Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(навчально-науковий інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

Здобувача вищої освіти Швеця Івана Анатолійовича
(ПІБ)
академічної групи 123м-23-1
(шифр)
спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури та параметрів комп'ютерної системи ІТ-компанії «Диверсіті ІТ» з ботом зворотного зв'язку»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Цвіркун Л.І.			
розділів:				
синтез системи	доц. Ткаченко С.М.			
розроблення програмного забезпечення	ас. Бешта Л.В.			
Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ _В.В. Гнатушенко
(підпис) (ініціали, прізвище)

«_____» _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра
(бакалавра, магістра)

здобувача вищої освіти _____Швецю І.А.____ **академічної групи** ____123м-23-1____
(прізвище та ініціали) (шифр)

спеціальності _____123 Комп'ютерна інженерія_____

за освітньою-професійною програмою _____«Комп'ютерна інженерія»_____
(офіційна назва)

на тему «Обґрунтування структури та параметрів комп'ютерної системи ІТ-компанії «Диверсіті ІТ» з ботом зворотного зв'язку»,
затверджену наказом ректора НТУ «Дніпровська політехніка» від 17 жовтня 2024 р. №1388-с

Розділ	Зміст	Термін виконання
Стан питання та постановка завдання	На основі матеріалів виробничих практик, інших науково-технічних джерел конкретизується предмет та мета роботи та виконується постановка завдання.	11.10.2024
Теоретичний	На основі аналізу ІТ-компанії формулюються та обираються методи й технології для розробки Telegram-боту та його бази даних, серверу	25.10.2024
Синтез системи	Визначаються цілі системи, формуються вимоги до системи та обирається апаратне забезпечення	15.11.2024
Розроблення програмного забезпечення	Розробка програмного забезпечення бота зворотного зв'язку	29.11.2024
Експериментальний розділ	Проведення і обробка результатів експериментів	06.12.2024

Завдання видано _____ проф. Л. І. Цвіркун
(підпис керівника) (ініціали, прізвище)

Дата видачі 06 вересня 2024 р.

Дата подання до екзаменаційної комісії 10.12.2024 р.

Прийнято до виконання _____ І.А. Швець
(підпис здобувача вищої освіти) (ініціали, прізвище)

РЕФЕРАТ

Пояснювальна записка: 110 с., 67 рис., 2 табл., 1 дод., 25 джерел.

КОМПАНІЯ, TELEGRAM, БОТ, КОМп'ЮТЕРНА МЕРЕЖА, БАЗА ДАНИХ, MONGO DB, PYTHON

Об'єкт розробки – комп'ютерна мережа IT-компанії «Диверсіті IT» з розробкою боту зворотного зв'язку.

Мета роботи – розробка та впровадження комп'ютерної мережі IT-компанії «Диверсіті IT», яка обладнана ботом зворотного зв'язку.

В першому розділі було розглянуто галузь характеристики та структуру компанії IT-компанії «Диверсіті IT», було проведено аналіз наявної проблеми компанії та існуючі системи вирішення проблеми, переглянуто проблеми сучасних систем. Додатково було визначено завдання на магістерську роботу.

В другому розділі було розглянуто загальні характеристики впровадженої комп'ютерної мережі IT-компанії «Диверсіті IT», яка обладнана ботом зворотного зв'язку. Проведено аналіз мов програмування для створення бота зворотного зв'язку, також аналіз баз даних для вирішення проблеми та розробка загальної моделі СМО.

В третьому розділі приведені основні цілі впровадження системи IT-компанії «Диверсіті IT» з її описом, визначено основні вимоги до системи, визначено структурну схему комплексу технічних засобів мережі, а також обгрунтовано застосування апаратних засобів.

В четвертому розділі було виконано опис алгоритму, згідно якого розробляється програмне забезпечення та фрагменти коду програмного забезпечення з поясненням них, до системи компанії «Диверсіті IT».

В п'ятому розділі було проведено експерименти над ботом зворотного зв'язку, перевірено функції на практиці.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СЗЗ − система зворотного зв’язку.

ПЗ − програмне забезпечення.

ПК − персональний комп’ютер.

ДБЖ − джерело безперебійного живлення.

ККД − коефіцієнт корисної дії.

ТЗ − технічне завдання.

СМО − система масового обслуговування.

ВСТУП

Сучасний стан розробки Telegram-ботів характеризується стрімким зростанням інтересу як з боку бізнесу. Це пов’язано з низкою факторів: мільйони активних користувачів по всьому світу роблять Telegram одним із найпопулярніших канaлів комунікації, наявність добре документованого API та великої кількості бібліотек для різних мов програмування, боти тепер використовуються для вирішення різноманітних завдань від автоматизації рутинних операцій до надання складних послуг.

В останні роки на світовому рівні спостерігаються активні зрушення в автоматизації процесів зворотного зв’язку завдяки чат-ботам. Ключові тенденції включають:

− використання штучного інтелекту та машинного навчання для покращення обробки запитів і надання відповідей;

− масштабовані системи зворотного зв’язку з використанням хмарних технологій;

− інтеграція з системами для збереження та обробки даних про клієнтів.

Актуальність розробки комп'ютерної мережі IT компанії "Диверсіті IT" з Telegram-ботом зворотного зв’язку полягає в необхідності забезпечення ефективного спілкування між користувачами та компанією "Диверсіті IT" в межах однієї мережі для збільшення замовлень. Бот допомагає прискорити процес збору заявок та замовлень.

Мета роботи полягає в розробці комп'ютерної мережі IT компанії "Диверсіті IT" та впровадженні Telegram-бота, який забезпечить користувачів швидким та зручним доступом до базової інформації про компанію та дозволить швидко зв’язатися в межах однієї мережі з адміністрації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

− побудова системи масового обслуговування;

− аналізувати існуючі рішення у сфері автоматизації зворотного зв’язку;

− розробити технічні вимоги до комп'ютерної системи;

− розробити алгоритм роботи телеграм боту;

− програмно реалізувати Telegram-бота, включаючи інтеграцію з базою даних;

− перевірити роботу комп'ютерної системи.

Об’єкт дослідження − комп’ютерна система з розробкою боту зворотного зв’язку у Telegram.

Предмет дослідження − методи автоматизації обробки запитів від користувачів за допомогою чат-бота, бази даних та організації комп’ютерної мережі для забезпечення стабільної роботи системи.

Методи дослідження − використання системного аналізу, вивчення інформації і тестування ботів Telegram.

Ідея роботи − створення Telegram-бота, який забезпечить зручний та ефективний канал комунікації між компанією та її клієнтами.

Встановлено, що використання Telegram-бота для збору та обробки зворотного зв’язку від користувачів значно скорочує середній час відповіді на запити в порівнянні з традиційними методами обслуговування клієнтів, при умові використання в межах однієї соц-мережі.

Обґрунтовано застосування MongoDB та Aiogram для створення бота зворотного зв’язку.

1 СТАН ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Стан питання

Комп’ютерні системи відіграють ключову роль у сучасному світі, забезпечуючи автоматизацію процесів, обробку великих обсягів інформації та швидке прийняття рішень. Вони є основою для функціювання різних галузей, таких як медицина, освіта, промисловість, транспорт і комунікації. Завдяки комп’ютерним системам компанії можуть оптимізувати свої робочі процеси, знижувати витрати та підвищувати продуктивність.

Розробка телеграм-ботів є сучасним і динамічним напрямом у галузі інформаційних технологій, що поєднує аспекти програмної інженерії, штучного інтелекту та інтерфейсів користувача. Завдяки простоті інтеграції, багатофункціональності та широкій поширеності месенджера Telegram, боти стають важливим інструментом автоматизації завдань, комунікації з користувачами та побудови бізнес-процесів.

З початку своєї появи в 2015 році API Telegram відкрив значні можливості для розробників, дозволяючи створювати програми, які інтегруються з платформою для виконання різноманітних завдань. Основні фактори, що сприяли активному розвитку ботів, включають:

− Telegram надає добре документований API, який дозволяє розробникам легко створювати та інтегрувати боти в різні сервіси;

− завдяки існуванню бібліотек для популярних мов, таких як Python, JavaScript, Java, PHP, розробка ботів стає доступною навіть для початківців;

− боти виконують рутинні завдання, знижуючи витрати часу та людських ресурсів у бізнесі, освіті та побуті;

− Telegram є однією з найпоширеніших платформ для спілкування, що забезпечує велику аудиторію користувачів для взаємодії через боти.

У науковій та прикладній площині дослідження в області телеграм-ботів охоплюють кілька ключових аспектів:

− використання NLP у телеграм-ботах дозволяє створювати інтерактивні системи, здатні розуміти запити користувачів природною мовою;

− поєднання телеграм-ботів із технологіями машинного навчання та нейронними мережами сприяє створенню персоналізованих сервісів;

− зважаючи на широкий доступ до даних через месенджери, особлива увага приділяється питанням безпеки передачі інформації та захисту даних користувачів;

− телеграм боти все частіше використовуються як інструменти для управління базами даних, проведення опитувань, управління IoT-пристроями тощо.

Таким чином, телеграм-боти є не лише інструментом для автоматизації, але й платформою для досліджень та інновацій, що сприяє розвитку технологій взаємодії людини з машиною.

1.2 Характеристика і структура об’єкта впровадження

1.2.1 Характеристика об’єкта IT-компанії «Диверсіті IT»

Компанія, для якої виконується дослідження та розробка бота зворотного зв’язку це IT-компанія «Диверсіті IT». IT-компанія, яка спеціалізується на розробці програмного забезпечення, наданні IT-послуг для бізнесу та має команду висококваліфікованих професіоналів з великим досвідом роботи у галузі програмної розробки. Компанія завжди відкрита до нових ідей та готова реалізувати будь-яку задумку своїх клієнтів.

Компанія «Диверсіті IT» володіє сучасними технологіями та інструментами для розробки програмного забезпечення та використовує найновіші технології та методики, що дозволяють надавати якісні та надійні рішення для своїх клієнтів.

«Диверсіті IT» має широкий діапазон клієнтів, що включає малі підприємства, середні та великі корпорації різних галузей та індустрій. Також завжди відкрита до співпраці з будь-якими клієнтами.

1.2.2 Організаційна структура компанії

Організаційна структура ІТ-компанії «Диверсіті ІТ» складається з:

– генеральний директор;

– менеджери проектів;

– розробники;

– дизайнери;

– тестувальники;

– консультанти з ІТ;

– аналітики.

Генеральний директор – відповідає за керування компанією в цілому, ведення стратегічного планування та взаємодію з клієнтами.

Менеджери проектів – забезпечують успішне виконання проектів, ведення команди розробників, контроль за термінами та якістю роботи.

Розробники – відповідають за програмну розробку продуктів, дотримання технічних вимог та якість коду.

Дизайнери – забезпечують високий рівень дизайну продуктів, створення користувацьких інтерфейсів та рекламної продукції.

Тестувальники – проводять тестування програмного забезпечення та виявляють помилки перед випуском продукту.

Консультанти з ІТ – надають консультації з питань технічної підтримки клієнтів та вирішення проблем з ІТ.

Аналітики – проводять аналіз ринку, клієнтів та конкурентів, допомагають в формулюванні стратегії та плануванні розвитку компанії.

Крім внутрішньої команди, компанія «Диверсіті ІТ» також може привласнювати зовнішніх консультантів та партнерів, які допомагають забезпечити виконання проектів та забезпечення потреб клієнтів у певних областях. Наприклад, компанія може співпрацювати з партнерами, які спеціалізуються на інтеграції різноманітного програмного забезпечення,

розширенні можливостей технічної інфраструктури, або наданні інших інноваційних послуг.

Загалом, успішність компанії залежить від того, наскільки добре вона використовує свої ресурси та здібності, технічні та інноваційні рішення, які вона використовує, а також від ефективної взаємодії з клієнтами та партнерами.

Схему організаційної структури IT-компанії наведено на рисунку 1.1.

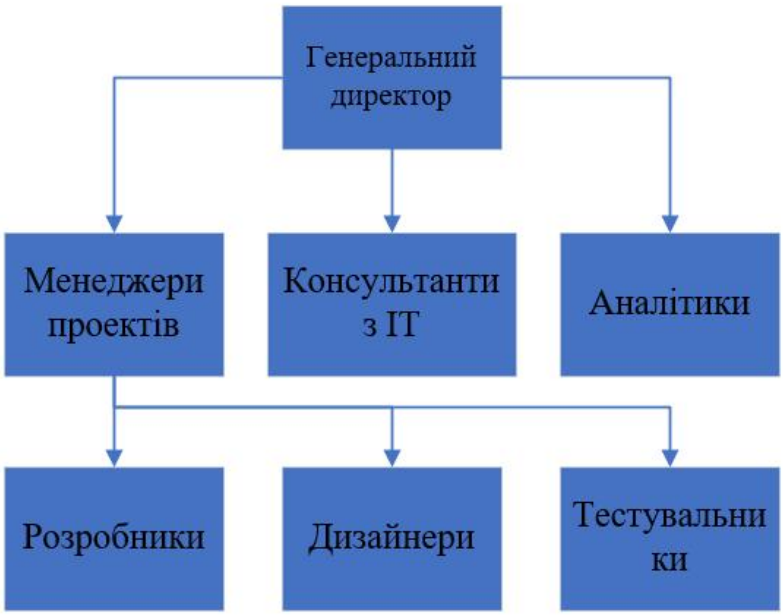

Рисунок 1.1 – Схема організаційної структури IT-компанії

1.3 Аналіз наявної проблеми компанії «Диверсіті IT» та існуючих систем

В Україні багато компаній надають послуги через Telegram, але не всі з них використовують Telegram-боти для зворотного зв’язку з клієнтами. Відсутність інтерактивного каналу для швидкого та ефективного зв’язку з компанією може призвести до втрати можливостей для оперативного вирішення запитів клієнтів. Це є проблемою, оскільки стандартні канали комунікації, такі як електронна пошта або телефон, часто вимагають значного часу для обробки запитів і можуть бути менш зручними для користувачів.

Впровадження Telegram-бота зворотного зв’язку для компанії «Диверсіті ІТ» дозволяє вирішити цю проблему, забезпечуючи швидку, автоматизовану комунікацію з клієнтами. Бот може оперативно приймати запити, надавати відповіді на часті питання, допомагати оформити замовлення або запитати додаткову інформацію, що значно спрощує процес для клієнтів і підвищує ефективність взаємодії.

Це рішення є актуальним, оскільки попит на автоматизовані системи зворотного зв’язку зростає в умовах сучасної цифровізації бізнесу. Telegram-боти вже стали стандартом для багатьох компаній, і наявність такого інструменту для зворотного зв’язку може стати важливим конкурентним перевагою для компанії.

Системи зворотного зв’язку в телеграм-ботах є важливим елементом взаємодії з користувачами, спрямованим на забезпечення ефективної комунікації, автоматизації процесів і вирішення специфічних запитів, в межах однієї соціальної мережі, для збільшення замовлень.

Існує декілька поширених форматів реалізації зворотного зв’язку в телеграм-ботах:

− створення інтерактивних форм, які запитують у користувачів конкретну інформацію (ім’я, контактні дані, опис проблеми тощо);

− перенаправлення повідомлень користувача до оператора для живої комунікації;

− автоматичне збереження та обробка запитів у базах даних для подальшого аналізу та опрацювання.

Сучасні телеграм-боти використовують алгоритми обробки природної мови (NLP) для розуміння запитів користувачів і формування відповідей. Найчастіше застосовуються:

− моделі на основі нейронних мереж (наприклад, GPT, BERT) для аналізу складних запитів;

− підключення до вебхуків для автоматичного оброблення даних, що надходять від користувачів.

Функціональні можливості:

− відповіді на часті запитання за допомогою заздалегідь заданих шаблонів;

− інформування користувача про статус виконання запиту (наприклад, підтвердження отримання заявки, статус вирішення проблеми).

Системи зворотного зв’язку, реалізовані через Telegram-боти, відзначаються широким спектром функціональних можливостей та переваг у порівнянні з іншими платформами.

Telegram активно використовується малими, середніми та великими підприємствами завдяки його відкритому API, високій швидкості обміну даними і широким можливостям для інтеграцій. Наприклад, компанії як «Monobank» (рисунок 1.2) використовують Telegram-ботів (https://t.me/monobankbot) для оперативного обслуговування клієнтів, що підвищує швидкість відповіді та знижує навантаження на кол-центри.

Рисунок 1.2 – Telegram-бот компанії MonoBank

У ботах інших компаній, наприклад, «Нова Пошта» (https://telegram.me/PrivatBank_help_bot), реалізовано функціонал інтерактивних форм (рисунок 1.3), які дозволяють клієнтам вводити дані про посилки, тощо. Це пришвидшує процес обробки.

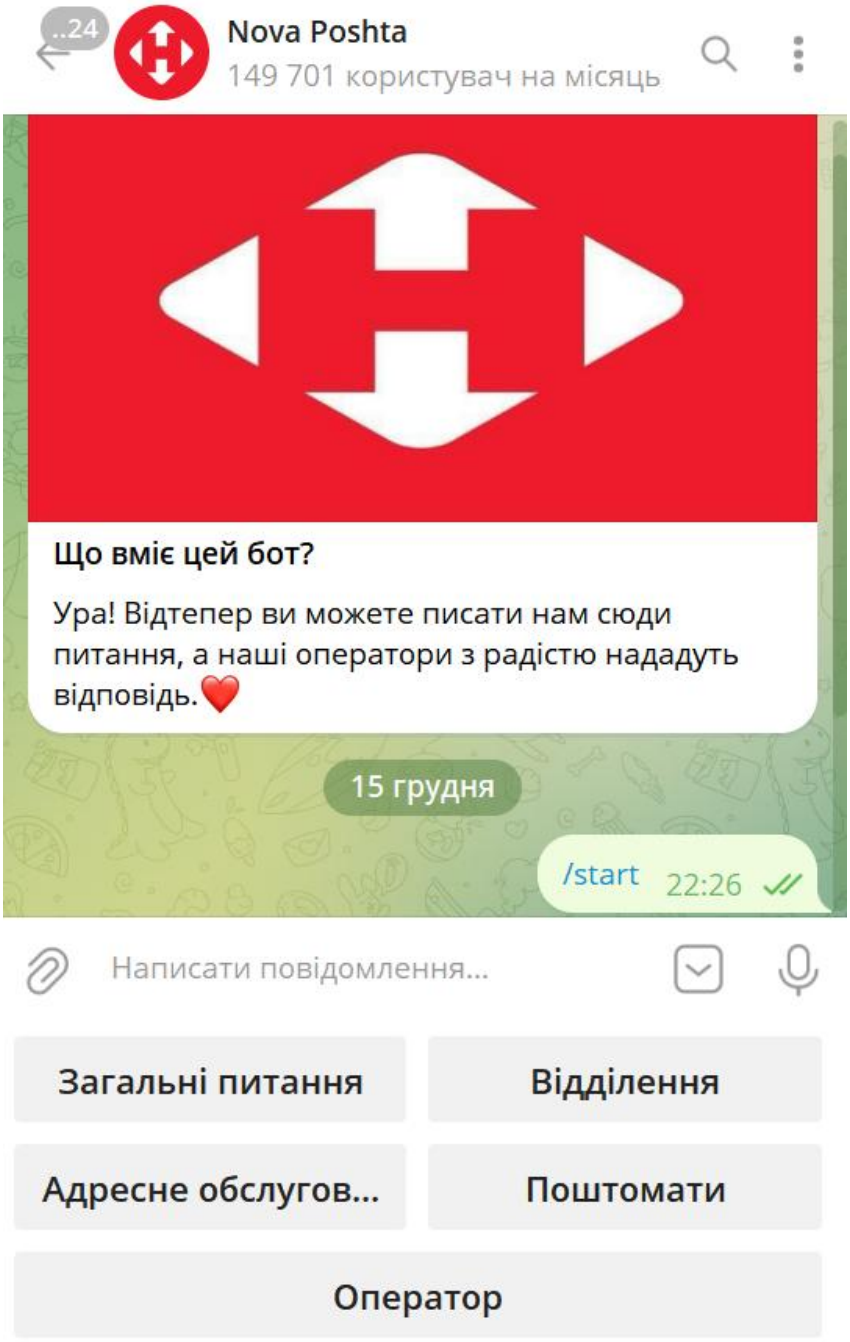

Рисунок 1.3 – Telegram-бот компанії Нова Пошта

Системи, такі як «ПриватБанк» (https://telegram.me/PrivatBank_help_bot на рисунку 1.4) або «Rozetka» (https://t.me/Rozetka_helpBot на рисунку 1.5), інтегрують Telegram-боти з базами даних та CRM-системами. Завдяки цьому можна не лише отримувати запити, але й автоматично обробляти їх, наприклад, надаючи клієнтам статус їх замовлення чи стан платежів.

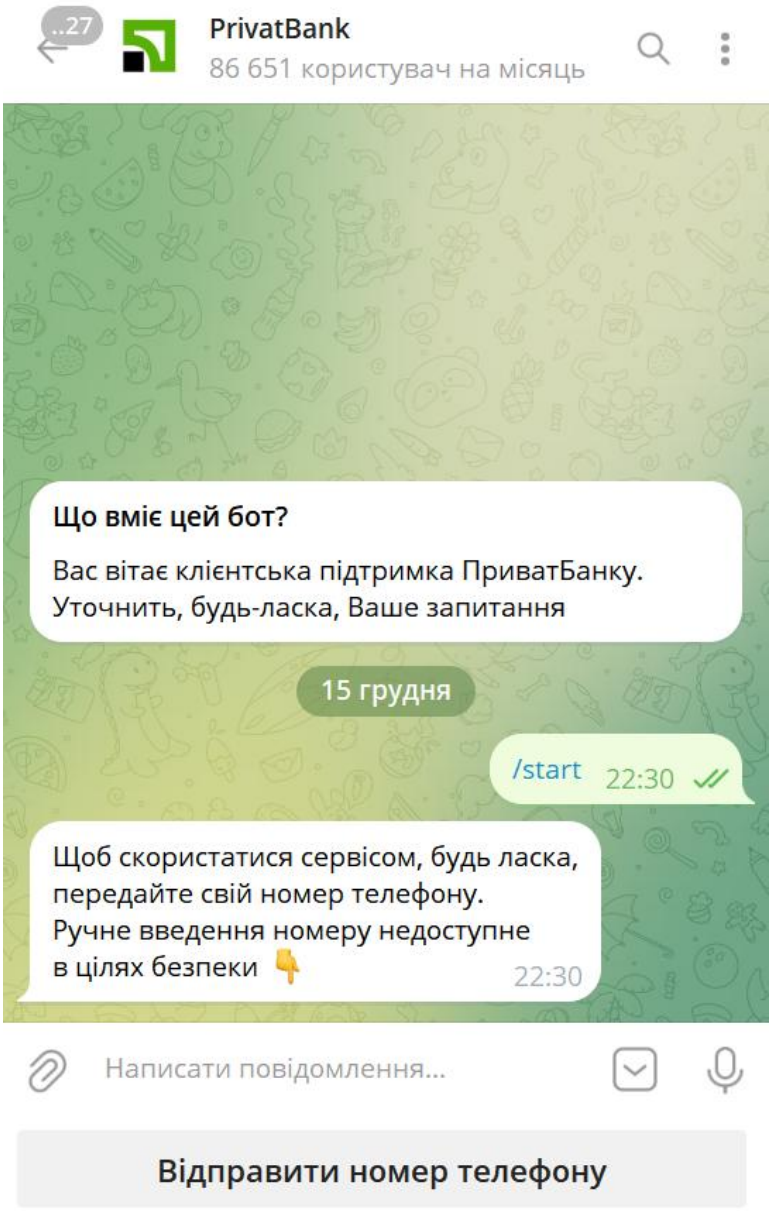

Рисунок 1.4 – Telegram-бот компанії «ПриватБанк»

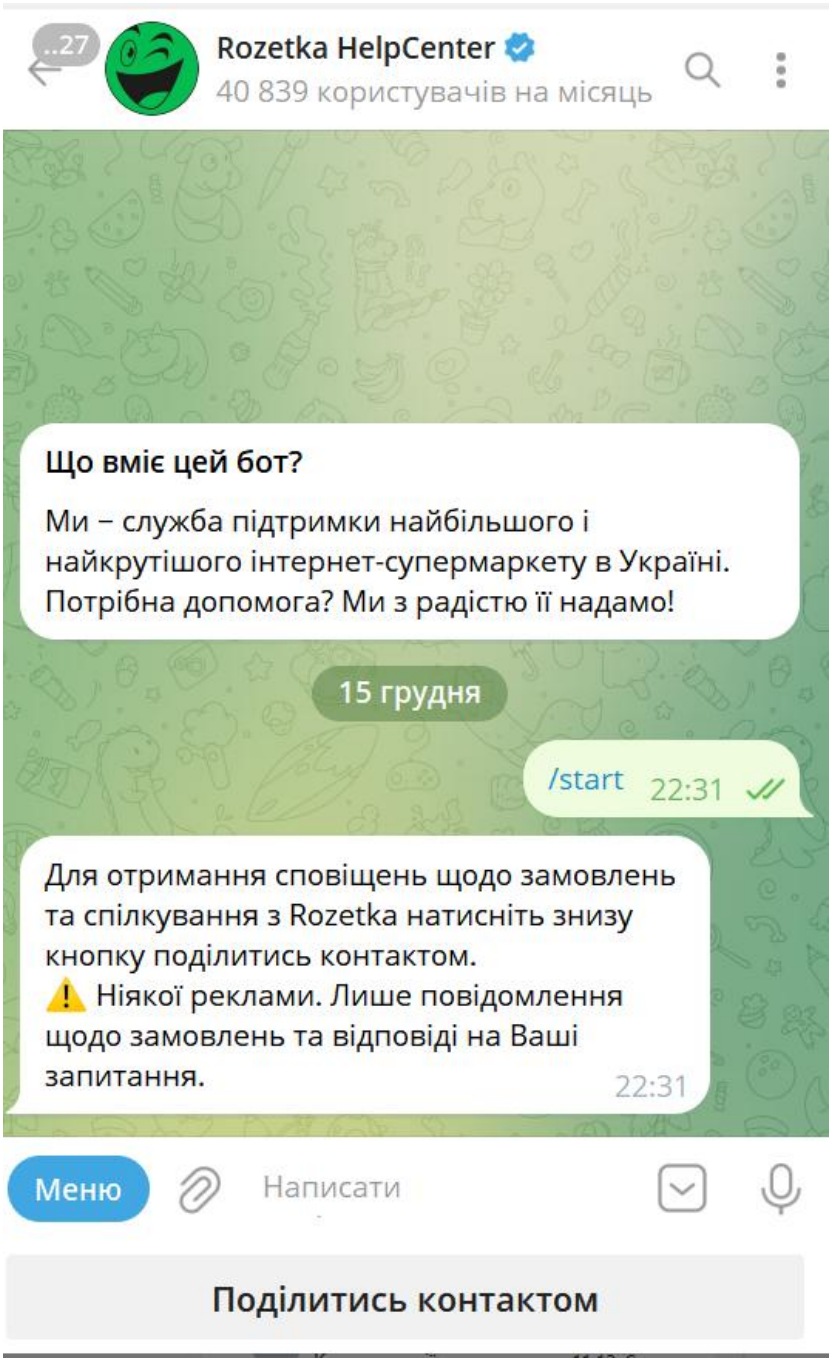

Рисунок 1.5 – Telegram-бот компанії «Rozetka»

Інтертоп – українська промислова компанія, що також використовує Telegram-бота (https://t.me/intertop_ukraine_bot на рисунку 1.6) для зворотного зв’язку з клієнтами. У їхньому випадку бот використовується для автоматизації комунікацій щодо замовлень і запитів.

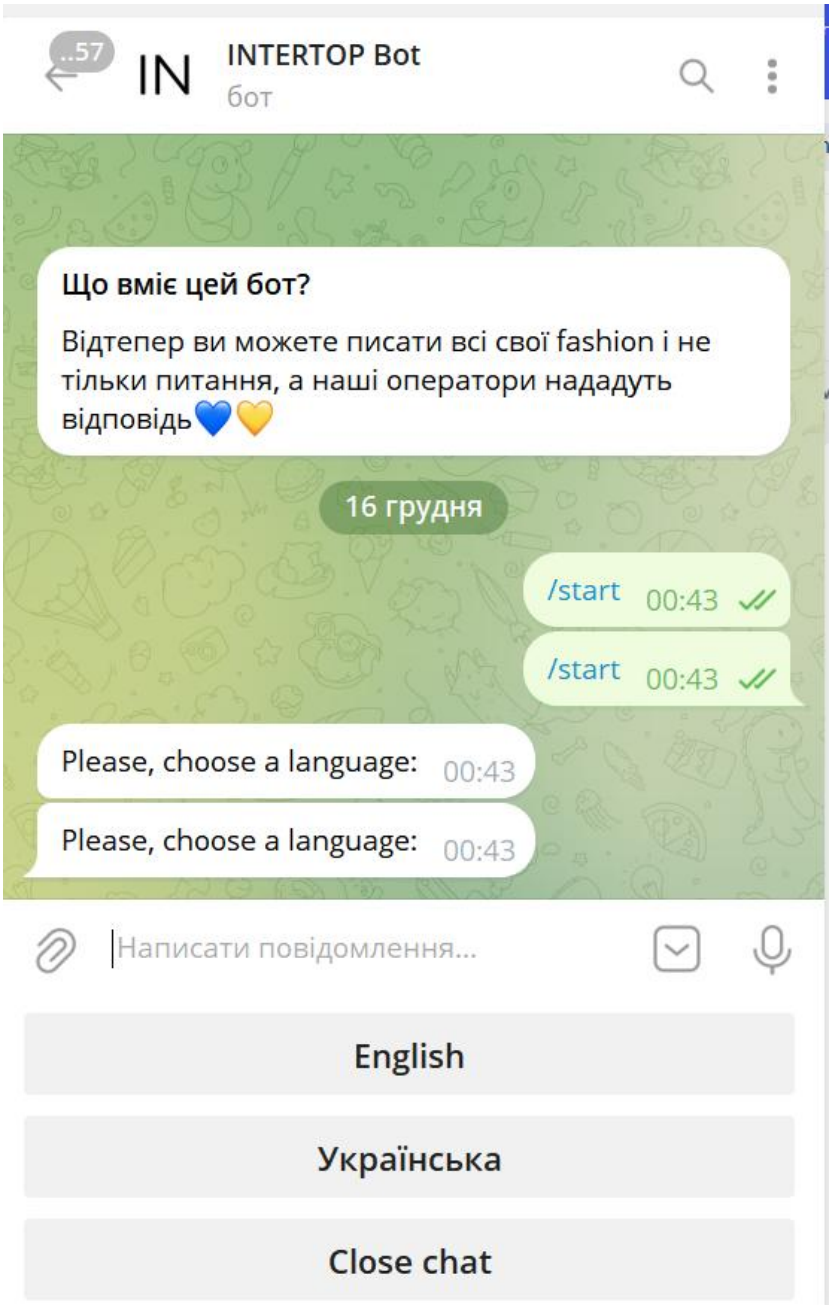

Рисунок 1.6 – Telegram-бот компанії Інтертоп

«Київстар» – один із найбільших операторів мобільного зв’язку в Україні, який також використовує Telegram-бота (https://t.me/Kyivstar_support_bot на рисунку 1.7) для надання підтримки своїм клієнтам. Бот дозволяє абонентам отримати інформацію про їхні тарифи, баланс, можливість поповнення рахунку, а також відправляти запити щодо послуг або технічних проблем. Крім того, через

бота можна отримати допомогу з налаштуванням мобільного інтернету або дзвінків.

Основні можливості бота:

– перевірка балансу та підключених послуг;

– поповнення рахунку;

– отримання інформації про актуальні тарифи;

– підтримка користувачів щодо налаштування телефону та інтернету.

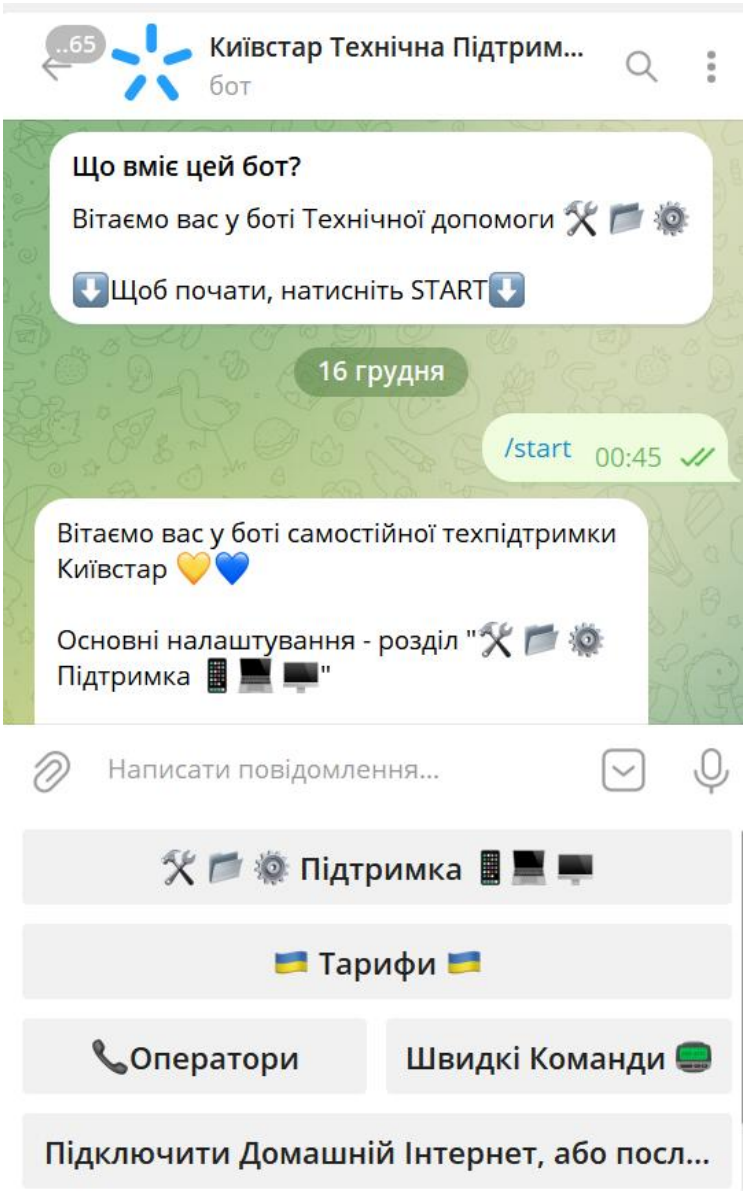

Рисунок 1.7 – Telegram-бот компанії «Київстар»

Запропонований Telegram-бот для «Диверсіті ІТ» розширює традиційні можливості систем зворотного зв’язку, додаючи такі унікальні особливості:

− різноманітність форматів зворотного зв’язку − текстові, голосові повідомлення, фото і відео, що дозволяє враховувати специфіку кожного запиту;

− функціонал для модерації користувачів, розсилки повідомлень та зняття затримки дозволяє ефективно керувати процесом взаємодії з клієнтами;

− використання MongoDB для зберігання даних забезпечує збереження великого обсягу інформації та її швидку обробку.

1.4 Проблеми сучасних систем

Основні проблеми сучасних телеграм-ботів:

− багато ботів створюються для виконання вузьких завдань і мають обмежену здатність обробляти складні сценарії;

− відсутність інтеграції з іншими системами (наприклад, ERP чи CRM) ускладнює автоматизацію складних бізнес-процесів;

− низький рівень штучного інтелекту;

− алгоритми обробки природної мови (NLP) іноді не здатні розпізнавати складні або контекстуальні запити;

− велике навантаження може призводити до збоїв у роботі бота;

− телеграм-боти залежать від стабільності серверів Telegram та API, що може створювати труднощі під час пікових навантажень;

− Telegram API має обмеження, такі як максимальна кількість запитів на секунду або ліміти на обсяг переданої інформації;

− у багатьох випадках боти не враховують індивідуальні особливості користувачів, що призводить до менш ефективної комунікації;

− відсутність адаптивності до специфічних вимог компанії;

− ризики витоку даних через недостатньо захищені канали зв’язку;

− використання ботів для фішингу, поширення шкідливого програмного забезпечення або атак на сервери компанії.

1.5 Постановка завдання роботи

Завданням для магістерської роботи є розробка Telegram-бота зворотного зв’язку.

Для вирішення цього завдання необхідно виконати наступні пункти:

− ознайомитися зі структурою компанії;

− вибір мови програмування та технологій для реалізації бота;

− розробка технічних вимог до телеграм боту;

− розробка технічних вимог до бази даних;

− розробка технічних вимог до серверу;

− розробка логіки обробки повідомлень користувачів;

− вибір комплексу технічних засобів системи;

− інтеграція бота з базою даних;

− інтерфейс для адміністраторів;

− реалізація системи сповіщень;

− забезпечення безпеки даних;

− тестування та налаштування системи.

Ці етапи допоможуть створити ефективного та безпечного Telegram-бота в комп’ютерній мережі компанії «Диверсіті ІТ», що буде корисним інструментом для компанії та відповідатиме всім технічним вимогам для збирання зворотного зв’язку.

1.6 Висновки до розділу

Розділ «Стан питання та постановка завдання» описує стан питання розробки Telegram-бота зворотного зв’язку, стислу інформацію компанії для якої розробка проводиться.

Далі в розділі було розглянуто існуючі СЗЗ, а також їх проблеми. Було виконано постановку завдання для магістерської роботи.

2 ТЕОРЕТИЧНИЙ РОЗДІЛ

2.1 Розробка моделі СМО для Telegram-бота зворотного зв’язку IT-компанії «Диверсіті ІТ»

Telegram-бот − це програмний агент, який працює в екосистемі Telegram і взаємодіє з користувачами через API Telegram. Система складається з кількох ключових компонентів, а саме:

− серверний компонент − обробляє запити користувачів, управляє логікою відповіді та інтегрується з базами даних;

− Telegram API − забезпечує зв’язок між сервером і клієнтським додатком Telegram;

− база даних − використовується для зберігання інформації про користувачів, запити та внутрішні параметри системи.

Для опису системи Telegram-бота зворотного зв’язку через модель системи масового обслуговування (СМО), необхідно визначити ключові компоненти системи та їх взаємозв’язок, щоб продемонструвати, як обробляються запити від користувачів.

Джерело запитів (користувачі Telegram):

− користувачі можуть надсилати текстові повідомлення, медіафайли (зображення, відео, документи), а також голосові повідомлення. Це вимагає різноманітних методів обробки та фільтрації інформації на стороні бота.

Система обробки запитів (сервер із ботом):

− Telegram-бот працює на основі асинхронної обробки (Aiogram);

− бот виконує два основні завдання: автоматичну обробку запитів (відповіді на стандартні питання) та переадресацію запитів до адміністратора.

Відповідно до моделі СМО, сервер можна представити як обслуговуючий пристрій із середнім часом обробки запиту μ (запити/хвилину).

Черга:

− якщо бот або сервер перевантажений, запити надходять до черги. Довжина черги визначається апаратними обмеженнями (наприклад, обсяг оперативної пам’яті та швидкодія сервера).

База даних (MongoDB):

− запити обробляються та повертаються до Telegram-бота.

Адміністратор (оператор):

− якщо запит неможливо автоматично опрацювати, він перенаправляється адміністратору, що є кінцевим обслуговуючим пристроєм із власним часом обробки.

Загальна схема СМО моделі зображено на рисунку 2.1

Рисунок 2.1 – Загальна схема СМО моделі

2.2. Огляд мов програмування для створення Telegram-ботів

2.2.1 Мова Python

Python − це одна з найпопулярніших мов програмування для розробки Telegram-ботів. Завдяки її простоті, зручності та багатій екосистемі, Python підходить як для новачків, так і для досвідчених розробників.

Aiogram − асинхронна бібліотека, яка забезпечує високу продуктивність, особливо для ботів із великою кількістю одночасних користувачів.

Python підтримує інтеграцію з базами даних (MongoDB, PostgreSQL, SQLite), хмарними сервісами, зовнішніми API та обробку великих обсягів даних.

За допомогою бібліотек, як-от asyncio та aiogram, Python дозволяє ефективно працювати з багатопотоковою обробкою запитів, що знижує затримки та підвищує продуктивність.

Python має величезну спільноту, що забезпечує доступ до документації, прикладів коду та активної підтримки.

Telegram-боти, написані на Python, можна розгортати на різних платформах: Windows, Linux, macOS.

Гнучкість Python дозволяє додавати новий функціонал до бота без необхідності значного переписування коду.

2.2.2 Мова JavaScript (Node.js)

Node.js − це середовище виконання JavaScript, яке дозволяє створювати високопродуктивні серверні додатки. Завдяки своїм асинхронним можливостям Node.js став одним із найкращих виборів для розробки Telegram-ботів. У цьому підрозділі розглядається використання JavaScript (Node.js) для створення Telegram-бота.

Переваги:

− використання неблокуючого вводу/виводу дозволяє обробляти численні запити одночасно, що є важливим для обслуговування багатьох користувачів;

− Node.js працює в одному потоці, що зменшує витрати на управління потоками, але дозволяє обробляти багато паралельних запитів через події;

− Node.js має величезну кількість модулів і бібліотек, які спрощують розробку (наприклад, Telegraf.js, node-fetch, Express);

− асинхронна модель дозволяє швидко обробляти багато одночасних запитів, що є критичним для популярних ботів;

− легка інтеграція з іншими сервісами: Node.js має велику кількість модулів для роботи з API, а також для виконання HTTP-запитів (axios, node-fetch).

Недоліки Node.js для Telegram-ботів:

− через однопоточність Node.js не підходить для інтенсивних обчислювальних завдань, але це можна вирішити через Worker Threads або винос обчислень в інший сервіс;

− у порівнянні з мовами на основі компіляції (наприклад, TypeScript або Java), JavaScript менш надійний через динамічну типізацію, що може призвести до помилок у великому коді.

2.2.3 Мова PHP

PHP − популярна серверна мова програмування, яка широко використовується для веб-розробки. Завдяки простоті у використанні та підтримці веб-серверів, PHP також підходить для створення Telegram-ботів, особливо у випадках інтеграції з існуючими веб-додатками або проектами, які вже використовують PHP.

Переваги:

− PHP працює практично на всіх веб-серверах (Apache, Nginx), що дозволяє легко налаштувати Telegram-бота без необхідності встановлення додаткового програмного забезпечення. Також має вбудовані функції для роботи з MySQL, PostgreSQL та іншими базами даних. Це спрощує реалізацію Telegram-ботів із функціоналом збереження даних, наприклад, запитів від користувачів;

− PHP має розширену підтримку для виконання HTTP-запитів (через cURL, file_get_contents) і обробки JSON, що є основними механізмами для взаємодії з Telegram API;

− PHP добре підходить для компаній, які вже мають веб-додатки на PHP, оскільки Telegram-бот може інтегруватися в їх інфраструктуру.

Telegram API підтримується низкою бібліотек на PHP, таких як:

− MadelineProto;

− Telegram Bot API PHP SDK;

− php-telegram-bot.

Недоліки використання PHP:

− PHP не підтримує багатопоточність, тому для роботи з високим навантаженням краще використовувати асинхронні мови, як-от Python або Node.js;

− для реалізації складних задач потрібна додаткова конфігурація або використання сторонніх бібліотек;

− PHP скрипти часто запускаються на веб-серверах, що може ускладнити їх розгортання порівняно з іншими мовами.

2.2.4 Мова Java

Java є універсальною мовою програмування, що забезпечує високу продуктивність, стабільність та широкі можливості для інтеграції з різними системами. Завдяки цим перевагам Java часто використовується для розробки складних та масштабованих рішень, включаючи Telegram-ботів.

Переваги:

− Java підходить для створення ботів, які працюють із великими обсягами даних та мають високі вимоги до швидкості обробки запитів;

− програми, написані на Java, можуть працювати на будь-якій платформі з встановленою Java Virtual Machine (JVM);

− Java дозволяє легко розширювати функціонал бота та адаптувати його до зростання навантаження;

− вбудована підтримка багатопоточності допомагає ефективно обробляти паралельні запити користувачів;

− Java забезпечує високий рівень безпеки, що є важливим для корпоративних рішень.

Недоліки:

− більший обсяг коду для виконання базових завдань.

2.2.5 Мова C#

C# − це сучасна об’єктозорієнтована мова програмування, яка активно використовується для створення корпоративних рішень, веб-додатків і Telegram-ботів. Завдяки платформі .NET і бібліотекам, C# забезпечує високу продуктивність і зручність розробки ботів.

Переваги:

− завдяки компіляції коду в байт-код C# забезпечує високу швидкість виконання та стабільність роботи;

− боти, створені на C#, підходять для обробки великих обсягів запитів і забезпечення безперебійної роботи;

− підтримка мультиплатформеності через .NET Core (Windows, Linux, MacOS);

− розширені можливості інтеграції з іншими сервісами Microsoft (Azure, SQL Server тощо);

− C# має вбудовані інструменти для управління пам’яттю (Garbage Collector), що знижує ризик помилок, пов’язаних із витоками пам’яті.

Найпопулярнішою бібліотекою для розробки Telegram-ботів на C# є Telegram.Bot. Вона забезпечує повний доступ до Telegram API, включаючи підтримку оновлень (update), команд і інтеграції з вебхуками.

Недоліки:

− потребує більше підготовки для запуску першого бота через залежності та налаштування бібліотек;

− для невеликих ботів використання C# може бути надмірним через його продуктивність і багатофункціональність. Простішим і швидшим варіантом буде Python;

− у порівнянні з Python або JavaScript, у спільноті C# менше готових прикладів та посібників, орієнтованих саме на розробку Telegram-ботів;

− запуск бота на C# через .NET може вимагати більше системних ресурсів, ніж легкі рішення на Python або Node.js;

− у хмарному середовищі сервери з підтримкою .NET часто коштують дорожче через підвищені вимоги до середовища виконання;

− якщо бот розгортається на Linux, може виникнути потреба в налаштуванні .NET Core, що потребує більше часу порівняно з іншими мовами, які встановлюються та налаштовуються простіше.

2.2.6 Мова Go

Go (Golang) − це мова програмування, розроблена компанією Google, відома своєю простотою, високою продуктивністю та підтримкою багатозадачності. Go є чудовим вибором для створення Telegram-ботів, завдяки своїм перевагам в обробці запитів з високою швидкістю та низьким споживанням ресурсів.

Переваги:

− є компільованою мовою, що дає їй перевагу в продуктивності порівняно з інтерпретованими мовами, такими як Python або JavaScript. Це особливо важливо для бота, який має обробляти великий потік запитів, адже Go може обробляти кілька запитів одночасно, не втрачаючи продуктивності;

− Go підтримує горутини − легковагі потоки, що дозволяє обробляти тисячі з’єднань одночасно, що робить її відмінним вибором для створення бота з високою навантаженістю;

− Go має дуже чистий і простий синтаксис, що дозволяє розробникам швидко освоїти мову і зосередитися на логіці бота, а не на складностях програмування;

− програми на Go компілюються в окремі виконувані файли, що робить їх легкими для розгортання на сервері без необхідності встановлення додаткових бібліотек чи залежностей;

− Go має вбудовану підтримку для масштабування, що дозволяє зручно створювати боти, які можуть обробляти великі навантаження, такі як великі об’єми запитів або численні користувачі.

Недоліки:

− на відміну від Python або JavaScript, Go має менше бібліотек і готових рішень для створення бота, що може призвести до необхідності розробляти власні компоненти для деяких функцій;

− Go має меншу спільноту розробників у порівнянні з Python або Node.js, що може ускладнити пошук допомоги або відповідей на питання;

− для деяких високо навантажених систем Go може вимагати значних ресурсів для підтримки паралельних процесів, хоча це рідко є проблемою для більшості ботів.

2.2.7 Висновки щодо вибору мови програмування для Telegram-бота

Python є найкращим вибором для розробки Telegram-ботів завдяки своїй простоті, широкому спектру бібліотек, підтримці асинхронних операцій та наявності величезної спільноти. Завдяки своїм інструментам і численним готовим рішенням Python дозволяє швидко створювати ефективні боти, що можуть обробляти великий обсяг запитів і взаємодіяти з користувачами в реальному часі.

2.3 Бази даних

2.3.1 Види баз даних

2.3.1.1 Реляційні бази даних (RDBMS)

Реляційні бази даних (RDBMS, Relational Database Management Systems) є одним з найпоширеніших типів баз даних, які використовуються для зберігання структурованих даних, що мають взаємозв’язки між собою. У випадку з Telegram-ботами реляційні бази даних можуть бути використані для зберігання інформації про користувачів, історії запитів, повідомлень та інших даних, які потребують організованого зберігання з можливістю пошуку та фільтрації.

Особливості реляційних баз даних:

− дані в реляційних базах зберігаються у вигляді таблиць, де кожна таблиця має рядки (записи) та стовпці (поля). Це дозволяє чітко структурувати інформацію;

− реляційні бази даних дозволяють встановлювати зв’язки між різними таблицями через зовнішні ключі (foreign keys), що забезпечує цілісність та узгодженість даних;

− для роботи з реляційними базами даних використовується мова запитів SQL (Structured Query Language), яка дозволяє ефективно здійснювати операції над даними − від вибору і вставки до оновлення та видалення;

− завдяки властивостям транзакцій (ACID), реляційні бази забезпечують збереження цілісності даних, навіть при відмовах системи чи інших непередбачених ситуаціях.

Переваги використання реляційних баз даних для Telegram-ботів:

− боти часто працюють із структурованими даними, такими як особисті дані користувачів, історія їхніх запитів, налаштування, або коментарі. Реляційні бази забезпечують зручну організацію та обробку таких даних;

− завдяки мові SQL, можна легко фільтрувати, сортувати, оновлювати й видаляти дані, а також здійснювати складні запити з кількома таблицями через JOIN;

− Telegram-бот може зберігати різні типи даних, такі як: історія чатів, персональні налаштування користувачів, продукти замовлення. Використовуючи зв’язки між таблицями, можна зберігати ці дані без дублювання;

− реляційні бази даних здатні обробляти великі обсяги даних, зберігаючи їх у структурованому вигляді, що дозволяє розширювати функціонал бота та забезпечувати високий рівень безпеки даних (шляхом застосування механізмів автентифікації, шифрування тощо).

2.3.1.2 Нереляційні бази даних (NoSQL)

Нереляційні бази даних (NoSQL) є інструментом для зберігання і обробки великих обсягів даних, особливо коли необхідно обробляти неструктуровану або напівструктуровану інформацію. Вони підходять для систем, де дані можуть змінюватися часто або не мають чіткої схеми, що є типовим для чат-ботів, зокрема Telegram-ботів. У випадку з Telegram-ботами, NoSQL бази даних можуть використовуватися для зберігання інформації про користувачів, запити, повідомлення, налаштування бота та інші метадані.

Дані бази даних не вимагають попередньо визначеної схеми для зберігання даних. Дані можуть бути збережені у різних форматах, таких як JSON, BSON, XML, що дозволяє зберігати неструктуровані або напівструктуровані дані. У Telegram-ботах це може бути корисно для зберігання повідомлень з різними типами вмісту (текст, фото, відео), де кожне повідомлення може мати свою структуру.

Нереляційні бази даних часто підтримують горизонтальну масштабованість. Це означає, що вони здатні обробляти більші обсяги даних, додавши нові сервери або вузли в кластер, що дозволяє зменшити навантаження на один сервер і збільшити продуктивність. Боти, що мають великий потік повідомлень, наприклад, у великих Telegram-каналах, можуть скористатися цією перевагою для масштабування.

Багато NoSQL баз даних оптимізовані для швидкого доступу і запису даних, що дозволяє обробляти запити на високу швидкість. Це особливо корисно для додатків, де критичним є час відгуку і необхідність обробки даних в реальному часі. Система зворотного зв’язку в Telegram-боті, що потребує миттєвого реагування на запити користувачів, може ефективно використовувати такі бази даних для швидкої обробки повідомлень.

2.3.2 База даних MySQL

MySQL − це одна з найпопулярніших реляційних баз даних з відкритим кодом, яка широко використовується для розробки веб-додатків, в тому числі для створення чат-ботів, сайтів, інтернет-магазинів і багатьох інших систем. Вона працює за принципом клієнт-сервер, де сервер зберігає дані, а клієнт взаємодіє з базою через запити на мові SQL. MySQL підтримує стандартні операції для роботи з реляційними даними, включаючи створення, читання, оновлення та видалення (CRUD-операції). Вона також забезпечує високу швидкість виконання запитів і надає багаті можливості для оптимізації запитів, що важливо при роботі з великими обсягами даних.

Однією з основних особливостей MySQL є її підтримка транзакцій, що забезпечує цілісність даних навіть при виникненні помилок або відмовах системи. База даних використовує систему зберігання даних InnoDB, що дозволяє проводити одночасне виконання багатьох запитів і забезпечує підтримку атомарності, консистентності, ізоляції та довговічності транзакцій (ACID). MySQL також має розвинену систему індексів, що дозволяє значно прискорити виконання складних запитів, і підтримує різні типи з’єднань між таблицями (JOIN), що дозволяє легко працювати з взаємопов’язаними даними.

2.3.3 База даних PostgreSQL

PostgreSQL − це об’єктно-реляційна система керування базами даних (СКБД), яка забезпечує високу надійність, масштабованість та підтримку складних запитів. Вона підтримує широкий спектр функцій, таких як транзакції ACID, повнотекстовий пошук, індексація, зберігання великих об’єктів (BLOB), а також можливість створення складних типів даних та користувацьких функцій. PostgreSQL є відкритим програмним забезпеченням, що надає можливість розширювати його функціональність за допомогою плагінів і розширень. Завдяки своїй надійності і здатності працювати з великими обсягами даних, PostgreSQL є

одним із найбільш популярних виборів для розробки складних корпоративних додатків і веб-сервісів.

Однією з основних переваг PostgreSQL є підтримка розширених можливостей для роботи з даними, таких як зберігання JSON, географічні дані (за допомогою розширення PostGIS) та можливість роботи з масивами. Крім того, PostgreSQL має сильну підтримку паралельної обробки запитів і реплікації, що дозволяє забезпечити високу доступність і масштабованість на великих інфраструктурах. Завдяки цим характеристикам, PostgreSQL ідеально підходить для використання у системах, що вимагають високої надійності і швидкості обробки великих даних, таких як телекомунікаційні та фінансові сервіси, а також чат-боти і веб-додатки.

2.3.4 База даних Redis

Redis − це база даних типу ключ-значення, яка зберігає дані в оперативній пам’яті, що робить її надзвичайно швидкою для операцій з читання та запису. Завдяки цьому Redis ідеально підходить для використання в системах, де необхідна висока продуктивність, зокрема для кешування, управління сесіями, черг повідомлень та інших тимчасових даних. Redis підтримує різні типи даних, такі як рядки, списки, множини, хеші та інші, що дозволяє гнучко працювати з різними структурами даних. Крім того, Redis забезпечує підтримку атомарних операцій, таких як списки черг та інші корисні функції, що робить його корисним для реалізації складних алгоритмів.

Одна з ключових особливостей Redis − це підтримка механізмів реплікації, що дозволяє створювати копії бази даних для підвищення доступності та відмовостійкості. Також Redis підтримує механізми персистентності, хоча і з орієнтацією на швидкість роботи в пам’яті, що дозволяє зберігати дані на диску для їх відновлення після перезавантаження. Redis підтримує публікацію/підписку (pub/sub), що дозволяє створювати масштабовані системи з обміну повідомленнями між різними компонентами. Завдяки своїй швидкості і гнучкості,

Redis активно використовується для обробки запитів у реальному часі, що робить його популярним вибором для Telegram-ботів, які потребують миттєвого реагування на запити користувачів.

2.3.5 База даних Mongo DB

MongoDB − це документо орієнтована база даних, яка зберігає дані у форматі BSON (Binary JSON). Цей формат дозволяє зберігати складні та неструктуровані дані, що робить MongoDB ідеальним рішенням для систем, де дані можуть бути різноманітними і не мають чіткої схеми. MongoDB підтримує гнучку модель даних, що дозволяє зберігати різні типи інформації в одному документі. Вона також підтримує індекси, що дозволяє ефективно здійснювати пошук і сортування даних. Система базується на принципі «відмови від транзакцій» на користь швидкості, що забезпечує високу продуктивність при великих навантаженнях.

Одна з основних переваг MongoDB − це її масштабованість. MongoDB підтримує горизонтальну масштабованість за допомогою шардінгу, що дозволяє розподіляти дані на кілька серверів і автоматично керувати ними. Це дозволяє забезпечити високу доступність і обробку великих обсягів даних без втрати продуктивності. Крім того, MongoDB має розвинену систему реплікації, що забезпечує високу надійність і доступність даних навіть у випадку відмови серверів. Завдяки своїм особливостям, MongoDB є популярним вибором для розробки високо навантажених і динамічних веб-додатків, таких як чат-боти і системи з великим обсягом даних.

2.3.6 Висновки щодо вибору баз даних для Telegram-бота

Для Telegram-бота зворотного зв’язку MongoDB є ідеальним вибором з кількох ключових причин:

− MongoDB є нереляційною базою даних (NoSQL), що дозволяє зберігати дані у форматі JSON-подібних документів. Це дає велику гнучкість у зберіганні

інформації про користувачів і їх запити, адже структура даних може змінюватися в будь-який момент без необхідності переконфігурації схеми. Для Telegram-бота, який обробляє різноманітні типи запитів, таких як текстові повідомлення, зображення, відео та файли, MongoDB дозволяє зручно зберігати ці дані з різними форматами та полями;

− Telegram-бот зворотного зв’язку може отримувати велику кількість запитів від користувачів, що вимагає швидкої обробки та зберігання даних. MongoDB підтримує горизонтальну масштабованість, що означає, що при зростанні навантаження можна додавати нові сервери для розподілу даних. Це робить MongoDB оптимальним вибором для проектів з високими вимогами до масштабованості та доступності;

− MongoDB має високу швидкість читання і запису завдяки своїй природі зберігання документів у вигляді BSON. Це дозволяє обробляти великі обсяги даних у реальному часі, що важливо для чат-бота, що взаємодіє з великою кількістю користувачів та обробляє численні запити одночасно;

− MongoDB має добре розвинені бібліотеки для багатьох мов програмування, у тому числі для Python, який зазвичай використовують для розробки Telegram-ботів. Завдяки асинхронній обробці запитів в MongoDB, боти можуть працювати ефективно, навіть коли кількість одночасних запитів від користувачів зростає;

− MongoDB має простий і зрозумілий API, що полегшує інтеграцію з іншими компонентами системи, такими як Telegram API. Оскільки з Telegram-ботом часто потрібно працювати з неструктурованими або змінними даними (наприклад, текст, файли, зображення), MongoDB є природним вибором, оскільки дозволяє легко адаптувати структуру зберігання інформації.

Таким чином, MongoDB є найкращим вибором для Telegram-бота зворотного зв’язку завдяки своїй гнучкості, масштабованості, швидкодії та простоті інтеграції з іншими компонентами системи.

2.4 Telegram API

API Telegram забезпечує інструменти для створення ботів, які можуть взаємодіяти з користувачами та виконувати різноманітні функції. Для роботи з API Telegram на Python використовується ряд бібліотек, що спрощують взаємодію з API. Ось детальний опис принципів роботи з API Telegram без акценту на конкретні команди.

Telegram API функціонує за принципом запитів та відповідей. Боти, що використовують API, можуть отримувати повідомлення, надсилати їх, а також взаємодіяти з різними компонентами платформи Telegram, такими як групи, канали, користувачі та інші боти. Всі запити до API Telegram здійснюються через HTTP або HTTPS, що забезпечує безпечний обмін даними.

Щоб почати використовувати API Telegram, необхідно створити бота та отримати токен аутентифікації. Токен є унікальним ідентифікатором, який бот використовує для ідентифікації в системі. Цей токен необхідно зберігати в безпечному місці, оскільки він надає доступ до бота.

Для взаємодії з API Telegram зазвичай використовуються бібліотеки, які реалізують усі методи, надані API. Наприклад, бібліотеки, такі як python-telegram-bot, aiogram або telepot, значно спрощують процес взаємодії з API.

Бот може отримувати повідомлення та інші оновлення через методи, такі як опитування або вебхуки. У випадку опитування бот періодично надсилає запити до API, щоб отримувати нові повідомлення. При використанні вебхуків бот реєструє URL, куди Telegram надсилає дані про нові події, наприклад такі як повідомлення.

Коли бот отримує нове повідомлення, він може обробити його, щоб визначити, що з ним робити. Це може включати перевірку вмісту повідомлення, ідентифікацію користувача, який надіслав повідомлення, та прийняття відповідних дій на основі отриманих даних.

Бот може надсилати повідомлення користувачам, групам або каналам. Для цього потрібно використовувати відповідний метод API, надаючи необхідні

параметри, такі як текст повідомлення, ID чату та інші опції форматування. Бот може надсилати текстові повідомлення, фотографії, відео, файли та інші типи контенту.

Боти можуть бути налаштовані на отримання команд від користувачів, наприклад, через текстові команди або кнопки в повідомленнях. Бот може реалізувати логіку, яка реагує на певні команди, забезпечуючи користувачів інформацією або виконуючи дії відповідно до запитів.

2.5 Бібліотеки Telegram

Щоб вибрати найбільш відповідний інструмент для розробки Telegram-бота, проведемо огляд популярних бібліотек, таких як python-telegram-bot, aiogram, та telepot. Основна увага приділялася простоті інтеграції, асинхронності, продуктивності та підтримці функціоналу Telegram API.

− python-telegram-bot − це одна з найпоширеніших бібліотек для створення ботів, що має велику документацію та активну спільноту розробників. Вона підтримує всі функції Telegram API та має просту структуру для розробки синхронних ботів. Недолік: обмежена підтримка асинхронних операцій, що може вплинути на продуктивність при великому навантаженні;

− aiogram − ця бібліотека з акцентом на асинхронність дозволяє значно покращити продуктивність при обробці великої кількості одночасних запитів. Вона є оптимальним вибором для проектів, де потрібно одночасно обслуговувати багато користувачів. Недолік: трохи складніша у використанні для початківців, оскільки потребує знань роботи з асинхронними функціями;

− Telepot − це легка бібліотека, що надає базовий функціонал для створення ботів. Підходить для невеликих проектів з простими вимогами. Недолік: менш популярна і має менше можливостей у порівнянні з іншими бібліотеками.

Після аналізу було виявлено, що для проекту Telegram-бота зворотного зв’язку найкращим вибором є aiogram, оскільки він забезпечує асинхронну

обробку запитів, що дозволяє ефективно обробляти велику кількість одночасних заявок користувачів.

2.6 Висновки до розділу

Розділ «Теоретичний розділ» описує загальні характеристики системи. Далі в розділі було виконано аналіз мов програмування, види баз даних та види бібліотек.

3 СИНТЕЗ СИСТЕМИ

3.1 Цілі впровадження та опис комп'ютерної системи ІТ-компанії «Диверсіті ІТ»

Комп'ютерна система ІТ-компанії «Диверсіті ІТ» призначена для взаємодії між підрозділами компанії. Також вона обладнана Telegram-ботом зворотного зв'язку.

Компанія створює індивідуальні програмні продукти, які повністю відповідають потребам клієнта.

Основні напрями діяльності:

− ERP-системи (Enterprise Resource Planning) − Комплексні рішення для управління ресурсами компанії. Функції: управління ланцюгами постачання, контроль витрат та фінансових потоків, планування виробництва та облік ресурсів;

− автоматизовані системи збору та обробки даних − Рішення для аналітики та моніторингу показників бізнесу у реальному часі. Приклад: програми для відстеження ефективності торгових точок.

Система спрямована на автоматизацію процесів зворотного зв'язку в соціальній мережі Telegram. Telegram-бот є інструментом, що допомагає оптимізувати комунікацію, спрощуючи збір і обробку заявок на прорахунок замовлень, запитів на консультації та повідомлень про проблеми.

Цілі системи клієнтам:

− переглядати інформацію про компанію − бот має надати основні відомості про діяльність компанії;

− отримати зворотний зв'язок написавши в Telegram-бота компанії повідомлення, відправивши фото, відео, файл, текст, голосове повідомлення з метою прорахунку замовлення від клієнта на розробку ПЗ з початковими даними для ТЗ або з метою консультації та виникненні проблем у клієнта з ПЗ наданим компанією раніше;

− отримати зворотній зв’язок, залишивши номер телефону для дзвінка від адміністрації з метою консультації по розробці ПЗ та можливість надати початкові дані для ТЗ або при виникненні проблем у клієнта з ПЗ наданим компанією раніше.

Цілі системи адміністрації:

− доступ до списку повідомлень, надісланих клієнтами, з можливістю відповідати на них, з метою надання консультації клієнту, узгодження ТЗ після його прорахунку або відповіді на виникненні проблем у клієнта з вже наданим ПЗ, методом надання відповіді через текстові повідомлення, маючи можливість також надіслати фото, відео, файл або голосові відповіді або дзвінок на запит клієнта;

− управління списком користувачів та клієнтів, які взаємодіяли з ботом;

− блокування та розблокування користувачів за необхідності;

− розсилка повідомлень клієнтам (за винятком заблокованих користувачів);

− зняття затримки повідомлень з користувачів або клієнтів.

Початкові дані від клієнта – клієнт зазвичай надає загальні вимоги або ідеї, оскільки не володіє необхідною кваліфікацією для розробки детального ТЗ. Це може включати:

− загальне бачення проекту;

− проблеми, які система повинна вирішити;

− вимоги щодо функціоналу.

Формування ТЗ виконавцем – виконавець компанія «Диверсіті ІТ» розробляє технічне завдання на основі отриманих початкових даних. Цей процес включає:

− збір та уточнення інформації від клієнта;

− спільне обговорення ідей, внесення правок та роз’яснення технічних нюансів;

− створення попередньої структури системи, що включає як функціональні блоки, так і апаратну реалізацію (сервери, бази даних, інтеграція з іншими системами);

− розрахунок бюджету, термінів реалізації та технічних характеристик «Диверсіті ІТ» інфраструктури.

Узгодження ТЗ – процес обговорення, внесення правок ТЗ між компанією та замовником. Обговорення та узгодження проводиться в боті зворотного зв’язку, телефонному режимі або зустрічі в офісі компанії «Диверсіті ІТ»

Зворотний зв’язок із клієнтами через «Telegram-бот зворотного зв’язку» здійснюється для забезпечення швидкої та ефективної комунікації між компанією «Диверсіті ІТ» та її клієнтами.

Запити від клієнтів:

− отримання консультацій щодо послуг компанії;

− надання початкових даних ТЗ;

− узгодження технічного завдання після його прорахунку;

− надання підтримки у разі виникнення проблем із вже впровадженим ПЗ.

Формати відповідей клієнту:

− текстові повідомлення;

− відправлення фото, відео, файлів або голосових відповідей;

− можливість здійснення телефонного дзвінка на запит клієнта;

− можливість зустрічі в офісі компанії на запит клієнта.

Етапи взаємодії із замовником:

а) збір вимог;

1) аналіз потреб клієнта і вимог до програмного забезпечення;

2) документування вимог;

3) уточнення обмежень проекту, зокрема часових рамок, бюджету та технічних аспектів;

б) погодження ТЗ;

1) складання ТЗ, яке містить усі ключові аспекти проекту: функціонал, архітектуру, інтерфейси та критерії прийняття;

2) обговорення ТЗ із замовником для внесення коригувань, якщо такі необхідні;

3) остаточне затвердження ТЗ як основного документа, який регламентує подальшу розробку та взаємодію;

в) етапи приймання робіт;

1) проведення проміжного тестування після завершення кожного етапу розробки для перевірки відповідності до погоджених вимог;

2) виконання фінального тестування перед здачею проєкту, щоб підтвердити готовність продукту до використання.

Передача замовнику усіх необхідних матеріалів, таких як: технічна документація з описом архітектури, функціональних можливостей та особливостей розробленого програмного забезпечення; інструкції з налаштування, експлуатації та адміністрування системи; посібник користувача, який пояснює, як працювати з програмним забезпеченням; інструкції з інтеграції продукту в існуючи IT-інфраструктуру замовника; надання структурованого, задокументованого вихідного коду проекту, відповідно до умов договору; код супроводжується інструкціями для розробників, які дозволяють легко вносити зміни або розширювати функціонал у майбутньому.

3.2 Формулювання технічних вимог до комп’ютерної системи

3.2.1 Вимоги до реалізації системи

У якості серверу комп’ютерної системи з ботом зворотнього зв’язку має використовуватися сервер DNS для зменшення витрат.

Telegram API має використовується для приймання та відправлення повідомлень від користувачів.

Aiogram фреймворк має забезпечувати реалізацію логіки взаємодії з ботом, включаючи обробку повідомлень, команд і запитів.

MongoDB має використовується як нереляційна база даних для зберігання інформації про клієнтів.

Структура бази даних повинна включати:

− id користувача Telegram;

− ім’я користувача;

− статус (активний/заблокований);

− статус (бот/не бот).

База даних клієнтів має використовуватися для виконання адмін-функцій, зокрема:

− надсилання повідомлень користувачам, які зберігаються в базі даних, за винятком тих, хто перебуває у списку заблокованих;

− забезпечення можливості блокування користувачів, які порушують правила використання системи (наприклад, займаються спамом);

− управління затримками в надсиланні повідомлень для клієнтів, з якими ведеться узгодження технічного завдання, що сприяє ефективному та швидкому обміну інформацією.

Серверна частина має забезпечувати обробку запитів, взаємодію з базою даних і управління функціоналом адміністрації.

Система повинна відповідати таким вимогам:

− система повинна працювати на різних операційних системах, таких як Windows та Linux;

− система має бути оптимізована для обробки до 100 запитів на секунду при 100 одночасних користувачах. під час пік-завантаження (до 200 одночасних користувачів) система повинна залишатися стабільною з рівнем помилок не більше 0.1%;

− час автоматичних відповідей бота не повинен перевищувати 1 секунду;

− час обробки повідомлень від користувачів до адміністрації бота має складати не більше 20 хвилин для консультацій або вирішення проблем з уже наданим ПЗ. У випадку необхідності прорахунку та уточнення ТЗ час обробки може становити 1-2 дні;

− всі вищезазначені вимоги повинні виконуватись у робочий час компанії, з 8:00 до 18:00;

– система повинна бути здатна обробляти невеликі обсяги даних, зберігаючи інформацію про клієнтів. Потенційний обсяг даних для кожного користувача не повинен перевищувати 1 МБ;

– система повинна забезпечувати безперервну роботу чат-бота, з мінімізацією часу простою. Для цього необхідно мати систему резервного живлення (ДБЖ), що забезпечує мінімум 12 годин роботи за відсутності електропостачання.

3.2.2 Вимоги до функцій виконуваних системою

На сервері DNS комп'ютерної системи має бути розгорнуто DNS сервіс компанії та Telegram-бот зворотного зв'язку.

Telegram-бот зворотного зв'язку компанії «Диверсіті ІТ» в соціальній мережі Telegram має надавати її користувачам/клієнтам такі функції:

− перегляд інформації про компанію «Диверсіті ІТ» − надання основних відомостей про діяльність компанії;

− надсилання повідомлень (фото, відео, файл, текст, голосове повідомлення) до адміністрації компанії «Диверсіті ІТ» для зворотного зв'язку з метою прорахунку замовлення від клієнта на розробку ПЗ та можливість надати початкові дані для ТЗ або з метою консультації та виникненні проблем у клієнта з ПЗ наданим компанією раніше;

− залишення номера телефону через відповідну кнопку та ввід номера, з метою передачі його ботом до адміністрації та очікування дзвінку від адміністрації з метою консультації по розробці ПЗ та можливість надати початкові дані для ТЗ або при виникненні проблем у клієнта з ПЗ наданим компанією раніше.

Окрім цього Телеграм бот має приймати такі формати у якості повідомлень до адміністрації або відповіді від адміністрації: фото, відео, голосові, текстові повідомлення, файли.

Для адміністрації має виводити адмін-панель з такими функціями:

− список всіх користувачів;

− блокування / розблокування користувачів;

− розсилка серед користувачів;

− зняття затримки з користувача.

3.2.3 Вимоги до видів забезпечення

3.2.3.1 Вимоги до інформаційного забезпечення

З метою забезпечення роботи комп'ютерної системи IT-компанії «Диверсіті IT» з Telegram-ботом зворотного зв'язку необхідно визначити вимоги до структури даних в базі даних та інформаційного обміну даних.

Структура бази даних користувачів нереляційної Mongo DB:

− унікальний ідентифікатор документа;

− Telegram ID користувача;

− ім'я користувача;

− стан користувача заблокований/не заблокований;

− стан перевірки на бота користувача.

Інформаційний обмін між компонентами системи відбувається наступним чином:

− користувач Telegram взаємодіє з ботом через Telegram API;

− бот обробляє запити користувача за допомогою бібліотеки Aiogram;

− у разі потреби збереження даних або внесення змін у базу даних MongoDB бот використовує бібліотеку Motor для виконання відповідних запитів.

3.2.3.2 Вимоги до лінгвістичного забезпечення.

Було розроблено модель обслуговування роботи серверу з телеграм ботом як модель масового обслуговування.

Телеграм бот зворотного зв’язку повинен підтримувати в інтерфейсі українську мову та емодзі для кращої візуалізації.

3.2.3.3 Вимоги до технічного забезпечення.

Для ефективної роботи комп’ютерної системи ІТ-компанії «Диверсіті ІТ» з Telegram-ботом зворотного зв’язку, створеного на Aiogram з використанням MongoDB та асинхронної обробки даних, сервер повинен мати наступні компоненти та характеристики.

Процесор (CPU):

− багатоядерний процесор для забезпечення одночасної обробки великої кількості запитів;

− процесор з високою тактовою частотою для швидкого виконання асинхронних операцій.

Оперативна пам’ять (RAM):

− від 8 ГБ і більше для обробки даних.

Дискова система:

− використання SSD для швидкої роботи з файлами, базою даних та логами;

− мінімальний обсяг 50 ГБ.

Мережеве з’єднання:

− низька затримка (Ping) і стабільний канал (мінімум 50 Мбіт/с);

− сервери з пропускною здатністю 100 Мбіт/с або вище для високих навантажень.

Джерело безперебійного живлення (ДБЖ):

− забезпечення роботу сервера при перебоях з електропостачанням або блекаутах.

3.2.3.4 Вимоги до організаційного забезпечення

Для ефективного впровадження та роботи в компанії «Диверсіті ІТ» Telegram-боту зворотного зв’язку необхідно призначити адміністратора або

декількох адміністраторів, які будуть працювати позмінно, та будуть відповідати на запити користувачів боту, додавати чи вилучати користувачів із чорного списку та здійснювати контроль за його роботою. Адміністратор або декілька адміністраторів мають працювати з 8:00 до 18:00 кожного дня для максимальної продуктивності.

3.2.4 Вимоги до захисту інформації

Для забезпечення захисту інформації необхідно виконувати такі вимоги:

− локальний сервер з локальною адресою − використання локального сервера без відкритого доступу до Інтернету мінімізує ризики витоку інформації про клієнтів. Це дозволяє зберігати дані в ізольованому середовищі, забезпечуючи їх безпеку та обмежуючи можливість доступу ззовні;

− захист від спаму − для зменшення ризиків спам-атак потрібно налаштувати затримку між повідомленнями від одного користувача. Це дозволяє уникнути швидкого надходження великої кількості запитів, які можуть призвести до перевантаження бота або сервера;

− захист від бота − перед початком роботи бота необхідно реалізувати механізм перевірки користувача на бота;

− Telegram API має вбудовані заходи безпеки, такі як HTTPS для всіх запитів до серверів Telegram, що гарантує захищене з’єднання між ботом та Telegram. Всі повідомлення, що надсилаються через API, також шифруються за допомогою TLS/SSL, що забезпечує їх захист під час передачі;

− MongoDB також надає вбудовані механізми безпеки, зокрема автентифікацію та авторизацію для доступу до бази даних. За замовчуванням MongoDB використовує role-based access control (RBAC), що дозволяє контролювати доступ до даних на основі ролей користувачів. Крім того, MongoDB підтримує шифрування даних на рівні файлів і мережевих з’єднань.

3.2.5 Вимоги до ергономіки системи

Для забезпечення комфортного використання Telegram-бота зворотного зв’язку, інтерфейс повинен бути максимально простим і інтуїтивно зрозумілим для користувачів усіх рівнів.

Інтерфейс також повинен включати додаткову візуалізацію у вигляді стикерів, щоб зробити процес взаємодії більш приємним та привабливішим.

3.2.6 Розробка схеми функціональної структури

Розробимо схему функціональної структури.

Додаток Телеграм − мобільний або десктопний додаток взаємодіє з користувачами.

Входи додатка: запити від бота.

Виходи додатка: відповіді від Telegram-бота.

Telegram API − набір інтерфейсів для взаємодії з платформою Telegram.

Його входи: повідомлення від користувачів, включаючи фото, відео, файли, голосові, текстові, натисканням кнопок або команди.

Його виходи: відповіді на повідомлення, кнопки, команди.

Телеграм бот зворотного зв’язку − програма мовою Python яка працює на основі Telegram API та взаємодіє з користувачами.

Його входи: команди від користувачів.

Його виходи: відповіді користувачам.

База даних − це організована система для зберігання, обробки та керування даними про користувачів боту в нашому випадку.

Її вхід: запит від API на зміни або перегляд бази даних.

Її виходи: дані що запитується.

Схема функціональної структури зображено на рисунку 3.1

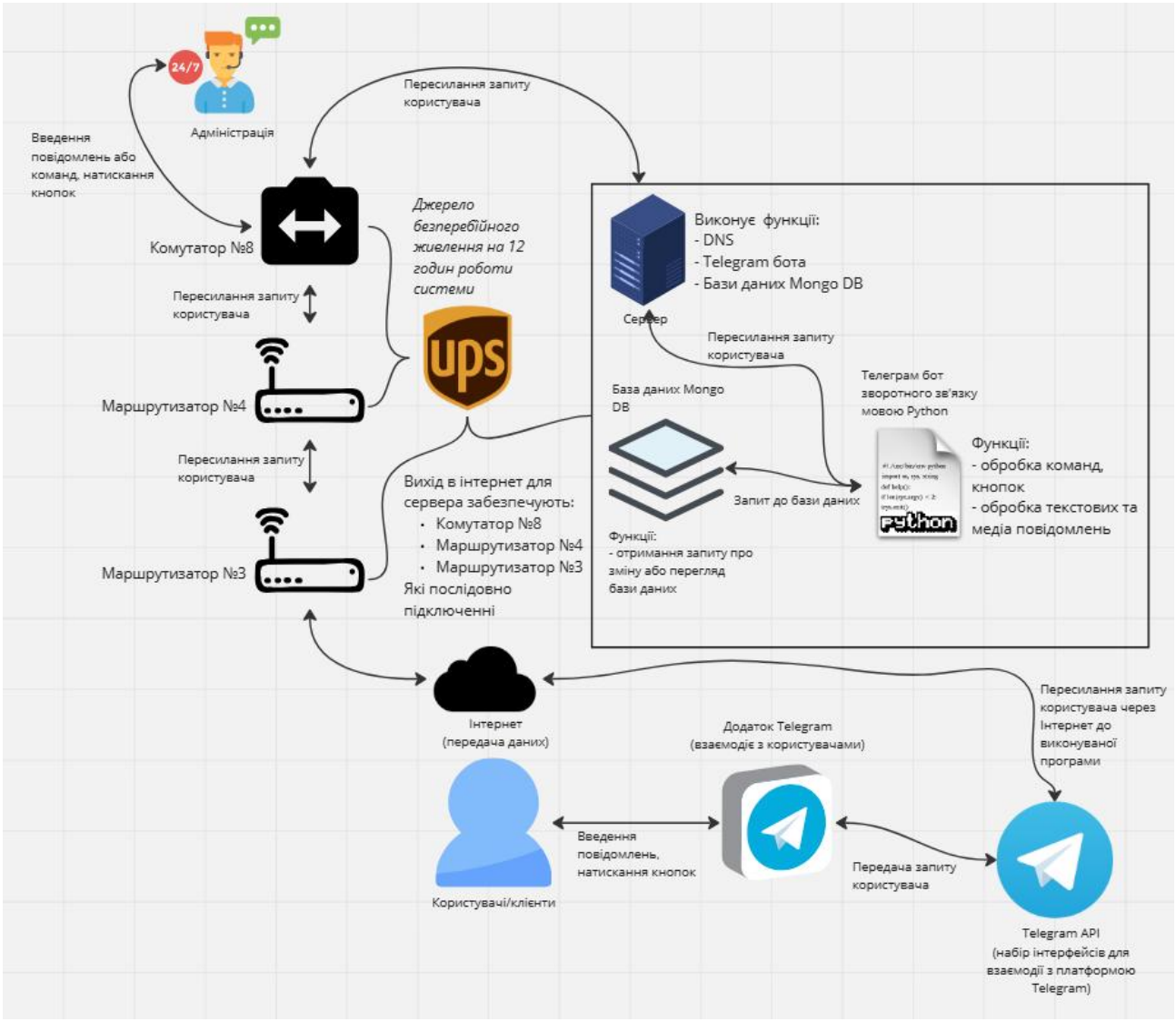

Рисунок 3.1 − Схема функціональної структури

3.3 Стислі відомості про комп'ютерну систему компанії «Диверсіті ІТ»

Мережа компанії «Диверсіті ІТ» розбита на 5 підмереж, кількість пристроїв мінімальну, під яку розраховане, зображено в таблиці 3.1.

Таблиця 3.1 − Розрахована кількість вузлів для підмереж

Підмережа	LAN1 Головний офіс	LAN2 Головний офіс	LAN3 Головний офіс	LAN4 Віддалений офіс	LAN5 Головний офіс
Кількість вузлів	50	27	79	39	76

LAN1− генеральний директор, відділ розробників.

LAN2 − відділ дизайнерів.

LAN3 − відділ консультантів з IT.

LAN4 − відділ тестувальників.

LAN5 − відділ менеджерів проектів, відділ аналітиків.

«Telegram-бот зворотного зв’язку» буде розгортатися в комп’ютерній мережі компанії «Диверсіті IT», схема якої зображено на рисунку 3.2.

В якості сервера для Telegram-боту зворотного зв’язку в комп’ютерній мережі буде використовуватися сервер DNS, з метою зменшення витрат, який знаходиться в підмережі LAN3, потужності якого дозволяють додатково розгорнути Telegram бота зворотного зв’язку.

Структура підключень сервера до інтернету виглядає таким чином:

− сервер DNS/Telegram-бота підключений до комутатора №8 за допомогою прямого кабелю;

− комутатор №8 підключений до маршрутизатора №4 за допомогою прямого кабелю;

− маршрутизатор №4 з’єднаний з маршрутизатором №3 через Serial DTE кабель;

− маршрутизатор №3 підключений до Інтернету через провайдера за допомогою технології NAT, забезпечуючи доступ до Інтернету для всіх підключених пристроїв у мережі компанії.

В під мережі LAN3 знаходиться відділ консультантів, з якого необхідно виділити 1-2 консультанти для адміністрування ботом зворотного зв’язку.

Вимоги до сервера DNS для мережі «Диверсіті IT»:

− сервер DNS повинен бути здатним обробляти великі обсяги запитів без зниження продуктивності, навіть при високій інтенсивності трафіку. Мінімальна кількість запитів, яку сервер має бути здатний обробляти на секунду, − 1000 запитів за секунду при навантаженні до 1000 одночасних користувачів. Під час

пік-завантаження (до 2000 одночасних користувачів) система повинна залишатися стабільною з рівнем помилок не більше 0.1%;

− час відповіді на DNS запит повинен бути не більше 50 мс.

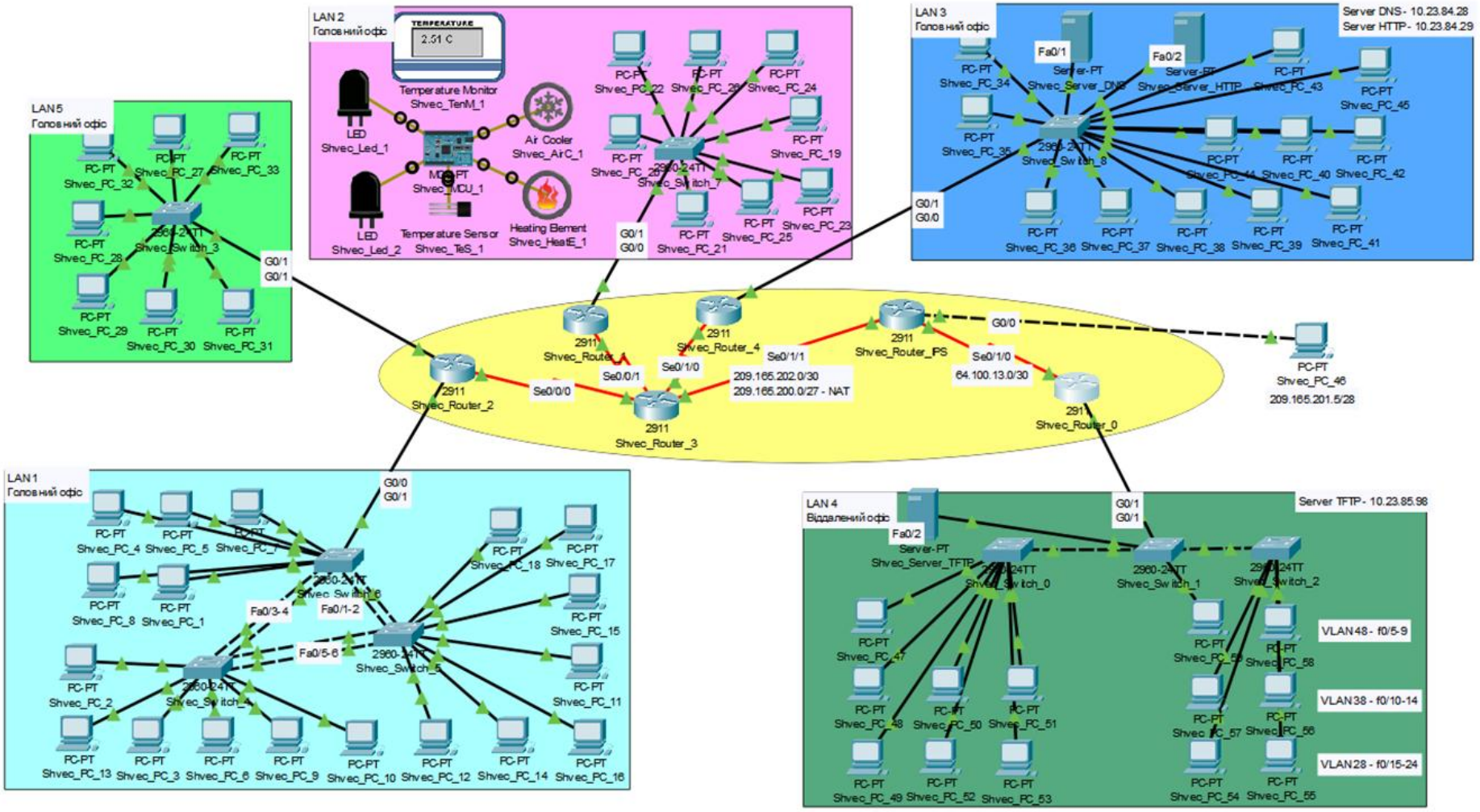

Рисунок 3.2 − Схема комп’ютерної мережі «Диверсіті ІТ»

3.4 Структурна схема комплексу технічних засобів «Диверсіті ІТ»

Переглянемо з’єднання в нашій мережі:

− між усіма маршрутизаторами використовується кабелі Serial DTE;

− між комутаторами та маршрутизаторами використовується прямі кабелі;

− між комутаторами та комутаторами використовується крос-кабель;

− між комутаторами та ПК використовується також прямий кабель.

Враховуючи топологічну схему комп’ютерної мережі в пункті 3.2, кількість підмереж та вузлів, додавання функцій Telegram-бота на сервер DNS, було розроблено схему комплексу технічних засобів комп’ютерної мережі компанії «Диверсіті ІТ» (рисунок 3.3).

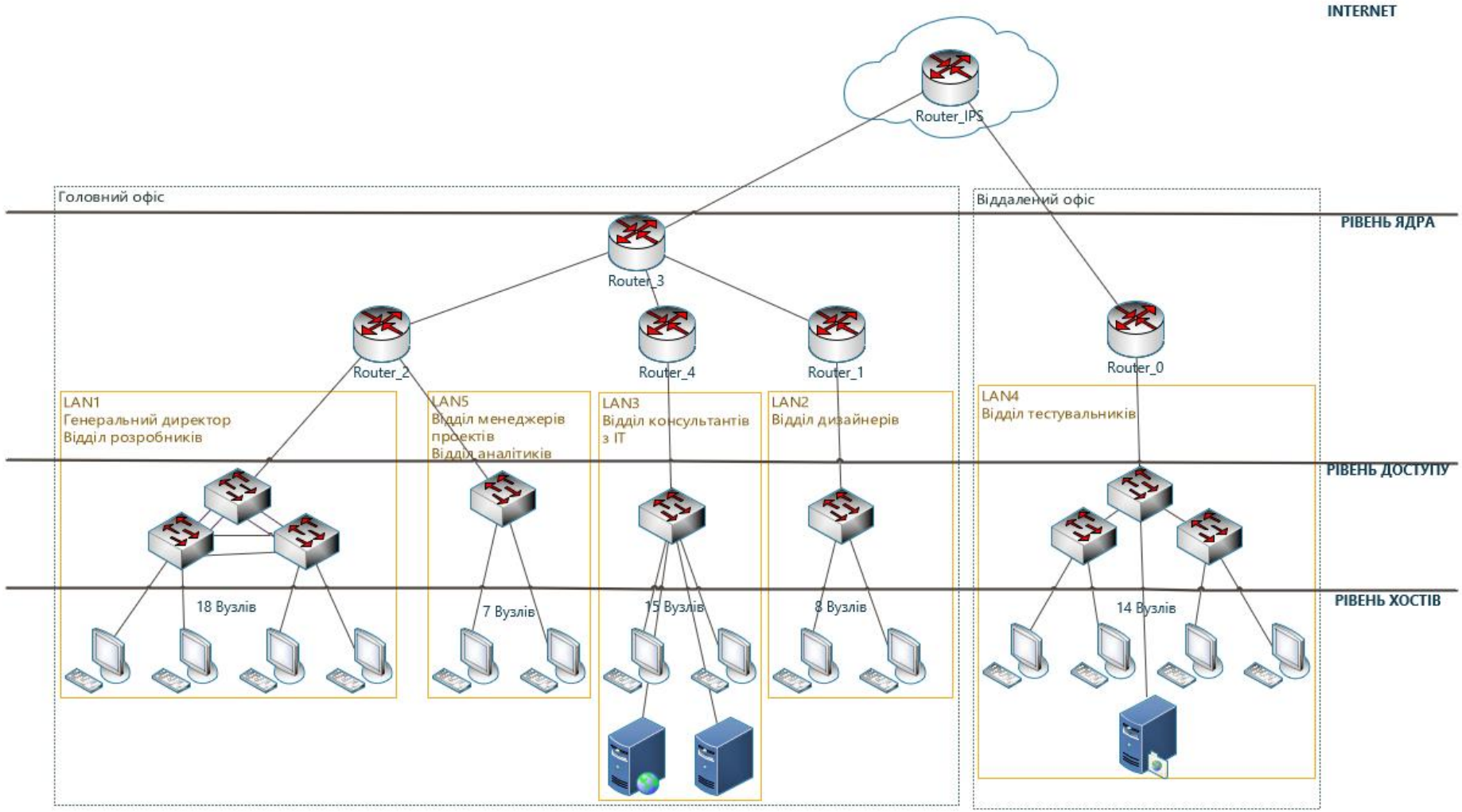

Рисунок 3.3 − Структурна схема комплексу технічних засобів мережі «Диверсіті IT»

3.5 Вибір та обґрунтування застосування апаратних засобів

Враховуючи вимоги до реалізації системи (див.п. 3.2.1) та враховуючи що для Telegram-боту зворотного зв’язку та DNS серверу буде використовуватися один фізичний сервер, враховуючи вимоги до сервера DNS (див.п. 3.3) оберемо сервер компанії Cisco, а саме модель Cisco UCS C240 M4 12 LFF 2U , яка вже встановлена в мережі LAN3 і використовується для сервера DNS. Основні переваги для системи зворотного зв’язку даного севрера:

− сервер оснащений двома процесорами Intel Xeon v4, кожен з яких підтримує до 12 ядер (24 потоки). Це забезпечує високу швидкість обробки асинхронних запитів від Telegram API, обробку бази даних MongoDB і виконання додатково серверних завдань (DNS) при високих навантаженнях;

− максимальна швидкість мережевого з’єднання до 1000 Мб/с забезпечує оперативний обмін даними між ботом і Telegram API, а також із клієнтами бази даних;

− сервер підтримує систему віддаленого управління CIMC (Cisco Integrated Management Controller), що дозволяє адміністраторам компанії з підмережі LAN1 оперативно реагувати на технічні збої чи необхідність оновлення програмного забезпечення.

Виконання ролі DNS-сервера та одночасно Telegram-боту дозволяє економити кошти, уникаючи придбання окремого сервера для цієї задачі.

Розрахунок енергоспоживання:

− блок живлення: 2 шт по 1200 вт, проте реальне споживання не більше 400 ват;

− мережеве обладнання, а саме 2 роутери по 120 ват та комутатор 50 ват під час реального споживання.

Загальна потужність виходить 700 ват. Для забезпечення роботи системи на 12 годин необхідно 8400 Ват, враховуючи відключення електроенергії під час блекаутів у країні.

В якості ДБЖ оберемо ДБЖ LogicPower LPY-PSW-2500VA+, який відповідає вимогам.

Даний ДБЖ розраховано до 1800 Вт та забезпечує чисту синусоїду. ДБЖ онлайн типу забезпечує завжди мережу 220в в не залежності яка напруга на вході, що важливо для серверів.

Також однієї з головних переваг ж те що даний ДБЖ підтримує тип акумулятора − Lipofe 4 , що являють собою на даний момент найбільш безпечним та довговічним варіантом.

Додатково ДБЖ має захист від перевантаження, короткого замикання, перезарядки.

У якості акумуляторів будуть використовуватися Lipofe 4 акумулятори − 4 шт 24в по 100Ah, підключені паралельно, що збільшує ємність, що дасть запас енергії близько 10k ват заявленої та близько 8.5к ват на розрядку, адже не варто висаджувати акумулятори менше 15% для продовження строку їх служби, враховуючи ККД БЖД близько 95%.

Для більш швидкої зарядки всіх акумуляторів одразу можна використовувати зарядний пристрій на 100-200A 29.2 ± 0.2 В.

Специфікація обладнання для комп’ютерної мережі компанії «Диверсіті ІТ», в якій розміщується сервер DNS / Telegram-бота, а також необхідне обладнання для виходу сервера до Інтернету та підрозділів компанії, що безпосередньо відповідають за функціювання компанії для розробки програмного забезпечення, наведено в таблиці 3.2. Враховано також обладнання для віддаленого офісу компанії.

Таблиця 3.2 − Специфікація обладнання

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документа, опитувального листа	Одиниці виміру	Кількість	Примітки
1	2	3	4	5	6
1.	Маршрутизатор RJ-45 Ports: 3 Тип слота розширення: HWIC, PVDM Технологія Ethernet: Гігабітний Ethernet Вхідна напруга: 110 В змінного струму 220 В змінного струму	Cisco 2911/K9	од.	6	
2.	Інтерфейсний модуль Синхронна максимальна швидкість (на порт): 8 Мбіт/с Послідовні протоколи EIA-232, EIA-449, EIA-530, EIA-530A, V.35 і X.21	Cisco HWIC-2T	од.	7	
3.	Комутатор Деталі порту/слота розширення: 24 x Fast Ethernet Network 2 x Gigabit Ethernet Uplink 2 x Gigabit Ethernet Expansion Slot Управління: RMON, SNMP v1,2c,3, VLAN, QoS, CLI, DHCP, Telnet, HTTP, HTTPS, Системний журнал Вхідна напруга: 110 В змінного струму 220 В змінного струму	Cisco WS-C2960-24LC-S	од.	9	Головний офіс − 6 од. Віддалений офіс – 3 од.

Продовження таблиці 3.2

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документа, опитувального листа	Одиниці виміру	Кількість	Примітки
1	2	3	4	5	6
4.	Сервер Кількість портів Ethernet 2 x RJ45 Швидкість з’єднання 1 Гбіт/с Тип процесора Intel Xeon E5 v3/v4	Cisco UCS C240 M4 12 LFF 2U	од.	3	Головний офіс – 2 од. Віддалений офіс – 1 од.
5.	Комп’ютер Відеокарта: GeForce RTX3060 Процесор: AMD Ryzen 5 5500 Обсяг оперативної пам’яті: 16 ГБ Обсяг SSD: 480 ГБ	COBRA Advanced (A55.16.H1S 4.36.16983)	од.	59	Головний офіс – 45 од. Віддалений офіс – 13 од.
6.	Монітор Максимальна роздільна здатність дисплея: 1920x1080 Інтерфейси: VGA, DisplayPort, 2 x HDMI Споживана потужність: У режимі роботи: 22 Вт У режимі очікування: 0.3 Вт У вимкненому стан: 0.3 Вт	AOC 24G2SAE/BK	од.	59	Головний офіс – 45 од. Віддалений офіс – 13 од.
7.	Клавіатура та мишка (набір) Розкладка: Eng / Ru / Ukr Максимальна роздільна здатність сенсора, dpi: 1000 Комплектація набору: 2 в 1 Тип клавіатури: Мембранні Комплектація: Клавіатура Миша, USB адаптер	Xiaomi Miiiw Wireless	од.	59	Головний офіс – 45 од. Віддалений офіс – 13 од.

Кінець таблиці 3.2

Позиція	Найменування і технічна характеристика	Тип, марка, позначення документа, опитувального листа	Одиниці виміру	Кількість	Примітки
1	2	3	4	5	6
8.	ДБЖ Кількість розеток 2 х євророзетки Вихідна потужність 1800 Вт Діапазон вхідної напруги під час роботи від мережі 140-275 В Потужність 2500 ВА Акумуляторна батарея Зовнішня	LogicPower LPY-PSW-2500VA+	од.	1	Головний офіс − 1 од.
9.	Акумулятор Номінальна напруга: 25.6 В Номінальна ємність: 100 А·год Місткість: 2560 Вт. Макс. Напруга заряду: 29.2 ± 0.2 В Макс. Струм заряду: 100 А	Jsdsolar LiFePo4 24V 100AH 2560Wh	од.	4	Головний офіс − 1 од.

3.6 Висновки до розділу

Розділ «Синтез системи» описує основні вимоги, цілі впровадження системи Telegram-боту зворотного зв’язку.

Також було розроблено функціональну схему структури та перераховано необхідне обладнання для реалізації системи.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ КОМПАНІЇ «Диверсіті ІТ»

4.1 Призначення й область застосування програмного забезпечення

Telegram-бот зворотного зв’язку створений для автоматизації взаємодії між користувачами та адміністрацією компанії. Основною метою бота є забезпечення швидкого збору заявок замовлень, а також надання оперативного зворотного зв’язку у разі виникнення питань чи проблем. Використання такого бота спрощує процес комунікації, зменшує навантаження на працівників підтримки й підвищує якість обслуговування клієнтів.

Це програмне забезпечення ідеально підходить для компаній, які вже ведуть Telegram-канали, оскільки воно дозволяє здійснювати всі взаємодії в межах однієї платформи. Таким чином, користувачі отримують зручний інструмент для зв’язку, а компанія − централізований канал збору даних. Бот також може використовуватися для надсилання повідомлень про акції, оновлення чи важливі події, що збільшує залучення клієнтів.

4.2 Обґрунтування технічних характеристик програми компанії «Диверсіті ІТ»

Основне завдання розробки Telegram-бота зворотного зв’язку − автоматизація процесу збору заявок, замовлень та зворотного зв’язку від клієнтів компанії. Враховуючи специфіку проекту, обмеження стосуються обробки даних: бот працює тільки з текстовими і стандартними мультимедійними форматами, підтримуваними Telegram API, а також передбачає роботу з обмеженою кількістю запитів у секунду згідно з лімітами платформи.

Алгоритм Telegram-бота будується за принципом обробки подій. Користувач надсилає запит, який надходить через Telegram API до основної програми, написаної мовою Python. Далі бот аналізує запит, визначає його тип і

виконує відповідну дію: запис у базу даних, надсилання відповіді користувачеві або перенаправлення запиту до адміністратора.

Дані користувачів зберігаються в MongoDB, що дозволяє працювати з великою кількістю заявок і легко масштабуватися. Носіями виступають SSD-диски, розподілені між системними файлами та базою даних для забезпечення швидкості операцій запису та читання.

4.3 Опис розробленого програмного забезпечення комп'ютерної системи ІТ-компанії «Диверсіті ІТ»

4.3.1 Загальні відомості

Програма отримала назву «Telegram-бот зворотного зв'язку». Її основне призначення − забезпечення автоматизованої взаємодії між компанією та користувачами через Telegram.

Для функціювання програми необхідне наступне програмне забезпечення.

Операційна система − Ubuntu Server або інша ОС, сумісна з Python та MongoDB.

Середовище виконання Python − версія 3.8 або новіша.

Система управління базами даних − MongoDB, що використовується для зберігання даних про користувачів.

Telegram API − для забезпечення інтеграції програми з месенджером Telegram.

Додаткові бібліотеки Python − Aiogram для взаємодії з Telegram API, Pymongo для роботи з MongoDB, logging для створення логів.

Програма написана мовою Python, що забезпечує її простоту, масштабованість і зручність підтримки. MongoDB обрано як базу даних завдяки її гнучкій структурі та можливості зберігати великі обсяги неструктурованих даних.

4.3.2 Функціональне призначення

Програма призначена для автоматизації взаємодії між компанією та її клієнтами. Основні класи завдань, які вирішує бот, включають прийом і обробку заявок.

Функціональні обмеження бота включають залежність від стабільного підключення до Інтернету, обмежену пропускну здатність API Telegram (до 100 запитів у секунду) та необхідність ручного втручання адміністратора.

4.3.3 Опис логічної структури програми

Телеграм-бот зворотного зв’язку, розроблений на Python, побудований за чіткими алгоритмами роботи, які поділяються на два основні напрями: для користувачів (рисунок 4.1) та для адміністрації (рисунок 4.2).

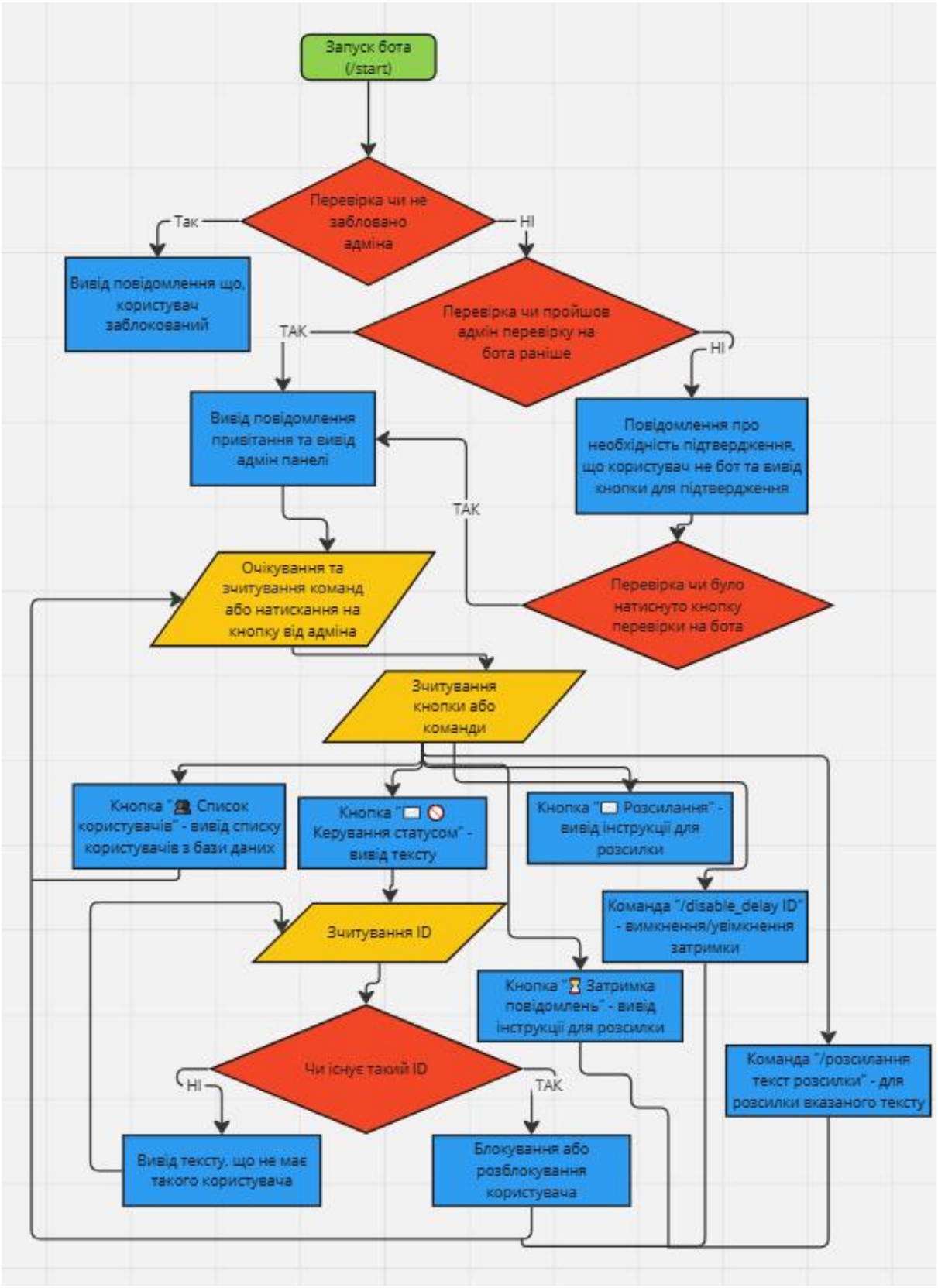

Рисунок 4.1 − Алгоритм роботи боту для користувачів

Для користувачів бот пропонує інтуїтивно зрозумілий процес взаємодії. При запуску програми користувач проходить первинну перевірку: чи він не заблокований у системі та чи не є ботом, підтверджуючи це натисканням відповідної кнопки. Після успішної перевірки користувач отримує доступ до функціоналу бота, який представлений трьома основними кнопками: «Про нас», «Передзвоніть мені» та «Зворотний зв’язок».

Кнопка «Про нас» надає інформацію про компанію у вигляді текстового повідомлення. Опція «Зворотний зв’язок» пропонує користувачеві залишити повідомлення для адміністрації, після чого вмикається режим затримки для запобігання спаму, дозволяючи користувачеві використовувати лише кнопки бота. Функція «Передзвоніть мені» дозволяє клієнту залишити свій номер телефону для зворотного дзвінка з боку адміністрації.

Для адміністрації бот забезпечує додатковий рівень перевірки: наявність прав адміністратора, перевірка на блокування та підтвердження, що не є ботом. Адміністративна панель представлена у вигляді чотирьох основних кнопок: «Список користувачів», «Керування статусом», «Розсилання» та «Затримка повідомлень».

Функція «Список користувачів» дозволяє переглядати актуальну базу користувачів, зареєстрованих у боті. Кнопка «Керування статусом» пропонує ввести ID користувача для його блокування або розблокування. Функція «Розсилання» дозволяє адміністраторам відправляти масові повідомлення, які отримують усі користувачі бота, окрім заблокованих. Опція «Затримка повідомлень» надає можливість відключити обмеження на текстові повідомлення для певних користувачів у разі потреби.

Крім цього, адміністрація має доступ до універсальних інструментів відповіді. Вона може надсилати користувачам текстові повідомлення, зображення, відео, аудіофайли, голосові повідомлення або документи − аналогічно до функціоналу, доступного для користувачів. Це дозволяє адміністраторам швидко й ефективно відповідати на запити та забезпечувати зручну взаємодію з клієнтами.

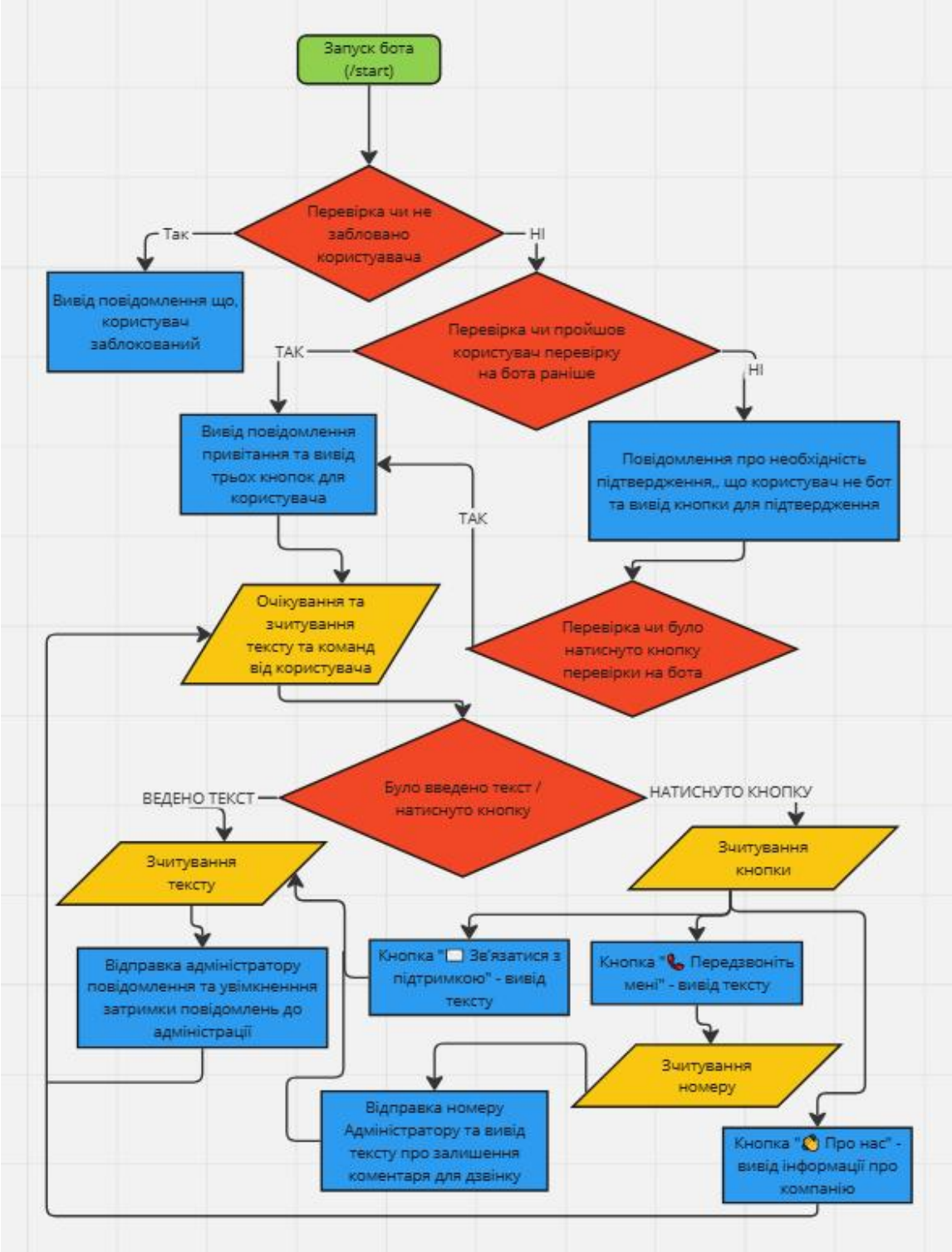

Рисунок 4.2 − Алгоритм роботи боту для адміністрації

Програма телеграм боту зворотного зв’язку побудована з використанням структурованої архітектури (рисунок 4.3), яка складається з кількох основних

файлів, написаних мовою Python. Такий підхід забезпечує легкість у розробці, підтримці та масштабуванні бота.

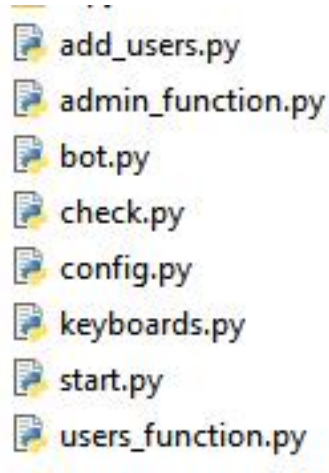

Рисунок 4.3 − Структурована архітектура

Головний файл − цей файл відповідає за імпорт необхідних бібліотек, налаштування логування для відстеження дій та помилок, а також за реалізацію основної логіки роботи бота. У ньому містяться алгоритми обробки запитів користувачів і відповіді на них.

Файл конфігурації − містить усі необхідні параметри конфігурації, включаючи токен бота, налаштування бази даних MongoDB, id адміністратора та адресу бази даних.

Файл старту − реалізує команду старту, яка ініціює перевірку на наявність користувача в базі даних, визначає, чи заблокований користувач, чи є він ботом, і додає його до бази даних у разі необхідності.

Файл адмін-функцій − містить функції адміністративної панелі, зокрема перегляд списку користувачів, управління їхнім статусом блокування, відправлення масових розсилок і вимкнення затримки повідомлень для окремих користувачів.

Файл користувацьких функцій − реалізує основний функціонал для користувачів, зокрема надання інформації про компанію, функцію зв’язку з оператором через бота або за допомогою дзвінка.

Файл перевірки − виконує перевірку статусу користувача: чи не заблокований він, чи пройшов необхідну перевірку на бота. Для адміністративної панелі файл також перевіряє, чи є користувач адміністратором.

Файл додавання користувачів − містить функції для додавання нових користувачів до бази даних, а також визначення структури даних, які зберігаються.

Файл кнопок − реалізує набір кнопок для взаємодії як з користувачами, так і з адміністративною панеллю, забезпечуючи інтуїтивно зрозумілий інтерфейс.

Такий поділ файлів забезпечує зручність в обслуговуванні коду, спрощує внесення змін і полегшує масштабування функціоналу бота в майбутньому.

Для створення телеграм бота використовуються основні бібліотеки (рисунок 4.4 − 4.5), які забезпечують необхідну функціональність. aiogram надає інструменти для взаємодії з Telegram API, обробки повідомлень та команд користувачів. Зокрема, Bot використовується для надсилання повідомлень, а Dispatcher керує отриманням і розподілом повідомлень. FSMContext дозволяє зберігати стан користувача для створення діалогів, а MemoryStorage використовується для збереження цих станів в пам’яті.

Бібліотеки logging і time допомагають вести журнал подій і встановлювати затримки в процесі взаємодії з користувачем. asyncio забезпечує асинхронне виконання, що дозволяє ефективно обробляти кілька запитів одночасно. Таким чином, ці бібліотеки працюють разом для створення функціонального та швидкодійного бота.

```
import logging
from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.dispatcher import FSMContext
from aiogram.types import Message
from aiogram.dispatcher.filters.state import State, StatesGroup
from aiogram import types
```

Рисунок 4.4 − Бібліотеки основні перша частина

```
import time
import asyncio
```

Рисунок 4.5 − Бібліотеки основні друга частина

Додатково, для забезпечення коректної роботи програми та її модульності, імпортуємо необхідні функції та змінні з інших файлів (рисунок 4.6), на які розбивається код. Зокрема:

− config.py містить важливі конфігурації, такі як API-токен для бота, id адміністратора, з'єднання з MongoDB та назву бази даних. З цього файлу імпортуються API_TOKEN, admin_id, MONGO_URI, DB_NAME та users_collection для налаштування підключення до бази даних;

− check.py надає функції для перевірки статусу користувачів, перевірки доступу адміністратора, а також для визначення, чи заблокований користувач. З цього файлу імпортуються функції check_user_status, check_admin_access, is_user_banned та is_admin;

− add_users.py містить функцію add_user, яка відповідає за додавання користувачів до бази даних;

− keyboards.py забезпечує створення клавіатур для користувачів та адміністрації. З цього файлу імпортуються функції user_menu, admin_menu та back_menu для відображення відповідних кнопок у боті;

− users_function.py містить функції для обробки запитів користувачів, таких як about_us1, contact_admin1, back_to_menu1, request_phone_number1 та handle_phone_number1, які забезпечують взаємодію з користувачем;

− start.py відповідає за стартову команду бота та перевірку CAPTCHA. Імпортуються функції start_command1 та process_recaptcha_verification1;

− admin_function.py містить функції для адміністративних операцій, таких як управління статусами користувачів, список користувачів, розсилки та відключення затримки. Імпортуються функції manage_status1, toggle_user_status1, list_users1, start_broadcast_info1, start_broadcast_info2, disable_delay1 та user_delay_disabled.

```
from config import API_TOKEN, admin_id, MONGO_URI, DB_NAME, users_collection
from check import check_user_status, check_admin_access, is_user_banned, is_admin
from add_users import add_user
from keyboards import user_menu, admin_menu, back_menu
from users_function import about_us1, contact_admin1, back_to_menu1, request_phone_number1, handle_phone_number1
from start import start_command1, process_recaptcha_verification1
from admin_function import manage_status1, toggle_user_status1, list_users1, start_broadcast_info1, start_broadcast_info2, disable_delay1, user_delay_disabled
```

Рисунок 4.6 − Бібліотеки додаткові

Далі ми налаштовуємо (рисунок 4.7) логування за допомогою logging.basicConfig(level=logging.INFO), що дозволяє вести журнал подій для моніторингу роботи бота. Потім створюємо об’єкт бота з використанням API_TOKEN, який є необхідним для взаємодії з Telegram API. Для збереження даних про користувачів використовуємо MemoryStorage(), що дозволяє зберігати тимчасову інформацію в пам’яті під час сесії. Також ініціалізуємо диспетчер dp, який керує усіма подіями, що відбуваються в боті. Окрім того, визначено клас Feedback, який містить стан awaiting_reply, що використовується для обробки зворотного зв’язку від користувача.

```
20  logging.basicConfig(level=logging.INFO) # Налаштування бота та диспетчера
21  bot = Bot(token=API_TOKEN)
22  storage = MemoryStorage()
23  dp = Dispatcher(bot, storage=MemoryStorage())
24
25  class Feedback(StatesGroup): # Стан для зворотного зв’язку
26      awaiting_reply = State()
27
```

Рисунок 4.7 − Логування, бот, диспетчер

На наступному фрагменті коду (рисунок 4.8) визначено кілька обробників повідомлень та запитів для бота на платформі Telegram, що реалізовані через Aiogram.

Перший обробник @dp.message_handler(commands=['start']) викликається при введенні команди /start користувачем. Він передає управління функції start_command1, що ініціює стартові налаштування бота, перевірку на бота або та блокування користувача.

Другий обробник @dp.callback_query_handler(lambda c: c.data.startswith('verify_')) реагує на callback-запити, що починаються з verify_. Якщо перевірка пройдена успішно, бот надсилає користувачу повідомлення про підтвердження, а також інформує про подальші дії, надаючи меню для взаємодії.

Інші обробники відповідають за функції в адмін-панелі бота, такі як керування статусом користувача, перегляд списку користувачів, розсилання повідомлень та затримку повідомлень.

```python
@dp.message_handler(commands=['start'])
async def start_command(message: types.Message):
    await start_command1(message)

@dp.callback_query_handler(lambda c: c.data.startswith('verify_'))
async def process_recaptcha_verification(callback_query: types.CallbackQuery):
    await process_recaptcha_verification1(callback_query)
    user_id = int(callback_query.data.split('_')[1])
    await bot.send_message(user_id, "Ви успішно підтвердили, що не є роботом!")
    await bot.send_message(user_id, "Ласкаво просимо! Ви можете поставити своє запитанн

@dp.message_handler(content_types=['text'], text='🚫 Керування статусом') # Обробник к
async def manage_status(message: types.Message):
    await manage_status1(message)

@dp.message_handler(lambda message: message.text.isdigit(), state='*') # Обробник для к
async def toggle_user_status(message: types.Message):
    await toggle_user_status1(message)

@dp.message_handler(content_types=['text'], text='👥 Список користувачів') # Обробник
async def list_users(message: types.Message):
    await list_users1(message)

@dp.message_handler(content_types=['text'], text='✉️ Розсилання') # Обробник кнопки дл
async def start_broadcast_info(message: types.Message):
    await start_broadcast_info1(message)

@dp.message_handler(commands=['розсилання'])
async def start_broadcast(message: types.Message):
```

Рисунок 4.8 − Деякі обробники адмін-панелі та старту

Після розсилки згідно коду бот видає статистику по користувачам які прийняли повідомлення . Розсилка робиться з невеликою затримкою (рисунок 4.9 − 4.10). Реалізована функція start_broadcast, яка відповідає за запуск процесу розсилки повідомлень всім активним користувачам бота. Коли адміністратор відправляє команду /розсилання, бот перевіряє, чи є користувач адміністратором та чи не заблокований він. Потім отримує текст повідомлення для розсилки, і якщо він не вказаний, бот попросить ввести текст.

Далі, бот отримує список всіх активних користувачів (які не заблоковані), та по черзі відправляє їм персоналізовані повідомлення з відповідними іменами користувачів. Після кожного відправленого повідомлення додається затримка в 1 секунду за допомогою await asyncio.sleep(1), що дозволяє уникнути

перевантаження сервера і забезпечує плавну роботу бота. Після завершення розсилки адміністратор отримує статистику про кількість успішно надісланих повідомлень та тих, що не вдалося відправити.

```python
broadcast_text = message.get_args() # Отримуємо текст розсилки
if not broadcast_text:
    await message.answer(
        "Будь ласка, вкажіть текст для розсилання після команди.\n"
        "Приклад: /розсилання Привіт, {ім'я}! Ми маємо для вас чудову новину!",
        parse_mode='Markdown',
    )
    return

users = await users_collection.find({"is_banned": False}).to_list(length=None) # Отримуємо список активних користувачів
success = 0     # Ініціалізуємо лічильники
failed = 0

for user in users:     # Розсилка повідомлень
    user_id = user['user_id']
    user_name = user.get("full_name", "користувач")
    personalized_text = broadcast_text.replace("{ім'я}", user_name)

    try:
        await bot.send_message(user_id, personalized_text)
        success += 1
    except Exception as e:
        logging.error(f"Не вдалося відправити повідомлення користувачу {user_id}: {e}")
        failed += 1

    await asyncio.sleep(1)  # Затримка між повідомленнями

await message.answer(     # Відправка статистики адміністратору
    f"Розсилка завершена.\n"
    f"Успішно: {success}\n"
    f"Не вдалося: {failed}"
)
```

Рисунок 4.9 − Фрагмент коду розсилки

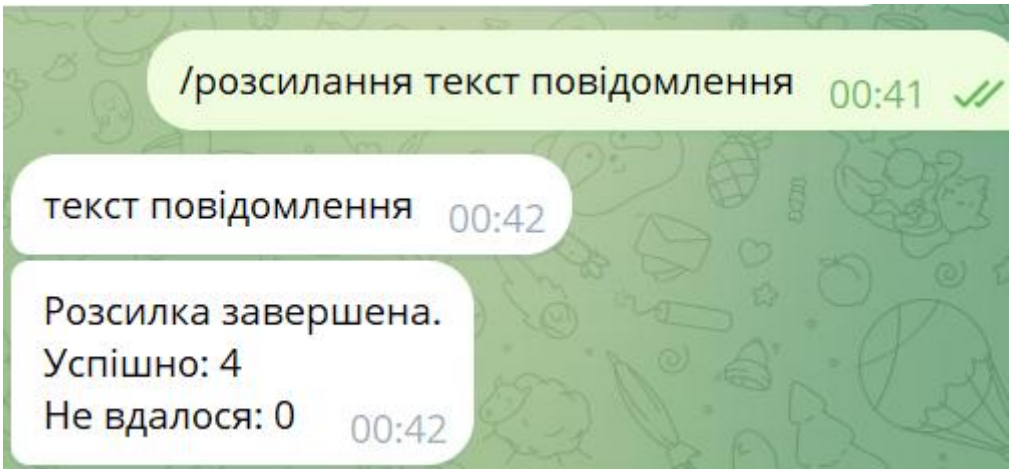

Рисунок 4.10 − Статистика після розсилки

Далі вказані обробники кнопок користувача в головному файлі (рисунок 4.11). Кожен обробник викликає відповідну функцію, залежно від тексту кнопки, яку натискає користувач. Наприклад, кнопка «Про нас» викликає функцію about_us1 для надання інформації, кнопка «Зв’язатися з підтримкою» запускає contact_admin1 для звернення до адміністрації, а кнопка «Передзвоніть мені» активує функцію request_phone_number1, щоб запросити номер телефону для зворотного зв’язку.

```
@dp.message_handler(content_types=['text'], text='🐠 Про нас') # Обробник для текстового повідомлення "🐠 Про нас"
async def about_us(message: Message):
    await about_us1(message)

@dp.message_handler(content_types=['text'], text='⌨ Зв’язатися з підтримкою') # Обробник для текстового повідомлення "⌨ Зв’язатися з підтримко
async def contact_admin(message: types.Message):
    await contact_admin1(message)

@dp.message_handler(content_types=['text'], text='🔙 Назад') # Обробник для текстового повідомлення "🔙 Назад"
async def back_to_menu(message: types.Message):
    await back_to_menu1(message)

@dp.message_handler(content_types=['text'], text='📞 Передзвоніть мені') # Обробник для текстового повідомлення "📞 Передзвоніть мені"
async def request_phone_number(message: types.Message):
    await request_phone_number1(message)

@dp.message_handler(lambda message: message.text and message.text.startswith("+"), content_types=['text']) # Обробник для отримання номера телефо
async def handle_phone_number(message: types.Message):
    await handle_phone_number1(message)         # Отримуємо введений номер телефону
    phone_number = message.text.strip().replace(" ", "")
```

Рисунок 4.11 − Обробники кнопок користувача

Наступний фрагмент коду приклад з боку обробки різних типів повідомлення від користувача, а саме на цьому фото це фото, відео (рисунок 4.12) Коли користувач надсилає фото, бот отримує останнє фото з повідомлення та надсилає його адміністраторам, додаючи інформацію про користувача, таку як ім’я та ID. Також додається кнопка «Відповісти» для зручності відповіді на це фото.

Аналогічно, коли користувач надсилає відео, воно обробляється аналогічним чином: отримується файл відео, і адміністраторам надсилається відео з описом користувача та кнопкою для відповіді. Цей процес дозволяє адміністраторам отримувати мультимедійні повідомлення та взаємодіяти з користувачами через бота.

```
elif message.photo: # Обробляємо фото
    photo = message.photo[-1].file_id
    for admin in admin_id:
        await bot.send_photo(
            admin,
            photo,
            caption=f"Фото від користувача:\n<b>Ім'я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
            parse_mode="HTML",
            reply_markup=types.InlineKeyboardMarkup().add(
                types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
        )
    )

elif message.video: # Обробляємо відео
    video = message.video.file_id
    for admin in admin_id:
        await bot.send_video(
            admin,
            video,
            caption=f"Відео від користувача:\n<b>Ім'я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
            parse_mode="HTML",
            reply_markup=types.InlineKeyboardMarkup().add(
                types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
            )
    )
```

Рисунок 4.12 − Обробка фото та відео користувача

На рисунку зображено фрагмент коду який згідно якого працює затримка для користувачів, як метод захисту від спаму (рисунок 4.13). Алгоритм працює таким чином: при отриманні нового повідомлення бот перевіряє, коли користувач надіслав своє останнє повідомлення. Якщо між повідомленнями менше ніж заданий інтервал (MESSAGE_TIMEOUT), перевіряється кількість повідомлень, які користувач надіслав за останні 30 секунд. Якщо користувач перевищує ліміт (MAX_MESSAGES), бот надсилає повідомлення з проханням почекати певну кількість секунд перед наступним відправленням.

Якщо між повідомленнями пройшло більше ніж 30 секунд, лічильник повідомлень скидається, і користувач може надсилати нові повідомлення без обмежень. Така затримка дозволяє уникнути надмірного спаму та забезпечити кращу взаємодію з користувачами.

```
current_time = time.time() # Отримуємо поточний час
if user_id in last_message_time: # Перевірка під час останнього повідомлення
    time_diff = current_time - last_message_time[user_id]
    if time_diff < MESSAGE_TIMEOUT:
        if user_id in message_count and message_count[user_id] >= MAX_MESSAGES: # Перевіряємо, скільки повідомлень надіслав користувач за останні
            await message.answer(f"Ви надто часто відправляєте повідомлення. Будь ласка, почекайте {MESSAGE_TIMEOUT - int(time_diff)} секунд.")
            return
        else:
            message_count[user_id] += 1 # Якщо повідомлень менше 2-х, збільшуємо лічильник
    else:
        message_count[user_id] = 1 # Якщо минуло більше 30 секунд, скидаємо лічильник повідомлень

else:
    message_count[user_id] = 1 # Якщо це перше повідомлення, просто додаємо

last_message_time[user_id] = current_time # Оновлюємо час останнього повідомлення
```

Рисунок 4.13 − Затримка користувачів

Далі в головному файлі ми обробляємо відповіді адміністратора до користувачів.

В файлі конфігурації (рисунок 4.14) ми використовуємо бібліотеку motor.motor_asyncio, яка є асинхронною обгорткою для роботи з MongoDB в Python, що дозволяє ефективно взаємодіяти з базою даних без блокування основного потоку виконання програми. Для підключення до бази даних використовуємо AsyncIOMotorClient, який створює асинхронне з’єднання з MongoDB. Далі визначаємо важливі конфігурації, такі як API_TOKEN для бота, admin_id для адміністратора, а також налаштування для підключення до MongoDB, де вказуємо MONGO_URI та ім’я бази даних DB_NAME. з’єднання з MongoDB створюється через mongo_client, і доступ до колекції користувачів отримується через users_collection для подальшої роботи з даними користувачів.

config.py > ...

```
1   import motor.motor_asyncio
2   import os
3   from motor.motor_asyncio import AsyncIOMotorClient
4
5   API_TOKEN = '7627528973:AAH3guWUGJ3jGAvfKlGmOOKEVven7EHaVvo'  # токен бота
6   admin_id = [1617533812]      # ID адміністратора
7   MONGO_URI = "mongodb://127.0.0.1:27017"  # MongoDB
8   DB_NAME = 'users'
9
10  mongo_client = AsyncIOMotorClient(MONGO_URI)
11  db = mongo_client[DB_NAME]
12  users_collection = db['users']
```

Рисунок 4.14 − Файл конфігурації

В стартовому коду ми робимо перевірки користувача та додавання його до бази даних (рисунок 4.15). Спочатку перевіряється, чи є користувач у базі даних. Якщо користувач заблокований, йому надсилається повідомлення про блокування, і доступ до бота припиняється. Якщо користувач ще не підтвердив, що він не бот, йому відправляється запит з кнопкою для підтвердження.

Якщо користувач вже підтвердив, що він не робот, йому відображається стандартне меню. Якщо користувач є адміністратором, то замість звичайного меню він отримує доступ до адмін-панелі. Після того, як користувач натискає кнопку для підтвердження, його статус оновлюється в базі даних через функцію add_user, що позначає його як підтвердженого користувача.

```python
async def start_command1(message: types.Message):
    # Перевіряємо, чи є користувач у базі даних
    user = await users_collection.find_one({"user_id": message.from_user.id})

    # Якщо користувач заблокований, надсилаємо повідомлення і виймаємо
    if user and user.get("is_banned", False):
        await message.answer("Ви заблоковані і не можете користуватися ботом.")
        return

    # Якщо користувач не підтвердив, що він не робот
    if not user or not user.get("is_human", False):
        # Створюємо кнопку для підтвердження
        keyboard = InlineKeyboardMarkup().add(
            InlineKeyboardButton("Я не робот", callback_data=f"verify_{message.from_user.id}")
        )

        # Запитуємо підтвердження
        await message.answer(
            "Щоб продовжити, натисніть кнопку, щоб підтвердити, що ви не бот.",
            reply_markup=keyboard
        )
    else:
        # Якщо користувач вже підтвердив, що не робот, показуємо стандартне меню
        # Вітання користувача
        if message.from_user.id in admin_id:
            await message.answer("Ласкаво просимо, адміністратор!", reply_markup=admin_menu)
        else:
            await message.answer("Ласкаво просимо! Ви можете поставити своє запитання або пропозицію ✍", reply_markup=user_menu)

async def process_recaptcha_verification1(callback_query: types.CallbackQuery):
    user_id = int(callback_query.data.split('_')[1])
    # Оновлюємо статус користувача в базі даних
    await add_user(user_id, callback_query.from_user.full_name, is_human=True)
```

Рисунок 4.15 − Стартовий код

В admin_function як бачимо обробляються майже усі функції адмін-панелі (рисунок 4.16).

```python
from aiogram import types
from config import users_collection  # Імпортуємо необхідні змінні
from keyboards import user_menu, back_menu, admin_menu
from check import check_admin_access, check_user_status, admin_id
from add_users import add_user

async def manage_status1(message: types.Message):

    user_id = message.from_user.id
    if not await check_admin_access(message):  # Перевірка прав адміністратора
        return
    if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
        return

    await message.answer("Введіть ID користувача для керування статусом блокування або розблокування:", re

async def toggle_user_status1(message: types.Message):

    user_id = message.from_user.id
    if not await check_admin_access(message):  # Перевірка прав адміністратора
        return
    if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
```

Рисунок 4.16 − Функції адмін-панелі

Функція manage_status1 дозволяє адміністратору керувати статусом користувача (блокуванням чи розблокуванням). Спочатку перевіряється, чи має користувач права адміністратора та чи не заблокований він. Якщо перевірки пройдено успішно, адміністратору відправляється запит на введення ID користувача для зміни його статусу. Також додається кнопка для повернення до попереднього меню.

У функції toggle_user_status1 адміністратор має можливість змінювати статус користувача, блокуючи або розблоковуючи його. Спочатку перевіряється, чи має користувач права адміністратора та чи не заблокований він. Потім, якщо користувач знайдений у базі даних, його статус (is_banned) змінюється: якщо він заблокований, він розблоковується, і навпаки. Після цього адміністратор отримує повідомлення про виконану операцію та відповідне меню для подальших дій.

Функція start_broadcast_info2 дозволяє адміністратору налаштовувати затримку для користувачів перед відправкою повідомлень. Адміністратор отримує інструкцію для введення ID користувача, для якого необхідно увімкнути або вимкнути затримку перед розсилкою повідомлень.

Функція disable_delay1 дозволяє адміністратору вимикати або вмикати затримку для конкретного користувача. Бот обробляє команду, що містить ID користувача, для якого потрібно змінити стан затримки. Якщо затримка вже вимкнена, вона буде увімкнена, і навпаки − якщо затримка увімкнена, вона буде вимкнена. У разі неправильного введення ID (не числовий формат), бот надає відповідне повідомлення про помилку.

В user_function як бачимо виконуються майже усі функції саме користувачів (рисунок 4.17). Функції реалізують основні взаємодії користувачів з ботом, такі як отримання інформації про компанію через about_us1, зв’язок з адміністратором через contact_admin1, повернення в головне меню через back_to_menu1, запит номера телефону через request_phone_number1, а також обробка введеного номера телефону через handle_phone_number1. Усі функції перевіряють статус користувача перед виконанням дій, що гарантує безпеку та правильність роботи бота.

```python
# users_function.py
from aiogram import types
from config import users_collection  # Імпортуємо необхідні змінні
from keyboards import user_menu, back_menu, admin_menu
from check import check_admin_access, check_user_status, admin_id

async def about_us1(message: types.Message):  # Функція для відправлення інформації про компанію
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return
    await message.answer("IT-компанії «Диверсіті IT» – це IT-компанія, яка спеціалізується на розробці

async def contact_admin1 (message: types.Message):  # Функція для зв'язку з адміністратором
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return
    await message.answer("Напишіть ваше повідомлення для адміністратора:", reply_markup=back_menu)

async def back_to_menu1(message: types.Message):  # Функція для повернення до головного меню
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return

    if message.from_user.id in admin_id:
        await message.answer("Ви повернулися в головне меню адміністратора.", reply_markup=admin_menu)
    else:
        await message.answer("Ви повернулися в головне меню користувача.", reply_markup=user_menu)

async def request_phone_number1(message: types.Message):  # Функція для запиту номера телефону
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return

    # Просимо користувача ввести номер телефону
    await message.answer("Будь ласка, введіть ваш номер телефону в форматі +380XXXXXXXXX:")

async def handle_phone_number1(message: types.Message):  # Функція для обробки введеного номера телефон
```

Рисунок 4.17 − Функції користувачів

По-перше, файл check.py зберігає код перевірки на бота та статус заблокування користувача (рисунок 4.18).

```python
async def is_user_banned(user_id):
    # Перевірка, чи заблокований користувач
    user = await users_collection.find_one({"user_id": user_id})
    return user['is_banned'] if user and 'is_banned' in user else False

async def check_user_status(user_id, message, users_collection):
    # Перевірка, чи заблокований користувач
    if await is_user_banned(user_id):
        await message.answer("Ви заблоковані і не можете користуватися ботом.")
        return False
    else:
    # Перевірка, чи існує користувач і чи підтвердив він, що не є ботом
        user = await users_collection.find_one({"user_id": user_id})
        if not user or not user.get("is_human", False):
            # Створення кнопки для підтвердження
            keyboard = InlineKeyboardMarkup().add(
                InlineKeyboardButton("Я не робот", callback_data=f"verify_{message.from_user.id}")
            )

            # Запит на підтвердження
            await message.answer(
                "Щоб продовжити, натисніть кнопку, щоб підтвердити, що ви не бот.",
                reply_markup=keyboard
            )
            return False
    return True
```

Рисунок 4.18 − Файл check.py

Функція is_user_banned перевіряє, чи заблокований користувач, повертаючи відповідне значення з бази даних. check_user_status перевіряє статус користувача, зокрема чи він заблокований або не підтвердив, що він не є ботом, та надає кнопку для підтвердження.

По-друге зберігає код перевірки на адміністратора (рисунок 4.19)

```python
async def is_admin(user_id: int) -> bool:
    return user_id in admin_id

async def check_admin_access(message):
    # Перевірка, чи є користувач адміністратором
    if not await is_admin(message.from_user.id):
        await message.answer("Ви не маєте доступу до адміністративної панелі.")
        return False
    return True
```

Рисунок 4.19 − Перевірка на адміністратора

Функція is_admin перевіряє, чи є користувач адміністратором, а check_admin_access забезпечує доступ до адміністративної панелі, перевіряючи права користувача. Ці перевірки допомагають керувати доступом до бота і запобігають небажаним діям користувачів або ботів.

У файлі add_users.py як бачимо зберігається структура інформації, яка збирається з кожного користувача (рисунок 4.20).

```
add_users.py > ...
1   from config import users_collection  # Імпортуємо колекцію користувачів з к
2
3   async def add_user(user_id, full_name, is_banned=False, is_human=False):
4       # Додаємо користувача до бази даних або оновлюємо його дані, якщо такий
5       user_data = {
6           "user_id": user_id,
7           "full_name": full_name,
8           "is_banned": is_banned,
9           "is_human": is_human
10      }
11      await users_collection.update_one(
12          {"user_id": user_id},
13          {"$set": user_data},
14          upsert=True
15      )
16
```

Рисунок 4.20 − Структура інформації про користувача

Функція add_user додає або оновлює інформацію про користувача в базі даних. Вона приймає параметри user_id, full_name, is_banned та is_human для збереження даних користувача. Якщо користувач з таким user_id вже існує в базі, то його дані оновлюються, якщо ні − додається новий запис. Оновлення або додавання відбувається за допомогою методу update_one, який використовує умову пошуку за user_id та оновлює або додає необхідні поля, якщо їх немає.

Також вже збережені дані можна переглянути через Mongo DB Compass (рисунок 4.21) або MongoDB Shell.

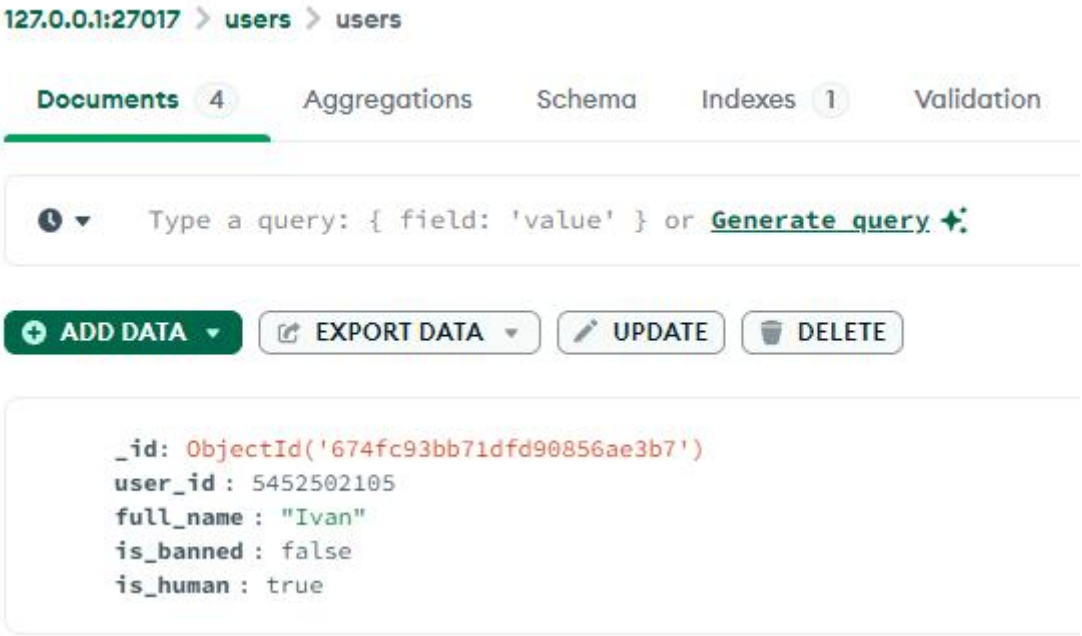

Рисунок 4.21 – Користувач з бази даних

В останньому файлі зберігаються кнопки користувача та адмін-панелі (рисунок 4.22).

keyboards.py > ...

```python
from aiogram import types

# Створення кнопок меню для користувача та адміністратора
user_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)  # Меню для користувача
user_menu.add(types.KeyboardButton('👋 Про нас'))  # Кнопка "Про нас"
user_menu.add(types.KeyboardButton('📩 Зв’язатися з підтримкою'))  # Кнопка "Зв’язатися з підтримкою"
user_menu.add(types.KeyboardButton('📞 Передзвоніть мені'))  # Кнопка "Передзвоніть мені"

admin_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)  # Меню для адміністратора
admin_menu.add(types.KeyboardButton('👥 Список користувачів'))  # Кнопка "Список користувачів"
admin_menu.add(types.KeyboardButton('🚫 Керування статусом'))  # Кнопка "Керування статусом"
admin_menu.add(types.KeyboardButton('📩 Розсилання'))  # Кнопка "Розсилання"
admin_menu.add(types.KeyboardButton('⏳ Затримка повідомлень'))  # Кнопка "Затримка повідомлень"

back_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)  # Меню для повернення
back_menu.add(types.KeyboardButton("🔙 Назад"))  # Кнопка "Назад"
```

Рисунок 4.22 – Усі кнопки боту

Меню користувача включає три кнопки: «Про нас», «Зв’язатися з підтримкою» та «Передзвоніть мені», що дозволяють користувачеві отримати інформацію, звернутися до адміністратора чи попросити передзвонити. Меню

адміністратора має чотири кнопки: «Список користувачів», «Керування статусом», «Розсилання» та «Затримка повідомлень», що дозволяють адміністратору здійснювати управлінські функції, такі як перегляд користувачів, керування їх статусами, розсилання повідомлень та налаштування затримки для повідомлень. Крім того, є кнопка «Назад», яка дозволяє повернутися до попереднього меню.

4.3.4 Використані технічні засоби

Для розробки програми використовувався Windows 10 як операційна система. Windows 10 є стабільною платформою для виконання Python-скриптів, а також для роботи з різними середовищами розробки.

Для написання коду та розробки бота використовувався Microsoft Visual Studio, що забезпечує підтримку Python через відповідні плагіни, що дає змогу ефективно писати, тестувати та налагоджувати програму.

Для роботи з базою даних MongoDB було використано MongoDB Compass, який є офіційний графічний інтерфейс для роботи з MongoDB. Compass надає зручний інтерфейс для перегляду і редагування даних, виконання запитів, аналізу структури бази даних і моніторингу її стану.

4.3.5 Виклик і завантаження

Telegram-бот зворотного зв’язку запускається на сервері за допомогою файлу або вручну через командний інтерфейс. Запуск виконується шляхом виконання скрипта Python, із зазначенням середовища виконання. Бот розгорнутий на сервері із постійним підключенням до мережі Інтернет для забезпечення безперервної роботи.

Головною точкою входу є файл bot.py, який містить ініціалізацію програмного забезпечення та основний цикл взаємодії з Telegram API. Після запуску цей файл:

− підключається до Telegram API за допомогою токена, виданого BotFather;

− реєструє хендлери для обробки вхідних повідомлень і команд користувачів;

− ініціалізує підключення до бази даних (MongoDB);

− створює логи роботи бота для моніторингу та діагностики.

Програма розташована на локальному накопичувачі сервера в папці /home/telegram-bot/. Вона завантажується в оперативну пам’ять при запуску. Програма потребує близько 50−200 МБ оперативної пам’яті залежно від кількості одночасних з’єднань і обсягу оброблюваних запитів.

Розмір вихідних файлів Telegram-бота становить приблизно 10 МБ, включаючи скрипти Python, конфігураційні файли та залежності (завантажені бібліотеки). Додатково, для роботи бази даних MongoDB потрібно близько 200 МБ дискового простору, який масштабуватиметься залежно від кількості користувачів та обсягу збережених даних.

4.3.6 Вхідні та вихідні дані

Вхідні дані − це повідомлення користувачів, які надходять у форматі, визначеному Telegram API. Вони фільтруються за типом (текст, медіа-файли).

Вихідні дані включають текстові повідомлення-відповіді, кнопки або меню Telegram Inline Keyboard, що надають користувачам зручний інтерфейс. Цей підхід обраний через його сумісність із платформою Telegram, а також можливість швидкої модифікації структури даних без зупинки роботи бота.

4.4 Висновки до розділу

Розділ «Розробка програмного забезпечення» описує область застосування ПЗ.

Далі в розділі було описано логічну структуру з приведенням фрагментів алгоритмів телеграм боту зворотного зв’язку та прикладами коду самого боту.

Було визначено функціональне призначення, описано загальні відомості, технічні засоби, вказано вхідні та вихідні дані.

5 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

5.1 Мета і завдання експерименту

Цей етап досліджень орієнтований на виявлення проблем і помилок, які можуть впливати на взаємодію користувачів з ботом.

Основна мета − перевірити, чи правильно бот виконує свої функції для кінцевих користувачів.

Перевірити, чи відповідає клієнтський функціонал технічним вимогам і чи всі функції працюють відповідно до специфікації.

Методи перевірки клієнтського функціоналу:

− виконання типових сценаріїв використання бота;

− перевірка всіх доступних команд, кнопок, меню.

Ціль перевірки клієнтського функціоналу: Виявити помилки у роботі бота, такі як некоректні відповіді, зависання, пропуск повідомлень або незрозуміле повідомлення користувачу.

Наступне тестування адміністративного функціоналу.

Цей етап спрямований на перевірку функцій, які доступні адміністраторам або модераторам бота. Мета − виявити потенційні проблеми у керуванні ботом, а також забезпечити правильність адміністрування та моніторингу користувацької активності.

Перше − це тестування прав доступу.

Мета тестування: перевірити, чи правильно обмежені функції доступу для адміністратора і користувачів.

Методи перевірки прав доступу:

− перевірка ролей і прав доступу адміністратора, модератора та звичайного користувача;

− тестування можливості несанкційованого доступу до адміністративних функцій.

Ціль перевірки прав доступу: переконатися, що адміністративний функціонал не дозволяє обходити обмеження або викрадати дані.

Аудит прав доступу: перевірка, чи можуть користувачі з несанкційованими правами виконати дії, доступні лише адміністраторам.

Таким чином, перевірка функціоналу бота з метою виявлення багів та обходу функціоналу є комплексним процесом, що включає тестування як клієнтської, так і адміністративної частини з різних аспектів.

5.2 Опис умов експерименту

5.2.1 Тестове середовище

Тестове середовище для перевірки Telegram-боту є:

– Telegram-бот працював використовуючи API Telegram;

– для взаємодії з базою даних буде застосовано графічний інтерфейс Mongo DB Compass.

5.2.2 Інструменти та методи тестування

Для телеграм бота тестування проводиться вручну шляхом надсилання текстових команд та отримання відповіді на них та аналіз змін в базі даних.

5.2.3 Тестові сценарії

В якості тестових сценаріїв буде перевірятися весь функціонал клієнтської та адміністративної частини, включали перевірку несанкційованого доступу та перегляд бази даних.

5.3 Хід експерименту

5.3.1 Перевірка функцій користувача

Першим кроком виконаємо запуск боту та як бачимо запускається перевірка на бота одразу (рисунок 5.1).

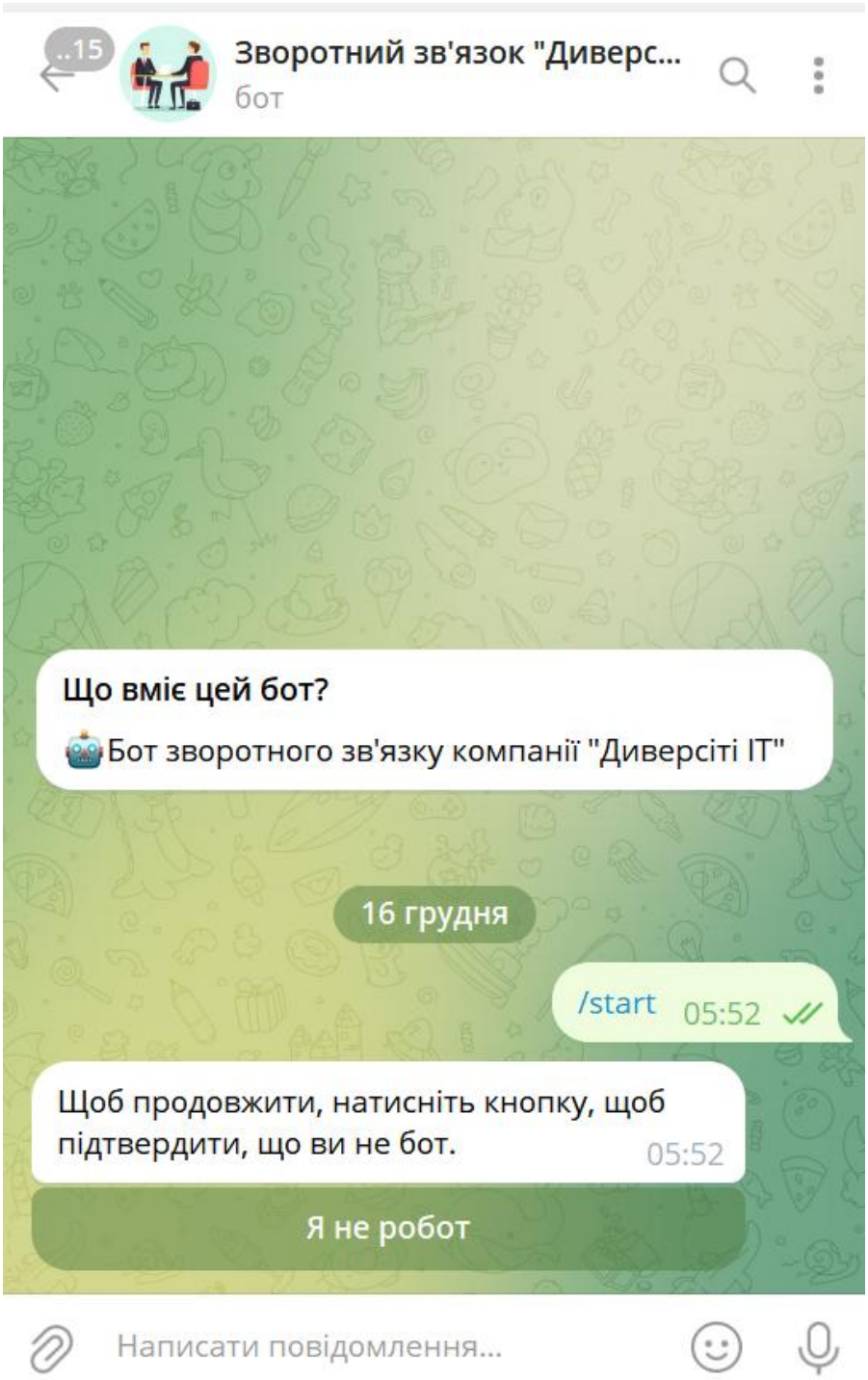

Рисунок 5.1 − Перевірка на бота

Після натискання на «я не робот» програма нас допустить до головного меню та внесе в базу даних інформацію про нас (рисунок 5.2 − 5.3).

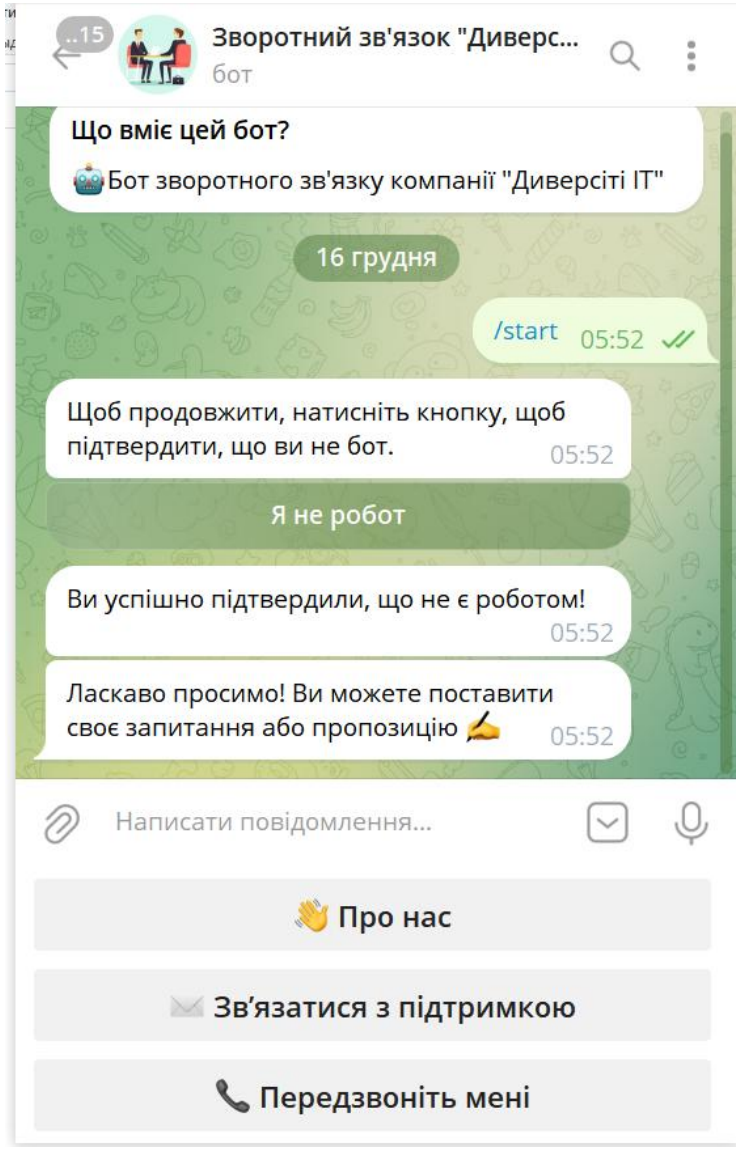

Рисунок 5.2 – Підтвердження, що користувач не робот

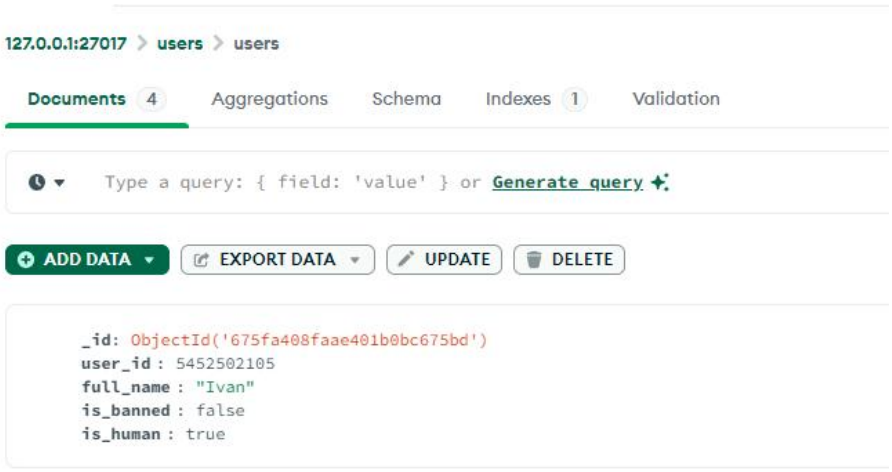

Рисунок 5.3 – Запис про користувача в базі даних

Тепер перевіримо, чи зможемо ми відправити повідомлення не пройшовши перевірку. Як бачимо, бот не дає користуватися функціоналом без перевірки (рисунок 5.4).

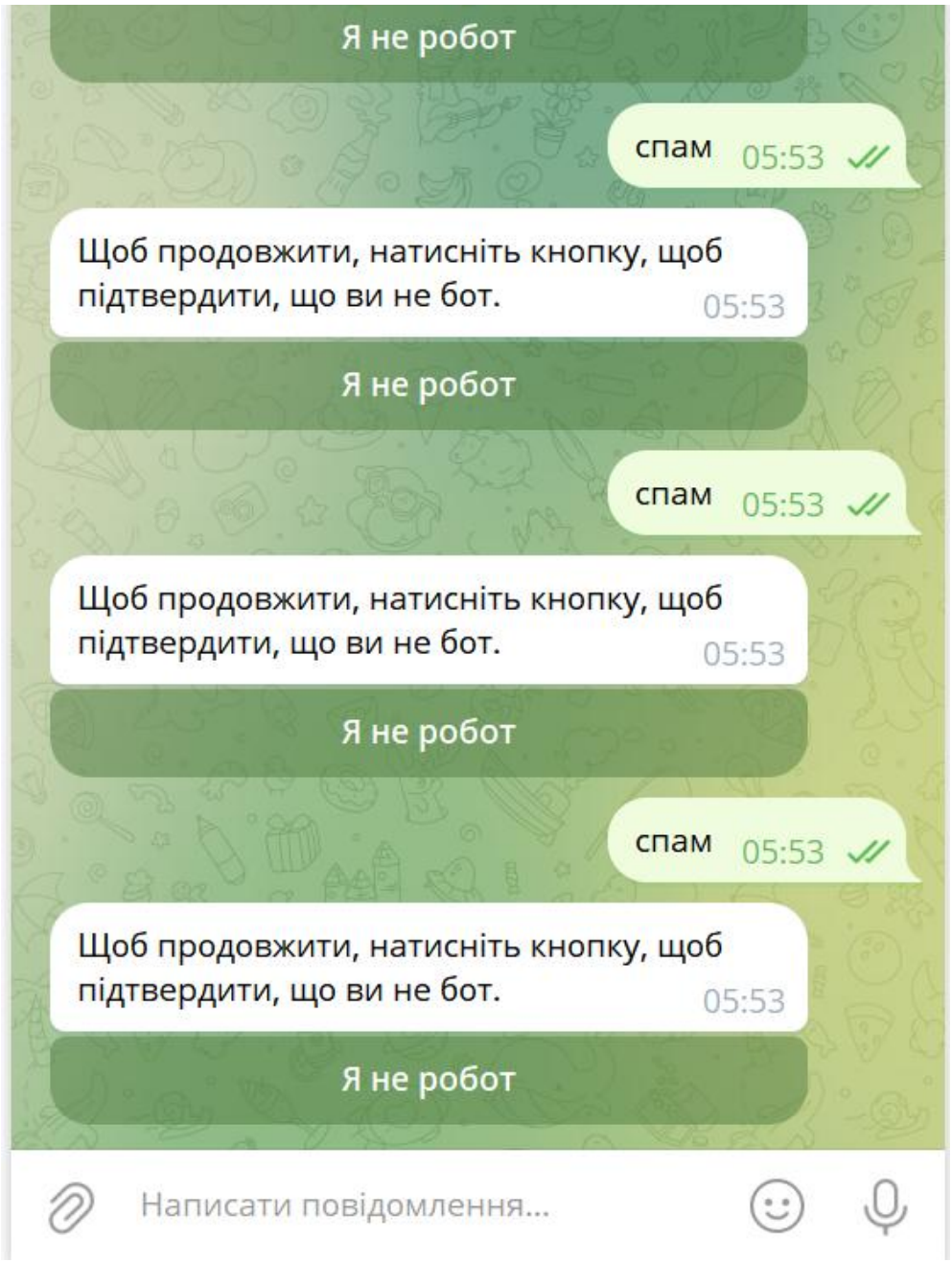

Рисунок 5.4 − Спам не пройшов перевірку на бота

Наприклад, ми видалимо бота з чатів (рисунок 5.5) та зайдемо повторно, то бот вже не бути повторно запрошувати підтвердження (рисунок 5.6).

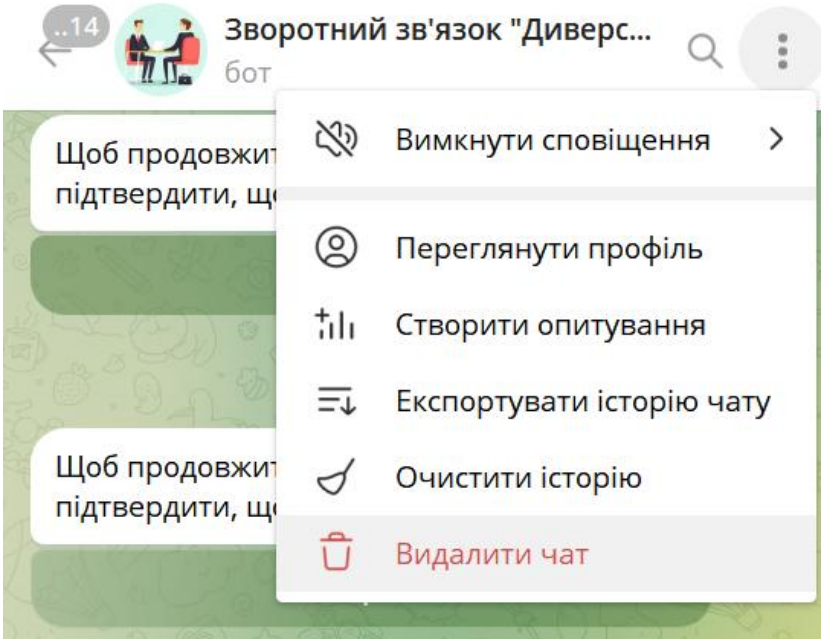

Рисунок 5.5 − Видалення чату з ботом

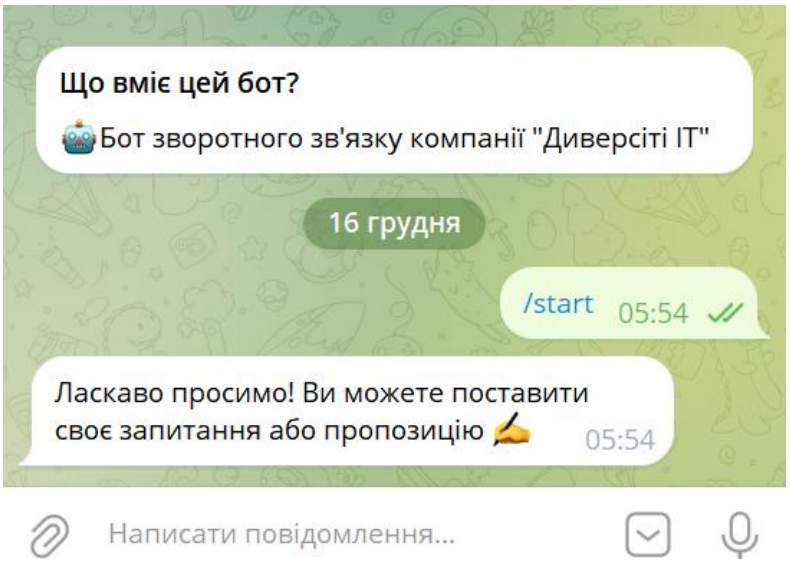

Рисунок 5.6 − Повторний запуск боту

Перевірка кнопки «Про нас» виводить інформацію про компанію (рисунок 5.7).

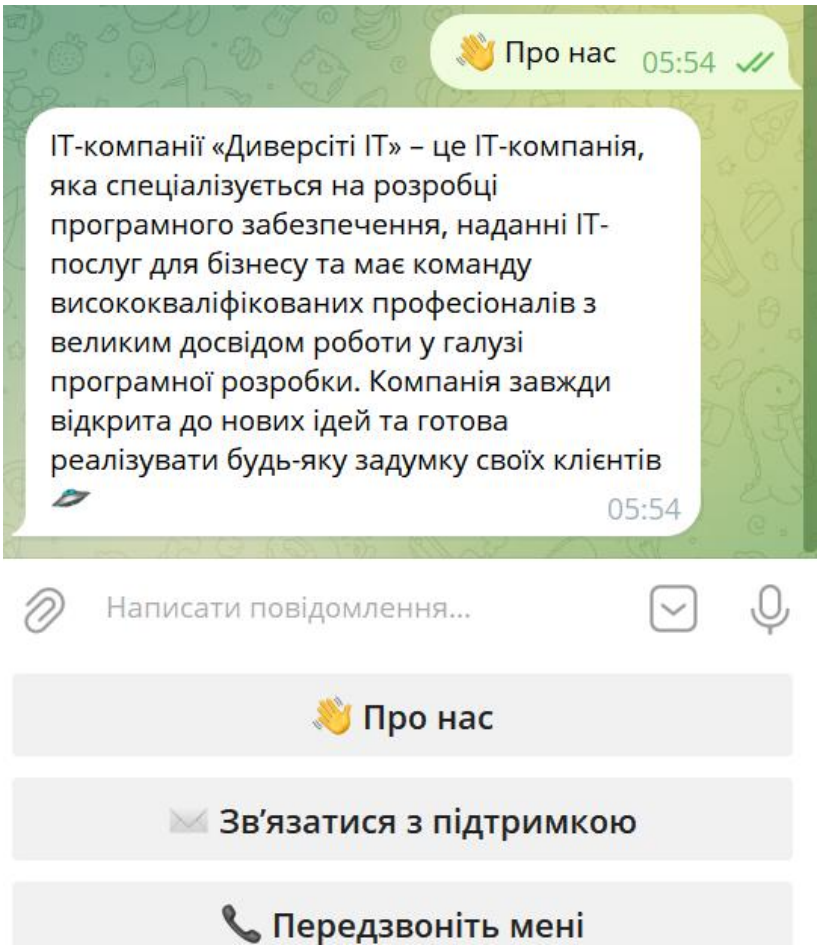

Рисунок 5.7 − Первірка кнопки «Про нас»

Після натискання кнопки «Зв’язатися з підтримкою» бот запропонує нам ввести повідомлення для відправлення (рисунок 5.8).

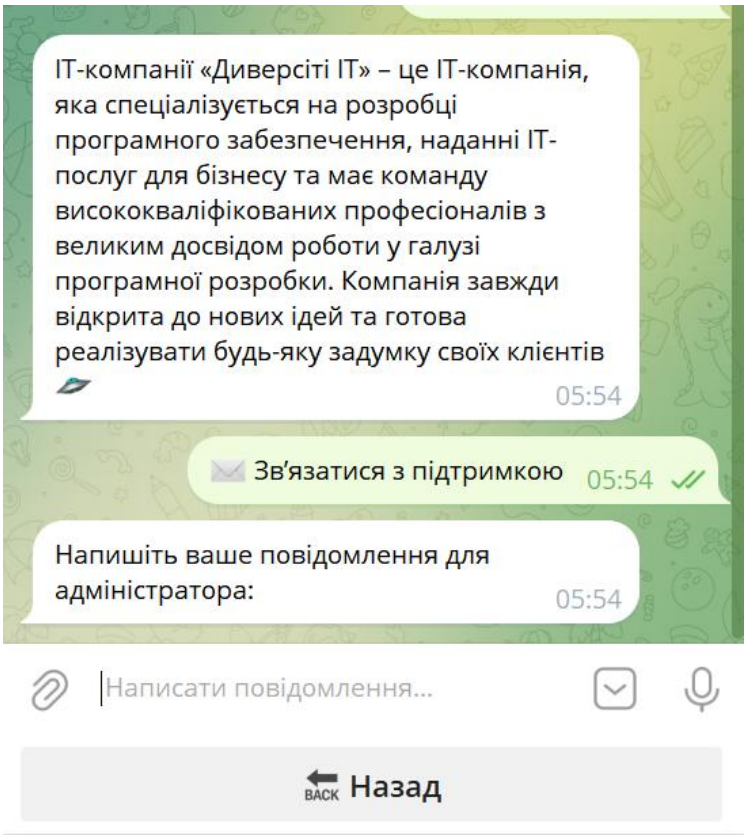

Рисунок 5.8 − Первірка кнопки «Зв'язатися з підтримкою»

Як бачимо, наші текстові повідомлення відсилаються до адміністрації (рисунок 5.9 та рисунок 5.10).

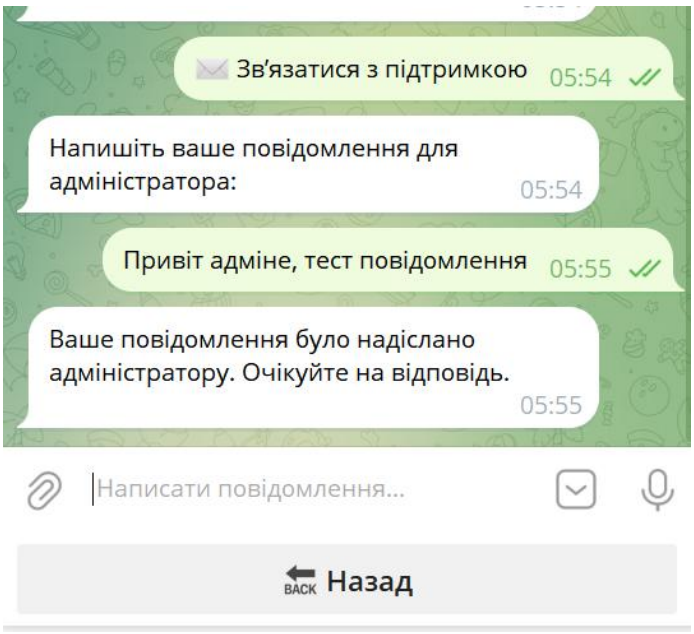

Рисунок 5.9 − Відправлення текстового повідомлення

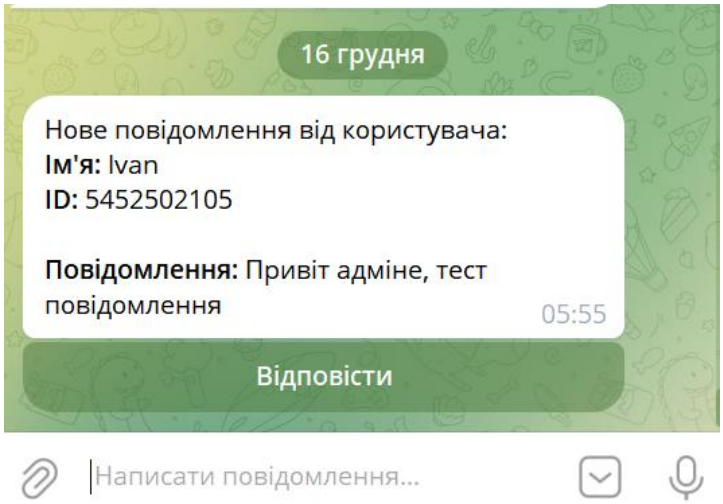

Рисунок 5.10 − Перевірка надходження до адміністрації

Перевіримо функціонал кнопки «Передзвоніть мені» (рисунок 5.11 та рисунок 5.12).

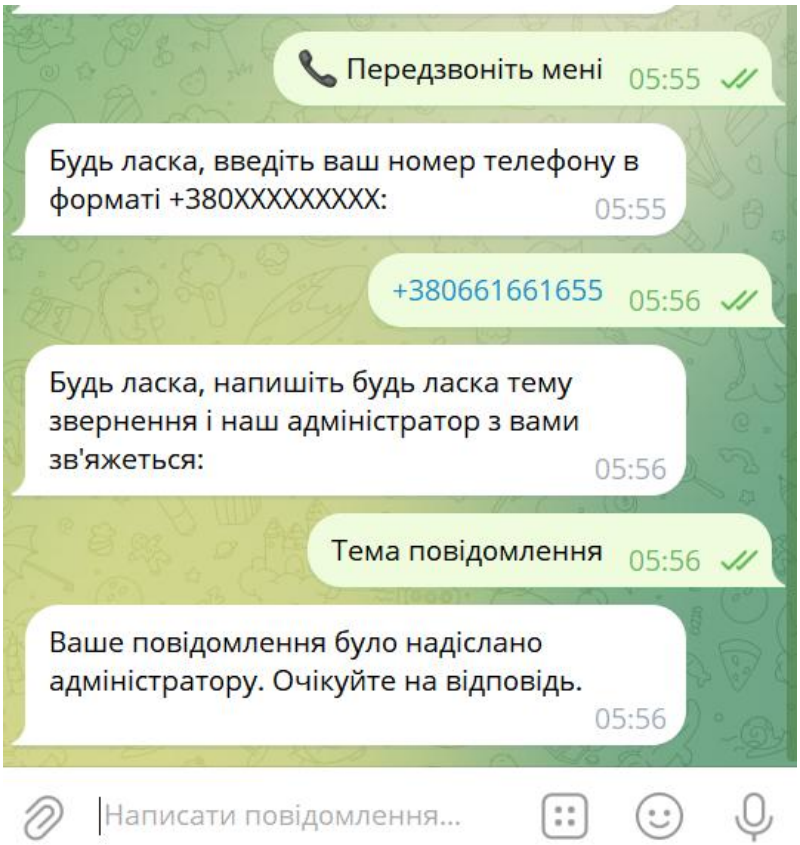

Рисунок 5.11 − Перевірка функцій кнопки «Передзвоніть мені»

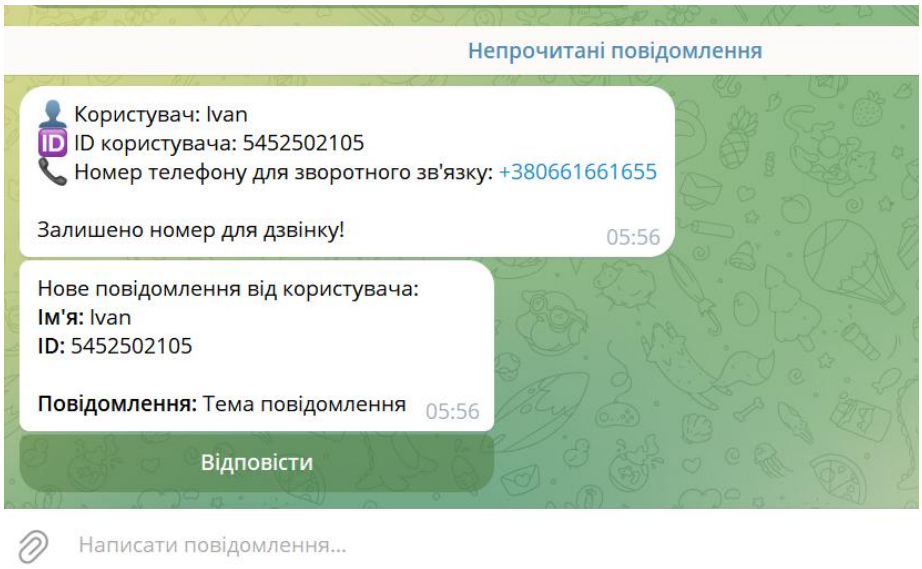

Рисунок 5.12 − Перевірка надходження інформації про зворотній дзвінок

Перевіримо, як працює затримка для користувачів (рисунок 5.13).

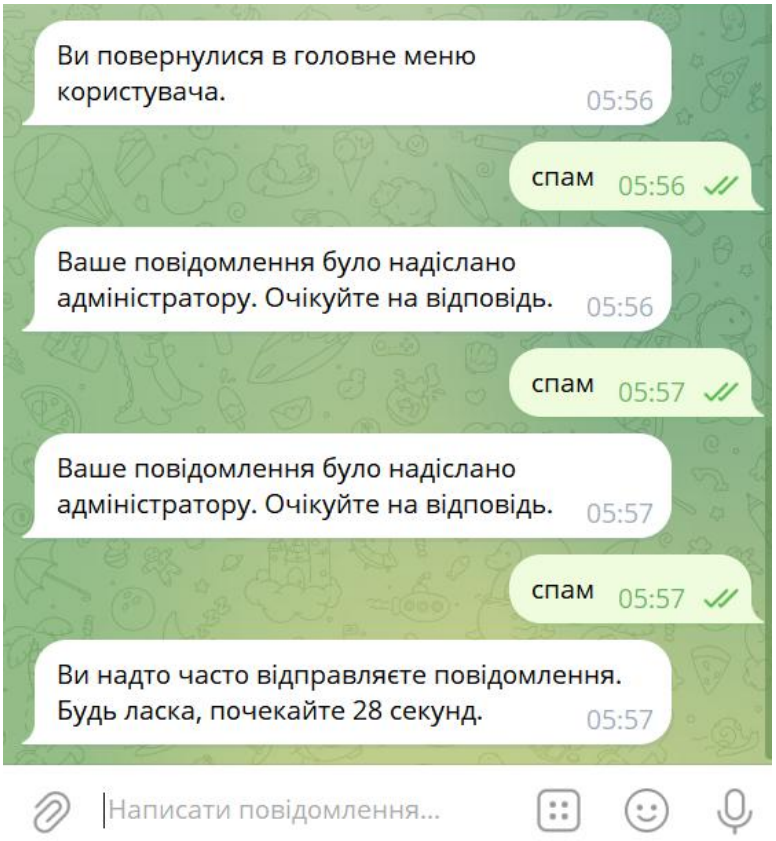

Рисунок 5.13 − Затримка користувачів

Тепер перевіримо, чи всі заявлені типи файлів підтримуються в боті зворотного зв’язку для відправлення їх адміністрації та чи всі надходять до адміністрації (рисунки 5.14 − 5.18).

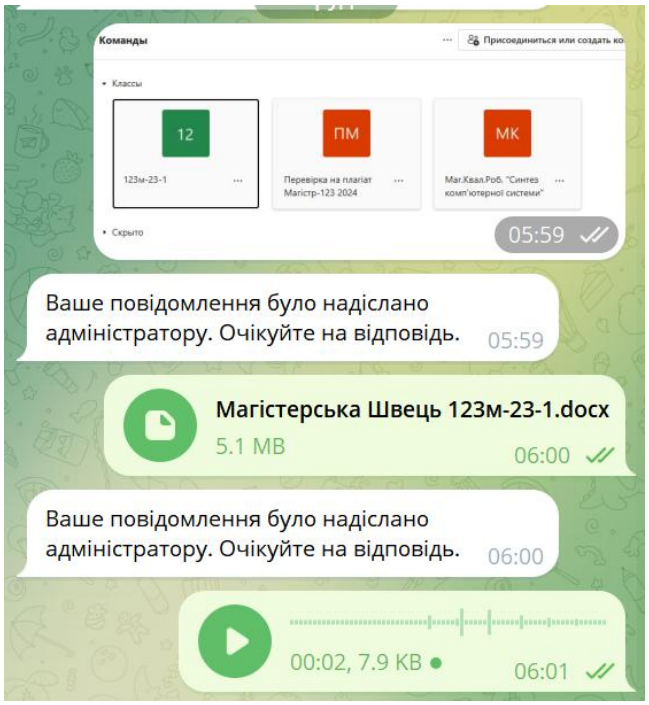

Рисунок 5.14 − Перевірка надсилання фото, файлу, голосового повідомлення

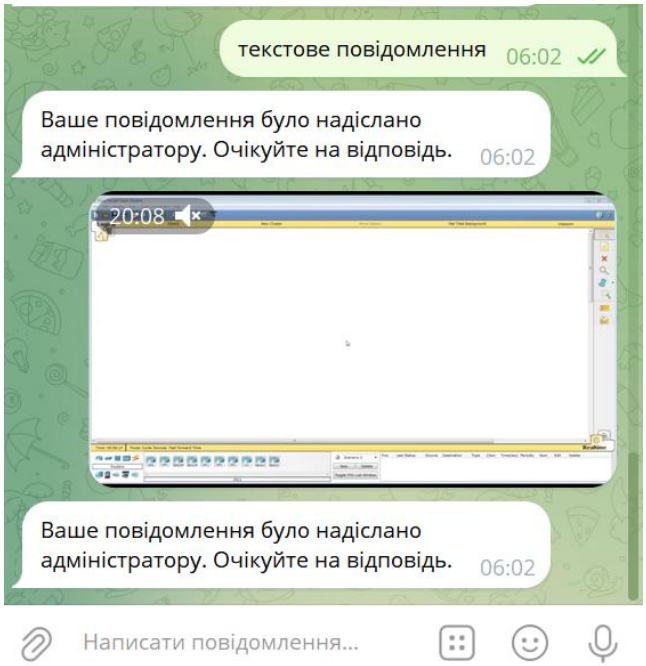

Рисунок 5.15 − Перевірка надсилання відео, текстового повідомлення

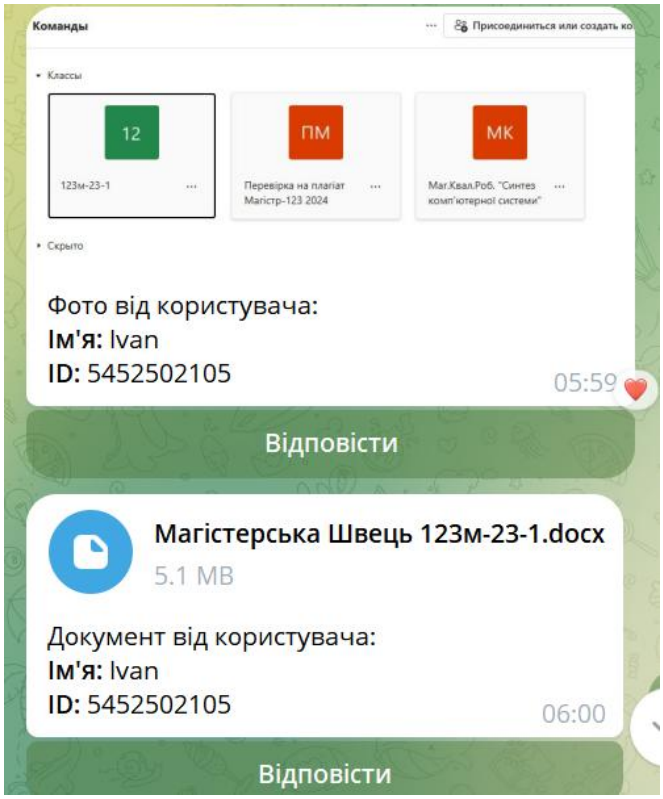

Рисунок 5.16 − Перевірка надходження до адміністрації документа, фото від користувача

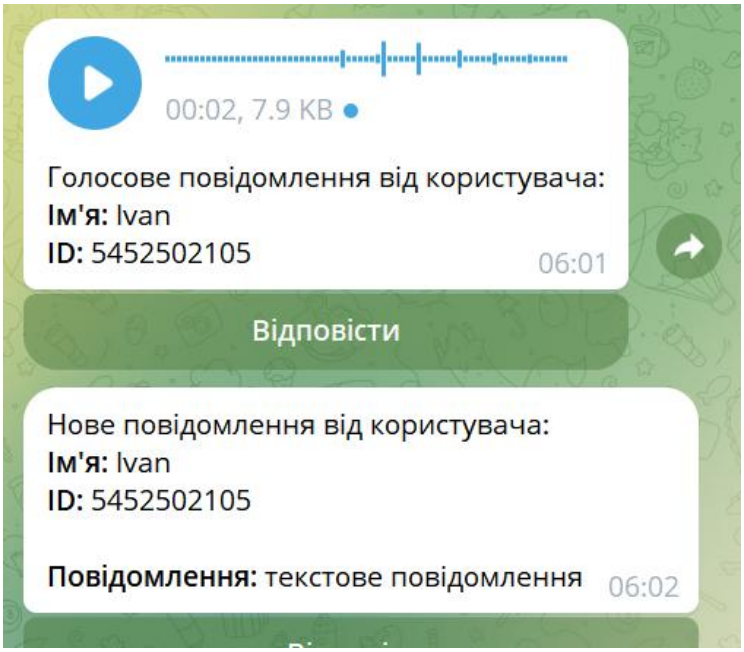

Рисунок 5.17 − Перевірка надходження до адміністрації текстового та голосового повідомлення від користувача

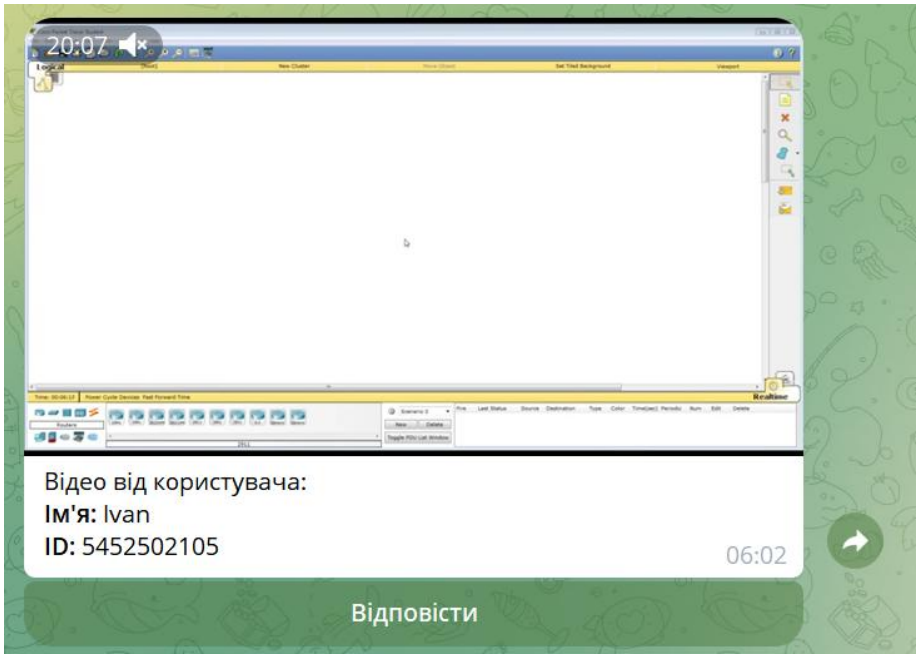

Рисунок 5.18 − Перевірка надходження до адміністрації відео від користувача

5.3.2 Первірка функцій адміністрації

Спочатку перевіримо виведення адмін-панелі при запуску бота (рисунок 5.19).

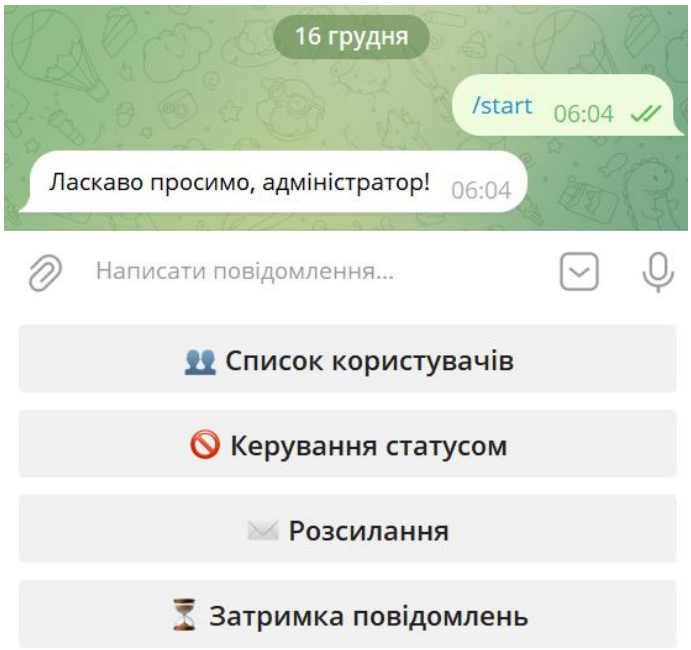

Рисунок 5.19 − Перевірка виводу адмін-панелі

Тепер перевіримо кнопку «Список користувачів» (рисунок 5.20).

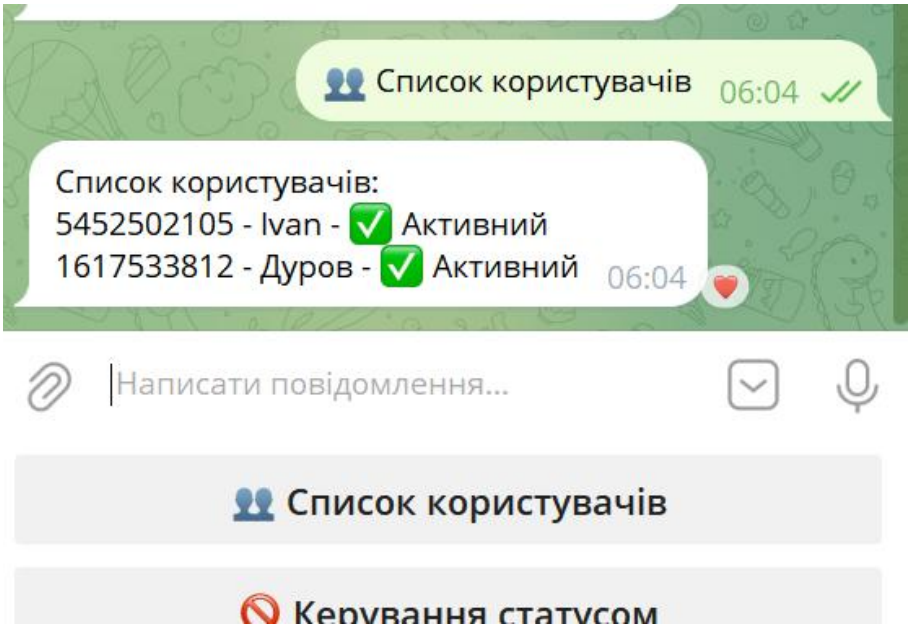

Рисунок 5.20 − Список користувачів

Виконаємо блокування користувача та перевіримо це у списку (рисунок 5.21).

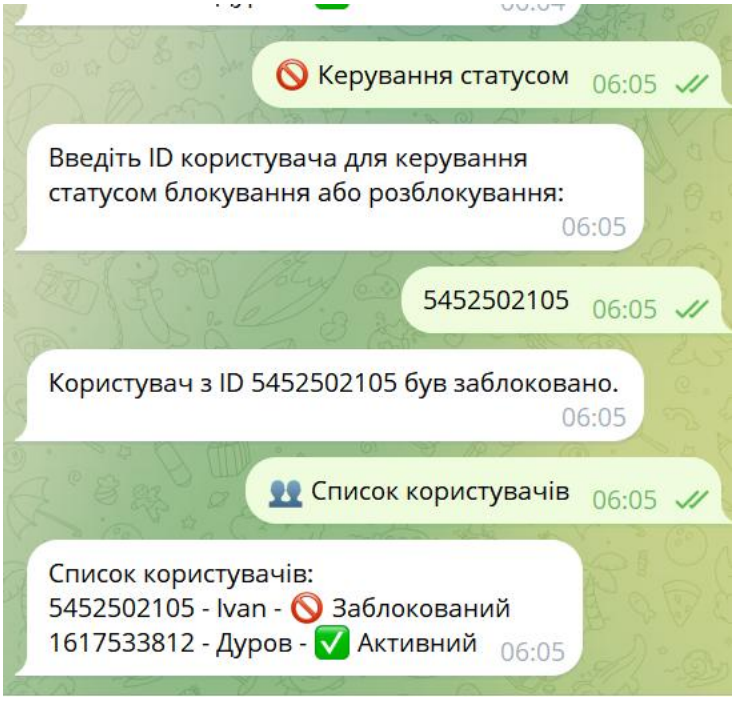

Рисунок 5.21 − Блокування користувача

Запустимо на ще одному акаунті телеграм-бота та перевіримо, як працює розсилка для заблокованих користувачів (рисунок 5.22 та 5.23).

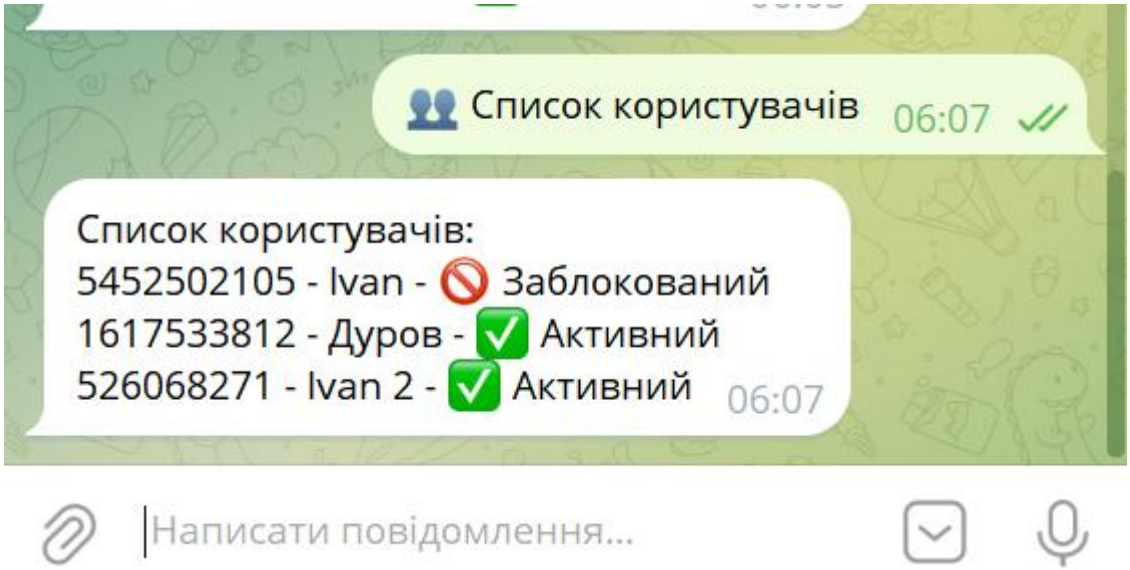

Рисунок 5.22 − Новий користувач боту

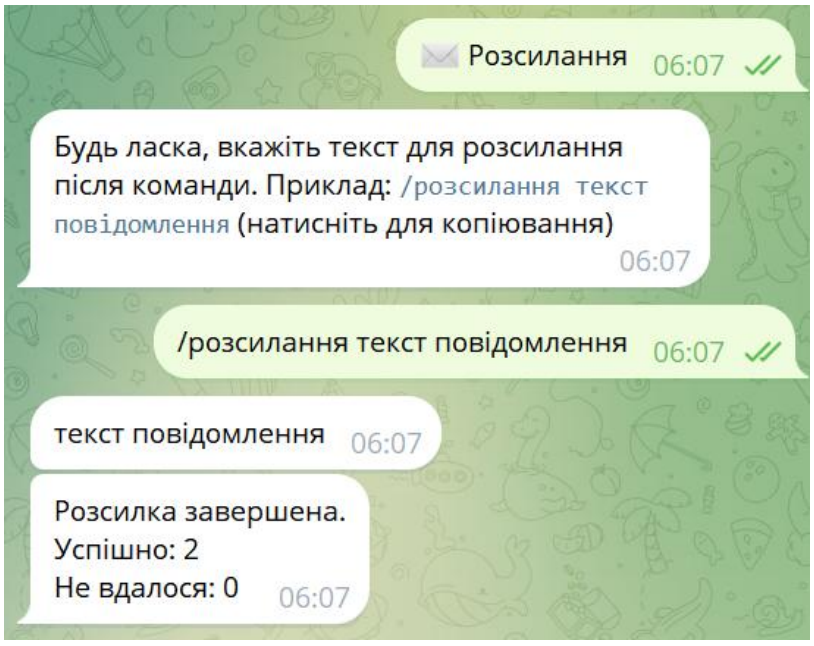

Рисунок 5.23 − Розсилання повідомлення

Як бачимо з вивід бота, що розсилка не велася на заблокованого користувача.

Перевіримо, чи може заблокований користувач надсилати повідомлення або натискати кнопки (рисунок 5.24).

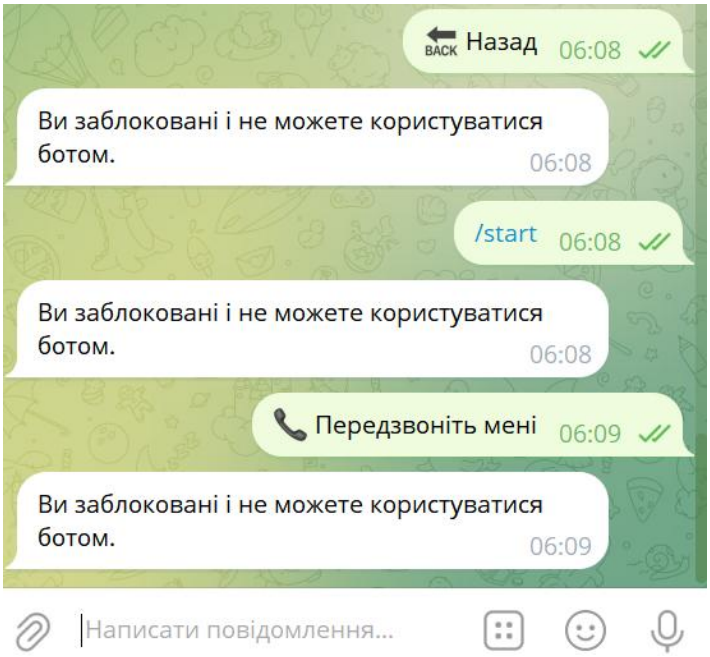

Рисунок 5.24 − Перевірка функцій, якщо користувач заблокований

Як бачимо, запис в базі даних також підтверджує що користувач заблокований (рисунок 5.25).

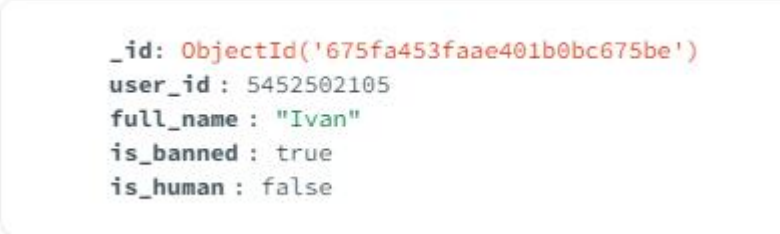

Рисунок 5.25 − Запис блокування користувача в базі даних

Після розблокування користувача, запис в базі даних також змінився (рисунок 5.26 та 5.27).

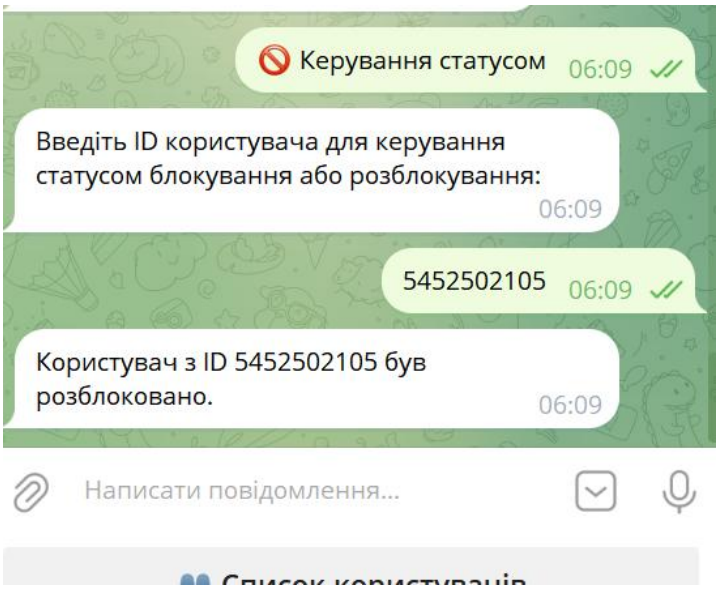

Рисунок 5.26 − Розблокування користувача

Рисунок 5.27 − Зміни в базі даних після розблокування

Перевіримо, чи працює вимкнення затримки для користувача (рисунок 5.28 та рисунок 5.29).

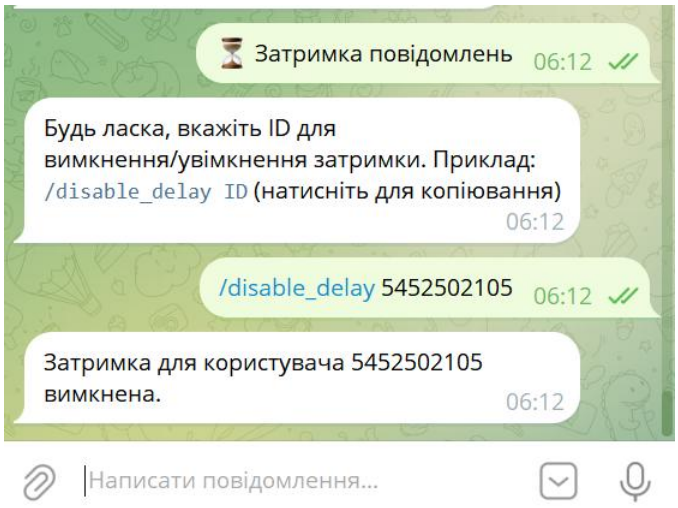

Рисунок 5.28 − Вимкнення затримки

Рисунок 5.29 − Спам повідомленнями без затримки

Тепер перевіримо, чи всі типи файлів, що були вказані, бот приймає у відповідь на повідомлення користувача (рисунок 5.30 −5.33).

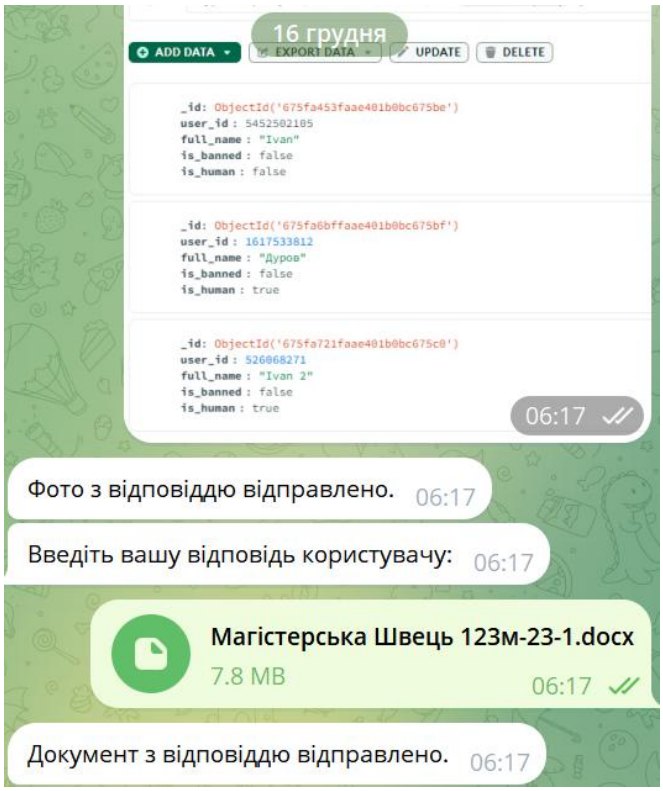

Рисунок 5.30 − Відповідь адміністратора файлом або фото

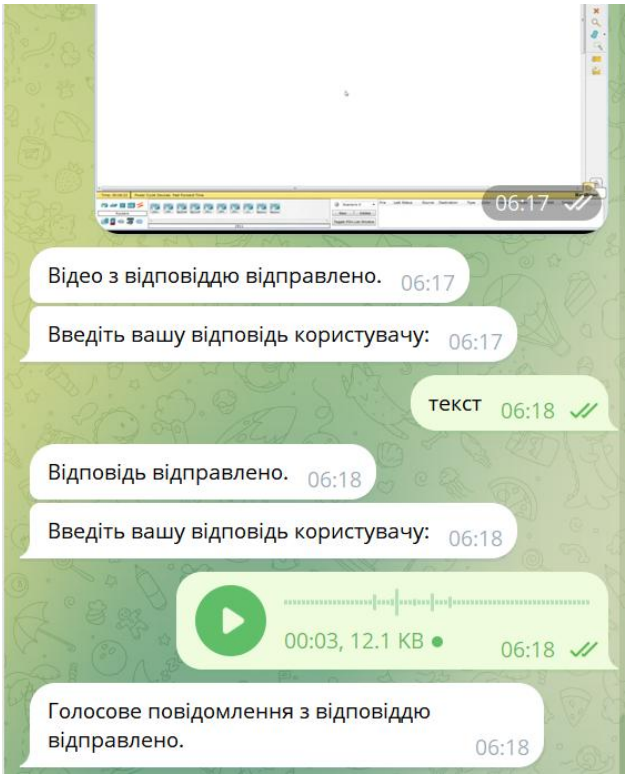

Рисунок 5.31 − Відповідь адміністратора відео або голосовим повідомленням

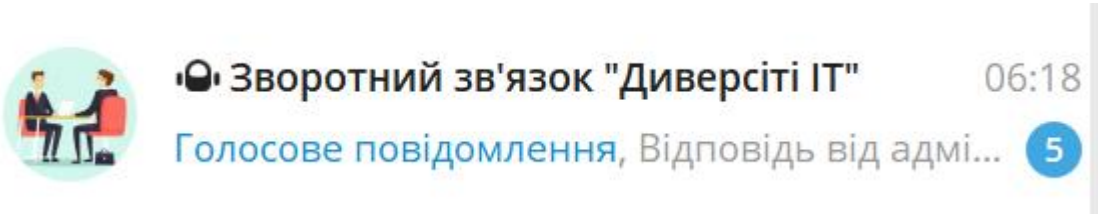

Рисунок 5.32 − Відповіді від адміністрації

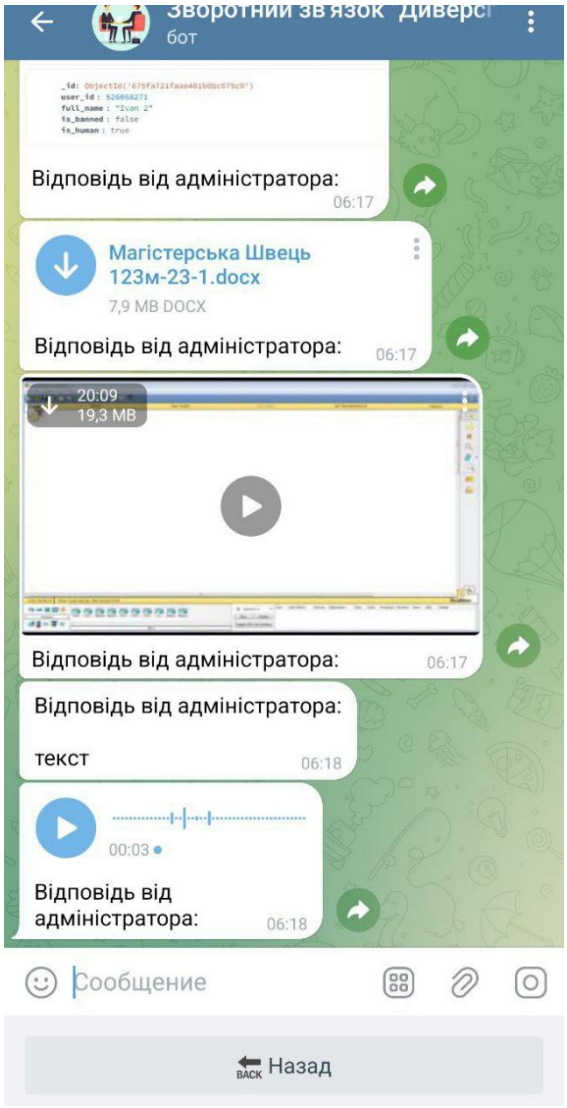

Рисунок 5.33 − Всі відповіді від адміністрації

5.3.3 Перевірка функцій несанкційованого доступу до адмін-панелі або адмін-команд

Спробуємо запустити адмінські функції ввівши команди або ввівши текст кнопки.

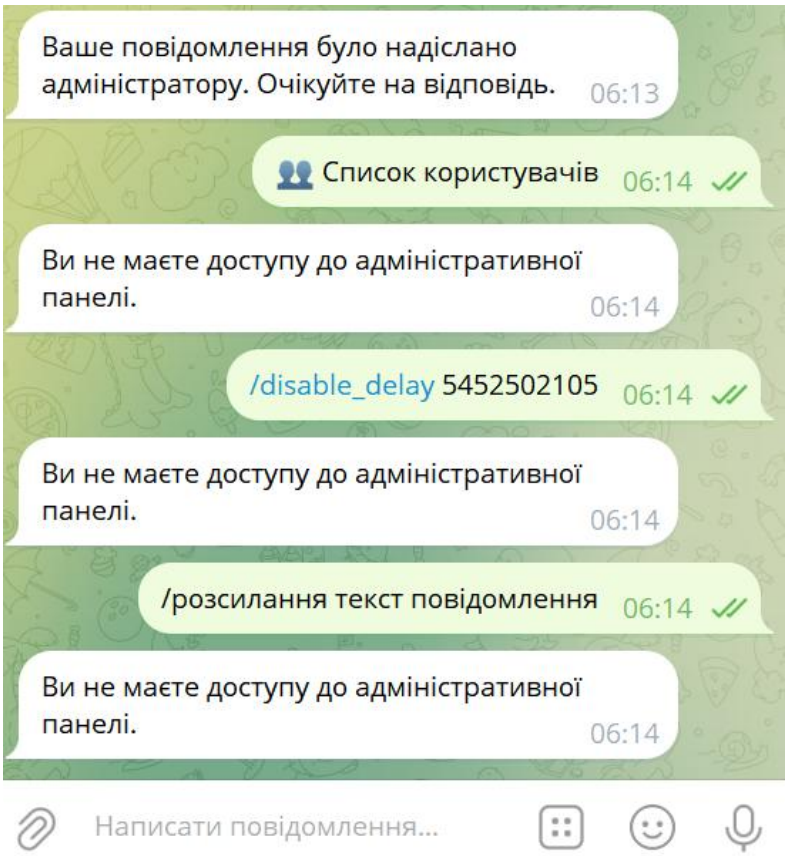

Рисунок 5.34 – Перевірка виконання адмін-функцій звичайним користувачем

5.4 Висновок до розділу

Розділ «Експериментальний» описує методи, мету та завдання проведення експериментів.

Під час чого було отримано позитивні результати, які свідчать про те, що ПЗ працює коректно.

ВИСНОВКИ

Кваліфікаційна робота є завершеною науковою роботою, в якій вирішена науково-практична задача аналізу компанії та вирішення її задач розробки комп’ютерної системи ІТ-компанії «Диверсіті ІТ» з ботом зворотного зв’язку.

Основні висновки і результати роботи полягають в наступному:

1. Проведено аналіз компанії «Диверсіті ІТ» та визначено її наявні проблеми, які потребують вирішення.

2. Проведено аналіз уже наявних рішень Telegram-ботів на ринку для розуміння їхніх переваг, недоліків та застосовності.

3. Сформульовано основні задачі, що необхідно вирішити в межах роботи.

4. Розглянуто комп’ютерну систему, як СМО; також було виконано аналіз різних мов програмування, баз даних і їхніх типів, що дозволило обрати оптимальні інструменти для реалізації системи.

5. Висунуто чіткі технічні та функціональні вимоги, які забезпечують ефективність роботи комп’ютерної системи ІТ-компанії «Диверсіті ІТ».

6. Розроблено схему функціональної структури, що відображає архітектуру та призначення компонентів системи.

7. Виконано розрахунки необхідного обладнання з урахуванням вимог до комп’ютерної системи ІТ-компанії «Диверсіті ІТ».

8. Сформовано алгоритми роботи програми, що забезпечують оптимальну функціональність і обробку даних боту зворотного зв’язку системи.

9. На основі визначених вимог та алгоритмів було розроблено повноцінне програмне забезпечення для боту зворотного зв’язку комп’ютерної системи ІТ-компанії «Диверсіті ІТ».

10. Роботу ПЗ було перевірено експериментальним шляхом, що підтвердило його коректність та відповідність вимогам.

Таким чином, усі поставлені завдання кваліфікаційної роботи було виконано у повному обсязі, а цілі досягнуто на 100%. Розроблена система є готовим рішенням для ефективної роботи компанії «Диверсіті ІТ».

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація Telegram Bot API . URL: https://core.telegram.org/bots/api

2. Бібліотека для Python, яка спрощує створення Telegram-ботів. URL: https://docs.python-telegram-bot.org/en/stable/

3. Як створити Telegram-бота за допомогою Python. URL: https://www.freecodecamp.org/ukrainian/news/yak-stvoryty-telehram-bota-za-dopomohoyu-python/

4. Як створити Telegram-бота на python. URL: https://www.youtube.com/watch?v=oCD9-2IQtko

5. Офіційна документація для MongoDB. URL: https://www.mongodb.com/docs/

6. Основи MongoDB. URL: https://devzone.org.ua/post/osnovy-mongodb

7. MongoDB та Telegram-інтеграція. URL: https://n8n.io/integrations/mongodb/and/telegram/

9. Цвіркун Л. І. Використання месенджерів як системи оповіщення користувачів локальних систем домашнього інтернету речей. / Л.І. Цвіркун, Л.В. Бешта, Ю.А. Миронов // Системні технології. Зб. наук. пр. НМЕтАУ. − 2021. − № 4. − с. 95−101.

10. Литвинов А. Л. Теорія систем масового обслуговування : навч. посібник; Харків. нац. ун-т міськ. госп-ва ім. О.М. Бекетова. – Харків : ХНУМГ ім. О.М. Бекетова, 2018. – 143 с.

11. Цвіркун Л.І. Проблеми масштабованості в розподіленій обробці даних для туманних ІТ-інфраструктур / Цвіркун Л.І. Соболевський І.О. // Міжнародна науково-технічна конференція «Інформаційні технології в металургії та машинобудуванні – ІТММ’2024», 10 − 11 квітня. – Дніпро, УДУНТ, 2024. – С. 390-410

12. Клієнт-серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

13. Rest api. URL: https://dou.ua/forums/topic/34550/

14. Створення асинхронних програм за допомогою Python (asyncio). URL: https://realpython.com/async-io-python/

15. Скалярне та векторне оптимізування запитів у MongoDB. URL: https://www.mongodb.com/docs/manual/aggregation/

16. Мікросервісна архітектура. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/

17. Логування в Python. URL: https://docs.python.org/uk/3/howto/logging.html

18. Паралельне виконання в Python. URL: https://docs.python.org/uk/3.11/library/concurrency.html

19. Створення спеціалізованого чат-боту. URL: https://dou.ua/forums/topic/46902/

20. Python з MongoDB. URL: https://www.mongodb.com/resources/languages/python

21. Індекси MongoDB та проблеми з ними. URL: https://dou.ua/forums/topic/39384/

22. Методи масштабування реляційних баз даних. URL: https://dou.ua/forums/topic/45890/?from=similar_posts_tech

23. MongoDB з Java. URL: https://www.mongodb.com/docs/languages/java/

24. MongoDB з C. URL: https://www.mongodb.com/docs/languages/c/

25. Моніторинг і логування Telegram-ботів з використанням Prometheus та Grafana. URL: http://surl.li/hwktto

Додаток А

Текст програми телеграм боту зворотного зв’язку

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ БОТА ЗВОРОТНОГО ЗВ’ЯЗКУ

Текст програми

804.02070743.24023-01 12 01

Листів 30

АНОТАЦІЯ

Цей документ включає в себе ПЗ для боту зворотного зв’язку.

Текст програми було реалізовано мовою Python.

Середовище розробки, яке використовувалось для розробки ПЗ – Visual Studio Code.

ЗМІСТ

1. Файл команди /start start.py

```
from aiogram import types
from config import users_collection
from add_users import add_user
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton # Додано імпорт
from check import admin_id
from aiogram.types import CallbackQuery
from keyboards import user_menu, back_menu, admin_menu

async def start_command1(message: types.Message):
    # Перевіряємо, чи є користувач у базі даних
    user = await users_collection.find_one({"user_id": message.from_user.id})

    # Якщо користувач заблокований, надсилаємо повідомлення і виймаємо
    if user and user.get("is_banned", False):
        await message.answer("Ви заблоковані і не можете користуватися ботом.")
        return

    # Якщо користувач не підтвердив, що він не робот
    if not user or not user.get("is_human", False):
        # Створюємо кнопку для підтвердження
        keyboard = InlineKeyboardMarkup().add(
            InlineKeyboardButton("Я не робот", callback_data=f"verify_{message.from_user.id}")
        )

        # Запитуємо підтвердження
        await message.answer(
```

```
            "Щоб продовжити, натисніть кнопку, щоб підтвердити, що ви не бот.",
            reply_markup=keyboard
        )
    else:
        # Якщо користувач вже підтвердив, що не робот, показуємо стандартне меню
        # Вітання користувача
        if message.from_user.id in admin_id:
            await message.answer("Ласкаво просимо, адміністратор!", reply_markup=admin_menu)
        else:
            await message.answer("Ласкаво просимо! Ви можете поставити своє запитання або пропозицію  ", reply_markup=user_menu)


async def process_recaptcha_verification1(callback_query: types.CallbackQuery):
    user_id = int(callback_query.data.split('_')[1])
    # Оновлюємо статус користувача в базі даних
    await add_user(user_id, callback_query.from_user.full_name, is_human=True)
```

2. Файл конфігурації config.py

```
import motor.motor_asyncio
import os
from motor.motor_asyncio import AsyncIOMotorClient

API_TOKEN = '7627528973:AAH3guWUGJ3jGAvfKlGmOOKEVven7EHaVvo'  # токен бота
admin_id = [1617533812]      # ID адміністратора
MONGO_URI = "mongodb://127.0.0.1:27017"  # MongoDB
DB_NAME = 'users'
```

```
mongo_client = AsyncIOMotorClient(MONGO_URI)
db = mongo_client[DB_NAME]
users_collection = db['users']
```

3. Файл користувацьких функцій users_function.py

```
# users_function.py
from aiogram import types
from config import users_collection  # Імпортуємо необхідні змінні
from keyboards import user_menu, back_menu, admin_menu
from check import check_admin_access, check_user_status, admin_id
```

async def about_us1(message: types.Message): # Функція для відправлення інформації про компанію

```
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return
    await message.answer("IT-компанії «Диверсіті IT» – це IT-компанія, яка спеціалізується на розробці програмного забезпечення, наданні IT-послуг для бізнесу та має команду висококваліфікованих професіоналів з великим досвідом роботи у галузі програмної розробки. Компанія завжди відкрита до нових ідей та готова реалізувати будь-яку задумку своїх клієнтів  ", reply_markup=user_menu)
```

async def contact_admin1 (message: types.Message): # Функція для зв’язку з адміністратором

```
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return
    await message.answer("Напишіть ваше повідомлення для адміністратора:", reply_markup=back_menu)
```

```
async def back_to_menu1(message: types.Message):  # Функція для повернення до головного меню
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return

    if message.from_user.id in admin_id:
        await message.answer("Ви повернулися в головне меню адміністратора.", reply_markup=admin_menu)
    else:
        await message.answer("Ви повернулися в головне меню користувача.", reply_markup=user_menu)

async def request_phone_number1(message: types.Message):  # Функція для запиту номера телефону
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return

    # Просимо користувача ввести номер телефону
    await message.answer("Будь ласка, введіть ваш номер телефону в форматі +380XXXXXXXXX:")

async def handle_phone_number1(message: types.Message):  # Функція для обробки введеного номера телефону
    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return
    # Відправляємо повідомлення адміну з номером телефону
```

4. Файл адмін-функцій admin_function.py

```
from aiogram import types
from config import users_collection  # Імпортуємо необхідні змінні
from keyboards import user_menu, back_menu, admin_menu
from check import check_admin_access, check_user_status, admin_id
from add_users import add_user

async def manage_status1(message: types.Message):

    user_id = message.from_user.id
    if not await check_admin_access(message):  # Перевірка прав адміністратора
        return
    if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
        return

    await message.answer("Введіть ID користувача для керування статусом блокування або розблокування:", reply_markup=back_menu)

async def toggle_user_status1(message: types.Message):

    user_id = message.from_user.id
    if not await check_admin_access(message):  # Перевірка прав адміністратора
        return
    if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
        return
```

```
        user_id_to_toggle = int(message.text)
        user = await users_collection.find_one({"user_id": user_id_to_toggle})

        if user:
            # Якщо користувач заблокований, розблокувати його, інакше заблокувати
            new_status = not user['is_banned']
            await add_user(user_id_to_toggle, user['full_name'], is_banned=new_status)
            action = "заблоковано" if new_status else "розблоковано"
            await message.answer(f"Користувач з ID {user_id_to_toggle} був {action}.", reply_markup=admin_menu)
        else:
            await message.answer("Користувач не знайдений.", reply_markup=admin_menu)

    async def list_users1(message: types.Message):

        user_id = message.from_user.id
        if not await check_admin_access(message):  # Перевірка прав адміністратора
            return
        if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
            return

        users = await users_collection.find().to_list(length=None)
        user_list = "\n".join(
            [f"{user['user_id']} - {user['full_name']} - {' Заблокований' if user.get('is_banned') else '✅ Активний'}" for user in users]
```

```
        )
        await message.answer(f"Список користувачів:\n{user_list}", reply_markup=admin_menu)

    async def start_broadcast_info1(message: types.Message):

        user_id = message.from_user.id
        if not await check_admin_access(message):  # Перевірка прав адміністратора
            return
        if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
            return

        await message.answer("Будь ласка, вкажіть текст для розсилання після команди. Приклад: `/розсилання текст повідомлення` (натисніть для копіювання)", parse_mode='Markdown')

    async def start_broadcast_info2(message: types.Message):
        user_id = message.from_user.id
        if not await check_admin_access(message):  # Перевірка прав адміністратора
            return
        if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
            return

        await message.answer("Будь ласка, вкажіть ID для вимкнення/увімкнення затримки. Приклад: `/disable_delay ID` (натисніть для копіювання)", parse_mode='Markdown')
```

```
# Словник для відключення затримки для користувача
user_delay_disabled = {}

async def disable_delay1(message: types.Message):
    # Перевірка прав адміністратора
    user_id = message.from_user.id
    if not await check_admin_access(message):
        return
    if not await check_user_status(user_id, message, users_collection):  # Перевірка статусу користувача
        return

    try:
        # Перевіряємо, що команда містить аргумент
        parts = message.text.split()
        if len(parts) < 2:
            await message.answer("Будь ласка, вкажіть ID користувача для вимкнення або увімкнення затримки.")
            return

        # Спробуємо перетворити аргумент у ціле число
        target_user_id = int(parts[1])

        # Перевіряємо поточний стан затримки
        if target_user_id in user_delay_disabled and user_delay_disabled[target_user_id]:
            # Включаємо затримку, якщо вона була вимкнена
            user_delay_disabled[target_user_id] = False
```

```
            await message.answer(f"Затримка для користувача {target_user_id} увімкнена.")
        else:
            # Відключаємо затримку, якщо вона була увімкнена або не задана
            user_delay_disabled[target_user_id] = True
            await message.answer(f"Затримка для користувача {target_user_id} вимкнена.")
    except ValueError:
        # Якщо аргумент не є числом
        await message.answer("ID користувача має бути числом. Будь ласка, перевірте введення.")
    except Exception as e:
        # Загальна обробка непередбачених помилок
        await message.answer(f"Сталася помилка: {str(e)}")
```

5. Файл додавання користувачів add_users.py

```
from config import users_collection  # Імпортуємо колекцію користувачів з конфігурації

async def add_user(user_id, full_name, is_banned=False, is_human=False):
    # Додаємо користувача до бази даних або оновлюємо його дані, якщо такий користувач уже існує
    user_data = {
        "user_id": user_id,
        "full_name": full_name,
        "is_banned": is_banned,
        "is_human": is_human
    }
    await users_collection.update_one(
        {"user_id": user_id},
```

```
        {"$set": user_data},
        upsert=True
    )
```

6. Головний файл запуску bot.py

```
import logging
from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.dispatcher import FSMContext
from aiogram.types import Message
from aiogram.dispatcher.filters.state import State, StatesGroup
from aiogram import types

from config import API_TOKEN, admin_id, MONGO_URI, DB_NAME, users_collection
from check import check_user_status, check_admin_access, is_user_banned, is_admin
from add_users import add_user
from keyboards import user_menu, admin_menu, back_menu
from users_function import about_us1, contact_admin1, back_to_menu1, request_phone_number1, handle_phone_number1
from start import start_command1, process_recaptcha_verification1
from admin_function import manage_status1, toggle_user_status1, list_users1, start_broadcast_info1, start_broadcast_info2, disable_delay1, user_delay_disabled

import time
import asyncio

logging.basicConfig(level=logging.INFO) # Налаштування бота та диспетчера
bot = Bot(token=API_TOKEN)
```

```
storage = MemoryStorage()
dp = Dispatcher(bot, storage=MemoryStorage())

class Feedback(StatesGroup): # Стан для зворотного зв’язку
    awaiting_reply = State()

@dp.message_handler(commands=['start'])
async def start_command(message: types.Message):
    await start_command1(message)

@dp.callback_query_handler(lambda c: c.data.startswith('verify_'))
async def process_recaptcha_verification(callback_query: types.CallbackQuery):
    await process_recaptcha_verification1(callback_query)
    user_id = int(callback_query.data.split('_')[1])
    await bot.send_message(user_id, "Ви успішно підтвердили, що не є роботом!")
    await bot.send_message(user_id, "Ласкаво просимо! Ви можете поставити своє запитання або пропозицію  ", reply_markup=user_menu)

@dp.message_handler(content_types=['text'], text='  Керування статусом’) # Обробник кнопки "  Керування статусом"
async def manage_status(message: types.Message):
    await manage_status1(message)

@dp.message_handler(lambda message: message.text.isdigit(), state='*') # Обробник для керування статусом користувача
async def toggle_user_status(message: types.Message):
    await toggle_user_status1(message)
```

```
@dp.message_handler(content_types=['text'], text='    Список користувачів') # Обробник кнопки "    Список користувачів"
async def list_users(message: types.Message):
    await list_users1(message)

@dp.message_handler(content_types=['text'], text='    Розсилання') # Обробник кнопки для розсилання
async def start_broadcast_info(message: types.Message):
    await start_broadcast_info1(message)

@dp.message_handler(commands=['розсилання'])
async def start_broadcast(message: types.Message):

    user_id = message.from_user.id
    if not await check_admin_access(message):
        return
    if not await check_user_status(user_id, message, users_collection):
        return

    broadcast_text = message.get_args() # Отримуємо текст розсилки
    if not broadcast_text:
        await message.answer(
            "Будь ласка, вкажіть текст для розсилання після команди.\n"
            "Приклад: /розсилання Привіт, {ім'я}! Ми маємо для вас чудову новину!",
            parse_mode='Markdown',
        )
        return
```

```
        users = await users_collection.find({"is_banned": False}).to_list(length=None)
# Отримуємо список активних користувачів
        success = 0     # Ініціалізуємо лічильники
        failed = 0

        for user in users:     # Розсилка повідомлень
            user_id = user['user_id']
            user_name = user.get("full_name", "користувач")
            personalized_text = broadcast_text.replace("{ім’я}", user_name)

            try:
                await bot.send_message(user_id, personalized_text)
                success += 1
            except Exception as e:
                logging.error(f"Не вдалося відправити повідомлення користувачу
{user_id}: {e}")
                failed += 1

            await asyncio.sleep(1)  # Затримка між повідомленнями

        await message.answer(     # Відправка статистики адміністратору
            f"Розсилка завершена.\n"
            f"Успішно: {success}\n"
            f"Не вдалося: {failed}"
        )

    @dp.message_handler(content_types=['text'], text='⏳ Затримка повідомлень')
# Обробник для затримки повідомлень
    async def start_broadcast_info(message: types.Message):
        await start_broadcast_info2(message)
```

```
@dp.message_handler(commands=['disable_delay']) # Команда для адміну, щоб вимкнути затримку користувача
async def disable_delay(message: types.Message):
    await disable_delay1(message)

@dp.message_handler(content_types=['text'], text='  Про нас') # Обробник для текстового повідомлення "  Про нас"
async def about_us(message: Message):
    await about_us1(message)

@dp.message_handler(content_types=['text'], text='  Зв’язатися з підтримкою') # Обробник для текстового повідомлення "  Зв’язатися з підтримкою"
async def contact_admin(message: types.Message):
    await contact_admin1(message)

@dp.message_handler(content_types=['text'], text='  Назад') # Обробник для текстового повідомлення "  Назад"
async def back_to_menu(message: types.Message):
    await back_to_menu1(message)

@dp.message_handler(content_types=['text'], text='  Передзвоніть мені') # Обробник для текстового повідомлення "  Передзвоніть мені"
async def request_phone_number(message: types.Message):
    await request_phone_number1(message)

@dp.message_handler(lambda message: message.text and message.text.startswith("+"), content_types=['text']) # Обробник для отримання номера телефону
async def handle_phone_number(message: types.Message):
```

```
    await handle_phone_number1(message)          # Отримуємо введений номер телефону
    phone_number = message.text.strip().replace(" ", "")

    if not phone_number.startswith("+"):
        await message.answer("Будь ласка, введіть правильний номер телефону в форматі +380123456789.")
        return

    for admin in admin_id:
        await bot.send_message(
            admin,
            f"    Користувач: {message.from_user.full_name}\n"
            f"   ID користувача: {message.from_user.id}\n"
            f"    Номер телефону для зворотного зв’язку: {phone_number}\n\n"
            f"Залишено номер для дзвінку!"
            )
    await message.answer("Будь ласка, напишіть будь ласка тему звернення і наш адміністратор з вами зв’яжеться:", reply_markup=back_menu) # Запитуємо повідомлення з зверненням

    last_message_time = {}   # Словник для збереження часу останнього повідомлення кожного користувача
    message_count = {}   # Словник для відстежування кількості повідомлень користувача за певний час
    MESSAGE_TIMEOUT = 30   # Час в секундах, який потрібно чекати перед відправкою наступного повідомлення (наприклад, 30 секунд)
    MAX_MESSAGES = 2   # Максимальна кількість повідомлень за 30 секунд
```

```
@dp.message_handler(lambda message: message.text or message.photo or message.video or message.audio or message.voice or message.document, content_types=[types.ContentType.TEXT, types.ContentType.PHOTO, types.ContentType.VIDEO, types.ContentType.AUDIO, types.ContentType.VOICE, types.ContentType.DOCUMENT])
async def forward_to_admin(message: types.Message):

    user_id = message.from_user.id
    if not await check_user_status(user_id, message, users_collection):
        return

    if user_id in user_delay_disabled and user_delay_disabled[user_id]: # Якщо затримку вимкнено для користувача, не перевіряємо час
        user_link = f'<a href="tg://user?id={user_id}">{user_id}</a>' # Надсилаємо повідомлення адміністратору без затримки
        if message.text: # Обробляємо текстове повідомлення
            for admin in admin_id:
                await bot.send_message(
                    admin,
                    f"Нове повідомлення від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}\n\n<b>Повідомлення:</b> {message.text}",
                    parse_mode="HTML",
                    reply_markup=types.InlineKeyboardMarkup().add(
                        types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                    )
                )

        elif message.photo: # Обробляємо фото
```

```
            photo = message.photo[-1].file_id
            for admin in admin_id:
                await bot.send_photo(
                    admin,
                    photo,
                    caption=f"Фото від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                    parse_mode="HTML",
                    reply_markup=types.InlineKeyboardMarkup().add(
                        types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                    )
                )

        elif message.video: # Обробляємо відео
            video = message.video.file_id
            for admin in admin_id:
                await bot.send_video(
                    admin,
                    video,
                    caption=f"Відео від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                    parse_mode="HTML",
                    reply_markup=types.InlineKeyboardMarkup().add(
                        types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                    )
                )

        elif message.audio: # Обробляємо аудіо
```

```
            audio = message.audio.file_id
            for admin in admin_id:
                await bot.send_audio(
                    admin,
                    audio,
                    caption=f"Аудіо від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                    parse_mode="HTML",
                    reply_markup=types.InlineKeyboardMarkup().add(
                        types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                    )
                )

        elif message.voice: # Обробляємо голосове повідомлення
            voice = message.voice.file_id
            for admin in admin_id:
                await bot.send_voice(
                    admin,
                    voice,
                    caption=f"Голосове повідомлення від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                    parse_mode="HTML",
                    reply_markup=types.InlineKeyboardMarkup().add(
                        types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                    )
                )
```

```
            elif message.document: # Обробляємо документ
                document = message.document.file_id
                for admin in admin_id:
                    await bot.send_document(
                        admin,
                        document,
                        caption=f"Документ від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                        parse_mode="HTML",
                        reply_markup=types.InlineKeyboardMarkup().add(
                            types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                        )
                    )

            await message.answer("Ваше повідомлення було надіслано адміністратору. Очікуйте на відповідь.")
            return

        current_time = time.time() # Отримуємо поточний час
        if user_id in last_message_time: # Перевірка під час останнього повідомлення
            time_diff = current_time - last_message_time[user_id]
            if time_diff < MESSAGE_TIMEOUT:
                if user_id in message_count and message_count[user_id] >= MAX_MESSAGES: # Перевіряємо, скільки повідомлень надіслав користувач за останні 30 секунд
                    await message.answer(f"Ви надто часто відправляєте повідомлення. Будь ласка, почекайте {MESSAGE_TIMEOUT - int(time_diff)} секунд.")
                    return
```

```
            else:
                message_count[user_id] += 1 # Якщо повідомлень менше 2-х, збільшуємо лічильник
        else:
            message_count[user_id] = 1 # Якщо минуло більше 30 секунд, скидаємо лічильник повідомлень

    else:
        message_count[user_id] = 1 # Якщо це перше повідомлення, просто додаємо

    last_message_time[user_id] = current_time # Оновлюємо час останнього повідомлення

    user_link = f'<a href="tg://user?id={user_id}">{user_id}</a>' # Надсилаємо повідомлення адміністратору
    for admin in admin_id:
        if message.text:
            await bot.send_message(
                admin,
                f"Нове повідомлення від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}\n\n<b>Повідомлення:</b> {message.text}",
                parse_mode="HTML",
                reply_markup=types.InlineKeyboardMarkup().add(
                    types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                )
            )
        elif message.photo:
```

```
            photo = message.photo[-1].file_id
            await bot.send_photo(
                admin,
                photo,
                caption=f"Фото від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                parse_mode="HTML",
                reply_markup=types.InlineKeyboardMarkup().add(
                    types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                )
            )
        elif message.video:
            video = message.video.file_id
            await bot.send_video(
                admin,
                video,
                caption=f"Відео від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                parse_mode="HTML",
                reply_markup=types.InlineKeyboardMarkup().add(
                    types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                )
            )
        elif message.audio:
            audio = message.audio.file_id
            await bot.send_audio(
                admin,
                audio,
```

```
                caption=f"Аудіо від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                parse_mode="HTML",
                reply_markup=types.InlineKeyboardMarkup().add(
                    types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                )
            )
        elif message.voice:
            voice = message.voice.file_id
            await bot.send_voice(
                admin,
                voice,
                caption=f"Голосове повідомлення від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                parse_mode="HTML",
                reply_markup=types.InlineKeyboardMarkup().add(
                    types.InlineKeyboardButton("Відповісти", callback_data=f"reply_{user_id}")
                )
            )
        elif message.document:
            document = message.document.file_id
            await bot.send_document(
                admin,
                document,
                caption=f"Документ від користувача:\n<b>Ім’я:</b> {message.from_user.full_name}\n<b>ID:</b> {user_link}",
                parse_mode="HTML",
                reply_markup=types.InlineKeyboardMarkup().add(
```

```
                    types.InlineKeyboardButton("Відповісти",
callback_data=f"reply_{user_id}")
                )
            )

        await message.answer("Ваше повідомлення було надіслано адміністратору. Очікуйте на відповідь.")  # Повідомлення користувачу, що його повідомлення надіслано адміністратору
        await asyncio.sleep(MESSAGE_TIMEOUT)  # Затримка на вказаний час (наприклад, 30 секунд)
        if user_id in message_count: message_count[user_id] = 0  # Скидаємо лічильник повідомлень користувача після закінчення часу

    @dp.callback_query_handler(lambda call: call.data and call.data.startswith("reply_")) # Обробник натискання на кнопку "Відповісти"
    async def start_reply(call: types.CallbackQuery, state: FSMContext):
        user_id = int(call.data.split("_")[1])
        await call.message.answer("Введіть вашу відповідь користувачу:")
        await state.update_data(user_id=user_id)
        await Feedback.awaiting_reply.set()

    # Обробник відповіді адміністратора
    @dp.message_handler(state=Feedback.awaiting_reply,
content_types=[types.ContentType.TEXT, types.ContentType.PHOTO, types.ContentType.DOCUMENT, types.ContentType.VIDEO, types.ContentType.AUDIO, types.ContentType.VOICE])
    async def process_reply(message: types.Message, state: FSMContext):
        data = await state.get_data()
        user_id = data.get("user_id")
```

```
        if message.photo:
            photo = message.photo[-1].file_id
            caption = f"Відповідь від адміністратора:\n\n{message.caption or ''}"
            await bot.send_photo(user_id, photo, caption=caption)
            await message.answer("Фото з відповіддю відправлено.")

        elif message.document:
            document = message.document.file_id
            caption = f"Відповідь від адміністратора:\n\n{message.caption or ''}"
            await bot.send_document(user_id, document, caption=caption)
            await message.answer("Документ з відповіддю відправлено.")

        elif message.video:
            video = message.video.file_id
            caption = f"Відповідь від адміністратора:\n\n{message.caption or ''}"
            await bot.send_video(user_id, video, caption=caption)
            await message.answer("Відео з відповіддю відправлено.")

        elif message.audio:
            audio = message.audio.file_id
            caption = f"Відповідь від адміністратора:\n\n{message.caption or ''}"
            await bot.send_audio(user_id, audio, caption=caption)
            await message.answer("Аудіо з відповіддю відправлено.")

        elif message.voice:
            voice = message.voice.file_id
            caption = f"Відповідь від адміністратора:\n\n{message.caption or ''}"
            await bot.send_voice(user_id, voice, caption=caption)
            await message.answer("Голосове повідомлення з відповіддю
відправлено.")
```

```
        else:
            await bot.send_message(user_id, f"Відповідь від адміністратора:\n\n{message.text}")
            await message.answer("Відповідь відправлено.")

        await state.finish()

    if __name__ == "__main__":
        from aiogram import executor
        executor.start_polling(dp)
```

7. Файл перевірки check.py

```
    from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton
    from aiogram.types import KeyboardButton, ReplyKeyboardMarkup
    from config import admin_id
    from config import users_collection

    async def is_user_banned(user_id):
        # Перевірка, чи заблокований користувач
        user = await users_collection.find_one({"user_id": user_id})
        return user['is_banned'] if user and 'is_banned' in user else False

    async def check_user_status(user_id, message, users_collection):
        # Перевірка, чи заблокований користувач
        if await is_user_banned(user_id):
            await message.answer("Ви заблоковані і не можете користуватися ботом.")
            return False
        else:
```

```
        # Перевірка, чи існує користувач і чи підтвердив він, що не є ботом
            user = await users_collection.find_one({"user_id": user_id})
            if not user or not user.get("is_human", False):
                # Створення кнопки для підтвердження
                keyboard = InlineKeyboardMarkup().add(
                    InlineKeyboardButton("Я не робот", callback_data=f"verify_{message.from_user.id}")
                )

                # Запит на підтвердження
                await message.answer(
                    "Щоб продовжити, натисніть кнопку, щоб підтвердити, що ви не бот.",
                    reply_markup=keyboard
                )
                return False
            return True

        async def is_admin(user_id: int) -> bool:
            return user_id in admin_id

        async def check_admin_access(message):
            # Перевірка, чи є користувач адміністратором
            if not await is_admin(message.from_user.id):
                await message.answer("Ви не маєте доступу до адміністративної панелі.")
                return False
            return True
```

8. Файл кнопок keyboards.py

```
from aiogram import types
```

```
    # Створення кнопок меню для користувача та адміністратора
    user_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)  # Меню для користувача
    user_menu.add(types.KeyboardButton('    Про нас'))  # Кнопка "Про нас"
    user_menu.add(types.KeyboardButton('   Зв’язатися з підтримкою'))  # Кнопка "Зв’язатися з підтримкою"
    user_menu.add(types.KeyboardButton('    Передзвоніть мені'))  # Кнопка "Передзвоніть мені"

    admin_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)  # Меню для адміністратора
    admin_menu.add(types.KeyboardButton('    Список користувачів’))  # Кнопка "Список користувачів"
    admin_menu.add(types.KeyboardButton('    Керування статусом’))  # Кнопка "Керування статусом"
    admin_menu.add(types.KeyboardButton('    Розсилання'))  # Кнопка "Розсилання"
    admin_menu.add(types.KeyboardButton('⧗   Затримка повідомлень'))  # Кнопка "Затримка повідомлень"

    back_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)  # Меню для повернення
    back_menu.add(types.KeyboardButton("    Назад"))  # Кнопка "Назад"
```