ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики і інформатики

«До захисту допущено»
Завідувач кафедри

_____ Ольга ДМИТРІЄВА
"7" червня 2022 р.

Випускна кваліфікаційна робота
бакалавра
(освітній ступень)

на тему
«Розробка телеграм боту для отримання актуальної інформації погодних умов в країні» зі спецчастиною
«Розробка програмних модулів, інтерфейсу і бази даних засобами Python, SQLite»

Виконав: студент 4 курсу, групи ІПЗ-18

Спеціальності 121 Інженерія програмного забезпечення

Васильєв А.Ю. _____
(прізвище та ініціали) (підпис)

Керівник доц. каф. ПМІ, к.т.н., доц. Ірина НАЗАРОВА _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент доц. каф. КІ, к.т.н., доц. Сергій КОВАЛЬОВ _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Нормоконтроль:

_____ Ірина НАЗАРОВА
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Дата 23.05.2022

Студент _____
(підпис)

Луцьк – 2022 р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерних наук і технологій
Кафедра прикладної математики та інформатики
Освітньо-кваліфікаційний рівень (освітній ступінь) бакалавр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ
Ольга ДМИТРІЄВА
_____
"____" _____2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Антону Васильєву
1. Тема проєкту (роботи) Розробка телеграм боту для отримання актуальної інформації погодних умов в країні
Спецчастина Розробка програмних модулів, інтерфейсу і бази даних засобами Python, SQLite
Керівник проєкту (роботи) Ірина Назарова, доц. каф. ПМІ, к.т.н., доц.
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "06" травня 2022 року №180
2. Строк подання студентом проєкту (роботи) 17.06.2022р.
3. Вихідні дані до проєкту (роботи) результати науково-дослідної роботи, матеріали переддипломної практики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) аналіз предметної області і постановка завдання;
2) проєктування програмної системи;
3) створення бази даних;
4) підключення до бази даних;
5) проєктування та розробка інтерфейсу;
6) розробка системи моніторингу даних веб-сайту;
7) розробка телеграм-боту;
8) тестування чат-боту.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1) блок-схема розробленої системи моніторингу даних веб-сайту;
2) діаграма варіантів використання;
3) екранні форми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 6 квітня 2022року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області і постановка завдання	10.04.2022	
2.	Проєктування системи	15.04.2022	
3.	Створення бази даних	18.04.2022	
4.	Розробка алгоритму веб-скрапінгу	21.04.2022	
5.	Проєктування та розробка інтерфейсу користувача	23.04.2022	
6.	Розробка системи моніторингу даних веб-сайту	28.04.2022	
7.	Тестування телеграм-боту	25.05.2022	
8.	Оформлення пояснювальної записки	01.06.2022	
9.	Підготовка слайдів презентації	05.06.2022	
10.	Підготовка демонстраційної версії розробленого програмного продукту	10.06.2022	
11.	Написання доповіді	12.06.2022	

Студент _____ Антон Васильєв
(підпис) (ім'я, прізвище)

Керівник роботи _____ Ірина Назарова
(підпис) (ім'я, прізвище)

АНОТАЦІЯ

Антон Васильєв. Розробка телеграм-боту для отримання актуальної інформації погодних умов в країні / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення ДВНЗ ДонНТУ, Покровськ, 2022.

Об’єктом дослідження є процеси проектування, розробки та тестування боту у месенджері Telegram.

Предметом дослідження є методи та алгоритми розробки чат-боту.

Метою роботи є розробка телеграм-боту для отримання актуальної інформації погодних умов в країні на мові програмування Python з використанням бази даних SQLite .

Практичне значення розробки телеграм-боту для отримання актуальної інформації погодних умов в країні полягає створені програмного додатку, основними завданнями якого є надання прогнозу погоди на поточний день, на п’ять днів, прогнозу погоди на дорозі та надання рекомендацій, щодо одягу у поточну погоду.

Методи дослідження, що застосовані при розробці телеграм-боту: метод порівнянь, метод проєктування програмного забезпечення, метод проєктування баз даних та емпіричний метод. Реалізація телеграм-боту здійснювалася завдяки основним положенням теорії кінцевих автоматів, теорії алгоритмів, теорії об’єктно-оріентованого програмування та модульного програмування

Ключові слова: телеграм-бот, Python, прогноз погоди, SQLite, Aiogram, кінцевий автомат, парсинг даних.

ANNOTATION

Anton Vasyliev. Development of a telegram-bot to obtain up-to-date information on weather conditions in the country / Final qualification work for obtaining the educational qualification level "bachelor" in specialty 121 Software Engineering of DonNTU, Pokrovsk, 2022.

The object of research is the processes of design, development and testing of the bot in the Telegram messenger.

The subject of research is the methods and algorithms of chatbot development.

The aim of the work is to develop a telegram bot to obtain up-to-date information on weather conditions in the country in the Python programming language using the SQLite database.

The practical significance of developing a telegram bot to obtain up-to-date information on weather conditions in the country is to create a software application, the main tasks of which are to provide weather forecast for the current day, five days, weather forecast and provide recommendations for clothing in current weather

Research methods used in the development of the telegram-bot: the method of comparisons, the method of software design, the method of database design and the empirical method. Implementation of the telegram-bot was carried out due to the basic provisions of the theory of finite state machines, the theory of algorithms, the theory of object-oriented programming and modular programming

Keywords: telegram bot, Python, weather forecast, SQLite, Aiogram, state machine, data parsing.

ЗМІСТ

ВСТУП

Те, що погода впливає на людину, факт незаперечний. Число людей, які відчувають зміни погоди постійно зростає.

При різкій зміні тиску, температури повітря, зміні вологості та магнітних бурях у людей з'являються головні болі, біль у суглобах, з'являється сонливість, погіршується настрій. Так проявляється метеозалежність. Через це, інформація про погодні умови та рекомендації до них є та будуть актуальні завжди.

В сучасних умовах все більше і більше людей проводять час у менеджерах, там вони спілкуються, отримують новини та різну інформацію.

Один з таких месенджерів Telegram [1], який застосований в роботі.

Завдяки сервісу «боти» у месенджері люди можуть отримувати корисну інформацію та користуватися різними функціями.

Виходячи з цього, темою кваліфікаційної роботи було обрано «Розробка телеграм боту для отримання актуальної інформації погодних умов в країні».

Метою роботи є розробка телеграм-боту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОСНОВНІ ЗАВДАННЯ РОБОТИ

1.1 Актуальність вибраної теми

Відслідковування прогнозу погоди завжди є актуальним, але є тенденція, яка свідчить про те, що через неправильне харчування та малорухливий спосіб життя люди все частіше стають метеозалежними і ця кількість людей зростає.

Тому важливо слідкувати за погодою та дотримуватися рекомендацій, щодо одягу.

Люди все частіше проводять час у смартфонах, а саме у месенеджерах, тому необхідність кросплатформених додатків зростає.

Один з таких кросплатформенних месенджерів – Telegram. Месенджер має багато переваг серед конкурентів, а саме: безпечність, канали, зручність, боти, стікери, анонімність. Відтак, Telegram стрімко набирає популярність та не має передумов для іншої тенденції (рисунок 1.1).

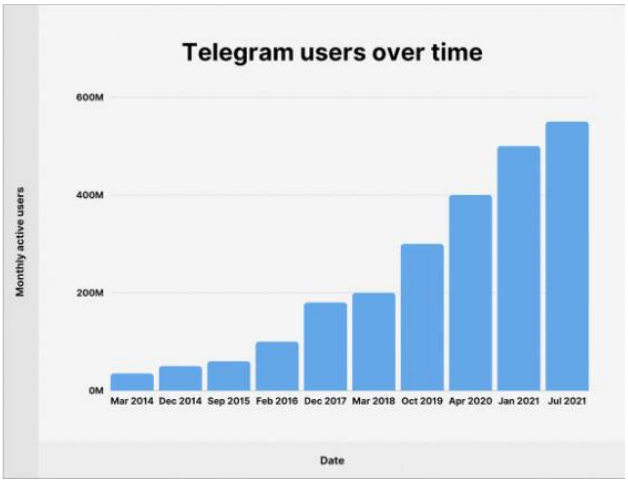

Рисунок 1.1 – Тенденція росту месенджера Telegram

Джерело: [2]

1.1 Об’єкт і предмет дослідження роботи

Об’єктом дослідження є процеси проектування, розробки та тестування боту у месенджері Telegram.

Предметом дослідження є методи та алгоритми розробки чат-боту.

1.2 Мета роботи

Метою роботи є розробка телеграм-боту для отримання актуальної інформації погодних умов в країні на мові програмування Python [3-5] з використанням бази даних SQLite [6].

1.3 Практичне завдання роботи

Практичне значення розробки телеграм-боту для отримання актуальної інформації погодних умов в країні полягає створені програмного додатку, основними завданнями якого є надання прогнозу погоди на поточний день, на п’ять днів, прогнозу погоди на дорозі та надання рекомендацій, щодо одягу у поточну погоду.

1.4 Методи дослідження

Методи дослідження, що застосовані при розробці телеграм-боту: метод порівнянь, метод проєктування програмного забезпечення, метод проєктування баз даних та емпіричний метод. Реалізація телеграм-боту здійснювалася завдяки основним положенням теорії кінцевих автоматів, теорії алгоритмів, теорії об’єктно-оріентованого програмування та модульного програмування

1.5 Обґрунтування методів дослідження

Емпіричний метод [7] дослідження був реалізований для отримання інформації, щодо структуризації системи та її проєктування. Метод порівнянь [7] був застосований за допомогою емпіричного методу для аналізу

конкурентного середовища. Метод проєктування програмного забезпечення було використано для процесу розробки. За допомогою методу проєктування баз даних були створенні бази даних.

1.6 Аналіз конкурентних продуктів

У месенджері Telegram існують боти, що надають інформацію прогнозу погоди, але їх функціонал частіш за все обмежений, не включає усі населені пункти та не має рекомендацій.

Таким прикладом є бот «Pogoda». На рисунках нижче відображено його функціонал (рисунок 1.2-1.3).

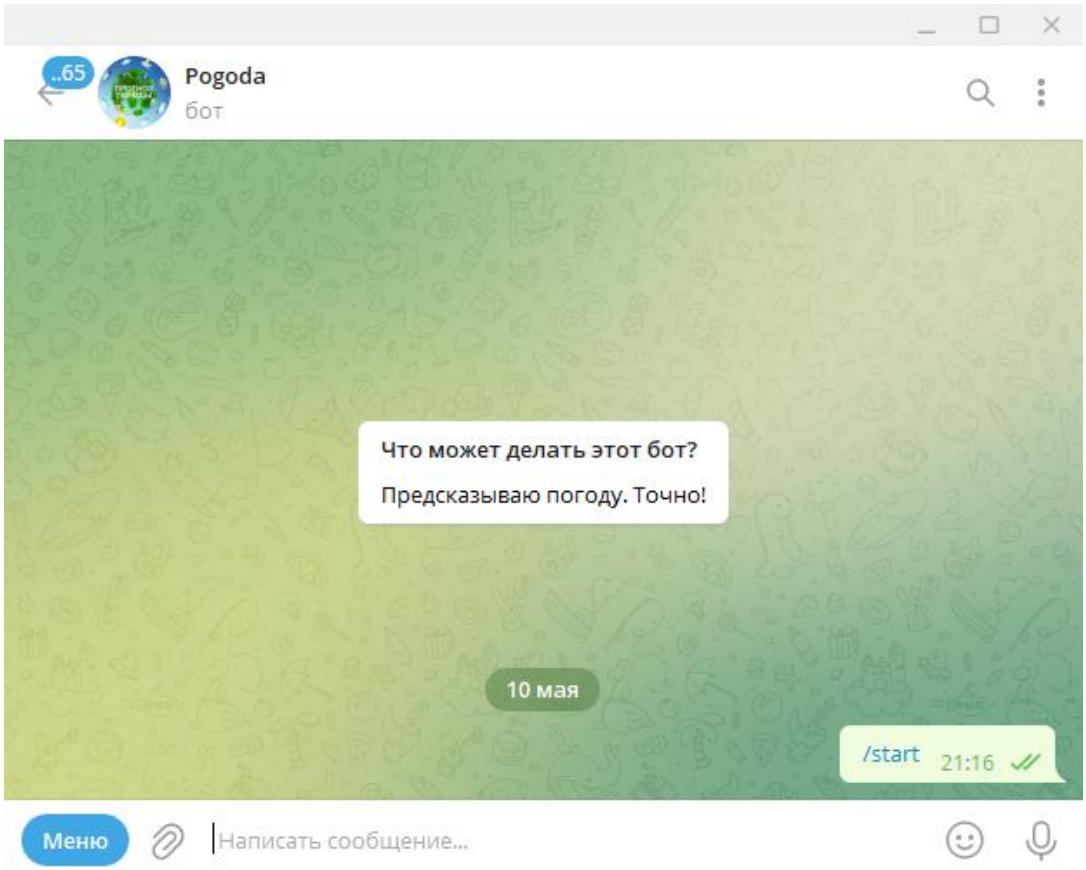

Рисунок 1.2 – Старт бота

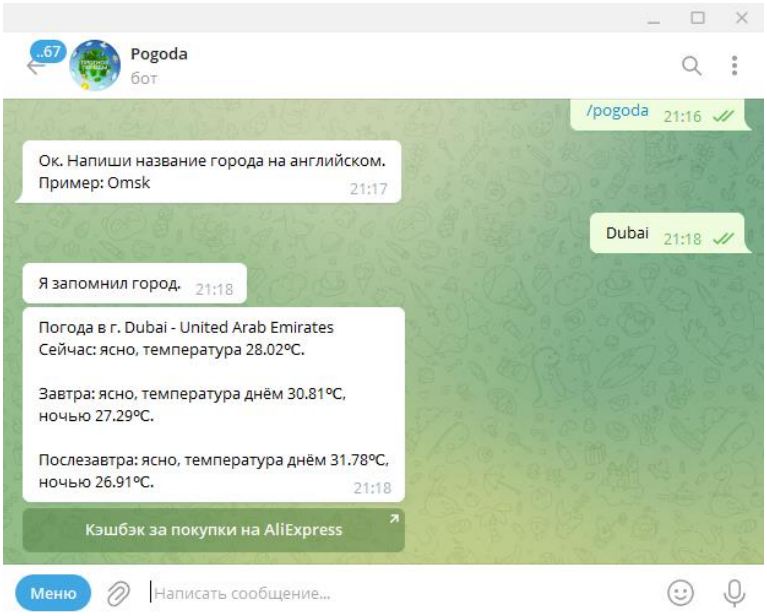

Рисунок 1.3 – Температура у заданому місті

Ще одним прикладом є бот з назвою «Погодник». На рисунках 1.4-1.5 нижче відображено його функціонал.

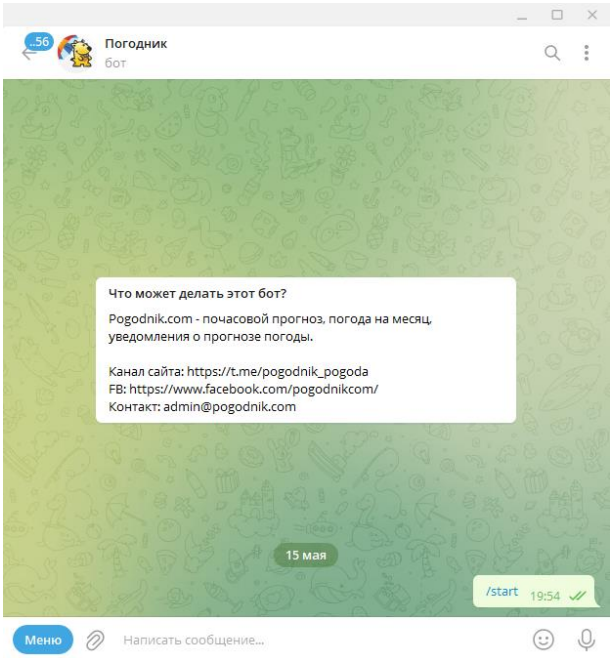

Рисунок 1.4 – Початок роботи з ботом

Функціонал бота та його інтерфейс відрізняється від попереднього конкурента.

Замість команд, завдяки яким відбувається взаємодія з ботом, в цьому телеграм-боті реалізовані кнопки, які є зручними для користувача. У боті відсутня інформація, щодо погодних умов на даний момент. Чат-бот буде надсилати лише інформацію, щодо погодних у майбутньому, що є недоліком. На рисунку 1.5 відображено взаємодію з телеграм-ботом.

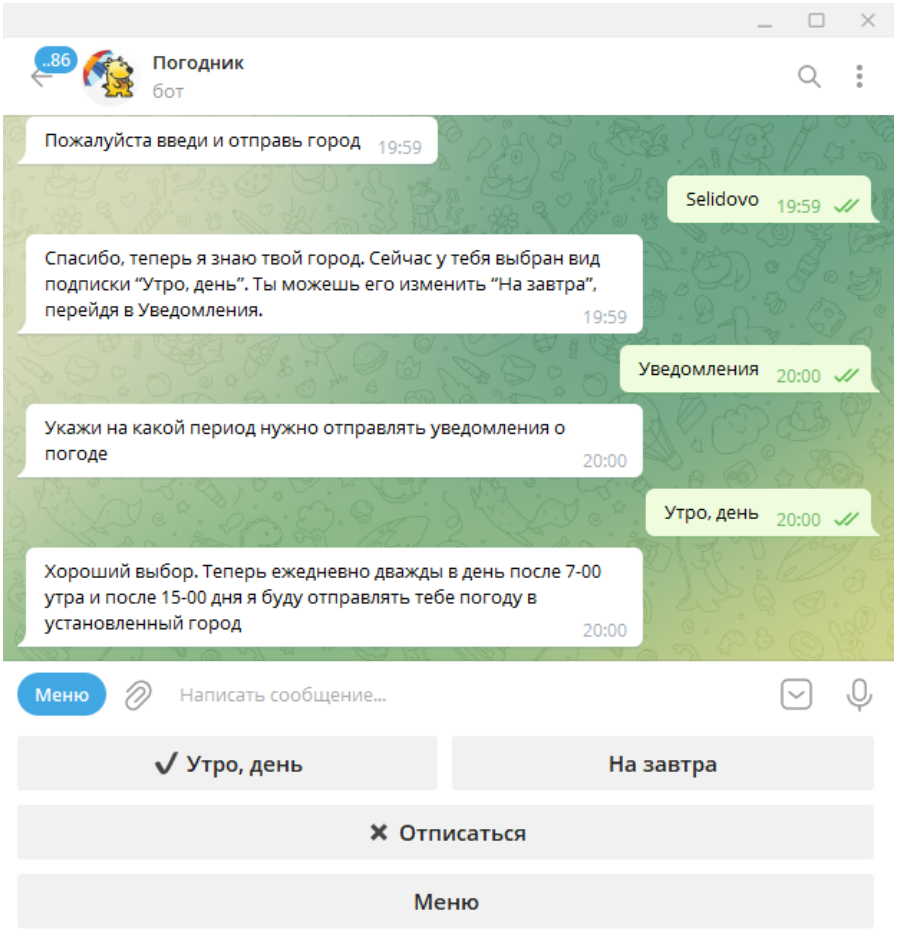

Рисунок 1.5 – Взаємодія з телеграм-ботом

Виконавши аналіз конкурентних продуктів можна зробити висновок, що існують конкурентні продукти, але їх опції обмежені і не включають можливі корисні функції.

1.7 Інструменти, засоби та технології для розробки телеграм-боту

Для отримання прогнозу погоди потрібно застосувати технологію API [8], яка дозволяє виконати HTTP-запит [9] та отримати HTTP-відповідь.

Телеграм-бот прогнозу погоди має функцію виведення прогнозу погоди на дорозі, для реалізації цієї функції потрібно виконати парсинг [10] та упорядковування даних.

Функціонал бота має змогу давати прогноз погоди за вибраним містом, змінювати місто, давати рекомендації, щодо одягу та має додаткові кнопки, через це необхідно запам’ятовувати місто та змінювати стани, тобто потрібно використовувати кінцевий автомат [11].

Для того, щоб запам’ятовувати місто потрібна база даних [12], в якій буде зберігатися інформація, щодо міста, яке увів користувач.

Рекомендації, щодо одягу представлені у вигляді фотографій, для зберігання фотографій використовується сервер телеграма. Завантаження фотографій до серверу телеграма виконується через команди адміністратора. Для кожної фотографії з сервера телеграму в базі даних створюється унікальний ID, який дозволяє звертатися до фотографії та видавати її користувачу.

Виходячи з цього програму можна розділити на модулі за допомогою модульного програмування [13] з використанням об'єктно-орієнтованого програмування [14]. Для наочності та зручності розробки програми слід використовувати UML-діаграми [15].

1.7.1 Технологія API

Для отримання даних з сайтів використовується технологія API [8], яка дозволяє надіслати до веб-сайту HTTP-запит [9] та отримати відповідь з необхідними даними. Ця технологія підтримується не на всіх веб-ресурсах, але її наявність значно спрощує процес отримання даних. На рисунку 1.6 наочно продемонстровано технологію API.

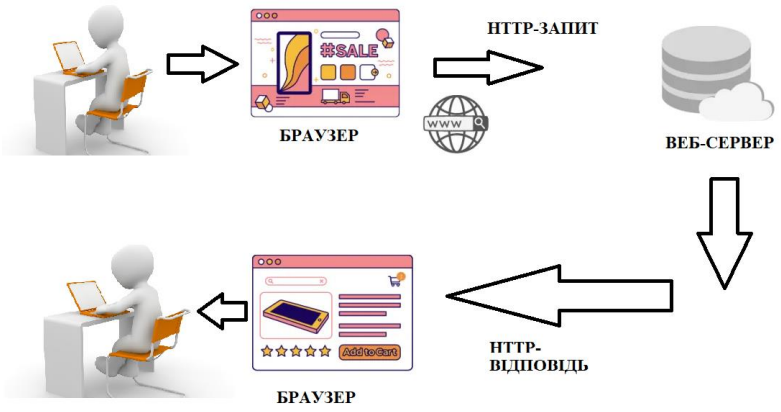

Рисунок 1.6 – Технологія API

1.7.2 Парсинг даних

Для отримання інформації даних зі сторінок веб-ресурсів було застосовано технологію парсингу даних [10], яка дозволяє зчитувати дані з веб-сторінок. На рисунку нижче відображено послідовність процесів технології (рис. 1.7).

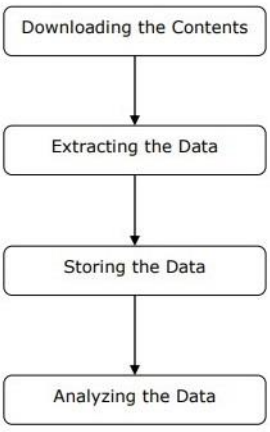

Рисунок 1.7 – Послідовність процесів парсингу даних

В роботі технологія парсингу даних реалізована за допомогою Document Object Model [16].

DOM – це об'єктна модель документа, яку браузер створює в пам'яті комп'ютера на підставі коду HTML, отриманого ним від сервера (рис. 1.8).

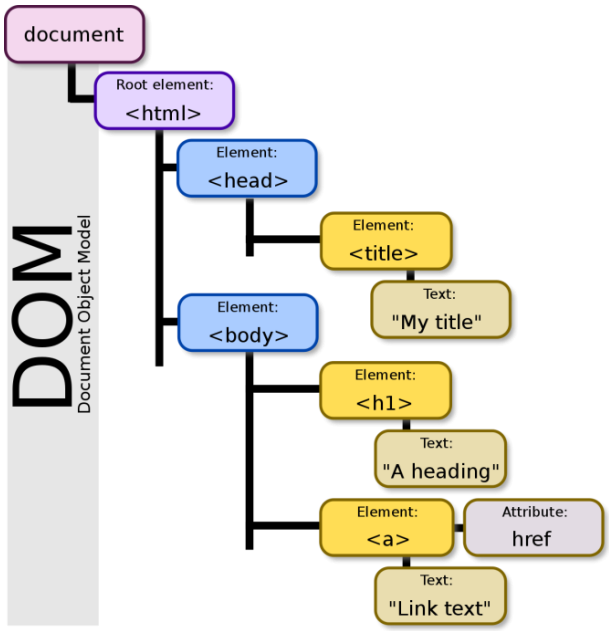

Рисунок 1.8 – Реалізація парсингу даних за допомогою DOM

Джерело: [16]

1.7.3 Кінцевий автомат

Для того, щоб змінювати стани об’єкту використовується кінцевий автомат.

Кінцевий автомат [11] – це модель, яка дозволяє змінювати шляхи стану об’єкта. Кінцевий автомат організовує поведінку програмного додатку.

На рисунку 1.9 зображено приклад роботи кінцевого автомату.

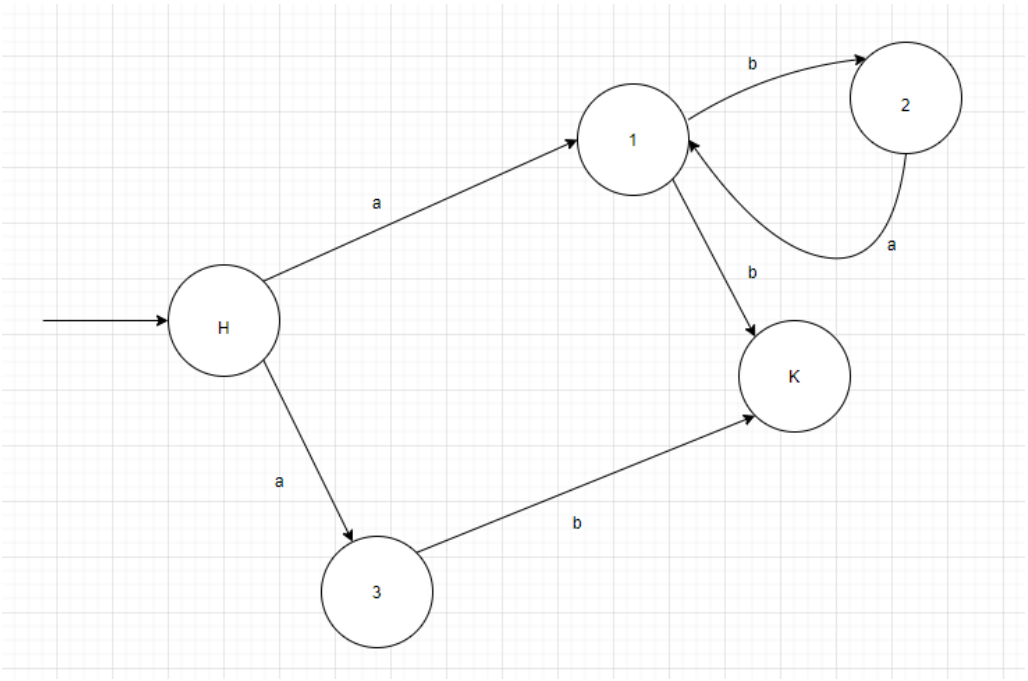

Рисунок 1.9 – Приклад роботи кінцевого автомату

1.7.4 Об’єктно-орієнтоване програмування

Об’єктно-орієнтоване програмування [14] – це метод програмування, при якому програма взаємодіє з об'єктами або частиною об'єктів. Головні функції ООП:

– структуризація коду;

– зменшення обсягу коду;

– читабельність.

У програмному коді було застосовано класи та глобальні змінні.

Клас – це елемент, який необхідний для конструювання програми.

Клас має поля та методи. Стан об’єкту визначається полями класу. Поведінка об’єкта визначається методами класу.

На рисунку 1.10 продемонстровано приклад оголошення класу та ініціалізація поля.

```
class FSMAdmin(StatesGroup):
    set_city = State()
    change_city = State()
    photo = State()
    weather_des = State()
    temp_low = State()
    temp_high = State()
```

Рисунок 1.10 – Оголошення класу та ініціалізація його поля

Глобальні змінні – це змінні, у якої область видимості є вся програма.

1.7.5 Модульне програмування

Для зручності програму було розбито на модулі. Головна мета модульного програмування [13] – це спрощення задач проєктування. Завдяки модульному програмуванню процес доопрацювання програми спрощується, завдяки тому, що доробляти можна окремі модулі, а не весь програмний продукт.

На рисунку 1.11 продемонстрована структура модульного програмування.

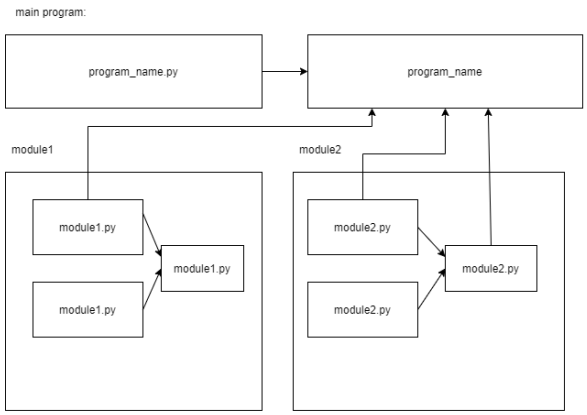

Рисунок 1.11 – Структура модульного програмування

1.7.6 UML мова та UML-діаграми

UML мова [15] – це графічна візуальна мова моделювання для проєктування та опису системи.

В UML для опису моделі системи є два основних поняття:

– сутність;

– відношення.

Сутність – це основні елементи моделей. Сутності поділяються на чотири типи, а саме:

– структурні;

– поведінкові;

– анотаційні;

– групуючі.

Відношення показує зв’язок між двома сутностями. Відношення поділяються на чотири типи:

– залежності;

– асоціація;

– реалізація;

– узагальнення.

Для демонстрації відношень сутностей використовується UML-діаграми. Вони поділяються на діаграми, що описують поведінку системи та діаграми, що описують фізичне розташування системи.

Діаграми поведінки системи поділяються на: діаграми станів, діаграми діяльності, діаграми об’єктів, діаграми кооперацій та діаграми послідовності.

Діаграми, що описуються фізичне розташування системи поділяються на: діаграми компонентів та діаграми розгортання.

1.8 Основні завдання роботи

Виходячи з поставленої мети необхідно виконати такі завдання:

– виконати аналіз предметної області;

– дослідити конкурентне середовище;

– виконати збір інформації, щодо погоди;

– виконати парсинг даних для прогнозу погоди на дорозі;

– розробити структуру кінцевого автомату;

– розробити структуру бази даних;

– отримати програмну реалізацію додатку на мові Pyhton.

1.9 Висновки за розділом

У цьому розділі було проведено системний аналіз кваліфікаційної роботи. Було обґрунтовано тему, її актуальність та визначені методи дослідження.

Проведено аналіз конкурентних продуктів, який показав, що частіш за все конкурентні продукти мають обмежений функціонал.

Були описані інструменти, технології та засоби, які використовуються при створені програмного продукту, а саме:

– технологія API;

– парсинг даних, реалізація якого виконана за допомогою Document Object Model;

– кінцевий автомат;

– об’єктно-орієнтоване програмування;

– модульне програмування;

– UML-діаграми.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Опис архітектури системи

Щоб створити у месенджері Telegram нового бота треба знайти бота – BotFather. Це офіційний інструмент для створення ботів та управління ними.

Для написання ботів існує Telegram Bot API. Щоб керувати Telegram Bot API потрібно обрати бібліотеку для мови Python. Найбільш популярні серед них:

- Telepot python-telegram-bot;
- Aiogram;
- Telegram Bot Service;
- Telebot.

Найбільш популярною бібліотекою для ботів на мові Python є – aiogram [17]. Вона асинхронна, використовує декоратори та містить зручні інструменти для розробки. Через це була обрана бібліотека aiogram.

Для зручності система була розбита на модулі, а саме:

- модуль обробки команд ;
- модуль роботи з базою даних;
- модуль запуску боту;
- модуль обробки кнопок.

Для користування ботом було створено команду /start, яка викликає головне меню кнопок.

2.1.1 Побудова UML-діаграм

Для побудови UML-діаграм [15] було обрано сервіс app.diagrams.net. За допомогою цього сервісу можливо створити усі необхідні UML-діаграми: діаграми використання, діаграми діяльності, діаграми розміщення.

Переваги apps.diagrams.net:

– графічне редагування;

– безкоштовне користування сервісом;

– широкий набір шаблонів;

– значний набір елементів;

– інтуїтивний інтерфейс;

– має усі найпопулярніші формати зберігання діаграм.

Архітектура телеграм-боту побудова так, що має дві сутності:

– адміністратор;

– користувач.

Адміністратор може використовувати дві команди:

– /download;

– /cancel.

Команда /download запускає стан, при якому адміністратор має змогу завантажити фотографії з одягом та вибрати параметри, які будуть відповідати саме цій фотографії.

Команда /cancel дозволяє адміністратору перервати процес завантаження фотографій.

На рисунку нижче представлена діаграма використання для актора «Адміністратор» (рисунок 2.1).

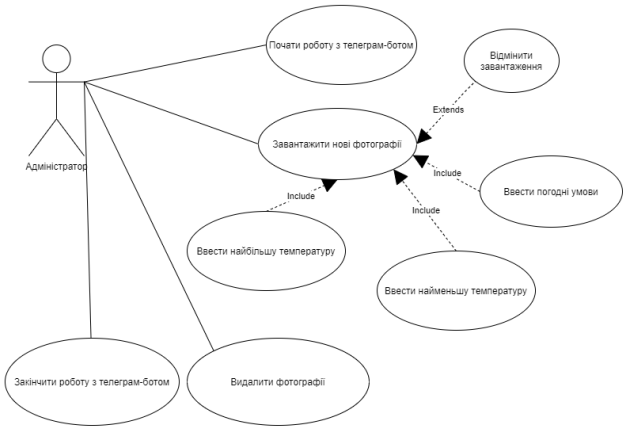

Рисунок 2.1 – Діаграма використання для актора «Адміністратор»

На рисунку 2.2 відображена діаграма діяльності для актора «Адміністратор».

Рисунок 2.2 – Діаграма діяльності для актора «Адміністратор»

Користувач має команду /start та основні три кнопки «Погода», «Погода на дорозі», «Змінити місто».

Команда /start запускає бота та дає змогу користувачу ввести місто. Команда /start також ініціалізує для користувача клавіатуру з трьома основними кнопками.

Кнопка «Погода» має підменю, яке складається з чотирьох кнопок:

– поточна погода;

– прогноз погоди на 5 днів;

– рекомендації щодо одягу;

– назад.

Кнопка «Поточна погода» видає користувачу інформацію, щодо поточної погоди у вибраному місті.

Кнопка «Прогноз погоди на 5днів» видає інформацію про погодні умови на п’ять днів.

Кнопка «Рекомендації щодо одягу» видає користувачу фотографію, відповідно до поточної погоди.

Кнопка «Назад» переміщає у головне меню.

На рисунку нижче продемонстрована діаграма використання користувача (рисунок 2.3).

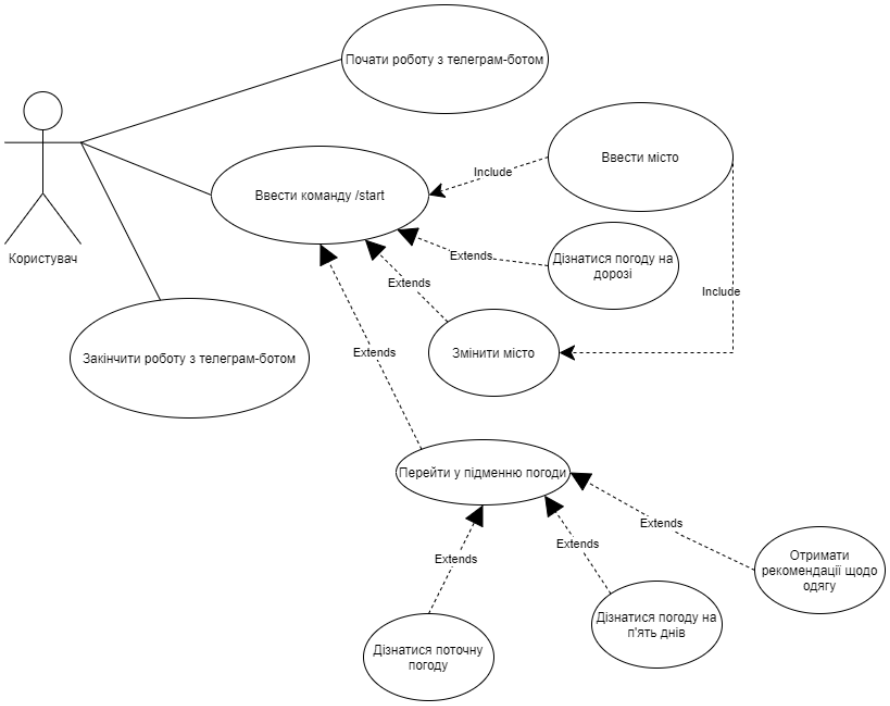

Рисунок 2.3 – Діаграма використання для користувача

Для роботи бота його було розміщено на локальному комп’ютері. Щоб користувачі могли взаємодіяти з телеграм-ботом їм потрібно увійти у месенджер Telegram телеграм на будь-якому девайсі. На рисунку нижче побудовано діаграму розміщення (рисунок 2.4).

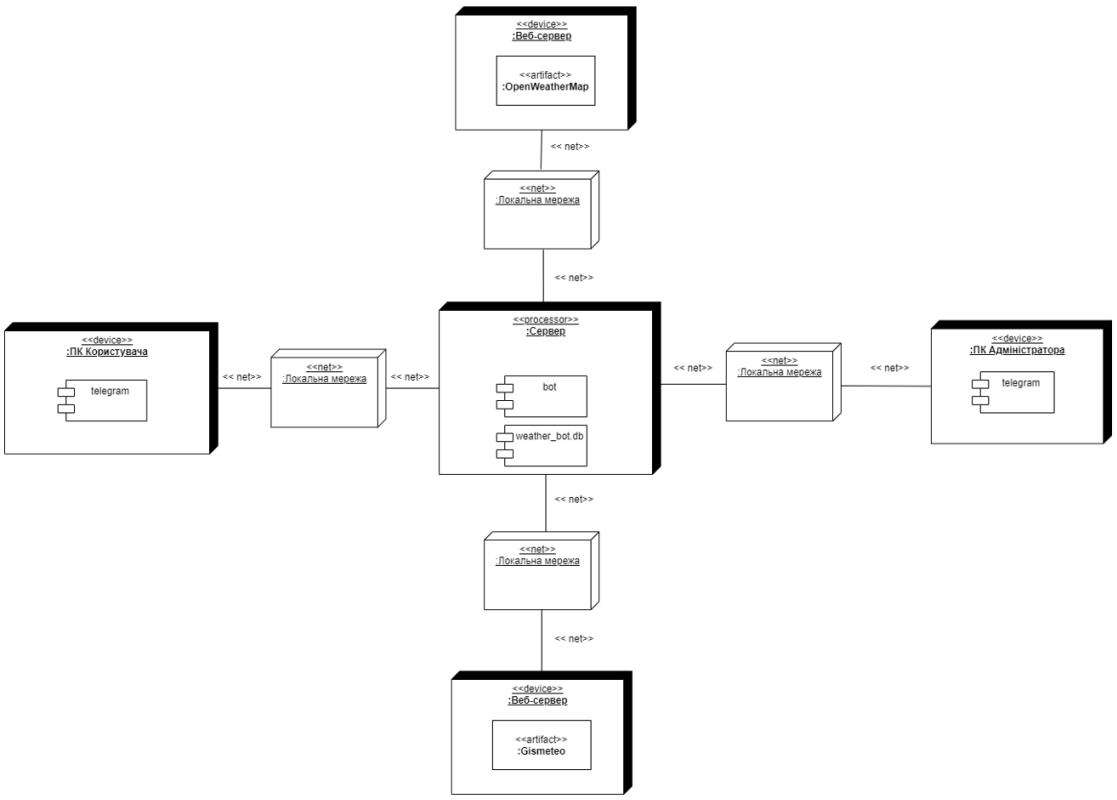

Рисунок 2.4 – Діаграма розміщення

2.2 Мова програмування

Для написання ботів використовують різні язики програмування, кожен з них має свої переваги, найбільш популярних язик програмування для створення ботів – Python [3-5], який дозволяє швидко та без проблемем виконати створення боту. Це пов'язано з широкими можливостями, які доступні при використанні стандартних бібліотек, так і з використання вже готов варіантів, таких як aiogram, які розраховані на роботу безпосередньо з Telegram.

2.3 Середовище розробки

Середовищем розробки бота на мові Python було обрано PyCharm [18].

PyCharm – це кросплатформене середовище розробки, яке дозволяє швидко проводити рефакторінг коду, а також використовувати зручний графічний відладчик. Має повний комплект засобів, які необхідні для ефективного програмування на мові Python.

Переваги Pycharm.

PyCharm – це редактор коду для мови Python. Він має такі функції: підсвічування синтаксису, автоматичне форматування, доповнення та відступи. Завдяки PyCharm можна перевіряти версії інтерпретатора мови на сумісність та використовувати шаблони коду.

В PyCharm можна використовувати інтерактивні консолі для відладчика та баз даних.

Редактор має велику кількість плагінів та його можна використовувати з різними трекерами.

Середовище розробки PyCharm має безкоштовну версію.

На рисунку 2.5 продемонстрований графічний дизайн PyCharm.

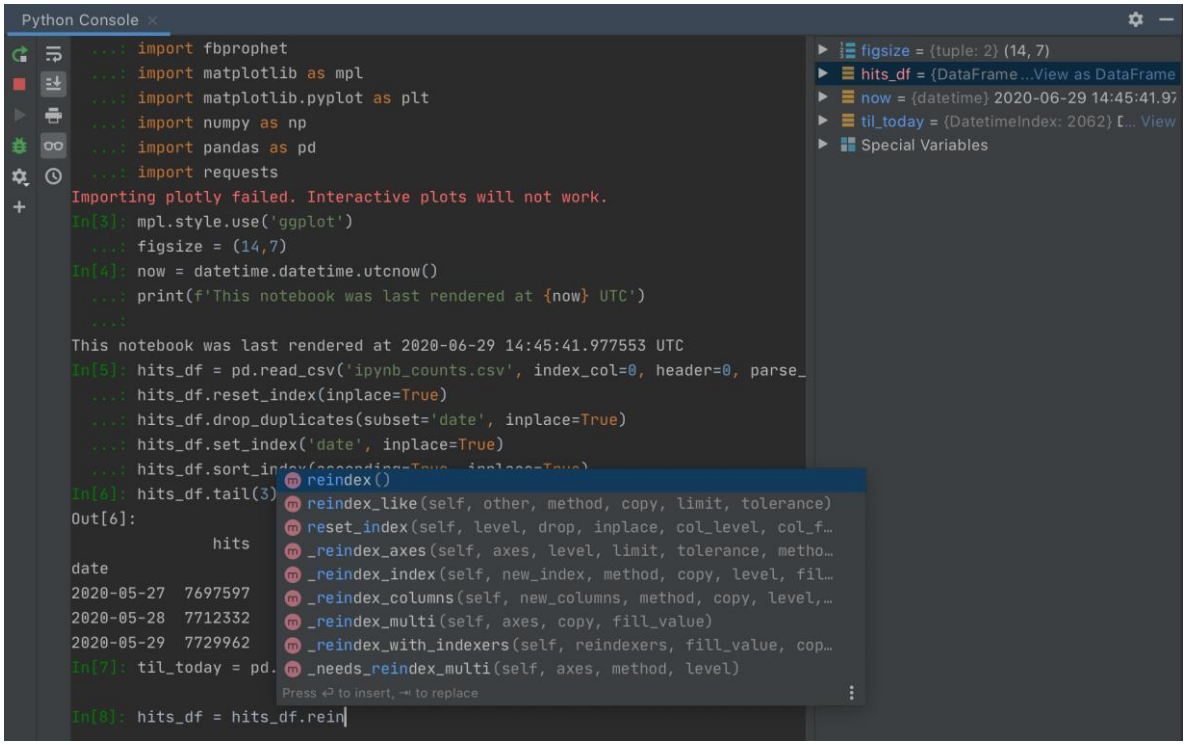

Рисунок 2.5 – Середовище розробки проекту

2.4 База даних

Для роботи з базами даних на мові Python був обраний модуль SQLite3.

SQLite [6] – це швидка однофайлова СКБД, яка не має сервера та дозволяє зберігати всю базу на одному пристрої, не потребує сторонніх бібліотек та служб.

Переваги SQLite:

- висока швидкість;
- зберігання даних в одному файлі;
- мінімалізм;
- доступність;
- кросплатформеність;
- автономність;
- малий розмір.

2.5 Графічний додаток для бази даних

Для зручності створення та редагування файлів бази даних був обраний графічний додаток DB Broswer [19].

DB Broswer – простий у використанні інструмент, який дозволяє підключатися до бази даних, переглядати, змінювати дані чи запускати sql сценарії. На рисунку 2.6 продемонстровано роботу програми.

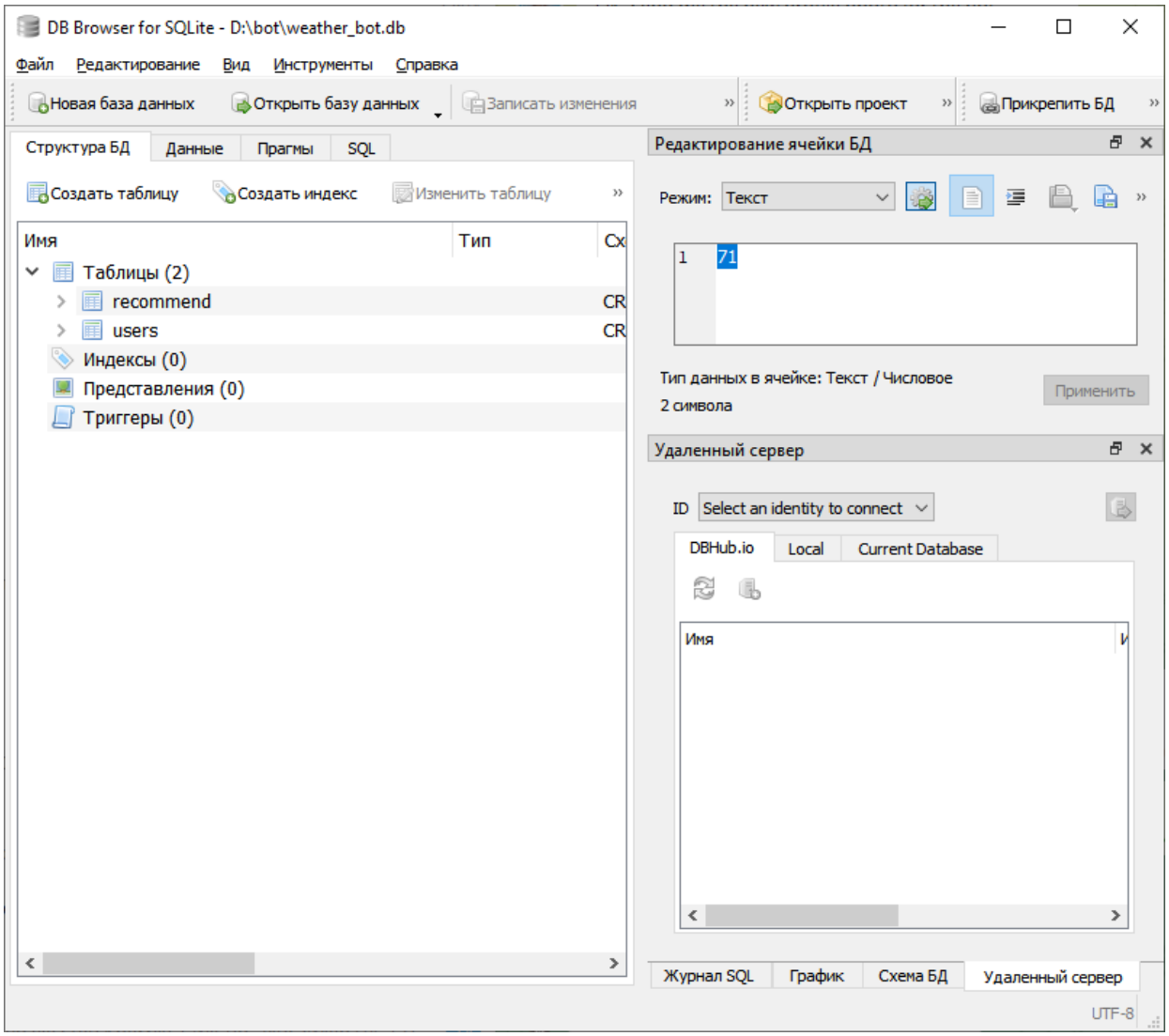

Рисунок 2.6 – DB Browser

2.6 Парсинг даних

Для зчитування даних з веб-ресурсу, а саме https://www.gismeteo.ua/, було реалізовано технологію парсингу даних. Було використано такі бібліотеки:

– BeautifulSoup;

– Requests.

BeautifulSoup – це пакет для аналізу документів html, який перетворює їх на синтаксичні дерева. За допомогою html та xml парсерів він видобуває потрібні дані.

Requests дає змогу реалізувати запити для сайтів. На рисунку 2.7 продемонстровано запит до сайту https://www.gismeteo.ua/.

```
1  import requests
2
3  res = requests.get('https:/gismeteo.ua')
4
5  print(res)
```

Рисунок 2.7 – Запит до сайту

2.7 Проєктування інтерфейсу

Для побудування макетів інтерфейсу Telegram боту було обрано веб-ресурс https://www.mockflow.com/. Цей веб-ресурс дозволяє реалізувати технологію вайрфрейму [20].

Переваги mockflow:

– автономний режим без доступу к облаку;

– інтуїтивність;

– великий набір шаблонів та інструментів;

– має безкоштовну версію.

Вайрфрейм – це технологія дизайну . Вайрфрейм повинен показувати:

– основну групу контенту;

– структуру інформації;

– опис та взаємодію між користувачем та інтерфейсом програмного продукту.

На рисунку 2.8 зображено макет telegram-бота, створений за допомогою mockflow.

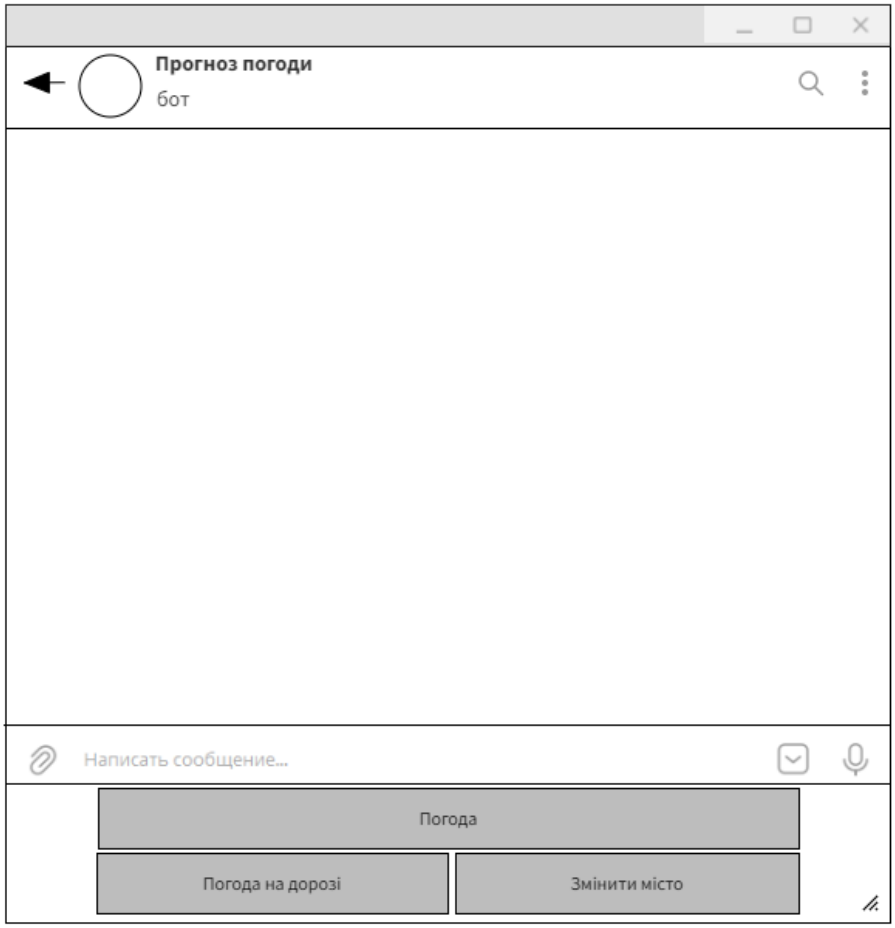

Рисунок 2.8 – Макет телеграм-боту

2.8 Висновки за розділом

У цьому розділі була описана архітектура системи, база даних, модулі системи та спроєктований інтерфейс системи.

Для користування API Telegram була обрана бібліотека aiogram, середовищем розробки було обрано PyCharm. Для роботи з базами даних було обрано SQLite, графічний редактор для якої DB Browser. Для зчитування даних с веб-ресурсу було реалізовано технологію веб-скрапінг. Для створення макету інтерфейсу було застосовано веб-ресурс mockflow.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ

3.1 Реєстрація та персоналізація телеграм-боту

Щоб створити нового бота спочатку потрібно його зареєструвати. Для цього потрібно знайти бота у телеграм, який відповідає за реєстрацію нових ботів та їх налаштування, а саме офіційний бот Telegram BotFather. На рисунку нижче відображено вікно з BotFather (рис. 3.1).

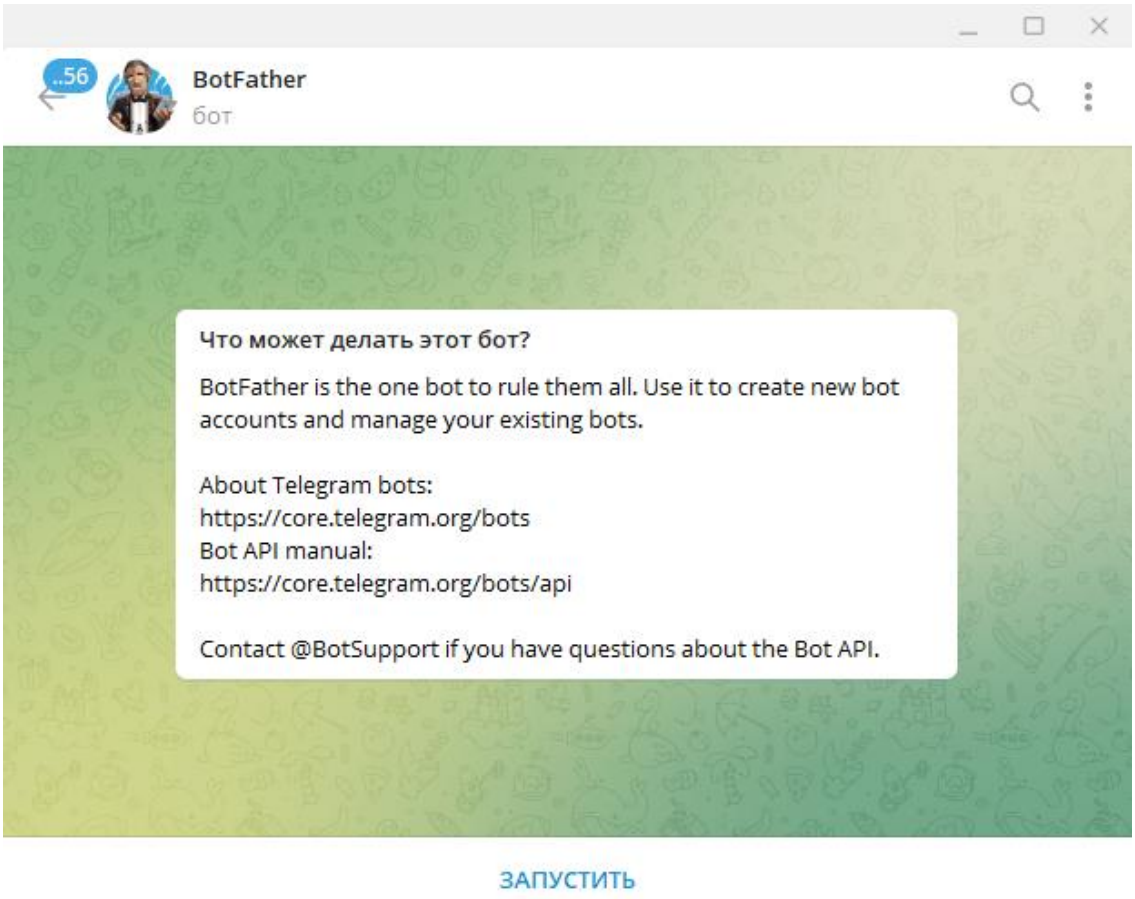

Рисунок 3.1 – Офіційний бот Telegram BotFather

Після натискання кнопки «запустити» потрібно ввести команду /newbot, ввести його назву та ім’я користувача.

Наступним етапом BotFather надасть токен, за допомогою якого можна буде керувати ботом. На рисунку 3.2 відображено процес створення боту та отримання токену.

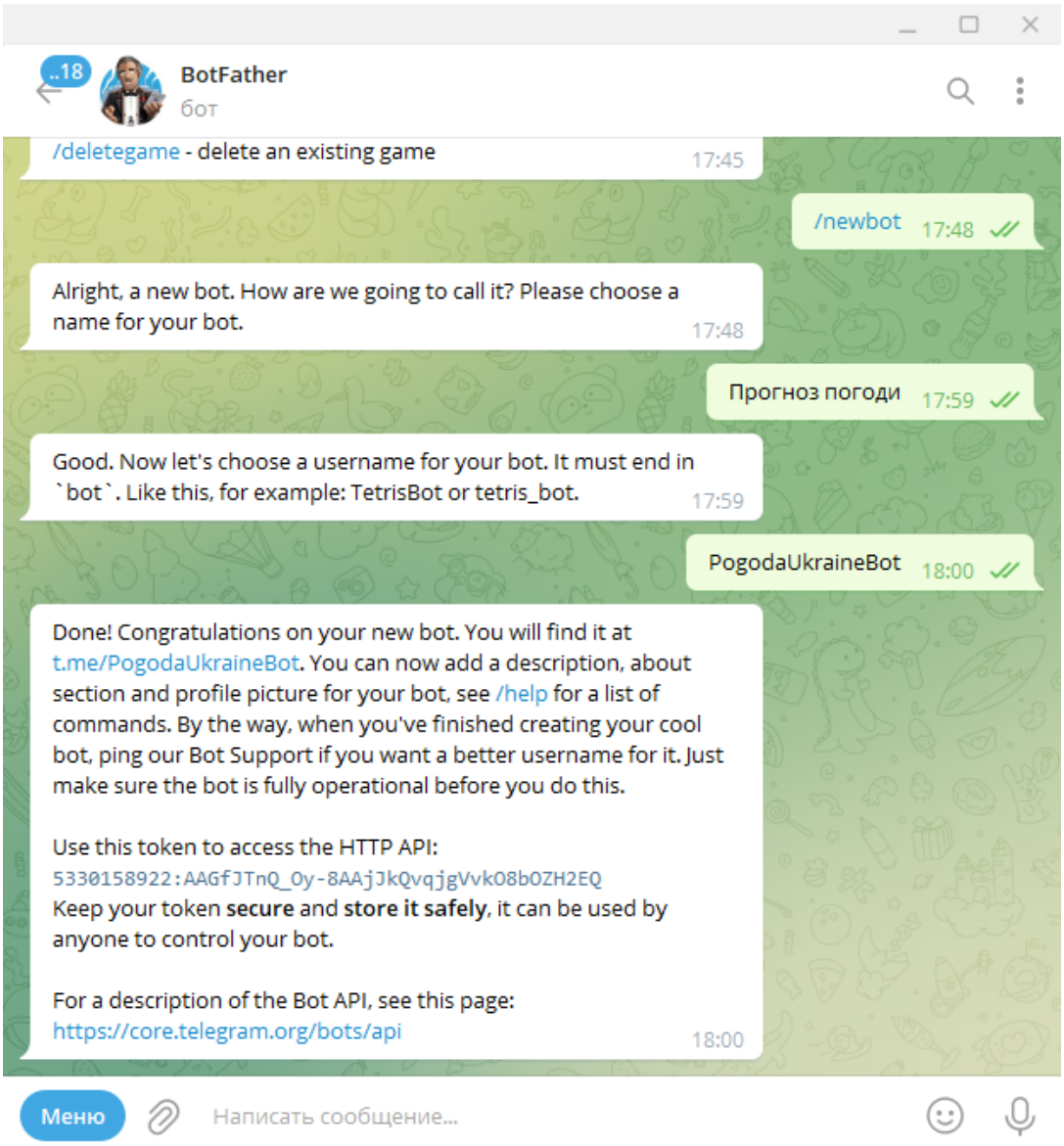

Рисунок 3.2 – Процес створення боту та отримання токену

Після цього потрібно виконати персоналізацію чат-боту, а саме налаштувати фото для бота, додати опис у вікно «Інформація про бота» та додати опис у вікні з телеграм ботом.

Для того щоб додати фото до чат-боту потрібно ввести команду /setuserpic та обрати бота, якому потрібно виконати надбудову. На рисунку 3.3 відображено процес налаштування фото для телеграм-боту «Прогноз погоди».

Рисунок 3.3 – Процес налаштування фото для телеграм-боту

Щоб додати опис у вікно з телеграм-ботом потрібно ввести команду /setdescription, обрати бота та надіслати текст, який є описом телеграм бота.

Для налаштування опису у вікні «Інформація про бота» з телеграм-ботом потрібно ввести команду /mybots, за допомогою меню вибрати пункт «Edit bot». Далі потрібно перейти до пункту «Edit About» та ввести опис чат-боту. На рисунку 3.4 продемонстровано виконані налаштування персоналізації.

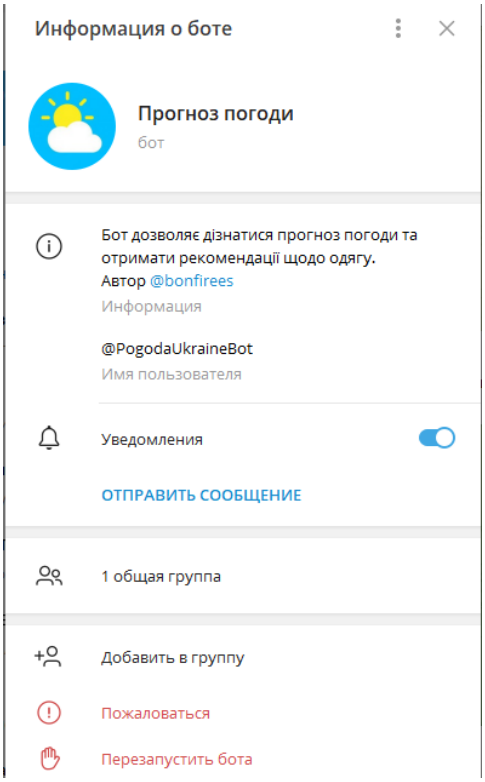

Рисунок 3.4 – Виконані налаштування

3.2 Початок роботи з PyCharm

Для початку потрібно створити новий проєкт, якому додамо назву bot та вкажемо шлях, де буде знаходитися проект з ботом. Для проекту використаємо віртуальне оточення Virtualenv і Python3.10. На рисунку нижче продемонстровано процес створення проєкту в PyCharm (рис. 3.5).

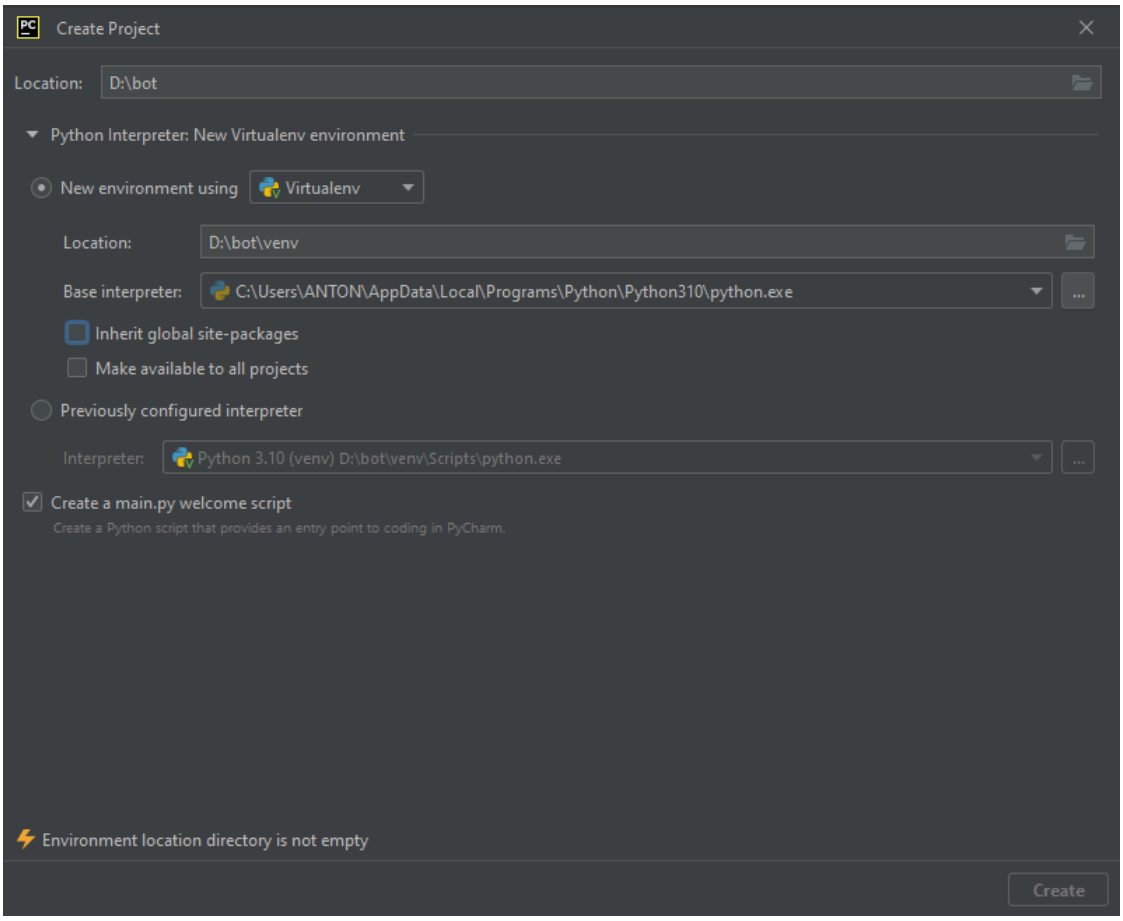

Рисунок 3.5 – Процес створення проекту

3.3 Модулі проекту

Проєкт створений та розбитий по модулям, окремі модулі відповідають за окремі частини. На рисунку 3.6 відображено усі модулі, які існують в програмному додатку.

Рисунок 3.6 – Модулі програмного додатку

3.3.1 Модуль botPogoda.py

Модуль botPogoda.py є головним модулем програми, в ньому створюється екземпляр боту, диспетчера та об’єкта пам’яті, завдяки яким буде виконуватися керування ботом. Для цього потрібно імпортувати клас Bot,

Dispatcher, MemoryStorage та виконавець executor, які імпортовані з бібліотеки aiogram.

Екземпляр боту (bot) – об’єкт класу Bot, який створюється конструктором з параметром token, який імпортований з модуля data_token.py.

Диспетчер (dis) – об’єкт класу Dispatcher, конструктор якого приймає в якості параметрів екземпляр боту та пам’яті.

Пам’ять (storage) – об’єкт класу MemoryStorage, який призначений для запам’ятовування повідомлень від користувача в оперативній пам’яті.

Також у цьому модулі застосовано асинхронну функцію on_startup для підключення бази даних. Для цього імпортовано модуль database.py

Усі обробники, які було розроблено в програмі реєструються викликом функції register_handler_handlers(dis), вона імпортується з модулю handler.py.

Лістинг представлено у додатку В.1.

3.3.2 Модуль buttons.py

Модуль buttons.py відповідає за створення кнопок та клавіатури. Для створення кнопок і клавіатури потрібно імпортувати класи ReplyKeyboardMarkup та KeyboardButton з бібліотеки aiogram.

Функціонал бота складається з семи кнопок:

- назад;
- погода;
- погода на дорозі;
- змінити місто;
- рекомендації щодо одягу;
- поточна погода;
- прогноз погоди на 5днів.

Лістинг представлено у додатку В.2.

3.3.3 Модуль data_token.py

Модуль data_token.py запроваджений для окремого зберігання API токенів телеграм-боту та погоди. Також у цьому модулі є ID користувача, яке необхідне для перевірки на адміністратора бота. Завдяки тому, що токени та айди користувача винесені в окремий модуль, це гарантує безпеку цих даних.

Лістинг представлено у додатку В.3.

3.3.4 Модуль database.py

Модуль розроблено для взаємодії з базою даних. Для роботи з базою даних необхідно імпортувати модуль sqlite, який є вбудованим інструментом Pyhton.

База даних містить дві таблиці: users та recommend.

Таблиця users містить такі поля:

- user_id з типом даних integer, є первинним ключем;
- first_name з типом даних text;
- city з типом даних text.

Таблиця recommend складається з таких полів:

- photo з типом даних text;
- weather_des з типом даних text;
- temp_low з типом даних real;
- temp_high з типом даних real.

На рисунку 3.7 відображена структура таблиць бази даних, за допомогою DB Browser.

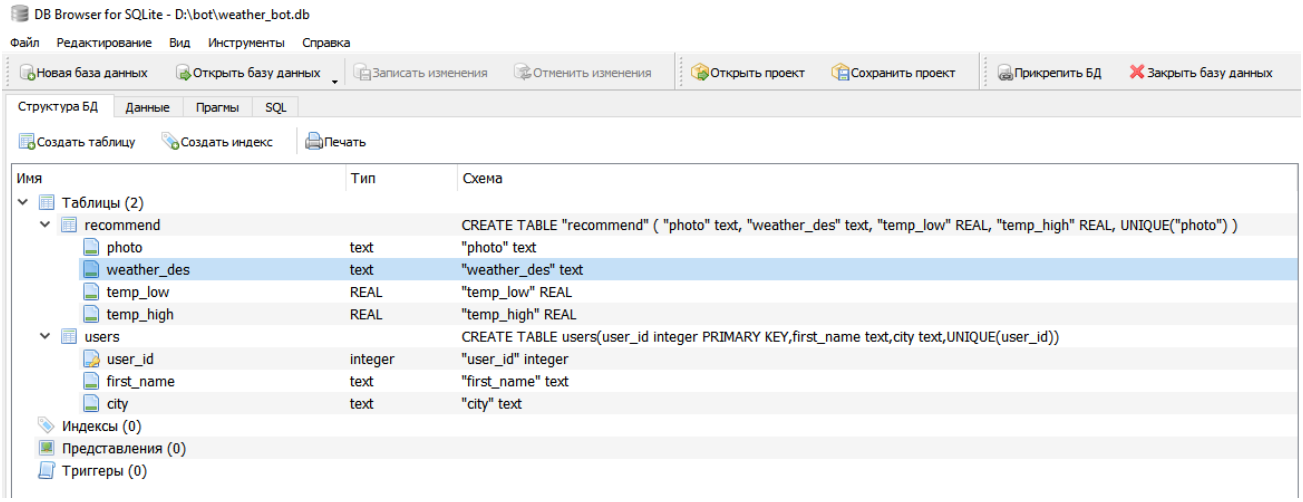

Рисунок 3.7 – Таблиці бази даних

Модуль database.py складається з таких функцій:

– sql_start – ця функція підключає базу даних, якщо така існує або створює нову, якщо такої не існує;

– add_photo – функція яка додає до бази даних фото та параметри за якими відбувається вибірка;

– sql_add_user – ця функція потрібна для занесення користувача до бази даних, а саме: його id, нікнейм у телеграм та місто, яке користувач ввів при запуску бота;

– sql_check_user – використовується для перевірки користувача, якщо він вже користувався ботом, містить аргументи user_id;

– sql_update_city – функція, яка заносить у базу даних нове місто, введене користувачем, має аргументи city та user_id;

– sql_get_city – призначена для того, щоб зчитати місто, яке ввів користувач для видання поточної погоди, погоди на п’ять днів, рекомендації, щодо одягу та погоди на дорозі;

– id_photo – ця функція потрібна для видання користувачу рекомендації щодо одягу.

При виклику цієї функції створюється тимчасова таблиця, в яку заносяться параметри поточної погоди для відправки фото за параметрами. По завершенню роботи функції тимчасова таблиця видаляється.

Лістинг представлено у додатку В.4.

3.3.5 Модуль handler.py

Цей модуль відповідає за обробку команд, кнопок та повідомлень. Також у цьому модулі створений клас машини станів. У кінцевому автоматі передбачено такі стани:

– set_city – стан у який переходить система для очікування введення назви міста;

– change_city – стан, у який переходить система для очікування введення нового міста;

– photo – стан, при якому система очікує від адміністратора завантаження фото;

– weather_des – стан введення опису параметру погоди;

– temp_low – стан, у який переходить система для введення найнижчої температури;

– temp_high – стан, при якому система очікує введення найвищої погоди.

Модуль має такі асинхронні функції:

– start_bot – відповідає за запуск бота командою /start, ініціалізує клавіатуру, в ході роботи функції кінцевий автомат приймає стан set_city;

– admin_start – функція, яка викликається за допомогою команди /download, її призначення складається у тому, що вона перевіряє чи є користувач адміністратором, кінцевий автомат приймає стан photo;

– cancel – викликається командою /cancel, функція дає змогу перервати процес завантаження фото у базу даних;

– load_photo – призначена для того, щоб занести фото, яке завантажує адміністратор у базу даних, кінцевий автомат приймає стан photo;

– load_des – функція для вводу опису параметру погоди у базу даних, кінцевий автомат приймає стан weather_des;

– load_ low_temp – відповідає за внесення у базу даних найнижчої температури, у ході роботи функції кінцевий автомат приймає стан temp_low;

– load_ high_temp – відповідає за внесення у базу даних найвищої температури, кінцевий автомат приймає стан temp_high;

– Set_city – функція, яка обробляє повідомлення користувача, коли введено місто, якщо місто вірне, то запис про id юзера, його нікнейм та місто додається до бази даних, інакше користувачу відправляється повідомлення «Місто введено невірно, введи ще раз», у ході роботи функції кінцевий автомат приймає стан set_city;

– change_city – функція яка обробляє кнопку «Змінити місто», при натисканні цієї кнопки користувач отримає повідомлення «Введіть нове місто»;

– new_city – викликається після отримання повідомлення про нове місто користувача, якщо місто вірне, то запис додається до бази даних, інакше потрібно ввести нове місто, кінцевий автомат має стан – change_city;

– road_weather – обробник кнопки «Погода на дорозі», при натисканні цієї кнопки користувач отримує інформацію про погоду на дорозі, інформація про місто, який ввів користувач береться з бази даних, інформація, щодо прогнозу погоди на дорозі отримується за допомогою парсингу даних;

– weather – функція, яка є обробником кнопки «Погода», gри натисканні кнопки користувачу відкривається клавіатура чотирма кнопками: «Поточна погода», «Прогноз погоди на 5 днів», «Рекомендації щодо одягу», «Назад»;

– rec_clothes – кнопка «Рекомендації щодо одягу», в залежності від міста, яке увів користувач, зчитуються погодні умови та температура, після

цього з бази даних користувачу видається фото, яке відповідає цим параметрам;

– cur_weather – функція є обробником кнопки «Поточна погода», в залежності від вибраного міста користувачу видається повідомлення про поточну погоду;

– cur5_weather – обробник кнопки «Прогноз погоди на 5 днів», користувачу при натисканні цієї кнопки видається прогноз погоди на 5 днів по заданому місту;

– back – функція, яка відповідає за обробку кнопки «Назад», при натисканні цієї кнопки користувач потрапляє у головне меню;

– get_weather – функція, яка є обробником повідомлень від користувача. Функція відповідає за видалення повідомлень від користувача.

Функція register_handler_handlers реєструє усі обробники кнопок, команд та повідомлень. Функція не є асинхронною.

Лістинг представлено у додатку В.5.

3.3.6 Модуль pars.py

Цей модуль потрібен для парсингу даних, щодо погоди на дорозі з веб-ресурсу https://www.gismeteo.ua/ua/. Для роботи з цим модулем потрібно імпортувати клас BeatifulSoup з модуля bs4 та пакет requests. BeatifulSoup потрібен для вилучення даних з HTML та XML файлів. Requests потрібен для отримання HTTP-запитів. У модулі є функція grw, яка відповідає за парсинг даних, за допомогою цієї функції виконується парсинг даних, з блоків на сайті вибирається вміст тегів. Для того, щоб потрапити на потрібно сторінку з містом використовується конструкція 'https://www.gismeteo.ua/ua/search/' + city + '/'.

Для алгоритму парсингу даних було побудовано блок-схему. На рисунку нижче відображено блок-схему алгоритму парсингу даних (рис. 3.8).

Лістинг представлено у додатку В.6.

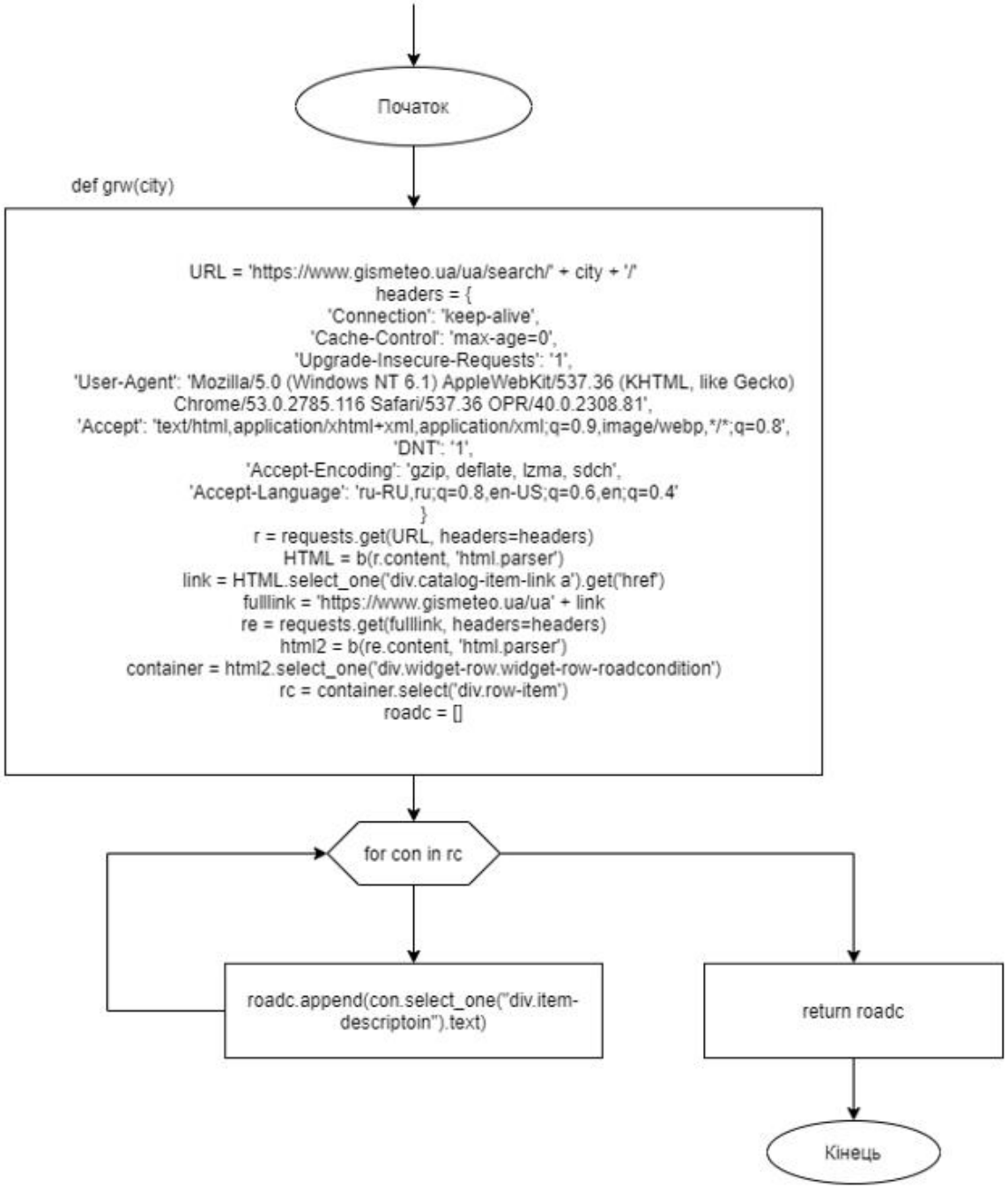

Рисунок 3.8 – Алгоритм парсингу даних

3.4 Висновки за розділом

У цьому розділі була описана розробка програмного засобу: реєстрація та персоналізація телеграм-боту, був створений проєкт, його код та розбитий на окремі модулі. Також була побудована блок-схема алгоритму парсингу даних.

4 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ І ЗАСОБИ БЕЗПЕКИ

4.1 Опис реалізації системи

Для того, щоб почати роботу з телеграм ботом потрібно натиснути кнопку «Запустити», яка відправить команду /start та виведе повідомлення «Привіт! Введи місто:». На рисунку 4.1 відображено початок робот з телеграм-ботом.

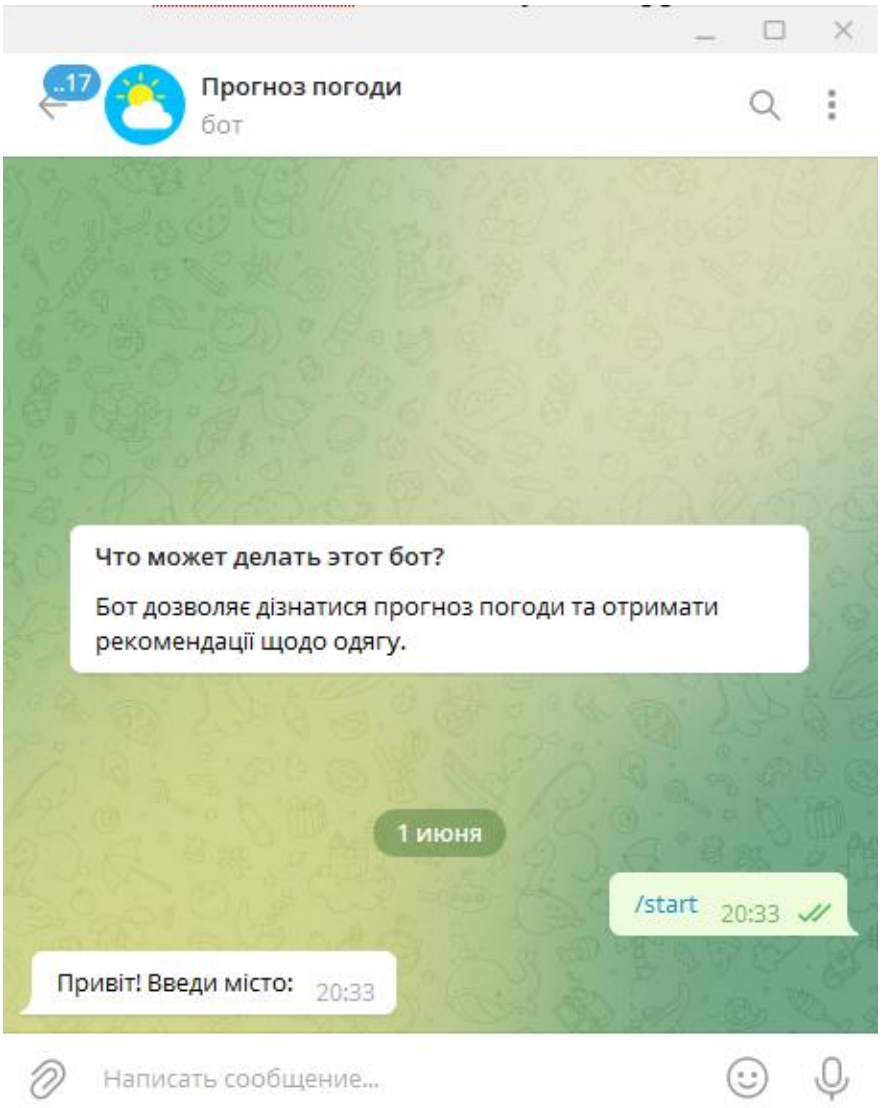

Рисунок 4.1 – Початок робот з телеграм-ботом «Прогноз погоди»

Якщо користувач введе місто неправильно, йому буде відправлено повідомлення «Місто введено невірно, введи ще раз». На рисунку нижче продемонстрований випадок, коли користувач ввів неправильне місто (рис. 4.2).

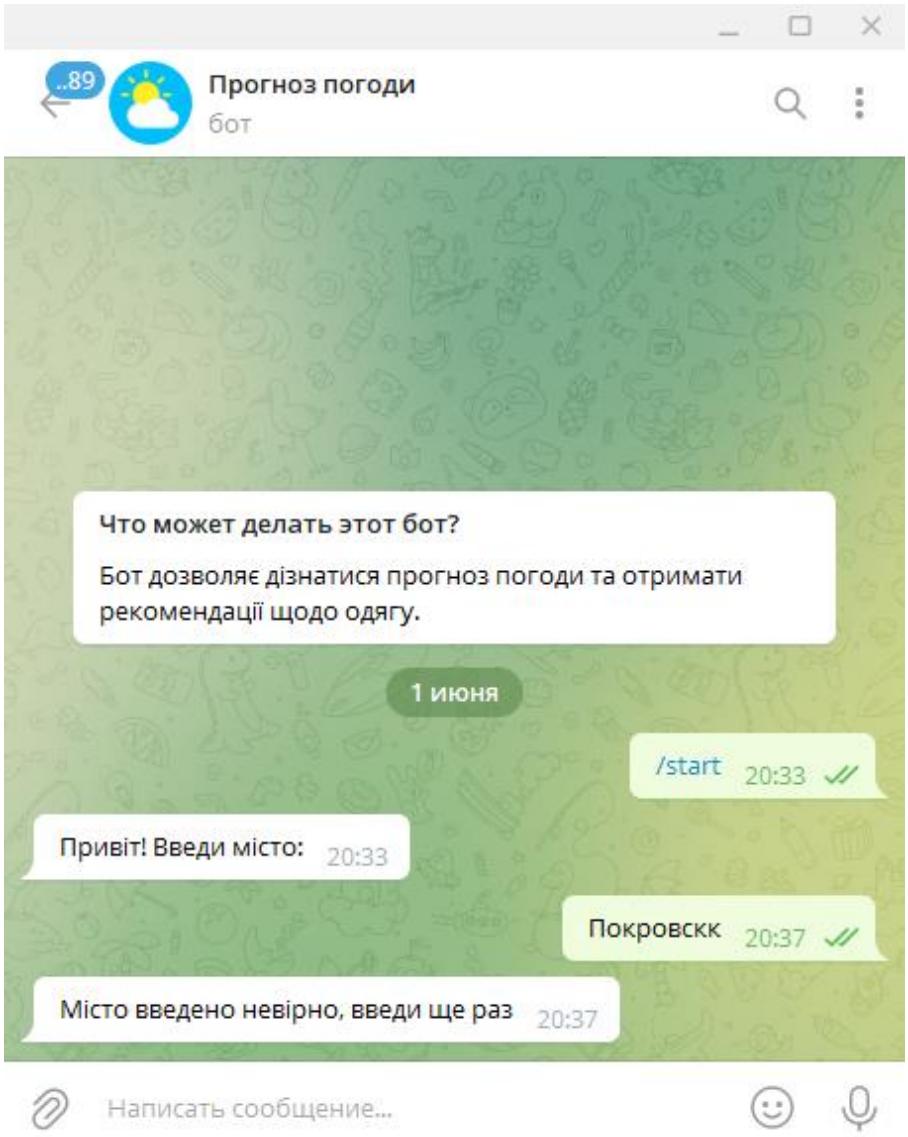

Рисунок 4.2 – Випадок, коли користувач ввів неправильне місто

Після того, як користувач введе коректне місто, йому буде відображено повідомлення «Вибране місто:місто» та ініціалізовано клавіатуру з трьома кнопками: «Погода», «Погода на дорозі», «Змінити місто». На рисунку 4.3 відображено випадок коли користувач ввів коректне місто.

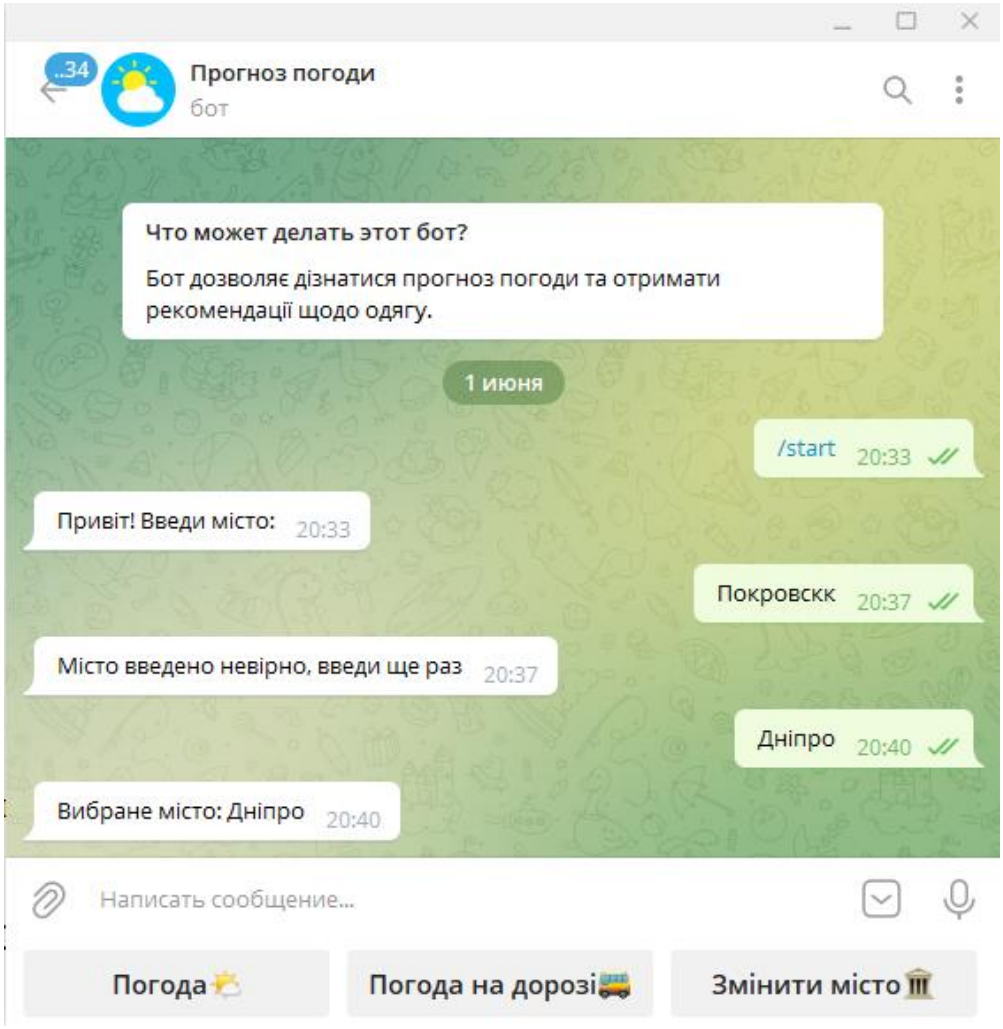

Рисунок 4.3 – Випадок, коли користувач ввів коректне місто

Якщо чат-боту відправити повідомлення , коли бот не просить нічого ввести, то буде видано повідомлення, що бот не приймає повідомлення. На рисунку нижче продемонстровано цей випадок (рис 4.4).

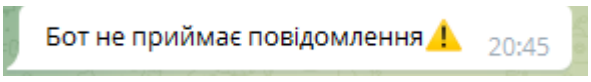

Рисунок 4.4 – Користувач відправив повідомлення

Коли користувач уперше запускає бота дані про нього додаються до бази даних (рис 4.5).

	user_id	first_name	city
	Фильтр	Фильтр	Фильтр
1	375481156	Евгений	Мирноград
2	425418688	Anton	Дніпро
3	448103922	Макс	Мохначка
4	474849206	Шизито Шизушкін	Селидове
5	627389380	Mura	Днепр
6	689137238	stepan	Харьков
7	1258005869	black queen	Днепр
8	1818103050	Nazar	Одеса
9	2066687797	Елена	Сочи
10	5001919548	Alex	львов
11	5038205185	Vasya	Львів
12	5187029537	Maja	Липовець
13	5225467129	Надія	Київ
14	5378729380	ВАСИЛЬ	ЖИТОМИР

Рисунок 4.5 – Додані записи про користувача

Через це коли користувач спробує запустити бота не в перший раз йому буде виведено повідомлення «Ти вже запустив бота» (рис 4.6).

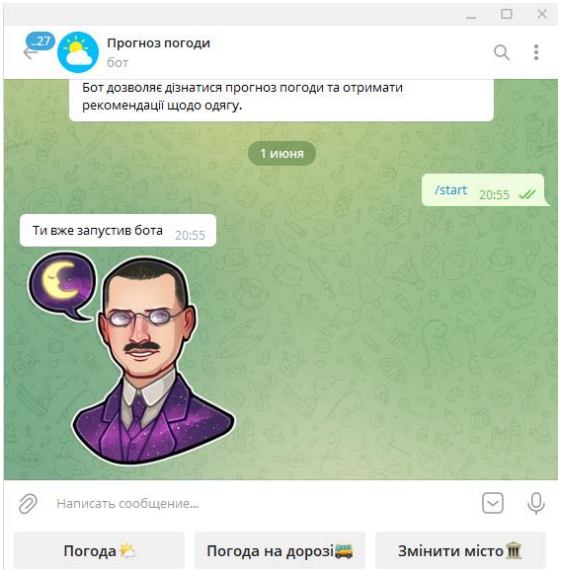

Рисунок 4.6 – Користувач не в перший раз запустив бота

Після того, як користувач коректне ввів місто та ініціалізувалась клавіатура, користувач має змогу змінити місто. Для цього потрібно натиснути кнопку «Змінити місто». Користувачу буде виведено повідомлення «Введіть нове місто». Якщо місто буде введено некоректно, то користувача буде

відправлено повідомлення «Місто введено невірно, введи ще раз», інакше – «Ви змінили місто на: місто». На рисунку 4.7 відображено процес зміни міста.

Рисунок 4.7 – Процес зміни міста

У цьому ж меню користувач може дізнатися інформацію, щодо погоди на дорозі, натиснувши кнопку «Погода на дорозі». На рисунку 4.8 продемонстровано отримання інформації, щодо погоди на дорозі.

Рисунок 4.8 – Інформація, щодо погоди на дорозі

При натисканні кнопки «Погода» користувач потрапляє у підменю погода де має змогу дізнатися поточну погоду, прогноз погоди на 5 днів та отримати рекомендації, щодо одягу.

Для того, щоб дізнатися поточну погоду користувач повинен натиснути кнопку «Поточна погода». На рисунку 4.9 продемонстровано вивід інформації, щодо поточної погоди.

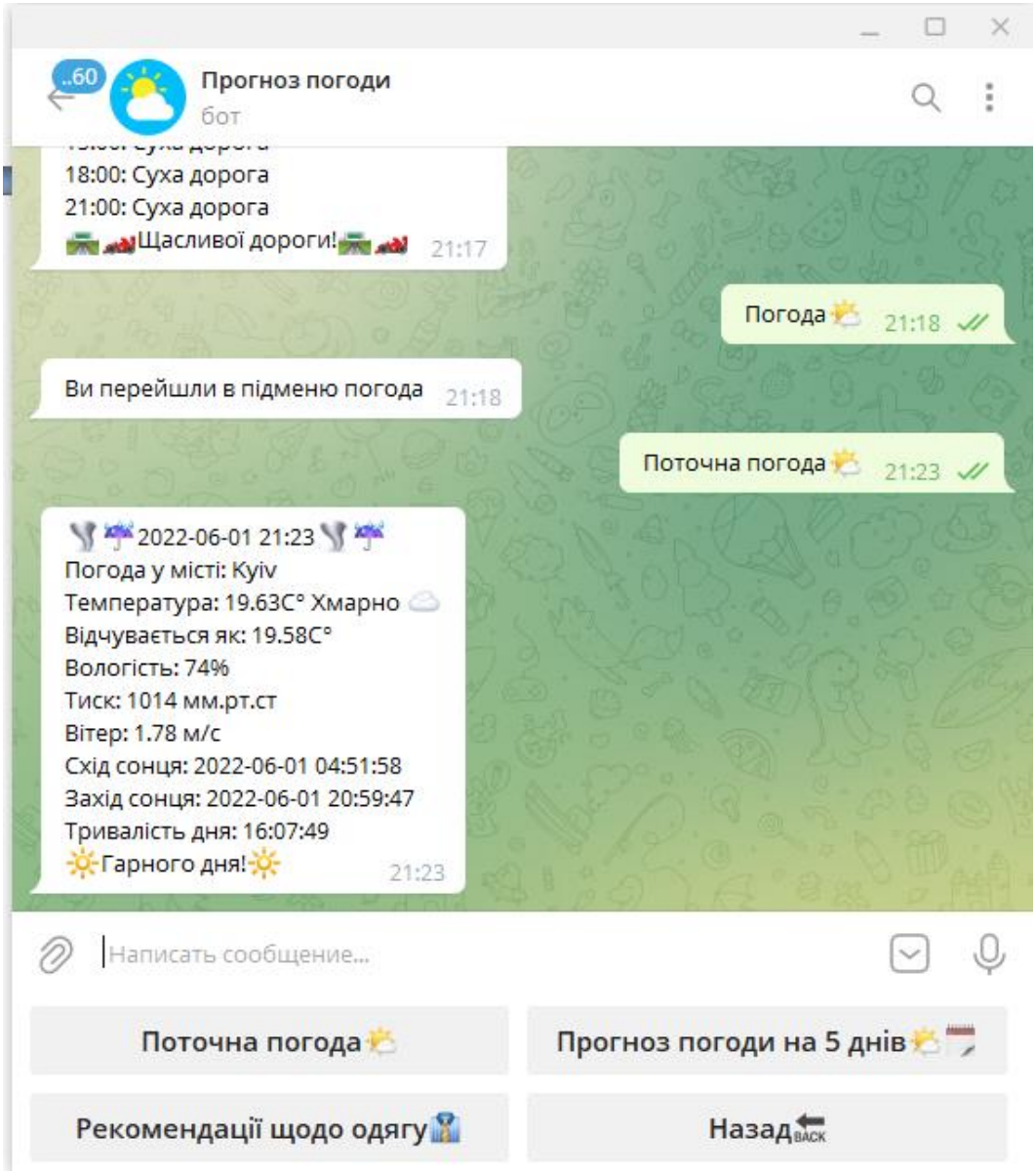

Рисунок 4.9 – Поточна погода

Щоб дізнатися прогноз погоди на п’ять днів користувачу потрібно натиснути кнопку «Прогноз погоди на 5 днів». Прогноз погоди на п’ять днів

буде виведений з інтервалом 3 години для більш точної інформації, щодо прогнозу погоди. На рисунках 4.10-4.11 відображено прогноз погоди на п’ять днів.

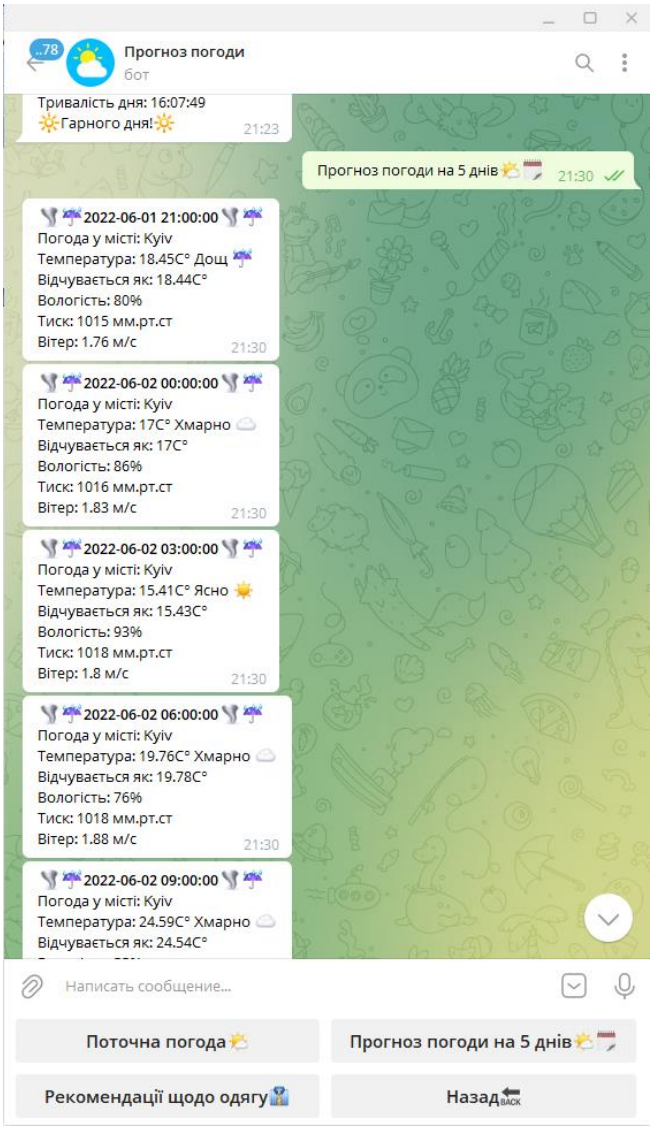

Рисунок 4.10 – Прогноз погоди на п’ять днів

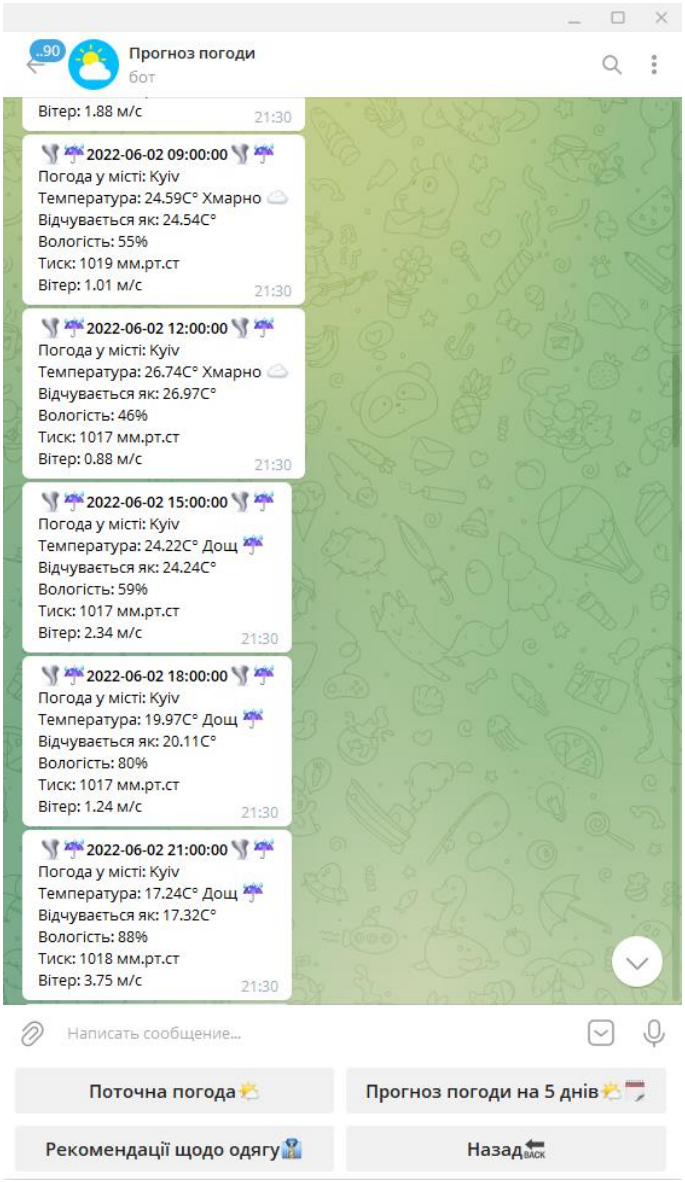

Рисунок 4.11 – Продовження

Щоб отримати рекомендації, щодо одягу потрібно натиснути кнопку «Рекомендації щодо одягу». Бот видасть фото, яке відповідає температурі та погодним умовах у заданому місті на поточний момент часу. На рисунку 4.12 продемонстровано процес видачі фото.

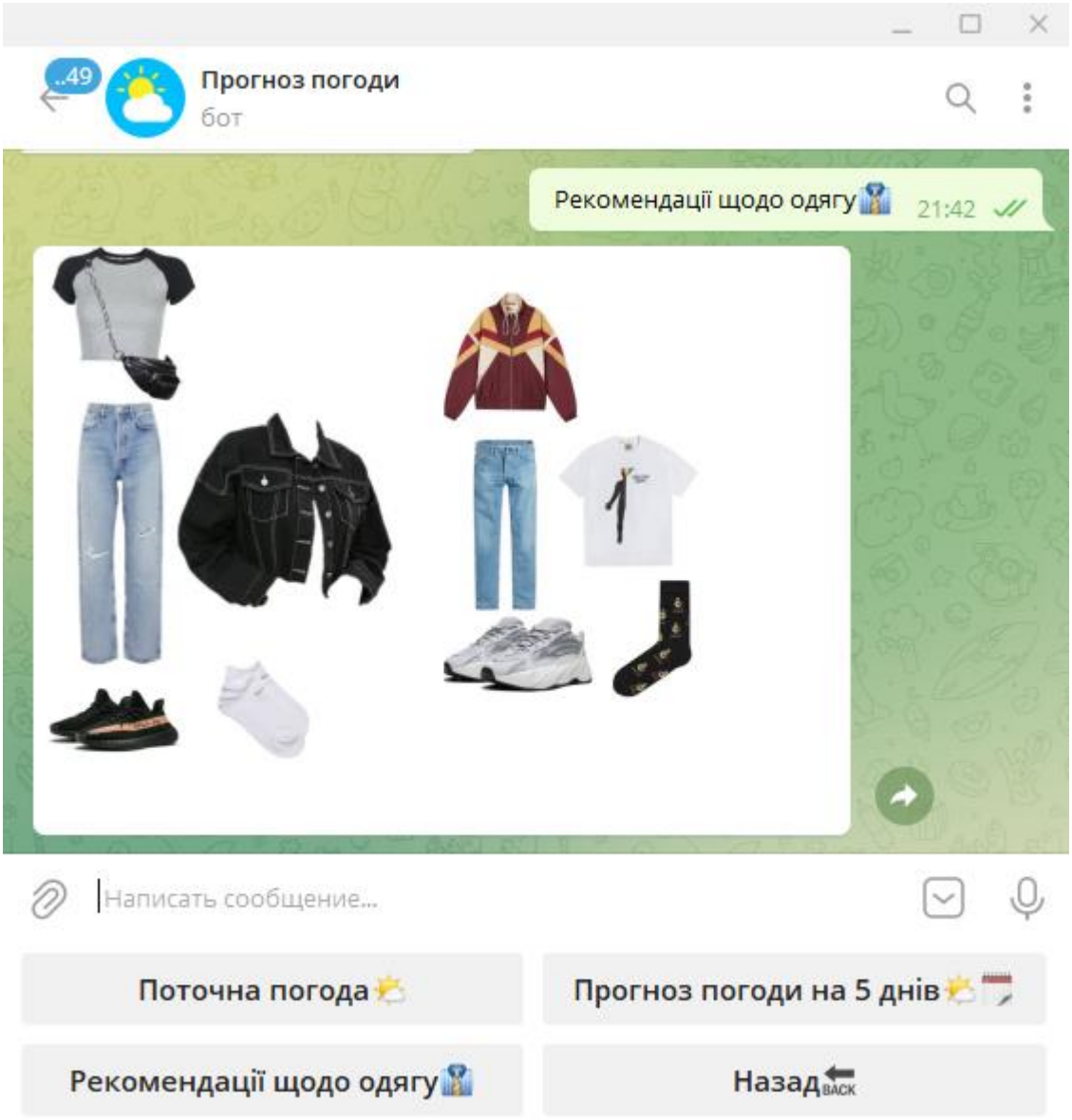

Рисунок 4.12 – Рекомендації щодо одягу

Для користувача також доступна кнопка «Назад», яка дає змогу перейти у головне меню.

У адміністратора є доступ до команди /download, яка дозволяє завантажити фото з певними параметрами до бази даних. Також є команда /cancel, яка дозволяє зупинити завантаження фото до бази даних. На рисунку 4.13 продемонстровано процес завантаження адміністратором фото з певними параметрами до бази даних.

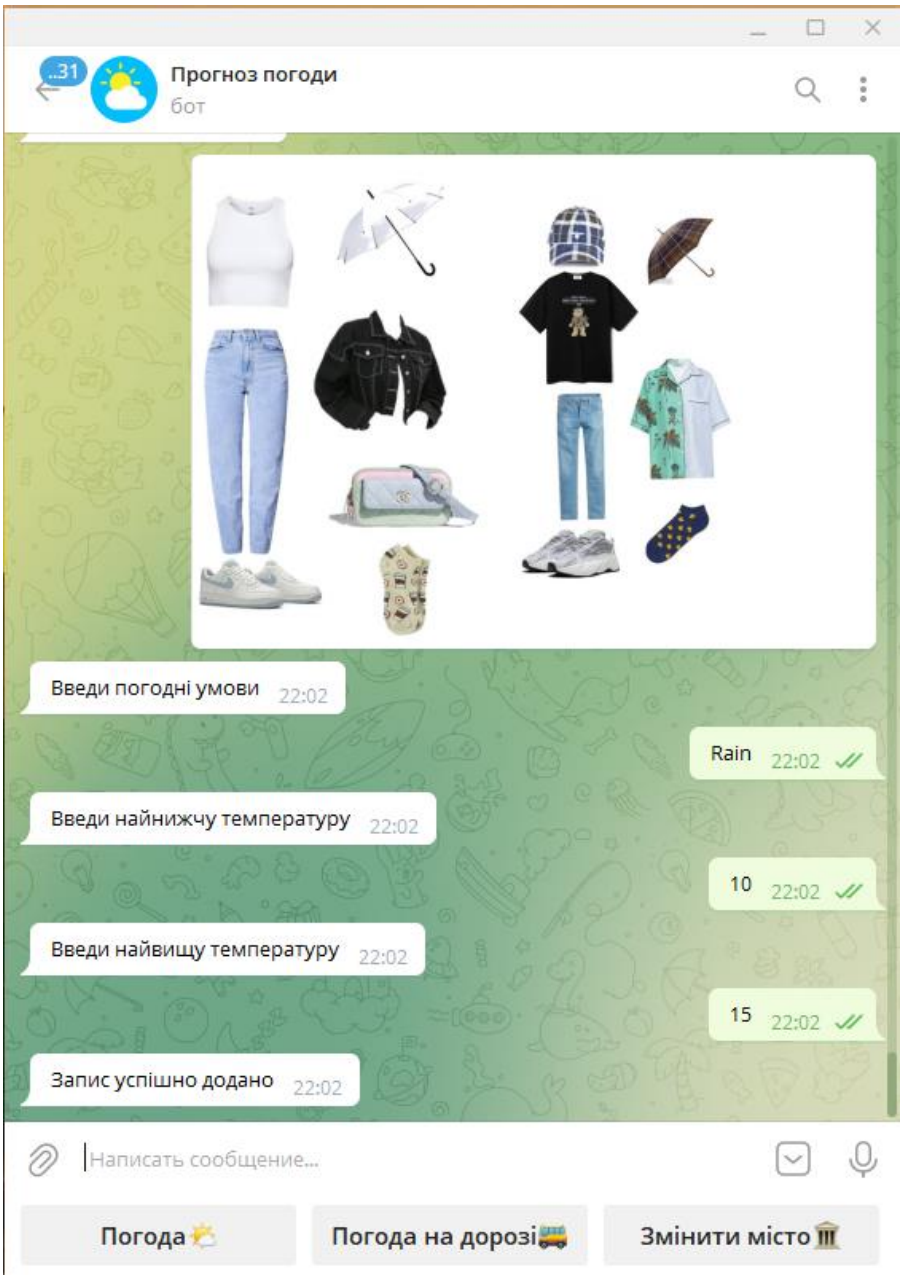

Рисунок 4.13 – Завантаження фото до бази даних

4.2 Засоби безпеки

Функціонал телеграм-боту передбачає дві сутності: адміністратор та користувач. У адміністратора є дві команди /download та /cancel. Для того, щоб користувач не мав змоги завантажувати нові фото до бази даних через команду /download у програмному коді реалізовано перевірку на адміністратора. Якщо ID користувача не є ID адміністратора, то користувач отримує повідомлення про те, що він не адміністратор і не дає змогу завантажувати нові фото до бази даних. Аналогічна перевірка є для команди /cancel.

Для безпеки токену погоди та токену взаємодії з телеграм-ботом, а також ID адміністратора усі дані були винесені в окремий модуль data_token.py.

4.3 Висновки за розділом

У цьому розділі було виконано опис розробленої програмної системи, були описані всі можливі сценарії взаємодії користувача з телеграм-ботом.

Також були описані засоби захисту, а саме перевірка на адміністратора та винесення токенів в окремий модуль.

5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

5.1 Функціональне тестування системи

Для перевірки модулів, які були створені під час розробки, необхідно провести тестування. Метод, який був обраний при тестуванні – функціональне тестування [22]. Цей метод тестування потрібен для того, щоб перевірити які функції програми реалізовані та наскільки вірно ці функції реалізовано. На рисунку 5.1 продемонстровано принцип функціонального тестування.

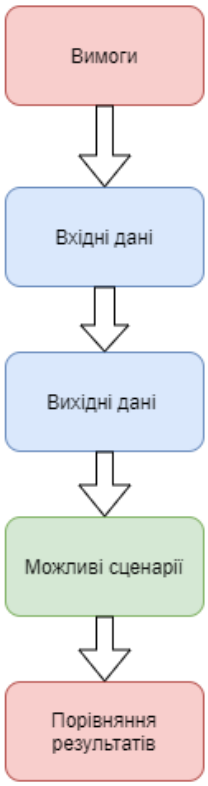

Рисунок 5.1 – Принцип функціонального тестування

Для тестування на мові python є бібліотека unittest, призначення цієї бібліотеки – написання тестів.

5.2 Тестування алгоритму парсингу даних

Дані про погоду на дорозі надаються завдяки алгоритму парсингу даних. Алгоритм передбачає, що функція, яка відповідає за парсинг даних – grw, повинна повертати значення типу list, тобто масив. Місто, яке вводить користувач повинно мати тип string, тобто рядок. Через це було написано два тести для перевірки цих умов. На рисунку 5.2 продемонстровані тести для алгоритму парсингу даних

```
test_pars.py
1   from pars import grw                                    ⚠4 ✓1
2
3   import unittest
4
5   class Test_pars(unittest.TestCase):
6
7       def setup(self):
8           pass
9       def teardown(self):
10          pass
11
12      def test_grw(self):
13          self.assertEqual(type(grw('Дніпро')), list)
14
15      def test_types(self):
16          self.assertRaises(TypeError, grw, 5 + 2j)
17          self.assertRaises(TypeError, grw, [1, 2])
18          self.assertRaises(TypeError, grw, [12])
19          self.assertRaises(TypeError, grw, True)
20
```

Рисунок 5.2 – Тести для алгоритму парсингу даних

Під час виконання тестів test_types не був складений, тому було виконано модифікацію коду для обробки виключення.

На рисунку 5.3 відображено результати складання тестів після модифікації коду.

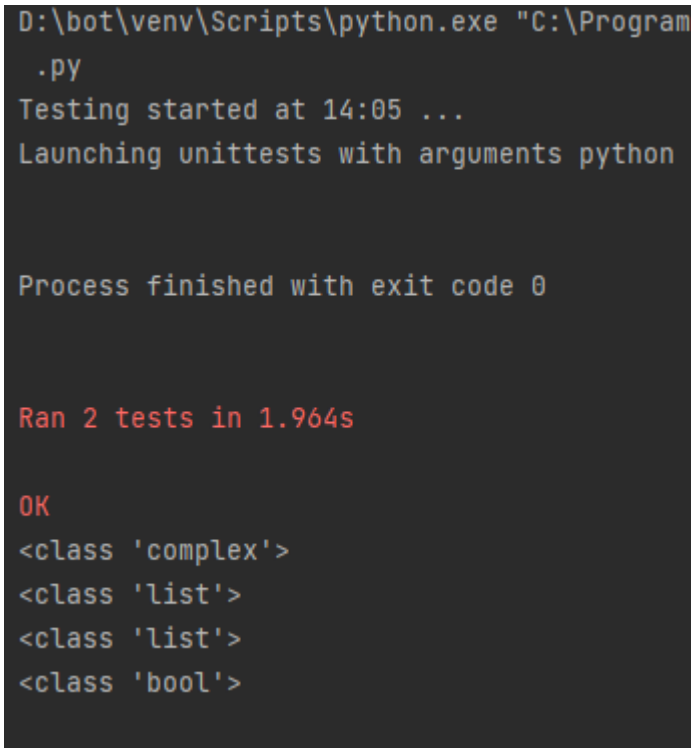

Рисунок 5.3 – Результат виконання тестів

5.3 Тестування підключення до API openweathermap

Для отримання даних про погоду потрібно відправити HTTP-запит до https://openweathermap.org/, якщо запит виконано успішно, то код запита повинен бути 200. Запит з застосування json повинен видавати тип даних dict, тобто словник. Виходячи з цього були написані два тести на перевірку цих умов (рис. 5.4).

```python
import unittest
import requests

from data_token import pog_token

class Test_api(unittest.TestCase):

    def setup(self):
        pass
    def teardown(self):
        pass

    def test_status_code(self):
        self.assertEqual(requests.get(f"http://api.openweathermap.org/data/2.5/weather?q={'Дніпро'}&appid="
                                      f"{pog_token}&units=metric&lang=ua&").status_code, 200)
    def test_type(self):
        self.assertEqual(type(requests.get(f"http://api.openweathermap.org/data/2.5/weather?q={'Дніпро'}&appid="
                                           f"{pog_token}&units=metric&lang=ua&").json()), dict)
```

Рисунок 5.4 – Тести для підключення до API openweathermap

Результати виконання тестів продемонстровані на рисунку 5.5.

```
D:\bot\venv\Scripts\python.exe "C:\Program F
Testing started at 15:00 ...
Launching unittests with arguments python -m


Process finished with exit code 0


Ran 2 tests in 0.217s

OK
```

Рисунок 5.5 – Результат виконання тестів

5.4 Висновки за розділом

У поточному розділі було описано метод тестування, а саме функціональне тестування.

Було написано два тести для алгоритму парсингу даних:

– тест на повертання функцією масиву;

– тест на тип даних міста.

Також було написано два тести для підключення до API openweather:

– тест успішного HTTP-запиту;

– тест на тип даних dict.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проведено аналіз вибраної теми, було визначено об’єкт і предмет дослідження, наведена мета робота, методи дослідження та їх обґрунтування. Був проведений аналіз конкурентних продуктів. Для розробки телеграм-боту були вибрані інструменти, засоби і технології, а саме:

– технологія API;

– технологія парсингу даних, яка була реалізована за допомогою Document Object Model;

– теорія кінцевих автоматів;

– метод програмування - об’єктно-орієнтоване програмування;

– теорія модульного програмування;

– графічна мова UML для побудови UML-діаграм.

Були визначені основні завдання роботи.

Під час проєктування системи було описано архітектуру системи, побудовано UML-діаграми:

– використання;

– діяльності;

– розміщення.

Також була обрана мова Python програмування та середовище розробки PyCharm. Для роботи з базами даних було обрано графічний додаток BD Browser. Для проєктування інтерфейсу було обрано веб-ресурс mockflow.

У ході розробки телеграм-боту було виконано реєстрацію чат-боту, налаштування його персоналізації, було описано усі модулі програми та функції.

Під час тестування програмної системи було використано функціональне тестування, було виконано два тести алгоритму парсингу даних та тести підключення до API веб-ресурсу openweather

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram [Електронний ресурс] – Режим доступу: https://web.telegram.org/z/

2. How many people use Telegram in 2022 [електронний ресурс] – Режим доступу: https://backlinko.com/telegram-users

3. Python [Електронний ресурс] – Режим доступу: https://www.python.org/

4. Эл Свейгарт «Автоматизация рутинных задач с помощью Python», 2017 – 573 с.

5. Свaруup Чилтур «Укус Питона» /пер. В. Смоляр, 2020 – 168 с.

6. SQLite3 [Електронний ресурс] – Режим доступу: https://www.sqlite.org/index.html;

7. Ф.О Чмиленко, Л.П. Жук Посібник до вивчення дисципліни «Методологія та організація наукових досліджень» Дніпропетровськ РВВ ДНУ», 2014 – 48 с.

8. API [Електронний ресурс] – Режим доступу: https://openweathermap.org/current

9. HTTP-запит [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods

10. Парсинг даних [Електронний ресурс] – Режим доступу: https://understandingdata.com/what-is-web-scraping/

11. Кінцевий автомат. Адаматський А. «Theory and Applications of Cellular Automata», 2008 – 636 с.

12. База даних [Електронний ресурс] – Режим доступу: https://www.oracle.com/cis/database/what-is-database/

13. Крістіан Хилл, пер. А. В. Снастина «Научное программирование на Python» 2021 – 646 с.

14. Об’єктно-орієнтоване програмування [Електронний ресурс] – Режим доступу: https://python-scripts.com/object-oriented-programming-in-python

15. UML-діаграми. Мартін Фаулер, Кендалл Скотт, пер. А. Петухова «UML. Основы. 3-е издание» 2018 – 192 с.

16. Document Object Model [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/Document_Object_Model/Introduction;

17. Aiogram [Електронний ресурс] – Режим доступу: https://docs.aiogram.dev/en/latest/

18. PyCharm [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/pycharm/

19. DB Browser [Електронний ресурс] – Режим доступу: https://sqlitebrowser.org/

20. Вайфрейм [Електронний ресурс] – Режим доступу: https://balsamiq.com/learn/articles/what-are-wireframes/

21. MockFlow [Електронний ресурс] – Режим доступу: https://www.mockflow.com/

22. Кей С. Хорстманн «Scala для нетерпеливых», 2012. – 416с.

ДОДАТОК А
ЗАУВАЖЕННЯ НОРМКОНТРОЛЕРА

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
ПЗ		Нещільне заповнення сторінок пояснювальної записки
Додаток В		Оформлення додатку невірне
С. 64-65		Помилки в оформленні списку використаних джерел

ДОДАТОК Б
ГРАФІЧНА ЧАСТИНА

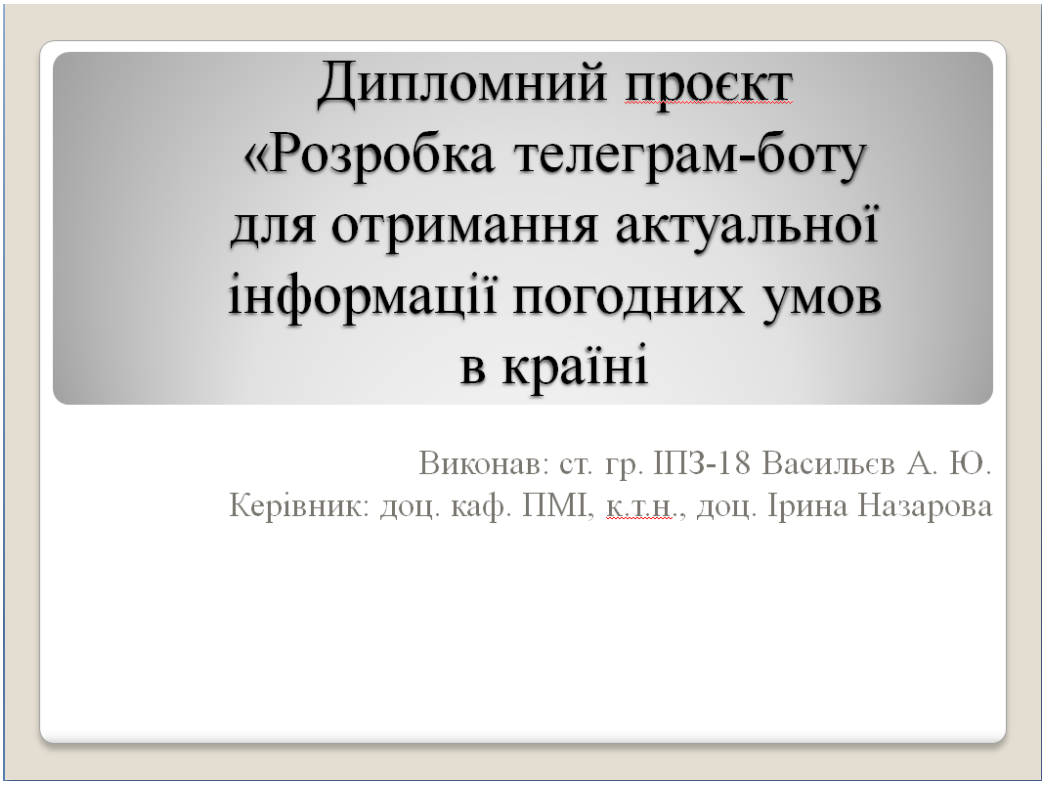

Рисунок Б.1 – Титульний сла йд

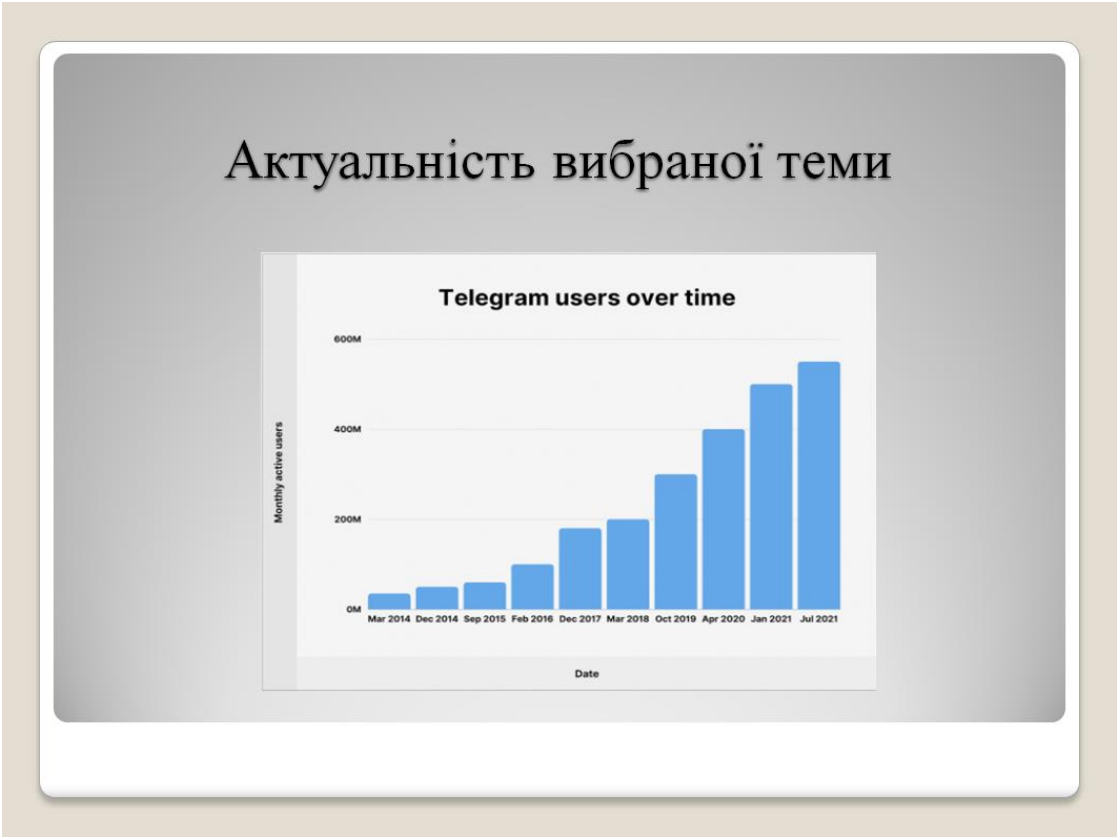

Рисунок Б.2 – Актуальність вибраної теми

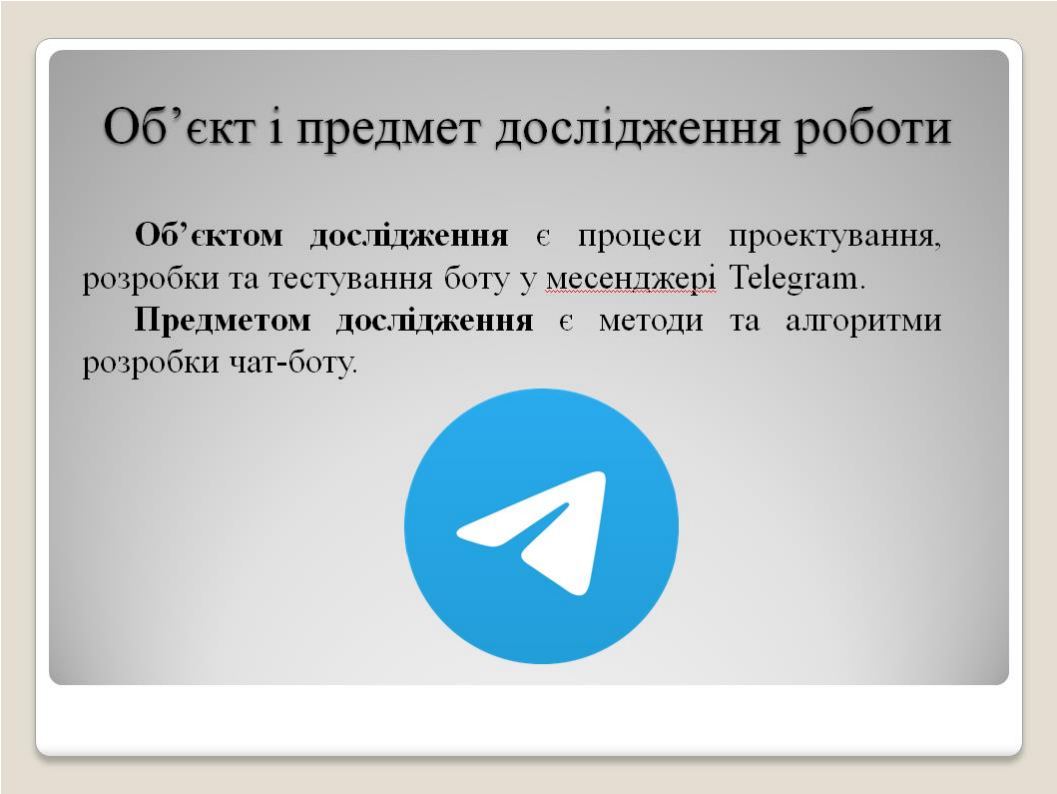

Рисунок Б.3 – Об’єкт і предмет дослідження роботи

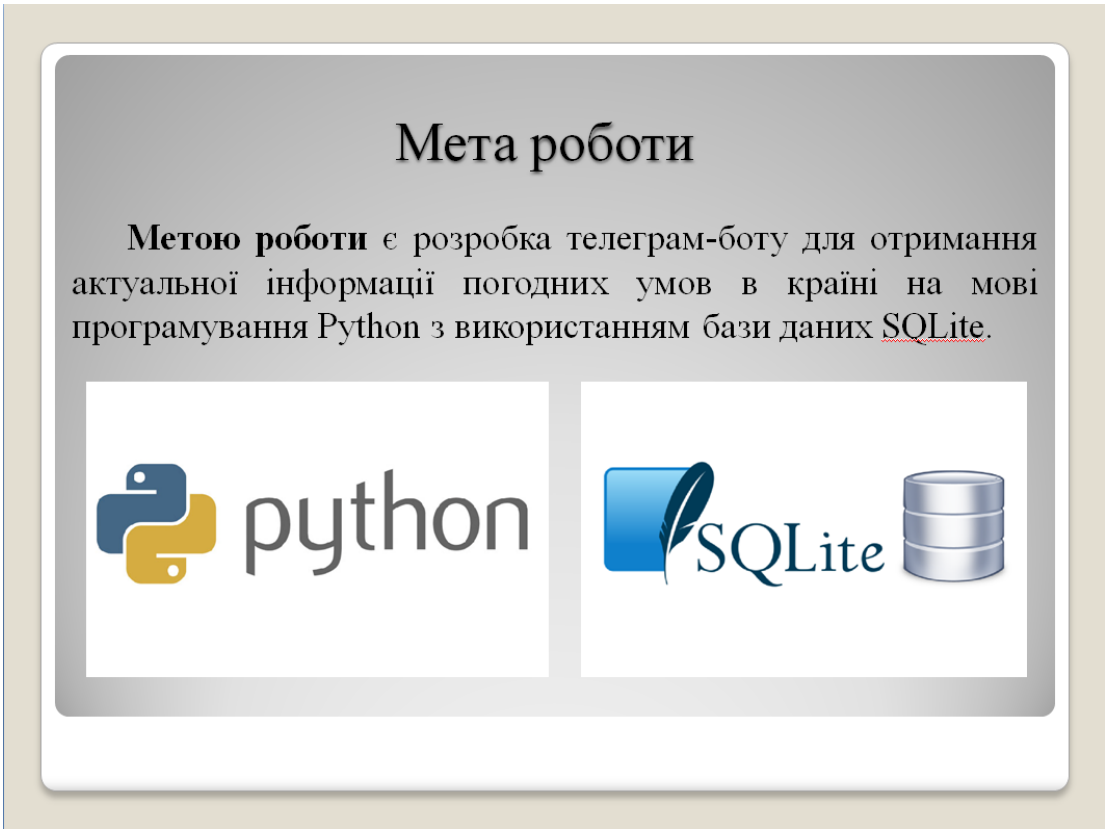

Рисунок Б.4 – Мета роботи

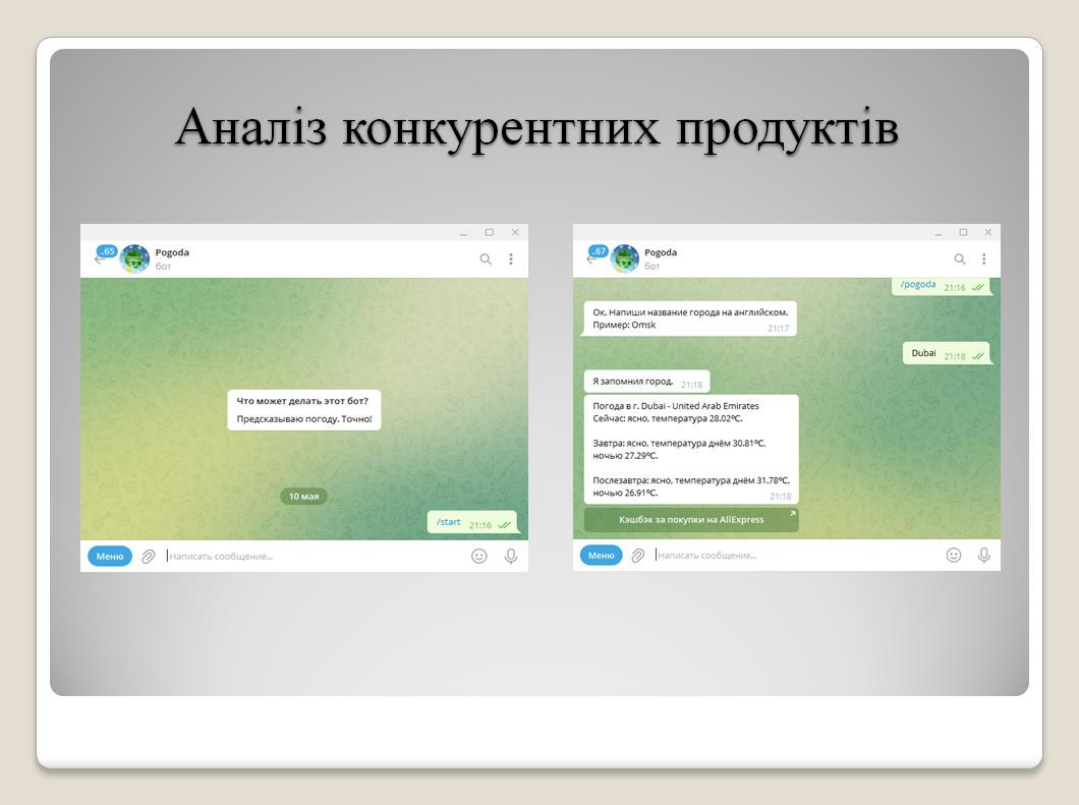

Рисунок Б.5 – Аналіз конкурентних продуктів

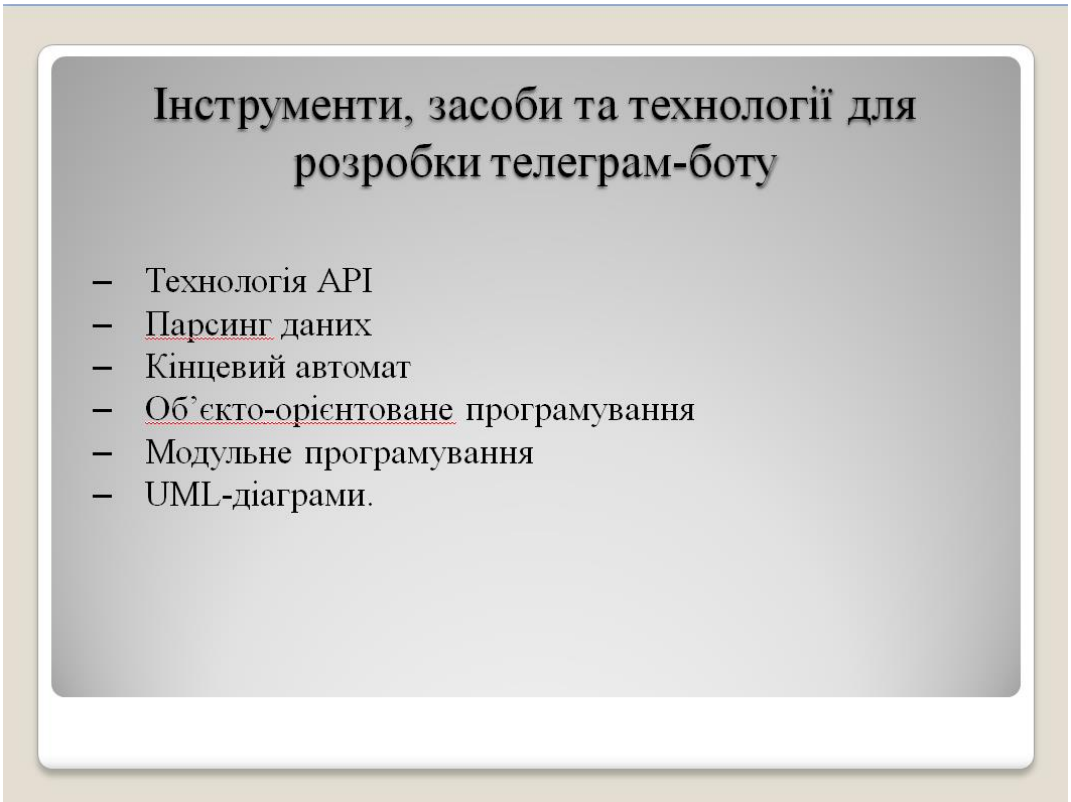

Рисунок Б.6 – Інструменти, засоби та технології для розробки телеграм-бот

Рисунок Б.7 – Діаграми використання

Рисунок Б.8 – Діаграми діяльності та розміщення

Рисунок Б.9 – Алгоритм парсингу даних

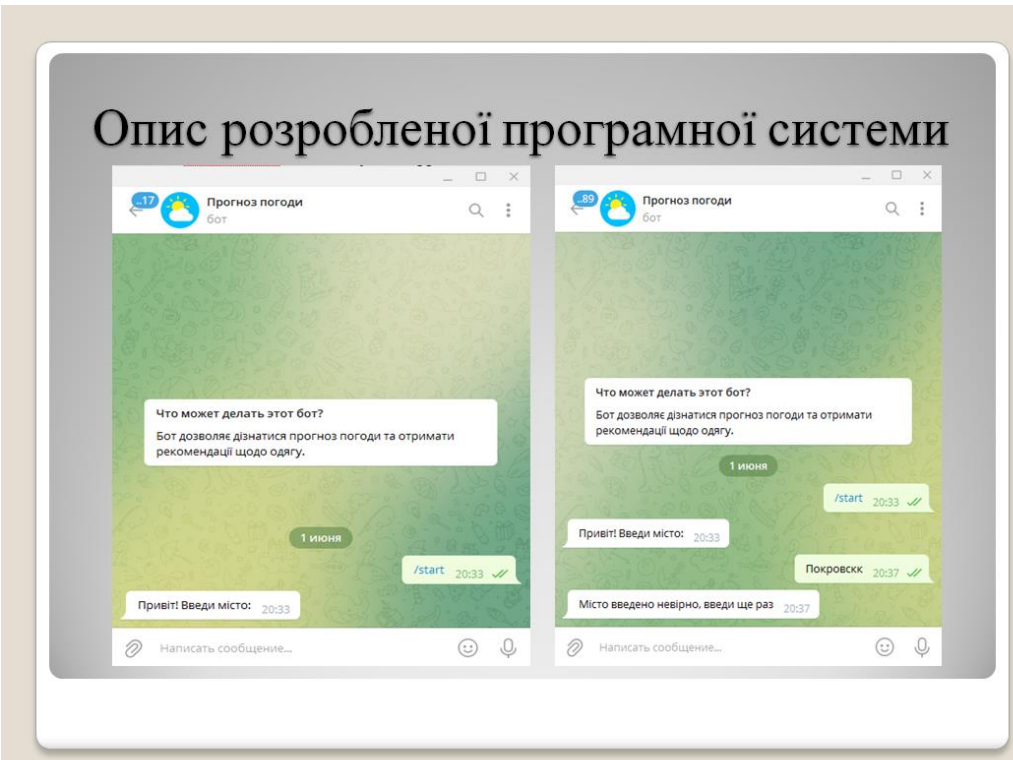

Рисунок Б.10 – Опис розробленої програмної системи

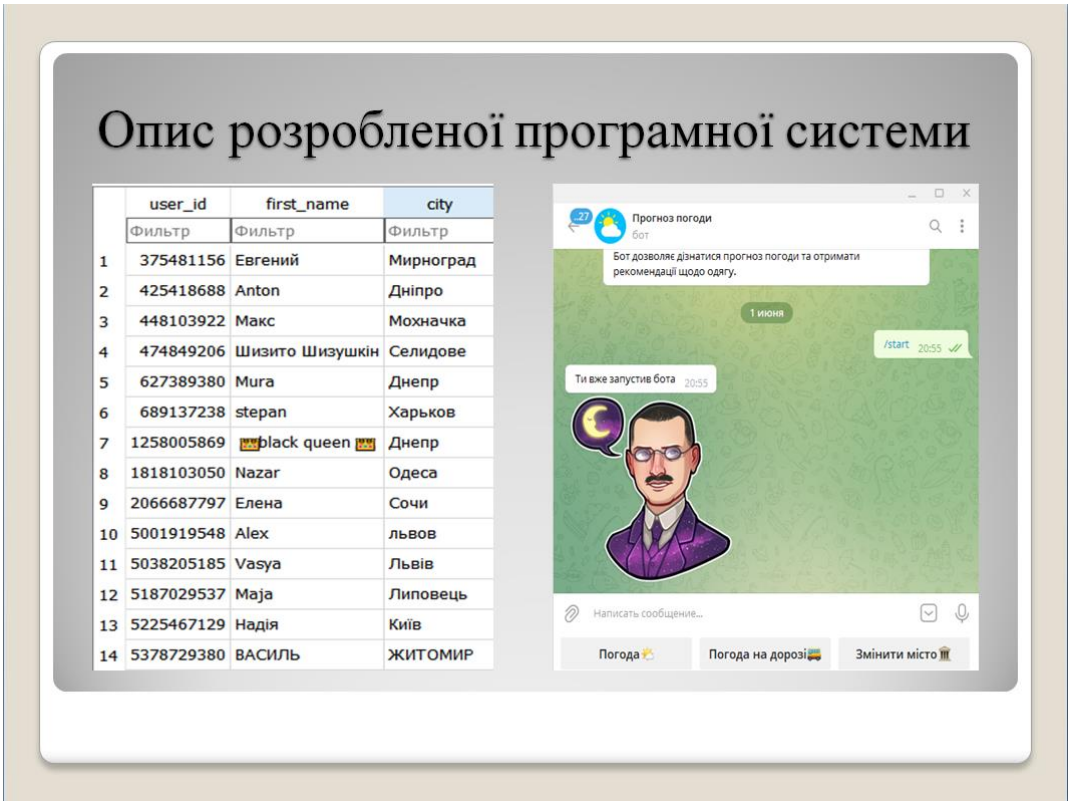

Рисунок Б.11 – Опис розробленої програмної системи

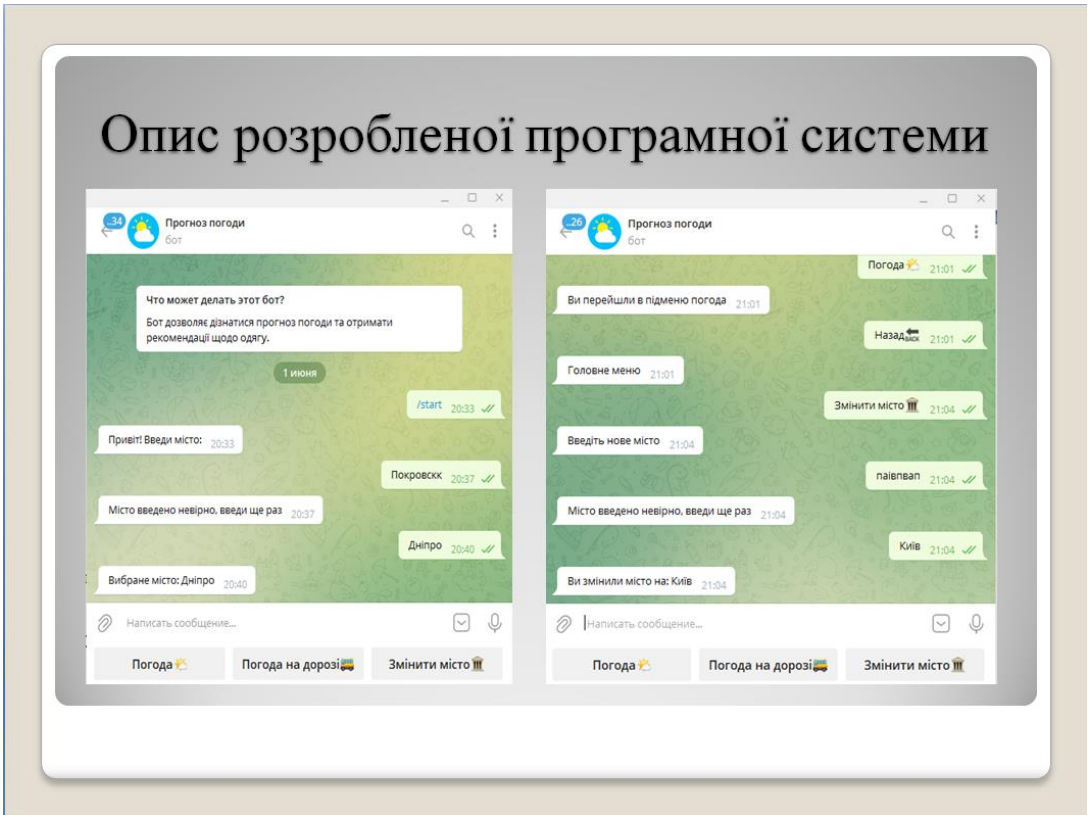

Рисунок Б.12 – Опис розробленої програмної системи

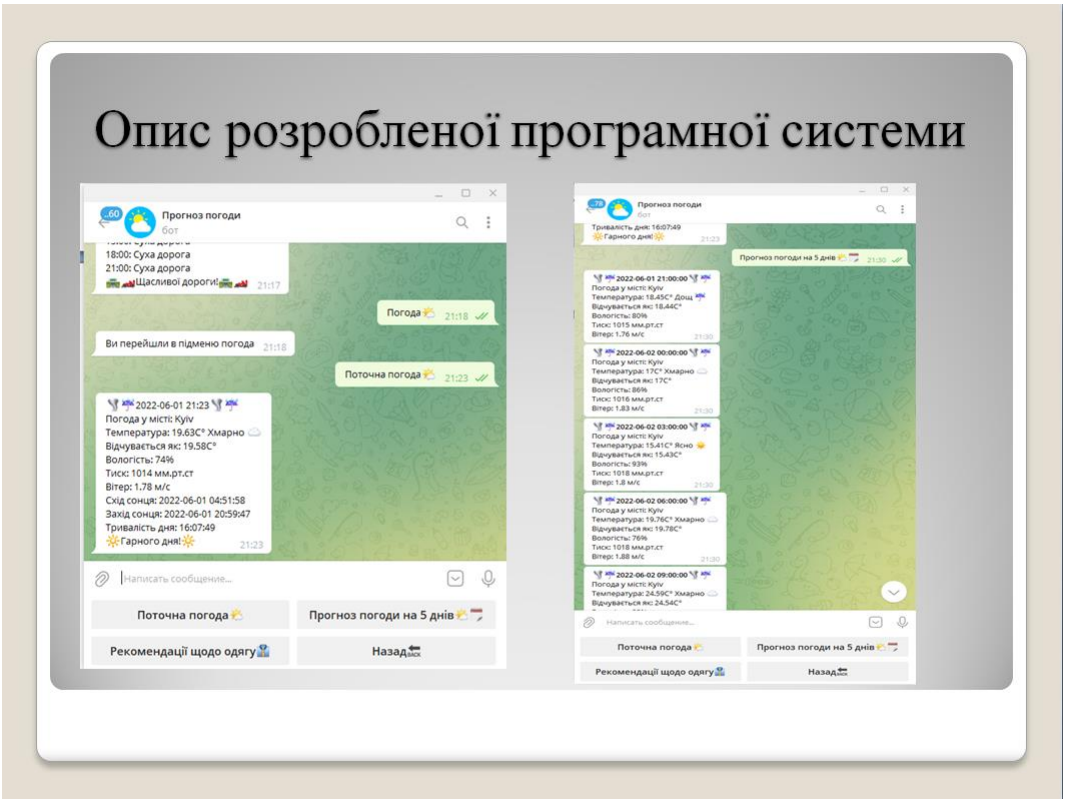

Рисунок Б.13 – Опис розробленої програмної системи

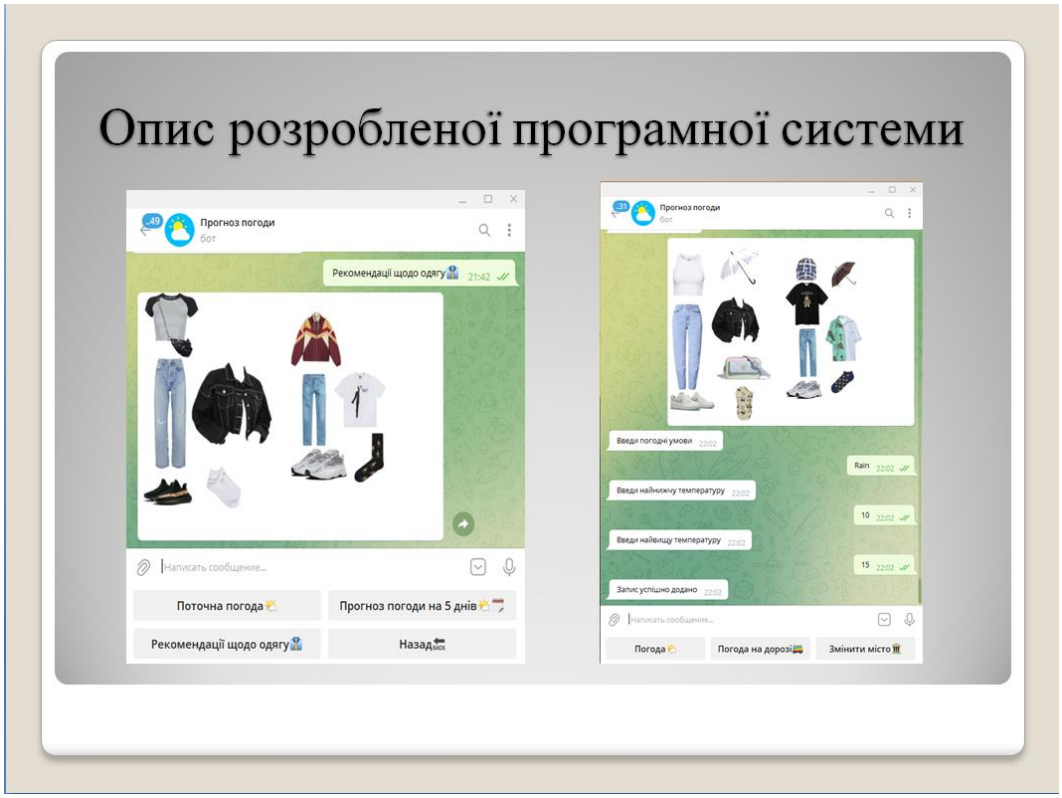

Рисунок Б.14 – Опис розробленої програмної системи

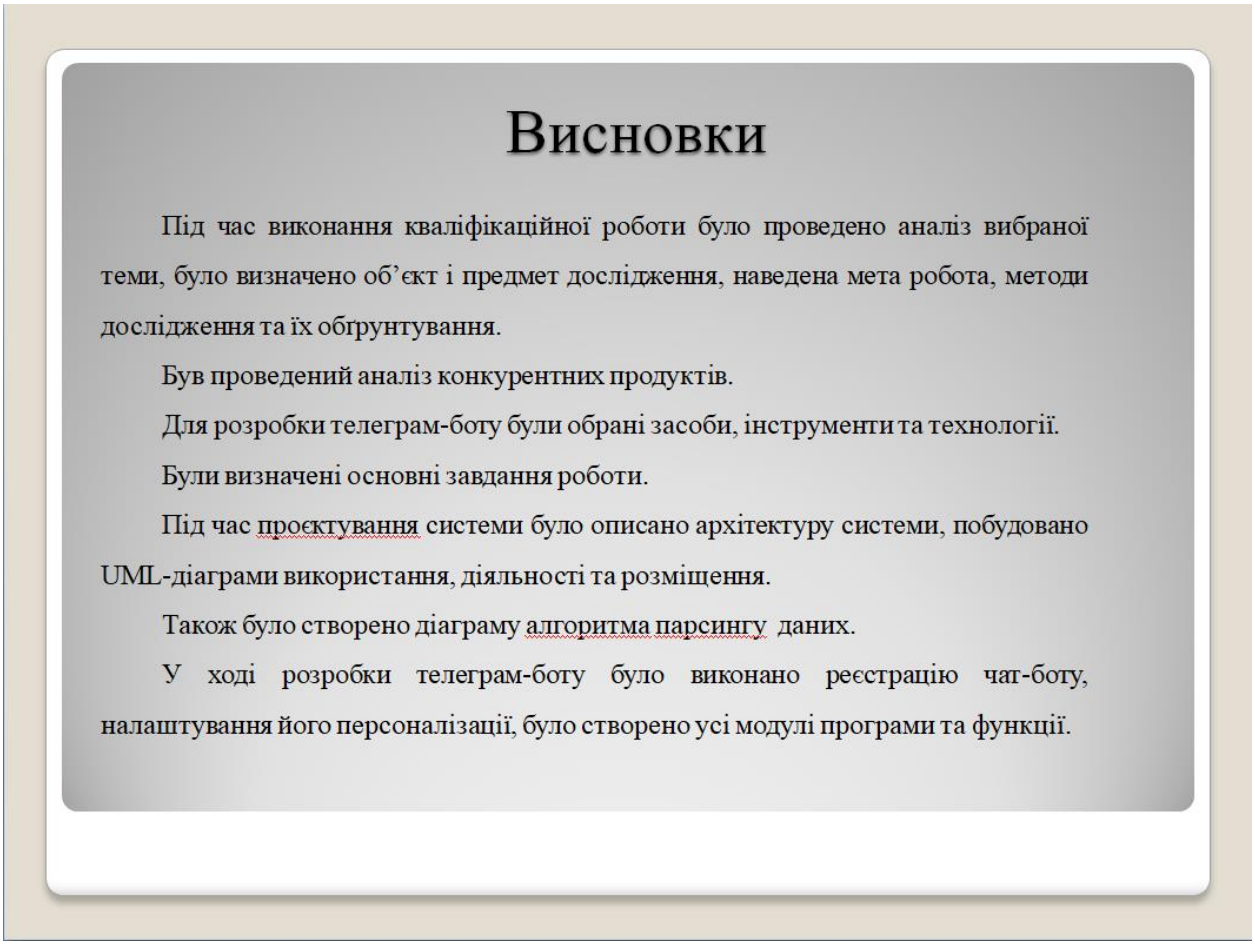

Рисунок Б.14 – Висновки

Додаток В.1

Лістинг модуля створення екземпляру бота

```
from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram.utils import executor
import database
from data_token import TOKEN
from aiogram.contrib.fsm_storage.memory import MemoryStorage
import handler

storage = MemoryStorage()
bot = Bot(token=TOKEN)
dis = Dispatcher(bot, storage=storage)


async def on_startup(_):
    print('Bot is working')
    database.sql_start()


handler.register_handler_handlers(dis)

if __name__ == '__main__':
    executor.start_polling(dis, skip_updates=True, on_startup=on_startup)
```

Додаток В.2

Лістинг модуля кнопок та клавіатури

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

back = KeyboardButton('Назад🔙')

b1 = KeyboardButton('Погода🌤')
b2 = KeyboardButton('Погода на дорозі🚌')
b3 = KeyboardButton('Змінити місто🏛')
kb_client = ReplyKeyboardMarkup(resize_keyboard=True)
kb_client.add(b1, b2, b3)

clothes = KeyboardButton('Рекомендації щодо одягу🎒')
cur_weather = KeyboardButton('Поточна погода🌤')
five_weather = KeyboardButton('Прогноз погоди на 5 днів🌤📆')
kb_weather = ReplyKeyboardMarkup(resize_keyboard=True, row_width=2)
kb_weather.add(cur_weather, five_weather, clothes, back)
```

Додаток В.3

Лістинг модуля токенів

TOKEN = "5330158922:AAGfJTnQ_Oy-8AAjJkQvqjgVvkO8bOZH2EQ"

pog_token = "777bf850deae51ec5120df6c3f6417d6"

ID = 425418688

Додаток В.4

Лістинг модуля взаємодії з базою даних

```
import sqlite3 as sq


def sql_start():
    global base, cur
    base = sq.connect('weather_bot.db')
    cur = base.cursor()
    if base:
        print('Database connected.')
    base.execute('CREATE TABLE IF NOT EXISTS users('
                 'user_id integer PRIMARY KEY,'
                 'first_name text,'
                 'city text,'
                 'UNIQUE(user_id)'
                 ');')
    base.commit()
    base.execute('CREATE TABLE IF NOT EXISTS recommend('
                 'photo text,'
                 'weather_des text,'
                 'temp_low real,'
                 'temp_high real,'
                 'UNiQUE(photo)'
                 ');')
    base.commit()


async def add_photo(state):
    async with state.proxy() as data:
        cur.execute('INSERT INTO recommend VALUES (?, ?, ?, ?)',
tuple(data.values()))
```

```
        base.commit()


def sql_add_user(user_id, first_name, city):
    try:
        cur.execute('INSERT INTO users (user_id,first_name,city) VALUES(?,?,?);',
                    (user_id, first_name, city))
        base.commit()
    except sq.IntegrityError as nu:
        print(nu)


def sql_check_user(user_id):
    return cur.execute('SELECT user_id from users WHERE user_id==?',
(user_id,)).fetchone()


def sql_update_city(city, user_id):
    cur.execute('UPDATE users SET city ==? WHERE user_id==?', (city, user_id))
    base.commit()


def sql_get_city(user_id):
    a = cur.execute('SELECT city FROM users WHERE user_id==?',
(user_id,)).fetchone()
    base.commit()
    return a


def id_photo(wd, t):
```

```
    cur.execute('CREATE TEMP TABLE IF NOT EXISTS Variable(wd text,t
real);')
    base.commit()
    cur.execute('INSERT INTO Variable VALUES(?,?)', (wd, t,))
    base.commit()
    a = cur.execute(
        'SELECT photo,wd,t FROM recommend,Variable WHERE
weather_des==wd AND t BETWEEN temp_low AND temp_high').fetchone()
    base.commit()
    cur.execute('DROP TABLE Variable')
    base.commit()
    return a
```

Додаток В.5

Лістинг модуля для обробки команд, кнопок, повідомлень

```
from aiogram import Dispatcher, types
from aiogram.dispatcher.filters.state import StatesGroup, State
from data_token import ID
import database
from data_token import pog_token
import requests
import datetime
from buttons import *
import pars
from aiogram.dispatcher import FSMContext


class FSMAdmin(StatesGroup):
    set_city = State()
    change_city = State()
    photo = State()
    weather_des = State()
    temp_low = State()
    temp_high = State()


async def start_bot(message: types.Message):
    if database.sql_check_user(message.from_user.id):
        await message.answer('Ти вже запустив бота', reply_markup=kb_client)
        await
message.answer_sticker(r'CAACAgIAAxkBAAEE0upij1YeZPOo3toKNTfDY0b
K3L4XdgAC9ggAAgi3GQLFcJickVO4KyQE')
    else:
        await message.answer("Привіт! Введи місто:")
        await FSMAdmin.set_city.set()
```

```
async def admin_start(message: types.Message):
    if message.from_user.id == ID:
        await FSMAdmin.photo.set()
        await message.answer('Завантаж фото')
    else:
        await message.answer('Вибач, але ти не адмін')


async def cancel(message: types.Message, state: FSMContext):
    if message.from_user.id == ID:
        now_state = await state.get_state()
        if now_state is None:
            return
        await state.finish()
        await message.answer('ОК')
    else:
        await message.answer('Вибач, але ти не адмін')


async def load_photo(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['photo'] = message.photo[0].file_id
    await FSMAdmin.next()
    await message.answer('Введи погодні умови')


async def load_des(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
```

```
        data['weather_des'] = message.text
    await FSMAdmin.next()
    await message.answer('Введи найнижчу температуру')


async def load_low_temp(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['temp_low'] = float(message.text)
    await FSMAdmin.next()
    await message.answer('Введи найвищу температуру')


async def load_high_temp(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['temp_high'] = float(message.text)

    await database.add_photo(state)
    await message.answer('Запис успішно додано')
    await state.finish()


async def set_city(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['city'] = message.text
        r = requests.get(

f"http://api.openweathermap.org/data/2.5/weather?q={message.text}&appid={pog
_token}&units=metric&lang=ua&")
        if r.status_code == 404:
            await message.answer('Місто введено невірно, введи ще раз')
```

```
        else:
            await message.answer('Вибране місто: ' +
data['city'],reply_markup=kb_client)
            database.sql_add_user(message.from_user.id,
message.from_user.first_name, data['city'])
            await state.finish()


async def change_city(message: types.Message):
    await message.answer('Введіть нове місто')
    await FSMAdmin.change_city.set()


async def new_city(message: types.Message, state: FSMContext):
    async with state.proxy() as data:
        data['city'] = message.text
        r = requests.get(

f"http://api.openweathermap.org/data/2.5/weather?q={message.text}&appid={pog
_token}&units=metric&lang=ua&")
        if r.status_code == 404:
            await message.answer('Місто введено невірно, введи ще раз')
        else:
            await message.answer('Ви змінили місто на: ' + data['city'])
            database.sql_update_city(data['city'], message.from_user.id)
            await state.finish()


async def road_weather(message: types.Message):
    try:
```

```
        city = database.sql_get_city(message.from_user.id)[0]
        rc = pars.grw(city)
        await message.answer((f"📅{datetime.datetime.now().strftime('%Y-%m-
%d')}📅\n"
                              f"Місто: {city}🏞\n"
                              f"0:00: {rc[0]}\n"
                              f"3:00: {rc[1]}\n"
                              f"6:00: {rc[2]}\n"
                              f"9:00: {rc[3]}\n"
                              f"12:00: {rc[4]}\n"
                              f"15:00: {rc[5]}\n"
                              f"18:00: {rc[6]}\n"
                              f"21:00: {rc[7]}\n"
                              f"📨🚙Щасливої дороги!📨🚙"
                              ))
    except:
        await message.answer('Функція тимчасово недоступна, спробуйте
пізніше⏰')


async def weather(message: types.Message):
    await message.answer('Ви перейшли в підменю погода',
reply_markup=kb_weather)


async def rec_clothes(message: types.Message):
    try:
        r = requests.get(
```

```
f"http://api.openweathermap.org/data/2.5/weather?q={database.sql_get_city(messa
ge.from_user.id)[0]}&appid={pog_token}&units=metric&lang=ua"
        )
        data = r.json()
        cur_weather = data["main"]["temp"]
        weather_description = data["weather"][0]["main"]
        id_photo = database.id_photo(weather_description, cur_weather)
        await message.answer_photo(photo=id_photo[0])
    except:
        await message.answer('Функція тимчасово недоступна, спробуйте
пізніше⏰')


async def cur_weather(message: types.Message):
    code_to_smile = {
        "Clear": "Ясно \U00002600",
        "Clouds": "Хмарно \U00002601",
        "Rain": "Дощ \U00002614",
        "Drizzle": "Сильний дощ \U00002614",
        "Thunderstorm": "Гроза \U000026A1",
        "Snow": "Сніг \U0001F328",
        "Mist": "Туман \U0001F32B"
    }
    try:
        r = requests.get(

f"http://api.openweathermap.org/data/2.5/weather?q={database.sql_get_city(messa
ge.from_user.id)[0]}&appid={pog_token}&units=metric&lang=ua&"
```

```
        )
        data = r.json()
        city = data["name"]
        cur_weather = data["main"]["temp"]

        weather_description = data["weather"][0]["main"]
        if weather_description in code_to_smile:
            wd = code_to_smile[weather_description]
        else:
            wd = "Подивись у вікно!"

        feels_like = data["main"]["feels_like"]
        humidity = data["main"]["humidity"]
        pressure = data["main"]["pressure"]
        wind = data["wind"]["speed"]
        sunrise_timestamp =
datetime.datetime.fromtimestamp(data["sys"]["sunrise"])
        sunset_timestamp = datetime.datetime.fromtimestamp(data["sys"]["sunset"])
        length_of_the_day = datetime.datetime.fromtimestamp(data["sys"]["sunset"])
- datetime.datetime.fromtimestamp(
            data["sys"]["sunrise"])

        await message.answer(f"🌪🍄{datetime.datetime.now().strftime('%Y-%m-
%d %H:%M')}🌪🍄\n"
                             f"Погода у місті: {city}\nТемпература: {cur_weather}C°
{wd}\n"
                             f"Відчувається як: {feels_like}C°\n"
                             f"Вологість: {humidity}%\nТиск: {pressure} мм.рт.ст\nВітер:
{wind} м/с\n"
```

```
                    f"Схід сонця: {sunrise_timestamp}\nЗахід сонця:
{sunset_timestamp}\nТривалість дня: {length_of_the_day}\n"
                    f"\U0001F506Гарного дня!\U0001F506"
                    )
    except:
        await message.answer('Функція тимчасово недоступна, спробуйте
пізніже⏰')


async def cur5_weather(message: types.Message):
    code_to_smile = {
        "Clear": "Ясно \U00002600",
        "Clouds": "Хмарно \U00002601",
        "Rain": "Дощ \U00002614",
        "Drizzle": "Сильний дощ \U00002614",
        "Thunderstorm": "Гроза \U000026A1",
        "Snow": "Сніг \U0001F328",
        "Mist": "Туман \U0001F32B"
    }
    try:
        r = requests.get(

f"http://api.openweathermap.org/data/2.5/forecast?q={database.sql_get_city(messa
ge.from_user.id)[0]}&appid={pog_token}&units=metric&lang=ua&"
        )
        data = r.json()

        city = data["city"]['name']
        cur_weather = data["list"]
```

```
        str = ""
        for i in cur_weather:
            weather_description = i["weather"][0]["main"]
            if weather_description in code_to_smile:
                wd = code_to_smile[weather_description]
            else:
                wd = "Подивись у вікно!"
            str = f"🌪☔<b>{i['dt_txt']}</b>🌪☔\n" \
                  f"Погода у місті: {city}\nТемпература: {i['main']['temp']}C° {wd}\n"
f"Відчувається як: {i['main']['feels_like']}C°\n" \
                  f"Вологість: {i['main']['humidity']}%\nТиск: {i['main']['pressure']}
мм.рт.ст\nВітер: {i['wind']['speed']} м/с\n"
            await message.answer(str, parse_mode='html')
    except Exception as ex:
        print(ex)
        await message.answer('Функція тимчасово недоступна, спробуйте
пізніше⏰')


async def back(message: types.Message):
    await message.answer('Головне меню', reply_markup=kb_client)


async def get_weather(message: types.Message):
    await message.delete()
    await message.answer("Бот не приймає повідомлення⚠      ")
```

```
def register_handler_handlers(dis: Dispatcher):
    dis.register_message_handler(start_bot, commands=['start'], state=None)
    dis.register_message_handler(admin_start, commands='download', state=None)
    dis.register_message_handler(cancel, state="*", commands='cancel')
    dis.register_message_handler(load_photo, content_types=['photo'],
state=FSMAdmin.photo)
    dis.register_message_handler(load_des, state=FSMAdmin.weather_des)
    dis.register_message_handler(load_low_temp, state=FSMAdmin.temp_low)
    dis.register_message_handler(load_high_temp, state=FSMAdmin.temp_high)
    dis.register_message_handler(set_city, state=FSMAdmin.set_city)
    dis.register_message_handler(change_city, text='Змінити місто🏛',
state=None)
    dis.register_message_handler(new_city, state=FSMAdmin.change_city)
    dis.register_message_handler(road_weather, text='Погода на дорозі🚌')
    dis.register_message_handler(weather, text='Погода🌤')
    dis.register_message_handler(rec_clothes, text='Рекомендації щодо одягу🧥')
    dis.register_message_handler(cur_weather, text='Поточна погода🌤')
    dis.register_message_handler(cur5_weather, text='Прогноз погоди на 5
днів🌤📆')
    dis.register_message_handler(back, text='Назад🔙')
    dis.register_message_handler(get_weather)
```

ДОДАТОК В.
ЛІСТИНГ МОДУЛЯ ПАРСИНГУ ДАНИХ

```
from bs4 import BeautifulSoup as b
import requests


def grw(city):
    if type(city) != str:
        raise TypeError
    else:
        URL = 'https://www.gismeteo.ua/ua/search/' + city + '/'
        headers = {
            'Connection': 'keep-alive',
            'Cache-Control': 'max-age=0',
            'Upgrade-Insecure-Requests': '1',
            'User-Agent': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/53.0.2785.116 Safari/537.36 OPR/40.0.2308.81',
            'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
            'DNT': '1',
            'Accept-Encoding': 'gzip, deflate, lzma, sdch',
            'Accept-Language': 'ru-RU,ru;q=0.8,en-US;q=0.6,en;q=0.4'
        }
        r = requests.get(URL, headers=headers)
        HTML = b(r.content, 'html.parser')
        link = HTML.select_one('div.catalog-item-link a').get('href')
        fulllink = 'https://www.gismeteo.ua/ua' + link
        re = requests.get(fulllink, headers=headers)
        html2 = b(re.content, 'html.parser')
        container = html2.select_one('div.widget-row.widget-row-roadcondition')
        rc = container.select('div.row-item')
        roadc = []
```

```
for con in rc:
    roadc.append(con.select_one('div.item-description').text)
return roadc
```