МАГІСТЕРСЬКА РОБОТА

МР. ІП - 29.00.00.000 ПЗ

Група ІПм-23-2

Кузь Павло

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Кузь Павло Васильович

(прізвище, ім'я, по батькові)

УДК ___004.942_
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи створення ботів для розміщення реклами в каналах та групах соціальних меседжерів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Кузь П.В.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Лютак Ігор Зіновійович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль
доц. Вовк Р.Б.
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Кузь Павлу Васильовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі та методи створення ботів для розміщення реклами в каналах та групах соціальних месенджерів**”**

керівник проекту (роботи) Лютак Ігор Зіновійович, д.т.н., професор

затверджені наказом закладу вищої освіти від **“** 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Архітектура, формальний опис та алгоритми функціонування рекламних ботів.

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

1. Опис розробки, постановка задач, аналіз джерел та системний аналіз
2. Аналіз сучасних технологій для поширення реклами за допомогою ботів.
3. Аналіз проблем управління розміщенням реклами
4. Формулювання вимог та алгоритмів функціонування бота
5. Програмна реалізація рішення

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Архітектура системи (рис. 2.1, ст. 28)
2. Структура бази даних рекламного бота (рис. 2.13, ст. 43)
3. Структура бази даних платіжної системи (рис. 2.12, ст. 40)
4. Діаграми класів платіжної системи та бота (рис. 3.2. - 3.3, ст. 50 – 51)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Опис розробки, постановка задач, аналіз джерел та системний аналіз	20.09.2024	виконано
2	Аналіз сучасних технологій для поширення реклами за допомогою ботів	01.10.2024	виконано
3	Аналіз проблем управління розміщенням реклами	12.10.2024	виконано
4	Формулювання вимог та алгоритмів функціонування бота	25.10.2024	виконано
5	Затвердження пояснювальної записки роботи завідувачем кафедри	05.11.2024	виконано

Студент – магіст _____
(підпис**)**

Керівник роботи _____
(підпис**)**

АНОТАЦІЯ

Магістерська робота: 96 с., 46 рис., 40 джерел.

Тема: Моделі та методи створення ботів для розміщення реклами в каналах та групах соціальних месенджерів.

Об’єкт дослідження: платформи та інструменти, що використовуються для управління рекламними оголошеннями в соціальних меседжерах, зокрема у каналах та групах, а також технології автоматизації цих процесів.

Мета роботи: є розробка ефективного інструменту для автоматизації процесу укладення та виконання рекламних контрактів у Telegram. Для цього буде створена система, яка надасть зручний інтерфейс для власників каналів для публікації цін та умов, а також для рекламодавців для налаштування рекламних форматів.

Предмет дослідження: інформаційні системи для управління рекламою з використанням API меседжера та блокчейну TRON для організації транзакцій.

Результати дослідження: проведено аналіз наявних систем управління рекламою, на основі якого запропоновано унікальну архітектуру інформаційної платформи та розроблено алгоритми її функціонування.

Висновок: результатом стала інформаційна платформа, яка спрощує управління рекламними кампаніями в TG і дозволяє компаніям ефективніше взаємодіяти зі своєю цільовою аудиторією; інтеграція з блокчейном TRON забезпечує високий ступінь безпеки і зручності для проведення фінансових операцій; а також нова система розрахована на використання блокчейну TRON. Запропоноване рішення є інноваційним інструментом, який може значно оптимізувати процес управління рекламою в месенджері.

АВТОРИЗАЦІЯ, ВЕРИФІКАЦІЯ, ТОКЕН АВТОРИЗАЦІЇ, СТЕЙБЛКОІН, БЛОКЧЕЙН, ЛІКВІДНІСТЬ, ФРЕЙМВОРК, ВЕБ-СЕРВЕР, РЕКЛАМНИЙ БОТ, ФРЕЙМВОРК.

ANNOTATION

Master's work: 96 p., 46 pic. , 40 sources.

Topic: Models and methods of creating bots for advertising in social messenger channels and groups.

Object of research: platforms and tools used to manage ads in social messengers, including channels and groups, as well as technologies for automating these processes. This includes both traditional advertising platforms and the latest approaches to automating campaign management through bots, using messenger APIs, and integration with distribution networks.

Purpose: create and optimize an effective information platform for managing advertising in messengers that will ensure user convenience, efficient management of advertising campaigns, and a high level of security of operations, especially when using financial transactions and personal data of users. The platform should allow for efficient ad customization, planning, targeting, and monitoring.

Subject of research: information systems for advertising management using the messenger API and the TRON blockchain for organizing transactions.

Research results: an analysis of existing advertising management systems was carried out, on the basis of which a unique architecture of the information platform was proposed and algorithms for its functioning were developed.

Conclusion: the result is an information platform that simplifies the management of advertising campaigns on Telegram and allows companies to interact more effectively with their target audience; integration with the TRON blockchain provides a high degree of security and convenience for financial transactions; and the new system is designed to use the TRON blockchain. The proposed solution is an innovative tool that can significantly optimize the process of advertising management in the messenger.

AUTHORIZATION, VERIFICATION, AUTHORIZATION TOKEN, STABLECOIN, BLOCKCHAIN, LIQUIDITY, FRAMEWORK, WEB SERVER, AD BOT, FRAMEWORK.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 9

ВСТУП 10

РОЗДІЛ 1

ОПИС РОЗРОБКИ, ПОСТАНОВКА ЗАДАЧ, АНАЛІЗ ДЖЕРЕЛ ТА СИСТЕМНИЙ АНАЛІЗ 13

1.1 Постановка задачі та характеристика розробки 13

1.2 Огляд літературних джерел 14

1.3 Системний аналіз 22

1.4. Висновки до розділу 26

РОЗДІЛ 2

СТРУКТУРИ, АЛГОРИТМИ ІНФОРМАЦІЙНИХ СИСТЕМ 28

2.1. Структури та архітектури систем 28

2.2. Блок-схеми опису алгоритмів 29

2.3. Діаграма функціональності систем 31

2.4. База даних PostgreSQL 39

2.5. Висновки до розділу 47

РОЗДІЛ 3

РОЗРОБКА МОДЕЛЕЙ І МЕТОДІВ ПРОГРАМНОГО РІШЕННЯ 48

3.1 Загальна структура програмного забезпечення 48

3.2. Діаграма класів 49

3.3. Використані сторонні бібліотеки та модулі 53

3.4. Розробка та опис програмних модулів 54

3.5. Розробка та опис інтерфейсу користувача 57

3.6. Альтернативні підходи, що розглядалися протягом розробки 60

3.7. Проблеми та незвичні ситуації, які виникали під час розробки та методи для їх протидії..... 60
3.8. Висновки до розділу 60
РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА..... 63
4.1. Інструкції для адміністратора, програміста, користувача 63
4.2. Вимоги до апаратно-програмного забезпечення 64
4.3. Тестування 64
4.4. Аналіз та оцінка отриманих результатів 74
4.5. Висновки до розділу 75
ВИСНОВКИ 77
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 78
ДОДАТКИ..... 81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP (Hyper Text Transfer Protocol) - це протокол передачі гіпертексту, який використовується для обміну інформацією між веб-серверами і клієнтськими програмами

РБ – Рекламний бот.

БД – база даних.

FK – зовнішній ключ

TG – назва меседжеру

ВСТУП

Актуальність роботи

Забезпечення надійного виконання та своєчасної оплати рекламних послуг є критично важливим для налагодження довгострокових вигідних партнерських відносин. В умовах сучасного цифрового простору Telegram стає одним з найбільш ефективних засобів для забезпечення виконання договірних зобов'язань, зокрема через використання посередників для автоматизації процесів. Зі зростанням популярності спільнот у Telegram виникають нові можливості для реклами, але й не менше проблем через невиконання умов договорів однією зі сторін. Тому створення надійних рекламних бото-систем, які гарантують виконання угод, є надзвичайно актуальним. Це дослідження спрямоване на вирішення цієї проблеми, що підкреслює його важливість.

Порівняння з відомими рішеннями проблеми

Сучасні системи для розміщення реклами в месенджерах зазвичай не забезпечують повної автоматизації та прозорості виконання рекламних договорів. Існуючі рішення часто стикаються з проблемами ненадійності та неефективності в управлінні рекламними контрактами. Відсутність єдиного посередника для контролю за виконанням зобов'язань створює ризики для учасників рекламного процесу. Однак розробка спеціалізованих ботів у Telegram відкриває нові можливості для покращення цих процесів.

Мета і задачі дослідження

Метою магістерської роботи є розробка ефективного інструменту для автоматизації процесу укладення та виконання рекламних контрактів у Telegram. Для цього буде створена система, яка надасть зручний інтерфейс для власників каналів для публікації цін та умов, а також для рекламодавців для налаштування рекламних форматів. Задачами дослідження є:

- Оцінка ефективності існуючих рекламних систем.
- Визначення вимог до системи автоматизації рекламних контрактів.

- Розробка алгоритмів функціонування рекламного бота.
- Створення прототипу для демонстрації роботи системи.

Об’єкт дослідження

Платформи та інструменти, що використовуються для управління рекламними оголошеннями в соціальних меседжерах, зокрема у каналах та групах, а також технології автоматизації цих процесів.

Предмет дослідження

Інформаційні системи для управління рекламою з використанням API меседжера та блокчейну TRON для організації транзакцій.

Методи дослідження

У дослідженні будуть застосовані методи аналізу існуючих рішень, проектування алгоритмів, розробки програмного забезпечення та тестування прототипів систем. Для порівняння ефективності розробленої системи з існуючими рішеннями використовуватимуться методи оцінки ефективності програмних продуктів.

Наукова новизна отриманих результатів

Наукова новизна полягає у розробці інноваційної системи автоматизації рекламних контрактів у Telegram, що гарантує виконання всіх умов угоди за допомогою спеціалізованого бота, що працює як надійний посередник між рекламодавцями та власниками каналів.

Практичне значення отриманих результатів

Практичне значення полягає в розробці ефективного інструменту для автоматизації процесу розміщення реклами, який забезпечує надійність рекламних контрактів, знижує ризики невиконання зобов'язань і підвищує довіру між рекламодавцями та власниками каналів. Система може бути використана для

покращення прозорості та ефективності рекламних кампаній у Telegram та інших месенджерах.

Структура магістерської роботи.

Магістерська робота викладена на 96 сторінках друкованого тексту, який складається з вступу, чотирьох розділів, висновків, списку використаних джерел (40 найменувань). Робота містить 46 рисунків.

РОЗДІЛ 1
ОПИС РОЗРОБКИ, ПОСТАНОВКА ЗАДАЧ, АНАЛІЗ ДЖЕРЕЛ ТА СИСТЕМНИЙ АНАЛІЗ

1.1 Постановка задачі та характеристика розробки

Система повинна надавати зручний інтерфейс для додавання ботів до каналів/груп, створення, обробки та виконання рекламних контрактів.

Система повинна мати наступні функціональні можливості

- додавання користувачів, каналів та груп до бази даних
- створення шаблонів публікації оголошень
- створення оголошень
- надання власникам каналів/груп доступу до управління та прийняття рекламних пропозицій для інших користувачів
- автоматична публікація оголошень
- поповнення та виведення коштів.

В якості мови програмування було обрано Python, а в якості бази даних для зберігання даних користувачів та рекламних контрактів - PostgreSQL. Для ефективної взаємодії з користувачами потрібен спеціальний інструмент, який дозволяє без проблем реєструвати та керувати базою даних.

Кожному користувачеві меседжера присвоюється унікальний персональний ідентифікатор, до якого зареєстрований бот може отримати доступ через іншого бота, @botfather.

Після реєстрації бота в обраному нами меседжері, необхідно додати користувачів, канали, групи до бази даних та розробити ключові команди для того щоб почати діалог з ботом. Завдяки меседжеру бот отримує всю необхідну інформацію, таку як повідомлення, списки користувачів у групах або каналах, додавання та видалення ботів, зміни в доступі користувачів та ботів тощо. Це дозволяє боту моментально наповнювати свою базу даних необхідною інформацією та перевіряти, чи має він усі права, необхідні для повноцінної роботи.

Головним інтерфейсом для взаємодії користувача з даним ботом є веб-компонент, який легко інтегрується у меседжер за допомогою WebApp. Глибока інтеграція в меседжер забезпечує розширений функціонал, створює дуже зручний інтерфейс і спрощує доступ до інформації. Як тільки користувач заходить у WebApp меседжеру, його дані перенаправляються на вебсторінки системи, усуваючи необхідність зайвої реєстрації або входу. Автоматизація рекламної системи в каналі чи групі меседжеру включає в себе договір, який укладається при створенні бота. Дана автоматизація дозволяє ретельно контролювати дотримання умов, зазначених у договорі між рекламодавцем і власником каналу чи групи.

На другому місці після платіжних систем, оскільки значної популярності набувають криптовалюти завдяки таким особливостям, як анонімність адрес, швидкість транзакцій та глобальна доступність. Криптовалютні транзакції, як правило, більш безпечні та прозорі завдяки технології блокчейн, що підвищує довіру та надійність, а використання в РБ криптовалют як основного платіжного засобу відповідає сучасним тенденціям та численним перевагам цифрових валют.

1.2 Огляд літературних джерел

1.2.1 Функціональність та визначення сервісів реклами

Advertising Services - це онлайн-платформа, яка надає широкий спектр інструментів для створення, управління та оптимізації рекламних кампаній в Інтернеті. Вони використовуються компаніями всіх розмірів і галузей для досягнення наступних маркетингових цілей, а саме:

- Збільшення продажів: Рекламні кампанії залучають трафік на веб-сайти та мобільні додатки, що спонукає користувачів здійснювати покупки.
- Підвищення впізнаваності бренду: реклама може просувати бренд, його продукти та послуги серед потенційних клієнтів.

- Генерувати ліди: рекламні кампанії дозволяють збирати контактні дані потенційних клієнтів і використовувати їх для подальшого маркетингу. Ключові можливості рекламної послуги.
- Створення рекламних кампаній: рекламодавці можуть створювати різні типи рекламних кампаній, включаючи текстові оголошення, іміджеві оголошення, відеорекламу та рекламу в соціальних мережах.
- Націлювання на аудиторію: рекламний сервіс може точно таргетувати рекламу на конкретну аудиторію на основі демографічних даних, інтересів, поведінки в Інтернеті тощо.
- Бюджет: рекламодавці можуть встановлювати бюджети на рекламні кампанії та контролювати їхні витрати.
- Відстеження та аналіз: рекламний сервіс надає детальну статистику ефективності рекламних кампаній, що дозволяє рекламодавцям відстежувати результати та вносити потрібні їм корективи.

1.2.2 Різновили сервісів реклами

Існує два основних типи рекламних послуг:

- Контекстна реклама: цей тип реклами показується користувачам на основі їхніх пошукових запитів та інтересів. Рекламодавці платять плату за кожен клік на їхнє оголошення (CPC).
- Медійна реклама: цей тип реклами показується користувачам на веб-сайтах і в мобільних додатках. Рекламодавці можуть платити за 1000 показів (CPM) або за клік (CPC).

Найпопулярнішими рекламними сервісами є:

- Google Ads: найпоширеніша платформа контекстної та медійної реклами.
- Реклама на Facebook: соціальна мережа Facebook дозволяє таргетувати рекламу на основі інтересів, демографічних даних та поведінки користувачів.
- Реклама Amazon: Рекламна платформа Amazon, яка дозволяє рекламодавцям рекламувати свої продукти на платформі Amazon.

- LinkedIn Ads: дана платформа соціальної мережі LinkedIn, яка дозволяє розміщувати таргетовану рекламу серед професіоналів у певних галузях.
- YouTube Ads: відеохостинг YouTube дозволяє показувати відеорекламу до, під час і після відео на YouTube.

1.2.3 Трактування поняття WebApp у меседжерах

WebApp - це веб-додаток, який інтегрується з платформою обраного нами меседжера. Це дає користувачам меседжеру отримувати доступ до додаткових функцій і сервісів безпосередньо з месенджера, не виходячи з нього.

WebApp має багато можливостей:

- Доступність: веб-додаток доступний через вбудований браузер, тому його можна використовувати на будь-якому пристрої, який використовує меседжер.
- Інтеграція: Веб-додатки інтегровуються з меседжером і можуть використовувати дані користувачів, а саме профілі та чати, для покращення користувацького досвіду.
- Гнучкість: Веб-додатки можна створювати за допомогою HTML, CSS та JavaScript, що дає розробникам можливість створювати широкий спектр функціональних можливостей.
- Безпека: Оскільки веб-додатки працюють в ізольованому середовищі, безпека даних користувачів гарантована;

WebApps меседжеру можна використовувати для різних цілей, а саме:

- Надання додаткових послуг: WebApps можна використовувати для онлайн-ігор, редагування фотографій, зберігання файлів тощо: WebApps можуть використовуватися для надання додаткових послуг користувачам legram, таких як онлайн-ігри, редагування фотографій, зберігання файлів тощо.
- Інтеграція зі сторонніми сервісами: WebApps можна використовувати для інтеграції Telegram зі сторонніми сервісами, такими як соціальні мережі, платіжні системи тощо.

- Автоматизація завдань: доступна можливість використовувати для автоматизації таких завдань, як планування зустрічей, створення нагадувань та інших.

1.2.4 Огляд технологій

Aiogram - бібліотека Python для розробки ботів TG, яка надає асинхронний API, що дозволяє легко створювати ботів, які можуть ефективно обробляти кілька запитів водночасно [2]. Дана бібліотека також пропонує велику кількість функцій, які допоможуть розробникам ботів, а саме:

- Розпізнавання команд і запитів
- Обробка повідомлень - Відображення клавіатури і меню
- Моніторинг стану користувача
- Зберігання даних
- Розгортання бота

Aiogram проста у використанні, потужна і гнучка, що робить її популярним вибором для розробників TG-ботів[1].

python-telegram-bot – дуже поширена у використанні бібліотека створення TG-ботів на основі Python, котра надає синхронний API для взаємодії з TG Bot API та підтримує усі стандартні функції, а саме: надсилання та отримання повідомлень, обробка команд, створення клавіатури, обробка мультимедіа та документів[3].

Telepot - також Python-бібліотека для створення TG –ботів. Вона підтримує синхронний та асинхронний API, що відповідно дозволяє розробникам обирати підхід, який являється найбільш підходящим для їхньої задумки[16].

Основний функціонал:

- Відправка та отримання повідомлень
- Обробка команд
- Відправка файлів та мультимедіа
- Підтримка Webhook

PyTelegramBotAPI або telebot – дана бібліотека одна із найпоширеніших бібліотек у даному плані [2]. А саме тому, що у неї простий у використанні синхронний API. Підтримка функцій таких , як

- Відправка та отримання повідомлень
- Створення клавіатури
- Обробка команд
- Відправлення мультимедії

Telethon - асинхронна бібліотека для взаємодії з Telegram API на мові Python, що дозволяє створювати клієнти Telegram, а також ботів, де основними можливостями є: надсилання та отримання повідомлень, обробка команд, завантаження та вивантаження медіафайлів та підтримка веб-хуків та опитувань [23].

Pyrogram - це асинхронна бібліотека, яка дозволяє створювати як ботів, так і клієнтів для взаємодії з Telegram API [24]. Ключові можливості включають в себе:

- надсилання та отримання повідомлень
- обробка команд
- завантаження та вивантаження медіафайлів
- підтримка веб-хуків та опитувань.

Описані вище бібліотеки надають можливість обирати розробникам найраціональніший інструмент для створення TG-бота відповідно до їх задач.

Фреймворк - програмний інструмент, котрий надає структуру та набір функцій для розробки ПЗ. Даний інструмент дозволяє розробникам зосередитися на логіці роботи програми. Фреймворки включають шаблони проектів, бібліотеки, інструменти та інші ресурси, котрі спрощують розробку ПЗ.

Типи фреймворків:

- Full-stack фреймворки: надають повний набір функцій для розробки, включаючи фронт- і бек-енд (Django, Ruby on Rails).
- Мікрофреймворки: надають лише мінімальну функціональність і дають гнучкість розробникам (Flask, Express).

- Асинхронні фреймворки: призначені для роботи з асинхронним програмуванням (FastAPI, Tornado) Django:

Django - це високо абстрактний веб-фреймворк на базі Python, що полегшує розробку складних веб-додатків[7]. Даний фреймворк пропонує широкий спектр функцій для допомоги розробникам, включаючи:

- Систему управління на основі моделей
- Маршрутизацію URL-адрес
- Шаблонізацію
- Систему обробки форм
- Автентифікацію та авторизацію
- Підтримку баз даних.

Django є найкращим вибором для розробників веб-додатків на Python, тоді як React - це бібліотека JavaScript для декларативних інтерфейсів користувача, що використовується для створення динамічних та інтерактивних веб-додатків[6]. React має такі переваги як:

- Компонентна модель
- Віртуальний DOM
- JSX
- Розширюваність
- Масштабна екосистема

React являється популярним вибором для front-end розробків, оскільки є простим та гнучким із можливістю масштабування.

База даних – зібрана колекція даних, котра зберігається і управляється таким чином, щоб забезпечити простий доступ, керування та оновлення. База даних використовують для збереження великої кількості даних, котра може бути ефективно оброблена і використана в будь-яких додатках, таких як вебсайти, мобільні додатки, системи управління підприємствами та інші. В даному випадку використовється база даних для зберігання даних користувачів, чатів, транзакцій.

Головні компоненти БД:

1. Дані: цифри або інформація, що зберігаються в БД. Дані можуть структуризуватися (таблиці з рядками і стовпцями) або неструктуризуватися (текстові документи, зображення).

2. СУБД: програмне забезпечення, котре дає змогу створювати, читати, обновлювати та видаляти дані в БД. СУБД також надає змогу управління користувацьким доступом, резервні копії, відновлення даних та безпеку.

3. Модель даних: структурний формат, в котрому організовані дані. Головні моделі даних включають в себе реляційні, документо-орієнтовані, графові та інші типи БД.

Типи БД:

1. Реляційні БД (SQL): облаштовують дані у виді таблиць, де дані взаємопов'язані. Використовують SQL для роботи з даними. Найвідомішими СУБД вважаються MySQL, PostgreSQL, Oracle, Microsoft SQL Server. Паревагами є структурність, змога важких запитів, забезпечення цільності даних, а недоліком менша гнучкість відносно до певних модель даних.[2]

2. Документо-орієнтовані БД або NoSQL: Зберігають дані у виді документів як JSON або BSON. Поширеними СУБД такого типу є MongoDB та CouchDB. Їх Перевагами являється гнучкість в збереженні різних типів даних та масштабованість, а недоліками менша кільість інструментів для важких запитів та транзакцій.

3. Графові БД: базуються на графовій структурі для організації та зберігання даних, де вузли представляють об'єкти, а ребра відношення між ними. Відомими СУБД вважаються Neo4j таAmazon Neptune.

Їх перевагами є ефективність для запитів ,котрі включають складні зв'язки між даними, а недоліками складність для розуміння та управління.

4. Ключ-значення бази даних: збереження даних у виді пар ключ-значення. Популярними СУБД вважаються Redis та DynamoDB. Їх перевагами є висока ефективність та простота при використанні, а недоліками мала можливість для важких запитів.

5. Стовпцеві бази даних: зберігають дані у вигляді стовпців замість рядків, що дозволяє ефективно обробляти великі обсяги даних. Популярні СУБД

включають Apache Cassandra, HBase. Їх перевагами є висока продуктивність для аналітичних запитів , а недоліками важкість у налаштуванні та управлінні.

БД є важливим для сучасних інформаційних систем тому, що забезпечуючи надійне зберігання та моментальний доступ до даних. Вибір БД залежить від певних потреб проекту, типу даних та вимог.

CSS-фреймворк – це набір визначених стилів, макетів та компонентів, котрі спрощують створення адаптивних та красивих сайтів. Фреймворки надають структуру та стандартизацію, дозволяючи розробникам швидко та легко використовувати узгоджені стилі до своїх вебзастосунків.[6]

Основні функції CSS-фреймворків: сіткова система (адаптивні макети, котрі автоматично підлаштовуються під будь-які розміри дисплею, що дає змогу легко створювати дизайн, котрий у висновку буде гарно виглядати на будь-яких пристроях, а саме в діапазоні від мобільних телефонів до великих моніторів), попередньо визначені стилі (готові для використання стилі для стандартних вебу, а саме кнопки, форми, таблиці, навігаційні меню та багато інших, що зменшує час на розробку і забезпечує балансний вигляд), компоненти (готові компоненти - модальні вікна, каруселі, спливаючі вікна та вкладки), адаптивність (засоби для створення адаптивних дизайнів що підлаштовуються під будь-які розміри екранів та девайсів, що доволі зручно до користування), кросбраузерність (гарантують одинаковий вигляду та роботу сайтів у будь-якому відомому браузері, відповідно це надає нам змогу уникнути проблем з сумісністю).

Одним із найпопулярніших CSS-фреймворків є Bootstrap, котрий забезпечує широкодіапазонну гаму компонентів для створення адаптивних сайтів.

Фреймворк Bulma базується на flexbox та надає сучастний та простий набір компонентів.

Foundation фреймворк надає потужні айтеми для створення адаптивних сайті заточених під мобільність.

Semantic UI використовує просту людську мову для опису дизайну, а це в свою чергу дає нам читабельний та легкий до розуміння код. На рисунку 1.1 зображені популярні CSS-фрейморки [41].

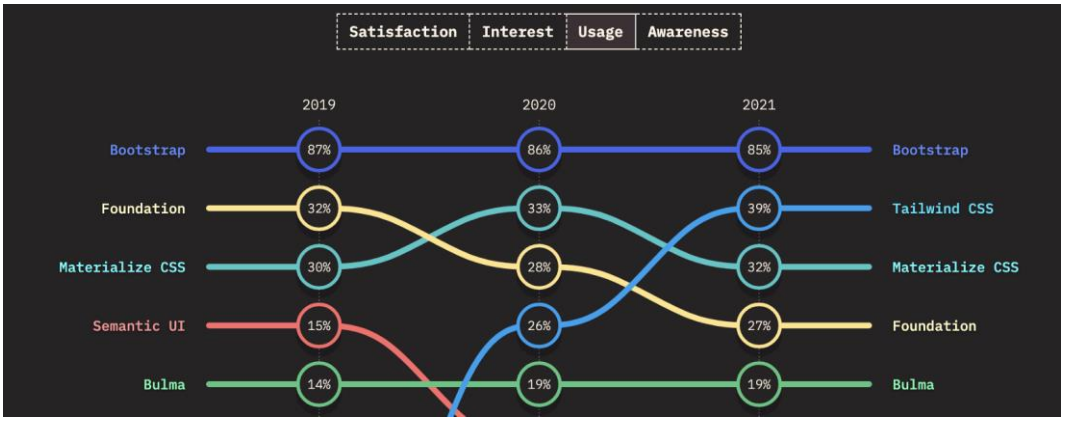

Рис. 1.1. Популярні CSS-фрейморки

CSS-фрейморки зпрощують розробку сайтів та забезпечують узгодження, адаптацію та швидкість створення, дозволяючи творцям фокусуватись на функціональності та досвіді використання.[5]

Даний вид бібліотек дозволяє розробника інтегрувати у свій додаток плаптіжки на основі блокчейн-мереж за допомогою зручного API, а також можливість розгортання смарт-контрактів та зберігання даних.

Одною із таких бібліотех є TronPy, яка відповідно базується на мові Python і створена для взаємодії з блокчейном TRON. Має в наявності зручний API, який дозволяє розробникам сворювати децентралізовані програми на платформі TRON. А також пропонує певний спектр функцій , що допомагають розробникам, а саме: створення та управління смарт-контрактами, взаємодія зі смарт-котрактами, розгортання децентралізованих програм та переказ відповідно системної монети TRON. TronPy являється поширеним вибором розробників децентралізовани програм TRON через його простоту, потужність та гнучкість[15].

1.3 Системний аналіз

1.3.1 Побудова та опис дерева проблеми

Дерево проблем – інструмент, котрий використовують для структуризованого опису проблем, їх причин і наслідків. Дає змогу спроектувати

складні проблеми, проаналізувати їх вихідні точки та знаходити можливі методи вирішення.

Працює дерево проблем наступним чином (додаток В):

1) Визначення головної проблеми, яка записується на вершині дерева.

2) Виявлення причин – записуються як гілки, що відходять від основної проблеми.

3) Деталізація причин – це коли кожна з причин може бути розділена на більш дрібні підпричини, що аналогічно відображаються як гілки на дереві.

4) Аналіз наслідків – дерево також може показувати наслідки проблем.

Центральним елементом багатьох методів планування проектів є аналіз дерева проблем. Даний метод допомагає знаходити вирішення, розклавши все по поличках як причин, так і наслідків відносно проблеми.

Згідно осовною проблемою являється необхідність організації надійної та ефективної системи для керування рекламними оголошеннями та транзакціями в TG за допомогою бота [38].

Гілки головної проблеми виглядають наступним чином:

1. Фінансові проблеми :

- Непердбачувані комісії за транкзакції, оскільки криптовалюта дуже волатильна та високі комісія за маленькі транкзакції.
- Фінансові втрати через помилки у транзакціях, а саме недоліки у верифікації платіжок та помилки в обробці їх запитів.
- Ризик замороження крипторахунків – причинами цього можуть бути зміни у політиці та технічні проблеми блокчейну.

2. Забезпечення ліквідності для покриття комісій – відсутність потрібної кількості валюти для оплати комісії та трабли з керуванням ліквідністю між кошельками.

3. Проблема з користувачами – неграмотно реалізований інтерфейс та дизайн без відповідного гайду користувача. Також імовірна проблема з реєстрацією та авторизацією, оскільки невідповідна надійність процесу та складнощі у відновленні даних для авторизації.

4. Технічні проблеми – інтеграція API TG через незадокументовані зміни та обмеження кількості запитів. Вразливість безпеки під час обробки даних юзера звідси відповідно випливає можливість несанкціонованого доступ до інформації. Також проблеми з масштабуванням серверної сторони через виликий потік запитів та обмеження ресурсного обладнання. Цілком можливе нестабільне з'єднання з блокчейном TRON, а саме погана швидкість підтвердження платежів та вразливість до кібератак. Сладність забезпечення хорошої продуктивності ПЗ, а саме неоптимізований код серверної сторони та проблеми з керуванням пам'ятю.[18]

1.3.2 Вибір та аналіз алгоритмів, методів та засобів розв'язання задачі

Каскадна модель — процес поступової розробки, в котрому розробка програмного забезпечення задумується набром етапів, котрі виконуються один за одним. Дана модель названа згідно з позицій, котрі займають різні фази, що складають проект, розміщені одна на одну та слідують за потоком виконання зверху вниз. Модель каскадного розвитку виникла в промисловості та будівництві, де зміни дороговартісні та важко реалізовані. Якщо ви створюєте матеріальний продукт, вносити зміни до вже створеного більш складніше, аніж у комп'ютерній програмі. У світі ПЗ інші методології розробки ще не були створені, тому каскадна модель, котра використовувалася в інших секторах, була адаптована.

Етапи каскадної моделі:

1. Вимоги до ПЗ – на даному етапі детально описуються всі функції та можливості, котрі повинна мати система. Даний етап важливий, оскільки будь-які зміни вимог на пізніших етапах розробки ПЗ можуть бути надто складними та дорогими.

2. Проектування – створюється архітектура системи та описується те, як вона буде працювати, а також детальні технічні специфікації, котрі описують усі компоненти системи.

3. Реалізація – відбувається написання коду ПЗ відповідно до специфікацій, що були розроблені на попередньому етапі.

4. Тестування – ПЗ ретельно тестується для виявлення та виправлення усіх помилок. Для проходження даного етапу використовують різні методи тестування, а саме такі як модульне тестування, інтеграційне тестування та системне тестування.

5. Встановлення та обслуговування сервісу – розгортається сервіс в середовищі користувачів і надаватиметься необхідна підтримка.

Може включати: встановлення та налаштування(розміщення сервісу на серверах або в хмарному середовищі користувача, налаштування параметрів згідно до потреб та інтеграцію з іншими системами), навчання та підтримка користувачів (надання інструкцій та допомоги з використанням сервісу, а також відповіді на їхні запитання та вирішення проблем), виправлення помилок в роботі сервісу, а також випуск оновлень з новими функціями та покращеннями та постійний моніторинг роботоспосібності сервісу, а також виявлення та усунення імовірних проблем, а також проведення планового обслуговування.[16]

А також потрібно обговорити інші аспекти цієї моделі, а саме переваги Каскадної моделі: простота та зрозумілість (легко сприймається та має чіткий поділ на етапи робить її доступною для розуміння та керування, полегшуючи комунікацію в команді), чітка структура (має чітко окреслені межі та цілі, що дозволяє чітко планувати роботу, розподіляти задачі та відстежувати прогрес, відповідно забезпечуючи чіткий порядок дій та передбачуваність результатів), раннє виявлення помилок(завдяки послідовності етапів помилки, як правило, виявляються на ранніх стадіях розробки, коли їх виправлення є значно простішим та дешевшим, що мінімізує ризики та економить ресурси, адже помилки, виявлені на пізніх етапах, можуть мати значно більші наслідки).

Недоліками каскадної моделі являються: відсутність гнучкості (жорстка структура Каскадної моделі робить її не дуже гнучкою. Зміни у вимогах до сервісу, які виникають на пізніх етапах проекту, можуть бути дуже складними та дорогими, що відповідно може призвести до затримок та додаткових витрат або навіть до необхідності переробити вже виконану роботу), тривалість розробки (послідовність етапів може значно збільшити час розробки, оскільки кожен етап має бути завершений перед переходом до наступного, що може призвести до

тривалих періодів очікування, коли інші члени команди не задіяні), високі ризики (каскадна модель пов'язана з високими ризиками, адже помилки, виявлені на пізніх етапах, можуть бути дуже дорогими у виправленні, що може призвести до перевищення бюджету, затримок проекту та незадоволення клієнтів).[22]

1.3.3 Схема системи з урахуванням інформаційних потоків

Структурна схема системи – графічне зображення архітектури, котре показує всі її компоненти та їх взаємозв'язки. Завдяки даній схемі можна чітко уявити організованість складової системи та як вони співпрацюють між собою та забезпечують функціональність додатку. На рисунку нижче зображено структурну схему для наглядної демонстрації:

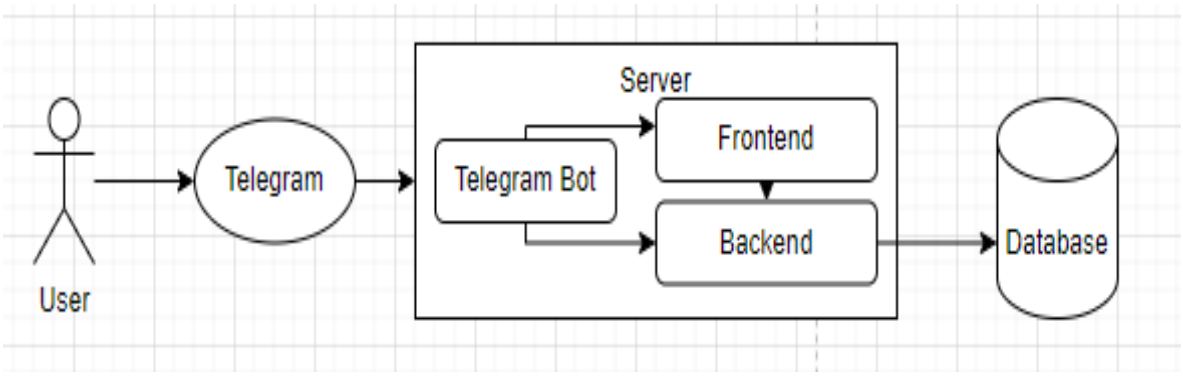

Рис. 1.2. Структурна схема системи

На рисунку 1.3, користувач взаємодіє зі системою через TG, а в подальшому через бота, котрий отримує потрібну інформацію про користувача і відправляє на серверну частину, для обробки і в подальшому в базу даних. TG бот може направити користувача на сайт через WebApp, де всі дії/зміни проходять через сервер і також зберігаються в базі даних.

Тобто вся система скрадається з 4 частин, а саме: TG бот, вебсайт, API та база даних.

1.4. Висновки до розділу

Під час оцінинки мети розробки та вимоги до завдань було сфокусовано

увагу на створення автоматизованої системи розміщення реклами в TG каналах та групах. Мета полягає у створені зручного та ефективного інструменту для власників каналів, де буде змога розміщувати розцінки та здійснювати угоди із замовниками реклами. Для замовників метою є отримання можливості обирати та налаштовувати рекламу згідно їх потреб.

Протягом огляду літератури було переглянуто різноманітну літературу та документацію, котра стосується розвитку автоматизованої системи розміщення реклами в TG каналах та групах. Цей огляд розширив розуміння контексту та вимог до даної системи.

Під час системного аналізу було побудовано дерево проблем і дерево цілей. Дерево проблем дало змогу визначити основні проблеми і питання, котрі потребували вирішення в процесі створення ПЗ. А також у реверсивному значені, дерево цілей було інструментом для визначення ключових цілей і завдань, потрібно для ефективного вирішення цих проблем.

Потім було проведено аналіз та обрано методи, алгоритми та інструменти для вирішення виявлених проблем. Це включало вибір технологій, фреймворків та інструментів, котрі найефективніше відповідають вимогам програми. Окрім того, ми розробили структурну схему системи з урахуванням інформаційних потоків, котра ілюструє організацію та взаємозв'язки між компонентами системи.

Тобто, за допомогою аналізу та вивчення літературних джерел зробили висновок на рахунок об'єкту розробки, постановки задачі та системного аналізу. Цей аналіз визначив критичні аспекти та напрямки розвитку системи, орієнтованої на криптовалютних інвесторів.

РОЗДІЛ 2
СТРУКТУРИ, АЛГОРИТМИ ІНФОРМАЦІЙНИХ СИСТЕМ

2.1. Структури та архітектури систем

Архітектура системи складається з TG-бота і веб-клієнтської частини, front-end та back-end.[20]

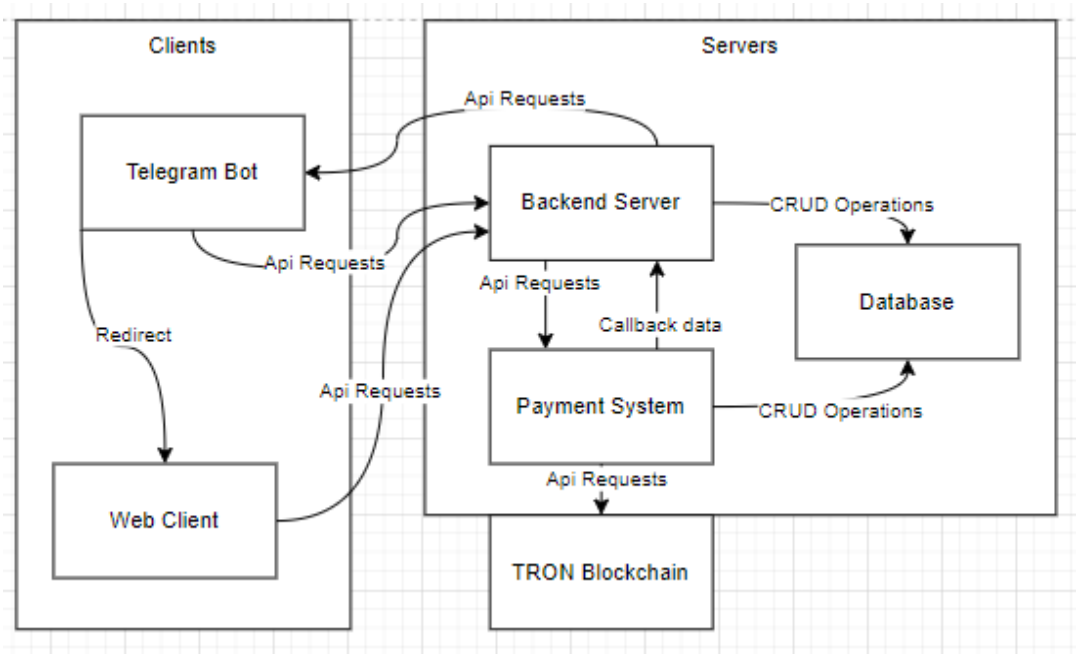

Рис. 2.1. Архітектура системи

Клієнти: TG Bot (компонент взаємодіє з користувачами через TG, отримуючи команди та надсилаючи запити до Back-end Server), Web Client(забезпечує користувачам зручний веб-інтерфейс для роботи з системою, здійснюючи запити до Back-end Server.

Сервери: Back-end Server (основний серверний компонент, який обробляє запити від TG Bot і Web Client та взаємодіє з іншими серверними компонентами та базою даних для забезпечення функціональності), database(сховище для даних системи, котре містить інформацію про юзера, чати, оголошення та платіжки),Payment System(компонент обробляє фінансові операції, включаючи

поповнення балансу та виведення коштів, що взаємодіє з блокчейном TRON для здійснення транзакцій).

Взаємодії між компонентами:

- TG Bot відправляє запити до Back-end Server для обробки команд користувача.
- Web Client надсилає запит до Back-end Server для взаємодії з системою через веб-інтерфейс.
- Back-end Server обробляє запити та передає дані до Payment System для обробки транзакцій, та взаємодіє з БД для збереження та отримання даних.
- Payment System надсилає запити до TRON Blockchain для здійснення транзакцій та повертає дані назад до Back-end Server.
- Database виконує операції CRUD у відповідь на запити від Back-end Server.

Дана архітектура забезпечує чітке розподілення задач між клієнтськими та серверними частинами відповідно забезпечує надійність і масштабованість системи РБ.

2.2. Блок-схеми опису алгоритмів

Для організації функціональності системи нашого рекламного бота створено декілька основних алгоритмів, котрі включають у себе різні аспекти взаємодії юзерів із системою, а також обробку транзакцій. Наведемо блок-схеми та описи основних алгоритмів:

1) Алгоритм розміщення рекламни – процес розміщення оголошення , протягом якого власник каналу отримує заявку на розміщення цього оголошення та підтверджує або відхиляє це розміщення. Якщо підтверджує, то оголошення буде розміщено на цьому каналі та статус заяви буде в свою чергу оновлюватись. Якщо власник відхилить то програмне рішення повідомить рекламодавця про те що власник відхилив оголошення. Це запорука гнучкої системи, що зображено на рисунку 2.2.

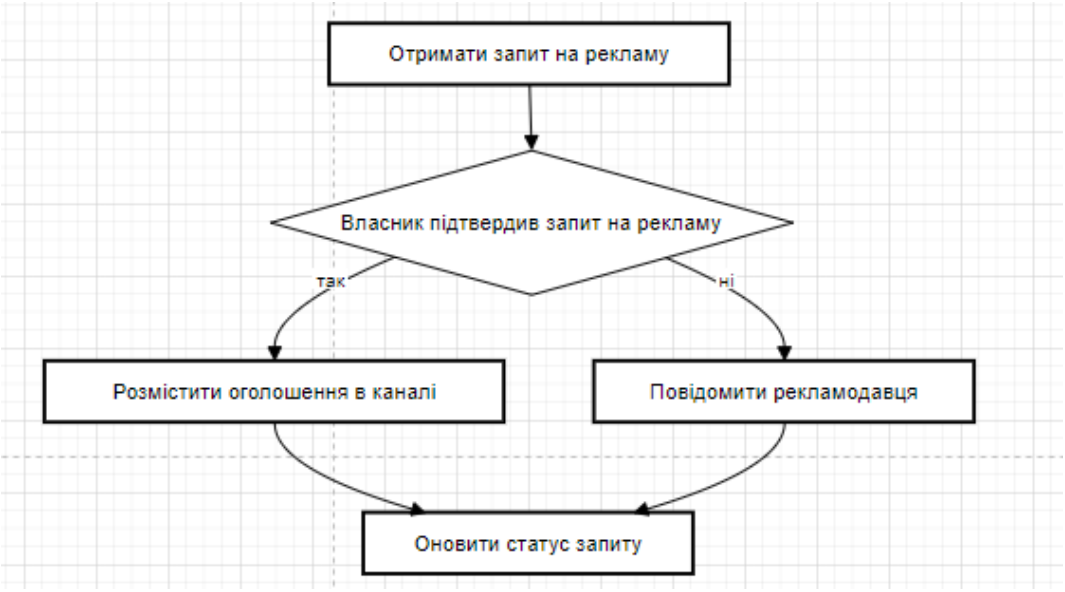

Рис. 2.2. Алгоритм розміщення рекламного оголошення

2) Алгоритм поповнення балансу – процес поповнення балансу користувача, котрий зображений на рисунку 2.3. Згідно даного алгоритму користувач подає запит на поповнення через веб-інтерфейс відповідно отримує реквізити для оплати, здійснюючи платіж, після чого транзакція обробляється та баланс користувача оновлюється.

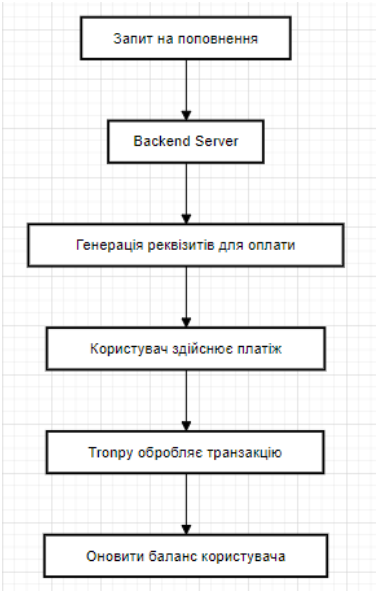

Рис. 2.3. Алгоритм поповнення балансу

3) Алгоритм виведення коштів – процес виведення коштів користувачем, при якому він подає запит на виведення, відповідно система

перевіряє баланс та генерує транзакцію, котра обробляється платіжною системою та зразу після цього баланс користувача оновлюється. Зображено нижче на рисунку. 2.4.

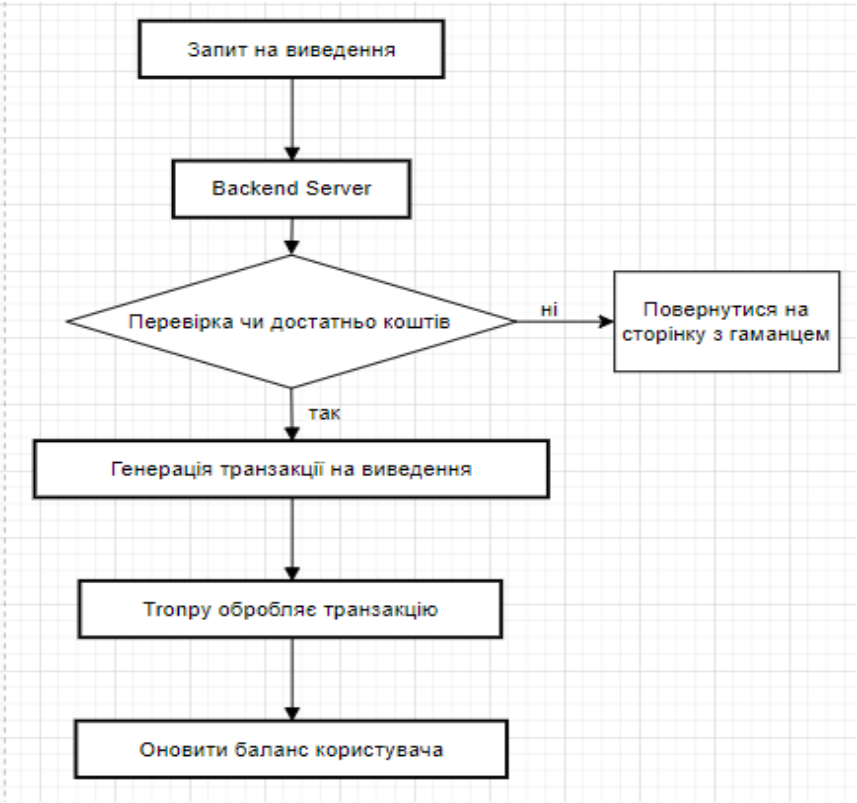

Рис. 2.4. Алгоритм виведення коштів

2.3. Діаграма функціональності систем

Створено use-case діаграму, в котрій демонструється взаємодія між акторами та системою, графічно описуючи різні сценарії використання розробленої мною системи, а саме які функції системи доступні різним типам юзерів.

Зображено на рисунку 2.5. три актора: рекламодавець, власник TG каналу/групи і адміністратор. Відмінність між рекламодавцем та власником каналу/групи полягає в тому, що рекламодавець має змогу створювати рекламні заявки, а власник каналу –оголошення. Адміністратор – спостерігає за діяльністю системи і чи вона працює стабільно та правильно, і має змогу управляти обліковими записами користувачів.

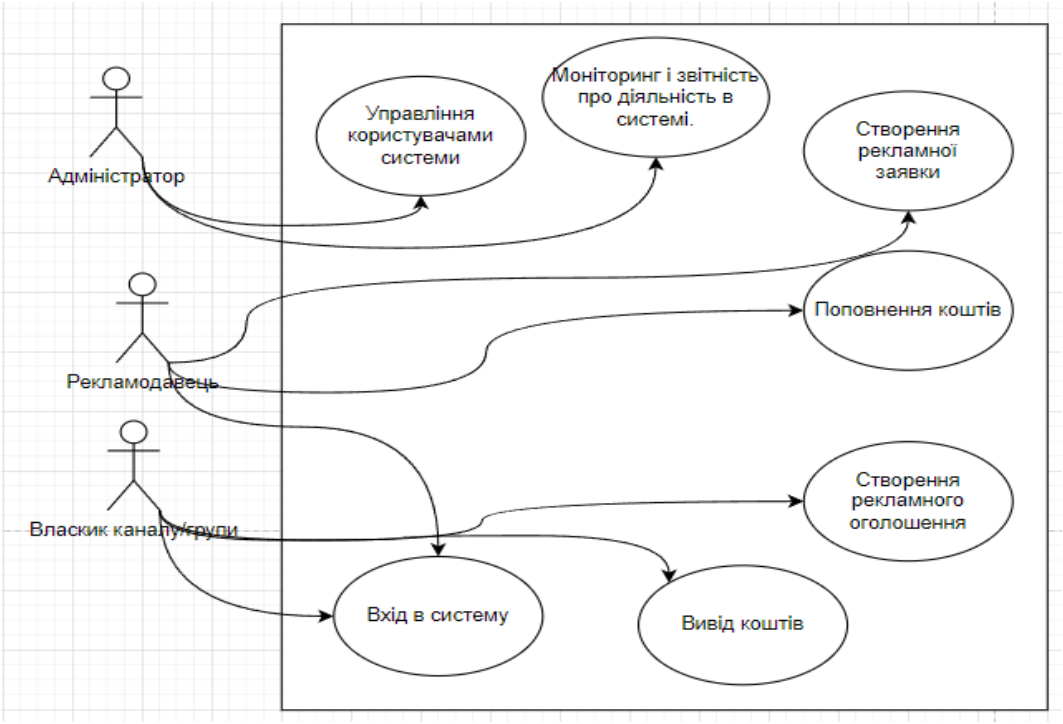

Рис. 2.5. Use-case діаграма рекламного бота

Діаграма послідовності описує взаємозвязок/взаємодію обєктів в певній послідовності. Описує які об'єкти беруть участь у взаємодії і в якій послідовності вони обмінюються повідомленнями.

Діграми послідовності використовуються щоб візуалізувати і зрозуміти яким чином різні частини системи взаємодіють одна з одною в конкретних сценаріях, що допомагає краще зрозуміти логіку процесів та забезпечити правильність реалізації програмного забезпечення та надати йому гнучкості.

Діаграма послідовності розділена на частини:

- Авторизація
- Створення рекламної заявки
- Створення рекламного оголошення і виконання рекламної угоди.
- Поповнення
- Вивід
- Платіжна система (поповнення та вивід)

Одною з основних елементів нашої системи є авторихація тому, що юзеру немає потреби проходити додаткової реєстрації, оскільки користувач ідентифікується по TG ID.

Користувач проходить реєстрацію в системі через команду “/start” – TG бот отримує Update від TG, де містяться, дані користувача, такі як: id, username, first_name, last_name та інші, після ці дані відправляються через бот до back-end системи, де обробляються і зберігаються в БД.

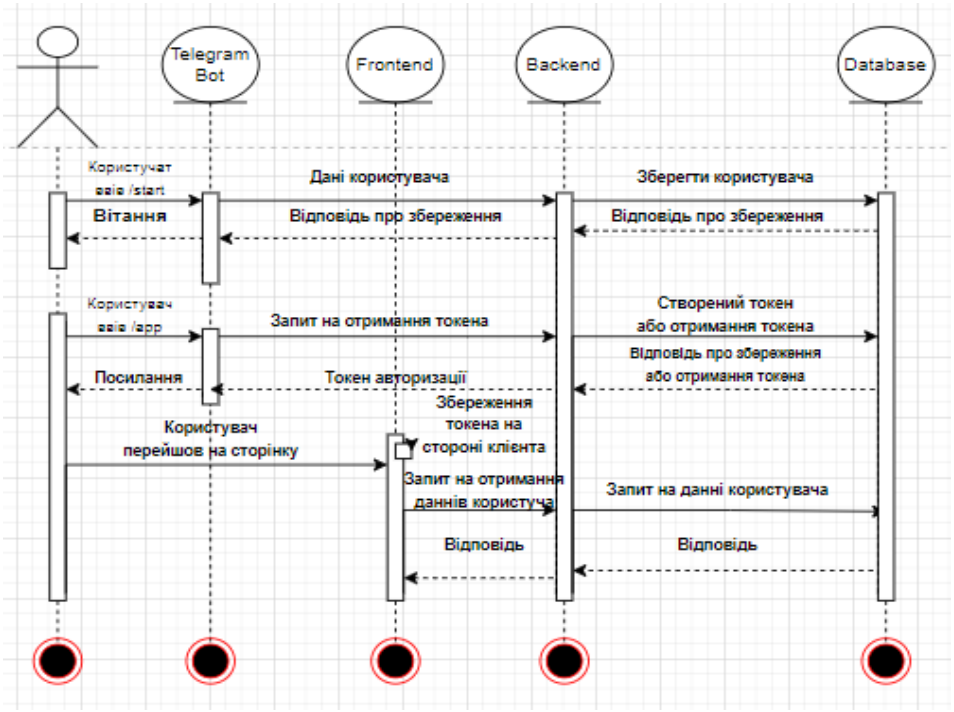

Рис. 2.6. Діаграма послідовності авторизації користувача.

Коли користувач вже був збережений, відповідно може получити посилання на сайт системи, де зможе використовувати послуги системи. Авторизація користувача на веб частині проходить дуже просто, оскільки ввівши команду «/app», телеграм бот надсилає запит до back-end для отримання токена авторизації – користувачу надається посилання на веб з токеном. Веб провіряє чи токен міститься в посиланні і зберігає його на стороні локально, для подальшого використання.

Діаграма послідовності створення рекламної заявки.

Користувач, який авторизувався, має змогу повноцінно взаємодіяти зі системою і можливість розмістити свою рекламу в обраному каналі/групі. А саме користувач отримує список доступних оголошень для розміщення реклами, вказавши фільтри. [31]

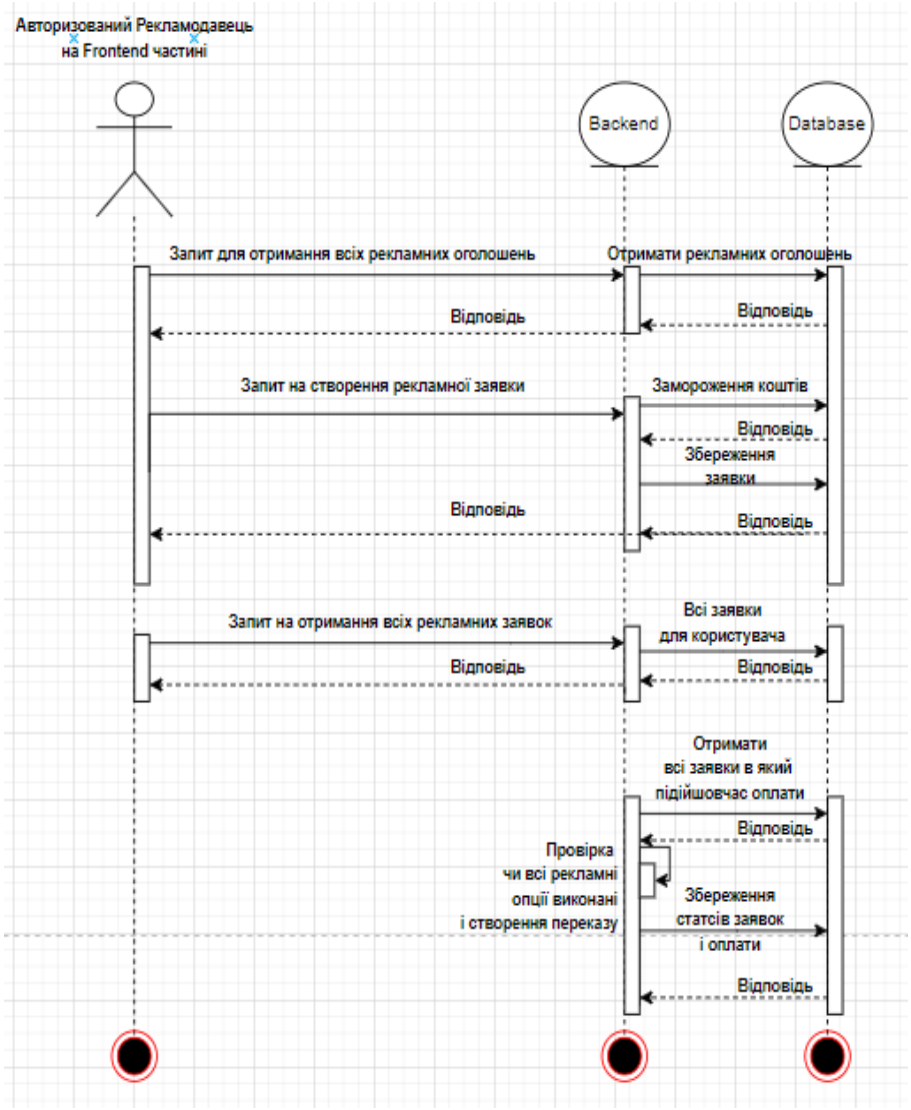

Рис. 2.7. Діаграма послідовності створення рекламної заявки.

Коли юзер отримав список, то відповідно обирає і створює замовлення, в якому він зможе обрати опції, котрі його зацікавили. Потім кошти користувача заморожуються до моменту відмови власника каналу від його заявки, або завершення рекламної угоди. Якщо рекламна угода була заключена, кошти списуються з його гаманця.

Діаграма послідовності створення і виконання рекламної угоди

Діаграма на рисунку 2.8 зображує наступне – авторизований користувач має змогу зайти на сторінку створення оголошення через телеграм канал групи що зєднано з бекендом та базою даних програмного рішення, де воно підтягне всі потрібні дані для девелоперів, а саме таке як: доступні чати/канали/групи, види реклами (пости).

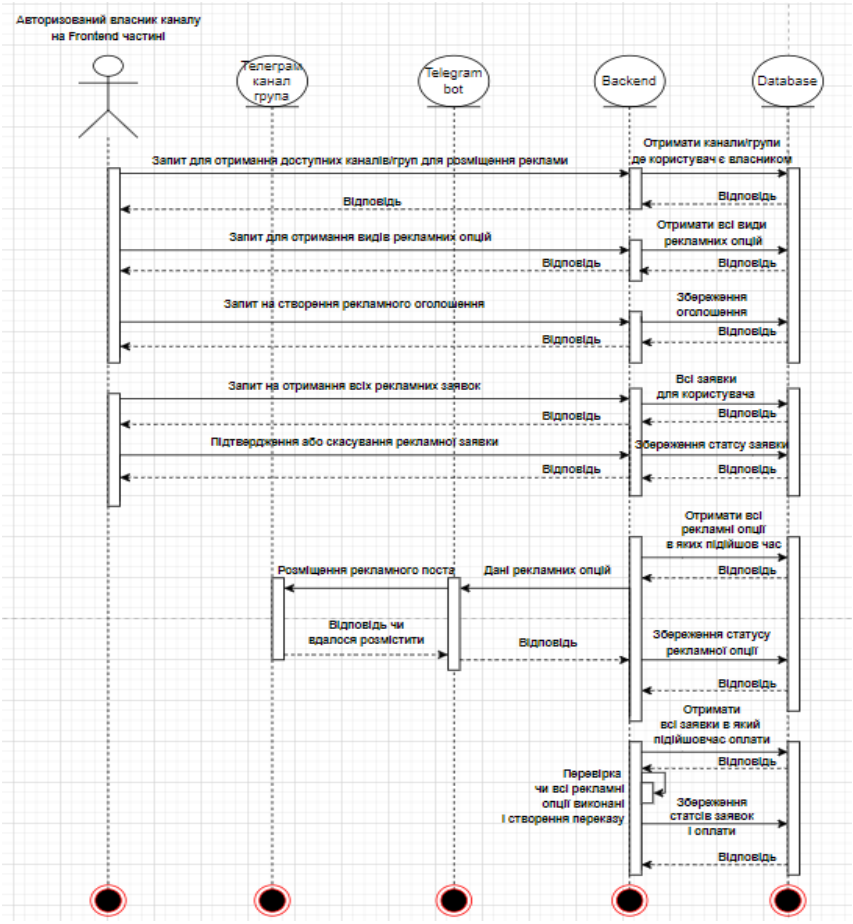

Рис. 2.8. Створення рекламного оголошення і виконання рекламної угоди

Користувач обирає чат, де буде розміщуватися реклама, вид реклами і її ціна. Після того, як користувач розмістив оголошення, йому можуть приходити рекламні заявки, котрі може прийняти або відхилити, якщо йому не підходить вміст реклами. Процес виконання угоди передбачає такі етапи: тримання постів(опцій) в який підійшов час для розміщення, розміщення рекламних постів та перевірка чи всі опції в угоді виконані і виконання оплати

Серверна частина витягує пости для розміщення та відправляє його TG боту, котрий пробує їх розмістити і як відповідь для Backend відправляє дані, який пост він зміг розмістити, а який ні (для прикладу: недостатньо прав в даному чаті), після цього всі стани зберігаються в БД.

Оплата виконується після того, коли усі опції угоди виконані і настав час оплати, котрий зазначається ще на моменті створення угоди. Коли оплата проходить таким чином – рекламодавцю стягується плата і йде поповнення гаманця власника каналу/групи.[2]

Діаграма послідовності поповнення і вивід коштів

Одні із найважливіших елементів даної системи це – поповнення та вивід коштів. На діаграмі поповнення відбувається за певними етапами, а саме:

1) Запит на поповнення – користувач робить запит до back-end для поповнення коштів, яка в свою чергу створює запит до платіжної системи.

2) Отримання адресу гаманця – створивши запит до платіжної системи, платіжна система створює заявку і відправляє її назад, для перевірки, після чого backend робить перевірку і відправляє підтвердження заявки і у відповідь отримує адрес гаманця.

3) Здійснення оплати – користувач після отримання адреси гаманця, має здійснити оплату.

4) Нарахування коштів – back-end очікує callback, запит про стан оплати і нараховує користувачу кошти, якщо оплата пройшла успішно.

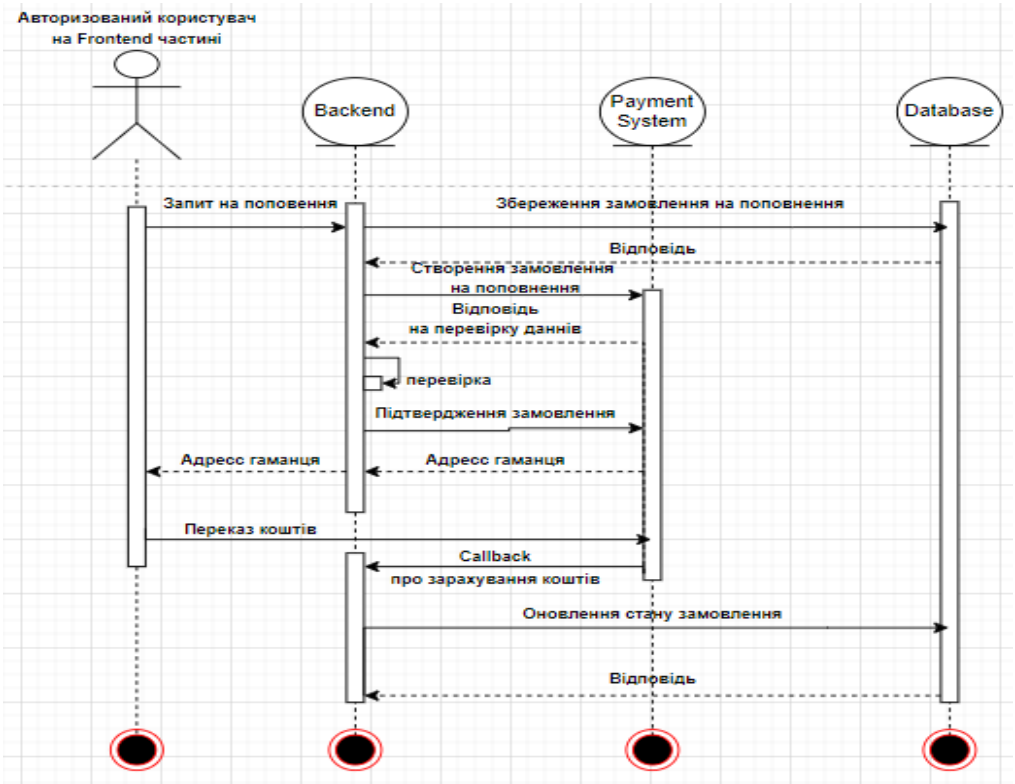

Рис. 2.9. Діаграма послідовності поповнення балансу

Дані етапи надають змогу користувачу швидко, легко та безпечно виконати поповнення балансу в рекламний бот, що дозволить в подальшому купляти рекламу.

А тепер приступимо до виведення коштів зароблених за допомогою реклами. Даний процес має також власні етапи, а саме: створення запиту на виплату (відповідно перейшовши у власни гаманець рекламного бота є можливість відправити запит на вивід коштів, вказавши реквізити), здійснення виплати (back-end отримавши запит на вивід – створює заявку і надсилає її на платіжну систему, підтверджує і очікує колбек, про стан виплати), отримання інформації про виплату(back-end отримав колбек про стан виплати та зберігає в базі даних).

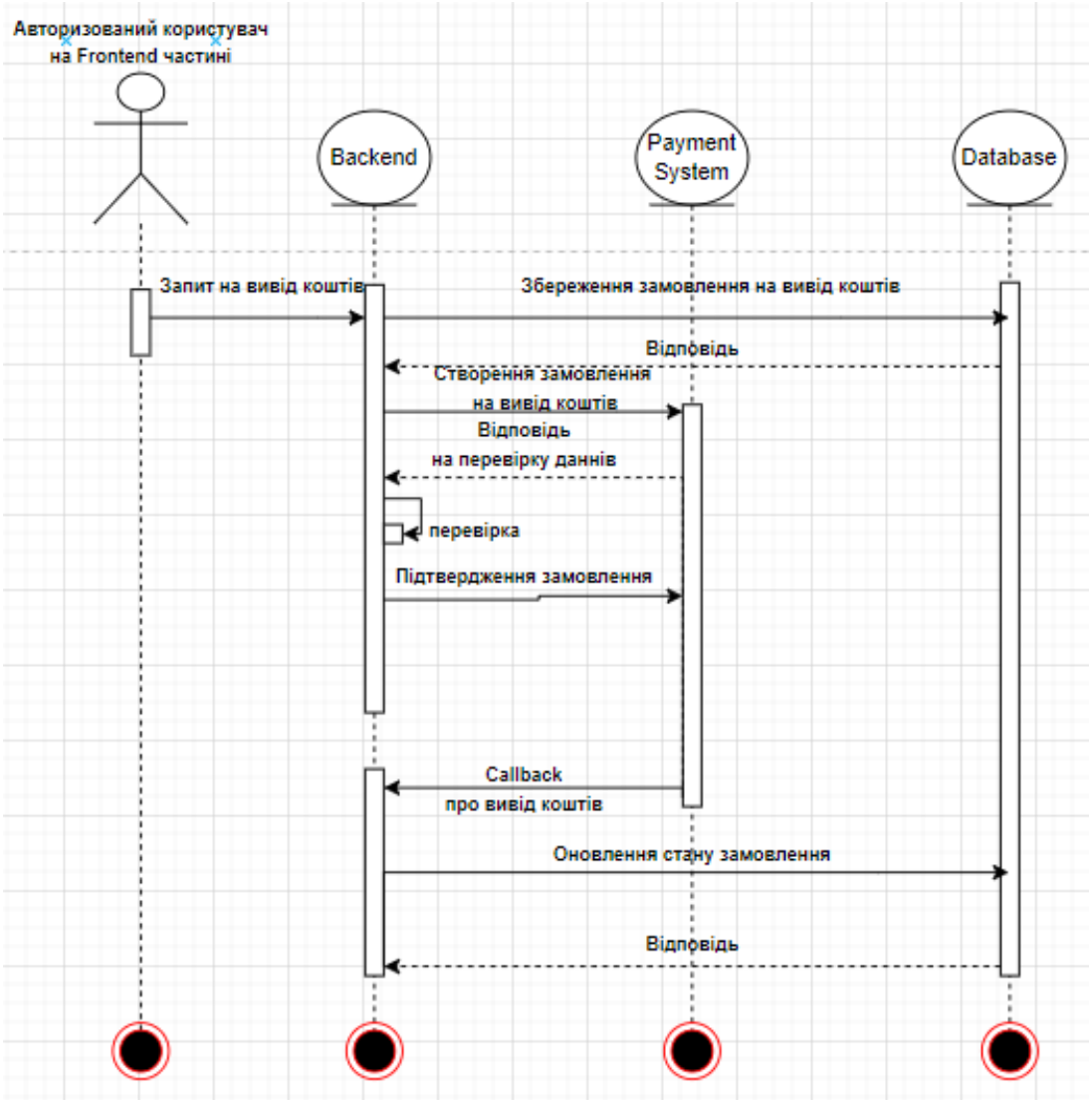

Рис. 2.10. Діаграма послідовності виводу коштів

У висновку поповнення та вивід являються досить простими для розуміння, а враховуючи факт того, що платіжна система реалізована через

блокчейн TROM, то це надасть безпеку користувачам від замороження коштів, що може бути при використанні платіжних систем поширеного характеру.

Діаграма послідовності платіжної системи: поповнення і вивід коштів

У платіжній системі користувач, тобто мерчант – сервіс, тому від повинен отримувати зручний API для взаємодії з платіжною системою, у нашому варіанті – крипто-платіжною системою.

Щоб поповнити баланс на рисинку 2.11. Мерчанту потрібно створити заявку на поповнення, котру отримує платіжна система та відповідно добавляє в базу даних, якщо усі дані введено правильно, то відправляє у відповідь деталі для перевірки мерчатну. Мерчант має перевірити чи всі дані валідні зі створеною заявкою, котру він відправив і надіслати підтвердження для реалізації цієї заявки, після того мерчант отримує реквізити для внесення депозиту.

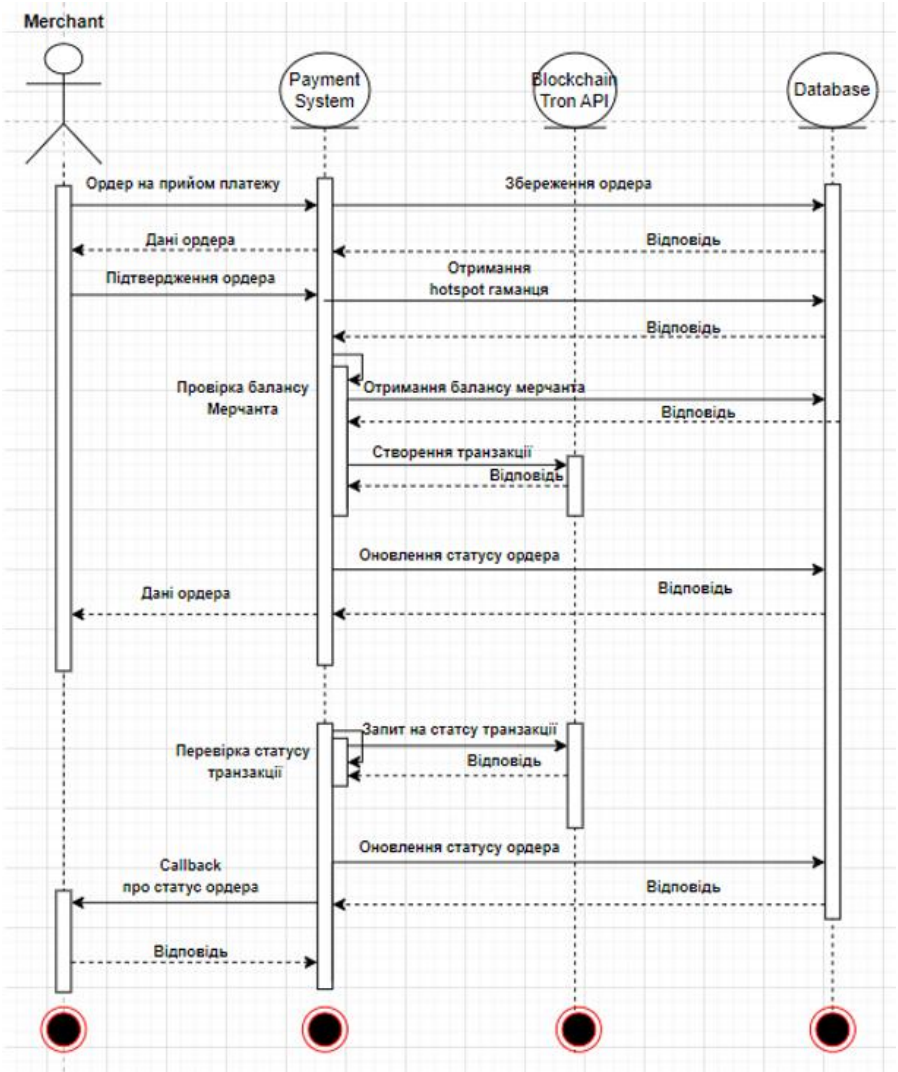

Рис. 2.11. Діаграма послідовності поповнення в платіжній системі

Після внесення оплати – платіжна система перевіряє чи прийшли кошти за реквізитами, якщо так – відправляє колбек зі статусом заявки, а якщо ні – очікує певний час, або скасовує заявку і надсилає колбек з неуспішним статусом заявки.

Як видно на діаграмі на рисунку 2.11. Вивід працює за подібною схемою, тільки добавляється частина зі збереженням транзакції та обирається головний гаманець. А А саме після створення та підтвердження заявки на виведення коштів – платіжна система створює відправлення в блокчейні та зберігає ідентифікатор транзакції, котрий буде провіряти чи платіж здійснено успішно, чи ні.

2.4. База даних PostgreSQL

PostgreSQL - реляційна система керування базами даних або СУБД , котру було обрано за такі плюси як: надійність, багаті можливості та активну спільноту розробників. Дана СУБД безкоштовною для використання, модифікації та розповсюдження та має ліцензію PostgreSQL License, котра має подібність до ліцензії MIT. При виборі також було важдиво, що PostgreSQL забезпечує повну підтримку ACID, оскільки це гарантія надійності транзакцій та збереження даних. Змога додавати нові типи даних, індекси, функції та мови процедур, котрі роблять систему гнучкою для моїх потреб. Вона підтримує всі стандартні SQL можливості і також багато розширень, включаючи складні запити, підзапити, перегляди, тригери та збережені процедури.

Дана СУБД пропонує синхронну та асинхронну реплікацію – дозволяє масштабувати БД для великих навантажень. А також включає розширені функції безпеки, як аутентифікація на базі ролей, SSL для реалізації HTTPS та контроль доступу на рівні рядків. Використання PostgreSQL дуже зручне, бо можна легко інтегруватися з різними інструментами та системами, включно з різними мовами Python, Java та інші, а також з іншими базами даних. Спільнота розробників та юзерів, котрі регулярно випускають оновлення та нові функції[10].

1) Для платіжної системи на базі TRON

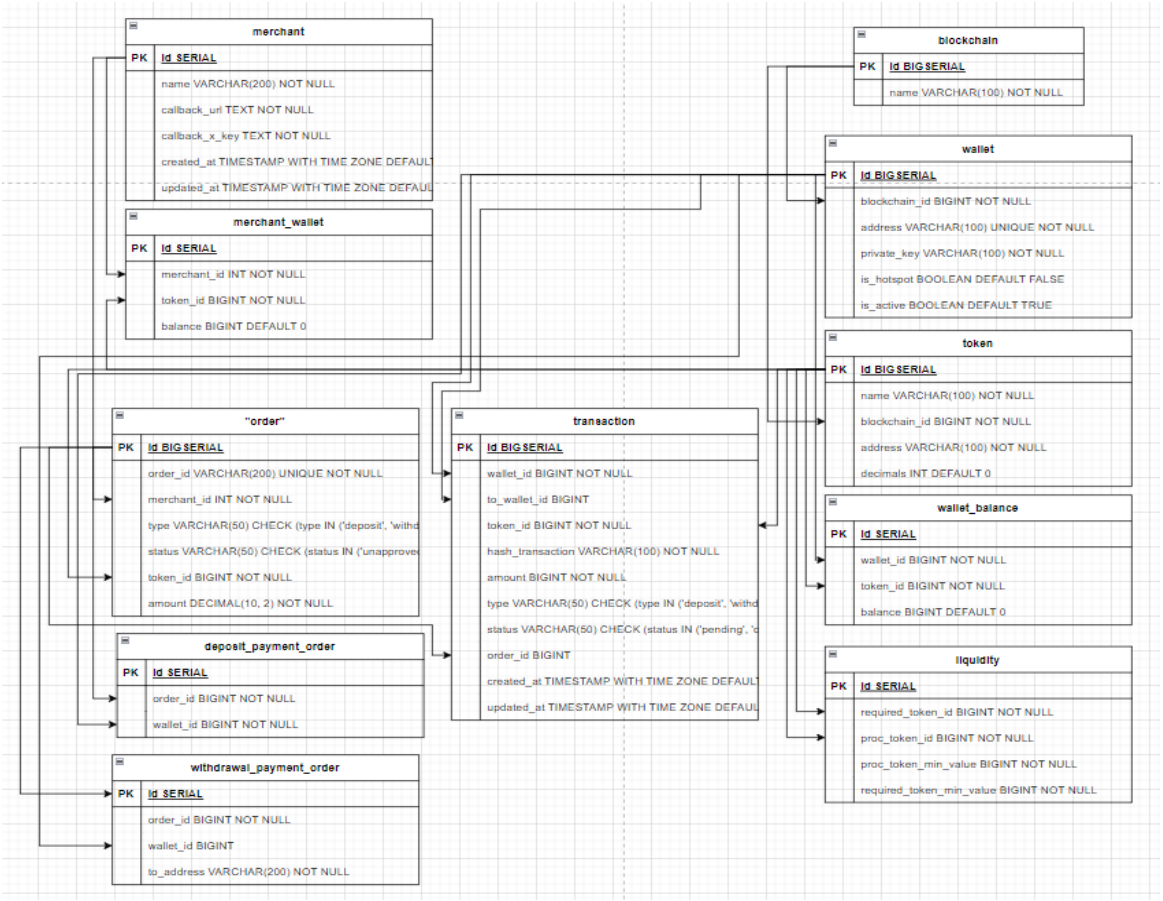

Рис. 2.12. Структура база даних для платіжної системи на блокчейні

У даній БД доступні такі таблиці й відповідно залежні поля:

Merchant - сервіс, або користувач, який буде взаємодіяти з нашою платіжною системою:

- id: унікальний ідентифікатор купця (PK).
- name: ім'я купця.
- callback_url: URL для зворотнього виклику.
- callback_x_key: ключ для зворотнього виклику.
- created_at: час створення запису.
- updated_at: час оновлення запису.

Merchant_wallet – баланс мерчанта:

- id: унікальний ідентифікатор (PK).
- merchant_id: ідентифікатор купця (FK, посилається на merchant).
- token_id: ідентифікатор токена (FK, посилається на token).
- balance: баланс.

Order – заявки на вивід, поповнення:

- id: унікальний ідентифікатор замовлення (PK).
- order_id: унікальний ідентифікатор замовлення(вказаний мерчантом).
- merchant_id: ідентифікатор купця (FK, посилається на merchant).
- type: тип замовлення (deposit, withdraw).
- status: статус замовлення (unapproved, approved).
- token_id: ідентифікатор токена (FK, посилається на token).
- amount: сума замовлення.

Deposit_payment_order – реквізити для поповнення:

- id: унікальний ідентифікатор (PK).
- order_id: ідентифікатор замовлення (FK, посилається на order).
- wallet_id: ідентифікатор гаманця для поповнення (FK, посилається на wallet).

Withdrawal_payment_order – реквізити для виведу:

- id: унікальний ідентифікатор (PK).
- order_id: ідентифікатор замовлення (FK, посилається на order).
- wallet_id: ідентифікатор гаманця (FK, посилається на wallet).
- to_address: адреса для виведення.

Transaction – транзакції, які здійснюються в блокчейні:

- id: унікальний ідентифікатор транзакції (PK).
- wallet_id: ідентифікатор гаманця (FK, посилається на wallet).
- to_wallet_id: ідентифікатор гаманця одержувача(FK, посилається на wallet).
- hash_transaction: хеш транзакції.
- amount: сума транзакції.
- type: тип транзакції (deposit, withdraw, transfer_in, transfer_out).
- status: статус транзакції (pending, completed, rejected).
- order_id: ідентифікатор замовлення (FK, посилається на order).
- created_at: час створення транзакції.
- updated_at: час оновлення транзакції.

Blockchain – блокчейни в який будуть здійснюватися операції:

- id: унікальний ідентифікатор блокчейну (PK).
- name: ім'я блокчейну.

Wallet – гаманці платіжної системи:

- id: унікальний ідентифікатор гаманця (PK).
- blockchain_id: ідентифікатор блокчейну (FK, посилається на blockchain).
- address: унікальна адреса гаманця.
- private_key: приватний ключ гаманця.
- is_hotspot: чи є гаманець головним(для виводу).
- is_active: чи активний гаманець(чи використовується для здійснення операції в блокчейні).

Token – валюти в блокчейні:

- id: унікальний ідентифікатор токена (PK).
- name: ім'я токена.
- blockchain_id: ідентифікатор блокчейну (FK, посилається на blockchain).
- address: адреса токена.
- decimals: кількість десяткових розрядів.

Wallet_balance – баланс гаманця:

- id: унікальний ідентифікатор (PK).
- wallet_id: ідентифікатор гаманця (FK, посилається на wallet).
- token_id: ідентифікатор токена (FK, посилається на token).
- balance: баланс токенів.

Liquidity – від якої суми токени з другорядного гаманця переводяться в головний, і скільки потрібно базової валюти для покриття комісії:

- id: унікальний ідентифікатор (PK).
- required_token_id: ідентифікатор токена забезпечення (FK, посилається на token).
- proc_token_id: ідентифікатор токена (FK, посилається на token).
- proc_token_min_value: мінімальна кількість токена для переказу його з другорядного гаманця на головний(hotspot).

- required_token_min_value: кількість токена забезпечення для покривання комісій блокчейна.

2) Схема БД для сервісу рекламного боту.

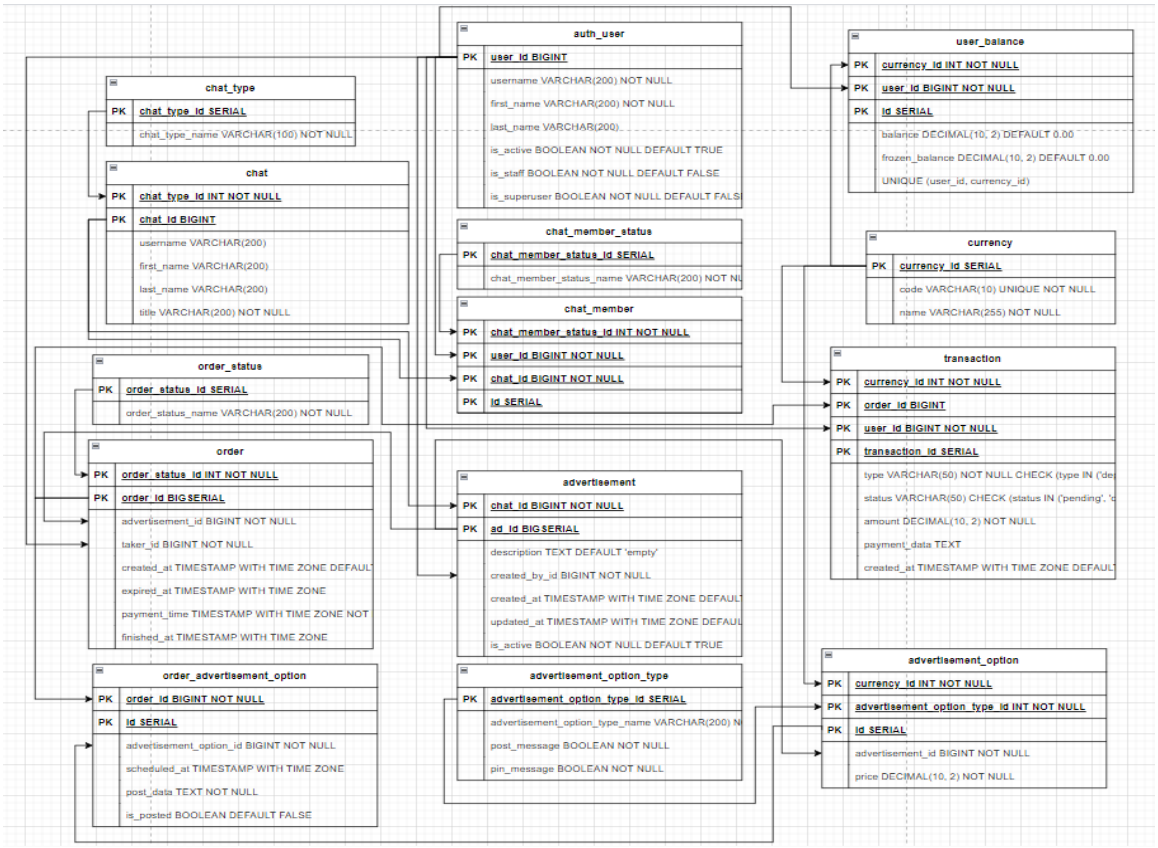

Рис. 2.13. Структура бази даних для сервісу рекламного бота

У БД зазначеній на вище наявні такі таблиці та поля:

Auth_user – телеграм користувачі:

- user_id: унікальний ідентифікатор користувача (PK).
- username: ім'я користувача.
- first_name: ім'я користувача.
- last_name: прізвище користувача.
- is_active: статус активності користувача.
- is_superuser: статус суперкористувача.

User_balance – баланс користувача:

- id: унікальний ідентифікатор (PK).
- currency_id: ідентифікатор валюти (FK, посилається на currency).
- user_id: ідентифікатор користувача (FK, посилається на auth_user).

- balance: баланс.
- frozen_balance: заморожений баланс.

Currency – вид валюти:

- id: унікальний ідентифікатор валюти (PK).
- code: код валюти.
- name: ім'я валюти.

Transaction – операції з коштами:

- id: унікальний ідентифікатор транзакції (PK).
- currency_id: ідентифікатор валюти (FK, посилається на currency).
- order_id: ідентифікатор замовлення (FK, посилається на order).
- user_id: ідентифікатор користувача (FK, посилається на auth_user).
- type: тип транзакції (deposit, withdraw, freeze, unfreeze).
- status: статус транзакції (pending, completed).
- amount: сума транзакції.
- payment_data: дані про оплату.
- created_at: час створення транзакції.

Chat_type – тип чата(група, канал):

- id: унікальний ідентифікатор типу чату (PK).
- chat_type_name: ім'я типу чату.

Chat - чат:

- id: унікальний ідентифікатор чату (PK).
- username: ім'я користувача.
- first_name: ім'я користувача.
- last_name: прізвище користувача.
- type: тип чату (FK, посилається на chat_type).

Chat_member_status – статуси учасників чата:

- id: унікальний ідентифікатор статусу учасника чату (PK).
- chat_member_status_name: ім'я статусу учасника чату.

Chat_member – учасник певного чата:

- id: унікальний ідентифікатор (PK).
- chat_id: ідентифікатор чату (FK, посилається на chat).

- user_id: ідентифікатор користувача (FK, посилається на auth_user).
- chat_member_status_id: ідентифікатор статусу учасника чату (FK, посилається на chat_member_status).

Order_status – статус рекламної заявки:

- id: унікальний ідентифікатор статусу замовлення (PK).
- order_status_name: ім'я статусу замовлення.

Order – рекламна заявка:

- id: унікальний ідентифікатор замовлення (PK).
- advertisement_id: ідентифікатор оголошення (FK, посилається на advertisement).
- status_id: ідентифікатор статусу замовлення (FK, посилається на order_status).
- created_at: час створення замовлення.
- expired_at: час закінчення замовлення.
- payment_time: час оплати.
- finished_at: час завершення замовлення.

Advertisement:

- id: унікальний ідентифікатор оголошення (PK).
- chat_id: ідентифікатор чату (FK, посилається на chat).
- created_by: ідентифікатор користувача, який створив оголошення(FK, посилається на auth_user).
- description: опис оголошення.
- created_at: час створення оголошення.
- updated_at: час оновлення оголошення.
- payment_window_minutes: тривалість платіжного вікна в хвилинах.
- is_active: статус активності оголошення.

Advertisement_option_type – типи рекламних опцій:

- id: унікальний ідентифікатор типу опції оголошення (PK).
- advertisement_option_type_name: ім'я типу опції оголошення.
- post_message: чи дозволено публікацію повідомлень.
- pin_message: чи дозволено закріплення повідомлень.

Advertisement_option – вид рекламної опції:

- id: унікальний ідентифікатор опції оголошення (PK).
- advertisement_id: ідентифікатор оголошення (FK, посилається на advertisement).
- advertisement_option_type_id: ідентифікатор типу опції оголошення (FK, посилається на advertisement_option_type).
- currency_id: ідентифікатор валюти (FK, посилається на currency).
- price: ціна.

Order_advertisement_option – вибрані опції в рекламній заявці:

- id: унікальний ідентифікатор (PK).
- order_id: ідентифікатор замовлення (FK, посилається на order).
- advertisement_option_id: ідентифікатор опції оголошення (FK, посилається на advertisement_option).
- scheduled_at: запланований час.
- posted_data: дані про публікацію.
- is_posted: статус публікації.

У БД для платіжної системи на блокчейні TRON врахував логіку взаємодії з блокчейном та зберігання заявок на ввід та вивід.

Майже в кожному блойкчейні може бути не один токен, де ми маємо зберігати їх, а ткож зберігати кількість потреби базової валюти TRX, щоб покрити комісію. Така структура дасть змогу ідентифікувати мерчанта, доступний йому баланс створення заявок на ввід та вивід монет, а також ліквідність.

Протягом розробки структури бази даних бота , проаналізовано які сутності юудуть зберігатися в БД і які їм належать. Замовлення зелжить від оголошення рекламного, а відповідно саме оголошення в свою чергу від чату – це дозволить правильно здійснювати оплату після завершення послуги рекламного замовлення тому, що за період виконання замовлення може змінить головний адміністратор чату/каналу/групи.

Для валюти створено окрему таблицю, котра дозволить в майбутньому добавити кілька платіжних способів і надасть гористувачам обширніший вибір.

2.5. Висновки до розділу

Системна конструкція (архітектура) включає зовнішню клієнтську частину, що складається з TG bot та веб клієнту, а також серверних компонентів таких, як: back-end server, БД та платіжна система. проаналізовано взаємозвязок між даними компонентами для за забезпечення хорошої роботи системи. Чітко визначена архітектура дозволяє розділити завдання між клієнтськими та серверними компонентами, що надає ефективість та надійність створеної системи

Написано кілька ключових інструкцій/алгоритмів, що забезпечують основний функціонал нашої системи, а саме: авторизацію користувачів, створення рекламних заявок, розміщення рекламних оголошень, депозит на баланс та виведення коштів. Описав алгоритми використовуючи блок-схеми, котрі позволяють краще вникнути у логіку процесів та надає змогу правильно реалізувати ПЗ.

Створено 2 структури БД для 2-ох систем, а саме для сервісу рекламного бота та пратіжної системи на базі TRON. Стосовно другої – врахував логіку взаємодії з блокчейном і зберігання та ввід/вивід, а також ліквідність токенів. СУБД – PostgreSQL дала змогу реалізувати надійні транзакції та зберігання даних, при тому з можливісь масштабування БД для великих нагрузок.

Отже, структура, алгоритми та інформаційне забезпечення системи РБ забезпечують її надійність, масштабованість та ефективність, надає змогу користувачам швидко та безпечно взаємодіяти з системою.

РОЗДІЛ 3
РОЗРОБКА МОДЕЛЕЙ І МЕТОДІВ ПРОГРАМНОГО РІШЕННЯ

3.1 Загальна структура програмного забезпечення

Щоб змоделювати структуру ПЗ, яку розроблено, використано діаграму компонентів. Дана діаграма дає змогу візуалізувати кожен компонент системи та їх звязки між собою, а це в свою чергу надає можливість кращого розуміння системної архітектури та взаємодії між елементами.

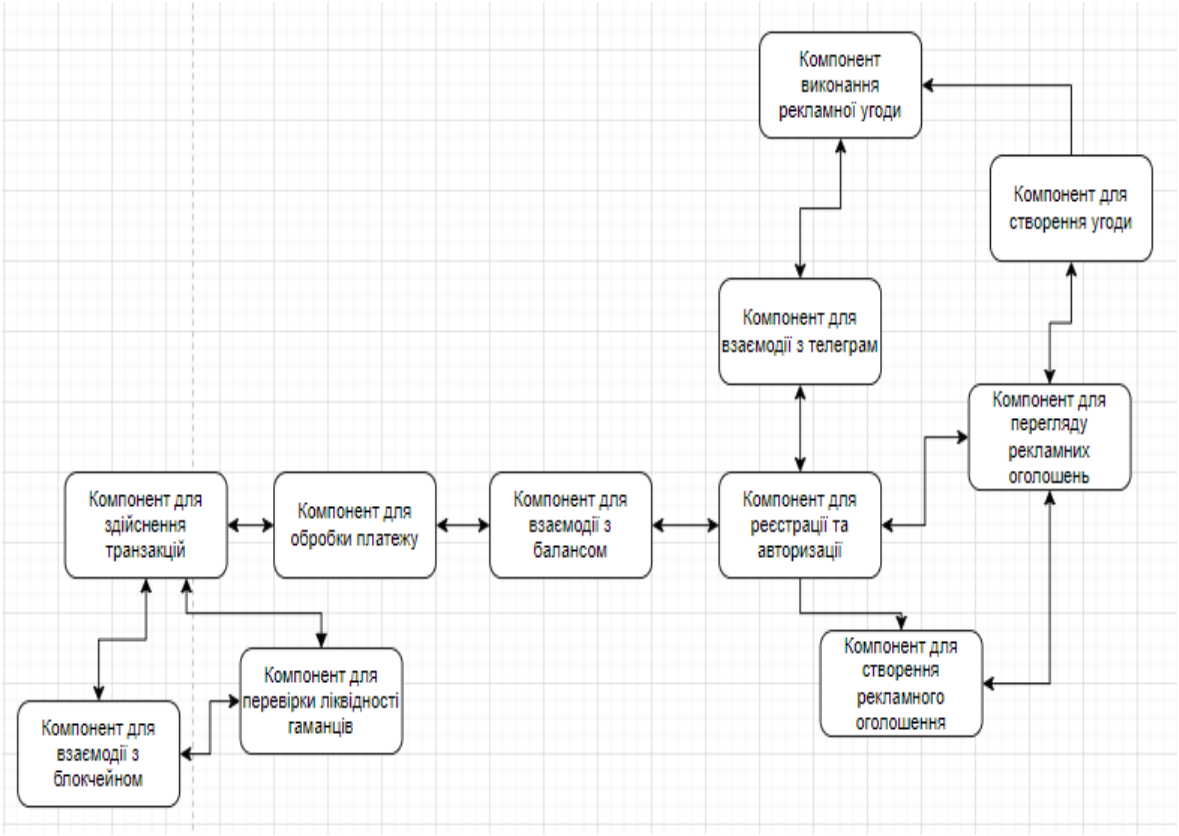

Рис. 3.1. Діаграма компонентів системи

В діаграмі зображеній вище, загалом присутньо 12 компонентів, а саме:

1) Взаємодії з TG;
2) Реєстрації та авторизації;
3) Створення рекламного оголошення;
4) Створення угоди;
5) Виконання рекламної угоди;

6) Взаємодії з балансом;
7) Обробки платежу;
8) Здійснення транзакцій;
9) Перевірки ліквідності гаманців;
10) Взаємодії з блокчейном;
11) Здійснення транзакцій;

Для організації взаємодії з TG API використав бібліотеку Aiogram. Дана бібліотека є сильним та зручним інструментом для розробки TG ботів. Надає асинхронний інтерфейс, котрий дає змогу обробляти неймовірну кількість запитів водночас, відповідно забезпечуючи високу продуктивність.

Tronpy – бібліотека для створення компоненту обробки платежів і здійснення транзакцій на базі блокчейну TRON. Надає зручний інтерфейс для взаємодії з блокченом. Що дозволить здійснювати транзакції для поповнення і виводу коштів зі системи.

Відносно реалізації інших компонентів використовув HTTPS запити, котрі нам допоміг Django і Django Rest Framework, що надають велику кількість готових рішень і полегшує розробку back-end системи[9].

3.2. Діаграма класів

Для створення діаграми класу яку використано для побудови програмного рішення було використано бібліотеку для Django – Django extension тому, що це потужний фреймворк, котрий в свою чергу надає з коробки всі потрібні медоти, в даних діаграмах немає методів що робить їх більш гнучкішими тому, що усі класи мають наслідування від самого батьківського класу model.Model. В ньому є такі класи як BlockChain, WalletFinanct, Wallet, Token, Transaction, merchWallet, Order, Merchant, DepositPaymentOrder. Вони написані за допомогою методології SOLID всі класи мають єдину відповідальність, відкритість-закритість, принцип лісков, розділення інтерфейсу та інверсія залежностей.

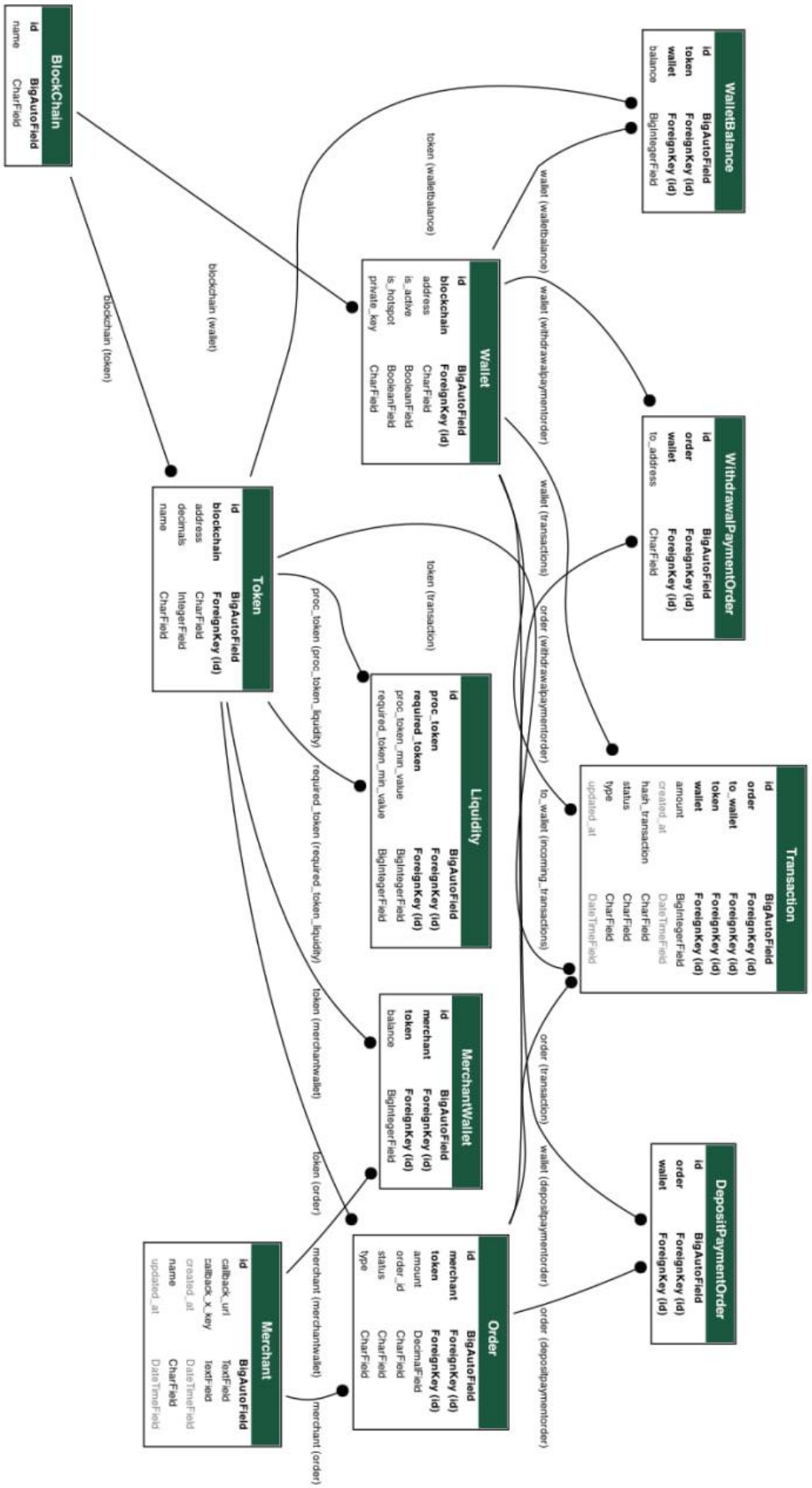

Рис. 3.2. Діаграма класів для платіжної системи

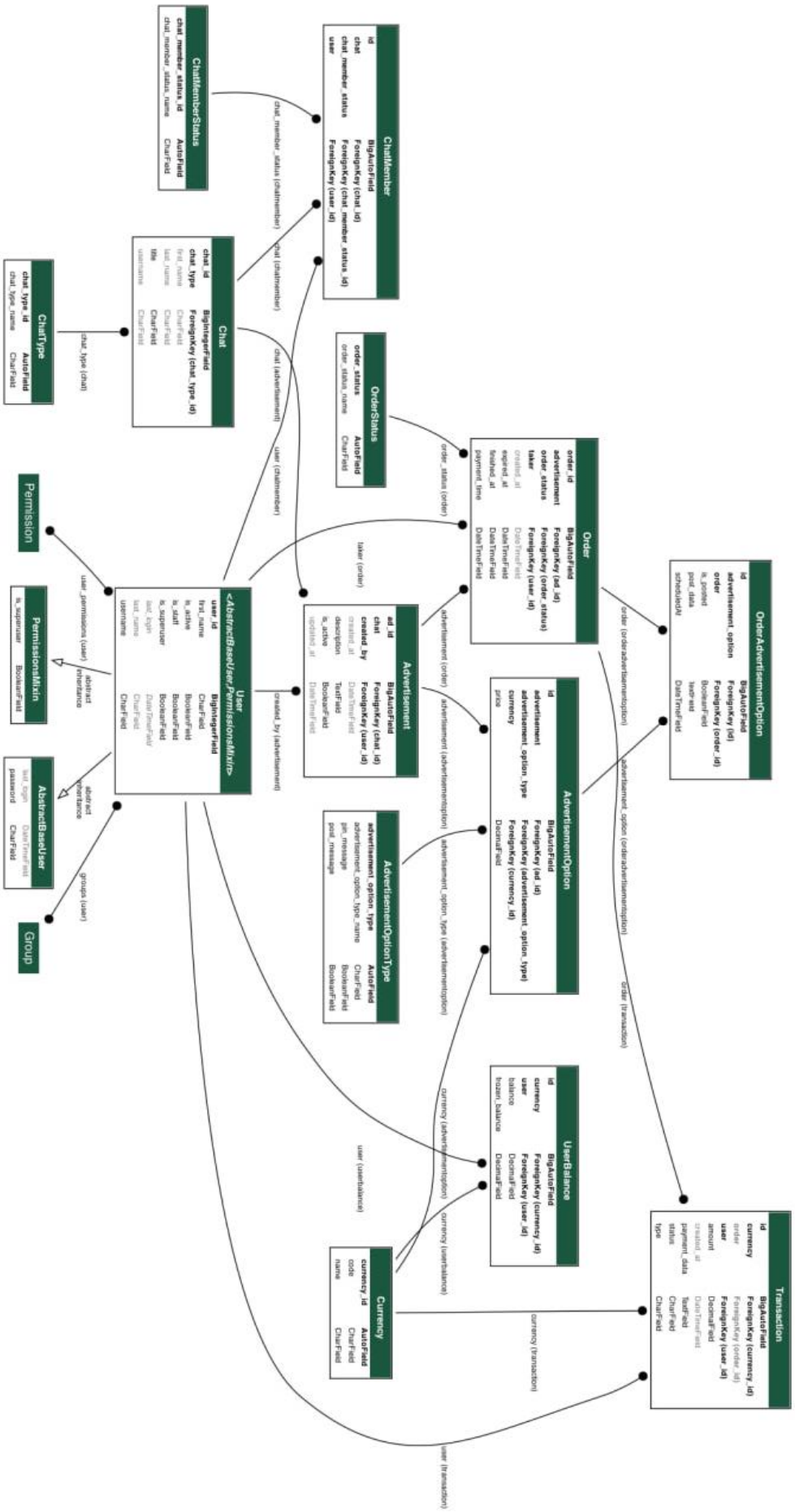

Рис. 3.3. Діаграма класів для рекламного бота

В діаграмі класі платіжної системи містяться наступні класи: Блокчейн, Токен, Мерчант, Баланс Мерчанта, Гаманець, Баланс Гаманця, Заявка, Транзакція.

У рекламного бота в діаграмі класів містяться такі класи: Користувач, Чат, Тип чата, Користувач Чата, Тип користувача чата, Валюта, Баланс Користувача, Рекламне Оголошення, Опції Рекламного Оголошення, Типи Опцій, Угоди, Статус Угоди, Транзакції, Опції Угоди.

Усі ці класи наслідують від `Model`, тому давайте розглянемо за що кожен клас відповідає.

Що може робити клас Model

Model є базовим класом, котрий надає основні функціональні можливості для інших класів. Включає наступні методи та функції:

1) Створення нових записів в базі даних:

- Метод create() – надає можливість створювати нові записи в базі даних. Приклад використання: Model.create(data).

2) Читання даних:

- Метод get() – отримування одиного запису з БД за вказаним критерієм. Приклад використання: Model.get(id=3).
- Метод all() – дозволяє отримати всі записи з бази даних для даного класу. Приклад використання: Model.all().

3) Оновлення даних:

- Метод update() – дає можливість оновлювати існуючі записи в базі даних. Приклад використання: Model.upda - te(id=3, data).

4) Видалення даних:

- Метод delete() – дає змогу видаляти записи з бази даних.

Приклад використання: Model.delete(id=3).

5) Видалення даних:

- Метод delete() – дає можливіть видаляти записи з бази даних.

Приклад використання: Model.delete(id=3).

6) Фільтрація даних по вказаним полям:

- Метод filter() – дає можливість фільтрувати записи на основі заданих критеріїв. Приклад використання: Model.filter(field=value).

7) Отримання всіх екземплярів класів, в яких міститься зовнішній ключ вказаного класу:

- Метод related() – дає змогу отримувати усі записи, котрі мають зовнішній ключ, що посилається на конкретний запис іншого класу. Приклад використання: Model.related(foreign_key=id).

8) Збереження даних:

- Метод save() - збереження зміни до запису в базі даних. Використовується після зміни властивостей об'єкта. Приклад використання: instance.save().

9) Валідація даних:

- Метод validate() – перевірка на коректність даних перед їх збереженням у базі даних.

10) Транзакційна підтримка:

- В цей перелік вхоять так методи: для початку, коміту та відкату транзакцій(дозволяє забезпечити цілісність даних при виконанні кількох взаємопов'язаних операцій).

3.3. Використані сторонні бібліотеки та модулі

Для розробки веб клієнтської частини використав сторонні модулі:

React (побудова користувацького інтерфейсу), Bootstrap (стилізації компонентів front-end) та Axios (виконання HTTP-запитів до back-end API)[17].

Щоб розробити серверну частину, а саме платіжної системи та рекламного бота використовував сторонні модулі: Django (головний веб-фреймворк для розробки back-end), Django REST framework (для створення Web API) , tronpy (для взаємодії з блокчейном TRON), django-cors-headers (Django-додаток для обробки заголовків сервера,котрі необхідні для Cross-Origin Resource Sharing).

Для розробки TG клієнтської частини були використані сторонні модулі такі як: aiogram (для розробки TG ботів), pydantic (валідація даних та управління

налаштуваннями), FastAPI (швидкий веб-фреймворк для створення API на Python 3.7+) та Uvicorn (створення веб-сервера ASGI для Python та використовується для запуску додатків FastAPI).

3.4. Розробка та опис програмних модулів

В якості БД обрав PostgreSQL, котра дала змогу зберігати дані і забезпечити їх цілісність і їх взаємозвязок, що дозволило зробити систему безпечнішою. А ще PostgreSQL добре працює взвязці з Django тому, що сам фреймворк зразу надає все потрібне для швидкого з'єднання з БД.

Структура веб-клієнтської частини проекту находить у файлі front на рисунку 3.4.

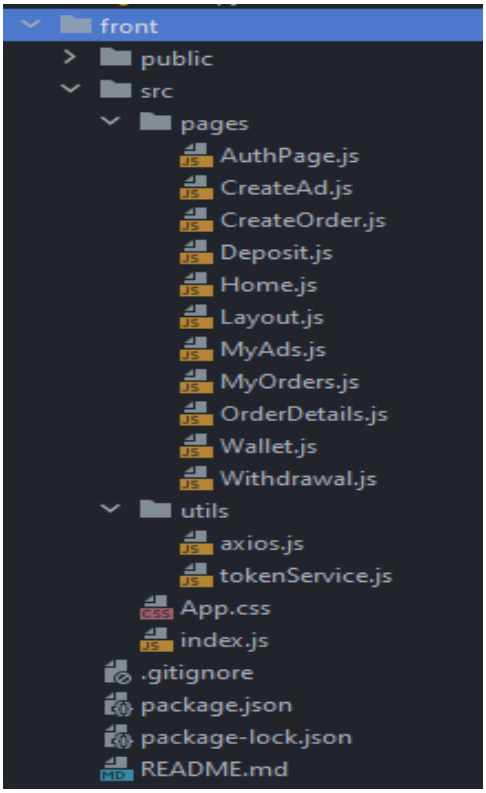

Рис. 3.4. Структура веб-клієнтської частини системи РБ

У папці front основний пакет для розробки клієнтської веб частини є src(source), котрий містить index.js, що виконує роль корневого компоненту, який в свою чергу слугує точкою старту для всього додатку.

Клієнстька частина TG містить декілька основних пакетів та модулів, а саме: main (запускає Телеграм клієнтську частину).

- server (отримує дані від серверної частини, для розміщення рекламних постів).
- loader (створює екземпляри об’єктів, котрі використаються для взаємодії з TG і Серверною частиною рекламного бота).
- utils.api (використовується для взаємодії з рекламного бота для збереження користувача, чата, користувача чата і отримання токена авторизації користувача)
- data.config (використовується для налаштування ключів і посилань) та handlers.start (обробляє дані від TG і використовує
- utils.api для зберігання даних юзерів).

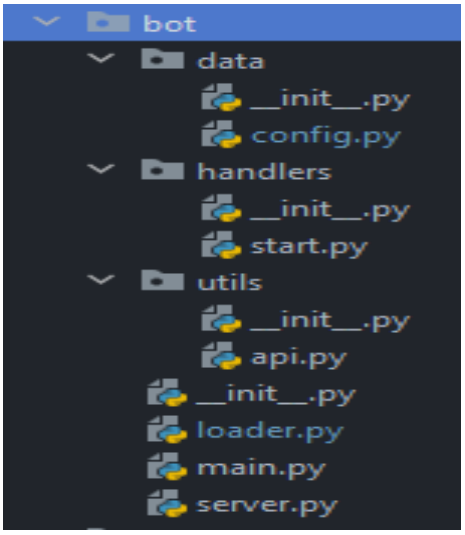

Рис. 3.5. Структура TG клієнтської частини та рекламного бота

Опис основних папок та модулів клієнтської TG частини рекламного боту:

- main – запускає клієнтську частину TG;
- data.config – використовується для налаштування ключів і посилань;
- utils.api – для взаємодії з рекламним ботом для збереження користувача, чата, користувача чата і отримання токена авторизації юзера;
- loader – створює екземпляри об’єктів, котрі будуть використані для взаємодії з TG та Серверною частиною рекламного бота;

- server – отримує дані від серверної частини, щоб розміщувати рекламні пости;
- handlers.start – обробляє дані від TG і використовує utils.api для збереження даних юзерів.

Серверної частини рекламного бота та платіжної системи, котра рахується ядром системи зображено на рисунку 3.6. Основна програма wallet, яка містить стандартні компоненти, такі як моделі, серіалізатори, тести, і views. Присутні файли конфігурації проєкту, включаючи settings.py, urls.py, а також файли для роботи з Celery (celery.py). У структурі також можна побачити файли для управління міграціями бази даних, реалізації API, а також використання зовнішніх сервісів, таких як Tron, що забезпечує функціональність платіжної системи.

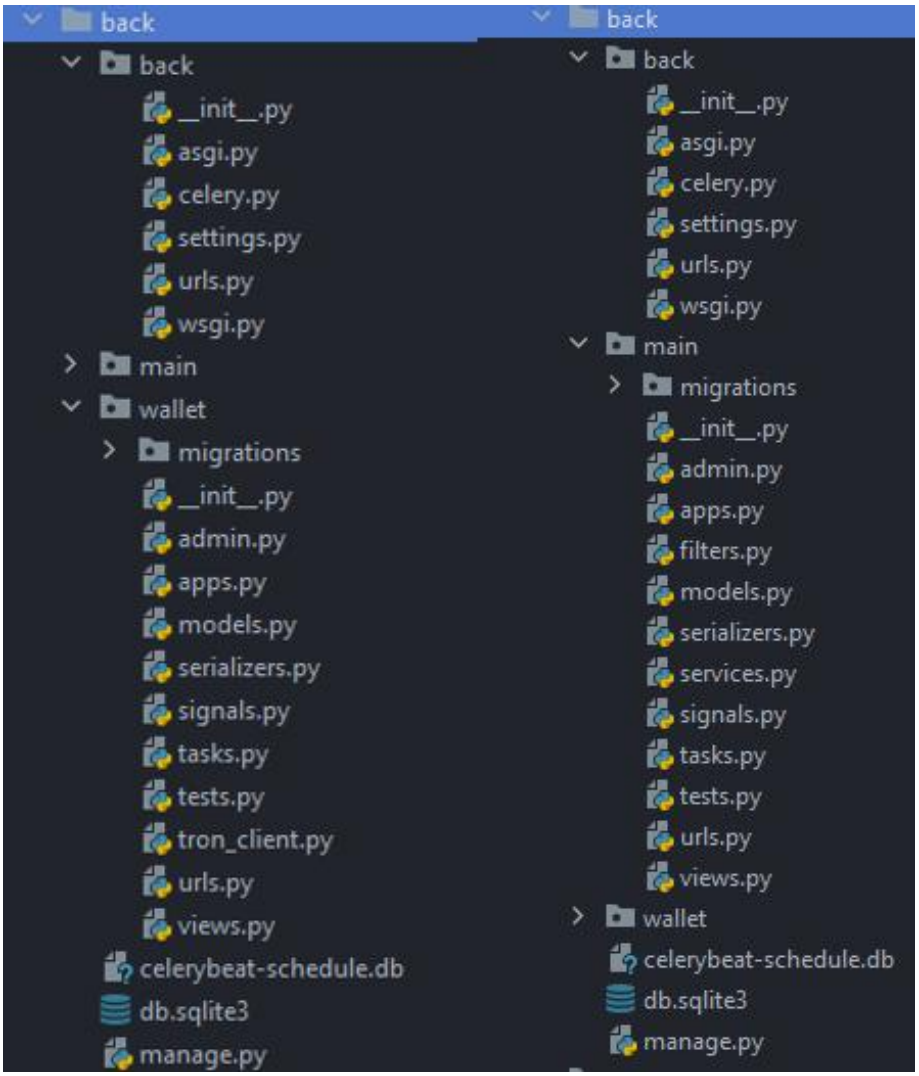

Рис. 3.6. Структура серверної частини бота

На серверній частині рекламного бота знаходяться модулі налаштування і модулі самого бота. Перелік основних: back.urls (), back.settings, back.celery, main.models, main.views, main.urls, main.signals, main.serializers, main.services, main.task, wallet.models, wallet.task, wallet.urls, wallet.urls, wallet.views, wallet.tron_client, wallet.serializers.

Через те , що у рекламного бота та серверної частини майже однакова стуктура, то виконаємо узагальнення: back.urls – модуль, який отримує всі API запити і переадресовує на дочірні(main.urls,wallet.urls), back.settings – налаштування проектів, де підключаються всі дочірні проекти і описані які сторонні модулі будуть підключені, back.celery – налаштування всіх процесів, які будуть запускатися у фоновому режимі, urls – для налаштування входів запитів і вказування, котра функція буде обробляти данні по певному посиланню, views –для обробки запитів приходящих з urls, models – модулі для створення класів котрі будуть взаємодіяти з базою даних, serializers – дозволяють зробити класи для валідації даних, що приходять у форматі JSON, tasks – описують процеси працюючі у фоному режимі, signals – в них створюються початкові сутності модулів, tron_client – для взаємодії з блокчейном.

3.5. Розробка та опис інтерфейсу користувача

Використав для розробки зручного TG і веб інтерфейсу 2 технології, а саме: React-фреймворк для front-end та бібліотеку aiogram для TG клієнтської частини.

У системі рекламного бота реалізував 10 сторінок інтерфейсів:

1) телеграм чат з ботом;
2) список рекламних оголошень;
3) створення рекламної угоди(або заявки);
4) список рекламних оголошень користувача;
5) створенням рекламного оголошення;
6) списком рекламнимних угод;
7) деталі рекламної угоди;

8) гаманець користувача;

9) поповнення коштів;

10) виводу коштів;

В TG-чаті з ботом, користувач може здійснювати комунікацію з ботом через команди: “/start” та “/app” .

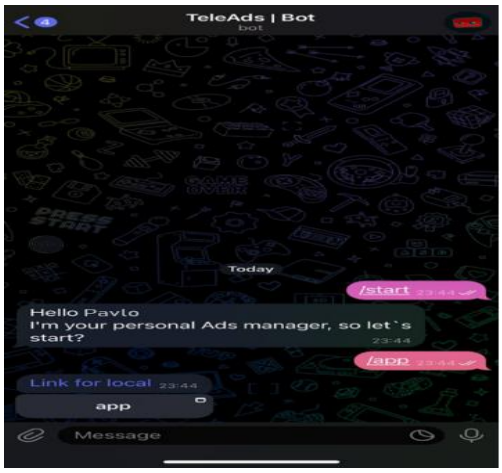

Рис. 3.7. TG-інтерфейс - чат з ботом

Юзер може зареєструватись в системі за допомогою команти “/start”, що в подальшому йому дає змогу зайти у веб-інтерфейс рекламного бота.

Головна сторінка веб-інтерфейсу висвічує список всіх актуальних рекламних оголошень виглядає наступним чином.

На сторінці користувач має змогу обрати потрібний йому канал чи групу, для того щоб розмістити свою рекламу там, де він хоче. Натискає кнопку “Create Order” – посилає на сторінку зі створенням заявки, де він повинен обрати потрібну опцію.

На даній сторінці, щоб користувач зміг обрати опцію потрібно натиснути кнопку “Add Option”, що дасть змогу обрати тип опції та вказати час, коли цей рекламний пост повинен бути розміщеним, вказати текст посту та якщо потрібно, то добавити кнопки під цим постом, для цього повинен натиснути на кнопку “Add Button” та вказати текст і лінк на кнопці. Потім він повинен вказати час оплати і натистути на конпку “Create Order”.

На цій сторінці можна переглянути список оголошення, котрі були створені, зробити їх активними, нажавши кнопку “Activate”, або не активними «Deactivat».

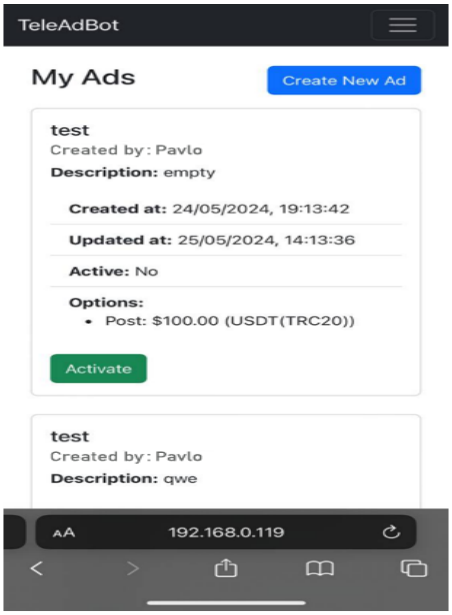

Рис. 3.8. Сторінка зі списоком рекламних оголошень користувача

Юзер натискає “Create New Ad” для переходу на сторінку зі створенням оголошення.

На цій сторінці користувач може обрати чат, в котрому буде розміщено рекламу, опис, обрати рекламні опції і вказати їх ціну.

На сторінці зі списом рекламнимних угод можна фільтрувати рекламні угоди і переходити до сторінки з деталями рекламної угоди кліком на кнопку “View Details”.

На сторінці з деталями рекламної угоди, коли статус ще знаходиться в режимі очікування «Pending» - власнику оголошення потрібно зробити підтвердження, щоб угода стала активною і перейшла в режим виконання.

На сторінці гаманця користувача можна переглянути список всіх активів і перейти на сторінку поповнення , нажавши кнопку «Deposit», або виводу коштів, нажавши кнопку «Withdrawal».

На сторінці внесення коштів користувач введе суму і нажме на кнопку “Deposit”, йому покаже реквізити і суму, куда він мусить здійснити переказ.

На сторінці виведення коштів користувачу потрібно буде вказати суму і реквізити, куда мають прийти гроші.

3.6. Альтернативні підходи, що розглядалися протягом розробки

Протягом розробки системи рекламного бота та платіжної системи також розглядав інші БД до використання. Тобто розглянуто ще MongoDB, оскільки в ньому можна зберігати дані різних структур, а в PostrgeSQL особливість фіксованої структури, котра дозволила зробити дані, які зберігаються в ній , цілісніші та надійнішими.

А також була морока з вибором блокчейну. Вибір був між TRON та Etherium. Обрав TRON, оскільки головним токеном, який буде використовуватися в способі оплаті є USDT, де 1 USDT еквівалентний 1 USD. Також в даному блокчеййні більша кількість транзакцій ніж в блокчейні Etherium та відповідно демократичніші комісії[14].

3.7. Проблеми та незвичні ситуації, які виникали під час розробки та методи для їх протидії

Виникала проблема ліквідності під час розробки платіжної системи. Тобто, коли кошти приходили на другорядні гаманці, а вивід виконувся з головного – відповідно появлялась дана проблема. вирішено дану проблему наступним чином – створено алгоритм перевірки балансу гаманців. В даному алгоритмі йдеться про те , що коли в другорядному гаманці назбирається певна сума – буде здійснено переказ на головний гаманець.

3.8. Висновки до розділу

В результаті дослідження встановив, що фрейворк джанго є більш продуктивнішим для роботи над реалізацією бота. Завдяки мікросервісній архітектурі надає високий рівень масштабованості та гнучкості що дозволяє

підлаштувати систему до росту кількості користувачів і обсягу великої кількості даних.

Змодельовано загальну структуру програмного забезпечення розробленої системи завдяки діаграм компонентів, котре дало змогу графічно візуалізувати взаємодію між компонентами та їх зв'язок.

Для кращого опису класів, атрибутів та їх взаємозвязків – створено діаграму класів. У системі, котру розроблено, діаграма класів містить класи для платіжної системи та рекламного бота. Класи платіжної системи включають: Блокчейн, Токен, Мерчант, Баланс Мерчанта, Гаманець, Баланс Гаманця, Заявка, Транзакція. Класи РБ включають: Користувач, Чат, Тип чату, Користувач Чату, Тип Користувача Чату, Валюта, Баланс Користувача, Рекламне Оголошення, Опції Рекламного Оголошення, Типи Опцій, Угоди, Статус Угоди, Транзакції, Опції Угоди.

Описано сторонні бібліотеки та розроблено модулі системи. Для зовнішньої (клієнтської частини) використав модулі React, Bootstrap та Axios. Для back-end (серверної сторони) використані такі бібліотеки як: Django, Django REST framework, Tronpy, django-cors-headers. Для TG клієнтської частини використані бібліотеки aiogram, pydantic, FastAPI, Uvicorn.

Розроблено програмні модулі системи. Структура клієнтської частини складається з пакету front, де основним з яких є src. Структура TG клієнтської частини включає основні пакети та модулі: loader, main, server, utils.api, handlers.start, data.config. Серверна частина рекламного бота та платіжної системи включає основні модулі: back.urls, back.settings, back.celery, main.models, main.views, main.urls, main.signals, main.serializers, main.services, main.task, wallet.models, wallet.task, wallet.urls, wallet.views, wallet.tron_client, wallet.serializers.

Розроблено та описано інтерфейс користувача. Для того щоб створити зручний та красивий інтерфейс використав React для веб-клієнтської частини та бібліотеку aiogram для TG клієнтської частини. У системі рекламного бота розробив такі сторінки інтерфейсу як: TG-чат з ботом, сторінка зі списком рекламних оголошень, сторінка створення рекламної угоди, сторінка зі списком

оголошень користувача, сторінка створення оголошення, сторінка зі списком рекламних угод, сторінка з деталями рекламної угоди, сторінка гаманця користувача, сторінка депозиту коштів, сторінка виводу коштів.

А ще протягом розробки розглянуто інші підходи, а саме вибір між базами даних PostgreSQL та MongoDB, а також блокчейнами Tron та Ethereum. У висновку обрав PostgreSQL та Tron, що краще відповідають вимогам мого проекту. Ще також вирішено проблему ліквідності гаманців, відповідно створивши алгоритм перевірки балансів[22].

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1. Інструкції для адміністратора, програміста, користувача

Встановлення та перенесення системи на іншу машину реалізовуватиметься за одним і тим же принципом тому, що система використовує кросплатформені мови програмування та фреймворки, що підтримуються усіма поширеними сьогодні браузерами.

Налаштував одночасний запуск frontend та backend-частини, а також проксі для зручного запуску.

Все працює на локальному сервісі, відповідно алгоритм встановлення виглядає наступним чином:

1) Запуск проекту в редакторі коду.
2) Запуск серверної частини за допомогою команди:

“python3.9 manage.py runserver”

3) Запуск веб-клієнтської частини за допомогою команди:

“npm start”

4) Запуск Celery для обробки фонових задач:

“celery -A back worker -l info”

“celery -A back beat -l info”

5) Запуск TG-бота за допомогою команди:

“python3.9 main.py”

Після виконання команд, описаних вище, з'явиться посилання на сайт, а саме http://localhost:8000/ для серверної, http://localhost:3000/ для веб-клієнтської і http://localhost:8001/ для TG-бота.

Інструкція користувача створена для того, щоб той самий користувач мав змогу зареєструватися у системі для отримання доступу до основного функціоналу ПЗ. Інтерфейс системи є лаконічним та не потребує інструкцій.

4.2. Вимоги до апаратно-програмного забезпечення

Для стабільної розробки нам потребність у хорошому залізі є обов’язковою, тому перечислимо вимоги:

- Процесор: сучасний багатоядерний та багатопоточний – для швидкості компіляції та обробки коду.
- ОЗП (оперативна пам’ять): мінімальна кількість 4 гб, але для ефективнішої роботи з великою кількістю компонентів нормальним об’ємом ОЗП буде 8-16 гб у двоканальному режимі.
- Накописувач: в ідеалі використовувати SSD типу SATA або NVMe m2, другий має більшу швидкість запису та зчитування даних, що значно прискорить завантаження та компіляцію ПЗ.

Перелічені вище вимоги дадуть змогу швидко та ефективно займатись як розробкою так і використанням системи рекламного бота.

4.3. Тестування

Використав Postman для тестування HTTP-запитів серверної частини розроблюваного ПЗ, котра надає змогу створювати різні типи запитів, додавати та змінювати параметри, виконувати аутентифікацію, отримувати та обробляти файли, аналізувати запити та переглядати дані у різних форматах (XML, JSON). Тестування проводив через методи: GET, POST.

У тестуванні велику роль зіграв Django, оскільк в ньому models і serializer дають змогу у відповідь на запити зазначити, котрих полів не хватає, або не правильного формату.

У випадку якщо дані не будуть збережені через не правильний формат, то Django повідомить у відповідь до здійсненого запиту з кодом відповіді 400, а при не авторизованому запиті з кодом 401. Наприклад, візьмем реєстрацію чату та користувача, а інші просто переглянемо відповіді при коректних даних.

Тестування коректності запиту реєстрації або створення облікового запису:

Якщо нового юзера немає в БД – система надасть нам відповідь з успішною реєстрацією користувача та відповідно збереже його в БД.

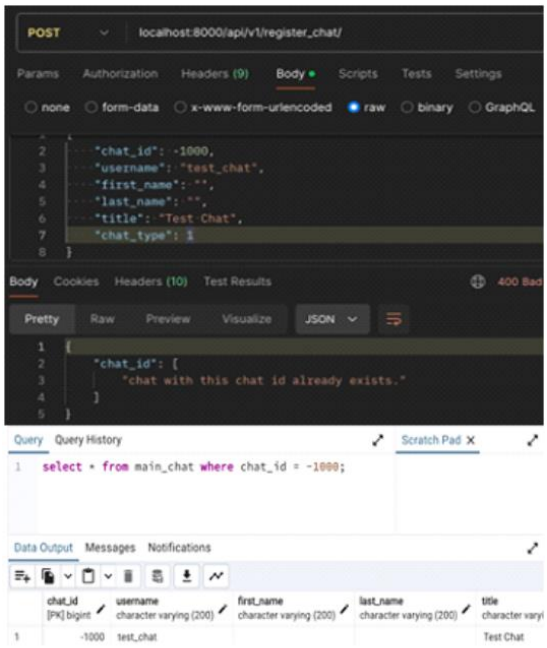

Рис. 4.1. Запит на реєстрацію користувача та дані в БД

У випадку якщо такий акаунт вжде існує, то система надасть відповідну відповідь.

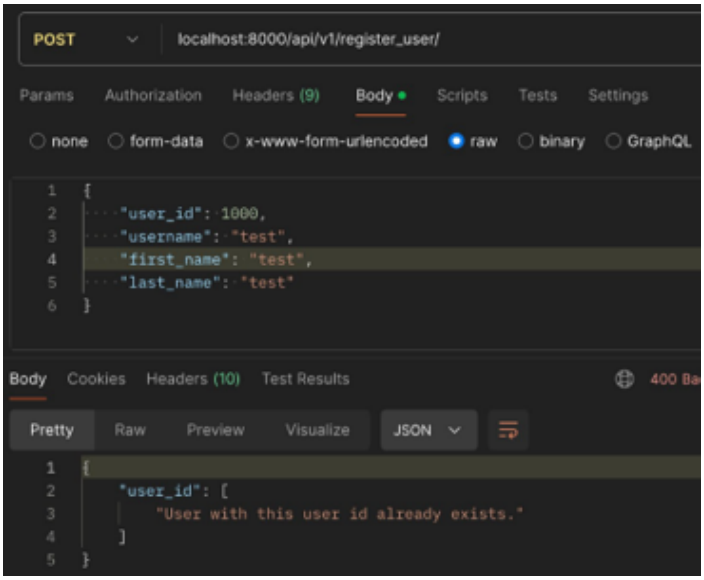

Рис. 4.2. Запит на створення акаунту, який вже існує

Тестування правильності запиту добавлення нового чату (реєстрації):

У випадку коли новий чат не існує в БД, то система дає сповіщення про успішну реєстрацією чату і зберігає його в БД

Якщо чат вже існує – система відповідає, що даний чат вже існує .

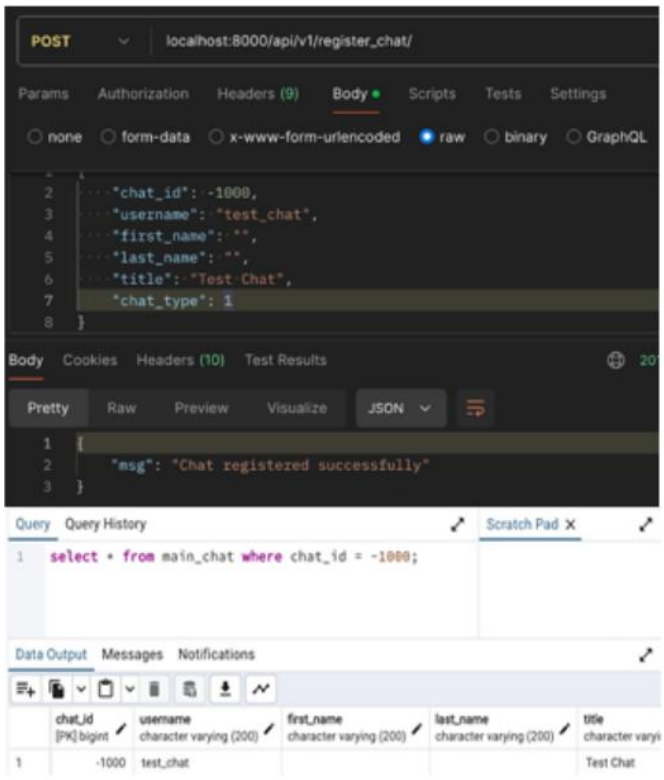

Рис. 4.3. Запит на додавання нового чату та дані про збереження

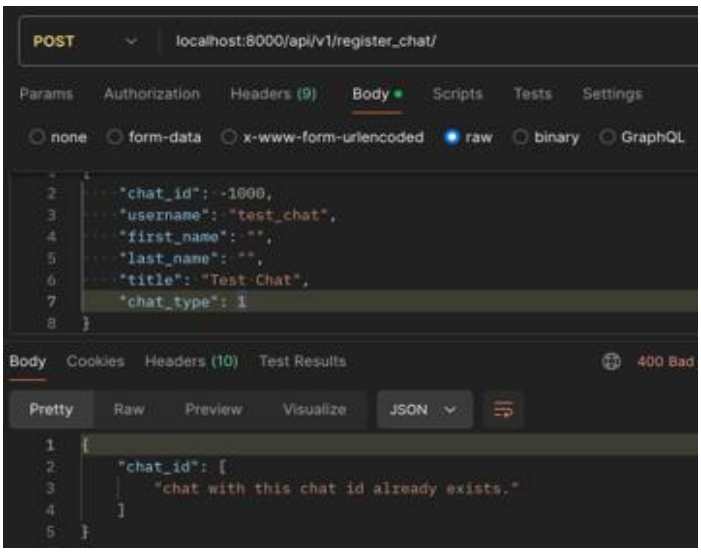

Рис. 4.4. Запит на додавання чату, що вже існує

Тестування правильності запиту на підвязку чату до користувача

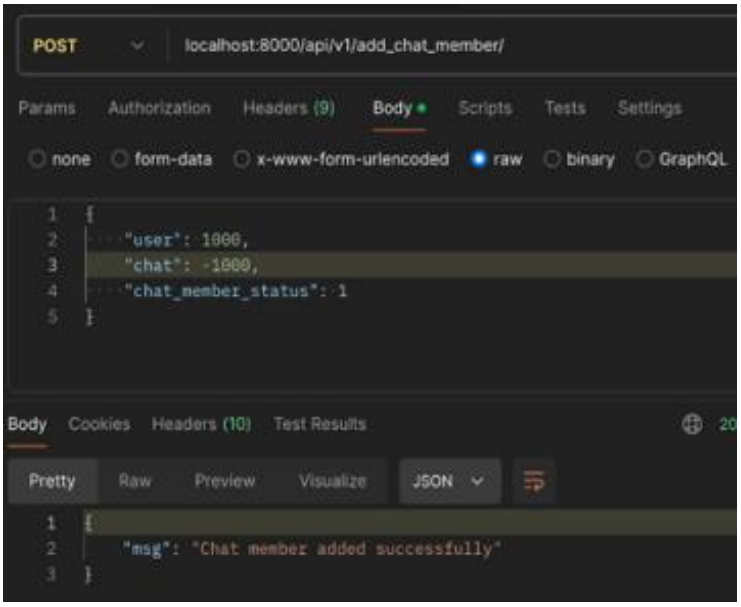

Рис. 4.5. Запит на підв’язку користувача до чату

Щоб перевірити наступні запитів на веб-клієнті системи, нам відповідно потрібен токен авторизації.

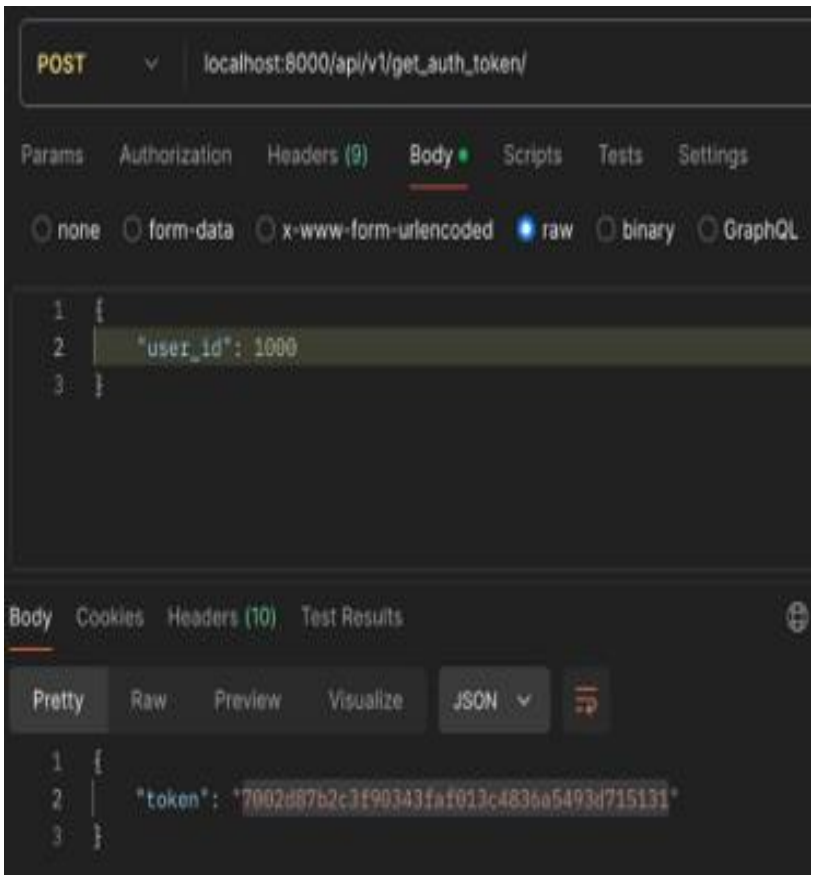

Рис. 4.6. Отримання токена авторизації

Запит на отримання всіх чатів з’єднаних з користувачем.

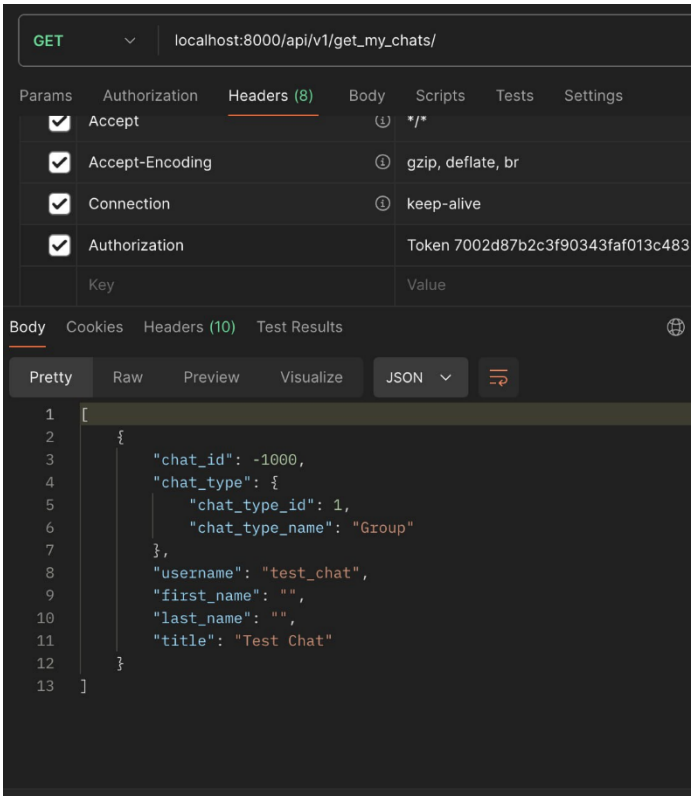

Рис. 4.7. Запит отримання всіх чатів з’єднаних з користувачем

Перевірка запита на створення рекламного оголошення

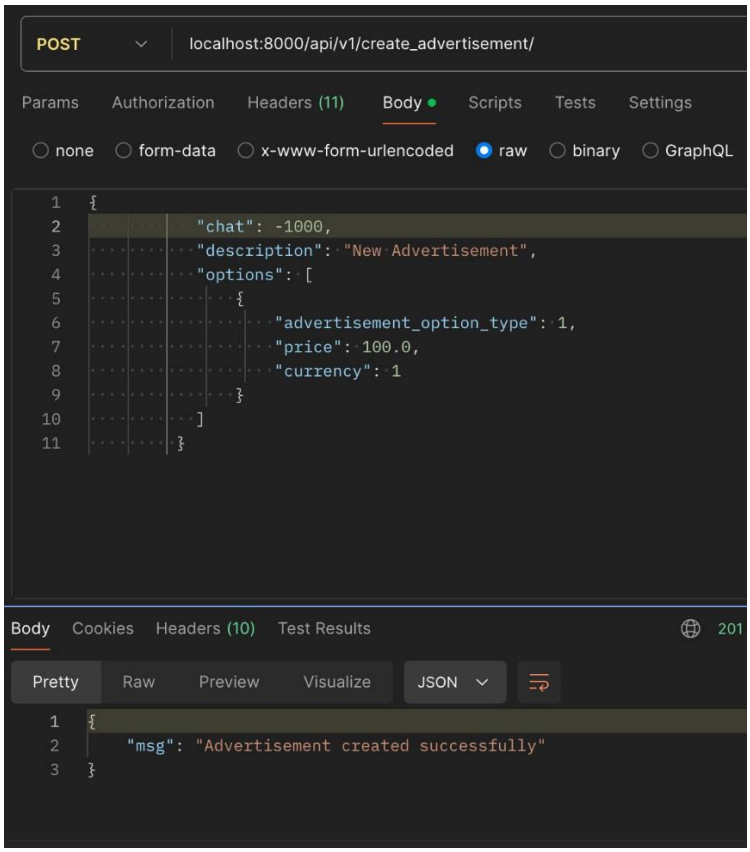

Рис. 4.8 Запит створення рекламного оголошення

Перевірка запиту на отримання всіх активних рекламних оголошень рекламодавця.

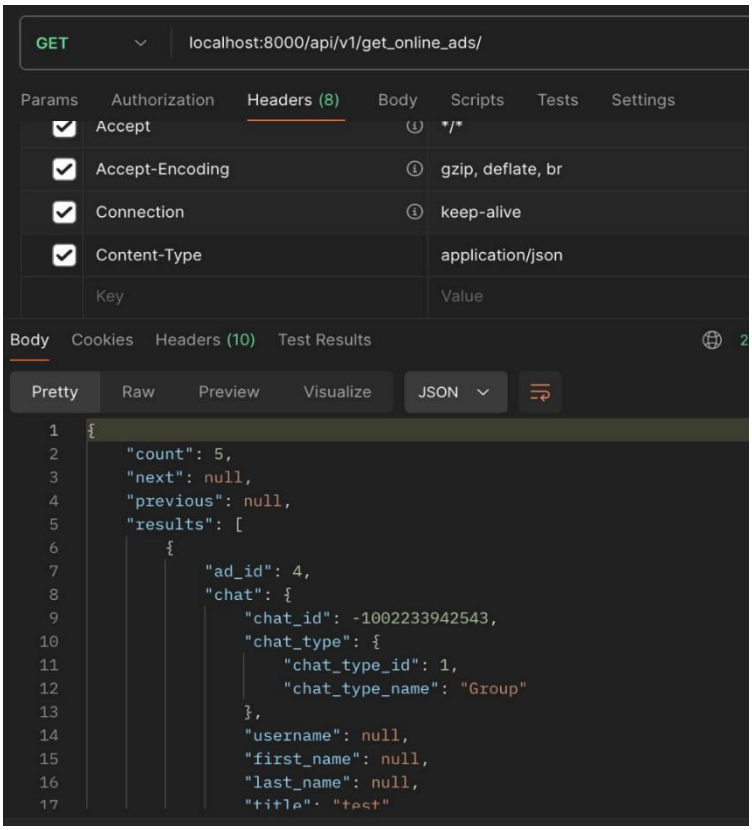

Рис. 4.9. Список всіх активних рекламних оголошень

Перевірка запиту для отримання усіх типів рекламних характеристик, які знадобляться для створення оголошення.

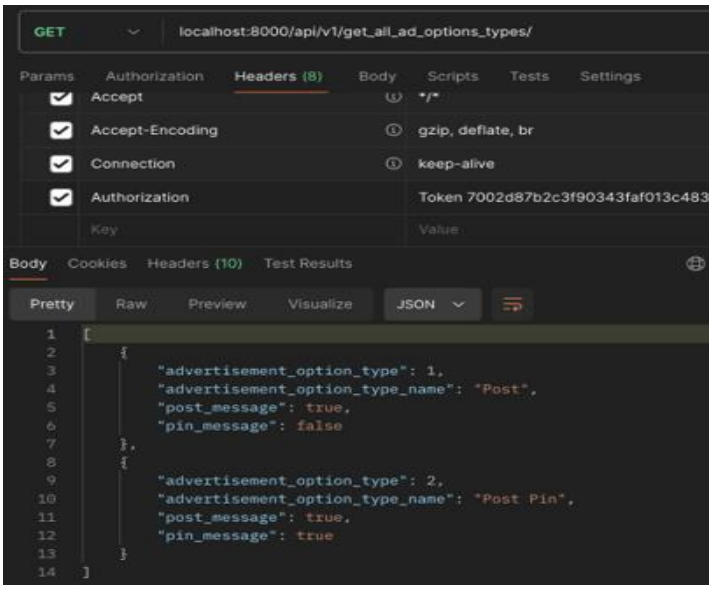

Рис. 4.10. Отримати всі типи рекламних опцій

Перевірка запиту на отримання інформації про юзера.

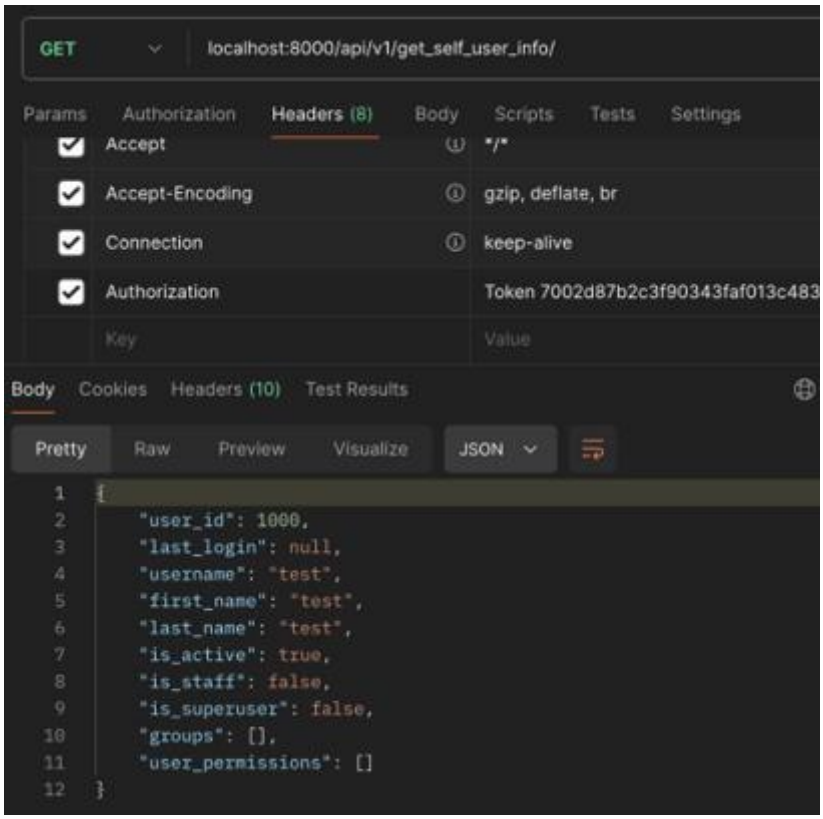

Рис. 4.11. Запит на отримання інформації про користувача

Перевірка запиту на отримання на усіх рекламних угод юзера.

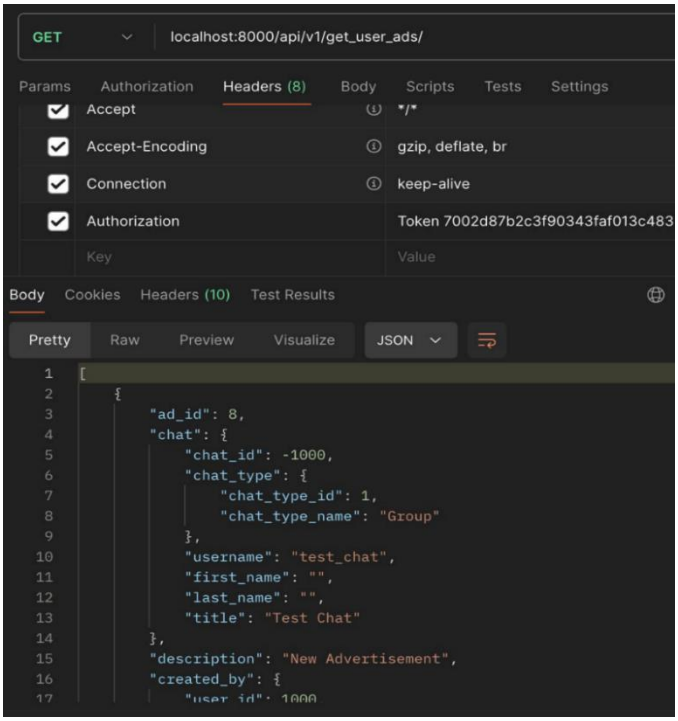

Рис. 4.12. Запит на отримання всі рекламних угод користувача

Перевірка запитів на перегляд гаманця користувача, його поповнення та вивід коштів.

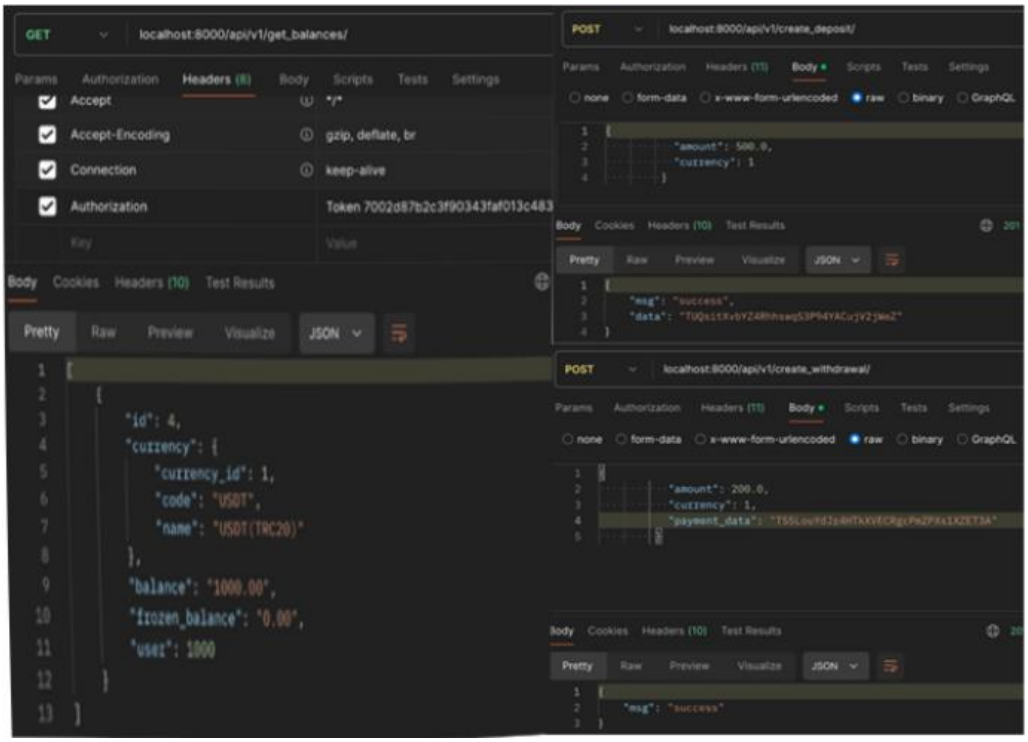

Рис. 4.13. Запити на перегляд, поповнення та вивід коштів з гаманця

Перевірка запитів на створення рекламної заявки та отримання списку усіх рекламних угод юзера.

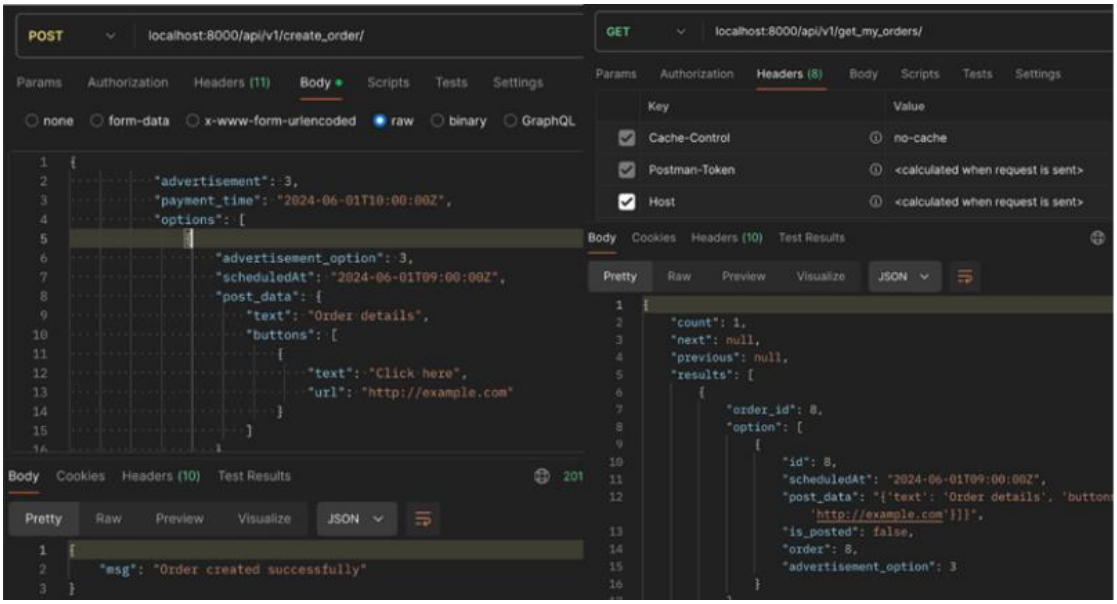

Рис. 4.14. Запити на створення рекламної угоди та перегляду всіх угод юзера

Виконуємо перевірку створення та підтвердження заявки на поповнення коштів. Після створення заявки платіжна система повинна у відповідь відправити деталі заявки для перевірки, потім рекламний бот повинен надати підтвердження для отримання реквізитів.

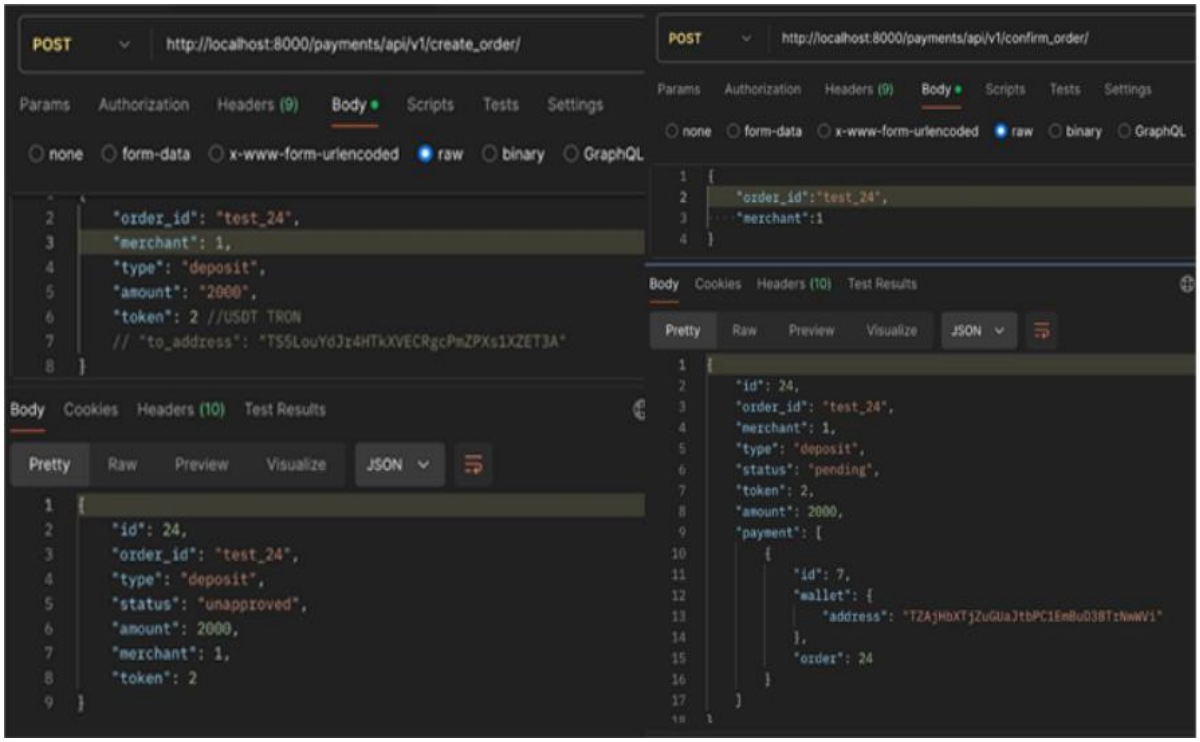

Рис. 4.15. Запити на створення та підтвердження заявки на поповнення

За аналогічним принципом працють також запити виводу коштів, але при створенні заявки на вивід потрібно вказати реквізити.

Виконуємо перевірку запиту на розміщення реклами через TG-бот.

Вітправляється набір постів, котрі повинні бути опубліковані, а у відповідь отримують пости, що були розміщені.

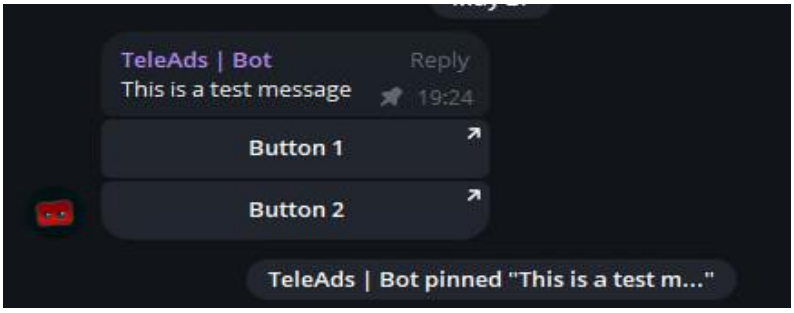

Рис. 4.16. Розміщений рекламний пост

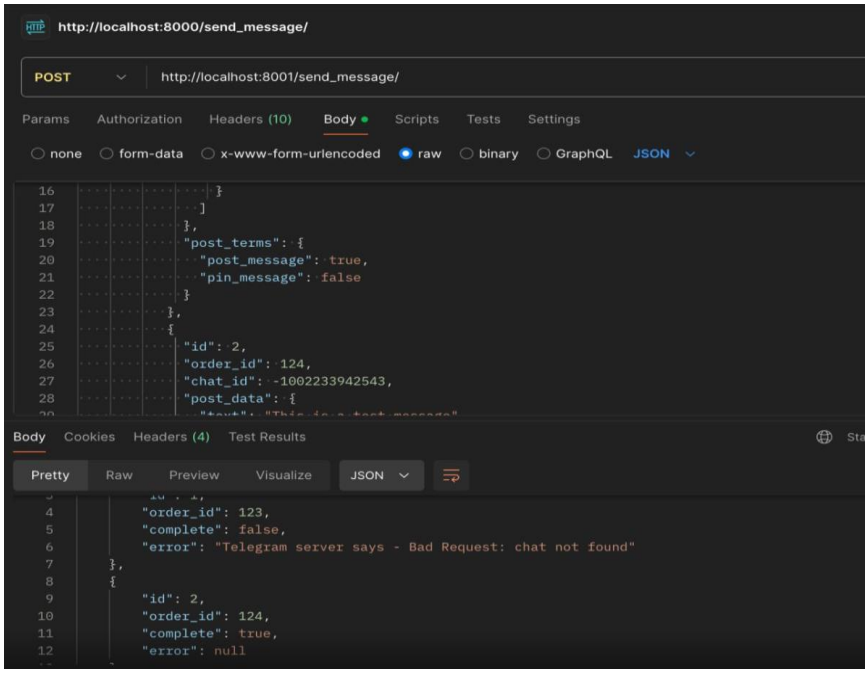

Рис. 4.17. Запит на розміщення рекламного поста

За допомогою Postman було проведено перевірку API, для того щоб впевнитися у правильному реагуванні системою на запити.

Також перевіримо поповенення та вивід коштів з рекламного бота та платіжної системи через веб для того, щоб впевнитись у коректному функціонуванні усіх компонентів ПЗ.

Проводимо перевірку поповнення за наступним алгоритмом дій: переходим на сторінку гаманця – нажимаєм на кнопку для депозиту – вводимо суму поповнення та нажимоємо кнопку для депозиту – отримуємо реквізити – робимо переказ – перевіряємо баланс гаманця.

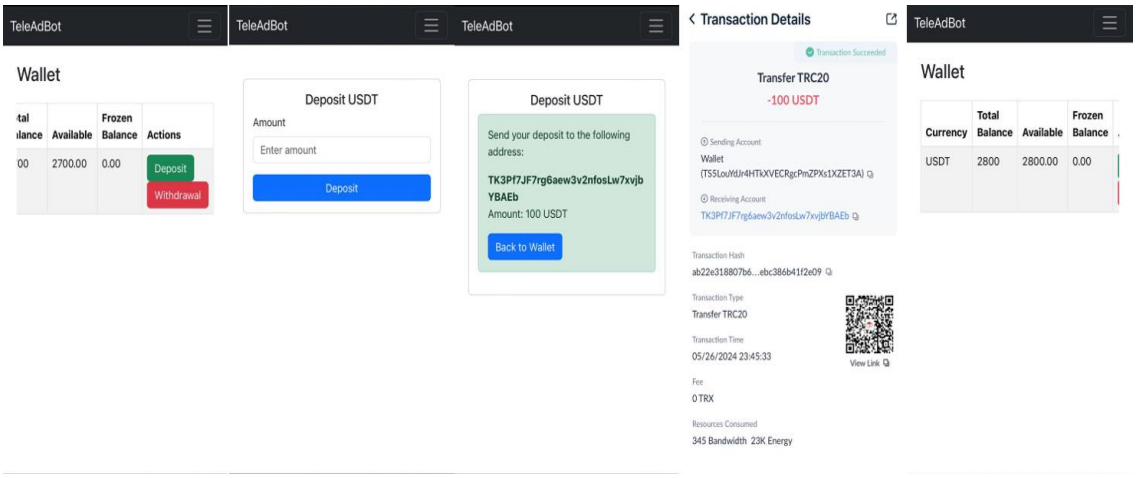

Рис. 4.18. Алгоритм поповнення коштів

Наступним кроком перевіримо вивід коштів за наступним алгоритмом дій: перехід на сторінку гаманця – нажати на клавішу “Вивести” – ввід суми виводу, реквізити та клік на клавішу “Вивести” – перевіряємо баланс криптогаманця.

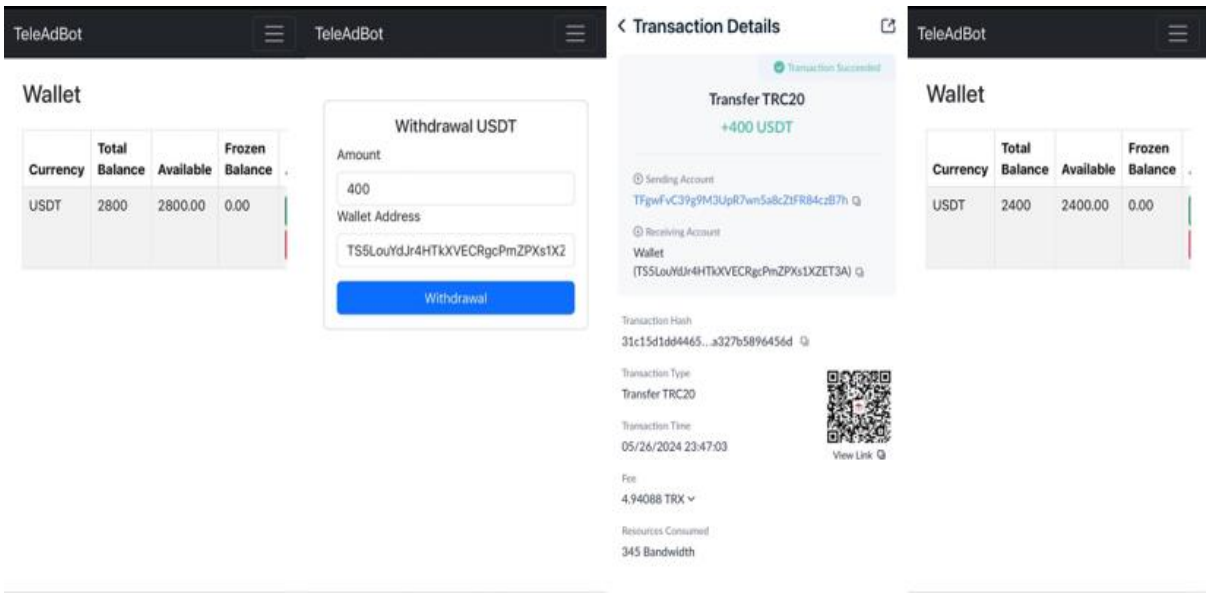

Рис. 4.19. Алгоритм виведення коштів

Отже, ми провели успішне тестування усіх запитів незахищеного протоколу передачі даних серверної частини розробленого нами ПЗ.

4.4. Аналіз та оцінка отриманих результатів

Провів тестування HTTP-запитів серверної частини системи рекламного бота, котре дало змогу впевнитись у правильному функціонуванні усіх компонентів системи. Також використовував платформу Postman для створення та виконання запитів, котре підтвердило, що система коректнцо обробляє різні типи запитів, здійснює аутентифікацію, обробляє файли та повертає дані у відповідних форматах. Результати тестування показали, що серверна частина стабільно працює під навантаженням, ефективно взаємодіє з базою даних і зовнішніми сервісами, а також забезпечує швидку обробку запитів без помилок.

Тестування підтвердило:

1) Запити на реєстрацію нових користувачів та чатів – успішно зберігають дані в БД відповідно забезпечують правильну обробку дублюючих записів.

2) Запити на підв’язування користувачів до чатів – правильно зв'язують дані у базі та надають відповідні токени авторизації.

3) Запити на створення та управління рекламними оголошеннями – надають змогу для зберігання, активації та деактивації та отримання рекламних оголошень.

4) Запити на фінансові операції – надають змогу для поповнення та виведення коштів та коректно обробляють транзакції через платіжну систему.

5) Запити від TG-бота – обробляють розміщення рекламних постів та відповідно повертають результати їх розміщення і можливі помилки.

Ще було перевірено роботоспосібність веб-системи для користувачів. Підтвердилась ефективна функціональність тестуванням процесу поповнення та виведення коштів через веб-інтерфейс та його залежних компонентів компонентів.

Загалом, результати тестування демонструють високу стабільність та надійність системи. Всі компоненти правильно взаємодіють між собою, забезпечуючи коректну роботу системи в цілому.

4.5. Висновки до розділу

В даному розділі розглянуто інструкції для користувачів, адміністраторів та програмістів, а саме: встановдення та налуштування системи рекламного бота та платіжної системи. Також визначив вимоги до конфігурації ПК, котрі забезпечать стабільну, швидку та ефективну розробку та запуск розробленою мної системи. За допомогою Postman було проведено тестування http-запитів, котре підтвердило правильну функціональність серверної частини ПЗ.

Згідно проведеного тестування було зроблено наступні висновки, а саме те, що всі компоненти системи рекламного бота правильно функціонують та відповідно забезпечують увесь потрібний функціонал для користувачів.

Отже, система рекламного бота відповідє вимогам проекту, демонструючи високу продуктивність та надійність, що має змогу надати можливість ефективної та комфортної взаємодії користувачів з ПЗ.

ВИСНОВКИ

У магістерській рооботі було розроблено автоматизовану систему для розміщення реклами в TG каналах та групах. Під час аналізу літературних джерел та існуючих систем виявив потребу в зручному інструменті для комфортного керування рекламними оголошеннями та фінансовими операціями на прощадці меседжеру TG. Визначити ключові задачі проекту та способи їх вирішення допомогла побудова дерева проблем та дерева цілей.

Архітектура розробленої системи, включає в себе такі клієнтські компоненти: Telegram Bot та Web Client. Серверні компоненти: Back-end Server, Database та Payment System. А також створено ключові алгоритми, котрі забезпечили функціональність системи, а саме такі як авторизація користувачів, створення рекламних заявок, управління оголошеннями та обробка транзакцій.

Також змодельовано загальну структуру програмного забезпечення завдяки діаграмам класів та описано використані бібліотеки та модулі.

Коректну роботу системи перевірено тестуванням компонентів, котре дало змогу впевнитись у їх задуманій фукціональності. Тестував http-запити за допомогою платформи Postman, що дало змогу підтвердити правильність та надійність роботи серверної частини. Згідно минулого твердження виявлено, що компоненти системи функціонують відповідно до вимог проекту та забезпечуючи необхідний функціонал.

Даний проект є актуальним та перспективним для ринку реклами на платформі TG, оскільки монетизація в даному меседжері не поширюється на наш регіон. Проект надає зручний інтерфейс для власників каналів та рекламодавців, дозволяючи ефективно керувати рекламними кампаніями та фінансовими операціями. Реалізована платіжна система на основі блокчейну TRON забезпечує додаткові переваги у вигляді швидкості та анонімності транзакцій.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. "Django for Beginners". William S. Vincent. 2022. Archived from the original on 15 January 2023. Retrieved 2023-03-10.
2. "Python Crash Course". Eric Matthes. No Starch Press. 2019. Archived from the original on 10 January 2023. Retrieved 2023-03-20.
3. "Two Scoops of Django 3.x". Daniel Roy Greenfeld, Audrey Roy Greenfeld. 2021. Archived from the original on 20 March 2023. Retrieved 2023-05-15.
4. "Flask Web Development". Miguel Grinberg. O'Reilly Media. 2018. Archived from the original on 15 March 2023. Retrieved 2023-05-25.
5. "Django REST Framework by Example". William S. Vincent. 2021. Archived from the original on 18 June 2023. Retrieved 2023-08-10.
6. "High Performance Django". Peter Baumgartner, Yann Malet. 2015. Archived from the original on 10 February 2023. Retrieved 2023-04-12.
7. "Python Programming: An Introduction to Computer Science". John Zelle. Franklin, Beedle & Associates. 2017. Archived from the original on 25 December 2022. Retrieved 2023-02-15.
8. "Mastering Django". Nigel George. 2020. Archived from the original on 12 March 2023. Retrieved 2023-04-01.
9. "Learning Python". Mark Lutz. O'Reilly Media. 2022. Archived from the original on 1 May 2023. Retrieved 2023-06-10.
10. "Django Design Patterns and Best Practices". Arun Ravindran. Packt Publishing. 2021. Archived from the original on 20 February 2023. Retrieved 2023-05-05.
11. "Automate the Boring Stuff with Python". Al Sweigart. No Starch Press. 2019. Archived from the original on 5 May 2022. Retrieved 2022-08-20.
12. "Pro Python". Marty Alchin, J. Burton Browning. Apress. 2020. Archived from the original on 15 June 2023. Retrieved 2023-08-10.
13. "Programming Python". Mark Lutz. O'Reilly Media. 2021. Archived from the original on 7 April 2023. Retrieved 2023-06-20.
14. "Effective Python: 59 Specific Ways to Write Better Python". Brett Slatkin. Addison-Wesley. 2021. Archived from the original on 25 March 2023. Retrieved 2023-05-18.

15. "Test-Driven Development with Python". Harry J.W. Percival. O'Reilly Media. 2017. Archived from the original on 18 July 2023. Retrieved 2023-08-15.

16. "Django Unleashed". Andrew Pinkham. Addison-Wesley. 2016. Archived from the original on 10 May 2023. Retrieved 2023-07-01.

17. "Python Tricks: A Buffet of Awesome Python Features". Dan Bader. Dan Bader Publishing. 2017. Archived from the original on 20 April 2023. Retrieved 2023-06-15.

18. "Django 4 By Example". Antonio Mele. Packt Publishing. 2022. Archived from the original on 12 March 2023. Retrieved 2023-05-22.

19. "Hands-On Django 4". Malcolm Maclean. Packt Publishing. 2023. Archived from the original on 15 June 2023. Retrieved 2023-07-10.

20. "Django RESTful Web Services". Gaston C. Hillar. Packt Publishing. 2018. Archived from the original on 10 January 2023. Retrieved 2023-03-05.

21. "Introduction to Machine Learning with Python". Andreas C. Müller, Sarah Guido. O'Reilly Media. 2017. Archived from the original on 1 April 2023. Retrieved 2023-05-10.

22. "Web Development with Django Cookbook". Aidas Bendoraitis. Packt Publishing. 2021. Archived from the original on 5 February 2023. Retrieved 2023-04-15.

23. "Deep Learning with Python". François Chollet. Manning Publications. 2021. Archived from the original on 18 July 2023. Retrieved 2023-08-05.

24. "The Django Book". Adrian Holovaty, Jacob Kaplan-Moss. 2022. Archived from the original on 20 March 2023. Retrieved 2023-06-01.

25. "Mastering Flask". Jack Stouffer. Packt Publishing. 2018. Archived from the original on 15 June 2023. Retrieved 2023-07-15.

26. "The Hitchhiker's Guide to Python". Kenneth Reitz, Tanya Schlusser. O'Reilly Media. 2021. Archived from the original on 10 March 2023. Retrieved 2023-05-01.

27. "Python for Data Analysis". Wes McKinney. O'Reilly Media. 2022. Archived from the original on 18 June 2023. Retrieved 2023-08-10.

28. "Django Admin Cookbook". AJ Bowen. 2020. Archived from the original on 12 February 2023. Retrieved 2023-04-12.

29. "Python and Django Full Stack Web Developer Bootcamp". Jose Portilla. Udemy. 2021. Archived from the original on 25 March 2023. Retrieved 2023-06-05.

30. "Django Up and Running". Andrew Pinkham. 2019. Archived from the original on 1 April 2023. Retrieved 2023-05-10.

31. "Pro Django". Marty Alchin. Apress. 2019. Archived from the original on 25 February 2023. Retrieved 2023-04-15.

32. "Full-Stack Django and React". Nikhil Sreenivasan. Packt Publishing. 2022. Archived from the original on 18 May 2023. Retrieved 2023-07-01.

33. "Real Python: Advanced Web Development with Django". Real Python Team. Real Python. 2022. Archived from the original on 5 June 2023. Retrieved 2023-07-20.

34. "Learning Web Development with Flask". Miguel Grinberg. 2021. Archived from the original on 12 May 2023. Retrieved 2023-06-15.

35. "Designing Microservices with Django". Armin Ronacher. O'Reilly Media. 2023. Archived from the original on 20 June 2023. Retrieved 2023-08-01.

36. "Securing Django Applications". Joseph Turner. Manning Publications. 2022. Archived from the original on 10 May 2023. Retrieved 2023-07-05.

37. "Django Testing Cookbook". F. Antony. Packt Publishing. 2021. Archived from the original on 15 April 2023. Retrieved 2023-06-10.

38. "Mastering Python Networking". Eric Chou. Packt Publishing. 2021. Archived from the original on 1 June 2023. Retrieved 2023-08-05.

39. "Django ORM Mastery". Andrew Burgess. 2022. Archived from the original on 10 March 2023. Retrieved 2023-05-01.

40. "Full Stack Python". Matthew Makai. Leanpub. 2020. Archived from the original on 20 May 2023. Retrieved 2023-07-10.

41. “ТОП-5 Кращих Фреймворків CSS”. (дата звернення: 27.11.2024) Режим доступу: https://itproger.com/ua/news/top-5-luchshih-freymvorkov-css

ДОДАТКИ

Додаток А

Фрагмент програмних лістингів

main/views.py

```
from django.contrib.auth import authenticate
from rest_framework.decorators import api_view, permission_classes
from rest_framework.generics import ListAPIView
from rest_framework.response import Response
from rest_framework import status
from rest_framework.authtoken.models import Token
from rest_framework.permissions import AllowAny, IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import viewsets
from rest_framework.filters import OrderingFilter
from rest_framework.pagination import PageNumberPagination

from .filters import OrderFilter, AdvertisementFilter
from .models import User, Chat, ChatMember, Advertisement,
AdvertisementOption, Order, AdvertisementOptionType, \
    OrderAdvertisementOption, OrderStatus, Currency, UserBalance,
ChatMemberStatus, ChatType
from .serializers import UserSerializer, ChatSerializer,
ChatMemberSerializer, AdvertisementSerializer, \
    AdvertisementOptionSerializer, OrderSerializer, \
    AdvertisementOptionTypeSerializer,
CreateAdvertisementSerializer, PreOrderAdvertisementSerializer, \
    CreateOrderSerializer, ConfirmOrderSerializer,
CurrencySerializer, UserBalanceSerializer, OrderDetailSerializer, \
    CreateDepositTransactionSerializer,
CreateWithdrawalTransactionSerializer,
AdvertisementUpdateStatusSerializer, \
    OrderStatusSerializer, ChatTypeSerializer
from .services import TronPaymentService
```

```
@api_view(['POST'])
@permission_classes([AllowAny])
def register_user(request):
    if request.method == 'POST':
        user_serializer = UserSerializer(data=request.data)
        if user_serializer.is_valid():
            user_serializer.save()
            return Response({'msg': 'User registered
successfully'}, status=status.HTTP_201_CREATED)
        return Response(user_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['POST'])
@permission_classes([AllowAny])
def register_chat(request):
    if request.method == 'POST':
        chat_serializer = ChatSerializer(data=request.data)
        if chat_serializer.is_valid():
            chat_serializer.save()
            return Response({'msg': 'Chat registered
successfully'}, status=status.HTTP_201_CREATED)
        return Response(chat_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['POST'])
@permission_classes([AllowAny])
def add_chat_member(request):
    if request.method == 'POST':
        chat_member_serializer =
ChatMemberSerializer(data=request.data)
        if chat_member_serializer.is_valid():
            chat_member_serializer.save()
            return Response({'msg': 'Chat member added
successfully'}, status=status.HTTP_201_CREATED)
```

```
        return              Response(chat_member_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_advertisement(request):
    if request.method == 'POST':
        data = request.data.copy()
        data['created_by'] = request.user.user_id
        ad_serializer = CreateAdvertisementSerializer(data=data)
        if ad_serializer.is_valid():
            chat = ad_serializer.validated_data['chat']
            chat_owner    =    ChatMember.objects.filter(chat=chat,
user=request.user,
chat_member_status__chat_member_status_name='Creator').exists()
            if chat_owner:
                ad_serializer.save()
                return   Response({'msg':   'Advertisement   created
successfully'}, status=status.HTTP_201_CREATED)
            else:
                return Response({'error': 'You are not the owner of
the chat'}, status=status.HTTP_403_FORBIDDEN)
        return                        Response(ad_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def advertisement_status_switch(request):
    if request.method == 'POST':
        ad_serializer                                              =
AdvertisementUpdateStatusSerializer(data=request.data)
        if ad_serializer.is_valid():
            try:
                ad                                                 =
Advertisement.objects.get(ad_id=ad_serializer.validated_data['ad_i
d'])
```

```
                chat_owner = ChatMember.objects.filter(
                    chat=ad.chat,
                    user=request.user,

chat_member_status__chat_member_status_name='Creator'
                ).exists()
                if chat_owner:
                    ad.is_active                                    =
ad_serializer.validated_data["is_active"]
                    ad.save()
                    return  Response({'msg':  'Advertisement  status
updated successfully'}, status=status.HTTP_200_OK)
                else:
                    return Response({'error': 'You are not the owner
of the chat'}, status=status.HTTP_403_FORBIDDEN)
            except Advertisement.DoesNotExist:
                return   Response({'error':   'Advertisement   not
found'}, status=status.HTTP_404_NOT_FOUND)
        return                        Response(ad_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_all_advertisements(request):
    if request.method == 'GET':
        advertisements = Advertisement.objects.all()
        serializer    =    AdvertisementSerializer(advertisements,
many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

@api_view(['POST'])
@permission_classes([AllowAny])
def get_auth_token(request):
    if request.method == 'POST':
        user_id = request.data.get('user_id')
```

```
        try:
            user = User.objects.get(user_id=user_id)
        except User.DoesNotExist:
            return Response({'error': 'Invalid user ID'},
status=status.HTTP_400_BAD_REQUEST)

        # Authenticate the user and create the token
        token, created = Token.objects.get_or_create(user=user)
        return Response({'token': token.key},
status=status.HTTP_200_OK)


    return Response({'error': 'Invalid method'},
status=status.HTTP_405_METHOD_NOT_ALLOWED)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_my_chats(request):
    if request.method == 'GET':
        chat_members =
ChatMember.objects.filter(user_id=request.user.user_id)
        chats =
Chat.objects.filter(chat_id__in=chat_members.values_list('chat_id'
, flat=True))

        serializer = ChatSerializer(chats, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_all_ad_options_types(request):
    if request.method == 'GET':
        ad_options = AdvertisementOptionType.objects.all()
```

```
        serializer = AdvertisementOptionTypeSerializer(ad_options,
many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_user_ads(request):
    if request.method == 'GET':
        try:
            creator_status                                          =
ChatMemberStatus.objects.get(chat_member_status_name='Creator')
            chat_member_ids                                         =
ChatMember.objects.filter(user=request.user,
chat_member_status=creator_status).values_list(
                'chat_id', flat=True)
            advertisements                                          =
Advertisement.objects.filter(chat_id__in=chat_member_ids)
            ads_serializer                                          =
AdvertisementSerializer(advertisements, many=True)
            return                    Response(ads_serializer.data,
status=status.HTTP_200_OK)
        except ChatMemberStatus.DoesNotExist:
            return Response([], status=status.HTTP_200_OK)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_self_user_info(request):

    if request.method == 'GET':
        user = request.user
        serializer = UserSerializer(user)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

```
class StandardResultsSetPagination(PageNumberPagination):
    page_size = 10
    page_size_query_param = 'page_size'
    max_page_size = 100
class GetOnlineAdsView(ListAPIView):
    serializer_class = AdvertisementSerializer
    permission_classes = [AllowAny]
    filter_backends = [DjangoFilterBackend, OrderingFilter]
    filterset_class = AdvertisementFilter
    ordering_fields = ['created_at', 'updated_at']
    pagination_class = StandardResultsSetPagination

    def get_queryset(self):
        return Advertisement.objects.filter(is_active=True)


@api_view(['GET'])
@permission_classes([AllowAny])
def get_chat_types(request):
    chat_types = ChatType.objects.all()
    serializer = ChatTypeSerializer(chat_types, many=True)
    return Response(serializer.data, status=status.HTTP_200_OK)

@api_view(['GET'])
@permission_classes([AllowAny])
def get_currencies(request):
    currencies = Currency.objects.all()
    serializer = CurrencySerializer(currencies, many=True)

    return Response(serializer.data, status=status.HTTP_200_OK)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_ad(request):
```

```
    if request.method == 'GET':
        ad_id = request.GET.get('ad_id')
        ad = Advertisement.objects.get(ad_id=ad_id)
        serializer = PreOrderAdvertisementSerializer(ad)
        return Response(serializer.data, status=status.HTTP_200_OK)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_order(request):
    if request.method == 'POST':
        request.data['taker'] = request.user.user_id
        serializer = CreateOrderSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg': 'Order created successfully'},
status=status.HTTP_201_CREATED)
        return                          Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)




class GetMyOrdersView(ListAPIView):
    serializer_class = OrderSerializer
    permission_classes = [IsAuthenticated]
    filter_backends = [DjangoFilterBackend, OrderingFilter]
    filterset_class = OrderFilter

    ordering_fields = ['created_at', 'expired_at', 'payment_time',
'finished_at']
    pagination_class = StandardResultsSetPagination

    def get_queryset(self):
```

```
        user = self.request.user
        try:
            creator_status                                          =
ChatMemberStatus.objects.get(chat_member_status_id=1)
            chat_member_ids  =  ChatMember.objects.filter(user=user,
chat_member_status=creator_status).values_list('chat_id',
flat=True)
            advertisements                                          =
Advertisement.objects.filter(chat_id__in=chat_member_ids)
            orders_as_owner                                         =
Order.objects.filter(advertisement__in=advertisements)
            orders_as_creator = Order.objects.filter(taker=user)
            return orders_as_owner | orders_as_creator
        except ChatMemberStatus.DoesNotExist:
            return Order.objects.none()

    def list(self, request, *args, **kwargs):
        queryset = self.filter_queryset(self.get_queryset())

        page = self.paginate_queryset(queryset)
        if page is not None:
            serializer = self.get_serializer(page, many=True)
            return self.get_paginated_response(serializer.data)

        serializer = self.get_serializer(queryset, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)


@api_view(['GET'])

@permission_classes([IsAuthenticated])
def get_order_statuses(request):
    order_statuses = OrderStatus.objects.all()
    serializer = OrderStatusSerializer(order_statuses, many=True)
```

```
    return Response(serializer.data, status=status.HTTP_200_OK)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def confirm_order(request):
    if request.method == 'POST':
        confirm_order_serializer                                    =
ConfirmOrderSerializer(data=request.data)
        if confirm_order_serializer.is_valid():
            user_id = request.user.user_id
            if
confirm_order_serializer.user_can_confirm_order(user_id):
                if confirm_order_serializer.data['confirm']:
                    confirm_order_serializer.confirm_order()
                else:
                    confirm_order_serializer.cancel_order()
                return         Response({'msg':         'Success'},
status=status.HTTP_200_OK)
            return  Response({'msg':  'User  cannot  confirm  the
order'}, status=status.HTTP_403_FORBIDDEN)
        return            Response(confirm_order_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
@permission_classes([AllowAny])
def get_currencies(request):
    if request.method == 'GET':
        serialize                                                   =
CurrencySerializer(Currency.objects.all(),many=True)
        return Response(serialize.data,status=status.HTTP_200_OK)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_balances(request):
```

```
    if request.method == 'GET':
        user = request.user
        user_balances = UserBalance.objects.filter(user=user)
        serializer      =      UserBalanceSerializer(user_balances,
many=True)
        return Response(serializer.data,status=status.HTTP_200_OK)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_order(request,order_id):
    if request.method == 'GET':
        user = request.user
        order = Order.objects.get(order_id=order_id)
        have_access = False
        if order.taker == user:
            have_access = True
        else:
            try:

ChatMember.objects.get(chat=order.advertisement.chat,    user=user,
chat_member_status=1)
                have_access = True
            except:
                return     Response({'msg':     'No     access'},
status=status.HTTP_404_NOT_FOUND)

        if have_access:
            serializer = OrderDetailSerializer(order)
            return
Response(serializer.data,status=status.HTTP_200_OK)


@api_view(['POST'])
@permission_classes([AllowAny])
def handel_payment_tron_callback(request):
```

```
    client = TronPaymentService()

    x_key = request.headers.get("X-KEY")
    if x_key == client.x_key:
        client.handle_transaction(data=request.data)
        return Response({'msg': 'Transaction handled
successfully'}, status=status.HTTP_200_OK)
    else:
        return Response({'msg': 'No access'},
status=status.HTTP_401_UNAUTHORIZED)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_deposit(request):
    request.data['user'] = request.user.user_id
    request.data['type'] = 'deposit'
    request.data['status'] = 'pending'
    serializer =
CreateDepositTransactionSerializer(data=request.data)
    if serializer.is_valid():
        transaction = serializer.save()
        client = TronPaymentService()
        resp = client.create_transaction(transaction)
        return Response(resp, status=status.HTTP_201_CREATED)
    return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_withdrawal(request):
    request.data['user'] = request.user.user_id
    request.data['type'] = 'withdrawal'
    request.data['status'] = 'pending'
    serializer =
CreateWithdrawalTransactionSerializer(data=request.data)
```

```
if serializer.is_valid():
    transaction = serializer.save()
    client = TronPaymentService()
    resp = client.create_transaction(transaction)
    return Response(resp, status=status.HTTP_201_CREATED)
return Response(serializer.errors, status=statu
```

Додаток Б

Фрагмент програмних лістингів

bot/main.py

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__fil
e__), '..')))

import asyncio
import uvicorn
from aiogram import types, Dispatcher
from bot.handlers import dp  # Правильний імпорт
from bot.callback_queries import dp as callback_dp  #
Перейменований dp для уникнення конфлікту
from bot.loader import bot
from bot.server import app

async def start_fastapi():
    config = uvicorn.Config(app, host="0.0.0.0", port=8001)
    server = uvicorn.Server(config)
    await server.serve()

async def start_aiogram():
    await dp.start_polling(bot, skip_updates=True)

async def main():
    await asyncio.gather(
        start_fastapi(),
        start_aiogram(),
    )

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

Додаток В

Дерево проблем

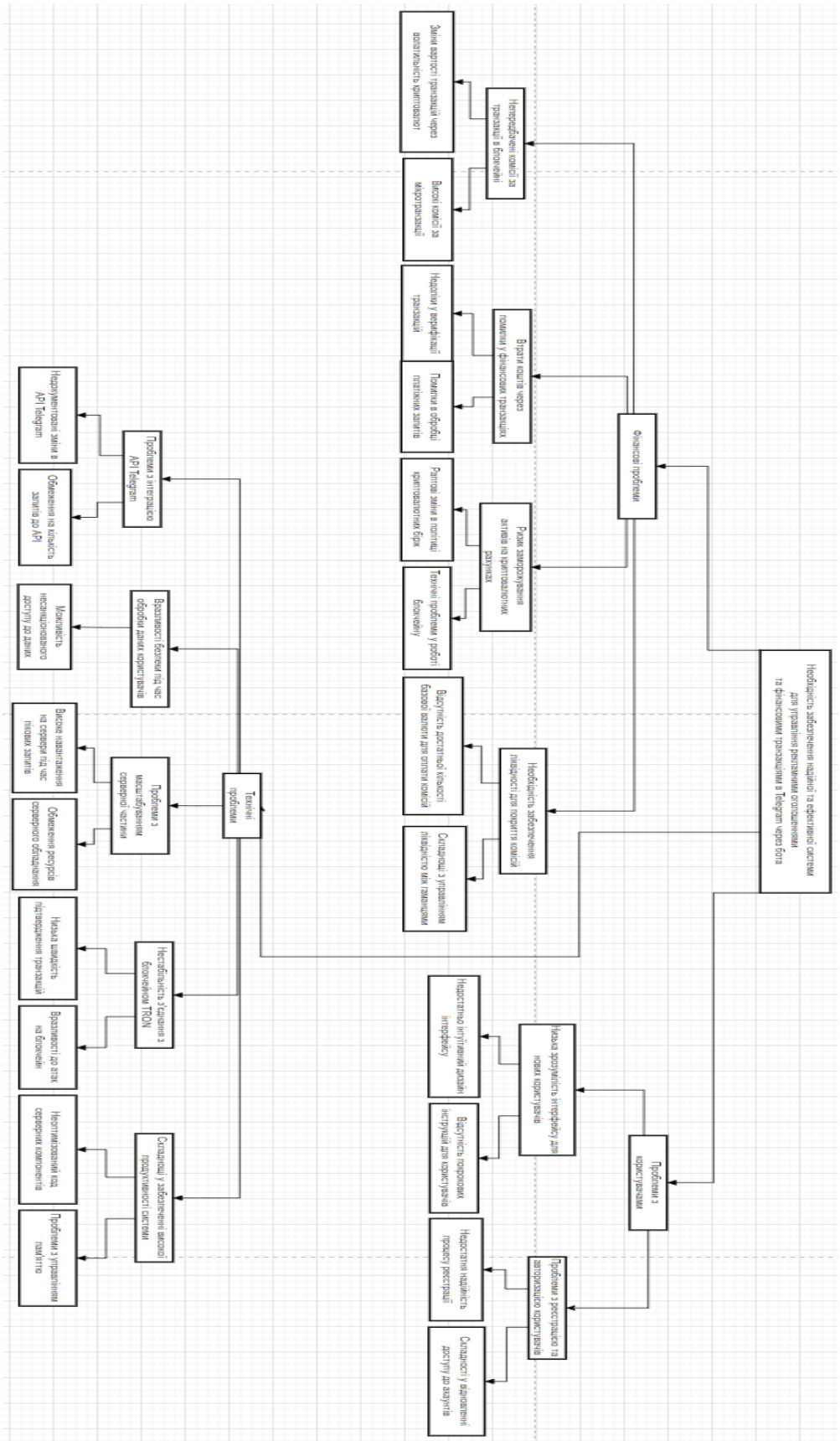

Рис. В.1. Дерево проблем