**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ**

**Інститут (факультет )** Автоматизації і комп'ютерних систем
**Кафедра** Інформаційних технологій, штучного інтелекту і кібербезпеки

| | |
|---|---|
| **«До захисту в ЕК»** | **«До захисту допущено»** |
| Директор інституту(декан факультету) | Завідувач кафедри |
| _____ Андрій Форсюк | _____ Сергій Грибков |
| (підпис) (ім'я та прізвище) | (підпис) (ім'я та прізвище) |
| «13» лютого 2023р. | «13» лютого 2023р. |

**КВАЛІФІКАЦІЙНА РОБОТА**
**НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА**

зі спеціальності 122 «Комп'ютерні науки»
(код та назва спеціальності)
освітньо-професійної програми Інформаційні управляючі системи та технології
на тему: Дослідження блокчейн технологій для обробки і передачі інформації з використанням криптографічних методів шифрування даних

Виконав: здобувач 2 курсу, групи ІС-2-3М

Жидко Анна Андріївна _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Горлова Тетяна Михайлівна _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти _____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент _____ _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач _____
(підпис)

Київ -2023р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних технологій, штучного інтелекту і кібербезпеки
Освітній ступінь магістр
**Спеціальність** 122 «Комп'ютерні науки»
(код і назва)
**Освітньо-професійна програма** Інформаційні управляючі системи та технології
(назва)

**ЗАТВЕРДЖУЮ**

**Завідувач**
**кафедри** Інформаційних технологій,
штучного інтелекту і кібербезпеки

Грибков С. В.
**« 11 » листопада 2022 року**

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

**Жидко Анни Андріївна**
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження блокчейн технологій для обробки і передачі інформації з використанням криптографічних методів шифрування даних*
керівник роботи *Горлова Тетяна Михайлівна доцент, к. т. н., с. н. с ,*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «11» листопада 2022 р. № 820-кс
2. Строк подання здобувачем роботи: 18.01.23
3. Вихідні дані до роботи: *Інформація про структуру блокчейн, криптографічні методи шифрування і криптовалюту, офіційна документація для програмних засобів, дані про розвиток блокчейн технологій в Україні.*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Аналіз доцільності використання блокчейн технологій, використання криптографії в блокчейні, забезпечення обробки конфіденційних даних користувача, процес взаємодії з мережами біткоїну та етеріуму.
5. Перелік графічного матеріалу:

Структура блокчейну, глобальні графіки криптовалют, база даних, асиметричне криптографічне шифрування, стандарти BIP32 та BIP39, інтерфейс користувача бота.

6. Консультанти розділів роботи:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 | доц. Горлова Т.М. | 11.11.22 | 12.11.22 |
| 2 | доц. Горлова Т.М. | 11.11.22 | 14.12.22 |
| 3 | доц. Горлова Т.М. | 11.11.22 | 28.12.22 |
| 4 | доц. Горлова Т.М. | 11.11.22 | 10.01.22 |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання: 11.11.2022

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів виконання кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз актуальності обраної теми в сучасному світі. | 19.11.22 - 30.11.22 | Виконано |
| 2 | Вивчення теоретичного матеріалу технології блокчейн. | 30.01.22 - 16.12.22 | Виконано |
| 3 | Створення моделі роботи майбутньої програми. | 16.12.22 - 23.12.22 | Виконано |
| 4 | Програмна реалізація запланованих функцій. | 23.12.22 - 17.01.23 | Виконано |
| 5 | Тестування розробленої комп'ютерної програми. | 17.01.23 - 20.01.23 | Виконано |
| 6 | Оформлення та фіксування кваліфікаційної роботи. | 20.01.23 - 24.01.23 | Виконано |
| 7 | Підготовка автореферату. | 24.01.23 - 25.01.23 | Виконано |
| 8 | Створення презентації. | 25.01.23 - 26.01.23 | Виконано |
| | | | |

**Здобувач** _____ Жидко А.А.
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ Горлова Т. М.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

В ході кваліфікаційної роботи був розроблений автоматизований бот, в основі діяльності якого лежить технологія блокчейн.

Об’єктом вивчення є використання технології блокчейн на прикладі таких криптовалют, як біткоїн та етеріум.

Реалізація програмної частини була виконана у середовищі Телеграм на мові програмування Python. Веб-додаток створений з використанням веб-фреймоврку Django. Запуск проекту відбувався за допомогою Docker-контейнерів.

Кваліфікаційна робота складається з 79 сторінок, містить 51 графічних зображень, 6 додатків, 14 літературних джерел.

**Ключові слова :** ТЕХНОЛОГІЯ БЛОКЧЕЙН, КРИПТОВАЛЮТА, БІТКОЇН, ЕТЕРІУМ, БОТ, КРИПТОГРАФІЯ, КРИПТОГАМАНЕЦЬ, АДРЕСА, СЕРВЕР, БАЗА ДАНИХ, МОВА ПРОГРАМУВАННЯ PYTHON, ТЕЛЕГРАМ

## SUMMARY

During the qualification work, an automated bot was developed, the basis of which is blockchain technology.

The object of study is the use of blockchain technology on the example of such cryptocurrencies as Bitcoin and Ethereum.

The software part was implemented in the Telegram environment using the Python programming language. The web application was created using the Django web framework. The project was launched using Docker containers.

Qualification work contains 79 pages course project, 51 pages of images, 6 additions, 14 literary sources.



**Keywords:** BLOCKCHAIN TECHNOLOGY, CRYPTOCURRENCY, BITCOIN, ETHEREIUM, BOT, CRYPTOGRAPHY, CRYPTOWALLET, ADDRESS, SERVER, DATABASE, PYTHON PROGRAMMING LANGUAGE TELEGRAM

**ЗМІСТ**

АНОТАЦІЯ ..... 3

SUMMARY ..... 4

ВСТУП ..... 6

РОЗДІЛ 1. ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ. ПОСТАНОВКА ЗАВДАННЯ ..... 8

РОЗДІЛ 2. ОСНОВНІ ПОНЯТТЯ ТЕХНОЛОГІЇ БЛОКЧЕЙН ..... 12

РОЗДІЛ 3. КРИПТОВАЛЮТА, ЯК ПРИКЛАД ВИКОРИСТАННЯ БЛОКЧЕЙН ТЕХНОЛОГІЇ ..... 16

3.1. Асиметричний криптографічний алгоритм шифрування даних ..... 16

3.2. Біткоїн, як перше покоління криптовалюти ..... 17

3.3. Етеріум, як криптовалюта другого покоління. ..... 18

3.3. Відмінність між Біткоїном та Етеріумом в межах магістрерської. ..... 19

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ ..... 22

4.1. Огляд використаних програмних засобів. ..... 22

4.2. Налаштування мережі біткоїну та етеріуму. ..... 22

4.3. Розроблення головного серверу для впровадження API. ..... 24

4.4. Написання автономного бота в середовищці Телеграм ..... 26

4.5. Бот підтримки користувача ..... 38

4.6. Веб-сторінка для адміністрування ..... 40

РОЗДІЛ 5. ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧА ..... 44

ВИСНОВКИ ..... 60

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ..... 61

ДОДАТКИ ..... 63

# ВСТУП

Розвиток інформаційних технологій збагачує та насичує життя кожної людини новими можливостями та викликами. З плином часу все в світі стає цифровим: наше спілкування, документи та звичайно гроші. У багатьох країнах вже проводиться тестування перших офіційних цифрових валют. Цифрові картини на аукціонах коштують мільйони доларів. Тобто епоха гаджиталізації набігає кінця, адже крізь є інтернет, всі мають смартфони, а це – термінали для входу до цифрового світу. Ми вступаємо до нової епохи – загальної цифровізації, крокуючи до порталу і це все завдяки технології з не дуже складною назвою – блокчейн.

В роботі досліджені основні моменти криптографічних методів шифрування, які використовуються в технології блокчейн, різниці між звичайними грошима та криптовалютою.

Варто зазначити, що на Всесвітньому економічному форумі у Давосі у 2018 р. Д.Тапскотт, що є одним з найголовніших експертів блокчейну у світі й автором бестселлера «Блокчейн-революція», презентував карту з 14-ма країнами-лідерами із впровадження блокчейн-технологій. Згідно, досліджуваних даних Україна посідає 14 місце серед провідних країн з запровадженню блокчейну-технології, що свідчить про готовність уряду та провідних компаній використовувати дану технологію [1].

Українська платформа Дія вже сьогодні використовує блокчейн для збереження критично важливих даних. У 2017 році Кабінет Міністрів уклав договір з американською компанією BitFury про переведення державних реєстрів на блокчейн, що має полегшити реєстрацію фізичних осіб та підприємств [7].

**Метою** кваліфікаційної роботи є дослідження внутрішніх процесів технології блокчейн із використанням програмного забезпечення та набутих навичок під час опанування вищої освіти ступеня магістр.

**Завданням дослідження** є створення автоматизованого бота, який відображатиме основні риси технології блокчейн.

**Об'єкт дослідження роботи** – розвиток та розповсюдження блокчейн технологій в життя.

**Предметом** дослідження є процес створення, передачі та обробки інформації в блокчейн з використанням методів шифрування даних на прикладі таких криптовалют, як біткоїн та етеріум.

В процесі написання дипломної роботи були використані такі **методи**: Метод спостереження, Метод експериментально-теоретичного рівня, Аналітичний метод, Розрахунковий метод.

**Науковою новизною роботи** є створення нового способу оброблення та здійснення грошових транзакцій.

**Практичною цінністю програми** є те, що будь-яка людина може створити криптовалютний гаманець для здійснення транзакцій в валюті Bitcoin та Ethereum. Власник бота зможе заробляти на відсотках від кожної транзакції. Комісія від транзакцій регулюється на адміністративній веб-сторінці в залежності від валюти.

**Апробація результатів магістерської роботи.**

Матеріали дослідження блокчейн технологій для обробки і передачі інформації з використанням криптографічний методів шифрування даних були опубліковані у IX Міжнародній науково-технічній Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами».

# РОЗДІЛ 1. ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ ДОСЛІДЖЕННЯ. ПОСТАНОВКА ЗАДАЧІ

Наша сучасність змінюється щохвилини. Сьогодні фінансові системи країн удосконалюються і прогресують у контексті розвитку і поширення ІТ-технологій. Так і з'явився аналог звичайним фіатним грошам - криптовалюта. Фіантні гроші – це тип грошей або валюти, цінність яких походить не від власної вартості, а від державного наказу використання їх як засоби платежу. Наразі система грошового обігу світу є нестабільною. Вона залежить від багатьох факторів, наприклад, фінансова криза у 2008-2009 рр. Саме тоді довіра до американського долара послабшала і люди почали замислюватись над іншими ідеями щодо створення нових валют [6].

Вперше застосовувався термін «криптовалюта», як альтернатива звичайним грошам, у 2009 році, коли Сатоші Накамото створив першу в світі криптовалюту – біткоїн. Криптовалюта – цифрова монета, яка захищена від підробки та яку можна зберігати в електронних гаманцях.

Криптовалюта є незалежною грошовою одиницею, яку ніхто не регулює і не здійснює контроль за рухом коштів, а отже заморожування рахунків не можливе. Вона стійка та незалежна від інфляції. Криптовалюта гарантує повну анонімність – всі дані про власника цифрового гаманця зашифровані, доступним є лише номер гаманця та значення суми, яка є на рахунку. Всі віртуальні монети неможливі підробити або здійснювати над ними будь-які шахрайські дії.

За допомогою графіку нижче показано відношення десяти найбільших криптовалют щодо загальної ринкової капіталізації. Оскільки Bitcoin, що позначений помаранчевим кольором, був першим активом, він залишається найбільшим за обсягом ринкової капіталізації, тому багато

людей стежать за його домінуванням на ринку. Простими словами, біткоїн – найпоширеніша криптовалюта за всі роки. [8]

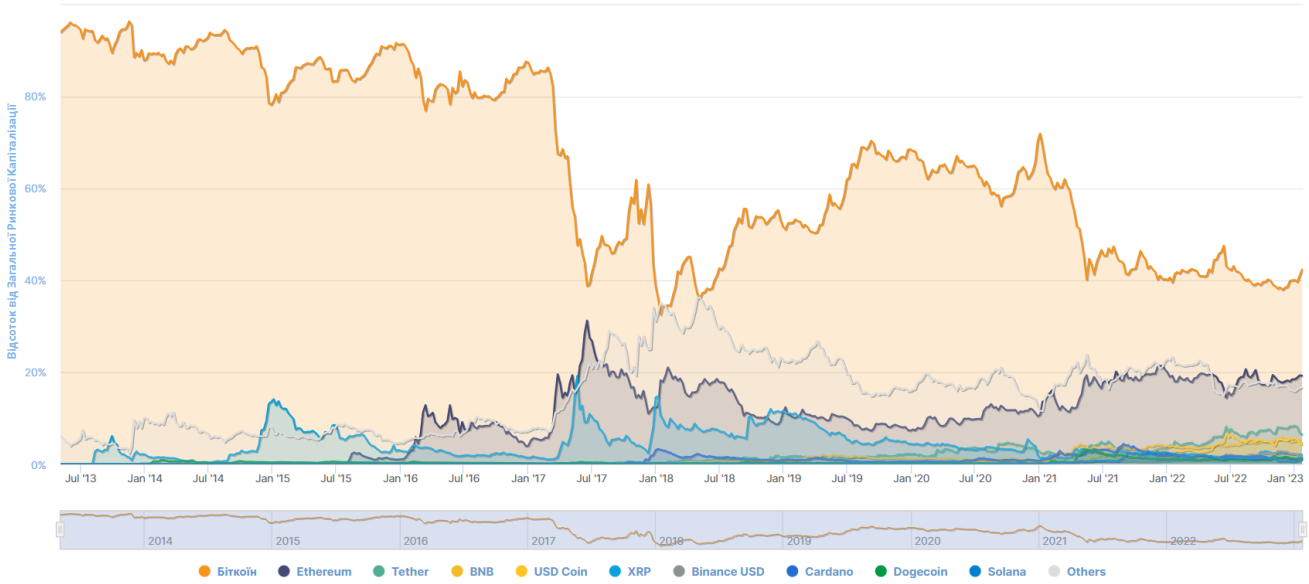


Рисунок 1.1. Відношення криптовалют щодо загальної ринкової капіталізації

В основі криптовалюти закладена технологія блокчейн. Нововведення цієї технології полягає в тому, що інформація про транзакції більш не зберігається в централізованій базі даних, а передається на комп'ютери всіх учасників мережі, які зберігають дані локально. Фактично технологія блокчейн – універсальний метод зберігання та обробки даних практично в будь-якій сфері діяльності. Великі компанії не можуть ігнорувати потенціал блокчейну, тому активно його впроваджують. Можливості застосування цієї технології в бізнесі та промисловості не знають кордонів, насамперед в криптовалютному середовищі.

На даний момент в світі налічується близько 1,7 тис. віртуальних валют. Понад 600 з них є активними, а ринкова вартість 30 валют більше $1 млрд. За 2023 рік Bitcoin виріс у вартості більше ніж в 20 разів. На сьогодні вартість одного біткоїну по курсу долара за 2023 становить $23 018.10.

Однак біткоїн – криптовалюта першого покоління. Він не був створений як занадто складна система. У свою чергу, друге покоління блокчейну здатне на більше. Яскравим прикладом є етеріум – криптовалюта та платформа для створення децентралізованих онлайн-сервісів із використанням блокчейну, що працюють на базі розумних контрактів. Етеріум та біткоїн надають можливість обмінюватися цифровими грошима, однак етеріум здатний на більше. За допомогою криптовалюти другого покоління ви зможете розгорнути власний код і взаємодіяти з додатками, створеними іншими користувачами. У зв'язку з тим, що його система дуже гнучка, Ethereum дозволяє запускати складні програми на своїй основі. Наразі вартість однієї найменшої одиниці етеріуму складає $1555.8.

Наведемо графіки зміни вартості біткоїну та етеріуму за всі роки.

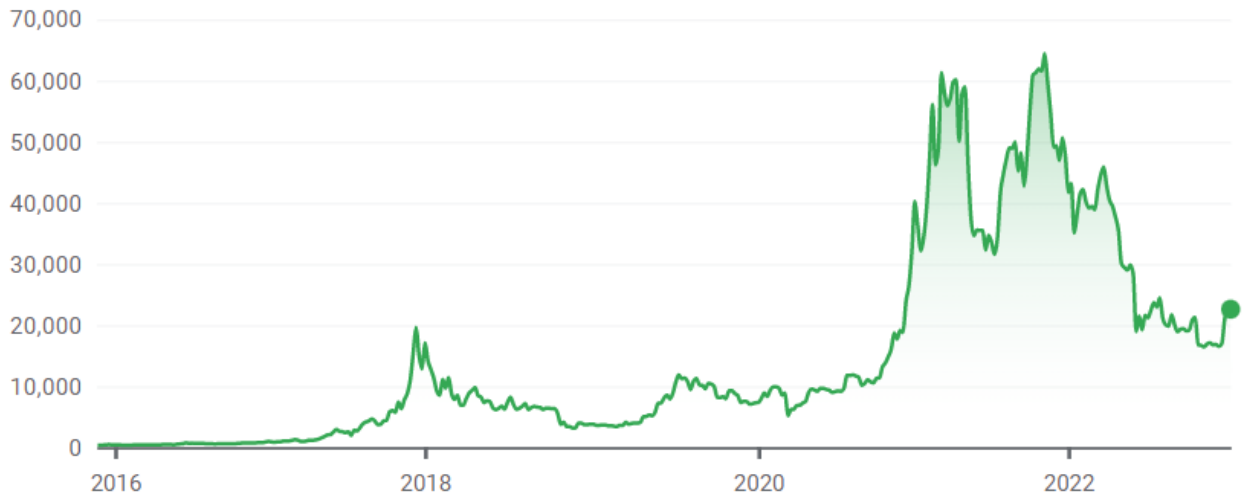


Рисунок 1.2. Графік зміни вартості біткоїну

На графіку вище наведено діапазон, починаючи з 2016 року, оскільки цінність біткоїну не змінювалась з моменту створення. Зростання вартості свідчить про активне використання користувачами цифрової валюти. Максимальна вартість була зафіксована 12 листопада 2021 року в розмірі $64 400.

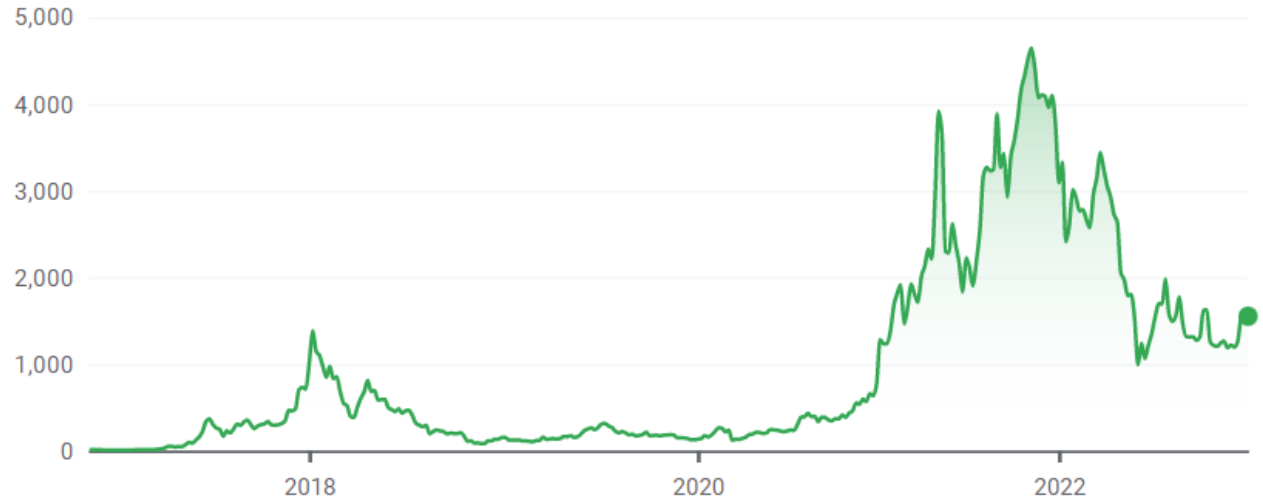


Рисунок 1.3. Графік зміни вартості етеріуму

На рисунку 1.3 наведено динаміку зміни ціни за криптовалютну одиницю етеріуму. Найвища вартість була 12 листопада 2021 року в розмірі $4644.63.

Таким чином, графіки допомагають зрозуміти, що вартість криптовалюти в світі досить висока, а використання її поширюється кожної миті.

Швидке розповсюдження криптовалют вже не може залишатися поза увагою держав, багато з яких досі не можуть визначитися яким чином слід її регулювати. Україна входить в ТОП-10 країн за кількістю користувачів криптовалюти. Криптовалюту в Україні офіційно не визнали та Секретар Ради національної безпеки і оборони України нещодавно заявив, що питання криптовалют більше не може залишатися поза увагою держави, зважаючи на стрімкий розвиток ринку валют у світі.

Таким чином, можна зробити висновок, що використання блокчейн технологій швидко набирає популярність і тому було прийняте рішення про детальне вивчення цієї технології.

Розглянемо детальніше як працює технологія блокчейн в цілому та на прикладі використання криптовалют.

## РОЗДІЛ 2. ОСНОВНІ ПОНЯТТЯ ТЕХНОЛОГІЇ БЛОКЧЕЙН

З назви блокчейн стає зрозумілим, що термін представляє собою ланцюжок блоків, всередині яких знаходиться певна інформація. Технологія блокчейн вперше була описана в 1991 році групою дослідників та призначалася для збереження цифрових документів, для того щоб уникнути підробку інформації, що в них зберігається. Така технологія майже не використовувалася, поки в 2009 році Сатоші Накомото не вдосконалив її для створення першої в світі криптовалюти – біткоїн. [10]

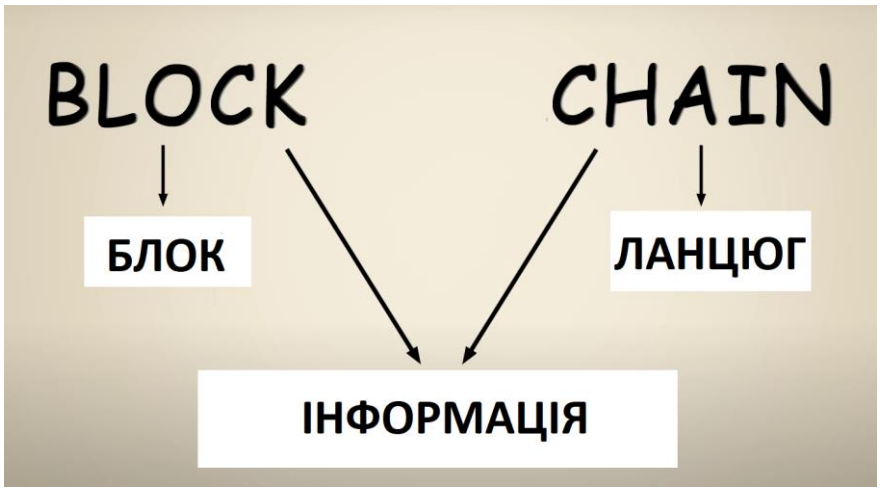


Рисунок 2.1. Розшифрування назви блокчейн технології

Блокчейн представляє собою розподілену книгу, яка повністю відкрита для кожної людини і має дуже важливу властивість – після того, як дані були записані в блокчейн, їх практично неможливо змінити.

Для того, щоб зрозуміти як це працює, розглянемо як працює блок. Він зберігає певні дані, хеш блоку та хеш минулого блоку. Дані, які зберігаються всередині блоку залежать від типу блокчейну. Наприклад, в біткоїні зберігаються подробиці про транзакцію: інформація про відправника, отримувача та кількості монет. Кожен блок має хеш, який можна порівняти з відбитком пальця – завжди унікальний. Він ідентифікує блок та всі його складові. [4]


Рисунок 2.2. Складові одного блоку

В ту мить як створюється новий блок, йому привласнюється хеш. Якщо всередині блоку щось змінюється, то це призведе до зміни хешу. Завдяки цьому хеш блоку дуже корисний, якщо ви хочете виявити зміну в блоках. Якщо хеш змінюються, то блок більш не є тим самим блоком.

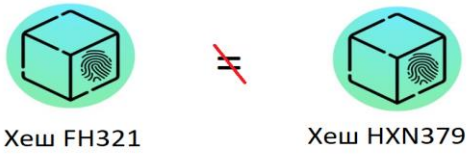


Рисунок 2.3. Зображення нерівності блоків з різними хешами

Представимо ланцюжок з трьох блоків. Кожен блок має хеш та хеш попереднього блоку. Перший блок особливий, оскільки він не вказує на попередній блок. Він має назву Генезис-блок.


Рисунок 2.4. Проста модель блокчейну

Уявимо, що ви хочете змінити блок номер два. Це звичайно змінить його хеш, тому блок номер три та наступні за ним блоки не дієвими, оскільки вони більш не будуть зберігати валідний хеш минулих блоків. Таким чином, зміна одного блока зробить наступні за ним блоки не дієвими. Однак використання одного хеш не достатньо для попередження фальсифікації, тому що комп'ютери в теперішній час мають великі потужності і можуть вичислити сотні хешів в секунду, тому з'являється можливість суттєво змінити блок та просто перерахувати всі хеші інших блоків, для того щоб зробити блокчейн дієвим через це фальсифікацію може ніхто не помітити. Для того, щоб запобігти цьому в блокчейні використовуються механізм, який називається «Доказ роботи» (англ. «Proof of work»). Суть цього механізму полягає в тому, що він значно зменшує час на генерацію нових блоків. Наприклад, якщо розглянути біткоїн, то він потребує близько десяти хвилин для того, щоб вирахувати «доказ роботи» та додати новий блок.

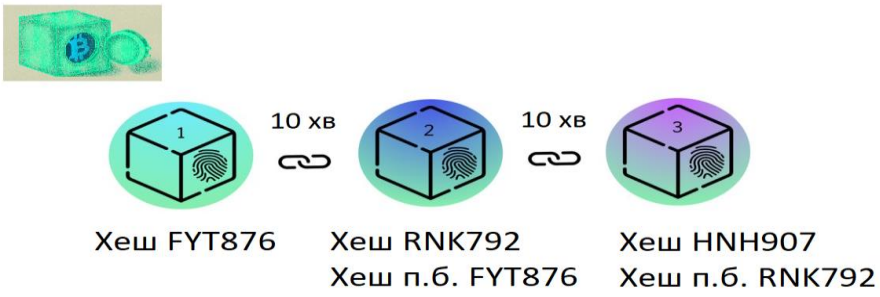


Рисунок 2.5. Використання «proof of work» в біткоїні

Завдяки цьому суттєво ускладняється зміна блоків, тому що якщо ви змінюєте один блок, вам потрібно буде перерахувати «доказ роботи» для решти блоків, що є дуже складним процесом.

Можна зробити висновок, що надійність блокчейну полягає в сукупності хешування і доказу роботи.

Є інший ефективний спосіб для забезпечення безпеки блокчейну – децентралізація. Він передбачає відсутність одного централізованого об’єкту, який буде керувати всім ланцюгом.

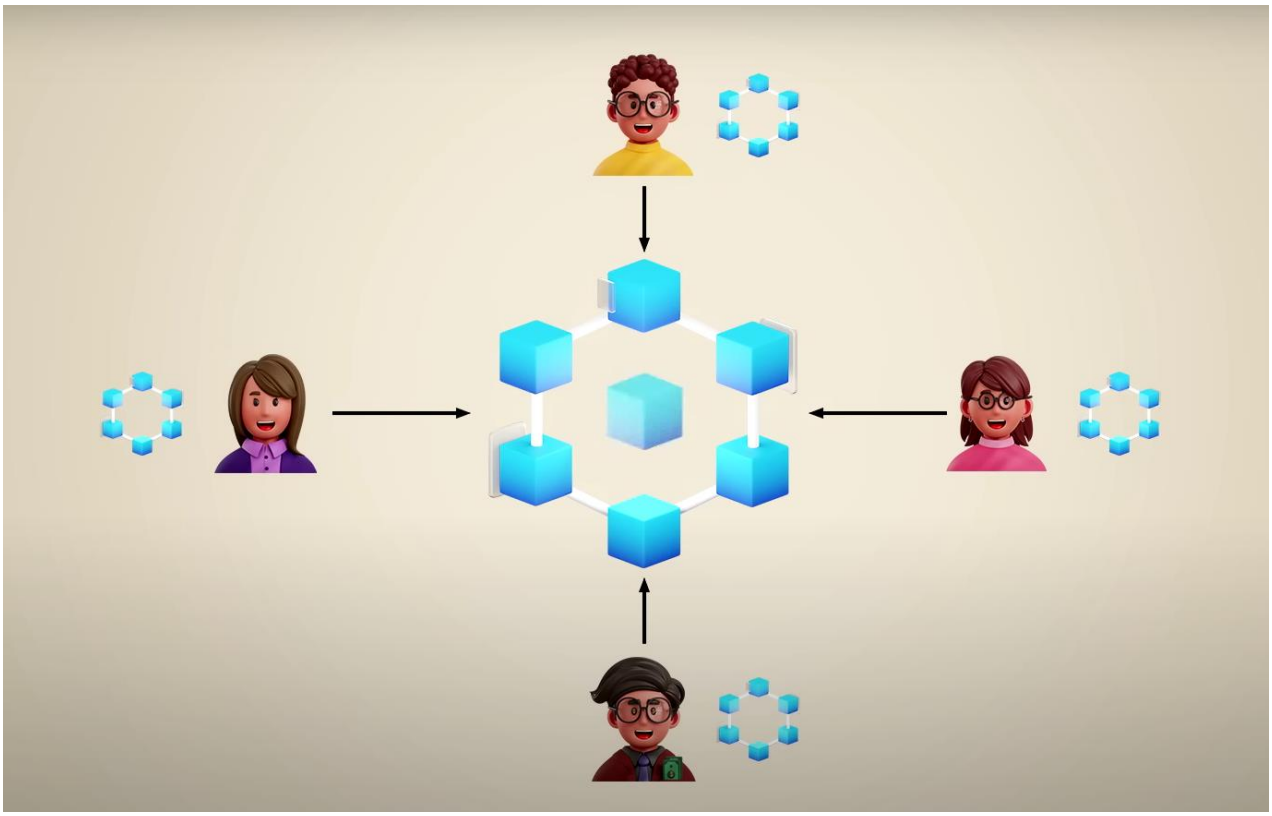

Рисунок 2.6. Децентралізація в блокчейні

Блокчейн використовує мережу з можливістю підключення абсолютно кожної людини. Наприклад, коли людина приєднується до мережі, вони отримує повну копію блокчейну. Якщо хтось створює новий блок, то такий блок відправляється всім учасникам мережі. Кожний вузол перевіряє цей блок, а саме його хеші для того, щоб переконатися в його правдивості. Якщо все гаразд, то кожний вузол додає новий блок до своєї копії блокчейну. Після чого всі вузли мережі досягають консенсусу та спільної згоди. Якщо блок фальсифікований, то такий блок буде відхиллено іншими вузлами мережі. [10].

Розглянемо найпоширеніше використання технології блокчейн – світ криптовалюти.

# РОЗДІЛ 3. КРИПТОВАЛЮТА, ЯК ПРИКЛАД ВИКОРИСТАННЯ БЛОКЧЕЙН ТЕХНОЛОГІЇ

Використання звичайної банківської картки для оплати давно стало для людства звичайною справою. З часом, така можливість з’явилась і у криптовалют. Однак, банки оберігають нас від людської помилки, а ось відповідальність за переказ коштів лежить тільки на вас.

Розглянемо детальніше, що таке криптовалюта.

В роботі розглядаються два різних види криптовалют – біткоїн та етеріум. Такі валюти були обрані, з метою зображення еволюції крипто-світу, починаючи з найпершого, біткоїну, та закінчуючи більш сучасною та швидкою криптовалютою етеріуму.

## 3.1. **Асиметричний криптографічний алгоритм шифрування даних**

Продемонструємо теорію асиметричного шифрування на простому прикладі. Нехай Аліса має важливий документ, який необхідно відправити Бобу. Вона використовує програму для шифрування для того, щоб захистити цей документ за допомогою паролю. Вона відправляє зашифрований документ Бобу, але той не може його розшифрувати, оскільки не знає ключа.

Для вирішення цієї проблеми використовується асиметричне шифрування. Найпопулярнішим є RSA algorithm. Аліса і Боб генерують пару публічного і приватного ключа, які пов’язані між собою математично. Важливо зауважити, що приватний ключ не може бути згенерований із публічного. Іншими словами, якщо ви знаєте чийсь публічний ключ, то ви не зможете отримати його приватний ключ. Тепер розглянемо як Боб та Аліса будуть використовувати асиметричне шифрування для передачі інформації.

Перший крок – це обмін приватними ключами. Аліса шифрує документ, використовуючи публічний ключ Боба. Далі вони відправляє зашифрований файл Бобу. Боб використовує свій приватний ключ для дешифровки

документа. Тільки він зможе розшифрувати. Навіть Аліса не зможе це зробити. Відповідальність за приватний ключ несе тільки його власник. Якщо зловмисник отримує приватний ключ, то він зможе розшифрувати всі повідомлення, що були зашифровані цим ключом.

.

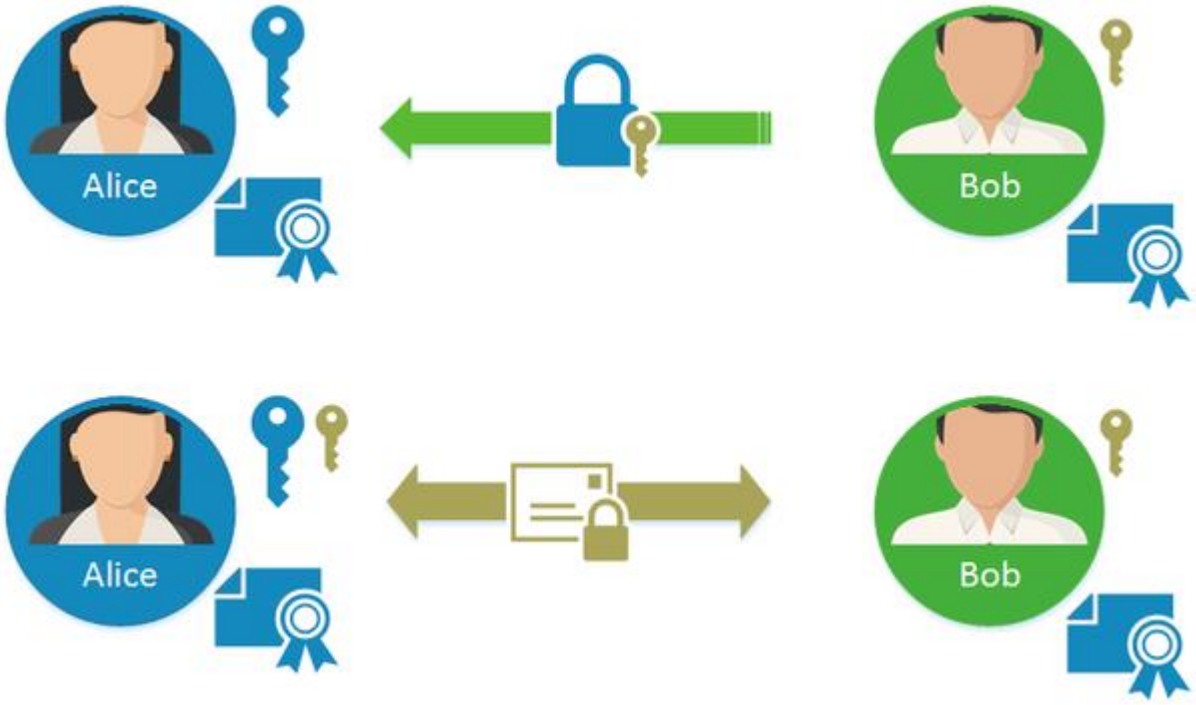


Рисунок 3.1. Схематичне зображення зашифрованого спілкування Аліси та Боба

## 3.2. **Біткоїн, як перше покоління криптовалюти**

Біткоїн – децентралізована електрона платіжна система, побудована на технології блокчейн, концепт якої був опублікований 2008 року Сатоші Накамото, і реалізований ним у 2009 році. Абревіатура біткоїну – BTC [10].

Процес передачі криптогрошей з одного гаманця на інший називається транзакцією.

На відміну від банка, де ви знаєте тільки про свої транзакції, в біткоїні всі знають про всі транзакції.

Розглянемо детальніше транзакцію біткоїнів на прикладі Аліси та Боба. Для здійснення транзакції необхідно вказати кому та від кого йдуть гроші та їхню кількість. Ці дані додаються до блоку блокчейну. За правилами біткоїну для того, щоб перевести кошти необхідно підписати транзакцію. Це відбувається за допомогою пари приватного та публічного ключа. Приватний ключ створює підпис, а приватний ключ надає можливість іншим людям перевірити її. Уявіть, що приватний ключ – це істинний пароль, а підпис – це посередник, який підтверджує, що ви знаєте пароль без необхідності його розкривати.

Публічний ключ виступає в ролі адреси, на яку відправляють кошти.

Відправляючи гроші людині, ви відправляєте їх на його публічний ключ. Для того, щоб витратити кошти, необхідно довести, що ви є власник цих коштів або те, що вони у вас взагалі є. Для цього генерується підпис із вашого приватного ключа. Інші вузли мережі перевіряють, що цей підпис відповідає вашому публичному ключу. За допомогою розрахунків, вони бачать, що у вас дійсно є приватний ключ, не знаючи його.

Зауважимо, що для кожної транзакції генерується новий підпис, тому зловмисник не може використати той самий підпис для іншої транзакції.

### 3.3. **Етеріум, як криптовалюта другого покоління**

Етеріум – криптовалюта, що базується на основі блокчейна, яка працює на базі розумних контрактів.  Абревіатура криптовалюти Ethereum – ETH (ether) [9].

Ідея була сформована Віталіком Бутеріном у 2013 році. Етеріум – піонер у використанні розумний або інтелектуальних контрактів на основі блокчейнів. При запуску на блокчейн розумний контракт стає схожим на самостійну комп’ютерну програму, яка автоматично виконується, коли виконуються певні умови.

Смарт-контракти Ethereum розробляються на одній з мов, спроектованих для трансляції в байт-код віртуальної машини Ethereum — Solidity (схожий на C або JavaScript), Vyper і Serpent (схожі на Python), LLL (низькорівнева версія Lisp), Mutan (заснований на Go).

Сторони підписують розумний контракт, використовуючи методи, аналогічні підписанню перерахунку коштів в криптовалютні мережі, що існують в наш час. Після підписання сторонами, контракт зберігається в блокчейні і вступає в силу.

Всі умови контракту повинні мати програмний опис і ясну логіку виконання

Для того, щоб розумні контракти могли існувати, потрібні певні умови:

- Використання широко поширених методів електронного підпису на основі публічних і приватних ключів (асиметричне шифрування).
- Існування відкритих, децентралізованих і довірчих сторонам контракту баз даних для виконуваних транзакцій, робота яких повністю виключає людський фактор. Як приклад: блокчейн в Bitcoin.
- Децентралізація середовища виконання розумного контракту. Як приклад: Ethereum, Codius, Counterparty.

### 3.4. **Відмінність між Біткоїном та Етеріумом в межах магістерської роботи**

Взаємодія з біткоїном обмежується стандартом BIP 32, який відповідає за генерацію пари публічного і приватного ключа [10].

BIP 32 — це пропозиція щодо вдосконалення біткойна, яка запровадила стандарт ієрархічних детермінованих (HD) гаманців і розширені ключі до біткойна. В основі стандарту лежить насіння англ «seed» — це частина даних, яка може бути використана для створення ієрархічного детермінованого (HD) гаманця. Це єдині дані, необхідні для відновлення

будь-яких приватних і відкритих ключів у гаманці, тому вони ефективні як резервна копія.

Насіння — це просто ентропія — випадковий рядок цифр. Початкове число використовується для створення єдиного розширеного закритого ключа англ «extended private key» (xprv), який називається головним закритим ключем англ «master private key». Цей приватний ключ можна використовувати для генерації дочірніх приватних ключів, а також відкритих ключів, дозволяючи гаманцю генерувати стільки пар ключів, скільки потрібно користувачеві.

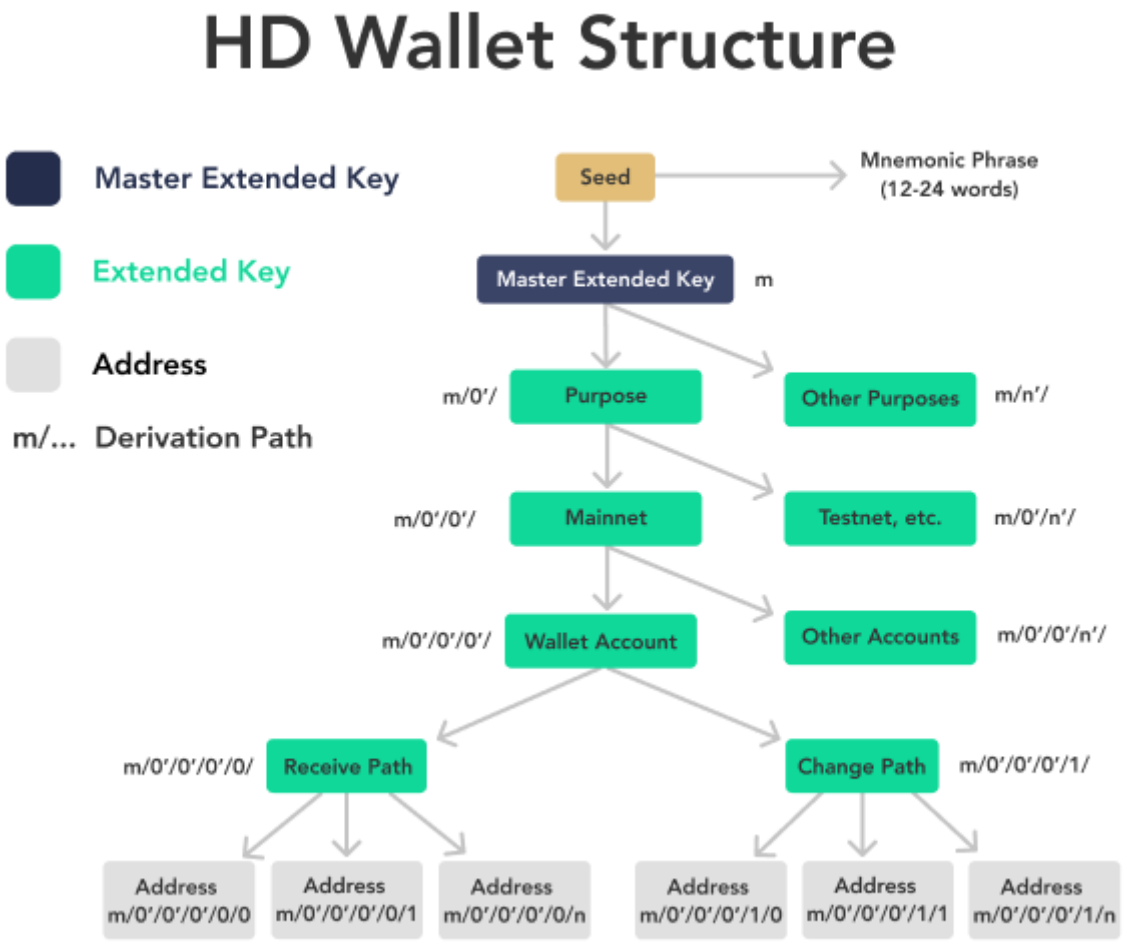


Рисунок 3.2. Схематичне зображення HD-криптогаманця

Етеріум в свою чергу використовує більш новий стандарт – BIP 39.

Етап у процесі налаштування нового криптовалютного гаманця – це випуск мнемонічної фрази або вихідних фраз із 12 до 24 слів. Ці початкові фрази необхідні для відновлення гаманця в разі втрати доступу до коштів користувачів.

Завдяки новому модернізованому стандарту BIP39 резервне копіювання та відновлення криптогаманців тепер можливі без використання складних приватних ключів. Користувачам не потрібно запам’ятовувати довгі складні символи, щоб отримати доступ до своїх гаманців. Натомість початкова фраза відновлення гаманця складається зі звичайних слів, які запам’ятовуються. Стандарт BIP39 покращує криптобезпеку, пропонуючи більш зручний формат для фраз відновлення, які з меншою ймовірністю будуть введені неправильно.

BIP 32. Насіння:

xprv9vQTsqDKJ8Qm9iiYYUyUFHNaGtj4ST3ZA2qoxQyLApLWYbCQos batAuQbhxyuZbh9ZixGwkEN3FhzKZHzBXArxTzpHDYRnUcF2skxwEZ NJr

BIP 39. Приклад мнемонічної фрази:

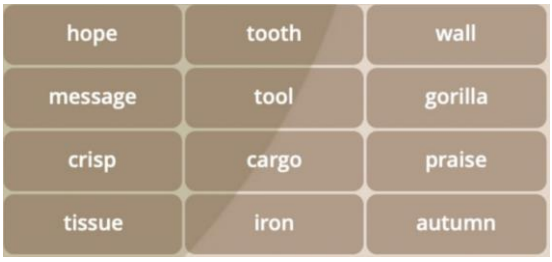


Рисунок 3.3. Мнемонічна фраза

Погодьтеся, що запам’ятати 12 слів набагато легше, аніж довільний набір символів як у BIP 32.

# РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

## 4.1. **Огляд використаних програмних засобів**

В процесі виконання кваліфікаційної роботи було використано наступний перелік програмних засобів

- Мова програмування Python
- Aiogram – сучасний та повністю асинхронний фреймоврк для розробки чат-ботів Telegram Bot API на Python 3.8. Асинхронність дозволяє сотням користувачів взаємодіяти з ботом без суттєвих затримок та гарантує високу швидкість обробки запитів на відміну від синхронних аналогів [1].
- Середовище Telegram та Telegram API. Хоча методи шифрування та прозорість Telegram були предметом критики, додаток для обміну повідомленнями користується популярністю через те, що він вважається більш приватним, ніж його популярні аналоги, що важливо при роботі з криптовалютою. Низька вартість розробки боту також вплинула на вибір цього середовища.
- Веб-фреймворк Django на разі є дуже популярним серед розробників завдяки зручності створення веб-додатків та API.
- Docker. Власне ядро докера дозволяє запустити практично будь-який додаток, безпечно ізольований у контейнері. При цьому не потрібно встановлювати ці додатки локально на комп’ютер. Всі контейнери запускаються лише однією командою.
- СУБД PostgreSQL для обробки та зберігання даних.
- Bitcoin Core та Geth (Go Ethereum) – клієнти для взаємодії з мережами біткоїну та етеріуму відповідно.

## 4.2. **Налаштування мережі біткоїну та етеріуму**

Для того, щоб під'єднатися до мережі біткоїну або етеріуму необхідно розгорнути вузол для обох мереж на локальному комп’ютері. Однак в процесі з’єднання вузла з мережею потрібно завантажити повну копію

блокчейну за всі роки, а це сотні гігабайтів. Більш того, до блокчейну додаються нові блоки щосекунди, що вимагає постійного розширення простору пам'яті. На допомогу розробникам блокчейн приходять тестові мережі – неповні копії блокчейну, які дозволяють виконувати ті ж самі дії над біткоїном або етеріумом.

Розгортання вузлів відбувається за допомогою програмного засобу Docker, що дозволяє запускати та виконувати програмні модулі ізольовано від локального комп'ютеру розробника – в так званих контейнерах. [3]

Контейнер для тестової мережі біткоїну завантажує образ bitcoind – Bitcoin Core Daemon – клієнт, який забезпечує процес створення та з'єднання локального комп'ютеру з мережею біткоїн. Наведемо програмний код нижче. [4]

```
bitcoin-core:
  image: kylemanna/bitcoind:latest
  ports:
    - 8333:8333
    - 8332:8332
  volumes:
    - ./services/bitcoin-core/bitcoin.conf:/bitcoin/.bitcoin/bitcoin.conf
    - ./services/bitcoin-core/bitcoin-core:/bitcoin/
```

Для етеріуму я також використовую докер-контейнер для взаємодії з клієнтом, який завантажує відповідний образ по аналогії з біткоїном. Geth (go-ethereum) — це реалізація Go Ethereum — шлюзу в децентралізовану мережу, написаний на мові програмування від компанії Google – Go [5].

Наведемо його код.

```
geth:
  image: ethereum/client-go:v1.10.13
  restart: unless-stopped
  ports:
   - "30303:30303"
   - "30303:30303/udp"
   - "8545:8545"
   - "8546:8546"
  volumes:
   - ./services/geth-docker/:/geth/
```

```
  - ./common_folder:/keystore
 stop_signal: SIGINT
 stop_grace_period: 2m
 command:
  - --syncmode=fast
  - --rinkeby
  - --http
  - --http.api
  - "personal,eth,net,web3"
  - --http.addr=0.0.0.0
  - --http.vhosts=*
  - --http.corsdomain=*
  - --keystore
  - "/keystore"
  - --datadir
  - "/chaindata"
  - --allow-insecure-unlock
 logging:
  driver: "json-file"
  options:
    max-size: "2m"
    max-file: "10"
```

Однак одних лиш контейнерів не достатньо для написання телеграм-бота. Необхідно забезпечити взаємодію вузлів із програмною мовою Python, а отже з головним сервером для бота.

Для біткоїну було реалізовано взаємодія з біткоїн-клієнтом за допомогою протоколу JSON-RPC (Додаток А.2), а етеріуму за допомогою протоколу Web3 (Додаток А.3). Було використано кастомний набір коду для спрощення спілкування між вузлами та головним сервером (Додаток А.1)

### 4.3. **Розроблення головного серверу для впровадження API**

Телеграм бот не може власноруч спілкуватися з вузлами криптовалют, для цього я прийняла рішення розробити веб-сервер. Сервер написаний на мові Python з використанням веб-фреймоврку Django, що дає змогу реалізувати API - це абревіатура «Application Programming Interface «(інтерфейс програмування додатків, програмний інтерфейс програми). База даних програми наведена в Додатку Г. API тут виступає в ролі «мосту» між телеграм ботом і вузлами біткоїна та етеріума. [2]

Сутність API являє собою набір url-адрес, за допомогою яких бот зможе отримувати, створювати або редагувати інформацію за допомогою HTTP - протоколу передачі даних, що використовується в комп'ютерних мережах. В середовищі Django був створений окремий додаток з назвою «user», який показує наступний список url-адрес.

```
from django.urls import path
from . import views
from . import eth_views
from . import btc_views


app_label = 'user'
urlpatterns = [

    path('delete-wallet/', views.DeleteWallet.as_view()),
    path('client-update/', views.UpdateClientView.as_view()),
    path('client/', views.ClientInfo.as_view()),

    # ETH
    path('create-eth-address/', eth_views.CreateETHAddress.as_view()),
    path('eth-wallet-create/', eth_views.CreateETHWalletView.as_view())
    path("get-eth-address/", eth_views.GetETHAddress.as_view()),
    path("get-eth-balance/", eth_views.GetETHBalance.as_view()),
    path('mnemonic/', views.GetWalletMnemonic.as_view()),

    # BTC
    path("get-btc-address/", btc_views.GetBTCAddress.as_view()),
    path('create-btc-wallet/', btc_views.CreateBTCWallet.as_view()),
    path('get-btc-balance/', btc_views.GetBTCBalance.as_view()),

]
```

Запуск серверу, бази даних та боту я реалізувала за допомогою докер-контейнерів (Додаток Б.1 та Б.2).

Розберемо приклад формування url-адреси для реалізації API. В налаштуваннях джанго необхідно вказати порт та домен, де буде розгорнутий сервер. Наприклад, для розробки я використовувала хост докер-контейнеру «server» та порт 8000. Таким чином, завершальний вигляд адреси виглядає так: хост:порт/ім’я url-адреси.

Розглянемо обробку http - запиту до головного серверу за адресою http://server:8000/api/v1/user/get-btc-balance/. За обробку адреса в джанго використовуються представлення або views – це своєрідні функції, які

повертають http-відповідь. Для адреси /get-btc-balance/ була розроблена наступна функція:

```
class GetBTCBalance(views.APIView):
    serializer_class = serializers.GetBTCBalanceSerializer

    def post(self, request, *args, **kwargs):
        # Перевіряємо отримані дані від бота на валідність
        serializer_data = self.serializer_class(data=request.data)
        if serializer_data.is_valid():
            wallet_id = serializer_data.validated_data.get('wallet_id')
            # Отримуємо екземпляр класу для взаємодії з вузлом біткоїна
            service = BitcoinService()
            response, status = service.get_balance(wallet_id=wallet_id)
            # Якщо бікоїн не відповідає повертаємо код помилки 400
            if not status:
                return Response(response, status=400)
            # Повертаємо успішний статус 200 та значення балансу від біткоїну
            return Response({**response, "code": 0}, status=200)
        # Повертаємо помилку 400 якщо дані гаманця були вказані невірно
        if "wallet_id" in serializer_data.errors:
            return Response({"code": 2}, status=400)
        # Повертаємо помилку 400 якщо отримані дані не валідні
        return Response(serializer_data.errors, status=400)
```

В середині функції був створений екземпляр класу BitcoinService для взаємодії з вузлом біткоїну. В залежності від даних, отриманих від бота, представлення повертає або успішний результат, або результат помилки.

Для валідації типів даних, отриманих від телеграм-бота, джанго використовує класи-серіалізатори, які наслідуються від вбудованого батьківського класу Serializer. В нашому випадку ми валідуємо поле wallet_id, яке обов’язково повинно бути типу char.

```
class GetBTCBalanceSerializer(serializers.Serializer):
    wallet_id = serializers.CharField()
```

По аналогії було створено інші адреси та представлення для можливості здійснення http-запитів від телеграм бота.

## 4.4. **Написання автономного бота в середовищі Телеграм**

Запустимо телеграм бота. Для цього необхідно створити керівний модуль, що використовує модуль конфігурації бота bot_app.py за допомогою фреймоворку aiogram (Додаток В.1). [1]

```
import asyncio

from aiogram.utils.executor import start_webhook
from aiogram.dispatcher.middlewares import BaseMiddleware
from aiogram import types

from bot_app import dp, bot
from config import *
import aiogram_logging
from services.check_client_info import check_fullname
from services.data_api_handler import post_info
import handlers


from services.set_commands import set_default_commands
from services.remove_extra_inline import remove_inline_markup
from services.throttling_middleware import ThrottlingMiddleware

import logging
from logging import handlers as logging_handlers


async def main_long_polling():
    """
    Запускає телеграм бот в режимі long-polling
    """
    print("Starting bot")

    try:
        await set_default_commands(bot)
        await dp.start_polling()
    finally:
        await dp.storage.close()
        await dp.storage.wait_closed()
        await bot.session.close()


    try:
        asyncio.run(main_long_polling())
    except (KeyboardInterrupt, SystemExit):
        print("Bot stopped!")


async def on_startup(dp):
    logging.warning(
        'Starting connection. ')

    await set_default_commands(bot)
    await bot.set_webhook(WEBHOOK_URL, drop_pending_updates=True)
```

```
async def on_shutdown(dp):
    logging.warning('Bye! Shutting down webhook connection')
```

Для запуску бота було використано докер-контейнери (Додаток В.2)

В програмному додатку телеграм-бота було створено конфігураційний файл config.py з переліком всіх назв кнопок, текстів повідомлень та інше (Додаток В.3)

Першим кроком взаємодії з телеграм ботом завжди будуть команди. Найпершою командою є /start. В боті було реалізовано таких список команд, які можна побачити після того, як ввести символ слешу.

```
async def set_default_commands(bot: Bot):
    """
    - Створює меню команд
    """
    return await bot.set_my_commands(
        commands=[
            BotCommand('/start', COMMAND_START),
            BotCommand('/help', COMMAND_HELP),
            BotCommand('/wallet', COMMAND_WALLET),
            BotCommand('/info', COMMAND_INFO),
            BotCommand('/settings', COMMAND_SETTINGS),
            BotCommand('/support', COMMAND_SUPPORT),
        ],
        scope=BotCommandScopeDefault()
    )
```

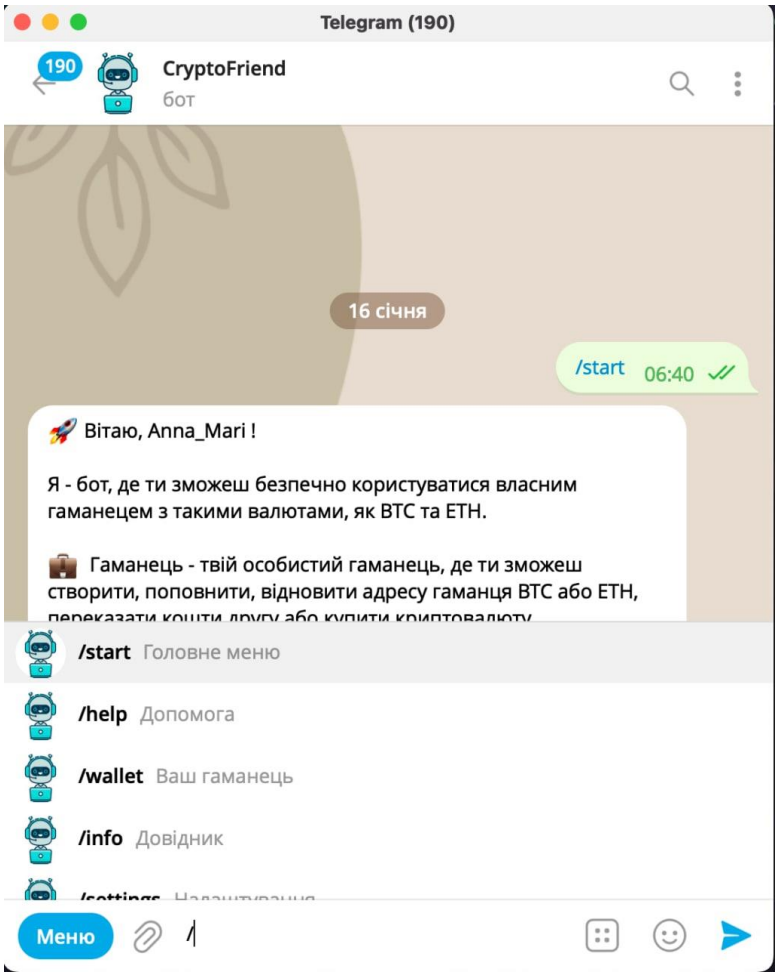


Рисунок 4.1. Меню команд боту

Для обробки кожної команди необхідно написати відповідну функцію. Наприклад, для команди /start було реалізовано наступний код.

```
@dp.message_handler(commands=['start'], state='*')
async def send_welcome(message: types.Message, state: FSMContext):
    """
     - Обробник викликається після введення /start
     - Бот надсилає інформацію про нового клієнта на сервер
    """
    await state.finish()
    full_name = await check_fullname(message)
    username = message.from_user.username
    await post_info(url=BOT_CLIENT_UPDATE_URL,
                    data={'user_id': message.from_user.id, 'username':
username,
                          'full_name': full_name, 'language':
message.from_user.language_code})
    new_user = username if username else full_name
    message_greetings = translate(text=MESSAGE_GREETINGS % new_user,
user_id=message.chat.id)
    await message.answer(text=message_greetings,
reply_markup=main_kb.get_main_keyboard(message.chat.id))
```

Після натискання команди /start одразу виникає повідомлення від бота та головне меню, що складається з кнопок Гаманець, Довідка, Налаштування та Підтримка.

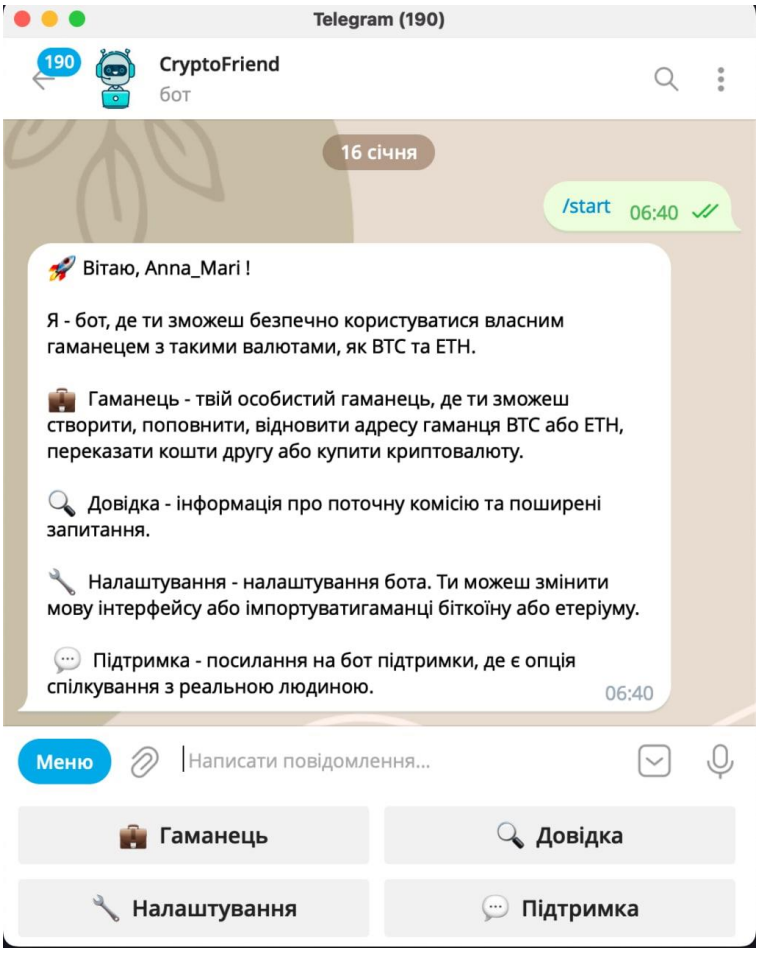


Рисунок 4.2. Головне меню боту

Програмна реалізація кнопок головного меню виглядає наступним чином. Спочатку реалізується функція, що повертає об’єкт клавіатури типу ReplyKeyboardMarkup, який потім вказується при виклику команди /start.

```
# Main ReplyKeyboard
def get_main_keyboard(user_id):
    main_keyboard = ReplyKeyboardMarkup(resize_keyboard=True, row_width=2)
    main_buttons_names = [BUTTON_WALLET,
                          BUTTON_INFO,
                          BUTTON_SETTINGS,
                          BUTTON_SUPPORT]
    main_buttons = []
    for button in main_buttons_names:
        # Переклад назви кнопок на обрану мову користувача
        main_buttons.append(translate(text=button, user_id=user_id))
    return main_keyboard.add(*main_buttons)
```

Припустимо користувач хоче створити адресу з валютою біткоїн. Йому необхідно натиснути на кнопку «Гаманець» та «Поповнити». Після цього необхідно обрати валюту.

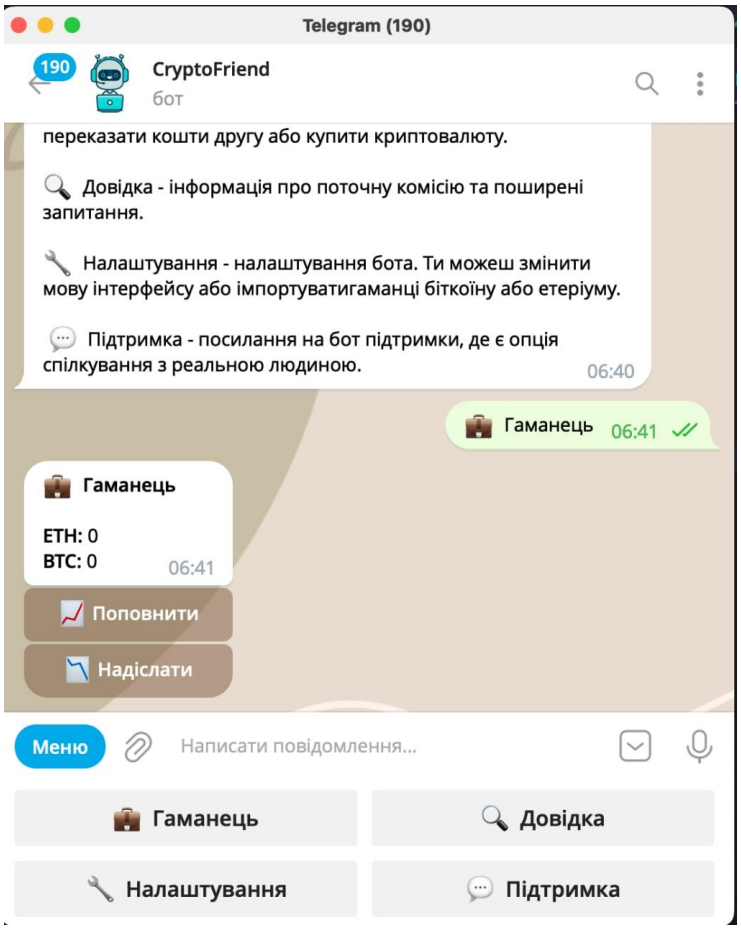


Рисунок 4.3. Реакція на натискання кнопки «Гаманець»

Список валют для вибору реалізується за допомогою іншого типу кнопок в телеграм – InlineKeyboardMarkup.

```
deposit_menu_cb = CallbackData('post', 'value')

deposit_menu_keyboard = InlineKeyboardMarkup(row_width=1)
deposit_buttons = [
    InlineKeyboardButton(text='BTC',
callback_data=deposit_menu_cb.new(value='BTC_deposit')),
    InlineKeyboardButton(text='ETH',
callback_data=deposit_menu_cb.new(value='ETH_deposit')),
]
deposit_menu_keyboard.add(*deposit_buttons)
```

Вигляд Inline-кнопок BTC та ETH можна побачити на малюнку.

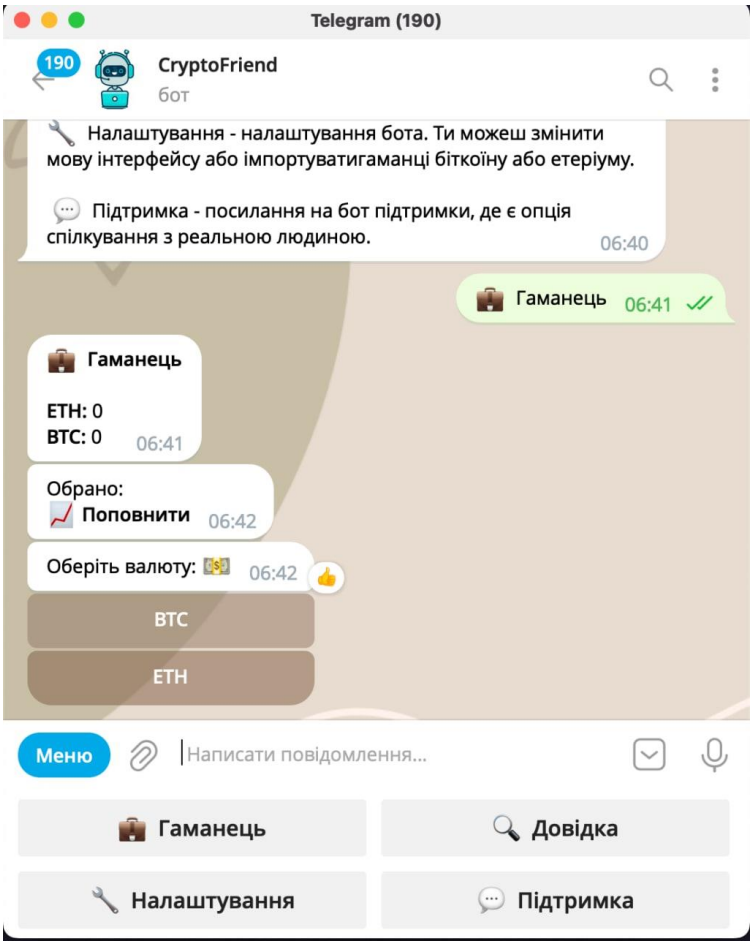


Рисунок 4.4. Зображення Inline-кнопок для вибору валюти

У випадку, якщо адреса не знайдена в базі даних серверу, бот пропонує її створити. Також є можливість імпортувати відповідну адресу, якщо та була створена раніше за межами боту.

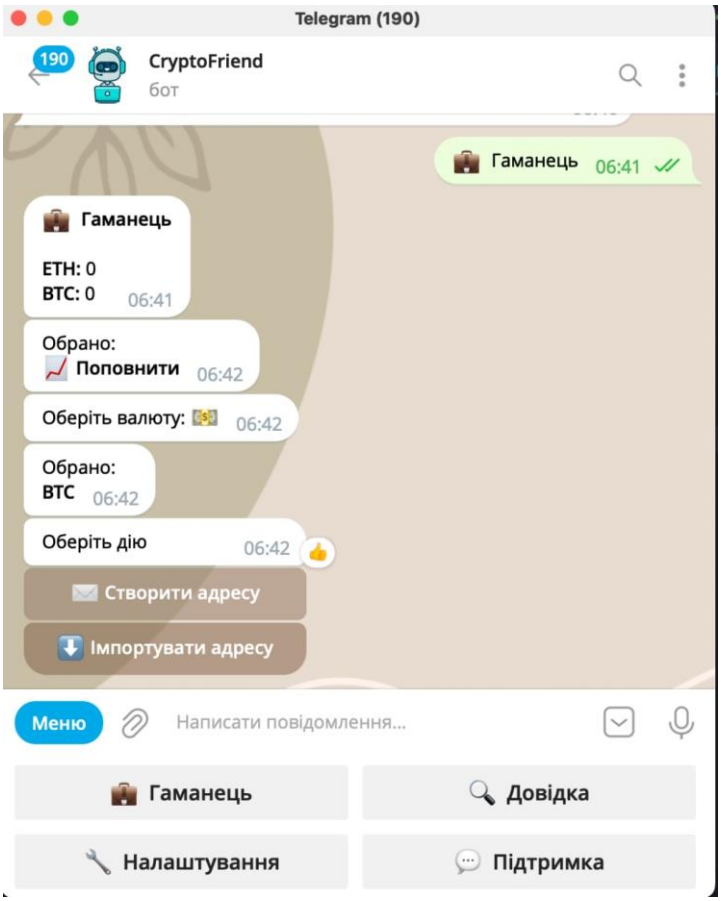


Рисунок 4.5. Inline-кнопки для створення та імпорту адреси

Після введення всіх даних бот робить пост-запит до головного серверу разом із даними від користувача. Сервер передає ці дані до мережі біткоїну, яка надає одразу створену адресу та приватний ключ.

Однак пароль це досить конфіденційні дані, які ніхто, навіть бот, не повинен зберігати. Для цього в бібліотеці aiogram існує кінцевий автомат (FSM) - це абстрактна машина, яка може перебувати в одному зі скінченної кількості станів у будь-який момент часу. Кінцевий автомат може переходити з одного стану в інший у відповідь на деякі вхідні дані; перехід з одного стану в інший називається переходом. Перевагою даної технології є те, що кожен стан – окрема змінна, яка може зберігати будь-яку інформацію в межах однієї машини. Після завершення життєвого циклу машини всі дані, що знаходилися раніше в її станах, стираються.

Простими словами, кінцевий автомат – структура зберігання тимчасових даних, а отже інформацію не потрібно зберігати в базі даних. Зауважимо, що всі конфіденційні дані бот не зберігає.

Програмна реалізація кінцевого автомату під час здійснення транзакції в мережі біткоїну вимагає використання програмного засобу redis в ролі тимчасового сховища даних. Запуск редіс відбувається в межах докер-контейнерів (Додаток В.5) [1]

Ініціація машини стану в середовищі aiogram.

```
class WithdrawState(StatesGroup):
    receiver = State()  # Адреса отримувача біткоїнів
    password = State()  # Пароль до криптогаманця відправника
    amount = State()  # Кількість біткоїнів
    confirm = State()  # Підтвердження транзакції True/False
```

Використання в межах транзакції.

```
@dp.callback_query_handler(ChatTypeFilter(types.ChatType.PRIVATE),

withdraw_kb.withdraw_menu_cb.filter(value='BTC_withdraw'), state='*')
async def create_btc_address(call: types.CallbackQuery, callback_data: dict,
state: FSMContext):
    """
    - Handler is called after pressing "Withdraw" -> "BTC" button
    - Handler open state by asking password to BTC wallet
    - Handler adds to the state memory current currency
    """
    await call.message.edit_reply_markup(reply_markup=None)
    message_selected = translate(text=MESSAGE_SELECTED,
user_id=call.message.chat.id)
    await call.message.answer(f'{message_selected}\n' + bold('BTC'),
parse_mode=ParseMode.MARKDOWN)

    # Встановлення стану пароль
    await WithdrawState.password.set()

    new_state = dp.current_state()
    await new_state.update_data(currency='BTC')
    message_enter_password = translate(text=MESSAGE_ENTER_PASSWORD,
user_id=call.message.chat.id)
    await call.message.answer(message_enter_password,

reply_markup=withdraw_kb.get_forgot_password_keyboard(call.message.chat.id))



@dp.message_handler(ChatTypeFilter(types.ChatType.PRIVATE),
state=WithdrawState.password)
```

```
async def enter_password(message: types.Message, state: FSMContext):
    """
    - Handler is called after entering password to ETH or BTC account
    - Handler open state by asking receiver address to make transaction
    - Deletes the message with the password
    """
    await state.update_data(password=message.text)

    # Встановлення стану отримувач
    await WithdrawState.receiver.set()

    message_enter_receiver_address =
translate(text=MESSAGE_ENTER_RECEIVER_ADDRESS, user_id=message.chat.id)
    await message.answer(message_enter_receiver_address)
    await message.delete()
```

По аналогії відбувається виклик станів сума та підтвердження транзакції (Додаток В.6)

В кінці обов’язково потрібно завершити роботу кінцевого автомату.

```
@dp.callback_query_handler(lambda call: call.data == BUTTON_TRANSACTION_SEND,
state=WithdrawState.confirm)
async def send_transaction(call: types.CallbackQuery, state: FSMContext):
    """
    - Handler is called after pressing on button "Send" transaction
    - Makes transaction via django API
    - Returns transaction hash
    """
    await call.message.edit_reply_markup(reply_markup=None)
    state_data = await state.get_data()
    address_receiver, password, amount = state_data.get('receiver'),
state_data.get('password'), state_data.get('amount')
    currency = state_data.get('currency')
    if currency == 'BTC':
        url = BOT_MAKE_TRANSACTION_BTC_URL
    else:
        url = BOT_MAKE_TRANSACTION_ETH_URL

    response = await post_info(url=url, data={'user_id':
call.message.chat.id, 'password': password,
                                              'address_receiver':
address_receiver, 'amount': amount,
                                              'currency': currency})
    code, tx_hash, error = response.get('code'), response.get('result'),
response.get('error')

    if code == 0:
        await call.message.answer(f'Transaction hash\n{tx_hash}')
    elif error:
        await call.message.answer(f'Error:\n*{error}*',
parse_mode=ParseMode.MARKDOWN)
    else:
        await call.message.answer(f'Error:\n*{ERROR_CODES.get(code)}*',
parse_mode=ParseMode.MARKDOWN)
```

```
    # Закриття стану кінцевого автомату
    await state.finish()
```

Бот успішно підтримує функцію зміни мови інтерфейсу за допомогою Google API. По замовчуванню інтерфейс відображається згідно з мовою, яку користувач встановив в налаштуваннях телеграму. Це відбувається за допомогою API запитів до головного джанго-серверу наступним чином. Після натискання команди /start бот передає серверу інформацію про користувача, в тому числі мову.

```
class UpdateClientView(views.APIView):
    permission_classes = (permissions.AllowAny, )
    serializer_class = serializers.ClientSerializer

    def post(self, request):
        serializer = self.serializer_class(data=request.data)
        # Валідація отриманих даних
        if serializer.is_valid():
            data = serializer.validated_data
            phone, email, username = data.get('phone'), data.get('email'),
data.get('username')
            full_name, language, user_id = data.get('full_name'),
data.get('language'), data.get('user_id')

            # Створення або редагування запису про користувача бота в базі
даних
            Client.objects.update_or_create(user_id=user_id,
                                            defaults={'phone': phone,
'email': email, 'username': username, 'full_name': full_name, 'language':
language})
            return Response({'result': True, "code": 0}, status=200)
        return Response(serializer.errors, status=400)
```

Функція для перекладу тексту.

```
from googletrans import Translator
from config import BOT_GET_CLIENT_URL
import requests


def translate(text, user_id=''):
    # Отримання інформації про користувача з БД серверу
    client_info = requests.post(url=BOT_GET_CLIENT_URL, json={'user_id':
user_id})

    # Отримання значення мови користувача
    client_language = client_info.json()['language']

    # Функція повертає перекладений текст від Google Translate
```

```
    translator = Translator()
    return translator.translate(src='uk', dest=client_language,
text=text).text
```

В результаті інтерфейс боту буде мати такий вигляд.

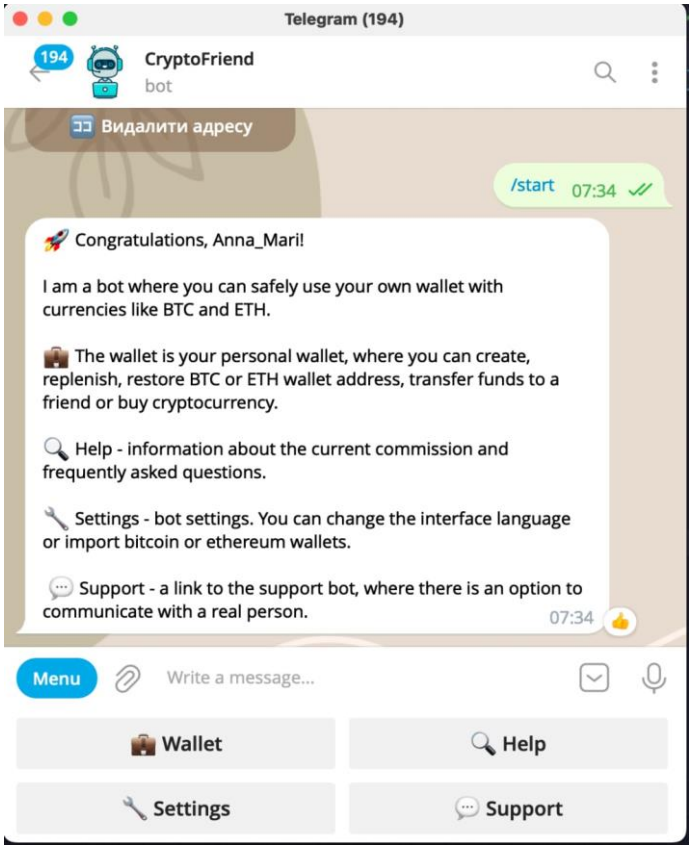


Рисунок 4.6. Інтерфейс англійською мовою

Якщо бот не впізнає повідомлення від користувача, то спрацьовує наступна функція.

```
@dp.message_handler(ChatTypeFilter(types.ChatType.PRIVATE), state='*',
content_types=ContentType.all())
async def unknown_message_reply(message: types.Message, state: FSMContext):
    """
     Викликається в разі невпізнання ботом повідомлення від користувача
    """
    await state.finish()
    message_unknown = translate(text=MESSAGE_UNKNOWN,
user_id=message.chat.id)
    await message.answer('/start ' + message_unknown)
```

Бот має захист від спаму повідомлень від користувача за допомогою класу BaseMiddleware – проміжне програмне забезпечення. Він не дозволяє користувачеві надсилати повідомлення до бота частіше, ніж за 1 секунду (Додаток В.7)


Рисунок 4.7. Захист від спаму

## 4.5. **Бот підтримки користувача**

В роботі було створено ще одного телеграм бота-підтримки, який здатен забезпечити користувачеві спілкування з реальною людиною.

Виникає логічне запитання: навіщо потрібно було створювати додаткового телеграм бота. Відповідь проста. Спілкування з реальною людиною виникає через групу в телеграмі, де знаходяться адміністратори боту – люди, які можуть відповісти на повідомлення користувача. Додавання основного боту до будь-якої групи в телеграмі небезпечно, оскільки відбувається обмін конфіденційних даних, тому таку можливість було вимкнено.

Користувач потрапляє до боту підтримки через головне меню основного боту.

Запуск та код боту заходиться Додатку Г.1 та Додатку Г.2


Рисунок 4.8. Взаємодія користувача з ботом підтримки

Повідомлення від боту відбувається за допомогою асинхронного пост запиту засобами Telegram API. Для цього необхідно знати токен основного боту та ідентифікатор створеної групи в телеграмі.

```
async def post_info(url, data):
    headers = await get_headers()
    async with aiohttp.ClientSession() as session:
        async with session.post(url=url, json=data, headers=headers) as resp:
            return await resp.json()



@dp.message_handler(state=AskQuestionState.question)
async def show_wallet(message: types.Message, state: FSMContext):

    await state.finish()
    await
post_info(url=f'https://api.telegram.org/bot{BOT_TOKEN}/sendMessage',
data={'chat_id': group_id, 'text': message.text})
```

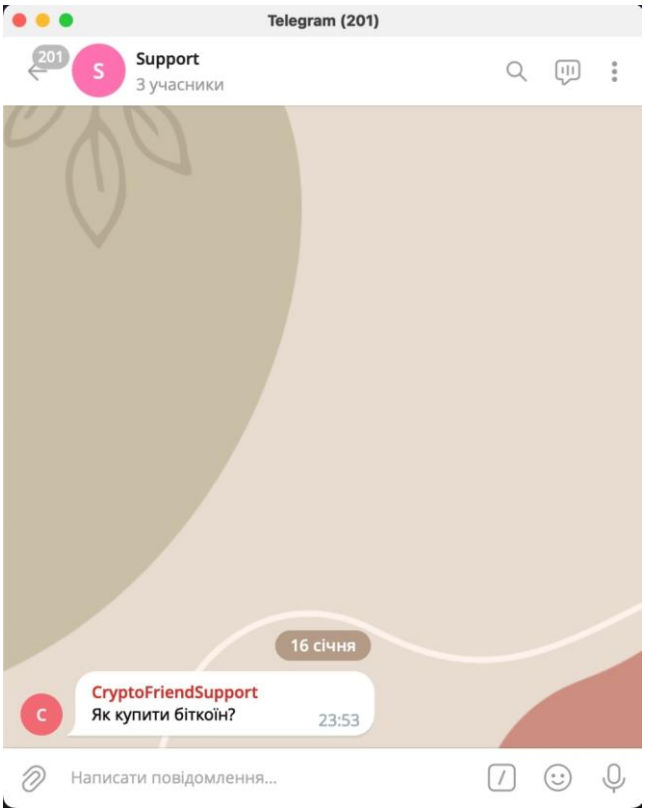


Рисунок 4.9. Відправка запитання від користувача до групи адміністраторів в телеграмі

## 4.6. **Веб-сторінка для адміністрування**

Однією з найпотужніших частин Django є автоматичний інтерфейс адміністратора. Він зчитує дані з таблиць бази даних, щоб забезпечити швидкий інтерфейс, де довірені користувачі можуть керувати вмістом.

Для того, щоб додати моделі (таблиці) до адмін-сторінки, необхідно впровадити наступний код в файлі admin.py кожного джанго-додатка в залежності від тих моделей, які потрібно побачити на сторінці.

```
from django.contrib import admin
from apps.applications.models import Currency, Client, ClientWallet,
InterfaceLanguage


class CurrencyAdmin(admin.ModelAdmin):
    search_fields = ('name',)
    list_display = ('name',)


class ClientAdmin(admin.ModelAdmin):
    list_display = (
```

```
        'user_id',
        'username',
        'full_name',
        'email',
        'created_at'
    )


class ClientWalletAdmin(admin.ModelAdmin):
    list_display = (
        'address',
        'client',
        'currency',
        'created_at'
    )


admin.site.register(Currency, CurrencyAdmin)
admin.site.register(Client, ClientAdmin)
admin.site.register(ClientWallet, ClientWalletAdmin)
admin.site.register(InterfaceLanguage)
```

З рештою адміністративна сторінка виглядає так.

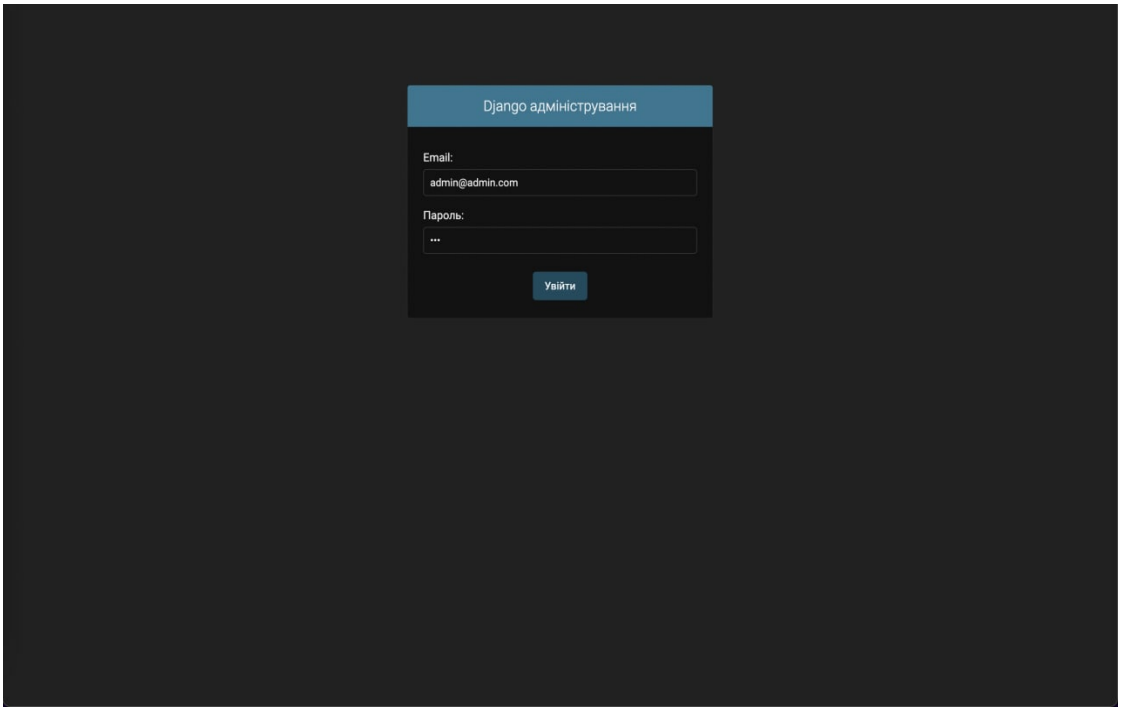


Рисунок 4.10. Сторінка логіну до адміністративної сторінки

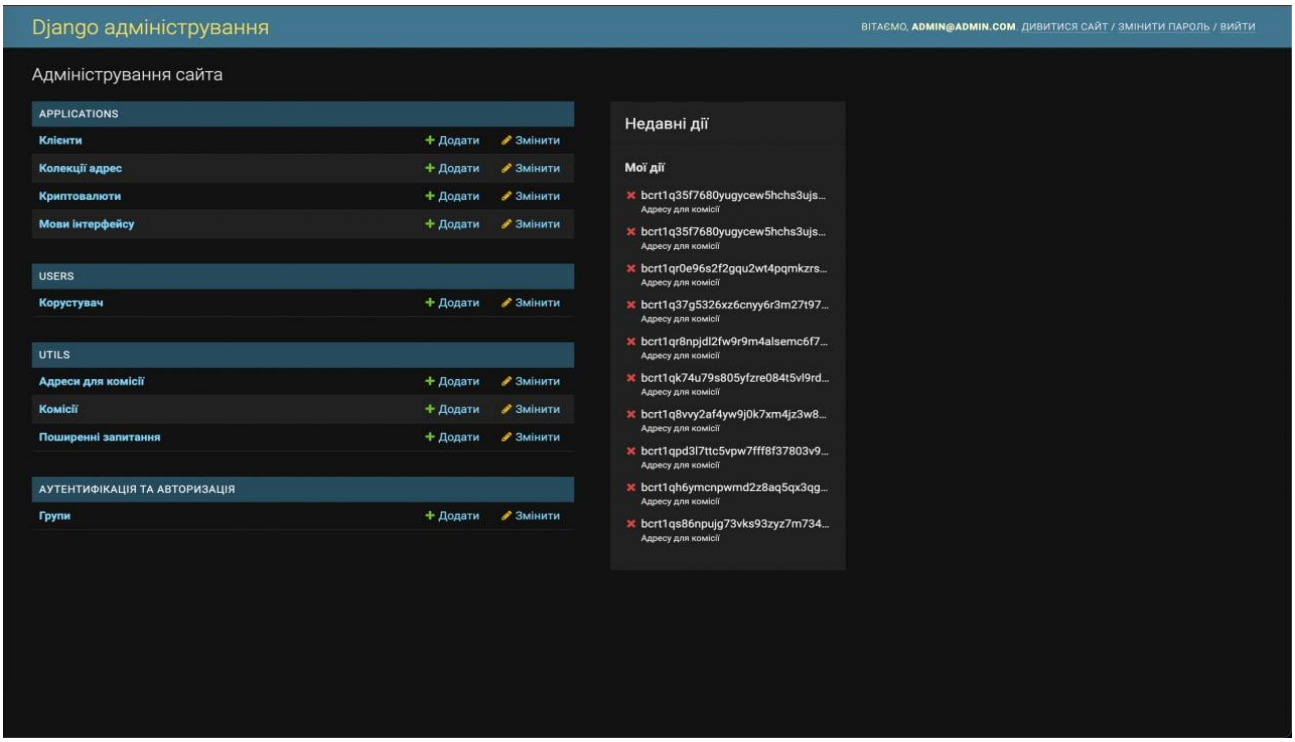


Рисунок 4.11. Сторінка для перегляду та редагування таблиць

Приклад того, як адміністратор може переглядати та редагувати таблиці в БД.

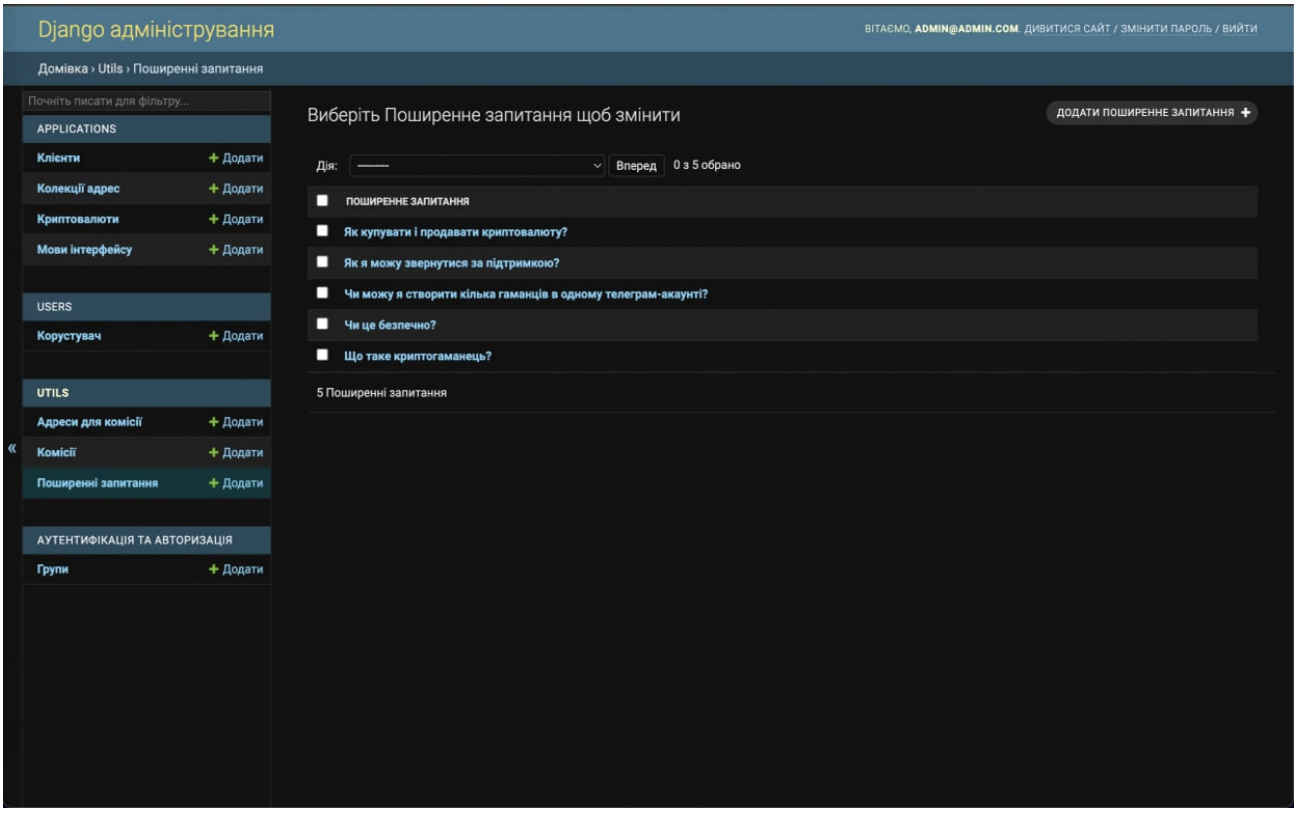

Рисунок 4.12.  Сторінка для перегляду та редагування таблиць

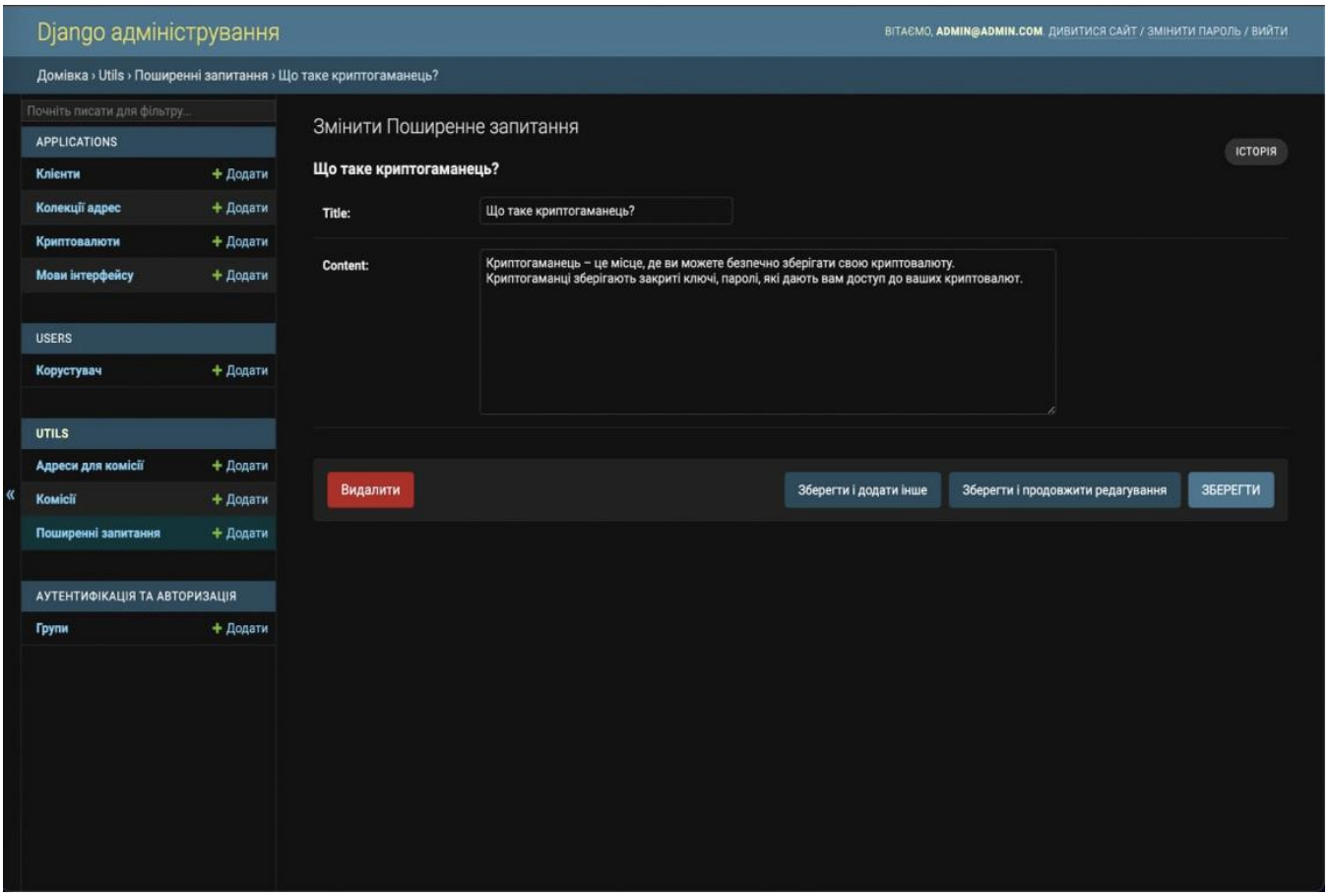


Рисунок 4.13.  Сторінка для перегляду та редагування таблиць

# РОЗДІЛ 5. ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧА

Після натискання на команду /start одразу виникає повідомлення від бота та головне меню, що складається з кнопок Гаманець, Довідка, Налаштування та Підтримка.

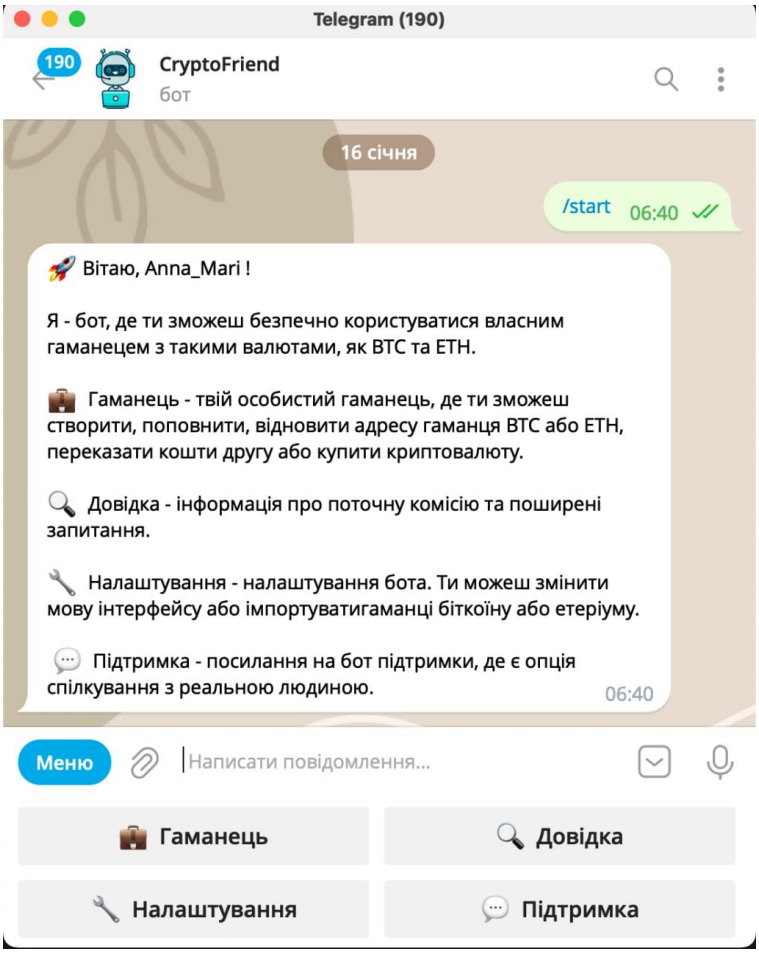


Рисунок 5.1. Головне меню бота

В боті реалізована додаткове меню команди, які можна побачити після того, як ввести символ слешу.

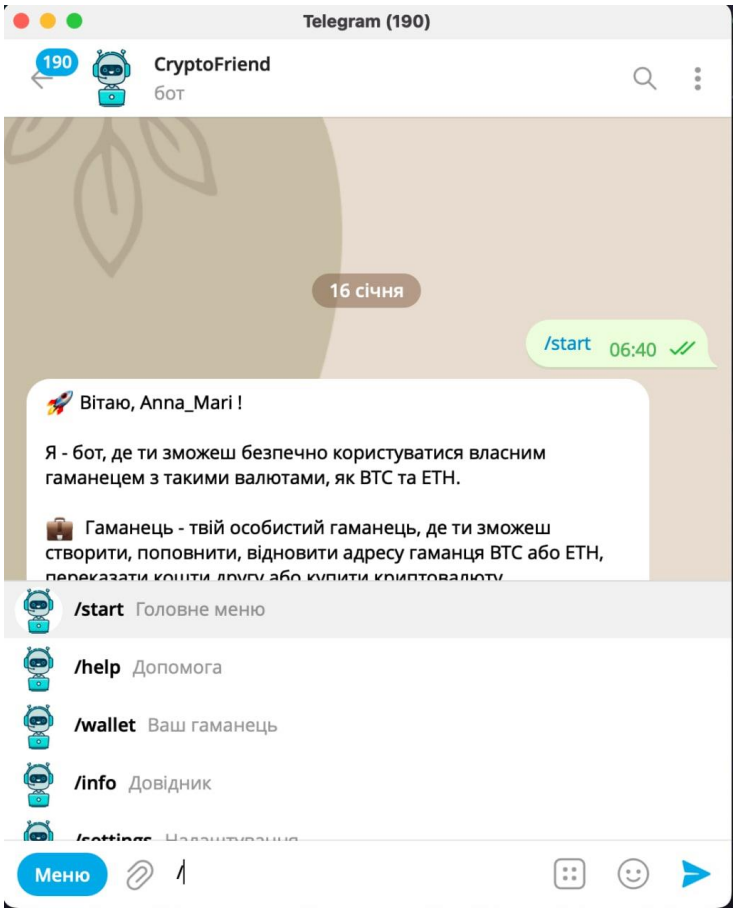


Рисунок 5.2. Меню команд

Якщо користувач хоче створити адресу з валютою біткоїн, йому необхідно натиснути на кнопку «Гаманець» - «Поповнити». Після цього обрати бажану валюту.

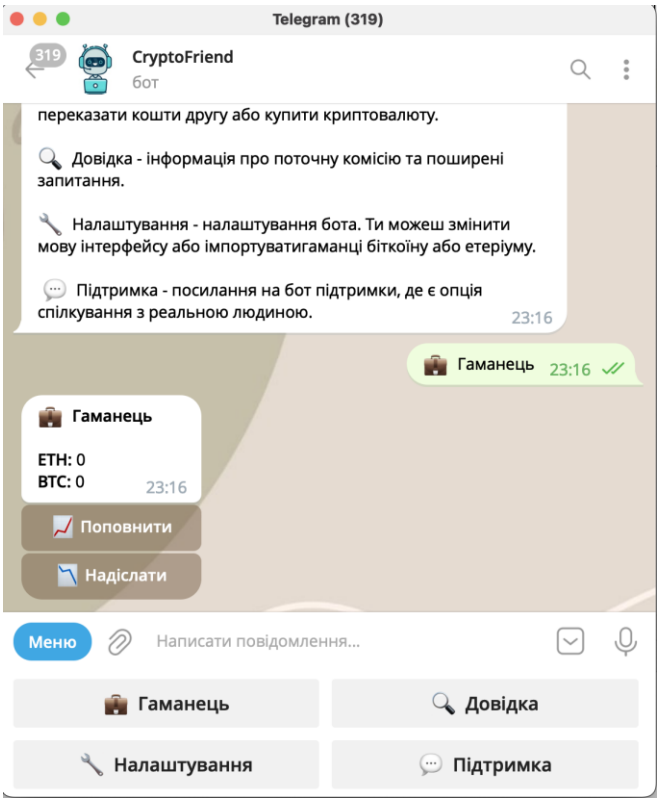


Рисунок 5.3. Реакція на натискання кнопки «Гаманець»

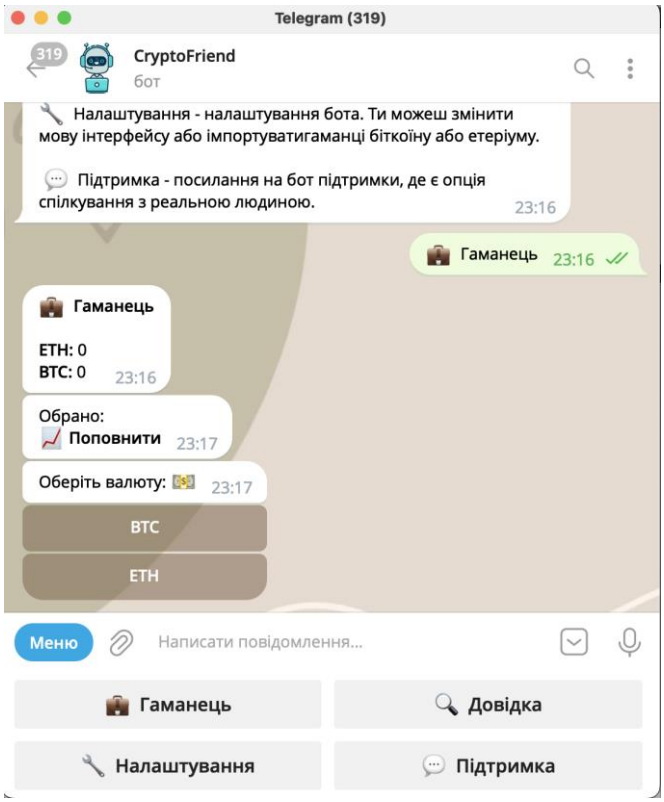


Рисунок 5.4. Вибір валюти

У випадку, якщо адреса не знайдена в базі даних серверу, бот пропонує її створити. Також є можливість імпортувати відповідну адресу, якщо та була створена раніше за межами боту.

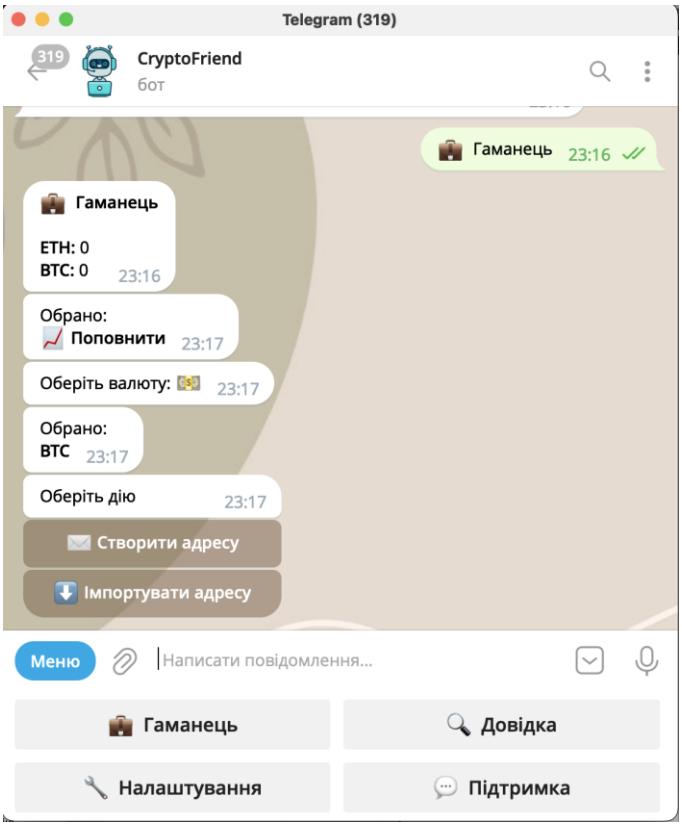


Рисунок 5.5. Можливість створити або імпортувати адресу

Бот передбачає можливість генерації надійного паролю. У разі, якщо користувач не задоволений наданим паролем, можна повторити генерацію знов або відмінити дію, або написати звичайним текстовим повідомленням пароль.

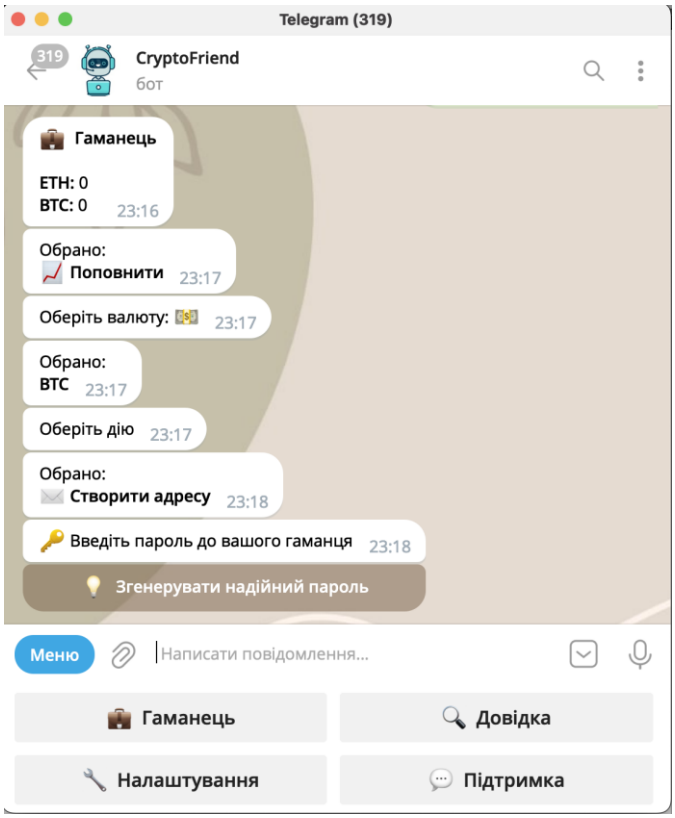


Рисунок 5.6. Можливість згенерувати надійний пароль

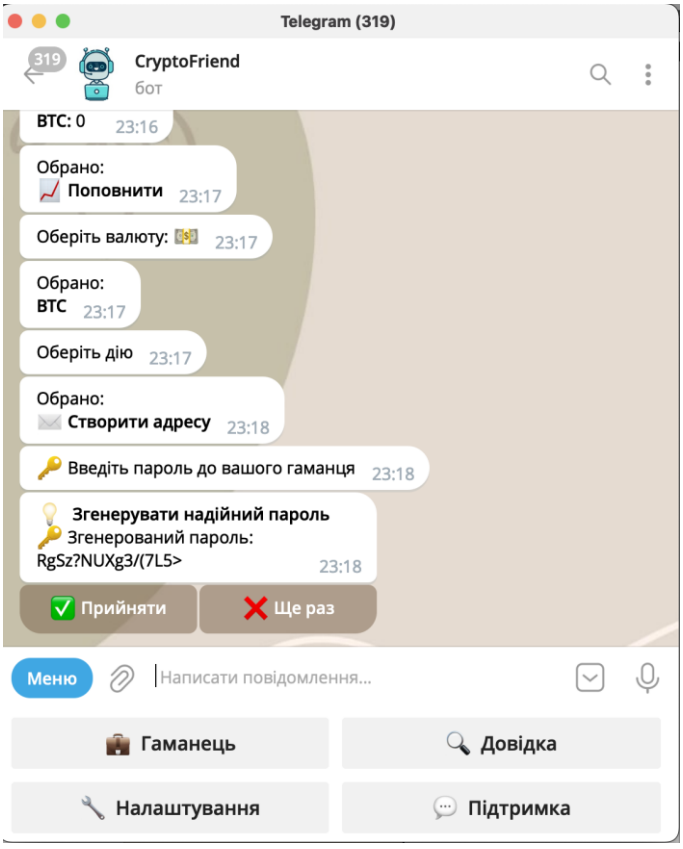


Рисунок 5.7. Підтвердження згенерованого паролю

Після введення всіх даних мережа біткоїн віддає користувачу приватний ключ до та публічну адресу до його біткоїн-гаманця. Бот попереджає, що приватний ключ не можна нікому віддавати.

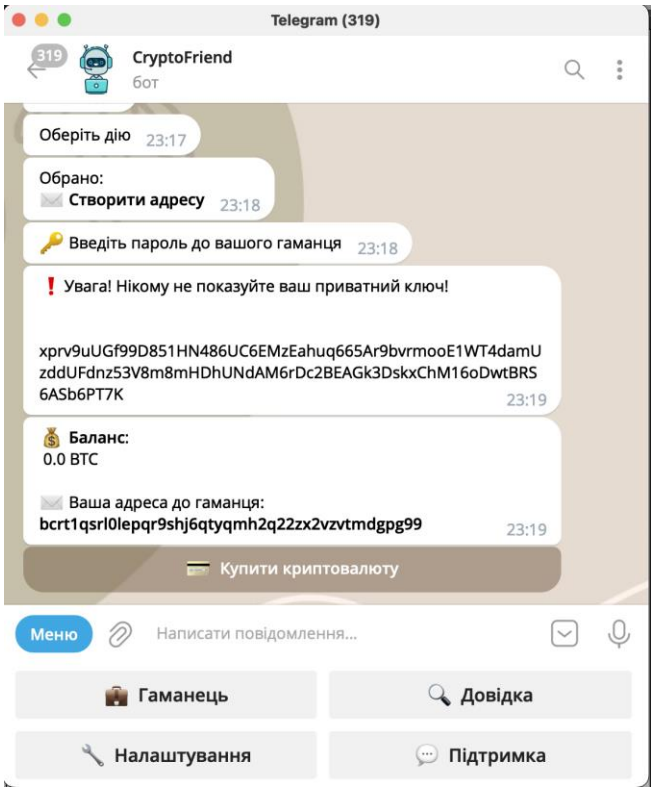


Рисунок 5.8. Результат створеної адреси в біткоїні

У користувача є можливість купити криптовалюту за звичайні, фіантні, валюти, натиснувши на кнопку «Купити криптовалюту». Після цього бот відправить вас до стороннього веб-сайту, наразі bitcoin.com, де можна купити необхідну кількість електронних монет. За покупку криптовалюти бот відповідальність не несе.

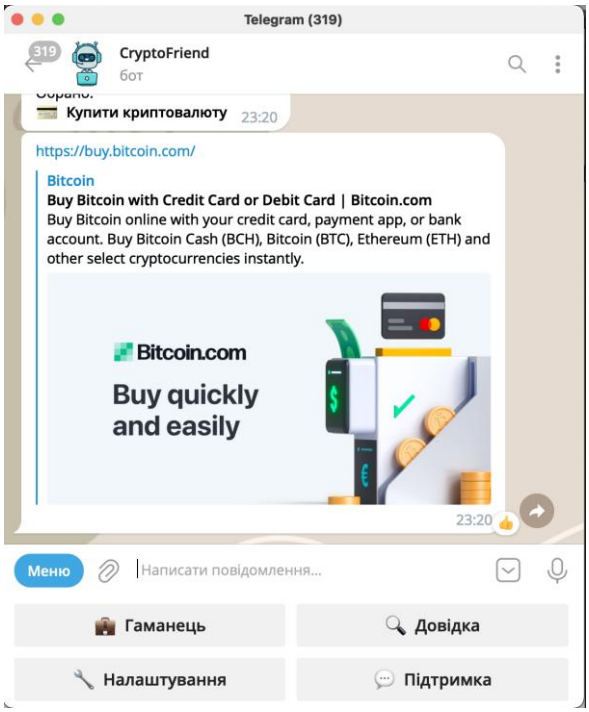


Рисунок 5.9. Посилання на сайт для придбання криптовалюти

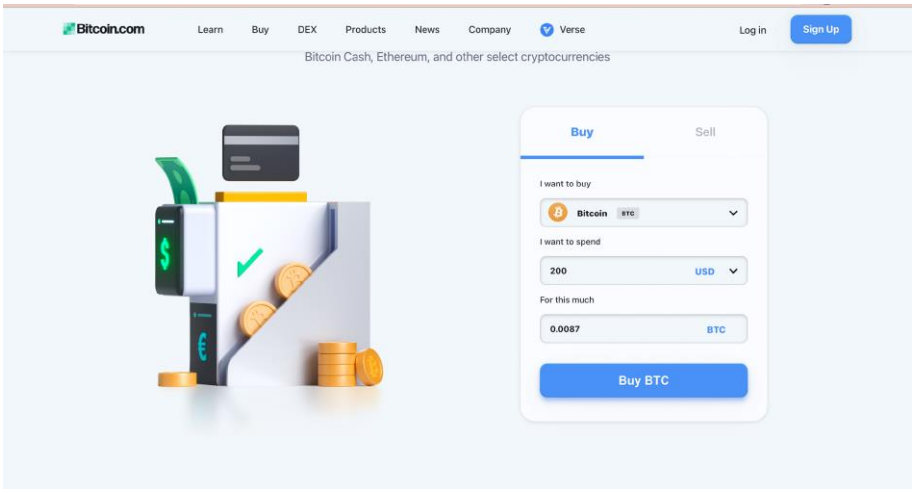


Рисунок 5.10. Приклад сайту для купівлі біткоїну

Аналогічні дії зі створенням адреси етеріуму. Користувачу необхідно натиснути на таку послідовність кнопок «Гаманець» - «Поповнити» - «Створити адресу» (Додаток Ж.1). Однак вигляд приватного ключа етеріуму відрізняється від біткоїну.

Замість рядка з набором символів та букв, мережа етеріуму відображає його послідовність 12 слів, мнемонічну фразу.

Після отримання паролю, бот надає користувачу його мнемонічну фразу, яку необхідно підтвердити, обравши правильний порядок слів на ітерактивній клавіатурі.

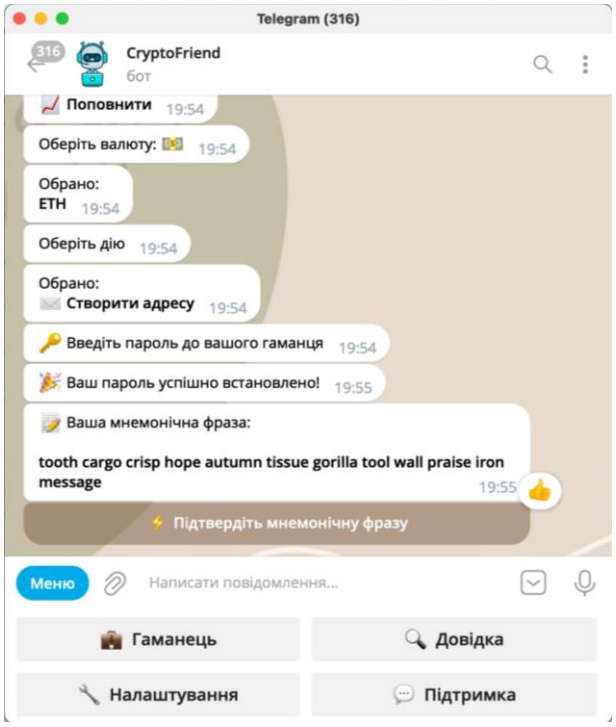


Рисунок 5.11. Кнопка для підтвердження мнемонічної фрази

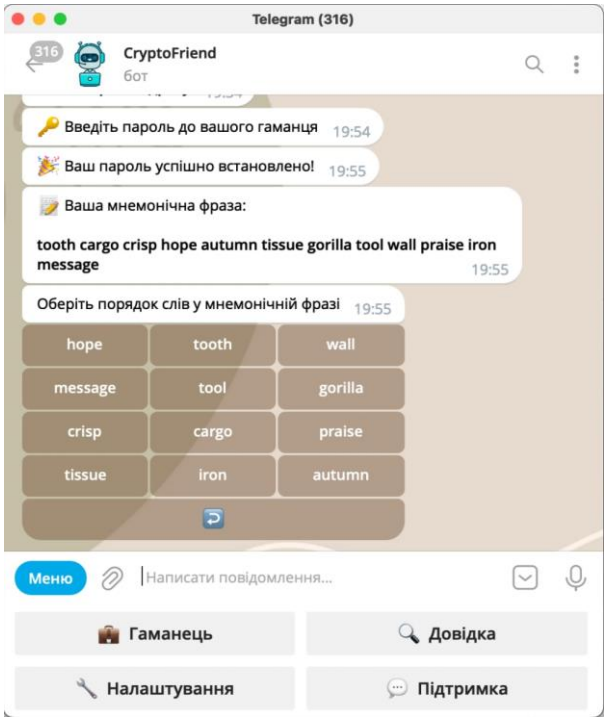


Рисунок 5.12. Ітерактивна клавіатура мнемонічної фрази

Інтерактивна працює таким чином, що після натискання обраного слово воно зникає, а решта не натиснутих слів лишається. Якщо користувач помилився та ввів невірну послідовність слів виникає відповідне повідомлення, яке водночас надає можливість повторити дію.

Мнемонічну фразу необхідно зберегти довільним чином, але в надійному місці, оскільки вона – ключ до електронного гаманця. Утративши її всі гроші гаманця зникнуть назавжди без можливості відновлення.

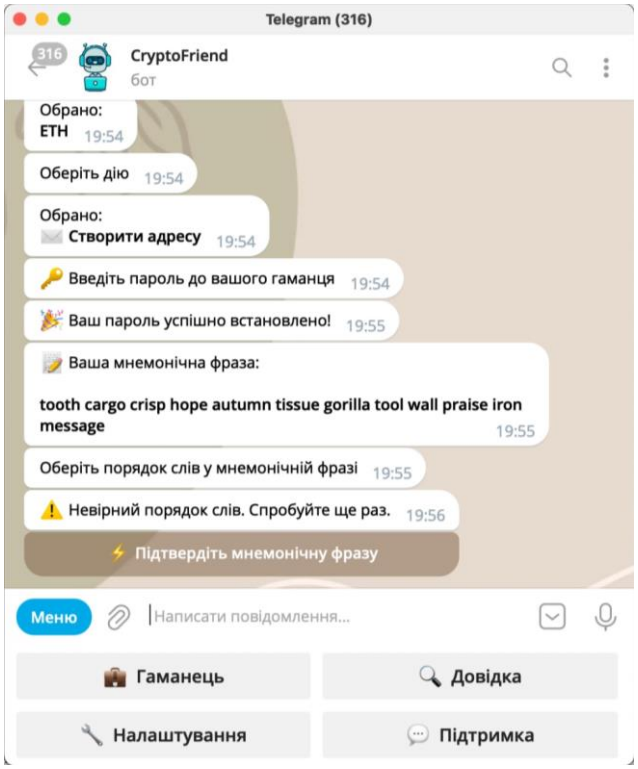


Рисунок 5.13. Повідомлення про помилку в послідовності слів мнемонічної фрази

Результатом правильно введеної мнемонічної фрази, користувач може побачити створену публічну адресу.

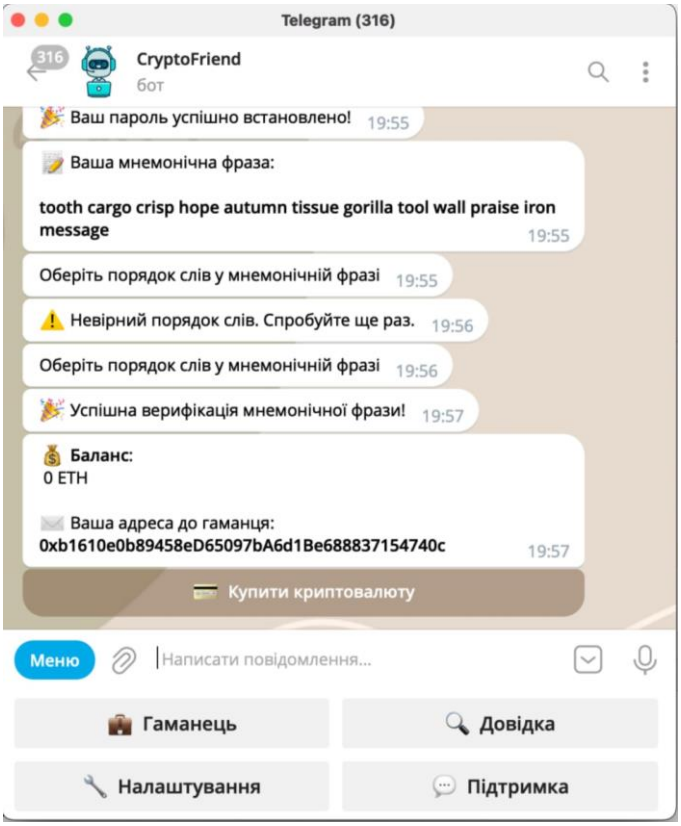


Рисунок 5.14. Створена публічна адреса етеріуму

Розкриємо можливість здійснення транзакції в межах однієї мережі на прикладі біткоїну. З головного меню рухаємось у напрямку «Гаманець» - «Надіслати» - «BTC» . Далі користувачу необхідно ввести пароль, який він створив на минулому кроці. Бот одразу стирає повідомлення, яке зберігало пароль для того, щоб уникнути витік даних. Є можливість відновлення паролю, що вимагає імпортування приватного ключа. Далі вводимо адресу отримувача та суму. Після цього бот надсилає повідомлення про підтвердження або скасування транзакції.

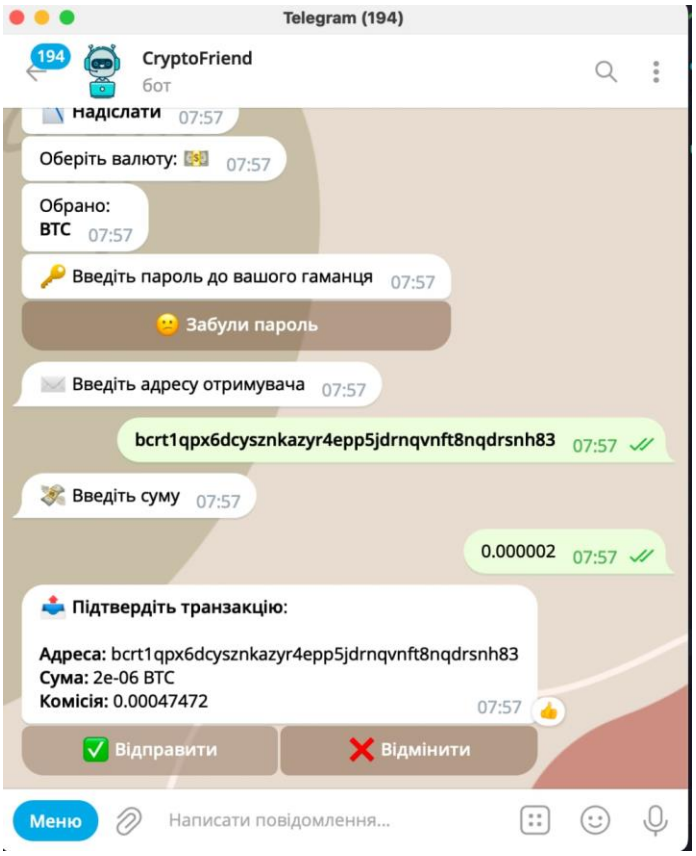


Рисунок 5.15. Можливість підтвердження даних для здійснення транзакції

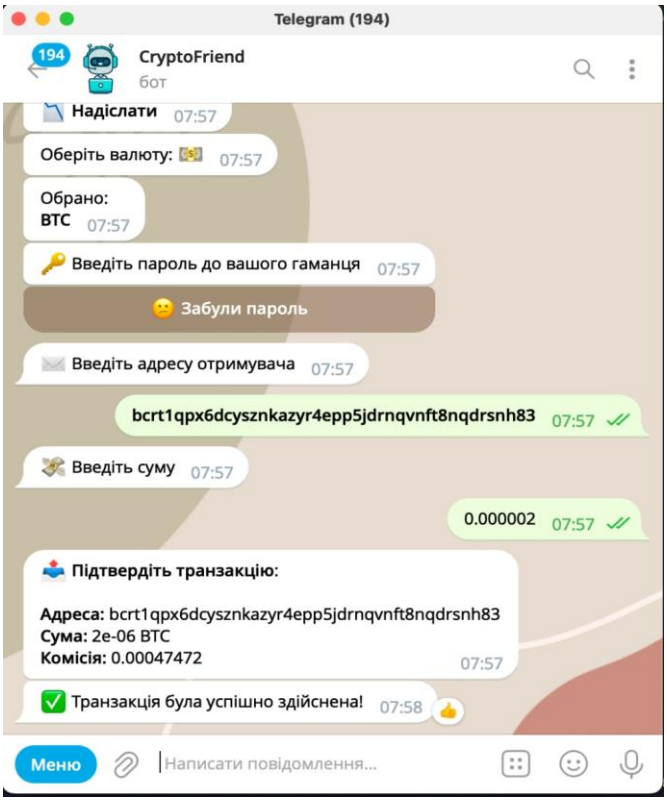


Рисунок 5.16. Повідомлення про статусу транзакції

Розглянемо кнопку «Довідка», яка містить в собі інформацію про комісію за кожний переказ коштів та поширенні запитання.

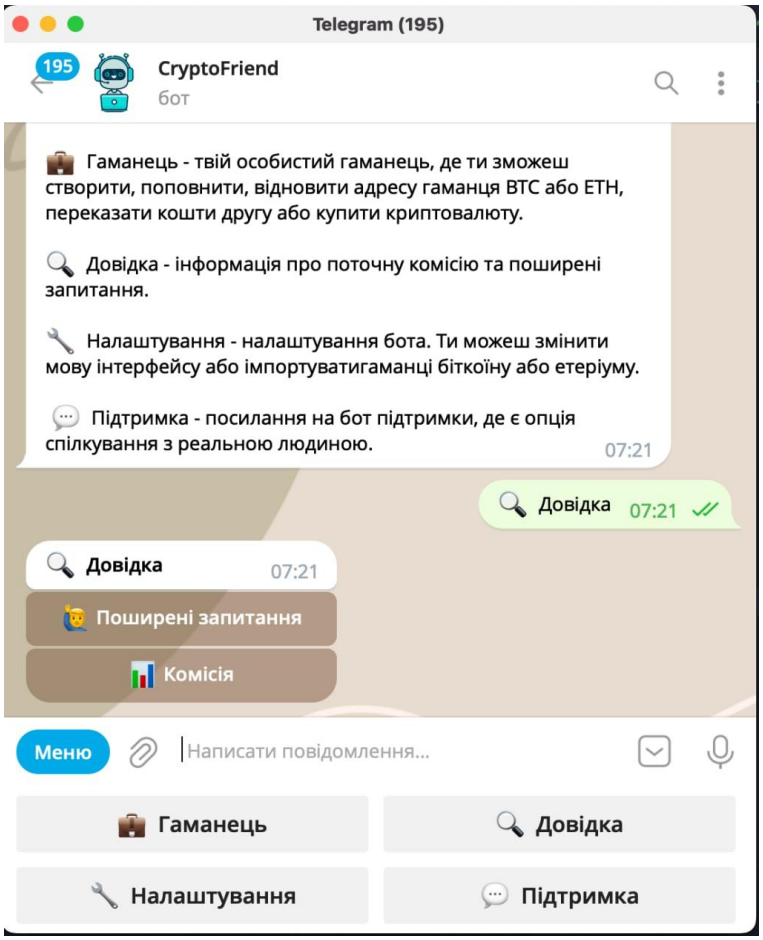


Рисунок 5.17. Реакція на натискання кнопки «Довідка»

Поширенні запитання представлені у вигляді кнопок, відповідь на які можна отримати, натиснувши на них.

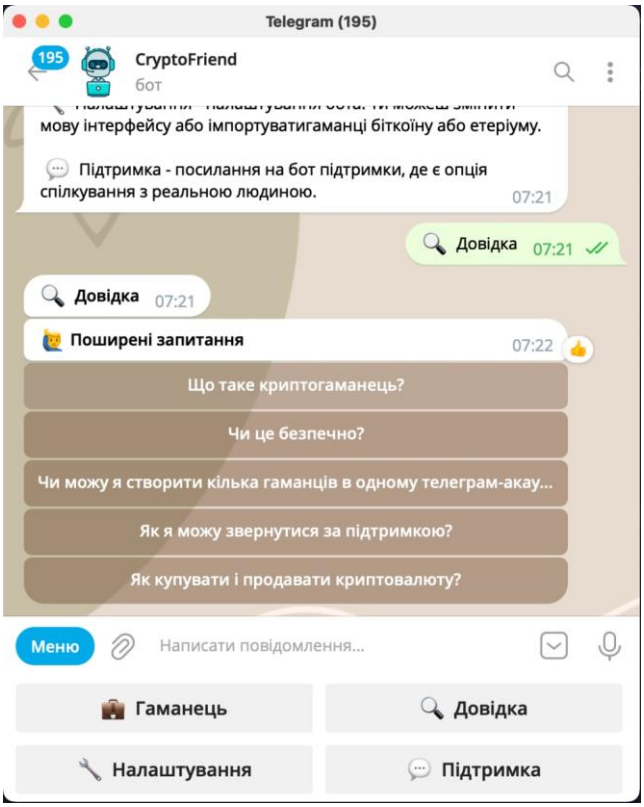


Рисунок 5.18. Список поширених запитань

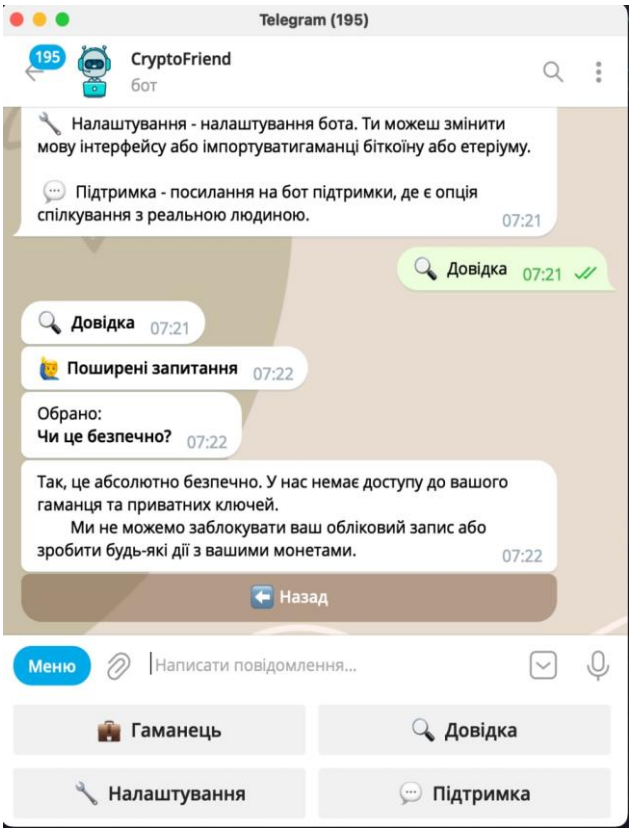


Рисунок 5.19. Відповідь на поширене запитання «Чи це безпечно»

Кнопка «Налаштування» дає змогу змінити мову, імпортувати або видалити адресу.

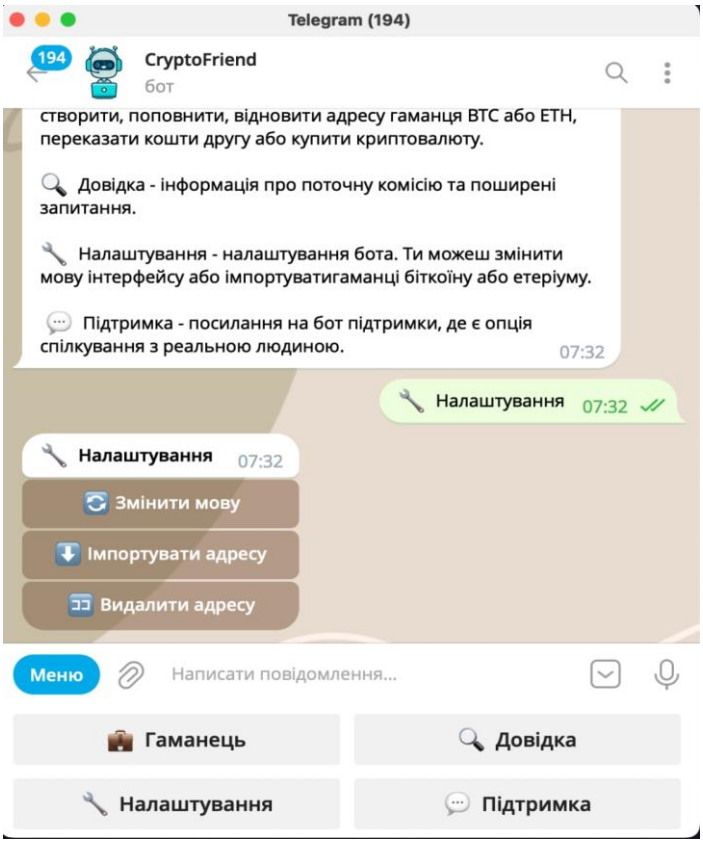


Рисунок 5.20. Реакція на натискання кнопки «Налаштування»

Розглянемо приклад зміни мови. За замовчуванням бот може змінювати мови інтерфейсу в залежності від того, яка мова стоїть в налаштуваннях до телеграм-акаунту.

Приклад зміни мови інтерфейсу на англійську:

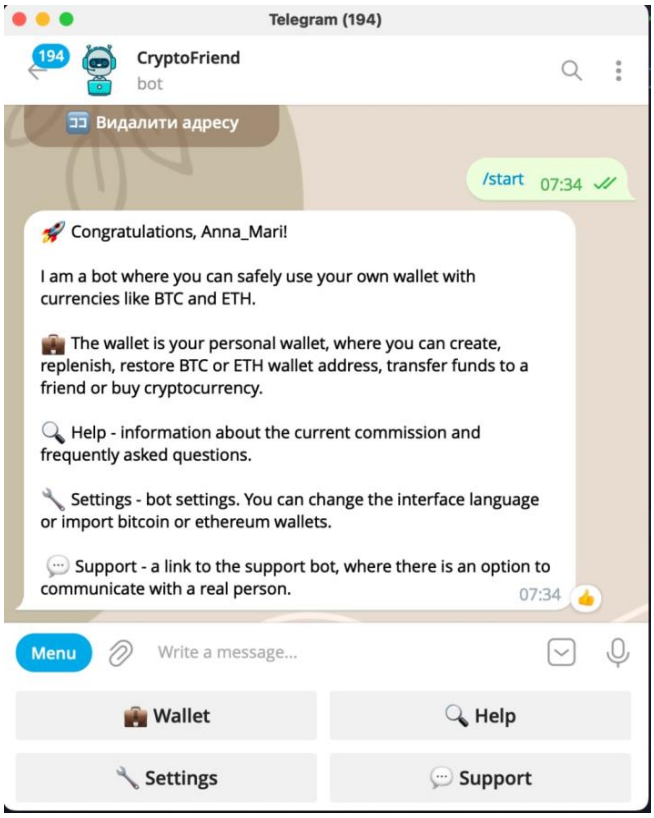


Рисунок 5.21. Приклад інтерфейсу англійською мовою

Після натискання на кнопку «Підтримка» користувач може перейти за посиланням до іншого боту підтримки та задати там питання. В свою чергу бот-підтримка є доданим до групи в телеграмі, де учасники групи можуть відповісти на всі запитання користувача.


Рисунок 5.22. Взаємодія користувача з ботом підтримки

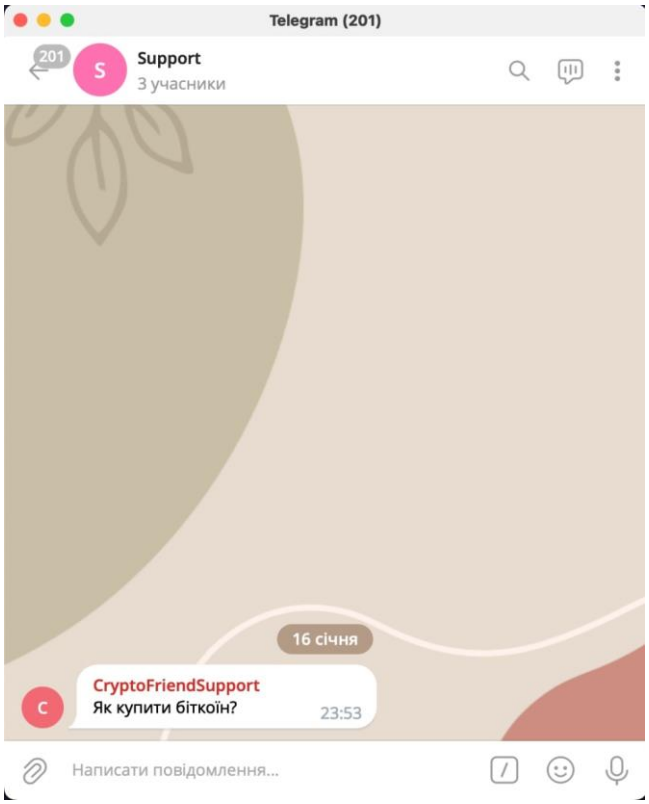


Рисунок 5.23. Відправка запитання від користувача до групи адміністраторів в телеграмі

# ВИСНОВКИ

В даній кваліфікаційній роботі “Дослідження блокчейн технологій для обробки і передачі інформації з використанням криптографічних методів шифрування даних” розглянути процеси створення, передачі та обробки інформації в блокчейн з використанням методів шифрування даних на прикладі таких криптовалют, як біткоїн та етеріум.

В роботі досліджені основні моменти криптографічних методів шифрування, які використовуються в технології блокчейн, різниці між звичайними грошима та криптовалютою.

Проведено дослідження внутрішніх процесів технології блокчейн із використанням програмного забезпечення, створений автоматизований бот, який відображає основні риси технології блокчейн.

В процесі написання дипломної роботи були використані методи спостереження, метод експериментально-теоретичного рівня, аналітичний метод, розрахунковий метод.

Науковою новизною роботи є створення нового способу оброблення та здійснення грошових транзакцій.

Практичною цінністю роботи є те, що будь-яка людина зможе створити криптовалютний гаманець для здійснення транзакцій в валюті Bitcoin та Ethereum. Власник бота зможе заробляти на відсотках від кожної транзакції. Комісія від транзакцій регулюється на адміністративній веб-сторінці в залежності від валюти.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. Офіційна документація фреймоврку aiogram [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aiogram.dev/uk_UA/latest/index.html
2. Офіційна документація фреймоврку Django [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com/
3. Офіційна документація Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/
4. Офіційна документація Bitcoin Core [Електронний ресурс] – Режим доступу до ресурсу: https://bitcoin.org/en/bitcoin-core/
5. Офіційна документація Geth [Електронний ресурс] – Режим доступу до ресурсу: https://geth.ethereum.org/
6. Що таке фіантні гроші [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A4%D1%96%D0%B0%D1%82%D0%BD%D1%96_%D0%B3%D1%80%D0%BE%D1%88%D1%96
7. Криптовалюта в світі: стан, регулювання та перспективи. Батракова Т.І., Турбарова Я.О.
8. Глобальні графіки криптовалют [Електронний ресурс] – Режим доступу до ресурсу: https://coinmarketcap.com/uk/charts/
9. Визначення Ethereum [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Ethereum
10. Binance academy [Електронний ресурс] – Режим доступу до ресурсу: https://academy.binance.com/en
11. Глобальні графіки криптовалют [Електронний ресурс] – Режим доступу до ресурсу: https://coinmarketcap.com/uk/charts/
12. Методичні рекомендації до викон. випускної кваліфікаційної роботи на здобуття освітнього ступеня «Магістр» спец. 122 «Комп’ютерні науки», освітньо-професійної програми «Інформаційні управляючі системи та

технології» ден. форми навчання [Електрон. ресурс] / уклад.
О. М. М'якшило, М. П. Костіков. – К.: НУХТ, 2022. – 28 с.
13. Визначення Ethereum [Електронний ресурс] – Режим доступу до ресурсу: https://business.diia.gov.ua/cases/tehnologii/blokcejn-ne-prosto-tehnologia-a-treta-hvila-internet-revolucii
14. Всесвітній економічний форум у Давосі [Електронний ресурс] – Режим доступу до ресурсу: https://www.dw.com/uk/%D0%B7%D1%

# ДОДАТКИ

## Додаток А. Розгортання вузлів біткоїну та етеріуму

1. Кастомний набір кодів для оброблення відповідей від біткоїну та етеріуму на сервері Django

```
ERROR_CODES = {
    0: 'Successful',
    1: 'Wrong password',
    2: 'Wallet does not exist',
    3: 'Wrong address format',
    4: "Wallet is already exists",
    5: "Client does not exist",
    6: "Insufficient funds",
    7: "Bitcoin node is not responding...",
    8: "Bitcoin core is reloading",
    9: "Ethereum node is not responding..."
}
```

2. Код для взаємодії з біткоїном по протоколу RPC на мові програмування Python

```
class BitcoinService:

    def __init__(self):
        self.node_url = settings.BITCOIN_NODE_URL
        self.session = requests.Session()
        self.session.auth = (
            settings.BITCOIN_NODE_USER,
            settings.BITCOIN_NODE_PASSWORD,
        )
        self.session.headers = {
            'content-type': 'text/plain;'
        }
        self.codes = {
            -18: 2,
            -14: 1,
            -6: 6,
            -28: 8
        }

    def create_wallet(self, wallet_id, password):
        """
        Створення гаманця
        https://developer.bitcoin.org/reference/rpc/createwallet.html
        """
        payload = {
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "createwallet",
            "params": [
                str(wallet_id),
                False,
                False,
                password,
                False,
                False,
                True
            ]
```

```
        }
        return self._request(url=self.node_url, data=json.dumps(payload))

    def create_address(self, wallet_id):
        """
        Створення адреси
        https://developer.bitcoin.org/reference/rpc/getnewaddress.html
        """
        payload = {
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "getnewaddress",
            "params": []
        }
        return self._request(url=f"{self.node_url}/wallet/{wallet_id}",
data=json.dumps(payload))

    def send_to_address(self, user_id, password, address, amount, comment:
str = ""):
        """
        Здійснення транзакції на іншу адресу
        Docs: https://developer.bitcoin.org/reference/rpc/sendtoaddress.html
        """
        unlock_wallet, status = self._unlock_wallet(user_id, password)
        if not status:
            return unlock_wallet, False

        payload = {
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "sendtoaddress",
            "params": [
                address,
                amount,
                comment
            ]
        }

        return self._request(url=f"{self.node_url}/wallet/{user_id}",
data=json.dumps(payload))

    def get_balance(self, wallet_id):
        """
        Отримати баланс гаманця
        https://developer.bitcoin.org/reference/rpc/getbalance.html
        """
        payload = {
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "getbalance",
            "params": []
        }
        return self._request(url=f"{self.node_url}/wallet/{wallet_id}",
data=json.dumps(payload))

    def echo_blockchain_info(self):
        """
        Отримання статусу та загальної інформації від біткоїн клієнту
        """
        payload = {
```

```
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "getblockchaininfo",
            "params": []
        }
        r = self.session.post(self.node_url, json.dumps(payload))
        return self._response(r)

    # Private
    def _unlock_wallet(self, wallet_id, passphrase):
        """
        Розблокувати гаманець перед його використанням
        https://developer.bitcoin.org/reference/rpc/walletpassphrase.html
        """
        payload = {
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "walletpassphrase",
            "params": [
                passphrase,  # wallet passphrase
                60  # unlock wallet for 60 seconds
            ]
        }
        r = self.session.post(f"{self.node_url}/wallet/{wallet_id}",
json.dumps(payload))
        return self._response(r)

    def _lock_wallet(self, wallet_id):
        """
        Заблокувати гаманець після його використання
        https://developer.bitcoin.org/reference/rpc/walletlock.html
        """
        payload = {
            "jsonrpc": "1.0",
            "id": str(uuid4()),
            "method": "walletlock",
            "params": []
        }

        return self._response(
            self.session.post(f"{self.node_url}/wallet/{wallet_id}",
json.dumps(payload))
        )

    def _response(self, response):
        """
        Обробка відповіді від біткоїн-клієнту
        """
        if type(response) is tuple:
            return response

        data = response.json()
        if response.status_code > 201:
            logger.error(json.dumps(data))

            if data["error"]["code"] not in self.codes.keys():
                return {"error": data['error']['message']}, False

            return {"code": self.codes[data['error']['code']]}, False
```

```
        return {"result": data.pop("result")}, True

    def _request(self, *args, **kwargs):
        """
        Здійснення запросу до біткоїн-клеінту
        """
        try:
            response = self.echo_blockchain_info()
            if response[0].get('code') == 8:
                return {"code": 8, "result": None}, False

            r = self.session.post(**kwargs)
            return self._response(r)
        except requests.exceptions.ConnectionError:
            return {"code": 7}, False
```

## 3. Код для взаємодії з біткоїном по протоколу RPC на мові програмування Python

```
def btc_exception(func):
    """Обгортає виключення для взаємодії з web3"""
    def wrap(*args, **kwargs):
        try:
            response = func(*args, **kwargs)
            return response
        except exceptions.InvalidAddress:
            return {"code": 3}, False
        except requests.exceptions.ConnectionError:
            return {"code": 9}, False
        except exceptions.CannotHandleRequest:
            return {"code": 9}, False
        except ValueError:
            return {"code": 6}, False
        except Exception as error:
            return {"error": error}, False
    return wrap


class GethService:

    def __init__(self) -> None:
        self.keystore_folder = settings.KEYSTORE_FOLDER
        self.w3 = Web3(Web3.HTTPProvider(settings.GETH_URL))

    @btc_exception
    def create_account(self, password: str):
        """
        Створення нового акаунту/адреси
        Docs:

https://web3py.readthedocs.io/en/stable/web3.geth.html?highlight=new_account#
web3.geth.personal.new_account
        :param password: str
        :return: ChecksumAddress|str
        """
        return self.w3.geth.personal.new_account(password), True

    @btc_exception
```

```
    def get_account_balance(self, address: str):
        """
        Отримання балансу від гаманця
        Docs:
https://web3py.readthedocs.io/en/stable/web3.eth.html?highlight=getBalance#we
b3.eth.Eth.getBalance
        :param address: str
        :return: int, decimal.Decimal
        """
        chs_address = self.w3.toChecksumAddress(address)
        return self.w3.fromWei(
            self.w3.eth.getBalance(chs_address),
            "ether"
        ), True

    def get_private_key(self, address: str, password: str, format_key: bool =
None) -> str:
        """
        Знаходить та відкриває приватний ключ користувача
        :param address: client address in string format
        :param password: client address password in str
        :param format_key: Formatting hex or not hex
        :return: hex bites or string
        """
        address = (address.replace('0x', '') if "0x" in address else
address).lower()
        filepath = ""
        for root, dirs, files in os.walk(self.keystore_folder):
            for file in files:
                if address in file:
                    filepath = os.path.join(self.keystore_folder, file)

        if not len(filepath):
            raise ValueError("Cant find file")

        with open(filepath) as keyfile:
            keyfile_json = keyfile.read()
            private_key = Account.decrypt(keyfile_json, password)
            return private_key if not format_key else private_key.hex()

    @btc_exception
    def send_to_address(self, user_id, address, password, amount, comment:
str = ""):
        """
        Здійснює транзакцію по мережі етеріум
        Docs:
        https://cryptomarketpool.com/send-a-transaction-to-the-ethereum-
blockchain-using-python-and-web3-py/
        :param user_id: user_id is used for find an address_sender
        :param address: receiver address
        :param password: sender password
        :param amount: value in ETH
        :param comment: comment to tx
        :return: transaction hash3
        """
        sender_address = ClientWallet.objects.get_eth_address(client=user_id)
        # You need to unlock your account to make transactions
        try:
            self.__unlock_account(sender_address, password)
        except ValueError:
```

```
            return {"code": 1}, False

        # You need to unpack account file from keystore to get private_key
        private_key1 = self.get_private_key(sender_address, password)
        nonce = self.w3.eth.getTransactionCount(sender_address)
        tx = {
            'chainId': self.w3.eth.chain_id,
            'nonce': nonce,
            'to': address,
            'value': self.w3.toWei(amount, 'ether'),
            'gas': 2000000,
            'gasPrice': self.w3.toWei('50', 'gwei')
        }
        signed_tx = self.w3.eth.account.sign_transaction(tx, private_key1)
        tx_hash = self.w3.eth.sendRawTransaction(signed_tx.rawTransaction)
        return self.w3.toHex(tx_hash), True

    def __unlock_account(self, address, password):
        """
        Розблаковує гаманець перед його використанням
        Docs:
https://web3py.readthedocs.io/en/stable/web3.eth.html?highlight=accounts#web3
.eth.Eth.accounts
        :return: list of accounts addresses in string
        """
        chs_address = self.w3.toChecksumAddress(address)
        self.w3.geth.personal.unlock_account(chs_address, password)

    def __lock_account(self, address):
        """
        Блокує гаманець після його використання
        Docs:
https://web3py.readthedocs.io/en/stable/web3.eth.html?highlight=accounts#web3
.eth.Eth.accounts
        :return: list of accounts addresses in string
        """
        chs_address = self.w3.toChecksumAddress(address)
        self.w3.geth.personal.lock_account(chs_address)
```

## Додаток Б. Розроблення головного серверу для впровадження API

1. Запуск серверу Django за допомогою Docker

```
server:
  build: ./app/
  command: python manage.py runserver 0.0.0.0:8000
  volumes:
    - ./app/:/usr/src/app/
    - ./common_folder:/keystore
    - ./services/bitcoin-core/bitcoin-core:/bitcoin/
    - ./logs/:/logs/
  ports:
    - 8000:8000
  env_file:
    - ./app/.env
  depends_on:
    - db
```

2. Запуск бази даних PostgreSQL за допомогою Docker

```
db:
  image: postgres:12.0-alpine
  volumes:
    - postgres_data:/var/lib/postgresql/data/
  ports:
    - 5432:5432
  environment:
    - POSTGRES_USER=admin
    - POSTGRES_PASSWORD=111
    - POSTGRES_DB=main
```

# Додаток В. Написання телеграм-боту

## 1. Модуль для налаштування телеграм боту

```
from aiogram import Bot, Dispatcher
from aiogram.contrib.fsm_storage.redis import RedisStorage2

from config import BOT_TOKEN, REDIS_HOST


bot = Bot(token=BOT_TOKEN)
storage = RedisStorage2(
    host=REDIS_HOST,
    port=6379)
dp = Dispatcher(bot, storage=storage)
```

## 2. Запуск телеграм бота в середовищі Docker

```
3.      bot:
  build: ./bot/
  command: python main.py
  volumes:
    - ./bot/:/usr/src/bot/
    - ./logs/:/logs/
  env_file:
    - ./bot/.env
  depends_on:
    - server
```

## 4. Конфігураційний файл з назвами кнопок, текстами повідомлень тощо

```
from aiogram.utils.emoji import emojize
from dotenv import load_dotenv
import os
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent
load_dotenv(BASE_DIR / '.env')

# GENERAL
BOT_TOKEN = os.getenv('BOT_TOKEN')
DEBUG = os.getenv('DEBUG')
REDIS_HOST = os.getenv('REDIS_HOST')

# API
BOT_FAQ_URL = os.getenv('BOT_FAQ_URL')

BOT_CREATE_ETH_ADDRESS_URL = os.getenv('BOT_CREATE_ETH_ADDRESS_URL')
BOT_GET_ETH_WALLET_URL = os.getenv('BOT_GET_ETH_WALLET_URL')
BOT_GET_ETH_BALANCE_URL = os.getenv('BOT_GET_ETH_BALANCE_URL')
BOT_CREATE_ETH_WALLET_URL = os.getenv('BOT_CREATE_ETH_WALLET_URL')
```

```
BOT_CLIENT_UPDATE_URL = os.getenv('BOT_CLIENT_UPDATE_URL')
BOT_GET_CLIENT_URL = os.getenv('BOT_GET_CLIENT_URL')

BOT_GET_WALLET_URL = os.getenv('BOT_GET_WALLET_URL')
BOT_GET_WALLET_MNEMONIC_URL = os.getenv('BOT_GET_WALLET_MNEMONIC_URL')

BOT_REVIEW_TX_URL = os.getenv('BOT_REVIEW_TX_URL')
BOT_MAKE_TRANSACTION_ETH_URL = os.getenv('BOT_MAKE_TRANSACTION_ETH_URL')
BOT_MAKE_TRANSACTION_BTC_URL = os.getenv('BOT_MAKE_TRANSACTION_BTC_URL')

BOT_CREATE_BTC_ADDRESS_URL = os.getenv('BOT_CREATE_BTC_ADDRESS_URL')
BOT_GET_BTC_ADDRESS_URL = os.getenv('BOT_GET_BTC_ADDRESS_URL')
BOT_CREATE_BTC_WALLET_URL = os.getenv('BOT_CREATE_BTC_WALLET_URL')
BOT_GET_BTC_BALANCE_URL = os.getenv('BOT_GET_BTC_BALANCE_URL')


# COMMANDS
COMMAND_START = 'Головне меню'
COMMAND_HELP = 'Допомога'
COMMAND_WALLET = 'Ваш гаманець'
COMMAND_SETTINGS = 'Налаштування'
COMMAND_SUPPORT = 'Підтримка'
COMMAND_INFO = 'Довідник'

# ERROR CODES
ERROR_CODES = {
    0: 'Successful',
    1: 'Wrong password',
    2: 'Wallet does not exist',
    3: 'Wrong address format',
    4: "Wallet is already exists",
    5: "Client does not exist",
    6: "Insufficient funds",
    7: "Bitcoin node is not responding...",
    8: "Bitcoin core is reloading",
    9: "Ethereum node is not responding..."
}

# MAIN MENU BUTTONS
BUTTON_WALLET = emojize(':briefcase:') + ' ' + ' Гаманець'
BUTTON_INFO = emojize(':mag:') + ' ' + ' Довідка'
BUTTON_SETTINGS = emojize(':wrench:') + ' ' + ' Налаштування'
BUTTON_SUPPORT = emojize(':speech_balloon:') + ' ' + ' Підтримка'

# GENERAL MESSAGES
MESSAGE_GREETINGS = emojize(':rocket:') + \
                        f" Вітаю, %s !\n\nЯ - бот, де ти зможеш безпечно
користуватися власним гаманецем " \
                        f"з такими валютами, як BTC та ETH.\n" \
                        f"\n{BUTTON_WALLET} - твій особистий гаманець, де ти
зможеш створити, поповнити, " \
                        f"відновити адресу гаманця BTC або ETH, переказати
кошти другу або купити криптовалюту.\n\n" \
                        f"{BUTTON_INFO} - інформація про поточну комісію та
поширені запитання.\n\n" \
                        f"{BUTTON_SETTINGS} - налаштування бота. Ти можеш
змінити мову інтерфейсу або імпортувати" \
                        f"гаманці біткоїну або етеріуму.\n\n " \
                        f"{BUTTON_SUPPORT} - посилання на бот підтримки, де є
опція спілкування з реальною людиною."
```

```
MESSAGE_HELP = emojize(':grey_question:') + f" Цей бот був створений для
взаємодії з біткоїном та етеріумом.\nБільше" \
                                          f"інформації ви можете знайти у
вкладці {BUTTON_INFO} або {BUTTON_SUPPORT}"
MESSAGE_UNKNOWN = 'для того, щоб побачити головне меню'
MESSAGE_SELECTED = 'Обрано: \n'
MESSAGE_FAILED_SORRY = emojize(':heavy_exclamation_mark:') + ' Вибачте. Щось
пішло не так. Спробуйте пізніше.'
MESSAGE_CHOOSE_CURRENCY = 'Оберіть валюту: ' + emojize(':dollar:')

# THROTTLING MESSAGES
MESSAGE_THROTTLING_EXCEEDED_LIMIT = emojize(':no_entry:') + \
                                    ' Перевищено ліміт повідомлень для
користувача, повторіть спробу пізніше.'
MESSAGE_THROTTLING_UNBLOCKED = emojize(':recycle:') + ' Розблаковано.'

# WALLET MESSAGES
MESSAGE_CHOOSE_SEED_PHRASE_ORDER = 'Оберіть порядок слів у мнемонічній фразі
'
MESSAGE_SEED_SUCCESSFUL_CONFIRM = emojize(':tada:') + ' Успішна верифікація
мнемонічної фрази!'
MESSAGE_SEED_WRONG_CONFIRM = emojize(':warning:') + ' Невірний порядок слів.
Спробуйте ще раз.'
MESSAGE_SEED_PHRASE = emojize(':pencil:') + ' Ваша мнемонічна фраза:\n\n'

MESSAGE_ENTER_PASSWORD = emojize(':key:') + ' Введіть пароль до вашого
гаманця'
MESSAGE_FORGOT_PASSWORD = 'FORGET PASSW'
MESSAGE_GENERATED_PASSWORD = emojize(':key:') + ' Згенерований пароль:'
MESSAGE_ENTER_AMOUNT = emojize(':money_with_wings:') + ' Введіть суму'
MESSAGE_ENTER_RECEIVER_ADDRESS = emojize(':envelope:') + ' Введіть адресу
отримувача'
MESSAGE_CHOOSE_ACTION = 'Оберіть дію '

MESSAGE_SUCCESSFUL_ADDRESS_CREATE = emojize(':tada:') + ' Ваш пароль успішно
встановлено!'

MESSAGE_YOUR_ADDRESS = emojize(':envelope:') + ' Ваша адреса до гаманця:\n'
MESSAGE_YOUR_PRIVATE_KEY = emojize(':exclamation:') + 'Увага! Нікому не
показуйте ваш приватний ключ!\n\n'
MESSAGE_BALANCE = emojize(':moneybag:') + ' *Баланс*'

MESSAGE_TRANSACTION_CONFIRM = emojize(':outbox_tray:') + \
                              ' *Підтвердіть транзакцію*:\n\n*Адреса:*
%s\n*Сума:* %s %s\n*Комісія: *%s'
MESSAGE_SEED_SUCCESSFUL_TRANSACTION = emojize(':white_check_mark:') + '
Транзакція була успішно здійснена!'

MESSAGE_WRONG_AMOUNT = emojize(':x:') + ' Невірна введена сума. Спробуйте ще
раз.'

MESSAGE_SUPPORT = 't.me/sarahboostersupport'

# BUY CRYPTO MESSAGE
MESSAGE_BUY_BTC = 'https://buy.bitcoin.com/'
MESSAGE_BUY_ETH = 'https://buy.bitcoin.com/'

# GENERAL BUTTONS
BUTTON_TRY_AGAIN = emojize(':leftwards_arrow_with_hook:')
BUTTON_BACK = emojize(':arrow_left:') + ' Назад'
```

```
BUTTON_YES = emojize(':white_check_mark:') + ' Прийняти'
BUTTON_NO = emojize(':x:') + ' Ще раз'

BUTTON_FAQ = emojize(':man_raising_hand:') + ' Поширені запитання'
BUTTON_FEE = emojize(':bar_chart:') + ' Комісія'

# WALLET BUTTONS
BUTTON_DEPOSIT = emojize(':chart_with_upwards_trend:') + ' Поповнити'
BUTTON_WITHDRAW = emojize(':chart_with_downwards_trend:') + ' Надіслати'
BUTTON_CREATE_ADDRESS = emojize(':envelope:') + ' Створити адресу'
BUTTON_IMPORT_ADDRESS = emojize(':arrow_down:') + ' Імпортувати адресу'
BUTTON_DELETE_ADDRESS = emojize(':koko:') + ' Видалити адресу'
BUTTON_BALANCE = emojize(':moneybag:') + ' Баланс'
BUTTON_BUY_CRYPTO = emojize(':credit_card:') + ' ' + ' Купити криптовалюту'
BUTTON_GENERATE_PASSWORD = emojize(':bulb:') + ' ' + ' Згенерувати надійний
пароль'

BUTTON_SEED_PHRASE_CONFIRM = emojize(':zap:') + ' Підтвердіть мнемонічну
фразу'

BUTTON_TRANSACTION_SEND = emojize(':white_check_mark:') + ' Відправити'
BUTTON_CANCEL = emojize(':x:') + ' Відмінити'
BUTTON_FORGOT_PASSWORD = emojize(':confused:') + ' Забули пароль'

# SETTINGS BUTTONS
BUTTON_CHANGE_LANGUAGE = emojize(':arrows_counterclockwise:') + ' Змінити
мову'
```

## 5. Запуск Redis в Docker-контейнері

```
redis:
  image: redis:6.2.6-alpine
  volumes:
    - redis_data:/var/lib/redis
  restart: always
  ports:
    - 6379:6379
  depends_on:
    - bot
```

## 6. Реалізації станів сума та підтвердження транзакції кінцевого автомату відповідно

```
dp.message_handler(ChatTypeFilter(types.ChatType.PRIVATE),
state=WithdrawState.receiver)
async def enter_amount(message: types.Message, state: FSMContext):
    """
    - Handler is called after entering receiver address
    - Handler open state by asking amount for transaction
    """
    await state.update_data(receiver=message.text)

    # Встановлення стану сума
    await WithdrawState.amount.set()

    message_enter_amount = translate(text=MESSAGE_ENTER_AMOUNT,
```

```
user_id=message.chat.id)
    await message.answer(message_enter_amount)


@dp.message_handler(ChatTypeFilter(types.ChatType.PRIVATE),
state=WithdrawState.amount)
async def check_amount(message: types.Message, state: FSMContext):
    """
    - Handler is called after entering amount for transaction
    - Checks type of amount - should be float
    - Makes transaction via django API
    - Returns transaction hash
    """
    amount = message.text
    try:
        amount_float = float(amount)
        await state.update_data(amount=amount_float)
        await confirm_transaction(message, state)

        # Встановлення стану підтвердження транзакції True/False
        await WithdrawState.confirm.set()
    except ValueError:
        message_wrong_amount = translate(text=MESSAGE_WRONG_AMOUNT,
user_id=message.chat.id)
        await message.answer(message_wrong_amount)
```

## 7. Middleware для блокування користувача, який надсилає повідомлення частіше однієї секунди

```
class ThrottlingMiddleware(BaseMiddleware):

    """
    Simple aiogram docks middleware
    Blocks one type of message handler if it exceeded limit seconds
    """

    # User can can send a message every limit seconds
    def __init__(self, limit=1, key_prefix='antiflood_'):
        self.rate_limit = limit
        self.prefix = key_prefix
        super(ThrottlingMiddleware, self).__init__()

    async def on_process_message(self, message: types.Message, data: dict):
        """
        This handler is called when dispatcher receives a message
        :param message:
        """
        # Get current handler
        handler = current_handler.get()

        # Get dispatcher from context
        dispatcher = Dispatcher.get_current()
        # If handler was configured, get rate limit and key from handler
        if handler:
            limit = getattr(handler, 'throttling_rate_limit',
self.rate_limit)
            key = getattr(handler, 'throttling_key',
f"{self.prefix}_{handler.__name__}")
```

```
        else:
            limit = self.rate_limit
            key = f"{self.prefix}_message"

        # Use Dispatcher.throttle method.
        try:
            await dispatcher.throttle(key, rate=limit)
        except Throttled as t:
            # Execute action
            await self.message_throttled(message, t)

            # Cancel current handler
            raise CancelHandler()

    async def message_throttled(self, message: types.Message, throttled:
Throttled):
        """
        Notify user only on first exceed and notify about unlocking only on
last exceed
        :param message:
        :param throttled:
        """
        handler = current_handler.get()
        dispatcher = Dispatcher.get_current()
        if handler:
            key = getattr(handler, 'throttling_key',
f"{self.prefix}_{handler.__name__}")
        else:
            key = f"{self.prefix}_message"

        # Calculate how much time is left till the block ends
        delta = throttled.rate - throttled.delta
        # Prevent flooding
        if throttled.exceeded_count <= 2:
            message_throttling_exceed_limit =
translate(text=MESSAGE_THROTTLING_EXCEEDED_LIMIT, user_id=message.chat.id)
            await message.answer(message_throttling_exceed_limit)

        # Sleep.
        await asyncio.sleep(delta)

        # Check lock status
        thr = await dispatcher.check_key(key)
```

# Додаток Г. Написання телеграм-боту підтримки

## 1. Програмний файл для запуску

```
support-bot:
  build: ./support-bot/
  command: python main.py
  volumes:
    - ./support-bot/:/usr/src/support-bot/
    - ./logs/:/logs/
  env_file:
    - ./support-bot/.env
  depends_on:
    - server
```

```
API_TOKEN = BOT_TOKEN

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

storage = RedisStorage2(
    host=REDIS_HOST,
    port=6379)

# Initialize bot and dispatcher
bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot, storage=storage)

group_id = '-603767100'


@dp.message_handler(commands=['start', 'help'])
async def send_welcome(message: types.Message):
    """
    This handler will be called when user sends `/start` or `/help` command
    """
    if message.chat.type == types.ChatType.PRIVATE:
        await message.answer(MESSAGE_GREETINGS % message.from_user.username,
reply_markup=get_main_keyboard(message.chat.id))
    else:
        print(message.chat.id)
        await message.answer(MESSAGE_GREETINGS2 % message.from_user.username,
reply_markup=get_main2_keyboard(message.chat.id))


@dp.message_handler(ChatTypeFilter(types.ChatType.PRIVATE), commands=['ask-
question'], state='*')
@dp.message_handler(lambda message: message.text == BUTTON_ASK_QUESTION,
state='*')
async def show_wallet(message: types.Message, state: FSMContext):

    await state.finish()
    await AskQuestionState.question.set()
    await message.answer(text='Напишіть ваше запитання')


@dp.message_handler(state=AskQuestionState.question)
async def show_wallet(message: types.Message, state: FSMContext):
```

```
    await state.finish()
    await
post_info(url=f'https://api.telegram.org/bot{BOT_TOKEN}/sendMessage',
data={'chat_id': group_id, 'text': message.text})



def get_main_keyboard(user_id):
    main_keyboard = ReplyKeyboardMarkup(resize_keyboard=True, row_width=2)
    main_buttons_names = [BUTTON_ASK_QUESTION,
                          BUTTON_SETTINGS]
    main_buttons = main_buttons_names
    return main_keyboard.add(*main_buttons)


def get_main2_keyboard(user_id):
    main_keyboard = ReplyKeyboardMarkup(resize_keyboard=True, row_width=2)
    main_buttons_names = [
                          BUTTON_SETTINGS]
    main_buttons = main_buttons_names
    return main_keyboard.add(*main_buttons)


def get_message_keyboard():
    password_keyboard = InlineKeyboardMarkup(row_width=1)
    password_buttons = [
        InlineKeyboardButton(text='Відповісти',
                             callback_data='generate_password_btc'),
        InlineKeyboardButton(text='Перекласти',
                             callback_data='generate_password_btc'),
    ]
    password_keyboard.add(*password_buttons)
    return password_keyboard


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

## 2. Функція для обробки команди /ask

```
@dp.message_handler(ChatTypeFilter(types.ChatType.PRIVATE), commands=['ask'],
state='*')
@dp.message_handler(lambda message: message.text == translate(BUTTON_WALLET,
message.chat.id), state='*')
async def show_wallet(message: types.Message, state: FSMContext):

    await state.finish()
    message_selected = translate(text=MESSAGE_SELECTED,
user_id=message.chat.id)
    button_support = translate(text=BUTTON_SUPPORT, user_id=message.chat.id)
    await message.answer(message_selected + bold(button_support),
parse_mode=ParseMode.MARKDOWN)
```

# Додаток Г. База даних

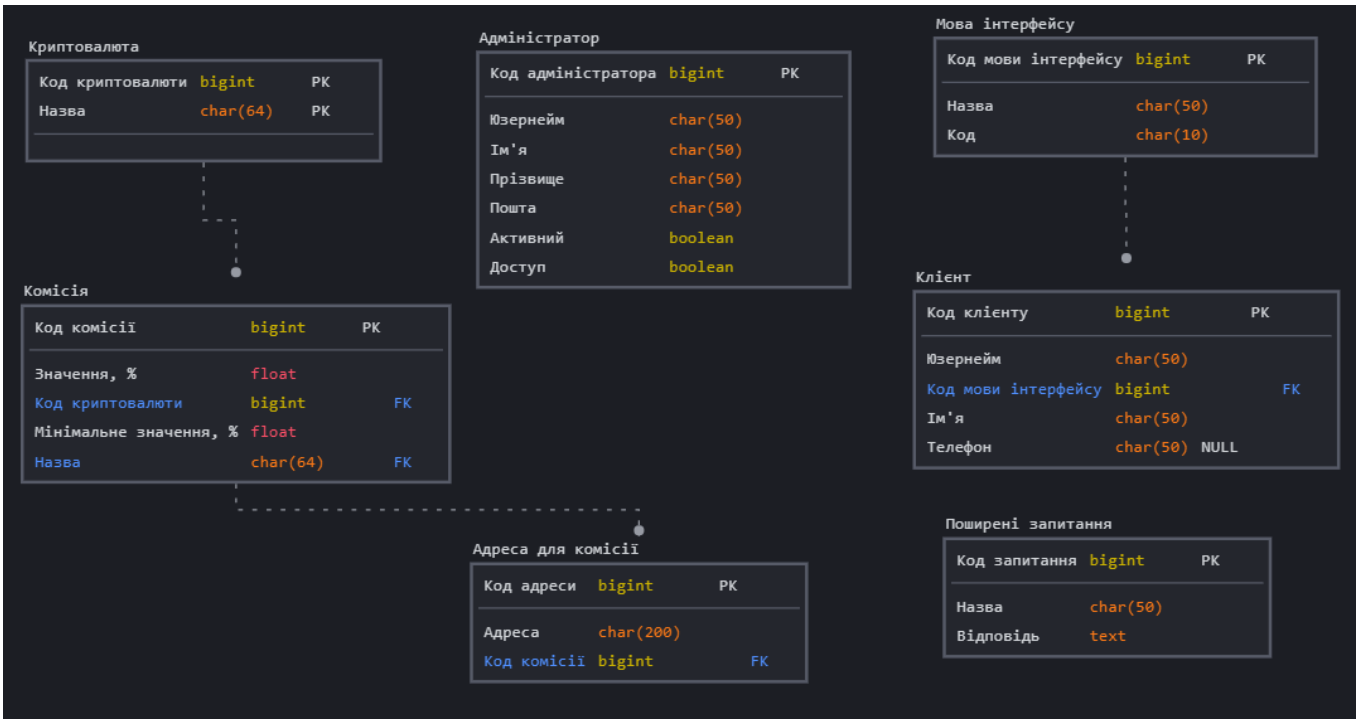


Рисунок 1. База даних програми

## Додаток Ж. Інструкція користувача

1. Створення адреси етеріуму.

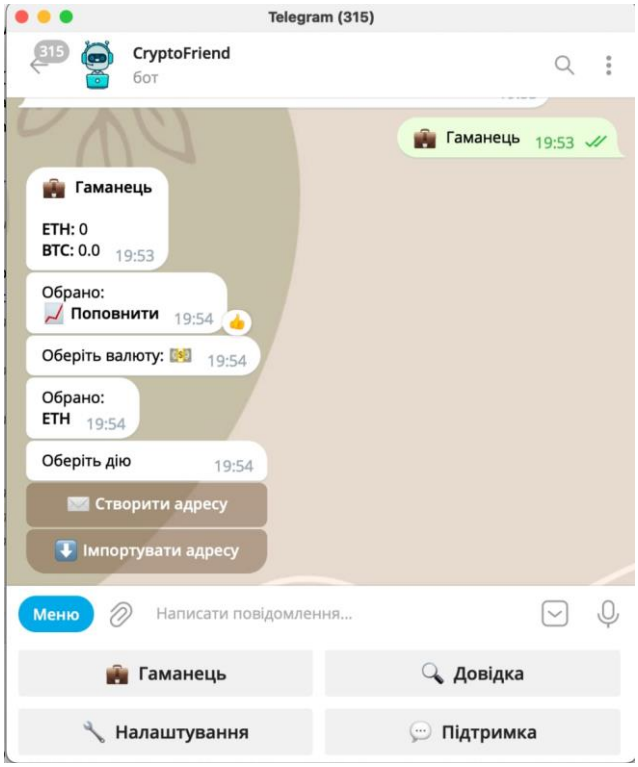


Рисунок 1. Можливість вибору створити або імпортувати адресу

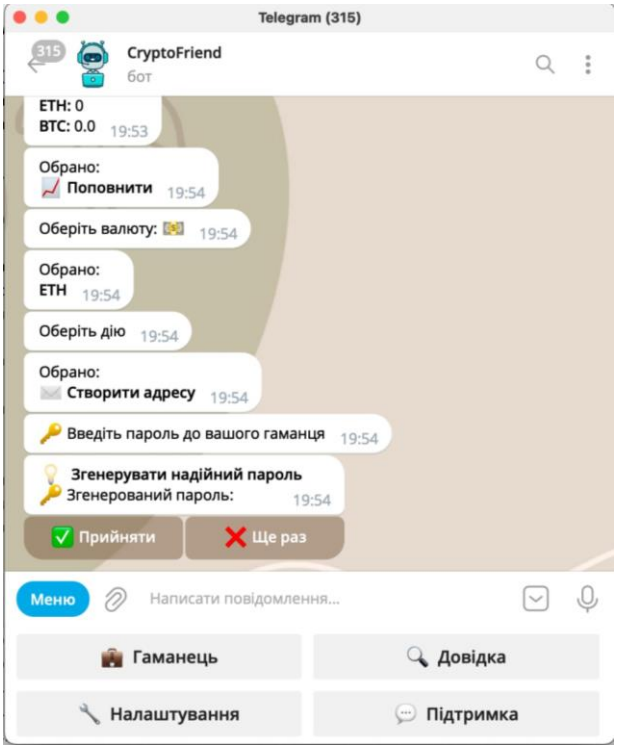


Рисунок 2. Генерація паролю до електронного гаманця