ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
СПЕЦІАЛЬНІСТЬ 121 «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи освітнього ступеня «бакалавр»

на тему: «**Телеграм-сервіс з логістики**»

Виконав студент **4**-го курсу, групи – **ПІ-60**
спеціальності 121 «Інженерія програмного забезпечення»

_____ Владислав ЧЕПЕЛЬ

Керівник _____ Валентин ЯНЧУК

Рецензент _____ Олена ЧИЖМОТРЯ

Житомир – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
СПЕЦІАЛЬНІСТЬ 121 «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри
інженерії програмного забезпечення
_____ Андрій МОРОЗОВ
«14» лютого 2022 р.

ЗАВДАННЯ
на випускну кваліфікаційну роботу

Студента Чепеля Владсилава Олександровича

Тема роботи: «Телеграм-сервіс з логістики»

Затверджена Наказом університету від «14» лютого 2022 р. № 112/с

Термін здачі студентом закінченої роботи: «26» червня 2022 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): Об'єктом дослідження є методи та засоби, необхідні для перевезення вантажу. Предметом дослідження є процес розробки повноцінного телеграм бота з використанням сучасних програмних технологій, таких як Python, PostgreSQL та HTML5.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Тамара ЛОКТІКОВА	14.02.22	14.02.22
2	Тамара ЛОКТІКОВА	14.03.22	14.03.22
3	Тамара ЛОКТІКОВА	02.05.22	02.05.22

Керівник _____ Валентин ЯНЧУК

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі Пошук, огляд та аналіз аналогічних розробок Формулювання технічного завдання Опрацювання літературних джерел	14.02.22 – 13.03.22	Виконано
2	Проектування структури	14.03.22 – 13.04.22	Виконано
3	Написання програмного коду, розробка бази даних	14.04.22 – 13.05.22	Виконано
4	Тестування	14.05.22 – 01.06.22	Виконано
5	Написання пояснювальної записки Виготовлення презентації	02.06.22 - 26.06.22	Виконано
6	Захист		

Студент _____ Владислав ЧЕПЕЛЬ
Керівник _____ Валентин ЯНЧУК

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра являє собою програмний комплекс телеграм-сервісу з логістики та пояснювальної записки до даного сервісу, що складається з 72 сторінок, 33 ілюстрацій та 11 таблиць.

Метою роботи є розробка автономного сервісу з доставки відповідних посилок або товару, який дозволить пришвидшити та додати зручності процесу логістичних послуг.

Під час реалізації проекту було визначено основні завдання для розробки, проаналізовано аналоги даної системи, обрано конкретні інструменти та покроково наведено сценарії роботи телеграм-сервісу. Весь процес було відображено в пояснювальній записці, а саме: описано об’єктну структуру системи, базу даних сервісу (PostgreSQL) та алгоритми роботи модулів реалізованих з використанням мови програмування Python. Крім цього перевірено функціонал та зручність інтерфейсу певним колом людей. В результаті розробленою системою вже користуються за призначенням.

КЛЮЧОВІ СЛОВА: ТЕЛЕГРАМ, ОГОЛОШЕННЯ, ПОСИЛКА, БОТ, АВТО, ЛОГІСТИКА.

ABSTRACT

The final qualifying work of the bachelor is a software package of an telegram-service on logistics and an explanatory note to this service, consisting of 72 pages, 33 illustrations and 11 tables.

The aim of the work is to develop an autonomous service for the delivery of appropriate parcels or goods, which will speed up and add convenience to the process of logistics services.

During the project implementation, the main tasks for development were identified, analogues of this system were analyzed, specific tools were selected and scenarios of telegram service operation were presented step by step. The whole process was summarized in a note, described the object structure of the system, database service (PostgreSQL database) and algorithms for modules implemented using the Python programming language. In addition, the functionality and usability of the interface were superficially tested by a certain group of people. As a result, the developed system is already used for its intended purpose.

KEY WORDS: TELEGRAM, ANNOUNCEMENT, PARCEL, BOT, CAR, LOGISTICS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (*з англ.* програмний інтерфейс програми)

CRUD – create, read, update, delete (*з англ.* створити, прочитати, оновити, видалити)

MVC – Model–View–Controller (*з англ*. Модель–Представлення–Контролер)

БД – база даних

СУБД – система управління базами даних

ТБ – Телеграм бот

ВСТУП

Актуальність обраної теми. Кожного дня люди використовують ті чи інші сервіси для того, щоб задовольнити свої потреби. Це може бути будь-що, сервіс з прослуховування музики, переглядів фільмів, покупки квитків, бронювання готелів і так далі.

Все дуже просто, в людей виникають потреби і через деякий час хтось може вирішити їхню проблему у вигляді відповідного сервісу. Тей, хто зробить це вчасно і якісно буде у виграші. Так само як і ми хочемо запропонувати суспільству наш сервіс задля взаємної вигоди усіх сторін та вирішення проблеми нагальної передачі посилок з точки А в точку Б.

Потрібно розуміти, що на сьогоднішній день існує досить велика конкуренція серед відповідних сервісів. Кожен бажає бути попереду, для цього розробляється буквально все. Це може бути зручний додаток з зрозумілим та стильним інтерфейсом або продуманий веб-сайт. Але ми вирішили зробити наш сервіс ще більш доступним та зручним для майбутніх користувачів, а саме: розмістити його в телеграмі у вигляді бота.

Останнім часом телеграм боти набирають велику популярність як серед малого бізнесу, так і серед великих компаній таких як «Укрзалізниця». Це робить вас більш гнучким та адаптивним до сучасного суспільства та його потреб. Телеграм є майже у всіх. Тому інтеграція вашого сервісу в телеграм бот надає вам широкий спектр можливостей. Головне правильно їх реалізувати та підтримувати.

Метою кваліфікаційної роботи є реалізація телеграм-сервісу з логістики.

Встановлена мета обумовлює наступні **завдання**:

- проаналізувати аналогічні сервіси;
- вибрати та обґрунтувати засоби реалізації продукту;
- визначити архітектуру та алгоритми для її реалізації;
- провести тестування готового програмного продукту;

Об’єктом дослідження є методи та засоби, необхідні для доставки посилки належним чином та в короткий термін.

Предметом дослідження є процес розробки повноцінного телеграм-сервісу доставки посилок(передач) з використанням сучасних програмних технологій, таких як Python, PostgreSQL та HTML5.

Практична значимість роботи полягає в тому, що розроблений телеграм бот може вирішити проблему зручності передачі товарів або посилок у якнайкоротший термін з вигодою для обох сторін.

Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із !! найменувань (!! стор.), !! додатків (!! стор.), !! таблиць та !! ілюстрацій по тексту. Повний обсяг роботи становить !! сторінки, з них !! сторінки основного тексту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ТЕЛЕГРАМ-СЕРВІСУ

1.1. Постановка задачі

Головною метою створення телеграм-сервісу є реалізація спеціальної платформи для здійснення перевозки посилок або товарів з вигодою для обох сторін. Ми прагнемо об’єднувати людей, які можуть перевезти щось та отримати за це кошти та людей, яким потрібно щось перевезти та заплатити за це адекватну ціну. А тей факт, що це можна зробити в одному з найпопулярніших та зручному месенджері, зробить цей сервіс доступним та зручним для усіх. Звичайно ж при умові правильної побудови системи та відповідного контролю.

В випускній роботі необхідно розробити повноцінний телеграм-сервіс з логістики. Для реалізації поставленого завдання, основними етапами є:

1. Провести аналіз вимог та визначити основний функціонал системи.
2. Реалізувати ядро системи управління сервісом.
   2.1. Написати програмні модулі для:
   - реєстрації користувачів в ТБ. Вона буде реалізовуватись майже непомітно. Всі необхідні дані ми отримаємо при надсиланні першого повідомлення від користувача. Повноцінна реєстрація буде завершена при наданні мобільного телефону користувачем;
   - можливості додання та редагування вже доданого авто;
   - коректної публікації оголошення з чіткими інформативними пунктами;
   - отримання інформації відносно доступних оголошень;
   - сповіщення щодо прийому оголошення як для замовника так і для виконавця;
   - перегляду статусу оприлюдненого оголошення;

   2.2.Реалізувати CRUD операції для всіх таблиць бази даних системи.
3. Реалізувати структуру збереження даних наступного виду:

3.1. Дані про авто (марка, модель, тип, колір, рік випуску, державний номер та короткий опис);

3.2. Дані про оголошення (місто та адреса відправки, місто та адреса прибуття, дата, зображення посилки, ціна та короткий опис);

4. Розробити зручне та зрозуміле меню для користувача

5. Здійснити перевірку введених даних

6. Розробити план тестування ТБ.

У результаті виконання поставленого завдання буде створено повнофункціональний телеграм-сервіс з логістики.

1.2. Аналіз аналогів програмного продукту

Насамперед ми хочемо дещо роз’яснити щодо нашого майбутнього сервісу з доставки. Ми прагнемо реалізувати сервіс по типу «BlaBlaCar», але для перевезення посилок. Тобто людина, яка їде в інше місто по своїм справам, може заодно й доставити вашу посилку.

І ми відверто можемо сказати, що саме таких сервісів немає. Тобто перед нами стоїть завдання роз’яснити людям що це таке і як воно працює. Чому воно спрацює? Тому що вищезгаданий «BlaBlaCar» - це дуже зручно, так само буде зручно й з нашим сервісом. Вони будуть схожі, але кардинально відрізняються за своїм призначенням.

Отже, так як ми розроблюємо сервіс з логістики, насамперед нам потрібно розібрати деякі його аналоги для прикладу.

Розпочнемо з найпопулярнішої української компанії з доставки – «Нова Пошта».

«Нова Пошта» наразі є лідером у сфері логістичних послуг в Україні. Ця компанія йде в ногу з часом та постійно розвивається у всіх планах. Ми сміливо можемо сказати, що «Нова Пошта» є прикладом того, як може виглядати ваш бізнес при правильному підході та розумінням сучасних потреб.

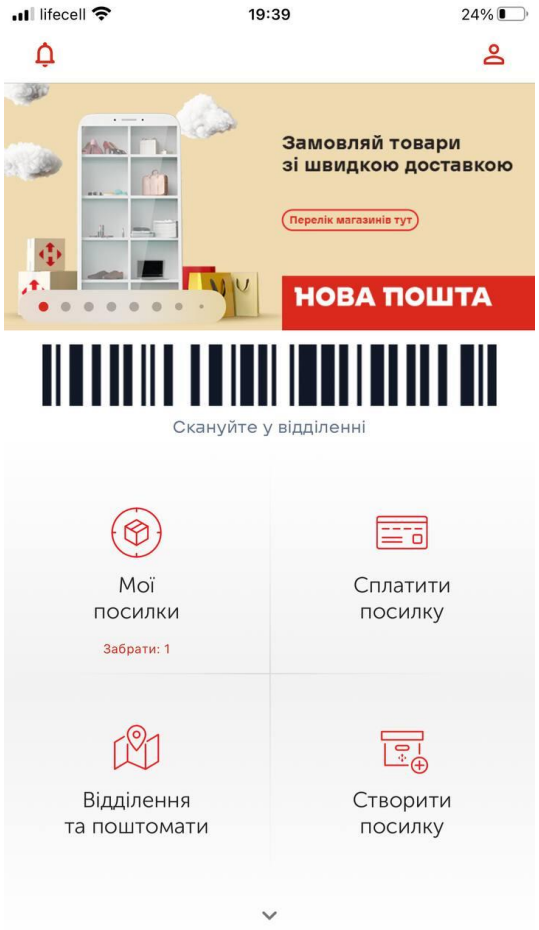

Рисунок 1.1 – Додаток «Нова Пошта»

Нова Пошта має досить зручний додаток. Ми особисто користуємось ним та можемо сказати, що на практиці він дійсно досить практичний та доволі простий у використанні. При вході у застосунок користувач може побачити головні розділи. Ми можемо переглянути інформацію про свої посилки, сплатити їх, ознайомитись з графіком та місцезнаходженням відділень або створити посилку для економії часу. Щодо оформлення тут також все гаразд, ми не можемо сказати, що в додатку є зайві елементи або незрозуміла структура. Отже, в цьому плані в них все гаразд.

Щодо недоліків, ми можемо наразі відмітити лише ціни за послуги. На сьогоднішній день ціни є досить не низькими. Якщо ваша посилка буде великих розмірів, вам доведеться заплатити значну суму за її доставку в інше місто.

Далі ми розглянемо також доволі популярну компанію з доставки (поштового зв'язку) «Укрпошта». Компанія працює майже по такому самому принципу як «Нова Пошта», але різниця між ними є, що ми й далі розберемо.

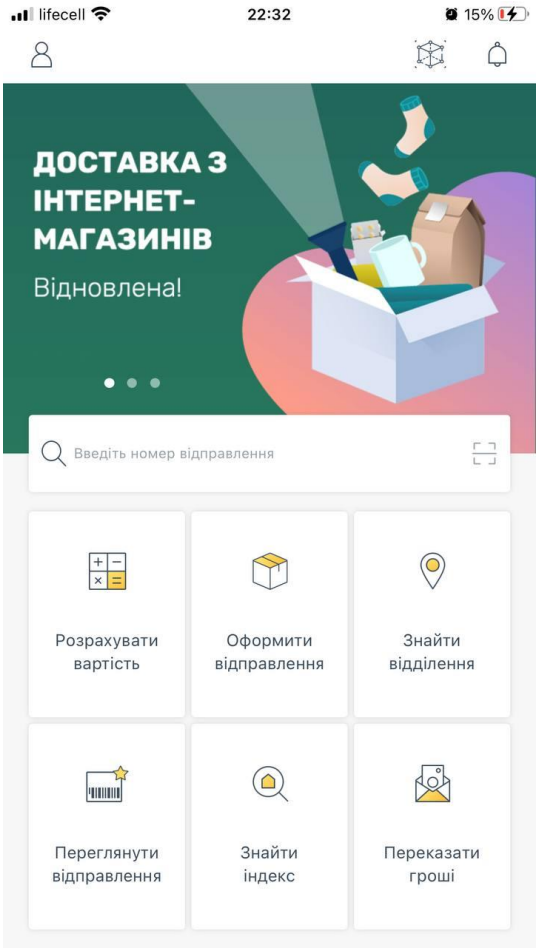

Рисунок 1.2 – Додаток «Укрпошта»

Почнемо з додатка. Виглядає він доволі непогано, на головній сторінці є відповідні розділи. Іконки та шрифт підібрано вдало. Реалізований доволі широкий асортимент послуг та функціонал з можливістю розрахунку вартості, оформлення відправлення та його перегляд. Але на практиці він менш зручний ніж додаток конкурента. Про що свідчить оцінка з AppStore: Нова Пошта – 4.7, а Укрпошта – 2.4. В цілому додаток розроблений з розумінням потреб користувача послуг.

Розбираючи недоліки даного сервісу, ми можемо виділити наступне: випадки з затримкою доставки та проблеми з додатком, про що свідчать коментарі з AppStore.

Ну і на останок ми розглянемо міжнародний онлайн-сервіс пошуку автомобільних попутчиків «BlaBlaCar».

Треба відмітити, що «BlaBlaCar» не пов’язаний з логістичними послугами, але це саме те, що ми хочемо реалізувати, але ми будемо шукати попутчиків не для власного переміщення, а для доставки нашого товару або будь чого іншого. Тому нам обов’язково потрібно розібрати даний онлайн-сервіс.

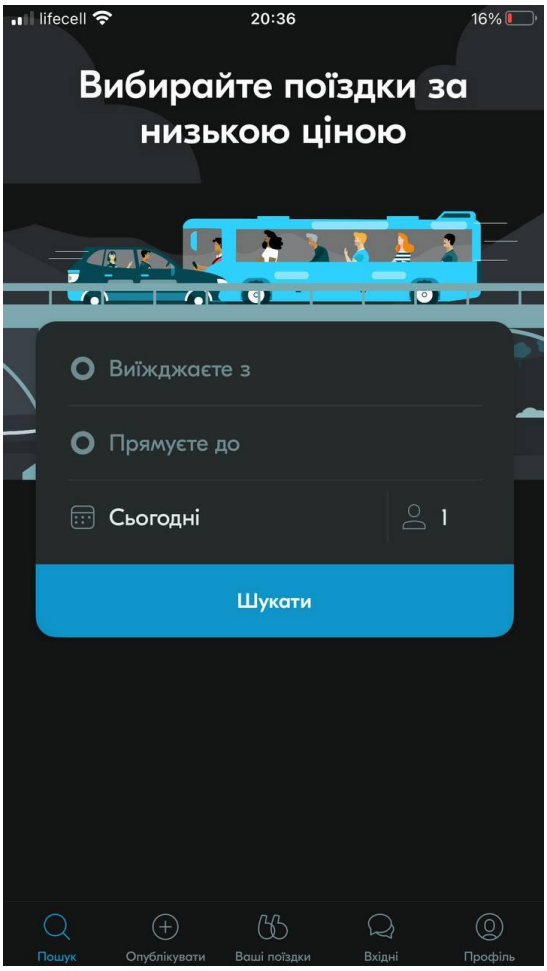

Рисунок 1.3 – Додаток «BlaBlaCar»

Даний сервіс також має доволі зручний мобільний застосунок. Тут все просто. Обираєте звідки вам потрібно виїхати, куди потрібно дібратись, обираєте дату, кількість місць та підбираєте варіанти для себе. Так і навпаки, ви можете опублікувати свою поїздку, і за допомогою даного сервісу підвезти людей з якими вам по дорозі. Потрібно відмітити зручність додатку, все працює швидко та належним чином. Підібраний сучасний та адаптивний стиль, який підкреслює всю структуру додатку. Знизу знаходяться всі головні розділи застосунку, що

дуже зручно для користувача. Також потрібно відмітити, що даний сервіс має свій внутрішній месенджер для комунікації з іншими користувачами додатку.

Щодо мінусів даного сервісу, ми можемо віднести лише те, що ми можемо потрапити на перевізника, який може вказати дещо хибну інформацію, запізнитись або зовсім не приїхати. Звичайно ми можемо подати на нього скаргу для подальшого його відсторонення, але все ж така ситуація може виникнути.

На основі визначених переваг та недоліків переглянутих додатків була складена таблиця для підбиття підсумків даного порівняння (Таблиця 1.1).

Таблиця 1.1

Порівняння розглянутих додатків

Назва	Нова Пошта	Укрпошта	BlaBlaCar
Інтерфейс	Зручний	Доволі зручний	Зручний
Функціонал	Широкий	Широкий	Достатній
Авторизація	Присутня	Присутня	Присутня
Стабільність роботи додатку	Хороша	Нормальна, але інколи виникають проблеми	Хороша
Швидкість	Чудова	Нормальна	Хороша
Реклама	Присутня	Відсутня	Присутня

Таким чином, основними вимогами щодо нового сервісу мають бути: наявність зручного та зрозумілого меню(інтерфейсу), стабільна робота сервісу та здійснення моніторингу якісного надання послуг.

1.3. Архітектура телеграм-сервісу з логістики

Архітектура телеграм бота описує смартфон, який спілкується з сервером Telegram, як міст між клієнтом і ботом сервера. Використовуються два основних терміни, а саме запити та оновлення. Сервер адміністратора виконує запити через сервер Telegram як міст між мобільним пристроєм і контрольованим сервером, потім сервер Telegram запускає процес опитування за протоколом HTTPS на

сервері, а бот-сервер надає оновлення з результатів запитів сервера адміністратора (рис.1.4).

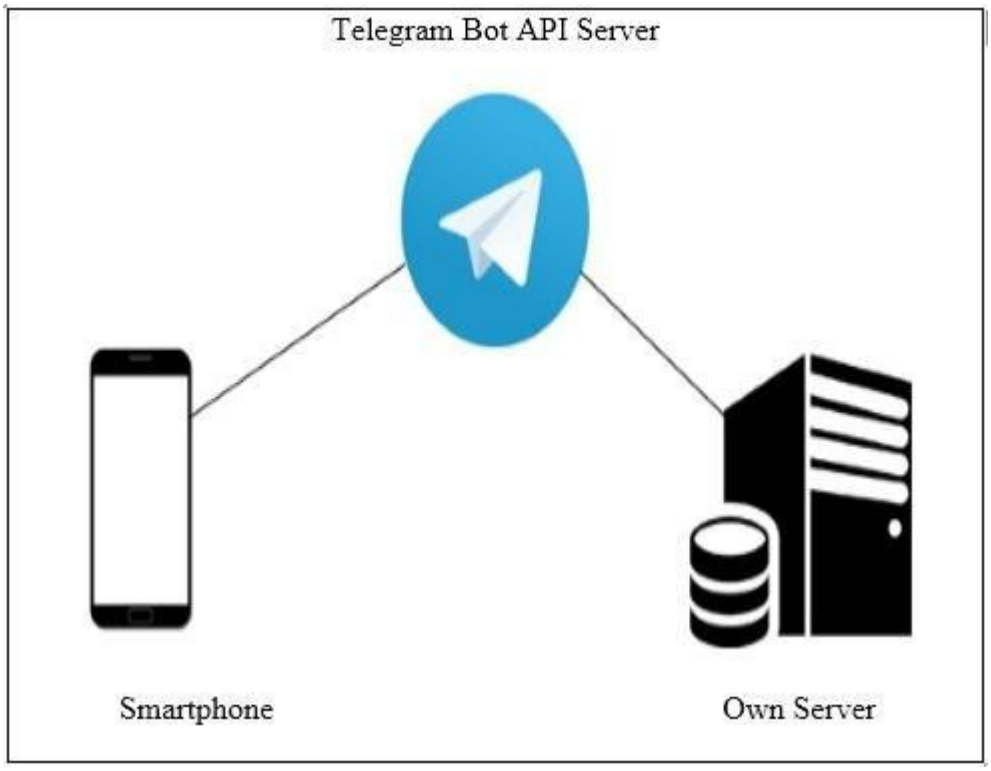

Рисунок 1.4 – Архітектура телеграм бота

Для цього дизайн виконується за допомогою API Telegram Bot як проміжного середовища для Admin Server. Telegram Bot API — це API (інтерфейс програмного програмування), який використовується для віддаленого моніторингу за допомогою ботів як програм, запущених на сервері. Telegram Bot API використовує протокол шифрування MTPproto, який був підготовлений для розробників як засіб безпеки.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для створення телеграм-сервісу з логістики було вирішено використовувати: мову розмітки гіпертекстових документів HTML, багатоцільову мову програмування Python, бібліотеку pyTelegramBotAPI, облачну PaaS-платформу Heroku, СУБД PostgreSQL та середовище програмування PyCharm.

Варто зупинитись на кожному з інструментів окремо і описати їх вклад в проект.

HTML (Hypertext Markup Language — Мова гіпертекстової розмітки) — це мова опису структури сторінок документів, яка дозволяє формотувати звичайний текст. Нам він корисний для створення абзаців та виділення жирним текстом для кращого сприйняття інформації.

Python — багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові програмування.

pyTelegramBotAPI (telebot) — спеціальна бібліотека для створення телеграм ботів на мові програмування Python. Це дуже зручний модуль для взаємодії з Telegram API. Розробники надали досить вичерпну документацію, тому з даною бібліотеко достатньо легко розібратись.

Heroku — це хмарний сервіс, де ви можете розгорнути свій застосунок, який має дуже простий користувацький інтерфейс, що дозволяє вам сфокусуватися на вивченні своєї платформи. Ще з великих переваг — те, що на Heroku можна розгортати свої застосунки безкоштовно. Саме за допомогою цієї хрманої платформи ми здійснимо деплой нашого ТБ.

PyCharm — інтегроване середовище розробки для мови програмування Python. Ми обрали саме це середовище через наступні можливості: статичний аналіз коду, підсвічування синтаксису і помилок, навігація серед проектів і сирцевого коду, вбудований відлагоджувач для Python, підтримка систем контролю версій та багато іншого, що дозволяє зручно створювати свій проект.

Порівняльний аналіз СУБД дозволяє раціонально обрати систему керування БД для проекту. В якості альтернативних варіантів будуть розглянуті такі СУБД як: MySQL, Microsoft SQL Server (надалі MS SQL) та PostgreSQL. Дві з яких безкоштовні – MySQL та PostgreSQL.

Усі вибрані СУБД підходять для проведення аналізу й порівняння, тому що реалізують реляційну модель даних. Максимально можливий обсяг збережених даних для кожної СУБД представлено у таблиці 1.2.

Таблиця 1.2

Максимально можливий обсяг збережених даних для кожної СУБД

	Розмір рядка	Розмір таблиці	Розмір БД
MySQL	64 KB	256 ТБ	∞
PostgreSQL	1.6 ТБ	32 ТБ	∞
MS SQL	∞	524258 ТБ	524258 ТБ

Критерій підтримування ОС наведено у таблиці 1.3.

Таблиця 1.3

Аналіз підтримуваних альтернативами операційних систем

	Windows	Linux	Unix	Android	Symbian
MySQL	+	+	+	+	+
PostgreSQL	+	+	+	-	+
MS SQL	+	-	-	-	-

Інформаційна система організації містить у собі чимало конфіденційної інформації. А для запобігання несанкціонованого доступу застосовуються різні способи та методи захисту. Проведемо порівняння систем забезпечення безпеки даних у таблиці 1.4 серед різних СУБД.

Таблиця 1.4

Аналіз систем забезпечення безпеки даних в альтернативах

	Ідентифікація	Захист від brute-force	Шифрування	Сертифікація безпеки
MySQL	+	-	+	-
PostgreSQL	+	+	+	+
MS SQL	+	-	+	+

Проведений аналіз даних СУБД (таблиця 1.5), таких як: MySQL, MS SQL та PostgreSQL показав, що для вирішення поставлених задач найбільше

підходить PostgreSQL. В нашому випадку PostgreSQL має ряд суттєвих переваг над іншими розглянутими СУБД, а саме: гнучкість, надійність та підтримка складних структур.

Для роботи з PostgreSQL ми будемо використовувати такий інструмент(IDE) як JetBrains DataGrip. Ми обрали саме DataGrip тому, що він має ряд переваг, таких як: першокласна підтримка баз даних, автоматизація рутинних задач та представлення розумних функцій IDE для продуктивної роботи.

Таблиця 1.5

Аналіз загальних показників СУБД

	MySQL	PostgreSQL	MS SQL
Логічна модель даних	Реляційна	Постреляційна (об'єктно-реляційна)	Реляційна
Фізична модель даних	Сторінкова		
Типи даних	Всі основні	Всі основні, довільні, множинні значення	Всі основні, розширені
Індекси	Всі основні, повнотекстовий	Звичайні, повнотекстовий	Всі основні, повнотекстовий
Мови маніпулювання	SQL	Розширений SQL	SQL, QBE, XQuery, багатомірні вирази (для OLAP)
Вбудовані мови програмування	ANSI C, ANSI C++	C, C++, Python, Java, Perl, PHP, .Net та ін	MS Visual Basic, C#
Генератор форм, звітів	Ні		Засоби побудови звітів
Транзакції	Так (в тому числі розподілені)	Так	
Тригери, процедури, що зберігаються	Так	Так (підтримка декількох мов програмування)	Так
Платформи	MS Windows, Unix, Linux, MacOS X, Novell NetWare та ін.	MS Windows, Linux, FreeBSD, Solaris, MacOS	Тільки MS Windows
Область застосування	Інформаційні системи масштабу підприємства		

Особливості	Можливість логічного об’єднання БД. Широкий вибір платформ. Реплікація	Відкриті вихідні тексти. Механізм наслідування. Масштабування. Реплікація.	Масштабування. Кластери. Реплікації. Розширена підтримка XML

Отже, було обрано для створення програмного комплексу такі інструменти та технології:

- мову розмітки гіпертекстових документів HTML5;
- багатоцільову мову програмування Python;
- бібліотеку pyTelegramBotAPI;
- облачну PaaS-платформу Heroku;
- середовище керування базами даних PostgreSQL;

Висновки до розділу 1

Проведений аналіз предметної області дозволив визначити основні можливості та функціонал телеграм-сервісу з логістики. З огляду на аналоги можна визначити такі головні особливості сервісу: доступність сервісу, спрощена реєстрація, можливість додавання авто та його редагування, перевірка введених даних та сповіщення при відповідних змінах.

Визначившись з архітектурою, було підібрано необхідний інструментарій та набір програм для розробки програмного продукту. Сюди входять: мова розмітки гіпертекстових документів HTML, багатоцільова мова програмування Python, бібліотека pyTelegramBotAPI, облачна PaaS-платформа Heroku та СУБД PostgreSQL.

Обрано наступні програмні продукти та інтегровані середовища розробки:

- середовище розробки PyCharm Community Edition 2019.3.2 ;
- функціональний пакет інструментів для роботи з БД JetBrains DataGrip;

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТЕЛЕГРАМ-СЕРВІСУ З ЛОГІСТИКИ

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи

Наш телеграм-сервіс ставить перед собою наступну ціль: знайти попутчика для людини, яка хоче через нього передати свою посилку чи щось інше задля економії часу та грошей. Майбутній сервіс має бути зручним як для виконавця, так і для замовника. Має бути присутня чітка інформація щодо самого перевезення, можливість вибору та перегляд стадій перевезення посилки.

Як вже зазначалось раніше, ми прагнено розробити аналог міжнародного-сервісу «BlaBlaCar», але з тією різницею, що попутчик потрібний буде для перевезення посилки.

Вимоги користувачів

1. Користувач, що не додав номер телефону
   1.1.Можливість перегляду оголошень
2. Користувач, що надав номер телефону
   2.1.Можливість додавання авто
   2.2.Можливість публікації оголошення
   2.3.Можливість приймати оголошення
   2.4.Можливість редагуваня інформації щодо авто
   2.5.Можливість перегляду актуальної інформації щодо опублікованого оголошення

Завдяки проведеному аналізу вимог до програмного забезпечення, було побудовано діаграму варіантів використання системи. Її зображено на рисунку 2.1.

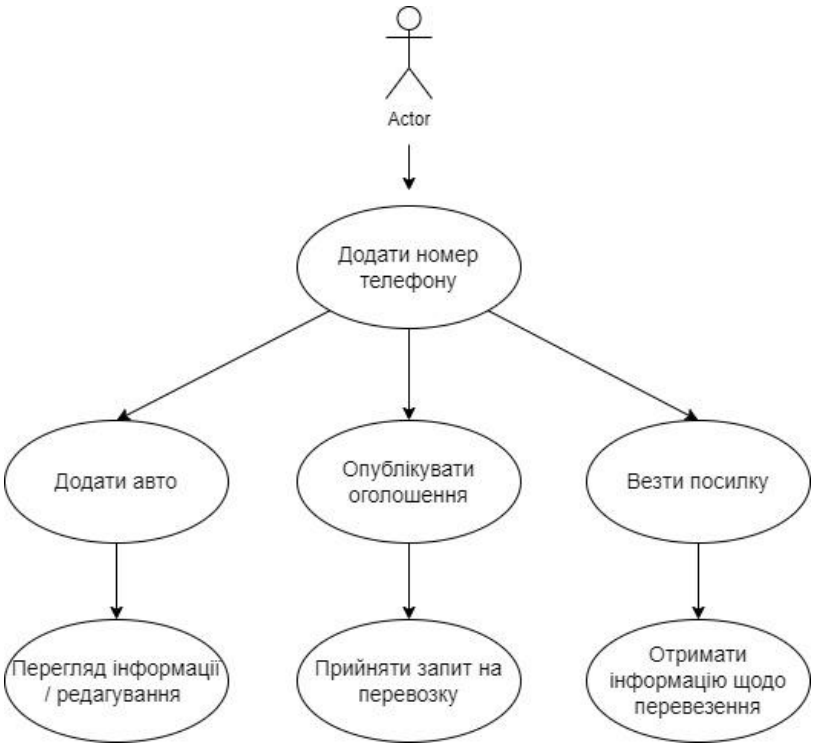

Рисунок 2.1 – Варіанти використання телеграм-сервісу

Функціональні вимоги:

1. Можливість повноціної реєстрації при надсиланні власного мобільного телефону.

2. Додавання авто та розміщення оголошень лише для користувачів, які надали свій номер телефону

3. Редагування, видалення та додавання авто.

4. Збереження оголошень в Базу даних.

5. Перегляд актуальної інформації щодо етапу перевозки оголошення.

Нефункціональні вимоги:

1. Навчання роботи з сервісом:

– Час, потрібний для ознайомлення з телеграм-сервісом для звичайних користувачів – 10-15 хвилин, а для досвідчених – 5-10 хвилин;
– Час відповіді системи для звичайних запитів не повинен перевищувати 1 секунду, а для більш складних запитів – 3 секунд;
– Інтерфейс телеграм-сервісу та головне меню повинно бути зрозумілим та зручним для всіх користувачів;

2. Продуктивність

Телеграм-сервіс має безперебійно працювати при 500 активних користувачів.

3. Надійність

– Доступність – час, потрібний для обслуговування системи не повинен перевищувати 10% від загального часу роботи;
– Середній час безперервної роботи – 10 - 15 робочих днів;
– Максимальна норма помилок та дефектів в роботі системи – 1 помилка на 1000 запитів користувача.

В проекті передбачено три класи, а саме: CarInfo, ParcelInfo та OfferParcel. В них передбачені спеціальні методи, які автоматично виконуються при створенні кожного нового екземпляра на основі відповідного класу.

Клас **CarInfo** – передбачений для створення екземляру на освнові введених даних про автомобіль відповідного користувача, після чого екземпляр заноситься в БД та видаляється (рис2.2).

```
class CarInfo:
    def __init__(self, car_brand):
        self.car_brand = car_brand
        self.car_model = None
        self.car_type = None
        self.car_color = None
        self.car_year = None
        self.car_number = None
        self.car_description = None
        self.id_car = None
```

Рисунок 2.2 – Клас CarInfo

Клас **ParcelInfo** - передбачений для створення екземляру на освнові введених даних при публікації оголошення відповідного користувача, після чого екземпляр заноситься в БД та видаляється за індетифікатором користувача(рис2.3).

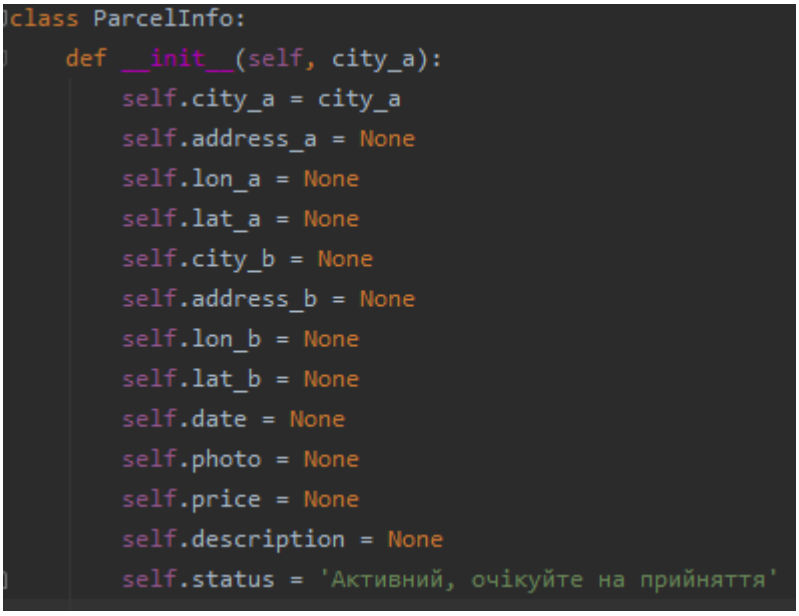

Рисунок 2.3 – Клас ParcelInfo

Клас **OfferParcel** - передбачений для створення екземляру на освнові запиту одного користувача на оголошення іншого, після чого екземпляр заноситься в БД та видаляється при підтвердженні запиту(рис2.3).

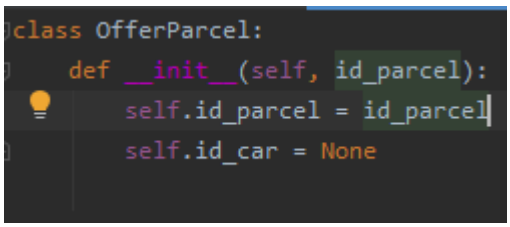

Рисунок 2.4 Клас OfferParcel

2.2. Розробка бази даних системи

Реалізована БД PostgreSQL складається з 4 таблиць, які містять у собі всі необхідні дані для повноцінної роботи ТБ. База даних має назву tudy-sudy_data.

Таблиця 2.1

Інформація про таблиці бази даних tudy-sudy_data

Назва таблиці	Призначення
car_info	Повна інформація про автомобіль відповідного користувача
parcel_info	Інформація щодо опублікованого оголошення користувачем
users	Зареєстровані користувачі сервісу
offer_parcel	Запити на перевезення відповідного оголошення

Схема Бази даних телеграм-сервісу з логістики наведена на рисунку 2.3.

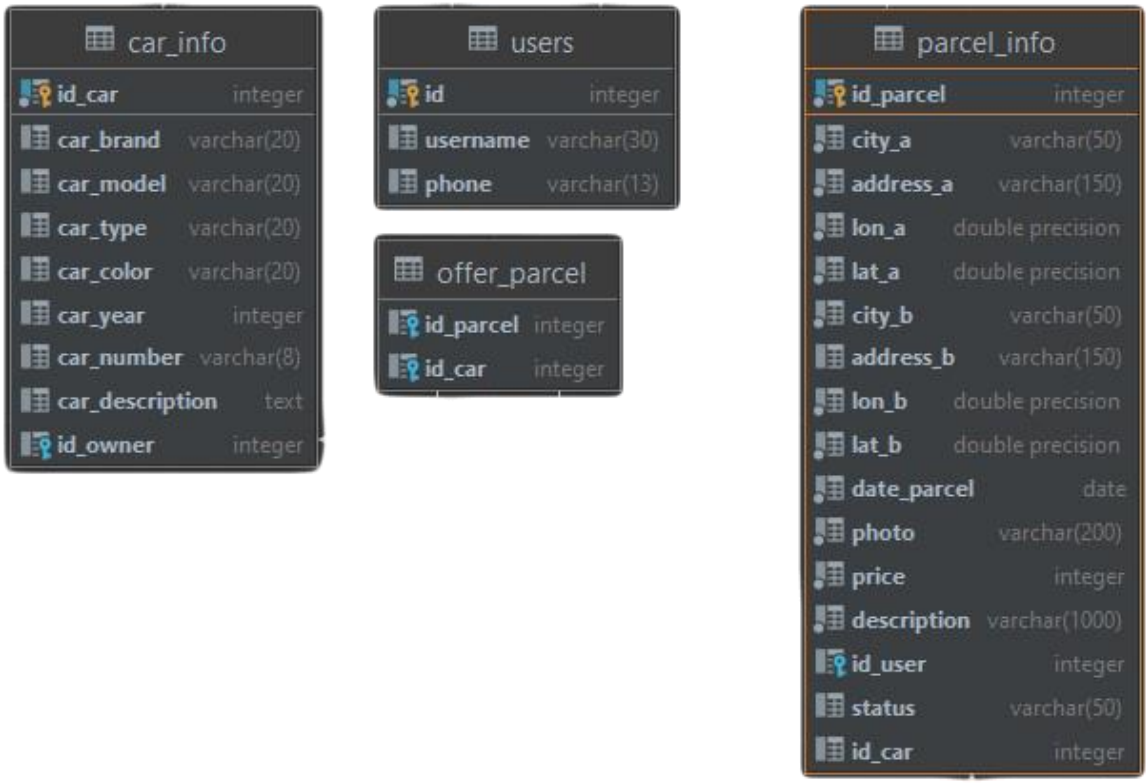

Рисунок 2.3 – Схема БД

Далі окремо опишемо таблиці реалізованої Бази даних.

Таблиця «car_info» зберігає повну інформацію про автомобіль. В цій таблиці зберігаються наступні дані: id_car(ідентифікатор авто), car_brand (марка автомобіля), car_model (модель авто), car_type (тип кузова), car_color(колір авто), car_year(рік випуску авто), car_number(державний номер авто), car_description(короткий поис автомобіля) та id_owner(ідентифікатор власника). Структура таблиці наведена у таблиці 2.2.

Таблиця 2.2

Опис полів таблиці «car_info»

Назва	Тип даних	Опис поля
id_car	int	Ідентифікатор автомобіля
car_brand	varchar(20)	Марка автомобіля
car_model	varchar(20)	Моедль автомобіля

car_type	varchar(20)	Тип кузова автомобіля
car_color	varchar(20)	Колір автомобіля
car_year	int	Рік випуску автомобіля
car_number	varchar(8)	Державний номер автомобіля
car_description	text	Короткий опис автомобіля
id_owner	int	Ідентифікатор власника автомобіля

Таблиця «parcel_info» є найбільшою і містить в собі інформацію про розміщені оголошення від користувачів. В цій таблиці зберігаються наступні дані: id_parcel (ідентифікатор оголошення), city_a (місто відправлення), address_a (адрес відправлення), lon_a (координати місця відправлення: довгота), lat_a (координати місця відправлення: широта), city_b (місто прибуття), address_b (адреса прибуття), lon_b (координати місця прибуття: довгота), lat_b (координати місця прибуття: широта), date_parcel (дата доставки), photo(зображення посилки), price(ціна за доставку), description (короткий опис посилки), id_user (власник оголошення), status (стадія перевезення посилки), id_car (ідентифікатор автомобіля, який перевозить посилку). Структура таблиці наведена нижче у таблиці 2.3.

Таблиця 2.3

Опис полів таблиці «parcel_info»

Назва	Тип даних	Опис поля
id_parcel	int	Ідентифікатор оголошення
city_a	varchar(50)	Місто відправлення
address_a	varchar(150)	Адреса відправлення
lon_a	float	Координати місця відправлення: довгота
lat_a	float	Координати місця відправлення: широта
city_b	varchar(50)	Місто прибуття
address_b	varchar(150)	Адреса прибуття
lon_b	float	Координати місця прибуття: довгота
lat_b	float	Координати місця прибуття: широта
date_parcel	date	Дата відправлення
photo	varchar(200)	Зображення посилки

price	int	Ціна за доставку посилки
description	varchar(1000)	Короткий опис посилки
id_user	int	Ідентифікатор власника оголошення
status	varchar(50)	Стадія перевезення посилки
id_car	int	Ідентифікатор автомобіля

Таблиця «users» містить в собі інформацію про зареєстрованих користувачів. В цій таблиці зберігаються наступні дані: id (ідентифікатор користувача), username (нікнейм користувача) та phone (мобільний телефон користувача). Структура таблиці представлена у таблиці 2.4.

Таблиця 2.4

Опис полів таблиці «users»

Назва	Тип даних	Опис поля
id	int	Ідентифікатор користувача
username	varchar(30)	Нікнейм користувача
phone	varchar(13)	Мобільний телефон користувача

Таблиця «offer_parcel» містить в собі інформацію про запити на перевезення відповідної посилки. В цій таблиці зберігаються наступні дані: id_parcel (ідентифікатор оголошення) та id_car (ідентифікатор автомобіля), її структура у таблиці 2.5.

Таблиця 2.5

Опис полів таблиці «offer_parcel»

Назва	Тип даних	Опис поля
id_parcel	int	Ідентифікатор оголошення
id_car	int	Ідентифікатор автомобіля

Отже реалізована БД забезпечує збереження, редагування та видалення інформації телеграм-сервісу. Вона складається з 4-ох таблиць: users, car_info, parcel_info та offer_parcel.

2.3. Реалізація телеграм-сервісу з логістики

Перед тим, як розпочинати розробку нашого телеграм бота, ми повинні отримати спеціальний токен від головного бота Telegram – BotFather. За допомогою команди /newbot ми можемо створити нового бота, після чого нам потрібно ввести його назву та юзернейм (з закінченням _bot). Бот вітає нас, присилає деяку інформацію та найголовніше наш токен (рис.2.4).

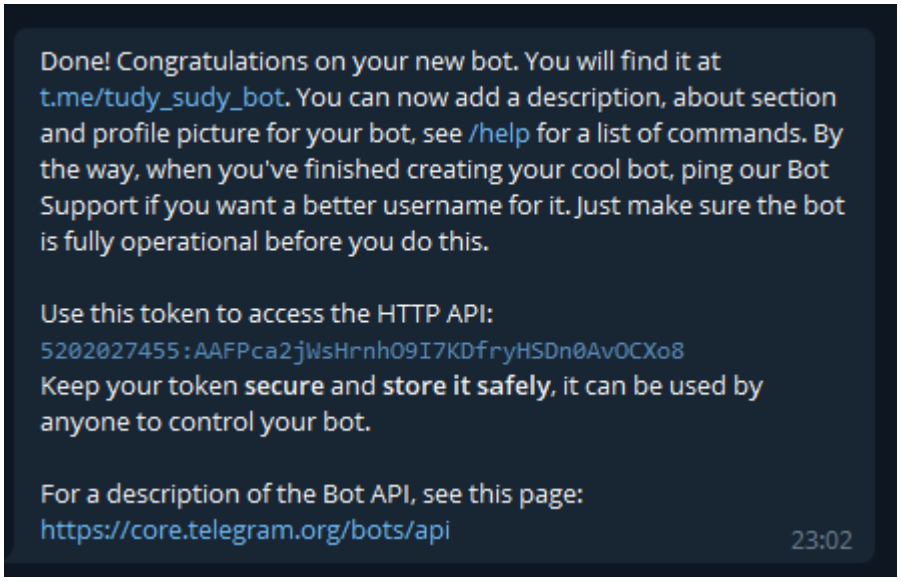

Рисунок 2.4 – Отримання токену

Для реалізації телеграм-сервісу з логістики було використано бібліотеку pyTelegramBotAPI (telebot). Ми її встановили за допомогою команди $ pip install pyTelegramBotAPI та імпортували в наш проект.

Спочатку створимо окремий файл під назвою configure.py для зберігання токену та в подальшому відповідних змінних(рис.2.5).

```
config = {
    'name': 'tudy_sudy_bot',
    'token': '5202027455:AAFPca2jWsHrnhO9I7KDfryHSDn0AvOCXo8',
    'db_URI': 'postgres://wrssjwwtnmfinf:1bff9e66522facf76187145b5d4e3f395b789860bea2ad9bcdf4763cc129579f@ec2-63-35'
              '-156-160.eu-west-1.compute.amazonaws.com:5432/dcnagi5ov45ph2',
}
```

Рисунок 2.5 – Зміст файлу configure.py

Надалі ми створимо екземляр класу telebot (обгортку над Telegram API) для того, щоб ми могли надсилати запити на сервер Telegram (рис.2.6).

```
bot = telebot.TeleBot(configure.config["token"])
```

Рисунок 2.6 – Створення екземпляру класу telebot

Екземпляр класу ми створили, тепер потрібно запустити нашого бота, це ми можеро реалізувати за допомогою відповідної команди (рис.2.7)

```
bot.polling(none_stop=True, interval=0)
```

Рисунок 2.7 – Команда запуску ТБ

Надалі для обробки повідомлень від користувача ми будемо використовувати спеціальні хендлери. Обробники повідомлень визначають фільтри, які повинно пройти повідомлення. Якщо повідомлення проходить фільтр, викликається декорована функція, а вхідне повідомлення передається як аргумент. Давайте розглянемо основні хендлери нашого ТБ у таблиці 2.6

Таблиця 2.6

Обробники повідомлень ТБ

Обробник повідомлення	Коли він спрацьовує та що в собі містить

`@bot.message_handler(commands=['start', 'help'])`
`def send_welcome(message):`
`  id_user = message.from_user.id`
`  username = message.from_user.username`
 	
`  cursor.execute(f'SELECT id FROM users WHERE id = {id_user}')`
`  result = cursor.fetchone()`
`  print(result)`
 	
`  if not result:`
`  sql = 'INSERT INTO users(id, username) VALUES(%s, %s)'`
`  val = (id_user, username)`
`  cursor.execute(sql, val)`
`  db_connection.commit()`
`  print(cursor.rowcount, "record inserted.")`
 	
`  bot.send_message(message.chat.id, f'Вітаємо, <b>{message.from_user.first_name}</b>',`
 	
`reply_markup=main_markup(message.from_user.id), parse_mode="HTML")`
	Даний обробник спрцьовує, коли користувач вперше розпочинає роботу з нашим ТБ або вводить команди /start та /help.
	Надалі бот перевіряє чи є даний користувач в нашій БД, якщо ж не має, то він додає його.

```
@bot.message_handler(func=lambda message: True)
def callback(message):
    cursor.execute(f'SELECT phone FROM users WHERE id =
{message.from_user.id}')
    phone = cursor.fetchone()[0]

    request_contact_markup =
types.ReplyKeyboardMarkup(resize_keyboard=True)

request_contact_markup.add(types.KeyboardButton('Надіслат
и номер телефону 📲', request_contact=True))

request_contact_markup.add(types.KeyboardButton('Назад🔙'
))

    if message.text == "Опублікувати оголошення 🗨️":
        if not phone:
            msg = bot.send_message(message.chat.id,
f'Перед тим як опублікувати оголошення, додайте свій
номер '

f'телефону 📲',

reply_markup=request_contact_markup)
            bot.register_next_step_handler(msg,
add_phone_advertisement)
        else:
            msg = bot.send_message(message.chat.id, f'В
якому місті знаходиться посилка?🛍️')
            bot.register_next_step_handler(msg,
parcel_city_a)

    elif message.text == "Додати своє авто 🚗":
...
```

Цей хендлер створений для того, щоб оброблювати введенні повідомелння від користувача.

В нашому випадку він нам знадобиться для того, щоб відпрацьовувати нажаття кнопок головного меню.

Для прикладу розберемо, що відбувається, коли користувач обрав розділ для публікації оголошення. Спочатку ТБ перевіряє чи користувач надав свій номер телефону, якщо ж ні, то він просить його про це.

Якщо ж користувач надав свій номер телефону, після цього спрацьовує функція `register_next_step_handler`, вона очікує ввід даних від користувача, після чого перевіряє їх та зберігає.

```
@bot.callback_query_handler(func=lambda call: True)
def callback(call):
    chat_id = call.message.chat.id
    message_id = call.message.message_id
    owner_id = call.from_user.id

    cursor.execute(f'SELECT id_car FROM car_info WHERE
id_owner = {call.from_user.id}')
    cars_id = cursor.fetchall()

    if call.data in cars_type:
        # noinspection PyBroadException
        try:
            car_info = car_info_dict[call.from_user.id]
            car_info.car_type = cars_type[call.data]

            msg = bot.send_message(call.message.chat.id,
f'<b>Якого кольору ваше авто?</b>', parse_mode='HTML')
            bot.register_next_step_handler(msg,
car_color_next_step)
        except Exception as e:
            print(e)
```

Цей обробник створений для реагування на натиснення інлайнових кнопок(ті, які прикріплені з повідомленням).

В даному прикладі ми хочемодізнатись який тип кузова обрав користувач так, як всі типи кузова ми виводимо за допомогою інлайнових кнопок.

```
            bot.reply_to(call.message, "Щось пішло не
так...")

    elif call.data in str(cars_id):
...
```

Також потрібно відмітити, що ми використовуємо адаптер бази даних Psycopg. Його основними характеристиками є повна реалізація специфікації Python DB API 2.0 і безпека потоків (кілька потоків можуть використовувати одне з'єднання). Отже, ми встановили бібліотеку psycopg2. Та в подальшому саме через неї добавляли, редагували та видаляли відповідні дані в БД (рис.2.8).

```
db_connection = psycopg2.connect(configure.config["db_URI"], sslmode="require")
cursor = db_connection.cursor()
```

Рисунок 2.8 – Створення екземпляру класу psycopg2 та cursor до нього

Також наведемо приклад вставки даних за допомгою даного екземпляру (рис.2.9).

```
sql = 'INSERT INTO users(id, username) VALUES(%s, %s)'
val = (id_user, username)
cursor.execute(sql, val)
db_connection.commit()
print(cursor.rowcount, "record inserted.")
```

Рисунок 2.9 – Вставка даних

Висновки до розділу 2

На даному етапі були створені відповідні класи для подальшого збереження їх екземплярів в Базу даних.

Спроектовано БД, що відповідає за збереження та колективний доступ до інформації сервісу. Реалізована БД забезпечує збереження та редагування інформації телеграм-сервісу. Вона складається з 4-х таблиць: users, car_info, parcel_info та offer_parcel.

Таблиця «parcel_info» є найбільшою і містить в собі інформацію про опубліковані оголошення користувачів сервісу. В цій таблиці зберігаються

наступні дані: id_parcel (ідентифікатор оголошення), city_a (місто відправлення), address_a (адрес відправлення), lon_a (координати місця відправлення: довгота), lat_a (координати місця відправлення: широта), city_b (місто прибуття), address_b (адреса прибуття), lon_b (координати місця прибуття: довгота), lat_b (координати місця прибуття: широта), date_parcel (дата доставки), photo(зображення посилки), price(ціна за доставку), description (короткий опис посилки), id_user (власник оголошення), status (стадія перевезення посилки), id_car (ідентифікатор автомобіля, який перевозить посилку).

В останньому пункті розділу наводиться детальний розгляд реалізованого коду нашого телеграм бота.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ТЕЛЕГРАМ-СЕРВІСОМ

3.1. Структура інтерфейсу. Інтерфейс та порядок роботи з телеграм-сервісом з логістики

На цій стадії виконання дипломного проекту буде поетапно продемонстровано користування телеграм ботом.

При початку роботи з телеграм ботом, він нас вітає та надсилає головне меню для подальшого вибору наших потреб в даному сервісі.

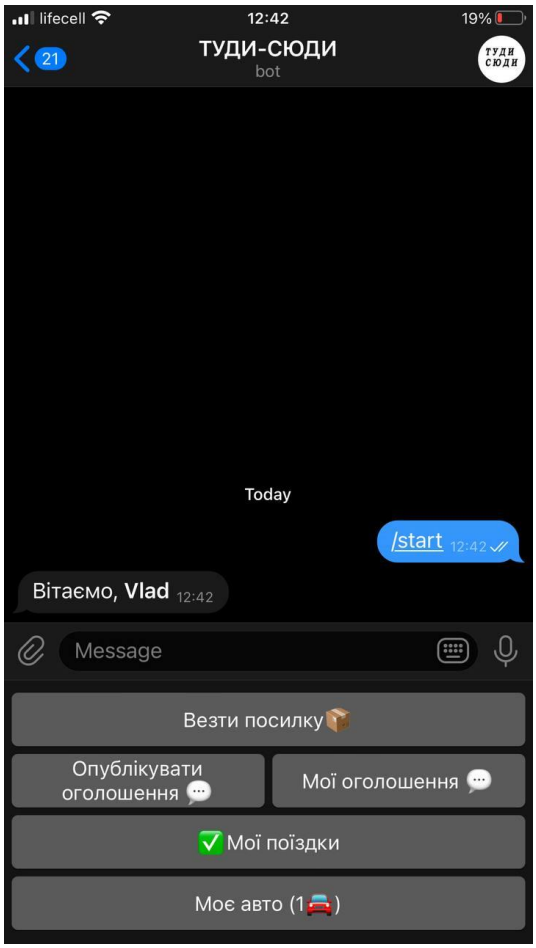

Рисунок 3.1 – Початок роботи з ботом

Головне меню ТБ складається з таких розділів як:

- Везти посилку – розділ, в якому ми в ролі перевізника можемо доставити посилку

- Опублікувати оголошення – розділ, в якому ми можемо опублікувати власне оголошення задля перевезення нашої посилки попутчиком
- Мої оголошення – розділ, в якому ми можемо переглянути власні оголошення, запити на їх перевезення та етапи перевезення посилки
- Мої поїздки – розділ, в якому ми помежо переглянути необхідну нам інформацію для перевезення схваленої посилки
- Моє авто / Додати своє авто – розділ, в якому ми можемо переглянути, редагувати інформацію про своє авто або взагалі видалити його. Якщо ви ще не додали авто в меню замість розділу «Моє авто» буде присутній розділ «Додати авто».

Перед тим, як отримати доступ до таких розділів, як «Опублікувати оголошення» та «Додати своє авто», користувачу потрібно надати свій номер телефону для закінчення реєстрації в ТБ та повноцінного використання ТБ (рис.3.2).

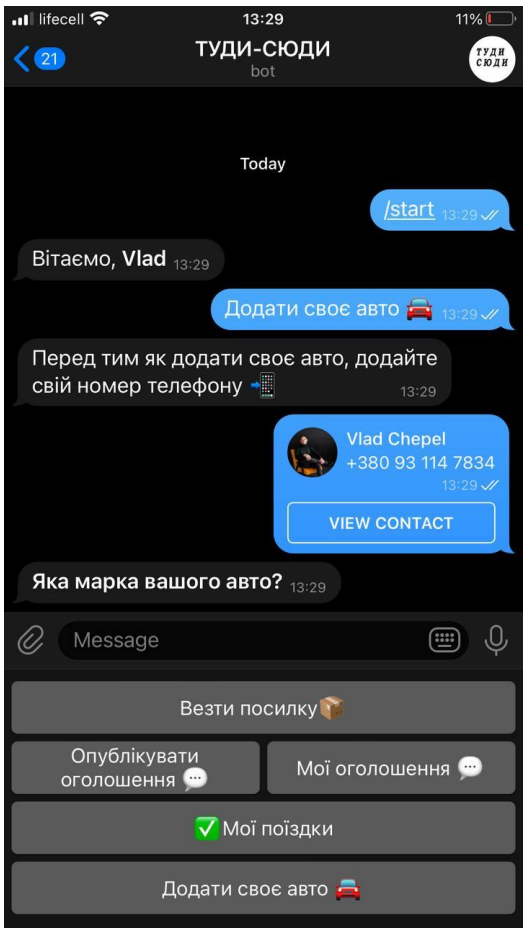

Рисунок 3.2 – Запит на надсилання номеру телефону

Після того, як користувач надав свій номер телефону, він має доступ до усіх розділів ТБ. Розпочнемо з розділу «Додати своє авто» (рис.3.3).

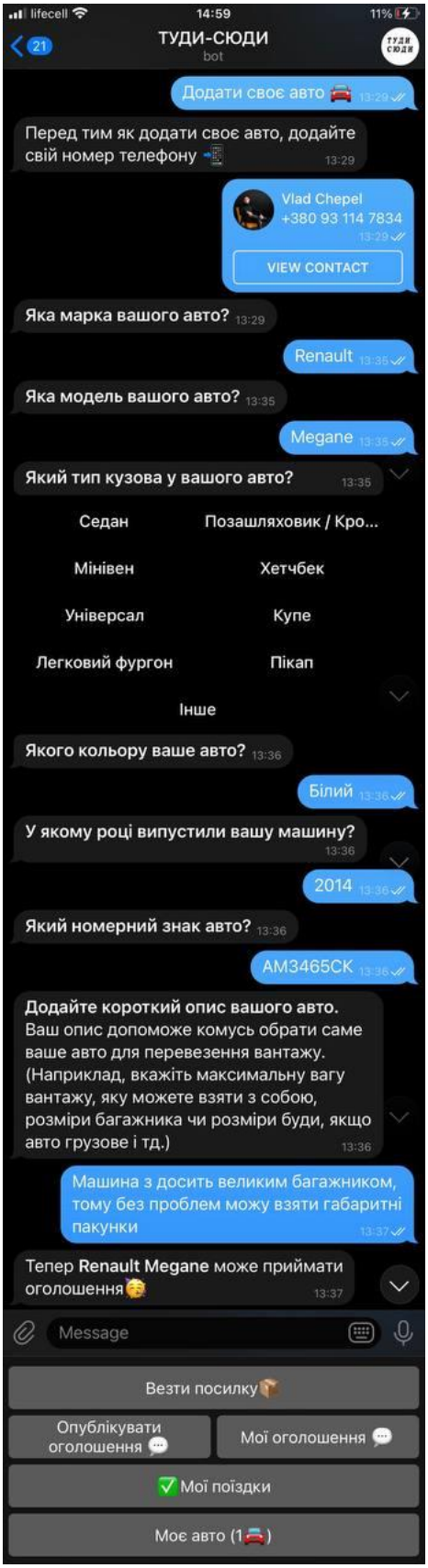

Рисунок 3.3 – Додання власного авто

ТБ у вигляді запитань хоче дізнатися у користувача повну інформацію щодо його авто, після вдалої перевірки введених значень авто додається у БД та виводиться повідомлення про те, що авто успішно додане та може приймати оголошення. Також тепер розділ «Додати своє авто» змінюється на «Моє авто» та в дужках сказується кількість доданих авто.

Давайте перейдемо до розділу «Моє авто».

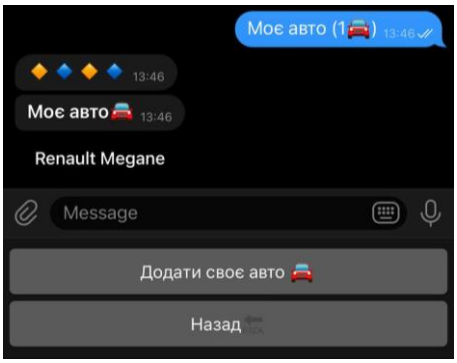

Рисунок 3.4 – Розділ «Моє авто»

При відкритті даного розділу виводиться перелік доданих авто. При натисканні на відповідне авто виводиться повна інформація з можливістю редагування та видалення автомобіля (рис.3.5).

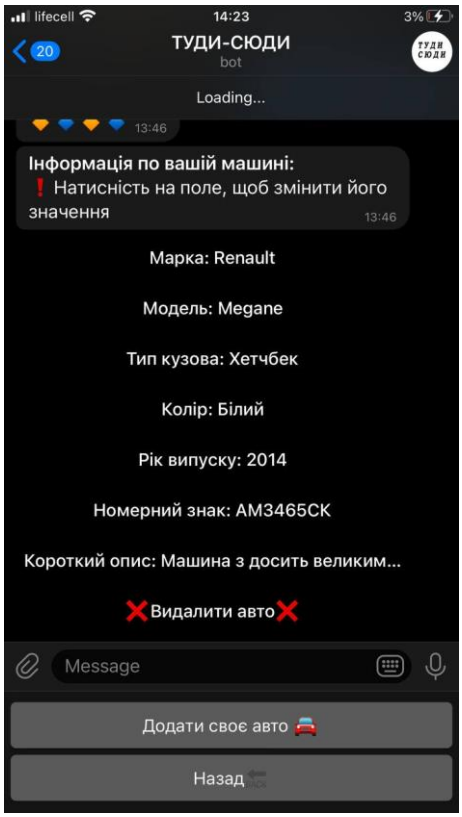

Рисунок 3.5 – Інформація по вибраному авто

Також варто відмітити те, що в даному випадку змінюється меню. Нам пропонується додати ще одне авто або вийти в головне меню ТБ.

Тепер давайте перейдемо до розділу «Опублікувати оголошення» (рис.3.6).

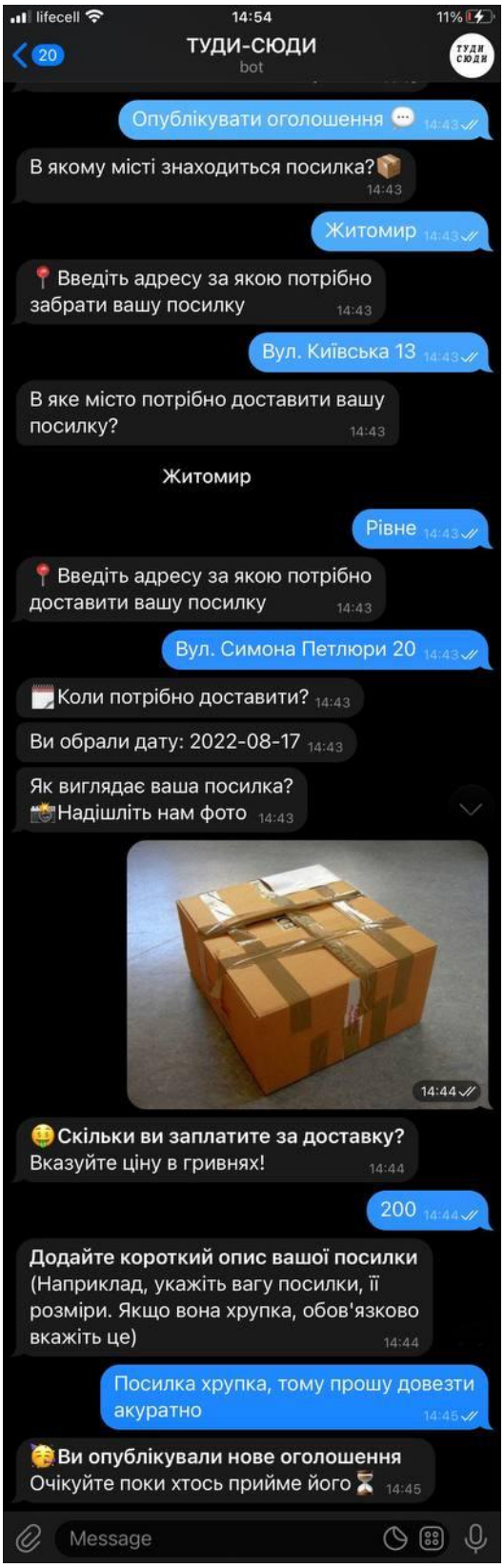

Рисунок 3.6 – Публікація оголошення

В даному випадку ТБ також у вигляді запитань дізнається всю інформацію про ваше оголошення. Опитування включає в себе доволі інформативні та чіткі пункти для того, щоб попутчик отримав всю необхідну йому інформацію для подальшого подання запиту на перевезення посилки.

Тепер можемо перейти в розділ «Мої оголошення» та ортимати перелік наших оголошень (рис.3.7).

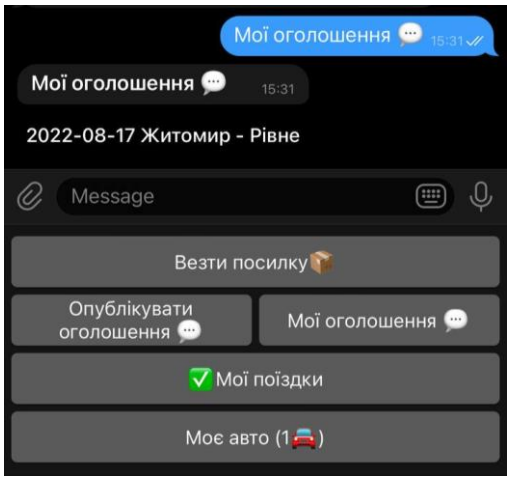

Рисунок 3.7 – Мої оголошення

Перелік виводиться у форматі дата, місто відправлення та місто прибуття. Якщо місто відправлення та місто прибуття співпадають, тоді перелік виводиться у форматі дата та місто.

При натисканні на відповідне оголошення, виводиться вся надана вами інформація, статус та можливість видалення посилки (рис3.8).

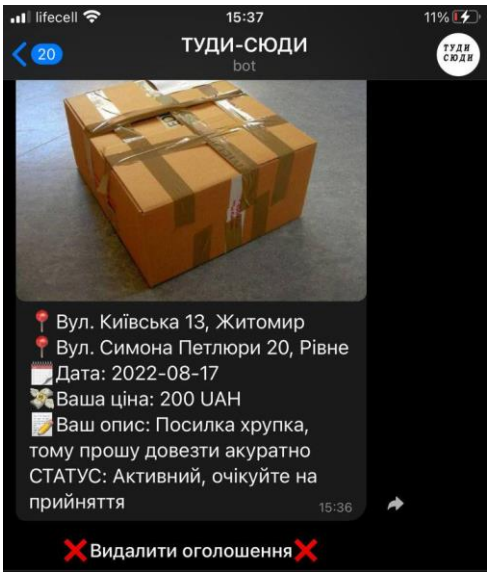

Рисунок 3.8 – Повна інформація власного оголошення

Наразі розглянемо розділ «Везти посилку». Це розділ для тих користувачів, які виступають в ролі попутчиків та можуть перевезти чиюсь посилку (рис.3.9).

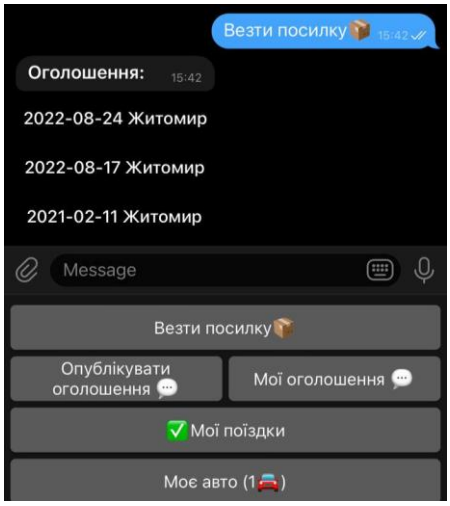

Рисунок 3.9 – Перелік доступних оголошень

При натисканні на даний розділ, ТБ надсилає нам опубліковані оголошення інших користувачів, яким потрібно доставити свою посилку. Перелік виводиться у форматі дата, місто відправки та місто прибуття. Якщо виводиться лише одне місто, це означає, що посилку потрібно доставити в межах міста. При переході на відповідне оголошення, ми отримуємо всю необхідну інформацію про посилку (рис.3.10).

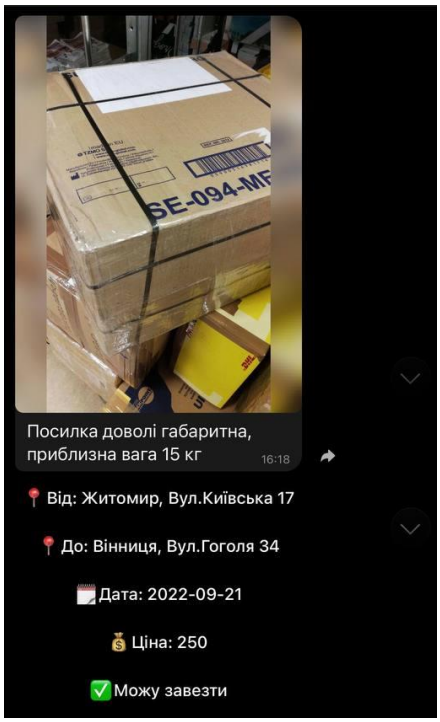

Рисунок 3.10 – Інформація про посилку

Тут наявна кнопка «Можу завезти», при натисканні на дану кнопку надсилається список доданих авто (рис.3.11).

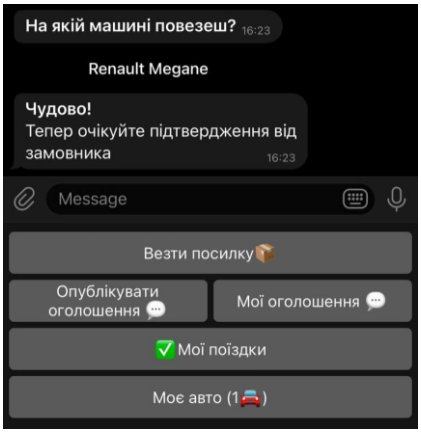

Рисунок 3.11 – Надсилання запиту на перевезення посилки

З даного списку ми обираємо авто, на якому ми можемо завезти посилку та отримуємо повідомлення, що запит успішно створений. Тепер нам потрібно очікувати підтвердження від власника.

Після того, як нам схвалили наш запит на перевезення, нам надходить повідомлення про це. Після чого, ми переходимо в розділ «Мої поїздки» та бачимо, що воно там з’явилось (рис.3.12).

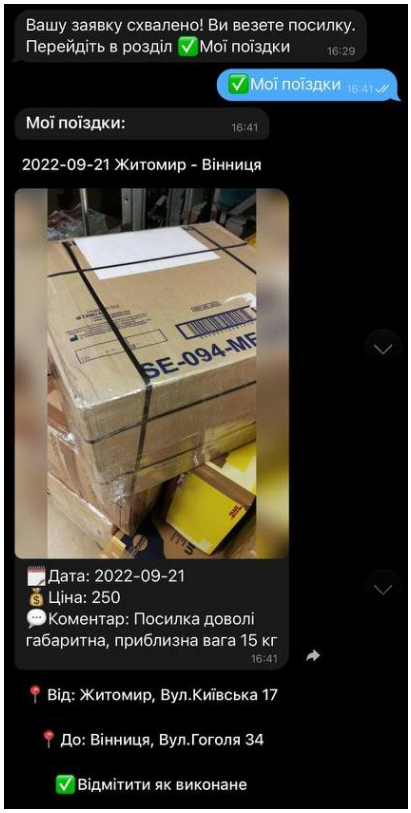

Рисунок 3.12 – Мої поїздки

Також потрібно відмітити, що при натисканні як на адресу відправки так і на адресу доставки нам надходить повідомлення з точним місцезнаходженням посилки на мапі (рис.3.13).

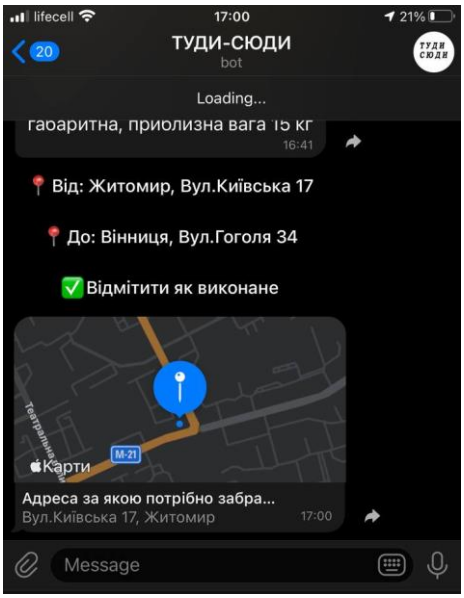

Рисунок 3.13 – Місцезнаходження посилки

Ми можемо натиснути на дану точку та відкрити її на зручній для нас мапі (рис.3.14).

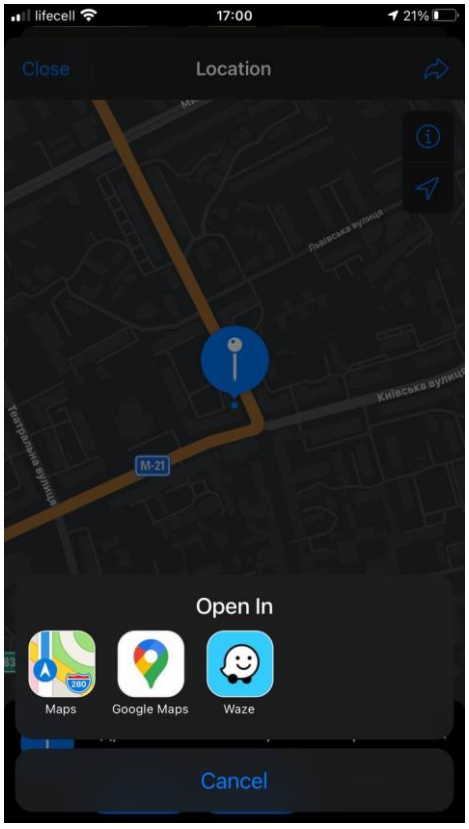

Рисунок 3.14 – Відкриття за допомогою мапи

3.2. Тестування роботи сервісу

Після реалізації проекту ми зосередили нашу увагу на перевірці введених даних від користувачів під час додавання авто та публікації оголошень. Також акцентували увагу на коретному виведені інформації та спрацювання відповідних кнопок. Наш телеграм-сервіс деякий час проходив тестування серед деякого кола людей. За цей період ми виявили можливі проблеми при роботі з ботом, врахували рекомендації зі сторони потенційних користувачів щодо зручності використання даного телеграм-сервісу.

Нагадаємо, що як такової реєстрації у нас немає. Це звязано з тим, що ми отримуємо деяку інформацію при першому надсиланні повідомлення від користувача. З цього повідомлення ми отримуємо його ідентифікатор та нікнейм в телеграмі. Для того, щоб закінчити реєстрацію, користувачу просто потрібно надіслати свій номер телефону. Для цього йому не потрібно вручну вписувати свій номер телефону, а лише на всього нажати на потрібну кнопку і все.

Але давайте розглянемо ситуацію, коли користувач замість надсилання номера телефону за допомогою кнопки, надсилає незрозумілий текст, це може статися випадково наприклад (рис.3.15).

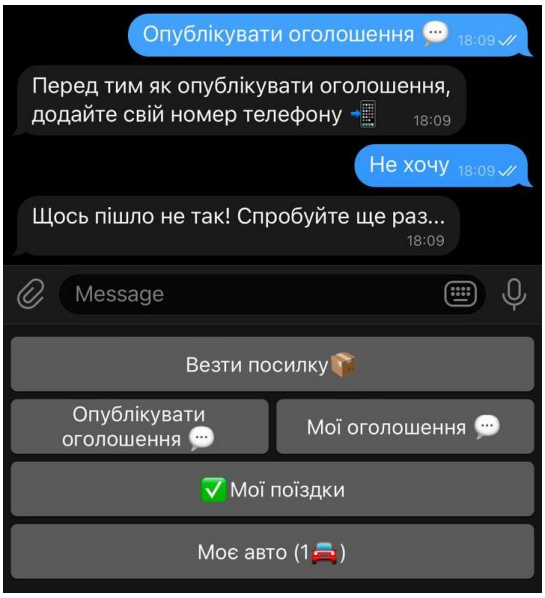

Рисунок 3.15 – Запит на надання номера телефону

В такій ситуації ТБ надсилає повідомлення, що щось пішло не так і вам варто спробувати ще раз. Ви будете отримувати це повідомлення поки не надішлете свій номер за допомогою відповідної кнопки.

Тепер давайте розберемо ввід даних при додаванні авто. Наприклад, коли ТБ запитує у користувача в якому році випустили його авто, користувач в такому разі може ввести тільки цифри, які не перебільшують на даний момент 2022-ий рік. Ввод чогось крім цифр недопустимий, про що телеграм бот нас інформує та попросить ввести дані знову (рис. 3.16).

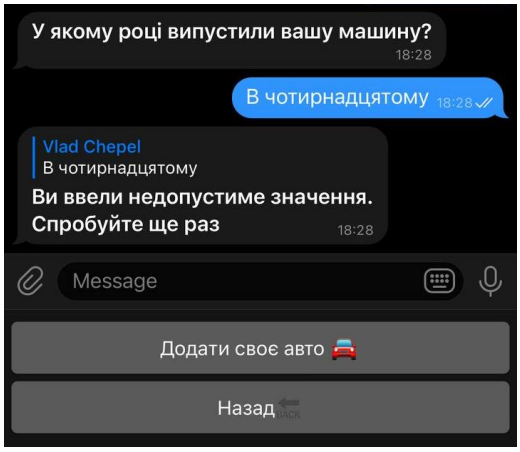

Рисунок 3.16 – Перевірка введених даних

Також, коли ТБ запитує у користувача якого кольору його авто, звсіно ж, що він не може вводити щось крім букв. При введенні чогось іншого ми отримаємо повідомлення про те, що ввели недопустиме значення (рис.3.17).

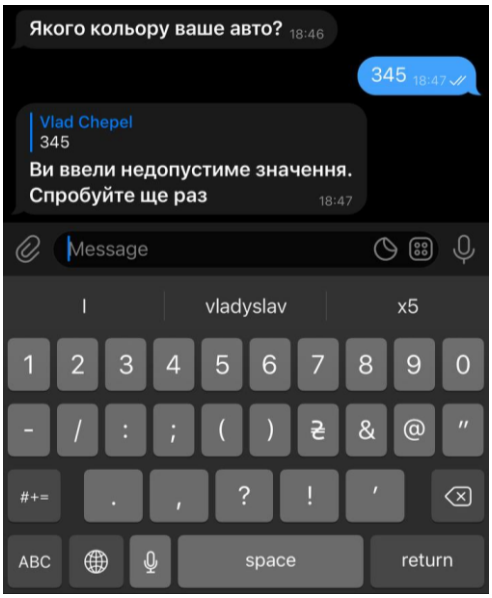

Рисунок 3.17 – Перевірка введених даних

Тепер давайте перейдемо до публікації оголошення. Тут ми також застосували ряд перевірок. Найголовніша з них – це перевірка адреси. Це дуже важливо, щоб при введені користувач не помилився та ввів дійсно ту адресу, яка йому потрібна. Для цього ми будемо використовувати так званий «геокодер». Це спеціальна бібліотека для отримання всіх даних по введенному адресу. Отже, давайте розберемо ситуацію при якій користувач неправильно ввів назву міста (рис.3.18).

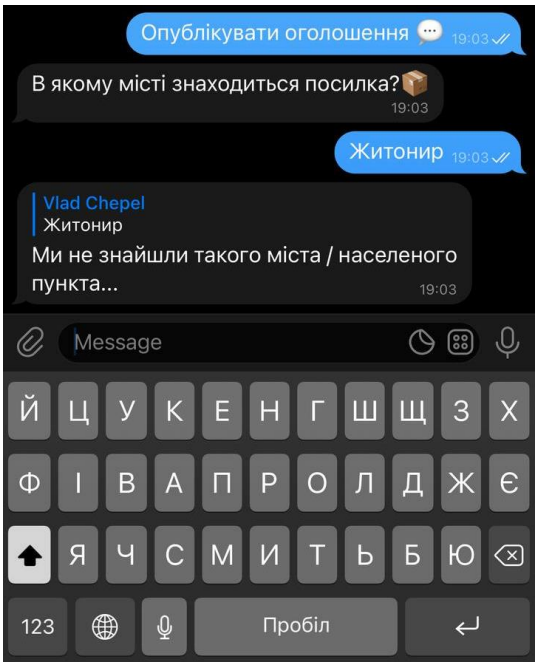

Рисунок 3.18 – Перевірка даних

Якщо користувач ввів неправильну назву міста або взагалі щось інше, то бот відправить вам повідомлення, що такого міста / населеного пункта він не знайшов. Після чого вам потрібно буде ввести коректну назву міста. Так само це працює й з адресом. Якщо ви введете некоректну адресу, ТБ повідомить вас про це. Також потрібно додати, що ми можемо вводити ті міста та населені пункти, що знаходяться тільки в Україні. При введені міста іншої країни, телеграм бот попросить ввести вас місто чи населений пункт в межах України (рис.3.19).

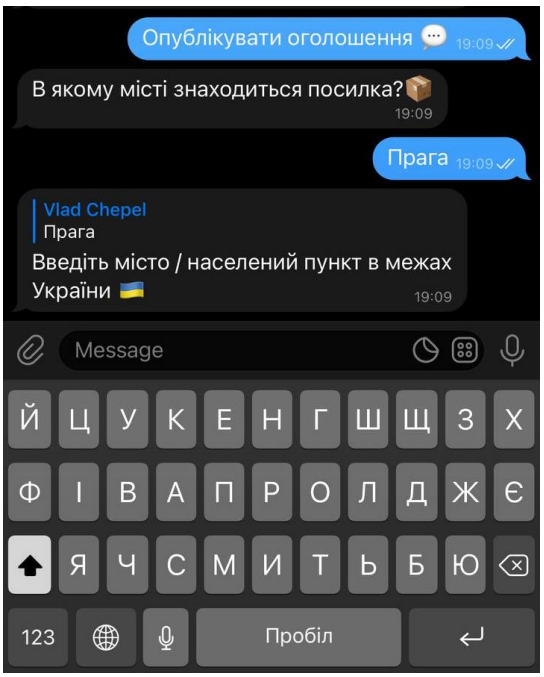

Рисунок 3.19 – Введення міста іншої країни

Коли ТБ попросить вас надіслати фото вашої посилки, а ви ввели текст чи надіслали відео, в такому разі ви також отримаєте повідомлення про те, що треба надіслати тільки фото (рис.3.20).

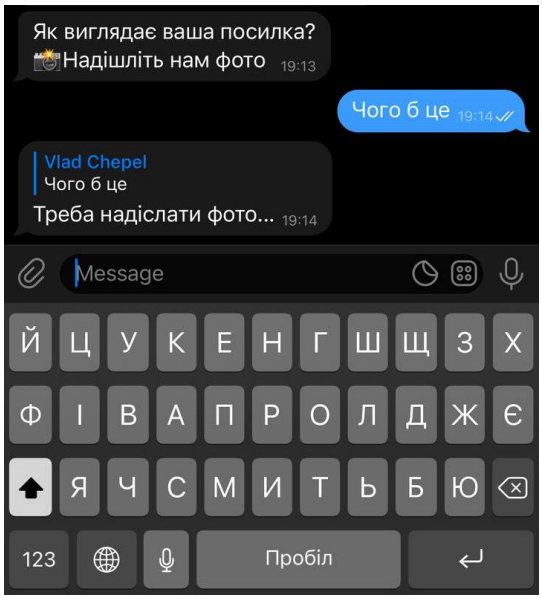

Рисунок 3.20 – Перевірка надсилання фото

Отже, з проведено тестування можна сказати, що розроблювальний програмний продукт повністю задовольняє всім початковим вимогам та готовий до використання.

Висновки до розділу 3

В даному розділі було детально описано інтерфейс телеграм-сервісу та проведено детальний опис всіх розділів головного меню.

Окрім цього було проведено функціональне та інтерфейсне тестування телеграм бота, що дало свій позитивний результат. Знайдені під час перевірок помилки були з часом виправлені, сервіс працює правильно та виконує покладені на нього задачі.

Після завершення роботи з сервісом всю роботу було задокументовано, а саме: процеси тестування та знешкодження проблем з інтерфейсом та функціоналом сервісу та процес покрокової коректної роботи з кінцевим варіантом телеграм бота.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано телеграм-сервіс з логістики.

1. Проведено аналіз аналогічних сервісів «Нова Пошта», «Укрпошта» та «BlaBlaCar». Визначено, що основними необхідними складовими є зручний та інтуїтивно зрозумілий додаток (в нашому випадку телеграм бот), стабільність роботи сервісу та підтримка сучасних технологій та рішень.

2. Для реалізації проекту було вибрано наступні засоби: мову розмітки гіпертекстових документів HTML5, багатоцільову мову програмування Python, бібліотеку pyTelegramBotAPI, облачну PaaS-платформу Heroku, середовище керування базами даних PostgreSQL та обрано наступні програмні продукти та інтегровані середовища розробки: середовище розробки PyCharm Community та функціональний пакет інструментів для роботи з базами даних JetBrains DataGrip.

3. Було проаналізовано функціональні та нефункціональні вимоги для створення телеграм-сервісу, для чого було побудовано діаграму варіантів використання.

4. Спроектовано Базу даних, яка відповідає всім вимогам нашого проекту. Реалізована БД забезпечує збереження, редагування та видалення інформації в нашому телеграм-сервісі. Вона складається з 4-х таблиць: users, car_info, parcel_info та offer_parcel. Таблиця «parcel_info» є найбільшою і містить в собі інформацію про опубліковані оголошення користувачів сервісу. В цій таблиці зберігаються наступні дані: id_parcel (ідентифікатор оголошення), city_a (місто відправлення), address_a (адрес відправлення), lon_a (координати місця відправлення: довгота), lat_a (координати місця відправлення: широта), city_b (місто прибуття), address_b (адреса прибуття), lon_b (координати місця прибуття: довгота), lat_b (координати місця прибуття: широта), date_parcel (дата доставки), photo(зображення посилки), price(ціна за доставку), description (короткий опис

посилки), id_user (власник оголошення), status (стадія перевезення посилки), id_car (ідентифікатор автомобіля, який перевозить посилку).

5. В кінці було проведено функціональне та інтерфейсне тестування телеграм-сервісу. Знайдені помилки та недоліки були виявлені та випралені, наразі сервіс працює правильно та виконує заплановані задачі.

Реалізований програмний комплекс готовий до широкого використання та розповсюдження серед потенційних користувачів, через свою зручність та інноваційність.

В подальшому даний сервіс може бути вдосконалений шляхом впровадження нового функціоналу та додаткової системи контролю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Путівник мовою програмування Python [Електроний ресурс] Режим доступу до ресурсу: http://pythonguide.rozh2sch.org.ua
2. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api
3. Асинхронный Telegram бот на языке Python 3 с использованием библиотеки aiogram [Електронний ресурс] – Режим доступу до ресурсу: https://surik00.gitbooks.io/aiogram-lessons/content/
4. eternnoir / pyTelegramBotAPI [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/eternnoir/pyTelegramBotAPI
5. examples/register_handler.py [Електронний ресурс] – Режим доступу до ресурсу:https://github.com/eternnoir/pyTelegramBotAPI/blob/master/examples/ register_handler.py
6. Питоном по телеграму [Електронний ресурс] – Режим доступу до ресурсу: https://xakep.ru/2021/11/28/python-telegram-bots/
7. Работа с PostgreSQL в Python [Електронний ресурс] – Режим доступу до ресурсу: https://khashtamov.com/ru/postgresql-python-psycopg2/
8. Psycopg – PostgreSQL database adapter for Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.psycopg.org/docs/#psycopg-postgresql-database-adapter-for-python
9. GeoPy’s documentation [Електронний ресурс] – Режим доступу до ресурсу: https://geopy.readthedocs.io/en/stable/
10. Calendar-telegram [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/unmonoqueteclea/calendar-telegram
11. Деплой Python Telegram бота на Heroku [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=E46jdft2ToA&list=LL&index=11&t=909s

ДОДАТКИ

Лістинг файлу main.py:

```
import configure
import telebot
from telebot import types
import psycopg2
from car_info import CarInfo
from parcel_info import ParcelInfo
from offer_parcel import OfferParcel
import re
from geopy.geocoders import Nominatim
from telegram_bot_calendar import DetailedTelegramCalendar, LSTEP

bot = telebot.TeleBot(configure.config["token"])
geolocator = Nominatim(user_agent="tudy-sudy")

db_connection = psycopg2.connect(configure.config["db_URI"], sslmode="require")
cursor = db_connection.cursor()

car_info_dict = {}
parcel_info_dict = {}
offer_dict = {}
current_car = {}

cars_type = {'sedan': 'Седан', 'SUV': 'Позашляховик', 'minivan': 'Мінівен',
'hatchback': 'Хетчбек',
             'universal': 'Універсал', 'coupe': 'Купе', 'passenger_van': 'Легковий
фургон', 'pickup': 'Пікап',
             'other': 'Інше'}


@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    id_user = message.from_user.id
    username = message.from_user.username

    cursor.execute(f'SELECT id FROM users WHERE id = {id_user}')
    result = cursor.fetchone()
    print(result)

    if not result:
        sql = 'INSERT INTO users(id, username) VALUES(%s, %s)'
        val = (id_user, username)
        cursor.execute(sql, val)
        db_connection.commit()
        print(cursor.rowcount, "record inserted.")

    bot.send_message(message.chat.id, f'Вітаємо,
<b>{message.from_user.first_name}</b>',
                     reply_markup=main_markup(message.from_user.id), parse_mode="HTML")


@bot.callback_query_handler(func=DetailedTelegramCalendar.func())
def cal(call):
    result, key, step = DetailedTelegramCalendar().process(call.data)
    if not result and key:
        bot.edit_message_text(f"Select {LSTEP[step]}",
                              call.message.chat.id,
                              call.message.message_id,
```

```
                                reply_markup=key)
    elif result:
        bot.edit_message_text(f"Ви обрали дату: {result}",
                              call.message.chat.id,
                              call.message.message_id)

        parcel_info = parcel_info_dict[call.from_user.id]
        parcel_info.date = result

        msg = bot.send_message(call.message.chat.id, f'Як виглядає ваша
посилка?\n📸Надішліть нам фото')
        bot.register_next_step_handler(msg, parcel_image)


@bot.message_handler(func=lambda message: True)
def callback(message):
    cursor.execute(f'SELECT phone FROM users WHERE id = {message.from_user.id}')
    phone = cursor.fetchone()[0]

    request_contact_markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    request_contact_markup.add(types.KeyboardButton('Надіслати номер телефону 📲',
request_contact=True))
    request_contact_markup.add(types.KeyboardButton('Назад🔙'))

    if message.text == "Опублікувати оголошення 💬":
        if not phone:
            msg = bot.send_message(message.chat.id, f'Перед тим як опублікувати
оголошення, додайте свій номер '
                                                    f'телефону 📲',
                                   reply_markup=request_contact_markup)
            bot.register_next_step_handler(msg, add_phone_advertisement)
        else:
            msg = bot.send_message(message.chat.id, f'В якому місті знаходиться
посилка?🏘')
            bot.register_next_step_handler(msg, parcel_city_a)

    elif message.text == "Додати своє авто 🚘":
        print(get_number_of_cars(message.from_user.id))
        if not phone:
            msg = bot.send_message(message.chat.id, f'Перед тим як додати своє авто,
додайте свій номер '
                                                    f'телефону 📲',
                                   reply_markup=request_contact_markup)
            bot.register_next_step_handler(msg, add_phone_car)
        else:
            msg = bot.send_message(message.chat.id, f'<b>Яка марка вашого авто?</b>',
parse_mode='HTML')
            bot.register_next_step_handler(msg, car_brand_next_step)

    elif re.match("Моє авто \(\d🚘\)", message.text):
        inline_markup = types.InlineKeyboardMarkup()

        cursor.execute(f'SELECT id_car, car_brand, car_model FROM car_info WHERE
id_owner = {message.from_user.id}')
        res = cursor.fetchall()

        for x in res:
            inline_markup.row(types.InlineKeyboardButton(f'{x[1]} {x[2]}',
callback_data=str(x[0])))
```

```
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        markup.row(types.KeyboardButton('Додати своє авто 🚗'))
        markup.row(types.KeyboardButton('Назад🔙'))
        bot.send_message(message.chat.id, f'🔻🔻🔻🔻', reply_markup=markup)

        bot.send_message(message.chat.id, f'<b>Моє авто🚗</b>',
reply_markup=inline_markup, parse_mode='HTML')

    elif message.text == 'Назад🔙':
        bot.send_message(message.chat.id, f'<b>Публікуй оголошення, додавай своє авто
та заробляй гроші💰</b>',
                         reply_markup=main_markup(message.from_user.id),
                         parse_mode='HTML')

    elif message.text == 'Везти посилку🎁':
        cursor.execute(f'SELECT id_car FROM car_info WHERE id_owner =
{message.from_user.id}')
        result = cursor.fetchone()

        if not result:
            bot.send_message(message.chat.id, f'<b>Спочатку додайте авто🚗</b>',
parse_mode='HTML')
            return

        bot.send_message(message.chat.id, f'<b>Оголошення:</b>',
reply_markup=get_all_parcel(message.from_user.id),
                         parse_mode='HTML')

    elif message.text == 'Мої оголошення 💬':
        cursor.execute(f'SELECT * FROM parcel_info WHERE id_user =
{message.from_user.id}')
        res = cursor.fetchall()

        if not res:
            bot.send_message(message.chat.id, f'<b>Тут поки пусто🗑</b>',
parse_mode='HTML')
        else:
            bot.send_message(message.chat.id, f'<b>Мої оголошення 💬</b>',
                             reply_markup=get_my_parcel_inline(message.from_user.id),
parse_mode='HTML')

    elif message.text == '✅Мої поїздки':
        bot.send_message(message.chat.id, f'<b>Мої поїздки:</b>',
                         reply_markup=get_agree_parcel_inline(message.from_user.id),
parse_mode='HTML')


def get_agree_parcel_inline(id_user):
    inline_markup = types.InlineKeyboardMarkup()

    cursor.execute(f'SELECT parcel_info.date_parcel, parcel_info.city_a,
parcel_info.city_b, parcel_info.id_parcel '
                   f'FROM parcel_info JOIN car_info ON '
                   f'parcel_info.id_car = car_info.id_car WHERE car_info.id_owner =
{id_user}')
    res = cursor.fetchall()

    for x in res:
```

```
        if x[1] == x[2]:
            inline_markup.row(types.InlineKeyboardButton(f'{x[0]} {x[1]}',
callback_data=f'parcel-agree-{x[3]}'))
        else:
            inline_markup.row(types.InlineKeyboardButton(f'{x[0]} {x[1]} - {x[2]}',
                                                         callback_data=f'parcel-agree-
{x[3]}'))

    return inline_markup


@bot.callback_query_handler(func=lambda call: True)
def callback(call):
    chat_id = call.message.chat.id
    message_id = call.message.message_id
    owner_id = call.from_user.id

    cursor.execute(f'SELECT id_car FROM car_info WHERE id_owner = {call.from_user.id}')
    cars_id = cursor.fetchall()

    if call.data in cars_type:
        # noinspection PyBroadException
        try:
            car_info = car_info_dict[call.from_user.id]
            car_info.car_type = cars_type[call.data]

            msg = bot.send_message(call.message.chat.id, f'<b>Якого кольору ваше
авто?</b>', parse_mode='HTML')
            bot.register_next_step_handler(msg, car_color_next_step)
        except Exception as e:
            print(e)
            bot.reply_to(call.message, "Щось пішло не так...")

    elif call.data in str(cars_id):
        send_car_info_call(call)

    elif re.match("brand-\d+", call.data):
        message = call.data
        car_id = message.split('-')[-1]
        current_car[owner_id] = car_id

        msg = bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                    text=f'<b>Яка марка вашого авто?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, change_brand)

    elif re.match("model-\d+", call.data):
        set_current_car(call)

        msg = bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                    text=f'<b>Яка модель вашого авто?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, change_model)

    elif re.match("color-\d+", call.data):
        set_current_car(call)

        msg = bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                    text=f'<b>Якого кольору ваше авто?</b>',
parse_mode='HTML')
```

```
        bot.register_next_step_handler(msg, change_color)

    elif re.match("year-\d", call.data):
        set_current_car(call)
        msg = bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                    text=f'<b>У якому році випустили вашу машину?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, change_year)

    elif re.match("number-\d+", call.data):
        set_current_car(call)
        msg = bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                    text=f'<b>Який номерний знак авто?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, change_number)

    elif re.match("description-\d+", call.data):
        set_current_car(call)
        msg = bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                    text=f'<b>Додайте короткий опис вашого авто.</b>
Ваш опис допоможе комусь '
                                         f'обрати саме ваше авто для перевезення
вантажу.\n(Наприклад, вкажіть '
                                         f'максимальну вагу вантажу, яку можете взяти з
собою, розміри '
                                         f'багажника чи розміри буди, якщо авто грузове
і тд.)',
                                    parse_mode='HTML')
        bot.register_next_step_handler(msg, change_description)

    elif re.match("type-\d+", call.data):
        set_current_car(call)

        inline_markup = types.InlineKeyboardMarkup()
        inline_markup.row(types.InlineKeyboardButton('Седан', callback_data='change-
sedan'),
                          types.InlineKeyboardButton('Позашляховик / Кросовер',
callback_data='change-SUV'))
        inline_markup.row(types.InlineKeyboardButton('Мінівен', callback_data='change-
minivan'),
                          types.InlineKeyboardButton('Хетчбек', callback_data='change-
hatchback'))
        inline_markup.row(types.InlineKeyboardButton('Універсал',
callback_data='change-universal'),
                          types.InlineKeyboardButton('Купе', callback_data='change-
coupe'))
        inline_markup.row(types.InlineKeyboardButton('Легковий фургон',
callback_data='change-passenger_van'),
                          types.InlineKeyboardButton('Пікап', callback_data='change-
pickup'))
        inline_markup.row(types.InlineKeyboardButton('Інше', callback_data='change-
other'))

        bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                              text=f'<b>Який тип кузова у вашого авто?</b>',
reply_markup=inline_markup,
                              parse_mode='HTML')

    elif re.match("change-\w+", call.data):
        message = call.data
```

```
        car_type = message.split('-')[-1]
        sql = f'UPDATE car_info SET car_type = \'{cars_type[car_type]}\' WHERE id_car =
{current_car[owner_id]}'
        cursor.execute(sql)
        db_connection.commit()

        print(cursor.rowcount, "record(s) affected")

        bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                              text=f'<b>Інформація по вашій машині:</b>\n!Натисність на
поле, щоб змінити його значення',

reply_markup=get_car_inline_delete(current_car[owner_id]), parse_mode='HTML')

    elif re.match("delete-\d+", call.data):
        set_current_car(call)

        sql = f'DELETE FROM car_info WHERE id_car = {current_car[owner_id]}'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) deleted")

        bot.edit_message_text(chat_id=chat_id, message_id=message_id, text=f'<b>Авто
успішно видалено</b>',
                              parse_mode='HTML')

        bot.send_message(call.message.chat.id, f'❌🚘',
                         reply_markup=main_markup(owner_id), parse_mode='HTML')

    elif re.match(f"{owner_id}-\w+", call.data):
        parcel_info = parcel_info_dict[call.from_user.id]

        if parcel_info.city_b is not None:
            return

        bot.clear_step_handler(call.message)

        message = call.data
        city_b = message.split('-')[-1]

        parcel_info = parcel_info_dict[owner_id]
        parcel_info.city_b = city_b

        msg = bot.send_message(call.message.chat.id, f'🔎Введіть адресу за якою
потрібно доставити вашу посилку')
        bot.register_next_step_handler(msg, parcel_address_b)

    elif re.match(f"my-parcel-\d+", call.data):
        id_parcel = get_id(call.data)

        bot.send_photo(chat_id=chat_id, photo=get_photo_parcel(int(id_parcel)),
caption=get_caption_parcel(id_parcel),
                       reply_markup=get_parcel_inline(id_parcel))

    elif re.match(f"delete-parcel-\d+", call.data):
        id_parcel = get_id(call.data)

        sql = f'DELETE FROM parcel_info WHERE id_parcel = {int(id_parcel)};'
        cursor.execute(sql)
        db_connection.commit()
```

```
        print(cursor.rowcount, "record(s) deleted")

        chat_id = call.message.chat.id
        message_id = call.message.message_id

        bot.delete_message(chat_id=chat_id, message_id=message_id)

        cursor.execute(f'SELECT * FROM parcel_info WHERE id_user =
{call.from_user.id}')
        res = cursor.fetchall()

        if not res:
            bot.send_message(chat_id, f'<b>Тут поки пусто🗑</b>', parse_mode='HTML')
        else:
            bot.send_message(chat_id, f'<b>Мої оголошення 💬</b>',
                             reply_markup=get_my_parcel_inline(call.from_user.id),
parse_mode='HTML')

    elif re.match(f"parcel-\d+", call.data):
        id_parcel = get_id(call.data)

        inline_markup, desc = get_parcel_info_inline(id_parcel)

        bot.send_photo(chat_id=chat_id, photo=get_photo_parcel(int(id_parcel)),
caption=desc,
                       reply_markup=inline_markup)

    elif re.match(f'parcel-address_a-\d+', call.data):
        id_parcel = get_id(call.data)

        cursor.execute(f'SELECT city_a, address_a, lon_a, lat_a FROM parcel_info WHERE
id_parcel = {int(id_parcel)}')
        res = cursor.fetchone()

        bot.send_venue(chat_id=chat_id, latitude=res[3], longitude=res[2],
address=f'{res[1]}, {res[0]}',
                       title=f'Адреса за якою потрібно забрати посилку\n')

    elif re.match(f'parcel-address_b-\d+', call.data):
        id_parcel = get_id(call.data)

        cursor.execute(f'SELECT city_b, address_b, lon_b, lat_b FROM parcel_info WHERE
id_parcel = {int(id_parcel)}')
        res = cursor.fetchone()

        bot.send_venue(chat_id=chat_id, latitude=res[3], longitude=res[2],
address=f'{res[1]}, {res[0]}',
                       title=f'Адреса за якою потрібно доставити посилку\n')

    elif re.match(f'take-parcel-\d+', call.data):
        id_parcel = get_id(call.data)

        offer = OfferParcel(id_parcel)
        offer_dict[call.from_user.id] = offer
        bot.send_message(chat_id, f'<b>На якій машині повезеш?</b>',
reply_markup=get_my_car_inline(call.from_user.id),
                         parse_mode='HTML')

    elif re.match(f'parcel-car-\d+', call.data):
        id_car = get_id(call.data)
```

```
        offer = offer_dict[call.from_user.id]
        offer.id_car = id_car

        sql = 'INSERT INTO offer_parcel(id_parcel, id_car) VALUES(%s, %s)'
        val = (offer.id_parcel, offer.id_car)
        cursor.execute(sql, val)
        db_connection.commit()
        print(cursor.rowcount, "record inserted.")

        bot.send_message(chat_id, f'<b>Чудово!</b>\nТепер очікуйте підтвердження від
замовника', parse_mode='HTML')

        bot.send_message(get_user_from_parcel(offer.id_parcel), '🥹Хтось хоче перевезти
вашу посилку.\n'
                                                                'Зайдіть в розділ Мої
оголошення 💬 та '
                                                                'підтвердіть запит')

        del offer_dict[call.from_user.id]

    elif re.match(f'offer-\d+-\d+', call.data):
        print(call.data)
        id_parcel = get_id_parcel(call.data)

        id_car = get_id(call.data)

        markup = types.InlineKeyboardMarkup()
        markup.row(types.InlineKeyboardButton('Погодити', callback_data=f'offer-yes-
{id_parcel}-{id_car}'))
        markup.row(types.InlineKeyboardButton('Відмовити', callback_data=f'offer-no-
{id_parcel}-{id_car}'))

        bot.send_message(chat_id=chat_id, text=get_car_text(id_car),
reply_markup=markup, parse_mode='HTML')

    elif re.match(f'offer-no-\d+-\d+', call.data):
        id_parcel = get_id_parcel(call.data)
        id_car = get_id(call.data)

        sql = f'DELETE FROM offer_parcel WHERE id_parcel = {int(id_parcel)} AND id_car
= {int(id_car)};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) deleted")

        chat_id = call.message.chat.id
        message_id = call.message.message_id

        bot.delete_message(chat_id, message_id)
        bot.send_photo(chat_id=chat_id, photo=get_photo_parcel(int(id_parcel)),
caption=get_caption_parcel(id_parcel),
                       reply_markup=get_parcel_inline(id_parcel))

    elif re.match(f'offer-yes-\d+-\d+', call.data):
        id_parcel = get_id_parcel(call.data)
        id_car = get_id(call.data)

        sql = f'UPDATE parcel_info SET id_car = {int(id_car)}, ' \
              f'status = \'Машину знайдено, очікуйте прибуття посилки\' WHERE id_parcel
```

```
= {int(id_parcel)};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")

        sql = f'DELETE FROM offer_parcel WHERE id_parcel = {int(id_parcel)} ;'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) deleted")

        chat_id = call.message.chat.id
        message_id = call.message.message_id

        bot.delete_message(chat_id, message_id)
        bot.send_photo(chat_id=chat_id, photo=get_photo_parcel(int(id_parcel)),
caption=get_caption_parcel(id_parcel),
                       reply_markup=get_parcel_inline(id_parcel))

        cursor.execute(f'SELECT id_owner FROM car_info WHERE car_info.id_car =
{id_car};')
        id_user = cursor.fetchone()[0]

        bot.send_message(chat_id=id_user,
                         text='Вашу заявку схвалено! Ви везете посилку. Перейдіть в
розділ ✅Мої поїздки')

    elif re.match(f'get-info-\d+', call.data):
        id_car = get_id(call.data)

        cursor.execute(f'SELECT users.username, users.phone FROM users '
                       f'JOIN car_info ON car_info.id_owner = users.id WHERE
car_info.id_car = {id_car};')
        user = cursor.fetchone()

        info = get_car_text(id_car) + f'\n<b>👤Власник авто</b>\n\t\t\t➔📱Номер телефону:
+{user[1]}\n' \
                                      f'\t\t\t✍Telegram: @{user[0]}'
        bot.send_message(chat_id=chat_id, text=info, parse_mode='HTML')

    elif re.match(f'parcel-done-\d+', call.data):
        id_parcel = get_id(call.data)

        sql = f'UPDATE parcel_info SET status = \'Посилку доставлено!\' WHERE id_parcel
= {int(id_parcel)};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")

        cursor.execute(f'SELECT * FROM parcel_info JOIN car_info ON '
                       f'parcel_info.id_car = car_info.id_car WHERE car_info.id_owner =
{call.from_user.id}')
        res = cursor.fetchone()

        inline_markup = types.InlineKeyboardMarkup()

        inline_markup.row(types.InlineKeyboardButton(f'🔎Від: {res[1]}, {res[2]}',
                                                     callback_data=f'parcel-address_a-
{int(res[0])}'))
        inline_markup.row(types.InlineKeyboardButton(f'🔎До: {res[5]}, {res[6]}',
                                                     callback_data=f'parcel-address_b-
```

```
{int(res[0])}'))

        chat_id = call.message.chat.id
        message_id = call.message.message_id

        bot.delete_message(chat_id=chat_id, message_id=message_id)
        bot.send_photo(chat_id=call.message.chat.id,
photo=get_photo_parcel(int(int(res[0]))),
                       caption=f'🗓Дата: {res[9]}\n💰Ціна: {res[11]}\n💬Коментар:
{res[12]}\n'
                               f'🚦Статус: {res[14]}', reply_markup=inline_markup)

        bot.send_message(chat_id=res[13], text='Вашу посилку доставлено')

    elif re.match(f'parcel-agree-\d+', call.data):
        cursor.execute(f'SELECT * FROM parcel_info JOIN car_info ON '
                       f'parcel_info.id_car = car_info.id_car WHERE
car_info.id_owner = {call.from_user.id}')
        res = cursor.fetchone()

        inline_markup = types.InlineKeyboardMarkup()

        inline_markup.row(types.InlineKeyboardButton(f'🔑Від: {res[1]}, {res[2]}',
                                                     callback_data=f'parcel-address_a-
{int(res[0])}'))
        inline_markup.row(types.InlineKeyboardButton(f'🔑До: {res[5]}, {res[6]}',
                                                     callback_data=f'parcel-address_b-
{int(res[0])}'))

        if res[14] == "Посилку доставлено!":
            bot.send_photo(chat_id=call.message.chat.id,
photo=get_photo_parcel(int(int(res[0]))),
                           caption=f'🗓Дата: {res[9]}\n💰Ціна: {res[11]}\n💬Коментар:
{res[12]}\n'
                                   f'🚦Статус: {res[14]}',
                           reply_markup=inline_markup)
        else:
            inline_markup.row(types.InlineKeyboardButton(f'✅Відмітити як виконане',
                                                         callback_data=f'parcel-done-
{int(res[0])}'))

            bot.send_photo(chat_id=call.message.chat.id,
photo=get_photo_parcel(int(int(res[0]))),
                           caption=f'🗓Дата: {res[9]}\n💰Ціна: {res[11]}\n💬Коментар:
{res[12]}',
                           reply_markup=inline_markup)

    else:
        pass


def get_id_parcel(message):
    mes = message
    return mes.split('-')[-2]


def get_user_from_parcel(id_parcel):
    cursor.execute(f'SELECT id_user FROM parcel_info WHERE id_parcel =
{int(id_parcel)}')
```

```
    return cursor.fetchone()[0]


def get_my_car_inline(id_owner):
    inline_markup = types.InlineKeyboardMarkup()

    cursor.execute(f'SELECT id_car, car_brand, car_model FROM car_info WHERE id_owner =
{id_owner}')
    res = cursor.fetchall()

    for x in res:
        inline_markup.row(types.InlineKeyboardButton(f'{x[1]} {x[2]}',
callback_data=f'parcel-car-{x[0]}'))

    return inline_markup


def get_id(message):
    mes = message
    return mes.split('-')[-1]


def get_parcel_info_inline(id_parcel):
    inline_markup = types.InlineKeyboardMarkup()

    cursor.execute(f'SELECT * FROM parcel_info WHERE id_parcel = {int(id_parcel)}')
    res = cursor.fetchone()

    description = res[12]

    inline_markup.row(types.InlineKeyboardButton(f'🔑Від: {res[1]}, {res[2]}',
                                                 callback_data=f'parcel-address_a-
{id_parcel}'))
    inline_markup.row(types.InlineKeyboardButton(f'🔑До: {res[5]}, {res[6]}',
                                                 callback_data=f'parcel-address_b-
{id_parcel}'))
    inline_markup.row(types.InlineKeyboardButton(f'🆔Дата: {res[9]}',
callback_data=f'.'))
    inline_markup.row(types.InlineKeyboardButton(f'💰Ціна: {res[11]}',
callback_data=f'parcel-price-{id_parcel}'))
    inline_markup.row(types.InlineKeyboardButton(f'✅Можу завезти',
callback_data=f'take-parcel-{id_parcel}'))

    return inline_markup, description


def get_car_text(id_car):
    cursor.execute(f'SELECT * FROM car_info WHERE id_car = {id_car}')
    car_data = cursor.fetchone()

    return f'<b>🚘Інформація про авто</b>\n' \
           f'\t\t\tМарка: {car_data[1]}\n' \
           f'\t\t\tМодель: {car_data[2]}\n' \
           f'\t\t\tТип кузова: {car_data[3]}\n' \
           f'\t\t\tКолір: {car_data[4]}\n' \
           f'\t\t\tРік випуску: {car_data[5]}\n' \
           f'\t\t\tНомерний знак: {car_data[6]}\n' \
           f'\t\t\tКороткий опис: {car_data[7]}'
```

```
def get_parcel_inline(id_parcel):
    inline_markup = types.InlineKeyboardMarkup()

    inline_markup.row(types.InlineKeyboardButton(f'❌Видалити оголошення❌',
callback_data=f'delete-parcel-{id_parcel}'))

    cursor.execute(f'SELECT id_car FROM parcel_info WHERE id_parcel =
{int(id_parcel)}')
    id_car = cursor.fetchone()[0]

    cursor.execute(f'SELECT * FROM offer_parcel WHERE id_parcel = {int(id_parcel)}')
    res = cursor.fetchall()

    if id_car:
        inline_markup.row(types.InlineKeyboardButton(f'⬇Вашу посилку відвезе⬇',
callback_data='.'))
        res = get_car_info(id_car)

        inline_markup.row(types.InlineKeyboardButton(f'{res[1]} {res[2]}',
                                                     callback_data=f'get-info-
{res[0]}'))

    elif res:
        inline_markup.row(types.InlineKeyboardButton(f'⬇Вашу посилку можуть
відвезти⬇', callback_data='.'))
        for car in res:
            car_info = get_car_info(car[1])

            inline_markup.row(types.InlineKeyboardButton(f'{car_info[1]}
{car_info[2]}',
                                                         callback_data=f'offer-
{id_parcel}-{car[1]}'))

    return inline_markup


def get_car_info(id_car):
    cursor.execute(f'SELECT id_car, car_brand, car_model FROM car_info WHERE id_car =
{id_car}')
    return cursor.fetchone()


def get_caption_parcel(id_parcel):
    cursor.execute(f'SELECT * FROM parcel_info WHERE id_parcel = {id_parcel}')
    res = cursor.fetchone()

    return f'🔑{res[2]}, {res[1]}\n' \
           f'🔑{res[6]}, {res[5]}\n' \
           f'🗓Дата: {res[9]}\n' \
           f'💸Ваша ціна: {res[11]} UAH\n' \
           f'📝Ваш опис: {res[12]}\n' \
           f'СТАТУС: {res[14]}'


def get_photo_parcel(id_parcel):
    cursor.execute(f'SELECT photo FROM parcel_info WHERE id_parcel = {id_parcel}')
    return cursor.fetchone()[0]
```

```
def get_all_parcel(id_user):
    inline_markup = types.InlineKeyboardMarkup()

    cursor.execute(f'SELECT date_parcel, city_a, city_b, id_parcel FROM parcel_info
WHERE id_car IS NULL AND '
                   f'id_user <> {id_user} ')

    res = cursor.fetchall()

    for x in res:
        if x[1] == x[2]:
            inline_markup.row(types.InlineKeyboardButton(f'{x[0]} {x[1]}',
callback_data=f'parcel-{x[3]}'))
        else:
            inline_markup.row(types.InlineKeyboardButton(f'{x[0]} {x[1]} - {x[2]}',
callback_data=f'parcel-{x[3]}'))

    return inline_markup


def get_my_parcel_inline(id_user):
    inline_markup = types.InlineKeyboardMarkup()

    cursor.execute(f'SELECT date_parcel, city_a, city_b, id_parcel FROM parcel_info '
                   f'WHERE id_user = {id_user}')
    res = cursor.fetchall()

    for x in res:
        if x[1] == x[2]:
            inline_markup.row(types.InlineKeyboardButton(f'{x[0]} {x[1]}',
callback_data=f'my-parcel-{x[3]}'))
        else:
            inline_markup.row(types.InlineKeyboardButton(f'{x[0]} {x[1]} - {x[2]}',
callback_data=f'my-parcel-{x[3]}'))

    return inline_markup


def get_country(address):
    country = address.split(',')[-1]
    return country


def parcel_city_a(message):
    try:
        location = geolocator.geocode(message.text)

        if get_country(location.address) != ' Україна':
            msg = bot.reply_to(message, "Введіть місто / населений пункт в межах
України UA")
            bot.register_next_step_handler(msg, parcel_city_a)
            return

        parcel_info = ParcelInfo(message.text)
        parcel_info_dict[message.from_user.id] = parcel_info

        msg = bot.send_message(message.chat.id, f'🔎Введіть адресу за якою потрібно
забрати вашу посилку')
        bot.register_next_step_handler(msg, parcel_address_a)
```

```
    except Exception as e:
        print(e)
        bot.reply_to(message, "Ми не знайшли такого міста / населеного пункта...")


def parcel_address_a(message):
    try:
        id_user = message.from_user.id

        location = geolocator.geocode(message.text)

        parcel_info = parcel_info_dict[message.from_user.id]
        parcel_info.address_a = message.text

        coords = geolocator.geocode(f'{parcel_info.address_a}, {parcel_info.city_a}')
        parcel_info.lon_a = coords.longitude
        parcel_info.lat_a = coords.latitude

        markup = types.InlineKeyboardMarkup()
        markup.row(types.InlineKeyboardButton(f'{parcel_info.city_a}',
callback_data=f'{id_user}-{parcel_info.city_a}'))

        msg = bot.send_message(message.chat.id, f'В яке місто потрібно доставити вашу
посилку?', reply_markup=markup)
        bot.register_next_step_handler(msg, parcel_city_b)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось ми не знайшли такої адреси...")


def parcel_city_b(message):
    try:
        location = geolocator.geocode(message.text)

        if get_country(location.address) != ' Україна':
            msg = bot.reply_to(message, "Введіть місто / населений пункт в межах
України UA")
            bot.register_next_step_handler(msg, parcel_city_a)
            return

        parcel_info = parcel_info_dict[message.from_user.id]
        parcel_info.city_b = message.text

        msg = bot.send_message(message.chat.id, f'🔑Введіть адресу за якою потрібно
доставити вашу посилку')
        bot.register_next_step_handler(msg, parcel_address_b)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Ми не знайшли такого міста / населеного пункта...")


def parcel_address_b(message):
    try:
        location = geolocator.geocode(message.text)
        print(location.address)

        parcel_info = parcel_info_dict[message.from_user.id]
        parcel_info.address_b = message.text

        coords = geolocator.geocode(f'{parcel_info.address_b}, {parcel_info.city_b}')
```

```
        parcel_info.lon_b = coords.longitude
        parcel_info.lat_b = coords.latitude

        bot.send_message(message.chat.id, f'🗓Коли потрібно доставити?')

        calendar, step = DetailedTelegramCalendar().build()
        bot.send_message(message.chat.id,
                         f"Select {LSTEP[step]}",
                         reply_markup=calendar)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось ми не знайшли такої адреси...")


def parcel_image(message):
    try:
        if message.content_type == 'photo':
            file_info = bot.get_file(message.photo[len(message.photo) - 1].file_id)

            parcel_info = parcel_info_dict[message.from_user.id]
            parcel_info.photo = file_info.file_id

            msg = bot.send_message(message.chat.id, f'💰<b>Скільки ви заплатите за
доставку?</b>\n'
                                                    f'Вказуйте ціну в гривнях!',
parse_mode='HTML')
            bot.register_next_step_handler(msg, parcel_price)
        else:
            msg = bot.reply_to(message, "Треба надіслати фото...")
            bot.register_next_step_handler(msg, parcel_image)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def parcel_price(message):
    try:
        price = message.text
        if not price.isdigit() or int(price) < 0:
            msg = bot.reply_to(message, f'<b>Ви ввели недопустиме значення. Спробуйте
ще раз</b>', parse_mode='HTML')
            bot.register_next_step_handler(msg, parcel_price)
            return

        parcel_info = parcel_info_dict[message.from_user.id]
        parcel_info.price = price

        msg = bot.send_message(message.chat.id, f'<b>Додайте короткий опис вашої
посилки</b>\n'
                                                f'(Наприклад, укажіть вагу посилки, її
розміри. '
                                                f'Якщо вона хрупка, обов\'язково
вкажіть це)', parse_mode='HTML')
        bot.register_next_step_handler(msg, parcel_description)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def parcel_description(message):
```

```
    try:
        id_user = message.from_user.id

        parcel_info = parcel_info_dict[message.from_user.id]
        parcel_info.description = message.text

        sql = 'INSERT INTO parcel_info(city_a, address_a, lon_a, lat_a, city_b,
address_b, lon_b, lat_b, ' \
              'date_parcel, photo, price, description, id_user, status) ' \
              'VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)'
        val = (parcel_info.city_a, parcel_info.address_a, parcel_info.lon_a,
parcel_info.lat_a, parcel_info.city_b,
               parcel_info.address_b, parcel_info.lon_b, parcel_info.lat_b,
parcel_info.date, parcel_info.photo,
               parcel_info.price, parcel_info.description, id_user, parcel_info.status)
        cursor.execute(sql, val)
        db_connection.commit()

        print(cursor.rowcount, "record inserted.")

        del parcel_info_dict[message.from_user.id]

        bot.send_message(message.chat.id, f'<b>🥳Ви опублікували нове оголошення</b>\n'
                                          f'Очікуйте поки хтось прийме його□',
parse_mode='HTML')
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def send_car_info_call(call):
    chat_id = call.message.chat.id
    message_id = call.message.message_id

    return bot.edit_message_text(chat_id=chat_id, message_id=message_id,
                                 text=f'<b>Інформація по вашій машині:</b>\n'
                                      f'❗Натисність на поле, щоб змінити його
значення',
                                 reply_markup=get_car_inline_delete(int(call.data)),
parse_mode='HTML')


def change_description(message):
    try:
        sql = f'UPDATE car_info SET car_description = \'{message.text}\' ' \
              f'WHERE id_car = {current_car[message.from_user.id]};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")
        send_car_info(message)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def change_number(message):
    try:
        sql = f'UPDATE car_info SET car_number = \'{message.text}\' WHERE id_car =
{current_car[message.from_user.id]};'
        cursor.execute(sql)
```

```
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")
        send_car_info(message)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def change_year(message):
    try:
        car_year = message.text
        if not car_year.isdigit() or int(message.text) > 2022 or int(message.text) <
1975:
            msg = bot.reply_to(message, f'<b>Ви ввели недопустиме значення. Спробуйте
ще раз</b>', parse_mode='HTML')
            bot.register_next_step_handler(msg, change_year)
            return

        sql = f'UPDATE car_info SET car_year = {int(message.text)} WHERE id_car =
{current_car[message.from_user.id]};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")

        send_car_info(message)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def change_color(message):
    try:
        sql = f'UPDATE car_info SET car_color = \'{message.text}\' WHERE id_car =
{current_car[message.from_user.id]};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")

        send_car_info(message)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
def change_model(message):
    try:
        sql = f'UPDATE car_info SET car_model = \'{message.text}\' WHERE id_car =
{current_car[message.from_user.id]};'
        cursor.execute(sql)
        db_connection.commit()
        print(cursor.rowcount, "record(s) affected")

        send_car_info(message)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
```

```
def change_brand(message):
    try:
        sql = f'UPDATE car_info SET car_brand = \'{message.text}\' WHERE id_car =
{current_car[message.from_user.id]};'
        cursor.execute(sql)
        db_connection.commit()

        print(cursor.rowcount, "record(s) affected")

        send_car_info(message)

    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


def send_car_info(message):
    return bot.send_message(message.chat.id,
                            f'<b>Інформація по вашій машині:</b>\n!Натисність на поле,
щоб змінити його значення',

reply_markup=get_car_inline_delete(current_car[message.from_user.id]),
parse_mode='HTML')


def set_current_car(call):
    message = call.data
    car_id = message.split('-')[-1]
    current_car[call.from_user.id] = car_id


def get_car_inline(id_car):
    cursor.execute(f'SELECT * FROM car_info WHERE id_car = {id_car}')
    car_data = cursor.fetchone()

    inline_markup = types.InlineKeyboardMarkup()

    inline_markup.row(types.InlineKeyboardButton(f'Марка: {car_data[1]}',
callback_data=f'brand-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Модель: {car_data[2]}',
callback_data=f'model-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Тип кузова: {car_data[3]}',
callback_data=f'type-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Колір: {car_data[4]}',
callback_data=f'color-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Рік випуску: {car_data[5]}',
callback_data=f'year-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Номерний знак: {car_data[6]}',
                                                 callback_data=f'number-
{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Короткий опис: {car_data[7]}',
                                                 callback_data=f'description-
{car_data[0]}'))

    return inline_markup


def get_car_inline_delete(id_car):
    cursor.execute(f'SELECT * FROM car_info WHERE id_car = {id_car}')
    car_data = cursor.fetchone()
```

```
    inline_markup = types.InlineKeyboardMarkup()

    inline_markup.row(types.InlineKeyboardButton(f'Марка: {car_data[1]}',
callback_data=f'brand-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Модель: {car_data[2]}',
callback_data=f'model-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Тип кузова: {car_data[3]}',
callback_data=f'type-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Колір: {car_data[4]}',
callback_data=f'color-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Рік випуску: {car_data[5]}',
callback_data=f'year-{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Номерний знак: {car_data[6]}',
                                                 callback_data=f'number-
{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'Короткий опис: {car_data[7]}',
                                                 callback_data=f'description-
{car_data[0]}'))
    inline_markup.row(types.InlineKeyboardButton(f'❌Видалити авто❌',
                                                 callback_data=f'delete-
{car_data[0]}'))

    return inline_markup


# noinspection PyBroadException
def car_brand_next_step(message):
    try:
        car_info = CarInfo(message.text)
        car_info_dict[message.from_user.id] = car_info

        msg = bot.send_message(message.chat.id, f'<b>Яка модель вашого авто?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, car_model_next_step)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
def car_model_next_step(message):
    try:
        car_info = car_info_dict[message.from_user.id]
        car_info.car_model = message.text

        bot.send_message(message.chat.id, f'<b>Який тип кузова у вашого авто?</b>',
reply_markup=get_type_inline(),
                         parse_mode='HTML')
        # bot.register_next_step_handler(msg, car_type_next_step)
    except Exception as e:
        bot.reply_to(message, "Щось пішло не так...")


def get_type_inline():
    inline_markup = types.InlineKeyboardMarkup()

    inline_markup.row(types.InlineKeyboardButton('Седан', callback_data='sedan'),
                      types.InlineKeyboardButton('Позашляховик / Кросовер',
callback_data='SUV'))
```

```
    inline_markup.row(types.InlineKeyboardButton('Мінівен', callback_data='minivan'),
                      types.InlineKeyboardButton('Хетчбек', callback_data='hatchback'))
    inline_markup.row(types.InlineKeyboardButton('Універсал',
callback_data='universal'),
                      types.InlineKeyboardButton('Купе', callback_data='coupe'))
    inline_markup.row(types.InlineKeyboardButton('Легковий фургон',
callback_data='passenger_van'),
                      types.InlineKeyboardButton('Пікап', callback_data='pickup'))
    inline_markup.row(types.InlineKeyboardButton('Інше', callback_data='other'))

    return inline_markup


# noinspection PyBroadException
def car_type_next_step(message):
    try:
        car_info = car_info_dict[message.from_user.id]
        car_info.car_type = message.text

        msg = bot.send_message(message.chat.id, f'<b>Якого кольору ваше авто?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, car_color_next_step)
    except Exception as e:
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
def car_color_next_step(message):
    try:
        color = message.text
        if color.isdigit():
            msg = bot.reply_to(message, f'<b>Ви ввели недопустиме значення. Спробуйте
ще раз</b>', parse_mode='HTML')
            bot.register_next_step_handler(msg, car_color_next_step)
            return
        car_info = car_info_dict[message.from_user.id]
        car_info.car_color = message.text

        msg = bot.send_message(message.chat.id, f'<b>У якому році випустили вашу
машину?</b>', parse_mode='HTML')
        bot.register_next_step_handler(msg, car_year_next_step)
    except Exception as e:
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
def car_year_next_step(message):
    try:
        car_year = message.text

        if not car_year.isdigit() or int(message.text) > 2022 or int(message.text) <
1975:
            msg = bot.reply_to(message, f'<b>Ви ввели недопустиме значення. Спробуйте
ще раз</b>', parse_mode='HTML')
            bot.register_next_step_handler(msg, car_year_next_step)
            return

        car_info = car_info_dict[message.from_user.id]
        car_info.car_year = car_year
```

```
        msg = bot.send_message(message.chat.id, f'<b>Який номерний знак авто?</b>',
parse_mode='HTML')
        bot.register_next_step_handler(msg, car_number_next_step)
    except Exception as e:
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
def car_number_next_step(message):
    try:
        car_info = car_info_dict[message.from_user.id]
        car_info.car_number = message.text

        msg = bot.send_message(message.chat.id, f'<b>Додайте короткий опис вашого
авто.</b> Ваш опис допоможе комусь '
                                                f'обрати саме ваше авто для перевезення
вантажу.\n(Наприклад, вкажіть '
                                                f'максимальну вагу вантажу, яку можете
взяти з собою, розміри '
                                                f'багажника чи розміри буди, якщо авто
грузове і тд.)',
                               parse_mode='HTML')
        bot.register_next_step_handler(msg, car_description_next_step)
    except Exception as e:
        bot.reply_to(message, "Щось пішло не так...")


# noinspection PyBroadException
def car_description_next_step(message):
    try:
        id_owner = message.from_user.id
        car_info = car_info_dict[id_owner]
        car_info.car_description = message.text

        sql = 'INSERT INTO car_info(car_brand, car_model, car_type, car_color,
car_year, car_number, car_description,' \
              ' id_owner) VALUES(%s, %s, %s, %s, %s, %s, %s, %s) RETURNING id_car'
        val = (car_info.car_brand, car_info.car_model, car_info.car_type,
car_info.car_color, car_info.car_year,
               car_info.car_number, car_info.car_description, id_owner)
        cursor.execute(sql, val)

        car_info.id_car = cursor.fetchone()[0]

        db_connection.commit()

        print(cursor.rowcount, "record inserted.")

        del car_info_dict[message.from_user.id]

        bot.send_message(message.chat.id, f'Тепер <b>{car_info.car_brand}
{car_info.car_model}</b> може приймати '
                                          f'оголошення🐱',
reply_markup=main_markup(message.from_user.id),
                         parse_mode="HTML")
    except Exception as e:
        print(e)
        bot.reply_to(message, "Щось пішло не так...")
```

```
# noinspection PyBroadException
def add_phone_car(message):
    try:
        if message.content_type == "contact":
            id_user = message.from_user.id
            phone = message.contact.phone_number

            sql = f'UPDATE users SET phone = {phone} WHERE id = {id_user};'
            cursor.execute(sql)
            db_connection.commit()

            msg = bot.send_message(message.chat.id, f'<b>Яка марка вашого авто?</b>',
reply_markup=main_markup(id_user),
                                   parse_mode='HTML')
            bot.register_next_step_handler(msg, car_brand_next_step)

        else:
            bot.send_message(message.chat.id, f'Щось пішло не так! Спробуйте ще
раз...',
                             reply_markup=main_markup(message.from_user.id),
parse_mode="HTML")
    except Exception as e:
        bot.reply_to(message, "Щось пішло не так...")


def add_phone_advertisement(message):
    if message.content_type == "contact":
        id_user = message.from_user.id
        phone = message.contact.phone_number

        sql = f'UPDATE users SET phone = {phone} WHERE id = {id_user};'
        cursor.execute(sql)
        db_connection.commit()

        msg = bot.send_message(message.chat.id, f'В якому місті знаходиться
посилка?🛍',
                               reply_markup=main_markup(id_user))
        bot.register_next_step_handler(msg, parcel_city_a)
    else:
        bot.send_message(message.chat.id, f'Щось пішло не так! Спробуйте ще раз...',
                         reply_markup=main_markup(message.from_user.id),
parse_mode="HTML")


def main_markup(id_owner):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)

    btn1 = types.KeyboardButton('Везти посилку🛍')
    btn2 = types.KeyboardButton('Опублікувати оголошення 💬')
    btn3 = types.KeyboardButton('Мої оголошення 💬')
    btn4 = types.KeyboardButton('✅Мої поїздки')

    if get_number_of_cars(id_owner) > 0:
        btn5 = types.KeyboardButton(f'Моє авто ({get_number_of_cars(id_owner)}🚘)')
    else:
        btn5 = types.KeyboardButton('Додати своє авто 🚘')

    markup.row(btn1)
    markup.row(btn2, btn3)
```

```
    markup.row(btn4)
    markup.row(btn5)

    return markup


def get_number_of_cars(id_owner):
    cursor.execute(f'SELECT COUNT(*) FROM car_info WHERE id_owner = {id_owner};')
    count = cursor.fetchone()[0]
    return count


bot.polling(none_stop=True, interval=0)
```

Додаток Б

Лістинг файлу configure.py:

```
config = {
    'name': 'tudy_sudy_bot',
    'token': '5202027455:AAFPca2jWsHrnhO9I7KDfryHSDn0AvOCXo833',
    'db_URI':
'postgres://wrssjwwtnmfinf:1bff9e66522facf76187145b5d4e3f395b789860bea2ad9bcdf4763cc129
579f@ec2-63-35'
             '-156-160.eu-west-1.compute.amazonaws.com:5432/dcnagi5ov45ph2',
}
```