграма розміщення.

При розробці програмної системи було детально описано роботу створених модулів, алгоритмів для побудови графіків та реалізацію машини станів і обробників помилок.

Для тестування системи було використано тести від CoinGecko для перевірки коректності роботи API та створено власний тест для обробника, який зчитує кількість валюти. Результати виявилися не задовільними через вже створений механізм обробки помилок.

Під час виконання кваліфікаційної роботи усі поставлені завдання були виконані в повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cryptoslate [електронний ресурс] – Режим доступу: https://cryptoslate.com/ukraine-has-the-highest-crypto-adoption-in-europe-and-fourth-globally/.

2. Triple-A [електронний ресурс] – Режим доступу: https://triple-a.io/crypto-ownership-ukraine/.

3. Chainalysis [електронний ресурс] – Режим доступу: https://blog.chainalysis.com/reports/2021-global-crypto-adoption-index/.

4. Statista [електронний ресурс] – Режим доступу: https://www.statista.com/statistics/234038/telegram-messenger-mau-users/.

5. Felderer M. Contemporary Empirical Methods in Software Engineering / M. Felderer, G. H. Travassos. – Springer, 2020. – 535 с

6. Shull F. Guide to Advanced Empirical Software Engineering / F. Shull, J. Singer, D. Sjøberg. – Springer, 2008. – 388 с

7. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – Worda, 2015. – 368 с

8. Бертран М. Объектно-ориентированное конструирование программных систем / М. Бертран. – Пер. с англ. – В. Биллиг, М. Дехтерь, Н. Супонев, Е. Павлова. – Русская Редакция, 2005. – 1204 с

9. Riel A. J. Object-Oriented Design Heuristics / A. J. Riel. – Addison-Wesley Professional, 1996. – 400 с

10. Мартін Р. Чиста архітектура: мистецтво розробки програмного забезпечення / Р Мартін. – Фабула, 2019. – 416 с

11. Buschmann F. Pattern-Oriented Software Architecture Volume 1: A System of Patterns / F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal.– Wiley, 1996. – 476 с

12. CtyptoChanBot [електронний ресурс] – Режим доступу: https://t.me/CryptoChanBot.

13. Лоре А. Проєктування веб-API / А. Лоре. – Пер. с англ. – Д. Беликова. – ДМК Прес, 2020. – 440 с.

14. Меджуи М. Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте / М. Меджуи, Э. Уайлд, Р. Митра, М. Амундсен – Пер. с англ. – А. Григорьева. – Питер, 2019. – 272 с

15. Tlgrm [електронний ресурс] – Режим доступу: https://tlgrm.ru/docs/bots.

16. Малышев К. Построение пользовательских интерфейсов / К. Малышев. – ДМК Пресс, 2021. – 268 с

17. Маттес Е. Пришвидшений курс Python / Е. Маттес. – ВСЛ, 2021. – 600 с

18. Васильєв О. Програмування мовою Python / О. Васильєв. – Навчальна книга Богдан, 2019. – 504 с

19. CoinGecko [електронний ресурс] – Режим доступу: https://www.coingecko.com/ru/.

20. Token API list [електронний ресурс] – Режим доступу: https://docs.google.com/spreadsheets/d/1wTTuxXt8n9q7C4NDXqQpI3wpKu1_5bGVmP9Xz0XGSyU/edit#gid=0.

21. Бібліотека Numpy [електронний ресурс] – Режим доступу: https://numpy.org/.

22. Бібліотека Pandas [електронний ресурс] – Режим доступу: https://pandas.pydata.org/.

23. Бібліотека Altair [електронний ресурс] – Режим доступу: https://altair-viz.github.io/.

24. Node.js [електронний ресурс] – Режим доступу: https://nodejs.org/uk/.

25. PyCharm [електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/pycharm/.

26. Draw.io [електронний ресурс] – Режим доступу: https://drawio-app.com/.

ДОДАТОК А.
ЗАУВАЖЕННЯ НОРМОКОНТРОЛЕРА

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
ПЗ		Нещільне заповнення сторінок пояснювальної записки
Зміст		Всі назви розділів, структурних часток, … повинні бути як у реченні, додаток В
С. 69-71		Оформлення використаних джерел містить помилки
Додаток В		Невірно оформлений (як розділ та підрозділи)

Додаток Б.
Графічна частина

Дипломний проєкт «Розробка телеграм-боту «Система моніторингу курсів криптовалют»

Виконав: ст. гр. ІПЗ-18 Оболонський м.в.

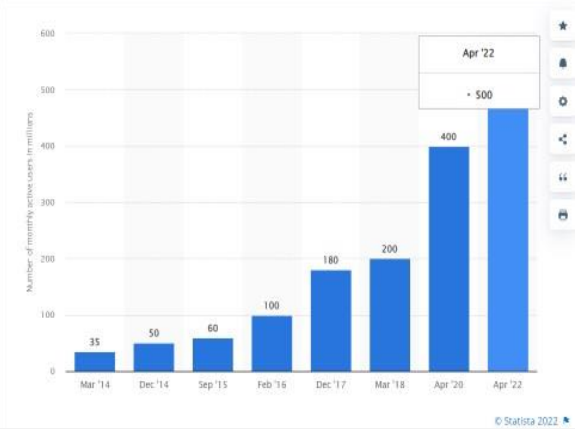
Керівник: зав. каф. ПМІ, д.т.н., проф. Дмитрієва О. А.

Рисунок Б1 – Титульний слайд

Актуальність обраної теми

Country	Index score	Overall index ranking	Ranking for individual weighted metrics feeding into Global Crypto Adoption Index		
			On-chain value received	On-chain retail value received	P2P exchange trade volume
Vietnam	1.00	1	4	2	3
India	0.37	2	2	3	72
Pakistan	0.36	3	11	12	8
Ukraine	0.29	4	6	5	40
Kenya	0.28	5	41	28	1
Nigeria	0.26	6	15	10	18
Venezuela	0.25	7	29	22	6
United States	0.22	8	3	4	109
Togo	0.19	9	47	42	2
Argentina	0.19	10	14	17	33
Colombia	0.19	11	27	23	12
Thailand	0.17	12	7	11	76
China	0.16	13	1	1	155
Brazil	0.16	14	5	7	113
Philippines	0.16	15	10	9	80
South Africa	0.14	16	18	16	62
Ghana	0.14	17	32	37	10
Russian Federation	0.14	18	8	6	122
Tanzania	0.13	19	60	45	4
Afghanistan	0.13	20	53	38	7

Звіт блок-чейн компанії Chainalysis



Графік користувачів Telegram

Рисунок Б2 – Актуальність обраної теми

Мета роботи та завдання дослідження

Метою роботи є розробка програмної системи моніторингу курсів криптовалют для месенджера Telegram.

Завданням дослідження є створення компактного помічника для роботи з криптовалютами, функціями якого є моніторинг курсів валют, візуалізація динаміки поведінки, калькуляція конвертування, обґрунтування пропозицій щодо обміну.

Рисунок Б3 – Мета роботи та завдання дослідження

Об’єкт та предмет дослідження

Об’єктом дослідження є процеси створення системи моніторингу даних, розробки та тестування боту у месенджері Telegram.

Предметом дослідження є методи та алгоритми функціонування системи моніторингу даних з незалежним агрегатором.

Рисунок Б4 – Об’єкт та предмет дослідження

Методи дослідження, технології та інструменти для розробки системи

- теорії алгоритмів ;
- емпіричні методи програмної інженерії ;
- об’єктно-орієнтованого програмування ;
- проєктування програмного забезпечення ;
- telegram-bot;
- інтерфейс прикладного програмування ;
- машина станів.
- агрегатор CoinGecko ;
- бібліотеки Aiogram, Altair, Numpy, Pandas;
- програмне середовище Node.js.

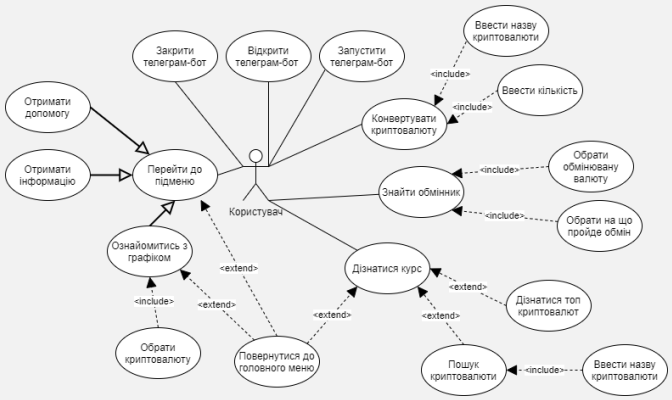
Рисунок Б5 – Методи дослідження, технології та інструменти для розробки системи

Основні завдання роботи

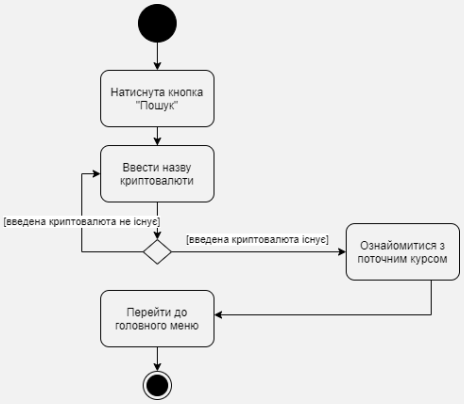
- провести об’єктний аналіз предметної області, дослідити сучасні технології створення телеграм-ботів;
- розглянути існуючи алгоритм розробки систем моніторингу даних та дослідити особливості систем моніторингу, які застосовуються на криптовалютномуринку, провести порівняльний аналіз розглянутих систем;
- спроєктувати програмну систему моніторингу курсів криптовалют для месенджера Telegram ;
- обґрунтувати програмні та апаратні вимоги до розроблюваної системи;
- отримати програмну реалізацію розроблювального додатку на мові Python з бібліотекою Aiogram та з незалежним агрегатором CoinGecko , який відстежує дані про криптовалюту;
- отримати програмну реалізацію модулів інтерфейсу телеграм -боту;
- провести тестування програмної системи для роботи зкриптовалютами.

Рисунок Б6 – Основні завдання роботи

UML -діаграми



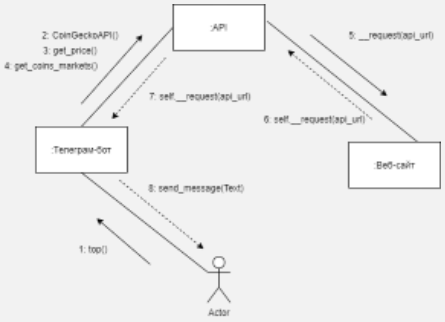
Діаграма варіантів використання для актора «Користувач»



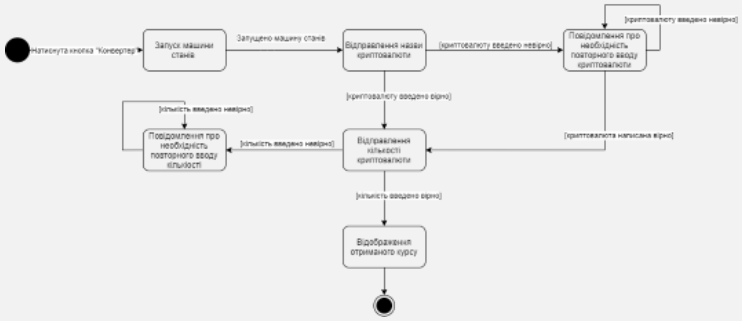
Діаграма послідовності для актора «Користувач»

Рисунок Б7 – UML-діаграми

UML -діаграми



Діаграма кооперації для актора «Користувач»



Діаграма станів для функції конвертування

Рисунок Б8 – UML-діаграми

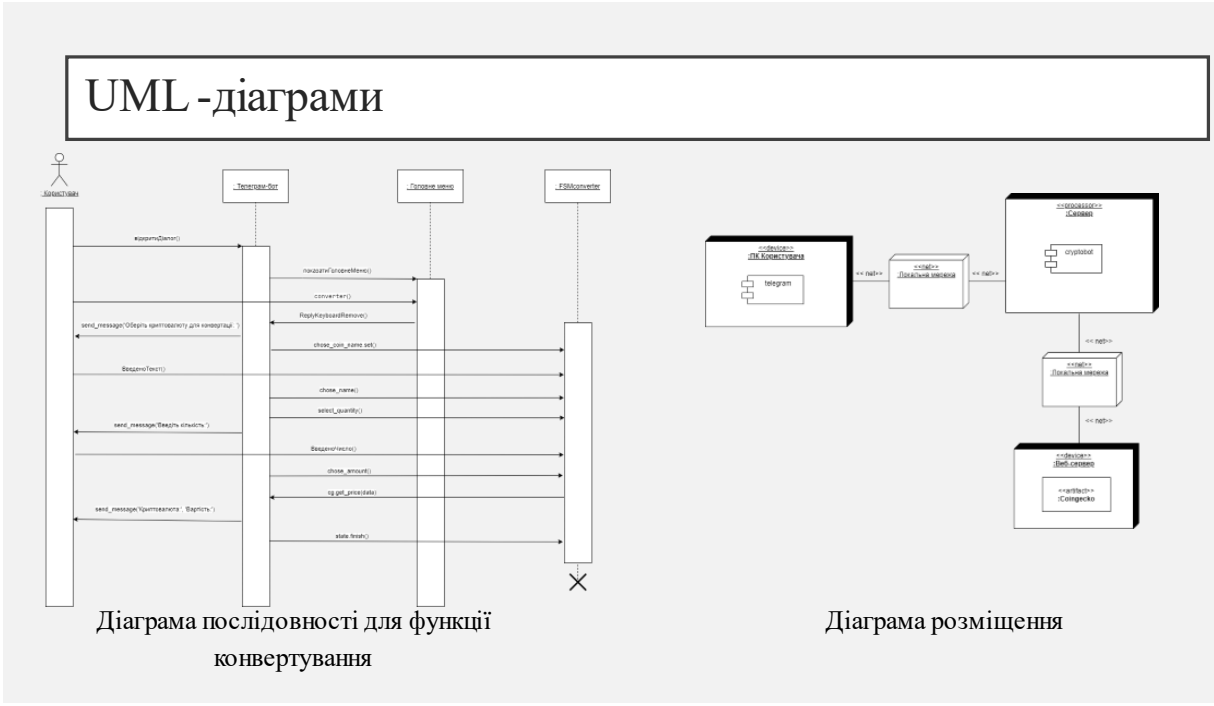


Рисунок Б9 – UML-діаграми

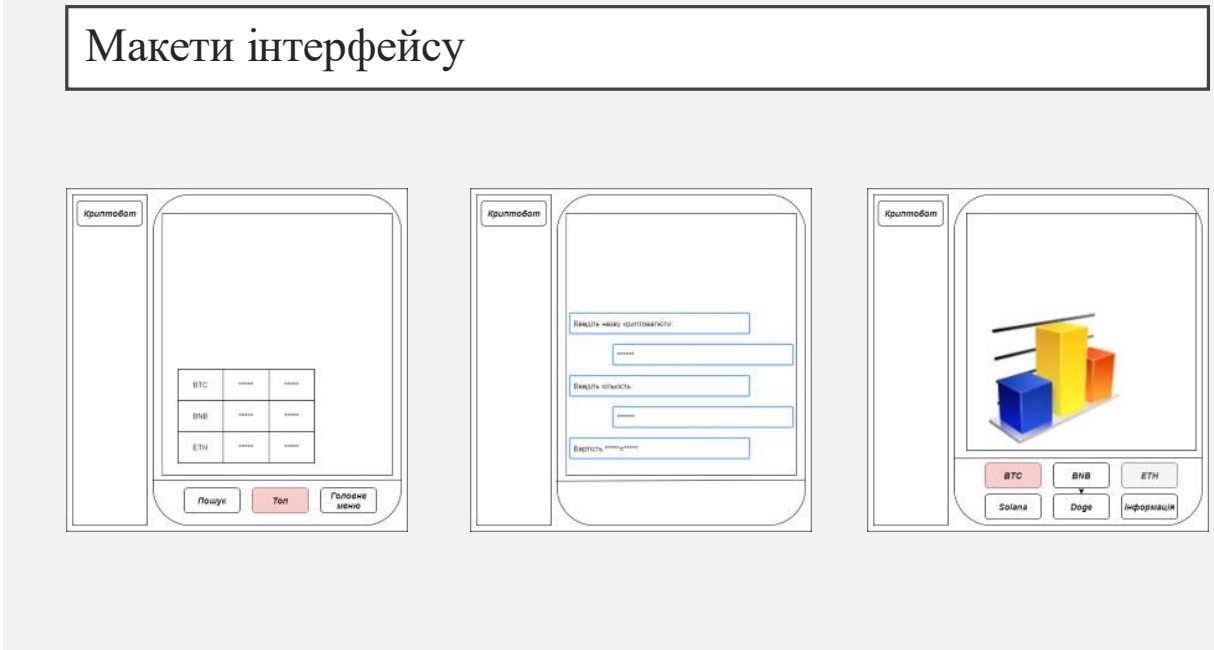
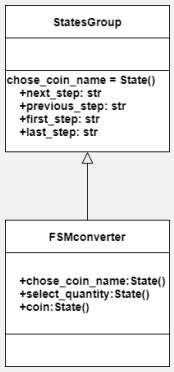


Рисунок Б10 – Макети інтерфейсу

Машина станів

Реалізація функцій конвертування та пошуку монети передбачає використання машини зі скінченною кількістю станів, через необхідність запам'ятовувати введену користувачем назву та кількість криптовалюти.



Діаграма класів машини станів

Рисунок Б11 – Машина станів

Блок-схема функції моніторингу даних криптовалюти та побудови графіку

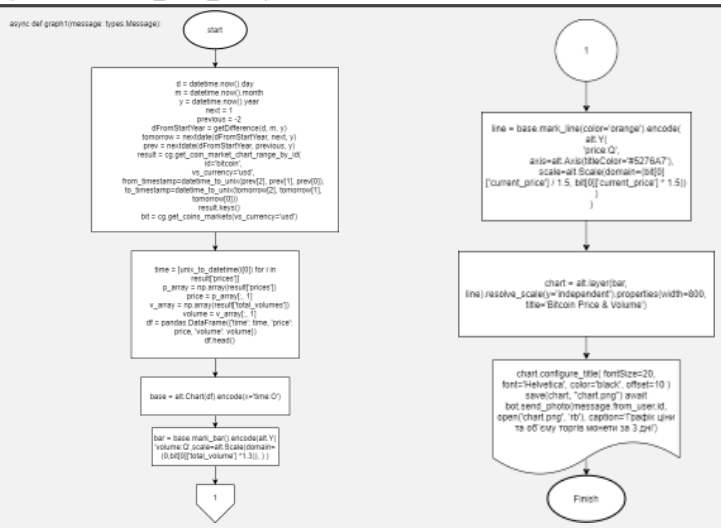


Рисунок Б12 – Блок-схема функції моніторингу даних криптовалюти та побудови графіку

Результати роботи

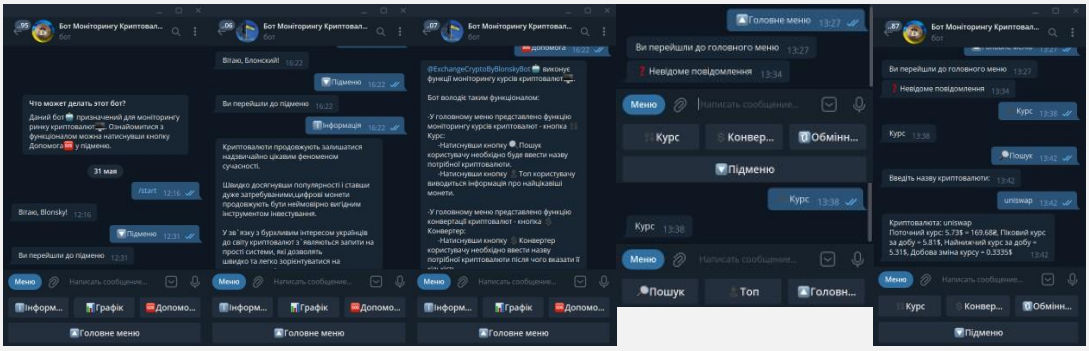


Рисунок Б13 – Результати роботи

Результати роботи

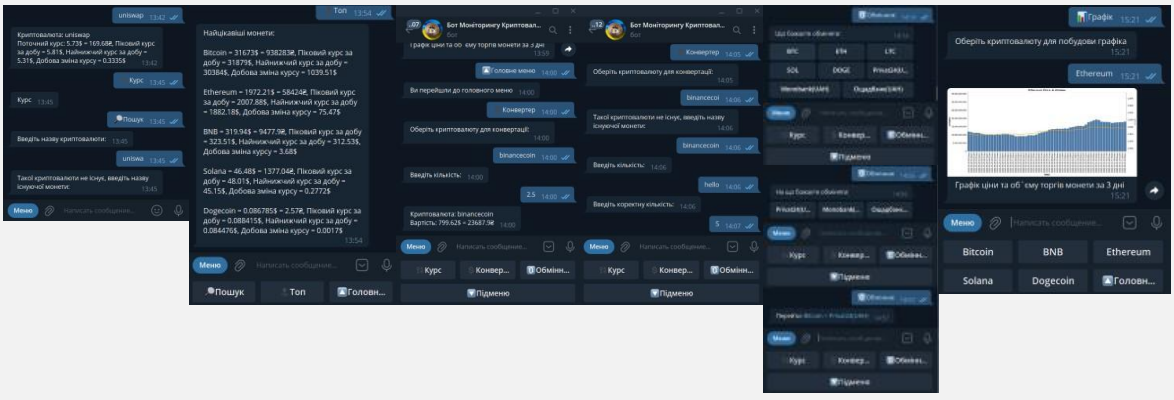


Рисунок Б14 – Результати роботи

Шляхи вдосконалення програмної системи

Основними шляхами для вдосконалення програмної системи є розширення її функціоналу. Новими можливостями для боту можуть слугувати функції визначення волатильності монети, пошук популярних монет за добу, виведення актуальних новин зі світу криптовалют. Також дані для системи моніторингу можна отримувати з певних бірж, а не агрегатора, який усереднює дані з декількох бірж.

Рисунок Б15 – Шляхи вдосконалення програмної системи

Висновки

Під час виконання кваліфікаційної роботи було проаналізовано тему роботи, її актуальність, визначено мету дослідження та практичне значення роботи, описано методи, об’єкт та предмет дослідження .

Визначено методи дослідження, технології та інструменти для розробки системи.

Також було спроєктовано макети інтерфейсу майбутньої системи та побудовано UML-діаграми.

У ході виконання кваліфікаційної роботи усі поставлені завдання були виконані в повному обсязі. Визначено шляхи для вдосконалення розробленої системи.

Рисунок Б16 – Висновки

Додаток В.1
Лістинг модуля generate

```
#create_bot.py
from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram import types
import option
from aiogram.contrib.fsm_storage.memory import MemoryStorage


storage = MemoryStorage()
bot = Bot(option.TOKEN)
dp = Dispatcher(bot, storage=storage)


async def set_default_commands(dp):
    await dp.bot.set_my_commands([
        types.BotCommand("start", "Запустити бота"),
        types.BotCommand("download", "Завантажити картки до БД (для
адміністратора)"),
        types.BotCommand("delete", "Очистити БД (для адміністратора)"),
        types.BotCommand("help", "Правила гри"),
        types.BotCommand("game", "Почати гру"),
        types.BotCommand("record", "ТОП-3+власний рекорд"),
    ])
```

Додаток В.2
Лістинг модуля main

```
from aiogram.utils import executor
from generate import commands
from generate import dp
import swap
import mainfunc
import graph
import exchange
import conv

async def start(_):
    print('Бот працює!')
    await commands(dp)

conv.register_conv_handlers(dp)
exchange.register_exchange_handlers(dp)
swap.register_swap_handlers(dp)
graph.register_graph_handlers(dp)
mainfunc.register_main_handlers(dp)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True, on_startup=start)
```

Додаток В.3
Лістинг модуля mainfunc

```
from aiogram import types
import markups as nav
from pycoingecko import CoinGeckoAPI
from aiogram.dispatcher.filters.state import State, StatesGroup
from aiogram.dispatcher import FSMContext
from aiogram import Dispatcher
from generate import bot


cg = CoinGeckoAPI()


class FSMconverter(StatesGroup):
    chose_coin_name = State()
    select_quantity = State()
    coin = State()


async def command_start(message: types.Message):
    await  bot.send_message(message.from_user.id, 'Вітаю,
{0.first_name}!'.format(message.from_user),
                            reply_markup=nav.mainMenu)


async def curs(message: types.Message):
    await bot.send_message(message.from_user.id, 'Курс',
reply_markup=nav.otherKurs)


async def mmenu(message: types.Message):
    await bot.send_message(message.from_user.id, 'Ви перейшли до головного
меню', reply_markup=nav.mainMenu)


async def other(message: types.Message):
    await bot.send_message(message.from_user.id, 'Ви перейшли до підменю',
reply_markup=nav.otherMenu)


async def obmennik(message: types.Message):
    await bot.send_message(message.from_user.id, 'Що бажаєте обміняти:',
reply_markup=nav.obMenu)
```

```
async def information(message: types.Message):
    await bot.send_message(message.from_user.id, 'Криптовалюти продовжують залишатися надзвичайно цікавим феноменом сучасності.\n\n'
                                                 'Швидко досягнувши популярності і ставши дуже затребуваними,цифрові монети продовжують бути неймовірно вигідним інструментом інвестування.\n\n'
                                                 'У зв`язку з бурхливим інтересом українців до світу криптовалют з`являються запити на прості системи, які дозволять  \n'
                                                 'швидко та легко зорієнтуватися на ринковому полі.\n\n'
                                                 'Тому основним завданням автора стало створення компактного, безкоштовного та публічного помічника для роботи з криптовалютами.\n\n'
                                                 'Основними завданнями якого є моніторинг курсів валют, візуалізація динаміки поведінки, калькуляція конвертування, обґрунтування пропозицій обміну.\n\n')
    await message.answer(
        text='Контакт автора: <a href="https://t.me/blonskym">Maxim Blonsky</a>',
        disable_web_page_preview=True, parse_mode='html')


async def help(message: types.Message):
    await bot.send_message(message.from_user.id,
'@ExchangeCryptoByBlonskyBot виконує функції моніторингу курсів криптовалют .\n\n'
                                                 'Бот володіє таким функціоналом:\n\n'
                                                 '-У головному меню представлено функцію моніторингу курсів криптовалют - кнопка Курс:\n'
                                                 '      -Натиснувши кнопку Пошук користувачу необхідно буде ввести назву потрібної криптовалюти.\n'
                                                 '      -Натиснувши кнопку Топ користувачу виводиться інформація про найцікавіші монети.\n\n'
                                                 '-У головному меню представлено функцію конвертації криптовалют - кнопка Конвертер:\n'
                                                 '      -Натиснувши кнопку Конвертер користувачу необхідно ввести назву потрібної криптовалюти після чого вказати її кількість.\n\n'
                                                 '-У головному меню представлено обмінник криптовалют - кнопка Обмінник:\n'
```

```
                                    '     -Натиснувши кнопку 🔃Обмінник
користувач може вибрати потрібну криптовалюту чи фіатну валюту, яку хоче
обміняти, \n'
                                    'після цього обрати на що користувач хоче
обміняти.\n\n'
                                    '-У підменю представлено побудову графіку
криптовалют - кнопка 📊Графік:\n'
                                    '     -Натиснувши кнопку 📊Графік та обравши
монету система побудує графік ціни та об`єму торгів криптовалюти.\n\n'
                                    '-У підменю представлено загальну інформацію
про систему - кнопка ℹ️Інформація. ')


async def graphMenu(message: types.Message):
    await bot.send_message(message.from_user.id, 'Оберіть криптовалюту для
побудови графіка',
                           reply_markup=nav.otherGraph)


async def converter(message: types.Message, state: FSMContext):
    await bot.send_message(message.from_user.id, 'Оберіть криптовалюту для
конвертації:',
                           reply_markup=types.ReplyKeyboardRemove())
    await FSMconverter.chose_coin_name.set()


async def clear(message: types.Message):
    await message.delete()
    await message.answer('❓Невідоме повідомлення')


def register_main_handlers(dp: Dispatcher):
    dp.register_message_handler(command_start, commands=['start'])
    dp.register_message_handler(curs, text='🎓Курс')
    dp.register_message_handler(mmenu, text='🔼Головне меню')
    dp.register_message_handler(other, text='🔽Підменю')
    dp.register_message_handler(obmennik, text='🔃Обмінник')
    dp.register_message_handler(information, text='ℹ️Інформація')
    dp.register_message_handler(help, text='🆘Допомога')
    dp.register_message_handler(graphMenu, text='📊Графік')
```

```
dp.register_message_handler(converter, text='💲Конвертер')
dp.register_message_handler(clear)
```

Додаток В.4
Лістинг модуля markups

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton,
InlineKeyboardMarkup, InlineKeyboardButton


btnMain = KeyboardButton('🔼Головне меню')

# --Main menu--
btnCur = KeyboardButton('🎓Курс')
btnConv = KeyboardButton('💲Конвертер')
btnObm = KeyboardButton('🔃Обмінник')
btnDrug = KeyboardButton('🔽Підменю')
mainMenu = ReplyKeyboardMarkup(resize_keyboard= True).add(btnCur,
btnConv, btnObm, btnDrug)


# --Other menu--
btnInfo = KeyboardButton('ℹ️Інформація')
btnGraf = KeyboardButton('📊Графік')
btnHelp = KeyboardButton('🆘Допомога')
otherMenu = ReplyKeyboardMarkup(resize_keyboard= True).add(btnInfo,
btnGraf, btnHelp, btnMain)

# --Graph menu--
btnGr = KeyboardButton('Bitcoin')
btnGr1 = KeyboardButton('BNB')
btnGr2 = KeyboardButton('Ethereum')
btnGr3 = KeyboardButton('Solana')
btnGr4 = KeyboardButton('Dogecoin')
otherGraph = ReplyKeyboardMarkup(resize_keyboard= True).add(btnGr, btnGr1,
btnGr2, btnGr3, btnGr4, btnMain)

# --Kurs menu--
btnSearch = KeyboardButton('🔎Пошук')
btnTop = KeyboardButton('🔝Топ')
otherKurs = ReplyKeyboardMarkup(resize_keyboard= True).add(btnSearch,
btnTop, btnMain)

# --Obmennik--
obBut1 = InlineKeyboardButton('BTC', callback_data='BTC')
obBut2 = InlineKeyboardButton('ETH', callback_data='ETH')
obBut3 = InlineKeyboardButton('LTC', callback_data='LTC')
obBut4 = InlineKeyboardButton('SOL', callback_data='SOL')
obBut5 = InlineKeyboardButton('DOGE', callback_data='DOGE')
```

```
obBut6 = InlineKeyboardButton('Privat24(UAH)', callback_data='Privat24')
obBut7 = InlineKeyboardButton('Monobank(UAH)', callback_data='Monobank')
obBut8 = InlineKeyboardButton('Ощадбанк(UAH)', callback_data='Ощадбанк')
obMenu = InlineKeyboardMarkup().add(obBut1, obBut2, obBut3, obBut4,
obBut5, obBut6, obBut7, obBut8)

# --PodObmennik--
podBut1 = InlineKeyboardButton('Privat24(UAH)', callback_data='UAH(btc)')
podBut2 = InlineKeyboardButton('Privat24(UAH)', callback_data='UAH(eth)')
podBut3 = InlineKeyboardButton('Privat24(UAH)', callback_data='UAH(ltc)')
podBut4 = InlineKeyboardButton('Privat24(UAH)', callback_data='UAH(sol)')
podBut5 = InlineKeyboardButton('Privat24(UAH)', callback_data='UAH(doge)')
podBut6 = InlineKeyboardButton('Monobank(UAH)',
callback_data='UAH(mbtc)')
podBut7 = InlineKeyboardButton('Monobank(UAH)',
callback_data='UAH(meth)')
podBut8 = InlineKeyboardButton('Monobank(UAH)', callback_data='UAH(mltc)')
podBut9 = InlineKeyboardButton('Monobank(UAH)', callback_data='UAH(msol)')
podBut10 = InlineKeyboardButton('Monobank(UAH)',
callback_data='UAH(mdoge)')
podBut11 = InlineKeyboardButton('Ощадбанк(UAH)',
callback_data='UAH(obtc)')
podBut12 = InlineKeyboardButton('Ощадбанк(UAH)',
callback_data='UAH(oeth)')
podBut13 = InlineKeyboardButton('Ощадбанк(UAH)',
callback_data='UAH(oltc)')
podBut14 = InlineKeyboardButton('Ощадбанк(UAH)',
callback_data='UAH(osol)')
podBut15 = InlineKeyboardButton('Ощадбанк(UAH)',
callback_data='UAH(odoge)')
podBut16 = InlineKeyboardButton('Bitcoin', callback_data='BTC(privat)')
podBut17 = InlineKeyboardButton('Bitcoin', callback_data='BTC(mono)')
podBut18 = InlineKeyboardButton('Bitcoin', callback_data='BTC(oshad)')
podMenu = InlineKeyboardMarkup().add(podBut1, podBut6, podBut11)
podMenu2 = InlineKeyboardMarkup().add(podBut2, podBut7, podBut12)
podMenu3 = InlineKeyboardMarkup().add(podBut3, podBut8, podBut13)
podMenu4 = InlineKeyboardMarkup().add(podBut4, podBut9, podBut14)
podMenu5 = InlineKeyboardMarkup().add(podBut5, podBut10, podBut15)
podMenu6 = InlineKeyboardMarkup().add(podBut16)
podMenu7 = InlineKeyboardMarkup().add(podBut17)
podMenu8 = InlineKeyboardMarkup().add(podBut18)
```

Додаток В.5
Лістинг модуля exchange

```
import markups as nav
from aiogram import Dispatcher
from generate import bot
from aiogram import types
from mainfunc import FSMconverter
from aiogram.dispatcher import FSMContext
from mainfunc import cg


async def top(message: types.Message):
    price = cg.get_price(ids='bitcoin, ethereum, binancecoin, solana, dogecoin',
vs_currencies='usd, uah')
    price2 = cg.get_coins_markets(ids='bitcoin, ethereum, binancecoin, solana,
dogecoin', vs_currency='usd')
    await bot.send_message(message.from_user.id, f'Найцікавіші монети:\n\n'
                                        f'Bitcoin = {price["bitcoin"]["usd"]}$ =
{price["bitcoin"]["uah"]}₴, Піковий курс за добу = {price2[0]["high_24h"]}$,
Найнижчий курс за добу = {price2[0]["low_24h"]}$, Добова зміна курсу =
{round(price2[0]["price_change_24h"], 4)}$\n\n'
                                        f'Ethereum = {price["ethereum"]["usd"]}$ =
{price["ethereum"]["uah"]}₴, Піковий курс за добу = {price2[1]["high_24h"]}$,
Найнижчий курс за добу = {price2[1]["low_24h"]}$, Добова зміна курсу =
{round(price2[1]["price_change_24h"], 4)}$\n\n'
                                        f'BNB = {price["binancecoin"]["usd"]}$ =
{price["binancecoin"]["uah"]}₴, Піковий курс за добу =
{price2[2]["high_24h"]}$, Найнижчий курс за добу = {price2[2]["low_24h"]}$,
Добова зміна курсу = {round(price2[2]["price_change_24h"], 4)}$\n\n'
                                        f'Solana = {price["solana"]["usd"]}$ =
{price["solana"]["uah"]}₴, Піковий курс за добу = {price2[3]["high_24h"]}$,
Найнижчий курс за добу = {price2[3]["low_24h"]}$, Добова зміна курсу =
{round(price2[3]["price_change_24h"], 4)}$\n\n'
                                        f'Dogecoin = {price["dogecoin"]["usd"]}$ =
{price["dogecoin"]["uah"]}₴, Піковий курс за добу = {price2[4]["high_24h"]}$,
Найнижчий курс за добу = {price2[4]["low_24h"]}$, Добова зміна курсу =
{round(price2[4]["price_change_24h"], 4)}$')


async def search(message: types.Message, state: FSMContext):
    await FSMconverter.coin.set()
    await bot.send_message(message.from_user.id, 'Введіть назву криптовалюти:',
                           reply_markup=types.ReplyKeyboardRemove())


async def chose_nam(message: types.Message, state: FSMContext):
```

```
    try:
        async with state.proxy() as data:
            data['name'] = message.text
            result = cg.get_price(ids=data['name'], vs_currencies='usd, uah')
            result2 = cg.get_coins_markets(vs_currency='usd')
            k = 0
            for r in result2:
                if r['id'] == data['name']:
                    break
                else:
                    k += 1
            await bot.send_message(message.from_user.id,
                                   f"Криптовалюта: {data['name']}\nПоточний курс:
{result[data['name']]['usd']}$ = {result[data['name']]['uah']}₴, Піковий курс за
добу = {result2[k]['high_24h']}$, Найнижчий курс за добу =
{result2[k]['low_24h']}$, Добова зміна курсу =
{round(result2[k]['price_change_24h'], 4)}$\n\n",
                                   reply_markup=nav.mainMenu)
    except KeyError as KE:
        print(KE)
        await bot.send_message(message.from_user.id, 'Такої криптовалюти не
існує, введіть назву існуючої монети:')
    else:
        await state.finish()


def register_exchange_handlers(dp: Dispatcher):
    dp.register_message_handler(top, text='🔝Топ')
    dp.register_message_handler(search, text='🔎Пошук')
    dp.register_message_handler(chose_nam, state=FSMconverter.coin)
```

Додаток В.6
Лістинг модуля graph

```
from aiogram import Dispatcher, types
from datetime import datetime
import numpy as np
import altair as alt
import pandas
from altair_saver import save
from generate import bot
from mainfunc import cg



def countLeapYears(d, m, y):
    years = y
    if m <= 2:
        years -= 1
    return years / 4 - years / 100 + years / 400


def getDifference(d, m, y):
    monthDays = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    n1 = y * 365 + d
    for i in range(m):
        n1 += monthDays[i]
    n1 += countLeapYears(d, m, y)
    n2 = y * 365 + 1
    for i in range(1):
        n2 += monthDays[i]
    n2 += countLeapYears(1, 1, y)
    return abs(n2 - n1) + 1


def nextdate(dFromStartYear, amount, y):
    res = dFromStartYear + amount
    m = 1
    d = res
    year = y
    monthDays = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if ((y % 4 == 0) or (y % 100 != 0) and (y % 400 == 0)):
        if (res >= 366):
            year = y + 1
            d = res - 366
    else:
        if (res >= 365):
            year = y + 1
```

```
            d = res - 365
    if ((year % 4 == 0) or (year % 100 != 0) and (year % 400 == 0)):
        monthDays[1] += 1
    for i in range(len(monthDays)):
        if (d > monthDays[i]):
            d = d - monthDays[i]
            m = m + 1
        else:
            break
    result = [int(d), int(m), int(year)]
    return result


def datetime_to_unix(year, month, day):
    dt = datetime(year, month, day)
    timestamp = (dt - datetime(1970, 1, 1)).total_seconds()
    return timestamp


def unix_to_datetime(unix_time):
    ts = int(unix_time / 1000 if len(str(unix_time)) > 10 else unix_time)  # /1000
handles milliseconds
    return datetime.utcfromtimestamp(ts).strftime('%Y-%m-%d
%H:%M%p').lower()


async def graph1(message: types.Message):
    d = datetime.now().day
    m = datetime.now().month
    y = datetime.now().year
    next = 1
    previous = -2
    dFromStartYear = getDifference(d, m, y)
    tomorrow = nextdate(dFromStartYear, next, y)
    prev = nextdate(dFromStartYear, previous, y)
    result = cg.get_coin_market_chart_range_by_id(
        id='bitcoin',
        vs_currency='usd',
        from_timestamp=datetime_to_unix(prev[2], prev[1], prev[0]),
        to_timestamp=datetime_to_unix(tomorrow[2], tomorrow[1], tomorrow[0]))
    result.keys()
    bit = cg.get_coins_markets(vs_currency='usd')
    # => dict_keys(['prices', 'market_caps', 'total_volumes'])
    time = [unix_to_datetime(i[0]) for i in result['prices']]
```

```
    p_array = np.array(result['prices'])
    price = p_array[:, 1]
    v_array = np.array(result['total_volumes'])
    volume = v_array[:, 1]
    df = pandas.DataFrame({'time': time, 'price': price, 'volume': volume})
    df.head()
    # Create y-axis
    base = alt.Chart(df).encode(x='time:O')
    # Create bars
    bar = base.mark_bar().encode(
        alt.Y(
            'volume:Q',
            scale=alt.Scale(domain=(0, bit[0]['total_volume'] * 1.5)),
        )
    )
    # Create line
    line = base.mark_line(color='orange').encode(
        alt.Y(
            'price:Q',
            axis=alt.Axis(titleColor='#5276A7'),
            scale=alt.Scale(domain=(bit[0]['current_price'] / 1.5, bit[0]['current_price']
* 1.5))
        )
    )
    # Build the chart
    chart = alt.layer(bar, line).resolve_scale(y='independent').properties(width=800,
title='Bitcoin Price & Volume')
    # Configure title
    chart.configure_title(
        fontSize=20,
        font='Helvetica',
        color='black',
        offset=10
    )
    save(chart, "chart.png")
    await bot.send_photo(message.from_user.id, open('chart.png', 'rb'),
                         caption='Графік ціни та об`єму торгів монети за 3 дні')


async def graph2(message: types.Message):
    # try:
    d = datetime.now().day
    m = datetime.now().month
    y = datetime.now().year
```

```
    next = 1
    previous = -2
    dFromStartYear = getDifference(d, m, y)
    tomorrow = nextdate(dFromStartYear, next, y)
    prev = nextdate(dFromStartYear, previous, y)
    result = cg.get_coin_market_chart_range_by_id(
        id='ethereum',
        vs_currency='usd',
        from_timestamp=datetime_to_unix(prev[2], prev[1], prev[0]),
        to_timestamp=datetime_to_unix(tomorrow[2], tomorrow[1], tomorrow[0]))
    result.keys()
    bit = cg.get_coins_markets(vs_currency='usd')
    # => dict_keys(['prices', 'market_caps', 'total_volumes'])
    time = [unix_to_datetime(i[0]) for i in result['prices']]
    p_array = np.array(result['prices'])
    price = p_array[:, 1]
    v_array = np.array(result['total_volumes'])
    volume = v_array[:, 1]
    df = pandas.DataFrame({'time': time, 'price': price, 'volume': volume})
    df.head()
    # Create y-axis
    base = alt.Chart(df).encode(x='time:O')
    # Create bars
    bar = base.mark_bar().encode(
        alt.Y(
            'volume:Q',
            scale=alt.Scale(domain=(0, bit[1]['total_volume'] * 2)),
        )
    )
    # Create line
    line = base.mark_line(color='orange').encode(
        alt.Y(
            'price:Q',
            axis=alt.Axis(titleColor='#5276A7'),
            scale=alt.Scale(domain=(bit[1]['current_price'] / 1.5, bit[1]['current_price']
* 1.5))
        )
    )
    # Build the chart
    chart = alt.layer(bar, line).resolve_scale(y='independent').properties(width=800,
title='Ethereum Price & Volume')
    # Configure title
    chart.configure_title(
        fontSize=20,
```

```
        font='Helvetica',
        color='black',
        offset=10
    )
    save(chart, "chart.png")
    await bot.send_photo(message.from_user.id, open('chart.png', 'rb'),
                         caption='Графік ціни та об`єму торгів монети за 3 дні')


async def graph3(message: types.Message):
    # try:
    d = datetime.now().day
    m = datetime.now().month
    y = datetime.now().year
    next = 1
    previous = -2
    dFromStartYear = getDifference(d, m, y)
    tomorrow = nextdate(dFromStartYear, next, y)
    prev = nextdate(dFromStartYear, previous, y)
    result = cg.get_coin_market_chart_range_by_id(
        id='binancecoin',
        vs_currency='usd',
        from_timestamp=datetime_to_unix(prev[2], prev[1], prev[0]),
        to_timestamp=datetime_to_unix(tomorrow[2], tomorrow[1], tomorrow[0]))
    result.keys()
    bit = cg.get_coins_markets(vs_currency='usd')
    # => dict_keys(['prices', 'market_caps', 'total_volumes'])
    time = [unix_to_datetime(i[0]) for i in result['prices']]
    p_array = np.array(result['prices'])
    price = p_array[:, 1]
    v_array = np.array(result['total_volumes'])
    volume = v_array[:, 1]
    df = pandas.DataFrame({'time': time, 'price': price, 'volume': volume})
    df.head()
    # Create y-axis
    base = alt.Chart(df).encode(x='time:O')
    # Create bars
    bar = base.mark_bar().encode(
        alt.Y(
            'volume:Q',
            scale=alt.Scale(domain=(0, bit[4]['total_volume'] * 2)),
        )
    )
    # Create line
```

```
    line = base.mark_line(color='orange').encode(
        alt.Y(
            'price:Q',
            axis=alt.Axis(titleColor='#5276A7'),
            scale=alt.Scale(domain=(bit[4]['current_price'] / 1.5, bit[4]['current_price']
* 1.5))
        )
    )
    # Build the chart
    chart = alt.layer(bar, line).resolve_scale(y='independent').properties(width=800,
title='BNB Price & Volume')
    # Configure title
    chart.configure_title(
        fontSize=20,
        font='Helvetica',
        color='black',
        offset=10
    )
    save(chart, "chart.png")
    await bot.send_photo(message.from_user.id, open('chart.png', 'rb'),
                         caption='Графік ціни та об`єму торгів монети за 3 дні')


async def graph4(message: types.Message):
    # try:
    d = datetime.now().day
    m = datetime.now().month
    y = datetime.now().year
    next = 1
    previous = -2
    dFromStartYear = getDifference(d, m, y)
    tomorrow = nextdate(dFromStartYear, next, y)
    prev = nextdate(dFromStartYear, previous, y)
    result = cg.get_coin_market_chart_range_by_id(
        id='solana',
        vs_currency='usd',
        from_timestamp=datetime_to_unix(prev[2], prev[1], prev[0]),
        to_timestamp=datetime_to_unix(tomorrow[2], tomorrow[1], tomorrow[0]))
    result.keys()
    bit = cg.get_coins_markets(vs_currency='usd')
    # => dict_keys(['prices', 'market_caps', 'total_volumes'])
    time = [unix_to_datetime(i[0]) for i in result['prices']]
    p_array = np.array(result['prices'])
    price = p_array[:, 1]
```

```
    v_array = np.array(result['total_volumes'])
    volume = v_array[:, 1]
    df = pandas.DataFrame({'time': time, 'price': price, 'volume': volume})
    df.head()
    # Create y-axis
    base = alt.Chart(df).encode(x='time:O')
    # Create bars
    bar = base.mark_bar().encode(
        alt.Y(
            'volume:Q',
            scale=alt.Scale(domain=(0, bit[8]['total_volume'] * 2)),
        )
    )
    # Create line
    line = base.mark_line(color='orange').encode(
        alt.Y(
            'price:Q',
            axis=alt.Axis(titleColor='#5276A7'),
            scale=alt.Scale(domain=(bit[8]['current_price'] / 1.5, bit[8]['current_price']
* 1.5))
        )
    )
    # Build the chart
    chart = alt.layer(bar, line).resolve_scale(y='independent').properties(width=800,
title='Solana Price & Volume')
    # Configure title
    chart.configure_title(
        fontSize=20,
        font='Helvetica',
        color='black',
        offset=10
    )
    save(chart, "chart.png")
    await bot.send_photo(message.from_user.id, open('chart.png', 'rb'),
                         caption='Графік ціни та об`єму торгів монети за 3 дні')


async def graph5(message: types.Message):
    # try:
    d = datetime.now().day
    m = datetime.now().month
    y = datetime.now().year
    next = 1
    previous = -2
```

```
    dFromStartYear = getDifference(d, m, y)
    tomorrow = nextdate(dFromStartYear, next, y)
    prev = nextdate(dFromStartYear, previous, y)
    result = cg.get_coin_market_chart_range_by_id(
        id='dogecoin',
        vs_currency='usd',
        from_timestamp=datetime_to_unix(prev[2], prev[1], prev[0]),
        to_timestamp=datetime_to_unix(tomorrow[2], tomorrow[1], tomorrow[0]))
    result.keys()
    bit = cg.get_coins_markets(vs_currency='usd')
    # => dict_keys(['prices', 'market_caps', 'total_volumes'])
    time = [unix_to_datetime(i[0]) for i in result['prices']]
    p_array = np.array(result['prices'])
    price = p_array[:, 1]
    v_array = np.array(result['total_volumes'])
    volume = v_array[:, 1]
    df = pandas.DataFrame({'time': time, 'price': price, 'volume': volume})
    df.head()
    # Create y-axis
    base = alt.Chart(df).encode(x='time:O')
    # Create bars
    bar = base.mark_bar().encode(
        alt.Y(
            'volume:Q',
            scale=alt.Scale(domain=(0, bit[9]['total_volume'] * 1.3)),
        )
    )
    # Create line
    line = base.mark_line(color='orange').encode(
        alt.Y(
            'price:Q',
            axis=alt.Axis(titleColor='#5276A7'),
            scale=alt.Scale(domain=(bit[9]['current_price'] / 1.5, bit[9]['current_price']
* 1.5))
        )
    )
    # Build the chart
    chart = alt.layer(bar, line).resolve_scale(y='independent').properties(width=800,
title='Dogecoin Price & Volume')
    # Configure title
    chart.configure_title(
        fontSize=20,
        font='Helvetica',
        color='black',
```

```
        offset=10
    )
    save(chart, "chart.png")
    await bot.send_photo(message.from_user.id, open('chart.png', 'rb'),
                         caption='Графік ціни та об`єму торгів монети за 3 дні')


def register_graph_handlers(dp: Dispatcher):
    dp.register_message_handler(graph1, text='Bitcoin')
    dp.register_message_handler(graph2, text='Ethereum')
    dp.register_message_handler(graph3, text='BNB')
    dp.register_message_handler(graph4, text='Solana')
    dp.register_message_handler(graph5, text='Dogecoin')
```

Додаток В.7
Лістинг модуля conv

```
import markups as nav
from aiogram import Dispatcher
from generate import bot
from aiogram import types
from mainfunc import FSMconverter
from aiogram.dispatcher import FSMContext
from mainfunc import cg



async def chose_name(message: types.Message, state: FSMContext):
    try:
        async with state.proxy() as data:
            data['name'] = message.text
            result = cg.get_price(ids=data['name'], vs_currencies='usd, uah')
        print(result[data['name']])
        await FSMconverter.select_quantity.set()
        await bot.send_message(message.from_user.id, 'Введіть кількість:')
    except KeyError as KE:
        print(KE)
        await bot.send_message(message.from_user.id, 'Такої криптовалюти не
існує, введіть назву існуючої монети:')
        await FSMconverter.chose_coin_name.set()


async def chose_amount(message: types.Message, state: FSMContext):
    try:
        async with state.proxy() as data:
            data['amount'] = float(message.text)
            result = cg.get_price(ids=data['name'], vs_currencies='usd, uah')
            p = round(data['amount'] * result[data['name']]['usd'], 2)
            p2 = round(data['amount'] * result[data['name']]['uah'], 2)
            await bot.send_message(message.from_user.id,
                                   f"Криптовалюта: {data['name']}\nВартість: {p}$ =
{p2}₴", reply_markup=nav.mainMenu)
            await state.finish()
    except ValueError as VE:
        print(VE)
        await bot.send_message(message.from_user.id, 'Введіть коректну
кількість:')
        await FSMconverter.select_quantity.set()
```

```
def register_conv_handlers(dp: Dispatcher):
    dp.register_message_handler(chose_name,
state=FSMconverter.chose_coin_name)
    dp.register_message_handler(chose_amount,
state=FSMconverter.select_quantity)
```

Додаток В.8
Лістинг модуля swap

```
from aiogram import Dispatcher, types
from aiogram.dispatcher.filters import Text
import markups as nav


async def obmbtc(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu)


async def obmetc(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu2)


async def obmltc(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu3)


async def obmsol(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu4)


async def obmdoge(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu5)


async def obmpri24(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu6)

async def obmmono(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu7)

async def obmoshad(callback: types.CallbackQuery):
    await callback.message.edit_text(text='На що бажаєте обміняти:',
reply_markup=nav.podMenu8)

async def obmprivat24btc(callback: types.CallbackQuery):
    await callback.message.delete()
```

```
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/BTC-
P24UAH/">Bitcoin > Privat24(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmprivat24eth(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/ETH-
P24UAH/">Ethereum > Privat24(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmprivat24ltc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/LTC-
P24UAH/">Litecoin > Privat24(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmprivat24sol(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/SOL-
P24UAH/">Solana > Privat24(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmprivat24doge(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/DGC-
P24UAH/">Dogecoin > Privat24(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmmonobtc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/BTC-
MONOBUAH/">Bitcoin > Monobank(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')
```

```
async def obmmonoeth(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/ETH-
MONOBUAH/">Ethereum > Monobank(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmmonoltc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/LTC-
MONOBUAH/">Litecoin > Monobank(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmmonosol(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/SOL-
MONOBUAH/">Solana > Monobank(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmmonodoge(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/DGC-
MONOBUAH/">Dogecoin > Monobank(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmoshbtc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/BTC-
OSHBUAH/">Bitcoin > Ощадбанк(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmosheth(callback: types.CallbackQuery):
    await callback.message.delete()
```

```
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/ETH-
OSHBUAH/">Ethereum > Ощадбанк(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmoshltc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/LTC-
OSHBUAH/">Litecoin > Ощадбанк(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmoshsol(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/SOL-
OSHBUAH/">Solana > Ощадбанк(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmoshdoge(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/DGC-
OSHBUAH/">Dogecoin > Ощадбанк(UAH)</a>',
        disable_web_page_preview=True, parse_mode='html')


async def obmpribtc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/P24UAH-
BTC/">Privat24(UAH) > Bitcoin</a>',
        disable_web_page_preview=True, parse_mode='html')

async def obmmonobtc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/MONOBUAH-
BTC/">Monobank(UAH) > Bitcoin</a>',
        disable_web_page_preview=True, parse_mode='html')
```

```
async def obmoshadbtc(callback: types.CallbackQuery):
    await callback.message.delete()
    await callback.message.answer(
        text='Перейти: <a href="https://exchangesumo.com/obmen/OSHBUAH-
BTC/">Ощадбанк(UAH) > Bitcoin</a>',
        disable_web_page_preview=True, parse_mode='html')


def register_swap_handlers(dp: Dispatcher):
    dp.register_callback_query_handler(obmbtc, Text(equals='BTC'))
    dp.register_callback_query_handler(obmetc, Text(equals='ETH'))
    dp.register_callback_query_handler(obmltc, Text(equals='LTC'))
    dp.register_callback_query_handler(obmsol, Text(equals='SOL'))
    dp.register_callback_query_handler(obmdoge, Text(equals='DOGE'))
    dp.register_callback_query_handler(obmpri24, Text(equals='Privat24'))
    dp.register_callback_query_handler(obmmono, Text(equals='Monobank'))
    dp.register_callback_query_handler(obmoshad, Text(equals='Ощадбанк'))
    dp.register_callback_query_handler(obmprivat24btc, Text(equals='UAH(btc)'))
    dp.register_callback_query_handler(obmprivat24eth, Text(equals='UAH(eth)'))
    dp.register_callback_query_handler(obmprivat24ltc, Text(equals='UAH(ltc)'))
    dp.register_callback_query_handler(obmprivat24sol, Text(equals='UAH(sol)'))
    dp.register_callback_query_handler(obmprivat24doge,
Text(equals='UAH(doge)'))
    dp.register_callback_query_handler(obmmonobtc, Text(equals='UAH(mbtc)'))
    dp.register_callback_query_handler(obmmonoeth, Text(equals='UAH(meth)'))
    dp.register_callback_query_handler(obmmonoltc, Text(equals='UAH(mltc)'))
    dp.register_callback_query_handler(obmmonosol, Text(equals='UAH(msol)'))
    dp.register_callback_query_handler(obmmonodoge,
Text(equals='UAH(mdoge)'))
    dp.register_callback_query_handler(obmoshbtc, Text(equals='UAH(obtc)'))
    dp.register_callback_query_handler(obmosheth, Text(equals='UAH(oeth)'))
    dp.register_callback_query_handler(obmoshltc, Text(equals='UAH(oltc)'))
    dp.register_callback_query_handler(obmoshsol, Text(equals='UAH(osol)'))
    dp.register_callback_query_handler(obmoshdoge, Text(equals='UAH(odoge)'))
    dp.register_callback_query_handler(obmpribtc, Text(equals='BTC(privat)'))
    dp.register_callback_query_handler(obmmonobtc, Text(equals='BTC(mono)'))
    dp.register_callback_query_handler(obmoshadbtc, Text(equals='BTC(oshad)'))
```

Додаток В.9
Лістинг модулю config

```
TOKEN = "5367727260:AAHGSu2s6NX8LCvsBkGQKCN5LkN1lqxg2kg"
```

Додаток В.10
Лістинг тесту обробника, який зчитує кількість валюти

```
import unittest
from exchange import chose_nam
from conv import chose_name, chose_amount
from mainfunc import FSMconverter


class Test(unittest.TestCase):
    def test_check(self):
        self.assertRaises(ValueError, chose_amount, -5, state=
FSMconverter.select_quantity)
        self.assertRaises(ValueError, chose_amount, 0, state=
FSMconverter.select_quantity)
```