МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) автоматизації і комп'ютерних систем
Кафедра інформаційних технологій, штучного інтелекту і кібербезпеки

«До захисту в ЕК»	**«До захисту допущено»**
Директор інституту(декан факультету)	Завідувач кафедри
_____ Андрій Форсюк	_____ Сергій Грибков
(підпис) (ім'я та прізвище)	(підпис) (ім'я та прізвище)
«12» лютого 2024р.	«12» лютого 2024р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності 122 "Комп'ютерні науки"
(код і назва спеціальності)

освітньо-професійної програми Інформаційні управляючі системи та технології

на тему: Дослідження та розроблення чат-бота для підтримки діяльності ТОВ «Буковинська Надія»

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

Васильченко Ілля Борисович _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Струзік Владислав Анатолійович _____
(прізвище , ім'я та по батькові повністю) (підпис)

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

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

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

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

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

Здобувач _____
(підпис)

Київ — 2024 р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) _ автоматизації і комп'ютерних систем _____
Кафедра __інформаційних технологій, штучного інтелекту і кібербезпеки ___
Освітній ступінь _магістра_____
Спеціальність __122 "Комп'ютерні науки"_____
(код і назва)
Освітньо-професійна програма ___ Інформаційні управляючі системи _____
та технології_____
(назва)

ЗАТВЕРДЖУЮ
Завідувач
кафедри Сергій ГРИБКОВ
_____
«19» грудня 2023 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

_____Васильченка Іллі Борисовича_____
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розроблення чат-бота для підтримки діяльності ТОВ «Буковинська Надія»_____
керівник роботи Струзік Владислав Анатолійович, доцент, кандидат технічних наук,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «19» грудня 2023 р. № 1006-КС____
2. Строк подання здобувачем роботи: 22.01.2024_____
3. Вихідні дані до роботи: Інформація про методи інтеграції месенджерів, документація до програмних бібліотек, публіцистична та наукова література_____
_____
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Дослідження та аналіз методів інтеграції месенджерів, дослідження реалізації методів Long Polling та Webhook у відповідному API, розробка математичної оцінки часу відповіді для методів Long Polling та Webhook, реалізація програмної частини чат-бота та його оптимізація.
_____
5. Перелік графічного матеріалу:
Архітектура роботи телеграм-бота, приклади графічного інтерфейсу, схема бази даних_____

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	доц. к.т.н.Струзік В.А.		
2	доц. к.т.н.Струзік В.А.		
3	доц. к.т.н.Струзік В.А.		

7. Дата видачі завдання: 19 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Опрацювання літератури та аналіз актуальності теми	19.12.2023	Виконано
2	Проведення попередніх аналізів та підготовка необхідних ресурсів для ефективного виконання дослідження методів інтеграції месенджерів.	26.12.2023	Виконано
3	Дослідження методів інтеграції месенджерів для чат-бота.	28.12.2023	Виконано
4	Розробка програмної частини чат-бота на мові програмування Python	07.01.2024	Виконано
5	Оформлення пояснювальної записки	20.01.2024	Виконано
6	Створення презентації	22.01.2024	Виконано
7	Оформлення автореферату	22.01.2024	Виконано

Здобувач _____ (підпис) Васильченко І. Б. (прізвище та ініціали)

Керівник роботи _____ (підпис) Струзік В. А. (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота "Дослідження та розроблення чат-бота для підтримки діяльності ТОВ «Буковинська Надія»" представляє дослідження та практичну реалізацію чат-бота для підприємства "Буковинська Надія". Робота виконана студентом Васильченко І. Б. і складається з 112 сторінок, 3 розділів, 31 рисунків, 1 таблиця, 4 додатки та 11 літературних джерел.

Кваліфікаційна робота "Дослідження та розроблення чат-бота для підтримки діяльності ТОВ «Буковинська Надія»" включає в себе вивчення та аналіз методів інтеграції месенджерів, дослідження реалізації методів Long Polling та Webhook у API обраного месенджера, а також аналіз впливу різних чинників на продуктивність чат-бота. У результаті цього дослідження була розроблена математична оцінка часу відповіді для обох методів.

Крім теоретичного аспекту, була виконана реалізація програмної частини чат-бота на мові програмування Python, який призначений для підтримки діяльності ТОВ «Буковинська Надія».

Ключові слова: ЧАТ-БОТ, ОПТИМІЗАЦІЯ ВЗАЄМОДІЇ, МАТЕМАТИЧНА ОЦІНКА, МЕСЕНДЖЕР, API, TELEGRAM, TELEGRAM API, LONG POLLING, WEBHOOK.

SUMMARY

The qualification work "Investigation and development of a chatbot to support the activities of “Bukovynska Nadiya” explores and practically implements a chatbot for the enterprise “Bukovynska Nadiya”. The work was carried out by the student Vasylchenko Illia and consists of 112 pages, 3 sections, 31 figures, 1 tables, 4 appendices, and 11 literary sources.

The qualification work “Investigation and development of a chatbot for support of the activity of “Bukovynska Nadiya” include the examination and analysis of messenger integration methods, investigation of the implementation of Long Polling and Webhook methods in the respective messenger API, as well as an analysis of the impact of various factors on the performance of the chatbot. As a result of this research, a mathematical assessment of response time for both methods was developed.

In addition to the theoretical aspect, the implementation of the software part of the chatbot was carried out in the Python programming language. This allowed for the creation of an efficient and stable system designed to support the activities of “Bukovynska Nadiya”.

Keywords: CHATBOT, INTERACTION OPTIMIZATION, MATHEMATICAL ASSESSMENT, MESSENGER, API, TELEGRAM, TELEGRAM API, LONG POLLING, WEBHOOK.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний бізнес, особливо у сфері обслуговування та реалізації товарів, стикається зі зростанням обсягів інтеракції з клієнтами через різноманітні онлайн-платформи. Завдяки цьому відкриваються безліч можливостей для розвитку, але разом з тим виникають проблеми управління та ефективної обробки великого потоку інформації. Серед цих викликів ключовою є проблема розподілу та оптимізації обробки повідомлень від клієнтів.

Підприємство "Буковинська Надія" стикається із завданням ефективного управління потоком повідомлень від клієнтів, що надходять з різних джерел продажу. З надлишковим обсягом повідомлень може виникати перенапруження ресурсів персоналу, збільшуватися час відповіді на запитання клієнтів та збільшувати ймовірність втрати важливих запитів.

Актуальність теми полягає в тому, що розробка та впровадження кросплатформеного чат-бота для оптимізації спілкування з клієнтами та інтеграції з існуючою CRM-системою може виявитися ключовим інструментом для вирішення наявних проблем. До основних задач чат-боту належать: автоматизований розподіл та обробка повідомлень; спрощення взаємодії з клієнтами з метою підвищення ефективності обслуговування. Актуальність дослідження полягає не лише в забезпеченні конкретного підприємства засобами для вирішення його конкретних завдань, але й у розширенні загального розуміння можливостей та переваг використання чат-ботів у бізнес-середовищі.

Зв'язок роботи з науковими програмами, планами, темами кафедри, університету. Наукова робота виконувалась згідно з науково-дослідною роботою на кафедрі інформаційних технологій, штучного інтелекту і кібербезпеки «Дослідження та використання сучасних інформаційних технологій для виконання функцій та завдань виробничого і організаційного управління підприємств харчової

галузі» (0120U105386 2020–2025 рр.) Національного університету харчових технологій.

Мета дослідження: обґрунтування та реалізація підвищення ефективності обслуговування клієнтів з використанням месенджерів.

***Завдання дослідження*:**

- провести аналіз різних методів інтеграції месенджерів для чат-бота;
- дослідити реалізацію методів Long Polling та Webhook у відповідному API Telegram та визначити переваги та недоліки кожного методу в контексті роботи з месенджером;
- визначити та проаналізувати фактори, що впливають на часові показники роботи чат-бота;
- провести аналіз результатів експериментальних досліджень ефективності роботи чат-бота.

Об'єктом дослідження є процес інтеграції месенджерів за допомогою чат ботів.

Предметом дослідження є методи інтеграції чат-бота, зокрема Long Polling та WebHook.

Для виконання поставлених завдань будуть використані такі *методи дослідження*:

1. емпіричний метод для дослідження сучасних методів та підходів побудови та інтеграції чат-бот платформ;
2. метод аналізу та синтезу для визначення методів інтеграції і впровадження чат-боту;
3. метод аналізу та порівняння методів взаємодії (Long Polling та WebHook) з месенджерським API;
4. метод природного експерименту при збору та накопичення інформації про використання кожного методу інтеграції чат-бота;

Наукова новизна є формування рекомендацій використання методів Long Polling та WebHook для взаємодії з месенджерським API в контексті розробки телеграм-бота.

Практичне значення. Створений чат-боту, що інтегрується з існуючою інформаційною системою ТОВ "Буковинська Надія" та забезпечує підвищення ефективності роботи з клієнтами. Впровадження розроблених стратегій взаємодії з API дозволить підприємству ефективно використовувати та підтримувати чат-бот, забезпечуючи стабільну та продуктивну роботу системи.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕСЕНДЖЕРІВ ТА ЇХ РОЛЬ ПРИ РОБОТІ З КЛІЄНТАМИ ПІДПРИЄМСТВА

1.1. Основна роль месенджерів

Месенджери - це програми або платформи для обміну текстовими повідомленнями, мультимедійним контентом та іншими елементами спілкування в режимі реального часу. Вони широко використовуються для спілкування між користувачами через Інтернет [7].

Першим популярним месенджером була "аська" — ICQ — та її мобільний аналог Jimm. Вона з'явилася у 1996 році і користувалася популярністю до 2005 року. З появою Skype та завдяки підтримці голосових і відеодзвінків він лідирував на ринку до 2012 року [1].

Для українців месенджери тепер не тільки сервіс для обміну повідомленнями, але й повноцінні джерела інформації. Так, за майже два роки великої війни українці стали вчетверо частіше читати новини саме з каналів у Telegram та Viber [1].

Месенджери стали невід'ємною частиною сучасного світу, перетворивши спосіб, яким люди спілкуються та обмінюються інформацією. Роль месенджерів у сучасному суспільстві вкрай важлива, оскільки вони відіграють центральну роль у забезпеченні миттєвого та ефективного зв'язку між користувачами.

До ключових аспектів ролі месенджерів в сучасному світі можна віднести:

- Миттєвість та глобальність. Месенджери надають можливість миттєвого обміну повідомленнями та мультимедійним контентом, зручніше і ефективніше, ніж традиційні форми комунікації. Це особливо важливо в умовах глобалізації, коли користувачі можуть перебувати в різних куточках світу.
- Групові чати та канали. Месенджери дозволяють створювати групові чати та канали для колективної комунікації та обміну інформацією. Це особливо корисно для організацій, команд, а також для розповсюдження новин та контенту.

- Чат-боти для автоматизації. Впровадження чат-ботів дозволяє автоматизувати взаємодію з користувачами, забезпечуючи широкий спектр сервісів від отримання інформації до здійснення покупок чи використання різноманітних послуг.
- Приватність та безпека. Месенджери надають користувачам можливість захисту особистої інформації та приватності, особливо за допомогою шифрування end-to-end, що стає важливою складовою в умовах зростання цифрового спілкування.
- Інновації та функціональність. Месенджери постійно вдосконалюються, додаючи нові функції та інноваційні можливості, такі як голосові та відеовиклики, розширений функціонал чат-ботів, інтеграція з іншими додатками та платформами.

1.2. Загальний огляд Telegram

Telegram – це месенджер, створений братами Павлом та Ніколою Дуровими, який вперше був представлений громадськості у 2013 році. Павло Дуров розробив Telegram з метою створення безпечного та конфіденційного інструменту зв'язку. [9]

Вже в перший рік свого існування завоював популярність завдяки своєму шифруванню та зручному інтерфейсу. Протягом наступних років Telegram регулярно оновлювався, додаючи нові функції та поліпшення для користувачів.

Засновники активно розвивали платформу, анонсуючи нові функції, такі як групові чати, канали для розповсюдження контенту, та удосконалюючи систему безпеки. Швидкість розробки та відповідь на вимоги користувачів допомогли Telegram стати одним із лідерів у світі месенджерів, з великою активною спільнотою користувачів по всьому світу.

Протягом років свого існування, Telegram став одним із найпопулярніших месенджерів у світі завдяки своїм унікальним функціональним можливостям та

акценту на конфіденційності користувачів. Платформа відрізняється від інших месенджерів шифруванням end-to-end, яке забезпечує безпеку та непроникність особистої інформації. [9]

Telegram надає користувачам широкий спектр можливостей, що забезпечують зручність та ефективність в комунікації. Деякі з його основних характеристик є [4]:

- обмін текстовими повідомлення та можливість надсилання мультимедійних файлів;
- створення груп та каналів для колективної комунікації або розповсюдження контенту;
- використання чат-ботів для автоматизації обробки повідомлень та надання різноманітних сервісів;
- шифрування end-to-end для забезпечення конфіденційності та захисту від несанкціонованого доступу;
- використання технології "Secret Chats" для створення повідомлень, які автоматично самознищуються після певного часу.

Telegram входить в конкурентну сферу месенджерів, і порівняння з іншими популярними платформами. Нижче подано порівняльний огляд Telegram з деякими іншими відомими месенджерами [7, 10]:

WhatsApp

- Шифрування. WhatsApp також використовує шифрування end-to-end, що забезпечує високий рівень безпеки.
- Публічні групи. Telegram надає більше можливостей для створення публічних груп та каналів, які можуть містити до 200 000 учасників.
- Робота з чат-ботами. Telegram відомий своїми широкими можливостями чат-ботів, що робить його привабливим для автоматизації завдань.

Signal

- Приватність. Signal отримав розповсюдженість своїм фокусом на приватності та анонімності користувачів, подібно до Telegram.
- Безпека. Обидва месенджери високо цінуються за забезпечення користувачів безпекою та конфіденційністю, але Signal вважається ще більш консервативним у цьому плані.

Facebook Messenger

- Інтеграція з соціальними мережами. Facebook Messenger має сильну інтеграцію з соціальною мережею Facebook, що дозволяє зручно обмінюватися контентом та інформацією.
- Боти. Обидва месенджери підтримують чат-ботів, але Telegram відомий більшою свободою та широкими можливостями їх використання.

Viber

- Розширений функціонал: Telegram вигідно виділяється своїм більш розширеним функціоналом, таким як групи, канали, та обробка мультимедійного контенту.
- Безпека: Обидва месенджери надають безпеку за допомогою шифрування, але Telegram частіше акцентується на цьому аспекті.

1.3. Чат-боти

Чат-бот – це програмне забезпечення, яке взаємодіє з користувачами через текстові чи голосові повідомлення. Його основна мета полягає в тому, щоб автоматизувати комунікацію із користувачем та надавати відповіді на його запитання або виконувати конкретні завдання.

Є два основних типи чат-ботів:

1) Правила та ключові слова (Rule-Based) - такі чат-боти працюють на основі заданих правил і визначених ключових слів. Вони реагують на конкретні запитання чи команди, ініційовані користувачем. Цей підхід може бути ефективним для простих завдань, але вони обмежені в розумінні контексту та не ефективні у вирішенні складних завдань.

2) Штучний інтелект (AI-Based) - такі чат-боти використовують алгоритми машинного навчання та обробки природної мови для аналізу та розуміння повідомлень користувачів. Вони можуть вдосконалювати свої відповіді в залежності від взаємодії з користувачем та набутого досвіду.

Основні функції чат-ботів:

- Автоматизація обслуговування клієнтів. Чат-боти можуть допомогти в розв'язанні запитань клієнтів, надавати інформацію про продукти або послуги, та вирішувати невеликі проблеми безпосередньо в чаті.
- Здійснення покупок та бронювань. Чат-боти можуть функціонувати як віртуальні асистенти для виконання операцій, таких як замовлення товарів, бронювання квитків або здійснення резервацій.
- Навчання та консультування. У чат-ботах, призначених для освітніх задач, є можливість отримати відповіді на питання, вивчати новий матеріал, або отримувати консультації за певною темою.
- Розваги та інтерактивні ігри. Чат-боти можуть стати віртуальними героями інтерактивних ігор, надаючи користувачам розвагу та взаємодію.
- Послуги у сфері здоров'я. Чат-боти можуть служити важливими засобами для надання медичної інформації, нагадувань про ліки, або навіть для проведення деяких медичних консультацій.
- Оперативна підтримка та інформування. Чат-боти дозволяють компаніям надавати оперативну підтримку клієнтам, відповідати на загальні питання, та надавати актуальну інформацію.

Чат-боти в Telegram грають ключову роль у розширенні функціоналу месенджера та наданні різноманітних сервісів користувачам. Вони можуть бути використані для різних завдань і надають можливості автоматизації, розваг, а також отримання інформації. Telegram API дозволяє розробникам створювати чат-ботів, які можуть взаємодіяти з користувачами та виконувати різноманітні функції. Широкі можливості чат-ботів роблять їх популярним інструментом для взаємодії з аудиторією у Telegram.

1.4 Дослідження можливостей Telegram API

Telegram API є невід'ємною частиною месенджера, яка надає розробникам широкий спектр можливостей для інтеграції та взаємодії з Telegram. Заснований на відкритих стандартах і протоколах, цей інтерфейс програмування дозволяє створювати різноманітні додатки, боти та сервіси, розширюючи можливості для взаємодії з різними користувачами. [8]

Telegram API забезпечує доступ до ключових можливостей месенджера, таких як надсилання повідомлень, адміністрування чатів, редагування профілю користувача, обмін медіафайлами, інтеграція зі сторонніми сервісами тощо. Відомий своєю простотою та ефективністю, Telegram API є потужним інструментом для розробників, що дозволяє їм використовувати можливості месенджера. [3]

Telegram API спроектовано на основі простої та ефективної архітектури, яка дозволяє розробникам легко взаємодіяти з месенджером. Він використовує RESTful API та протоколи, такі як HTTP та HTTPS, для передачі даних між клієнтами та серверами Telegram.

Захист конфіденційності - це основний принцип Telegram API полягає в забезпеченні високого рівня захисту конфіденційності даних користувачів. Це досягається за допомогою шифрування та інших технологій безпеки.

Telegram API розроблено так, щоб бути простим та доступним для розробників будь-якого рівня кваліфікації. Це допомагає швидко створювати та розгортати додатки без зайвих труднощів.

Регулярні оновлення та підтримка Telegram API забезпечує вдосконалення функціональності та виправлення можливих помилок. Підтримка розробників є ключовою складовою, що дозволяє ефективно використовувати API в реальних проектах.

Взаємодія з ботом в Telegram відбувається через ряд методів та команд, які дозволяють користуватися функціоналом чат-бота. Також, боти можуть взаємодіяти з користувачами, надсилаючи повідомлення та використовуючи різні елементи, такі як клавіатури, стікери та мультимедійний контент.

Чат-боти в Telegram використовують різні методи для обробки текстових повідомлень від користувачів. Основні етапи обробки текстових повідомлень включають:

- Отримання повідомлення. Бот отримує текстове повідомлення від користувача через Telegram API.
- Аналіз тексту. Бот може використовувати алгоритми обробки природної мови (NLP) для аналізу тексту та визначення його сутності та інтенції.
- Розпізнавання команд. Якщо текст містить команду (наприклад, "/start" або "/help"), бот може виконати відповідну дію.
- Формування відповіді. Бот генерує текстову відповідь в залежності від змісту повідомлення та виконаних аналізів.
- Відправлення відповіді користувачеві. Згенерована відповідь відправляється користувачеві через Telegram API.

Основні етапи обробки повідомлення з зображеннями:

- Отримання зображення від користувача. Бот отримує зображення від користувача через Telegram API.

- Аналіз зображення (опціонально). Бот може використовувати алгоритми обробки зображень для аналізу контенту, наприклад, розпізнавання об'єктів чи визначення особливостей.
- Відправлення відповіді з зображенням. Бот може відповісти на отримане зображення, надіславши користувачеві власне зображення чи текстову відповідь.
- Генерація стікера (опціонально). На основі отриманого зображення бот може створити стікер, що може бути використаний для подальшої взаємодії.

Основні етапи обробки повідомлення з аудіо:

- Отримання аудіофайлу від користувача. Бот отримує аудіофайл від користувача через Telegram API.
- Аналіз аудіо (опціонально). Бот може використовувати алгоритми аудіоаналізу для визначення особливостей, таких як розпізнавання мови або аналіз частот.
- Відправлення відповіді з аудіофайлом. Бот може відповісти, надіславши користувачеві аудіофайл, що містить голосове повідомлення або музичний контент.

Основні етапи обробки повідомлення з відео:

- Отримання відеофайлу від користувача. Бот отримує відеофайл від користувача через Telegram API.
- Аналіз відео (опціонально). Бот може використовувати алгоритми обробки відео для визначення особливостей, таких як розпізнавання об'єктів або аналіз динаміки.
- Відправлення відповіді з відеофайлом. Бот може відповісти, надіславши користувачеві відеофайл або відповідне повідомлення.

Забезпечення безпеки даних у Telegram чат-ботах включає в себе використання шифрування передачі даних, безпечних каналів обміну та обмеження доступу до API-ключів. Особисті дані зберігаються мінімально, з чіткою політикою

конфіденційності, що визначає правила використання та захисту інформації. Крім того, забезпечується можливість видалення особистих даних користувачів за їхнім бажанням. Важливим є також стале вдосконалення заходів безпеки для ефективного захисту конфіденційної інформації та підтримки високого рівня довіри користувачів.

1.5 Постановка задачі на дослідження

В сучасному світі месенджери стають не лише інструментами для особистого спілкування, але й потужними засобами для ведення бізнесу та взаємодії з клієнтами.

Високий рівень конкуренції на ринку вимагає від підприємств постійного вдосконалення методів комунікації. У цьому контексті, впровадження чат-ботів стає ключовим рішенням, спрямованим на оптимізацію взаємодії із клієнтами та підвищення рівня їх обслуговування.

Метою дослідження є обґрунтування та реалізація підвищення ефективності обслуговування клієнтів з використанням месенджерів, а саме чат-ботів. Враховуючи вищезазначене завданням на дослідження є наступне:

- провести аналіз різних методів інтеграції месенджерів для чат-бота;
- дослідити реалізацію методів Long Polling та Webhook у відповідному API Telegram та визначити переваги та недоліки кожного методу в контексті роботи з месенджером;
- визначити та проаналізувати фактори, що впливають на часові показники роботи чат-бота;
- провести аналіз результатів експериментальних досліджень ефективності роботи чат-бота.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРОБЛЕННЯ ЧАТ-БОТА

2.1 Огляд мов програмування для реалізації чат-бота

При реалізації чат-бота для Telegram, важливим аспектом є вибір мови програмування та відповідних бібліотек для взаємодії з Telegram API. Нижче подано огляд популярних мов та бібліотек для створення чат-ботів.

Python є однією з найпопулярніших мов програмування для розробки чат-ботів для Telegram. Зручний синтаксис, багаті бібліотеки та спеціалізовані фреймворки, такі як aiogram чи python-telegram-bot, активна спільнота роблять Python популярним вибором для швидкої реалізації функцій ботів.

Бібліотеки Python для створення чат-ботів [3]:

- AIOGram - це асинхронна бібліотека для створення Telegram ботів у Python. Вона використовує asyncio та забезпечує швидку та ефективну обробку подій та відповідей в чат-боті. Особливості: асинхронна реалізація для оптимізації роботи бота; підтримка всіх основних можливостей Telegram API; зручний інтерфейс для роботи з асинхронним кодом.
- pyTelegramBotAPI - це проста та легка у використанні бібліотека для розробки чат-ботів у Python. Вона забезпечує простий інтерфейс для взаємодії з Telegram API та дозволяє розробникам легко відповідати на події та обробляти повідомлення. Особливості: простота використання; підтримка різноманітних функцій Telegram API; локальна обробка подій та відповідей.йним контентом.
- python-telegram - це проста та легка бібліотека для створення Telegram ботів. Вона надає базовий функціонал для обробки повідомлень та подій. Особливості: легка у використанні та розумінні; базовий функціонал для обробки повідомлень та команд.

- python-telegram-bot - це офіційна бібліотека Telegram для Python. Вона має обширний функціонал та надає розробникам доступ до всіх можливостей Telegram API, таких як робота з повідомленнями, клавіатурами, фотографіями тощо. Особливості: підтримка всіх можливостей Telegram API; асинхронні функції для ефективної роботи з подіями; розширені можливості для роботи з клавіатурами та іншими елементами.

JavaScript/TypeScript є популярними мовами для створення застосунків, включаючи чат-боти. З використанням бібліотеки Telegraf або Botpress можна легко розробляти Telegram ботів з використанням JavaScript або TypeScript.

Бібліотеки JavaScript для створення чат-ботів [3]:

- Telegraf - це потужна та зручна бібліотека для Node.js, призначена для розробки чат-ботів у Telegram. Вона надає простий інтерфейс та підтримує багатофункціональність, включаючи роботу з командами, клавіатурами та іншими функціями Telegram API.
- Botpress - є відкритою бібліотекою для створення чат-ботів, яка базується на Node.js та використовується для розробки різних видів ботів, включаючи ті, що працюють у Telegram.
- Telegram.js - це бібліотека для розробки чат-ботів у Telegram, вона надає основні засоби для взаємодії з Telegram API та дозволяє розробникам створювати прості та ефективні боти для цієї платформи.

Java є розповсюдженою мовою програмування. Для неї існує бібліотека TelegramBots API, що дозволяє зручно створювати чат-боти на Java з високою стабільністю та продуктивністю. Вона має широкий функціонал та забезпечує простий інтерфейс для взаємодії з Telegram API [3].

Java відома своєю високою рівнем безпеки. Це важливо при роботі з особистими даними користувачів, що є актуальним аспектом для чат-ботів.

Мова має велику та активну спільноту розробників, а також докладну офіційну документацію, що полегшує вивчення та вирішення питань.

Ruby разом з фреймворком telegram-bot-ruby надає простий та зрозумілий спосіб для розробки Telegram ботів. Ruby сприяє швидкому вирішенню завдань. Бібліотека telegram-bot-ruby використовується для реалізації чат-ботів у Telegram на Ruby. Вона надає простий та зрозумілий інтерфейс для взаємодії з Telegram API та включає в себе різні функції, такі як обробка повідомлень, клавіатури, команди тощо [3].

Мова програмування **C#** може використовуватися для створення чат-ботів за допомогою бібліотеки Telegram.Bot. Вона надає зручний інтерфейс для взаємодії з Telegram API та підтримує різні функціональності, такі як робота з повідомленнями, клавіатурами, зображеннями тощо. Інтеграція з .NET. C# входить в склад платформи .NET, що робить його доступним для розробки веб-застосунків та сервісів, таких як чат-боти. Розробники можуть використовувати різноманітні функції та бібліотеки .NET для полегшення розробки [3].

2.2 Огляд методів реалізації чат-ботів

Існує 2 популярні методи реалізації чат-ботів, а саме Long Polling та WebHook.

Long Polling є одним із методів організації взаємодії між сервером Telegram та ботом. Цей метод базується на тривалому очікуванні (polling) відповідей від сервера [2].

Long Polling - це простий та ефективний метод для створення чат-ботів, особливо якщо бот повинен реагувати на події миттєво та при цьому бути легким у реалізації та підтримці.

Основні принципи роботи Long Polling:

- клієнт (бот) відправляє запит на сервер Telegram і очікує відповідь;
- сервер зберігає запит, поки не з'являться нові дані або до завершення тайм-ауту;

- якщо є нові події (повідомлення, оновлення), сервер відправляє їх клієнту;

Основні переваги Long Polling [5, 6]:

- легко реалізовується, особливо з використанням бібліотек, таких як pyTelegramBotAPI;
- забезпечує стабільну роботу бота, оскільки він використовує надійний зв'язок із сервером Telegram.

Основні недоліки Long Polling [5, 6]:

- для отримання нових повідомлень бот повинен періодично відправляти запити, що може викликати затримки та збільшення завантаження сервера;
- якщо немає нових подій, запит може очікувати відповіді до завершення тайм-ауту, що також може вплинути на час відгуку бота.

Рекомендації для використання:

- Long Polling є добрим вибором для створення простих та надійних чат-ботів.
- Варто розглядати опції зберігання стану бота для уникнення втрати даних при перезапуску.

Бібліотека pyTelegramBotAPI дозволяє легко реалізувати Long Polling з допомогою методу bot.polling(), який автоматично обробляє нові події та повідомлення.

Використання WebHook - це ще один метод реалізації взаємодії між ботом та сервером Telegram [2].

Принцип роботи наступний:

- бот реєструє WebHook, надаючи URL-адресу, на яку сервер Telegram буде відправляти дані;
- коли відбувається подія (нове повідомлення або інше оновлення), сервер Telegram надсилає POST-запит на вказану URL-адресу бота;

- бот отримує цей POST-запит, обробляє отримані дані та реагує відповідним чином.

До основних переваг можливо віднести [5, 6]:

- Швидкість відповіді. Використання WebHook дозволяє отримувати миттєві повідомлення, оскільки сервер Telegram активно повідомляє бота про події.
- Зменшення навантаження. Запити надсилаються лише при виникненні подій, що дозволяє зменшити навантаження на сервер бота.

До основних недоліків є можливо віднести [5, 6]:

- Реалізація WebHook може бути складнішою в порівнянні з Long Polling, оскільки вимагає правильного налаштування веб-сервера та наявності SSL-сертифікату для забезпечення безпеки.
- Потрібен веб-сервер, який буде отримувати вхідні запити до бота.

Бібліотека pyTelegramBotAPI надає можливість використання WebHook за допомогою методу bot.setWebhook(url), який налаштовує WebHook та повідомляє сервер Telegram про адресу для надсилання оновлень.

Використання WebHook рекомендується для більш потужних та продуктивних чат-ботів, які можуть надсилати багато повідомлень або обробляти складні запити.

2.3. Порівняння та обґрунтування вибору методів

Однією з ключових задач при розробці чат-ботів є вибір методу комунікації між сервером бота та сервером месенджера. В загальному випадку постає вибір між методами Long Polling та WebHook. Для кращого розуміння переваг і недоліків кожного методу, ми пропонуємо порівняльний аналіз за ключовими критеріями у Таблиці №1.

Таблиця 1. Порівняння методів Long Polling та WebHook

Критерій	**Long Polling**	**WebHook**
Швидкість відповіді	Може бути деяка затримка у відповіді бота, оскільки він регулярно опитує сервер.	Миттєва відповідь на події, оскільки сервер Telegram активно надсилає оновлення на вказану URL-адресу.
Завантаження сервера	Може створювати більше завантаження на сервер, оскільки бот регулярно надсилає запити.	Завантаження зменшується, оскільки сервер Telegram ініціює зв'язок лише при виникненні подій.
Складність налаштувань	Легший у встановленні та реалізації.	Складніше встановлення, оскільки потрібно правильно налаштувати веб-сервер та SSL-сертифікат.
Надійність	Надійний метод для створення простих та стабільних ботів.	Також надійний, але вимагає ретельної конфігурації.
Вимоги до інфраструктури	Здебільшого не вимагає спеціалізованих інфраструктурних рішень.	Потрібен наявний веб-сервер, що приймає запити до бота.
Застосування	Добре підходить для простих та менш навантажених ботів.	Рекомендований для більш потужних ботів, які можуть обробляти багато запитів та подій.

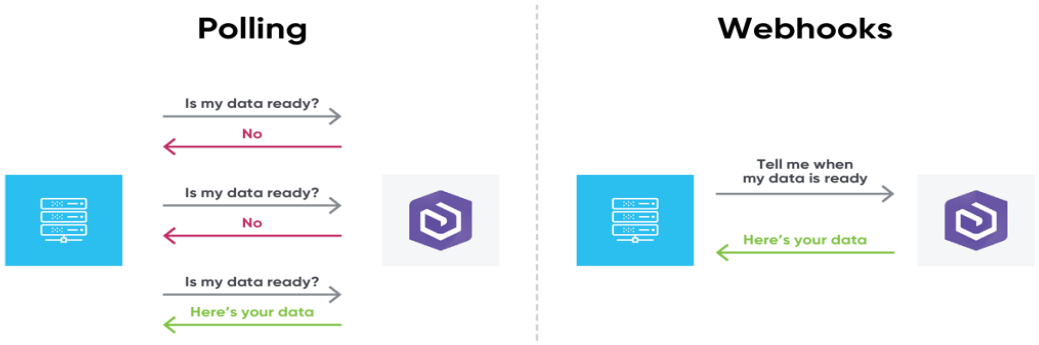

Рисунок 2.1 - Приклад роботи методів Long Polling та методу WebHook та їх порівняння [2]

2.4. Математична оцінка часу відповіді для методів Long Polling та WebHook

У сучасному світі, де ефективність та швидкість є ключовими показниками успішності багатьох програмних рішень, вибір методу взаємодії з месенджерським API стає важливим етапом розробки. Два основних методи, які використовуються для отримання відповідей від API, це Long Polling та WebHook.

Метод Long Polling:

Нехай T[LP] - час відправки клієнтом запиту, T[WAIT_LP] - час очікування відповіді. K[LP] - коефіцієнт, який враховує додаткові часові затрати для Long Polling,

Час відповіді оцінюються:

$$T[RESPONSE\_LP] = T[LP] + T[WAIT\_LP] + K[LP]. \quad (1)$$

Де T[RESPONSE_LP] - час відповіді для Long Polling.

Метод WebHook

Нехай T[WH] - час, коли сервер бота отримує вхідний HTTP-запит, T[PROCESS_WH] - час обробки та відправки відповіді. K[WH] - коефіцієнт, який враховує додаткові часові затрати для WebHook.

Час відповіді розраховується:

$$T[RESPONSE\_WH] = T[WH] + T[PROCESS\_WH] + K[WH] \quad (2)$$

Де T[RESPONSE_WH] - час відповіді для WebHook.

Умова переваги WebHook над Long Polling:

$$T[WH] + T[PROCESS\_WH] + K[LP] < T[LP] + T[WAIT\_LP] + K[WH] \quad (3)$$

Декомпозиція процесів взаємодії чат-бота з сервером Telegram дозволяє визначити ключові параметри та часові витрати , що впливають на швидкість та ефективність роботи чат-бота.

Аналіз життєвого циклу методу Long Polling (LP) включає в себе:

- Час відправки запиту (T[LP])
- Час очікування відповіді (T[WAIT_LP])
- Додатковий час (K[LP]).

Аналіз життєвого циклу методу метод WebHook (WH) включає в себе:

- Час отримання HTTP-запиту (T[WH])
- Час обробки та відправки відповіді (T[PROCESS_WH])
- Додатковий час (K[WH]).

Математична оцінка дозволяє вирішити:

- Визначення часу відповіді для кожного методу.

- Оптимізація параметрів для покращення продуктивності чат-бота.
- Аналіз впливу різних чинників на часові показники роботи чат-бота.
- Об'єктивно визначити переваги та недоліки кожного методу, що є важливим для вибору оптимального способу взаємодії чат-бота з сервером Telegram

В ході проведення експерименту було виявлено наступні дані:

Метод Long Polling:

- T[LP] = 1сек (час відправки запиту)
- T[WAIT_LP] = 1.5сек (час очікування відповіді)
- K[LP] = 0.5сек (Додатковий час на відправку та отримання пустого запиту, якщо немає оновлень. Наявність повторних запитів або спроб в разі втрати з'єднання.)

Метод WebHook:

- T[WH] = 0.5сек (час отримання HTTP-запиту)
- T[PROCESS_WH] = 0.8сек (час обробки та відправки відповіді)
- K[WH] = 1.1сек (Додатковий час на обробку та відправку відповіді через веб-сервер. Часові затрати на відновлення повідомлення, оскільки існує можливість втрати підключення при великому навантаженні)

Розрахунок Часу Відповіді:

Метод Long Polling:

$$T[RESPONSE\_LP] = 1 + 1.5 + 0.5 = 3 \quad (4)$$

Метод WebHook:

$$T[PROCESS\_WH] = 0.5 + 0.8 + 1.1 = 2.4 \quad (5)$$

Переваги WebHook над Long Polling:

$$0.5 + 0.8 + 1.1 < 1 + 1.5 + 0.5 \quad (6)$$

Обираючи між Long Polling та використанням WebHook, розробник повинен враховувати конкретні потреби свого проекту. Long Polling підходить для простих чат-ботів, тоді як використання WebHook може бути більш доцільним в роботі з більш потужними та вимогливими застосунками.

У даному прикладі застосування математичної оцінки для порівняння часу відповіді методів Long Polling та WebHook в Telegram API надає нам наступні висновки: Метод WebHook має середній час відповіді (2.4), що відчутно менший, ніж у методі Long Polling (3). У випадку, коли важливий низький час відповіді та ефективність, вибір методу WebHook може бути більш обґрунтованим.

2.5. Висновок до розділу 2

У другому розділу цієї роботи було проведено глибоке дослідження різних аспектів розробки чат-ботів.

Огляд мов програмування для реалізації чат-бота надав чітке уявлення про наявність різних інструментів. Виявлено, що Python є високо популярною та ефективною мовою для цієї цілі.

Дослідження методів реалізації чат-ботів привів до висновку, що обидва методи мають свої переваги та недоліки. Long Polling відрізняється простотою реалізації, але може викликати певні затримки в обробці подій. З іншого боку WebHook надає більш ефективну реакцію на події, але вимагає додаткових налаштувань. Порівняння вибору методів дозволяє нам зрозуміти, що для конкретного проекту краще вибрати метод взаємодії, який відповідає його вимогам та можливостям.

У останньому підрозділі роботи було представлено математичну оцінку, що відображає життєвий цикл отримання та відправки запитів на Telegram API за допомогою методів Long Polling та WebHook. Для проведення експерименту та реалізації чат-бота в даній роботі було обрана мова Python разом із бібліотекою

pyTelegramBotAPI. Обрана комбінація визначається рядом факторів, які включають простоту використання, велику спільноту розробників та наявність команди розробників на підприємстві на якому буде впроваджений чат-бот.

Завдяки проведеному експерименту на основі математичної оцінки для реалізації чат-боту було обрано метод WebHook.

РОЗДІЛ 3. РОЗРОБКА ЧАТ-БОТА

3.1. Обґрунтування вибору Python

Вибір Python для розробки Telegram бота обґрунтовується кількома ключовими перевагами, які важливі для ефективної та успішної реалізації проекту.

По-перше, Python визначається своєю простотою та лаконічністю синтаксису, це важливо для швидкої розробки та підтримки чат-ботів.

По-друге, мова Python має широкий вибір бібліотек та фреймворків для різних завдань.

Крім того, на підприємстві вже є команда розробників, яка має досвід та навички в роботі з Python. Це сприяє ефективній реалізації проекту, оскільки команда готова використовувати всі переваги мови.

Узагальнюючи, вибір Python є логічним, оскільки ця мова програмування відповідає вимогам проекту та враховує наявність кваліфікованої команди розробників. Також буде використано бібліотеку pyTelegramBotAPI.

3.2. Етапи розробки та імплементації чат-бота

Створення нового бота в Telegram здійснювалося з використанням BotFather сервісу, який надає можливість реєстрації та отримання токену для бота. Цей токен буде ключовим елементом для ідентифікації та взаємодії з чат-ботом.

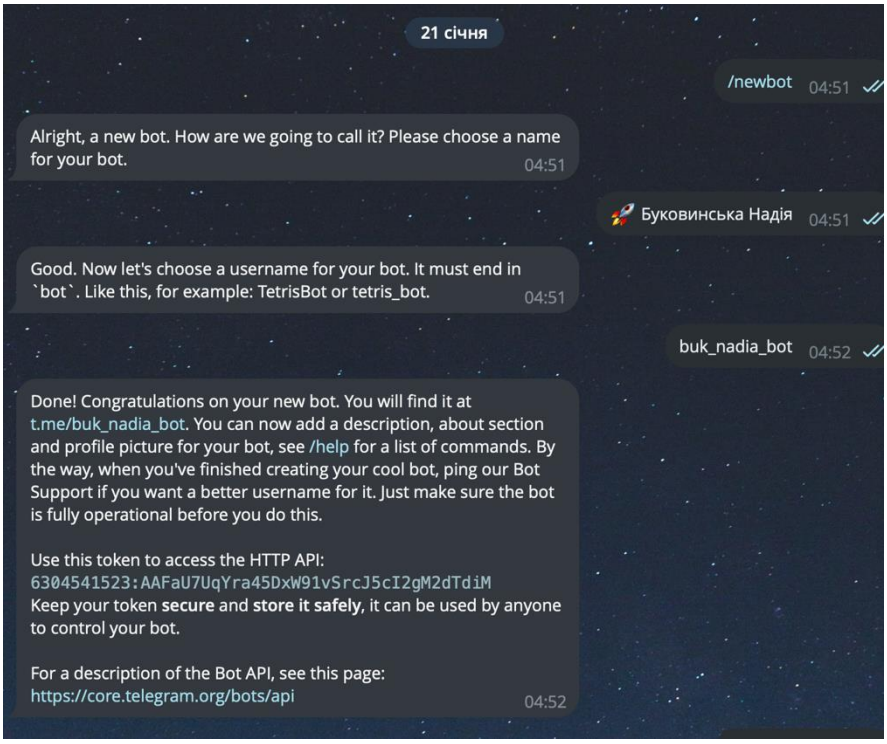

Рисунок 3.1 - Створення телеграм боту за допомогою BotFather

Структура та логіка роботи чат-бота обґрунтовується основними функціями, які бот повинен виконувати. Наприклад, вітання нових користувачів, надання інформації та обробку різних команд.

Спочатку необхідно створити модуль конфігурації, який в собі буде включати глобальні налаштування для роботи бота, параметри для зв’язку з базою даних, а також меню (рис. 3.2-3.4).

```
BOT_TOKEN = '6304541523:AAFaU7UqYra45DxW91vSrcJ5cI2gM2dTdiM'

WEBHOOK_HOST = '<ip/host where the bot is running>'
WEBHOOK_PORT = 8443  # 443, 80, 88 or 8443 (port need to be 'open')
WEBHOOK_LISTEN = '0.0.0.0'  # In some VPS you may need to put here the IP addr

WEBHOOK_SSL_CERT = './webhook_cert.pem'
WEBHOOK_SSL_PRIV = './webhook_pkey.pem'

WEBHOOK_URL_BASE = "https://%s:%s" % (WEBHOOK_HOST, WEBHOOK_PORT)
WEBHOOK_URL_PATH = "/%s/" % BOT_TOKEN


CHANNEL_LOG = -1001943003392
CHAT_NEW_MESSAGE = -1001943003392
```

Рисунок 3.2 - Конфігурація бота

```
USER_STATUS = {
    'start': 'Головне меню',
    'messages': '🔥 Мої переписки',
    'pin_chats': '⚡ Закреплені за мною чати',
    'guest_chats': '👨 Чати с неавторизованими клієнтами',
    'history_messages': '📁 Історія моїх переписок',
    'history_pin_chats': '📁 Історія моїх закріплених чатів',
    'history_guest_chats': '📁 Історія моїх чатів з гостями',
    'history': '📁 Історія всіх чатів',
    'send-message': 'send-message',
    'in-chat': 'in-chat',
    'activate-chat-message': 'activate-chat-message',
    'change_activate_sku': 'change_activate_sku',
}
```

Рисунок 3.3 - Конфігурація статусів для користувачів боту

global_settings.py database_settings.py

```
host = "localhost"
port = 3306
user = "user1"
password = "@6KU33]+^P_xrMCj&'"
db_name = "buk_nadia"
```

Рисунок 3.4 - Параметри підключення до бази даних

Програмний код, який задає конфігурацію для меню бота наведено в **додатку В.14.**

З метою забезпечення швидкого доступу до даних користувачів та їх адміністрування, був розроблений модуль управління користувачами бота. Код цього модуля представлено **в додатку В.1.**

Цей модуль, що включає в себе класи User та UserManager, створений для динамічного зберігання та оновлення інформації про кожного користувача у системі. Забезпечуючи ізольований підхід до обробки даних користувачів, модуль дозволяє взаємодіяти з базою даних та забезпечує високий рівень ефективності.

Основні функції модуля включають в себе створення користувачів, оновлення їх інформації, а також надання зручного доступу до цих даних для інших компонентів бота. Використовуючи принципи патерну Singleton, модуль забезпечує єдиний екземпляр класу UserManager, що дозволяє ефективно взаємодіяти з усіма користувачами з одного екземпляра класа.

Основні елементи цього модуля.

Клас User:

Атрибути:

- *telegram_id* - ідентифікатор користувача в Telegram;
- *status* - поточний статус користувача;
- *last_id_chat* - ідентифікатор останнього чату, з яким взаємодіяв користувач;
- *lvl* - рівень користувача;
- *db_id_user* - ідентифікатор користувача в базі даних;
- *info* - інформація про користувача, яка отримується з бази даних;

Методи:

- *__init__*(self, telegram_id) - конструктор класу, призначений для ініціалізації об'єкту користувача;
- *update*(self) - оновлює інформацію про користувача з бази даних;

- *__setattr__*(self, name, value) - метод, який викликається при встановленні атрибутів, і відповідає за оновлення цих даних у базі даних;

Клас UserManager (з реалізацією патерну Singleton):

Атрибути:

- *all_users* - словник, що містить всіх користувачів, де ключем є telegram_id;
- *last_updates* - словник для відстеження останнього оновлення інформації про користувачів;

Методи:

- *get_user*(self, telegram_id) -> User - повертає об'єкт користувача за його ідентифікатором, оновлюючи інформацію з бази даних при необхідності;
- *get_all_users*(self) -> dict - повертає словник усіх користувачів;

Модуль управління користувачами дозволяє легко отримувати та оновлювати інформацію про користувачів бота, забезпечуючи зручний інтерфейс для взаємодії з базою даних та управління їхніми атрибутами. Призначений для оптимізації роботи бота та забезпечення швидкого доступу до даних користувачів.

Модуль управління потоками, який складається з класу LockManager, розроблений для ефективного контролю за потоками в асинхронному середовищі. Код цього модуля представлено **в додатку В.2.** Основна мета цього модуля - забезпечити безпечну та синхронізовану роботу з різними потоками, що може бути особливо важливим в асинхронних програмах.

Ключові елементи цього модуля:

Клас LockManager:

Атрибути:

- *locked* - словник, який вказує, чи заблокований потік з певною назвою;

- *lock* - загальне блокування для управління доступом до словника та забезпечення його консистентності;

Методи:

- *get_lock*(self, name) -> threading.Lock - повертає об'єкт блокування для певного потоку за його ім'ям, якщо блокування для цього потоку вже існує, повертає його; інакше створює нове та повертає його.

Цей модуль відіграє важливу роль в синхронізації асинхронних процесів, де різні частини програми можуть виконуватися паралельно в окремих потоках. Клас LockManager вирішує проблему забезпечення потокобезпечності та управління блокуваннями для взаємовиключення між потоками, забезпечуючи безпечний доступ до спільних ресурсів.

Використання блокувань для синхронізації потоків важливо для уникнення конфліктів та непередбачених станів у виконанні асинхронних операцій. Оптимальне використання модуля управління потоками сприяє підтримці консистентності та стабільності в асинхронних додатках.

Розроблено модуль для виконання команд, пов'язаних із зміною рівнів доступу користувачів, адмініструванням груп та контролем за їхніми можливостями.

Опис коду модуля управління користувачами. Код відображено **в Додатку В.3**.

Функція обробки текстових повідомлень (рис. 3.5).

Умова Виклику:

- Викликається для повідомлень, які мають текстовий контент та відповідають командам /approve, /ban, /admin.

Функції:

- Перевіряє рівень доступу користувача.

- Змінює рівень доступу іншого користувача відповідно до команди /approve, /ban, /admin.
- Відправляє повідомлення про результат виконання операції.

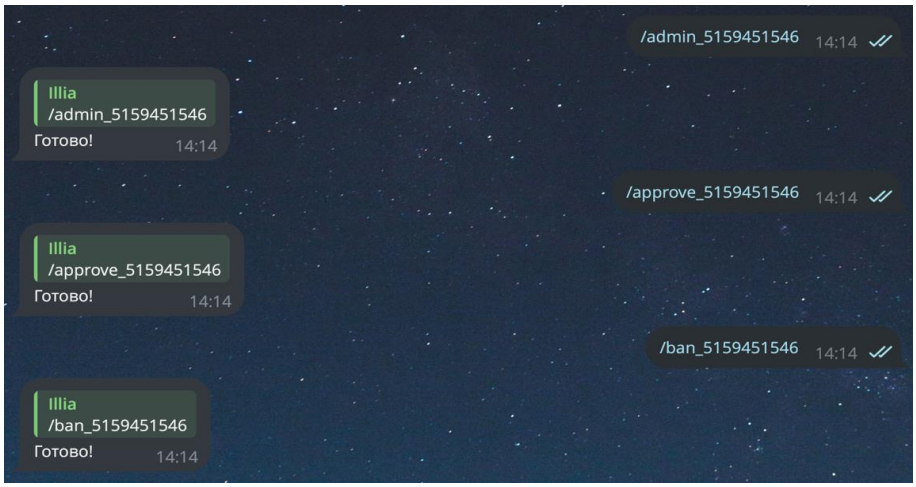

Рисунок 3.5 - Приклад управління користувачами, надання прав доступу

Функції обробки команд (/all_users, /clean) (рис. 3.6-3.7).

Умова Виклику:

- Викликається для команд /all_users та /clean.

Функції:

- Перевіряє рівень доступу користувача.
- Команда /all_users - відправляє інформацію про всіх користувачів.
- Команда /clean - запускає скрипт для видалення історії та відправляє повідомлення про успішний запуск скрипта.

Рисунок 3.6 - Приклад запуску скрипта для автоматизованого очищення чату.

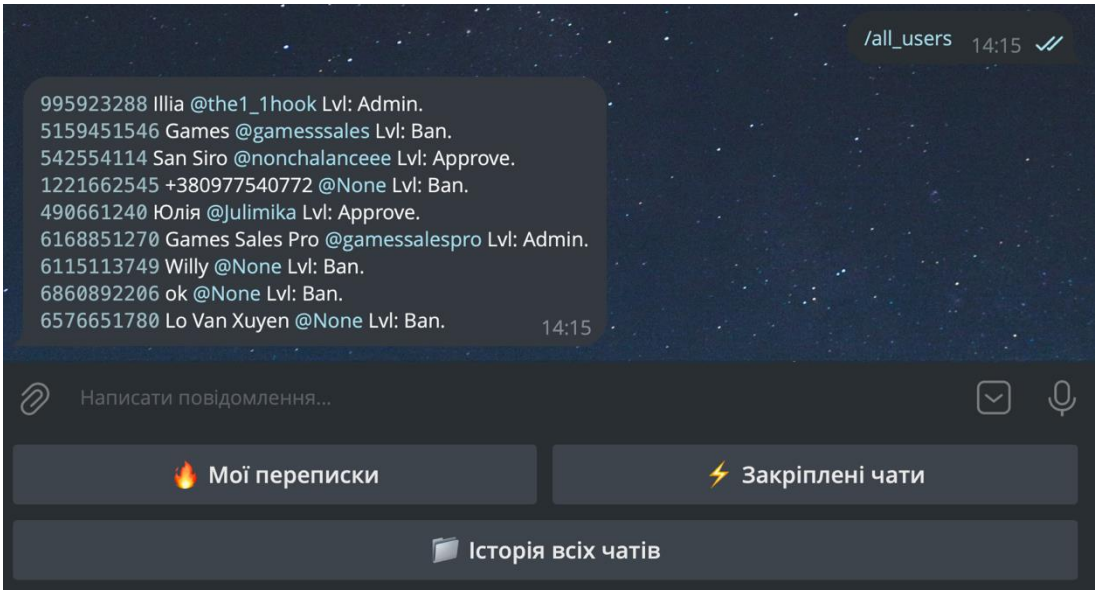

Рисунок 3.7 - Приклад управління користувачами, перегляд всіх користувачів бота.

Функція обробка відповідей на кнопки (callback_query = offset)

Умова Виклику:

- Викликається при отриманні callback_query, пов'язаного з пагінацією користувачів (/offset).

Функції:

- Перевіряє рівень доступу користувача.
- Викликає функцію send_users для відправлення списку користувачів з урахуванням пагінації.

Особливості:

- Усі функції мають перевірку рівня доступу користувача (if user.lvl != -1:), що робить їх недоступними для використання користувачами з обмеженим доступом.
- Використовується багатопотоковість для запуску скрипту видалення історії (threading.Thread(target=scripts.delete_history).start()).

Опис коду модуля управління чатами від лиця адміністратора бота. Код відображено **в додатку В.4:**

Функція get_connected_users(message)

Тип Обробки:

- Обробка текстових повідомлень.

Умова Виклику:

- Викликається для повідомлень, які мають текстовий контент та відповідають команді /get_connected_users_{chat_id}.

Функції:

- Перевіряє рівень доступу користувача.
- Отримує інформацію про підключених користувачів до вказаного чату.
- Відправляє інформацію про чат та його підключених користувачів у вигляді текстового повідомлення.

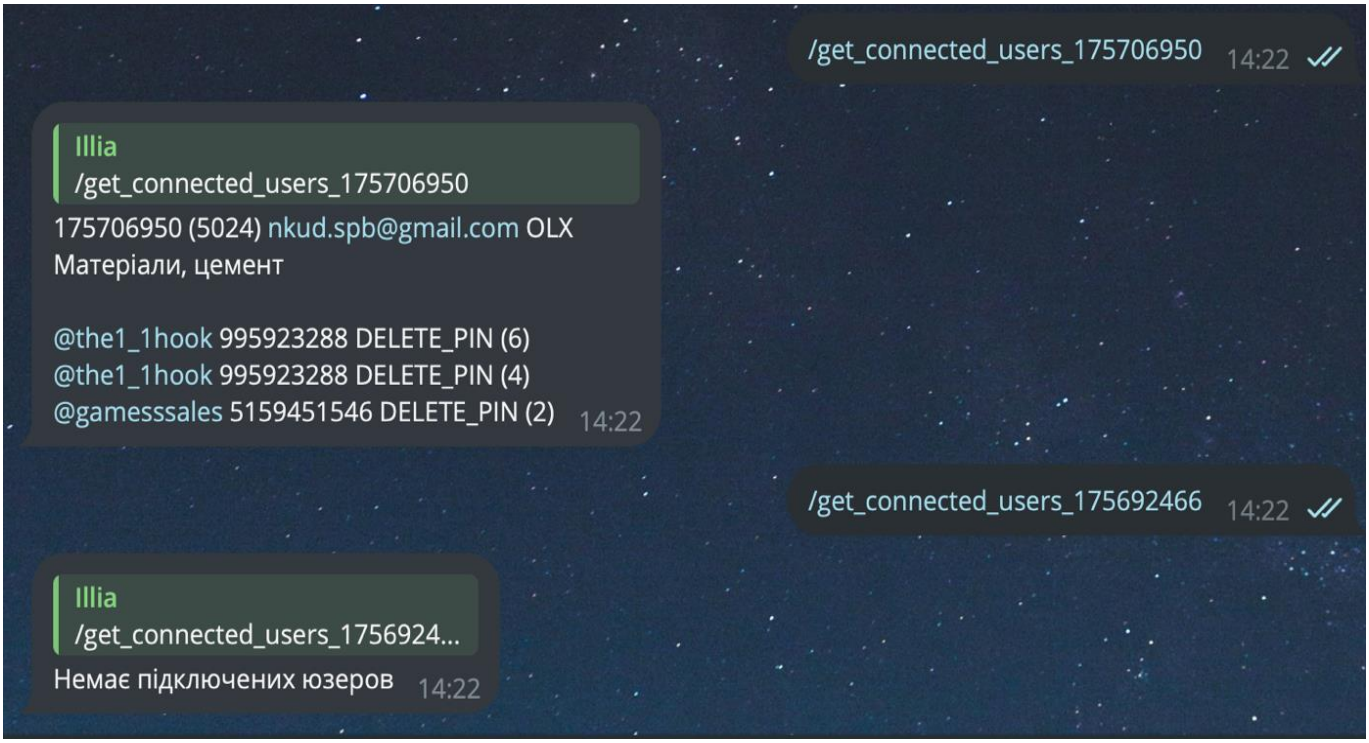

Рисунок 3.8 - Приклад управління чатами, перегляд прав доступу

Функція connect_to_chat(message)

Тип Обробки:

- Обробка текстових повідомлень.

Умова Виклику:

- Викликається для повідомлень, які мають текстовий контент та відповідають команді /connect_to_chat_{chat_id}.

Функції:

- Перевіряє рівень доступу користувача.
- Отримує інформацію про чат, до якого потрібно підключити користувача.
- Виконує підключення користувача до чату з урахуванням різних станів підключення (PIN, ACTIVATE_KEY).
- Оновлює статус та останній ID чату для користувача.
- Відправляє інформацію про чат та його підключених користувачів у вигляді текстового повідомлення.

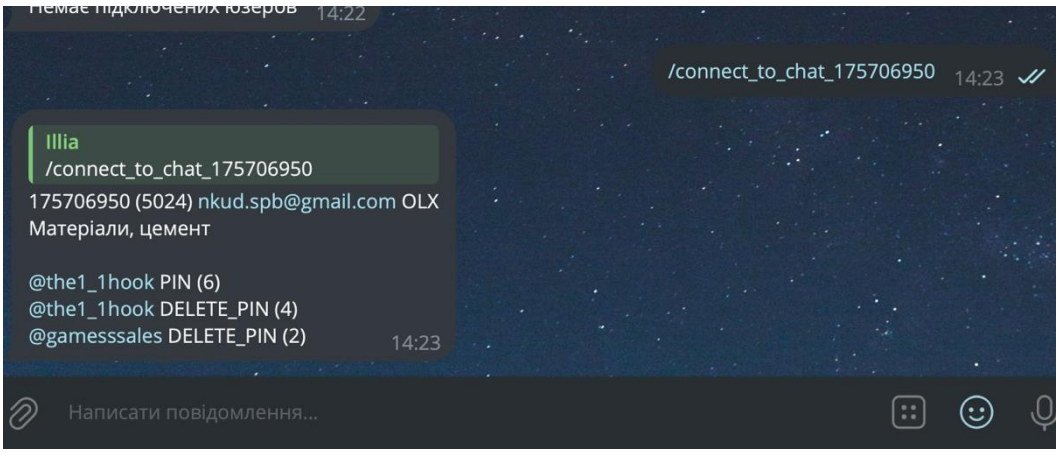

Рисунок 3.9 - Приклад підключення до чату, за допомогою адмін команди

Функція get_chat(message)

Тип Обробки:

- Обробка текстових повідомлень.

Умова Виклику:

- Викликається для повідомлень, які мають текстовий контент та відповідають команді /get_chat_{chat_id}.

Функції:

- Перевіряє рівень доступу користувача.
- Отримує інформацію про вказаний чат.
- Відправляє інформацію про чат у вигляді текстового повідомлення.

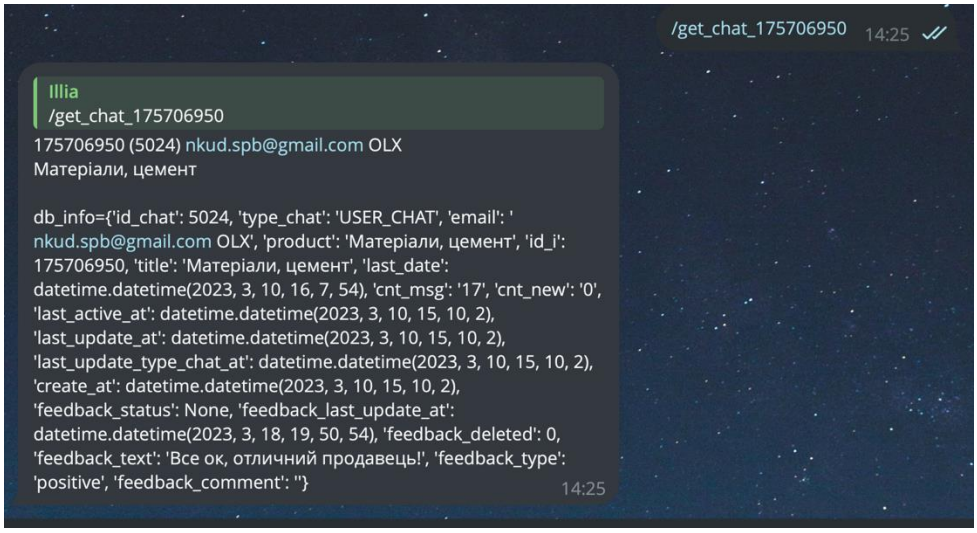

Рисунок 3.10 - Приклад управління чатом, перегляд інформації з бази даних

Інформація про чат.

Реалізовано модуль, що відображає дані з CRM системи про існуючі чати та повідомлення. Забезпечено можливість відправки повідомлень, отримання статистики та іншої інформації, пов'язаної із чатами.

Код цього модуля відображено в **додатках В.6, В.7, В.8, В.9.**

Функція send_status_read:

- Обробляє команду /send_status_read для відправки статусу "прочитано" в CRM-систему.
- Використовується замок для унікальності операцій з кожним чатом.
- Здійснює відправку запиту до CRM-системи та обробляє результат.
- Запускає потік для асинхронного отримання нових повідомлень у чаті.

Функція open_chat_call:

- Обробляє callback-запити для відкриття чату.
- Встановлює активний чат для користувача та переходить до статусу відправки повідомлення.

Функція open_chat:

- Обробляє команди /open_chat для відкриття конкретного чату за його ідентифікатором.
- Встановлює активний чат та переходить до статусу відправки повідомлення.

Функція history_chat:

- Обробляє команду /history_chat для відображення історії чату.
- Використовує ідентифікатор чату, збережений у статусі користувача.

Функція pagination_message:

- Обробляє callback-запити для пагінації повідомлень у чаті.
- Використовує ідентифікатор чату та зміщення для отримання певної кількості повідомлень.

Функція send_message_to_chat:

- Обробляє команду /send_message_to_chat для переходу до статусу відправки повідомлення.
- Отримує інформацію про поточний активний чат користувача з бази даних.
- Встановлює новий статус користувача для відправки повідомлення у вибраний чат.

- Надсилає користувачеві повідомлення з інструкціями та клавіатурою відправлення повідомлення.

Функція back_to_chat:

- Обробляє команду /back_to_chat для повернення до меню чату після відправлення повідомлення.
- Отримує інформацію про поточний активний чат користувача з бази даних.
- Встановлює новий статус користувача для перегляду меню чату.
- Вибирає відповідне меню чату (в залежності від типу чату) та надсилає його користувачеві.

Управління підключенням менеджерів. Модуль дозволяє підключати менеджерів до чатів та закріплювати їх за цими чатами на 24 години. Забезпечено зручний інтерфейс для управління цим процесом. Код цього модуля відображено в **додатку В.10.**

Для забезпечення коректної роботи в умовах конкурентної взаємодії використовуються механізми блокування (threading.Lock).

Функція history_transfer_data:

- Обробляє вибір менеджера для підключення до чату або відповіді на повідомлення.
- Отримує інформацію про чат та користувача з параметрів виклику.
- Забезпечує конкурентний доступ до блокувань для запобігання конфліктів.
- Проводить перевірку та обробку стану підключення менеджера до чату.
- Виводить інформацію та надсилає повідомлення користувачу та менеджеру згідно вибору.

Функція connect:

- Обробляє вибір менеджера для підключення до чату або відповіді на повідомлення.
- Отримує інформацію про чат та користувача з параметрів виклику.
- Забезпечує конкурентний доступ до блокувань для запобігання конфліктів.
- Проводить перевірку та обробку стану підключення менеджера до чату.
- Виводить інформацію та надсилає повідомлення користувачу та менеджеру згідно вибору.

Модуль головного меню:

У модулі головного меню реалізовано обробку подій головного меню чат-бота, яке містить різні кнопки та відповіді на їхні натискання. Основна функціональність полягає в наданні користувачам можливості вибору різних опцій та отримання інформації. Код цього модуля відображено **в додатку В.11.**

Функція messages:

- Відкриває меню чатів для користувача.
- Отримує список активних чатів користувача та виводить їх кількість та перелік.
- Використовує конфігураційне меню MESSAGES_MENU.

Функція history_messages:

- Відкриває історію повідомлень для користувача.
- Отримує різні типи історії повідомлень залежно від обраної опції (PIN, UN_PIN, UN_ACTIVATE_KEY).
- Використовує конфігураційне меню для кожного типу історії.

Функція pin_chats:

- Відкриває меню закріплених чатів для користувача.

- Отримує список активних чатів, які закріплені за користувачем.
- Використовує конфігураційне меню PIN_CHATS_MENU.

Функція history:

- Відкриває загальне меню історії чатів.
- Отримує загальний перелік чатів без розбіжностей за типами.
- Використовує загальне конфігураційне меню.

Функція next_by_callback:

Обробляє вибір користувача в інтерактивному режимі.

- Змінює виведені чати відповідно до натискання кнопок "Далі".
- Використовується для пагінації списків чатів залежно від контексту.

Модуль реалізований для зручного та інтуїтивно зрозумілого взаємодії з користувачем, надаючи йому доступ до опцій та інформації через відповідні кнопки.

3.3. Інтеграція чат-бота з CRM-системою KeepingCRM

Інтеграція з CRM-системою визначає новий рівень зручності та функціональності для обробки та відстеження клієнтських взаємодій у реальному часі.

Код відповідає за інтеграцію чат-бота з CRM-системою KeepingCRM. Для цього використовується API системи для отримання тимчасового токена, доступу до чатів, повідомлень та інших функцій. Основні функції:

Функція get_token(token=config.API_KEY) необхідна для отримання тимчасового токена від CRM-системи:

```
def get_token(token=config.API_KEY):
    global good_token, good_token_time
    headers = {
        'Content-Type': "application/json",
        'Accept': "application/json"
    }
```

```
        timestamp = time.time()
        sign            =            sha256((token           +
str(round(timestamp))).encode('utf-8')).hexdigest()

        token_json_request = {
            "seller_id": 479276,
            "timestamp": timestamp,
            "sign": sign
        }

        req                                                    =
requests.post(url=f'https://buknadia.keeping.com/api/apilog
in', json=token_json_request, headers=headers)
        if str(req.json().get('retval')) == '0':
            good_token = req.json().get('token')
            good_token_time = datetime.datetime.now()

        return json.loads(req.text).get('token')
```

Функція check_token() необхідна для перевірки актуальності та наявності тимчасового токена:

```
    def check_token():
        global good_token, good_token_time
        if  good_token_time  is  None  or  good_token_time  +
datetime.timedelta(hours=1) < datetime.datetime.now():
            tk = get_token()
        else:
            tk = good_token
        return tk
```

Функція get_chats(tk=None, page=1, filter_new=0) необхідна для отримання інформації про чати користувачів від CRM-системи:

```
    def get_chats(tk=None, page=1, filter_new=0):
        if tk is None:
            tk = check_token()
```

```
        headers = {
            'Content-Type': "application/json",
            'Accept': "application/json"
        }
        json_request = {
            'pagesize': 20,
            'page': page,
            'filter_new': filter_new,

        }

        req                                                    =
requests.get(url=f'https://buknadia.keeping.com/api/debates
/v2/chats?token={tk}',
                             params=json_request,
                             headers=headers)

        return req.json()
```

Функція get_messages(tk=None, id_i=None) необхідна для отримання повідомлень для конкретного чату від CRM-системи:

```
    def get_messages(tk=None, id_i=None):
        if tk is None:
            tk = check_token()

        headers = {
            'Content-Type': "application/json",
            'Accept': "application/json"
        }
        json_request = {
            'count': 200,
            'hidden': 0,
        }

        req                                                    =
requests.get(url=f'https://buknadia.keeping.com/api/debates
/v2?token={tk}&id_i={id_i}',
                             params=json_request,
                             headers=headers)
        return req.json()
```

Функція insert_file(tk=None, file=None) необхідна для відправлення файлу до CRM-системи для чату:

```
    def insert_file(tk=None, file=None):
        if tk is None:
            tk = check_token()

        files = [
            ('files[]', open(file, 'rb')),
        ]

        req                                                    =
requests.post(url=f'https://buknadia.keeping.com/api/debate
s/v2/upload-preview?token={tk}&lang=ru-RU',
                              files=files, )
        return req.json()
```

Функція send_message(tk=None, id_i=None, message=None, files=None) необхідна для відправлення повідомлення до конкретного чату від CRM-системи:

```
    def   send_message(tk=None,   id_i=None,   message=None,
files=None):
        if tk is None:
            tk = check_token()

        headers = {
            'Content-Type': "application/json",
            'Accept': "application/json"
        }

        json_request = {
            "message": message,
        }
        if files is not None:
            json_request["files"] = files

        req                                                    =
requests.post(url=f'https://buknadia.keeping.com/api/debate
s/v2/?token={tk}&id_i={id_i}', json=json_request,
                              headers=headers)

        return req
```

Функції send_seen(tk=None, id_i=None), get_purchase(tk=None, invoice_id=None) використовується для відправлення підтвердження перегляду та отримання інформації про покупку:

```
    def send_seen(tk=None, id_i=None, ):
       if tk is None:
           tk = check_token()

       headers = {
           'Content-Type': "application/json",
           'Accept': "application/json"
       }

       req                                                    =
requests.post(url=f'https://buknadia.keeping.com/api/debate
s/v2/seen?token={tk}&id_i={id_i}',
                              headers=headers)

       return req
```

3.4. Архітектура та принцип роботи чат-бота для ТОВ“Буковинська Надія”

Цей чат-бот є ключовим інструментом для впровадження ефективної системи комунікації та обробки інформації, спрямованої на покращення відносин з клієнтами та оптимізацію внутрішніх процесів підприємства. Складові інформаційної системи відображені на рисунку . 3.11.

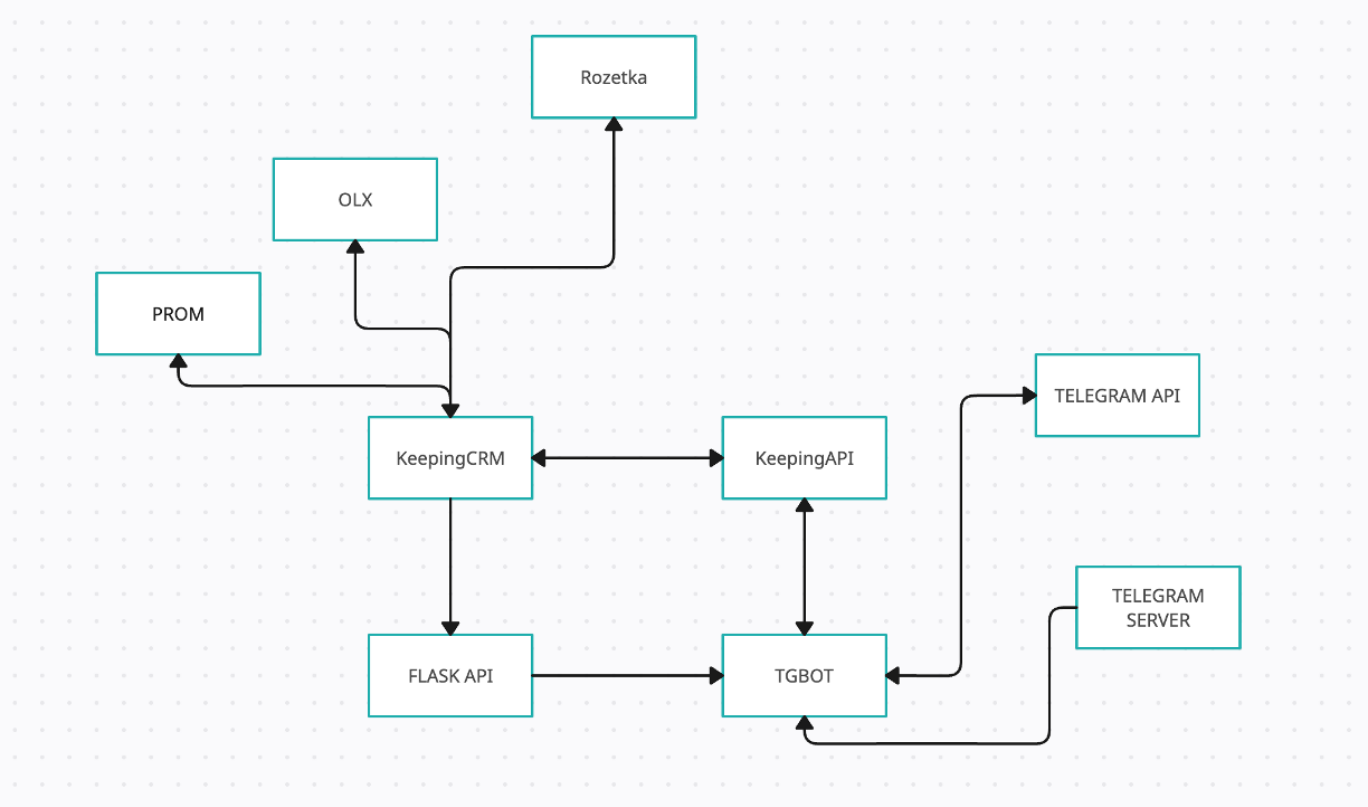

Рисунок 3.11 - Складові інформаційної системи

Складові інформаційної системи для ТОВ “Буковинська Надія” та їх призначення:

- KeepingCRM, KeepingAPI - забезпечує інтеграцію з ресурсами продажу та має модулі для редагування товарів та цін;
- OLX, Prom, Rozetka - ресурси для продажу товарів і послуг;
- TELEGRAM API використовується для розсилки повідомлень до спільного чату. Повідомлення також розсилаються в особисті чати менеджерів;
- TELEGRAM SERVER відправляє WebHook у TGBOT;
- FLASK API. Використання фреймворка Flask для реалізації WebHook на сервері. Сигнал від CRM системи про надходження повідомлення активує серверний скрипт;
- TGBOT. Бот, що використовує Telegram API для розсилки повідомлень до спільного чату. Повідомлення також розсилаються в особисті чати менеджерів.

Основні функції чат-боту:

- отримання сигналу від CRM системи про надходження нового повідомлення;
- аналіз та обробка повідомлення та запускає скрипт для обробки отриманого повідомлення;
- оновлення бази даних через імпортовані дані з з CRM;
- розсилка повідомлень до спільного чату та особистих чатів менеджерів;
- відповіді від менеджерів надсилаються на сервер для подальшої обробки;
- бот використовує написані правила для автоматизованої підтримки діяльності підприємства.

3.5. Сервер на Flask для отримання сигналу від CRM про надходження нового повідомлення

Для ефективного вирішення завдань та покращення функціоналу чат-боту впроваджено додатковий функціонал, пов'язаний із CRM. У цьому контексті розглядається важливий етап обробки та взаємодії з новими повідомленнями від CRM за допомогою WebHook.

Націлення на оптимізацію та автоматизацію відбувається через розробку спеціального серверу, який служить для обробки надходження нових повідомлень від CRM. Сервер приймає дані у форматі JSON, і в результаті запускає відповідний скрипт для вставки нових повідомлень в базу даних системи чат-боту. Виокремлений процес вставки дозволяє ефективно обробляти великі потоки даних та підтримувати гнучку роботу системи.

Основний маршрут для обробки надходження нових повідомлень від CRM - /messages/{LINK_KEY}. При отриманні POST-запиту, витягується ідентифікатор

чату (InvoiceId) з тіла JSON-запиту. Після чого викликається os.system для запуску виконання скрипту insert_new_message_by_webhook.py в іншому потоці.

Код для отримання сигналу від CRM:

```
import datetime
import logging
import os
import threading

from flask import jsonify, render_template, Flask, request, Response
from flask_cors import CORS
from config import LINK_KEY

import time

app = Flask(__name__)
CORS(app)

@app.before_request
def handle_preflight():
   if request.method == "OPTIONS":
       res = Response()
       res.headers['X-Content-Type-Options'] = '*'
       return res

logging.basicConfig(filename=f'log/debug_{datetime.date.today()}.log', level=logging.DEBUG,
                    format='%(asctime)s    %(levelname)s %(name)s %(message)s')
logging.Formatter.converter = time.gmtime

@app.errorhandler(404)
def not_found(e):
   return render_template("404.html"), 404

@app.route(f"/messages/{LINK_KEY}",     methods=["GET", "POST"])
def messages(**kwargs):
   try:
       logging.warning(f'{request.json=}')
```

```
            id_chat = request.json.get('InvoiceId')
            if not id_chat:
                raise ValueError('not InvoiceId')

            threading.Thread(target=os.system(f'python3
/root/booknadia_messanger/insert_new_message_by_webhook.py
{id_chat}'))
        except:
            pass
        return jsonify(ok=True), 200
```

3.6. Вставка нових повідомлень

Основний скрипт вставки повідомлень взаємодіє з базою даних, зберігаючи нові повідомлення, і надсилає їх відповідним менеджерам та чатам чат-бота.

Алгоритм роботи скрипта, код відображений в додатку В.12:

- Здійснюється отримання ID чату з новим повідомленням від CRM.
- Перевіряється наявність ID чату в базі даних чат-бота, створення нового чату, якщо його ще немає.
- Зберігаються дані про нові повідомлення в базі даних.
- Нові повідомлення надсилаються відповідним чатам та менеджерам чат-бота.

Функція get_chats:

- Отримання списку чатів з CRM для подальшої обробки.
- Перевірка та порівняння інформації про чати в базі даних та від CRM.
- Виклик функції get_messages для кожного чату, де відбувається отримання та обробка повідомлень.

Функція get_messages:

- Отримання повідомлень для конкретного чату з CRM.

- Порівняння інформації про повідомлення в базі даних та отримані від CRM.
- Вставка нових повідомлень до бази даних чат-бота.
- Відправка повідомлень до відповідних чатів Telegram.

3.7. Реалізація метод WebHook для взаємодії з Telegram API

Метод WebHook є ефективним рішенням, яке було досліджено раніше, і його впровадження дозволяє покращити продуктивність та надійність чат-бота. Код висвітлено **в додатку В.13**

Основні етапи використання методу WebHook включають:

- Створення Flask додатку: У коді реалізовано веб-сервер на основі Flask, який відповідає за обробку вхідних HTTP-запитів від Telegram. Flask дозволяє легко налаштовувати маршрути та обробляти вхідні дані.
- Налаштування маршрутів: За допомогою Flask визначені два маршрути: один для верифікації WebHook при його налаштуванні та інший для обробки вхідних оновлень від Telegram.
- Обробка вхідних оновлень: При отриманні вхідного оновлення від Telegram через метод WebHook, викликається функція webhook(), яка перевіряє тип контенту та обробляє отримані дані за допомогою основного бот-модуля main_bot.
- Забезпечення безперебійної роботи WebHook: Реалізовано цикл start(), який використовується для постійного перевстановлення WebHook, забезпечуючи стабільну роботу чат-бота. При виникненні помилок або втраті з'єднання процес автоматично відновлюється.
- Використання методу WebHook спрощує взаємодію з Telegram API, зменшуючи витрати на ресурси та забезпечуючи швидку та ефективну обробку вхідних повідомлень чат-бота для підприємства "Буковинська Надія".

3.8. Функціональні можливості чат-бота та їх взаємодія з різними джерелами продажу

Чат-бот має ряд функціональних можливостей, які полегшують взаємодію з різними джерелами продажу та забезпечують ефективне управління замовленнями та клієнтським сервісом.

Взаємодія з CRM-системою (KeepingCRM):

- Отримання інформації про замовлення. Бот взаємодіє з CRM-системою, щоб отримувати дані про замовлення, клієнтів та інші важливі дані.
- Відправлення повідомлень в CRM. Бот може відправляти повідомлення та оновлення в CRM-систему для комунікації з клієнтами.
- Обробка замовлень та чатів. Відправлення повідомлень користувачам: Бот дозволяє надсилати повідомлення клієнтам, вирішувати питання та надавати інформацію про замовлення.
- Відстеження статусу замовлення. Бот може надсилати користувачам статус їх замовлення та інформацію про доставку.
- Управління чатами. Користувачам надається можливість переглядати історію чатів для відстеження попередніх обговорень та рішень.

3.9. Висновок до розділу 3

В цьому розділі були визначені та обґрунтовані ключові аспекти процесу розробки чат-бота для ТОВ “Буковинська Надія”. Обрана мова програмування Python та бібліотека pyTelegramBotAPI стали основою для реалізації чат-бота з урахуванням їхніх переваг.

Процес розробки чат-бота був ретельно проаналізований та розкритий на етапи, включаючи інтеграцію з CRM - системою KeepingCRM, використання серверу на Flask для обробки сигналів від CRM, реалізацію методу WebHook для

взаємодії з TelegramAPI, а також реалізації самого чат-бота для підтримки діяльності підприємства ТОВ “Буковинська Надія”.

Важливим аспектом використання чат-бота є той факт, що реалізація бота відбувалася за допомогою методу WebHook, який було детально проаналізовано в розділі 2. Такий вибір і його успішна імплементація в роботі чат-бота є підтвердженням того, що наукове дослідження методів розроблення чат-ботів відобразилося на реальній практиці та принесло позитивні результати для функціональності бота.

Наслідки використання розробленого чат-бота для підприємства можуть бути значущими. Впровадження чат-бота дозволить забезпечити ефективну комунікацію з клієнтами, підвищити ефективність обробки звернень і замовлень, тобто забезпечити оперативну взаємодію з клієнтами. Це призведе до покращення рівня обслуговування клієнтів та зростання їх задоволеності.

Крім того, використання чат-бота може допомогти у впровадженні інноваційних методів взаємодії з клієнтами та підвищити конкурентоспроможність підприємства на ринку.

В цілому, чат-бот може стати ефективним інструментом для оптимізації бізнес-процесів.

ВИСНОВКИ

У даному дослідженні було проведено аналіз різних методів інтеграції месенджерів для чат-бот.

Проведений аналіз факторів, що впливають на часові показники роботи чат-бота, вказують на те, що методи мають різні показники характеристик. У методу Long Polling основним фактором є час очікування відповіді, що може викликати затримки в обробці подій. З іншого боку, метод WebHook вирізняється низьким часом обробки та швидкістю отримання повідомлень, що зменшує час відповіді.

За результатами дослідження було зроблено висновок що обидва методи мають свої переваги та недоліки. Long Polling є простим у використанні, але може призводити до затримок. З іншого боку, Webhook є більш ефективним, але вимагає більше уваги до налаштувань та безпеки.

Вибір між Long Polling та Webhook повинен ґрунтуватися на конкретних вимогах проекту та інфраструктурних можливостях. У даному випадку впроваджено Webhook, оскільки його переваги в контексті швидкої та ефективної взаємодії з месенджером виявилися більш суттєвими для успішної реалізації чат-бота.

Проведений аналіз експериментальних дослідження ефективності роботи чат-бота з використанням методів Long Polling та WebHook дав змогу виявити, що метод WebHook має середній час відповіді (2.4 секунди), що відчутно менший, ніж у методі Long Polling (3 секунди).

Процес розробки чат-бота був ретельно проаналізований та розкритий на етапи, включаючи інтеграцію з CRM - системою KeepingCRM, використання серверу на Flask для обробки сигналів від CRM, реалізацію методу WebHook для взаємодії з TelegramAPI, а також реалізацію самого чат-бота для підтримки діяльності підприємства ТОВ “Буковинська Надія”.

Обрано метод WebHook для реалізації чат-бота, оскільки його характеристики відповідають вимогам до ефективної та оперативної взаємодії з Telegram API. Цей вибір забезпечує швидку реакцію на події та покращує загальні показники чат-бота для оптимальної взаємодії з користувачами.

У відділі продажів ТОВ «Буковинська Надія», у випадку нового повідомлення, в CRM надходило сповіщення з посиланням на чат. Якщо обидва менеджера одночасно клікнули на це посилання, вони відповідали на одне й те саме повідомлення. В результаті внесених змін, після першої відповіді чат автоматично закріплюється за конкретним менеджером. Інші менеджери отримують інформацію про це закріплення, що унеможливлює подвійні відповіді на одне повідомлення. Такий підхід сприяє кращій організації роботи, уникненню непорозумінь та покращенню загальної продуктивності взаємодії з клієнтами.

Реалізувавши цей бот, з'явилась можливість відповідати на повідомлення з будь-якого пристрою, на якому встановлено месенджер Telegram. Це означає, що менеджери можуть працювати з чат-ботом не лише на робочому місці, але і з інших пристроїв, таких як особисті комп'ютери, планшети чи мобільні телефони, навіть перебуваючи в громадському транспорті або у будь-яких інших місцях. Це дозволяє забезпечити більшу гнучкість та мобільність у роботі, зробити відповіді на повідомлення швидшими та зручнішими для команди менеджерів.

Внесок у підтримку діяльності ТОВ «Буковинська Надія» включав в себе значне покращення розподілу чатів між менеджерами, спрямоване на підвищення ефективності та уникнення ситуацій одночасної відповіді на повідомлення обома менеджерами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Полякова А. Telegram, Viber, WhatsApp, Signal — яким месенджерам можна довіряти [Електронний ресурс] / Аліна Полякова // Економічна правда. – Режим доступу до ресурсу: https://www.epravda.com.ua/publications/2017/12/15/632183 (дата звернення: 19.12.2023).
2. Brian Busch, Evaluating Webhooks vs. Polling. [Електронний ресурс] / Brian Busch // Dzone. – Режим доступу до ресурсу: https://dzone.com/articles/evaluating-webhooks-vs-polling (дата звернення: 19.12.2023).
3. Telegram Bot API [Електронний ресурс] // Telegram. – Режим доступу до ресурсу: https://core.telegram.org/bots/api (дата звернення: 19.12.2023).
4. Telegram FAQ [Електронний ресурс] // Telegram. – Режим доступу до ресурсу: https://telegram.org/faq (дата звернення: 19.12.2023).
5. Code With Ben. SurvTech. Webhooks vs Polling in APIs [Електронний ресурс] / Code With Ben // YouTube. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=7ID6U6uzNTk (дата звернення: 19.12.2023).
6. When to Use Webhooks, WebSocket, Pub/Sub, and Polling [Електронний ресурс] // Hookdeck. – Режим доступу до ресурсу: https://hookdeck.com/webhooks/guides/when-to-use-webhooks (дата звернення: 19.12.2023)
7. Красильнікова О.І. Що таке месенджери, які у них можливості - топ популярних месенджерів [Електронний ресурс] / Красильнікова О.І. // ВсеОсвіта. – Режим доступу до ресурсу: https://vseosvita.ua/library/so-take-mesendzeri-aki-u-nih-mozlivosti-top-popularnih-mesendzeriv-281047.html (дата звернення: 19.12.2023)
8. Amazon Web Services. What is an API (Application Programming Interface)? [Електронний ресурс] // Amazon Web Services. – Режим доступу до ресурсу: https://aws.amazon.com/what-is/api/ (дата звернення: 19.12.2023).
9. Wikipedia. Telegram (software) [Електронний ресурс] // Wikipedia. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Telegram_(software) (дата звернення: 19.12.2023).
10. Stegner, Ben. 6 Reasons Why You Might Want to Stop Using Telegram [Електронний ресурс] / Ben Stegner // MakeUseOf. – Режим доступу до ресурсу: https://www.makeuseof.com/reasons-stop-using-telegram/ (дата звернення: 19.12.2023).
11. Методичні рекомендації до викон. випускної кваліфікаційної роботи на здобуття освітнього ступеня «Магістр» спец. 122 «Комп’ютерні науки», освітньо-професійної програми «Інформаційні управляючі системи та технології» ден. форми навчання [Електронний ресурс] / укладачі О. М. М’якшило, М. П. Костіков. – К.: НУХТ, 2022. – 28 с.

ДОДАТКИ

Додаток А - Архітектура модулів, моделі, скрипти та утиліти

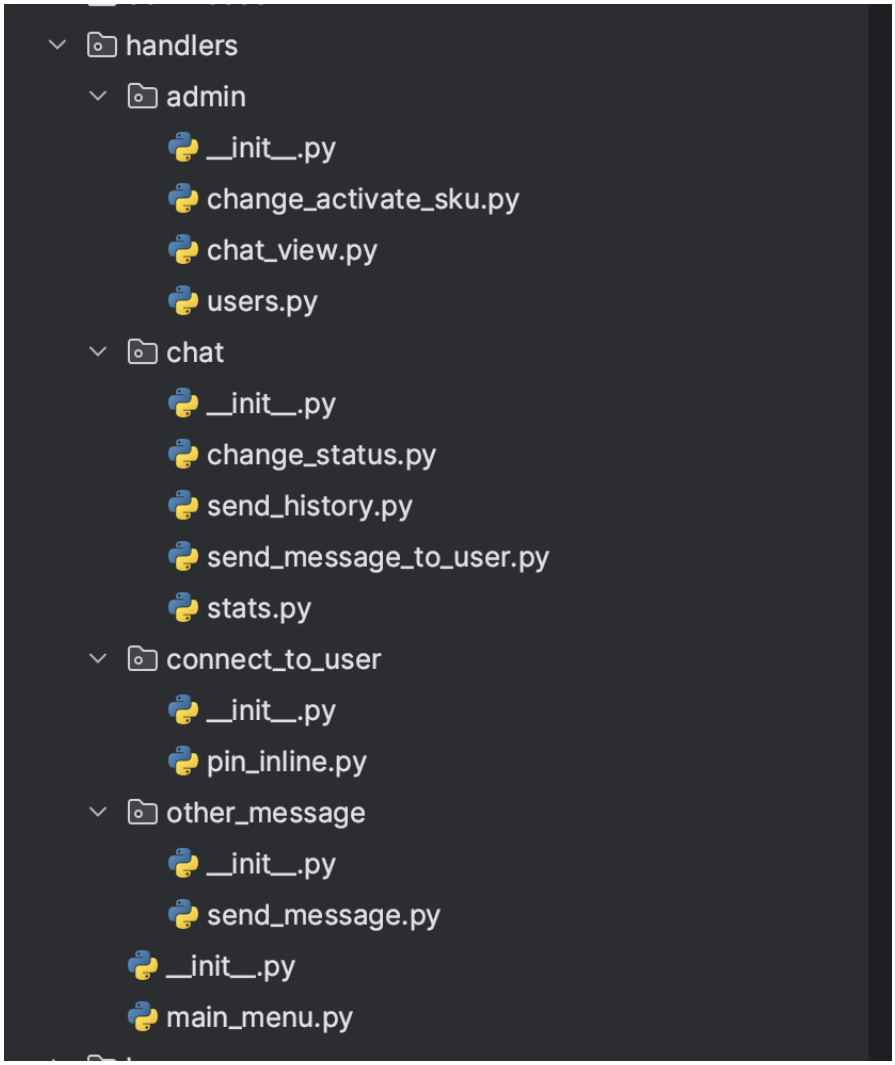

Рисунок А.1 - Архітектура модулів для управління повідомленням в боті

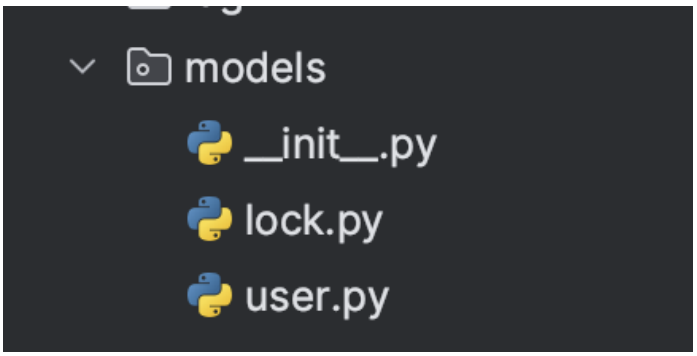

Рисунок А.2 - Список моделей використаних в боті

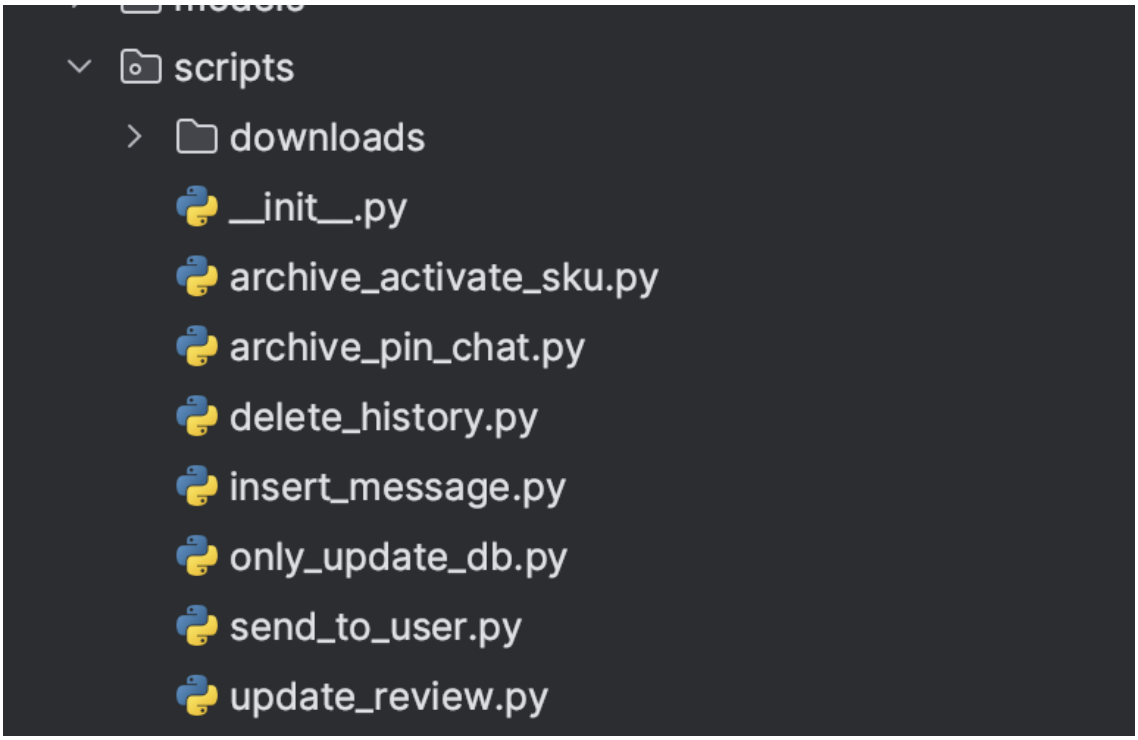

Рисунок А.3 - Список допоміжних скриптів

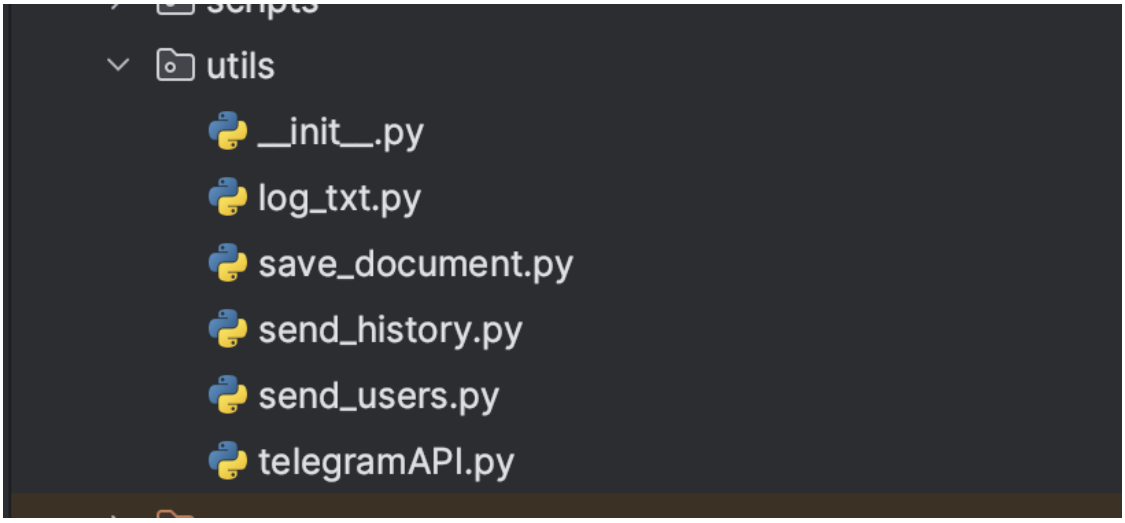

Рисунок А.4 - Список допоміжних утиліт

Додаток В - Код програми

1. Модель User

```
import datetime

import database

class Singleton(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton,
cls).__call__(*args, **kwargs)
        return cls._instances[cls]

class User:
    status = None
    last_id_chat = None
    lvl = None
    db_id_user = None
    info = None

    def __init__(self, telegram_id):
        self.telegram_id = telegram_id
        self.update()

    def update(self, ):
        self.info =
database.get_info_user(self.telegram_id)
        if self.info is None:
            return

        super().__setattr__("status",
self.info.get('last_status'))
        super().__setattr__("lvl", self.info.get('lvl'))
        super().__setattr__("db_id_user",
self.info.get('id_user'))
        super().__setattr__("last_id_chat",
self.info.get('last_id_chat'))

    def __setattr__(self, name, value):
```

```
            if name == 'status':
                database.set_user_status(id_user=self.db_id_
user, status=value)

            if name == 'lvl':
                database.set_user_lvl(id_user=self.db_id_use
r, lvl=value)

            if name == 'last_id_chat':
                database.set_last_id_chat(id_user=self.db_id
_user, last_id_chat=value)

            object.__setattr__(self, name, value)

    class UserManager(metaclass=Singleton):
        all_users = {}
        last_updates = {}

        def get_user(self, telegram_id) -> User:
            if        str(telegram_id)        not        in
list(self.all_users.keys()):
                self.all_users[str(telegram_id)]           =
User(str(telegram_id))
                self.last_updates[str(telegram_id)]        =
datetime.datetime.now()

            if     self.last_updates[str(telegram_id)]     +
datetime.timedelta(minutes=1) < datetime.datetime.now():
                self.all_users[str(telegram_id)].update()
                self.last_updates[str(telegram_id)]        =
datetime.datetime.now()

            return self.all_users[str(telegram_id)]

        def get_all_users(self) -> dict:
            return self.all_users
```

2. Модель Lock

```
    import datetime
    import threading
```

```
    class Singleton(type):
       _instances = {}

       def __call__(cls, *args, **kwargs):
           if cls not in cls._instances:
               cls._instances[cls]    =    super(Singleton,
cls).__call__(*args, **kwargs)
           return cls._instances[cls]

    class LockManager(metaclass=Singleton):
       locked = {}
       lock = threading.Lock()

       def get_lock(self, name) -> threading.Lock:
           self.lock.acquire()
           try:
               if name not in self.locked:
                   self.locked[name] = threading.Lock()
                   print(self.locked[name])
               return self.locked[name]
           finally:
               self.lock.release()
```

3. Код модуля управління користувачами admin.user

```
    import threading

    import models
    import scripts
    import database
    from main import bot
    import utils

    @bot.message_handler(content_types=['text'],
                         func=lambda                message:
message.text.split('_')[0] in ['/approve', '/ban', '/admin']
                         )
    def get_message(message):
       user                                                =
models.UserManager().get_user(message.chat.id)
       if user.lvl != -1:
```

```
            return

        user_change                                                 =
models.UserManager().get_user(message.text.split('_')[1])
        if user_change.info is None:
            bot.reply_to(message, 'Немає такого юзера')
            return

        user_change.lvl = 0 if message.text.split('_')[0] ==
'/ban' else (-1 if message.text.split('_')[0] == '/admin'
else 1)
        bot.reply_to(message,
f'Готово!\n{user_change.__dict__}')

    @bot.message_handler(commands=['all_users'])
    def search(message):
        user                                                        =
models.UserManager().get_user(message.chat.id)
        if user.lvl != -1:
            return

        utils.send_users(bot, message.chat.id, 0, None)

    @bot.callback_query_handler(func=lambda           call:
str(call.data).split('_')[0] == 'offset')
    def callback_inline(call):
        user                                                        =
models.UserManager().get_user(call.message.chat.id)
        if user.lvl != -1:
            return

        call_split = str(call.data).split('_')
        utils.send_users(bot=bot,
telegram_id=call.message.chat.id,
offset=int(call_split[1]),
                         message_id=call.message.id)

    @bot.message_handler(commands=['clean'])
    def search(message):
        user                                                        =
models.UserManager().get_user(message.chat.id)
        if user.lvl != -1:
            return
```

```
        threading.Thread(target=scripts.delete_history).star
t()
        bot.reply_to(message, 'Запуск скрипта успішний')
```

4. Код модуля управління чатами admin.chat_view

```
    import models
    import database
    from main import bot
    import utils
    from config import USER_STATUS

    @bot.message_handler(
        content_types=['text'],
        func=lambda                                    message:
"_".join(message.text.split('_')[:-1])                    ==
'/get_connected_users'
                        )
    def get_connected_users(message):
        user                                                =
models.UserManager().get_user(message.chat.id)
        if user.lvl != -1:
            return

        chat  =  database.get_chat(message.text.split('_')[-
1])
        if not chat:
            bot.reply_to(message, 'Немає такого чата')
            return

        users                                               =
database.get_connected_users(chat['id_chat'])
        if not users:
            bot.reply_to(message,     'Немає     підключених
юзеров')
            return
        users_message        =        [f"@{user['username']}
{user['telegram_id']}                  {user['status_pin']}
({user['id_pin']})" for user in users]
        bot.reply_to(
```

```
            message,
            f'{chat["id_i"]}              ({chat["id_chat"]})
{chat["email"]}\n{chat["title"]}\n\n'
            + "\n".join(users_message)
        )

    @bot.message_handler(
        content_types=['text'],
        func=lambda                                    message:
"_".join(message.text.split('_')[:-1])                    ==
'/connect_to_chat'
                            )
    def connect_to_chat(message):
        user                                                =
models.UserManager().get_user(message.chat.id)
        if user.lvl != -1:
            return

        chat  =  database.get_chat(message.text.split('_')[-
1])
        if not chat:
            bot.reply_to(message, 'Немає такого чата')
            return

        users                                               =
database.get_connected_users(chat['id_chat'])
        new_connect = True
        update_connect_pin = False
        update_connect_un_activate = False

        for connected_user in users:
            if     str(connected_user['telegram_id'])     ==
str(message.from_user.id):
                if  connected_user['status_pin']  in  ['PIN',
'ACTIVATE_KEY', ]:
                    new_connect = False
                    update_connect_pin = False
                    update_connect_un_activate = False
                    break

                if      connected_user['status_pin']      in
['UN_PIN', 'DELETE_PIN']:
```

```
                    update_connect_pin                          =
connected_user['id_pin']
                    new_connect = False
                    break

                if      connected_user['status_pin']      in
['UN_ACTIVATE_KEY', ]:
                    update_connect_un_activate                  =
connected_user['id_pin']
                    new_connect = False
                    break

        if new_connect:
            database.connect_chat(
                id_chat=chat['id_chat'],
                id_user=user.db_id_user,
                type_connect='PIN'
            )

        elif update_connect_pin is not False:
            database.update_connect_chat(
                id_pin=update_connect_pin,
                type_connect='PIN')

        elif update_connect_un_activate is not False:
            database.update_connect_chat(
                id_pin=update_connect_pin,
                type_connect='ACTIVATE_KEY')

        user.status  =  USER_STATUS['send-message']  +  '_'  +
str(chat["id_chat"])
        user.last_id_chat = chat["id_chat"]

        users                                                   =
database.get_connected_users(chat['id_chat'])
        users_message        =        [f"@{user['username']}
{user['status_pin']} ({user['id_pin']})" for user in users]
        bot.reply_to(
            message,
            f'{chat["id_i"]}              ({chat["id_chat"]})
{chat["email"]}\n{chat["title"]}\n\n'
            + "\n".join(users_message)
        )
```

```
    @bot.message_handler(
        content_types=['text'],
        func=lambda                                    message:
"_".join(message.text.split('_')[:-1]) == '/get_chat'
                        )
    def get_chat(message):
        user                                                =
models.UserManager().get_user(message.chat.id)
        if user.lvl != -1:
            return

        chat  =  database.get_chat(message.text.split('_')[-
1])
        if not chat:
            bot.reply_to(message, 'Немає такого чата')
            return

        bot.reply_to(
            message,
            f'{chat["id_i"]}              ({chat["id_chat"]})
{chat["email"]}\n{chat["title"]}\n\ndb_info={chat}'
        )
```

5. Код допоміжних функцій відправки користувачів бота в чат адміністратору

```
    import database
    from telebot import types

    def send_users(bot, telegram_id, offset, message_id):
        limit = 20
        all_request    =    database.get_users(offset=offset,
limit=limit+1)
        # telegram_id, username, first_name, lvl, language,
limit_query
        navigation = types.InlineKeyboardMarkup()
        message = ''
        for user in all_request[:20]:
```

```
            message += f'<code>{user["telegram_id"]}</code>
{user["first_name"]} @{user["username"]} ' \
                       f'Lvl: {"Ban" if user["lvl"] == 0 else
("Admin" if user["lvl"] == -1 else "Approve")}.\n'
        if offset == 0 and len(all_request) != limit + 1:
            pass
        elif offset != 0 and len(all_request) != limit + 1:
            navigation.add(types.InlineKeyboardButton('Left'
, callback_data=f'offset_{offset - limit}'))
        elif offset > 0 and len(all_request) == limit + 1:
            navigation.add(types.InlineKeyboardButton('Left'
, callback_data=f'offset_{offset - limit}'),
                           types.InlineKeyboardButton('Right
', callback_data=f'offset_{offset + limit}'))
        elif offset == 0 and len(all_request) == limit + 1:
            navigation.add(types.InlineKeyboardButton('Right
', callback_data=f'offset_{offset + limit}'))

        if message_id is None:
            bot.send_message(telegram_id, message,
                             reply_markup=navigation,
                             parse_mode='html')
        else:
            bot.edit_message_text(text=message,
                                  chat_id=telegram_id,
                                  message_id=message_id,
                                  reply_markup=navigation,
                                  parse_mode='html')
```

6. Інформація про чат. Код про зміну статусу чата:

```
import datetime

import database
import models
from models import UserManager
from config import MENU, USER_STATUS
import config
from main import bot

@bot.message_handler(func=lambda message: message.text
in [MENU['edit_status_activate_key'], ])
def edit_status_activate_key(message):
```

```
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        chats = database.get_chats_users_arr(
            id_user=user.db_id_user,
            status_pin=['ACTIVATE_KEY', 'UN_ACTIVATE_KEY'])

        for chat in chats:
            if            str(chat['id_chat'])            ==
str(user.last_id_chat):
                if chat['type_chat'] == 'ACTIVATE_SKU':
                    database.update_type_chat(chat['id_chat'
], 'ACTIVATE_SKU_DONE')
                    message_text     =     f'Статус     чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                                  f'зміненний з "В роботі"
на "Закінчений"'

                elif          chat['type_chat']           ==
'ACTIVATE_SKU_DONE':
                    database.update_type_chat(chat['id_chat'
], 'ACTIVATE_SKU')
                    message_text     =     f'Статус     чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                                  f'зміненний             з
"Закінчений" на "В роботі"'
                    database.update_connect_chat(chat['id_pi
n'], 'ACTIVATE_KEY')

                else:
                    message_text  =  f'Статус  вашего  чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                                  f"не може бути змінений з
\"{chat['type_chat']}\"."

                bot.reply_to(
                    message,
                    message_text,
                    parse_mode='html'
                )
                return
```

```
        message_text                    =                f'Чат
#<code>{user.last_id_chat}</code> не є закріпленим за Вами'
        bot.reply_to(
            message,
            message_text,
            parse_mode='html'
        )

    @bot.message_handler(func=lambda message: message.text
in [MENU['un_status_chat'], ])
    def un_status_chat(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        chats = database.get_chats_users_arr(
            id_user=user.db_id_user,
            status_pin=['PIN', 'UN_PIN'])
        for chat in chats:
            if            str(chat['id_chat'])            ==
str(user.last_id_chat):
                if chat['type_chat'] == 'USER_CHAT':
                    database.update_connect_chat(chat['id_pi
n'], 'DELETE_PIN')
                    message_text     =     f'Статус     чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                                    f'зміненний             с
"{chat["status_pin"]}" на "DELETE_PIN"'
                    break
                else:
                    message_text           =           f'Чат
#<code>{chat["id_i"]}</code>   ({chat["email"]})   забороняє
видалення закрепа'
                    break
        else:
            message_text               =               f'Чат
#<code>{user.last_id_chat}</code> не є закріпленим за Вами'
            bot.reply_to(
                message,
                message_text,
                parse_mode='html'
            )
        try:
```

```
            bot.send_message(
                config.CHANNEL_LOG,
                '#CHANGE_PIN\n' + message_text,
                parse_mode='html'
            )
        except:
            pass

    @bot.message_handler(
        content_types=['text'],
        func=lambda message:
"_".join(message.text.split('_')[:-1]) ==
'/change_status_chat'
                            )
    def change_status_chat(message):
        user =
models.UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        chat = database.get_chat(message.text.split('_')[-
1])
        if not chat:
            bot.reply_to(message, 'Нет такого чата')
            return

        if chat['type_chat'] == 'ACTIVATE_SKU':
            database.update_type_chat(chat['id_chat'],
'ACTIVATE_SKU_DONE')
            message_text = f'Статус чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                           f'зміненний з "В роботі" на
"Закінчений"'

        elif chat['type_chat'] == 'ACTIVATE_SKU_DONE':
            database.update_type_chat(chat['id_chat'],
'ACTIVATE_SKU')
            message_text = f'Статус чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                           f'зміненний з "Закінчений" на "В
роботі"'

        else:
```

```
            database.update_type_chat(chat['id_chat'],
'ACTIVATE_SKU_DONE')
            message_text    =    f'Статус    вашого    чата
#<code>{chat["id_i"]}</code> ({chat["email"]}) ' \
                              f"був         змінений         з
\"{chat['type_chat']}\" на Закінчений (ACTIVATE_SKU_DONE)."

        bot.reply_to(
            message,
            message_text,
            parse_mode='html'
        )
        return
```

7. Інформація про чат. Код для перегляду історії чата:

```
    import database
    import crm
    import models
    from models import UserManager
    from config import MENU, USER_STATUS
    import config
    from main import bot
    from telebot import types
    import utils
    import threading
    import scripts

    @bot.message_handler(func=lambda message: message.text
in [MENU['send_status_read'], ])
    def send_status_read(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        chat = database.get_chat_by_id(user.last_id_chat)
        try:
            r = crm.send_seen(id_i=chat["id_i"])
            if r.status_code == 200:
                bot.reply_to(
                    message,
                    text=f'Готово, {chat["id_i"]}',
```

```
                )
                try:
                    threading.Thread(target=scripts.get_mess
ages,
                                     args=(None,
chat["id_chat"], chat["id_i"], chat)).start()
                except:
                    pass
                database.reset_cnt_new(chat['id_chat'])
                return
            bot.reply_to(
                message,
                text=f'Помилка {r.status_code}, {chat} ',
            )
        except Exception as ex:
            bot.reply_to(
                message,
                text=f'Помилка      {ex.__class__.__name__},
{ex}, {chat} ',
            )

    @bot.callback_query_handler(func=lambda           call:
call.data.split('_')[0] in
                                                ['open-
chat', 'open-chat-new', 'open-history-chat'], )
    def open_chat_call(call):
        user = UserManager().get_user(call.from_user.id)
        if user.lvl == 0:
            return

        id_chat = call.data.split('_')[-1]
        user.last_id_chat = int(id_chat)
        chat = database.get_chat_by_id(user.last_id_chat)
        user.status  =  USER_STATUS['send-message']  +  '_'  +
str(chat["id_chat"])

        bot.send_message(
            chat_id=call.from_user.id,
            text=f'Ви відповідаєте на ордер #{chat["id_i"]}
{chat["email"]}',
            reply_markup=config.SEND_MESSAGE_MENU
        )
```

```
        utils.send_history(bot,
telegram_id=call.from_user.id, id_chat=id_chat)

    @bot.message_handler(func=lambda                message:
message.text.split('_')[:2] == ['/open', 'chat'])
    def open_chat(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        id_chat = int(message.text.split('_')[-1])
        chat = database.get_chat(id_chat)
        user.last_id_chat = chat["id_chat"]
        user.status   =   USER_STATUS['in-chat']   +   '_'   +
str(chat["id_chat"])

        chat_menu = config.CHAT_MENU3

        bot.send_message(
            message.chat.id,
            text=f'Відкрито   меню   чата   {chat["id_i"]},
{chat["email"]}',
            reply_markup=chat_menu
        )
        utils.send_history(bot, telegram_id=message.chat.id,
id_chat=chat["id_chat"])

    @bot.message_handler(func=lambda message: message.text
in [MENU['history_chat'], ])
    def history_chat(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        id_chat = user.status.split('_')[-1]
        utils.send_history(bot, telegram_id=message.chat.id,
id_chat=id_chat)

    @bot.callback_query_handler(func=lambda           call:
call.data.split('_')[0] in ['pag-message', ], )
    def pagination_message(call):
        user = UserManager().get_user(call.from_user.id)
        if user.lvl == 0:
```

```
            return

        id_chat = call.data.split('_')[-2]
        offset = int(call.data.split('_')[-1])
        utils.send_history(bot,
telegram_id=call.from_user.id, id_chat=id_chat,
                           message_id=call.message.id,
offset=offset)
```

8. Код для відправки повідомлень до клієнтів

```
    import database
    import models
    from models import UserManager
    from config import MENU, USER_STATUS
    import config
    from main import bot
    from telebot import types
    import utils

    @bot.message_handler(func=lambda message: message.text
in [MENU['send_message_to_chat'], ])
    def send_message_to_chat(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        chat = database.get_chat_by_id(user.last_id_chat)
        user.status = USER_STATUS['send-message'] + '_' +
str(chat["id_chat"])
        bot.send_message(
            chat_id=message.chat.id,
            text=f'Ви відповідаєте на ордер #{chat["id_i"]}
{chat["email"]}',
            reply_markup=config.SEND_MESSAGE_MENU
        )

    @bot.message_handler(func=lambda message: message.text
in [MENU['back_to_chat'], ])
    def back_to_chat(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return
```

```
        chat = database.get_chat_by_id(user.last_id_chat)
        user.status = USER_STATUS['in-chat'] + '_' +
str(user.last_id_chat)

        chat_menu = config.CHAT_MENU
        if chat['type_chat'] in
config.CHOICE_CHAT_MENU.keys():
            chat_menu =
config.CHOICE_CHAT_MENU[chat['type_chat']]

        bot.send_message(
            message.chat.id,
            text=f'Відкрито меню чата {chat["id_i"]},
{chat["email"]}',
            reply_markup=chat_menu
        )
```

9. Код для відображення статистики по чатам

```
    import datetime

    import database
    import models
    from models import UserManager
    from config import MENU, USER_STATUS
    import config
    from main import bot
    from telebot import types
    import utils

    @bot.message_handler(func=lambda message: message.text
in [MENU['stats_pin_chats'], ])
    def stats_pin_chats(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        chats = database.get_activate_chats_users_arr()
        message_text = []
        need_check = []
```

```
        good_ids = []
        for chat in chats:
            type_chat = ''
            if chat['type_chat'] == 'ACTIVATE_SKU':
                if chat['status_pin'] == 'UN_ACTIVATE_KEY':
                    need_check.append(chat)
                    continue
                elif chat['status_pin'] is None:
                    type_chat = '⭐'
                elif chat['status_pin'] == 'ACTIVATE_KEY':
                    type_chat = '🔲'
            elif chat['type_chat'] == 'ACTIVATE_SKU_DONE':
                if chat['status_pin'] == 'UN_ACTIVATE_KEY':
                    continue

                if chat['status_pin'] is None \
                        and chat['last_update_type_chat_at']
+ datetime.timedelta(days=1) < datetime.datetime.now():
                    continue

                type_chat = '🔲'

            if type_chat in ['🔲', '🔲']:
                message_text.append(f'{type_chat}
{chat["create_at"].strftime("%d.%m                %H:%M")}
<code>{chat["id_i"]}</code> '
                                    f'@{chat["username"]}
{chat["first_name"]}, '
                                    f'{chat["title"]}')
            else:
                message_text.append(f'{type_chat}
{chat["create_at"].strftime("%d.%m                %H:%M")}
<code>{chat["id_i"]}</code> '
                                    f'{chat["title"]}')
            good_ids.append(chat['id_chat'])
        for chat in need_check:
            if chat["id_chat"] in good_ids:
                continue

            type_chat = '🔴'
```

```
            message_text.append(f'{type_chat}
{chat["create_at"].strftime("%d.%m                %H:%M")}
<code>{chat["id_i"]}</code> ' \
                                f'{chat["title"]}')
        if len(message_text) == 0:
            bot.reply_to(message, 'Немає такого чата')
            return

        bot.reply_to(
            message,
            "\n".join(message_text),
            parse_mode='html'
        )
```

10. Управління підключенням менеджерів

```
    import threading

    import database
    import models
    from models import UserManager
    from config import MENU, USER_STATUS
    import config
    from main import bot
    import utils
    from telebot import types

    locks = {}
    locks2 = {}
    lock_type1 = threading.Lock()
    lock_type2 = threading.Lock()

    @bot.callback_query_handler(func=lambda          call:
call.data.split('_')[0] in ['reply-message', ], )
    def history_transfer_data(call):
        user = UserManager().get_user(call.from_user.id)
        if user.lvl == 0:
            return

        id_chat = call.data.split('_')[-1]
        try:
            id_m = int(call.data.split('_')[-2])
```

```
        except:
            id_m = None
        chat = database.get_chat_by_id(id_chat)
        if not chat:
            try:
                bot.answer_callback_query(
                    callback_query_id=call.id,
                    text='Поточного чата поки немає в базі'
                )
            except:
                pass
            return

        lock_type1.acquire()
        if id_chat not in locks.keys():
            locks[id_chat] = threading.Lock()
        lock_type1.release()

        locks[id_chat].acquire()
        try:
            chat = database.get_chat_by_id(id_chat)

            if id_m is not None:
                messages                                        =
database.get_messages_digi_by_chat(chat['id_chat'])[::-1]
                status = False
                for message in messages:
                    if message['id_m'] == id_m:
                        status = True
                        continue
                    if not status:
                        continue
                    if message['seller'] == 1:
                        try:
                            bot.answer_callback_query(
                                callback_query_id=call.id,
                                text='На це повідомлення вже
відповідали'
                            )
                        except:
                            pass
                        open_chat                               =
types.InlineKeyboardMarkup()
```

```
                        open_chat.add(
                            types.InlineKeyboardButton(
                                f'👉 Відкрити чат',
                                callback_data=f'open-
chat_{chat["id_chat"]}'),
                        )
                        bot.edit_message_text(
                            message_id=call.message.id,
                            chat_id=call.message.chat.id,
                            text=call.message.text + '\n\nHa
це повідомлення вже відповідали без натиснення кнопки',
                            entities=call.message.entities,
                            reply_markup=open_chat
                        )
                        return

            new_connect = True
            update_connect_pin = False
            update_connect_un_activate = False
            connected_users                                 =
database.get_connected_users(id_chat=chat['id_chat'])
            utils.send_message(
                chat_id=config.CHANNEL_LOG,
                message=f'{chat=}\n{connected_users=}'
            )
            for connected_user in connected_users:
                if   str(connected_user['telegram_id'])   ==
str(call.from_user.id):
                    if    connected_user['status_pin']    in
['PIN', 'ACTIVATE_KEY', ]:
                        new_connect = False
                        update_connect_pin = False
                        update_connect_un_activate = False
                        break

                    if    connected_user['status_pin']    in
['UN_PIN', 'DELETE_PIN']:
                        update_connect_pin                 =
connected_user['id_pin']
                        new_connect = False
                        break
```

```
                    if    connected_user['status_pin']    in
['UN_ACTIVATE_KEY', ]:
                        update_connect_un_activate         =
connected_user['id_pin']
                        new_connect = False
                        break

            else:
                if len(connected_users) > 0:
                    managers          =          [f'Менеджер:
@{connected_user["username"]},
{connected_user["telegram_id"]}'   for   connected_user   in
connected_users if connected_user['status_pin'] in ['PIN',
'ACTIVATE_KEY', ]]
                    if len(managers) > 0:
                        bot.edit_message_text(
                            message_id=call.message.id,
                            chat_id=call.message.chat.id,
                            text=call.message.text + f'\n\n'
+ "\n".join(managers),
                            entities=call.message.entities
                        )
                        try:
                            bot.answer_callback_query(
                                callback_query_id=call.id,
                                text='Цей      чат      уже
закреплений за іншним менеджером'
                            )
                        except:
                            pass
                        return

            user.status = USER_STATUS['send-message'] + '_'
+ str(chat["id_chat"])
            user.last_id_chat = chat["id_chat"]
            if new_connect:
                database.connect_chat(
                    id_chat=chat['id_chat'],
                    id_user=user.db_id_user,
                    type_connect='PIN'
                )

            elif update_connect_pin is not False:
```

```
                database.update_connect_chat(
                    id_pin=update_connect_pin,
                    type_connect='PIN')

            elif update_connect_un_activate is not False:
                database.update_connect_chat(
                    id_pin=update_connect_pin,
                    type_connect='ACTIVATE_KEY')

            if call.message.text is not None:
                bot.edit_message_text(
                    message_id=call.message.id,
                    chat_id=call.message.chat.id,
                    text=call.message.text + f'\n\nМенеджер:
@{call.from_user.username}, {call.from_user.first_name}',
                    entities=call.message.entities
                )
            else:
                bot.edit_message_caption(
                    message_id=call.message.id,
                    chat_id=call.message.chat.id,
                    caption=call.message.caption           +
f'\n\nМенеджер:                @{call.from_user.username},
{call.from_user.first_name}',
                    caption_entities=call.message.caption_en
tities
                )

            utils.send_history(bot,
telegram_id=call.from_user.id, id_chat=chat["id_chat"])
            bot.send_message(
                chat_id=call.from_user.id,
                text=f'Ви     відповідаєте     на     ордер
#{chat["id_i"]} {chat["email"]}\n',
                reply_markup=config.SEND_MESSAGE_MENU
            )
            try:
                bot.answer_callback_query(
                    callback_query_id=call.id,
                    text='Перейдіть в бота для відпправки
повідомлення вказаному клієнту.'
                )
            except:
```

```
                pass
        finally:
            locks[id_chat].release()

    @bot.callback_query_handler(func=lambda           call:
call.data.split('_')[0] in ['connect', ], )
    def connect(call):
        user = UserManager().get_user(call.from_user.id)
        if user.lvl == 0:
            return

        id_i = call.data.split('_')[-1]
        chat = database.get_chat(id_i)
        if not chat:
            try:
                bot.answer_callback_query(
                    callback_query_id=call.id,
                    text='Поточного чата поки немає в базі'
                )
            except:
                pass
            return

        lock_type2.acquire()
        if id_i not in locks2.keys():
            locks2[id_i] = threading.Lock()
        lock_type2.release()

        locks2[id_i].acquire()
        try:
            chat = database.get_chat(id_i)
            new_connect = True
            update_connect_pin = False
            connected_users                                    =
database.get_connected_users(id_chat=chat['id_chat'])
            for connected_user in connected_users:
                if   str(connected_user['telegram_id'])   ==
str(call.from_user.id):
                    if    connected_user['status_pin']    in
['ACTIVATE_KEY', ]:
                        new_connect = False
                        update_connect_pin = False
                        break
```

```
                    if    connected_user['status_pin']    in
['UN_PIN', 'UN_ACTIVATE_KEY', 'PIN', 'DELETE_PIN']:
                        update_connect_pin                  =
connected_user['id_pin']
                        new_connect = False
                        break

            else:
                if len(connected_users) > 0:
                    managers          =          [f'Менеджер:
@{connected_user["username"]},
{connected_user["telegram_id"]}' for
                                    connected_user            in
connected_users if
                                    connected_user['status_pin']
in ['ACTIVATE_KEY', ]]
                    if len(managers) > 0:
                        bot.edit_message_text(
                            message_id=call.message.id,
                            chat_id=call.message.chat.id,
                            text=call.message.text + f'\n\n'
+ "\n".join(managers),
                            entities=call.message.entities
                        )
                        try:
                            bot.answer_callback_query(
                                callback_query_id=call.id,
                                text='Цей      чат      вже
закріпленний за іншим менеджером'
                            )
                        except:
                            pass
                        return

            user.status = USER_STATUS['send-message'] + '_'
+ str(chat["id_chat"])
            user.last_id_chat = chat["id_chat"]
            if new_connect:
                database.connect_chat(
                    id_chat=chat['id_chat'],
                    id_user=user.db_id_user,
                    type_connect='ACTIVATE_KEY'
```

```
                )

            elif update_connect_pin is not False:
                database.update_connect_chat(
                    id_pin=update_connect_pin,
                    type_connect='ACTIVATE_KEY')

            #
database.update_type_chat(id_chat=chat["id_chat"],
type_chat='ACTIVATE_SKU_IN_WORK')
            utils.send_history(bot,
telegram_id=call.from_user.id, id_chat=chat["id_chat"])
            bot.edit_message_text(
                message_id=call.message.id,
                chat_id=call.message.chat.id,
                text=call.message.text   +   f'\n\nМенеджер:
@{call.from_user.username}, {call.from_user.first_name}',
                entities=call.message.entities
            )
            bot.send_message(
                chat_id=call.from_user.id,
                text=f'Ви     відповідаєте     на     ордер
#{chat["id_i"]} ({chat["email"]})',
                reply_markup=config.SEND_MESSAGE_MENU
            )
            try:
                bot.answer_callback_query(
                    callback_query_id=call.id,
                    text='Перейдіть  в  бота  для  відпправки
повідомлення вказаному клієнту.'
                )
            except:
                pass

        finally:
            locks2[id_i].release()
```

11. Модуль головного меню

```
import database
import models
from models import UserManager
from config import MENU, USER_STATUS
```

```
    import config
    from main import bot
    from telebot import types

    @bot.message_handler(func=lambda message: message.text
in [MENU['messages'], ])
    def messages(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        user.status = USER_STATUS['messages']

        chats = database.get_chats_users(user.db_id_user,
'PIN')
        bot.send_message(message.chat.id,
                         f'Відкрито меню чатів',
                         parse_mode='html',
                         reply_markup=config.MESSAGES_MENU)

        if not chats or len(chats) == 0:
            bot.send_message(message.chat.id,
                             f'У Вас немає активних чатів.',
                             parse_mode='html',
                             reply_markup=config.MESSAGES_ME
NU)
            return

        bot.send_message(message.chat.id,
                         f'У Вас всього {len(chats)} чатів.',
                         parse_mode='html',
                         reply_markup=config.get_keyboard_ch
ats(
                             chats=chats,
                             offset=0,
                             type_board='next-pin'
                         ))

    @bot.message_handler(func=lambda message: message.text
in [MENU['history_messages'], MENU['history_pin_chats'], ])
    def history_messages(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
```

```
            return

        user.status = USER_STATUS['history_messages']

        data = {
            MENU['history_messages']: {
                'arr': ['PIN', 'UN_PIN'],
                'type_board': 'next-h-pin',
                'reply_markup': config.MESSAGES_MENU
            },
            MENU['history_pin_chats']: {
                'arr': 'UN_ACTIVATE_KEY',
                'type_board': 'next-h-activate-key',
                'reply_markup': config.PIN_CHATS_MENU
            }
        }
        if type(data[message.text]['arr']) is str:
            chats                                                =
database.get_chats_users(user.db_id_user,
data[message.text]['arr'])
        else:
            chats                                                =
database.get_chats_users_arr(user.db_id_user,
data[message.text]['arr'])

        if not chats or len(chats) == 0:
            bot.send_message(message.chat.id,
                             f'У Вас немає архівних чатів.',
                             parse_mode='html',
                             reply_markup=data[message.text]
['reply_markup'])
            return

        bot.send_message(message.chat.id,
                         f'У Вас всього {len(chats)} чатів.',
                         parse_mode='html',
                         reply_markup=config.get_keyboard_ch
ats(
                             chats=chats,
                             offset=0,
                             type_board=data[message.text]['
type_board']
                         ))
```

```
    @bot.message_handler(func=lambda message: message.text
in [MENU['pin_chats'], ])
    def pin_chats(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        user.status = USER_STATUS['pin_chats']

        chats  =  database.get_chats_users(user.db_id_user,
'ACTIVATE_KEY')
        bot.send_message(message.chat.id,
                         f'Відкрито меню чатів',
                         parse_mode='html',
                         reply_markup=config.PIN_CHATS_MENU)

        if not chats or len(chats) == 0:
            bot.send_message(message.chat.id,
                             f'У Вас немає активних чатів.',
                             parse_mode='html',
                             reply_markup=config.PIN_CHATS_M
ENU)
            return

        bot.send_message(message.chat.id,
                         f'У Вас всього {len(chats)} чатів.',
                         parse_mode='html',
                         reply_markup=config.get_keyboard_ch
ats(
                             chats=chats,
                             offset=0,
                             type_board='next-activate-key'
                         ))

    @bot.message_handler(func=lambda message: message.text
in [MENU['history'], ])
    def history(message):
        user = UserManager().get_user(message.chat.id)
        if user.lvl == 0:
            return

        user.status = USER_STATUS['pin_chats']
```

```
        chats = database.get_chats()

        if not chats or len(chats) == 0:
            bot.send_message(message.chat.id,
                             f'У Вас немає активних чатів.',
                             parse_mode='html',
                             reply_markup=config.HOME_MENU)
            return

        bot.send_message(message.chat.id,
                         f'Всього {len(chats)} чатів.',
                         parse_mode='html',
                         reply_markup=config.get_keyboard_ch
ats(
                             chats=chats,
                             offset=0,
                             type_board='next-all'
                         ))

    @bot.callback_query_handler(func=lambda           call:
call.data.split('_')[0] in ['next-h-pin',

                    'next-h-activate-key',

                    'next-activate-key',

                    'next-pin',

                    'next-all',

                    ], )
    def next_by_callback(call):
        user = UserManager().get_user(call.from_user.id)
        if user.lvl == 0:
            return
        data = {
                'next-h-pin': ['PIN', 'UN_PIN'],
                'next-h-activate-key': 'UN_ACTIVATE_KEY',
                'next-activate-key': 'ACTIVATE_KEY',
                'next-pin': 'PIN',
                'next-all': None,
        }
```

```
        if type(data[call.data.split('_')[0]]) is str:
            chats                                                    =
database.get_chats_users(user.db_id_user,
data[call.data.split('_')[0]])
        elif data[call.data.split('_')[0]] is None:
            chats = database.get_chats()
        else:
            chats                                                    =
database.get_chats_users_arr(user.db_id_user,
data[call.data.split('_')[0]])

        bot.edit_message_reply_markup(
            call.message.chat.id,
            call.message.id,
            reply_markup=config.get_keyboard_chats(
                chats=chats,
                offset=int(call.data.split('_')[-1]),
                type_board=call.data.split('_')[0]
            )
        )
```

12. Вставка нових повідомлень

```
import logging

import config
import database
import crm
import datetime
import time
import utils

def main():
    try:
        tk = crm.get_token()
        start_at = datetime.datetime.now()
        check = True
        while True:
            try:
                get_chats(tk, 1, check)
                check = False
            except Exception as ex:
```

```
                    if       (datetime.datetime.now()       -
start_at).total_seconds() >= 50:
                        raise Exception(ex)
                    continue
                break
        except Exception as ex:
            utils.send_message(
                chat_id=config.CHANNEL_LOG,
                message=f'#ERROR update_messages {ex}, next
iter',
            )

    def get_chats(tk, page=1, check=True):
        logging.warning(f'{page=}')
        error = 0
        next_page = False
        cnt_change = 0
        chats = crm.get_chats(tk=tk, page=page)
        for chat in chats.get('chats'):
            if    str(chat['cnt_msg'])    ==    '1'    and
str(chat['cnt_new']) == '0':
                continue
            check_messages = True
            logging.warning(f'{chat=}')
            info_chat = database.get_chat(id_i=chat['id_i'])
            its_new = False
            if not info_chat:
                its_new = True
                database.create_chat(
                    email=chat['email'],
                    product=chat['product'],
                    id_i=chat['id_i'],
                    title=chat['product'],
                    last_date=chat['last_date'],
                    cnt_msg=chat['cnt_msg'],
                    cnt_new=chat['cnt_new']
                )
                info_chat                                    =
database.get_chat(id_i=chat['id_i'])
                next_page = True

            if  its_new  or  str(info_chat['last_date'])  !=
str(chat['last_date'])   or   str(info_chat['cnt_msg'])   !=
```

```
str(chat['cnt_msg'])    or    str(info_chat['cnt_new'])    !=
str(chat['cnt_new']):
                try:
                    cnt_change = get_messages(
                        tk,    id_chat=info_chat['id_chat'],
id_i=info_chat['id_i'], info_chat=info_chat
                    )
                except Exception as ex:
                    logging.warning(ex)
                    error += 1
                    continue
                else:
                    database.update_chat(
                        info_chat['id_chat'],
                        last_date=chat['last_date'],
                        cnt_msg=chat['cnt_msg'],
                        cnt_new=chat['cnt_new'])
                next_page = True

            if cnt_change > 0:
                next_page = True

        if   datetime.datetime.strptime(chats.get('chats')[-
1]['last_date'],        '%Y-%m-%d        %H:%M:%S')        <
datetime.datetime.now() - datetime.timedelta(days=30):
            return

        # get_chats(tk, page=page+1)
        if next_page:
            return get_chats(tk, page=page+1)

        if error == 0:
            return

        if   datetime.datetime.strptime(chats.get('chats')[-
1]['last_date'],        '%Y-%m-%d        %H:%M:%S')        <
datetime.datetime.now() - datetime.timedelta(days=1):
            return

        get_chats(tk, page=page+1)

    def get_messages(tk, id_chat, id_i, info_chat):
        messages = crm.get_messages(tk, id_i)
```

```
        connected_users = None
        cnt_change = 0
        for message in messages:
            message_info                                    =
database.get_message_digi(id_m=message['id'])
            if not message_info:
                cnt_change += 1

                database.create_message_digi(
                    id_chat=id_chat,
                    id_m=message.get('id'),
                    message_text=f"{message.get('message')}"
.replace('<br>', '\n'),
                    seller=message.get('seller'),
                    buyer=message.get('buyer'),
                    deleted=message.get('deleted'),
                    date_written=message.get('date_written')
,
                    date_seen=message.get('date_seen'),
                    is_file=message.get('is_file'),
                    filename=message.get('filename'),
                    url=message.get('url'),
                    is_img=message.get('is_img'),
                    preview=message.get('preview')
                )
                if message.get('buyer') == 1:
                    if not connected_users:
                        connected_users = []
                        db_connected_users                  =
database.get_connected_users(id_chat)
                        for       db_connected_user       in
db_connected_users:
                            if
db_connected_user['status_pin'] in ['PIN', 'ACTIVATE_KEY']:
                                connected_users.append(db_co
nnected_user)

                    if      not      connected_users      or
len(connected_users) == 0:
                        link_for_reply_channel = {
                            'inline_keyboard':    [[{'text':
'✉ Відповісти',
```

```
                                                    'callback_
data':  f'reply-message_{message.get("id")}_{id_chat}'},  ],
]}
                        message_add = ''
                        disable_notification = False
                    else:
                        link_for_reply_channel = None
                        disable_notification = True
                        message_add = '\n'
                        for         connected_user         in
connected_users:
                            message_add  +=  f'\nЗакріплений
менеджер:                       @{connected_user["username"]}
{connected_user["first_name"]}'

                    link_for_reply_user = {
                        'inline_keyboard':        [[{'text':
'⚡Відкрити чат',
                                                    'callback_data
': f'open-chat-new_{id_chat}'}, ], ]}

                    message_template = '📪 Повідомлення від
клієнта.\n\n' \
                                       f'{info_chat["title"]
}\n' \
                                       f'Рахунок:         <a
href="https://buknadia.keeping.com/purchases_inv_detail?id_
i={id_i}&view=debate">{id_i}</a> ' \
                                       f'Клієнт:
{info_chat["email"]}\n'

                    if not message.get('is_img'):
                        if message.get('is_file'):
                            this_message                    =
f'{message_template}' \
                                                f'<b>Повідомлення
:</b>                                                   <a
href="{message.get("url")}">{message.get("message")}</a>',
                        else:
                            this_message                    =
f'{message_template}<b>Повідомлення:</b>
{message.get("message")}'
```

```
                        utils.send_message(
                            chat_id=config.CHAT_NEW_MESSAGE,
                            message=f'{this_message}{message
_add}',
                            link=link_for_reply_channel,
                            insert_id_chat=id_chat,
                            id_m=message.get('id'),
                            disable_notification=disable_not
ification
                        )

                        for         connected_user         in
connected_users:
                            utils.send_message(
                                chat_id=connected_user['tele
gram_id'],
                                message=this_message,
                                link=link_for_reply_user,
                                insert_id_chat=id_chat,
                                id_m=message.get('id')
                            )

                    elif message.get('is_img'):
                        utils.send_photo(
                            chat_id=config.CHAT_NEW_MESSAGE,
                            caption=f'{message_template}<b>П
овідомлення:</b> {message.get("message")}{message_add}',
                            link=link_for_reply_channel,
                            photo=message.get('url'),
                            insert_id_chat=id_chat,
                            id_m=message.get('id'),
                            disable_notification=disable_not
ification
                        )

                        for         connected_user         in
connected_users:
                            utils.send_photo(
                                chat_id=connected_user['tele
gram_id'],
                                caption=f'{message_template}
<b>Повідомлення:</b> {message.get("message")}',
```

```
                                    link=link_for_reply_user,
                                    photo=message.get('url'),
                                    insert_id_chat=id_chat,
                                    id_m=message.get('id')
                                )

                logging.warning(f'create_message:
{message}')
                # print('create_message')
        return cnt_change

    if __name__ == '__main__':
        main()
```

13. Код з використанням методу WebHook для взаємодії з TelegramAPI

```
    import datetime
    import flask
    import telebot
    import config
    from config import WEBHOOK_SSL_CERT, WEBHOOK_SSL_PRIV,
WEBHOOK_URL_PATH, WEBHOOK_URL_BASE
    from config import WEBHOOK_LISTEN, WEBHOOK_PORT
    import main as main_bot
    import handlers
    import time
    import logging

    import utils

    app = flask.Flask(__name__)

    logging.basicConfig(filename=f'log/debug_{datetime.date
time.now()}.log', level=logging.INFO,
                        format='%(asctime)s    %(levelname)s
%(name)s %(message)s')
    logging.Formatter.converter = time.gmtime

    @app.route('/', methods=['GET', 'HEAD'])
    def index():
        return ''

    @app.route(WEBHOOK_URL_PATH, methods=['POST'])
```

```
    def webhook():
        if   flask.request.headers.get('content-type')    ==
'application/json':
            json_string                                    =
flask.request.get_data().decode('utf-8')
            update                                         =
telebot.types.Update.de_json(json_string)
            main_bot.process_new_updates([update])
            return ''
        else:
            flask.abort(403)

    def start():
        while True:
            try:
                main_bot.remove_webhook()
                time.sleep(0.5)
                main_bot.set_webhook(url=WEBHOOK_URL_BASE  +
WEBHOOK_URL_PATH,
                                     certificate=open(WEBHOO
K_SSL_CERT, 'r'))

                app.run(
                    host=WEBHOOK_LISTEN,
                    port=WEBHOOK_PORT,
                    debug=False,
                    ssl_context=(WEBHOOK_SSL_CERT,
WEBHOOK_SSL_PRIV),
                )

            except Exception as ex:
                utils.send_message(
                    chat_id=config.CHANNEL_LOG,
                    message=f'#ERROR {ex}'
                )
                logging.error(str(ex))
                time.sleep(5)
                continue

    if __name__ == '__main__':
        start()
```

14. Код конфігації меню:

```
from telebot import types

MENU = {
            'messages': '🔥 Мої переписки',
            'pin_chats': '⚡ Закріплені чати',
            'stats_pin_chats': '📊 Статистика',
            'guest_chats': '😸 Чати з гостями',

            'history_messages': '📁 Історія моїх переписок',
            'history_pin_chats': '📁 Історія моїх закріплених чатів',
            'history_guest_chats': '📁 Історія моїх чатів з гостями',
            'history': '📁 Історія всіх чатів',

            'history_chat': '📁 Історія чата',
            'send_message_to_chat': '📤 Написати повідомлення',
        'send_status_read': '✅ Надіслати статус "Прочитано"',
        'edit_status_activate_key': '✅ Змінити статус діалогу',
            'un_status_chat': '🤦‍♂️ Зняти з мене закріп',
            'back': '◀ Назад',
            'back_to_messages': '🔙 Назад',
            'back_to_chat': '◀ Вернутися назад до чату',

            }

NONE_KEYBOARD = types.ReplyKeyboardRemove()
```

```
    HOME_MENU                                                    =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    HOME_MENU.row(MENU['messages'], MENU['pin_chats'], )
    # HOME_MENU.row(MENU['guest_chats'], MENU['history'], )
    HOME_MENU.row(MENU['history'], )


    MESSAGES_MENU                                                =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    MESSAGES_MENU.row(MENU['history_messages'],
MENU['back'], )


    PIN_CHATS_MENU                                               =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    PIN_CHATS_MENU.row(MENU['history_pin_chats'],
MENU['stats_pin_chats'], MENU['back'], )


    GUEST_CHATS_MENU                                             =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    GUEST_CHATS_MENU.row(MENU['history_guest_chats'],
MENU['back'], )


    CHAT_MENU                                                    =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    CHAT_MENU.row(MENU['history_chat'],
MENU['send_message_to_chat'], )
    CHAT_MENU.row(MENU['send_status_read'],
MENU['un_status_chat'], MENU['back'], )


    CHAT_MENU2                                                   =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    CHAT_MENU2.row(MENU['history_chat'],
MENU['send_message_to_chat'], )
    CHAT_MENU2.row(MENU['edit_status_activate_key'],
MENU['send_status_read'], MENU['back'], )


    CHAT_MENU3                                                   =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    CHAT_MENU3.row(MENU['history_chat'],
MENU['send_message_to_chat'], )
```

```
    CHAT_MENU3.row(MENU['send_status_read'],  MENU['back'],
)


    CHOICE_CHAT_MENU = {
       'USER_CHAT': CHAT_MENU,
       'ACTIVATE_SKU': CHAT_MENU2,
       'ACTIVATE_SKU_ARCHIVE': CHAT_MENU,
       'ACTIVATE_SKU_DONE': CHAT_MENU2,
    }


    SEND_MESSAGE_MENU                                          =
types.ReplyKeyboardMarkup(resize_keyboard=True)
    SEND_MESSAGE_MENU.row(MENU['send_status_read'],
MENU['back_to_chat'], )
```

Додаток Г – Приклади інтерфейсу користувача

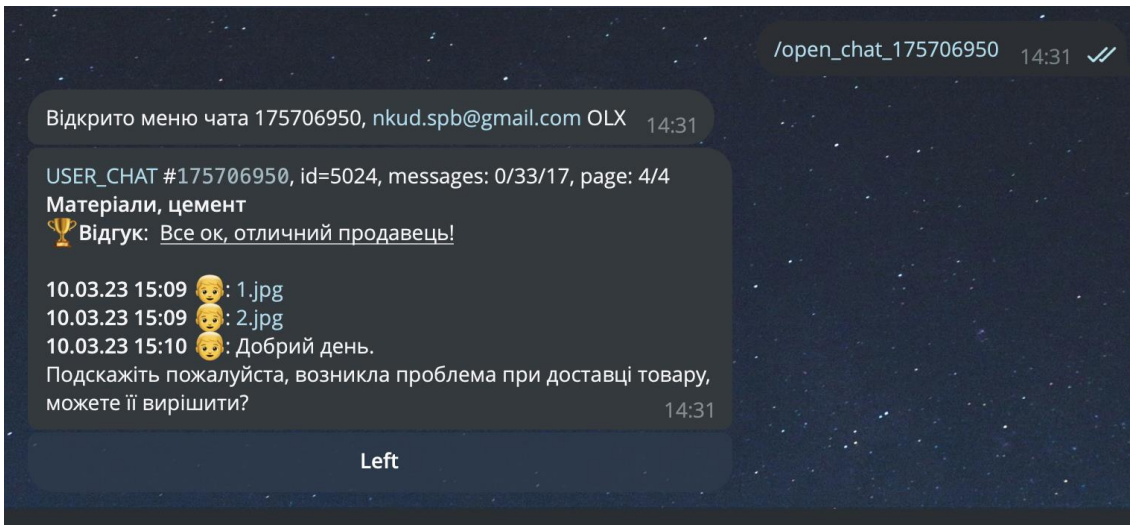

Рисунок Г.1 - Перегляд історії чату

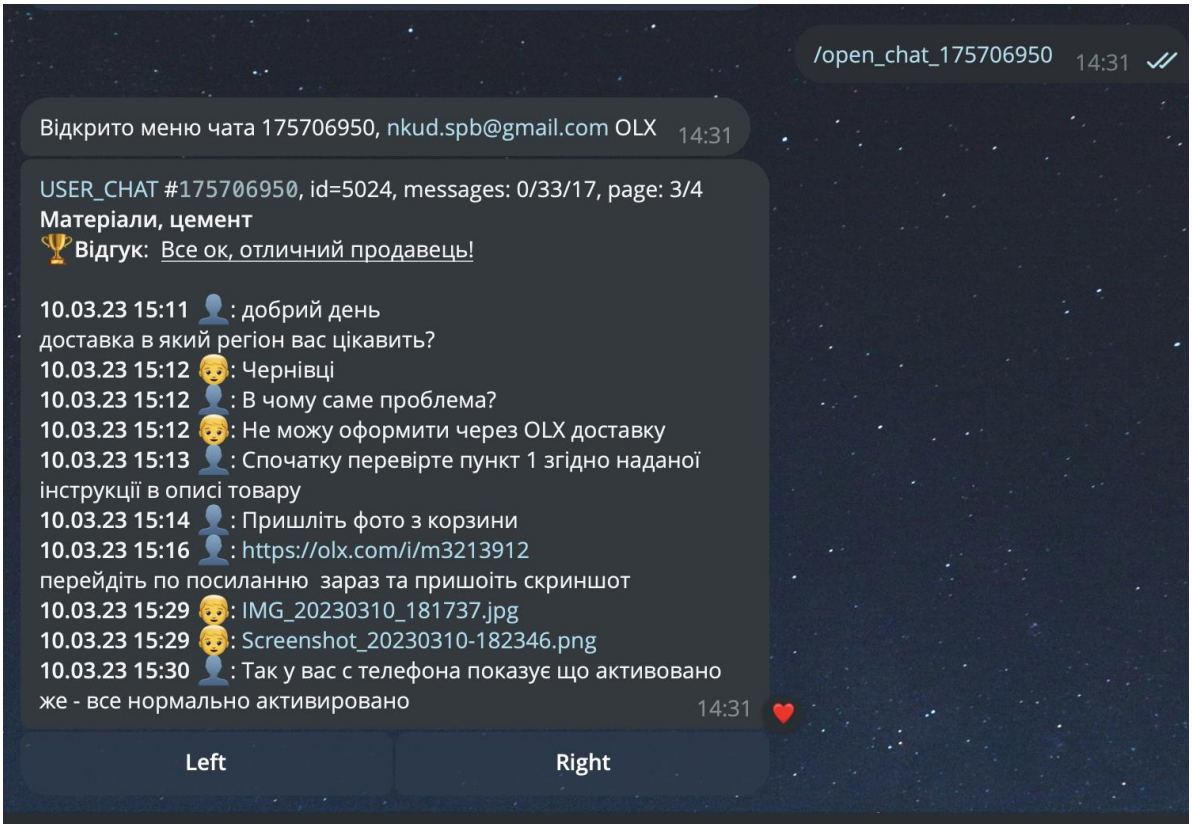

Рисунок Г.2 - Перегляд історії чату, після переходу на попередню сторінку

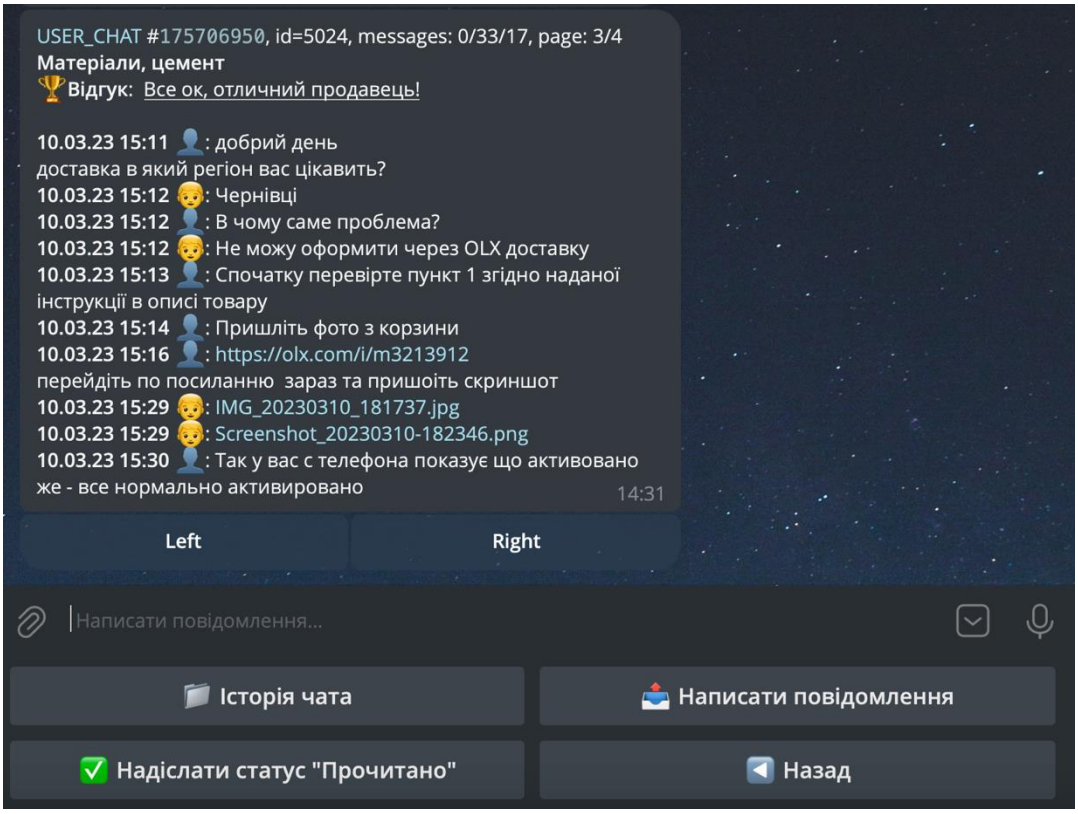

Рисунок Г.3 - Кнопки для користувача, під час перегляду історії чата

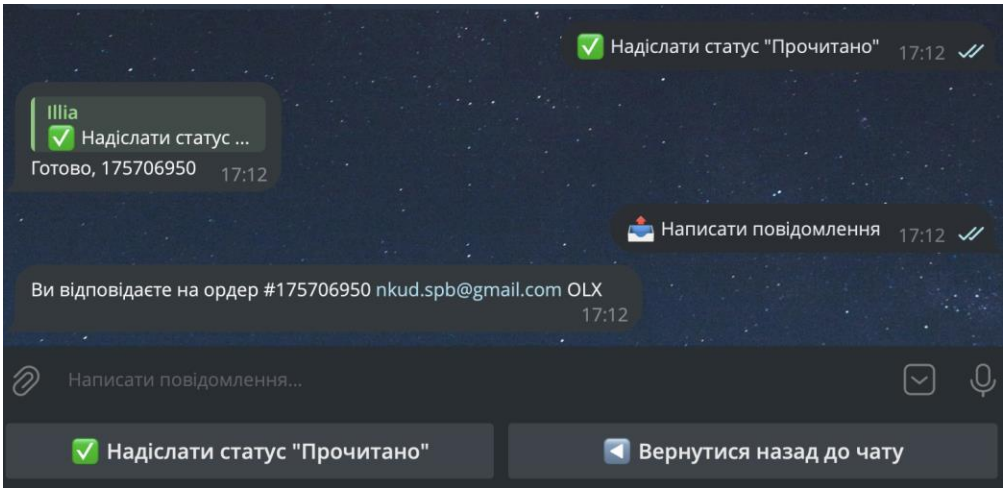

Рисунок Г.4 - Надсилання статусу «Прочитано» на повідомлення клієнта

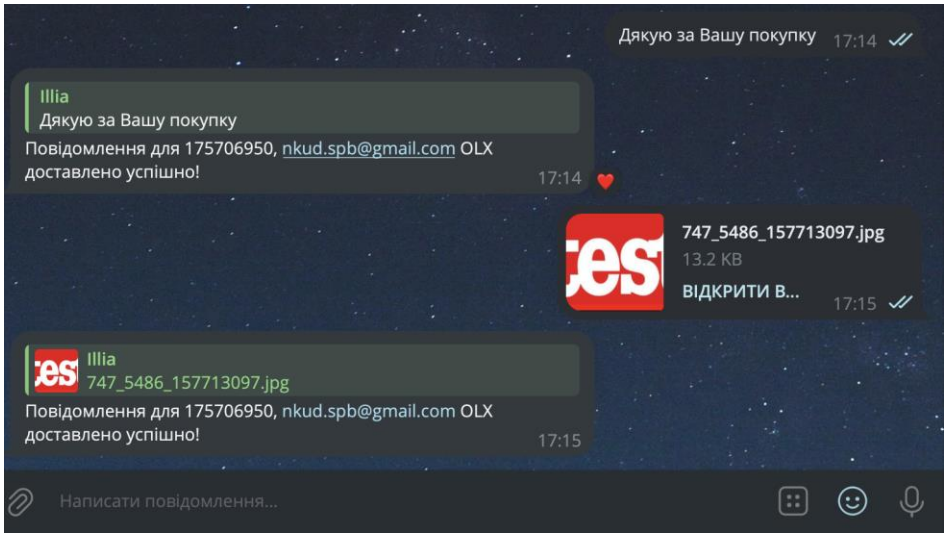

Рисунок Г.5 - Надсилання текстового повідомлення та зображення

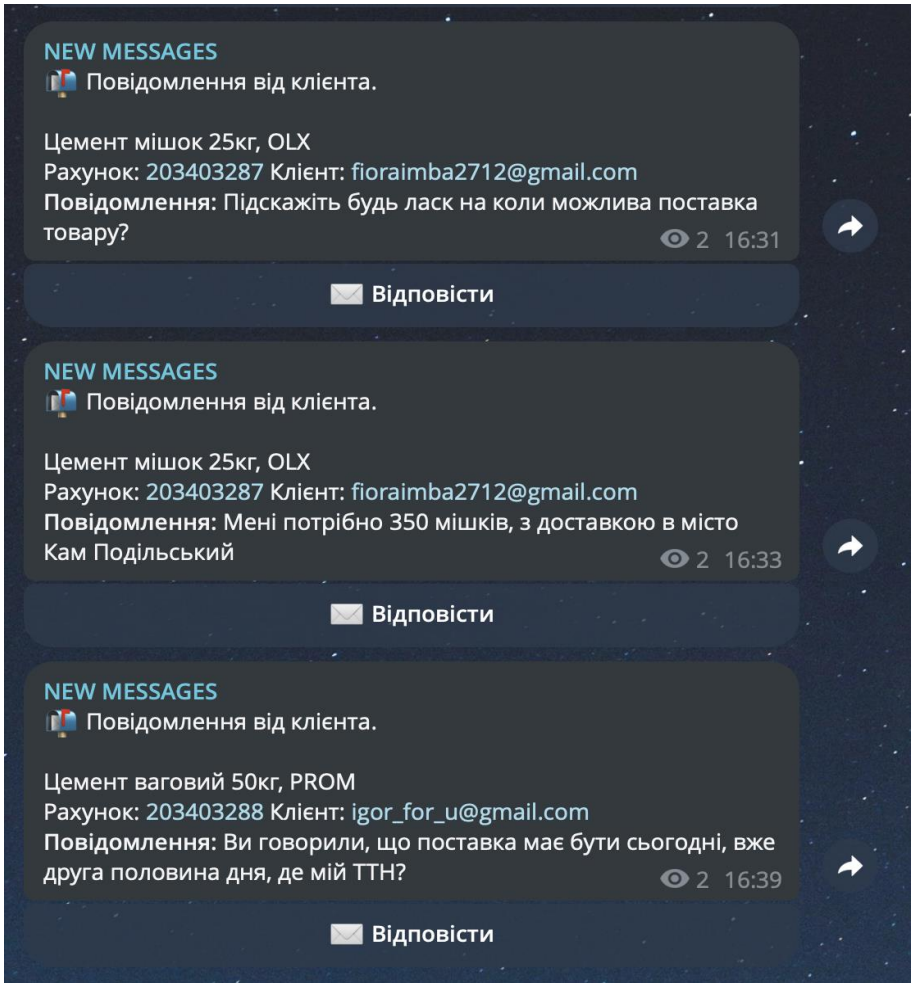

Рисунок Г.6 - Вигляд чату з усіма новими повідомленнями

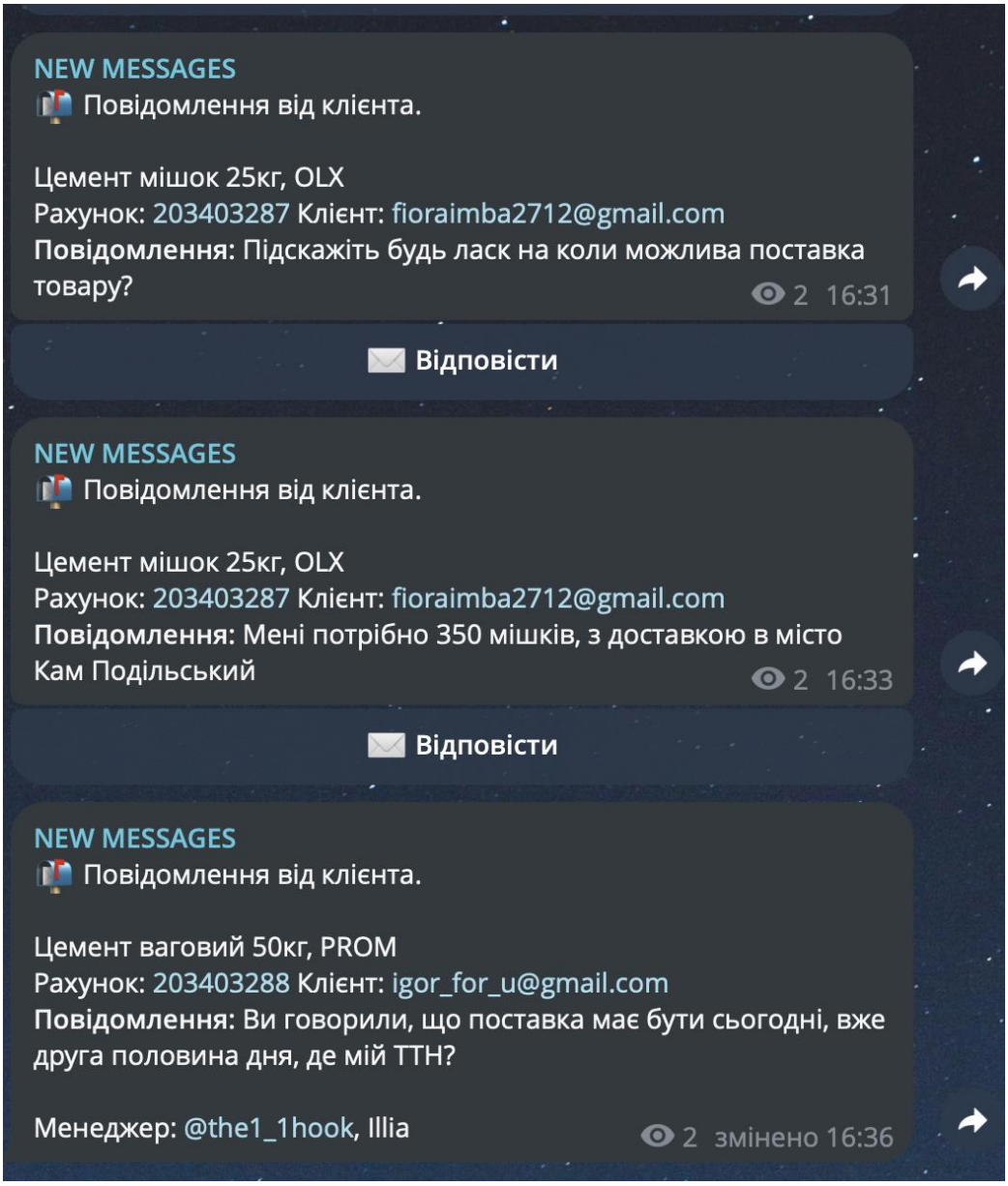

Рисунок Г.7 - Вигляд чату з усіма новими повідомленнями, після натиснення кнопки «Відповісти»

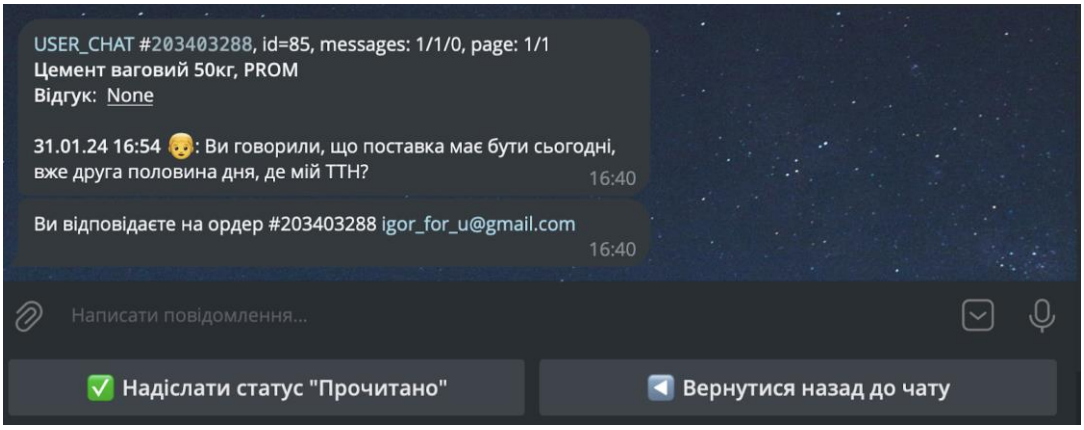

Рисунок Г.8 - Вигляд чату з ботом, після натиснення кнопки «Відповісти»

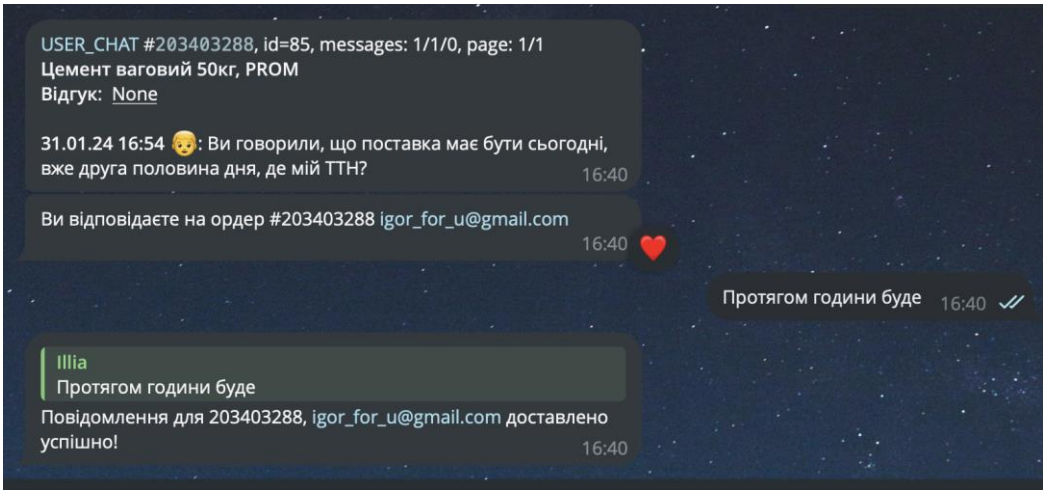

Рисунок Г.9 - Вигляд відповіді на повідомлення

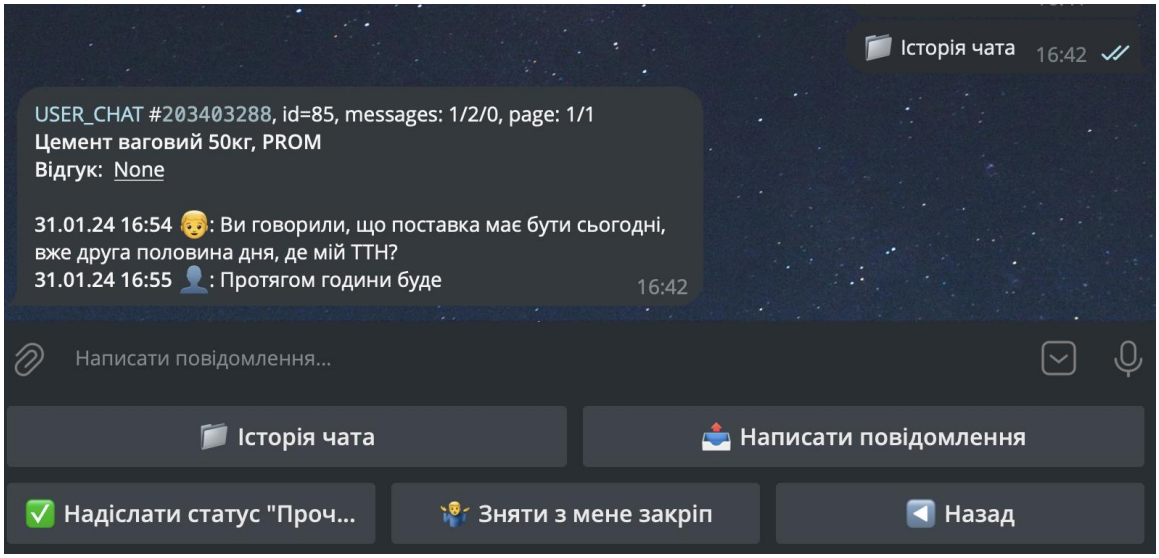

Рисунок Г.10 - Вигляд чату з ботом, після відповіді на повідомлення

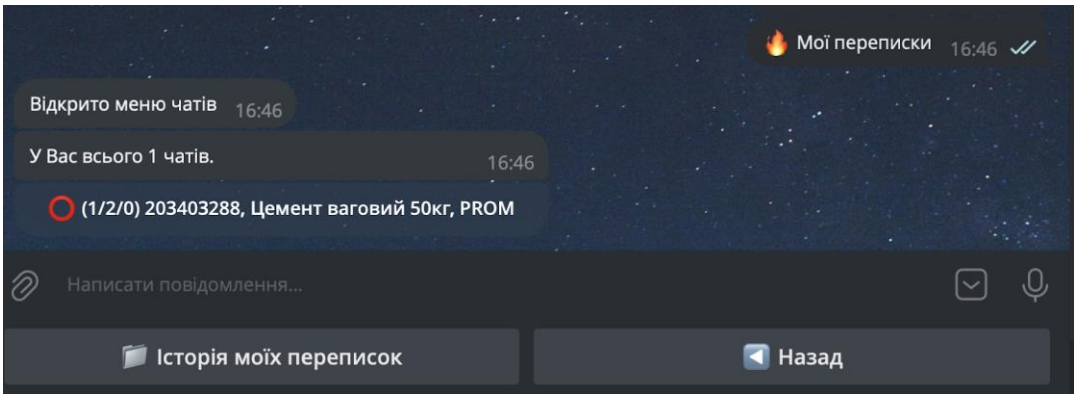

Рисунок Г.11 - Вигляд моїх переписок, після привязання чату за собою

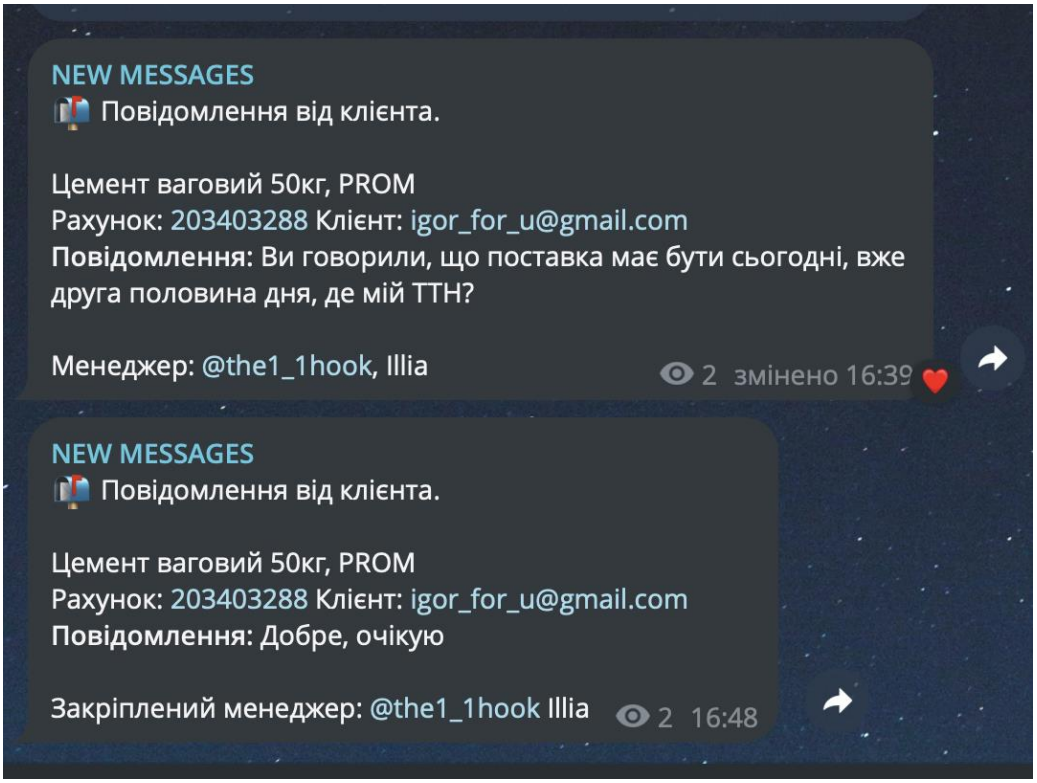

Рисунок Г.12 - Вигляд нового повідомлення, якщо чат вже раніше був закріплений за менеджером

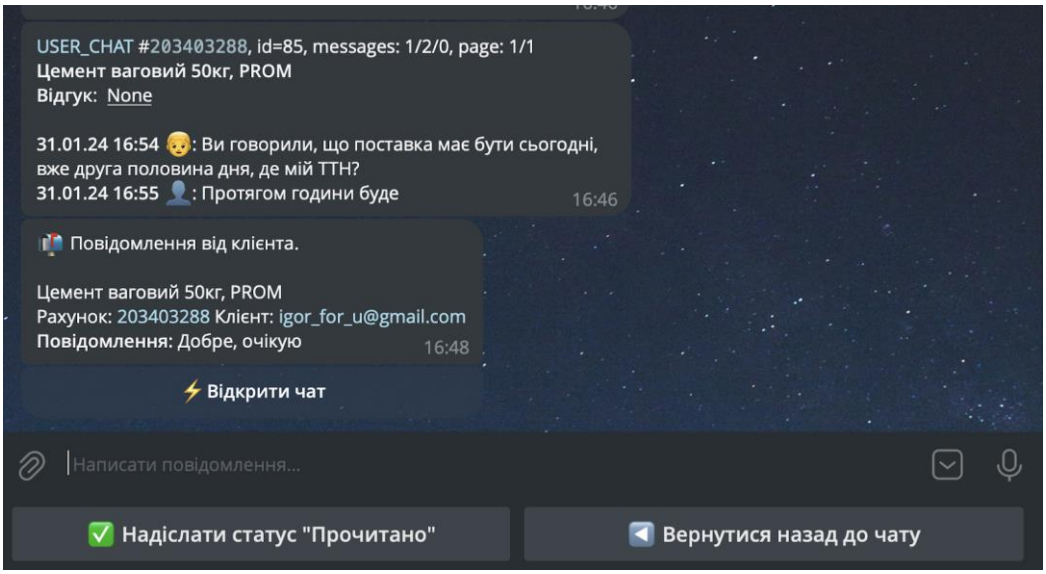

Рисунок Г.13 - Вигляд нового повідомлення, в чаті менеджера, який був закріплений за чатом

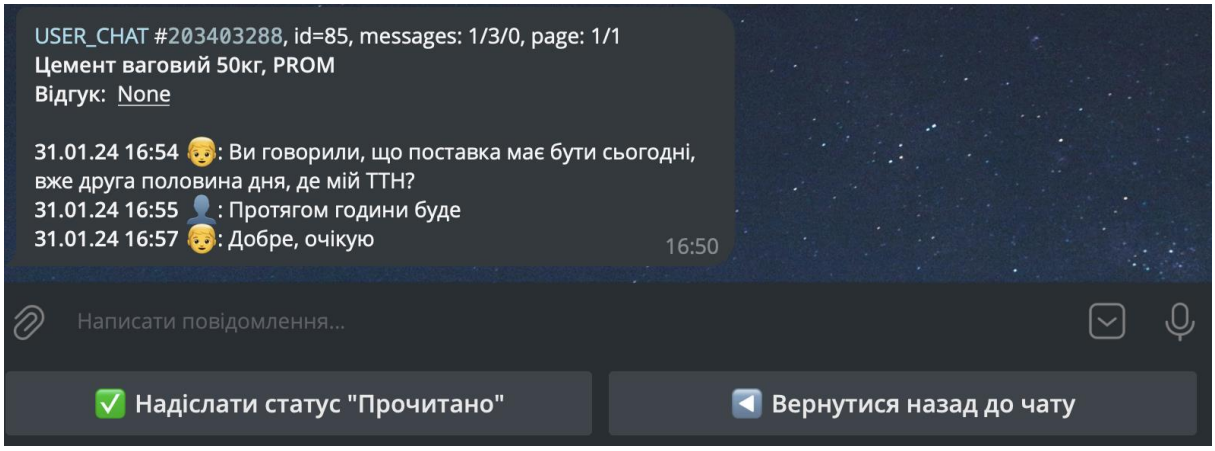

Рисунок Г.14 - Вигляд історії, після нового повідомлення, в чаті менеджера, який був закріплений за чатом

Додаток Д - Схема бази даних

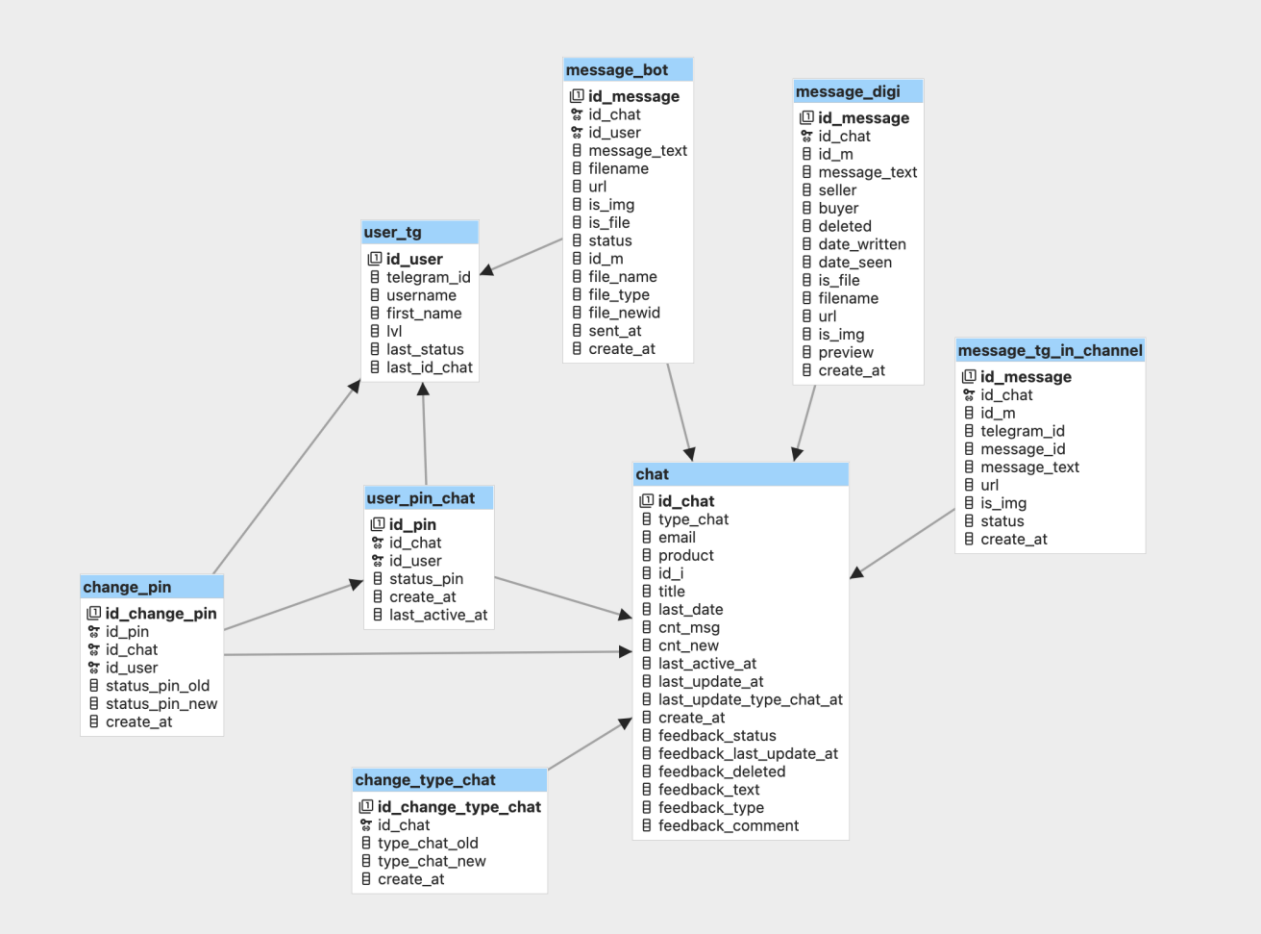

Рисунок Д.1 - Схема бази даних