МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система визначення прогнозу погоди для заданої місцевості»

здобувача групи ІН - 03 Лебедки Дмитра Сергійовича

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

Дмитро Лебедка

(підпис)

Керівник,
Старший викладач комп'ютерних наук,
кандидат технічних наук

Олег Берест

(підпис)

СУМИ – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН-03 Лебедки Дмитра Сергійовича

1. Тема роботи: «Інформаційна система визначення прогнозу погоди для заданої місцевості»
затверджую наказом по СумДУ від «22» квітня 2024 р. № 0414-IV
2. Термін здачі здобувачем кваліфікаційної роботи до 1 червня 2024 року

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються для розробки інформаційної системи у вигляді телеграм боту.
3) Розробка інформаційної системи визначення прогнозу погоди для заданої місцевості.
4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження	06.05.24 – 10.05.24	
2	Огляд технологій, що використовуються для розробки інформаційної системи у вигляді телеграм боту	11.05.24 – 17.05.24	
3	Розробка інформаційної системи визначення прогнозу погоди для заданої місцевості	18.05.24 – 20.05.24	
4	Аналіз отриманих результатів	21.05.24 – 28.05.24	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	27.05.24 – 31.05.24	

Здобувач вищої освіти _____ Керівник _____

АНОТАЦІЯ

Записка: 42 стр., 17 рис., 1 додаток, 10 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки прогноз погоди може допомогти людям приймати кращі рішення щодо свого повсякденного життя. Завдяки постійному розвитку технологій прогнозування погоди ці системи стають ще більш корисними та цінними.

Об’єкт дослідження — інформаційна система визначення прогнозу погоди для заданої місцевості у вигляді телеграм боту.

Мета роботи — розробка інформаційної системи прогнозу погоди для заданої місцевості.

Методи дослідження — алгоритми створення інформаційних систем з використанням такого стеку технологій: Python + PostgreSql.

Результати — розроблено інформаційну систему прогнозу погоди для заданої місцевості у вигляді телеграм боту, у користувача є можливість отримувати актуальні дані про погодні умови для конкретного місця. Інформаційна система створена на базі мови програмування Python та СУБД PostgreSql.

ІНФОРМАЦІЙНА СИСТЕМА, ТЕЛЕГРАМ БОТ, ПРОГНОЗ ПОГОДИ, PYTHON, POSTGRESQL.

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки прогноз погоди може допомогти людям приймати кращі рішення щодо свого повсякденного життя. Завдяки постійному розвитку технологій прогнозування погоди ці системи стають ще більш корисними та цінними.

Об’єкт дослідження. Інформаційна система визначення прогнозу погоди для заданої місцевості у вигляді телеграм боту.

Предмет дослідження. Методологія створення інформаційної системи у вигляді телеграм боту.

Гіпотеза. Користувачі Telegram будуть частіше використовувати чат-бота погоди, який надає їм точну та персоналізовану інформацію про погоду для їхнього місцезнаходження.

Новизна. Розробка подібної інформаційної системи у вигляді телеграм бота створить можливість допомогти людям отримувати точну та персоналізовану інформацію про погоду для їхнього місцезнаходження.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, інформаційного та програмного забезпечення, візуалізації роботу додатку, висновків та списку використаних джерел.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз принципів побудови погодних ботів

Погодні боти є корисними та популярними інструментами для надання інформації про погоду користувачам. Для їх успішної розробки та функціонування важливо дотримуватися певних принципів та кращих практик.

Основні принципи та аналіз побудови погодних ботів:

- Збір даних**:** Використання даних з метеорологічних агентств та погодних API (наприклад, OpenWeatherMap).
- Інтеграція**:** Підключення до платформ через API та вебхуки (Telegram, Facebook Messenger, вебсайти).
- Алгоритми**:** Обробка природної мови (NLP) для розуміння запитів, розробка логіки відповідей.
- Інтерфейс**:** Інтуїтивний дизайн, зручні меню та графічні елементи.
- Безпека**:** Захист даних, дотримання конфіденційності користувачів.

Завдяки цьому погодні боти можуть надавати точну та своєчасну інформацію, підвищуючи комфорт і безпеку користувачів.

1.2 Інструменти для розробки

Для створення Telegram ботів існують кілька платформ та інструментів, які спрощують процес розробки, налаштування та управління ботами. Ось деякі з них:

1. BotFather (рис. 3.1) - бот у телеграмі, який допомагає створенню власного чат-боту. Він дає можливість користувачам, які хочуть створити й налаштувати бота для своїх цілей, при цьому є дуже непоганий додатковий функціонал[1].

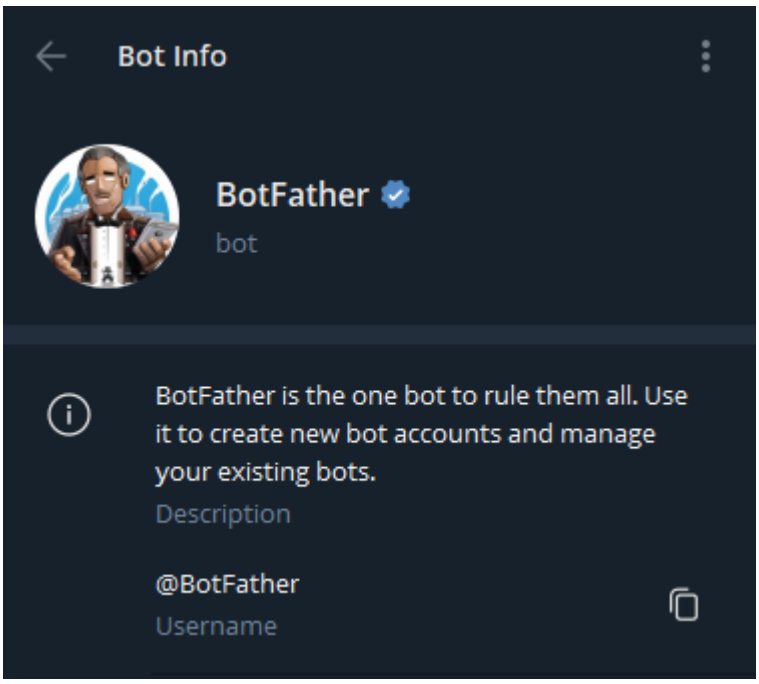

Рисунок 1.1 – BotFather офіційний інструмент телеграм для створення ботів

Деякі основні функції:

- Створення нових ботів: BotFather дозволяє створювати нових Telegram-ботів, надаючи їм ім'я та опис.
- Отримання токена бота: BotFather видає токен бота, який використовується для доступу до API Telegram і керування ботом.
- Налаштування бота: BotFather дозволяє налаштовувати різні параметри бота, такі як ім'я, опис, фото профілю та команди.
- Створення команд бота: BotFather дозволяє створювати команди для вашого бота, які користувачі можуть використовувати для взаємодії з ним.
- Отримання інформації про бота: BotFather дозволяє отримувати інформацію про вашого бота, таку як кількість підписників, активних користувачів та останні оновлення.

Переваги:

- Простота використання: BotFather надає простий та зрозумілий інтерфейс для створення та налаштування ботів, навіть для тих, хто не має досвіду в програмуванні.
- Широкі можливості: За допомогою BotFather можна створювати ботів з різноманітними функціями, від простих автоматичних відповідей до складних інтерактивних ботів з використанням кнопок, клавіатур та інтерфейсів.
- Підтримка Telegram API: BotFather надає доступ до Telegram API, що дозволяє використовувати всі можливості платформи Telegram для реалізації функціональності вашого бота.

Недоліки:

- Обмежений функціонал: Хоча BotFather має деякі базові можливості для створення ботів, він обмежує можливості для реалізації складних або дуже специфічних функцій.
- Необхідність додаткового програмування: Для створення деяких більш складних ботів може бути необхідне додаткове програмування на мовах, таких як Python або JavaScript, після налаштування через BotFather.
- Обмежені можливості адміністрування: BotFather не надає розширених інструментів для адміністрування бота після його створення, таких як аналітика використання, керування підписниками тощо[2].

2. Bot frameworks (рис. 1.2) - це програмні платформи, які допомагають розробникам створювати ботів для різних платформ обміну повідомленнями, таких як Telegram, Facebook Messenger, WhatsApp та інші. Вони надають набір інструментів, бібліотек та API, які спрощують процес розробки ботів, економлять час та зусилля[3].

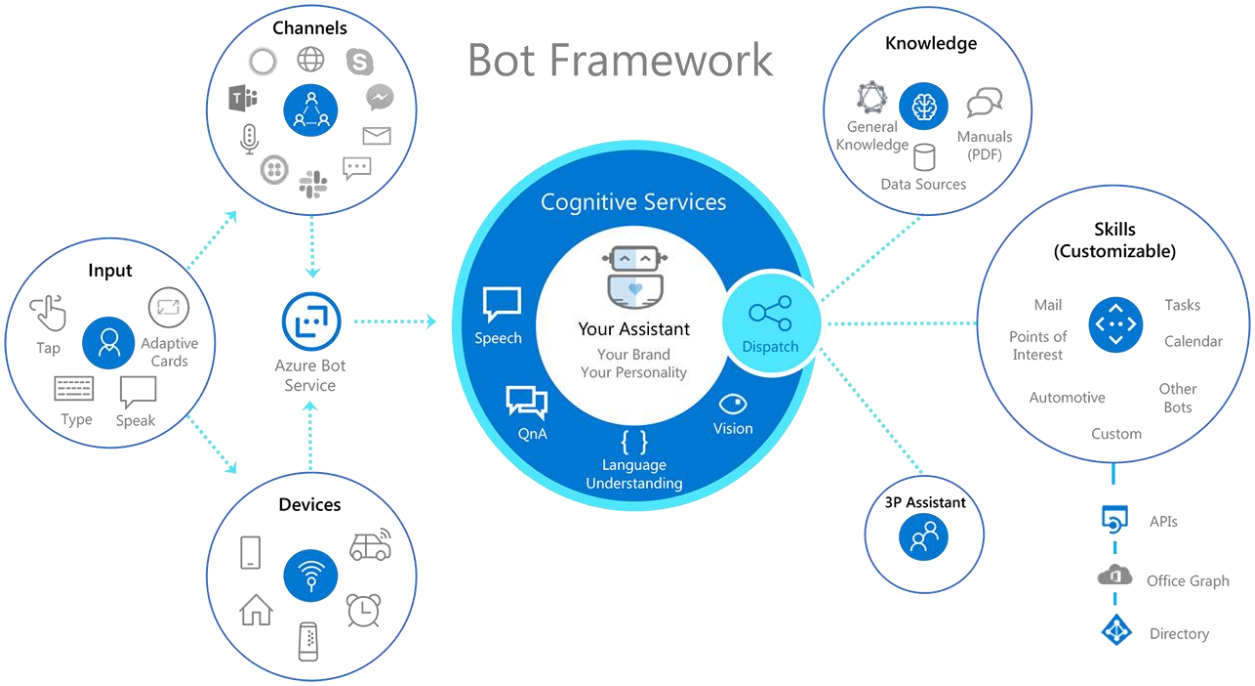

Рисунок 1.2 – Bot Frameworks

Переваги:

- Зниження складності: Bot frameworks позбавляють розробників від необхідності писати код з нуля, надаючи готові модулі та компоненти для виконання поширених завдань.
- Доступ до API: Bot frameworks надають доступ до API платформ обміну повідомленнями, роблячи інтеграцію з ними простою.
- Спільнота та підтримка: Більшість bot frameworks мають активну спільноту розробників та офіційну підтримку, що може допомогти у вирішенні проблем.
- Покращена масштабованість: Bot frameworks дозволяють легко масштабувати ботів для обробки великої кількості користувачів та запитів.
- Економія часу: Завдяки готовим рішенням, розробка ботів стає швидшою та ефективнішою.

Недоліки:

- Вивчення нового інструменту: Використання бот-фреймворків може вимагати вивчення нового інструменту або програмного середовища, що може бути часом і ресурсами затратним.

- Обмежені можливості: Деякі бот-фреймворки можуть мати обмежені можливості або бути менш гнучкими у порівнянні з іншими способами розробки ботів.
- Залежність від розробника: Використання бот-фреймворка може зробити вас залежними від певного розробника або компанії, що розробляє фреймворк, та обмежити вашу свободу вибору або масштабованість вашого проекту[4].

Перед вибором bot framework важливо чітко визначити свої потреби та бюджет, а також дослідити доступні варіанти, щоб знайти той, який найкраще відповідає вашим вимогам.

3. Chatfuel (рис. 1.3) - це платформа для створення чат-ботів без необхідності програмування. Ви можете використовувати графічний інтерфейс для створення відповідей бота, встановлювати різні дії в залежності від вхідних повідомлень, додавати кнопки, зображення та інші медіа.

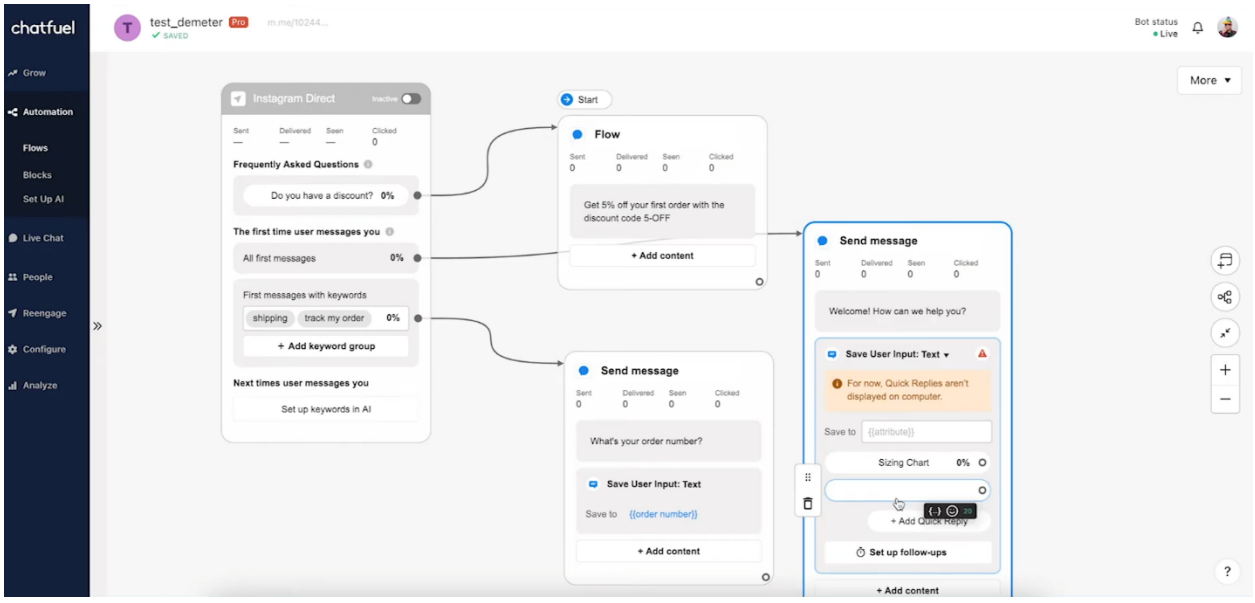

Рисунок 1.3 – Chatfuel інтерфейс

Переваги:

- Безкоштовний план: Chatfuel пропонує безкоштовний план з обмеженими функціями, що робить його чудовим вибором для початківців.
- Швидке створення: Завдяки готовим шаблонам та компонентам можна створювати ботів за лічені хвилини.
- Простота використання: Chatfuel має інтуїтивно зрозумілий інтерфейс drag-and-drop, який дозволяє створювати ботів без кодування.
- Підтримка спільноти: Chatfuel має активну спільноту користувачів та офіційну підтримку, що може допомогти у вирішенні проблем.
- Багатофункціональність: Chatfuel пропонує широкий спектр функцій, включаючи збір даних, розгалуження діалогів, інтеграцію з іншими платформами та багато іншого.

Недоліки:

- Обмежена масштабованість: Chatfuel ідеально підходить для швидкого створення простих чат-ботів, для складніших проектів або великих обсягів трафіку вам може знадобитися більш масштабований підхід.
- Обмежені можливості програмування: Хоча Chatfuel надає широкий функціонал для створення ботів без програмування, він може бути обмеженим для тих, хто має специфічні потреби та вимагає розширених можливостей програмування.
- Залежність від Facebook: Використання Chatfuel обмежено лише платформою Facebook Messenger, тому якщо вам потрібно створити бота для інших месенджерів або каналів, вам може знадобитися використовувати інші інструменти [5].

Загалом, Chatfuel є корисним інструментом для швидкого створення чат-ботів для Facebook Messenger з базовою або середньою складністю, але важливо враховувати його обмежені можливості в порівнянні з більш розширеними платформами.

4. Dialogflow (раніше API.AI) (рис. 1.4) - інструмент від Google для створення розумних чат-ботів. Він дозволяє аналізувати вхідні

повідомлення користувачів, розпізнавати їх наміри та відповідати згідно цих намірів. Dialogflow підтримує різні мови та має вбудовані інтеграції з іншими платформами.

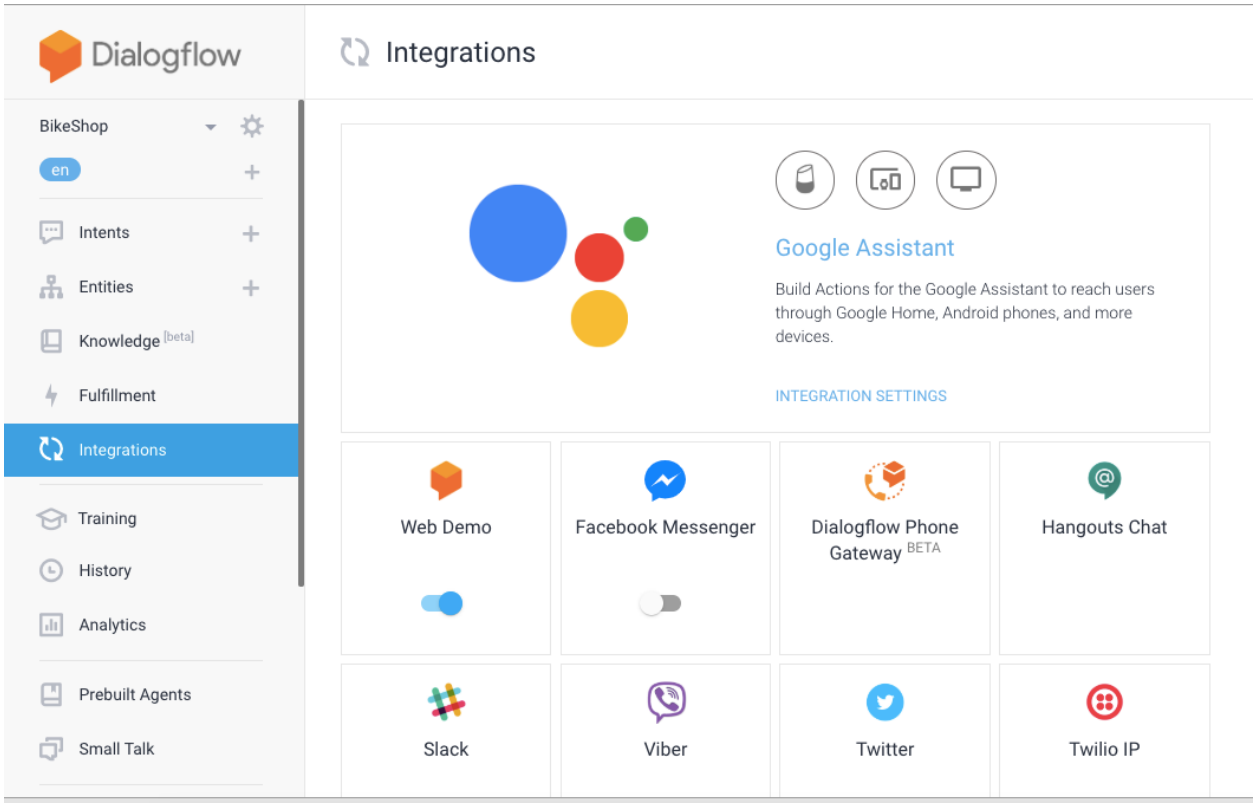

Рисунок 1.4 – Dialogflow інтерфейс

Основні функції:

- Інтеграція з різними платформами: Dialogflow підтримує інтеграцію з різними платформами, такими як Facebook Messenger, Telegram, Skype, Twitter, Google Assistant, Amazon Alexa та іншими.
- Мовознавство та розуміння природної мови (NLU): Dialogflow використовує потужні алгоритми машинного навчання для аналізу тексту користувача та розуміння його намірів.
- Створення розміток розмов: За допомогою Dialogflow ви можете створювати розмітки розмов, що дозволяє визначати ключові слова та фрази, які користувач може використовувати під час взаємодії з ботом.

Переваги:

- Гнучкість: Dialogflow можна використовувати для створення чат-ботів та голосових помічників для різних платформ та випадків використання.
- Потужні можливості NLU: Dialogflow має одні з найкращих можливостей NLU на ринку, що дозволяє створювати чат-ботів, які розуміють природну мову.
- Інтеграція з Google Cloud: Dialogflow легко інтегрується з іншими продуктами Google Cloud, такими як Dialogflow.
- Простота використання: Dialogflow має простий у використанні візуальний інтерфейс, який робить його доступним навіть для початківців.
- Масштабованість: Dialogflow може масштабуватися для обробки великої кількості користувачів та запитів[6].

Недоліки:

- Витрати на використання: При використанні платних планів Dialogflow ви можете зіткнутися з витратами на обробку текстових запитів та інші сервіси Google Cloud.
- Залежність від інфраструктури Google: Використання Dialogflow означає залежність від інфраструктури та сервісів Google, що може викликати певні обмеження та обмежити вашу гнучкість у виборі інструментів та технологій.
- Обмеженість безкоштовного плану: Безкоштовний план Dialogflow обмежує кількість запитів на обробку тексту та інші функції, що може бути недостатньо для великих або активних проектів[7].

Перед використанням Dialogflow важливо врахувати його вартість, криву навчання та залежність від Google Cloud.

5. ManyChat (рис. 1.5) - ще одна платформа для створення чат-ботів, яка спрощує процес розробки та управління. ManyChat дозволяє додавати відповіді, кнопки, каруселі, зображення та багато іншого без необхідності програмування.

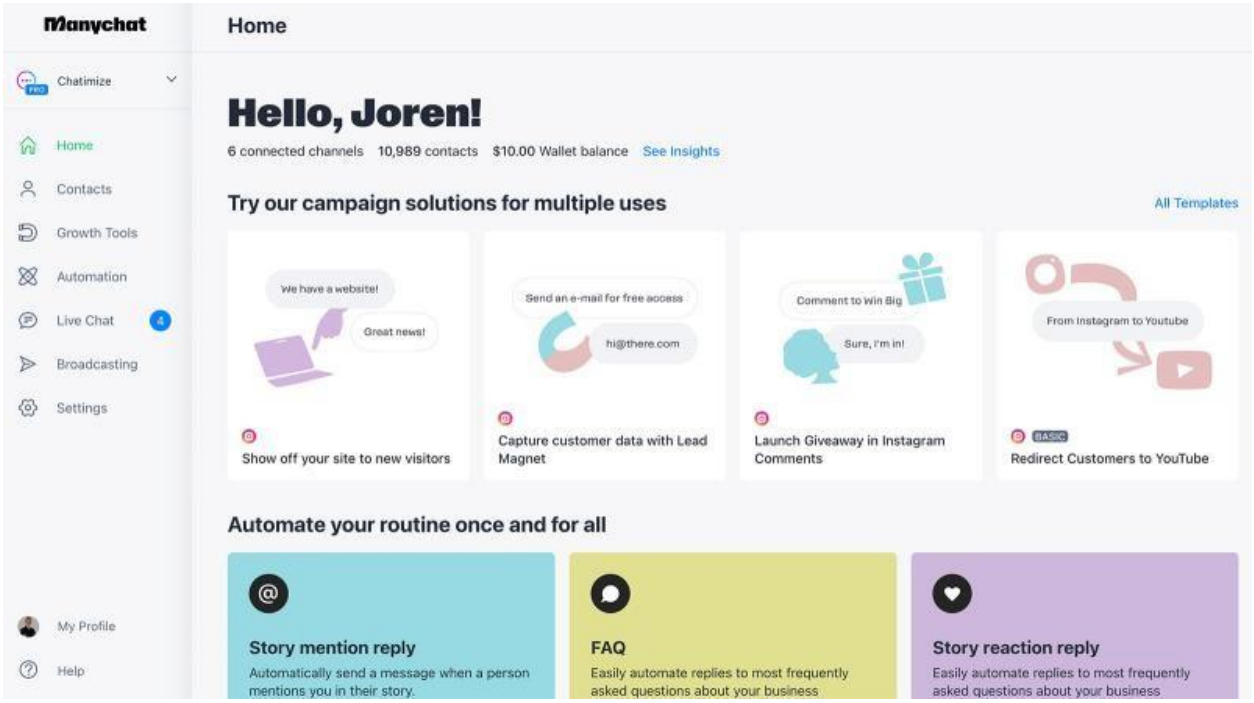

Рисунок 1.5 – ManyChat інтерфейс

Основні функції:

- Створення автоматизованих чат-ботів: ManyChat надає зручний веб-інтерфейс та інструменти для створення чат-ботів без програмування. Ви можете налаштовувати ботів для надання автоматичних відповідей, запитань та взаємодії з користувачами.
- Використання віджетів та кнопок: ManyChat дозволяє додавати віджети та кнопки для спрощення взаємодії з користувачами, що полегшує навігацію та виконання дій у боті.
- Аналітика та відстеження: Платформа надає засоби для аналізу використання бота, такі як статистика повідомлень, конверсія та інші метрики, що дозволяє вам вдосконалювати взаємодію з користувачами[8].

Переваги:

- Багатофункціональність: ManyChat пропонує широкий спектр функцій, які роблять його чудовим вибором для підприємств, які хочуть створювати чат-ботів для різних цілей.
- Підтримка спільноти: ManyChat має активну спільноту користувачів та офіційну підтримку, що може допомогти у вирішенні проблем.

- Простота використання: ManyChat має простий у використанні візуальний інтерфейс drag-and-drop, який робить його доступним навіть для початківців.
- Безкоштовний план: ManyChat пропонує безкоштовний план з обмеженими функціями, що робить його чудовим вибором для початківців.

Недоліки:

- Обмежені можливості безкоштовної версії: Безкоштовний план ManyChat має обмеження щодо кількості підписників та функціональності, що може бути обмежуючим для деяких користувачів.
- Залежність від платформи: ManyChat побудований на інфраструктурі Facebook Messenger, тому ви залежите від змін та обмежень, які накладає Facebook.
- Обмежена гнучкість: Деякі користувачі можуть вважати, що ManyChat має обмежені можливості для реалізації складних або специфічних сценаріїв взаємодії з користувачами.

Загалом, ManyChat є потужним інструментом для створення та управління чат-ботами з великим набором функцій, проте важливо розуміти його обмеження та залежність від платформи Facebook[9].

Ці інструменти надають можливості для створення ботів на різних рівнях складності, від простих текстових ботів до більш складних інтерактивних рішень.

1.3 Постановка задачі

За допомогою мови програмування Python потрібно створити інформаційну систему у вигляді телеграм-бота для отримання погодної інформації, дотримуючись таких етапів побудови:

- Вибір інструментів розробки: Провести огляд доступних інструментів для створення інформаційної-системи у вигляді телеграм-бота і обрати оптимальні.

- Розробка бази даних: Створити базу даних для зберігання інформації про місцезнаходження користувача
- Розробка структури бота: Розробити зручну структуру телеграм-бота, включаючи створення меню для користувачів.
- Розробка алгоритму роботи бота: Розробити алгоритм, який дозволить користувачам запитувати погодну інформацію для конкретного місця та отримувати актуальні дані.
- Тестування бота: Провести тестування телеграм-бота, переконатися, що він працює безперебійно та зручно для користувачів, і відповідає всім поставленим завданням.

Після завершення цих етапів, користувачі зможуть зручно та безперебійно отримувати погодну інформацію через інформаційну систему у вигляді телеграм-бота.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Проектування бази даних

База даних є неодмінною частиною кожного проекту. Вона використовується для зберігання та обробки різноманітної інформації, включаючи дані про клієнтів, продукцію та інші аспекти. При виборі СУБД для погодного боту, існує багато альтернатив, таких як PostgreSQL, MySQL, Microsoft SQL Server, SQLite, MongoDB, Redis, IBM DB2 та Elasticsearchx[10].

У даному випадку для розробки погодного боту буде використована СУБД PostgreSQL. Ця база даних відома своєю надійністю, відкритим кодом та активною спільнотою, яка завжди готова надати допомогу та відповісти на всі питання, пов'язані з її використанням[11].

Створено ER-діаграму (рис. 2.1) бази даних для нашої інформаційної системи.

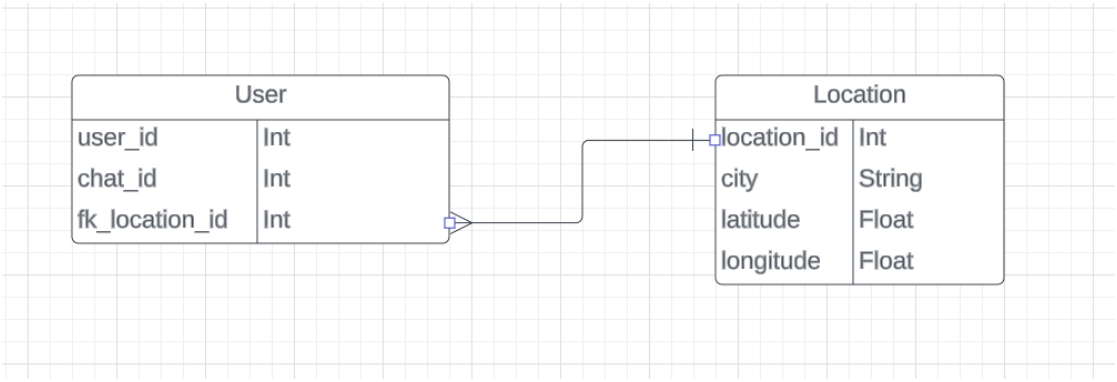

Рисунок 2.1 – Схема бази даних телеграм боту

Таблиця User містить:

1. user_id - унікальний номер користувача в базі
2. chat_id - унікальний номер чату з ним
3. fc location id – посилання на його локацію

Таблиця Location містить:

1. location_id – унікальний номер в базі

2. city – назва міста
3. latitude – широта
4. longitude – довгота

2.2 Огляд інструментів для розробки

1. PyCharm (рис. 2.2) - інтегроване середовище розробки, де буде зручно писати код на мові Python. Має у своєму складі зручні надбудови[12].

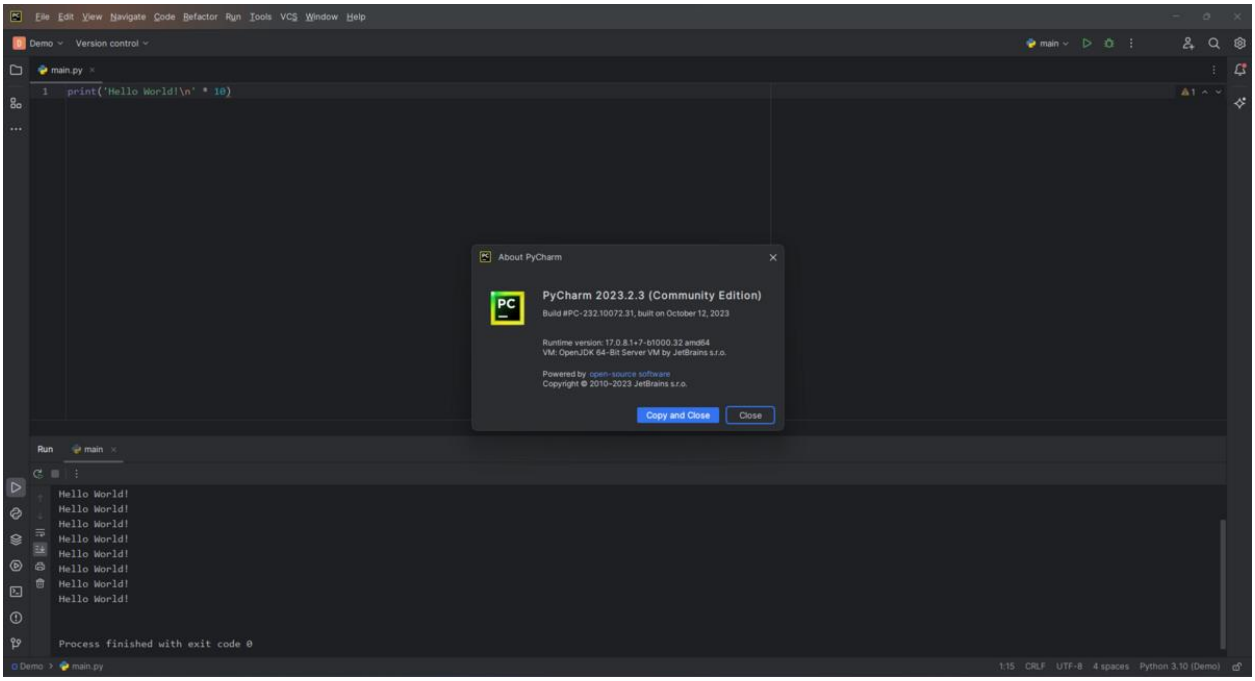

Рисунок 2.2 – Інтерфейс PyCharm

Можливості PyCharm:

- Аналіз коду;
- Інтеграція з Git;
- Редагування та автодоповнення коду;
- Відладка та тестування;
- Розширюваність;
- Наукові обчислення.

До переваг можна віднести:

- Простий у використанні;
- Активна спільнота;
- Безкоштовний та платний варіанти;
- Регулярні оновлення;

- Потужні функції.

Недоліків не так і багато, але вони є:

- Відсутність деяких функцій;
- Складність деяких функцій;
- Високі системні вимоги[13].

2. PGAdmin (рис. 2.3) - зручний інструмент для роботи з базою даних PostgreSQL, що має приємний та зрозумілий інтерфейс користувача[13].

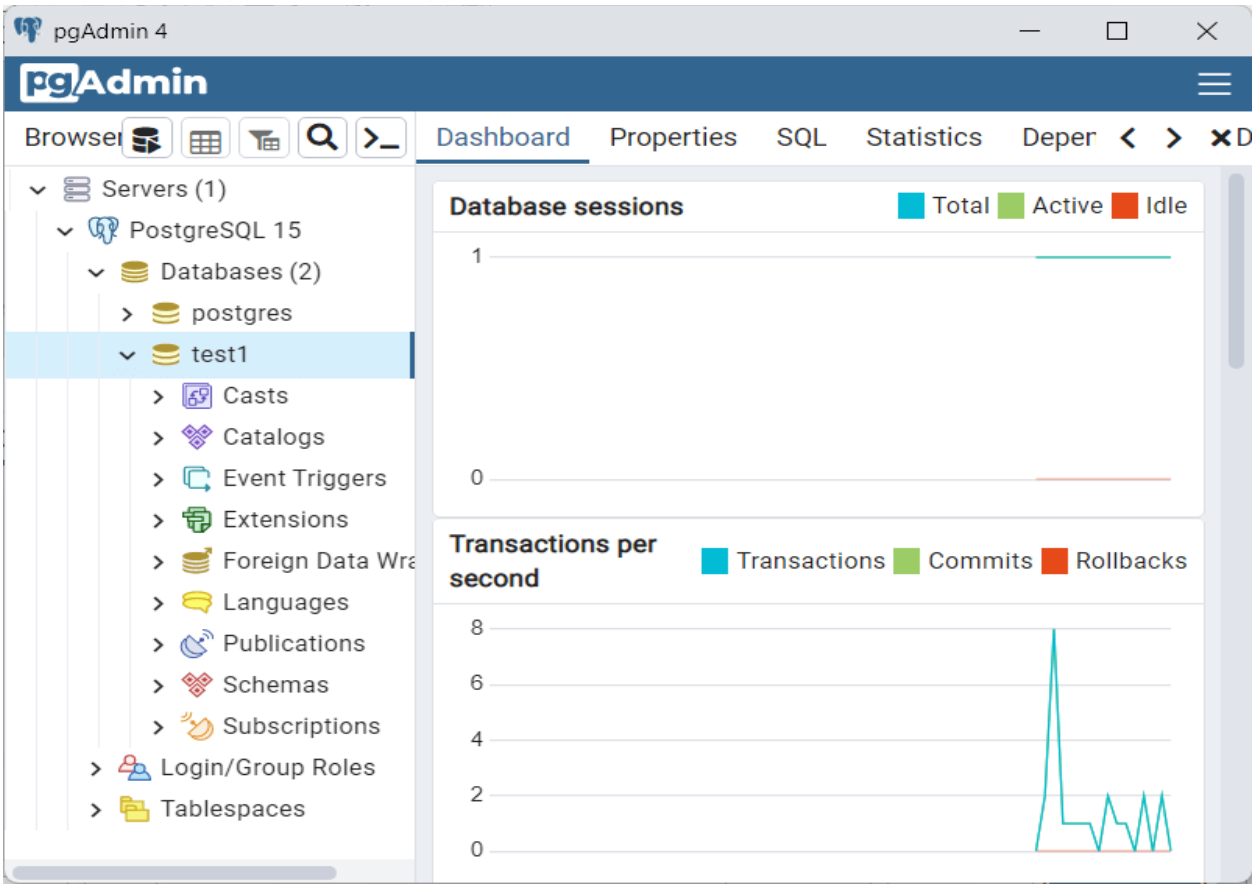

Рисунок 2.3 – Інтерфейс pgAdmin 4

PGAdmin має такі можливості:

- Візуалізація даних;
- Імпорт та експорт даних;
- Підтримка віддалених серверів;
- Виконання SQL запитів;
- Створення та керування об'єктами бази даних.

Переваги:

- Простий у використанні;
- Активна спільнота;
- Регулярні оновлення;
- Безкоштовний та відкритий;
- Потужні функції.

Недоліки:

- Обмежені можливості порівняно з комерційними інструментами;
- Можливість збоїв та проблем з продуктивністю[14].

2.3 Клієнтська частина система

Для взаємодії користувача з нашим погодним ботом ми будемо використовувати комбінацію інтерактивних кнопок, рукописного вводу та команд, які пропонує сам Telegram.

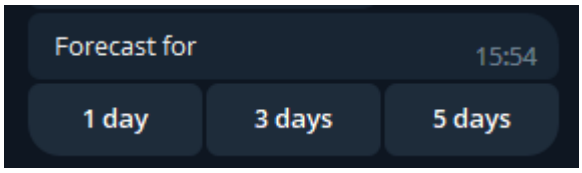

Рисунок 2.4 – Приклад інтерактивних кнопок

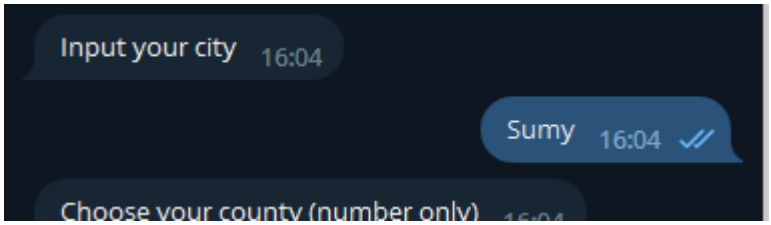

Рисунок 2.5 – Приклад взаємодії за допомогою вводу користувача

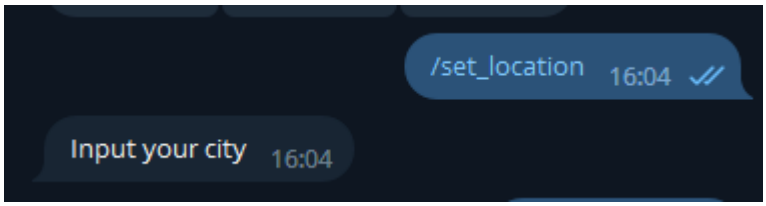

Рисунок 2.6 – Приклад взаємодії за допомогою команд Telegram

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Розробка дизайну

Use Case (сценарій використання) — це список дій, які юзер може виконувати для роботи з програмою або додатком, задля досягнення поставленої задачі.

Тестування варіантів використання виконується для виявлення логічних прогалин або помилок у веб-програмі, які важко виявити під час тестування певних модулів або частин веб-програми.

У основі своїй варіанти використання описують, що саме система робить, а не як[15].

Розроблено Use Case діаграму (рис. 3.1) для нашої інформаційної системи у вигляді телеграм боту.

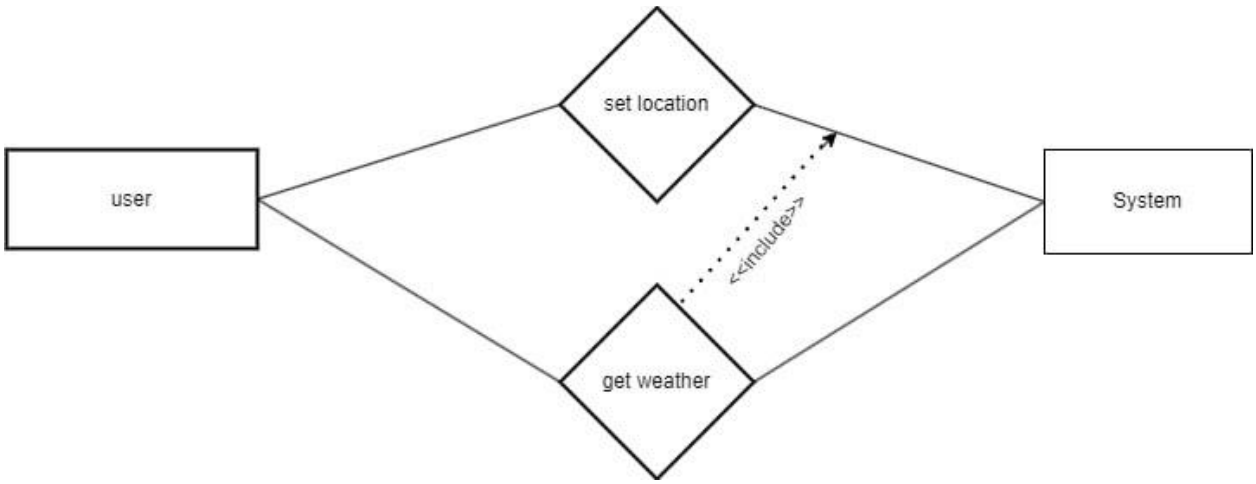

Рисунок 3.1 – Use Case діаграма

З нашої діаграми варіантів використання визначимо умови для взаємодії з ботом:

1. Користувач повинен виконати будь-яку операцію, щоб отримати результат.
2. Користувач має обрати локацію, щоб виконати запит прогнозу погоди.
3. Користувач повинен обрати певну кількість днів, для отримання прогнозу погоди.

3.2 Програмна реалізація

Зробимо опис основних файлів: bot.py, keyboard.py, database.py, geocoding_API.py, weather.py (Додаток А), які були створені для роботи інформаційної системи визначення прогнозу погоди для заданої місцевості у вигляді телеграм боту:

1) bot.py – основний додаток телеграм-бота.

```
import asyncio

from aiogram import Bot, Dispatcher

from configparser import ConfigParser

from handlers import city_input, main_commands, weather
```

Імпортуються необхідні модулі. asyncio використовується для асинхронного програмування, aiogram – це бібліотека для роботи з ботами Telegram, ConfigParser використовується для читання конфігураційних файлів, а handlers – це модуль, що містить обробники команд та повідомлень (наприклад, city_input, main_commands, weather).

```
config = ConfigParser()

config.read('config.ini')

TOKEN = config.get('auth', 'TOKEN')
```

Створюється об'єкт ConfigParser для читання конфігураційного файлу config.ini, звідки витягується токен для бот з секції auth (змінна TOKEN).

```
bot = Bot(token=TOKEN)
```

Створюється екземпляр бот з використанням токена.

```
async def main():
    dp = Dispatcher()

    dp.include_routers(
        weather.router,
        city_input.router,
        main_commands.router
    )



    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)
```

У цій функції створюється Dispatcher - об'єкт, який керує обробкою вхідних повідомлень та команд. Потім до нього додаються маршрутизатори (обробники команд) з модулів weather, city_input та main_commands. await bot.delete_webhook(drop_pending_updates=True) видаляє вебхук і скидає всі очікувані оновлення, щоб бот почав роботу з нуля. await dp.start_polling(bot) запускає процес отримання та обробки нових оновлень (повідомлень та команд) від Telegram.

```
if __name__ == '__main__':
    asyncio.run(main())
```

Запуск програми.

2) keyboard.py – inline клавіатура для взаємодії з користувачем.

```
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup
```

Імпортуються класи InlineKeyboardButton та InlineKeyboardMarkup із модуля aiogram.types. InlineKeyboardButton використовується для створення кнопок, а InlineKeyboardMarkup для створення розмітки клавіатури.

```
buttons = [
  InlineKeyboardButton(text="1 day", callback_data='get_1_day_weather'),
  InlineKeyboardButton(text="2 days", callback_data='get_2_days_weather'),
  InlineKeyboardButton(text="3 days", callback_data='get_3_days_weather')
  ]
```

Створюється список кнопок buttons, де кожна кнопка представлена об'єктом InlineKeyboardButton. В даному випадку створюються три кнопки з текстами "1 day", "2 days" та "3 days". Відповідні дані зворотного виклику: get_1_day_weather, get_2_days_weather та get_3_days_weather.

```
mainMenu = InlineKeyboardMarkup(inline_keyboard=[buttons])
```

Створюється об'єкт InlineKeyboardMarkup, який є розміткою клавіатури. Аргумент inline_keyboard приймає перелік списків кнопок. Це дозволяє організовувати кнопки в рядки та стовпці. У цьому випадку inline_keyboard=[buttons] означає, що всі три кнопки будуть в одному рядку.

3) database.py – файл роботи з базою даних, створення движка, запис та отримання локації користувача

```
from sqlalchemy import create_engine, Integer, Column, String, Float
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import services.geocoding_API as geocoding_API
```

Імпортуються модулі SQLAlchemy до роботи з базою даних, і модуль geocoding_API, який, надає функції отримання координат міста за його назвою.

```
engine = create_engine('postgresql+psycopg2://postgres:12345@localhost:5432/postgres')
connection = engine.connect()
```

Створюється двигун для підключення до бази даних PostgreSQL із використанням бібліотеки psycopg2. Потім встановлюється з'єднання з базою даних.

```
class User(Base):
    __tablename__ = 'users'
    user_id = Column(Integer, primary_key=True)
    chat_id = Column(Integer)
    fk_location_id = Column(Integer)
class Location(Base):
    __tablename__ = 'locations'
    location_id = Column(Integer, primary_key=True)
    city = Column(String)
    latitude = Column(Float)
    longitude = Column(Float)
```

Ці класи визначають структури таблиць users і locations. Модель User містить інформацію про користувача, включаючи chat_id та зовнішній ключ fk_location_id, який посилається на запис у таблиці locations. Модель Location містить інформацію про місцезнаходження, включаючи city, latitude та longitude.

```
def set_location(chat_id: int, country: str, city: str):
    coord = geocoding_API.get_city_coords(country, city)
    session = Session()
    if (session.query(Location).filter(Location.latitude == coord['lat'], Location.longitude == coord['lng'])
            .first() is None):
        location = Location(city=city, latitude=coord['lat'], longitude=coord['lng'])
        session.add(location)
        session.commit()
    fk_location = session.query(Location).filter(Location.latitude == coord['lat'],
                                                 Location.longitude == coord['lng']).first()
    if session.query(User).filter(User.chat_id == chat_id).first() is None:
        user = User(chat_id=chat_id, fk_location_id=fk_location.location_id)
        session.add(user)
    else:
        a = session.query(User).filter(User.chat_id == chat_id).first()
        a.fk_location_id = fk_location.location_id
    session.commit()
```

Функція для установки розташування користувача. Вона отримує координати міста за допомогою функції geocoding_API.get_city_coords, зберігає місцезнаходження в базі даних, якщо воно ще не існує, та пов'язує користувача з цим місцезнаходженням. Якщо користувач вже існує, оновлюється його розташування.

```
def get_location(chat_id: int) -> dict[str, float]:
    session = Session()
    loc = session.query(User).filter(User.chat_id == chat_id).first()
    if loc is None:
        return UserNotFound
    else:
        loc = session.query(Location).filter(Location.location_id == loc.fk_location_id).first()
        coord = {'lat': loc.latitude, 'lng': loc.longitude}
        return coord
```

Функція для отримання розташування користувача. Повертає координати розташування користувача за його chat_id. Якщо користувач не знайдено, повертається виключення UserNotFound.

4) Geocoding_API.py – файл роботи з API геокодінга для отримання координат локації введеної користувачем.

```
from configparser import ConfigParser
import requests
import json
```

Імпортуються необхідні модулі. ConfigParser використовується для роботи з конфігураційними файлами, requests – для надсилання HTTP-запитів, та json – для обробки JSON-відповідей.

```
config = ConfigParser()
config.read('config.ini')
GEOCODING_API_KEY = config.get('auth', 'GEOCODING_API_KEY')
```

Створюється об'єкт ConfigParser, читається файл config.ini, та витягується ключ API для геокодування із секції auth.

```
def get_city_coords(country: str, city: str) -> dict[str, float]:
    req = requests.get(base_url + f"&q={city}").text
    hits = json.loads(req)['hits']
    for item in hits:
        if 'country' in item:
            if item['country'] == country and item['name'] == city:
                point = item['point']
                coord = {'lat': point['lat'], 'lng': point['lng']}
                return coord
```

Функція для отримання координат міста, вона приймає назву країни та міста як аргументи та повертає словник з координатами міста (широта та довгота).

```
def get_country_list(city: str) -> dict[int, str]:
    req = requests.get(base_url + f"&q={city}")
    hits = json.loads(req.text)['hits']
    n = 1
    location_options = {}
    for item in hits:
        if 'country' in item:
            if item['country'] not in location_options.values():
                location_options[n] = f'{item["country"]}'
                n += 1
    return location  options
```

Ця функція приймає назву міста та повертає словник, де ключами є порядкові номери, а значення – назви країн, в яких є місто з таким ім'ям.

5) weather.py – файл для роботи з програмною API, отримання прогнозу погоди на 1, 2 та 3 дні.

```
from datetime import datetime, timedelta
from configparser import ConfigParser
import requests
import json
import services.database as database
```

Імпортуються необхідні модулі. datetime та timedelta для роботи з датою та часом, ConfigParser для роботи з конфігураційними файлами, requests для

виконання HTTP-запитів, json для обробки JSON-відповідей, services.database для взаємодії з базою даних (передбачається, що цей модуль надає функції для роботи з базою даних).

```
config = ConfigParser()
config.read('config.ini')
WEATHER_API_KEY = config.get('auth', 'WEATHER_API_KEY')
base_url = "https://api.weatherapi.com/v1/forecast.json"\
    "?key=" + WEATHER_API_KEY
```

Читання конфігурації та налаштування API.

```
def weather_1_day(chat_id: int) -> str:
    try:
        coord = database.get_location(chat_id)
    except database.UserNotFound:
        return "Input your location (/set_location)"
    req = requests.get(base_url + f"&lat={coord['lat']}&lon={coord['lng']}").text
    json_dict = json.loads(req)
    day = json_dict['daily'][0]
    summary_weather = f"{datetime.fromtimestamp(day['dt']).strftime('%d.%m.%Y')} \n" \
                      f"Temperature: {day['temp']['min']} - {day['temp']['max']} \n" \
                      f"Weather: {day['weather'][0]['description']} \n" \
                      f"Humidity: {day['humidity']}%\n"
    return summary_weather
```

Функція для отримання прогнозу погоди на один день.

```
def weather_several_days(chat_id: int, days: int) -> str:
    try:
        coord = database.get_location(chat_id)
    except database.UserNotFound:
        return "Input your location (/set_location)"
    resp = requests.get(base_url + f"&q={coord['lat']},{coord['lng']}&days={days}")
    json_dict = json.loads(resp.text)
    forecast_days = json_dict['forecast']['forecastday']
    summary_weather = ''
    for item in forecast_days:
        date = item['date']
        day = item['day']
        max_temp = day['maxtemp_c']
        min_temp = day['mintemp_c']
        condition = day['condition']['text']
        humidity = day['avghumidity']
        summary_weather += f"Date: {date} \n" \
                           f"Temperature: {min_temp} - {max_temp} \n" \
                           f"Condition: {condition} \n" \
                           f"Humidity: {humidity}%\n\n"
    return summary_weather
```

Функція для отримання прогнозу погоди на кілька днів.

3.3 Робота додатку

У процесі створення додатку отримана працездатна інформаційна система прогнозу погоди для заданої місцевості у вигляді телеграм боту.

Вводимо команду /set_location та назву міста щоб зазначити місце розташування користувача, обираємо країну зі списку в якій знаходиться це місто (рис. 3.2).

Рисунок 3.2 – Встановлення розташування користувача

Обираємо кількість днів для прогнозу погоди (рис. 3.3).

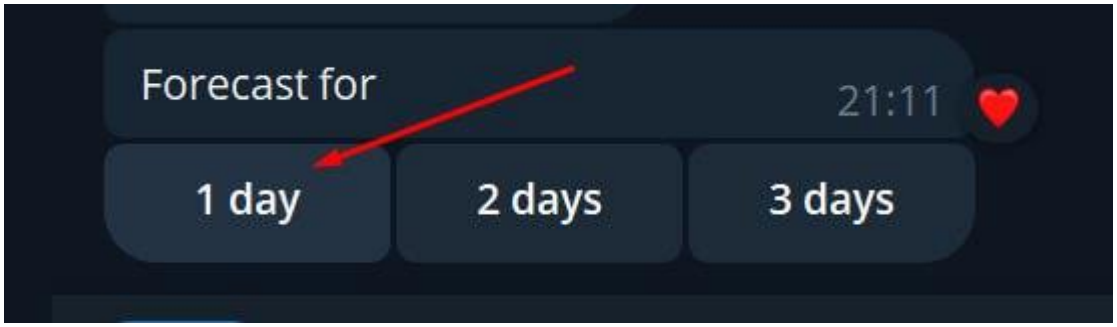

Рисунок 3.3 – Вибір

Отримуємо прогноз на 1, 2 або 3 дні на вибір (рис. 3.4, 3.5, 3.6). Він містить такую інформацію: дату, мінімальну та максимальну температуру на день, стан погоди та вологість повітря.

Рисунок 3.4 – Прогноз погоди на 1 день

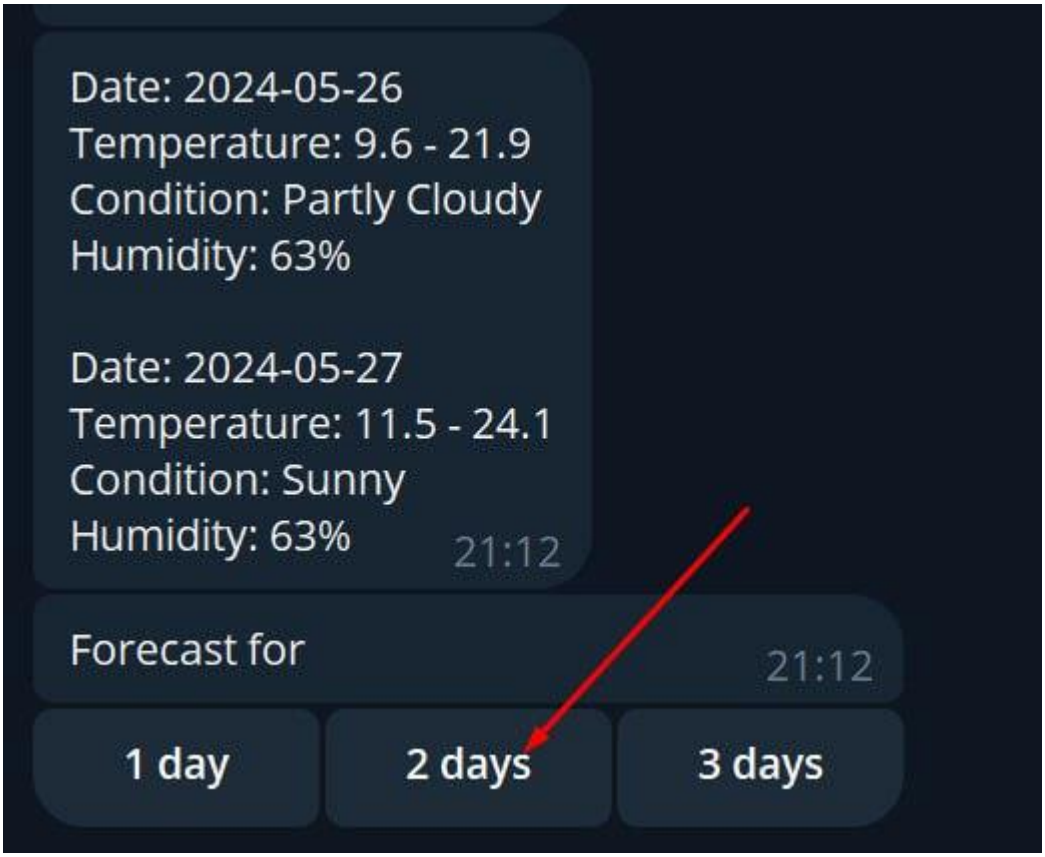

Рисунок 3.5 – Прогноз погоди на 2 дні

Рисунок 3.6 – Прогноз погоди на 3 дні

4 ПОДАЛЬШЕ ПОКРАЩЕННЯ ПРОЕКТУ

Розробка інформаційних систем для прогнозування погоди набуває все більшої популярності завдяки зручності та оперативності отримання інформації про погоду. Поточна система, яка надає прогнози на 1, 2 та 3 дні, вже є корисним інструментом для користувачів. Проте існує багато можливостей для її вдосконалення з метою підвищення функціональності, точності прогнозів та зручності користування. В даному розділі розглянемо кілька потенційних напрямків для покращення нашої інформаційної системи.

4.1 Покращення точності прогнозів

Одним з основних напрямків покращення інформаційної системи є підвищення точності прогнозів погоди. Це можна досягти за допомогою ряду методів, які охоплюють інтеграцію з більш надійними погодними API, використання технологій машинного навчання та аналізу великих даних.

Інтеграція з Надійними Погодними API

Для підвищення точності прогнозів погоди важливо інтегрувати систему з більш надійними та точними погодними API. На ринку існують кілька провідних постачальників погодних даних, таких як Dark Sky, AccuWeather та Weather Underground. Ці сервіси забезпечують детальні та точні прогнози на основі великої кількості метеорологічних станцій та супутникових даних. Інтеграція з такими API дозволить значно підвищити точність прогнозів та охопити ширший спектр погодних умов.

Використання Машинного Навчання

Машинне навчання надає великі можливості для аналізу погодних даних та створення більш точних моделей прогнозування. Використання історичних даних про погоду, таких як температура, вологість, тиск, швидкість та напрям вітру, дозволяє створювати моделі, які можуть передбачати майбутні погодні умови з високою точністю. Застосування методів регресії, нейронних мереж

та інших алгоритмів машинного навчання допоможе оптимізувати прогнозування погоди та враховувати локальні метеорологічні особливості.

Аналіз Великих Даних

Сучасні технології дозволяють збирати та аналізувати великі обсяги метеорологічних даних з різних джерел, таких як метеостанції, супутники, радари та інші сенсори. Аналіз цих даних у реальному часі дозволяє враховувати зміни погодних умов та швидко адаптувати прогнози. Використання технологій Big Data та хмарних обчислень допоможе підвищити продуктивність та точність прогнозування.

Моделі Мікроклімату

Ще одним напрямком покращення точності прогнозів є врахування мікрокліматичних особливостей регіонів. Мікрокліматичні моделі дозволяють враховувати локальні географічні та кліматичні особливості, такі як рельєф місцевості, близькість до водних об'єктів, лісові масиви тощо. Це дозволить створювати більш точні прогнози для конкретних місцевостей, особливо у випадках, коли загальні моделі можуть бути недостатньо точними.

Використання Додаткових Даних

Для підвищення точності прогнозів можна також використовувати додаткові дані з різних джерел, такі як дані про дорожні умови, авіаційні дані, морські прогнози тощо. Ці дані допоможуть краще розуміти та передбачати зміни погодних умов, особливо в умовах швидко змінюваних погодних ситуацій.

4.2 Розширення функціональності

Розширення функціональності інформаційної системи для прогнозування погоди є важливим кроком для підвищення її корисності та задоволення потреб користувачів. Існує кілька напрямків, у яких можна розширити функціональність системи, щоб надати користувачам більш детальну та корисну інформацію.

Прогноз на Тиждень

Одним із найочевидніших покращень є додавання можливості отримання прогнозу на тиждень вперед. Це дозволить користувачам краще планувати свої дії, враховуючи майбутні погодні умови. Така функція може бути реалізована через інтеграцію з погодними API, які надають дані про довгострокові прогнози. Додатково можна надавати графічні інтерпретації прогнозу, що зробить інформацію більш зрозумілою та візуально привабливою.

Оповіщення про Зміни Погоди

Важливою функцією може стати автоматичне оповіщення користувачів про значні зміни погодних умов, такі як штормові попередження, різке зниження або підвищення температури, сильні вітри чи інші небезпечні метеорологічні явища. Це допоможе користувачам вчасно реагувати на потенційні небезпеки та приймати відповідні заходи. Оповіщення можуть надходити у вигляді push-повідомлень, електронних листів або SMS.

Детальніші Прогнози

Крім загального прогнозу, можна надати користувачам більш детальну інформацію про погодні умови, такі як вологість, тиск, швидкість і напрям вітру, індекс ультрафіолетового випромінювання, видимість тощо. Це дозволить користувачам отримувати повнішу картину про поточні та майбутні погодні умови і краще планувати свої дії.

Інтерактивна Карта Погоди

Додавання інтерактивної карти погоди, яка показуватиме поточні та прогнозовані погодні умови в режимі реального часу, значно покращить користувацький досвід. Користувачі зможуть вибирати різні регіони, щоб побачити локальні прогнози, а також переглядати анімації змін погоди

протягом дня або тижня. Така функція може бути особливо корисною для подорожуючих або професіоналів, які залежать від точних погодних даних.

Голосові Команди

Для зручнішого користування системою можна додати підтримку голосових команд. Це дозволить користувачам взаємодіяти з системою, використовуючи голосові запити, що зробить процес отримання прогнозів швидшим та зручнішим. Інтеграція з популярними голосовими асистентами, такими як Google Assistant чи Amazon Alexa, розширить можливості використання системи.

Геолокаційні Запити

Автоматичне визначення локації користувача для надання прогнозу без необхідності ручного введення місцезнаходження є ще однією корисною функцією. Це можна реалізувати за допомогою GPS або IP-геолокації. Така функція зробить процес отримання прогнозу більш зручним, особливо для користувачів, які часто переміщуються.

Інтеграція з Календарем

Можливість інтеграції з календарем користувача для автоматичного додавання прогнозів погоди до подій може стати значним покращенням. Користувачі зможуть отримувати погодні прогнози для своїх запланованих заходів та подій, що допоможе краще підготуватись до них. Це може бути корисним як для особистих, так і для професійних заходів.

Індивідуальні Налаштування

Додавання можливості налаштування системи під індивідуальні потреби користувачів також є важливим аспектом. Користувачі зможуть налаштувати, які саме параметри погоди їх цікавлять, як часто вони хочуть отримувати

оновлення та в якому форматі. Це дозволить зробити систему більш гнучкою та зручною для різних категорій користувачів.

4.3 Покращення інтерактивності

Інтерактивність є ключовим аспектом для підвищення зручності та ефективності використання інформаційної системи для прогнозування погоди. Розширення інтерактивних можливостей дозволить користувачам більш активно взаємодіяти із системою, отримувати більш персоналізовані та оперативні відповіді на свої запити. У цьому розділі розглянемо кілька напрямків для покращення інтерактивності системи.

Голосові Команди

Впровадження підтримки голосових команд є одним із найсучасніших і найбільш зручних способів взаємодії з інформаційною системою. Голосові асистенти, такі як Google Assistant, Amazon Alexa або Apple Siri, стають все більш популярними, і інтеграція з ними дозволить користувачам отримувати прогнози погоди, не використовуючи клавіатуру або екран. Це особливо корисно для водіїв, людей, що займаються спортом, або тих, хто зайнятий іншими справами і не може користуватися руками для введення даних.

Геолокаційні Запити

Автоматичне визначення локації користувача є важливою функцією, яка значно спрощує процес отримання прогнозу погоди. Замість того, щоб вручну вводити свою локацію, користувач може дозволити системі доступ до геолокації свого пристрою. Це дозволить автоматично надавати точні прогнози для поточного місцезнаходження користувача, що особливо зручно для людей, які часто подорожують або змінюють своє місце перебування.

Інтерактивна Карта Погоди

Додавання інтерактивної карти погоди, яка показуватиме поточні та прогнозовані погодні умови в режимі реального часу, значно покращить користувацький досвід. Користувачі зможуть переміщатися по карті, обирати різні регіони для перегляду локальних прогнозів, а також переглядати анімації змін погоди протягом дня або тижня. Така карта може включати шари з додатковою інформацією, такою як температура, опади, вітер та інші параметри.

Інтерактивні Сповіщення

Інтерактивні сповіщення можуть стати ще одним ефективним засобом покращення інтерактивності системи. Наприклад, користувачі можуть налаштувати сповіщення про зміни погоди або небезпечні метеорологічні умови, такі як шторми або сильні опади. Такі сповіщення можуть надходити у вигляді push-повідомлень на смартфон, електронних листів або SMS. Користувачі також можуть взаємодіяти зі сповіщеннями, наприклад, підтверджуючи їх отримання або запитуючи додаткову інформацію.

Інтеграція з Соціальними Мережами

Інтеграція з соціальними мережами, такими як Facebook, Twitter або Instagram, дозволить користувачам ділитися прогнозами погоди зі своїми друзями та підписниками. Це може бути корисним для інформування великої кількості людей про погодні умови, особливо у випадку надзвичайних ситуацій. Користувачі зможуть публікувати прогнози, сповіщення про зміни погоди або інші важливі повідомлення безпосередньо зі системи.

Інтерактивні Віджети

Впровадження інтерактивних віджетів, які можна розмістити на головному екрані смартфона або на робочому столі комп'ютера, дозволить користувачам швидко отримувати актуальну інформацію про погоду. Такі віджети можуть

показувати поточну температуру, прогноз на найближчі дні, а також інтерактивні елементи для оновлення даних або переходу до детальнішої інформації.

Персоналізовані Налаштування
Додавання можливості персоналізованих налаштувань дозволить користувачам налаштувати систему під свої індивідуальні потреби. Користувачі зможуть вибирати, які параметри погоди їх цікавлять, як часто вони хочуть отримувати оновлення, та в якому форматі. Це зробить систему більш гнучкою та зручною для різних категорій користувачів.

4.4 Оптимізація коду

Покращення продуктивності та стабільності системи є також важливим аспектом. Це можна досягнути за рахунок:

Оптимізації запитів до API: Зменшення кількості запитів до API шляхом кешування результатів та використання менш ресурсомістких методів.

Поліпшення обробки помилок: Додавання більш детальних повідомлень про помилки та можливостей для їх автоматичного виправлення.

Збільшення масштабованості: Забезпечення можливості обробки більших обсягів запитів при зростанні кількості користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи та за допомогою мови програмування Python створено інформаційну систему у вигляді телеграм-бота для отримання погодної інформації.

- Було обрано інструменти розробки: Python як мову програмування, PyCharm – середа розробки, PostgreSQL – СУБД для інформаційної системи.
- Розробка бази даних: Створено базу даних для зберігання інформації про місцезнаходження користувача
- Розробка структури бота: Розроблено зручну структуру телеграм-бота, включаючи створення меню для користувачів.
- Розробка алгоритму роботи бота: Розроблено алгоритм, який дозволяє користувачам запитувати погодну інформацію для конкретного місця та отримувати актуальні дані.
- Тестування бота: Проведено тестування телеграм-бота, він працює безперебійно та зручно для користувачів, і відповідає всім поставленим завданням.

В результаті користувачі можуть зручно та безперебійно отримувати погодну інформацію через інформаційну систему у вигляді телеграм-бота. Проект у майбутньому буде розширятися.

СПИСОК ЛІТЕРАТУРИ

1. Як створити телеграм бота через BotFather? [Електронний ресурс]. URL: https://www.https://ukr-bot.com/yak-stvoryty-telehram-bota-cherez-botfather/ (дата звернення: 22.04.2023).
2. BotFather. Можливості, команди та функціонал. [Електронний ресурс]. URL: https://gerabot.com/article/botfather_mozhlivosti_ta_funkcional (дата звернення: 22.04.2023).
3. All You Need to Know About Bot Frameworks: A Complete Guide. Maruti Techlabs [Електронний ресурс]. URL: https://marutitech.com/complete-guide-bot-frameworks/ (дата звернення: 23.04.2023).
4. Основи Microsoft Bot Framework. [Електронний ресурс]. URL: https://learn.microsoft.com/azure/bot-service/bot-builder-basics?view=azure-bot-service-4.0 (дата звернення: 23.04.2023).
5. Вибір бот-платформи: Chatfuel | Бізнес Майстерня. Бізнес Майстерня [Електронний ресурс]. URL: https://www.bizmaster.xyz/2020/02/vybir-bot-platformy-Chatfuel.html (дата звернення: 23.04.2023).
6. Введення в Dialogflow: налаштування та обмеження. *Розумні чат-боти | magdamagla* [Електронний ресурс]. URL: https://magdamagla.com/posts/dialogflow-settings/df-settings/ (дата звернення: 25.04.2024).
7. Недоліки Dialogflow. Складнощі при створенні чат-ботів. [Електронний ресурс]. URL: https://magdamagla.com/posts/dialogflow-cons/df-cons/ (дата звернення: 25.04.2024).
8. Вибір бот-платформи: ManyChat | Бізнес Майстерня. Бізнес Майстерня [Електронний ресурс]. URL: https://www.bizmaster.xyz/2020/02/vybir-bot-platformy-manychat.html (дата звернення: 25.04.2024).
9. Маничат vs Chatbot. [Електронний ресурс]. URL: https://ciroapp.com/uk/versus/manychat-vs-chatbot/ (дата звернення: 25.04.2024).

10. База даних (БД) – Що це таке? Визначення бази даних | Wiki HOSTiQ.ua. *HOSTiQ Wiki*. URL: https://hostiq.ua/wiki/ukr/database/ (дата звернення: 18.05.2024).
11. Postgresql що це і як застосовується в різних напрямках. FoxmindEd [Електронний ресурс]. URL: https://foxminded.ua/postgresql-shcho-tse/ (дата звернення: 18.05.2023).
12. Основи Pycharm. Уроки для початківців. W3Schools українською. W3Schools українською. Безплатні уроки онлайн для початківців, школярів та студентів [Електронний ресурс]. URL: https://w3schoolsua.github.io/hyperskill/pycharm-basics.html#gsc.tab=0 (дата звернення: 25.04.2024).
13. Кращі IDE для Python в 2023 році. [Електронний ресурс]. URL: https://mate.academy/blog/python/ide-for-python-2023/ (дата звернення: 25.04.2024).
14. Графічний клієнт pgAdmin [Електронний ресурс]. URL: https://krypton.com.ua/vvedenye-v-postgresql/grafichnyj-kliyent-pgadmin/ (дата звернення: 25.04.2024).
15. Що таке Use Case та для чого вони потрібні | Онлайн-курси від компанії QATestLab [Електронний ресурс]. URL: https://training.qatestlab.com/blog/technical-articles/what-is-a-use-case-and-what-are-they-for/ (дата звернення: 30.04.2024).

ДОДАТОК А

ЛІСТИНГ ІНФОРМАЦІЙНОЇ СИСТЕМИ

bot.py

```python
import asyncio
from aiogram import Bot, Dispatcher

from configparser import ConfigParser
from handlers import city_input, main_commands, weather

config = ConfigParser()
config.read('config.ini')
TOKEN = config.get('auth', 'TOKEN')

bot = Bot(token=TOKEN)

async def main():
    dp = Dispatcher()

    dp.include_routers(
        weather.router,
        city_input.router,
        main_commands.router
    )


    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)

if __name__ == '__main__':
    asyncio.run(main())
```

keyboard.py

```python
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup

buttons = [
  InlineKeyboardButton(text="1 day", callback_data='get_1_day_weather'),
  InlineKeyboardButton(text="2 days", callback_data='get_2_days_weather'),
  InlineKeyboardButton(text="3 days", callback_data='get_3_days_weather')
  ]

mainMenu = InlineKeyboardMarkup(inline_keyboard=[buttons])
```

states.py

```python
from aiogram.fsm.state import State, StatesGroup

class Location_states(StatesGroup):
    input_city = State()
    counry = State()
```

database.py

```python
from sqlalchemy import create_engine, Integer, Column, String, Float
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

import services.geocoding_API as geocoding_API

engine = create_engine('postgresql+psycopg2://postgres:12345@localhost:5432/postgres')
connection = engine.connect()

class UserNotFound(Exception):
    pass

Base = declarative_base()


class User(Base):
    __tablename__ = 'users'

    user_id = Column(Integer, primary_key=True)
    chat_id = Column(Integer)
    fk_location_id = Column(Integer)


class Location(Base):
    __tablename__ = 'locations'

    location_id = Column(Integer, primary_key=True)
    city = Column(String)
    latitude = Column(Float)
    longitude = Column(Float)


Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)


def set_location(chat_id: int, country: str, city: str):
    coord = geocoding_API.get_city_coords(country, city)
    session = Session()

    if (session.query(Location).filter(Location.latitude == coord['lat'], Location.longitude == coord['lng'])
            .first() is None):
        location = Location(city=city, latitude=coord['lat'], longitude=coord['lng'])
        session.add(location)
        session.commit()

    fk_location = session.query(Location).filter(Location.latitude == coord['lat'],
                                                 Location.longitude == coord['lng']).first()

    if session.query(User).filter(User.chat_id == chat_id).first() is None:
        user = User(chat_id=chat_id, fk_location_id=fk_location.location_id)
        session.add(user)
    else:
        a = session.query(User).filter(User.chat_id == chat_id).first()
        a.fk_location_id = fk_location.location_id

    session.commit()
```

```python
def get_location(chat_id: int) -> dict[str, float]:
    session = Session()
    loc = session.query(User).filter(User.chat_id == chat_id).first()

    if loc is None:
        return UserNotFound
    else:
        loc = session.query(Location).filter(Location.location_id == loc.fk_location_id).first()
        coord = {'lat': loc.latitude, 'lng': loc.longitude}
        return coord
```

geocoding_API.py

```python
from configparser import ConfigParser
import requests
import json

config = ConfigParser()
config.read('config.ini')
GEOCODING_API_KEY = config.get('auth', 'GEOCODING_API_KEY')
base_url = "https://graphhopper.com/api/1/geocode?locale=en&debug=true" + f"&key={GEOCODING_API_KEY}"


def get_city_coords(country: str, city: str) -> dict[str, float]:
    req = requests.get(base_url + f"&q={city}").text
    hits = json.loads(req)['hits']

    for item in hits:
        if 'country' in item:
            if item['country'] == country and item['name'] == city:
                point = item['point']
                coord = {'lat': point['lat'], 'lng': point['lng']}
                return coord


def get_country_list(city: str) -> dict[int, str]:
    req = requests.get(base_url + f"&q={city}")
    hits = json.loads(req.text)['hits']
    n = 1
    location_options = {}
    for item in hits:
        if 'country' in item:
            if item['country'] not in location_options.values():
                location_options[n] = f'{item["country"]}'
                n += 1

    return location_options
```

services → weather.py

```python
from configparser import ConfigParser
import requests
import json

config = ConfigParser()
config.read('config.ini')
GEOCODING_API_KEY = config.get('auth', 'GEOCODING_API_KEY')
base_url = "https://graphhopper.com/api/1/geocode?locale=en&debug=true" + f"&key={GEOCODING_API_KEY}"


def get_city_coords(country: str, city: str) -> dict[str, float]:
    req = requests.get(base_url + f"&q={city}").text
    hits = json.loads(req)['hits']

    for item in hits:
        if 'country' in item:
            if item['country'] == country and item['name'] == city:
                point = item['point']
                coord = {'lat': point['lat'], 'lng': point['lng']}
                return coord


def get_country_list(city: str) -> dict[int, str]:
    req = requests.get(base_url + f"&q={city}")
    hits = json.loads(req.text)['hits']
    n = 1
    location_options = {}
    for item in hits:
        if 'country' in item:
            if item['country'] not in location_options.values():
                location_options[n] = f'{item["country"]}'
                n += 1

    return location_options
```

city_input.py

```python
from aiogram import Router, F
from aiogram.types import Message
from aiogram.filters import Command
from aiogram.fsm.context import FSMContext
import re

from services.database import set_location
from keyboard import mainMenu

from states import Location_states
import services.geocoding_API as geocoding_API

router = Router()

@router.message(Command(commands=["set_location"]))
async def enter_location_config(message: Message, state: FSMContext):
  await message.answer("Enter your city")
  await state.set_state(Location_states.input_city)


@router.message(Location_states.input_city)
async def input_city(message: Message, state: FSMContext):
        city_name = message.text  # city name
        pattern = r'^[A-Za-z]+$'

        if re.match(pattern, city_name):
                await state.update_data(city=city_name)

                locations_dict = geocoding_API.get_country_list(city_name)
                await state.update_data(locations_dict=locations_dict)

                location_list_answer = 'Choose your county (number only):\n'
                for i in locations_dict:
                        location_list_answer += f'{i}.{locations_dict[i]}\n'

                await state.set_state(Location_states.counry)
                await message.answer(location_list_answer)
        else:
                await message.answer('City name must contain only english letters. Try again')


@router.message(Location_states.counry)
async def choose_counry(message: Message, state: FSMContext):
    location_number = message.text

    if location_number.isdigit() and int(location_number) > 0:
        data = await state.get_data()
        location_dict = data.get('locations_dict')
        city = data.get('city')

        set_location(
            chat_id=message.chat.id,
            country=location_dict[int(location_number)],
            city=city
    )

        await message.answer('Thank you! Now you can get weather forecast!', reply_markup=mainMenu)
        await state.clear()
    else:
            await message.answer('Choose number from list')
```

main_commands.py

```python
from aiogram.types import Message
from aiogram import Router
from aiogram.filters import Command
from keyboard import mainMenu

router = Router()

@router.message(Command(commands=["start", "help"]))
async def commands_start(message: Message):
    await message.reply("Hello, I'm weather bot. \n \n"
                        "At first you should set your location \n\n"
                        "/set_location")
@router.message(Command(commands=["menu"]))
async def send_menu(message: Message):
    await message.answer(text='Forecast for ', reply_markup=mainMenu)

@router.message()
async def command_nof_found(message: Message):
    await message.reply('Command not found!')

```

handlers → weather.py

```python
from aiogram import types
from aiogram import Router, F
from aiogram.types import CallbackQuery

from bot import bot
from keyboard import mainMenu


import services.weather as weather


router = Router()


@router.callback_query(F.data.startswith('get_1_day_weather'))
async def get_weather(call: CallbackQuery):
        await bot.delete_message(chat_id=call.from_user.id, message_id=call.message.message_id)
        await call.message.answer(text=f'{weather.weather_several_days(call.from_user.id, 1)}')
        await call.message.answer(text='Forecast for ', reply_markup=mainMenu)


@router.callback_query(F.data.startswith('get_2_days_weather'))
async def get_2_days_weather(call: CallbackQuery):
        await bot.delete_message(chat_id=call.from_user.id, message_id=call.message.message_id)
        await call.message.answer(text=f'{weather.weather_several_days(call.from_user.id, 2)}')
        await call.message.answer(text='Forecast for', reply_markup=mainMenu)


@router.callback_query(F.data.startswith('get_3_days_weather'))
async def get_3_days_weather(call: CallbackQuery):
    await bot.delete_message(chat_id=call.from_user.id, message_id=call.message.message_id)
    await call.message.answer(text=f'{weather.weather_several_days(call.from_user.id, 3)}')
    await call.message.answer(text='Forecast for', reply_markup=mainMenu)
```