Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ТЕЛЕГРАМ БОТ ДЛЯ ОБЛІКУ ФІНАНСІВ З ВИКОРИСТАННЯМ БІБЛІОТЕКИ AIOGRAM
TELEGRAM BOT FOR FINANCIAL ACCOUNTING USING THE AIOGRAM LIBRARY

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Гринюк Максим Романович

(підпис)

Керівник:
к.т.н., старший викладач
Гринюк Сергій Васильович

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Гринюку Максиму Романовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Телеграм бот для обліку фінансів з використанням бібліотеки Aiogram*

Керівник роботи *Гринюк Сергій Васильович, к.т.н., старший викладач*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Загальні відомості про Телеграм ботів
Детальний аналіз засобів розробки
Розробка Телеграм боту для обліку фінансів
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Знімки екрану інтерфейсу користувача телеграм бота та його аналоги
Лістинги коду у вигляді рисунків
Структурні та архітектурні схеми, діаграми
Моделі бази даних

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Загальні відомості про Телеграм ботів	*Гринюк С.В.*		
Детальний аналіз засобів розробки	*Гринюк С.В.*		
Розробка Телеграм боту для обліку фінансів	*Гринюк С.В.*		
Висновки	*Гринюк С.В.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	10.11.2022 р.	Виконано
2.	*Огляд літератури на тему розробки ігор*	22.11.2022 р.	Виконано
3.	*Дослідження предметної області*	12.01.2023 р.	Виконано
4.	*Вибір засобів розробки*	14.02.2023 р.	Виконано
5.	*Розробка та тестування телеграм-боту*	26.03.2023 р.	Виконано
6.	*Оформлення матеріалів роботи*	03.04.2023 р.	Виконано
7.	*Нормоконтроль*	20.05.2023 р.	Виконано
8.	*Інструментальна перевірка на академічний плагіат*	31.05.2023 р	Виконано
9.	*Представлення кваліфікаційної роботи бакалавра до захисту*	06.06.2023 р.	Виконано

Здобувач вищої освіти _____ (Гринюк М.Р.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Гринюк С.В.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Гринюк М.Р. Телеграм бот для обліку фінансів з використанням бібліотеки Aiogram. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, одного додатку.

У розділі 1 були надані загальні відомості про предметну область та виконано порівняльний аналіз аналогічних програм та інструментів для обліку фінансів. Аналізуючи ці інструменти, виявлено їх переваги та недоліки.

У розділі було 2 проведено детальний аналіз засобів розробки. Було обґрунтовано вибір технологій для реалізації Telegram–бота, окреслено функціонально–структурну схему роботи Telegram–боту, а також створено модель бази даних PostgreSQL для зберігання необхідної інформації.

У розділі 3 було проведено безпосередню розробку Telegram–бота. Було створено сам бот та спроектовано його структурної схему, розглянуто механізми захисту даних користувача для забезпечення безпеки і конфіденційності, а також розглянуті методи тестування Telegram–боту для впевненості в його правильному функціонуванні.

Об'єкт дослідження – технології розробки сучасних веб–інструментів.

Предмет дослідження – Телеграм бот для обліку фінансів з використанням бібліотеки Aiogram і мови програмування Python.

Метою роботи є розробка ефективного Телеграм бота для забезпечення зручного обліку фінансів та контролю мультивалютного грошового портфоліо, а також дослідження основних понять, сфер застосування та аналогічних програм та інструментів для обліку фінансів.

Ключові слова: Telegram–бот, облік фінансів, курси валют, Python, Aiogram, веб–технології.

ANNOTATION

Hryniuk M.R. Telegram bot for financial accounting using the Aiogram library. Manuscript.

Bachelor's qualification work of the "Computer Engineering" educational program, specialization 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

The qualification work consists of an introduction, three sections, conclusions, a list of references, and one appendix.

Section 1 provides general information about the subject area and a comparative analysis of similar programs and tools for financial accounting. By analyzing these tools, their advantages and disadvantages were identified.

Section 2 presents a detailed analysis of the development tools. The selection of technologies for implementing the Telegram bot is justified, and the functional–structural scheme of the bot's operation is outlined. Furthermore, a PostgreSQL database model is created to store the necessary information.

Section 3 focuses on the actual development of the Telegram bot. The bot itself is created, and its structural scheme is designed. Mechanisms for ensuring user data protection and security are discussed, along with methods for testing the Telegram bot to ensure proper functionality.

Object of research – the development technologies of modern web tools.

Subject of research – Telegram bot for financial management using the Aiogram library and the Python programming language.

The aim of the work is to develop an efficient Telegram bot that provides convenient financial management and control of a multi–currency portfolio. Additionally, the study aims to explore the fundamental concepts, application areas, and similar programs and tools for financial management.

Keywords: Telegram bot, financial management, currency exchange rates, Python, Aiogram, web technologies.

ЗМІСТ

ВСТУП

Актуальність теми дослідження полягає в зростаючій популярності та використанні Телеграм ботів в якості інструмента для виконання практично будь–яких завдань. У сучасному світі, де швидкість і доступність інформації відіграють ключову роль, використання месенджерів, зокрема Telegram, стає все більш популярним способом спілкування та отримання різних сервісів. Водночас, люди все більше усвідомлюють важливість ефективного управління своїми фінансами, а також потребу в точній інформації про стан їхнього грошового портфеля.

Метою роботи є розробка ефективного Телеграм бота для забезпечення зручного обліку фінансів та контролю мультивалютного грошового портфоліо, а також дослідження основних понять, сфер застосування та аналогічних програм та інструментів для обліку фінансів.

Об'єкт дослідження – технології розробки сучасних веб–інструментів.

Предмет дослідження – Телеграм бот для обліку фінансів з використанням бібліотеки Aiogram і мови програмування Python.

Були сформовані наступні цілі на кваліфікаційну роботу:

– дослідити загальну теоретичну базу і сфері розробки телеграм-ботів.

– створити функціонально-структурну схему розробки.

– реалізувати взаємодію з базою даних PostgreSQL для збереження даних про грошове портфоліо та курси валют.

– протестувати та підтвердити працездатність бота, виконавши тестування з різними сценаріями використання.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ ПРО ТЕЛЕГРАМ БОТІВ

1.1 Основні поняття та сфери застосування Телеграм ботів

Проблема, розглянута у даній роботі, полягає у створенні інструмента для обліку власних фінансів, а конкретніше, відстежування стану мультивалютного грошового портфеля. Ця задача може бути важкою для людей, що володіють іншими грошовими валютами, окрім національної, оскільки потрібно моніторити кілька валют одночасно, а також зміну курсу цих валют у часі.

Для вирішення цієї проблеми було вирішено розробити Telegram–бота, який дозволить користувачам відстежувати стан свого портфеля в реальному часі. Для цього можна використовувати мову програмування Python та бібліотеку Aiogram, яка дозволить створити зручний інтерфейс для взаємодії з користувачем.

Телеграм боти – це програмні агенти, які можуть автоматично виконувати різні завдання в Телеграмі, інтерактивно спілкуватися з користувачами та надавати їм різні послуги. Вони можуть бути використані для автоматизації різних процесів, які пов'язані зі збором інформації, комунікацією з користувачами, організацією опитувань та багатьох інших завдань [1].

Принцип роботи телеграм ботів полягає в тому, що вони підключаються до серверів Телеграму та обмінюються повідомленнями з користувачами [2]. Таким чином, користувач може взаємодіяти з ботом через повідомлення, команди або кнопки, які можуть бути вбудовані в повідомлення. Телеграм бот може обробляти запити користувачів, відправляти відповіді, створювати сповіщення, а також збирати інформацію та передавати її до відповідних систем або баз даних [3].

Телеграм боти можуть бути створені за допомогою різноманітних мов програмування, включаючи Python, Java, Node.js та інші [4]. Для розробки телеграм бота зазвичай використовують спеціальні бібліотеки, які дозволяють легко підключатися до API Телеграму та використовувати його функціонал.

Телеграм боти можуть бути використані для різних цілей, включаючи бізнес, рекламу, маркетинг, освіту та розваги. Наприклад, бот може надавати користувачам інформацію про послуги чи товари, збирати замовлення, розповідати новини, створювати опитування та інше. Завдяки телеграм ботам, користувачі можуть легко та зручно здійснювати різні дії, отримувати швидку відповідь та здійснювати операції безпосередньо з Телеграму, що дуже зручно та ефективно для багатьох користувачів. Приклади сучасних Телеграм–ботів зображені на рисунках 1.1 та 1.2.

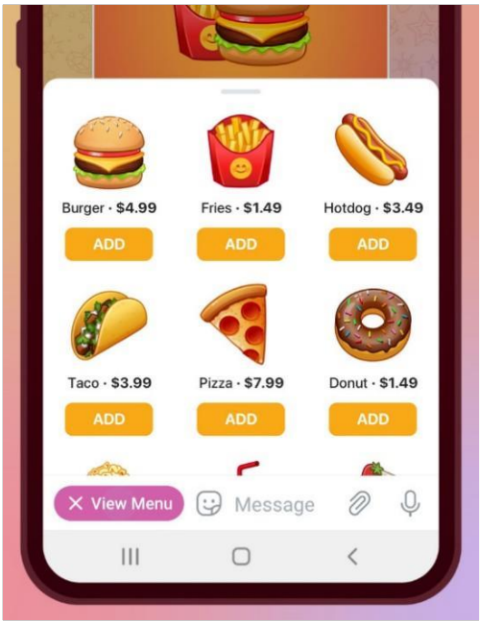

Рисунок 1.1 – Телеграм–бот для замовлення їжі

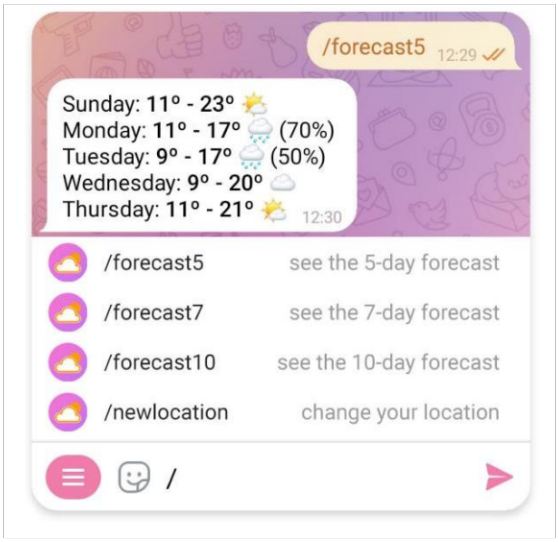

Рисунок 1.2 – Телеграм–бот для прогнозу погоди

З точки зору системних вимог, для роботи бота необхідний доступ до Інтернету та Telegram API, а також виділений веб–сервер для хостингу бота.

Одним з головних напрямів інформаційних потоків є отримання даних про курси валют з веб–ресурсів, наприклад, з фінансових сайтів або API провайдерів курсів валют. Для забезпечення коректної роботи бота необхідно стежити за оновленнями цих даних та забезпечувати їхню актуальність.

Функціональний опис вихідного об'єкту може включати такі можливості, як:

– додавання валют до портфеля та видалення їх з нього;

– відстеження поточного стану портфеля та зміни його в часі;

– відображення курсів валют у реальному часі.

Створення інструменту для відстеження стану мультивалютного грошового портфеля має доцільність, оскільки це може допомогти багатьом людям контролювати свої фінансові інвестиції та забезпечити більш точне інвестування валют. Така інформаційна система може бути корисною для будь–якого користувача, який має мультивалютний грошовий портфель.

Крім того, створення такої інформаційної системи може зробити процес відстеження стану портфеля більш ефективним та автоматизованим. Користувачам не доведеться вручну відслідковувати зміни курсів валют та стану свого портфеля, що може займати багато часу та зусиль. Завдяки боту, користувачі можуть отримувати інформацію про свій портфель в режимі реального часу та отримувати сповіщення про зміни стану портфеля, що може значно полегшити процес інвестування.

Таким чином, створення телеграм бота для відстеження стану мультивалютного грошового портфеля засобами мови Python з використанням бібліотеки Aiogram є доцільним та може бути корисним для широкого кола користувачів [5].

1.2 Порівняльний аналіз аналогічних програм та інструментів для обліку фінансів та контролю мультивалютного грошового портфоліо

Існують різні програми та рішення для відстеження стану мультивалютного грошового портфоліо, які можуть бути використані як аналоги для проектування телеграм бота. Деякі з них:

– Personal Capital (рис. 1.3) – це онлайн–сервіс для відстеження фінансового стану, який надає можливість додавати різні види інвестицій та рахунків. Програма має вбудовану функцію аналізу портфоліо та можливість створення користувацьких звітів.

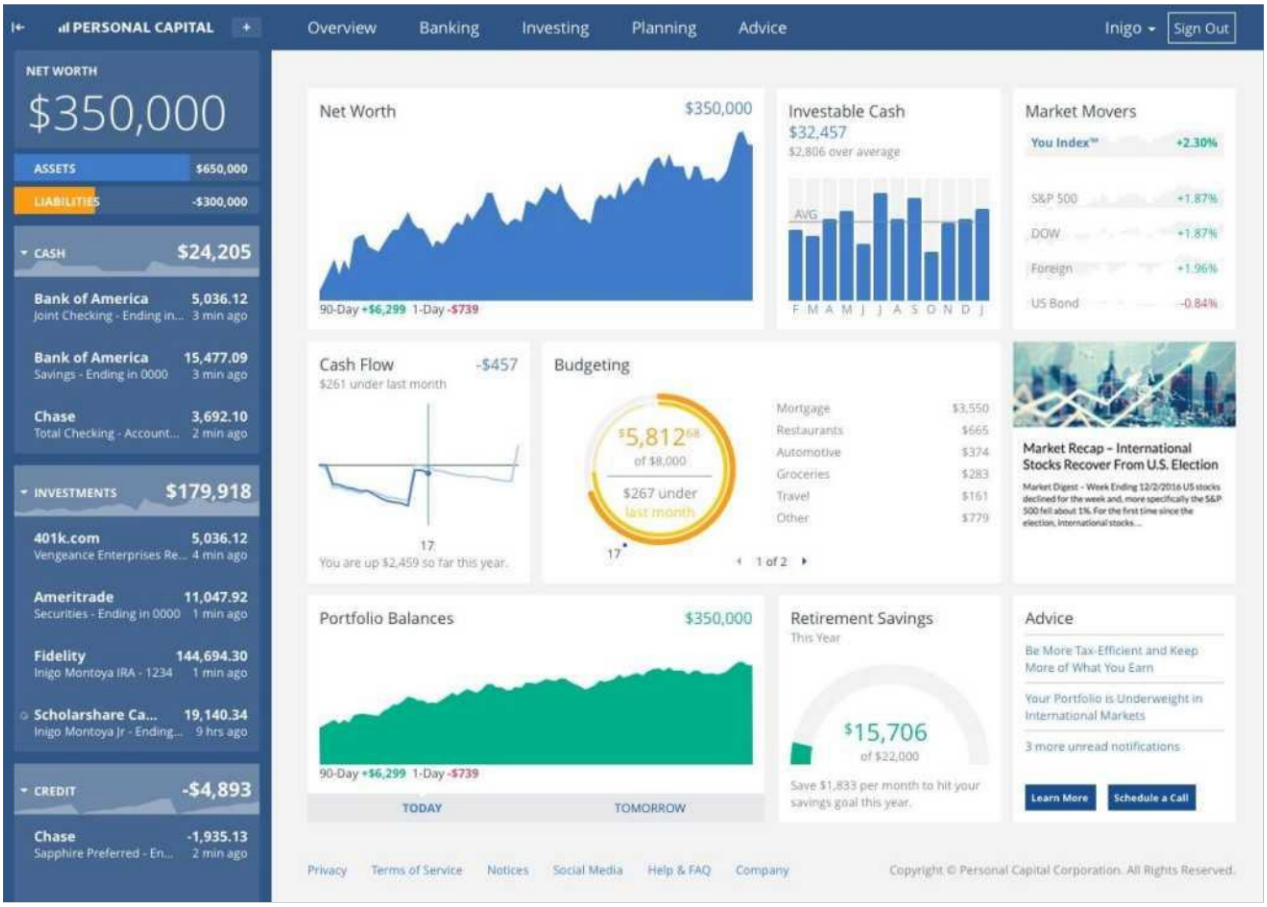

Рисунок 1.3 – Інтерфейс Personal Capital

– Mint (рис. 1.4) – це онлайн–сервіс для відстеження фінансового стану та керування фінансами. Вона дозволяє додавати різні банківські рахунки та інвестиції, має функцію автоматичного визначення категорій витрат.

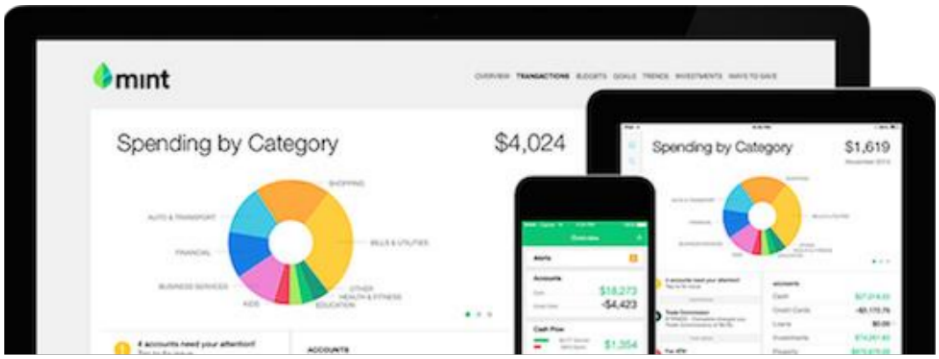

Рисунок 1.4 – Інтерфейс Mint

– Yahoo Finance (рис. 1.5) – це онлайн–сервіс, який надає можливість відстежувати фінансові новини та стан інвестиційного портфелю. Сервіс має різні інструменти для аналізу ринку та інвестицій, та надає користувачам можливість створення власних звітів.

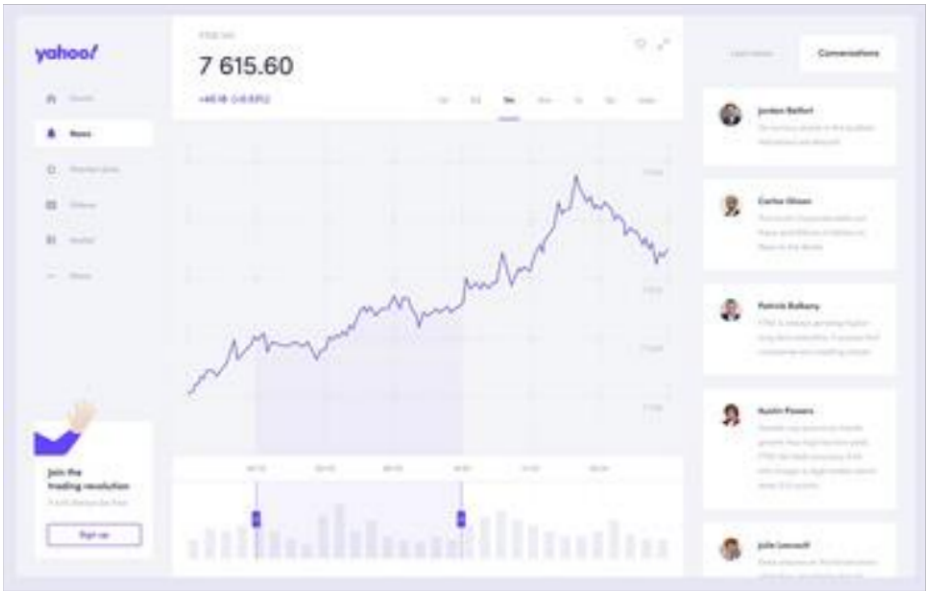

Рисунок 1.5 – Інтерфейс Yahoo Finance

Оскільки немає аналогічних телеграм ботів, або програм, які надають можливість відстеження стану мультивалютного грошового портфоліо за допомогою телеграм бота, розробка такого інструменту є цілком виправданою.

Проектований телеграм–бот має декілька переваг порівняно з існуючими рішеннями. Наприклад, він буде безкоштовним, оскільки використовуватиме

платформу Telegram API. Крім того, він буде більш універсальним, оскільки пропонуватиме функціональність для відстеження різних видів валют (в тому числі криптовалют). Також, користувачі зможуть використовувати бота безпосередньо у своєму месенджері, що дозволить їм зручно і швидко відстежувати своє портфоліо.

1.3 Аналіз та вибір засобів для розробки

Для розробки телеграм–боту для відстеження стану мультивалютного грошового портфоліо можна використати різноманітні засоби та бібліотеки.

Зазвичай для розробки телеграм ботів використовуються мови програмування, такі як Python, JavaScript, Java, PHP, Ruby, Go та інші. Однак, найпопулярнішою серед програмістів мовою для розробки телеграм ботів є Python. Вона має багато корисних бібліотек, таких як aiogram, pyTelegramBotAPI, Telebot та інші, що значно спрощують розробку телеграм ботів. Порівняння цих бібліотек представлено у таблиці 1.1.

Таблиця 1.1 – Порівняння Python–бібліотек для розробки Telegram–ботів

Функції/Характеристики	aiogram	pyTelegramBotAPI	Telebot
Підтримка асинхронної розробки	Так	Ні	Ні
Документація	Добре документована	Добре документована	Добре документована
Кількість користувачів на GitHub	3.6k	6.4k	4.8k
Підтримка inline–режиму	Так	Так	Так
Можливість отримувати статистику	Так	Ні	Ні
Кількість опцій	Велика	Велика	Велика
Складність вивчення	Висока	Середня	Середня

У загальному, всі три бібліотеки є добре документованими, підтримують відправку повідомлень з клавіатурами, роботу з медіа–файлами, inline–режим та кілька ботів в одному боті. Однак, aiogram відрізняється тим, що вона підтримує асинхронну розробку. PyTelegramBotAPI та Telebot є менш складними для вивчення, але перша має більшу кількість користувачів на GitHub. Через підтримку асинхронної розробки, була обрана саме бібліотека aiogram.

Асинхронне програмування – це підхід до розробки програм, що дозволяє виконувати багато задач одночасно, не блокуючи виконання коду. У традиційному синхронному програмуванні, код виконується послідовно, з закінченням кожної операції, перш ніж перейти до наступної. Це може призвести до блокування програми, якщо деякі завдання займають багато часу на виконання.

Асинхронне програмування дозволяє виконувати різні задачі одночасно, користуючись іншим підходом. Замість блокування програми, під час очікування результату, виконується наступне завдання. Таким чином, багато завдань можна виконувати одночасно. Це може збільшити продуктивність програми і знизити час її відгуку.

Асинхронне програмування зазвичай використовується в ситуаціях, коли програма повинна здійснювати багато операцій введення/виведення, наприклад, взаємодіяти з мережею або файловою системою, або коли виконання завдань може зайняти багато часу. Використання асинхронного програмування може допомогти знизити час відгуку програми і підвищити її продуктивність.

Крім того, Python є дуже простою та легко зрозумілою мовою програмування, що дозволяє розробникам швидко та ефективно створювати телеграм ботів.

Щодо баз даних, можна використовувати різноманітні системи, наприклад, PostgreSQL, MySQL або SQLite [6]. Їх порівняння представлене у таблиці 1.2. Вибір конкретної системи залежить від вимог до продукту та можливостей, що надає кожна система.

Таблиця 1.2 – Порівняння баз даних

Характеристика	PostgreSQL	MySQL	SQLite
Ліцензія	PostgreSQL License	GNU GPL	Public Domain
Тип системи	Реляційна	Реляційна	Реляційна
Розмір установки	~ 50 МБ	~ 400 МБ	~ 2 МБ
Максимальний розмір бази даних	безлімітний	32 ТБ	140 ТБ
Мови програмування для процедурного коду	C, C++, Java, Perl, Python, Ruby, Tcl	C, C++, Java, Perl, Python, Ruby	Немає
Підтримка підзапитів	Так	Так	Так
Підтримка віконних функцій	Так	Так	Ні
Підтримка FULL OUTER JOIN	Так	Так	Ні
Підтримка партиційних таблиць	Так	Так	Ні
Підтримка геоданих	Так	Так	Ні
Підтримка реплікації	Так	Так	Так
Підтримка шифрування	Так	Так	Так

Для зберігання даних була обрана система БД PostgreSQL, а саме хмарне рішення від Neon. Вони надають безкоштовний виділений хмарний сервер бази даних, з наступним тарифом: 10 можливих гілок БД, до 3 ГБ даних на кожну гілку та 100 обчислювальних годин на місяць. Даних можливостей цілком вистачить для розробки телеграм бота [7].

Для забезпечення роботи бота необхідний сервер, на якому буде запущена програма безпосередньо самого бота. Для цього можна використати різноманітні платформи, такі як Heroku, AWS, DigitalOcean тощо [8].

У порівнянні з існуючими рішеннями, такими як Wealthsimple або Personal Capital, телеграм бот має свої переваги, такі як легкість використання, невеликі витрати на розробку та підтримку, можливість швидкої інтеграції з телеграм–каналом та іншими сервісами. Однак, такий бот може мати обмежені можливості порівняно з повноцінною інвестиційною платформою, тому він може бути корисним для користувачів, які шукають просте та зручне рішення для відстеження свого портфоліо.

РОЗДІЛ 2
ДЕТАЛЬНИЙ АНАЛІЗ ЗАСОБІВ РОЗРОБКИ

2.1 Обґрунтування обраних технологій

Вирішення поставленого завдання, яке полягає у розробці телеграм бота для обліку фінансів, або іншими словами, відстеження стану мультивалютного грошового портфоліо, засобами мови Python з використанням бібліотеки aiogram та базою даних PostgreSQL, є доцільним з науково–технічного погляду з наступних причин:

– python є однією з найпопулярніших мов програмування у світі. Вона має велику кількість бібліотек та інструментів, які значно спрощують розробку програм. Бібліотека aiogram є потужним інструментом для розробки телеграм ботів на мові Python;

– postgreSQL є потужною та надійною системою керування базами даних. Вона підтримує багато функцій, які забезпечують високу продуктивність та безпеку. Також, PostgreSQL має відкритий вихідний код та широку спільноту, що дозволяє швидко вирішувати проблеми та забезпечувати підтримку [9];

– використання телеграм бота для відстеження стану мультивалютного грошового портфоліо дозволяє зручно та швидко отримувати інформацію про фінансовий стан користувача, а також забезпечує збереження цієї інформації у базі даних. Бот дозволяє вести звітність та забезпечує зручний доступ до інформації з будь–якого місця з доступом до мережі Інтернет;

– у порівнянні з іншими схемами, обрана схема розробки засобів мови Python з використанням бібліотеки aiogram та базою даних PostgreSQL є більш гнучкою та забезпечує високу продуктивність та безпеку. Також, вона є більш доступною з точки зору розробки та використання [10].

Також до переваг використання PostgreSQL можна віднести широкі можливості по оптимізації та налаштуванню бази даних, що дозволяє досягти високої продуктивності та ефективності роботи програми. Крім того, PostgreSQL має добру підтримку у спільноті розробників та активно розвивається [11].

Бібліотека aiogram також є оптимальним вибором для розробки телеграм ботів на мові Python. Вона має досить простий та зрозумілий інтерфейс, добре документована та має широкі можливості для розробки функціоналу бота, включаючи роботу з клавіатурами, розсилку повідомлень, обробку зображень та багато іншого.

Усі ці фактори разом зробили PostgreSQL та бібліотеку aiogram оптимальними виборами для розробки телеграм бота для обліку власних фінансів (відстеження стану мультивалютного грошового портфоліо).

2.2 Функціонально–структурна схема роботи Телеграм боту

Для отримання даних про актуальні курси, будуть використані такі способи взаємодії з веб–сторінками як API–запити та парсинг вебсайтів.

API (або Application Programming Interface) запити (рис 2.1) – це команди, які відправляються веб–додатком або мобільним додатком до API, щоб отримати певні дані або здійснити певну дію. API – це набір програмних інтерфейсів, що дозволяють різним програмам взаємодіяти один з одним, надаючи специфічний спосіб виконання певних дій.

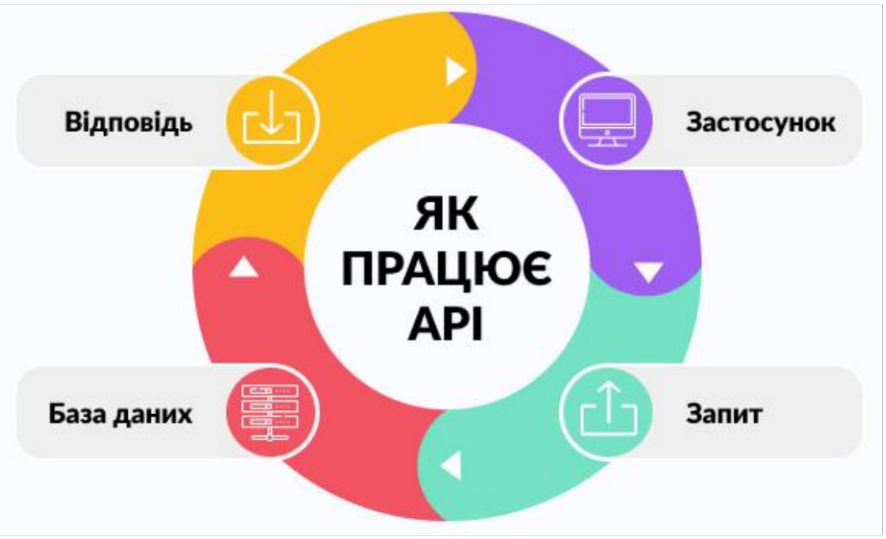

Рисунок 2.1 – Круговорот API запиту

API запити можуть бути виконані з різних джерел, таких як веб–браузер, програми для настільних комп'ютерів або мобільні додатки. Зазвичай, API запити передаються в форматі HTTP запитів, таких як GET, POST, PUT або DELETE.

Для використання HTTP запитів за допомогою Python буде використана бібліотека requests. Бібліотека requests в мові Python – це модуль для взаємодії з мережевими запитами, зокрема, з HTTP–запитами. Ця бібліотека дозволяє здійснювати HTTP–запити (GET, POST, PUT, DELETE та інші) до веб–сайтів і отримувати відповіді з сервера. Зразок використання API–запиту за допомогою бібліотеки requests представлений у рисунку 2.2.

```
#API-запит для конвертування валют
def main(amount, cur1, cur2):
    amount = float(comma_to_dot(amount))
    cur1 = cur1.upper()
    cur2 = cur2.upper()
    url = 'https://v6.exchangerate-api.com/v6/' + API_KEY + '/latest/' + cur1
    response = requests.get(url)
    data = response.json()
    currency = data['conversion_rates'][cur2]
    result = currency * amount
    if cur1 in currency_lst:
        cur1 = cur1 + ' (' + currency_lst[cur1] + ')'
    if cur2 in currency_lst:
        cur2 = cur2 + ' (' + currency_lst[cur2] + ')'
    return str(amount) + " " + cur1 + " = %.2f" % result + ' ' + cur2, result
```

Рисунок 2.2 – Функція для конвертування валют

Завдяки простому та зрозумілому синтаксису, бібліотека requests є дуже популярною серед розробників Python. Вона має декілька зручних функцій, таких як автоматична робота з cookies, підтримка аутентифікації, передача параметрів запиту, управління сесіями, зручне форматування та передача даних у JSON форматі та інше.

Парсинг веб–сайтів (рис 2.3) – це процес автоматичного отримання та обробки даних з веб–сайтів. Зазвичай використовуються програми–парсери, що

можуть зчитувати та аналізувати HTML–код веб–сторінок, а також витягувати з них різні дані.

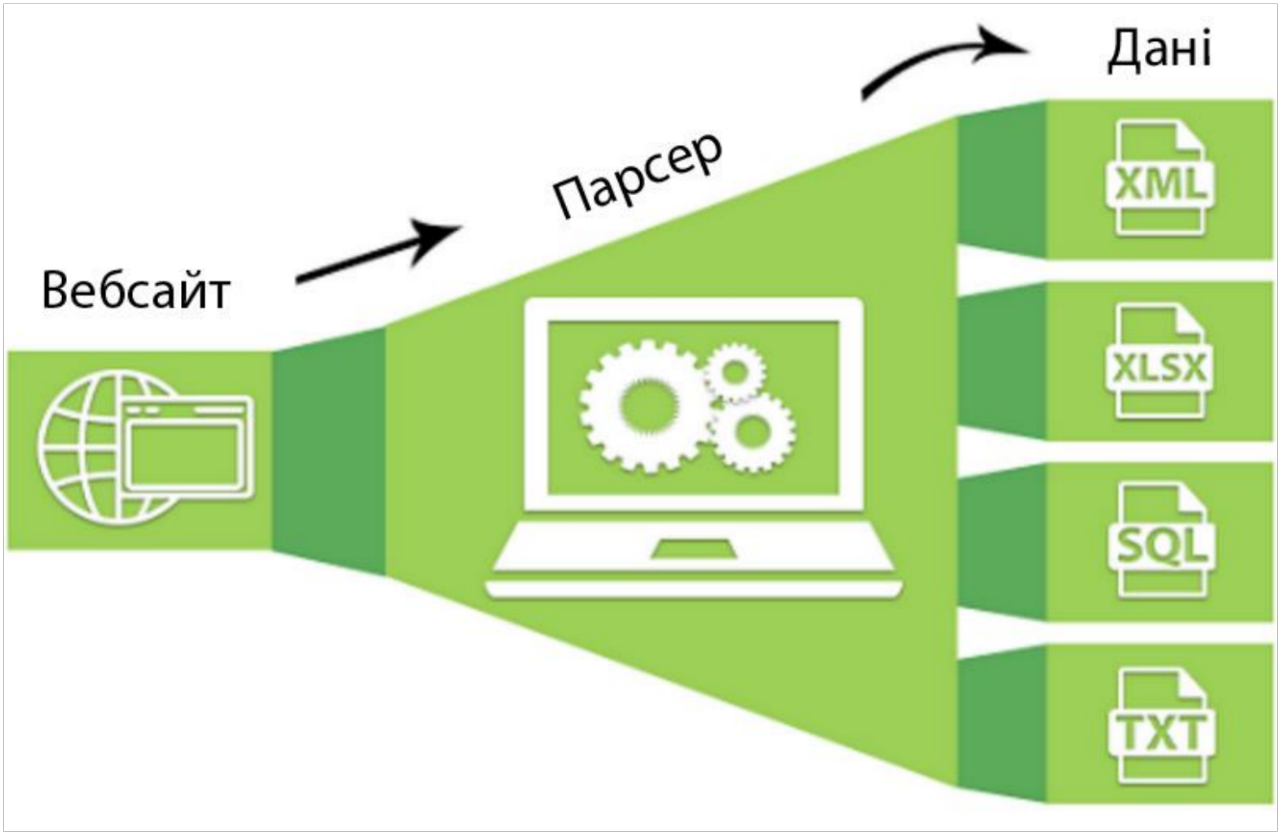

Рисунок 2.3 – Принцип парсингу веб–сайтів

Парсери можуть використовуватись для збору даних з різних джерел, таких як сторінки з пошукових систем, соціальних мереж, онлайн–магазинів тощо. Вони дозволяють витягувати з сайту не тільки текстову інформацію, але й зображення, відео, аудіофайли та інші медіа–елементи. Парсинг веб–сайтів може мати різні цілі, такі як збір даних для аналізу конкурентів, моніторинг цін на товари, відслідковування новин та інформації в різних галузях, наприклад, фінансах, спорті, політиці тощо.

Для парсингу веб–сайтів за допомогою мови програмування Python буде використана бібліотека Beautiful Soup 4. Beautiful Soup – це бібліотека для парсингу HTML і XML даних у Python. Вона дозволяє зручно знаходити, витягувати та модифікувати дані з веб–сторінок, які можуть бути складними з

точки зору HTML коду. Зразок парсингу вебсайту, використаний в коді телеграм–боту, представлений на рисунку 2.4.

```
#Парсинг скороченої назви криптовалюти з сайту CoinMarketCap
def get_short(name):
    url = "https://coinmarketcap.com/all/views/all/"
    html = requests.get(url).content
    soup = BeautifulSoup(html, 'html.parser')
    answer = soup.find(title = name).text
    return answer
```

Рисунок 2.4 – Функція для парсингу веб–сайту

Beautiful Soup забезпечує широкі можливості знаходження елементів на веб–сторінках за допомогою CSS селекторів, регулярних виразів, та інших методів. Вона також дозволяє екстрагувати дані з HTML атрибутів, створювати власні об'єкти даних зі знайдених елементів, та багато іншого.

Beautiful Soup є потужною та популярною бібліотекою для парсингу веб–сторінок, та добре підходить для багатьох задач в області веб–скрапінгу та обробки даних з Інтернету.

2.3 Створення моделі бази даних PostgreSQL

Для взаємодії з базою даних буде використана бібліотека Psycopg2, яка використовується для з'єднання з базами даних PostgreSQL [12]. Вона надає інтерфейс взаємодії з PostgreSQL з допомогою мови програмування Python, дозволяючи виконувати різні операції з базою даних, такі як вставка, оновлення, вилучення даних, а також виконання складних запитів і транзакцій [13].

Psycopg2 забезпечує стабільну, ефективну та безпечну реалізацію протоколу взаємодії між Python та PostgreSQL. Вона підтримує багато функцій PostgreSQL, таких як операції з бінарними даними, підготовлені запити, транзакції, роботу з курсорами, керування схемами бази даних та багато інших.

Psycopg2 є однією з найпопулярніших бібліотек для взаємодії з PostgreSQL у середовищі Python і використовується в багатьох веб–додатках, програмах аналізу даних, системах керування базами даних та багатьох інших застосуваннях, які використовують базу даних PostgreSQL.

Основні команди, які можна використовувати з psycopg2, включають:

– підключення до бази даних: psycopg2.connect(). Ця команда дозволяє встановити з'єднання з базою даних PostgreSQL, передаючи URL–адресу, користувача, пароль та інші параметри підключення;

– створення курсора: connection.cursor(). Ця команда створює курсор, який використовується для взаємодії з базою даних. Курсор виконує SQL–запити та отримує результати [14];

– виконання SQL–запитів: cursor.execute(). Ця команда виконує SQL–запит на базі даних, передаючи його як рядок в аргументі. Можна використовувати параметри, щоб передати значення до запиту та забезпечити захист від SQL–ін'єкції [15];

– отримання результатів: cursor.fetchone(), cursor.fetchmany(), cursor.fetchall(). Ці команди використовуються для отримання результатів виконаного запиту. fetchone() повертає один рядок результату, fetchmany(n) повертає n рядків результату, а fetchall() повертає всі рядки результату;

– збереження змін: connection.commit(). Ця команда зберігає зміни до бази даних після виконання запитів на вставку, оновлення або видалення даних;

– закриття з'єднання: connection.close(). Ця команда закриває з'єднання з базою даних після виконання всіх запитів та роботи з курсорами.

Для хостингу бази даних була обрана платформа Neon. Neon (рис. 2.5) – це повністю керований серверний PostgreSQL з безсерверним режимом роботи, який має безкоштовний тариф, що розділяє зберігання та обчислення, і пропонує сучасні можливості розробника, такі як безсерверність, гілки та необмежене зберігання даних. Neon дозволяє розробникам створювати гілку їх бази даних PostgreSQL, яка легко створюється з майже нульовим навантаженням, оскільки вона реалізована за допомогою техніки «копіювати при записі» (copy–on–write)

в сховищі Neon. Neon повністю сумісний з будь–яким додатком, що використовує офіційну версію PostgreSQL, і наразі підтримує PostgreSQL 14 та PostgreSQL 15. Neon дозволяє розробникам створювати гілки своєї бази даних Postgres, поліпшуючи їх робочий процес розробки [16, 17].

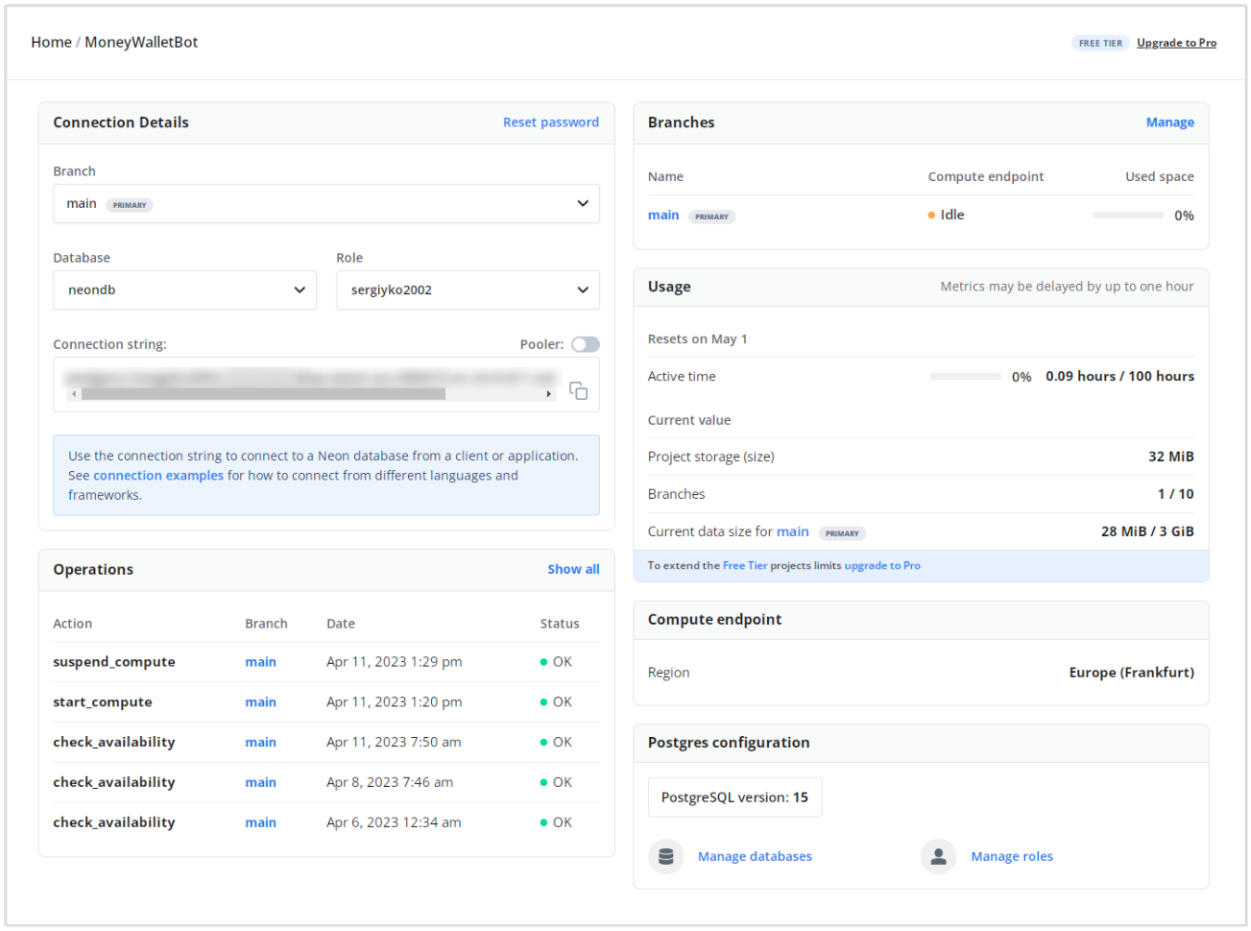

Рисунок 2.5 – Панель керування Neon

Взаємодія з БД відбувається за допомогою SQL–запитів. SQL (Structured Query Language) є мовою програмування, яка використовується для керування реляційними базами даних (RDBMS). SQL дозволяє взаємодіяти з базами даних, створювати, змінювати та видаляти дані, виконувати пошук і аналіз даних, а також керувати структурою бази даних [18].

Основні SQL–запити включають:

– SELECT: Використовується для вибору даних з бази даних. Запит SELECT дозволяє вибирати дані з однієї або декількох таблиць, вказувати умови фільтрації, сортування, об'єднання та агрегації даних.

– INSERT: Використовується для вставки нових записів в таблицю бази даних. Можна вказати значення для всіх або окремих стовпців.

– UPDATE: Використовується для оновлення наявних записів в таблиці бази даних. Можна встановлювати нові значення для вказаних стовпців та вказувати умови оновлення.

– DELETE: Використовується для видалення записів з таблиці бази даних. Можна вказати умови видалення для видалення певних записів.

Принцип взаємодії з базами даних за допомогою SQL полягає в тому, що програми використовують SQL–запити для взаємодії з базою даних. Запити передаються до бази даних, де вони виконуються, і результати повертаються програмі. SQL також дозволяє визначати структуру бази даних, включаючи таблиці, стовпці, індекси, ключі, обмеження, тригери та інші об'єкти бази даних.

Інформаційна модель БД являє собою кілька таблиць, такі як users, unregistered, crypto_usr, banks_usr та cash_usr. Вони містять різні поля, такі як id, first_name, last_name, nickname, capital, usd_capital, status, currency, amount, mark, name, які використовуються для зберігання різних видів інформації, таких як дані про користувачів, криптовалюти, їх банківські та готівкові рахунки [19].

Логічна модель БД містить наступні сутності та їх взаємозв'язки:

– users: сутність, яка містить дані про зареєстрованих користувачів, такі як ідентифікатор, ім'я, прізвище, нікнейм, капітал та статус (адміністратор чи користувач);

– unregistered: сутність, яка містить дані про незареєстрованих користувачів, такі як ідентифікатор, ім'я, прізвище, нікнейм;

– crypto_usr: сутність, яка містить дані про криптовалютний портфель користувачів, такі як валюта та її кількість;

– banks_usr: сутність, яка містить дані про банківський рахунок користувачів, такі як валюта та її кількість;

– cash_usr: сутність, яка містить дані про готівковий портфель користувачів, такі як валюта, кількість та значок валюти.

Фізична модель бази даних включає в себе деталі реалізації БД на фізичному рівні, такі як типи даних, розміри полів, індекси, обмеження цілісності тощо, та виглядає наступним чином [20].

Таблиця users:

– id – ціле число (integer), первинний ключ (primary key) для ідентифікації користувача;

– first_name – текстове поле (varchar) для зберігання імені користувача;

– last_name – текстове поле (varchar) для зберігання прізвища користувача;

– nickname – текстове поле (varchar) для зберігання нікнейму користувача;

– capital – текстове поле (varchar) для зберігання загального капіталу користувача;

– usd_capital – текстове поле (varchar) для зберігання капіталу користувача в доларах США;

– status – текстове поле (varchar) для зберігання статусу користувача.

Таблиця unregistered:

– id – ціле число (integer), первинний ключ (primary key) для ідентифікації незареєстрованого користувача;

– first_name – текстове поле (varchar) для зберігання імені незареєстрованого користувача;

– last_name – текстове поле (varchar) для зберігання прізвища незареєстрованого користувача;

– nickname – текстове поле (varchar) для зберігання нікнейму незареєстрованого користувача.

Таблиці crypto_usr, banks_usr та cash_usr:

– currency – текстове поле (varchar), первинний ключ (primary key) для зберігання назви валюти;

– amount – текстове поле (varchar) для зберігання кількості валюти (як у цілому так і у дробовому форматі);

– mark – текстове поле (varchar), що зберігає значок національної валюти.

У коді телеграм–боту вся взаємодія з БД відбувається через функції та команди, описані у файлі db.py. При запуску бота першим ділом відбувається ініціалізація бази даних (відповідний код представлений на рисунку 2.6).

```
import psycopg2
DB_URL = ***
base = psycopg2.connect(DB_URL, sslmode = "require")
db = base.cursor()
db.execute('CREATE TABLE IF NOT EXISTS users (id integer PRIMARY KEY, first_name varchar,
last_name varchar, nickname varchar, capital varchar, usd_capital varchar, status varchar)')
db.execute('CREATE TABLE IF NOT EXISTS unregistered (id integer PRIMARY KEY, first_name
varchar, last_name varchar, nickname varchar)')
base.commit()
```

Рисунок 2.6 – Код ініціалізації бази даних

ERD діаграма бази даних представлена на рисунку 2.7.

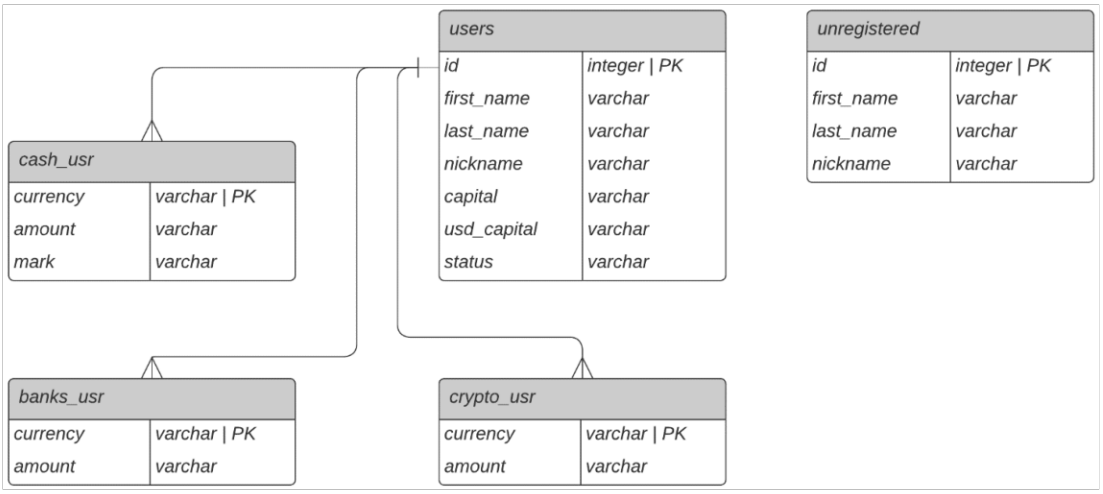

Рисунок 2.7 – ERD діаграма БД

Цей код виконує підключення до бази даних у рядку base = psycopg2.connect(DB_URL, sslmode = «require»). Цей рядок коду встановлює з'єднання з базою даних PostgreSQL за допомогою URL–адреси DB_URL, яка містить інформацію про розташування бази даних, і налаштування безпеки (sslmode). Далі створюється курсор db = base.cursor(), який використовується для взаємодії з базою даних. Після цього відбувається створення таблиць db.execute('CREATE TABLE IF NOT EXISTS users ...') та db.execute('CREATE

TABLE IF NOT EXISTS unregistered ...') – ці рядки коду створюють дві таблиці в базі даних з назвами «users» та «unregistered» відповідно. Параметр IF NOT EXISTS дозволяє створювати таблиці, тільки якщо вони ще не існують. Рядок base.commit() зберігає зміни до бази даних після виконання запитів на створення таблиць [21].

РОЗДІЛ 3
РОЗРОБКА ТЕЛЕГРАМ БОТУ ДЛЯ ОБЛІКУ ФІНАНСІВ

3.1 Створення Telegram–боту та проектування його структурної схеми

Перед розробкою бота, спершу необхідно зареєструвати його в системі ботів Telegram. Для цього необхідно знайти бота @BotFather в телеграмі, який є офіційним ботом для створення нових ботів. Можна використати пошук або зайти на профіль @BotFather за посиланням: https://t.me/BotFather. Далі потрібно запустити діалог з @BotFather, натиснувши на кнопку «Start» або написавши команду /start (рис. 3.1), та слідувати за вказівкам. Бот запропонує ряд команд для налаштування вашого бота, наприклад: /newbot для створення нового бота, /setname для встановлення імені бота, /setdescription для встановлення опису бота, тощо. Після налаштування буде отримано токен бота, який знадобиться для взаємодії з API телеграму. Саме цей токен і буде використаний у коді.

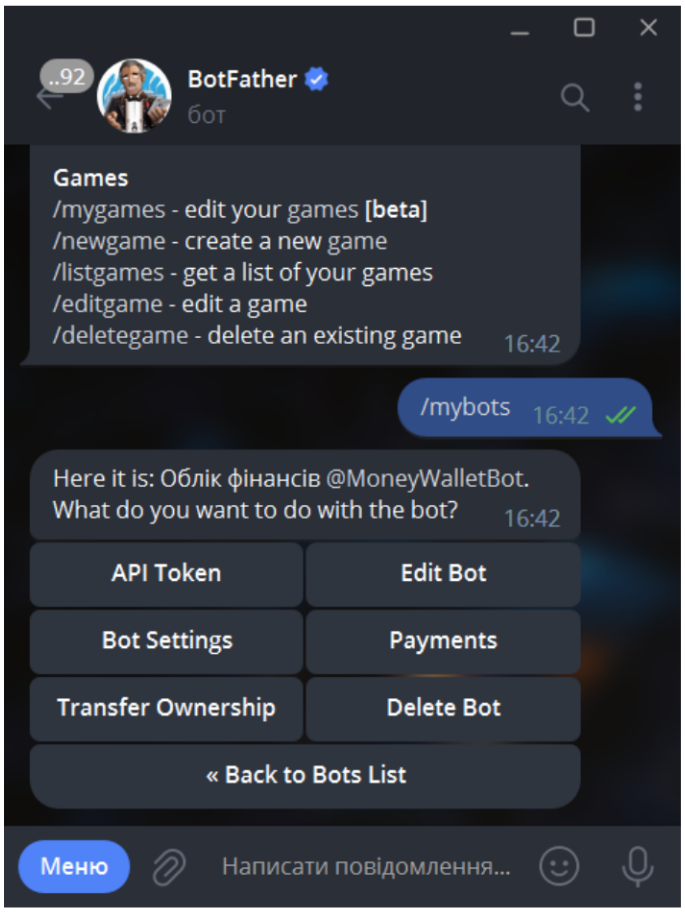

Рисунок 3.1 – Меню взаємодії з BotFather

Після отримання токена можна розпочинати розробку бота. Для початку, необхідно зрозуміти його структуру. Для збереження даних користувачів буде використана база даних, для отримання актульних курсів валют – зовнішні джерела (криптовалютна біржа Binance для отримання курсів криптовалют, та сервіс ExchangeRate–API для отримання актульних курсів фіатних валют). Таким чином, структурна схема бота буде складатися з 3 структурних елементів, що зображені на рисунку 3.2.

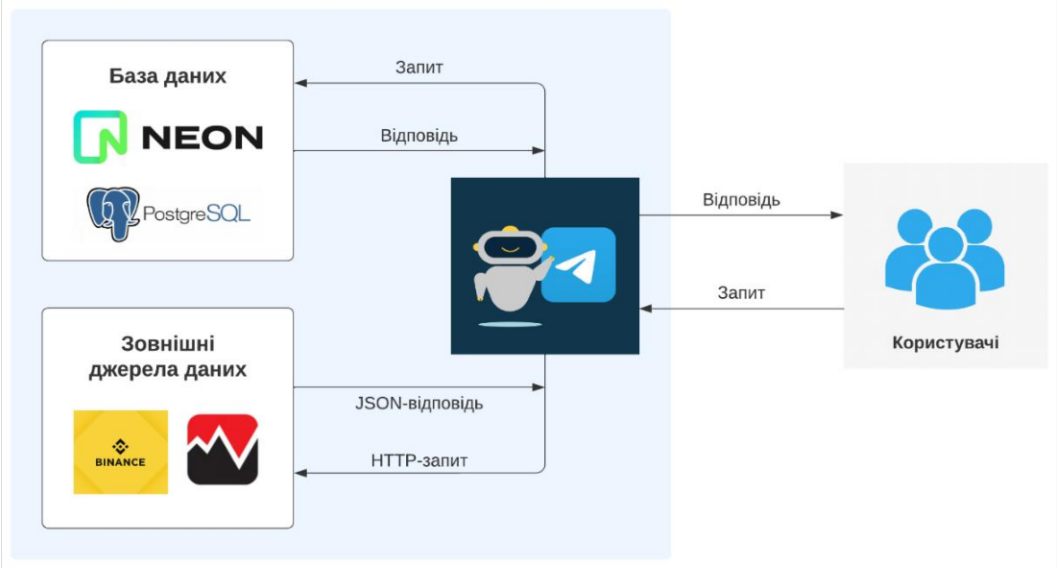

Рисунок 3.2 – Загальна структурна схема телеграм–боту

Розпочинати розробку слід зі встановлення усіх необхідних бібліотек. Можна встановлювати їх кожну по–окремості, використовуючи команду pip install «назва бібліотеки», а можна вписати назви усіх необхідних бібліотек у файл requirements.txt (рис. 3.3) та запустити команду pip install –r /path/to/requirements.txt.

```
1  aiogram
2  json
3  fake_useragent
4  requests
5  beautifulsoup4
6  psycopg2
```

Рисунок 3.3 – Вміст файлу requirements.txt

pip – це популярний пакетний менеджер для мови програмування Python, який використовується для встановлення, оновлення та керування сторонніми бібліотеками (пакетами) Python. Назва pip походить від скорочення «Pip Installs Packages» (пакети встановлює Pip).

pip дозволяє легко встановлювати сторонні бібліотеки Python з репозиторію PyPI (Python Package Index), який є офіційним репозиторієм пакетів Python. За допомогою pip можна встановлювати, оновлювати та видаляти багато різних бібліотек Python, що дозволяє розширити можливості мови та використовувати готові рішення, створені іншими розробниками.

Весь основних код бота буде описаний у файлі bot.py (додаток А). Початок розробки необхідно розпочати з ініціалізації бота (рис. 3.4) за допомогою попередньо отриманого токену.

```
# Configure logging
logging.basicConfig(level=logging.INFO)

# Initialize bot and dispatcher
bot = Bot(token = API_TOKEN)
server = Flask(__name__)
storage = MemoryStorage()
dp = Dispatcher(bot, storage=MemoryStorage())
```

Рисунок 3.4 – Ініціалізація бота

Принцип взаємодії бота та користувача полягає у тому, що в боті працюють обробники подій, створенні за допомогою @dp.message_handler. Цей декоратор використовується для реєстрації функції, яка буде виконуватися, коли бот отримує повідомлення від користувача. Повідомлення може бути текстовим, фотографією, відео або іншим типом повідомлення, які може надіслати користувач [22].

Цей декоратор має кілька параметрів, таких як regexp, commands, content_types та інші, які дозволяють налаштувати фільтри для відповідних типів повідомлень. Коли бот отримує повідомлення, яке відповідає цим фільтрам,

функція–обробник, прикріплена до @dp.message_handler, буде автоматично виконана з отриманим повідомленням в якості аргументу.

Приклад використання декоратора наведено на рисунку 3.5.

```
@dp.message_handler(lambda message:
message.text.startswith(' Команди'))
@dp.message_handler(commands=['start'])
async def commands(message: types.Message):
    if registered(message.from_user.id) == 1:
        commands_msg = mycommands.main()
        await bot.send_message(message.from_user.id,
commands_msg, reply_markup = menu.mainMenu)
    elif registered(message.from_user.id) == 0:
        db.ask_registration(message.from_user.id,
message.from_user.first_name, message.from_user.last_name,
message.from_user.username)
        await bot.send_message(message.from_user.id,
not_allowed_msg)
```

Рисунок 3.5 – Обробник повідомлення

Цей код використовує декоратор @dp.message_handler з різними параметрами для реєстрації двох обробників повідомлень.

Перший обробляє повідомлення, які починаються з тексту « Команди». Це досягається за допомогою лямбда–функції lambda message: message.text.startswith(' Команди'), яка є фільтром для текстових повідомлень, що починаються з даного тексту. Якщо введений текст повідомлення починається з « Команди», то виконується функція.

Другий обробник обробляє повідомлення, що містять команду /start. Це досягається за допомогою параметра commands=['start'], який вказує на обробку команди /start. Якщо користувач відправляє команду /start, то виконується функція (рис. 3.6).

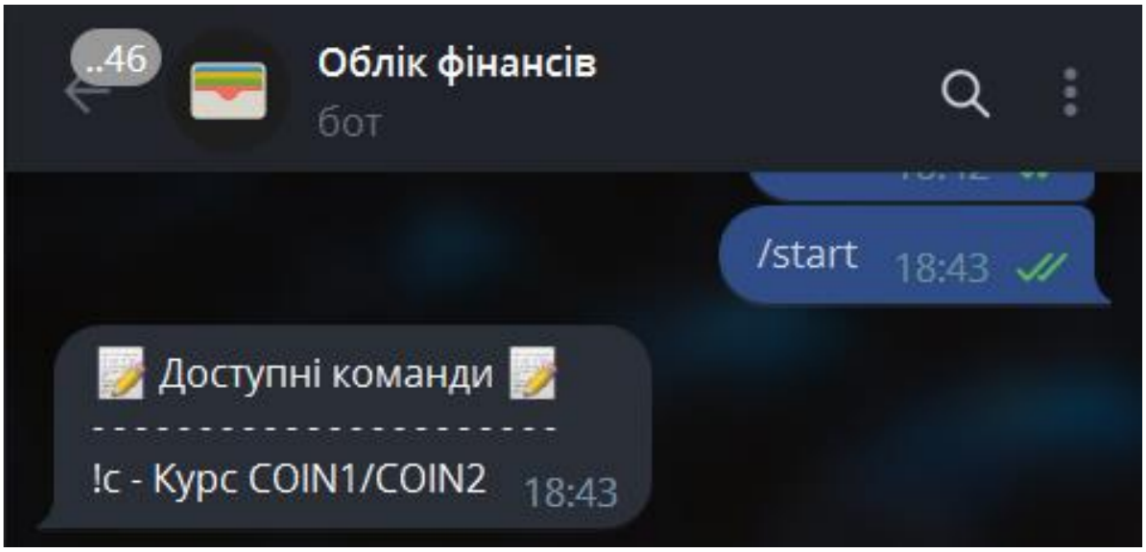

Рисунок 3.6 – Обробка команди /start

Обидва обробники в результаті призводять до виконання тої ж функції, яка відправляє повідомлення з командами користувачеві, якщо він зареєстрований, або відправляє повідомлення про незареєстрованого користувача і викликає функцію реєстрації нового користувача, якщо він не зареєстрований.

Повністю вся логіка взаємодії бота з користувачем побудована на цьому принципі обробки повідомлень. Для відправки повідомлення зі сторони бота, використовується команда await bot.send_message(message.from_user.id, commands_msg, reply_markup = menu.mainMenu) з трьома параметрами. Перший параметр відповідає за адресата, якому буде надіслано повідомлення (використовується унікальний ID користувача, від якого бот отримав повідомлення), другий параметр відповідає за вміст повідомлення, яке надсилає бот, а третій параметр, за розмітку клавіатури, яка буде показана користувачу при відправці ботом повідомлення.

Розмітка клавіатури в aiorgram створюється за допомогою об’єктів ReplyKeyboardMarkup, які в свою чергу містять об’єкти KeyboardButton. У даному боті ці об’єкти прописані у файлі navigation.py, що імпортується у файл bot.py командою «import Tools.navigation as menu». Приклад створення головного меню бота представлено на рисунку 3.7, а результат – на рисунку 3.8.

```
btnCurrency = KeyboardButton('📊 Курси валют')
btnWallet = KeyboardButton('💰 Мій гаманець')
btnPortfolio = KeyboardButton('📑 Портфоліо')
btnConverter = KeyboardButton('↔️ Конвертер валют')
btnFuel= KeyboardButton('⛽ АЗС Ціни')
btnOther = KeyboardButton('⚙️ Інше')
mainMenu = ReplyKeyboardMarkup(resize_keyboard = True).row(btnCurrency,
btnPortfolio).add(btnWallet).row(btnFuel, btnConverter).add(btnOther)
```

Рисунок 3.7 – Код, що створює розмітку головного меню

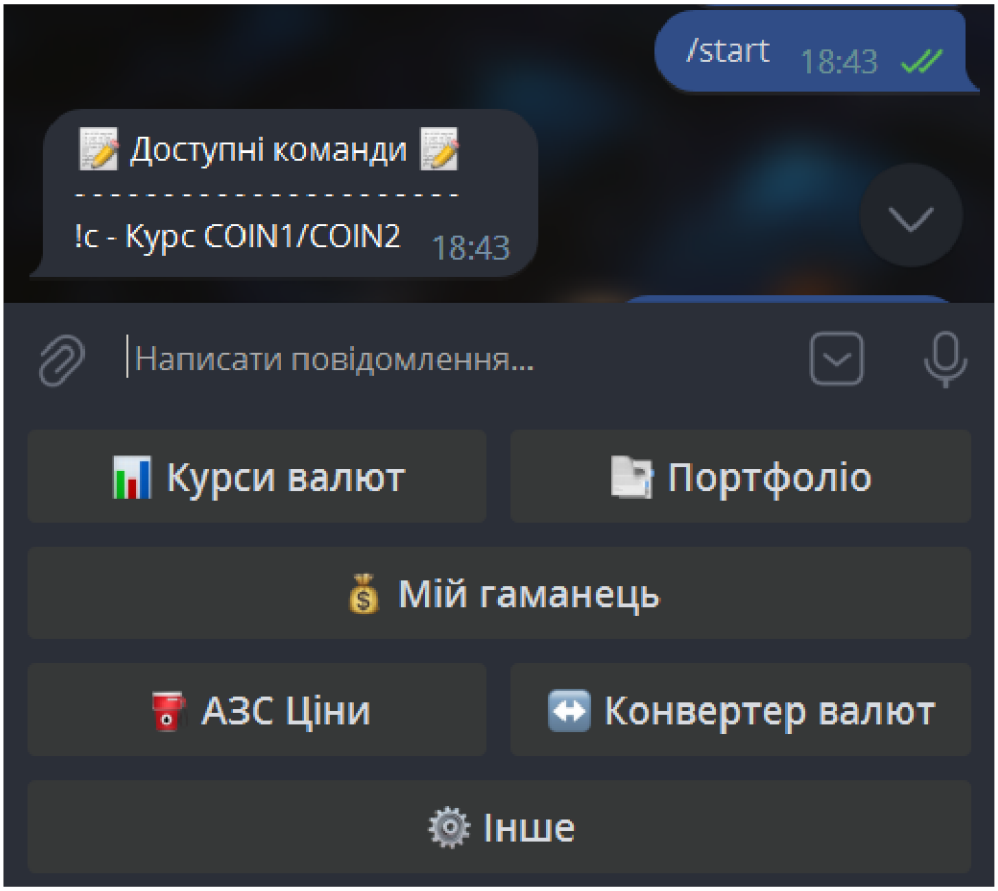

Рисунок 3.8 – Головне меню бота

Таким чином був розроблений весь наступний функціонал бота. При надсиланні команди, обробник повідомлень відловлював команду та викликав необхідну функцію. Додатково був додатний функціонал парсингу цін на пальне (рис. 3.9) та можливість швидкого отримання курсу необхідної валютної пари шляхом введення команди «!c валюта1/валюта2» (рис. 3.10).

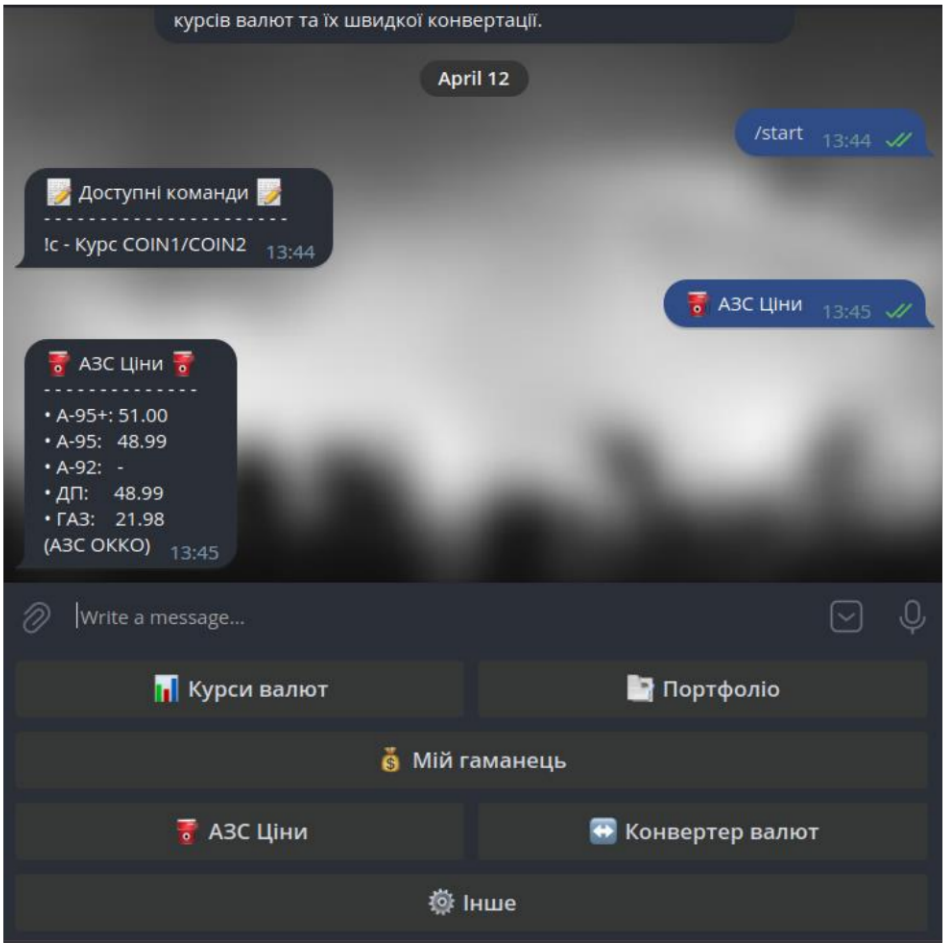

Рисунок 3.9 – Парсинг цін на пальне

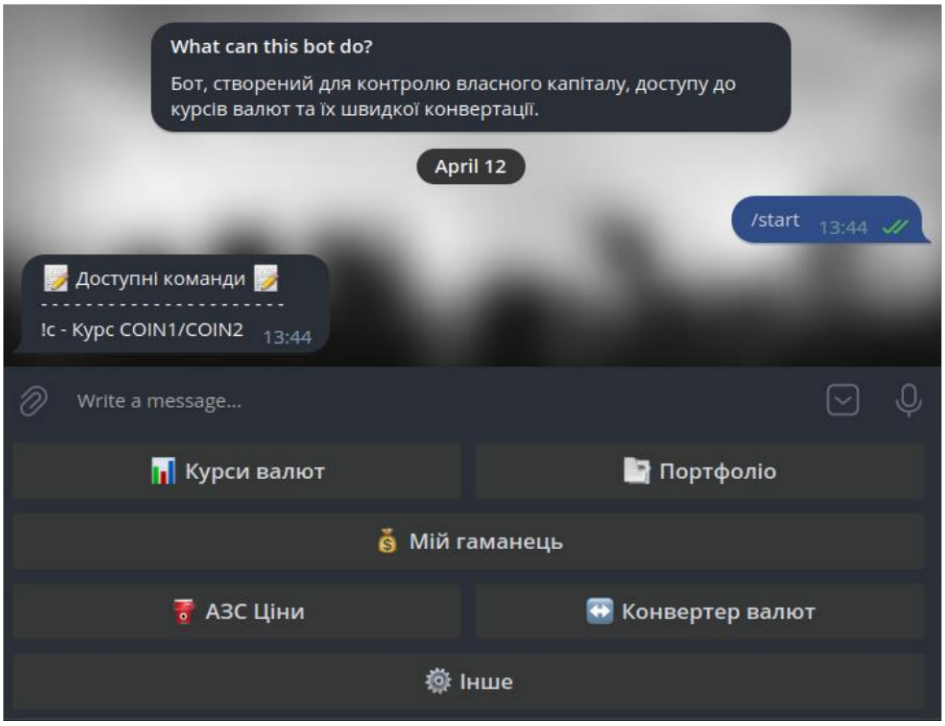

Рисунок 3.10 – Команда для отримання курсу валют

Для редагування портфоліо, необхідно перейти в розділ «Інше» та викликати команду «Змінити портфоліо», та слідувати вказівкам бота (рис. 3.11, рис. 3.12). Варто звернути увагу, що кожен раз бот змінює розмітку клавіатури на необхідну.

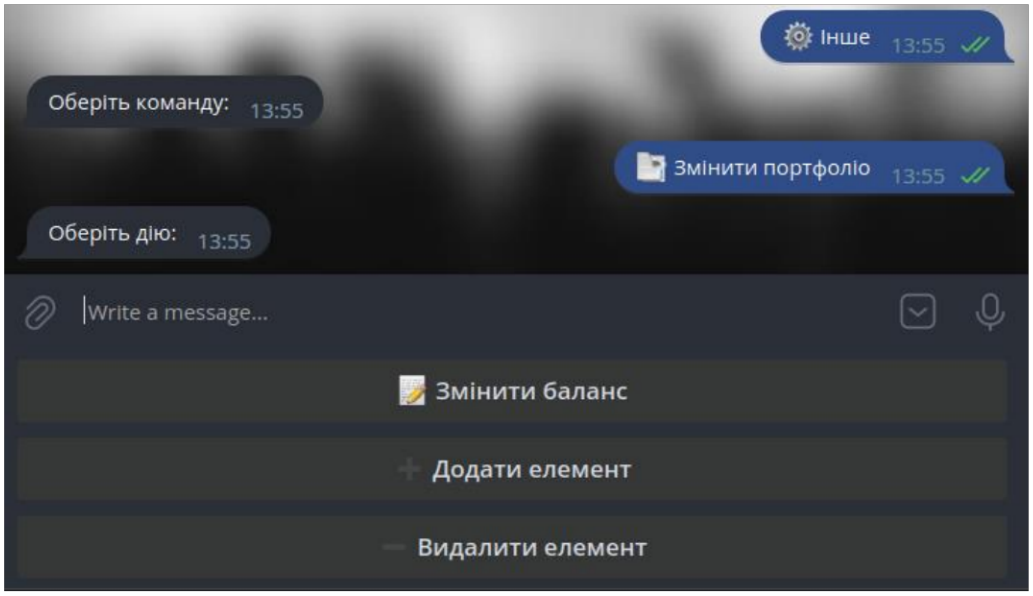

Рисунок 3.11 – Меню зміни портфоліо

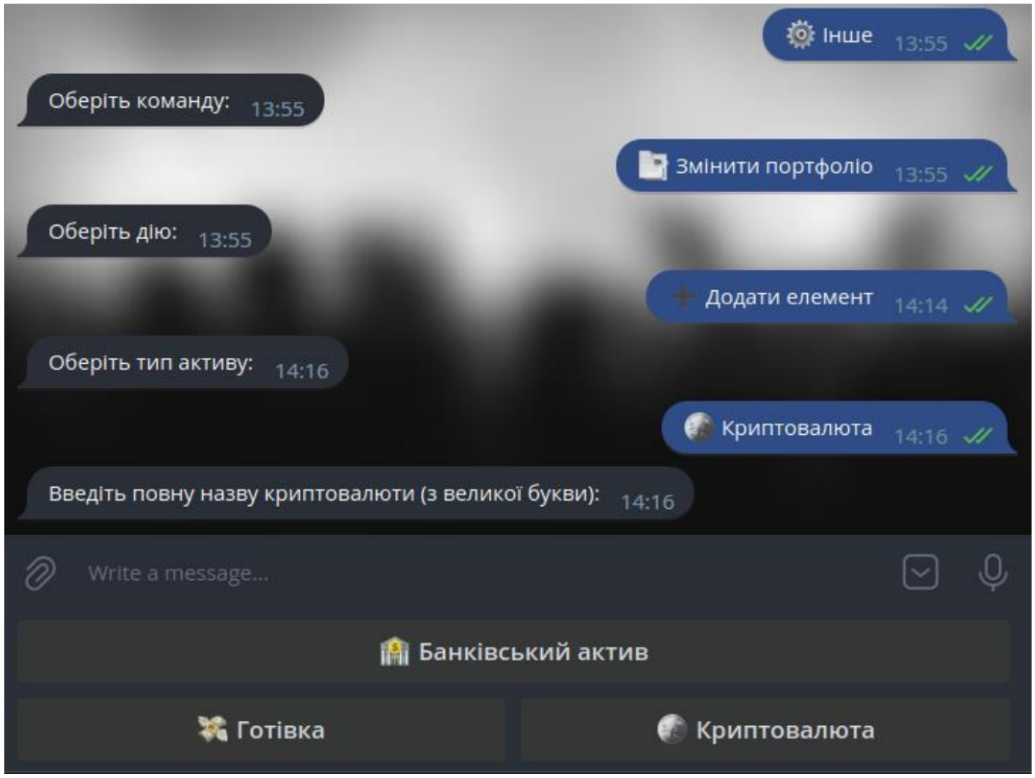

Рисунок 3.12 – Заповнення даних про актив

Актуальні курси доданих в портфоліо валют бот відображає при виклику команди «Курси валют» (рис 3.13). Після виклику команди, виконується функція main(id) (рис. 3.14) файлу currencies.py, яка отримує з бази даних список усіх криптовалют користувача, та для кожного елементу списку отримує дані про актуальний курс.

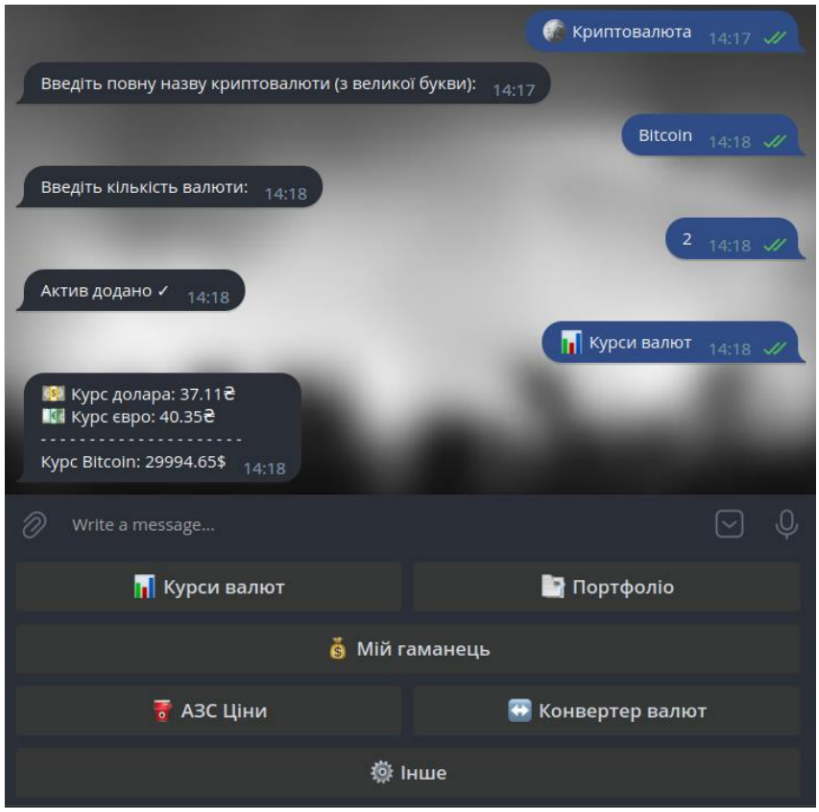

Рисунок 3.13 – Відображення актульних курсів валют

```
def main(id):
    currencies = ''
    for string in strings:
        currencies += string
    lst = []
    try:
        db.db.execute('SELECT currency FROM crypto_' + str(id))
        lst = db.db.fetchmany()
    except BaseException as e:
        print(e)
    for item in lst:
        currencies += '\nКурс ' + item[0] + ': %.2f$' %
binance(converter.get_short(item[0]), "USDT")
    return currencies
```

Рисунок 3.14 – Отримання даних про крипто–активи користувача

За аналогічним принципом при виклику команди «Портфоліо» або «Мій гаманець» бот отримує з бази даних дані про абсолютно усі активи користувача, отримує для кожного актуальний курс, конвертує його кількість, та виводить користувачеві результат.

3.2 Механізми захисту даних користувача

Враховуючи, що вся взаємодія відбувається всередині месенджеру Telegram, то варто розглянути саме його механізми захисту. Telegram – це месенджер, який надає можливість забезпечити високий рівень захисту для користувачів, телеграм–ботів та їх даних. Основні принципи захисту в Телеграмі включають:

– шифрування даних: Телеграм використовує шифрування на кінцевих точках (end–to–end encryption) для особистих чатів, що означає, що тільки відправник та одержувач можуть переглядати вміст повідомлення, а навіть сама компанія не має доступу до цих даних;

– захист від перехоплення: Телеграм захищає дані користувачів від перехоплення шляхом використання TLS–шифрування для передачі даних між клієнтами та серверами;

– двофакторна автентифікація: Телеграм підтримує можливість встановлення двофакторної автентифікації, що дозволяє користувачам захистити свої акаунти від несанкціонованого доступу;

– аудит безпеки: Телеграм регулярно проводить аудити безпеки для виявлення можливих проблем.

Також, варто зазначити, що при ідентифікації користувача, бот використовує внутрішній ID (а не логін) користувача у базі даних Telegram. Цей ідентифікатор абсолютно унікальний та незмінний для кожного профілю, а тому виключена будь–яка можливість отримання одним користувачем даних іншого користувача.

3.3 Методи тестування Телеграм боту

При розробці телеграм–ботів, тестування є важливою складовою процесу розробки, щоб переконатися в якості та надійності боту перед його впровадженням. Основні принципи тестування телеграм–ботів включають:

– функціональне тестування. Під час даного процесу необхідно переконатися, що всі функції боту виконуються правильно відповідно до вимог і специфікацій. Це може включати тестування різних сценаріїв взаємодії з ботом, перевірку відповідності результатів очікуваним, тестування різних варіацій введення від користувачів, та інше:

– навантажувальне тестування. Це передбачає виконання тестів з великою кількістю запитів або великим обсягом даних, щоб переконатися, що бот може витримати навантаження та працювати стабільно в реальному використанні;

– тестування взаємодії з API Телеграму. Це передбачає перевірку взаємодії боту з API Телеграму, таку як відправка та отримання повідомлень, керування чатами, робота з клавіатурами, статистика відправлення повідомлень, та інші функції;

– безпека та захист даних. Під час цього тестування необхідно переконатися, що бот відповідає вимогам безпеки, таким як захист від несанкціонованого доступу, використання шифрування, обробка чутливих даних користувачів відповідно до вимог законодавства та політики конфіденційності.

Системними вимогами для роботи системи буде будь який пристрій, що має можливість запускати Telegram онлайн або нативно, а також стабільне Інтернет–з’єднання.

Для хостингу бота підійде практично будь–яка платформа, так як телеграм–боти не вимагають високого рівня продуктивності системи. Єдине зауваження, в перспективі, зі зростанням кількості користувачів присутня вірогідність того, що все ж таки буде необхідно провести вдосконалення хостинг–платформи. Рівень необхідної продуктивності системи для

забезпечення стабільної роботи буде прямо дорівнювати рівню зростання кількості користувачів.

3.4 Інструкція користувача з використання Телеграм боту

Для того, щоб почати використовувати телеграм–бота, користувачу необхідно знайти його через пошук за назвою @MoneyWalletBot (рис. 3.15) або за посиланням https://t.me/MoneyWalletBot. Після запуску, користувач одразу побачить перед собою головне меню бота (рис. 3.16).

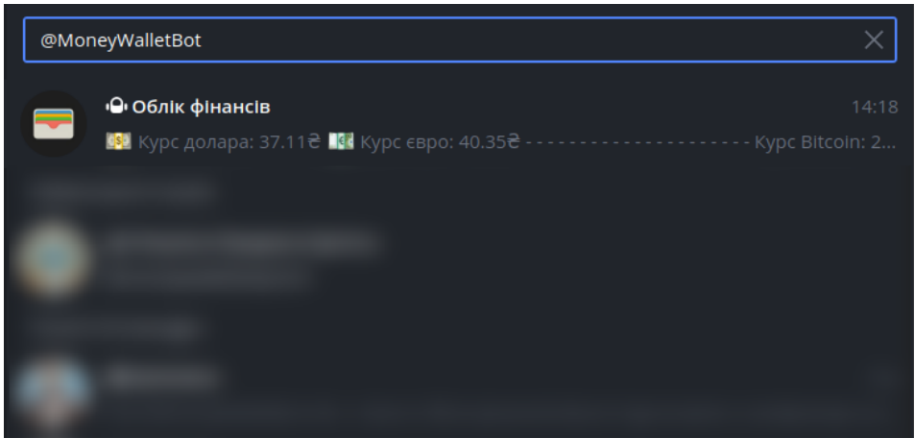

Рисунок 3.15 – Знаходження бота через пошук

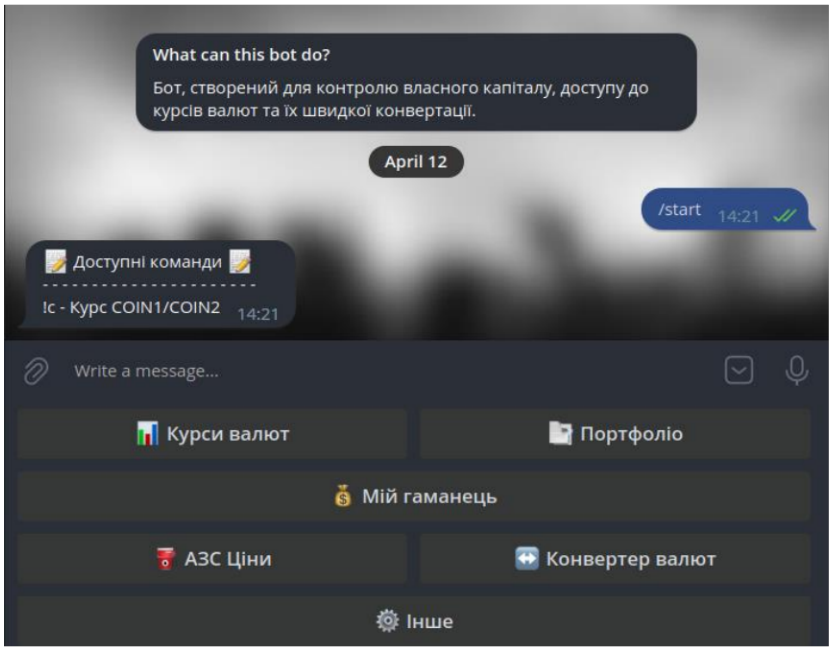

Рисунок 3.16 – Головне меню бота

Після цього необхідно додати активи у розділ «Портфоліо». Для цього необхідно натиснути кнопку «Інше» (рис. 3.17), далі «Змінити портфоліо» (рис. 3.18), «Додати елемент» (рис. 3.19), обрати тип активу, ввести його назву, баланс та валюту (рис. 3.20).

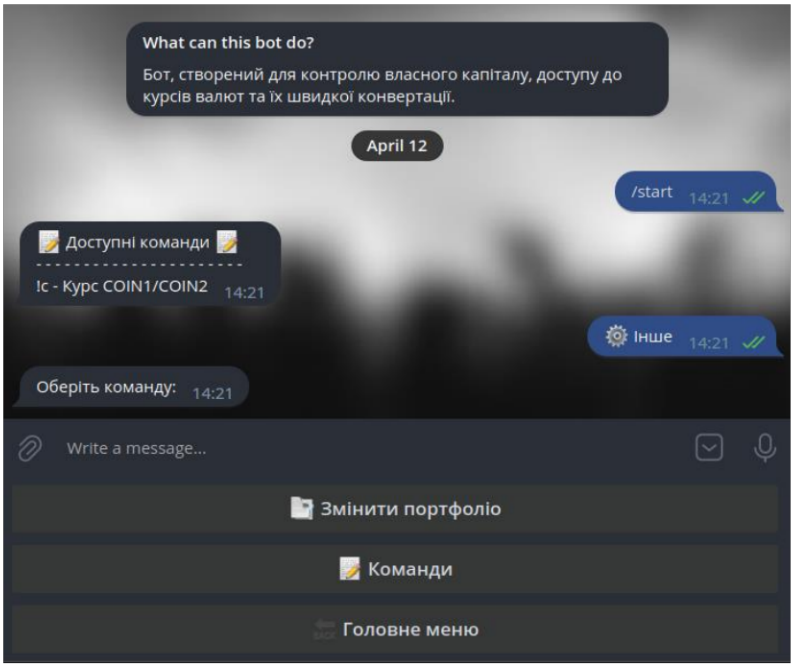

Рисунок 3.17 – Меню «Інше»

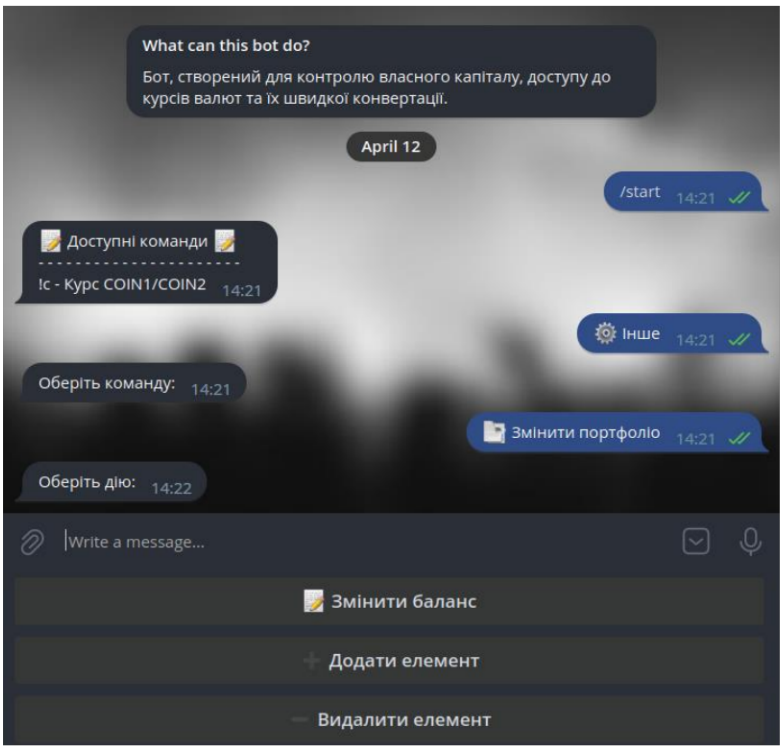

Рисунок 3.18 – Редактор портфоліо

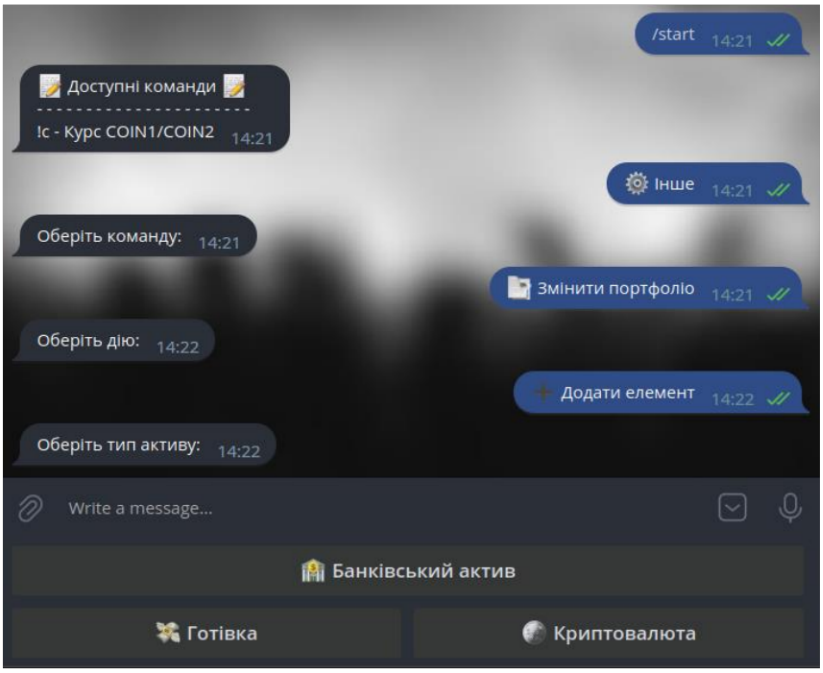

Рисунок 3.19 – Додавання елементу

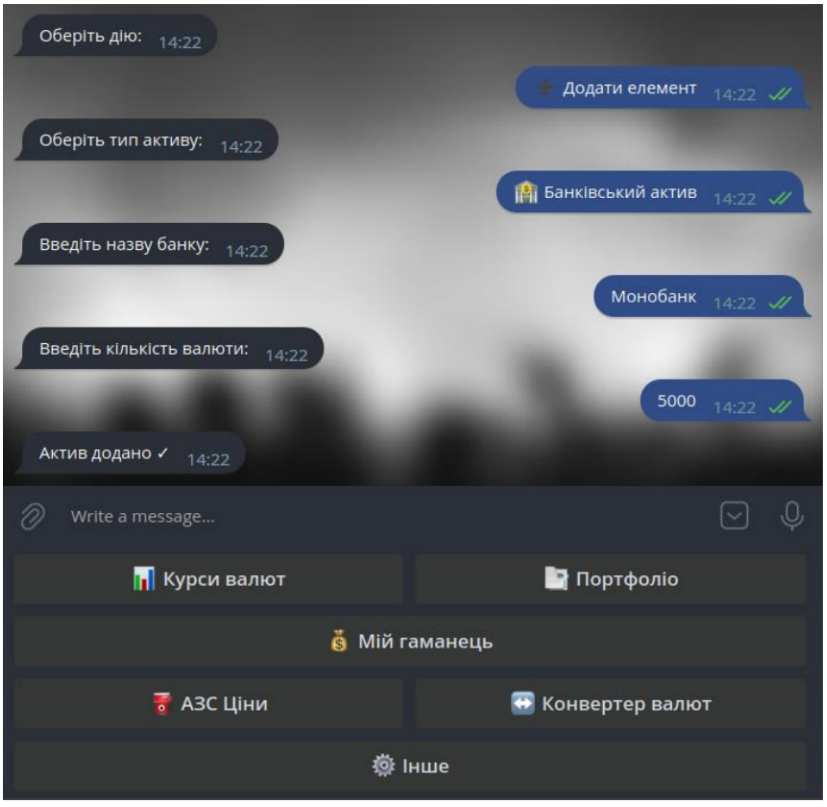

Рисунок 3.20 – Доданий актив

Після цього можна звернути увагу, що відповідний актив з’явився у розділі «Портфоліо» (рис. 3.21), його баланс додався до загального у розділі «Гаманець» (рис. 3.22), а відповідна валюта з’явилась у меню «Курси валют» (рис. 3.23).

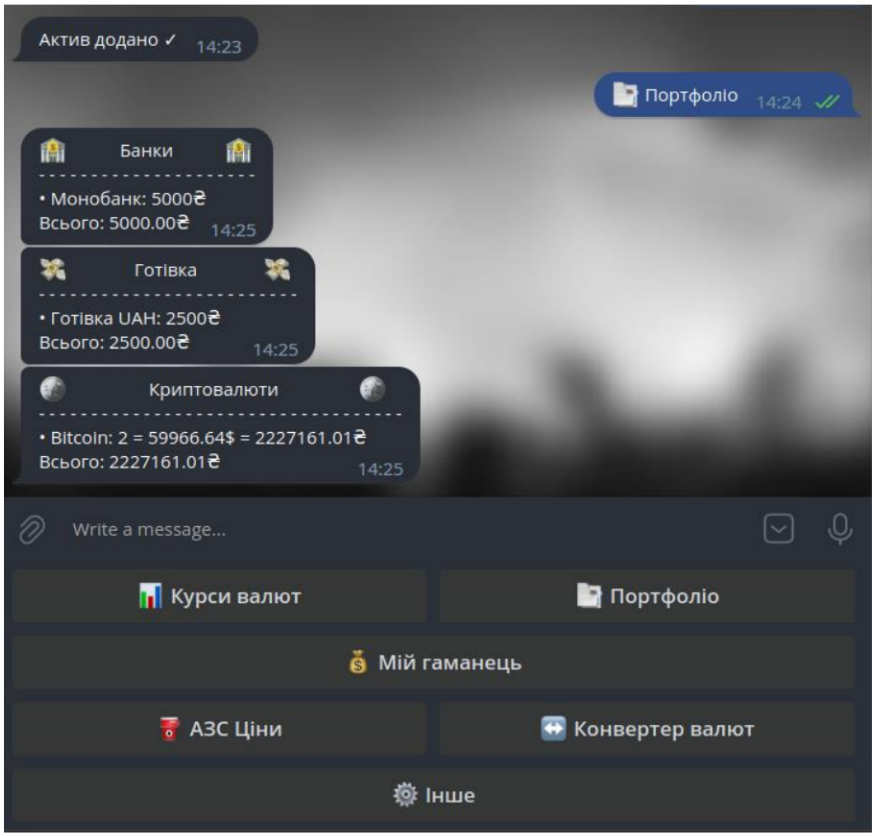

Рисунок 3.21 – Портфоліо користувача

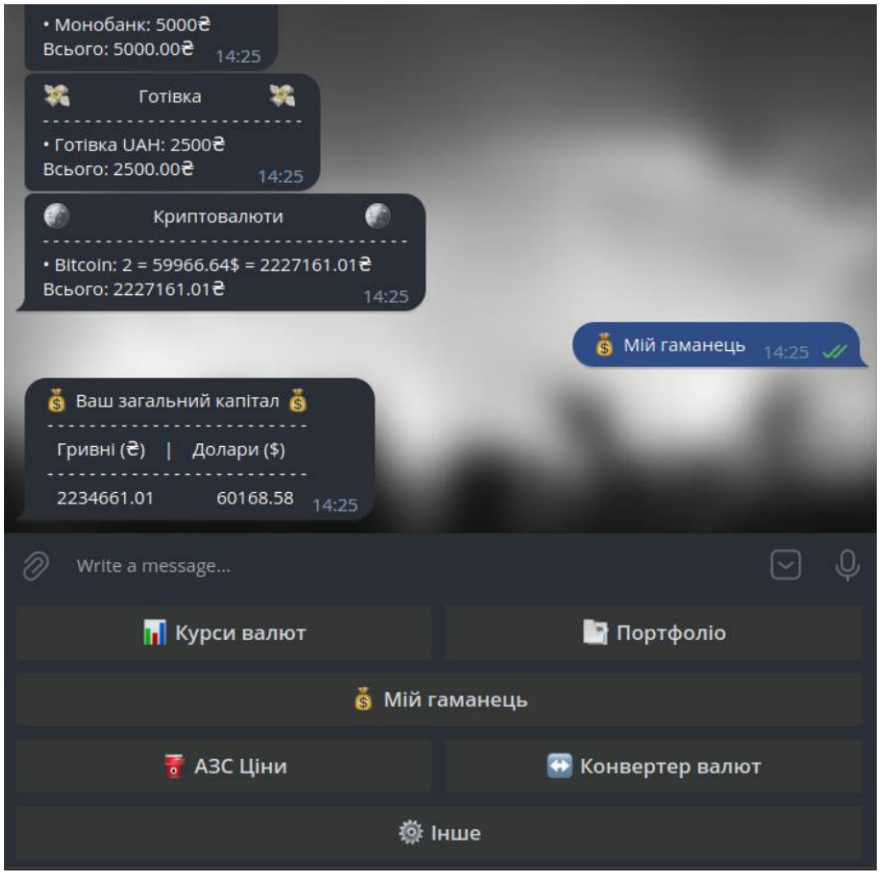

Рисунок 3.22 – Гаманець користувача

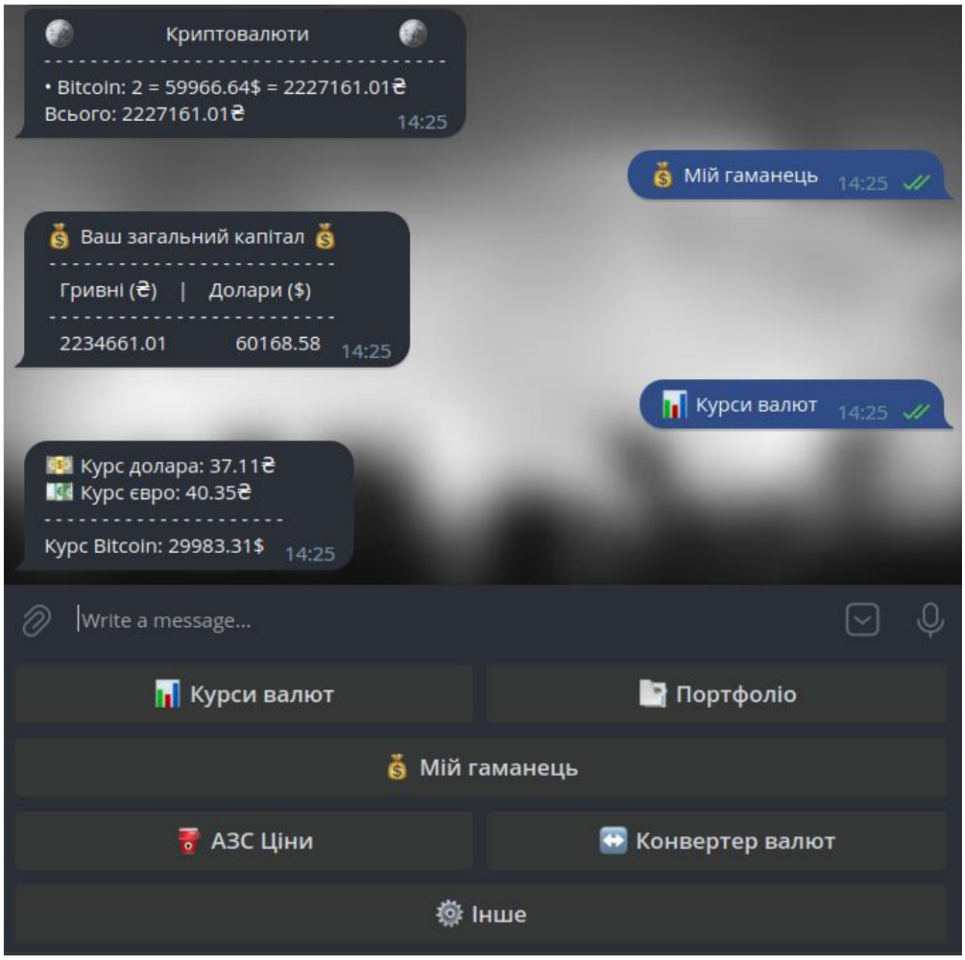

Рисунок 3.23 – Поява нової валюти у меню «Курси валют»

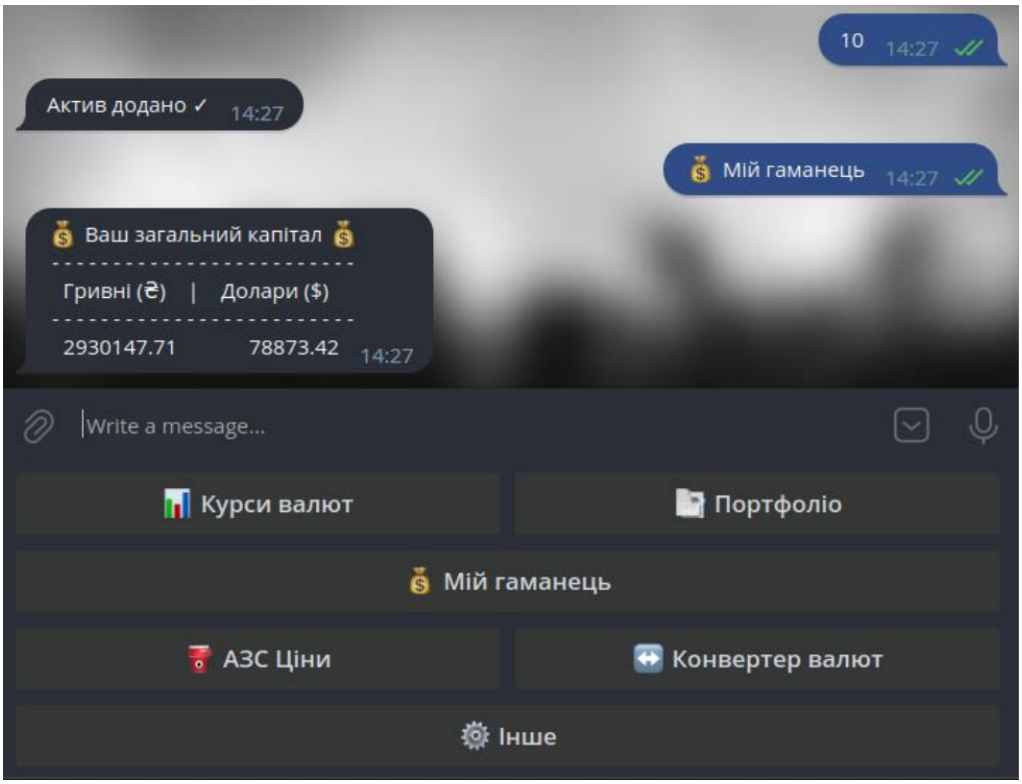

Рисунок 3.20 – Збільшення балансу у гаманці

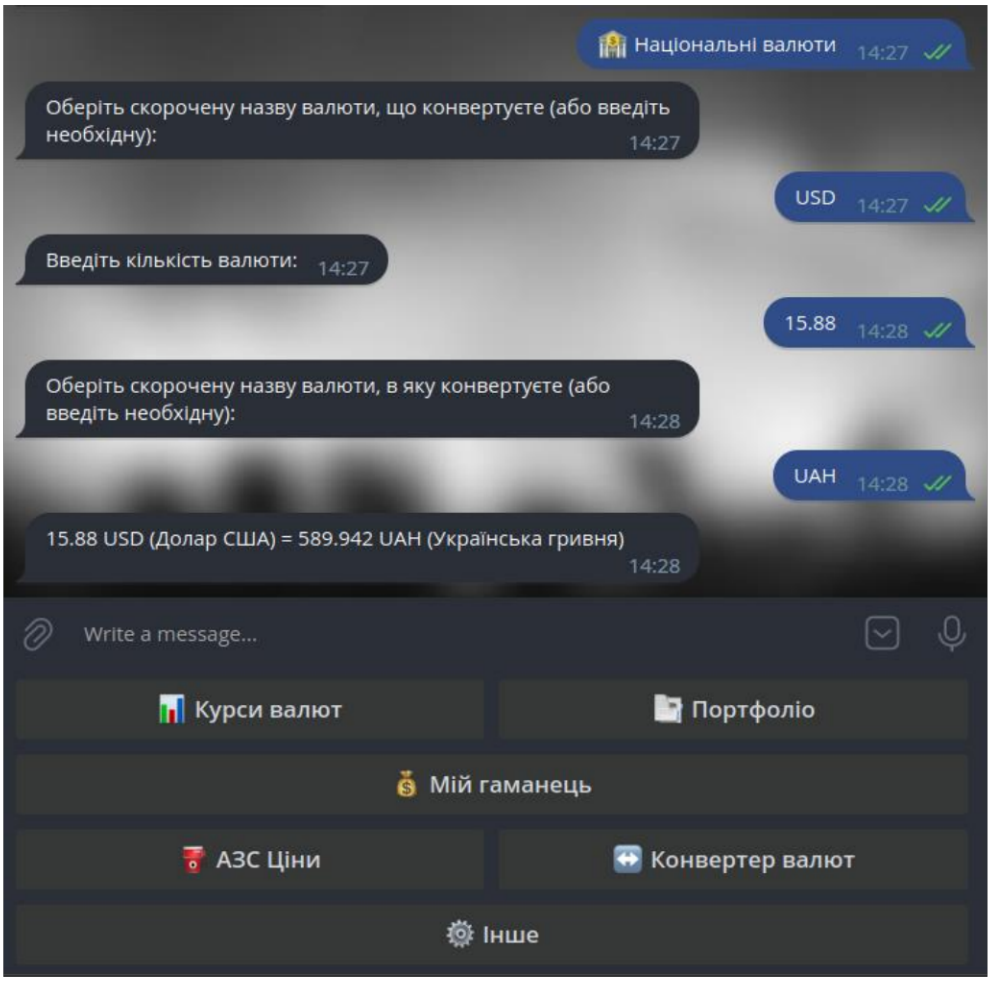

Рисунок 3.21 – Конвертер валют

З кожним новим доданим активом, загальний баланс у меню «Мій гаманець» відповідно збільшується. При необхідності є можливість скористатися конвертером валют, отриманням актуального курсу необхідної пари, та іншими додатковими функціями.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи провів детальне дослідження та розробку Телеграм бота для обліку фінансів з використанням бібліотеки Aiogram та мови програмування Python.

В ході дослідження проаналізував основні поняття та сфери застосування Телеграм ботів у контексті обліку фінансів та обґрунтував вибір технологій для розробки Телеграм бота, включаючи використання бібліотеки Aiogram та бази даних PostgreSQL. Порівняльний аналіз аналогічних програм та інструментів дозволив виявити їх переваги та недоліки.

У процесі реалізації розробив функціонально–структурну схему роботи Телеграм бота, що дозволяє ефективно взаємодіяти з користувачем та забезпечувати зручний облік фінансів, створив модель бази даних PostgreSQL для зберігання необхідної інформації про фінансові транзакції та мультивалютне грошове портфоліо та розробив безпечні механізми захисту даних користувача, що гарантують конфіденційність та надійність використання Телеграм бота.

Після реалізації провів тестування розробленого об’єкту для перевірки його функціональності та надійності роботи та розробив інструкцію з використання, яка дозволить користувачам швидко ознайомитися з основними функціями та можливостями бота.

У висновку, розроблений Телеграм бот виявився справді зручним та ефективним інструментом для обліку фінансів, а саме відстеження стану мультивалютного портфоліо, в якому доступні такі функції як конвертер валют, моніторинг курсів валют, моніторинг балансу власних активів, та ряд додаткових функцій.

Телеграм-боти для обліку фінансів зручні та легкі у використанні, можуть автоматизувати фінансові операції. Однак, вони мають обмежену функціональність порівняно з повноцінними фінансовими програмами, а також можуть створювати ризик безпеки та не мати достатньої підтримки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Товстенюк В., Лавренчук С. В. Інформаційний чатбот для месенджеру Telegram у галузі освіти. Комп'ютерно–інтегровані технології: освіта, наука, виробництво. (41). 2020. С. 180–185.

2. Telegram Bot API. Telegram. URL: https://core.telegram.org/bots/api (дата звернення: 12.12.2022).

3. Білик, І. Ю. Телеграм–бот для роботи з курсами валют та відслідковування витрат : випускна кваліфікаційна робота : 123 "Комп’ютерна інженерія". Чернігів, 2020. 70 с.

4. Боти: вступ для розробників. Telegram. URL: https://core.telegram.org/bots (дата звернення: 25.12.2022).

5. Кравчук О. Розробка телеграм ботів на Python. Тернопіль: ТНТУ, 2022. 1 с.

6. Devecosystem databases 2022. JetBrains. URL: https://www.jetbrains.com/lp/devecosystem–2022/databases/ (дата звернення: 05.01.2023).

7. Смальченко Н.Г. Оптимізація роботи СУБД PostgreSQL: від рівня сервера до рівня. Київ: Національний Університет «Києво–Могилянська академія», 2021. 35 с.

8. Як використовувати Django, PostgreSQL та Docker. Ithillel Blog. URL: https://blog.ithillel.ua/articles/how–to–use–django–postgresql–and–docker (дата звернення: 09.01.2023).

9. PostreSQL Documentation. PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 15.01.2023).

10. Головачов І. А. Реляційні та нереляційні бази даних в бізнесі. Наукові розробки молоді на сучасному етапі : тези доповідей XVII Всеукраїнської наукової конференції молодих вчених та студентів (26–27 квітня 2018 р., Київ). К. : КНУТД, 2018. Т. 3 : Економіка інноваційної діяльності підприємств. С. 319–320.

11. Гулковський М. М., Бурак Н. Є. Сучасні системи управління базами даних. 2020. URL: http://sci.ldubgd.edu.ua:8080/jspui/handle/123456789/7446 (дата звернення: 28.01.2023).

12. How to Use PostgreSQL in Python. FreeCodeCamp. URL: https://www.freecodecamp.org/news/postgresql–in–python/ (дата звернення: 02.02.2023).

13. Зінов'єва, І.С., Артемчук, В.О. Сучасні підходи до подальшої еволюції концепції баз даних // Dynamics of the development of world science. Abstracts of the 3d International scientific and practical conference. Perfect Publishing. Vancouver, Canada, 2019. 9 с.

14. SQL Tutorial. W3Schools. URL: https://www.w3schools.com/sql/ (дата звернення: 05.02.2023).

15. SQL. Вікіпедія. URL: https://uk.wikipedia.org/wiki/SQL (дата звернення: 09.02.2023).

16. DBeaver. DBeaver Community. URL: https://dbeaver.io/ (дата звернення: 13.02.2023).

17. DBeaver. Вікіпедія. URL: https://uk.wikipedia.org/wiki/DBeaver (дата звернення: 17.02.2023).

18. Гангала О. М. Метод блокування незареєстрованого SQL–трафіку на бази даних PostgreSQL та MySQL: кваліфікаційна робота бакалавра за спеціальністю 125 — Кібербезпека / О. М. Гангала. – Тернопіль : ТНТУ, 2022. – 52 с.

19. Федоров О. В. Веб–застосування для побудови запитів до бази даних PostgreSQL із використанням графічного інтерфейсу : кваліфікаційна робота освітнього рівня «Бакалавр». Київ: КПІ, 2022. 141 с.

20. Бази даних. PostgreSQL. CyberBionic. URL: https://edu.cbsystematics.com/ua/courses/postgresql (дата звернення: 27.02.2023).

21. Найшвидший спосіб завантажити дані в PostgreSQL за допомогою Python. Codeguida. URL: https://codeguida.com/post/1996 (дата звернення: 02.03.2023).

22. Документація Telegram Bot. Python Telegram Bot. URL: https://docs.python–telegram–bot.org/en/stable/ (дата звернення: 07.03.2023).

ДОДАТКИ

Додаток А

Лістинг коду файлу bot.py

```
from http import server
import logging, time
from aiogram import Bot, Dispatcher, executor, types
from aiogram.dispatcher import FSMContext
from aiogram.contrib.fsm_storage.memory import MemoryStorage
import Tools.navigation as menu
from Tools import scrapper, converter, states, calendar_data
from Menu import currencies, mycommands, wallet, fuel, prt, weather, spammer
import db, os
from aiogram.utils.executor import start_webhook
from flask import Flask, request

# Configure logging
logging.basicConfig(level=logging.INFO)

# Initialize bot and dispatcher
bot = Bot(token = API_TOKEN)
server = Flask(__name__)
storage = MemoryStorage()
dp = Dispatcher(bot, storage=MemoryStorage())

error_msg = "На жаль, сталася помилка 😒"
not_allowed_msg = "На жаль, у вас відсутній доступ для того, щоб користуватися
мною 😒"

def registered(id):
    reg = db.is_registered(id)
    if reg == 1: return 1
    if reg == 0: return 0

def admin(id):
    reg = db.is_admin(id)
    if reg == 1: return 1
    if reg == 0: return 0

#---Команди
@dp.message_handler(lambda message: message.text.startswith('📝 Команди'))
@dp.message_handler(commands=['start'])
async def commands(message: types.Message):
    if registered(message.from_user.id) == 1:
        commands_msg = mycommands.main()
        await bot.send_message(message.from_user.id, commands_msg, reply_markup
= menu.mainMenu)
    elif registered(message.from_user.id) == 0:
        db.ask_registration(message.from_user.id, message.from_user.first_name,
message.from_user.last_name, message.from_user.username)
        await bot.send_message(message.from_user.id, not_allowed_msg)


#---Курс валюти(команда)
@dp.message_handler(lambda message: message.text.startswith('!c '))
async def currency(message: types.Message):
    if registered(message.from_user.id) == 1:
        string = message.text[3:].upper()
        try:
            answer, tkn = scrapper.scrap(string)
            await bot.send_message(message.from_user.id, "Курс " + tkn[0] +
tkn[1] + tkn[2] + ": %.2f" % float(answer['price']))
```

```
        except:
            await bot.send_message(message.from_user.id, error_msg)


#---Курси валют
@dp.message_handler(lambda message: message.text.startswith('📊 Курси валют'))
async def currency_stats(message: types.Message):
    if registered(message.from_user.id) == 1:
        currencies_msg = currencies.main(message.from_user.id)
        await bot.send_message(message.from_user.id, currencies_msg)


#---Гаманець
@dp.message_handler(lambda message: message.text.startswith('💰 Мій гаманець'))
async def mywallet(message: types.Message):
    if registered(message.from_user.id) == 1:
        wallet_msg = wallet.main(message.from_user.id)
        await bot.send_message(message.from_user.id, wallet_msg, reply_markup =
menu.mainMenu)


#--Портфоліо
@dp.message_handler(lambda message: message.text.startswith('💼 Портфоліо'))
async def myportfolio(message: types.Message):
    if registered(message.from_user.id) == 1:
        try:
            prt1, prt2, prt3 = prt.main(message.from_user.id)
            await bot.send_message(message.from_user.id, prt1[0], reply_markup =
menu.mainMenu)
        except Exception as e:
            msg = "Помилка: " + str(e)
            await bot.send_message(message.from_user.id, msg, reply_markup =
menu.mainMenu)
        try:
            prt1, prt2, prt3 = prt.main(message.from_user.id)
            await bot.send_message(message.from_user.id, prt2[0], reply_markup =
menu.mainMenu)
        except Exception as e:
            msg = "Помилка: " + str(e)
            await bot.send_message(message.from_user.id, msg, reply_markup =
menu.mainMenu)
        try:
            prt1, prt2, prt3 = prt.main(message.from_user.id)
            await bot.send_message(message.from_user.id, prt3[0], reply_markup =
menu.mainMenu)
        except Exception as e:
            msg = "Помилка: " + str(e)
            await bot.send_message(message.from_user.id, msg, reply_markup =
menu.mainMenu)


#---Ціни на паливо
@dp.message_handler(lambda message: message.text.startswith('⛽ АЗС Ціни'))
async def weather_stats(message: types.Message):
    if registered(message.from_user.id) == 1:
        fuel_msg = fuel.main()
        await bot.send_message(message.from_user.id, fuel_msg, reply_markup =
menu.mainMenu)


#---Інше
@dp.message_handler(lambda message: message.text.startswith('⚙ Інше'))
async def other_menu(message: types.Message):
```

```
    if registered(message.from_user.id) == 1:
        other_msg = "Оберіть команду:"
        await bot.send_message(message.from_user.id, other_msg, reply_markup =
menu.otherMenu)

#----------------------------------Конвертер валют------------------------------
---------
@dp.message_handler(lambda message: message.text.startswith('↔ Конвертер
валют'))
async def convert(message: types.Message):
    if registered(message.from_user.id) == 1:
        converter_msg = "Оберіть тип валюти:"
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.currenciesMenu)
        await states.convert.ctype.set()

@dp.message_handler(state = states.convert.ctype)
async def ctype(message: types.Message, state: FSMContext):
    answer = message.text
    await state.update_data(ctype = answer)
    converter_msg = "Оберіть скорочену назву валюти, що конвертуєте (або введіть
необхідну):"
    data = await state.get_data()
    if data['ctype'] == '🏦 Національні валюти':
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.national_moneyMenu)
    elif data['ctype'] == ' Криптовалюти':
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.crypto_moneyMenu)
    await states.convert.from_cur.set()

@dp.message_handler(state = states.convert.from_cur)
async def from_cur(message: types.Message, state: FSMContext):
    answer = message.text
    await state.update_data(from_cur = answer)
    converter_msg = "Введіть кількість валюти:"
    data = await state.get_data()
    if data['ctype'] == '🏦 Національні валюти':
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.national_moneyMenu)
    elif data['ctype'] == ' Криптовалюти':
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.crypto_moneyMenu)
    await states.convert.amount.set()

@dp.message_handler(state = states.convert.amount)
async def amount(message: types.Message, state: FSMContext):
    answer = message.text
    await state.update_data(amount = answer)
    converter_msg = "Оберіть скорочену назву валюти, в яку конвертуєте (або
введіть необхідну):"
    data = await state.get_data()
    if data['ctype'] == '🏦 Національні валюти':
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.national_moneyMenu)
    elif data['ctype'] == ' Криптовалюти':
        await bot.send_message(message.from_user.id, converter_msg, reply_markup
= menu.crypto_uah_moneyMenu)
    await states.convert.to_cur.set()

@dp.message_handler(state = states.convert.to_cur)
async def amount(message: types.Message, state: FSMContext):
    answer = message.text
```

```
    await state.update_data(to_cur = answer)
    data = await state.get_data()
    converter_msg = scrapper.state_handler(data)
    await bot.send_message(message.from_user.id, converter_msg, reply_markup =
menu.mainMenu)
    await state.finish()

#---------------------------------Зміна портфоліо-----------------------------
---------
@dp.message_handler(lambda message: message.text.startswith('📑 Змінити
портфоліо'))
async def func(message: types.Message):
    if registered(message.from_user.id) == 1:
        msg = "Оберіть дію:"
        await bot.send_message(message.from_user.id, msg, reply_markup =
menu.editPrtMenu)
        await states.portfolio.action.set()

@dp.message_handler(lambda message: message.text.startswith('➕ Додати
елемент'), state = states.portfolio.action)
async def func(message: types.Message, state: FSMContext):
    cmd = "add"
    await state.update_data(mode = cmd)
    msg = "Оберіть тип активу:"
    await bot.send_message(message.from_user.id, msg, reply_markup =
menu.typePrtMenu)
    await states.portfolio.ctype.set()

@dp.message_handler(lambda message: message.text.startswith('🏦 Банківський
актив'), state = states.portfolio.ctype)
async def func(message: types.Message, state: FSMContext):
    answer = "banks_"
    await state.update_data(ctype = answer)
    msg = "Введіть назву банку:"
    await bot.send_message(message.from_user.id, msg)
    await states.portfolio.currency.set()

@dp.message_handler(state = states.portfolio.currency)
async def func(message: types.Message, state: FSMContext):
    answer = message.text
    await state.update_data(currency = answer)
    data = data = await state.get_data()
    if data['mode'] == "add":
        msg = "Введіть кількість валюти:"
        await bot.send_message(message.from_user.id, msg)
        await states.portfolio.amount.set()
    if data['mode'] == "edit":
        msg = "Введіть кількість валюти:"
        await bot.send_message(message.from_user.id, msg)
        await states.portfolio.amount.set()
    elif data['mode'] == "del":
        try:
            prt.del_item(message.from_user.id, data['currency'], data['ctype'])
            msg = 'Актив видалено ✓'
        except Exception as e:
            msg = "Помилка: " + str(e)
        await bot.send_message(message.from_user.id, msg, reply_markup =
menu.mainMenu)
        await state.finish()


@dp.message_handler(lambda message: message.text.startswith('💸 Готівка'), state
= states.portfolio.ctype)
```

```
async def func(message: types.Message, state: FSMContext):
    answer = "cash_"
    await state.update_data(ctype = answer)
    msg = "Оберіть скорочену назву валюти (або введіть необхідну):"
    await bot.send_message(message.from_user.id, msg, reply_markup =
menu.national_moneyMenu)
    await states.portfolio.currency.set()

@dp.message_handler(lambda message: message.text.startswith(' Криптовалюта'),
state = states.portfolio.ctype)
async def func(message: types.Message, state: FSMContext):
    answer = "crypto_"
    await state.update_data(ctype = answer)
    msg = "Введіть повну назву криптовалюти (з великої букви):"
    await bot.send_message(message.from_user.id, msg)
    await states.portfolio.currency.set()

@dp.message_handler(state = states.portfolio.amount)
async def func(message: types.Message, state: FSMContext):
    answer = message.text
    data = await state.get_data()
    if data['mode'] == "edit":
        try:
            prt.edit_item(message.from_user.id, data['currency'], answer,
data['ctype'])
            msg = 'Баланс змінено ✓'
        except Exception as e:
            msg = "Помилка: " + str(e)
    elif data['mode'] == "add":
        try:
            prt.add_item(message.from_user.id, data['currency'], answer,
data['ctype'])
            msg = 'Актив додано ✓'
        except Exception as e:
            msg = "Помилка: " + str(e)
    await bot.send_message(message.from_user.id, msg, reply_markup =
menu.mainMenu)
    await state.finish()

@dp.message_handler(lambda message: message.text.startswith('➖ Видалити
елемент'), state = states.portfolio.action)
async def func(message: types.Message, state: FSMContext):
    cmd = "del"
    await state.update_data(mode = cmd)
    msg = "Оберіть тип активу:"
    await bot.send_message(message.from_user.id, msg, reply_markup =
menu.typePrtMenu)
    await states.portfolio.ctype.set()

@dp.message_handler(lambda message: message.text.startswith('📝 Змінити
баланс'), state = states.portfolio.action)
async def func(message: types.Message, state: FSMContext):
    cmd = "edit"
    await state.update_data(mode = cmd)
    msg = "Оберіть тип активу:"
    await bot.send_message(message.from_user.id, msg, reply_markup =
menu.typePrtMenu)
    await states.portfolio.ctype.set()

if __name__ == '__main__':
    #start_webhook(
    #    dispatcher=dp,
    #    webhook_path=APP_URL,
```

```
#     on_startup=on_startup,
#     on_shutdown=on_shutdown,
#     skip_updates=True,
#     host=WEBAPP_HOST,
#     port=WEBAPP_PORT,
#)

#server.run(host="0.0.0.0", port = int(os.environ.get("PORT", 5000 )))
executor.start_polling(dp, skip_updates=True)
```