ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Телеграм-бот для виявлення підозрілих транзакцій у мережах на основі блокчейн»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ростислав НОВИК
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД-42
Ростислав НОВИК
Ім`я, ПРИЗВІЩЕ

Керівник: Андрій ГАШКО
Ім`я, ПРИЗВІЩЕ

Рецензент:
Ім`я, ПРИЗВІЩЕ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Бакалавр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК

«____» _____2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____ Новику Ростислав Сергійовичу

(прізвище, ім`я, по-батькові здобувача)

1.Тема квалафікаційної роботи: Телеграм-бот для виявлення підозрілих транзакцій у мережах на основі блокчейн

керівник кваліфікаційної роботи Андрій ГАШКО

(Ім`я, ПРИЗВІЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 20__ р. №00

2. Строк подання кваліфікаційної роботи «___» _____ 20__ р.

3. Вхідні дані до кваліфікаційної роботи:

-

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

-

5.Ілюстративний матеріал: *презентація*

6. Дата видачі завдання «___» _____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури	07.03.2025	
2	Розробка концепції телеграм-боту	20.03.2025	
3	Аналіз технологій для знаходження найбільш підходящих	22.03.2025	
4	Розробка каркасу бота	27.03.2025	
5	Інтешрація сторонніх API, та створення аналізу крипто-транзакцій та гаманців	05.04.2025	
6	Оформлення дипломної роботи	25.04.2025	

Здобувач вищої освіти _____ Ростислав НОВИК

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Андрій ГАШКО

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 80 стор., 40 рис., 16 джерел.

Мета дослідження - розробка Telegram-боту, для легкого, та швидкого аналізу транзакцій, та гаманців у різних мережах Blockchain, заради запобігання розповсюдженню санкціонних монет.

Об'єкт дослідження - процес створення сервісу перевірки безпечності транзакцій у мережах, на основі Telegram-боту

Предмет дослідження - Telegram-бот для перевірки підозрілих транзакцій у мережах Blockchain

У рамках дослідження було створено Telegram-бота для перевірки транзакцій у мережах Blockchain, який включає автоматичну обробку криптоплатежів, ручну перевірку транзакцій/гаманців на мові Python. Розробка бота здійснювалася через інтеграцію з блокчейн технологіями через бібліотеки та використання сторонніх API-сервісів. Під час розрообки було створено логіку перевірки транзакцій, ручна та автоматична перевірка, створено базу данних та її архутекруту для зберігання всх необхідних даних. Проведено аналіз найбільш популярних мереж, задля більшого охвату користувачів, та подальша їх інтеграція.

КЛЮЧОВІ СЛОВА: TELEGRAM, BOT, BLOCKCHAIN, МЕРЕЖА, API

ABSTRACT

Textual part of the qualification work for obtaining a bachelor's degree: 80 pages, 40 figures, 16 sources.

The research goal is to develop a Telegram bot for easy and quick analysis of transactions and wallets across various Blockchain networks, in order to prevent the spread of sanctioned coins.

The object of research is the process of creating a transaction safety verification service in networks, based on a Telegram bot

The subject of research is a Telegram bot for checking suspicious transactions in Blockchain networks

As part of the research, a Telegram bot was created for verifying transactions in Blockchain networks, which includes automatic processing of crypto payments, manual verification of transactions/wallets in Python. The bot development was carried out through integration with blockchain technologies via libraries and the use of third-party API services. During development, transaction verification logic was created, manual and automatic verification, a database and its architecture were created for storing all necessary data. An analysis of the most popular networks was conducted to reach more users, and their subsequent integration.

KEYWORDS: TELEGRAM, BOT, BLOCKCHAIN, NETWORK, API

ЗМІСТ

ВСТУП

Актуальність теми. Перевірка безпечності криптовалюних транзакцій є вкрай актуальною на данних момент. В сучасному світі дуже багато криптовають яка знаходиться під санкціями по різним причинам, наприклад, якщо вона була задіяна у нелегальній діяльності, такій як відмивання коштів, торгівля забороненими речами, зв'язана з казино, і так далі. Та зважаючи на це все, потрібні інструменти, за допомогою яких, можна було б оминати санкціонні кошти, щоб не натрапити під ризик.

В якості інтерфуйсу був вибраний саме Telegram-бот, тому-що це швидкий і зручний для всіх месенджер, з інтуітивно зрозумілим інтерфейсом, та швидким доступом. Значно швидше зайти у бота, та перевірити транзакцію, а ніж шукати сайт та прогружати його. Також використання саме Telegram-боту в якості інтерфейсу, дає можливість зробити систему автоматичної перевірки транзакцій. Для авто-перевірки, користовачу потрібно лише зберігти публічну адресу гаманця, всі надалі вхідні транзакції будуть автоматично перевірені, та надіслана аналітика.

Об'єкт дослідження - процес розробки Telegram-боту для виявлення підозрілих транзакцій

Предмет дослідження - розробка Telegram-боту для надання користувачам інструмента виявлення підозрілих транзакцій у мережах Blockchain

Мета дослідження - метою дослідження є створення інструменту для всіх користувачів, за допомогою якого можна виявити підозрілі транзакції у різних мережах Blockchain, та подальшого запобігання потрапляння санкціонних коштів на свої гаманці. Дослідження спрямоване на створення ефективного інструменту для моніторинга, який дозвонялє своєвчасно ідентифікувати підозрілі транзакції. Дане дослідження має практичну цінність, так як направлене на підвищення безпеки користувачів Blockchain мереж, за допомогою надання їм інструменту для своєвчасного виявлення пізрілих транзакцій.

Наукова новизна - наукова новизна дослідження полягає у розробці комплексного підходу до виявлення підозрілих транзакцій через інтерфейс Telegram-боту, що має наступні інноваційні аспекти:

1. Вперше серед подібних інструментів реалізовано підтримку багатьох мереж з можливістю автоматичного визначення конкретної мережі за адресою гаманця.

2. Розроблено унікальну систему визначення відостку ризику транзакції, зважаючи на багато аспектів.

3. Максимально зручний, мінімалістичний, та інтуітивно зрозумілий інтерфейс бота.

4. Додано інструмент автоматичної перевірки всіх вхідних транзакцій, заданого гаманця, для звичайного користувача.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

БД - База даних

Chain - Блокчейн мережа

РОЗДІЛ 1. Аналіз методів виявлення підозрілих транзакцій у Blockchain-мережах

1.1. Сучасний стан Blockchain технологій та ринку криптовалют

1.1.1 Еволюція Blockchain та її роль у сучасній економіці

Blockchain як технологія виникла у 2008 році з публікацією книги Сатоші Накамото “Bitcoin: A Peer-to-Peer Electronic Cash System” [1]. Спочатку розроблений як основа для криптовалюти Bitcoin, Blockchain пройшов значну еволюцію у багатогранну технологію з чималим спектром застосувань.

Еволюцію Blockchain можна розділити на наступні етапи:

1. *Blockchain 1.0 (2009-2013)* - перше покоління представлене Bitcoin та іншими криптовалютами, з метою цифрових платежів. Інновацію на той момент стало вирішення проблеми подвійних витрат без контролюючого органу.

2. *Blockchain 2.0 (2014-2019)* - Поява смарт-контрактів та платформ для їх реалізації, основною та найвідомішою з який і по наш час залишається Ethereum. Ця інновація дозволила перейти до складних програмованих угод(смарт-контрактів), які починали автоматично виконуватися, при досяганні заданих умов.

3. *Blockchain 3.0 (2020-теперішній час)* - розвиток систем із підвищеною масштабованістю, енергоефективністю та здатністю взаємодії між різними блокчейнами. Визначними представниками. З'явилися нові механізми консенсусу, такі як Proof of Stake (PoS), Delegated Proof of Stake (DPoS) та інші. Визначними представниками третього покоління є Solana, Avalance, TON, та оновлений Ethereum 2.0

У сучасній економіці блокчейн починає відігравати все більш значущу роль, впливаючи на різні сектори:

- *Фінансовий сектор:* Окрім створення нового класу активів, трансформуються традиційні фінансові послуги, через розвиток децентралізації фінансів (DeFi), включаючи в себе торгівлю, кредитування та інші фінансові операції, без участі традиційних посередників.

- *Міжнародні платежі та грошові перекази:* Blockchain значно спростив та прискорив міжнародні транзакції, зменшив витрати та час проведення операцій, порівняно з банківськими переказами.

- *Ланцюги поставок та логістика:* При використанні Blockchain

підвищується прозорість, відствежуваність, та безпека в ланцюгах, що вкрай важливо для контролю якості продукції, та боротьби з контрафактом.

- *Право власності та ідентифікація:* Технологія створила нові методи для реєстрації та підтвердження права власності через NFT та децентралізовані системи ідентифікації.

- *Державне управління:* Деякі країни вже почали впроваджувати Blockchain, для оптимізації державних послуг, наприклад голосування, реєстрація майна, видача документів та інше.

- *Охорона здоров'я:* Технологія забезпечує безпечне зберігання та обмін медичними даними, що в свою чергу підвищує їх доступність, не втрачаючи при цьому конфіденційності.

Станом на 2025 рік Blockchain-технології продовжують розвиватися в напрямку вирішення ключових питань, таких як масштабованість, енергоспоживання та регуляторні аспекти. Згідно з даними Gartner, до 2025 року включно загальна додана вартість бізнесу від використання Blockchain перевищить $176 млрд, а до 2030 року має досягнути $3,1 трлн [2]. Ці прогнози підкреслюють стратегічну важливість технології для майбутнього розвитку глобальної економіки.

1.1.2 Порівняльний аналіз популярних блокчейн-мереж

Сучасний ринок криптовалюти характеризується різноманіттям мереж, кожна з яких володіє унікальними технічними характеристиками, перевагами та обмеженнями. Розглянемо ключові особливості найпопулярніших мереж.

Bitcoin (BTC) - залишається першою та найціннішою за ринковою капіталізацією криптовалютою.

- *Час блоку:* приблизно 10 хвилин
- *Пропускна здатність:* 7 транзакцій на секунду (TPS)
- *Особливості:* Висока безпека, обмежена програмованість, обмежена емісія (21 млн монет)
- *Використання:* Переважно як цифрове золото, засіб збереження вартості та платіжний засіб

Навіть не зважаючи на обмежену програмованість Bitcoin порівняно з іншими мережами, продовжує залишатися еталоном надійності та безбеки.

Ethereum (ETH) - мережа, яка перша впровадила повноцінні смарт-контракти, та залишається наразі другою за капіталізацію, після Bitcoin.

- *Час блоку:* близько 12 секунд

- *Пропускна здатність:* 15-30 TPS (основний шар), тисячі TPS у рішеннях Layer 2
- *Особливості:* Розвинена екосистема смарт-контрактів, підтримка ERC-стандартів токенів (ERC-20, ERC-721, ERC-1155 та ін.), активний розвиток рішень масштабування Layer 2 (Optimism, Arbitrum, zkSync)
- *Використання:* DeFi-додатки, NFT, децентралізовані біржі (DEX), ігри, соціальні мережі та інші dApps

Після переходу на PoS (в рамках оновлення Ethereum 2.0), значно знизилось енергоспоживання мережі, що підготувало основу для подальших оптимізацій, та потенційно дозволить досягти тисяч TPS на базовому рівні.

Tron (TRX) - високопродуктивна Blockchain-платформа, орієнтована на розважальну індустрію та цифровий контент.

- *Час блоку:* 3 секунди
- *Пропускна здатність:* близько 2000 TPS
- *Особливості:* Висока швидкість та низькі комісії, сумісність зі смарт-контрактами Ethereum (віртуальна машина TVM), система Super Representatives (27 валідаторів)
- *Використання:* Платежі, токенізація активів, стримінгові платформи, ігри, стейблкоїни

Мережа здобула велику популярність за рахунок низьких комісій, що зробило її привабливою для транзакцій зі стейблкоїнами, зокрема USDT-TRC20

TON (The Open Network) - платформа розроблена командою Telegram, в майбутньому передана спільноті.

- *Час блоку:* близько 5 секунд
- *Пропускна здатність:* потенційно мільйони TPS завдяки динамічному шардінгу
- *Особливості:* Багаторівнева архітектура, підтримка мікроплатежів, тісна інтеграція з екосистемою Telegram, власна мова програмування FunC
- *Використання:* Платежі в Telegram, децентралізовані застосунки, NFT, ігри, веб3-сервіси

TON має великий потенціал, за рахунок інтеграції з одним із найпопулярніших месенджерів - Telegram, який налічує понад 900 млн користувачів, та амбітну технічну архітектуру.

Solana (SOL) - високопродуктивна мережа з акцентом на швидкість та низьку вартість транзакцій.

- *Час блоку:* 400 мілісекунд
- *Пропускна здатність:* теоретично до 65,000 TPS
- *Особливості:* Унікальний механізм PoH для синхронізації часу, паралельна обробка транзакцій, висока продуктивність при низьких витратах
- *Використання:* DeFi, NFT-маркетплейси, ігри, децентралізовані біржі, мобільні dApps

Solana виділяється серед інших блокчейнів надзвичайно високою швидкістю транзакцій та низькими комісіями, хоча періодично і стикається з проблемами стабільності мережі.

Порівняльна таблиця характеристик розглянутих Blockchain-мереж:

Характеристика	BTC	ETH	Tron	TON	SOL
Механізм консенсусу	PoW	PoS	DPoS	PoS	PoH + PoS
Час блоку	~10 хв	~12 сек	~3 сек	~5 сек	~400 мс
Пропускна здатність	7 TPS	15-30 TPS	~2000 TPS	Потенційно мільйони	До 65,000 TPS
Середня комісія (2025)	$1-15	$0.5-5	$0.01-0.1	$0.01-0.05	$0.0001-0.001
Програмованість	Обмежена	Висока	Висока	Висока	Висока
Енергоспоживання	Високе	Низьке	Низьке	Низьке	Низьке
Масштабованість	Обмежена	Середня	Висока	Дуже висока	Висока

1.1.3 Особливості функціонування токенізованих активів (USDT, USDC) у різних екосистемах

Токенізовані активи, особливо стейблкоїни, відіграють ключову роль у сучасній криптоекономіці, забезпечуючи стабільність у плаваючому середовищі криптовалют. Серед всіх найбільшу популярність мають USDT та USDC, які чітко прив’язані до долара США.

USDT (Tether) - є найпоширенішим серед стейблкоїнів, з ринковою капіталізацією понад $110 млрд.
Особливості:

- **Багатоланцюгова реалізація:** USDT функціонує одночасно в різних блокчейн-мережах:
  - ERC20
  - TRC20
  - SOL
  - BEP20
  - та інші.
- **Забезпечення:** Tether Limited [4] заявляє, що кожен USDT забезпечений еквівалентом 1 долара США в резервах компанії, хоча структура цих резервів (готівка, комерційні папери, казначейські облігації) періодично викликає дискусії.

- **Особливості функціонування в різних мережах:**
    - ERC20 - Перша та досі популярна версія USDT, але висока вартість газу в періоди навантаження мережі, робить її менш привабливою для малих транзакцій, у порівнянні з іншими.
    - TRC20 - Найпопулярніша для щоденних переказів, через низькі комісії та швидку обробку транзакцій. За даними Tether, в мережі Tron знаходиться більше 50% всього обсягу USDT.
    - SOL - Відносно нова реалізація, яка пропонує високу швидкість та дуже низьку вартість комісії.

USDC (USD Coin) - випущений Circle [3] та біржею Coinbase, є другим за розміром стейблкоїном з капіталізацією, що сягає понад $70 млрд.

- **Багатоланцюгова реалізація:** USDT функціонує одночасно в різних блокчейн-мережах:
    - ERC20
    - TRC20
    - SOL
    - BEP20
    - та інші.
- **Забезпечення:** USDC позиціонує себе, як повністю забезпечений доларовими резервами, що регулярно проходять аудит, надаючи щомісячні звіти про резерви, які складаються переважно з готівки та короткострокових казначейських облігацій США.
- **Особливості функціонування:**
    - ERC20 **-** Основна версія USDC з найвищою ліквідністю.
    - TRC20 – USDC в даній мережі менш поширена в порівнянні з USDT, але також пропонує переваги у вигляді низьких комісій.
    - SOL - Швидко популяризується в мережі, завдяки високій швидкості та низьким комісіям.

Ключові аспекти функціонування стейблкоїнів у різних блокчейн-екосистемах:

1. Міжмережеві мости та своп-сервіси
2. Різні стандарти токенів
3. Вартість та швидкість транзакцій
4. Безпека та ризики
5. Регуляторні аспекти

Токенізовані активи, в особливості стейблкоїни, є невід'ємною частиною інфраструктури крипто-ринку, вони забезпечують проміжну ланку між традиційними фінансами та криптоекономікою

1.1.4 Статистика розвитку криптовалютного ринку та обсяги транзакцій

Криптовалютний ринок демонструє стабільне зростання, незважаючи на періодичні підйоми та спади. Аналіз статистичних даних дає змогу оцінити масштаби та динаміку розвитку даного сектору.

Загальна ринкова капіталізація

Станом на початок 2025 року загальна капіталізація крипто-ринку перевалила за $3 трлн, що є значно більше, ніж у 2021 році - $1 трлн, та на своєму історичному максимумі наприкінці 2021 року - $2,7 трлн. Основними факторами зростання є:

- Інституціональне прийняття банками, хедж-фондами та публічними компаніями
- Регуляторна визначеність у провідних економіках
- Впровадження криптовалют, як легального платіжного засобу в частині країн світу
- Розвиток інфраструктури та зручних інтерфейсів для користувачів

Розподіл капіталізації за основними криптоактивами (початок 2025 року)

Криптоактив	Приблизна капіталізація	Частка ринку
Bitcoin (BTC)	$1.2 трлн	40%
Ethereum (ETH)	$600 млрд	20%
Стейблкоїни (USDT, USDC, та ін.)	$240 млрд	8%
Solana (SOL)	$120 млрд	4%
TON	$75 млрд	2.5%
Tron (TRX)	$30 млрд	1%

Інші	$735 млрд	24.5%

Обсяги транзакцій у різних блокчейн-мережах

Щоденні обсяги транзакцій є вкрай важливим показником, для визначення активності мережі, який допомагає зрозуміти її реальне використання:

1. *Bitcoin:*
   - Середня кількість транзакцій на добу: 400,000-600,000
   - Середній добовий обсяг у вартісному вираженні: $10-15 млрд
   - Середній розмір блоку: 1.5-2 МБ
2. *Ethereum*:
   - Середня кількість транзакцій на добу: 1.2-1.5 млн
   - Загальний обсяг заблокованої вартості (TVL) у DeFi: $100+ млрд
   - Середня вартість газу: 20-30 Gwei
3. *Tron*:
   - Середня кількість транзакцій на добу: 4-6 млн
   - Обсяг переказів USDT-TRC20: $10-15 млрд щодня
   - Активних адрес: 15+ млн
4. *TON*:
   - Середня кількість транзакцій на добу: 2-3 млн
   - Швидке зростання завдяки інтеграції з Telegram
   - Активних адрес: 25+ млн
5. *Solana*:
   - Середня кількість транзакцій на добу: 30-50 млн
   - Активних програм: 1000+
   - Користувачів DeFi-додатків: 2+ млн щодня

Обсяги транзакцій стейблкоїнів

- **USDT (всі мережі):** $60-80 млрд щоденний обсяг переказів
  - USDT-TRC20 (Tron): 55-65% від загальних обсягів
  - USDT-ERC20 (Ethereum): 20-25%
  - USDT на інших мережах: 10-25%
- **USDC (всі мережі):** $20-30 млрд щоденний обсяг переказів
  - USDC-ERC20 (Ethereum): 60-70%
  - USDC-SOL (Solana): 15-20%
  - USDC на інших мережах: 10-25%

Динаміка зростання користувачів

Кількість користувачів у криптовалюті продовжує стабільно рости:

- Загальна кількість криптовалютних гаманців: понад 500 млн (2025)
- Активні користувачі: близько 200 млн
- Географічний розподіл: значне зростання у країнах, що розвиваються, особливо в Азії, Африці та Латинській Америці

Інституціональне прийняття
Важливим фактором для зростання ринку є збільшення інституціонального прийняття:

- Публічні компанії
- Фінансові установи
- Венчурні інвестиції у блокчейн-стартапи

Тенденції використання
Аналіз тенденцій використання криптовалют демонструє значне розширення сфер для застосування:

- Платежі та грошові перекази: 35% транзакцій
- Інвестиції та заощадження: 30%
- DeFi та стейкінг: 20%
- NFT та ігри: 10%
- Інші використання: 5%

Статистика показує, що криптовалютний ринок на даний момент продовжує свій розвиток, інтегруючись у традиційну фінансову систему та наше повсякденне життя. Зростання обсягів транзакцій, особливо у мережах з низькими комісіями, створює вкрай сприятливе середовище для інновацій, але нажаль і збільшує ризики використання цих платформ для незаконної діяльності, що підкреслює важливість розробки ефективних інструментів для виявлення підозрілих транзакцій.

1.2. Проблема шахрайства та зловживань у криптовалютних операціях

1.2.1. Типологія шахрайських схем у блокчейн-мережах

Розвиток Blockchain-технологій, окрім інноваційних можливостей, створив нові способи для різноманітних зловживань. Найпоширеніші типи шахрайських схем у Blockchain-мережах включають:

1. **Фішинг та соціальна інженерія**
   - Створення фальшивих сайтів
   - Розсилка шахрайських повідомлень
   - Фальшиві оголошення
   - Підміна адрес гаманців у буфері обміну
2. **Скам-проєкти та шахрайські ICO/IDO**
   - Проєкти, що запускаються для збору коштів без наміру розвитку
   - Інвестиційні піраміди під виглядом криптопроєктів
   - Rug pulls
3. **Технічні маніпуляції**
   - штучне збільшення ціни з подальшим продажем активів
   - використання інформації транзакцій в пулі для отримання переваги
   - Флеш-позики задля маніпуляцій цінами в DeFi
4. **Відмивання коштів**
   - "Міксери" та "тумблери" для приховування походження коштів
   - Крос-чейн мости для заплутування слідів
   - Багатоетапні переводи через безліч адрес
   - Використання приватних криптовалют
5. **Децентралізовані фінансові злочини**
   - Вразливості смарт-контрактів
   - Маніпуляції з оракулами цін
   - Флеш-кредитні атаки
   - Експлуатація помилок при запуску нових DeFi-протоколів
6. **Інфраструктурні атаки**
   - Фальшиві криптогаманці та розширення браузера
   - Компрометація біржових облікових записів
   - SIM-свопінг для отримання доступу до двофакторної аутентифікації
   - Атаки на криптобіржі та платформи з управління активами

Згідно з даними компанії Chainalysis, обсяг криптовалют, отриманих злочинним шляхом, у 2024 році перевищував $20 млрд, що складає приблизно 0,4% від загального обсягу транзакцій. Незважаючи на невелику частку, це значні суми, які потребують ефективних систем моніторингу та виявлення [5].

1.2.2 Аналіз найпоширеніших методів протиправної діяльності з використанням криптовалют

Криптовалюти, завдяки своїм особливостям, дуже привабливі для різноманітних видів незаконної діяльності. Розглянемо найбільш поширені способи:

Фінансування тероризму та екстремізму:

- Використання стейблкоїнів для міжнародних переказів
- Децентралізовані платформи для збору коштів
- P2P-обміни, що дозволяють обійти KYC-процедури

Торгівля забороненими товарами:

- Використання криптовалют у даркнеті
- Платежі за контрабанду та наркотики
- Складні ланцюжки транзакцій для приховування продавців та покупців

Обхід санкцій та валютних обмежень:

- Використання криптоплатежів для міжнародних переказів в обхід санкцій
- Створення підставних компаній та гаманців для таємного фінансування
- Міжланцюгові своп-операції для ускладнення відстеження

Маркери протиправної активності в транзакціях:

- Постійні переміщення коштів малими сумами
- Використання міксерів
- Операції з адресами, позначеними як пов'язані з злочинною діяльністю
- Транзакції з темних бірж
- Особливі патерни руху коштів: моментальне роздрібнення сум, консолідація з багатьох джерел

1.2.3. Економічні наслідки шахрайських операцій для користувачів та ринку в цілому

Шахрайські операції у цій сфері мають негативні наслідки як окремим користувачам, так і ринку в цілому.

Наслідки для користувачів:

- Пряма втрата коштів (фішинг і тд.)
- Втрата довіри до цифрових фінансів
- Складність доведення факту шахрайства через особливості блокчейн-транзакцій

Наслідки для ринку:

- Зниження довіри інвесторів та користувачів до криптовалютного ринку
- Волатильність ринку через масштабні шахрайські схеми
- Посилення тиску регуляторних органів, що може обмежувати інновації
- Репутаційні збитки для всієї індустрії
- Відтік інвесторів через високі ризики

За даними дослідження Cambridge Centre for Alternative Finance, щорічні втрати від шахрайства в криптосфері становлять 2-5% загальної капіталізації ринку. Особливо вразливими є нові користувачі та ринки, що розвиваються [6].

1.2.4. Правові аспекти протидії шахрайству у сфері криптовалют

Правове регулювання криптовалют та протидія шахрайству значно розвилися за останні роки, але все ще залишаються специфічні виклики

Міжнародне регулювання:

- Рекомендації FATF (Група розробки фінансових заходів боротьби з відмиванням грошей) щодо віртуальних активів
- Міжнародна співпраця правоохоронних органів (Інтерпол, Європол)

Національне регулювання:

- Різні підходи: від повної заборони до ліцензування
- Впровадження KYC та AML процедур
- Нормативні вимоги до криптовалютних бірж та обмінників
- Оподаткування криптовалютних операцій

Правові виклики:

- Технологічна складність

- Швидкий розвиток нових схем
- Юридичні обмеження для міжнародних операціях
- Конфлікт між приватністю користувачів та необхідністю контролю
- Складність доказу замислу та встановлення відповідальних осіб

Правові механізми захисту:

- Блокування підозрілих гаманців на регульованих платформах
- Маркування підозрілих адрес у публічних базах даних
- Судові процеси для повернення викрадених активів
- Застосування законодавства про кіберзлочинність

Створення систем моніторингу для виявлення підозрілих транзакцій є ключовим елементом протидії шахрайству. Технологічні рішення на кшталт Telegram-ботів для виявлення зараженої криптовалюти можуть значно підняти ефективність ідентифікації протиправних операцій.

1.3. Аналіз API-інтерфейсів для отримання даних з блокчейн-мереж

1.3.1. Огляд доступних API для роботи з різними блокчейн-мережами

Для ефективної роботи з блокчейн-даними розроблений Telegram-бот використовує комбінацію API для моніторингу транзакцій та спеціалізоване API для аналітики. Розглянемо основні API, що застосовуються в системі .

Etherscan API

API, що використовується в розробленому боті для відстеження транзакцій у мережі ERC-20 для токенів ETH, USDT, USDC.

- **Функціональність:** Надає доступ до списку транзакцій транзакцій гаманця ERC20
- **Покриття:** Ethereum та всі токени стандарту ERC-20

- **Недоліки:** Відсутність WebSocket-інтерфейсу для отримання повідомлень про нові транзакції

Blockchair API
Другий API, що використовується в системі для моніторингу транзакцій у різних мережах, таких як BTC, TRC20.

- **Функціональність:** Надає доступ до списку транзакцій транзакцій гаманця BTC, TRC20
- **Покриття:** BTC, TRX та всі токени стандарту TRC-20
- **Недоліки:** Відсутність WebSocket-інтерфейсу для отримання повідомлень про нові транзакції

Аналітичне API
Для глибокого аналізу транзакцій та гаманців на предмет підозрілої активності в системі використовується спеціалізоване API

- **Функціональність:** Аналіз транзакцій та гаманців з використанням спеціалізованих алгоритмів для виявлення підозрілих транзакцій
- **Покриття:** Підтримка всіх мереж та токенів, інтегрованих у бота
- **Ключові переваги:** Спеціалізовані алгоритми виявлення шахрайства

1.4 Висновки до першого розділу

У першому розділі було проведено комплексний аналіз теоретичної частини, технологій та методів, пов'язаних з виявленням підозрілих транзакцій у Blockchain. На основі наведених даних можна зробити наступні висновки:

1. Сучасний криптовалютний ринок характеризується різноманіттям Blockchain-мереж (Ethereum, Tron, TON, Solana, Bitcoin ...) та токенізованих активів (USDT, USDC ...), кожен з яких має свої особливості, що вимагає розробки спеціалізованих підходів до аналізу у кожній екосистемі.
2. Проблема шахрайства у криптовалютній сфері набуває все більшої актуальності з огляду на збільшення обсягів транзакцій та залучення нових користувачів та інвесторів. Існуючі типи шахрайських схем кожен день еволюціонують, що вимагає розробки адаптивних механізмів їх виявлення.
3. Аналіз доступних API сервісів для різних Blockchain-мереж показав наявність широких можливостей для отримання даних про транзакції та гаманці, проте виявив суттєві відмінності в структурі даних, обмеженнях та функціональності між різними сервісами.

4. Дослідження методології аналізу даних дозволило визначити ключові критерії та алгоритми для виявлення підозрілих транзакцій, які базуються на аналізі історії взаємодій, оцінці репутації гаманців та виявленні аномальних патернів у послідовностях транзакцій.
5. Інтеграція методів отримання даних через API з алгоритмами аналізу транзакцій та механізмами взаємодії через Telegram, є перспективною для розробки доступного та ефективного інструменту виявлення підозрілих транзакцій у різних мережах.

Таким чином, проведений аналіз підтверджує актуальність та технічну здійсненність розробки Telegram-бота для виявлення підозрілих транзакцій у різноманітних Blockchain-мережах. Визначені в цьому розділі теоретичні підходи, методи та технології формують основу для практичної реалізації системи, яка буде детально розглянута в наступних розділах.

РОЗДІЛ 2. Проектування та розробка Telegram-бота для виявлення підозрілих транзакцій

2.1. Каркас Telegram-бота

2.1.1. Аналіз бібліотек для взаємодії з Telegram API

Для розробки бота необхідно обрати бібліотеку, яка забезпечить взаємодію з Telegram API. У таблиці 2.1 представлено порівняльний аналіз популярних бібліотек для розробки ботів на базі месенджера Telegram.

Таблиця 2.1. Порівняльний аналіз бібліотек для розробки Telegram-ботів

Бібліотека	**Переваги**	**Недоліки**	**Підтримка асинхронності**
python-telegram-bot	Висока популярність, детальна	Дещо високий поріг входу для початківців	Так

	документація, підтримка всіх функцій Telegram Bot API		
Telebot	Простота використання, низький поріг входу	Обмежена підтримка деяких нових функцій Telegram Bot	Частково
aiogram	Повністю асинхронна, сучасний підхід, підтримка всіх функцій Telegram Bot API	Складніша архітектура для простих ботів	Так
Telethon	Підтримка як клієнтського API, так і Bot API	Складніший, орієнтований більше на клієнтське API	Так

Для розробки бота була обрана бібліотека aiogram, оскільки вона забезпечує:

1. Повну асинхронність, що критично для обробки великої кількості запитів одночасно
2. Високу продуктивність навіть при значному навантаженні
3. Сучасний підхід до структурування коду
4. Підтримку всіх необхідних функцій Telegram API
5. Активний розвиток бібліотеки

Aiogram використовує парадигму асинхронного програмування на основі Python asyncio, що дає змогу ефективно обробляти велику кількість запитів без блокувань основного потоку. Це особливо важливо для бота, який буде виконувати тривалі операції з блокчейн-API і одночасно підтримувати взаємодію з користувачами.

Базовий приклад структури бота з використанням aiogram

```python
from aiogram import Bot, Dispatcher, types
from aiogram.filters.command import Command
import asyncio

bot = Bot(token='TOKEN')
dp = Dispatcher()

@dp.message(Command('start'))
async def start_reg(message: types.Message) -> None:
    await message.answer("Hello!")

async def main():  1 usage
    await dp.start_polling()

if __name__ == '__main__':
    asyncio.run(main())
```

Рис 2.1.1.1 Приклад базової структури Telegram бота з aiogram

2.1.2. Визначення ключових сценаріїв використання та взаємодії з ботом

Для забезпечення ефективної взаємодії користувачів з ботом було розроблено набір ключових сценаріїв використання.

Реєстрація та авторизація користувача:

- Початкова реєстрація нового користувача
- Авторизація існуючого користувача
- Управління профілем користувача

Ручна перевірка транзакцій

- Перевірка транзакції за її хешем
- Перевірка гаманця за його адресою
- Вибір блокчейн-мережі для перевірки (USDT-ERC20, USDC-ERC20, ETH, USDT-TRC20, USDC-TRC20, TRX, TON, SOL, BTC)
- Отримання детального звіту про транзакцію/гаманець

Автоматична перевірка транзакцій

- Моніторингове меню
- Додавання гаманця на моніторинг
- Налаштування параметрів моніторингу
- Отримання автоматичних сповіщень про нові транзакції
- Управління списком гаманців для моніторингу

Перегляд аналітики та звітів

- Перегляд історії перевірок
- Експорт звітів

Налаштування та допомога

- Зміна налаштувань бота
- Отримання довідкової інформації
- Зв'язок з підтримкою

2.1.3. Схематичне створення інтуїтивно зрозумілого інтерфейсу

Інтерфейс Telegram-бота розроблено з урахуванням UX дизайну та орієнтацією на інтуїтивну зрозумілість.

При вводі стандартної команди /start в боті, йде вибір мови інтерфейсу бота, далі користувач потряпляє в головне меню.

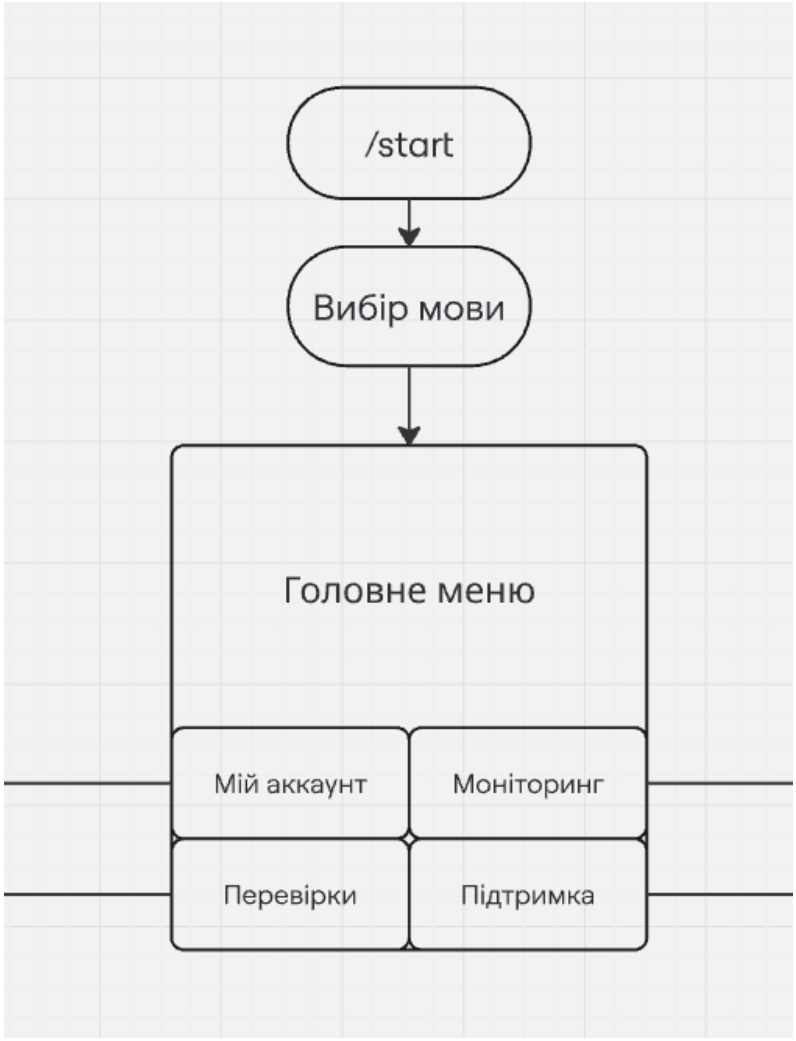


Рис. 2.1.3.1 Схематично головне меню

Дивлячись на зображення Рис. 2.1.3.1 першим пунктом меню йде “Мій аккаунт”, при натисканні на який ми потрапляємо в меню користувача, див Рис. 2.1.3.2. Наступною кнопкою в меню є Моніторинг, детальніше про яку описано на малюнку, див Рис 2.1.3.3. Перейшовши на другий ряд, ми можемо побачити можливість переходу в меню Перевірок, див Рис. 2.1.3.4.
Також в боті є Технічна підтримка. При натисканні на “Підтримка” користувач потрапляє в меню написання зверненя див Рис. 2.1.3.5.

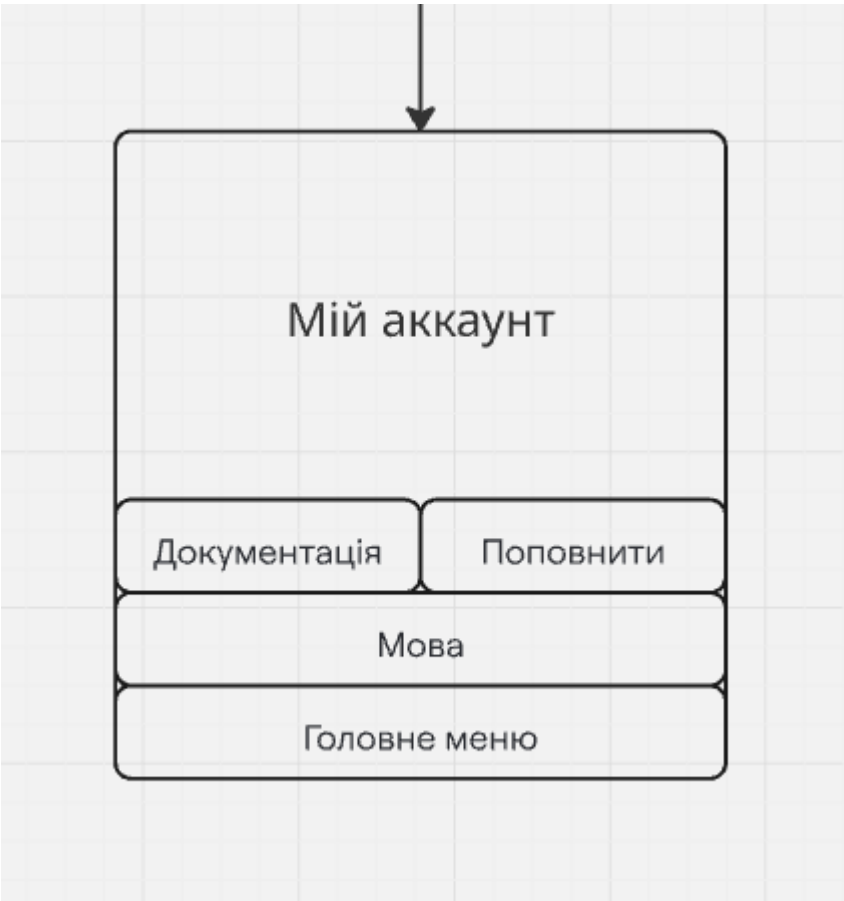


Рис. 2.1.3.2 Схематично меню “Мій аккаунт”

У цьому меню є можливість ознайомитися з документацією бота, поповнення рахунку, зміни мови інтерфусу та кнопку повернення в головне меню.

При натисканні на “Документація”, користувачу приходить повідомлення з інструкцією по користуванню ботом.

Пункт поповнити дає користовачу змогу поповнити баланс перевірок.

Наступна кнопка мова дає користовачу можливість змінити мову бота на запропоновану із списку.

Рис 2.1.3.3 Схематично меню “Моніторинг”

В меню моніторинг ми можемо побачити можливість Додавання гаманця, видалення, та перегляду списку всіх гаманців.

Рис 2.1.3.4 Схематично меню “Перевірки”

В меню перевірок ми можемо зробити перевірку транзакції, гаманця. Також користуючись наведеними кнопками ми можемо дістати PDF файл з історією усіх перевірок користувача.

Логіку перевірки Транзакцій та Гаманців можна побачити на налюнках Рис. 2.1.3.6 та Рис. 2.1.3.7 відповідно.

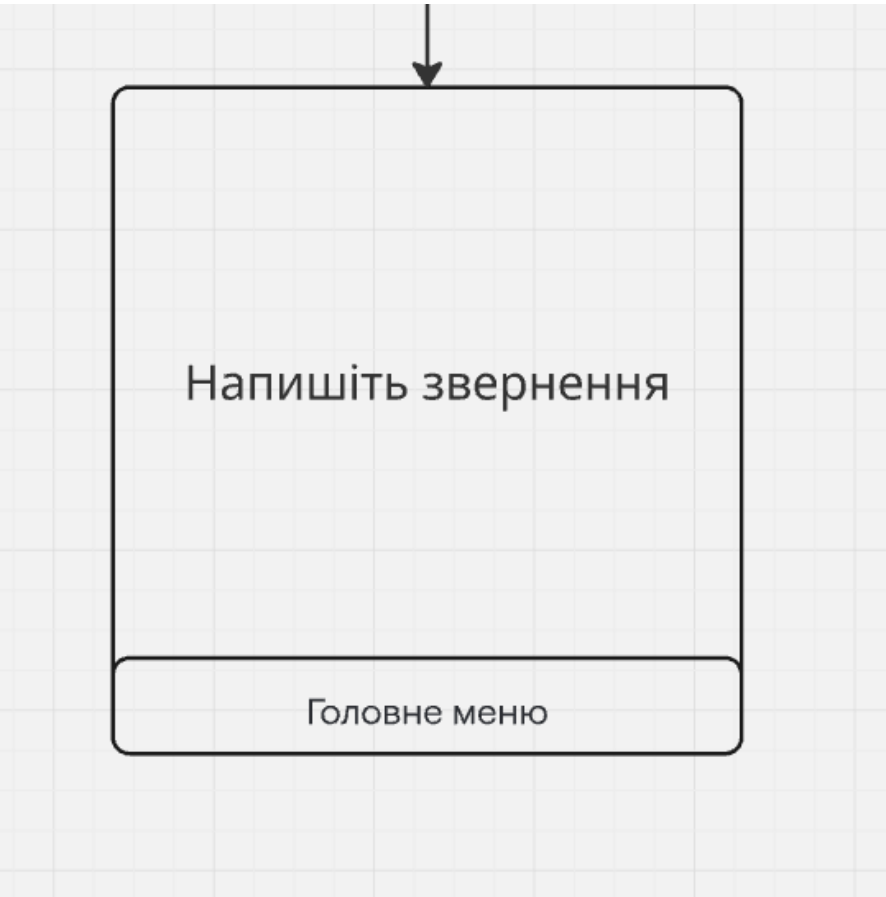


Рис 2.1.3.4 Схематично меню “Підтримка”

В данній секції користувач може написати своє звернення в Технічну Підтримку, яке далі буде надіслано Адміністратору.

Рис 2.1.3.6 Схематично меню перевірки транзакції

В меню перевірки транзакцій користувач обирає мережу, у якій хоче перевірити транзакцію, далі потрібно ввести хєш адресу транзакції, та отримати аналітику по ній.

Рис 2.1.3.7 Схематично меню перевірки гаманця

При перевірці гаманця користувачу треба лише ввести адресу гаманця, далі бот автоматично визначає мережу, та скидає аналітику по гаманцю.

2.1.4. Реалізація меню бота

1. Прописуємо команду /start, яка одразу перенаправляє на вибір мови, див Рис. Рис. 2.1.4.1

```python
@dp.message(Command('start'))  1 usage
async def start_reg(message: types.Message, state: FSMContext) -> None:
    await state.set_state(menu.choose_language)
    await choose_language(message, state)
```

Рис. 2.1.4.1

2. Далі потрібно прописати функцію для вибору мови інтерфейса, див Рис. 2.1.4.2, Рис. 2.1.4.3, Рис. 2.1.4.4

```python
@dp.message(menu.choose_language)  3 usages (2 dynamic)
async def choose_language(message: types.Message, state: FSMContext) -> None:
    try:
        if message.text == 'EN GB':
            await choose_user_language(message.from_user.id, two: 'eng')
        elif message.text == 'CN CN':
            await choose_user_language(message.from_user.id, two: 'cn')
        elif message.text == 'TR TR':
            await choose_user_language(message.from_user.id, two: 'tr')
        elif message.text == 'UA UA':
            await choose_user_language(message.from_user.id, two: 'ua')

        elif message.text == '/start':
            reg = await check_add_users(message.from_user.id)
            if reg == 'not':
                await add_users(message.from_user.id, message.from_user.username, message.from_user.username, two1: 'eng', two2: 3, asd: 0)
                try:
                    await bot.send_message(chat_id=GROUP_ID,
                                           text=f"<b>Username:</b> <u>@{await take_username(message.from_user.id)}</u>\n"
                                                f"<b>Действие:</b> <u>Первый раз зашел в бота</u>\n\n"
                                                f"<b>Количество проверок:</b> <u>{await take_checks(message.from_user.id)}</u>",
                                           parse_mode="html"
                                           )
```

Рис. 2.1.4.2

```
            except:
                pass
        else:
            await choose_user_language(message.from_user.id, two: 'eng')

    else:
        await message.answer(text="Unknown language")
        await start_reg(message)
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [
                KeyboardButton(text=f"{await take_text(message.from_user.id, dsafv: 'Ознакомлен')}")
            ]
        ],
        resize_keyboard=True,
        input_field_placeholder="Что хотите проверить?",
        selective=True
    )
    await bot.send_message(chat_id=message.from_user.id,
```

Рис. 2.1.4.3

```
    await bot.send_message(chat_id=message.from_user.id,
                           text=f"{await take_text(message.from_user.id, dsafv: 'Перед использованием BitOne Bot пожалуйста ознакомьтесь и дайте своё согласие. Ознакомится с
                               f"<a href='http://bitone.com.ua'><b><u>{await take_text(message.from_user.id, dsafv: 'тут')}</u></b></a>",
                           parse_mode='html',
                           reply_markup=keyboard)
        await state.set_state(menu.rules)
    except TypeError:
        pass
```

Рис. 2.1.4.4

3. Додаємо обробник кнопки Ознайомлений див Рис 2.1.4.5

```
@dp.message(lambda message: message.text in take_lang_list( dasda: 'Ознакомлен', ''))
async def choose_language_sel(message: types.Message, state: FSMContext) -> None:
    await start_headler(message, state)
```

Рис. 2.1.4.5

4. Після авторизації користувач перекидається в головне меню, тож додамо його, див Рис. 2.1.4.6

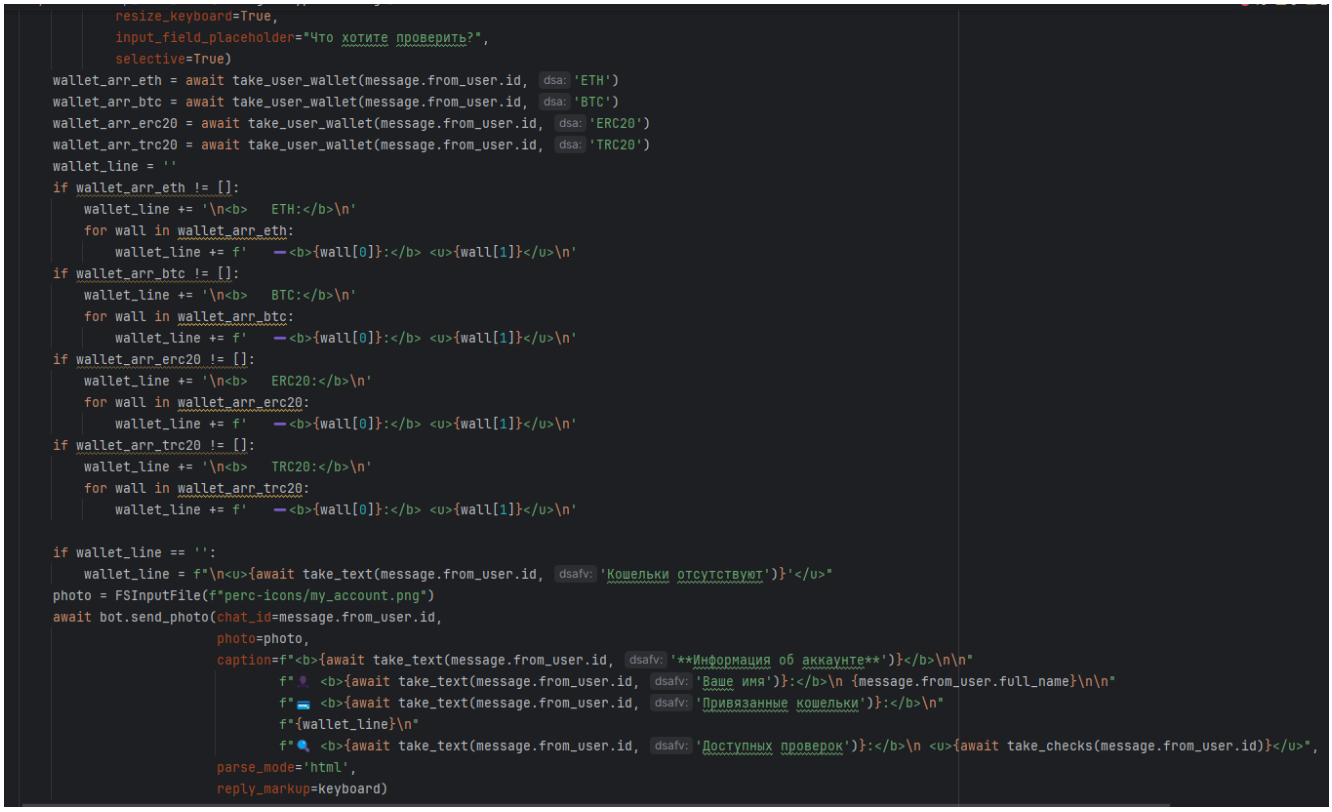
```python
@dp.message(lambda message: message.text in take_lang_list( dasda: 'Назад', ' ↩') or message.text in take_lang_list( dasda: 'Вернутся в главное меню', ' ↩'))  1 usage
async def start_headler(message: types.Message, state: FSMContext) -> None:
    await state.clear()
    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [
                KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Мой аккаунт')} ⚙"),
                KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Мониторинг')}  📈")
            ],
            [
                KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Проверки')} 🔍"),
                KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Поддержка')} 📩")
            ],
        ],
        resize_keyboard=True,
        input_field_placeholder="Что хотите проверить?",
        selective=True
    )
    await bot.send_message(chat_id=message.from_user.id,
                           text=f"{await take_text(message.from_user.id,  dsafv: 'Пожалуйста, воспользуйтесь меню под полем ввода:')}\n\n",
                           reply_markup=keyboard)
```

Рис. 2.1.4.6

5. Далі таким самим чином прописуємо меню “Мій аккаунт”, “Моніторинг”, “Перевірки”, “Підтримка”. Нижче в малюнках Рис. 2.1.4.7 та Рис. 2.1.4.8 наведено на прикладі меню “Мій аккаунт”

```python
@dp.message(lambda message: message.text in take_lang_list( dasda: 'Мой аккаунт', ' ⚙'))
async def temp_back_menu(message: types.Message) -> None:
    if str(message.from_user.id) in ADMINS:
        keyboard = ReplyKeyboardMarkup(
            keyboard=[
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Правила использования')} 📜"),
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Пополнить')} 💰")
                ],
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Язык')} 🌎")
                ],
                [
                    KeyboardButton(text=f"/{await take_text(message.from_user.id,  dsafv: 'Админ_проверка')}")
                ],
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Вернутся в главное меню')} ↩")
                ]
            ],
            resize_keyboard=True,
            input_field_placeholder="Что хотите проверить?",
            selective=True)
    else:
        keyboard = ReplyKeyboardMarkup(
            keyboard=[
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Правила использования')} 📜"),
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Пополнить')} 💰")
                ],
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Язык')} 🌎")
                ],
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id,  dsafv: 'Вернутся в главное меню')} ↩")
                ]
            ],
            resize_keyboard=True,
```

Рис. 2.1.4.7



```
        resize_keyboard=True,
        input_field_placeholder="Что хотите проверить?",
        selective=True)
wallet_arr_eth = await take_user_wallet(message.from_user.id, dsa: 'ETH')
wallet_arr_btc = await take_user_wallet(message.from_user.id, dsa: 'BTC')
wallet_arr_erc20 = await take_user_wallet(message.from_user.id, dsa: 'ERC20')
wallet_arr_trc20 = await take_user_wallet(message.from_user.id, dsa: 'TRC20')
wallet_line = ''
if wallet_arr_eth != []:
    wallet_line += '\n<b>   ETH:</b>\n'
    for wall in wallet_arr_eth:
        wallet_line += f'    ➖<b>{wall[0]}:</b> <u>{wall[1]}</u>\n'
if wallet_arr_btc != []:
    wallet_line += '\n<b>   BTC:</b>\n'
    for wall in wallet_arr_btc:
        wallet_line += f'    ➖<b>{wall[0]}:</b> <u>{wall[1]}</u>\n'
if wallet_arr_erc20 != []:
    wallet_line += '\n<b>   ERC20:</b>\n'
    for wall in wallet_arr_erc20:
        wallet_line += f'    ➖<b>{wall[0]}:</b> <u>{wall[1]}</u>\n'
if wallet_arr_trc20 != []:
    wallet_line += '\n<b>   TRC20:</b>\n'
    for wall in wallet_arr_trc20:
        wallet_line += f'    ➖<b>{wall[0]}:</b> <u>{wall[1]}</u>\n'

if wallet_line == '':
    wallet_line = f"\n<u>{await take_text(message.from_user.id, dsafv: 'Кошельки отсутствуют')}'</u>"
photo = FSInputFile(f"perc-icons/my_account.png")
await bot.send_photo(chat_id=message.from_user.id,
                     photo=photo,
                     caption=f"<b>{await take_text(message.from_user.id, dsafv: '**Информация об аккаунте**')}</b>\n\n"
                             f"👤 <b>{await take_text(message.from_user.id, dsafv: 'Ваше имя')}:</b>\n {message.from_user.full_name}\n\n"
                             f"💳 <b>{await take_text(message.from_user.id, dsafv: 'Привязанные кошельки')}:</b>\n"
                             f"{wallet_line}\n"
                             f"🔍 <b>{await take_text(message.from_user.id, dsafv: 'Доступных проверок')}:</b>\n <u>{await take_checks(message.from_user.id)}</u>",
                     parse_mode='html',
                     reply_markup=keyboard)
```

Рис. 2.1.4.8

2.2. Взаємодія із Базою Даних

2.2.1. Проектування Бази Даних для зберігання інформації

Для ефективного зберігання та управління даними, необхідними для роботи бота, спроектовано реляційну базу даних. У якості СУБД обрано MySQL.

Структура бази даних включає наступні основні таблиці:

1. users - для зберігання користувачів
2. dictionary - для зберігання перекладів
3. last_trans - додані на моніторинг гаманці
4. transaction_history - історія всіх перевірок

SQL-скрипт для створення основних таблиць:

```
CREATE TABLE IF NOT EXISTS `users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`user_id` varchar(32) NOT NULL,
`username` varchar(50) NOT NULL,
```

```
`first_name` varchar(50) NOT NULL,
`language` varchar(5) NOT NULL,
`checks` int (10) NOT NULL,
`checks_group` varchar (32),
`checks_group_name` varchar (100),
`referal_code` varchar (10),
`notifications` BOOLEAN,
`count_autochecks` INT (20) DEFAULT 100,
PRIMARY KEY (`id`));


CREATE TABLE IF NOT EXISTS `dictionary` (
`id` int (11) NOT NULL AUTO_INCREMENT,
`name` varchar(300) NOT NULL,
`eng` varchar (300) NOT NULL,
`deu` varchar (300) NOT NULL,
`spa` varchar (300) NOT NULL,
`ua` varchar (300) NOT NULL,
PRIMARY KEY (`id`));

CREATE TABLE IF NOT EXISTS `last_trans` (
`id` int (11) NOT NULL AUTO_INCREMENT,
`user_id` varchar (32) NOT NULL,
`chain` varchar (5) NOT NULL,
`name_address` varchar(20) NOT NULL,
`address` varchar (50) NOT NULL,
`last_hash` varchar (80) NOT NULL,
PRIMARY KEY (`id`));

CREATE TABLE IF NOT EXISTS `transaction_history` (
`id` int (11) NOT NULL AUTO_INCREMENT,
`user_id` varchar (32) NOT NULL,
`chain` varchar (15) NOT NULL,
`type` varchar (15) NOT NULL,
`address` varchar (50) NOT NULL,
`procent` int (3) NOT NULL,
PRIMARY KEY (`id`));
```

2.2.2. Створення інструменту взаємодії БД з ботом

Для забезпечення ефективної взаємодії між ботом та базою даних була використана бібліотека mysql-connector для зв'язку з БД.

Для початку була створена функція підключеннят до БД див Рис. 2.2.2.1

```
async def create_connection():
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host,
            user=user,
            port=port,
            passwd=password,
            database=database
        )
        print("Підключення до БД MySQL прошло успішно")
    except Error as e:
        print(f"Відбулася помилка '{e}'")
    return connection
```

Рис. 2.2.2.1

Наступним кроком були створені всі необхідні функції для редагування таблиць. Для прикладу як створюються дані функції наведено створення функції для запиту адрес користувача доданих на моніторинг, див Рис. 2.2.2.2.

```
async def take_user_wallet(user_id, chain):  4 usages
    connection = await create_connection()
    cursor = connection.cursor()
    wallet_arr = []
    try:
        cursor.execute(f"SELECT name_address, address FROM last_trans WHERE user_id = '{user_id}' AND chain = '{chain}'")
        arr = cursor.fetchall()
        wallet_arr += arr
    except Error as e:
        print(f"Произошла ошибка '{e}'")
    finally:
        cursor.close()
        connection.close()
    return wallet_arr
```

Рис. 2.2.2.2

2.3. Інтеграція API для взаємодії із Blockchain-мережами

2.3.1. Розробка модулів для взаємодії з API

Для забезпечення взаємодії з блокчейн-мережами розроблено модуль, що використовує спеціалізований API для виявлення підозрілих транзакцій та аналізу.

API, що використовується для аналізу блокчейн-транзакцій, надає два основних ендпоінти:

1. /check_address?address={wall_address}&chain={chain}&api_key={ACL_API_KEY} - для перевірки гаманця
2. /check_trans?hash={hash}&chain={chain}&api_key={ACL_API_KEY} - для перевірки транзакції за її хешем

Основна перевага такого підходу полягає в уніфікації взаємодії з різними Blockchain-мережами та отриманні стандартизованих результатів аналізу, що спрощує інтеграцію та подальшу обробку даних.

Для взаємодії з API розроблено дві прості асинхронні функції див Рис 2.3.1.1

```python
async def analyt_address(chain, wall_address):
    url = f"http://127.0.0.1:8000/check_address?address={wall_address}&chain={chain}&api_key={API_KEY}"
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            response = await resp.json()
    return response

async def analyt_trans(hash, chain):
    url = f"http://127.0.0.1:8000/check_trans?hash={hash}&chain={chain}&api_key={API_KEY}"
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            response = await resp.json()
    return response
```

Рис 2.3.1.1

Ці функції забезпечують простий та ефективний спосіб взаємодії з API для аналізу транзакцій та гаманців. Використання саме асинхронної бібліотеки для API запитів (з використанням `aiohttp`) дозволяє ефективно обробляти запити навіть при великому навантаженні на бота.

Приклад відповіді API при перевірці адреси гаманця (check_address):

{

```
"status": "Succeced",
        "error": false,
        "date": "2025-05-17 07:14:38.356250",
        "sender": "0x7b3db39d379ed60055e6507ff96fff4197f28081",
        "pip_amount": {
                "Incoming_amount": "24.0189 ETH",
                "Outgoing_amount": "23.9991 ETH",
                "len_transactions": 80
        },
        "risk": 10,
        "risk_level": "Low",
        "chain": "ETH",
        "chain_out": "ETH",
        "hot_address": {
                "Exchange": "deposit"
        },
        "last_transmissionawait": "2025-05-16 01:07:11",
        "low_risk": {
                "input": {
                        "Exchange": 53.85,
                        "Wallet": 46.15
                },
                "output": {
                        "Exchange": 100
                }
        },
        "medium_risk": {
                "Exchange | High Risc": 0.55
        },
        "high_risk": {
                "Sanctions": 1.25
        },
        "other_risk": {},
        "amount": "0.0198",
        "address_counterparty_list": [
                {
                        "name": "nexo",
                        "amount": 115120.888,
                        "percent": 50.01
```

```
            },
            {
                "name": "Unknown",
                "amount": 115075.975,
                "percent": 49.99
            }
        ]
}
```


Приклад відповіді API при перевірці транзакції (check_trans):


```
{
    "status": "Succeced",
    "error": false,
    "date": "2025-05-17 07:16:01.803053",
    "sender": "0xdb64043a31a3b237b7d9e00e21c64730d2d184c5",
    "pip_amount": "0.0001 ETH",
    "risk": 12,
    "risk_level": "Low",
    "chain": "ETH",
    "chain_out": "ETH",
    "hot_address": {},
    "last_transmissionawait": "2025-05-06 06:55:23",
    "low_risk": {
        "input": {
            "Exchange": 50,
            "Swap": 50
        },
        "output": {
            "Wallet": 100
        }
    },
    "medium_risk": {},
    "high_risk": {
        "Sanctions": 0.4
    },
    "other_risk": {},
    "amount": "0.0267",
    "address_counterparty_list": [
        {
```

```
                "name": "MetaMask",
                "amount": 51.061,
                "percent": 37.765
            },
            {
                "name": "MEXC",
                "amount": 49.947,
                "percent": 36.941
            },
            {
                "name": "Unknown",
                "amount": 34.199,
                "percent": 25.294
            }
        ],
        "transaction_hash":
"0xb68e1c4415431258bc8ffcc91a0c394c27fde4f65a761c3e93572be2a06ea895",
        "recipient": "0x9291678c1442fdff2ee2b01c0a7c1c873ba008f2"
}
```


2.3.2. Оптимізація запитів та управління обмеженнями API

Для інтеграції API з ботом реалізовано спеціальні обробники команд, що відповідають за перевірку гаманців та транзакцій. Розглянемо основні обробники та їх функціональність:

Перевірка гаманця

Процес перевірки гаманця складається з наступних етапів:

1. Користувач обирає опцію "Перевірки" -> “Перевірка гаманця”
2. Бот запитує адресу гаманця
3. Користувач вводить адресу гаманця
4. Система визначає тип блокчейн-мережі за форматом адреси за допомогою функції check_chain_from_address
5. Користувач обирає конкретний токен із запропонованих в мережі для перевірки,
6. Система викликає функцію check_wallet_main, яка використовує analyt_address для отримання аналізу гаманця
7. Результати аналізу форматуються та відправляються користувачу

Весь лянцюг обробки можна подивитися на наступних малюнках
1 этап обробки - Рис. 2.3.2.1
2 этап обробки - Рис. 2.3.2.2, Рис. 2.3.2.3,
3 этап обробки - Рис. 2.3.2.4, Рис 2.3.2.5, Рис 2.3.2.6

```python
@dp.message(lambda message: message.text in take_lang_list( text: 'Проверка кошелька',  smile: ' 💰'), menu.choose_checks_type)
async def check_wallet(message: types.Message, state: FSMContext) -> None:
    kb = [
        [KeyboardButton(text=f"{await take_text(message.from_user.id,  text_name: 'Вернутся в главное меню')} ↩")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)
    photo = FSInputFile(f"perc-icons/check_trans.png")
    await bot.send_photo(chat_id=message.from_user.id,
                         photo=photo,
                         caption=f"<b>{await take_text(message.from_user.id,  text_name: 'Введите адрес кошелька')}</b>",
                         parse_mode='html',
                         reply_markup=keyboard)
    await state.set_state(wallet.wallet_check)
```

Рис. 2.3.2.1

```python
@dp.message(wallet.wallet_check)
async def input_wall_address(message: types.Message, state: FSMContext) -> None:
    address = message.text
    await message.answer(text=f"{await take_text(message.from_user.id,  text_name: 'Ожидайте, идёт проверка')}...",
                         parse_mode='html')
    chain = await check_chain_from_address(address)
    if chain == 'TRC20':
        tokens = ['USDT', 'USDC', 'TRX']
    elif chain == 'ERC20':
        tokens = ['USDT', 'USDC', 'ETH', 'BNB']
    elif chain == 'BTC':
        tokens = ['BTC']
    elif chain == 'SOL':
        tokens = ['SOL', 'USDT-SOL', 'USDC-SOL']
    else:
        photo = FSInputFile(f"perc-icons/check_trans.png")
        await bot.send_photo(chat_id=message.from_user.id,
                             photo=photo,
                             caption=f"<b>InvalidAddress</b>",
                             parse_mode='html')
        await state.clear()
        await start_headler(message, state)

    await state.update_data(address=address)
    await state.update_data(chain=chain)

    kb = [
        [],
        [
            KeyboardButton(text=f"{await take_text(message.from_user.id,  text_name: 'Вернутся в главное меню')} ↩")
        ]
    ]
```

Рис. 2.3.2.2

```
    await state.update_data(address=address)
    await state.update_data(chain=chain)

    kb = [
        [],
        [
            KeyboardButton(text=f"{await take_text(message.from_user.id, text_name: 'Вернутся в главное меню')} ↩")
        ]
    ]
    for token in tokens:
        kb[0].append(KeyboardButton(text=f"{token}"))
    keyboard = ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)
    photo = FSInputFile(f"perc-icons/check_trans.png")
    await bot.send_photo(chat_id=message.from_user.id,
                         photo=photo,
                         caption=f"<b>Select Token:</b>",
                         parse_mode='html',
                         reply_markup=keyboard)
    await state.set_state(wallet.wallet_check_in_chain)
```

Рис. 2.3.2.3

```
@dp.message(lambda message: message.text in ['ETH', 'USDT', 'BTC', 'TRX', 'USDC', 'BNB', 'SOL', 'USDT-SOL', 'USDC-SOL'], wallet.wallet_check_in_chain)
async def wallet_in_chain(message: types.Message, state: FSMContext) -> None:
    state_data = await state.get_data()
    token = message.text
    wall_address = state_data['address']
    chain = state_data['chain']

    if chain == 'TRC20' or chain == 'ERC20':
        if token == 'USDT' or token == 'USDC':
            out_token = f"{token}-{chain}"
        else:
            out_token = token
    else:
        out_token = token

    kb = [
        [
            KeyboardButton(text=f"{await take_text(message.from_user.id, text_name: 'Вернутся в главное меню')} ↩")
        ]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)

    await message.answer(text=f"{await take_text(message.from_user.id, text_name: 'Ожидайте, идёт проверка')}...",
                         reply_markup=keyboard)

    json = await check_wallet_main(wall_address, out_token, message.from_user.id)
    await mess_low_checks(message.from_user.id)
```

Рис. 2.3.2.4

```
if json['error']:
    await message.answer(text=f"{json['final_text']}",
                         parse_mode='html')
elif json['status'] == 'sanctions':
    await message.answer(text=f"{json['final_text']}",
                         parse_mode='html')
    if message.from_user.id not in DONT_MESS_GROUP:
        await bot.send_message(text=f"{json['adm_text']}",
                               chat_id=GROUP_ID,
                               parse_mode='html')
else:
    if 'adm_text_alarm' in json:
        await message.answer(chat_id=GROUP_ID,
                             text=f"{json['adm_text_alarm']}",
                             parse_mode='html')

    try:
        photo = FSInputFile(f"perc-icons/{json['procent']}.png")
        builder = InlineKeyboardBuilder()
        builder.add(types.InlineKeyboardButton(
            text=f"{await take_text(message.from_user.id, text_name: 'Полная информация')}",
            callback_data="detail_wall_eth")
        )
        await callback_data_add(wall_address, out_token)
        if 'final_text_2_page' not in json:
            await bot.send_photo(chat_id=message.from_user.id,
                                 photo=photo,
                                 caption=json['final_text'],
                                 parse_mode='html',
                                 reply_markup=builder.as_markup())
            if 'adm_text' in json:
```

Рис. 2.3.2.5

```
                                 reply_markup=builder.as_markup())
            if 'adm_text' in json:
                await bot.send_photo(chat_id=GROUP_ID,
                                     photo=photo,
                                     caption=json['adm_text'],
                                     parse_mode='html')
        else:
            await bot.send_photo(chat_id=message.from_user.id,
                                 photo=photo,
                                 caption=json['final_text'],
                                 parse_mode='html',
                                 reply_markup=builder.as_markup())
            await message.answer(text=json['final_text_2_page'],
                                 parse_mode='html')
            if 'adm_text' in json:
                await bot.send_photo(chat_id=GROUP_ID,
                                     photo=photo,
                                     caption=json['adm_text'],
                                     parse_mode='html')
                await bot.send_message(chat_id=GROUP_ID,
                                       text=json['adm_text_2_page'],
                                       parse_mode='html')
    except TypeError:
        await message.answer(text='ERROR',
                             parse_mode='html')

await state.clear()
await start_headler(message, state)
```

Рис. 2.3.2.6

Перевірка транзакції

Процес перевірки транзакції схожий до перевірки гаманця:

1. Користувач обирає опцію "Перевірки" -> “Перевірка транзакції”
2. Бот запитує мережу в якій була проведена транзакція
3. Користувач вибирає мережу із запропонованих
4. Бот запитує хеш транзакції
5. Користувач надсилає хеш адресу транзакції
6. Система викликає функцію check_transaction_hash, яка використовує analyt_trans для отримання аналізу транзакції
7. Результати аналізу форматуються та відправляються користувачу

Весь лянцюг обробки можна подивитися на наступних малюнках
1 этап обробки - Рис. 2.3.2.7, Рис. 2.3.2.8
2 этап обробки - Рис. 2.3.2.9
3 этап обробки - Рис. 2.3.2.10, Рис 2.3.2.11, Рис 2.3.2.12

```python
@dp.message(lambda message: message.text in ['USDT-TRC20', 'BTC', 'USDT-ERC20', 'ETH', 'USDC', 'TRX'], transaction.transaction_check)
async def check_transaction(message: types.Message, state: FSMContext) -> None:
    chain = message.text
    if chain == 'USDC':
        await state.update_data(coin=chain)
        keyboard = ReplyKeyboardMarkup(
            keyboard=[
                [
                    KeyboardButton(text="TRC20"),
                    KeyboardButton(text="ERC20")
                ],
                [
                    KeyboardButton(text=f"{await take_text(message.from_user.id, text_name: 'Вернутся в главное меню')} ↩")
                ]
            ],
            resize_keyboard=True,
            input_field_placeholder="Что хотите проверить?",
            selective=True
        )
        photo = FSInputFile(f"perc-icons/check_trans.png")
        await bot.send_photo(chat_id=message.from_user.id,
                             photo=photo,
                             caption=f"{await take_text(message.from_user.id, text_name: 'Выберите подходящую сеть и проверьте кошелек, чтобы рассчитать риск со
                             parse_mode='html',
                             reply_markup=keyboard)
        await state.set_state(transaction.usdc_chain)
    else:
```

Рис. 2.3.2.7

```python
    await state.set_state(transaction.usdc_chain)
else:
    await state.update_data(chain=chain)
    kb = [
        [KeyboardButton(text=f"{await take_text(message.from_user.id, text_name: 'Вернутся в главное меню')} ↩")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)
    photo = FSInputFile(f"perc-icons/check_trans.png")
    await bot.send_photo(chat_id=message.from_user.id,
                         photo=photo,
                         caption=f"<b>{await take_text(message.from_user.id, text_name: 'Введите код транзакции')} {chain}:</b>",
                         reply_markup=keyboard,
                         parse_mode='html')
    await state.set_state(transaction.check_tx_eth)
```

Рис. 2.3.2.8

```python
@dp.message(lambda message: message.text in ['TRC20', 'ERC20'], transaction.usdc_chain)
async def check_transaction(message: types.Message, state: FSMContext) -> None:
    chain = message.text
    state_data = await state.get_data()
    coin = state_data['coin']
    out_chain = f"{coin}-{chain}"
    await state.update_data(chain=out_chain)
    kb = [
        [KeyboardButton(text=f"{await take_text(message.from_user.id, text_name: 'Вернутся в главное меню')} ↩")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)
    photo = FSInputFile(f"perc-icons/check_trans.png")
    await bot.send_photo(chat_id=message.from_user.id,
                         photo=photo,
                         caption=f"<b>{await take_text(message.from_user.id, text_name: 'Введите код транзакции')} {out_chain}:</b>",
                         reply_markup=keyboard,
                         parse_mode='html')
    await state.set_state(transaction.check_tx_eth)
```

Рис. 2.3.2.9

```python
@dp.message(transaction.check_tx_eth)
async def check_transaction(message: types.Message, state: FSMContext) -> None:
    tx_hash = message.text
    state_data = await state.get_data()
    chain = state_data['chain']
    kb = [
        [KeyboardButton(text=f"{await take_text(message.from_user.id, text_name: 'Вернутся в главное меню')} ↩")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)
    await message.answer(text=f"{await take_text(message.from_user.id, text_name: 'Ожидайте, идёт проверка')}...",
                         reply_markup=keyboard)

    json = await check_transaction_hash(tx_hash, chain, message.from_user.id)
    await mess_low_checks(message.from_user.id)

    if json['error']:
        await message.answer(text=f"{json['final_text']}",
                             parse_mode='html')
    elif json['status'] == 'sanctions':
        await message.answer(text=f"{json['final_text']}",
                             parse_mode='html')
        if message.from_user.id not in DONT_MESS_GROUP:
            await bot.send_message(text=f"{json['adm_text']}",
                                   chat_id=GROUP_ID,
                                   parse_mode='html')
    else:
        if 'adm_text_alarm' in json:
            await message.answer(chat_id=GROUP_ID,
                                 text=f"{await take_text(message.from_user.id, text_name: 'Ожидайте, идёт проверка')}...",
                                 parse_mode='html')

        else:
```

Рис. 2.3.2.10

```
        else:
            if 'adm_text_alarm' in json:
                await message.answer(chat_id=GROUP_ID,
                                     text=f"{json['adm_text_alarm']}",
                                     parse_mode='html')

            photo = FSInputFile(f"perc-icons/{json['procent']}.png")
            builder = InlineKeyboardBuilder()
            builder.add(types.InlineKeyboardButton(
                text=f"{await take_text(message.from_user.id, text_name: 'Полная информация')}",
                callback_data="detail_trans")
            )
            await callback_data_add(json['address'], chain)
            if 'final_text_2_page' not in json:
                await bot.send_photo(chat_id=message.from_user.id,
                                     photo=photo,
                                     caption=json['final_text'],
                                     parse_mode='html',
                                     reply_markup=builder.as_markup())
                if 'adm_text' in json:
                    await bot.send_photo(chat_id=GROUP_ID,
                                         photo=photo,
                                         caption=json['adm_text'],
                                         parse_mode='html')
            else:
                await bot.send_photo(chat_id=message.from_user.id,
                                     photo=photo,
                                     caption=json['final_text'],
                                     parse_mode='html',
                                     reply_markup=builder.as_markup())
                await message.answer(text=json['final_text_2_page'],
                                     parse_mode='html')
                if 'adm_text' in json:
```

Рис 2.3.2.11

```
        else:
            await bot.send_photo(chat_id=message.from_user.id,
                                 photo=photo,
                                 caption=json['final_text'],
                                 parse_mode='html',
                                 reply_markup=builder.as_markup())
            await message.answer(text=json['final_text_2_page'],
                                 parse_mode='html')
            if 'adm_text' in json:
                await bot.send_photo(chat_id=GROUP_ID,
                                     photo=photo,
                                     caption=json['adm_text'],
                                     parse_mode='html')
                await bot.send_message(chat_id=GROUP_ID,
                                       text=json['adm_text_2_page'],
                                       parse_mode='html')

    await state.clear()
    await start_headler(message, state)
```

Рис 2.3.2.12

РОЗДІЛ 3: ТЕСТУВАННЯ TELEGRAM-БОТА

У даному розділі представлено результати розгортання та тестування розробленого Telegram-бота.

3.1. Розгортання і впровадження системи

3.1.1. Налаштування серверного середовища та розгортання бота

Для забезпечення стабільної роботи Telegram-бота було обрано сервер з Ubuntu.

Розгортання серверної частини бота відбувалося за наступним алгоритмом:

1. Оновлення системи та встановлення необхідних залежностей, див Рис 3.1.1.1

```
sudo apt update
sudo apt upgrade -y
sudo apt install python3.10 python3-pip python3.10-venv nginx supervisor -y
```

Рис 3.1.1.1

2. Створення віртуального середовища Python та встановлення бібліотек, див Рис 3.1.1.2

```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

Рис 3.1.1.2

3. Налаштування Supervisor для забезпечення автоматичного перезапуску бота у випадку помилок, див Рис 3.1.1.3

```
[program:telegram_bot]
command=/home/user/venv/bin/python /home/user/bot/main.py
directory=/home/user/bot
user=user
autostart=true
autorestart=true
stderr_logfile=/var/log/telegram_bot.err.log
stdout_logfile=/var/log/telegram_bot.out.log
```

Рис 3.1.1.3

Така конфігурація серверного середовища забезпечує:

- Автоматичний перезапуск бота у випадку збоїв
- Логування всіх подій для подальшого аналізу та оптимізації
- Високу доступність системи 24/7

3.1.2. Процес реєстрації бота у Telegram

Для створення бота в Telegram було використано BotFather — офіційного бота Telegram для керування ботами. Процес реєстрації включав наступні кроки:

- Отримання токена бота через BotFather, див Рис 3.1.2.1
- Налаштування опису та зображення бота для покращення користувацького досвіду, див Рис 3.1.2.2 та Рис 3.1.2.3

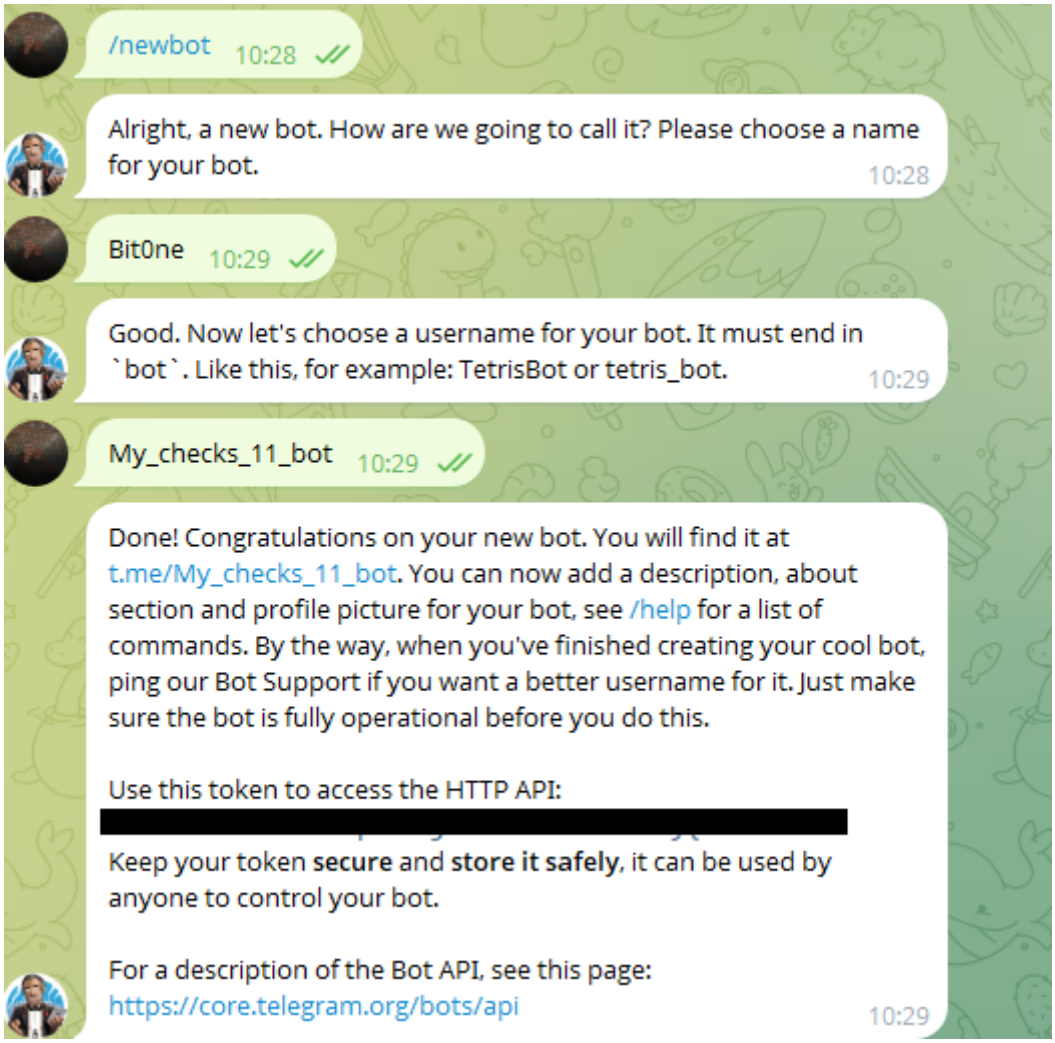


Рис 3.1.2.1

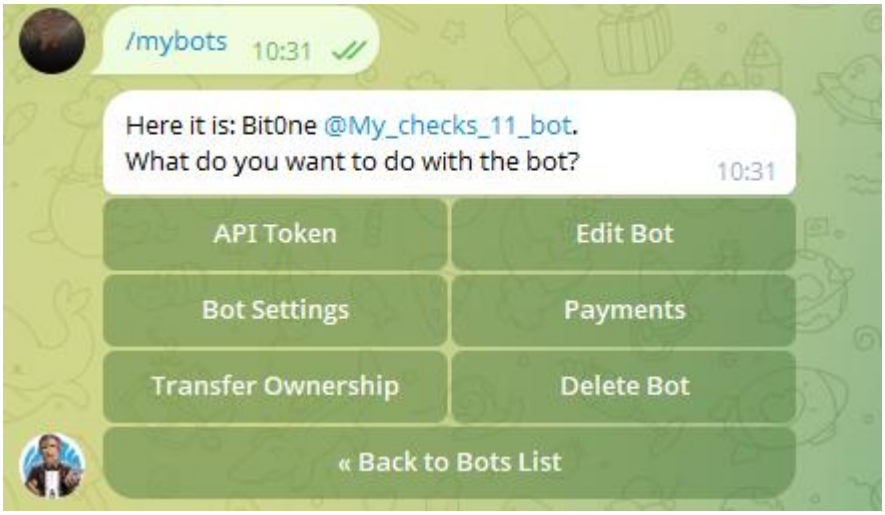


Рис 3.1.2.2

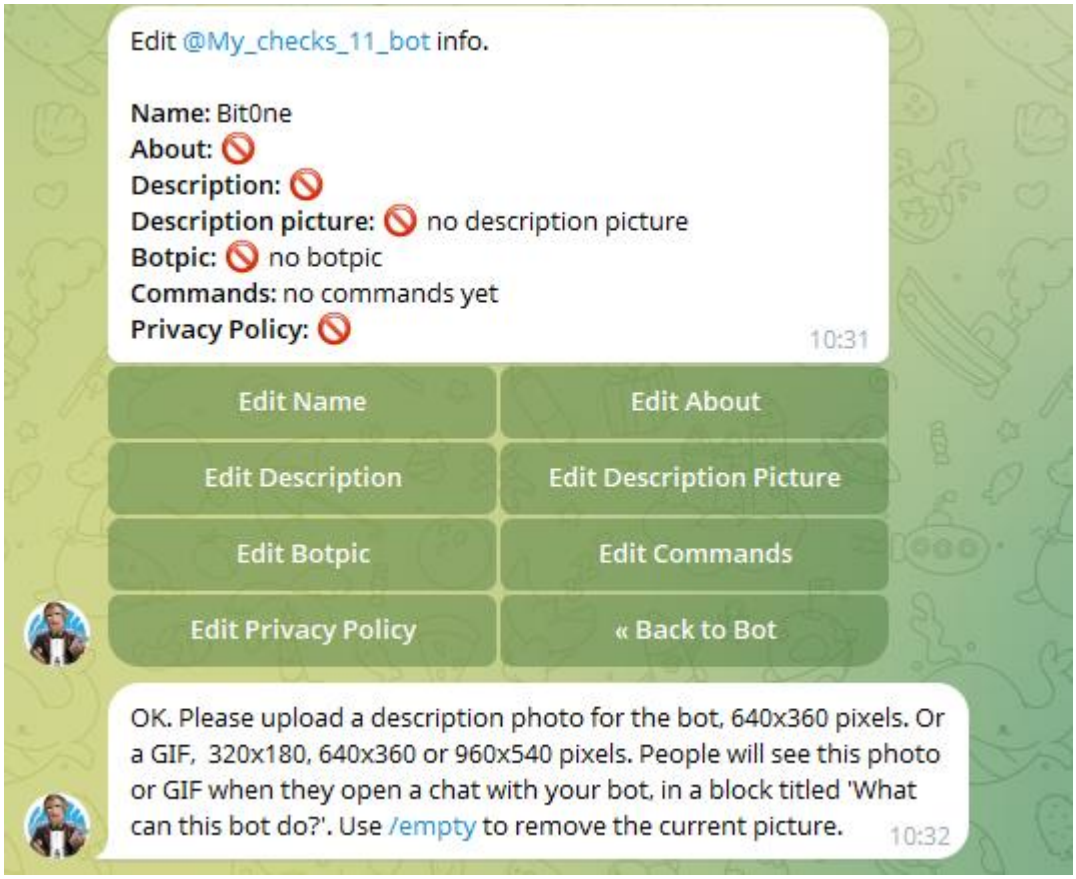


Рис 3.1.2.3

3.1.3. Опис інтерфейсу бота та інструкцій для користувачів

Інтерфейс розробленого Telegram-бота спроектовано з урахуванням UX дизайну та особливостей Telegram.

На Рис. 3.1.3.1 представлено головне меню бота, яке відображається після команди /start та вибору мови інтерфейсу.

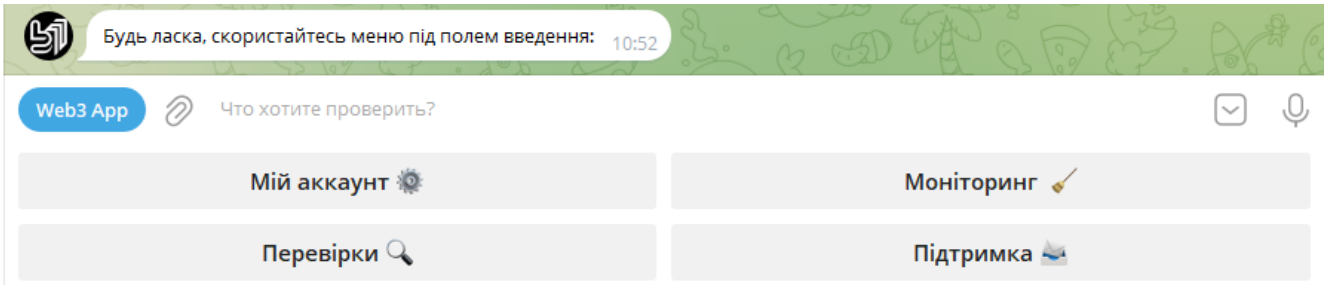


Рис. 3.1.3.1

При переході у розділ Мій аккаунт, див Рис. 3.1.3.2, бот надсилає інформацію про користувача - його ім’я, гаманці додані на перевірку, та кількість доступних перевірок.
На клавіатурі є Документація, поповнення, мова, та повернення в головне меню.

1. Кнопка Документація надсилає користувачу правила користування ботом.
2. Кнопка Поповнити перекидаю в меню поповнення балансу перевірок.
3. Кнопка Мова доволяє користовачу змінити мову інтерфейсу.
4. Кнопка Повернутися в головне меню, повертає користувача в головне меню вибору дії.

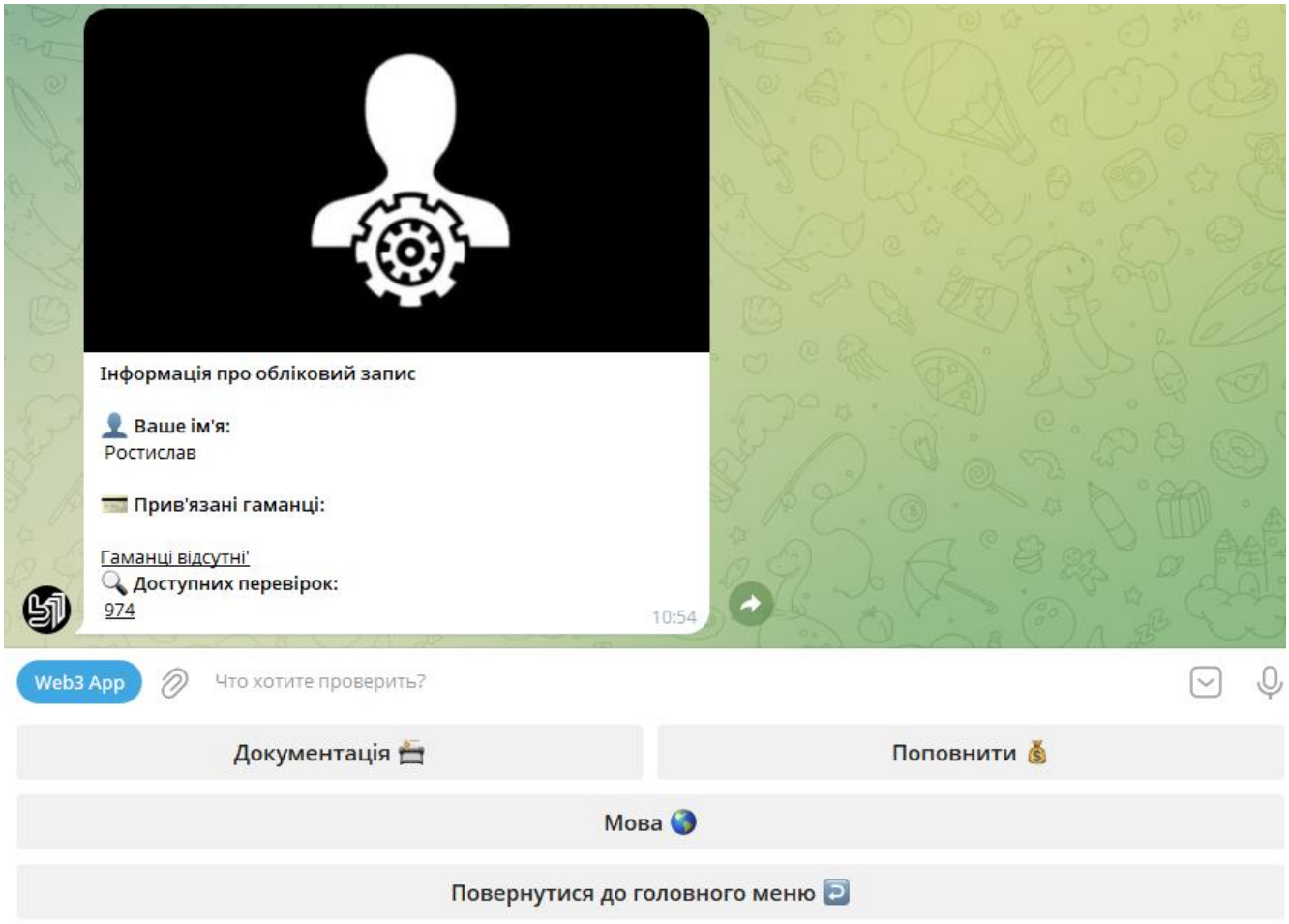


Рис. 3.1.3.2

Наступний розділ Моніторинг, див Рис 3.1.3.3, дає користовачу можливість керувати гаманцями на моніторингу.
На клавіатурі є кнопки Мої гаманці, Додати гаманець, Видалити гаманець, та Повернутися в головне меню

1. Кнопка Мої гаманці надсилає користовачу список доданих їм гаманців на моніторинг.
2. Кнопка Додати гаманець, дає користовачу додати новий гаманець на моніторинг, ввівши його адресу та назву.
3. Кнопка Видалити гаманець, видаляє гаманець з моніторингу, та надалі нові транзакції по ньому не будуть надсилатися користовачу.
4. Кнопка Повернутися в головне меню, повертає користувача в головне меню бота

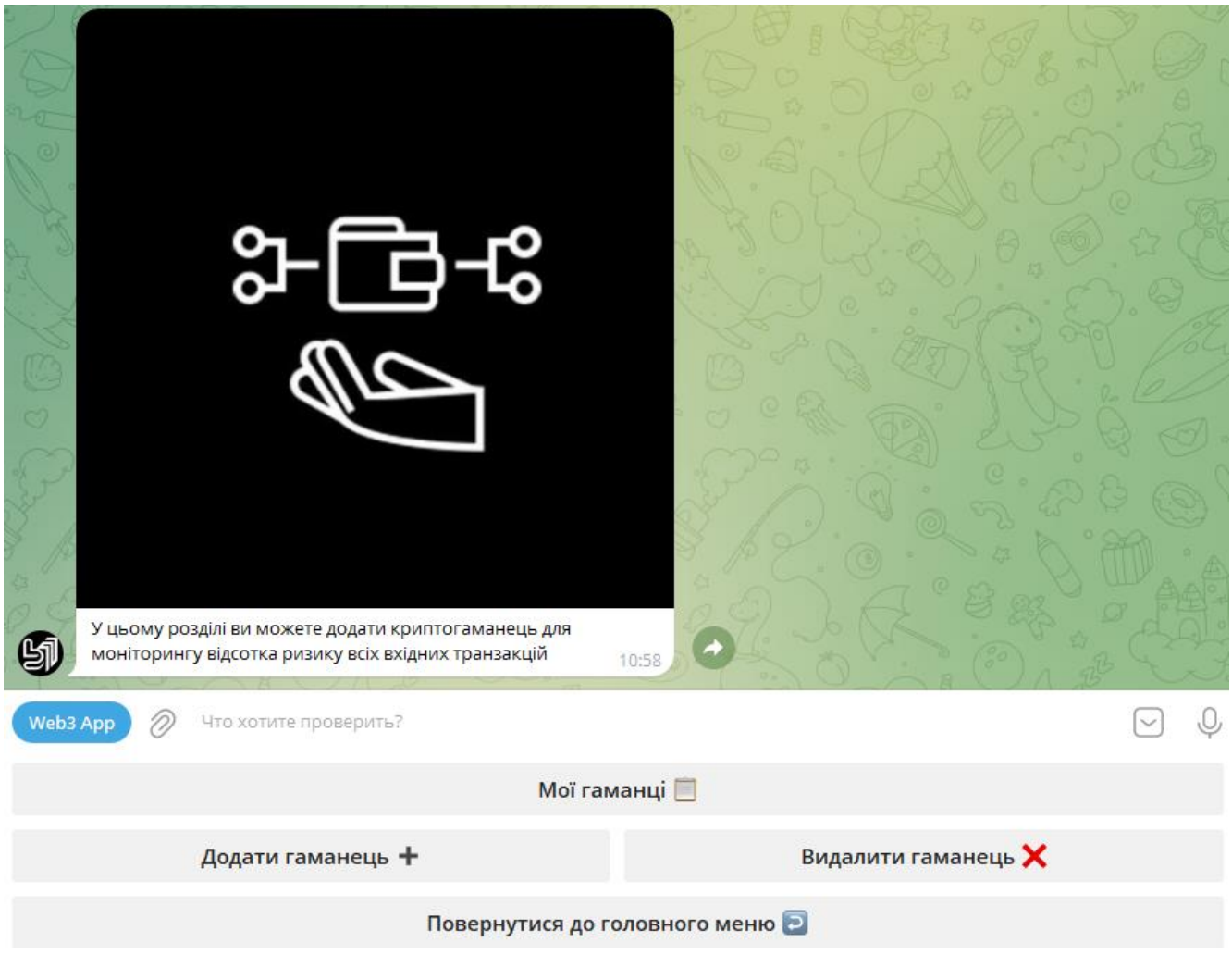


Рис 3.1.3.3

Розділ Перевірки перенаправляє пористувача в меню перевірок, див Рис. 3.1.3.4

На клавіатурі є кнопки Перевірка транзакції, Перевірка гаманця, Історія перевірок, та Повернутися до головного меню

1. Кнопка перевірка транзакції, дає користовачу можливість перевірити транзакцію.
2. Кнопка Перевірка гаманця, переадресує користувача на перевірки гаманців.
3. Кнопка Історія перевірок надсилає користувачу PDF файл з повною історією його перевірок.
4. Кнопка повернутися до головного меню, повертає користувача у головне меню бота

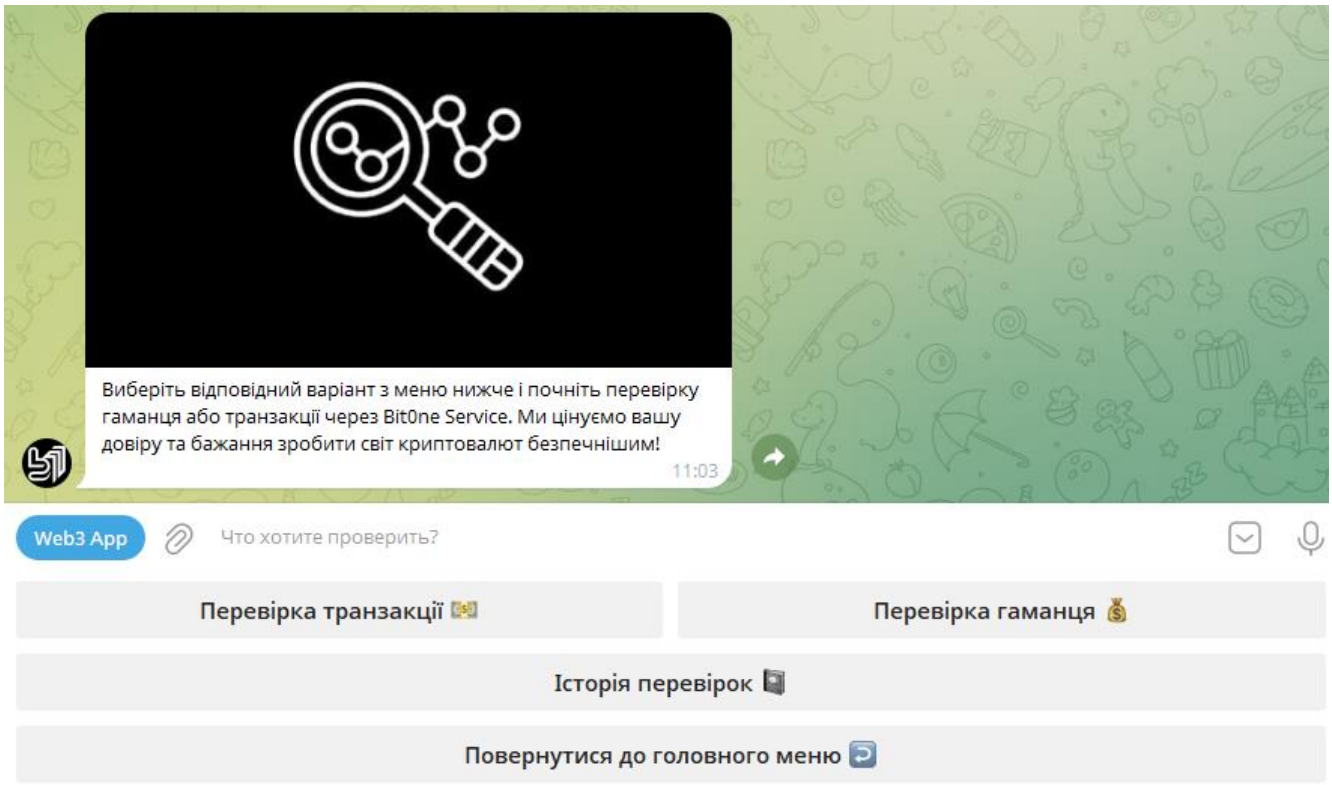


Рис. 3.1.3.4

Останній розділ підтримка дає користувачу написати в Технічну Підтримку бота. При натисканні на дану кнопку, див Рис. 3.1.3.4, користувач пише своє повідомлення, яке надсилається адміністратору, на яке адмін може відповісти у бота, або написати користувачу в особисті повідомлення.

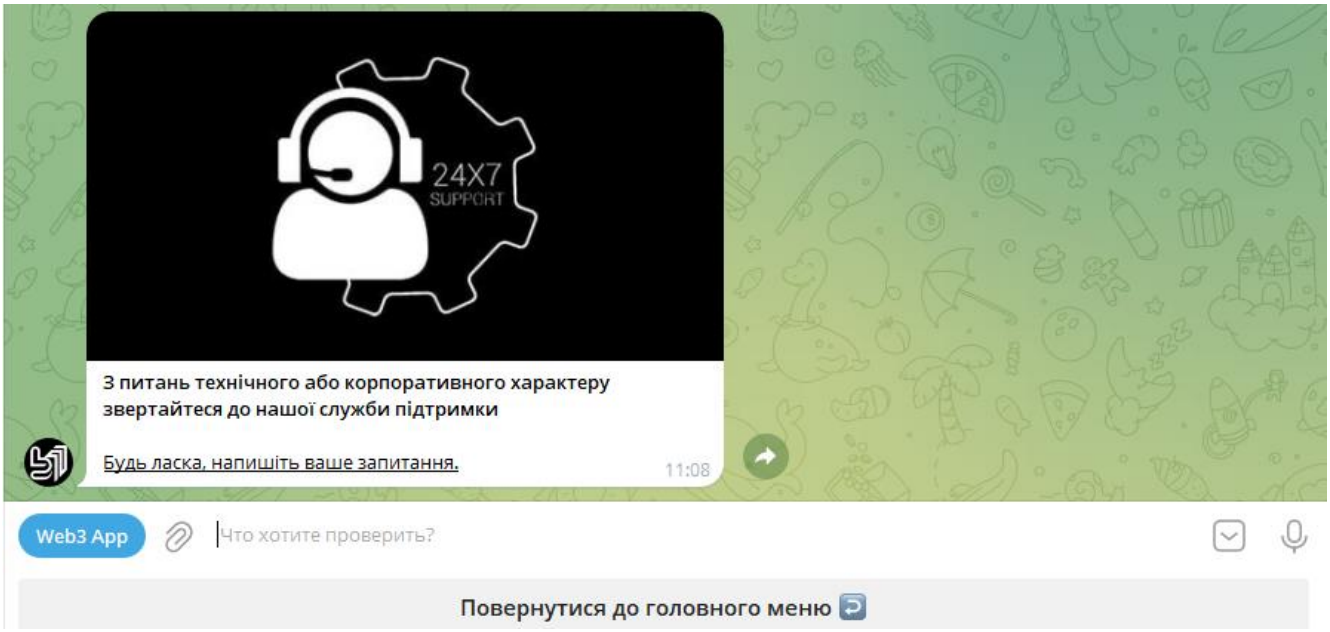


Рис. 3.1.3.5

Розроблений інтерфейс забезпечує інтуїтивно зрозумілу навігацію та ефективну взаємодію з ботом, що підтверджено результатами тестування користувацького досвіду.

3.2. Тестування функціональності Telegram-бота

3.2.1. Сценарії тестування перевірки транзакцій

Тестування функціоналу перевірки транзакцій проводилося за наступними сценаріями:

1. перевірка валідної транзакції:
    a. Вибір мережі для перевірки
    b. Введення коректного хешу транзакції
    c. Перевірка коректності відображення результату аналізу
2. перевірка невалідного хешу транзакції:
    a. Вибір мережі
    b. Введення некоректного хешу транзакції
    c. Перевірка відображення повідомлення про помилку

Нижче наведені малюнки Рис. 3.2.1.1, Рис. 3.2.1.2, Рис. 3.2.1.3, на них можна побачити процес перевірки транзакції, та кінцевий результат.

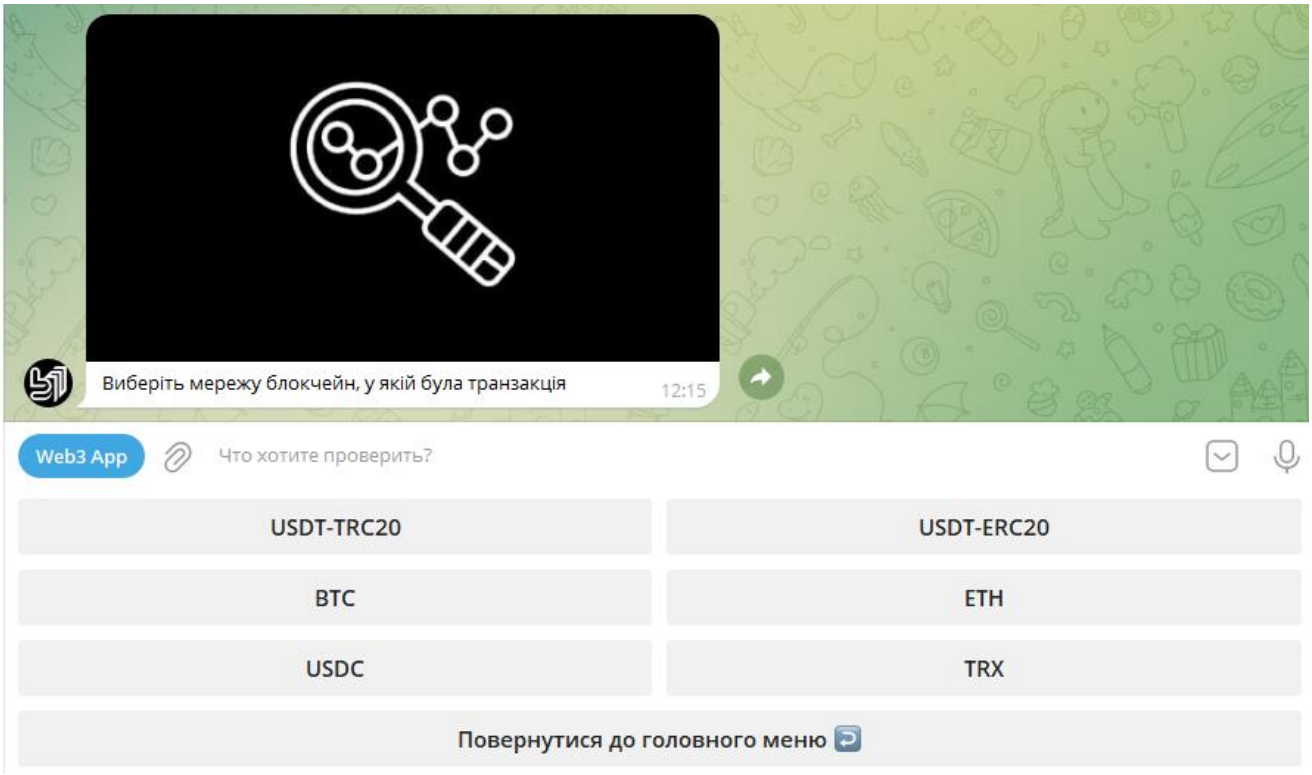


Рис. 3.2.1.1

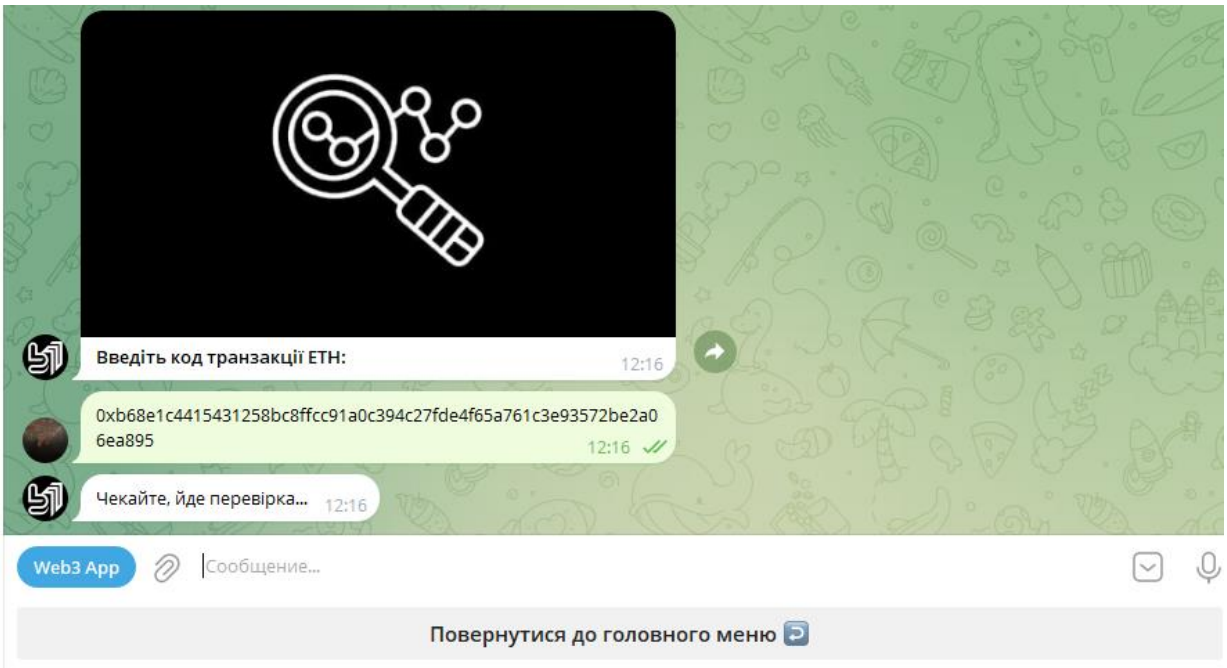


Рис. 3.2.1.2

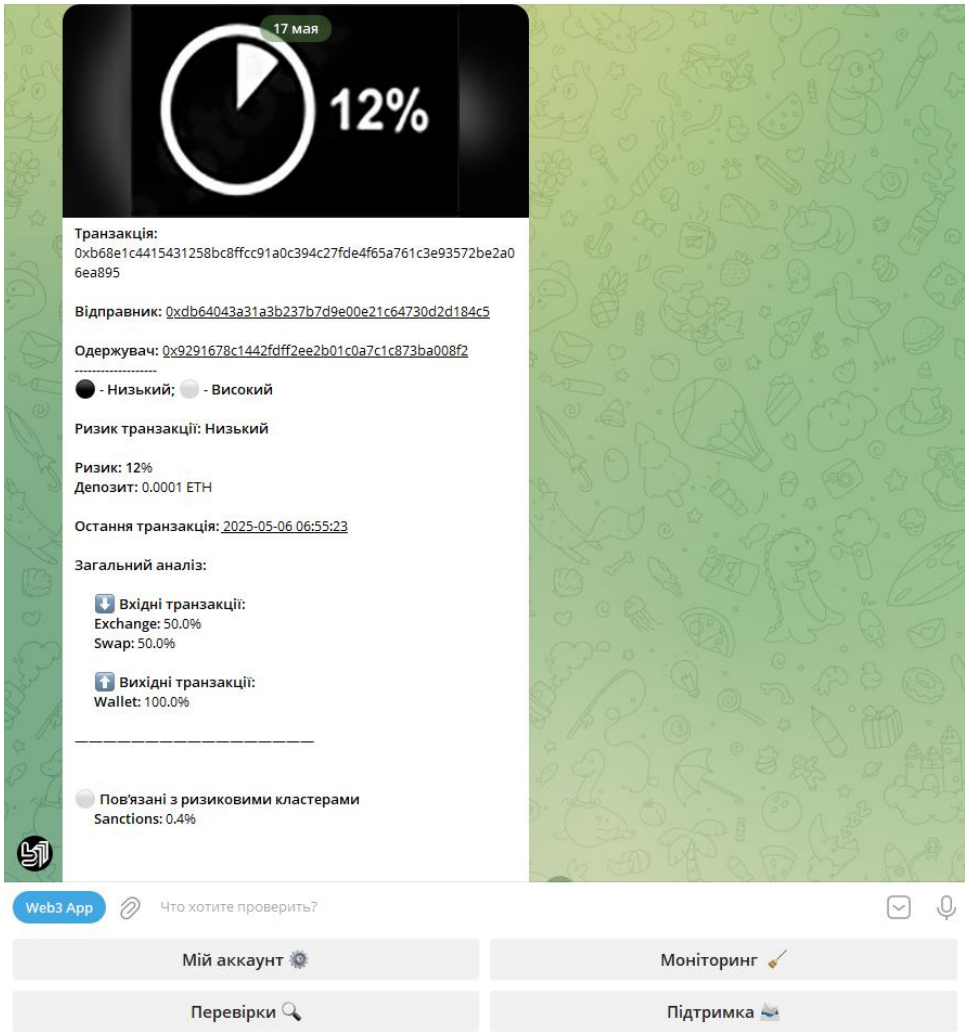


Рис. 3.2.1.3

Далі спробуємо ввести невалідний хеш транзакції.
Як видно на Рис. 3.2.1.4, невалідних хеш не проходить, та видає попередження.

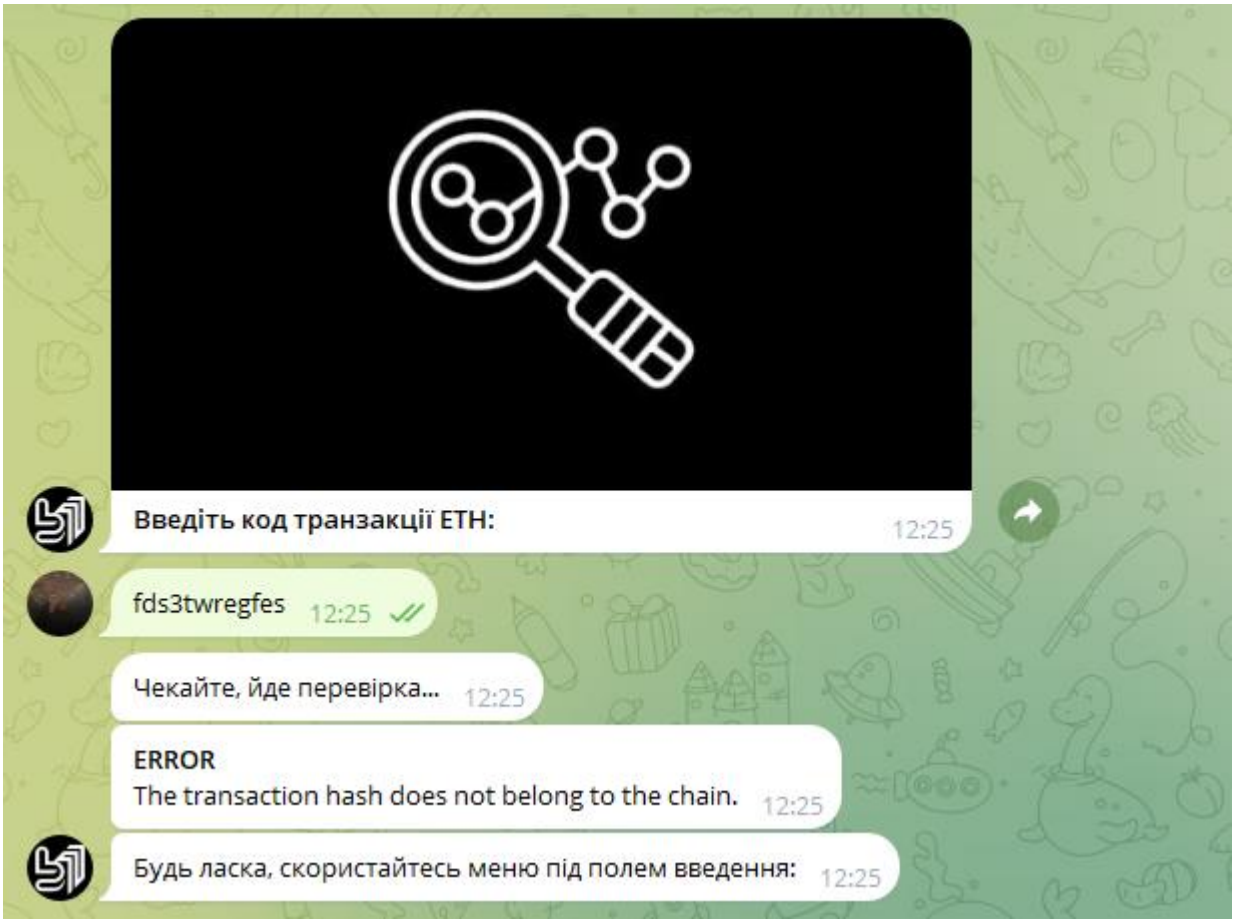


Рис. 3.2.1.4

Таблиця 3.2.1 Результати тестування перевірки транзакцій у різних мережах

Мережа	Базовий сценарій	Сценарій з невалідним хешем	Результат
ETH	Успішно	Успішно	Успішно
USDT-ERC20	Успішно	Успішно	Успішно
USDC-ERC20	Успішно	Успішно	Успішно
TRX	Успішно	Успішно	Успішно
USDT-TRC20	Успішно	Успішно	Успішно
USDC-TRC20	Успішно	Успішно	Успішно
TON	Успішно	Успішно	Успішно
BTC	Успішно	Успішно	Успішно
SOL	Успішно	Успішно	Успішно

За результатми тестування у мережах приведених в Таблиці 3.2.1, можемо зробити висновок, що перевірки транзакцій працюють корректно.

3.2.2. Сценарії тестування перевірки гаманця

Тестування функціоналу перевірки гаманців проводилося за наступними сценаріями:

3. перевірка валідної адреси гаманця:
    a. Введення коректного адресу гаманця
    b. Вибір токену для перевірки
    c. Перевірка коректності відображення результату аналізу
4. перевірка невалідної адреси гаманця:
    a. Введення некоректного хешу транзакції
    b. Перевірка відображення повідомлення про помилку

Нижче наведені малюнки Рис. 3.2.2.1, Рис. 3.2.2.2, Рис. 3.2.2.3, на них можна побачити процес перевірки гаманця, та кінцевий результат.

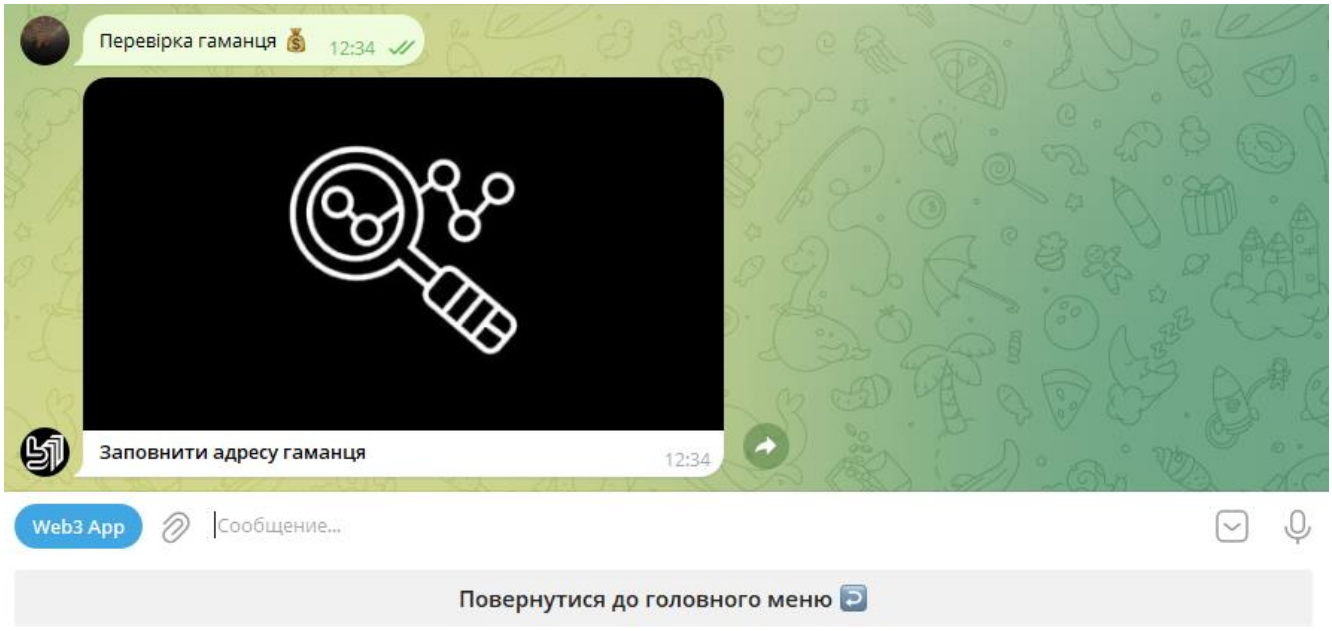


Рис. 3.2.2.1

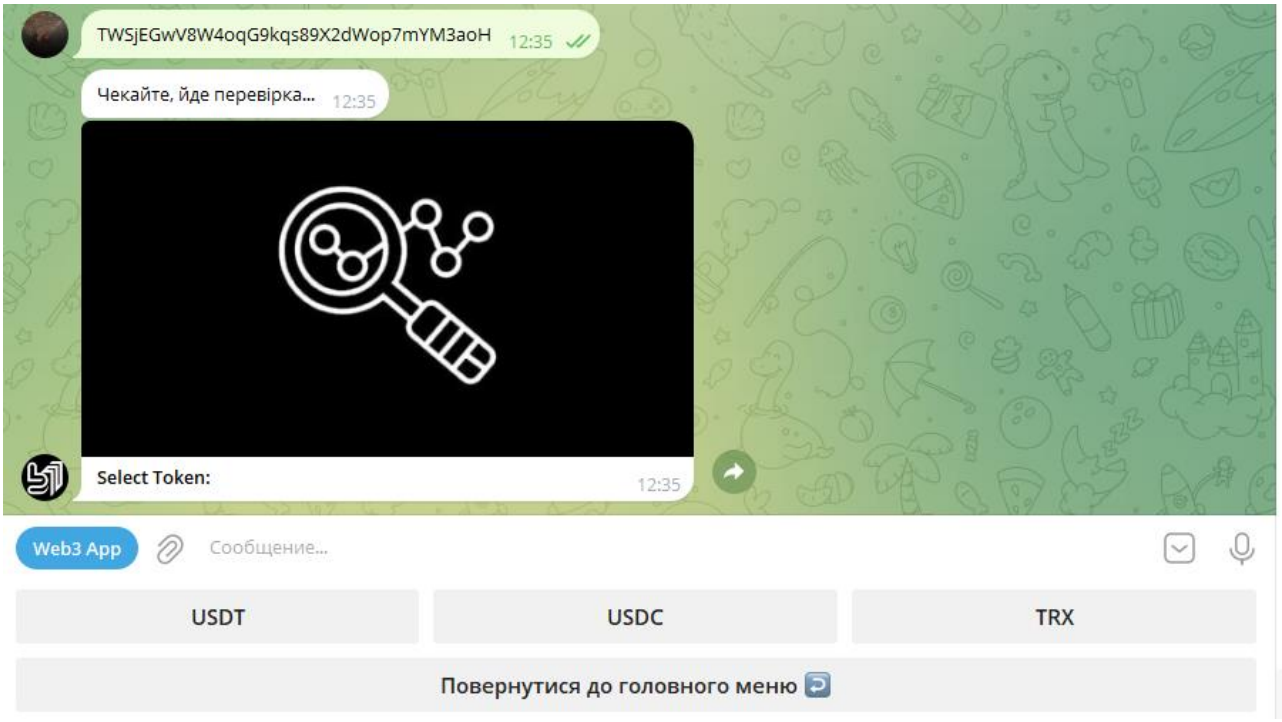


Рис. 3.2.2.2

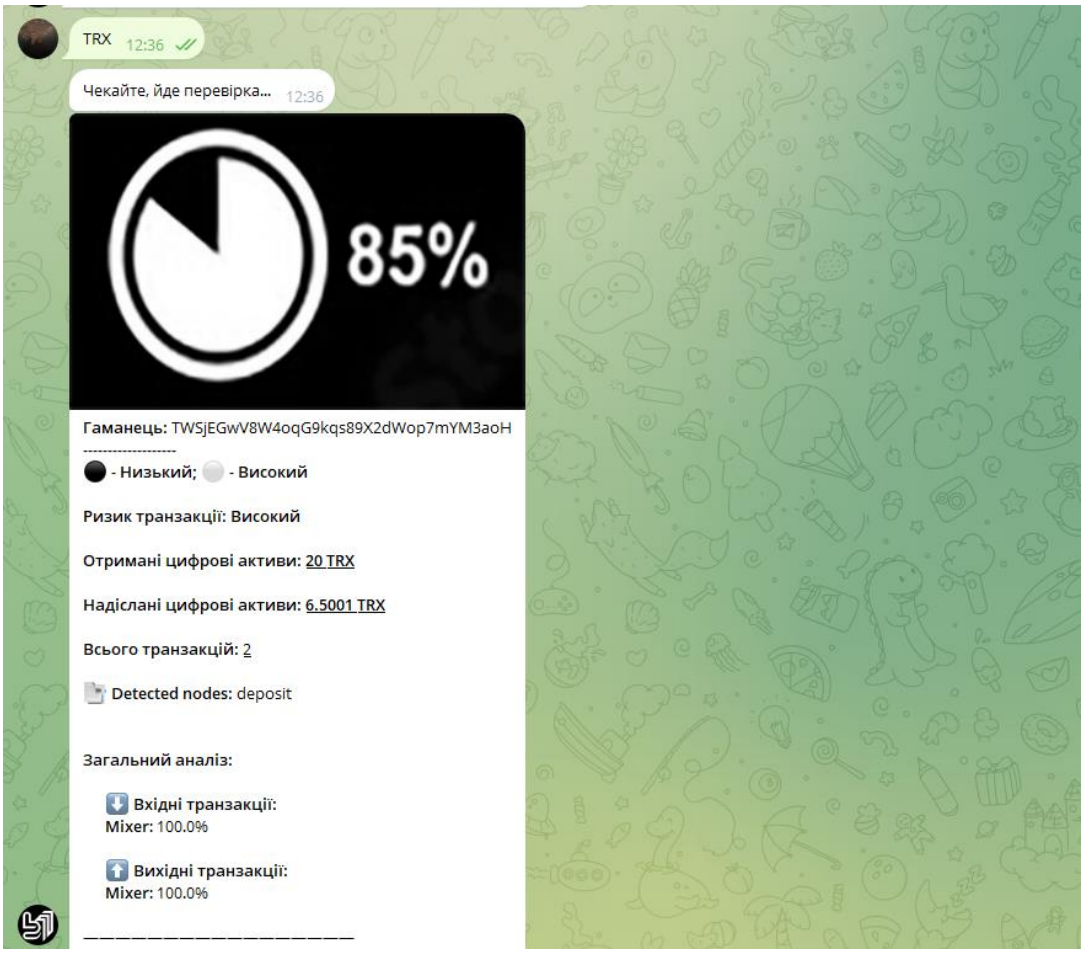


Рис. 3.2.2.3

Далі спробуємо ввести невалідну адресу.
Як видно на Рис. 3.2.2.4, невалідна адреса не проходить, та видає попередження.

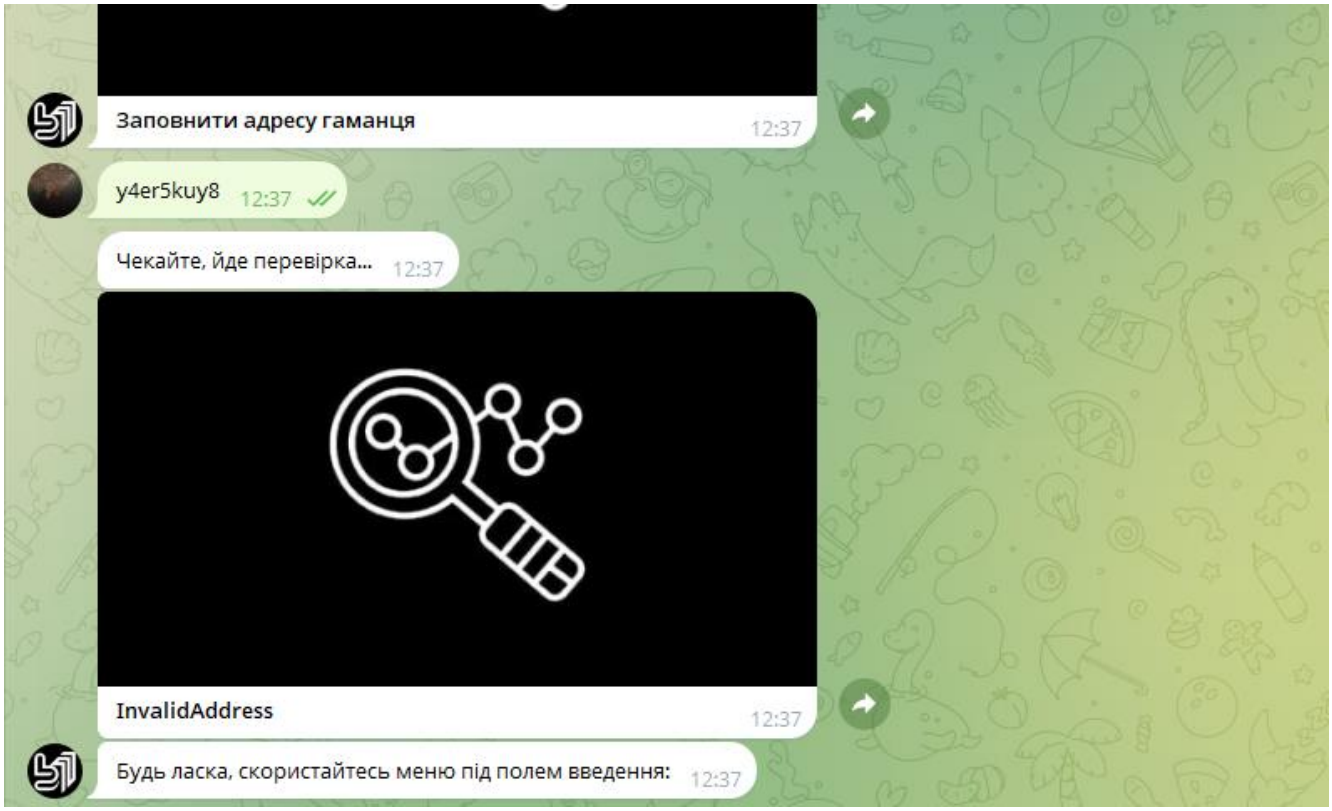


Рис. 3.2.2.4

Таблиця 3.2.2 Результати тестування перевірки гаманців у різних мережах

Мережа	**Базовий сценарій**	**Сценарій з невалідною адресою**	**Результат**
ETH	Успішно	Успішно	Успішно
USDT-ERC20	Успішно	Успішно	Успішно
USDC-ERC20	Успішно	Успішно	Успішно
TRX	Успішно	Успішно	Успішно
USDT-TRC20	Успішно	Успішно	Успішно
USDC-TRC20	Успішно	Успішно	Успішно
TON	Успішно	Успішно	Успішно
BTC	Успішно	Успішно	Успішно
SOL	Успішно	Успішно	Успішно

За результатми тестування у мережах приведених в Таблиці 3.2.2, можемо зробити висновок, що перевірки адрес працюють корректно.

ВИСНОВОК

У результаті виконання роботи було розроблено Telegram-бота для виявлення підозрілих транзакцій у різних Blockchain-мережах. Дослідження та практична реалізація дозволили отримати наступні результати:

Проведено аналіз сучасного стану Blockchain-технологій та проблем, які пов'язані із шахрайством у криптовалютних операціях. Визначено основні типи шахрайських схем та методи протиправної діяльності з використанням криптовалют. Було розглянуто правові аспекти протидії шахрайству у крипто сфері та можливі економічні наслідки таких операцій для користувачів і ринку в цілому.

Розроблено методологію аналізу даних про транзакції для виявлення підозрілої активності. Визначено критерії класифікації транзакцій як потенційно підозрілих, створено алгоритми обробки та аналізу даних, отриманих через API.

Спроектовано та розроблено архітектуру Telegram-бота з використанням сучасних підходів до розробки програмного забезпечення. Обрано бібліотеку aiogram для взаємодії з Telegram API, що забезпечує асинхронність. Розроблено інтуїтивно зрозумілий інтерфейс для зручності користувачів.

Впроваджено функціонал ручної перевірки транзакцій і гаманців, а також моніторинг гаманців, з автоматичною перевіркою нових вхідних транзакцій.

Проведено тестування розробленого Telegram-бота. Виконано функціональне тестування основних можливостей бота. Результати тестування підтвердили ефективність розробленого рішення.

Практична цінність результатів

Розроблений Telegram-бот має значну практичну цінність і може бути використаний:

- Криптовалютними біржами та обмінниками
- Правоохоронними органами для розслідування фінансових злочинів
- Індивідуальними користувачами криптовалют для перевірки надійності контрагентів перед здійсненням транзакцій

Отже, розроблений Telegram-бот є ефективним інструментом для виявлення підозрілих транзакцій у різних блокчейн-мережах, що допомагає значно підвищити безпеку та прозорість криптовалютних операцій. Система демонструє

високу точність та надійність вихідних даних, а також має зручний та інтуїтивно зрозумілий інтерфейс, що робить її доступною для широкого кола користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[2] Gartner (2023). Forecast: Blockchain Business Value, Worldwide, 2017-2030. Gartner Research.

[3] Circle (2025). USDC Reserves Report. https://www.circle.com/en/usdc/reserves

[4] Tether (2025). Transparency Report. https://tether.to/en/transparency/

[5] Chainalysis. (2024). The 2024 Crypto Crime Report. Chainalysis Inc.

[6] Cambridge Centre for Alternative Finance. (2023). Global Cryptoasset Regulatory Landscape Study. University of Cambridge.

Wikipedia - https://uk.wikipedia.org/

Державний університет інформаційно-комунікаційних технологій
Кафедра Інформаційних систем та технологій

Кваліфікаційна робота
на тему:

«Телеграм-бот для виявлення підозрілих транзакцій у мережах на основі блокчейн»

на здобуття освітнього ступеню бакалавра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: Новик Р. С., ІСД-42
Науковий керівник роботи
Гашко А. О.

Київ 2025

🔒 Актуальність теми

Проблеми безпеки:

💰 Зростання шахрайства
$20+ млрд у 2024 році

🕳️ Відмивання коштів
Через криптовалюти

⚠️ Санкційні активи
В блокчейн-мережах

🔍 Складність виявлення
Для звичайних користувачів

Чому Telegram-бот?

- 🚀 Швидкий доступ
- 📱 Зручний інтерфейс
- 🔄 Автоматичний моніторинг
- 🌍 Широка аудиторія

🎯 Мета та завдання дослідження

Мета:

Створення інструменту для виявлення підозрілих транзакцій у різних блокчейн-мережах та запобігання потрапляння санкціонних коштів

Підтримувані мережі:

- Bitcoin (BTC)
- Ethereum (ETH)
- USDT-ERC20
- USDC-ERC20
- Tron (TRX)
- USDT-TRC20
- USDC-TRC20
- TON
- Solana (SOL)

Основні завдання:

1. **Аналіз методів** виявлення підозрілих транзакцій
2. **Дослідження API** для блокчейн-мереж
3. **Розробка архітектури** Telegram-бота
4. **Реалізація функціоналу** перевірки та моніторингу
5. **Система моніторингу** автоматичних перевірок
6. **Тестування** та оптимізація системи

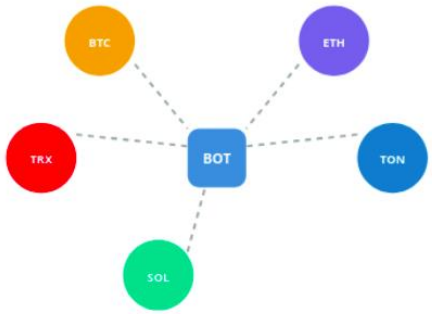
🔬 Об'єкт та предмет дослідження

Об'єкт дослідження:

Процес створення сервісу перевірки безпечності транзакцій у блокчейн-мережах на основі Telegram-боту

Предмет дослідження:

Telegram-бот для перевірки підозрілих транзакцій у мережах Blockchain



Мультимережева інтеграція

⚠️ Аналіз проблеми шахрайства

🎣 Фішинг та соціальна інженерія

- Фальшиві сайти
- Підміна адрес гаманців
- Шахрайські повідомлення

💰 Скам-проєкти

- Шахрайські ICO/IDO
- Rug pulls
- Інвестиційні піраміди

🔧 Технічні маніпуляції

- Pump & dump схеми
- Флеш-кредитні атаки
- MEV атаки

💸 Відмивання коштів

- Використання міксерів
- Багатоетапні переводи
- Крос-чейн операції

$20B+
Обсяг шахрайства у 2024

2-5%
Від загальної капіталізації

0.4%
Від усіх транзакцій

🏗️ Архітектура рішення

Технологічний стек:

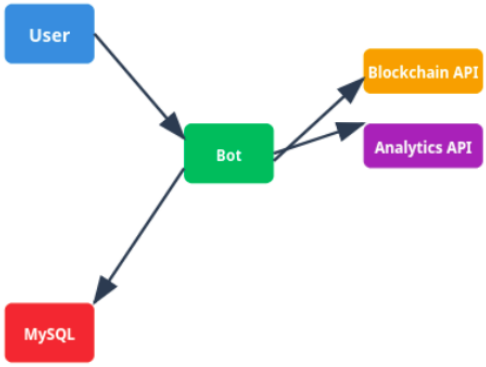
Python 3.8+ | aiogram | aiohttp | MySQL | Ubuntu Server | Supervisor

API інтеграції:

🔗 **Etherscan API**
Ethereum мережа

🔗 **Blockchair API**
Bitcoin та Tron

🔗 **Спеціалізоване API**
Аналітика транзакцій



Асинхронна архітектура

⚙️ Функціональність бота

🔍 Ручна перевірка

- Перевірка транзакцій за хешем
- Аналіз гаманців за адресою
- Автовизначення мережі
- Детальний звіт з ризиками

📊 Автоматичний моніторинг

- Додавання гаманців
- Перевірка нових транзакцій
- Миттєві сповіщення
- Управління списком

📈 Аналітика

- Історія перевірок
- Статистика користування
- Експорт звітів у PDF
- Детальні метрики

🌐 Багатомовність

- Українська мова
- Англійська мова
- Німецька мова
- Іспанська мова

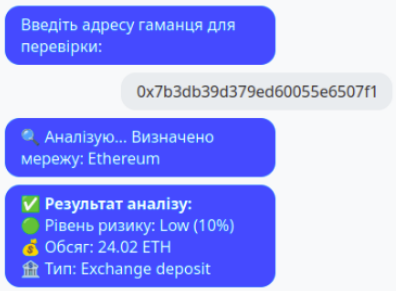
Результат аналізу включає:

Low Risk Medium Risk High Risk

📱 Інтерфейс користувача

Головне меню

Приклад діалогу



Алгоритм перевірки:

🗄️ База даних та архітектура

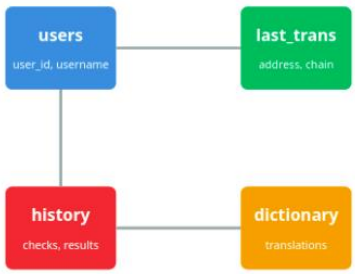
Структура БД:

👥 users
- Інформація користувачів
- Баланс перевірок
- Налаштування мови

📊 last_trans
- Гаманці на моніторингу
- Останні транзакції
- Мережі та адреси

📈 transaction_history
- Історія перевірок
- Статистика
- Результати аналізу

🌐 dictionary
- Переклади
- Багатомовна підтримка
- Локалізація



MySQL Реляційна БД

🧪 Тестування системи

Сценарії тестування:

✅ Перевірка транзакцій
- Валідні хеші - успішно
- Невалідні хеші - коректна обробка
- Всі мережі протестовані

✅ Перевірка гаманців
- Валідні адреси - успішно
- Невалідні адреси - обробка помилок
- Автовизначення мережі

✅ Навантажувальне тестування
- Стабільна робота під навантаженням
- Асинхронна обробка запитів
- Автоматичне відновлення

Результати тестування мереж:

Всі тести пройдені успішно

Висновки та перспективи

Розроблено:

- Повнофункціональний Telegram-бот
- Підтримка 9 різних токенів/мереж
- Система автоматичного моніторингу
- Багатомовний інтерфейс

Наукова новизна:

- Вперше реалізовано підтримку багатьох мереж з автовизначенням
- Унікальна система оцінки ризику транзакцій
- Інтуїтивний мінімалістичний інтерфейс
- Автоматична перевірка вхідних транзакцій

Перспективи розвитку:

- Додавання нових блокчейн-мереж
- Інтеграція з існуючими фінансовими системами
- Розширення функціоналу моніторингу