Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Об'єктно-орієнтоване програмування»

Тема «Розробка та розгортання некастодіального криптогаманця для мережі TON, інтегрованого з Telegram»

Студента__2__ курсу AI-223 групи
Спеціальності 122 – «Комп’ютерні науки»
_____Муляр Даніїл Дмитрович_____
(прізвище та ініціали)
Керівник ст. викл. к.т.н. Годовіченко М.__
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії
_____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)

м. Одеса – 2024 рік

Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

ЗАВДАННЯ

НА КУРСОВУ РОБОТУ

Студенту Муляр Даніїлу Дмитровичу група AI-223

1. Тема роботи

«Розробка та розгортання некастодіального криптогаманця для мережі TON, інтегрованого з Telegram»

2. Термін здачі студентом закінченої роботи 20.06.2024

3. Зміст розрахунково-пояснювальної записки: огляд технологій, взаємодія з блокчейном TON, архітектура та компоненти системи, розгортання додатку, інструкція користучава

Завдання видано 29.05.24 _____
(підпис викладача)

Завдання прийнято до виконання 29.05.24 _____
(підпис студента)

АНОТАЦІЯ

В курсовій роботі розроблено та розгорнуто веб-додаток TonPlex — некастодіальний криптогаманець для мережі TON, інтегрований з Telegram. Проект використовує FastAPI, SQLAlchemy, aiogram та Jinja2 для реалізації гнучкої та асинхронної архітектури. Особливу увагу приділено взаємодії з блокчейном TON та забезпеченню безпеки даних. У роботі описані основні етапи розробки, розгортання та налаштування додатку, а також розглянуті перспективи подальшого розвитку проекту.

ABSTRACT

In this coursework, a web application called TonPlex was developed and deployed. TonPlex is a non-custodial cryptocurrency wallet for the TON network, integrated with Telegram. The project uses FastAPI, SQLAlchemy, aiogram, and Jinja2 to implement a flexible and asynchronous architecture. Special attention is given to interaction with the TON blockchain and ensuring data security. The work describes the main stages of development, deployment, and configuration of the application, as well as the prospects for further development of the project.

ЗМІСТ

ВСТУП

Проєкт являє собою некостодіальний криптогаманець мережі TON, реалізований у вигляді веб-додатку всередині Telegram. Це веб-оболонка, яка дозволяє користувачам взаємодіяти з блокчейном через їхній гаманець, використовуючи мій інтерфейс.

Основною причиною для розробки цього криптогаманця та занурення у цю тему став щирий інтерес автора до світу Web3. Не розуміючи, як взаємодіяти з ним через код, автор поставив собі за мету зробити більше, ніж міг, і досягти того, чого раніше не вмів, у короткі строки та з повною віддачею процесу.

Вибір на користь некостодіального підходу обумовлений тим, що він здається найбільш канонічним для блокчейн-технологій, втілюючи їх основні принципи — прозорість та свободу. Користувач, бачачи на балансі свого некостодіального гаманця певну суму, може бути впевнений, що вона дійсно належить йому. Перенесення гаманця в інший інтерфейс здійснюється просто шляхом введення сід-фрази, без необхідності переводити кожен актив окремо.

Проєкт спочатку задумувався як навчальний і не має конкретної цільової аудиторії. Проте з урахуванням поточного функціоналу — імпорт та створення гаманця, прийом і відправка коштів — цей гаманець може підійти широкому колу користувачів. Інтерфейс розроблено дружелюбним, інтуїтивно зрозумілим і стильним, що робить його придатним для всіх, хто зацікавлений у використанні криптовалют.

1 ОГЛЯД ТЕХНОЛОГІЙ

Метою цієї роботи було вийти за межі своїх поточних навичок і досягти більшого, ніж було можливо раніше, в короткі строки. Спочатку маючи непогане розуміння роботи бекенду, мені також потрібно було освоїти фронтенд розробку.

Python

Основною мовою програмування для цього проєкту був обраний Python, оскільки він є моєю улюбленою мовою. Python надає широкий спектр бібліотек і фреймворків, необхідних для реалізації проєкту.

FastAPI

Для запуску додатку використовувався фреймворк FastAPI, який відрізняється неймовірною простотою, гнучкістю та підтримкою асинхронного програмування. FastAPI дозволяє швидко створювати потужні та продуктивні веб-додатки.

SQLAlchemy

Для взаємодії з базою даних PostgreSQL використовувалася бібліотека SQLAlchemy на драйвері asyncpg. SQLAlchemy надає всі необхідні інструменти ORM, спрощуючи роботу з базою даних та забезпечуючи високу продуктивність.

Aiogram

Розробка веб-додатку всередині Telegram включала взаємодію з ботом, для чого був обраний aiogram. Цей фреймворк є найпотужнішим для роботи з Telegram Bot API, і я добре з ним знайомий. Aiogram забезпечує зручне та гнучке створення і управління ботами.

Jinja2

Спочатку планувалося писати фронтенд окремо на React, однак, ознайомившись із цим інструментом, я зрозумів, що його освоєння займе більше

часу, ніж я очікував. Тому я вирішив використовувати Jinja2 для генерації статичних сторінок. Jinja2 дозволяє легко інтегрувати шаблони в Python-додатки та генерувати динамічні веб-сторінки.

HTML, CSS, JavaScript

Фронтенд був реалізований за допомогою HTML, CSS та JavaScript. Ці технології забезпечують створення користувацького інтерфейсу.

TON SDK та PyTONAPI

Взаємодія з блокчейном TON була реалізована за допомогою бібліотек tonsdk та PyTONAPI. Ці інструменти на Python дозволили інтегрувати функціонал блокчейна в проект. Попереднє вивчення лекцій про блокчейн TON дало мені розуміння його внутрішнього устрою, що виявилося надзвичайно корисним, враховуючи відсутність документації по цим бібліотекам. Робота з ними нагадувала подорож по тривимірному простору з глибокими впадинами, високими горами і кротовими норами, де високорівнева абстракція представляла собою гору, а низькорівнева — впадину, все це змішувалося в такий незрозумілий і кривий ландшафт, що після того як я розібрався, то більше почав цінувати раніше знайомі інструменти європейських та американських розробників. Це було непросто, але результат того вартий.

2 ВЗАЄМОДІЯ З БЛОКЧЕЙНОМ TON

2.1 Загальне описання блокчейну

Блокчейн представляє собою розподілену базу даних, яка зберігає інформацію про всі транзакції, що відбуваються в мережі. Основна перевага блокчейну полягає в його децентралізації, що означає відсутність єдиної точки контролю. Це забезпечує високий рівень безпеки і прозорості, оскільки дані, записані в блокчейн, практично неможливо змінити або видалити.

2.2 Ключові особливості блокчейну TOH

Блокчейн TOH (The Open Network) унікальний у своєму роді і відрізняється від інших блокчейнів своєю багаторівневою архітектурою. TOH складається з чотирьох основних рівнів:

1. **Мастерчейн (Masterchain)**: Головний блокчейн мережі TOH, що містить метаінформацію про всі інші ланцюжки (воркчейни). Мастерчейн координує роботу всієї мережі.
2. **Воркчейни (Workchains)**: Підпорядковані ланцюжки мастерчейну, кожен з яких може мати свої власні правила і структури даних.
3. **Шардчейни (Shardchains)**: Воркчейни можуть бути розбиті на шардчейни для паралельної обробки транзакцій, що значно збільшує пропускну здатність мережі.
4. **Аккаунтчейни (Accountchains)**: Всередині кожного шардчейну знаходяться аккаунтчейни, які містять дані і код смарт-контрактів конкретних акаунтів.

Блокчейн TOH є першим асинхронним блокчейном, який спроектований так, що може безкінечно масштабуватися, забезпечуючи високу продуктивність і швидкість обробки транзакцій.

2.3 Інструменти для взаємодії з TOH

Для взаємодії з блокчейном TOH я використовую два основних інструменти:

1. **TON SDK (tonsdk)**: Використовується для генерації та відправлення зовнішніх повідомлень у блокчейн, а також для створення та серіалізації гаманців.

2. **TON API (PyTONAPI)**: Це бібліотека, яка є обгорткою для мови Python і дозволяє парсити дані з блокчейну та всієї інфраструктури TOH.

Для роботи з цими інструментами я розробив два ключові класи: TONWalletManager і AsyncTONApiClient.

2.4 TONWalletManager

Клас TONWalletManager відповідає за створення та управління гаманцями, а також за виконання транзакцій.

Основні методи класу:

1. **create_wallet**: Створює новий гаманець, генерує мнемонічну фразу і повертає об'єкт TonWalletModel з деталями гаманця.

2. **get_wallet**: Відновлює гаманець на основі наданої мнемонічної фрази.

3. **deploy_wallet**: Ініціалізує гаманець у блокчейні, відправляючи початкове повідомлення для активації гаманця.

4. **transfer**: Виконує відправку транзакції з вказаного гаманця на іншу адресу.

2.5 Детальний опис методу transfer

Метод **transfer** відповідає за відправку транзакцій і включає наступні кроки:

1. **Отримання seqno**: Першим кроком метод викликає **get_seqno**, щоб отримати поточний номер послідовності транзакцій для гаманця. Seqno (sequence number) — це порядковий номер транзакції, який збільшується на одиницю після кожної відправки. Це запобігає повторному виконанню однієї і тієї ж транзакції та забезпечує їх унікальність.

2. **Ініціалізація гаманця**: Якщо функція не вдається викликати метод **get_seqno**, це означає, що гаманець не ініціалізований. У цьому випадку метод викликає **deploy_wallet**, щоб ініціалізувати гаманець у блокчейні, відправляючи початкове повідомлення для активації гаманця. Після цього метод знову викликає **transfer**.

3. **Формування і відправка транзакції**: Після успішного отримання seqno метод формує повідомлення про трансфер за допомогою **create_transfer_message**, вказуючи адресу призначення, суму і номер послідовності. Повідомлення про трансфер відправляється в блокчейн за допомогою **raw_send_message**. Якщо повідомлення успішно відправлено, транзакція логірується.

2.6 Додаткові пояснення

Створення гаманця в блокчейні TOH — це процес створення порожньої адреси в пам'яті. Щоб адреса стала гаманцем, необхідно його проініціалізувати, відправивши в блокчейн ініціалізаційне повідомлення. До отримання першого TOHкоіна і його відправки адреса буде неініціалізованою.

Кожна адреса в мережі TOH може мати кілька форматів. Основні типи адрес: bounceable і nonbounceable. Адреса у форматі bounceable підтримує

функцію відскоку, що означає повернення коштів при помилці. Адреси bounceable і nonbounceable схожі, але мають різний бітовий мешок в кінці. Для отримання перших монет необхідно представити адресу у форматі nonbounceable, щоб кошти надійшли на рахунок.

2.7 AsyncTONApiClient

Клас **AsyncTONApiClient** призначений для парсингу даних з блокчейну і всієї інфраструктури TOH. Він збирає дані про адреси, поточні котирування і іншу необхідну інформацію, яка може знадобитися користувачу в інтерфейсі. Основні методи класу включають: отримання балансу адреси. отримання історії транзакцій. отримання поточних котирувань токенів.

3 АРХІТЕКТУРА ТА КОМПОНЕНТИ СИСТЕМИ

3.1. Загальна структура проекту

Проєкт організований у вигляді набору директорій, кожна з яких представляє окремий компонент системи. Загальна структура проєкту представлена на наступному фото (рис. 3.1).

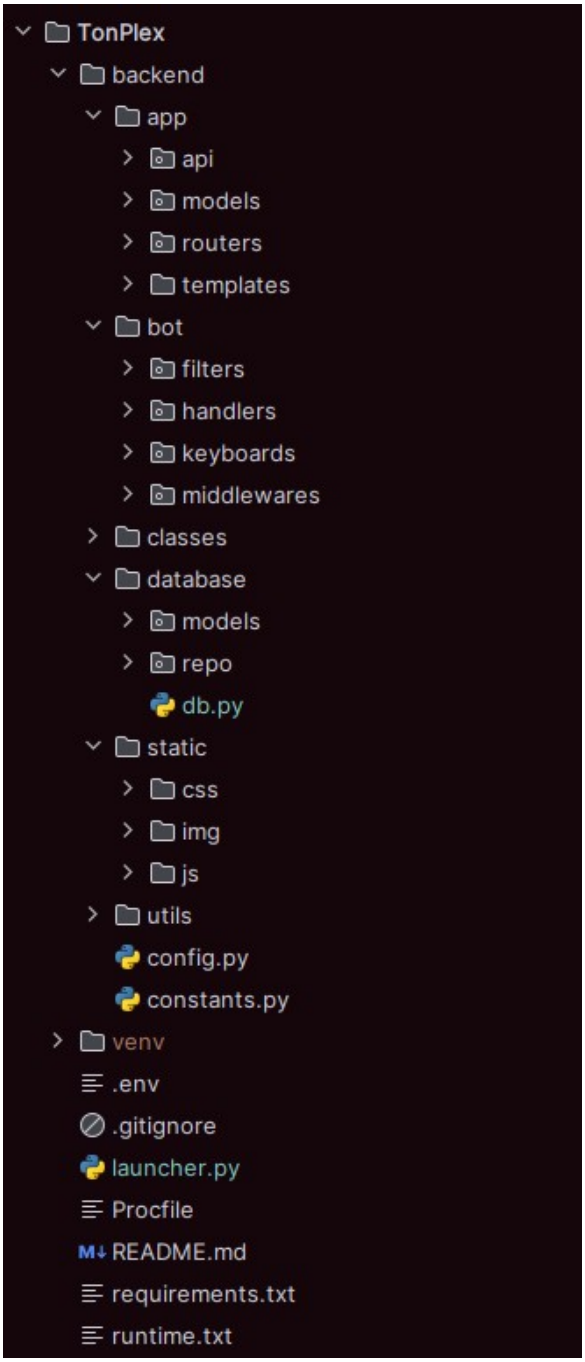

Рис. 3.1 — Структура проекту

3.2. Бекенд

У корені проєкту знаходяться наступні ключові файли:

- **.env**: Файл для зберігання змінних оточення, необхідних для конфігурації застосунку.

- **.gitignore**: Список файлів та директорій, які не повинні бути включені до репозиторію git.

- **launcher.py**: Містить ключові функції для зв'язування та розгортання всього проєкту, утворюючи єдине, більш потужне ціле.

- **Procfile**: Конфігураційний файл для розгортання застосунку на платформі Railway.

- **README.md**: Документація проєкту, що включає опис, інструкції щодо встановлення та використання.

- **requirements.txt**: Список залежностей проєкту, які повинні бути встановлені.

- **runtime.txt**: Вказує версію Python, що використовується для запуску застосунку.

Весь код знаходиться в директорії backend, яка містить наступні піддиректорії та файли:

Директорія App:

- **api**: Реалізовані роутери POST-запитів.
    - wallet.py
    - wallet_setup.py
    - __init__.py

- **models**: Модулі pydantic для валідації отримуваних і повертаємих даних.
    - mnemonics.py
    - transactions.py

    - user.py
    - wallet.py
    - __init__.py

- **routers**: Реалізовані роутери GET-запитів, які генерують веб-шаблони для користувачів.
    - wallet.py
    - wallet_setup.py
    - welcome.py
    - __init__.py

- **templates**: Містить шаблони HTML-сторінок.
    - base.html
    - import_wallet.html
    - redirect.html
    - wallet.html
    - wallet_created.html
    - welcome.html

Директорія Bot

Директорія bot містить модулі для Telegram-бота:

- **filters**: Фільтри для обробки вхідних повідомлень.
    - private_chat.py
    - __init__.py

- **handlers**: Обробники різних команд та подій.
    - starting.py
    - __init__.py

- **keyboards**: Клавіатури для взаємодії з користувачем.
    - start.py
    - __init__.py

- **middlewares**: Проміжні шари для обробки запитів.
    - config.py

- __init__.py

Директорія Classes

Директорія classes містить окремі класи, які можуть використовуватись у всьому застосунку:

- **atomic_manager.py**: Реалізує клас для забезпечення атомарності деяких операцій.

- **encryption_manager.py**: Реалізує клас, який шифрує і дешифрує дані, що будуть записані до бази даних.

- **ton_api_client.py**: Реалізує клас для парсингу необхідних даних з блокчейну.

- **ton_wallet_manager.py**: Реалізує клас для взаємодії з блокчейном, включаючи створення, серіалізацію та деплой гаманців, а також відправку транзакцій.

Директорія Database

Директорія database реалізує підключення до бази даних за допомогою SQLAlchemy. Вона містить дві піддиректорії:

- **models**: Представлена ORM структура таблиці користувачів.
  - user.py
  - __init__.py

- **repo**: Реалізовано клас для взаємодії з таблицею користувачів.
  - user.py
  - __init__.py

Директорія **Static**

Директорія static містить всі необхідні допоміжні файли та ресурси для шаблонів:

- **CSS**: Стилі.
  - base_styles.css

    - import_styles.css
    - wallet.css
    - wallet_created.css
    - welcome_styles.css

- **Images**: Зображення.

- **JavaScript**: Скрипти.
    - import_wallet.js
    - redirect.js
    - wallet.js
    - wallet_created.js
    - welcome.js

Директорія Utils

Директорія utils містить допоміжні модулі з простими функціями, які не вписуються в інші модулі, але можуть ними використовуватись:

- **logo.py**
- **shorten_address.py**

Також, у директорії backend містяться два важливих модулі:

- **config.py**: Представлені датакласи конфігураційних сутностей, такі як різні ключі, необхідні для кожного сервісу. Далі кожен з цих датакласів включається в один єдиний конфігураційний клас, який ініціалізується та використовується всією програмою.

- **constants.py**: Використовується для зберігання констант, які можуть знадобитись системі.

3.3. Фронтенд

Хоча проєкт не розділений на окремий фронтенд, він включає шаблони HTML і статичні файли, такі як CSS і JavaScript. Ці файли використовуються для створення користувацького інтерфейсу і забезпечення інтерактивності веб-застосунку. У майбутньому планується реалізація повноцінного фронтенду на React.

3.4. Інтеграція з Telegram ботом

Директорія bot містить модулі для управління Telegram-ботом. У поточній реалізації участь телеграм бота не обов'язкова, все, що робить цей компонент, це лише реагує на команду старту бота та надсилає повідомлення з кнопкою запуску веб-застосунку. Однак ця директорія містить всі необхідні папки та модулі для подальшого масштабування бота в майбутньому.

3.5. API та маршрутизація

Застосунок має такі ендпоінти (Рис 3.1):

GET-запити (обробляються в директорії app/routers і повертають згенерований шаблон):

1. /wallet: Інтерфейс гаманця
2. /wallet_setup/import: Імпорт гаманця
3. /wallet_setup/created: Гаманець створено
4. /: Кореневий маршрут
5. /redirect: Перенаправлення користувача

POST-запити (обробляються в директорії app/api):

1. /api/wallet/send_transaction: Відправка транзакції з гаманця
2. /api/wallet_setup/create: Створення гаманця
3. /api/wallet_setup/check_mnemonics: Перевірка мнемонічної фрази

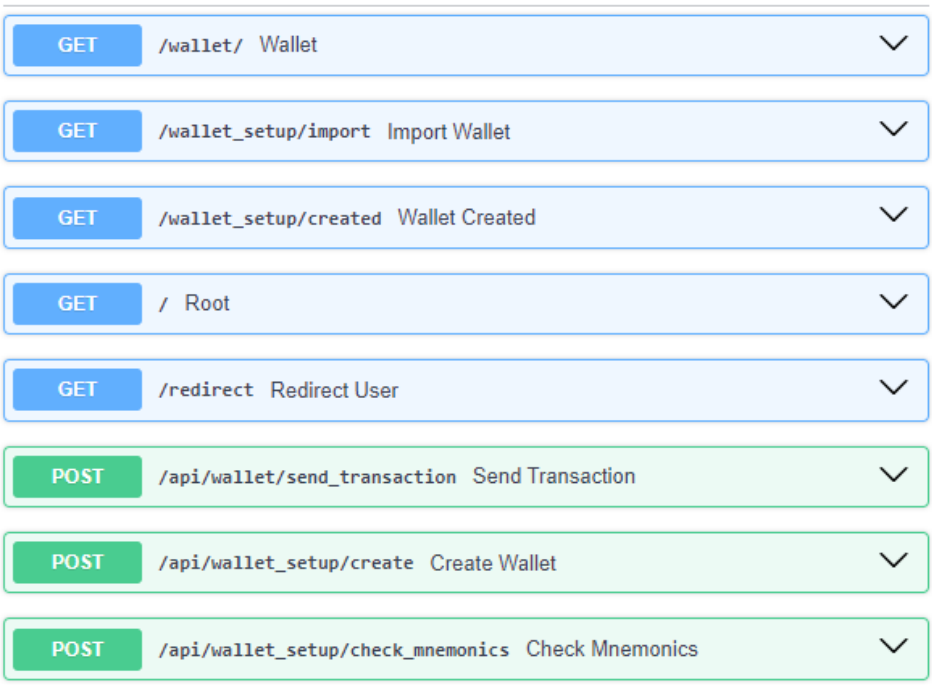

Рис. 3.2 — Структура проекту

Ці ендпоінти надають інтерфейси для взаємодії користувачів із застосунком через GET і POST-запити. GET-запити використовуються для отримання та відображення даних, а POST-запити — для відправки даних на сервер і виконання відповідних операцій.

3.6. Безпека

Безпека в програмі забезпечується двома ключовими класами: atomic_manager та encryption_manager.

1. **Atomic Manager**: слідкує за атомарністю операцій, що запобігає виникненню неконсистентних станів у випадку збоїв або помилок. Цей клас я колись написав сам і повсюдно його використовую. Він дозволяє створювати блокування асинхронних операцій на будь-якому рівні. У поточній реалізації проєкту я не використав цей клас, оскільки не знайшов йому застосування, але все ж залишив для подальшої розробки.

2. **Encryption Manager**: Клас encryption_manager.py відповідає за шифрування і дешифрування даних. Зокрема, він використовується для шифрування мнемонічної фрази кожного користувача перед збереженням її в базу даних. Це забезпечує захист конфіденційної інформації і запобігає несанкціонованому доступу до гаманців користувачів.

Хоча додаткові заходи безпеки наразі не впроваджені, я усвідомлюю вразливості системи, особливо пов'язані з фронтенд-частиною, з якою я ще мало знайомий. На початку розробки я вважав, що підмінити аргументи, які передаються в GET-запити (які генерують веб-сторінки) у Telegram веб-застосунку, неможливо, оскільки Telegram не розкриває подібних даних. Однак існують інструменти, які дозволяють втручатися в роботу застосунку, як якщо б це був звичайний сайт.

При поточній конфігурації проєкту зловмиснику достатньо змінити ID користувача у запиті /wallet&id=id, щоб отримати доступ до сторінки гаманця іншого користувача і взаємодіяти з нею, що може призвести до крадіжки коштів. Щоб уникнути цього, я планую більш глибоко вивчити методи безпечної передачі даних у GET і POST-запитах, а також спробувати захистити сторінку від

розкриття конфіденційних даних.

Зараз ID користувача передається через офіційний скрипт Telegram:

```
<script src="https://telegram.org/js/telegram-web-app.js"></script>
```

Цей скрипт дозволяє отримати ID користувача, який відкрив застосунок. Для підвищення безпеки я планую двічі перевіряти ID користувача і маскувати його в тілі запиту при отриманні даних гаманця, а не передавати всі дані одразу при генерації сторінки через GET-запит.

Щойно я зрозумів, що вже зараз можу просто шифрувати і мнемонічну фразу та айді користувача, у такому разі підміна айді користувача не буде проблемою, оскільки система все одно його не розпізнає.

4 РОЗГОРТАННЯ ДОДАТКА

4.1. Вибір платформи

Для розгортання застосунку було обрано сервіс Railway. Він надає простий та функціональний інтерфейс, що дозволяє легко керувати проєктами. Платформа дозволяє підключити GitHub акаунт і автоматично запускати нові збірки при кожному коміті, що робить процес розгортання та оновлення застосунку більш зручним та ефективним.

4.2. Підготовка проекту

Перед розгортанням проєкту на Railway були виконані наступні кроки по підготовці:

1. **Налаштування змінних оточення**: Файл .env містить наступні змінні оточення, необхідні для коректної роботи застосунку:

BOT_NAME
BOT_TOKEN
ADMINS
USE_REDIS
POSTGRES_USER
POSTGRES_PASSWORD
POSTGRES_DB
DB_HOST
ENCRYPTION_KEY
TONAPI_KEY

2. **Підготовка файлів конфігурації**:

- **Procfile**: вказує команду для запуску застосунку. Railway буде шукати цей файл у корені проєкту.

  web: python launcher.py

- **runtime.txt**: вказує версію Python, що використовується для запуску застосунку:

  python-3.9.6

4.3. Процес розгортання

Процес розгортання проєкту на Railway включав наступні етапи:

1. **Створення акаунта**: Реєстрація та створення акаунта на платформі Railway.
2. **Підключення GitHub акаунта**: Підключення GitHub акаунта для доступу до репозиторіїв проєкту.
3. **Створення білда**: Створення нового білда обраного репозиторію. Railway автоматично запускає процес збірки при кожному коміті.
4. **Ініціалізація бази даних**: База даних розгорнута на платформі neon.tech і працює незалежно від Railway. Ініціалізація бази даних відбувається безпосередньо в проєкті з використанням змінних оточення, вказаних у файлі .env.

4.4. Налаштування та запуск застосунку

Після успішного завершення збірки та розгортання проєкту на Railway були виконані наступні кроки:

1. **Повідомлення про готовність**: Railway повідомив про завершення збірки і надав посилання на веб-застосунок.

2. **Налаштування Telegram бота**: Посилання на веб-застосунок було інтегровано в Telegram бота, що дозволяє користувачам взаємодіяти з застосунком через інтерфейс Telegram.

Таким чином, розгортання застосунку на Railway пройшло успішно завдяки зручності та функціональності платформи, а також можливості автоматичного запуску збірок і простоті інтеграції з GitHub.

5 ІНСТРУКЦІЯ КОРИСТУВАЧА

5.1. Взаємодія користувача з ботом

Користувач починає взаємодію з додатком через Telegram-бота @TonPlexBot (Рис. 5.1). Для початку роботи з ботом користувач викликає команду /start. Бот обробляє команду і надсилає користувачу повідомлення з кнопкою, що містить посилання на веб-додаток.

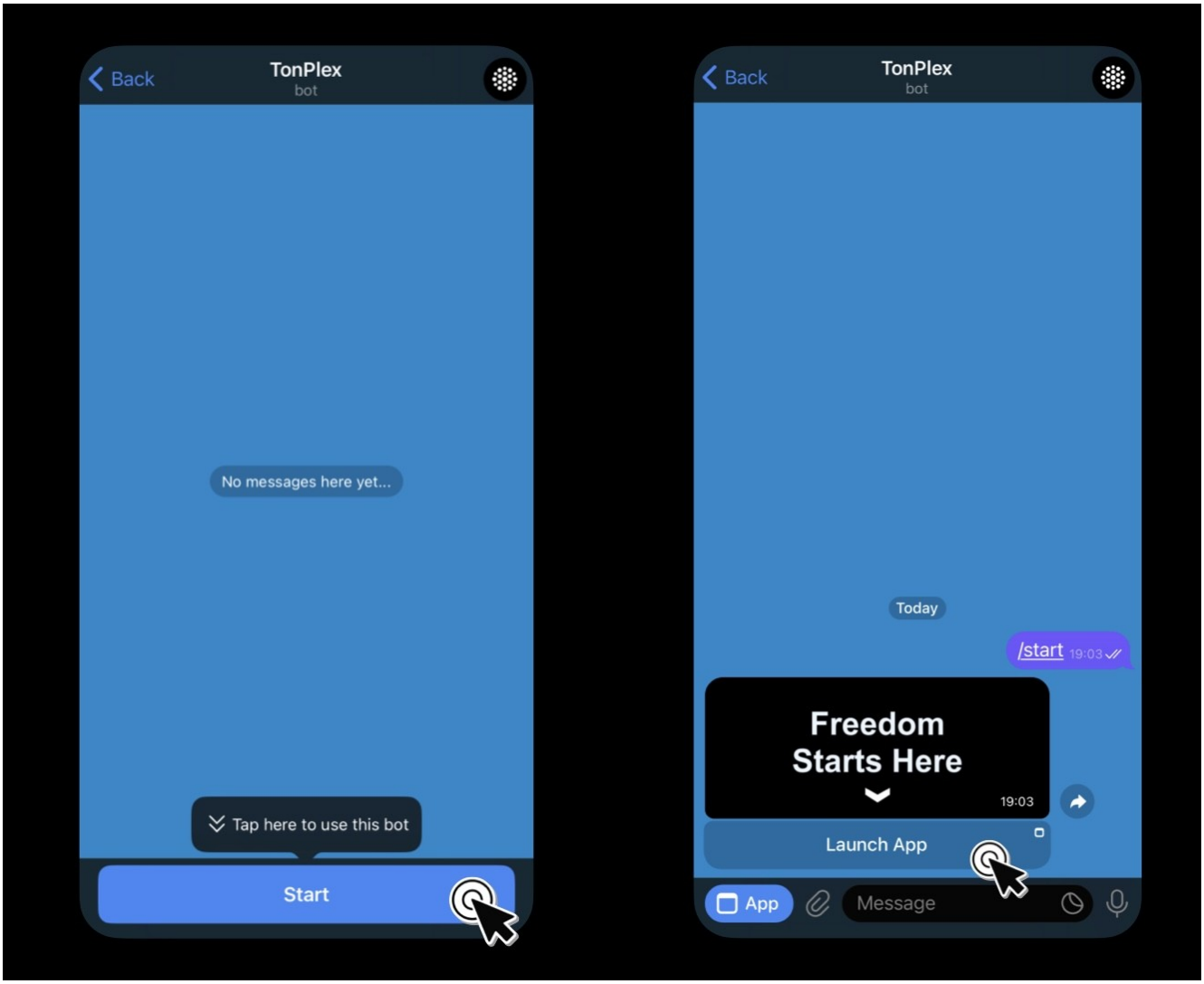

Рис. 5.1 — Старт бота

5.2. Запуск веб-застосунку

Після натискання на кнопку додаток відкривається. В цей момент користувач переходить на кореневий ендпоінт додатку, де система перевіряє наявність гаманця у користувача. Якщо гаманця немає, користувача перенаправляють на сторінку з пропозицією створити новий гаманець або імпортувати існуючий. (Рис. 5.2)

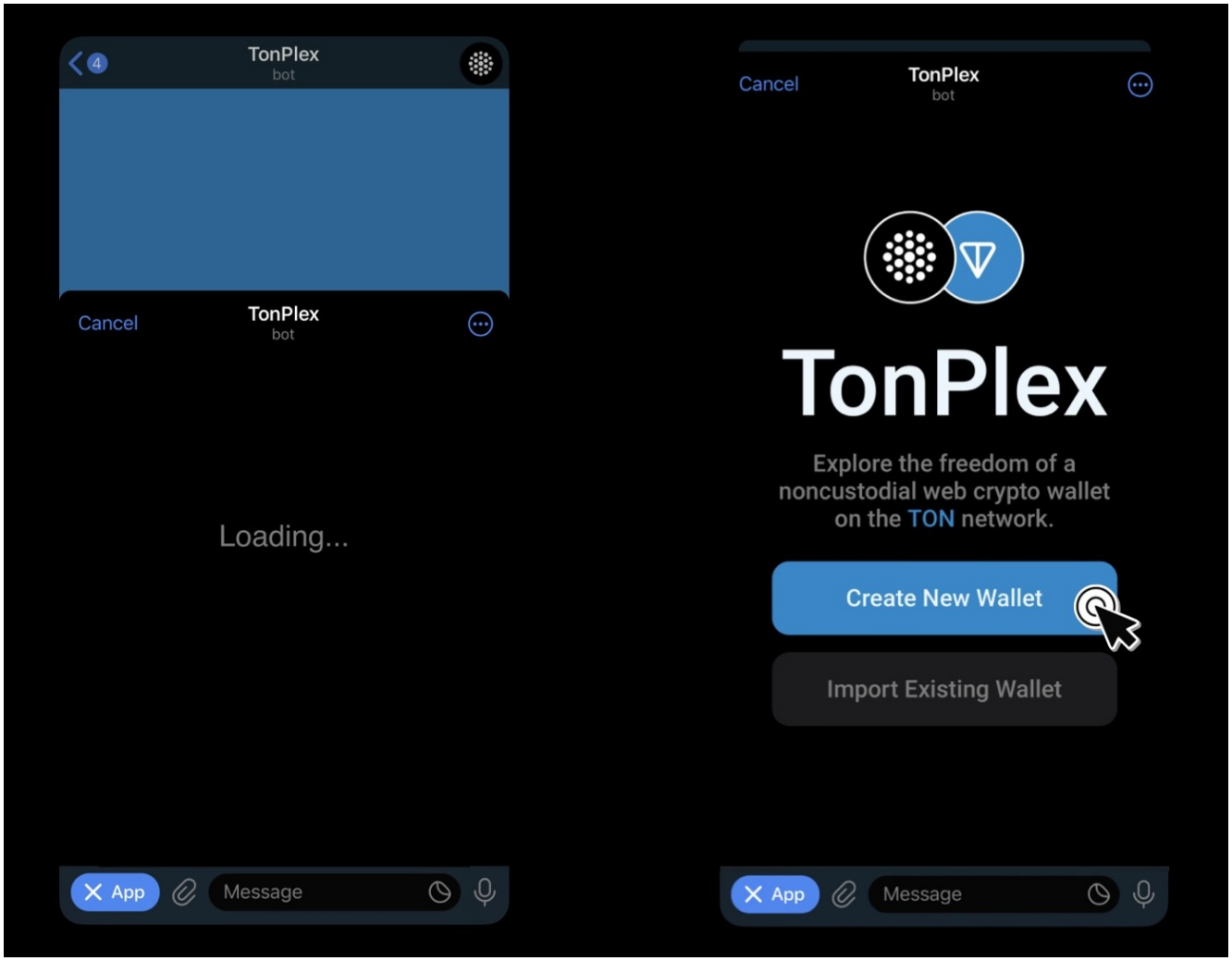

Рис. 5.2 — Запуск веб-додатку

5.3. Налаштування та використання гаманця

Якщо користувач обирає створення нового гаманця, він натискає відповідну кнопку інтерфейсу. Додаток створює гаманець і перенаправляє користувача на сторінку з сид-фразою. (Рис. 5.3) Рекомендується записати фразу, але цей крок можна пропустити. Надалі користувач зможе дізнатися свою фразу від гаманця за потреби. Додаток також дозволяє скопіювати фразу в буфер обміну. На цьому етапі користувач вже зареєстрований в базі даних разом зі своїм гаманцем, і при повторному запуску додатку буде відображатися веб-інтерфейс його гаманця.

Якщо користувач обирає імпорт існуючого гаманця, додаток запропонує ввести мнемонічну фразу. Після успішного введення система записує дані користувача в базу даних, і далі все буде аналогічно сценарію створення нового гаманця.

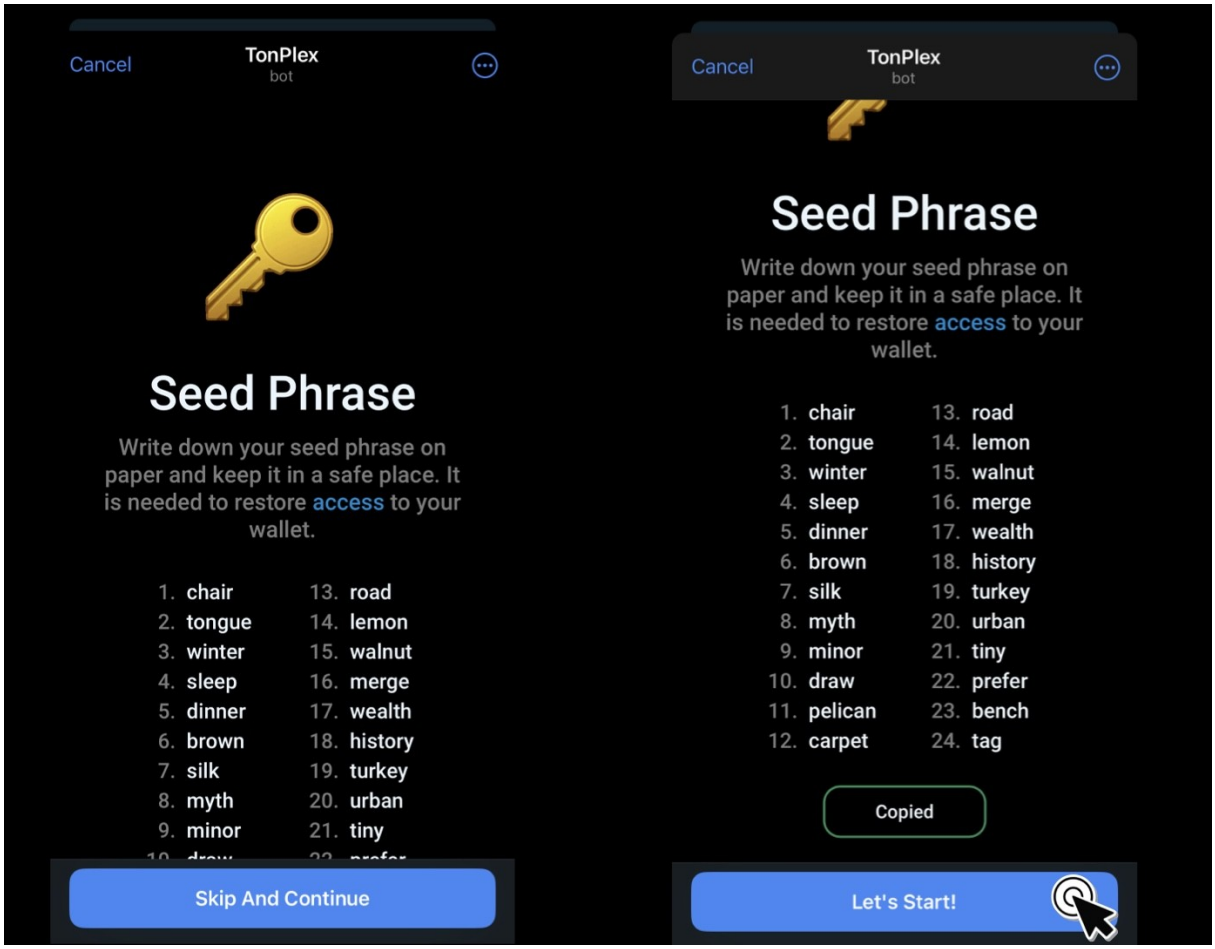

Рис. 5.3 — Створення гаманця

Тепер користувачу доступний веб-інтерфейс його гаманця, за допомогою якого він може приймати Toncoin і жетони, відправляти Toncoin, переглядати історію транзакцій і відкривати сид фразу свого гаманця.

Враховуючи специфіку блокчейну TON, новостворений гаманець — це просто порожня адреса в мережі без задеплоєного смарт-контракту інтерфейсу гаманця, тому до першої транзакції додаток буде видавати користувачу адресу його гаманця у форматі non-bounceable (відмінність можна помітити за трьома останніми символами на адресі), інакше відправлені монети будуть повернуті відправнику та не зможуть надійти на рахунок гаманця. (Рис. 5.4)

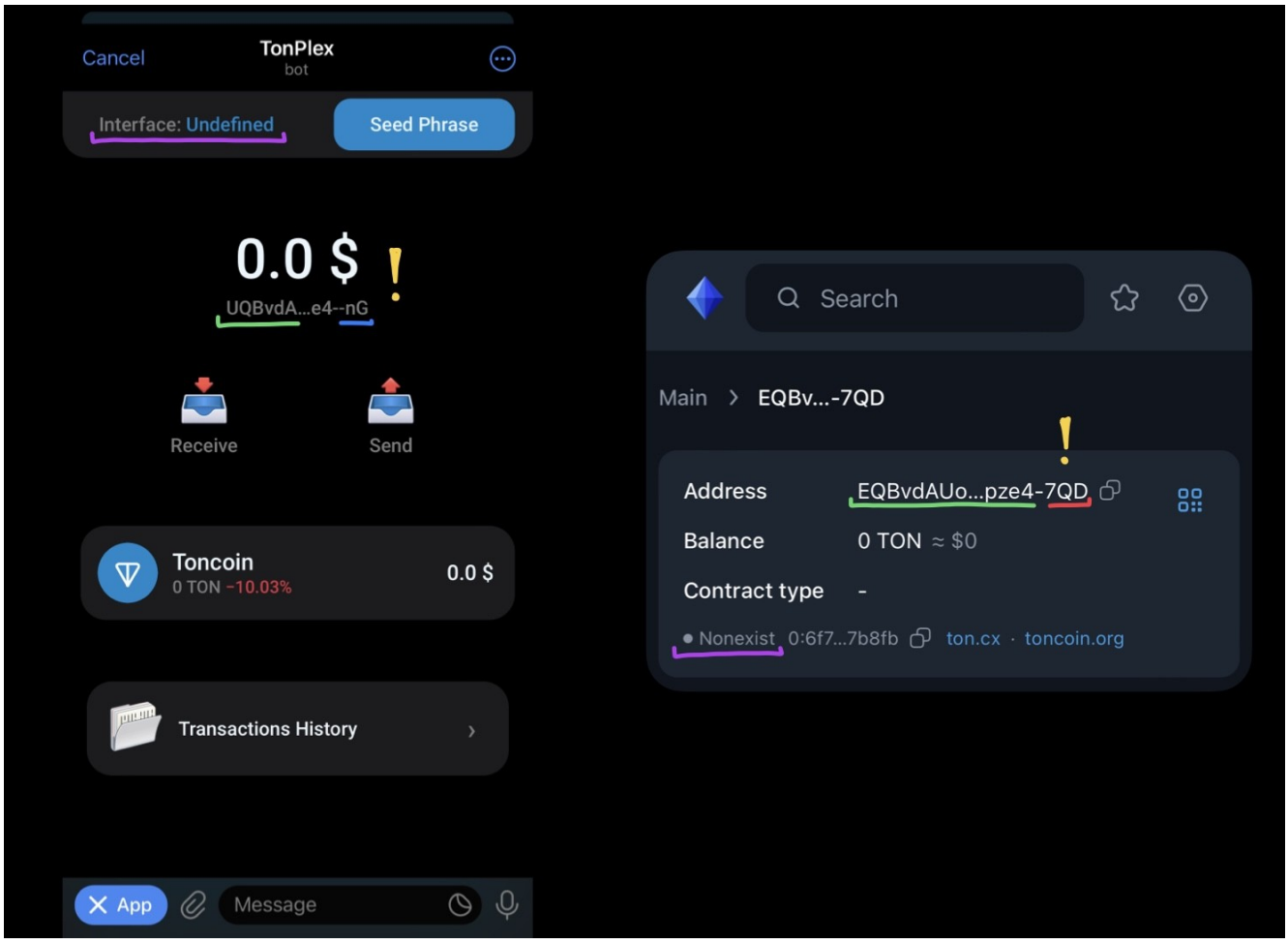

Рис. 5.4 — Веб-інтерфейс гаманця
та його представлення у Tonviewer

5.4. Прийом Toncoin та жетонів

Для прийому перших монет користувач натискає кнопку "Receive". Спливає вікно з адресою гаманця, яку користувач може скопіювати. (Рис. 5.5)

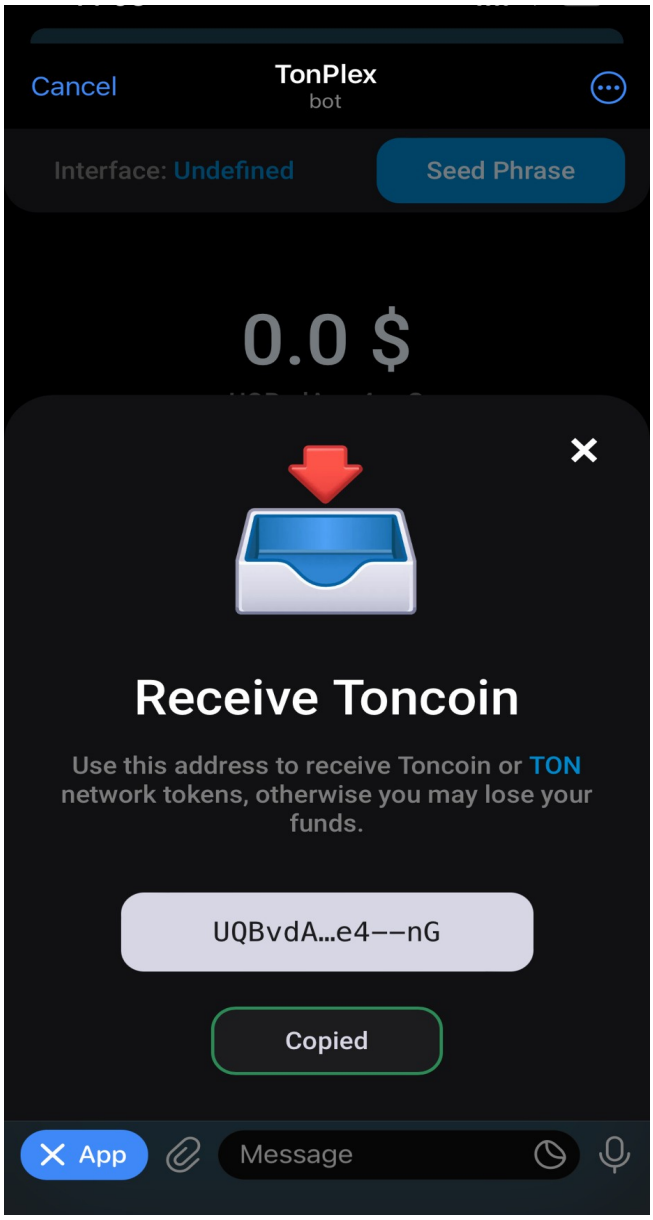

Рис. 5.5 — Модальне вікно для копіювання гаманця

Повідомлення про надходження коштів наразі не реалізовані, проте можна підписати кожного користувача на повідомлення про транзакції в його гаманці і надсилати повідомлення через Telegram-бота.

На адресу гаманця був відправлений перший Toncoin, який успішно зарахувався на раніше неіснуючу (unexist) адресу пам'яті завдяки формату non-bounceable. Тепер наша адреса має статус uninit. Цей статус означає, що адреса зареєстрована в блокчейні, але ще не була ініціалізована. Адреса існує в мережі, і на ньому можуть бути токени, але він ще не містить коду контракту або інші дані, необхідні для повної функціональності. Також адреса змінилася в браузері блокчейну, це пов'язано з форматом останньої виконаної транзакції. (Рис. 5.6)

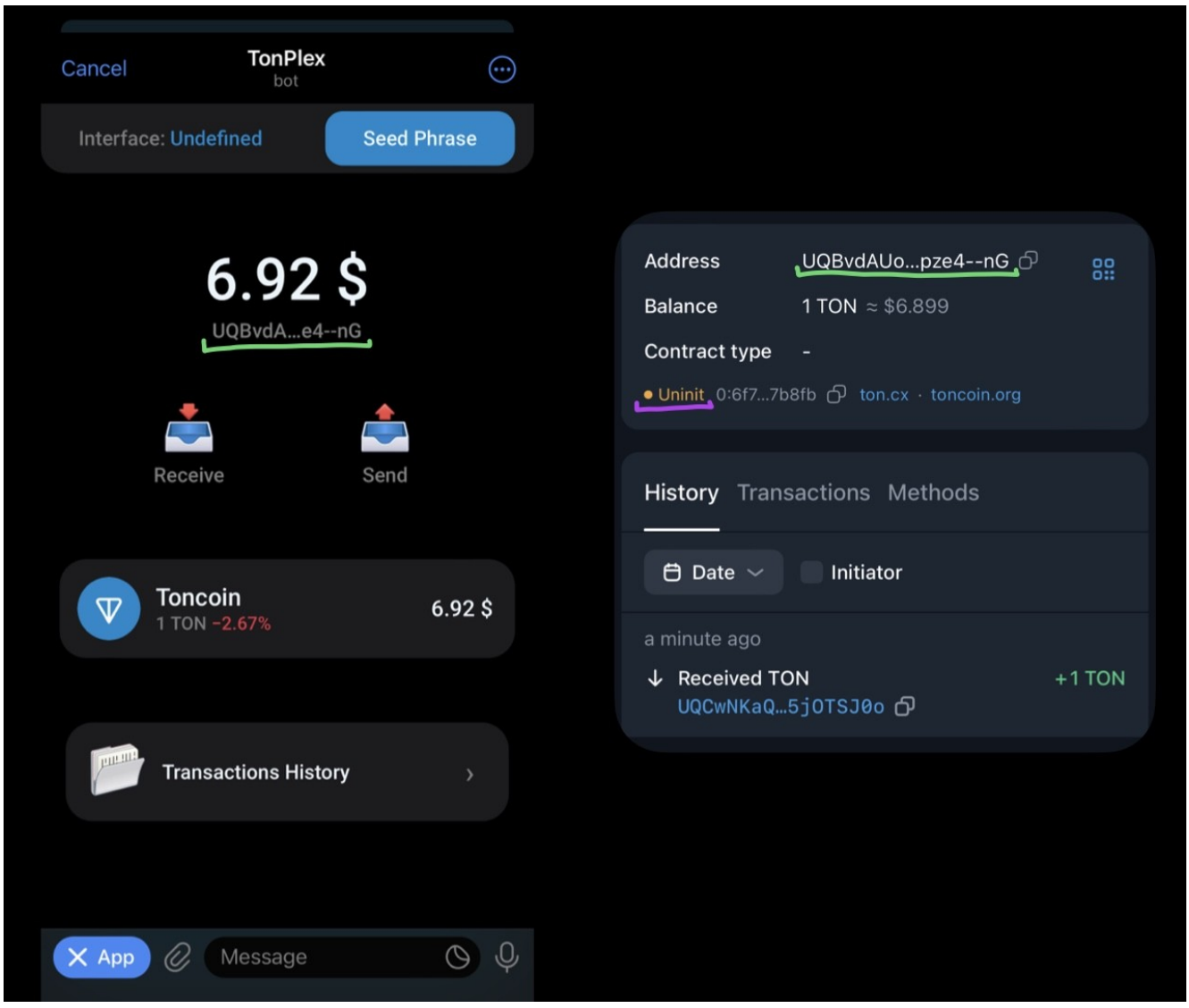

Рис. 5.6 — Отримання першого Toncoin

5.5. Відправка Toncoin

Щоб відправити Toncoin з гаманця, користувач натискає кнопку "Send". Спливає модальне вікно з формою, яка містить наступні поля: сума, адреса отримувача і memo (коментар до транзакції, необов'язковий для більшості переказів). Після заповнення необхідних даних користувач натискає кнопку "Confirm". (Рис. 5.7)

Додаток починає обробку транзакції, формуючи повідомлення і відправляючи його в блокчейн. Про статус транзакції користувача повідомить інше модальне вікно, яке спливає після підтвердження.

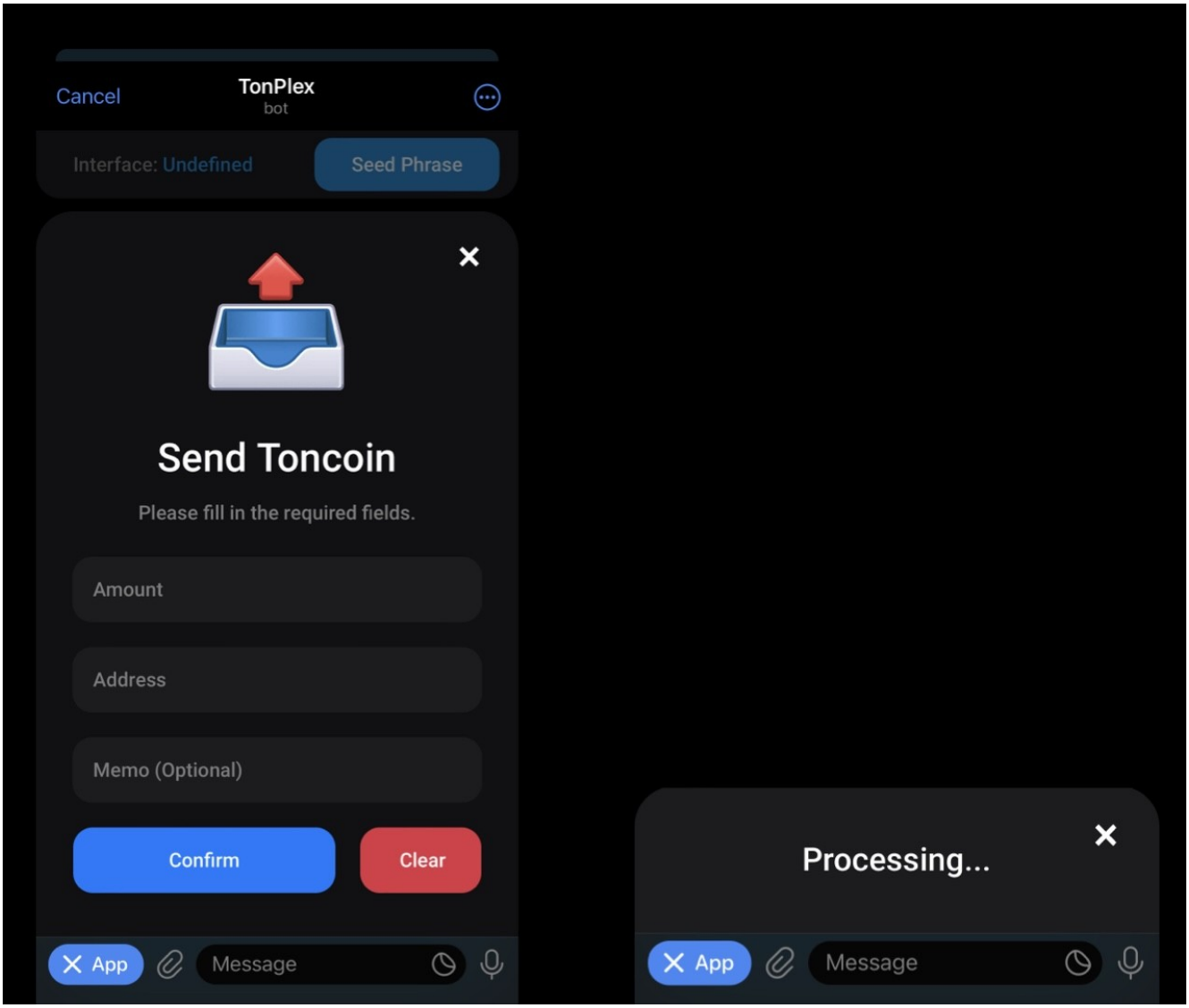

Рис. 5.7 — Відправка Toncoin

Перша транзакція викличе метод деплою гаманця, після чого гаманець буде проініціалізований і в смарт-контракт буде завантажено вибраний інтерфейс гаманця (в даному випадку wallet_v4r2) (Рис. 5.8). Після цього додаток вже буде видавати адресу гаманця для отримання в форматі bounceable.

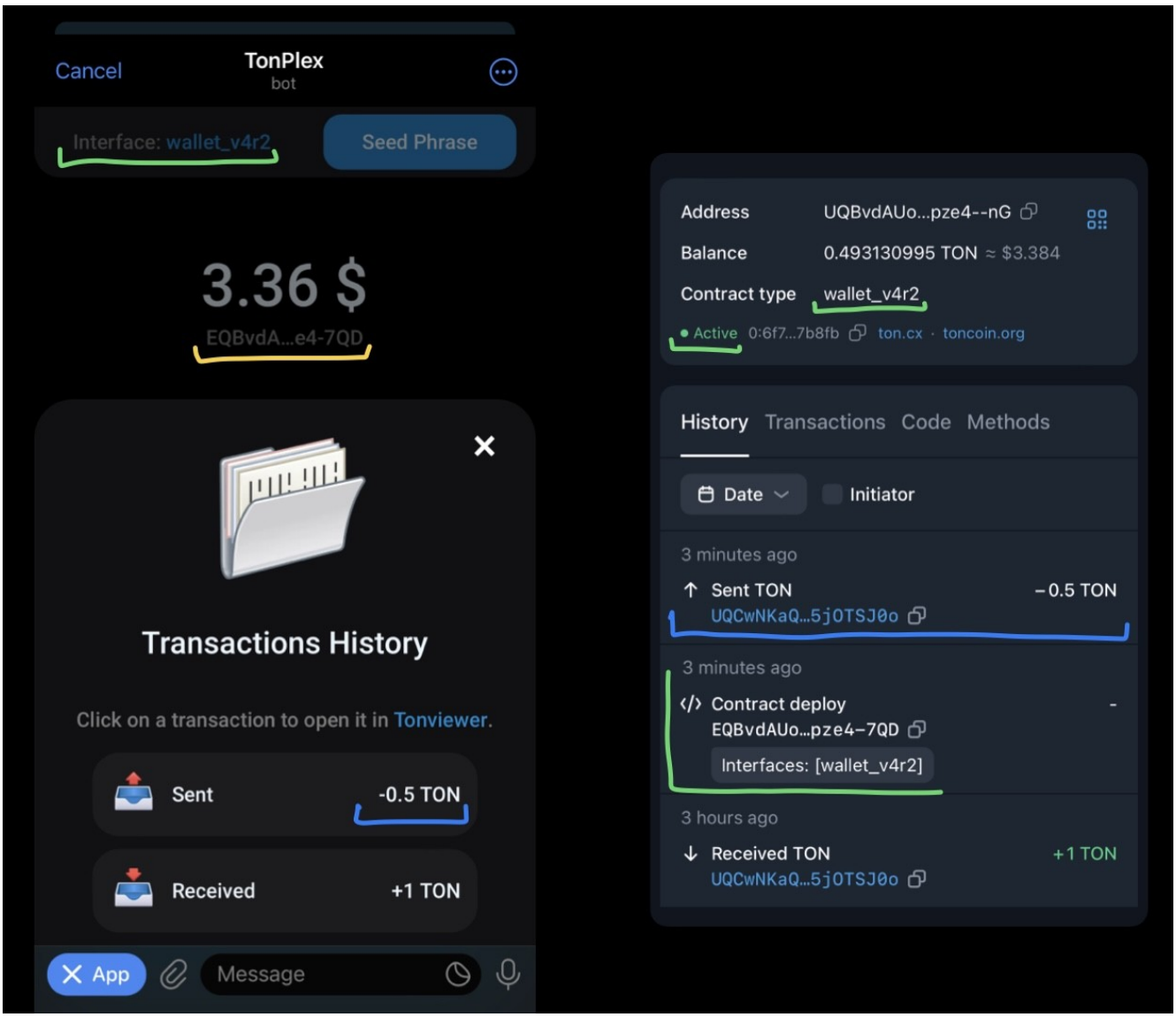

Рис. 5.8 — Деплой гаманця та історія транзакцій

Додаток також підтримує перегляд жетонів акаунту. Вони будуть відображатися в інтерфейсі, але поки що з ними не можна взаємодіяти.

ВИСНОВКИ

Під час виконання курсової роботи було розроблено та розгорнуто веб-додаток TonPlex, який представляє собою некастодіальний криптогаманець для мережі TON, інтегрований з Telegram. Я обрав платформу Railway для розгортання проекту через її простоту і функціональність, що дозволило легко підключити репозиторій та автоматизувати процес збирання і розгортання.

Під час роботи над проектом я опанував багато нових технологій та інструментів. Бекенд додатка був реалізований за допомогою FastAPI, SQLAlchemy та aiogram, що дозволило створити гнучку та асинхронну архітектуру. Для фронтенда використовувалися Jinja2, HTML, CSS та JavaScript, що забезпечило інтуїтивно зрозумілий та стильний інтерфейс.

Особливу увагу було приділено взаємодії з блокчейном TON. Використовуючи бібліотеки tonsdk та PyTONAPI, я реалізував функціональність створення, імпорту та управління гаманцями, а також відправлення та отримання транзакцій. У процесі роботи я глибше вивчив особливості блокчейна TON, включаючи його архітектуру та принципи роботи.

Безпека додатка була забезпечена за допомогою класів atomic_manager та encryption_manager, які гарантують атомарність операцій і шифрування даних мнемонічної фрази користувача. Тим не менш, я усвідомив необхідність подальшого покращення безпеки, особливо в частині захисту даних у GET та POST-запитах.

Проект TonPlex є прикладом успішного застосування різних технологій для створення функціонального та безпечного криптогаманця. У подальшому планується розширення функціональності додатка, включаючи підтримку взаємодії з токенами, NFT та реалізацію сповіщень про транзакції. Також я

планую розробити реактивний веб-інтерфейс, який підтримує всі сучасні можливості, надані Telegram для своїх додатків, що мені не вдалося повністю реалізувати через обмеження використовуваної бібліотеки JavaScript. Перспективи зростання та розвитку додатка величезні.

Цей проект дав мені цінний досвід у розробці веб-додатків і роботі з блокчейном, що, безсумнівно, стане основою для подальшого професійного зростання і розвитку в галузі програмування та блокчейн-технологій.

Метою роботи було об'єднати безліч технологій і подивитися, як вони працюють разом. Я хотів зробити більше, ніж міг, те, чого не вмів. На мою думку, мету було досягнуто. Оцінюю свій внесок у цю роботу на 100%, оскільки вона була мені надзвичайно цікава і я був повністю залучений у процес. Веб3 і DeFi — це майбутнє.