МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

_____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: Telegram-bot для автоматизації взаємодії працівників СТО з клієнтами

Здобувача групи ІТ-03 Немийного Олексія Володимировича
(шифр групи) (прізвище, ім'я, по батькові)

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

_____ Олексій Немийний
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник Кузнєцов Едуард Генадійович к.т.н доцент _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Компютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖЕНО
В. о. зав. кафедри ІТ

_____Світлана ВАЩЕНКО
«__» _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРУ СТУДЕНТУ

Немийному Олексію Володимировичу

1 Тема роботи Telegram-bot для автоматизації взаємодії працівників СТО з клієнтами

керівник роботи ___Кузнєцов Едуард Генадійович к.т.н доцент_____

затверджені наказом по університету від « 07 »__травня__2024 р. __№0482-VI__

2 Строк подання студентом роботи « 26 » __травня__ 2024 р.

3 Вхідні дані до роботи__введення даних клієнтом про себе, надання списку послуг, вибір послуги,перевірка статусу_____

4 Зміст розрахунково-пояснювальної записки (перелік питань , які потрібно розробити)___ Розглянути актуальні дослідження в галузі, проаналізувати існуючі решення, моделювання роботи бота в нотаціях, розробити програмне рішення, протестувати програмний продукт_____

5 Перелік графічного матеріалу (з точним значенням обов'язкових кресл.) контекстна діаграма у нотації IDEF0, діаграма декомпозиції 1-го рівня у нотації IDEF0, діаграма варіантів використання, логічна діаграма БД.

6. Консультанти розділів роботи:

Розділ	**Консультант**	**Підпис, дата**	
		Завдання видав	**Завдання прийня**

7. Дата видачі завдання __08.04.24_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Проаналізувати існуючі рішення	08.04.24 - 12.04.24	
2	Вибір засобів розробки та планування меню бота	15.04.24 - 19.04.24	
3	Планування функціоналу	22.04.24 - 26.04.24	
4	Створення функціоналу телеграм-боту	29.04.24 - 3.05.24	
5	Створення баз даних та перевірка на її заповнення	06.05.24 - 10.05.24	
6	Перевірка на працездатнійсть системи	13.05.24 - 17.05.24	
7	Тестування	20.05.24 - 21.05.24	
8	Написання пояснювальної записки	22.05.24 – 26.05.24	

Студент _____ Олексій НЕМИЙНИЙ

Керівник роботи _____ к.т.н доцент Едуард КУЗНЄЦОВ

АНОТАЦІЯ

Тема кваліфікаційної роботи бакалавра «Telegram-bot для автоматизації взаємодії працівників СТО з клієнтами».

Пояснювальна записка складається зі вступу, 3 розділів, списку використаних джерел із 25 найменувань, висновків, додатків. Загальний обсяг роботи 70 сторінок, у тому числі 59 сторінки основного тексту, 2 сторінки списку використаних джерел, 21 сторінок додатків.

Актуальніст теми автомобільних сервісів та інновацій у цій галузі є надзвичайно високою в сучасному світі, оскільки автомобілі відіграють важливу роль у повсякденному житті мільйонів людей по всьому світу. Зростаючий обсяг автопарку, технологічні зміни у сфері автомобільного виробництва та зростання вимог до безпеки та зручності управління автомобілем стимулюють розвиток нових технологій та сервісів для забезпечення найвищого рівня обслуговування. Тому розробка обслуговування для сервісу СТО та надання послуг віддалено в режимі реального часу є дуже важливим аспектом

Мета роботи розробити телеграм бота для взаємодії працівників СТО з клієнтами. Основна мета цього бота це щоб клієнт міг змогу вибрати віддалено послугу та стати в чергу , а при виконанні робіт на самому сервісі , будуть змінюватися статуси виконання послуг, що клієнти в свою чергу зможуть відстежувати самостійно через телеграм бота.

Результат роботи – цей телеграм-бот було попрактиковано не сервісі СТО з знайомими для розуміння зручності як зі сторони клієнта так і зі сторони адміністратора.

Ключові слова: телеграм-бот, месенджер Telegram, база даних, токен, програмний продукт.

ЗМІСТ

ВСТУП

Актуальність теми. Тема автомобільних сервісів та інновацій у цій галузі є надзвичайно актуальною в сучасному світі, оскільки автомобілі відіграють важливу роль у повсякденному житті мільйонів людей по всьому світу. Зростаючий обсяг автопарку, технологічні зміни у сфері автомобільного виробництва та зростання вимог до безпеки та зручності управління автомобілем стимулюють розвиток нових технологій та сервісів для забезпечення найвищого рівня обслуговування.

Об'єкт дослідження – онлайн технології підтримання діяльності комерційних послуг.

Предмет дослідження - розробка та впровадження телеграм-боту для автомобільного сервісу, що спрощує взаємодію клієнтів з сервісним центром, надаючи зручний спосіб замовлення та контролю ремонтних робіт.

Мета дослідження - Розробка ефективного та функціонального телеграм-боту для автомобільного сервісу з метою поліпшення якості обслуговування клієнтів, забезпечення швидкого доступу до інформації про послуги, а також оптимізації робочих процесів у сервісному центрі.

Відповідно до мети було поставлено наступні **завдання:**

- Розглянути та проаналізувати існуючі рішення в області онлайн обслуговування;
- Моделювання роботи бота;
- Розробити забеспечення телеграм боту;
- Розробити прогрмане рішення
- Протестувати розроблену програму

Структура роботи. Робота складається з трьох розділів, восьми підрозділів, висновків та списку використаних джерел. Загальний обсяг роботи - 60 сторінок

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд актуальних досліджень і публікацій

Незважаючи на те, що чат-бот здається нещодавнім модним словом, він існує відтоді, як люди розробили спосіб взаємодії з комп'ютерами. Перший чат-бот був представлений ще до того, як був розроблений перший персональний комп'ютер. Він отримав назву Eliza і був розроблений у лабораторії штучного інтелекту MIT Йозефом Вайзенбаумом у 1966 році.

Еліза видавала себе за психотерапевта. Еліза перевірила ключові слова у введених користувачами даних і запустила правила перетворення результату. Ця конкретна методологія генерації відповідей все ще широко використовується при створенні чат-ботів. Після Елізи «Перрі» написав психіатр Кеннет Колбі, який тоді працював у Стенфордському університеті, намагаючись змоделювати людину з параноїдальною шизофренією.

ALICE, або просто Alicebot, спочатку був розроблений Річардом Воллесом у 1995 році та був натхненний Елізою. Хоча він не пройшов тест Тьюринга, ALICE залишався одним із найсильніших у своєму роді та тричі був нагороджений премією Лебнера, щорічним конкурсом ШІ.

У першому десятилітті 21 століття SmarterChild був створений ActiveBuddy. Це була перша спроба створити чат-бота, який міг би не лише забезпечувати розваги, але й надавати користувачам більше корисної інформації, такої як інформація про акції, спортивні результати, цитати з фільмів та багато іншого. Він існував у AOL і Windows Live Messenger, ним користувалися понад 30 мільйонів людей. Пізніше він був придбаний Microsoft у 2007 році за 46 мільйонів доларів. SmarterChild є попередником Siri від Apple і S Voice від Samsung [1].

Siri — це інтелектуальний особистий помічник, який був розроблений як побічний проект компанією SRI International, а пізніше застосований Apple у своїй iOS 5 для iPhone. Це невід'ємна частина екосистеми iOS. Siri дозволяє користувачам брати участь у випадкових розмовах, надаючи корисну інформацію про погоду, акції та квитки в кіно. Такі технічні гіганти, як Samsung і Google, також пішли по стопах Apple, розробивши своївласні помічники AI, S Voice і Google Allo відповідно.

Є також голосові домашні помічники, такі як Amazon Alexa та Google Home, які є іншим представником чат-ботів.

Останні розробки чат-ботів

З огляду на історію, компанії завжди створювали власні індивідуальні чат-боти на основі штучного інтелекту, щоб служити цілям своїх кінцевих користувачів. Останніми роками ця тенденція змінилася: у червні 2015 року Telegram відкрив свою бот-платформу, дозволяючи розробникам створювати чат-ботів, які обслуговують користувачів із численними послугами, такими як опитування, новини, ігри, інтеграція та розваги. Крім того, у грудні 2015 року Slack, програмне забезпечення для крос-платформної командної співпраці, оголосило про використання ботів. Запуск Slack своєї платформи для ботів став каталізатором, який спонукав інші компанії почати інвестувати в цей новий канал залучення користувачів.

Як один із найбільших гравців на цьому ринку, Facebook випустив свою платформу Messenger у квітні 2016 року під час конференції розробників F8. Хоча Facebook трохи запізнився на вечірку, це найбільше вплинуло на галас чат-ботів. Велику роль у цьому відіграла можливість охопити 1 мільярд активних користувачів через Messenger.

Щоб назвати ще кілька, Skype, Kik і WeChat є іншими основними гравцями в обміні повідомленнями, які випустили свої платформи для розробників для публікації чат-ботів.

Підводячи підсумок, якщо уявити собі подорож чат-ботів із 1960-х років до сьогодні, то можна побачити, що те, що колись було фантазією про можливість спілкування з неживою віртуальною істотою, тепер є частиною нашого повсякденного життя.

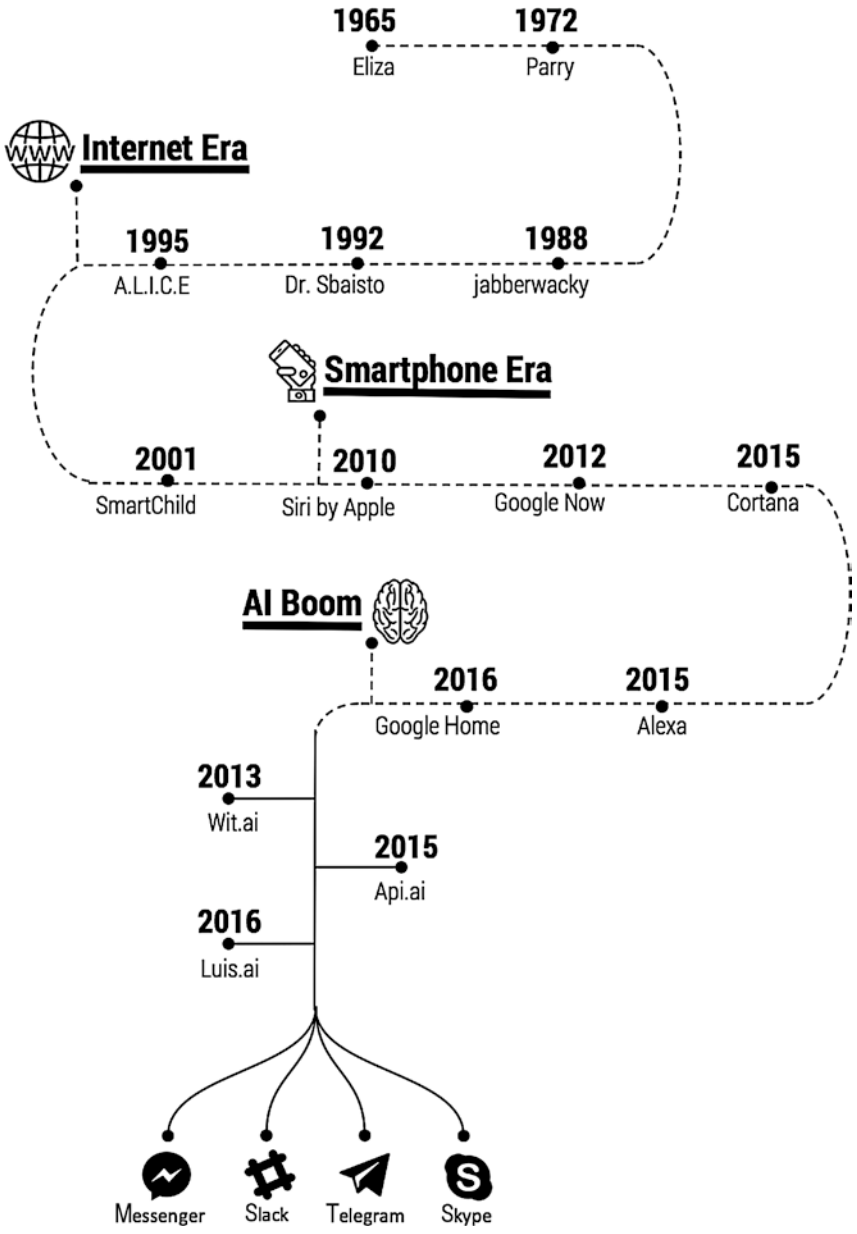

Рисунок 1.1 - Розквіт чат-ботів

Останнім часом чат-боти стали модним словом, і багато людей думають, що це через ажіотаж щодо штучного інтелекту, створений Facebook, який відкрив

свою платформу Messenger для розробників для створення ботів. Може здатися, що чат-боти стали сенсацією за дуже короткий проміжок часу, але насправді це поєднання різних факторів, які відбулися з початку 2000-х до сьогодні [2].

У цьому розділі ми розглянемо фактори, які сприяли нещодавньому зростанню чат-ботів, і зрозуміємо, який у цьому сенс. Щоб дати вам зрозуміти, куди рухаються чат-боти, чимало незалежних дослідників прогнозують, що до кінця 2017 року приблизно одна третина запитів у службу підтримки потребуватиме певного втручання людини, а решта дві третини будуть повністю обробляється системами ШІ.

Зростання кількості користувачів Інтернету

Кількість людей, які використовуютьІнтернет у 2000 році становив 300 мільйонів(www.internetworldstats.com/emarketing.htm). У 2017 році ця кількість зросла до 3,7 мільярда (www.internetworldstats.com/emarketing.htm). Застосування Інтернету зростає на 49,6 відсотка, і оскільки щодня все більше людей підключаються до мережі, потужність Інтернету зростає. Зросла не тільки кількість людей, які користуються Інтернетом, але й час, проведений усіма в Інтернеті, зростає. Дорослі проводять в Інтернеті в середньому близько 28 годин на тиждень, збираючи інформацію, спілкуючись із друзями в соціальних мережах або просто споживаючи мультимедійний контент. Із зростанням використання та збільшенням кількості людей Інтернет за оцінками, згенерував близько 1,2 мільйона терабайт даних (1 терабайт дорівнює 1000 гігабайтам). 2007 рік ознаменував появу великих даних, що означає, що існує багато даних, які можна видобути для пошуку інформації, і інструменти для цього все ще активно розробляються великими підприємствами по всьому світу. Одним із ключових компонентів інтелектуального чат-бота є доступ до даних, які можна використовувати для відповідей на запити користувачів.

Щоб чат-бот був успішним, до нього має мати доступ багато людей. В Інтернеті є кілька платформ, які можуть збільшити такі цифри. За місяць

послугами Facebook скористалися понад 1,7 мільярда людей і швидко усвідомили потенціал обміну бізнес-повідомленнями за допомогою чат-ботів.

Технологічний прогрес

Усі дані, які щодня генеруються користувачами Інтернету, виявляться марними, якщо не буде доступних інструментів для використання даних у навчальних цілях. За останні кілька років було благо для сфери машинного навчання та штучного інтелекту. На початку 2000-х років сфера машинного навчання розвивалася з додаванням глибокого навчання, яке допомагає комп'ютерним машинам «бачити» та розуміти речі в тексті, зображеннях, аудіо, або відео. Провідні технологічні компанії підштовхнули розвиток ШІ, щоб використовувати потужність дешевих обчислень для вирішення складних проблем. Ми стали свідками того, що показник надійності алгоритмів машинного навчання був достатньо високим, щоб їх можна було розгортати в виробниче середовище, де досвід реальних користувачів покращується завдяки використанню цих послуг; це стало можливим завдяки наявності великих наборів даних.

Перехід теоретичних завдань машинного навчання до практичної реалізаціїдопоміг інтернет-компаніям використовувати машинне навчання для розвитку свого бізнесу [5].

Усі провідні світові технологічні компанії зробили свій внесок у те, щоб зробити алгоритми машинного навчання відкритими для будь-кого, щоб використовувати та створювати цікаві програми. TensorFlow із відкритим кодом Google як програмне забезпечення та хмарний сервіс, який був важливою віхою в машинному навчанні, оскільки він забезпечив можливості машинного навчання, які могли використовувати всі, хто має базове розуміння програмування. Інші компанії просуваються в тому ж напрямку, щоб зробити машинне навчання доступним для всіх. Наприклад, Microsoft Azure запустила платформу даних/машинного навчання у своїй хмарній пропозиції, а Amazon додала моделі

машинного навчання у свою хмарну пропозицію AWS. Netflix започаткував культуру конкуренції розробників, створивши моделі, які дають кращу впевненість, ніж алгоритми Netflix, для пропозицій фільмів. Kaggle взяв ідею від Netflix і перетворив себе на платформу машинного навчання для початківців розробників, щоб вчитися на існуючих великих наборах даних і створювати потужні висновки.

Екосистема розробника

У 2003 році в Сполучених Штатах було близько 670 000 розробників, а до 2013 року це число зросло до 1 мільйона розробників. З 2003 до 2013 року кількість робочих місць у розробниках програмного забезпечення зросла на 50 відсотків. Спільнота розробників зростає експоненціально просуває екосистему програмного забезпечення з відкритим кодом, щоб допомогти розробити або вдосконалити існуючі інструменти та фреймворки для розробників. Удосконалення та легка доступність інструментів і фреймворків призвели до швидкої розробки додатків, що є ключовим компонентом для легкого випробування нових ідей і швидкого зазнавання невдач. Екосистема API розвивалася протягом останнього десятиліття, і сьогодні цілком можливо отримати API для будь-якої області застосування, починаючи від інформації про погоду і закінчуючи критичними медичними даними.

Тепер розробники можуть створювати чат-боти, які розуміютьприродною мовою з легкістю. Коли чат-бот розуміє, що сказав користувач, він отримує необхідне інформації за допомогою API або пошуку в базі даних. Поточна екосистема розробників і API виявляється золотою. Розробників, які створюють чат-боти, стимулюють, щоб мати можливість генерувати прибуток для покриття витрат на розробку, а Facebook, Skype, Slack, веб-сайти та мобільні програми формують платформу, на якій розробники можуть розгортати свої чат-боти [7].

Платформи обміну повідомленнями

Чат-боти потрапили в центр уваги завдяки двом гравцям: Facebook і Slack. Оскільки Telegram відкрив свій додаток для розробників для створення та розгортання ботів у червні 2015 року, Facebook оголосив про чат-ботів на своїй платформі під час конференції розробників F8 у квітні 2016 року, що викликало інтерес у розробників у всьому світі. Усі популярні платформи обміну повідомленнями надають розробникам величезну базу споживачів, яку можна використовувати для надання різноманітних послуг через чат.

1.2 Аналіз програмних рішень аналогів

У сучасному світі боти стали не лише популярними в месенджерах, а й в автомобільній індустрії. Боти у автомобільних сервісах відкривають нові можливості для власників автомобілів, дозволяючи легко та зручно керувати ремонтом та обслуговуванням свого транспортного засобу. Ці боти часто пропонують широкий спектр послуг, включаючи діагностику, технічне обслуговування, ремонтні роботи, а також можливість запланувати візит до сервісного центру через месенджер.

Завдяки ботам у автомобільних сервісах власники автомобілів можуть ефективно керувати технічним станом свого транспортного засобу, отримуючи рекомендації щодо необхідного ремонту, оцінки вартості послуг та можливість записатися на прийом до сервісного центру в зручний для них час.

Ці боти також спрощують взаємодію з сервісним центром, забезпечуючи швидкий та ефективний спосіб зв'язку для питань щодо технічного стану автомобіля та організації обслуговування.

Одними з існуючих наразі рішень є телеграм-боти, такі як:

- CarFixBot
- AutoServiceBot
- CarDoctorBot
- AutoRepairBot

Таблиця 1.1 – Порівняння існуючих рішень

	CarFixBot	AutoServiceBot	CarDoctorBot	AutoRepairBot
Особливості	Надає широкий спектр послуг з ремонту автомобілів, включаючи діагностику, технічне обслуговування та ремонт. Пропонує можливість запису на прийом та отримання оцінок.	Пропонує різноманітні послуги з ремонту та технічного обслуговування автомобілів, включаючи діагностику, зміну масла та ротацію шин.	Надає діагностику, рекомендації з ремонту та запис на прийом.	Пропонує послуги з ремонту та технічного обслуговування автомобілів, можливо, включаючи діагностику, заміну деталей та заплановане обслуговування.

Продовження таблиці 1.1 – Порівняння існуючих рішень

Інтерфейс користувача	Простий інтерфейс для зручної взаємодії, можливо, з опціями для легкої навігації.	Простий інтерфейс для легкої взаємодії, можливо, з командами у чаті.	Інтерфейс, призначений для легкої взаємодії, з командами у чаті або керованими підказками.	Інтерфейс, розроблений для зручного використання, з командами у чаті або варіантами меню.

Таким чином, було розглянуто поняття чат-ботів, а також проаналізовано існуючі аналогічні рішення на ринку.

1.3. Мета та задачі дослідження

Мета дослідження - це розробка ефективного та функціонального телеграм-боту для автомобільного сервісу з метою поліпшення якості обслуговування клієнтів, забезпечення швидкого доступу до інформації про послуги, а також оптимізації робочих процесів у сервісному центрі.

Ознайомившись з предметною областью було посталено наступні задчі для реалізації:

1. Розглянути актуальні дослідження в галузі
2. Проаналізувати існуючі рішення
3. Моделювання роботи бота в нотаціях
4. Розробити прогрмане рішення
5. Протестувати програмний продукт

Основою для розробки нашого проекту буде мова програмування Python, оскільки вона є однією з найпопулярніших мов у програмуванні. Python відзначається своєю простотою і лаконічністю синтаксису –це робить його ідеальним вибором для розробників різного рівня підготовки. Зокрема, Python часто обирають для створення телеграм-ботів, і на це є кілька вагомих причин.

Однією з бібліотек для цієї мети є python-telegram-bot. Вона дає зручний інтерфейс для взаємодії з Telegram API, а це дозволяє розробникам швидко і ефективно налаштовувати ботів. Завдяки цій перевазі Python є найкращим варіантом для розробки телеграм-боту.

Відносно бази даних – для цього проекту було обрано SQLite. Для її роботи не потрібна необхідність в окремому сервері. Всі данні будуть зберігатися у одному файлі. При умові навіть великих обсягів даних вона забеспечую швидкий доступ до інформації. Оскільки у нашому проекті буде інтенсивно використовуватись база даних та до неї буде висока частота запитів , тому вона нам просто необхідна для використання.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ

2.1 Аналіз мов програмування

Python - це високорівнева, інтерпретована, динамічна мова програмування, яка здатна вирішувати різноманітні завдання, від простих сценаріїв до складних веб-додатків та наукових обчислень. Ось детальний огляд переваг і недоліків використання Python:

Переваги Python:

Простий синтаксис: Python має зрозумілий і простий для вивчення синтаксис, що робить його ідеальним вибором для початківців та професіоналів.

Широка підтримка спільноти: Python має велику та активну спільноту, яка регулярно розвиває бібліотеки та фреймворки, що полегшують розробку програм.

Багато бібліотек: Python має велику екосистему бібліотек для різних цілей, таких як робота з базами даних, обробка даних, машинне навчання, веб-розробка тощо.

Портативність: Python підтримується на багатьох платформах, включаючи Windows, macOS і різні дистрибутиви Linux, що робить його універсальним і доступним для розробників.

Швидкість розробки: Python дозволяє розробникам швидко прототипувати та впроваджувати програми завдяки своїй простоті та зручному синтаксису.

Недоліки Python:

Виконання на великих об'ємах даних: У порівнянні з деякими іншими мовами програмування, такими як C++ або Java, Python може бути повільним у виконанні на великих об'ємах даних.

Неідеальна підтримка паралельного програмування: Хоча Python має модулі для паралельного програмування, такі як multiprocessing та threading, вони не завжди ефективні через особливості інтерпретатора Python.

Не підходить для деяких задач високої продуктивності: У деяких областях, де вимагається висока продуктивність, таких як вбудовані системи або ігри, Python може бути не найкращим вибором через свою інтерпретовану природу.

Незважаючи на ці недоліки, Python залишається однією з найпопулярніших мов програмування завдяки своїм перевагам у сфері швидкості розробки, доступності та потужності.

JavaScript (JS) - це мова програмування, яка використовується для розробки веб-додатків, включаючи фронтенд і бекенд, а також для створення мобільних додатків та додатків для IoT (Internet of Things). Ось детальний огляд переваг і недоліків використання JavaScript:

Переваги JavaScript:

Веб-додатки та інтерактивність: JavaScript використовується для створення інтерактивних елементів веб-сторінок, таких як анімація, форми, ігри та інші візуальні ефекти.

Широка підтримка браузерами: JavaScript підтримується всіма сучасними браузерами, що робить його ідеальним для розробки веб-додатків, які працюють на будь-якому пристрої.

Node.js для серверного програмування: JavaScript може бути використаний для розробки серверних додатків за допомогою платформи Node.js, що дозволяє використовувати той же код як на фронтенді, так і на бекенді.

Багатофункціональність: JavaScript є мовою високого рівня, яка має багато вбудованих функцій і бібліотек для роботи з різними типами даних, включаючи рядки, масиви, об'єкти тощо.

Широке використання: JavaScript використовується не лише для веб-розробки, але й для мобільних додатків (за допомогою фреймворків, таких як React Native або Ionic) та для розробки додатків для IoT.

Недоліки JavaScript:

Перенесення помилок на сторону клієнта: Оскільки JavaScript виконується на стороні клієнта, помилки в коді можуть призвести до некоректної роботи веб-сторінки або додатку.

Інтерпретована мова програмування: JavaScript є інтерпретованою мовою, що може призвести до меншої швидкодії в порівнянні з компільованими мовами програмування.

Крос-браузерна сумісність: Існує проблема крос-браузерної сумісності, коли деякі функції JavaScript можуть працювати по-різному на різних браузерах.

Безпека: JavaScript може бути вразливим до атак, таких як внедрення коду (injection) або cross-site scripting (XSS) атаки, якщо не враховувати правила безпеки програмування.

Хоча JavaScript має свої недоліки, він залишається однією з найбільш популярних мов програмування завдяки своїм перевагам у сфері веб-розробки, доступності та розширюваності.

PHP (Hypertext Preprocessor) - це мова програмування, яка використовується для розробки веб-додатків та веб-сайтів. Ось детальний огляд переваг і недоліків використання PHP:

Переваги PHP:

Простота вивчення: PHP має простий і зрозумілий синтаксис, що робить його досить легким для вивчення, навіть для початківців.

Широке використання: PHP є однією з найпоширеніших мов програмування для веб-розробки і використовується для створення різних типів веб-додатків, від простих блогів до великих електронних комерційних платформ.

Багатофункціональність: PHP має велику кількість вбудованих функцій та розширень, що полегшує роботу з базами даних, обробкою форм, створенням зображень тощо.

Зручність інтеграції: PHP добре інтегрується з різними СУБД, такими як MySQL, PostgreSQL, SQLite тощо, що дозволяє розробникам легко працювати з базами даних.

Велика спільнота: PHP має велику та активну спільноту розробників, що забезпечує широкий доступ до документації, форумів підтримки та багато інших ресурсів.

Недоліки PHP:

Безпека: PHP має деякі вразливості, такі як вразливості типу переповнення буфера та вразливості SQL-ін'єкції, що можуть стати причиною атак на веб-додатки.

Низька продуктивність: У порівнянні з деякими іншими мовами програмування, такими як Java або C++, PHP може мати меншу продуктивність та швидкодію, особливо для великих та складних додатків.

Неявна типізація: PHP має слабку підтримку статичної типізації, що може призвести до непередбачуваного поведінки програми та помилок.

Нестабільність функціоналу: Історично PHP мала проблеми зі стабільністю деяких функцій та розширень, що може призвести до проблем при оновленні або міграції додатків на нові версії PHP.

Хоча PHP має свої недоліки, вона залишається однією з найпопулярніших мов програмування для веб-розробки завдяки своїм перевагам у сфері доступності, багатофункціональності та широкому використанню.

Ruby - це динамічна, об'єктно-орієнтована мова програмування, яка була створена в Японії в 1995 році Юкіхіро Мацумото. Ось детальний огляд переваг і недоліків використання Ruby:

Переваги Ruby:

Простий та читабельний синтаксис: Ruby має природній та зрозумілий синтаксис, що робить код легким для читання та розуміння. Це сприяє швидкому розвитку та підтримці проектів.

Об'єктно-орієнтованість: Ruby повністю об'єктно-орієнтована мова, де все є об'єктом. Це спрощує роботу з даними та створення більш структурованих програм.

Багата бібліотека: Ruby має велику кількість готових бібліотек та фреймворків, які значно полегшують розробку різноманітних додатків, включаючи веб-додатки, мобільні додатки та інші.

Ruby on Rails: Ruby має відомий та потужний веб-фреймворк Ruby on Rails, який прискорює розробку веб-додатків та надає велику кількість зручних інструментів та бібліотек для роботи з базами даних, маршрутизації, аутентифікації тощо.

Динамічна типізація: Ruby має динамічну типізацію, що означає, що змінні не повинні бути явно оголошені з типом даних, що полегшує написання коду та зменшує кількість помилок.

Недоліки Ruby:

Швидкодія: У порівнянні з деякими іншими мовами програмування, такими як C++ або Go, Ruby може бути повільним виконуваним мовом через його інтерпретований характер.

Ресурсомісткість: Ruby може вимагати багато ресурсів системи, що може призвести до проблем з продуктивністю на великих проектах або під навантаженням.

Крос-платформенність: Хоча Ruby підтримується на багатьох платформах, розгортання додатків на Windows може викликати проблеми через різницю у середовищах [15].

Помірна популярність: Хоча Ruby має велику та віддану спільноту, вона може бути менш популярною, ніж інші мови програмування, що може вплинути на доступність ресурсів та документації.

Хоча Ruby має свої недоліки, вона залишається потужним та популярним інструментом для розробки різноманітних додатків, зокрема веб-додатків, завдяки своїм перевагам у сфері простоти, гнучкості та багатофункціональності.

Go, також відома як Golang, - це мова програмування, розроблена в компанії Google, яка поєднує в собі ефективність компілюваних мов, таких як C або C++, з простотою і сучасністю скриптових мов програмування. Ось детальний огляд переваг і недоліків використання Go:

Переваги Go:

Швидкість виконання: Go володіє вражаючою швидкістю виконання завдяки своїй компілюваній природі, що робить його ідеальним вибором для високонавантажених додатків та сервісів.

Ефективність використання ресурсів: Go відомий своєю ефективністю використання ресурсів системи, що дозволяє економити пам'ять і CPU ресурси на великих проектах.

Простий синтаксис: Синтаксис Go простий та легкий для вивчення, що робить його популярним серед початківців та досвідчених розробників.

Конкурентність: Go має вбудовану підтримку конкурентності, що дозволяє легко створювати паралельні процеси та оброблювати великий обсяг одночасних запитів.

Стандартна бібліотека: Go має потужну стандартну бібліотеку, яка включає в себе багато корисних інструментів для роботи з мережами, шифруванням, обробкою JSON і багато іншого.

Недоліки Go:

Брак обратної сумісності: З часом Go може змінюватися, що може призвести до проблем з обратною сумісністю між версіями, особливо для великих проектів.

Молода екосистема: Хоча Go має швидко зростаючу спільноту та екосистему, вона може бути менш розвиненою, ніж у деяких інших мов програмування.

Неідеальна підтримка generics: У Go не було повноцінної підтримки generics до версії 1.18, що може призвести до більш складного коду у деяких випадках.

Менша кількість бібліотек: Незважаючи на швидке зростання, екосистема Go все ще може мати менше бібліотек порівняно з іншими мовами програмування.

Не зважаючи на ці недоліки, Go залишається потужним та ефективним інструментом для розробки різноманітних додатків, особливо у сферах високонавантажених систем та обробки одночасних запитів.

2.2 Аналіз баз даних

MySQL - це одна з найпопулярніших відкритих систем управління базами даних (СУБД), яка широко використовується для зберігання та керування реляційними базами даних. Ось детальний огляд особливостей MySQL:

1. Надійність та швидкість:

MySQL відома своєю високою надійністю та швидкодією, що робить її популярним вибором для великих веб-сайтів, систем управління контентом та корпоративних додатків.

Вона оптимізована для роботи з великим обсягом даних та високими навантаженнями, що дозволяє їй ефективно обробляти тисячі запитів за короткий час.

2. Розширюваність:

MySQL підтримує різні методи розширення, включаючи горизонтальне та вертикальне масштабування, реплікацію та розподілені транзакції.

Це дозволяє розгортати бази даних MySQL на серверах з великою кількістю ядер та пам'яті для забезпечення високої продуктивності та доступності.

3. Функціональність:

MySQL має широкий набір функцій, включаючи підтримку транзакцій, індексацію даних, зберігані процедури, тригери, представлення, операції з JSON, регулярні вирази та інші.

Вона підтримує різні типи даних, такі як числа, рядки, дати, часи, зображення, географічні дані, що робить її універсальним рішенням для різних типів додатків.

4. Легкість використання:

MySQL має простий та зрозумілий SQL-синтаксис, який робить легкою роботу з базою даних для розробників усіх рівнів.

Вона підтримує велику кількість інструментів адміністрування, таких як phpMyAdmin, MySQL Workbench, що полегшує управління базою даних.

5. Відкритість та спільнота:

MySQL - це відкрита СУБД з активною спільнотою розробників, яка надає регулярні оновлення, виправлення помилок та нові функції.

Вона має широку документацію та велику кількість ресурсів для навчання та підтримки.

В цілому, MySQL є потужним, надійним та широко використовуваним рішенням для зберігання та управління реляційними базами даних у різних типах програм.

PostgreSQL - це потужна об'єктно-реляційна система управління базами даних (СУБД), яка відкрита, гнучка та розширювана. Ось детальний огляд особливостей PostgreSQL:

1. Надійність та стабільність:

PostgreSQL відома своєю високою надійністю та стабільністю. Вона має вбудовану підтримку транзакцій, журналізації та відновлення після збоїв, що робить її ідеальним рішенням для критичних застосунків.

2. Розширюваність:

PostgreSQL підтримує різні методи розширення, включаючи горизонтальне та вертикальне масштабування, реплікацію, розподілені транзакції та розподілені обробки запитів.

Це дозволяє гнучко налаштовувати та розгортати бази даних для відповіді на різні потреби та обсяги даних.

3. Функціональність:

PostgreSQL має багатий набір функцій, включаючи підтримку тригерів, збережених процедур, виразів, віконних функцій, географічних типів даних, операцій з JSON та інші.

Вона підтримує також різні типи даних, такі як рядки, числа, дати, часи, зображення, географічні дані тощо.

4. Сумісність:

PostgreSQL підтримує стандарт SQL ANSI, що робить її сумісною з багатьма іншими СУБД і дозволяє легко переносити додатки з однієї системи на іншу.

Вона також підтримує різні рівні ізоляції транзакцій, включаючи серіалізовану ізоляцію, що забезпечує високий рівень консистентності даних.

5. Розвиток та спільнота:

PostgreSQL має активну та велику спільноту розробників, яка надає регулярні оновлення, виправлення помилок та нові функції.

Вона має докладну документацію, різноманітні ресурси для навчання та підтримки, а також різноманітні розширення та допоміжні інструменти [16].

Узагальнюючи, PostgreSQL є потужною та надійною СУБД, яка підходить для різних типів додатків та завдань. Вона забезпечує широкий набір функцій, високу продуктивність та надійність, що робить її відмінним вибором для великих та критичних застосунків.

MongoDB - це документоорієнтована система управління базами даних (NoSQL), яка використовує JSON-подібні документи для зберігання даних. Ось детальний огляд особливостей MongoDB:

1. Схема документів:

MongoDB використовує гнучку схему документів, що дозволяє зберігати дані у вигляді JSON-подібних об'єктів, які можуть мати різні поля та структури.

Це дозволяє легко змінювати структуру даних та додавати нові поля без необхідності зміни схеми бази даних.

2. Горизонтальне масштабування:

MongoDB підтримує горизонтальне масштабування, що дозволяє розгортати бази даних на кількох серверах та автоматично розподіляти навантаження між ними.

Це дозволяє підтримувати великі обсяги даних та високу доступність в розподілених середовищах.

3. Швидкодія та продуктивність:

MongoDB відома своєю високою швидкодією та продуктивністю завдяки використанню індексації даних, кешування та іншим оптимізаціям.

Вона також підтримує запити до бази даних у форматі мови запитів MongoDB (MongoDB Query Language), яка дозволяє швидко та ефективно отримувати дані.

4. Гнучкість та розширюваність:

MongoDB має широкий набір функцій, таких як підтримка транзакцій, агрегаційних функцій, текстового пошуку, геопросторових операцій, операцій з масивами та інші.

Вона також підтримує різні формати даних, такі як документи, масиви, геодані та інші, що дозволяє зберігати різнорідні дані.

5. Спільнота та інструменти:

MongoDB має велику та активну спільноту розробників, яка надає регулярні оновлення, виправлення помилок та нові функції.

Вона має також широкий вибір інструментів для адміністрування, моніторингу та розробки, таких як MongoDB Compass, MongoDB Atlas, MongoDB Shell та інші.

Узагальнюючи, MongoDB є потужною та гнучкою системою управління базами даних, яка підходить для різних типів додатків та завдань. Вона надає широкий набір функцій, високу швидкодію та надійність, що робить її відмінним вибором для розробки сучасних додатків.

Redis - це швидка та потужна система управління даними типу ключ-значення, яка часто використовується для кешування, сесій та повідомлень у веб-додатках. Ось детальний огляд особливостей Redis:

1. Простота та ефективність:

Redis має простий та легкий у використанні інтерфейс команд, який дозволяє швидко зберігати та отримувати дані у форматі ключ-значення.

Вона працює у пам'яті, що дозволяє отримувати швидкий доступ до даних та виконувати операції з ними з великою швидкодією.

2. Типи даних:

Redis підтримує різні типи даних, включаючи рядки, хеші, списки, множини та сортовані множини.

Це дозволяє зберігати різнорідні дані та використовувати їх для різних цілей, від кешування до обробки потоків даних.

3. Висока доступність:

Redis підтримує реплікацію та шардування, що дозволяє розгортати кластери Redis на кількох серверах та забезпечувати високу доступність та надійність.

Вона також має вбудовану підтримку механізму моніторингу та автоматичного відновлення, що дозволяє швидко виявляти та виправляти збої.

4. Підтримка транзакцій:

Redis підтримує транзакції, що дозволяє виконувати групу команд атомарно та забезпечувати консистентність даних.

Вона також підтримує різні операції над множинами даних, такі як об'єднання, перетин, різниця тощо.

5. Гнучкість та розширюваність:

Redis має широкий вибір розширень та модулів, які дозволяють розширювати його функціональність та використовувати його для різних цілей.

Вона також має велику спільноту розробників, яка надає різноманітні інструменти та бібліотеки для роботи з Redis.

Узагальнюючи, Redis є потужним та ефективним інструментом для кешування, сесій та повідомлень у веб-додатках. Вона надає швидку доступність до даних, гнучкість та високу доступність, що робить її відмінним вибором для великих та складних проектів.

SQLite - це легка, портативна та самодостатня двійкова файлова база даних, яка зберігається у одному файлі на локальному пристрої. Вона широко використовується в різних сферах програмування, включаючи мобільні додатки, веб-розробку та інші області, де потрібно легко вбудовувати базу даних без необхідності установки окремого сервера баз даних.

Однією з основних переваг SQLite є його простота використання. Встановлення та налаштування SQLite дуже просте, і його можна легко інтегрувати у проекти будь-якої складності. Він не вимагає запуску окремого

сервера баз даних, що робить його ідеальним варіантом для локального зберігання невеликих обсягів даних.

SQLite також має добру швидкодію. Оскільки база даних зберігається у вигляді одного файлу на диску, доступ до даних є дуже ефективним. Це особливо важливо для мобільних додатків, де обмежені ресурси пристрою можуть впливати на продуктивність.

Крім того, SQLite підтримує багато стандартних функцій SQL, що дозволяє виконувати різноманітні операції з даними, такі як вибірка, вставка, оновлення та видалення. Це робить його дуже потужним інструментом для роботи з базами даних у різних програмних середовищах.

Загалом, SQLite є важливим інструментом для розробників програмного забезпечення, які потребують простої та ефективної бази даних для своїх проектів. Він забезпечує широкі можливості для зберігання та обробки даних у різних програмних середовищах, дозволяючи розробникам створювати потужні та ефективні програми.

Таким чином, для розробки програмного рішення було обрано мову програмування Python та БД SQLite.

2.3 Моделювання діаграм діяльності

Діаграма використання (use case diagram) - це тип діаграми в аналізі та проектуванні програмного забезпечення, який допомагає моделювати взаємодію між системою та її користувачами або зовнішніми системами. Головна мета діаграми використання - показати, які дії (використовуючи варіанти використання) можуть виконувати різні актори (користувачі або системи) в межах системи.

На діаграмі використання кожен варіант використання представлений у вигляді овалів, що називаються «використанням». Актори, які взаємодіють з системою, зображуються у вигляді піктограм зовнішнього об'єкта (наприклад, людина або інша система).

Діаграма використання допомагає створити загальне уявлення про те, як система має взаємодіяти зі своїми користувачами та іншими системами. Вона дозволяє ідентифікувати основні функціональні можливості системи та визначити, як вони пов'язані з акторами. Це зручний інструмент для аналізу вимог до системи та уточнення їх перед розробкою програмного забезпечення.

Під час проектування додатку було розроблено наступну діаграму використання:

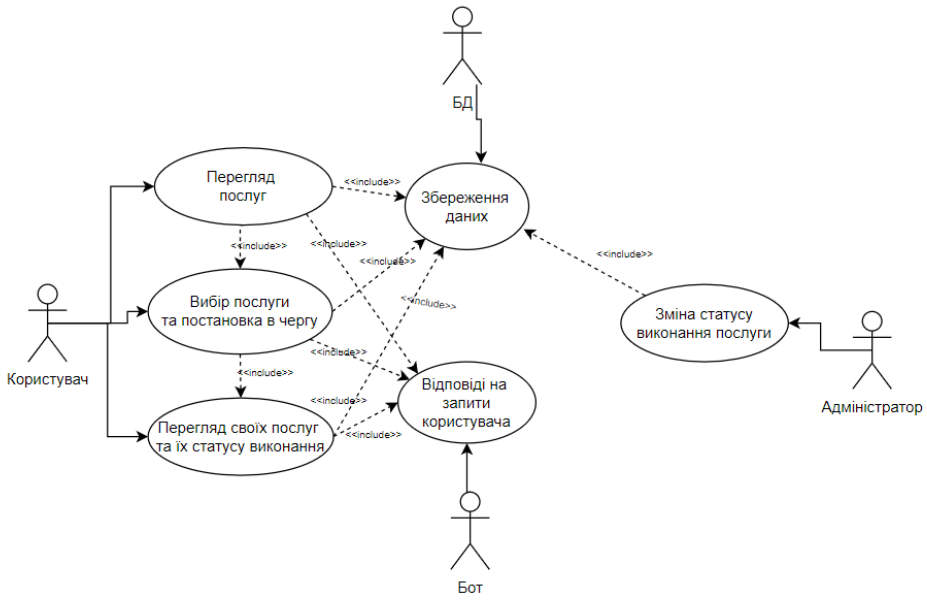

Рисунок 2.1 – Діаграма використання

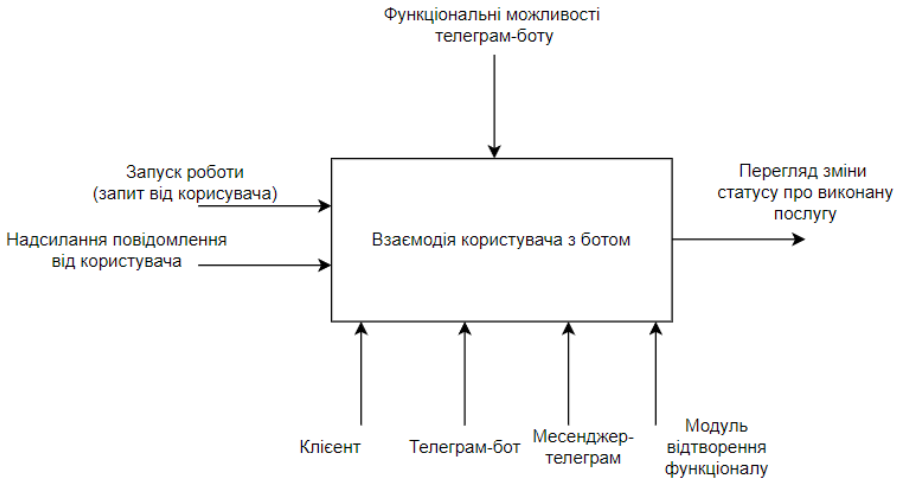

Рисунок 2.2 – Діаграма IDEF0. Концептуальний рівень

Для діаграми визначені наступні дані:

Вхідні: запуск роботи, надсиланя повідомлення;

Управління: функціональні можливості телеграм-боту;

Механізми: клієнт, телеграм-бот, месенджер-телеграм, модуль відтворення функціоналу;

Вихідні: перегляд змін статусу про виконану послугу;

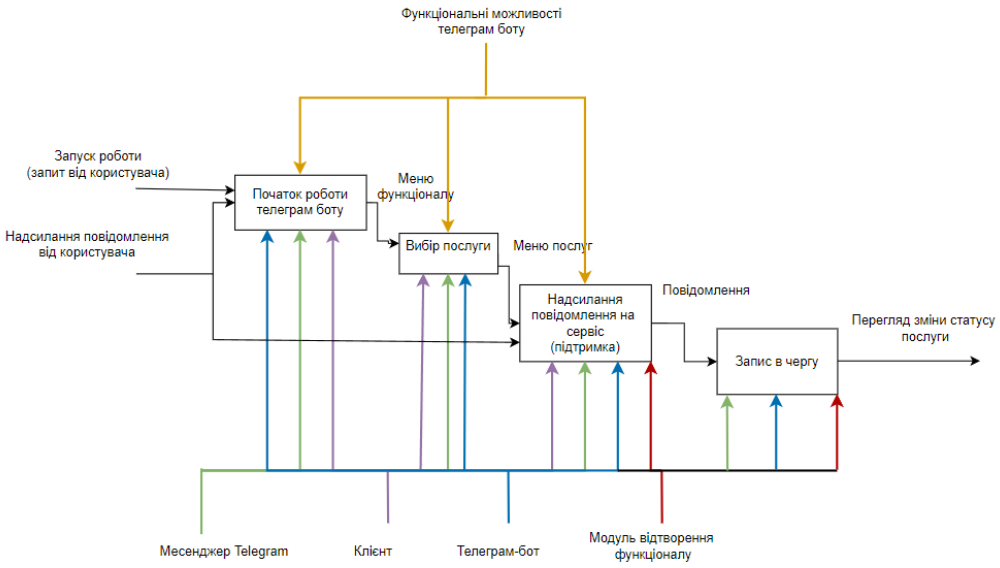

Рисунок 2.3 – Декомпозиція внутрішніх потоків

На Рисунку 2.3 показано діаграму декомпозиції внутрішніх потоків , які розроблені для деталізації попередньої діаграми , а саме – IDEF0.

2.5 Проектування бази даних

ER-діаграма - це графічне зображення сутностей і зв'язків між ними у базі даних. Це один з ключових інструментів при моделюванні баз даних, який дозволяє візуалізувати структуру і взаємозв'язки між об'єктами даних.

У ER-діаграмі сутності представлені у вигляді прямокутників, а зв'язки між ними - у вигляді ліній з позначками, що показують, як сутності пов'язані між собою. Кожна сутність має атрибути, які визначають її властивості. Зв'язки між сутностями вказують на те, як вони взаємодіють одна з одною і яка їхньою природою є ця взаємодія: один до одного, один до багатьох або багато до багатьох.

ER-діаграми допомагають при розробці та розумінні структури баз даних, їх використовують для аналізу вимог, проектування бази даних, оптимізації та відлагодження. Вони є важливим інструментом для комунікації між аналітиками, розробниками програмного забезпечення та замовниками при роботі над проектами баз даних.

Розробляючи схему бази даних, було створено наступну діаграму:

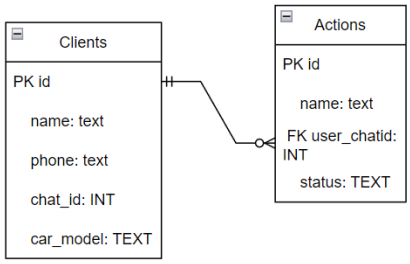

Рисунок 2. – ER-діаграма

На даній діаграмі показано лише дві таблиці: Clients, Actions. До першої відносяться такі дані: name, phone, chat_id, car_model. До дргуої такі дані: name,user_chatid, status. Перша таблиця була створена з метою запису клієнтів , друга – для запису послуг клієнта.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Розробка коду програми

```
import telebot
import sqlite3
```

Лістинг 3.1 – Імпортування модулів

Цей фрагмент коду імпортує дві бібліотеки: telebot та sqlite3.

telebot: Ця бібліотека є інтерфейсом для роботи з Telegram Bot API у Python. Вона дозволяє створювати, налаштовувати та запускати Telegram-ботів. Користуючись telebot, ви можете отримувати повідомлення від користувачів, відправляти їм повідомлення, обробляти команди, реагувати на клавіатури та інше.

sqlite3: Ця бібліотека надає інтерфейс для взаємодії з базами даних SQLite в Python. SQLite - це легка вбудована база даних, яка не потребує окремого сервера та може бути використана для збереження та обробки даних в локальних застосунках. За допомогою sqlite3 ви можете створювати, зчитувати, оновлювати та видаляти дані у базі даних SQLite, а також виконувати запити SQL.

Отже, ці дві імпортовані бібліотеки дозволять вам створити Telegram-бота з використанням telebot та взаємодіяти з базою даних SQLite за допомогою sqlite3.

```
# Function to read token from a file
def read_token():
    with open("token.txt", "r") as file:
        return file.read().strip()
```

Лістинг 3.2 – зчитування токену з текстового файлу

Ця функція read_token() призначена для зчитування токену (API token) з файлу token.txt. Ось її пояснення:

open("token.txt", "r"): Функція open() відкриває файл з назвою "token.txt" у режимі читання ("r").

with ... as file: Контекстний менеджер with використовується для автоматичного закриття файлу після завершення роботи з ним. Він створює об'єкт файлу, який називається file, для подальшої роботи з ним всередині блоку with.

file.read().strip(): Метод read() читає вміст файлу, а метод strip() видаляє всі пробіли та символи нового рядка з кінця та початку рядка. Ось як це працює:

file.read(): Цей виклик читає весь вміст файлу.

strip(): Він видаляє пробіли (включаючи символи нового рядка) з початку та кінця рядка.

return file.read().strip(): Зчитаний токен повертається з функції read_token(). Це означає, що будь-який фрагмент коду, який викликає цю функцію, отримає токен для подальшого використання в програмі.

Загалом, ця функція є досить зручним способом отримання токену з файлу, оскільки вона дозволяє зберігати токен у окремому файлі, що полегшує його зміну та уникнення розголошення відкритого тексту.

```
# Initialize the bot with the token read from file
bot = telebot.TeleBot(read_token())
```

Лістинг 3.3 – Ініціалізація боту

Цей рядок коду ініціалізує об'єкт бота (bot) з використанням токену, який зчитується з файлу за допомогою функції read_token(). Ось як це працює:

read_token(): Ця функція викликається для отримання токену з файлу token.txt, який містить API токен для вашого бота.

telebot.TeleBot(...): Конструктор класу TeleBot ініціалізує об'єкт бота для взаємодії з Telegram Bot API. В якості аргументу передається токен, отриманий з файлу за допомогою read_token().

Отже, після виконання цього рядка коду ви матимете створений об'єкт бота, який буде готовий до використання з отриманим токеном для взаємодії з Telegram API.

```
def connect_database():
    conn = sqlite3.connect('clients.db')
    cursor = conn.cursor()
    return conn, cursor
```

Лістинг 3.4 – Підключення бази даних

Ця функція connect_database() створює підключення до бази даних SQLite і повертає об'єкти підключення та курсора. Ось як вона працює:

sqlite3.connect('clients.db'): Функція connect() модуля sqlite3 використовується для підключення до бази даних SQLite. У цьому випадку ми передаємо шлях до бази даних 'clients.db'.

conn.cursor(): Після успішного підключення до бази даних ми створюємо об'єкт курсора, який використовується для виконання операцій бази даних, таких як виконання запитів SQL.

return conn, cursor: Обидва об'єкти, підключення до бази даних (conn) та курсор (cursor), повертаються з функції. Це дозволяє використовувати їх у подальших операціях з базою даних.

Отже, виклик цієї функції дозволить отримати підключення до бази даних та курсор, який можна використовувати для виконання SQL-операцій у вашій програмі.

```
# Function to close database connection
def close_database(conn):
    conn.close()
```

Лістинг 3.5 – Закриття з’єдннаня з БД

Ця функція close_database(conn) призначена для закриття з'єднання з базою даних SQLite. Ось як вона працює:

conn.close(): Цей виклик методу close() зберігає всі зміни, зроблені у базі даних під час роботи з нею, і закриває з'єднання. Після виклику цього методу подальша взаємодія з базою даних буде неможлива, поки з'єднання не буде відновлено.

Функція не повертає жодного значення, оскільки просто виконує дію закриття з'єднання.

Виклик цієї функції із переданим об'єктом з'єднання (conn) допоможе коректно закрити з'єднання з базою даних після того, як ви закінчите роботу з нею.

На Рисунку 3.6 Показано функцію яка створює меню бота із заданих в ній команд

```
def addCommandsMenu(chatId):
    c1 = telebot.types.BotCommand(command='start', description='Registration')
    c2 = telebot.types.BotCommand(command='actions', description='Add action')
    c3 = telebot.types.BotCommand(command='myactions', description='List of my actions')
    c4 = telebot.types.BotCommand(command='support', description='Support')
    c5 = telebot.types.BotCommand(command='admin', description='Admin')
    c6 = telebot.types.BotCommand(command='services', description='services')
    bot.set_my_commands([c1,c2,c3, c4,c5,c6])
    bot.set_chat_menu_button(chatId, telebot.types.MenuButtonCommands('commands'))
```

Рисунок 3.6 – Відображення меню

На Рисунку 3.7 можна спостерігати стоворення дух таблиць: для клієнтів та послуг, також показано їх ініціалізацію.

```python
def clientsTable():
    conn, cursor = connect_database()
    cursor.execute('''CREATE TABLE IF NOT EXISTS clients (
                      id INTEGER PRIMARY KEY AUTOINCREMENT,
                      chat_id INTEGER,
                      name TEXT,
                      phone TEXT,
                      car_model TEXT)''')

    conn.commit()
    close_database(conn)
#Функція створення таблиці та її полів для обраних послуг клієнта(при умові що даної таблиці не існує)
def actionsTable():
    conn, cursor = connect_database()
    cursor.execute('''CREATE TABLE IF NOT EXISTS actions (
                      id INTEGER PRIMARY KEY AUTOINCREMENT,
                      user_chatid INTEGER,
                      name TEXT,
                      status TEXT)''')
    conn.commit()
    close_database(conn)
# ініціалізація таблиць
def startDB():
     clientsTable()
     actionsTable()
```

Рисунок 3.7 – Створення таблиць та їх ініціалізвація

support_chat_id = 123456789: Ця змінна містить ідентифікатор чату, в який будуть переслані повідомлення від користувачів службі підтримки. Його потрібно замінити 183765493 на фактичний ідентифікатор чату служби підтримки при початку роботи , тобто розмежувати користувачів на сервісі хто за що відповідає.

forward_text = f"Message from user [{client[3]}] {client[2]}:\n{message.text}": Цей рядок формує текстове повідомлення, яке буде переслане до служби підтримки. Воно містить ідентифікатор чату користувача та текст його повідомлення.

bot.send_message(support_chat_id, forward_text): Цей виклик методу send_message() відправляє повідомлення до чату служби підтримки із зазначеним текстом. Це повідомлення містить текст, який був отриманий від користувача.

bot.send_message(message.chat.id, "Your message has been forwarded to customer support. They will get back to you soon."): Цей рядок відправляє відповідне повідомлення користувачеві, яке підтверджує, що його повідомлення було переслано до служби підтримки.

Ця функція дозволяє користувачам звертатися до служби підтримки через вашого бота та ефективно обробляти їхні запити. Приклад такої функції можна побачити на рисунку 3.8.

```python
def handle_support_chat(message):
    client = getClientById(message.chat.id)
    if not client:
        bot.send_message(message.chat.id, "Клієнта не знайдено.")
        return
    print(f"Received support message from {message.chat.id}: {message.text}")
    support_chat_id = 644443347  # Replace with your support chat ID
    forward_text = f"Message from user [{client[3]}] {client[2]}:\n{message.text}"
    bot.send_message(support_chat_id, forward_text)
    bot.send_message(message.chat.id, "Your message has been forwarded to customer support. They will get back to you soon.")
```

Рисунок 3.8 – Відправка повідомлення в службу підтримки

Було створено функцію checkUserExist(chatId). Ця функція вивкликається перед виконанням кожної команди для перевірки чи є клієнт в системі, якщо ні тоді команда не буде виконуватись. Це було спеціально зроблено , тому що користувас може запустити бота і почати «клацати» всі команди , а при наявності такої перевірки йому буде надіслано сповіщення про те щоб він зареєструвався в системі , а тільки потім йому будуть доступні всі інші команди.

```python
def checkUserExist(chatId):
    client = getClientById(chatId)
    if not client:
            bot.send_message(chatId, "Ви не зареєстровані. Скористуйтесь командою /start для того щоб почати користуватися ботом.")
            return False
    else:
        return True
```

Рисунок 3.9 – Функція для перевірки

Ця функція handle_start(message) обробляє команду /start та починає процес реєстрації клієнта та його постановки в чергу. Ось як вона працює:

@bot.message_handler(commands=['start']): Цей декоратор вказує, що функція буде обробляти команду /start.

print(f"Received /start command from {message.chat.id}"): Цей рядок виводить в консоль повідомлення про отриману команду /start, разом із ідентифікатором чату користувача, який її надіслав.

bot.send_message(message.chat.id, "Для реєстрації введіть своє ім'я:"): Цей виклик методу send_message() відправляє повідомлення користувачеві з проханням ввести своє ім'я для реєстрації.

bot.register_next_step_handler(message, register_name): Цей виклик методу register_next_step_handler() реєструє функцію register_name як наступний крок для обробки повідомлення від користувача після отримання його імені. Тобто після того, як користувач введе своє ім'я, виконається функція register_name.

Отже, ця функція реалізує перший крок процесу реєстрації клієнта та постановки його в чергу, запрошуючи користувача ввести своє ім'я.

```python
@bot.message_handler(commands=['start'])
def handle_start(message):
    print(f"Received /start command from {message.chat.id}")
    client = getClientById(message.chat.id)
    if not client:
        bot.send_message(message.chat.id, "Для реєстрації введіть своє ім'я:")
        bot.register_next_step_handler(message, register_name)
    else:
        bot.send_message(message.chat.id, "Ви вже зареєстровані")

def register_name(message):
    print(f"{message.chat.id} registered name: {message.text}")
    name = message.text
    bot.send_message(message.chat.id, "Введіть свій номер телефону:")
    bot.register_next_step_handler(message, register_phone, name)

def register_phone(message, name):
    print(f"{message.chat.id} registered phone: {message.text}")
    phone = message.text
    bot.send_message(message.chat.id, "Введіть марку та модель автомобіля:")
    bot.register_next_step_handler(message, register_car_model, name, phone)
```

Рисунок 3.10 – Реєстрація клієнта

```python
def register_car_model(message, name, phone):
    print(f"{message.chat.id} registered car model: {message.text}")
    car_model = message.text
    conn, cursor = connect_database()
    cursor.execute("INSERT INTO clients (chat_id, name, phone, car_model) VALUES (?, ?, ?, ?)",
                   (message.chat.id, name, phone, car_model))
    conn.commit()
    close_database(conn)
    bot.send_message(message.chat.id, "Ви успішно зареєстровані і поставлені у чергу.")
    addCommandsMenu(message.chat.id)
```

Рисунок 3.11 – Реєстрація клієнта(2)

Ці три функції (register_name, register_phone, register_car_model) виконують послідовний процес реєстрації клієнта із збереженням інформації в базі даних та повідомленнями користувачу під час кожного кроку. Ось як вони працюють:

register_name(message): Ця функція отримує повідомлення з ім'ям від користувача, яке він ввів після команди /start. Вона зберігає це ім'я, відправляє запит на введення номеру телефону та реєструє наступний крок обробки як register_phone.

register_phone(message, name): Ця функція отримує повідомлення з номером телефону користувача, яке він ввів після запиту. Вона зберігає цей номер телефону, відправляє запит на введення марки та моделі автомобіля та реєструє наступний крок обробки як register_car_model.

register_car_model(message, name, phone): Ця функція отримує повідомлення з маркою та моделлю автомобіля користувача, яке він ввів після запиту. Вона зберігає ці дані у базі даних, відправляє користувачеві повідомлення про успішну реєстрацію та відображає меню доступних команд.

Ці функції дозволяють взаємодіяти з користувачем під час реєстрації та ефективно зберігати інформацію про клієнтів у базі даних.

```python
@bot.message_handler(commands=['services'])
def handle_services(message):
    print(f"Received /services command from {message.chat.id}")

    services_list = "\n".join([f"{service}: {price}" for service, price in services_prices.items()])

    bot.send_message(message.chat.id, "Список послуг та їхня ціна у гривнях:\n" + services_list)
```

Рисунок 3.12 – Список послуг

Ця функція handle_services(message) відповідає на команду /services, надсилаючи користувачу список доступних послуг разом з їхніми цінами. Ось як вона працює:

@bot.message_handler(commands=['services']): Цей декоратор вказує, що функція буде обробляти команду /services.

print(f"Received /services command from {message.chat.id}"): Цей рядок виводить в консоль повідомлення про отриману команду /services, разом із ідентифікатором чату користувача, який її надіслав.

services_list = "\n".join([f"{service}: {price}" for service, price in services_prices.items()]): Цей рядок генерує список послуг та їх цін зі словника services_prices. Він формує рядок, в якому кожен елемент словника представлений у вигляді "назва_послуги: ціна_послуги", розділених символом нового рядка (\n).

bot.send_message(message.chat.id, "Список послуг та їхня ціна у гривнях:\n" + services_list): Цей виклик методу send_message() відправляє повідомлення користувачеві зі списком доступних послуг та їх цін. Це повідомлення містить заголовок "Список послуг та їхня ціна у гривнях:", після якого йде список послуг із їх цінами.

```python
services_prices = {
    "Заміна масла": "500 грн",
    "Перевірка гальм": "300 грн",
    "Розвал-сходження": "400 грн",
    "Заміна фільтра повітря": "200 грн",
    "Діагностика двигуна": "600 грн",
    "Чистка та обробка тормозів": "350 грн",
    "Заміна свічок запалювання": "250 грн",
    "Перевірка системи охолодження": "400 грн",
    "Заміна глушника": "700 грн",
    "Виправлення дефектів ходової частини": "450 грн",
}
```

Рисунок 3.13 – Словник

```python
@bot.message_handler(commands=['admin'])
def handle_admin(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    if message.chat.id != admin_id:
        bot.send_message(message.chat.id, "Ви не маєте доступу до цієї команди.")
        return

    conn, cursor = connect_database()
    cursor.execute("SELECT name, phone, car_model FROM clients")
    clients = cursor.fetchall()
    close_database(conn)

    if not clients:
        bot.send_message(message.chat.id, "Немає зареєстрованих клієнтів.")
        return
# client[1]=phone client[0] = name, client[2] = car_model
    clients_list = "\n" join([f"[{client[1]}]  {client[0]}, {client[2]}" for client in clients])
    bot.send_message( (parameter) message: Any ієнтів:\n" + clients_list)
    bot.send_message(message.chat.id, "Введіть номер клієнта для зміни статусу:")
    bot.register_next_step_handler(message, select_client)
```

Рисунок 3.14 – Можливості адміністратора

На рисунку 3.14 показано @bot.message_handler(commands=['admin']): Цей декоратор вказує, що функція буде обробляти команду /admin. Далі виконуються перевірки: чи користувач має доступ до цієї команди, чи в БД є зареєстровані клієнти. Функція дозволяє адміністратору подивитися всіх користувачів які зареєстровані в системі та дозволяє змінити статус послуги клієнта. conn, cursor = connect_database(): Цей рядок встановлює з'єднання з базою даних.

conn.commit(): Цей рядок зберігає зміни, внесені до бази даних. close_database(conn): Цей рядок закриває з'єднання з базою даних. Нижче буде наведено функції які відповідають за цей фунціонал.

```
def select_client(message):
    try:
        phone = (message.text)
        client = getClientByPhone(phone)

        if not client:
            bot.send_message(message.chat.id, "Клієнта з таким номером не знайдено. Спробуйте ще раз.")
            return
        actions = getClientActions(client[1])
        showClientActions(message.chat.id, actions)
        if not actions:
            return
        bot.send_message(message.chat.id, "Введіть номер дії для зміни статусу:")
        bot.register_next_step_handler(message, select_client_action, client[1], )
    except ValueError:
        bot.send_message(message.chat.id, "Невірний формат. Введіть номер послуги у вигляді числа.")
```

Рисунок 3.15 – Зміна статусу

На рисунку 3.15 відображена функція яка показує послуги клієнта та дозволяє змінити статус обраної послуги.

```
def select_client_action(message, clientId):
    try:
        actionId = int(message.text)
        conn, cursor = connect_database()
        cursor.execute("SELECT * FROM actions WHERE id = ?", (actionId,))
        action = cursor.fetchone()
        close_database(conn)

        if not action:
            bot.send_message(message.chat.id, "Такої послуги немає.")
            return
        keyboard = telebot.types.ReplyKeyboardMarkup(one_time_keyboard=True)
        keyboard.row('в черзі', )
        keyboard.row(  'в роботі',  )
        keyboard.row(  'виконано')
        bot.send_message(message.chat.id, "Оберіть статус:", reply_markup=keyboard)
        bot.register_next_step_handler(message, update_client_action_status, clientId,actionId )
    except ValueError:
        bot.send_message(message.chat.id, "Невірний формат. Введіть номер послуги у вигляді числа.")
```

Рисунок 3.16 – Зміна статусу(2)

На рисунку 3.16 відображено функцію для вибору послуги клієнта та відображення статусів.

```python
def update_client_action_status(message, clientId , actionId):
    status = message.text

    if status not in ['в черзі', 'в роботі', 'виконано']:
        bot.send_message(message.chat.id, "Невірний статус. Будь ласка, оберіть один з запропонованих статусів.")
        return

    conn, cursor = connect_database()
    cursor.execute("UPDATE actions SET status = ? WHERE user_chatid = ? and id = ?", (status, clientId,actionId))
    conn.commit()
    close_database(conn)

    bot.send_message(message.chat.id, "Статус клієнта успішно змінено.")
    bot.send_message(message.chat.id, f"Статус виконання робіт: {status}")
```

Рисунок 3.17 – Зміна статусу(3)

На рисунку 3.17 відображено функцію для зміни статусу послуги клієнта.

```python
@bot.message_handler(commands=['actions'])
def handle_actions(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    keyboard = telebot.types.ReplyKeyboardMarkup(one_time_keyboard=True)
    keyboard.row('Заміна масла', )
    keyboard.row( 'Перевірка гальм', )
    keyboard.row(  'Розвал-сходження', )
    keyboard.row( 'Заміна фільтра повітря' )
    keyboard.row( 'Діагностика двигуна' )
    keyboard.row( 'Чистка та обробка тормозів' )
    keyboard.row('Заміна свічок запалювання', )
    keyboard.row(  'Перевірка системи охолодження',)
    keyboard.row(  'Заміна глушника', )
    keyboard.row(  'Виправлення дефектівходової частини', )
    bot.send_message(message.chat.id, "Оберіть послугу:", reply_markup=keyboard)
    bot.register_next_step_handler(message, create_client_action)
```

Рисунок 3.18 – Створення обраної послуги

На рисунку 3.18 показано як були створені кнопки, щоб користувач міг вибрати послугу із за пропонованих натиснувши на неї. @bot.message_handler(commands=['actions']): Цей декоратор вказує, що функція буде обробляти команду /actions.

```python
def create_client_action(message):
    name = message.text
    default_status = 'в черзі'
    if name not in ['Заміна масла', 'Перевірка гальм', 'Розвал-сходження','Заміна фільтра повітря','Діагностика двигуна','Чистка та обробка тормозів
        bot.send_message(message.chat.id, "Такої послуги немає.")
        return

    conn, cursor = connect_database()
    cursor.execute("INSERT INTO actions (user_chatid, name, status) VALUES (?, ?, ?)",
                   (message.chat.id, name, default_status))
    conn.commit()
    close_database(conn)

    bot.send_message(message.chat.id, "Обрана послуга створена.", reply_markup=telebot.types.ReplyKeyboardRemove())
    bot.send_message(message.chat.id, f"Послуга: {name} -> додана зі станусом: {default_status}")
```

Рисунок 3.19 – Створення обраної послуги клієнту

На рисунку 3.20 буде показано фрагмент коду за допомогою якого відбувається відображення усіх послуг які клієнт використовував. @bot.message_handler(commands=['myactions']): Цей декоратор вказує, що функція буде обробляти команду /myactions.

```python
# Відображення усіх послуг які клієнт використовував
@bot.message_handler(commands=['myactions'])
def handle_myactions(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    clientId = message.chat.id
    actions  = getClientActions(clientId)
    showClientActions(clientId, actions)
```

Рисунок 3.20 – Відображення усіх послуг клієнта

На рисунку 3.21 та на рисунку 3.22 буде показано функції для счітування послуг, відображення послуг, обирання клієнта за ID та обирання клієнта за номером телефону.

```python
# Зчитування обраних послуг клієнта по ID
def getClientActions(clientId):
    conn, cursor = connect_database()
    cursor.execute("SELECT id, user_chatid, name, status FROM actions WHERE user_chatid = ?", (clientId,))
    actions = cursor.fetchall()
    close_database(conn)
    return actions
# Відображення послуг клієнта по ID
def showClientActions(clientId, actions):
    if not actions:
        bot.send_message(clientId, "Ви ще не користувались нашими послугами.")
        return
    clients_list = "\n".join([f"[{action[0]}] - {action[2]}, статус:{action[3]};" for action in actions])
    bot.send_message(clientId, "Список виконаних послуг:\n" + clients_list)

# Обрати клієнта з бази даних по його ID
def getClientById(clientId):
    conn, cursor = connect_database()
    cursor.execute("SELECT * FROM clients WHERE chat_id = ?", (clientId,))
    client = cursor.fetchone()
    close_database(conn)
    return client
```

Рисунок 3.21 – зчитування та відображення

```python
#Обрати клієнта з бази даних по його телефону
def getClientByPhone(phone):
    conn, cursor = connect_database()
    cursor.execute("SELECT * FROM clients WHERE phone = ?", (phone,))
    client = cursor.fetchone()
    close_database(conn)

    return client
```

Рисунок 3.22 – зчитування та відображення(2)

```
# Головна функція, яка постійно перевіряє наявність нових
повідомлень
print("Bot is running...")
bot.polling(none_stop=True)
```

Лістинг 3.21 – Очікування нових повідомлень

Ця остання частина коду виконує функцію, яка стежить за наявністю нових повідомлень у чаті та обробляє їх. Ось як вона працює:

print("Bot is running..."): Цей рядок просто виводить в консоль повідомлення про те, що бот запущено і працює.

bot.polling(none_stop=True): Цей рядок запускає процес перевірки наявності нових повідомлень у чаті. Функція polling() постійно виконується, перевіряючи сервер Telegram на наявність нових повідомлень для бота. Параметр none_stop=True вказує, що бот має продовжувати роботу навіть у випадку помилки.

Ця остання функція дозволяє боту постійно працювати, обробляючи нові повідомлення, які надходять у чат. Це основний цикл бота, який дозволяє йому надавати відповіді та взаємодіяти з користувачами.

3.3 Робота користувача з Telegram-ботом

Початок роботи Telegram-боту

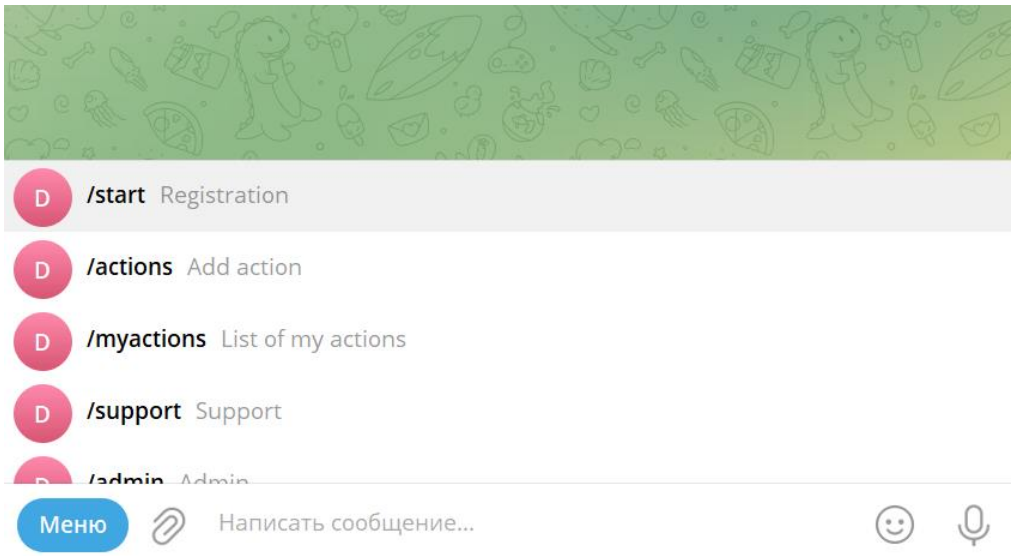

Рисунок 3.1 – Початок роботи з ботом (використання команди «/start»)

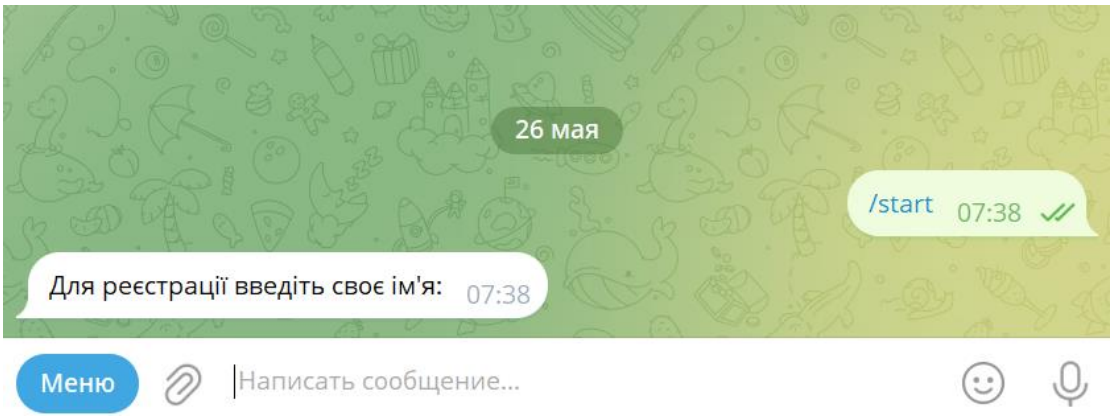

Рисунок 3.2 – Початок роботи з ботом

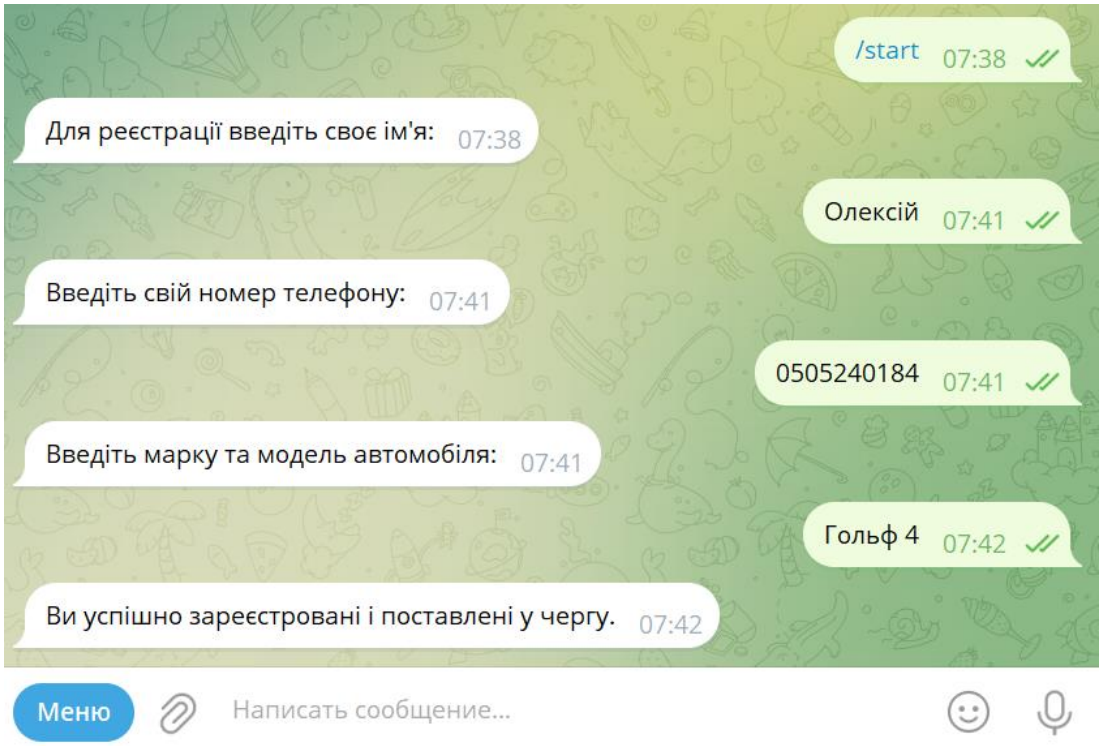

Рисунок 3.3 – Успішна реєстрація користувача

Під час реєстрації, вводимо на запит бота ім’я, номер телефону та назву автомобіля.

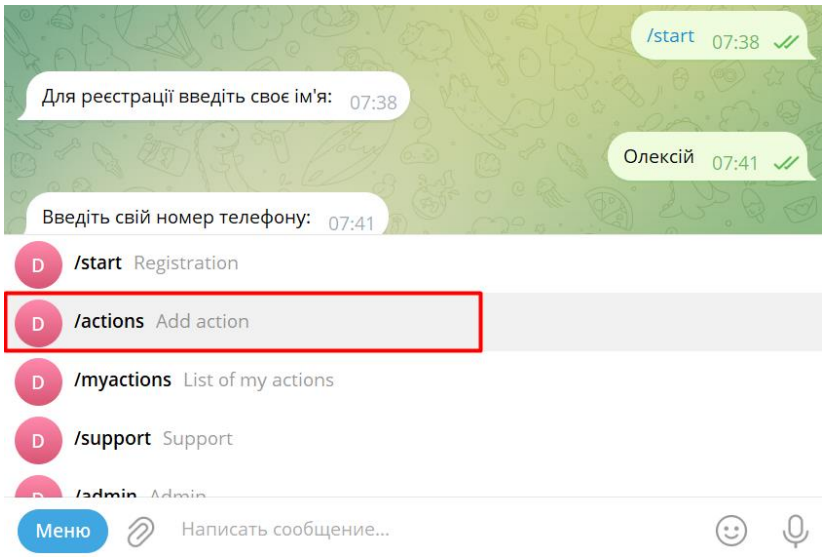

Рисунок 3.4 – Команда «actions»

Для відображення послуг які має сервіс потрібно використати команду «/actions». Потім користувач може обрати те що йому потрібно приклад можна спостерігати нижче на Рисуноку 3.5 – Команда «/actions»

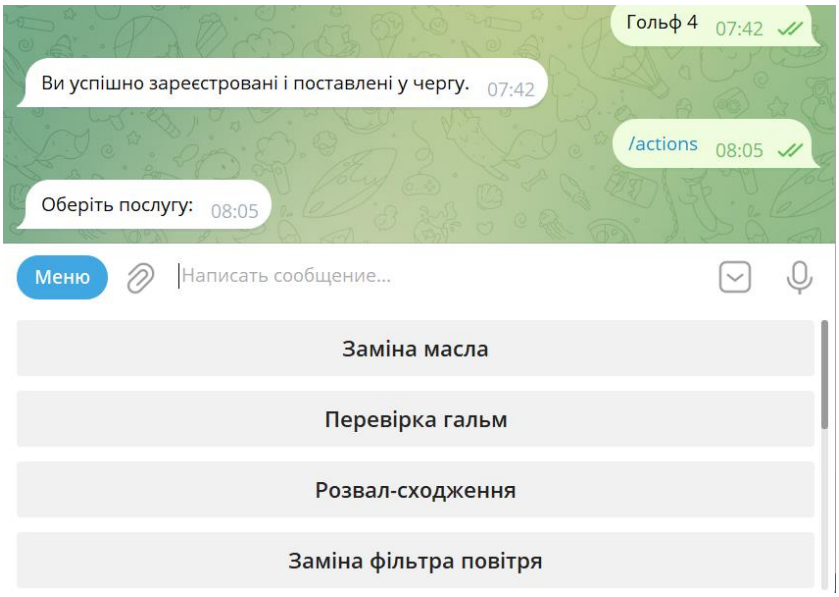

Рисунок 3.5 – Команда «actions»

Користувач обирає послугу і вона додається в чергу приклад можна побачити на Рисунку 3.6 – Додавання обраної послуги в чергу

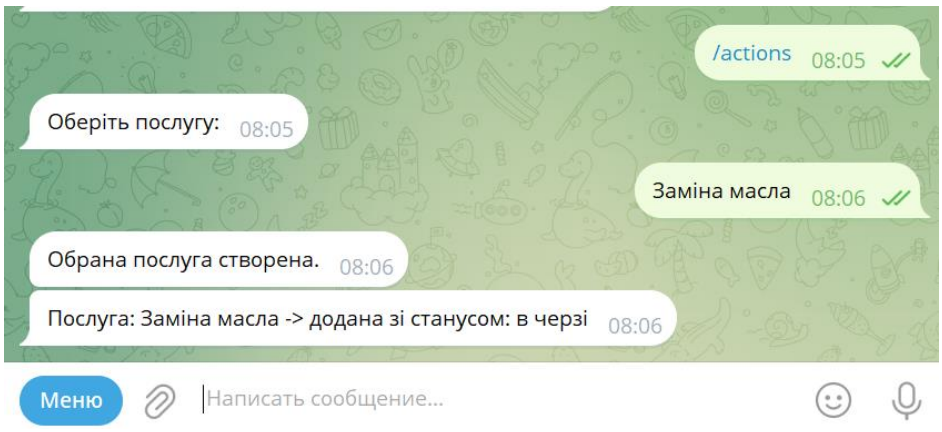

Рисунок 3.6 – Додавання обраної послуги в чергу

Користувач може переглянути всі послуги що він замовляв натиснувши на команду «/myactions»

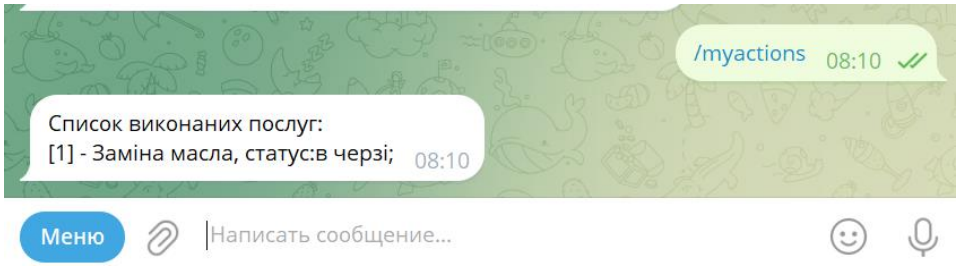

Рисунок 3.7 – Перегляд послуг що замовлялися

Також користувач може скористатися командою «/support» для опису своєї проблеми та негайного надсилання повідомлення на сервіс. Приклад роботи проілюстровано нижче на Рисунку 3.8 – Виконання команди «/support» та Рисунку 3.9 – Виконання команди «/support».Адміністратор

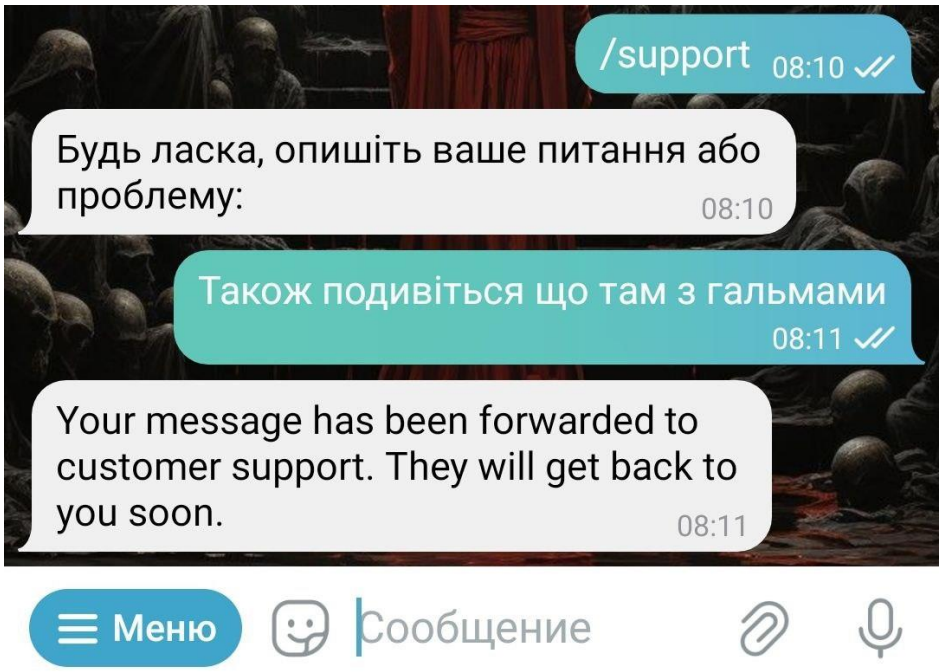

Рисунок 3.8 – Виконання команди «/support»

При використанні такої команди на акаунт адміністратора або сапорта буде надіслано повідомлення з вказаною проблемою , та інформацією про користувача. Приклад на Рисунку 3.9 – Виконання команди «/support». Адміністратор.

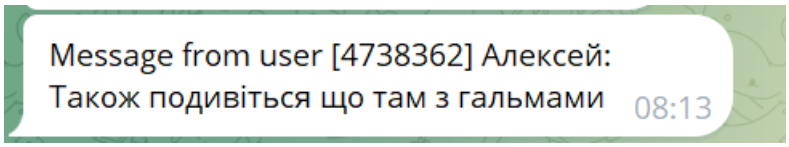

Рисунок 3.9 – Виконання команди «/support».Адміністратор.

Команда «/admin» для користувача не є доступною , при умові що він її використає зявиться сповіщення про розмежування прав. Приклад на Рисунку 3.10 – «/admi» для користувача.

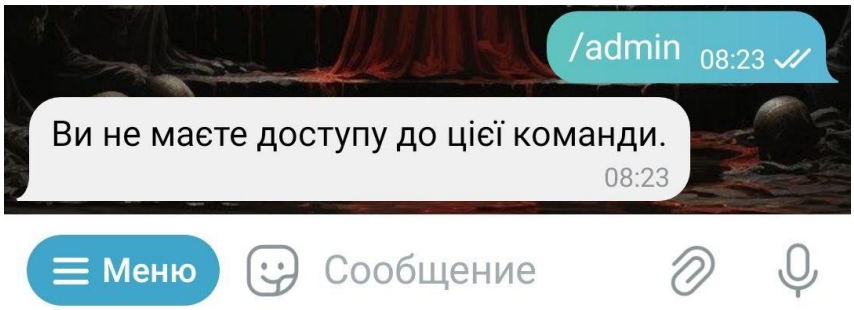

Рисунок 3.10 – «/admi» для користувача.

3.4 Робота адміністратора з Telegram-ботом

Роглянемо роботу телеграм-бота зі сторони адміністратора. Рисунок 3.11 – Робота адміністратора.

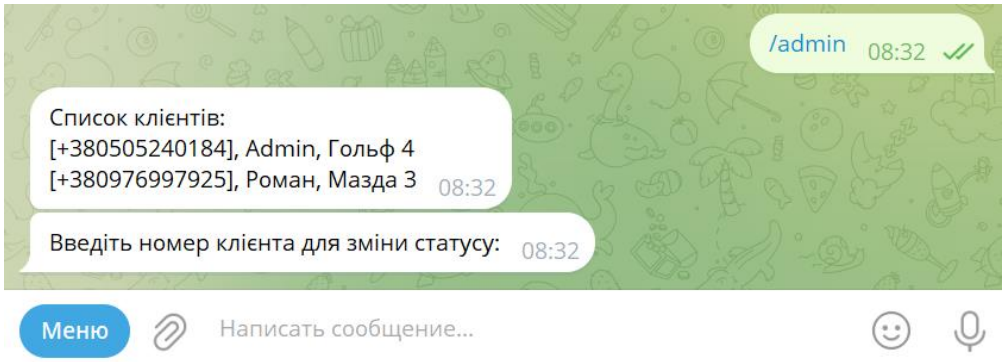

Рисунок 3.11 – «/admi» для адміністратора.

Виконавши цю команду адміністратор бачить список клієнтів та інформацію про них. Адмін може змінювати статус виконання послуги окремого клієнта. Приклад можемо спостерігати на Рисунку 3.12 – Зміна статусу роботи, Рисунок 3.13 – Зміна статусу роботи (2), Рисунок 3.14 – Відображення змін у користувача.

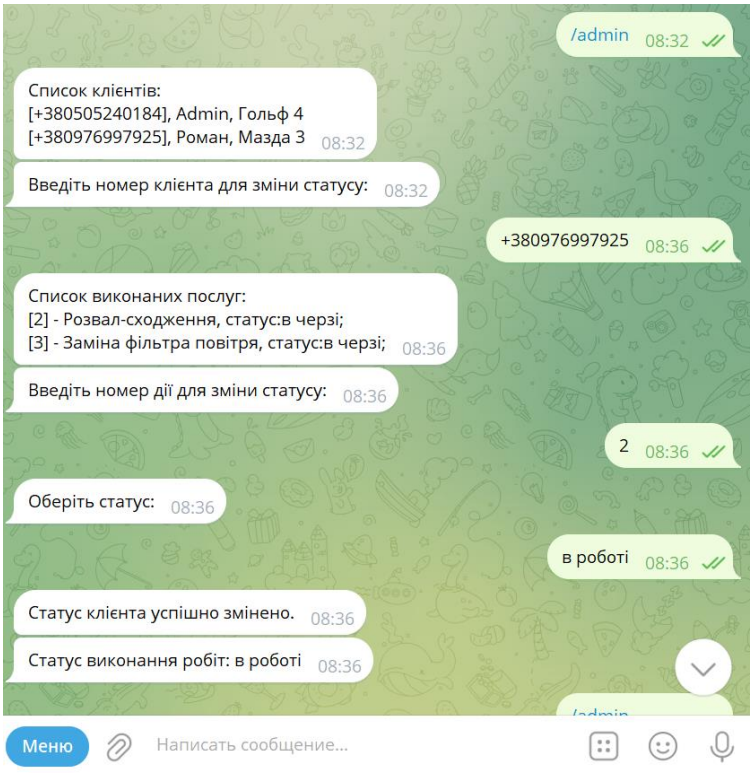

Рисунок 3.12 – Зміна статусу роботи.

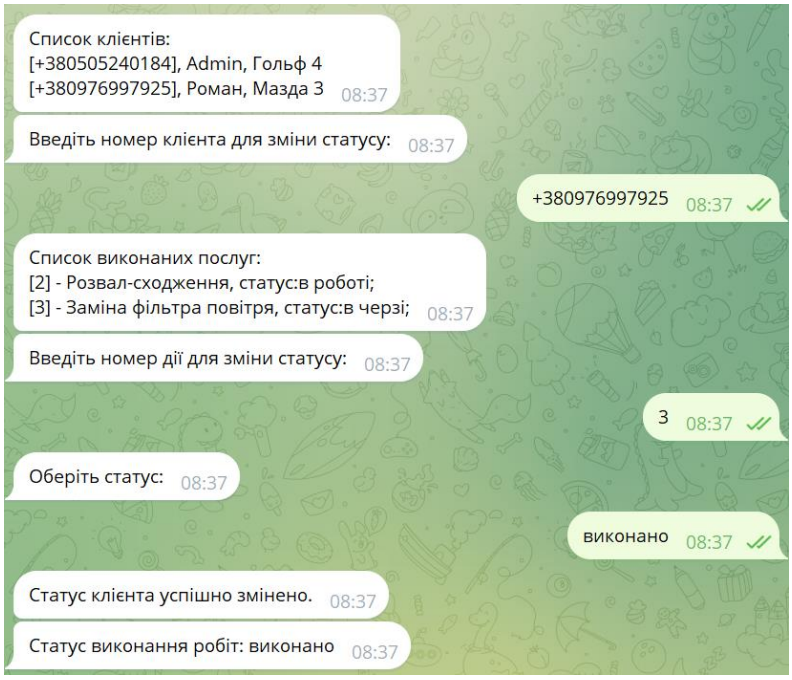

Рисунок 3.13 – Зміна статусу роботи (2)

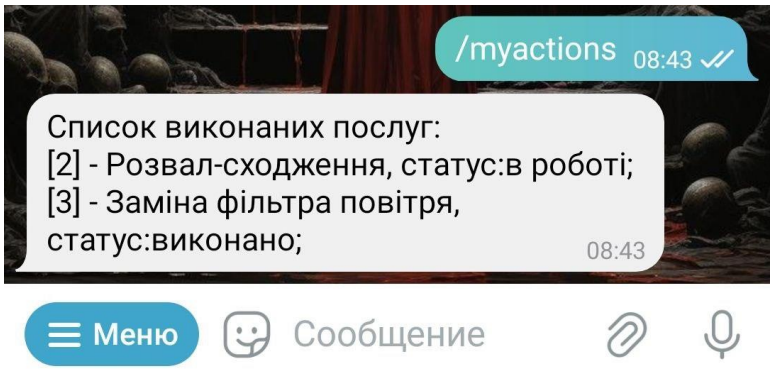

Рисунок 3.14 – Відображення змін у користувача.

3.4 Робота підтримки з Telegram-ботом

Користувач може надіслати повідомлення в підтримку для вирішення проблеми , або надання інформації відносно робти. Користувач це може зробити за допомогою команди «/support» , таким чином зявляється повідомлення від

сервісу , а користувач має змогу написати його та надіслати внаслідок чого відбудеться негайне сповіщення у робітника на сервісі. Приклад роботи можна спостерігати нижче на Рисунку 3.15 – Ствренння запиту у підтримку

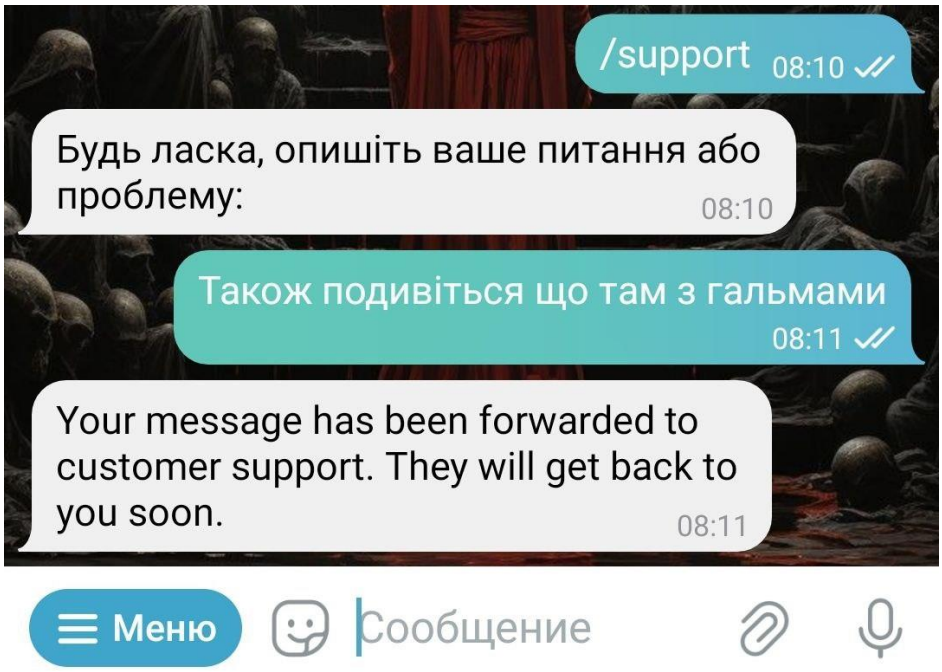

Рисунок 3.15 – Створення запиту у підтримку

На Рисунку 3.15 можемо спостерігати як користувач вводить повідомлення, а на Рисунку 3.16 можемо побачити як це відбувається зі сторони робітника.

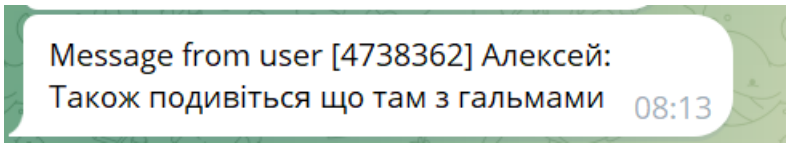

Рисунок 3.15 – Надходження запиту у підтримку

3.4 Тестування Telegram-боту

Перший тест. Зробимо тестування відносно перевірки на реєстрацію – тобто коли користувач уже зареєстрований , чи зможе він зареєструватися знову. Отже приклад показано на Рисунку 3.16.

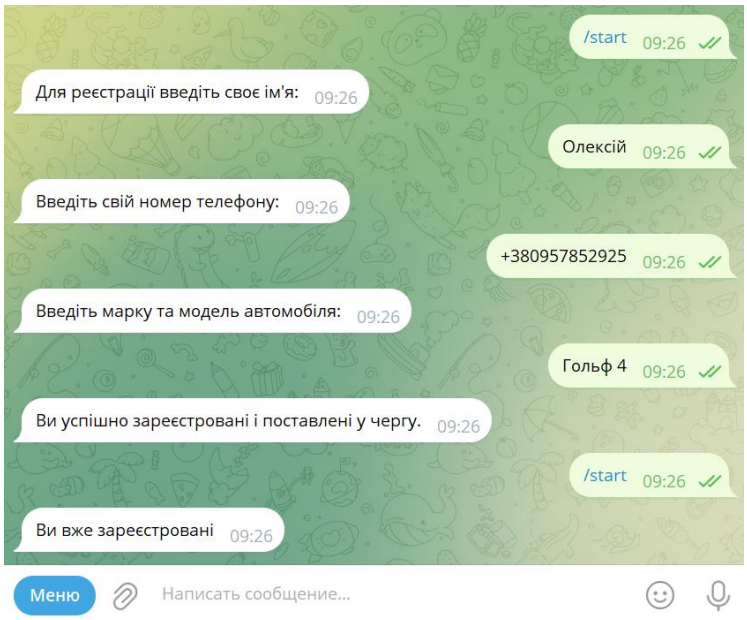

Рисунок 3.16 – Перевірка на реєстрацію

Тест виконано з гарним результатом оскільки було виконано перевірку та надіслано сповіщення про те що цей користувач уже зареєстрований.

Другий тест. Зробимо тестування відносно виконання команди на перевірку послуг які уже є – тобто коли користувач уже зареєстрований але не обрав послуги , а відразу натиснув на перевірку послуг які є в наявності. Приклад проілюстровано на Рисунку 3.17

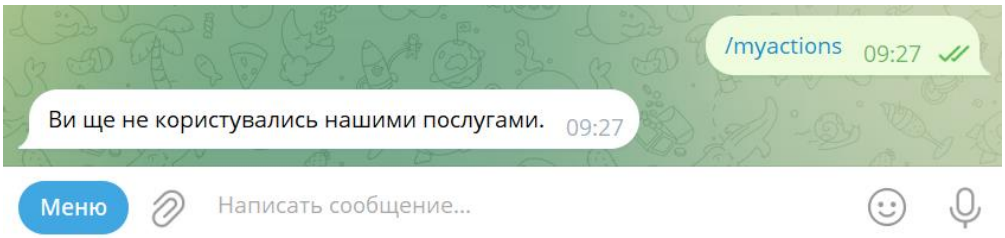

Рисунок 3.17 – Перевірка свої послуг

Тест виконано з гарним результатом , оскільки була виконана перевірка і на діслано сповіщення про те що користувач ще не користувався нашими послугами – тобто не обрав будь якої послуги зі списку.

Третій тест. Зробимо тестування відносно всіх команд при умові що користувач не зареєстрований , а відразу почина користуватися всіма командами.

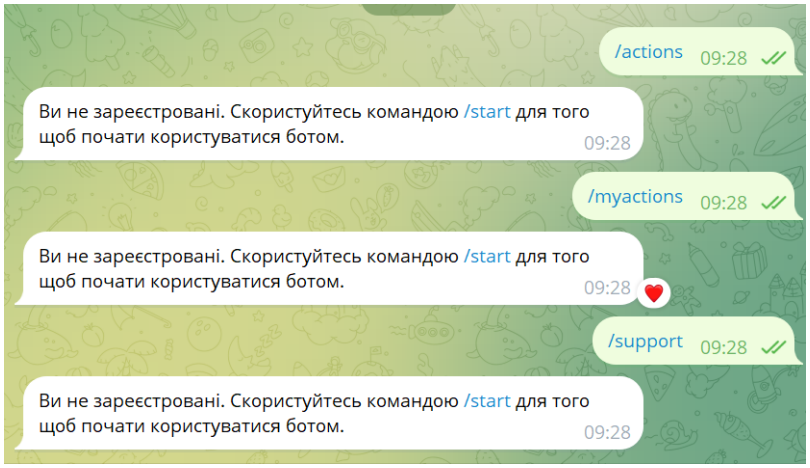

Рисунок 3.18 – Не зареєстрований користувач.

Тест виконано з гарним результатом оскільки була виконана перевірка і надіслано сповіщення відносно того щоб користувач натисну певну команду для реєстрації.

Четвертий тест. Зробимо тестування відносно користувача який намагається виконати команду «/admin»

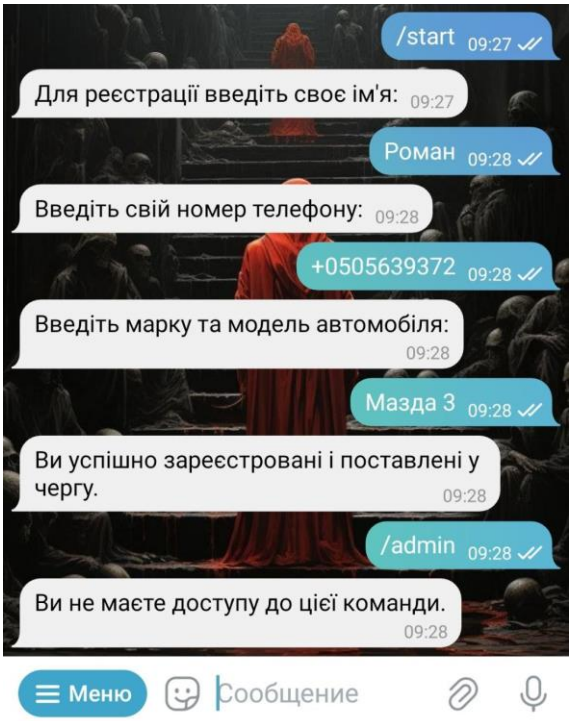

Рисунок 3.19 – Функціонал адміністратора

Тест виконано з гарним результатом, оскільки було зроблено перевірку чи це є адміністратор , якщо ні то буде виводитись сповіщення що користувач не має змоги до цієї команди, що в нашому випаку і вийшло.

Таким чином, було спроектовано, розроблено і протестовано телеграм-бот для зв’язку клієнтів СТО з сервісом та вдосконалення його роботи.

ВИСНОВКИ

Тема розробки телеграм-боту для сервісу СТО є надзвичайно важливою в контексті сучасних технологічних та соціальних тенденцій. В умовах стрімкого розвитку цифрових технологій та швидкого темпу життя, власники автомобілів шукають ефективні та зручні способи керування обслуговуванням своїх транспортних засобів. Телеграм-бот, який інтегрується з сервісом технічного обслуговування, може вирішити цю проблему, забезпечуючи власникам автомобілів зручний і швидкий спосіб отримання необхідної інформації та замовлення послуг.

Однією з головних переваг такого бота є можливість забезпечення безпосереднього та оперативного спілкування між клієнтами та представниками СТО. Це дозволяє зменшити час очікування на відповідь, уникнути зайвих телефонних дзвінків і спростити процес планування обслуговування автомобіля.

Також важливою перевагою є можливість забезпечення постійного доступу до інформації про стан автомобіля та статус виконаних ремонтних робіт через телеграм-бот. Це робить процес обслуговування більш прозорим та зручним для власників автомобілів, а також дозволяє їм бути в курсі всіх змін та оновлень у режимі реального часу.

У першому розділі даної кваліфікаційної роботи було проаналізовано предметну область, зокрема, було розглянуто поняття чат-ботів, а також проаналізовані існуючі аналоги це дозволило краще зрозуміти клієнта та його потреби як користувача.

Другий розділ був присвячений аналізу і вибору засобів розробки телеграм-ботів, мов програмування та баз даних. Для розробки прогрманої реалізації було обрано мову програмування Python та СУБД SQLite. Також було виконанно моделювання роботи бота в нотаціях IDEF0, use case, декомпозиція внутрішніх

потоків. Проведено аналіз та проектування баз даних – створено логічну модель бази даних. Це все дозволяє краще зрозуміти логіку та функціонал продукту.

У третьому розділі було розроблено програмне рішення, а саме спроектовано систему, використовуючі найсучасніші методики, розроблено програмний код боту, а також протестовано створене програмне рішення ,що в свою чергу призвело до покрещення роботи чат-боту та виправленню помилок.

Крім того, важливість цієї теми полягає в можливості вдосконалення робочих процесів в автомобільних сервісах. Використання технології телеграм-боту може допомогти оптимізувати процеси прийому та обробки замовлень, забезпечити краще управління ресурсами та підвищити загальну ефективність роботи СТО.

Отже, в контексті швидко розвиваючогося світу автомобільні сервіси повинні адаптуватися до нових технологічних можливостей для задоволення потреб сучасних клієнтів, і розробка телеграм-боту для СТО є ключовим елементом цього процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Developers Guide. URL: https://devguide.python.org/ (дата звернення: 10.05.2024)

2. Документація веб-фреймоворка Django. URL: https://docs.djangoproject.com/en/3.0/ (дата звернення: 10.05.2024)

3. Документація бази даних PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 10.05.2024)

4. Створення Telegram Bot. URL: https://codeguida.com/post/410

5. Telegram Bots. URL: https://core.telegram.org/bots (дата звернення: 10.05.2024)

8. Документація Telegram Bot API. URL: https://core.telegram.org/bots/api#available-methods (дата звернення: 10.05.2024)

9. Документація бібліотеки pyTelegramBotAPI. URL: https://github.com/eternnoir/pyTelegramBotAPI (дата звернення: 10.05.2024)

10. Документація бази даних rozklad KPI API. URL: https://api.rozklad.org.ua/ (дата звернення: 10.05.2024)

11. State Design Pattern. URL: https://sourcemaking.com/design_patterns/state (дата звернення: 10.05.2024)

12. Starting a Django Project. URL: https://realpython.com/django-setup/ (дата звернення: 10.05.2024)

13. SQL vs ORM. URL: https://habr.com/ru/company/pgdayrussia/blog/328690/ (дата звернення: 10.05.2024)

14. PostgreSQL - CREATE Database. URL: https://www.tutorialspoint.com/postgresql/postgresql_create_database.htm (дата звернення: 10.05.2024)

15. Virtual Environments and Packages. URL: https://docs.python.org/3/tutorial/venv.html (дата звернення: 10.05.2024)

16. Python Secrets Module. URL: https://pynative.com/python-secrets-module/ (дата звернення: 10.05.2024)

17. Types of Chatbot Technology. URL: https://medium.com/voice-tech-podcast/types-of-chatbot-technology-72d095df2540 (дата звернення: 10.05.2024)

18. Skills and Technologies Driving Chatbot Innovation Technology. URL: https://jasoren.com/skills-and-technologies-drivingchatbot-innovation/ (дата звернення: 10.05.2024)

19. Chatbot fundamentals. URL: https://www.artificial-solutions.com/chatbots#1 (дата звернення: 10.05.2024)

20. Andy English Bot. URL: https://t.me/andyrobot (дата звернення: 10.05.2024)

21. Pomodoro Bot. URL: https://t.me/pomodoro_timer_bot/ (дата звернення: 10.05.2024)

22. Словники Dictionaries. URL: https://t.me/dictsbot (дата звернення: 10.05.2024)

23. Telegram APIs. URL: https://core.telegram.org/api (дата звернення: 10.05.2024)

24. Bots: An introduction for developers. URL: https://core.telegram.org/bots (дата звернення: 10.05.2024)

25. Боти в Telegram що це таке і як вони працюють. URL: https://sharkdevelop.com/boty-v-telegram/ (дата звернення: 10.05.2024)

ДОДАТОК А.

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку телеграм-боту
«Telegram-bot для автоматизації взаємодії
працівників СТО з клієнтами ”»

ПОГОДЖЕНО:

Доцент кафедри інформаційних технологій

_____ Кузнєцов Е.Г.

Студент групи ІТ-03

_____ Немийний О.В.

Суми 2023

1. Призначення й мета створення телеграм-бота для покращення комунікування між працівниками на сервісі та клієнтами

1.1 Призначення програмного продукту

Даний програмний продукт має надавати повноцінну інформацію відносно статусу виконання роботи та полегшити запис до майстрів , тобто запис в чергу та реєстрація.

1.2 Мета створення web-додатку

Метою даного продукту є залучення більшого обсягу клієнтів за такою перевагою як : зручний запис в чергу , зручний перегляд стану виконання роботи та бігато чого іншого для зручного комунікування.

1.3 Цільова аудиторія

До цільової аудиторії телеграм-боту можна віднести практично всіх людей, що мають свій власний автомобіль , які зацікавленні в їх обслуговування та покращення стану автомобіля.

2 Вимоги до web-додатку

2.1 Вимоги до web-додатку в цілому

2.1.1 Вимоги до структури й функціонування web-додатку

Web-додаток має бути доступним в мережі Інтернет під доменним іменем gloss.zzz.com.ua. Web-додаток повинен складатися із взаємозалежних розділів із чітко розділеними функціями.

Каталог Послуг:

Описова Інформація: Кожена послуга повинна мати докладний опис, включаючи ціну та характеристику.

Зручний Інтерфейс:

Легкість Навігації: Забезпечення простого та зрозумілого інтерфейсу для клієнтів, який дозволяє легко переглядати та вибирати послуги.

Команда Support:

Онлайн запитання: Можливість відправити питання через чат-підтримку в режимі реального часу.

2.1.2 Вимоги до персоналу

Від персоналу не має вимагатися особливих технічних навичок для підтримки й експлуатації телеграм-боту, окрім загальних навичок роботи з персональним комп'ютером і стандартним веб-браузером.

Персонал, що працює з телеграмом, повинен мати базові навички користування комп'ютером або телефоном .

2.1.3 Вимоги до збереження інформації

Уся інформація надана у телеграм-боті буде зберігатися у базі даних реалізованій засобами системи управління базами даних sqlLite та не буде використовуватися з якимись неадекватними намірами.

2.1.4 Вимоги до розмежування доступу

Розроблюваний телеграм-бот має встановлювати обмеження доступу для різних категорій користувачів, забезпечуючи безпеку та конфіденційність

інформації. Доступ до функціоналу розмежовується за наступними групами користувачів: адміністратор та зареєстрований клієнт.

Адміністратор:

Має необмежений доступ до всієї інформації та функціоналу боту.

Має права перегляду, додавання, редагування та видалення будь-якої інформації.

Зареєстрований клієнт:

Має можливість переглядати інформацію в боті, обирати та переглядати доступні послуги.

Має права на додаткові функції, визначені в групі користувачів як "Зареєстровані".

2.2 Структура телеграм-боту

2.2.1 Загальна інформація про структуру телеграм-боту

Структура телеграм-боту охоплює ряд важливих компонентів для забезпечення зручного користування.

До складу програмного продукту входять:

Головна сторінка:

Містить навігаційне меню для швидкого доступу.

Показує список команд для використання , та зручне меню внизу для швидкого доступу до них.

Містить команду зв’язку з адміністрацією, команду для підтвердження замовлення певної послуги.

Каталог з послугами:

Показує кнопки з послугами.

Також містить команду для зв’язку з робочими для зручності комунікування.

2.2.2 Навігація

По середині при вході до боту розташоване навігаційне меню з командами, яке фіксується для зручності користувача. Це меню містить команди на всі доступні функції програмного продукту.

2.2.3 Управління контентом

Адміністратор має повний доступ до редагування інформації , додавання послуг , зміни «цінової політики» , створення додаткових команд, редагування уже створених команд , змінювати інтерфейс.

2.2.4 Дизайн та структура додатку

Дизайн боту виконаний у мінімалістичному та сучасному стилі, має стандартний вигляд чату. Використовуються стандартні кольори які були встановлені користувачем заздалегідь. Розташування елементів є зручним та логічним, надаючи користувачеві приємний інтерфейс.

2.2.5 Система навігації (карта web-додатку)

Карта web-додатку зображена на рисунку А.2.

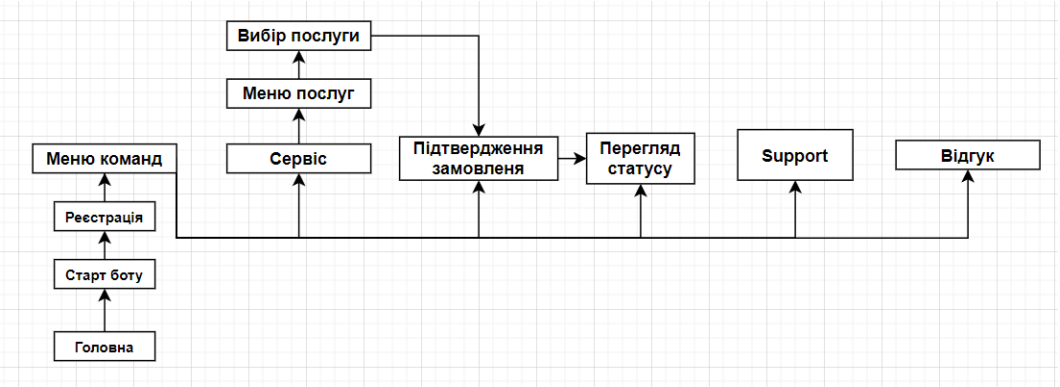

Рисунок А.2 – Карта web-додатку

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Потреби користувача, визначені на основі рішення замовника, представлені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	Потреби користувача	Джерело
ТХ-01	Створення замовлення та стати в чергу	Клієнт
ТХ-02	Надіслання жалоби	Клієнт
ТХ-03	Перегляд та пошук послуги	Клієнт
ТХ-04	Отримання інорфмації про статус	Клієнт
ТХ-05	Реєстрація	Клієнт
ТХ-06	Зворотній зв'язок та підтримка	Усі користувачі
ТХ-07	Редагування даних	Адміністратор

2.3.2 Функціональні вимоги

На основі потреб користувача були визначені такі функціональні вимоги:

-Реєстрація та авторизація користувачів;

-Відображення меню послуг;

-Відображення статусу замовлення;

-Можливість зворотного зв’язку з адміністратором;

-Адміністрування інформації;

-Підтвердження або відмова в послузі;

-Зберігання статусу;

-Залишення відгука про програмний продукт

2.3.3 Системні вимоги

Даний розділ визначає системні вимоги, визначені розробником.

Проаналізувавши потреби користувачів було визначено наступні вимоги:

- Реєстрація та авторизація користувачів:

Система повинна надавати можливість користувачам створювати облікові записи та авторизуватися для доступу до особистого простору.

- Перегляд та пошук послуг:

Має бути реалізована функція для швидкого та зручного доступу для перегляду меню послуг та їх вибір .

- Створення нової заявки :

Додаток повинен забезпечувати можливість користувачам створювати, новий заказ та показувати його статус.

- Створення зворотного зв’язку:

При виникненні проблеми по вині робочих чи взагалі , у користувача повинна бути можливість надання інформації адміністратору відносно своєї проблеми , тобто команда для реалізації сповіщення про проблему.

- Залишення відгуку та його перегляд

Користувач має змогу залишити свій відгук про програмний продукт та в майбутньому переглянути інші відгуки інших користувачів.

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Реалізація додатку відбувається з використанням:

- Python

- SQLite
- botFather

3 Склад і зміст робіт зі створення web-додатку

Докладний опис етапів роботи зі створення web-додатку наведено в таблиці А.3.

Таблиця А.3 – Етапи створення web-додатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Розробка дизайну інтерфейсу	0 день
2	Створення архітектури	7 дні
3	Розробка основної логіки	1 день
4	Реалізація функції реєстрації	2 дні
5	Реалізація функції оформлення замовлень	2 дні
6	Реалізація функції показу меню послуг	2 дні
7	Реалізація функції звортного звязку	2 день
8	Реалізація функції залишення відгука	2 день
9	Реалізація функції support	2 день
10	Компіляція	1 день
11	Тестування програмного продукту	7 дні
12	Написання супровідної документації	7 дні
13	Реліз додатку	2 день
	Загальна тривалість робіт	37 днів

ДОДАТОК Б.
ПЛАНУВАННЯ РОБІТ

Метою даної роботи є розробка програмного продукту – Telegram-bot для автоматизації взаємодії працівників СТО з клієнтами.

Для досягнення мети проекту потрібно виконати такі завдання як:

- провести аналіз продуктів які уже є на «ринку» та використовуються для того щоб визначити їх переваги та недоліки
- виконати планування та покрокові пукти виконання-написання боту
- спробувати роботу на своєму прикладі приїхавши на СТО та здійснивши виконавши функціонал боту

Деталізація мети проекту методом SMART.

Перше , що потрібно зробити – це ще на концептуальному етапі визначити мету проекту методом SMART . Даний метод покращує аналіз . Приклад можна спостерігати на Таблиці 1.

Таблиця Б.1 − Деталізація мети методом SMART

Specific (конкретна)	Створити Телеграм-бота для малих підприємців, та можливо часних робітників що спеціалізуються на виконанні послуг для ремонту авто та інших послуг, з можливістю здійснення зручної та зрозумілої комунікації між робітниками та клієнтами. Також для зручності самих клієнтів.
Measurable (вимірювана)	Забезпечити можливість створення та редагування розширеними функціональними можливостями.

продовження Таблиці Б.1 – Деталізація мети методом SMART

Achievable (досяжна)	Технічне завдання від замовника , наявність програмістів та тестувальників , які будуть виконувати поставлене завдання , також використовується ПЗ Visual Studio Code,постійний зв'язок з замовником.
Relevant (реалістична)	Мета проекту пов'язана з бізнес-стратегією та його розвитком в онлайн-середовищі. Запуск Телеграм-боту є актуальним та важливим кроком для підвищення конкурентоспроможності, залучення нових клієнтів та підвищення обсягів виконаних робіт.
Time-framed (обмежена у часі)	Завершення розробки до 22 травня 2024 р.

Планування змісту структури робіт. Основним інструментом для планування змісту структури робіт служить WBS діаграма − графічне подання згрупованих елементів проекту у вигляді пакета робіт, які ієрархічно пов’язані з продуктом проекту. Побудуємо структуру WBS, у якій детально опишемо роботи, які потрібно виконати на кожному етапі створення проекту. Виконаємо декомпозицію робіт для даного проекту. Взагалом – це процес розподілу проекту на менші, керовані компоненти або завдання. Діаграма WBS зображена на рис. Б.1.

Планування структури організації, для впровадження готового проекту (OBS). Після побудови WBS розробимо організаційну структуру виконавців OBS. Організаційна структура проекту стосується тільки внутрішньої організаційної структури проекту і не стосується відносин проектних груп чи учасників з батьківськими організаціями. Діаграма OBS зображена на рис. Б.2. Список виконавців, що функціонують в проекті знаходиться в табл. Б.2.

Таблиця Б.2 − Виконавці проекту

Роль	Ім’я	Проектна роль
Розробник	Немийний О.В.	Виконує розробку основного функціоналу проекту та інтерфейс користувача
Проектувальник	Немийний О.В.	Проектує структуру запитів та команд телеграм-бота, розробляє дизайн програмного продукту.
Тестувальник	Немийний О.В.	Відповідає за тестування функціоналу та дизайну додатку, перевірку моделі на адекватність.
Косультант проекту	Немийний О.В.	Формує завдання на розробку проекту.
Менеджер проекту	Немийний О.В.	Відповідає за виконання термінів, розподіл ресурсів та завдань між учасниками. Виконує збір та аналіз даних.

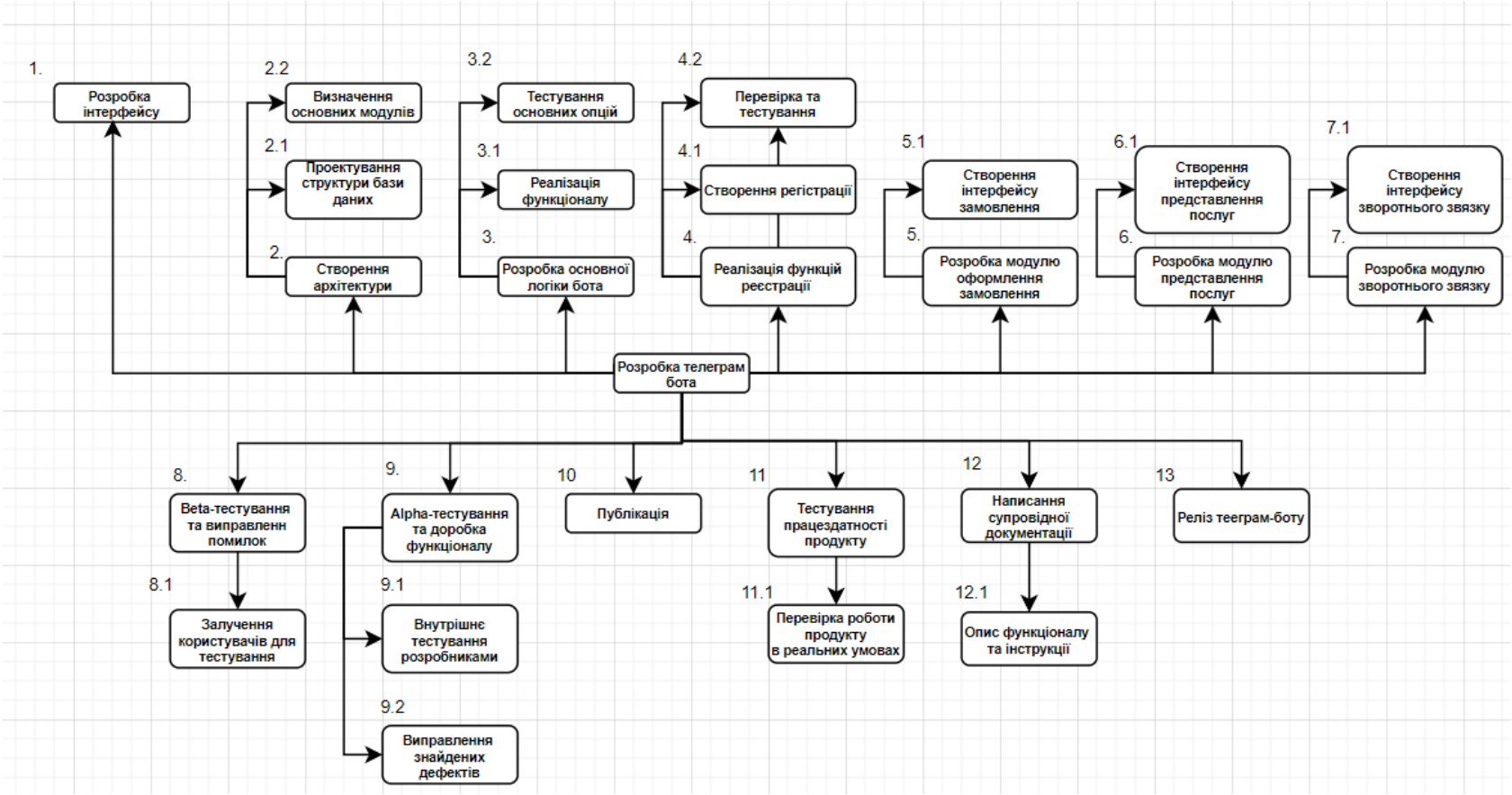

Рисунок Б.1 – WBS.Структура робіт проекту

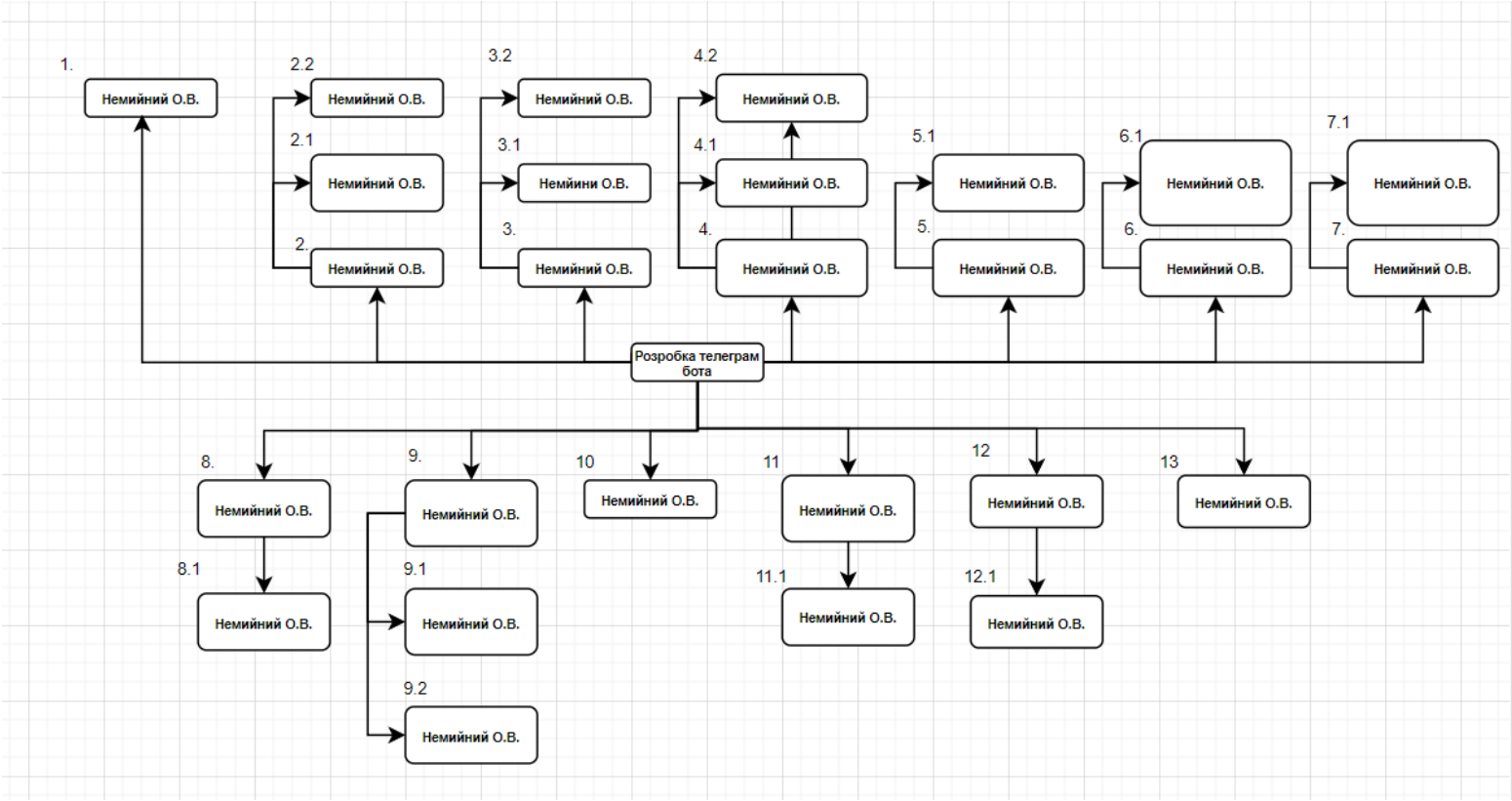

Рисунок Б.2 – Організаційна структура проекту (OBS)

Діаграма Ганта. Далі побудуємо календарний план виконання проекту. Найпоширеніший формат графіка в будь-якій галузі — діаграма Ганта. Цей графік дозволяє менеджерам проекту і всій команді розробників візуалізувати графіки часу і взаємозв'язок між окремими завданнями та етапами роботи над проектом.

Побудова календарного графіку (діаграми Ганта) є одним з важливих етапів планування проекту, що виглядає як розклад виконання робіт з реальним розподілом дат. Завдяки йому можна отримати достовірне уявлення про тривалість процесів з обмеженнями у ресурсах.

Календарний графік проекту представлено на Рисунку Б3 – діаграма.

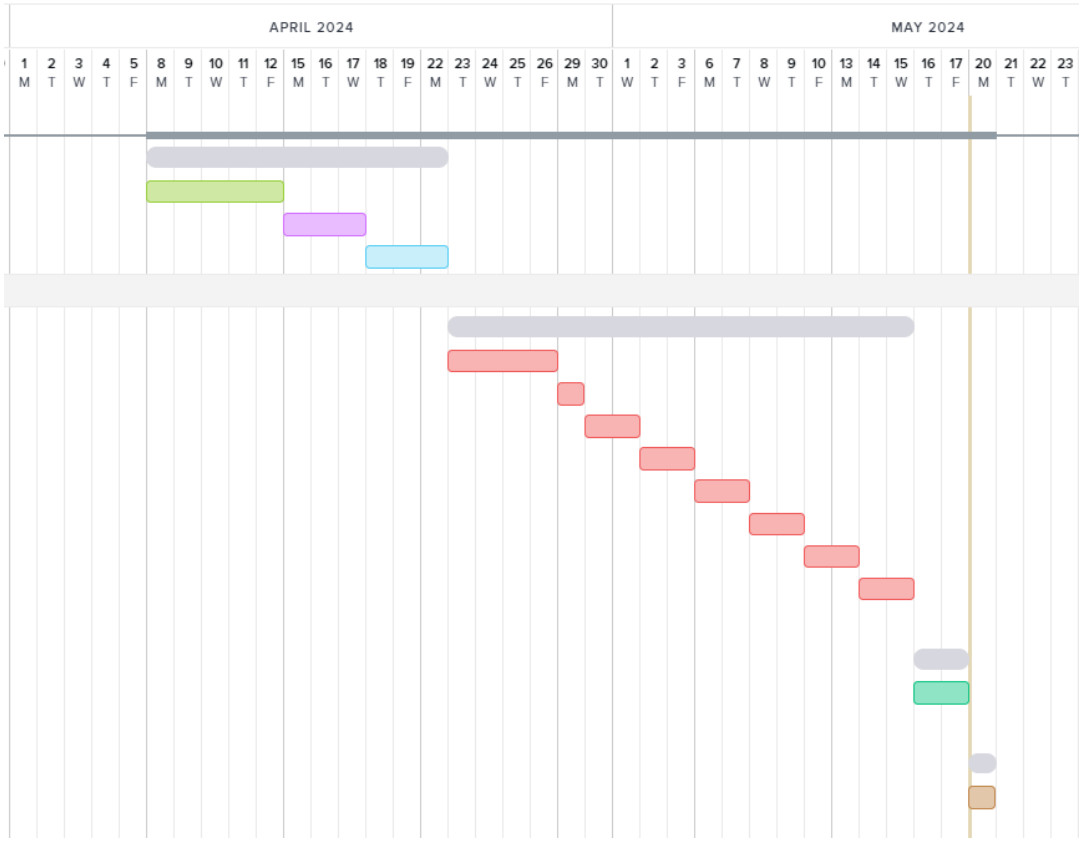

Рисунок Б.3 – Діаграма Ганта

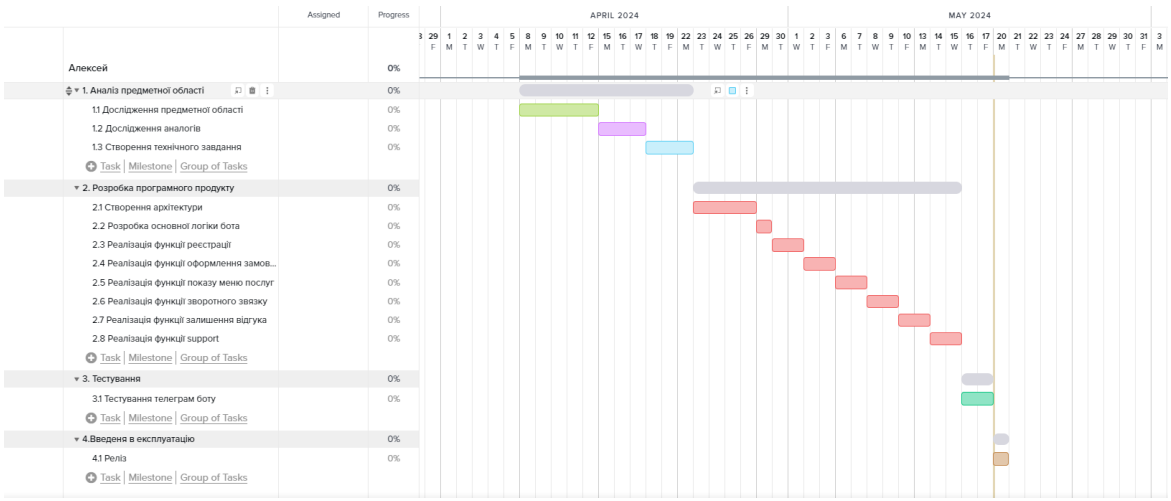

Рисунок Б.4 – Продовження діаграми Ганта

Аналіз ризиків. Виконаємо якісну і кількісну оцінку ризиків роботи. При якісній оцінці визначимо ризики, що потребують швидкого реагування. Така оцінка визначить ступінь важливості ризику і дозволить вибрати спосіб реагування. Кількісна оцінка ризиків буде виконана для більш повної ідентифікації ризиків та ступеня їхнього впливу на виконання проекту. Кількісна і якісна оцінка ризиків можуть використовуватися окремо або разом, залежно від наявного часу і бюджету, необхідності в кількісній або якісній оцінці ризиків. У табл. R.5 знаходиться класифікація ризиків за показниками ймовірності виникнення ризику та величині втрат.

Далі виконаємо планування реагування на ризики — це розробка методів і технологій зниження негативного впливу ризиків на проект. Визначимо ефективність розробки реагування на проект, визначимо чи будуть наслідки впливу ризику на проект позитивними або негативним.

Оцінюємо ризики за показниками. На основі оцінки будуємо матрицю ймовірності виникнення ризиків та впливу ризику, що зображена на рис. R.6.

У таблиці R.3 надано перелік ризиків даного проекту.

Таблиця R.3 Ризики проєкту

№	Опис ризику
1	Виникнення непорозумінь між замовником та розробником
2	Недостатньо описане завдання для розроблення
3	Відключення світла
4	В певний період часу розробник може бути не працездатним
5	Низька кваліфікація розробників
6	Нераціональне виділеня часу для розробки
7	Дуже часте внесення змін

Таблиця R.4 – Результати визначення ймовірності , впливу та рангу ризиків проекту

№	Назва ризику	Ймовірність (0.1-0.9)	Вплив (0.05-0.8)	Ранг
1	Виникнення непорозумінь між замовником та розробником	0.1	0.3	0.02
2	Недостатньо описане завдання для розроблення	0.5	0.5	0.2
3	Відключення світла	0.1	0.3	0.02

продовження таблиці R.4

4	В певний період часу розробник може бути не працездатним	0.2	0.8	0.14
5	Низька кваліфікація розробників	0.5	0.6	0.26
6	Нераціональне виділеня часу для розробки	0.3	0.6	0.18
7	Дуже часте внесення змін	0.45	0.6	0.29

Таблиця R.5 – Шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	Ймовірність виникнення	Вплив ризику	Типи ризику
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Потрібно знизити негативний вплив ризиків на проект , а для цього потрібно план як реагувати на цей вплив.

Матриція ймовірності та впливу згідно проекту була отримана в результаті планування заходів реагування на ризики.

- зелений колір – прийнятні ризики;
- жовтий колір – виправданні ризики;
- червоний колір – недопустимі ризики.

Таблиця R.6 – Матриція ймовірності та впливу згідно проекту

Ймовірність ризику	Вплив загорзи ризику				
	Дуже малий	Малий	Середній	Великий	Дуже великий
	0.04	0.1	0.2	0.5	0.9
0.9					
0.7					
0.5				R2(0,21)	R6(0,26)
0.3					R7(0,9)
0.1			R1(0,01),R3(0,01)		R4(0,16)R5(0,19)

Ризики проекту за рівнем представлені у таблиці R.7 , а у таблиці R.8 описано ризики та дії які потрібно буде робити , щоб усунути їх.

Таблиця R.7 – Шкала оцінювання ризику за рівнем ризику

№	Назва	Межі	Ризик які входять
1	Прийняті	0,05<R<0,06	1,3
2	Виправдані	0,05<R<0,16	4,5
3	Недопустимі	0,12<R<0,68	2,6,7

Таблиця R.8 – Ризики та дії для їхнього усунення

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_1	Відкритий	Непорозуміння між розробником та замовником	Низька	Середній	3	1. Налагодити гарні відносини між розробником та керівником. 2. Дотримуватися ділового етикету спілкування. 3. Створити комфортні умови для співпраці	Попередження	При виявлення непорозуміння потрібно вияснити, що саме стало причиною непорозуміння обсудити її та створити здорову атмосферу в колективі.
RS_2	Відкритий	Недостатньо описане завдання для розроблення	Середня	Високий	6	1. Швидким чином звязатися з замовником , та обговорити всі пункти які необхідно розуміти для розроблення. 2. Домовитися про зустрічі для контролю 3. Тимчасові контрольні надсилання виконаної частини	Попередження	При умові якщо було виявлено невідповідність готового продукту з тим що замовляв замовник, швидко виділити всі пункти які не відповідають замовлення та виправити їх

Продовження таблиці R.8

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_3	Відкритий	Відключення світла	Низька	Середній	3	1. Виконувати роботу на ноутбуках, або інших пристроях з автономними джерелами живлення. 2. Закупити переносні генератори та потужні павербанків 3. Мати резервне джерело мобільного інтернету.	Прийняття	Переміщення офісу або робітників у населений пункт з стабільної енергосистемою
RS_4	Відкритий	В певний період часу розробник може бути не працездатним	Низька	Високий	3	.	Прийняття	

RS_5	Відкритий	Низька кваліфікація розробників	Середня	Високий	5	1. В першу чергу повинна будти людина яка буде оцінбвати навички робітників. 2. Якщо така ситуація виникла , швидко знайти робітників через іншу компанію , або через знайомих	Пом'якшення	За умови , якщо робота була виконана дуже не якісно , або взагалі не виконана потрібно негайно звільнити працівника і в швидкому темпі знайти нового на різних платформах , або ж через знайомих

Продовження таблиці R.8

ID	Статус ризику	Опис ризику	Ймовірність виникнення	Вплив ризику	Ранг ризику	План А	Тип стратегії реагування	План Б
RS_6	Відкритий	Нераціональне виділеня часу для розробки	Низька	Високий	3	1.Створити план роботи з вказаними завдання та датами їх виповнення. 2.Кожну неділю, або інший період часу в залежності від проекту -	Попередження	Розробник та замовник повинні зв'язатися та знайти компроміс для встановлення нових термінів , для виконання

						звязуватися з командою , та робити аналіз хто що виконав , а хто ще не встигає , для розуміння чи виконається проект у вказаний термін		програмного продукту
RS_7	Відкритий	Дуже часте внесення змін у технічне завдання	Середня	Високий	5	1. Першоетапно створити пункти які необхідно виконати і чітко слідувати ним. 2. Запитати у замовника що він хоче бачити в кінцевому результат і на цьому зійтись – поставити чіткі завдання	Попередження	Розробник та замовник повинні зв’язатися та знайти компроміс для встановлення нових умов , для виконання програмного продукту.

ДОДАТОК В. ЛІСТИНГ КОДУ

```
import telebot
import sqlite3
def read_token():
    with open("token.txt", "r") as file:
        return file.read().strip()
def connect_database():
    conn = sqlite3.connect('clients.db')
    cursor = conn.cursor()
    return conn, cursor
def close_database(conn):
    conn.close()
def clientsTable():
    conn, cursor = connect_database()
    cursor.execute('''CREATE TABLE IF NOT EXISTS clients (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        chat_id INTEGER,
                        name TEXT,
                        phone TEXT,
                        car_model TEXT)''')

    conn.commit()
    close_database(conn)
def actionsTable():
    conn, cursor = connect_database()
    cursor.execute('''CREATE TABLE IF NOT EXISTS actions (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        user_chatid INTEGER,
                        name TEXT,
```

```
                        status TEXT)''')
    conn.commit()
    close_database(conn)
def startDB():
     clientsTable()
     actionsTable()
bot = telebot.TeleBot(read_token())

startDB()
def getClientActions(clientId):
    conn, cursor = connect_database()
    cursor.execute("SELECT id, user_chatid, name, status FROM
actions WHERE user_chatid = ?", (clientId,))
    actions = cursor.fetchall()
    close_database(conn)
    return actions
def showClientActions(clientId, actions):
    if not actions:
        bot.send_message(clientId, "Ви ще не користувались нашими
послугами.")
        return
    clients_list = "\n".join([f"[{action[0]}] - {action[2]},
статус:{action[3]};" for action in actions])
    bot.send_message(clientId, "Список виконаних послуг:\n" +
clients_list)
def getClientById(clientId):
    conn, cursor = connect_database()
    cursor.execute("SELECT * FROM clients WHERE chat_id = ?",
(clientId,))
    client = cursor.fetchone()
```

```
    close_database(conn)
    return client
def getClientByPhone(phone):
    conn, cursor = connect_database()
    cursor.execute("SELECT * FROM clients WHERE phone = ?",
(phone,))
    client = cursor.fetchone()
    close_database(conn)

    return client
def addCommandsMenu(chatId):
    c1 = telebot.types.BotCommand(command='start',
description='Registration')
    c2 = telebot.types.BotCommand(command='actions',
description='Add action')
    c3 = telebot.types.BotCommand(command='myactions',
description='List of my actions')
    c4 = telebot.types.BotCommand(command='support',
description='Support')
    c5 = telebot.types.BotCommand(command='admin',
description='Admin')
    c6 = telebot.types.BotCommand(command='services',
description='services')
    bot.set_my_commands([c1,c2,c3, c4,c5,c6])
    bot.set_chat_menu_button(chatId,
telebot.types.MenuButtonCommands('commands'))
def checkUserExist(chatId):
    client = getClientById(chatId)
    if not client:
```

```
            bot.send_message(chatId, "Ви не зареєстровані.
Скористуйтесь командою /start для того щоб почати користуватися
ботом.")
            return False
    else:
        return True
def handle_support_chat(message):
    client = getClientById(message.chat.id)
    if not client:
        bot.send_message(message.chat.id, "Клієнта не знайдено.")
        return
    print(f"Received support message from {message.chat.id}:
{message.text}")
    support_chat_id = 644443347  # Replace with your support chat
ID
    forward_text = f"Message from user [{client[3]}]
{client[2]}:\n{message.text}"
    bot.send_message(support_chat_id, forward_text)
    bot.send_message(message.chat.id, "Your message has been
forwarded to customer support. They will get back to you soon.")
@bot.message_handler(commands=['start'])
def handle_start(message):
    print(f"Received /start command from {message.chat.id}")
    client = getClientById(message.chat.id)
    if not client:
        bot.send_message(message.chat.id, "Для реєстрації введіть
своє ім'я:")
        bot.register_next_step_handler(message, register_name)
    else:
        bot.send_message(message.chat.id, "Ви вже зареєстровані")
```

```
def register_name(message):
    print(f"{message.chat.id} registered name: {message.text}")
    name = message.text
    bot.send_message(message.chat.id, "Введіть свій номер
телефону:")
    bot.register_next_step_handler(message, register_phone, name)

def register_phone(message, name):
    print(f"{message.chat.id} registered phone: {message.text}")
    phone = message.text
    bot.send_message(message.chat.id, "Введіть марку та модель
автомобіля:")
    bot.register_next_step_handler(message, register_car_model,
name, phone)

def register_car_model(message, name, phone):
    print(f"{message.chat.id} registered car model:
{message.text}")
    car_model = message.text
    conn, cursor = connect_database()
    cursor.execute("INSERT INTO clients (chat_id, name, phone,
car_model) VALUES (?, ?, ?, ?)",
                   (message.chat.id, name, phone, car_model))
    conn.commit()
    close_database(conn)
    bot.send_message(message.chat.id, "Ви успішно зареєстровані і
поставлені у чергу.")
    addCommandsMenu(message.chat.id)
services_prices = {
```

```
    "Заміна масла": "500 грн",
    "Перевірка гальм": "300 грн",
    "Розвал-сходження": "400 грн",
    "Заміна фільтра повітря": "200 грн",
    "Діагностика двигуна": "600 грн",
    "Чистка та обробка тормозів": "350 грн",
    "Заміна свічок запалювання": "250 грн",
    "Перевірка системи охолодження": "400 грн",
    "Заміна глушника": "700 грн",
    "Виправлення дефектів ходової частини": "450 грн",
}
@bot.message_handler(commands=['services'])
def handle_services(message):
    print(f"Received /services command from {message.chat.id}")

    services_list = "\n".join([f"{service}: {price}" for service,
price in services_prices.items()])

    bot.send_message(message.chat.id, "Список послуг та їхня ціна у
гривнях:\n" + services_list)
@bot.message_handler(commands=['support'])
def handle_support(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    print(f"Received /support command from {message.chat.id}")
    bot.send_message(message.chat.id, "Будь ласка, опишіть ваше
питання або проблему:")
    bot.register_next_step_handler(message, handle_support_chat)
```

```
admin_id = 644443347
@bot.message_handler(commands=['admin'])
def handle_admin(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    if message.chat.id != admin_id:
        bot.send_message(message.chat.id, "Ви не маєте доступу до цієї команди.")
        return

    conn, cursor = connect_database()
    cursor.execute("SELECT name, phone, car_model FROM clients")
    clients = cursor.fetchall()
    close_database(conn)

    if not clients:
        bot.send_message(message.chat.id, "Немає зареєстрованих клієнтів.")
        return
    clients_list = "\n".join([f"[{client[1]}], {client[0]}, {client[2]}" for client in clients])
    bot.send_message(message.chat.id, "Список клієнтів:\n" + clients_list)
    bot.send_message(message.chat.id, "Введіть номер клієнта для зміни статусу:")
    bot.register_next_step_handler(message, select_client)
def select_client(message):
    try:
        phone = (message.text)
```

```
        client = getClientByPhone(phone)

        if not client:
            bot.send_message(message.chat.id, "Клієнта з таким
номером не знайдено. Спробуйте ще раз.")
            return
        actions = getClientActions(client[1])
        showClientActions(message.chat.id, actions)
        if not actions:
            return
        bot.send_message(message.chat.id, "Введіть номер дії для
зміни статусу:")
        bot.register_next_step_handler(message,
select_client_action, client[1], )
    except ValueError:
        bot.send_message(message.chat.id, "Невірний формат. Введіть
номер послуги у вигляді числа.")
def select_client_action(message, clientId):
    try:
        actionId = int(message.text)
        conn, cursor = connect_database()
        cursor.execute("SELECT * FROM actions WHERE id = ?",
(actionId,))
        action = cursor.fetchone()
        close_database(conn)

        if not action:
            bot.send_message(message.chat.id, "Такої послуги
немає.")
            return
```

```
        keyboard =
telebot.types.ReplyKeyboardMarkup(one_time_keyboard=True)
        keyboard.row('в черзі', )
        keyboard.row(  'в роботі',  )
        keyboard.row(  'виконано')
        bot.send_message(message.chat.id, "Оберіть статус:",
reply_markup=keyboard)
        bot.register_next_step_handler(message,
update_client_action_status, clientId,actionId )
    except ValueError:
        bot.send_message(message.chat.id, "Невірний формат. Введіть
номер послуги у вигляді числа.")
def update_client_action_status(message, clientId , actionId):
    status = message.text

    if status not in ['в черзі', 'в роботі', 'виконано']:
        bot.send_message(message.chat.id, "Невірний статус. Будь
ласка, оберіть один з запропонованих статусів.")
        return

    conn, cursor = connect_database()
    cursor.execute("UPDATE actions SET status = ? WHERE user_chatid
= ? and id = ?", (status, clientId,actionId))
    conn.commit()
    close_database(conn)

    bot.send_message(message.chat.id, "Статус клієнта успішно
змінено.")
    bot.send_message(message.chat.id, f"Статус виконання робіт:
{status}")
```

```
@bot.message_handler(commands=['actions'])
def handle_actions(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    keyboard =
telebot.types.ReplyKeyboardMarkup(one_time_keyboard=True)
    keyboard.row('Заміна масла', )
    keyboard.row( 'Перевірка гальм', )
    keyboard.row(  'Розвал-сходження', )
    keyboard.row( 'Заміна фільтра повітря' )
    keyboard.row( 'Діагностика двигуна' )
    keyboard.row( 'Чистка та обробка тормозів' )
    keyboard.row('Заміна свічок запалювання', )
    keyboard.row(  'Перевірка системи охолодження',)
    keyboard.row(  'Заміна глушника', )
    keyboard.row(  'Виправлення дефектівходової частини', )
    bot.send_message(message.chat.id, "Оберіть послугу:",
reply_markup=keyboard)
    bot.register_next_step_handler(message, create_client_action)
def create_client_action(message):
    name = message.text
    default_status = 'в черзі'
    if name not in ['Заміна масла', 'Перевірка гальм', 'Розвал-
сходження','Заміна фільтра повітря','Діагностика двигуна','Чистка
та обробка тормозів','Заміна свічок запалювання', 'Перевірка
системи охолодження', 'Заміна глушника', 'Виправлення дефектів
ходової частини']:
        bot.send_message(message.chat.id, "Такої послуги немає.")
        return
```

```
    conn, cursor = connect_database()
    cursor.execute("INSERT INTO actions (user_chatid, name, status)
VALUES (?, ?, ?)",
                   (message.chat.id, name, default_status))
    conn.commit()
    close_database(conn)

    bot.send_message(message.chat.id, "Обрана послуга створена.",
reply_markup=telebot.types.ReplyKeyboardRemove())
    bot.send_message(message.chat.id, f"Послуга: {name} -> додана
зі статусом: {default_status}")
@bot.message_handler(commands=['myactions'])
def handle_myactions(message):
    exist = checkUserExist(message.chat.id)
    if exist == False:
        return
    clientId = message.chat.id
    actions  = getClientActions(clientId)
    showClientActions(clientId, actions)
print("Bot is running...")
bot.polling(none_stop=True)
```