Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка інтелектуального чат-бота для обробки запитів користувачів із використанням GPT та Python

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Федорків М. В.
(підпис) (прізвище та ініціали)

Керівник _____ Мудрик І. Я.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль 2025

2025

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Федорківу Максиму Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтелектуального чат-бота для обробки запитів користувачів із використанням GPT та Python

Керівник роботи Мудрик Іван Ярославович, док. Філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Потреба замовника, вимоги замовника.

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ

Аналіз предметної області та постановка задачі

Теоретичні основи NLP та обґрунтування вибору GPT

Реалізація Telegram-бота з підключенням до API та інтеграцією з базою даних

Безпека життєдіяльності основи охорони праці

Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

Схема архітектури Telegram-бота

Структура таблиць у Notion (скріншоти з поясненням)

Діаграма логіки обробки повідомлень користувача

Інтерфейс генерації ключів в OpenAI (скріншот)

Фрагменти коду (основні модулі)

Скріни прикладів роботи бота (запит, відповідь, запис)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Лазарюк В. В., доц. каф. МТ		
основи охорони праці			
Нормконтроль	Стоянов Ю. М., к. т. н.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	**Примітка**
1	Ознайомлення з завданням кваліфікаційної роботи		
2	Збір та аналіз інформації за темою дослідження		
3	Формування структури пояснювальної записки		
4	Розробка технічного завдання та вибір методів реалізації		
5	Реалізація чат-бота: архітектура, модулі, підключення API		
6	Тестування функціоналу чат-бота		
7	Написання розділів пояснювальної записки (1–3 розділи)		
8	Написання розділу 4: «Охорона праці та безпека життєдіяльності»		
9	Оформлення висновків, списку використаних джерел, додатків		
10	Перевірка роботи керівником, внесення правок		
11	Нормоконтроль		
12	Перевірка кваліфікаційної роботи на плагіат		
13	Попередній захист кваліфікаційної роботи		
14	Захист кваліфікаційної роботи		

Студент _____ *Федорків М. В..*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Мудрик І. Я.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Федорків М. В. Інтелектуальний чат-бот для обробки запитів користувачів із використанням GPT та Python, ТНТУ ім. Івана Пулюя, Тернопіль 2025.

Пояснювальна записка містить, 20 рисунків, 16 джерел та 2 додатки – загалом 68 сторінок.

Об'єкт дослідження: Процес розробки інтелектуального чат-бот для обробки запитів користувачів із використанням GPT та Python.

Мета: створити повнофункціональну систему, яка дозволяє автоматизувати процес взаємодії з клієнтами, зменшити навантаження на оператора та забезпечити зручний механізм бронювання місць.

Сучасні системи обміну повідомленнями, зокрема Telegram, активно використовуються не лише для приватного спілкування, а й як інструменти автоматизації бізнес-процесів. Одним із ключових напрямів такого використання є взаємодія користувачів із чат-ботами, здатними обробляти запити в реальному часі. Особливої актуальності набуває поєднання ботів з технологіями штучного інтелекту, які дозволяють зробити діалог максимально природним та ефективним. У межах цієї роботи розглянуто процес створення Telegram-бота з інтеграцією моделі GPT для консультування користувачів, а також реалізовано функціонал обробки записів на перевезення з використанням хмарної бази даних Notion.

Ключові слова: бот, чат-бот, штучний інтелект, Telegram-бот, GPT, OpenAI, Notion, обробка природної мови, перевезення, системи миттєвого обміну повідомленнями, запис клієнтів, логістика.

ABSTRACT

Fedorkiv M. V. Intelligent Chatbot for Processing User Requests Using GPT and Python, Ternopil Ivan Puluj National Technical University, Ternopil, 2025. The explanatory note contains 20 figures, 16 references and 2 appendices — a total of 68 pages.

Object of research: the process of developing an intelligent chatbot for processing user requests using GPT and Python.

Purpose: To develop a full-featured system that automates client interaction, reduces operator workload, and provides a convenient mechanism for booking seats. Modern messaging systems, particularly Telegram, are increasingly used not only for personal communication but also as tools for automating business processes. One of the key applications is user interaction with chatbots capable of processing queries in real time. The integration of chatbots with artificial intelligence technologies has become especially relevant, as it allows for natural and efficient dialogue. This work explores the development of a Telegram bot integrated with the GPT model for user consultation, as well as the implementation of a booking system using the cloud-based Notion database.

Keywords: bot, chatbot, artificial intelligence, Telegram bot, GPT, OpenAI, Notion, natural language processing, transportation, instant messaging systems, client booking, logistics.

ПЕРЕЛІК СКРОЧЕНЬ

ПЗ – Програмне забезпечення

AI – Штучний інтелект (від англ. Artificial Intelligence)

API – Інтерфейс прикладного програмування (Application Programming Interface)

GPT – Generative Pre-trained Transformer (Генеративний попередньо навчений трансформер)

NLP – Обробка природної мови (Natural Language Processing)

GUI – Графічний інтерфейс користувача (Graphical User Interface)

ID – Ідентифікатор

JSON – JavaScript Object Notation (формат обміну даними)

HTTP – Протокол передачі гіпертексту (HyperText Transfer Protocol)

URL – Уніфікований покажчик ресурсу (Uniform Resource Locator)

BOT – Автоматизований програмний агент (від англ. bot)

TG – Telegram (месенджер, платформа для реалізації бота)

OpenAI – Компанія-розробник моделей GPT

Notion – Хмарний сервіс для зберігання та обробки даних

.env – Файл середовища для зберігання конфіденційних змінних

CLI – Інтерфейс командного рядка (Command Line Interface)

CPU – Центральний процесор (Central Processing Unit)

UI – Інтерфейс користувача (User Interface)

UX – Досвід користувача (User Experience)

log – Журнал подій або помилок

ЗМІСТ

ВСТУП

З огляду на розвиток систем штучного інтелекту та зростаючу роль людино-машинної взаємодії, чат-боти стають ключовим елементом цифрових сервісів.

Сфера міжнародних пасажирських перевезень потребує постійного вдосконалення засобів комунікації між перевізником і клієнтом. Зі зростанням кількості поїздок між Україною та країнами Європейського Союзу зростає й обсяг запитів щодо маршрутів, графіків, вільних місць, умов поїздок. У більшості випадків такі звернення мають однотипний характер, що спричиняє перевантаження операторів та знижує ефективність обслуговування. Використання автоматизованих рішень, зокрема чат-ботів, дозволяє значно покращити якість взаємодії з клієнтами, пришвидшити обробку запитів і забезпечити цілодобову підтримку.

Telegram є однією з найзручніших і найпопулярніших платформ для створення чат-ботів завдяки відкритому API, широким можливостям інтеграції та підтримці сучасних підходів до обробки повідомлень. Платформа дозволяє реалізувати швидку й адаптивну взаємодію між користувачем і ботом у текстовому форматі без потреби у встановленні додаткового програмного забезпечення. У поєднанні з методами обробки природної мови такі боти здатні краще розуміти запити, адаптуватися до формулювань користувача, уточнювати деталі та формувати більш точну відповідь.

Розробка Telegram-бота для інформування пасажирів про перевезення в межах Європи є актуальним завданням як з практичної, так і з наукової точки зору. Практична значущість полягає в можливості підвищення ефективності роботи компанії, яка займається перевезеннями, зменшення навантаження на персонал і покращення клієнтського досвіду. З наукового боку, розглядається комплекс завдань, пов’язаних із програмною інженерією, включаючи проєктування архітектури програмного забезпечення, використання Telegram Bot API, інтеграцію з базами даних та побудову логіки діалогу на основі запитів природною мовою.

Основною метою роботи є створення чат-бота у месенджері Telegram для автоматизованого консультування клієнтів щодо перевезень Європою з можливістю перевірки доступних місць на конкретну дату та формування заявки.

Задля досягнення поставленої мети було проаналізовано предметну область, вивчено аналогічні рішення, сформовано вимоги до функціоналу, спроєктовано архітектуру програмного рішення, реалізовано базовий функціонал та проведено його тестування.

У центрі дослідження перебуває процес надання інформаційної підтримки клієнтам у сфері міжнародних перевезень, а предметом виступає Telegram-бот як інструмент автоматизації цієї взаємодії. У рамках реалізації проєкту використано методи аналізу предметної області, моделювання поведінки користувача, тестування програмного продукту, а також практичної інтеграції з базою даних, реалізованою на платформі Notion. Такий підхід дозволяє забезпечити динамічну перевірку доступних місць, додавання нових записів до бази та гнучку обробку запитів у режимі реального часу.

Результатом роботи стало програмне рішення, у якому реалізовано алгоритм розпізнавання дат, облік зайнятих місць, автоматизоване формування нових записів та інтеграцію з даними без необхідності ручного втручання. Рішення є адаптивним і придатним до масштабування, його функціонал може бути використаний у практичній діяльності перевізників, які працюють у сфері пасажирських перевезень між Україною та країнами Європи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика об’єкта

У процесі створення будь-якого програмного забезпечення надзвичайно важливим етапом є визначення вимог до системи. Саме на цій стадії формуються очікування кінцевих користувачів, визначаються функціональні та нефункціональні характеристики майбутнього продукту, а також здійснюється моделювання основних сценаріїв використання. Для розробки Telegram-бота, який автоматизує взаємодію з клієнтами у сфері міжнародних перевезень, цей етап відіграє ключову роль, оскільки забезпечує відповідність програмного рішення реальним потребам цільової аудиторії [5].

Загальний підхід до проєктування системи починається з аналізу поточної проблемної ситуації. У більшості випадків компанії, що займаються перевезеннями, здійснюють комунікацію з клієнтами через звичайні месенджери або телефонні дзвінки. Такий підхід не дозволяє масштабувати роботу при збільшенні кількості запитів, оскільки кожне звернення вимагає ручної обробки. Це створює затримки у відповіді, збільшує навантаження на персонал і знижує загальну якість обслуговування. Як наслідок, виникає необхідність у створенні програмного агента, здатного автоматично відповідати на типові запитання клієнтів, надавати інформацію про поїздки, перевіряти наявність вільних місць та здійснювати попередній запис на поїздку [6].

На основі цього аналізу було сформовано загальні вимоги до функціоналу чат-бота. Серед них: розуміння запитів користувача у довільній текстовій формі, автоматична перевірка кількості вільних місць на зазначену дату, взаємодія з базою даних Notion [4], можливість внесення нових записів до цієї бази, а також формування підтвердження для користувача. Крім того, враховується потреба в обробці різних варіацій запитів, що вимагає залучення елементів обробки природної мови [2]. Система повинна бути здатна інтерпретувати повідомлення на

кшталт «чи є місця на 10 червня?» або «можна записатись на 14 число?» навіть без чіткого формулювання [3].

До нефункціональних вимог, які також слід враховувати при проєктуванні, належать стабільність роботи бота, цілодобова доступність, безпека збереження даних, захист від спаму та непередбачуваних дій користувача, а також мінімальна затримка у відповідях. Бот має працювати в режимі реального часу, без потреби у втручанні адміністратора, крім випадків, що потребують ручної перевірки. Також важливо забезпечити можливість масштабування функціоналу — наприклад, додавання нагадувань про поїздку або інформування про зміни в розкладі [9].

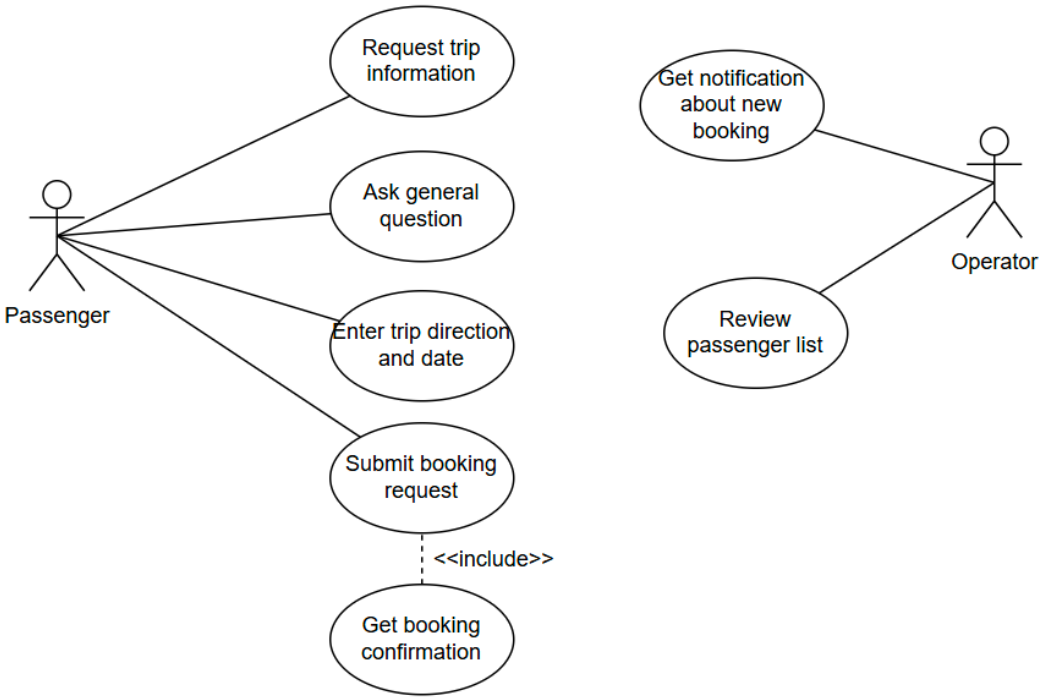

Рис. 1.1 – Діаграма варіантів використання Telegram-бота для перевезень.

Крім use-case, ще одним важливим елементом проєктування є побудова діаграми взаємозв’язків між компонентами системи, зокрема між Telegram API [3], обробником повідомлень, логікою обробки запитів, базою даних Notion [4] та іншими допоміжними модулями. Така архітектурна схема дозволяє візуалізувати внутрішню логіку системи, визначити ключові точки обміну даними та передбачити вузькі місця, які можуть впливати на продуктивність [6].

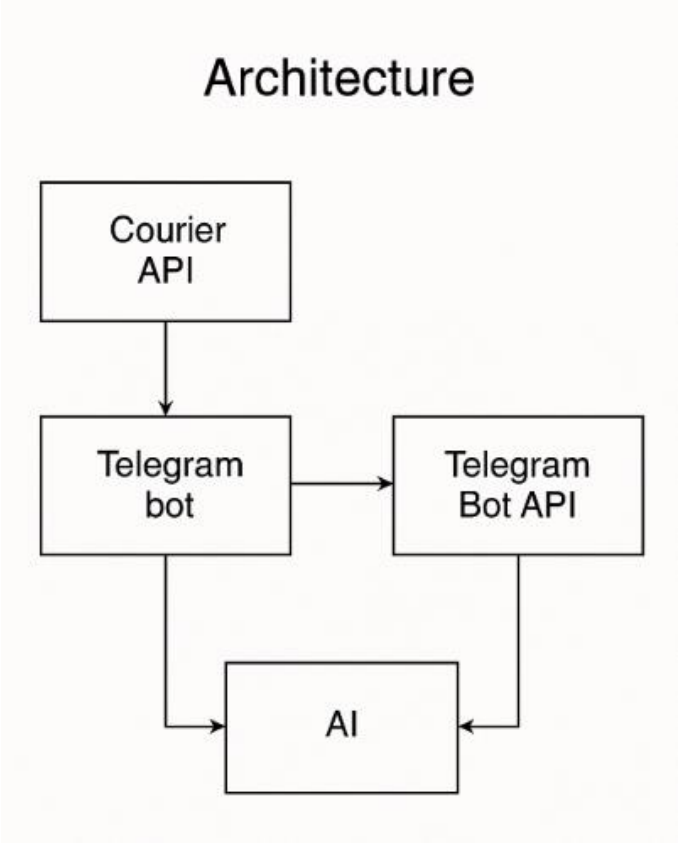

Рис 1.2 – Архітектурна схема Telegram-бота для перевезень.

Процес формування вимог тісно пов’язаний із вивченням цільової аудиторії. Основними користувачами чат-бота є клієнти, які планують міжнародну поїздку, мають доступ до Telegram, та хочуть оперативно отримати відповідь без необхідності телефонного дзвінка. Зазвичай це українські заробітчани, студенти або мандрівники, які зацікавлені в простому та зрозумілому інтерфейсі. Тому особливу увагу слід приділити зручності взаємодії, зрозумілим відповідям та мінімізації кількості кроків до отримання результату. Наприклад, після уточнення дати та кількості місць бот повинен одразу надати відповідь про доступність і запропонувати підтвердити запис у два кліки [3, 4].

Таким чином, у процесі проєктування Telegram-бота визначено ключові функціональні та нефункціональні вимоги, сформовано загальне уявлення про архітектуру системи, а також окреслено сценарії взаємодії користувача з ботом. Це створює основу для подальшого технічного проєктування, вибору технологій та реалізації конкретних модулів системи [5].

1.2 Аналіз існуючих технічних рішень у сфері чат-ботів

Сфера створення чат-ботів активно розвивається протягом останнього десятиліття. З моменту появи перших автоматизованих помічників, які відповідали на запити у формі шаблонних повідомлень, технології зробили значний прорив у напрямку адаптивної, контекстно залежної взаємодії. Чат-боти почали застосовуватись у багатьох сферах — від технічної підтримки в інтернет-магазинах до автоматизованих юридичних консультацій, фінансових сервісів і транспортних компаній [5]. Основна перевага чат-бота полягає у можливості заміни або доповнення живого оператора, заощаджуючи ресурси компанії та час користувача [6].

Аналіз сучасного ринку показує, що більшість реалізованих ботів базуються на простих скриптах зі строго заданими діалоговими гілками. Подібні рішення легко реалізовуються, мають передбачувану логіку, але не здатні ефективно працювати з неструктурованими повідомленнями чи новими формами запитів. Їх застосування обмежується лише тими випадками, де користувачі діють за шаблоном, що не завжди відповідає реальним умовам використання, особливо у транспортній галузі [6], де повідомлення можуть бути довільними за змістом і формулюванням.

Більш гнучкі чат-боти використовують зовнішні платформи з підтримкою штучного інтелекту. Одними з найвідоміших прикладів є сервіси Dialogflow від Google, Microsoft Bot Framework, IBM Watson Assistant, а також інтеграції з моделями GPT (Generative Pre-trained Transformer) [1, 2]. Ці інструменти забезпечують можливість розпізнавання наміру користувача (intent recognition), побудову діалогів на основі машинного навчання, а також адаптацію відповідей до контексту спілкування. Усе це дозволяє створювати більш «людяний» і розумний інтерфейс для взаємодії [2].

Серед найпоширеніших реалізацій у сфері логістики можна виділити Telegram-боти, що дозволяють бронювати доставку товарів, відстежувати статус

посилки або замовляти поїздки. Прикладом є боти для служб таксі (наприклад, Uklon Bot або Bolt Bot), які взаємодіють з користувачем у кілька кроків: отримують адресу, уточнюють деталі, надсилають підтвердження [3, 4]. Хоча такі боти використовують переважно фіксовану структуру діалогу, деякі з них інтегрують NLP-моделі для покращення взаємодії [2]. У закордонному досвіді поширеним є використання WhatsApp або Messenger-ботів у компаніях із вантажними або пасажирськими перевезеннями, зокрема в Іспанії, Німеччині та Польщі [6].

На основі аналізу можна зробити висновок, що ефективні чат-боти мають відповідати таким критеріям: мінімізація кроків взаємодії, адаптація до стилю спілкування користувача, висока швидкість відповіді, а також доступність 24/7. Важливо також забезпечити прозорість логіки обробки запиту — користувач повинен чітко розуміти, що від нього очікується, і як діяти далі [5].

У контексті розробки чат-бота для перевезень між країнами Європи необхідно врахувати не лише загальні принципи побудови ботів, а й специфіку сфери. Вона полягає у наявності розкладу поїздок, обмеженої кількості місць, частих запитів на конкретні дати та необхідності оперативно реагувати на бронювання [4]. Існуючі рішення рідко враховують цю специфіку в повному обсязі — більшість з них призначені для або внутрішньої логістики, або орієнтовані на іншомовні ринки. Отже, виникає потреба у створенні вузькоспеціалізованого рішення, адаптованого до української мови, реалій та очікувань користувачів, з можливістю розуміння довільних фраз, розпізнавання дат, перевірки вільних місць у базі та запису нового клієнта [2, 4].

Для візуального розуміння відмінностей між типами чат-ботів можна подати порівняльну таблицю, де зіставлено основні характеристики шаблонного бота, ботів на NLP-платформах і ботів із інтеграцією AI-моделей (наприклад, GPT) [1].

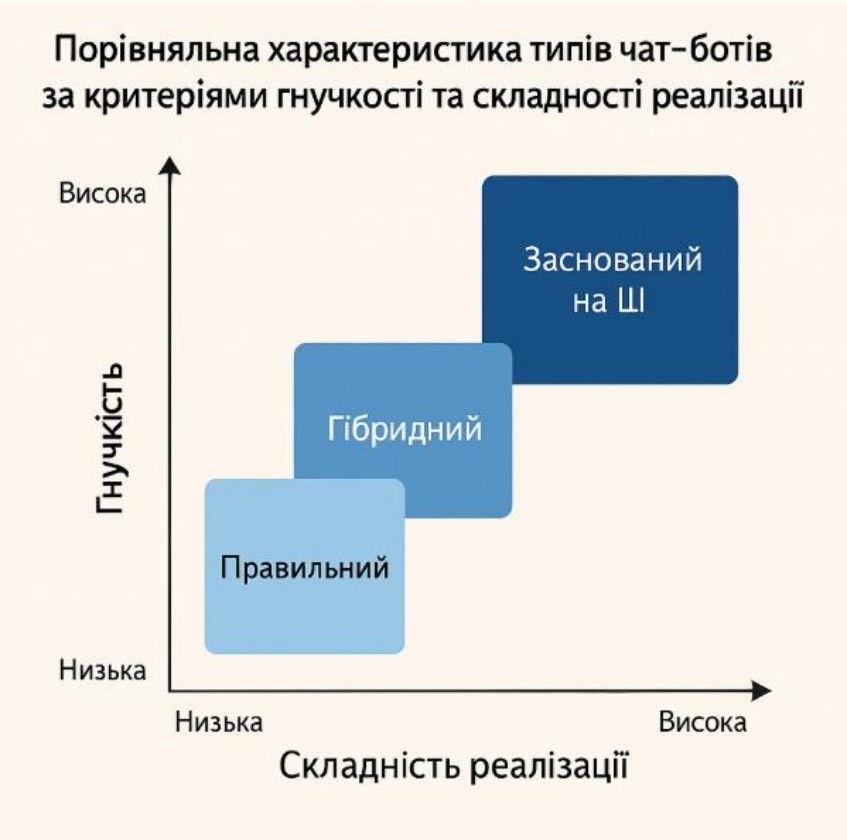

Рис. 1.3 – Порівняльна характеристика типів чат-ботів за критеріями гнучкості та складності реалізації.

Варто відзначити, що попри наявність великої кількості реалізацій чат-ботів, існує суттєвий розрив між універсальними платформами та рішеннями, спеціалізованими для конкретної бізнес-моделі перевезень. Саме це формує постановку науково-технічного завдання даної дипломної роботи — створення Telegram-бота з можливістю інтеграції до зовнішньої бази даних, підтримкою обробки природної мови та гнучкої логіки обробки запитів, з урахуванням українського контексту і потреб користувача [1, 2].

1.3 Стан проблеми та постановка задачі дослідження

Проблема ефективної комунікації між підприємствами, які надають послуги перевезення, та їхніми клієнтами стоїть досить гостро у сучасних умовах

цифровізації. Хоча більшість компаній уже перейшли до спілкування з користувачами через месенджери, такі як Telegram або Viber, сам процес відповіді на запити в багатьох випадках залишається ручним [3]. Це не лише знижує швидкість обслуговування, але й обмежує можливість масштабування, адже зі збільшенням кількості клієнтів різко зростає навантаження на операторів. Водночас частина запитів є повторюваною, типовою та піддається автоматизації.

Незважаючи на доступність інструментів для створення чат-ботів, у транспортній галузі все ще спостерігається недостатній рівень впровадження інтелектуальних рішень [4]. У багатьох компаніях, що надають послуги пасажирських перевезень між Україною та країнами Європи, запис клієнтів досі ведеться у вигляді повідомлень в месенджерах, а інформація щодо доступних місць перевіряється вручну. Це створює низку ризиків: втрату клієнтів через затримку у відповіді, помилки при бронюванні, неповну або застарілу інформацію щодо графіку руху чи завантаженості транспорту. Такий підхід є застарілим і суперечить сучасним трендам автоматизації обслуговування [4].

З боку користувача також існують певні труднощі. Зокрема, клієнти не завжди розуміють, які саме формулювання необхідно використовувати, щоб отримати відповідь, а у разі великого потоку запитів — можуть не дочекатися відповіді взагалі [2]. Бажання отримати просту та швидку відповідь через зручний інтерфейс, без очікування або дзвінків, є абсолютно природним у цифрову епоху. Відповідно, виникає необхідність у системі, яка могла б самостійно розпізнавати запити, розуміти значення повідомлення, визначати необхідні дії та швидко реагувати на них.

Серед додаткових викликів, які постають при реалізації такого рішення, варто згадати необхідність роботи з довільними фразами, різними форматами дат (наприклад, "на 15 число", "у п’ятницю", "цього тижня"), різними мовами повідомлень (українська, іноді російська або англійська), а також забезпечення коректної логіки перевірки зайнятих місць. Це вимагає застосування методів обробки природної мови (NLP), а також використання зовнішніх баз даних, які зберігатимуть актуальну інформацію у структурованому вигляді [1, 4]. У цьому

контексті інтеграція з платформою Notion як джерелом динамічних даних є практичним і гнучким рішенням [4].

На основі викладеного стану проблеми можна сформулювати науково-технічне завдання дипломної роботи. Його суть полягає у створенні Telegram-бота для автоматизації процесу консультування клієнтів транспортної компанії, що займається міжнародними перевезеннями. Бот повинен бути здатен обробляти повідомлення користувача у довільній формі, розпізнавати дати, перевіряти кількість вільних місць у базі даних на певну дату, а також записувати нових пасажирів до бази [2, 4]. Крім того, система має забезпечувати гнучкість розширення функціоналу, стабільну роботу, адаптивність до стилю повідомлень клієнтів і простоту у подальшому супроводі.

Це завдання передбачає комплексну реалізацію — від збору та аналізу вимог до розробки програмної архітектури, написання коду, тестування й оптимізації функціоналу. Водночас реалізація проєкту дозволяє внести конкретний внесок у вирішення проблеми цифровізації комунікації в галузі перевезень, створивши приклад функціонального, адаптивного та практично орієнтованого програмного рішення [2, 4].

1.4 Етапи розвитку ШІ в NLP

Щоб зрозуміти технічну основу майбутнього рішення, проаналізуємо розвиток підходів до обробки мови в межах штучного інтелекту.

Історія розвитку штучного інтелекту тісно пов’язана з прагненням людства створити машини, здатні розуміти і генерувати природну мову [2]. Проблематика обробки текстової інформації, інтерпретації людських висловлювань та побудови логіки діалогу почала активно вивчатися ще в середині XX століття. З того часу відбулося кілька ключових етапів, які визначили напрям подальшого розвитку інтелектуальних систем, зокрема у сфері Natural Language Processing (NLP) [7].

Перший етап розвитку ШІ в обробці мови охоплює період 1950–1970-х років, коли домінували так звані «rule-based» або системи на основі правил. Вони ґрунтувались на заздалегідь визначених граматичних конструкціях, словниках та логічних операторах. Одним із перших прикладів такої системи стала ELIZA – чат-бот, створений у 1966 році Джозефом Вейценбаумом, який імітував поведінку психотерапевта [7]. Вона не розуміла змісту фраз, але змогла створити ілюзію спілкування завдяки набору шаблонів. Аналогічні системи використовувались для перекладу текстів, аналізу синтаксису та створення діалогових моделей, однак їхні можливості були дуже обмеженими через відсутність адаптивності.

Наступним важливим етапом став розвиток статистичних методів у 1980–1990-х роках. Завдяки зростанню обчислювальної потужності стало можливим обробляти великі текстові корпуси для виведення закономірностей. Підхід, відомий як Statistical NLP, дозволив системам «вивчати» мову на основі реальних даних, а не лише правил [2]. Методики на кшталт n-грамного моделювання та байєсівської класифікації суттєво покращили якість автоматичного перекладу, пошуку інформації та розпізнавання тексту. Хоча статистичні моделі були точнішими за rule-based, вони все ще мали обмеження у розумінні контексту.

На початку 2000-х років з’явився гібридний підхід, який поєднував статистичні моделі з частково ручним налаштуванням. Це дало змогу створювати більш точні системи, особливо у вузькоспеціалізованих галузях, таких як медичний аналіз тексту чи юридична лінгвістика. Проте справжній прорив настав у другій половині 2010-х років із приходом нейронних мереж, що дали старт сучасній епосі глибокого навчання [1, 2].

Станом на сьогодні, найпотужніші системи NLP базуються на глибоких нейронних мережах, зокрема трансформерах. Моделі, такі як BERT (від Google), GPT (від OpenAI), RoBERTa, T5 та інші, дозволяють не лише генерувати зв’язні тексти, а й аналізувати наміри, враховувати контекст попередніх реплік, будувати діалоги, перекладати, резюмувати та класифікувати повідомлення з високим рівнем точності [2, 6, 7]. GPT, зокрема, став прикладом того, як великі мовні моделі (LLM

– Large Language Models) можуть взаємодіяти з користувачем у реальному часі, підтримуючи діалог, виконуючи команди та навіть генеруючи програмний код.

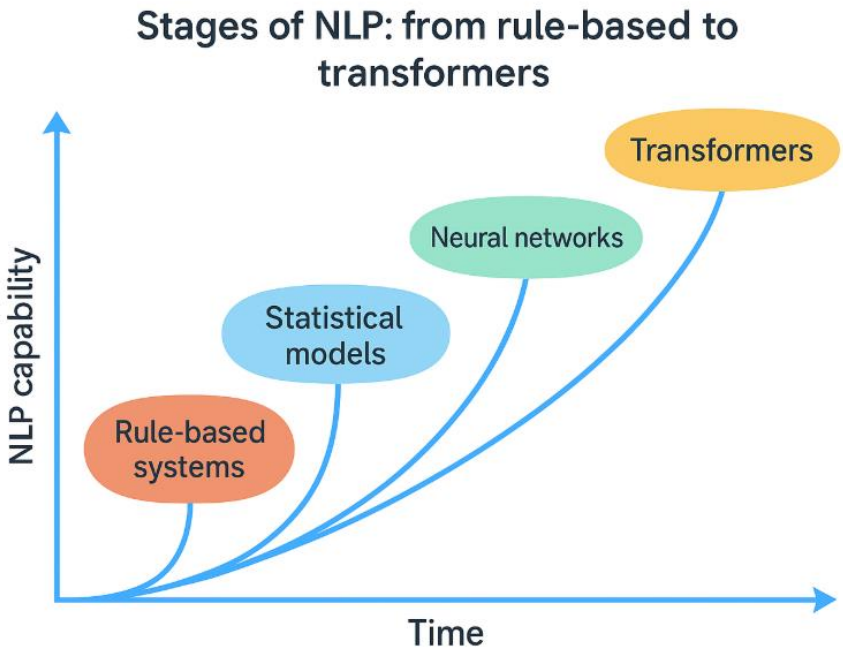

Рис. 1.4 – Етапи розвитку NLP: від rule-based систем до трансформерів.

Кожен етап розвитку ШІ в NLP відзначається поступовим переходом від ручного налаштування до самонавчання та генеративних можливостей. Якщо на початкових етапах системи могли лише наслідувати мовні патерни, то сучасні моделі здатні навчатися на мільярдах слів і будувати складні логічні зв’язки між частинами тексту [6]. Це відкриває нові горизонти для практичного використання таких моделей, зокрема у створенні адаптивних чат-ботів, які можуть самостійно аналізувати запит, визначати його мету та формувати відповідь у зрозумілій формі.

Використання таких моделей у поєднанні з платформами обміну повідомленнями, як-от Telegram, дозволяє створювати інтелектуальні рішення, що змінюють підхід до обслуговування клієнтів [1, 2]. Саме тому в рамках цієї дипломної роботи розглядається можливість інтеграції таких технологій у процеси, пов’язані з наданням інформаційних послуг у сфері пасажирських перевезень

1.5 Підходи до NLP

Обробка природної мови (NLP) є однією з найскладніших та водночас найперспективніших галузей штучного інтелекту. На різних етапах розвитку цієї галузі використовувалися різні підходи до побудови моделей, кожен із яких мав свої переваги, недоліки та специфіку застосування. Перехід від rule-based систем до трансформерів є не лише технологічною еволюцією, але й переходом до якісно нового рівня взаємодії між людиною і машиною [1].

Моделі на основі правил (rule-based systems) будуються на заздалегідь визначених шаблонах, правилах і словниках. Вони є передбачуваними, простими у реалізації та легкими в обслуговуванні [2]. Зазвичай такі системи включають аналіз синтаксису, морфології, визначення частин мови та побудову фразових структур. Незважаючи на свою простоту, rule-based моделі не здатні опрацьовувати великі обсяги даних або навчатися з прикладів, а також не враховують контекст, що значно обмежує їхнє застосування в умовах реального спілкування з користувачем.

Наступним кроком стали статистичні моделі (statistical models), які базуються на обробці великих текстових корпусів. Вони використовують імовірнісні методи для передбачення наступного слова, побудови фраз, виявлення співвідношення між словами. Одним із поширених підходів є використання n-грам — послідовностей із n слів, на основі яких обчислюється ймовірність появи наступного слова [4]. Хоча статистичні моделі дозволили зробити значний прорив у якості перекладу, класифікації тексту та синтезі мови, вони все ще не мали механізмів глибокого розуміння або семантики.

Поява штучних нейронних мереж (neural networks) значно розширила можливості NLP. Такі моделі здатні вивчати складні залежності між словами, будувати векторні представлення (embeddings) слів, враховувати контекст і навіть навчатися на непозначених даних [5]. Моделі на основі RNN (Recurrent Neural Networks) і LSTM (Long Short-Term Memory) змогли розширити горизонт пам’яті, що дало змогу обробляти цілі речення та абзаци. Проте ці мережі мали обмеження

в паралельній обробці даних і довготривалих залежностях, що зумовило потребу в нових підходах.

Справжню революцію спричинила поява трансформерів (transformers), які вперше були описані в статті “Attention is All You Need” (2017) [6]. Ці моделі замінили рекурентну обробку на механізм уваги (self-attention), що дозволяє паралельно аналізувати всі слова у тексті, визначати важливість кожного слова у контексті, а також будувати глобальні взаємозв’язки. На базі трансформерів з’явилися моделі нового покоління — GPT, BERT, T5, які демонструють видатні результати у генерації, класифікації, перекладі, аналізі настроїв, пошуку інформації та побудові діалогів [7].

Comparison of NLP Models

RULE-BASED	STATISTICAL	NEURAL	TRANSFORMER
ADAPTIVITY	No	Yes	Yes
NEED FOR DATA	Low	Moderate	High
PROCESSING SPEED	Fast	Medium	Slow
CONTEXT UNDERSTANDING	None	Moderate	High
SCALABILITY	Low	Moderate	High

Рис. 1.5 – Порівняння моделей NLP: rule-based, statistical, neural, transformer

Особливістю трансформерів є те, що вони не тільки дозволяють аналізувати вхідний текст, а й здатні генерувати зв’язні логічні тексти у відповідь. Саме це робить їх ідеальними для створення інтелектуальних чат-ботів, які повинні не лише

відповідати на типові питання, але й розуміти запити в довільній формі, адаптуватися до стилю користувача, підтримувати діалог і формувати рекомендації. Більш того, сучасні трансформери легко адаптуються до нових доменів завдяки донавчанню на спеціалізованих даних або використанню prompt-інженерії без модифікації самої моделі [6].

Таким чином, розуміння еволюції NLP-моделей дозволяє обґрунтовано вибрати саме трансформерну архітектуру (зокрема GPT) як базу для інтелектуальної взаємодії у Telegram-боті, орієнтованому на обробку текстових запитів у сфері перевезень [1].

1.6 Архітектура GPT та її сильні сторони

Моделі GPT (Generative Pre-trained Transformer) є представниками великомасштабних трансформерних архітектур, що стали проривом у сфері обробки природної мови [6]. Архітектура GPT базується на механізмі самоуваги (self-attention), який дозволяє моделі враховувати всі слова у вхідній послідовності одночасно, визначаючи, які з них є найбільш важливими для поточного прогнозу. Це значно підвищує точність моделі, дозволяючи краще розуміти контекст запиту [7].

Однією з ключових особливостей GPT є двоетапний підхід до навчання: спочатку модель проходить етап попереднього навчання (pre-training) на великих обсягах текстових даних, а потім – донавчання (fine-tuning) або використання через prompt-інженерію для виконання конкретних завдань [7]. Такий підхід дозволяє моделі не лише засвоїти граматику і структуру мови, а й опанувати закономірності побудови логічних і змістовних зв’язків у тексті.

Архітектура GPT складається з великої кількості шарів трансформера, кожен з яких має механізм самоуваги, нормалізації, активаційної функції та механізм feed-forward (прямого проходження сигналу) [6]. Вхідні токени (слова, розбиті на

частини) кодуються у вигляді векторів, які зберігають як значення, так і позицію слів у реченні. Потім ці вектори передаються через послідовні шари трансформера, де формується контекстне представлення кожного токена.

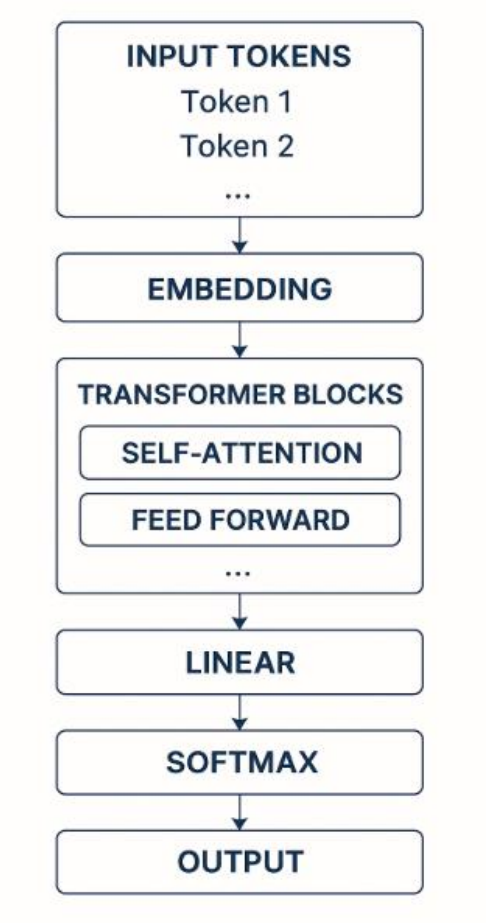

Рис. 1.6 – Спрощена архітектура GPT-моделі з основними компонентами: embedding, self-attention, feed-forward layers.

На практиці GPT використовується як мовна модель, яка на основі вхідного тексту генерує продовження з високою мовною природністю [6]. У контексті чат-ботів це дозволяє моделі не просто надавати шаблонні відповіді, а формувати осмислені тексти, які враховують контекст розмови. GPT здатна розпізнавати наміри користувача, відповідати на запитання, переказувати інформацію, аналізувати емоційне забарвлення повідомлення та навіть підтримувати довгі діалоги [7].

Застосування GPT у чат-ботах має низку переваг:

- Модель здатна працювати з довільними формулюваннями, без потреби в суворій структурі запиту.
- Забезпечується природна генерація мови, що створює враження спілкування з живою людиною.
- GPT адаптується до стилю спілкування користувача, покращуючи досвід взаємодії.
- Можливість масштабування на різні мови, тематики й сценарії без переписування логіки [6]

Завдяки цим характеристикам GPT-моделі є придатними для інтеграції в системи автоматизованої взаємодії з користувачем, зокрема в месенджерах, таких як Telegram, де важливо підтримувати гнучку, адаптивну та контекстно чутливу комунікацію

1.7 Обґрунтування вибору GPT для проєкту

Вибір моделі обробки природної мови є ключовим етапом при створенні чат-бота, особливо у випадках, коли взаємодія з користувачем повинна бути адаптивною, контекстною та наближеною до природного діалогу [6]. У попередніх підпунктах було розглянуто основні покоління NLP-моделей: rule-based системи, статистичні підходи, нейронні мережі та трансформери [16]. Кожен із цих типів моделей має свої переваги й недоліки, що доцільно враховувати при виборі інструменту для практичного впровадження.

Rule-based системи мають найменшу складність реалізації та високу швидкість роботи, проте повністю позбавлені гнучкості [7]. Вони здатні обробляти лише заздалегідь відомі шаблони, що робить їх непридатними для відкритих діалогів з користувачами, які формулюють запити у довільній формі. Такі рішення могли б підійти для внутрішніх службових ботів із фіксованим набором команд, але не для проєкту, що орієнтований на реальних клієнтів транспортної компанії.

Статистичні моделі є більш універсальними, проте також не враховують повноцінного контексту і не дозволяють будувати багатокрокову логіку спілкування. Вони підходять для задач класифікації чи фільтрації, але недостатньо потужні для генерації осмислених діалогів [7].

Нейронні мережі (зокрема, LSTM або GRU) дозволяють враховувати контекст, проте мають обмеження при роботі з довгими послідовностями, а також не завжди дають стабільні результати без складної підготовки даних і навчання [7]. Крім того, їхня реалізація вимагає великих ресурсів і значного часу на підготовку.

Трансформери, а саме GPT-моделі, забезпечують найвищу якість генерації тексту та розуміння запитів користувача [6]. Вони дозволяють будувати відповіді з урахуванням усієї історії діалогу, підтримують роботу з різними мовами, легко масштабуються та не потребують повного перенавчання для кожного нового застосування. Замість цього достатньо скористатися правильно сформульованими запитами (prompt engineering), що значно пришвидшує впровадження таких моделей у практичну систему.

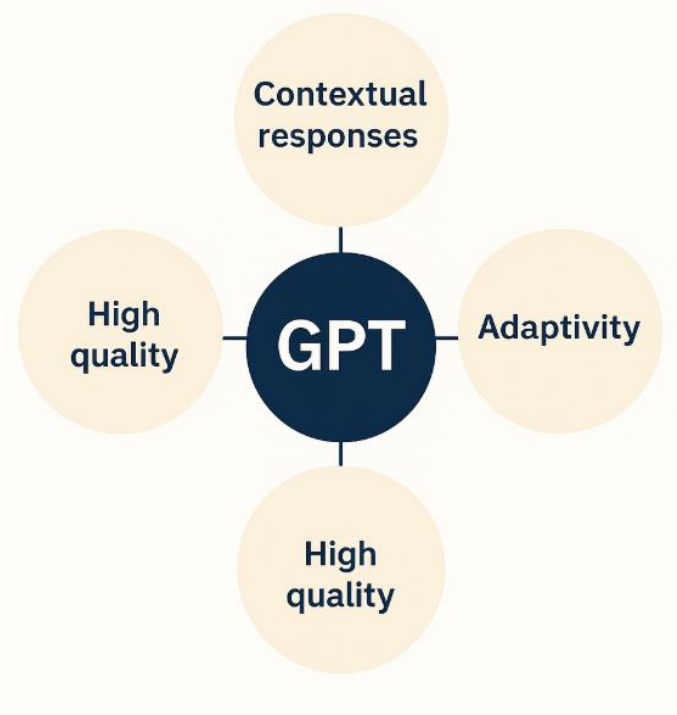

Рис. 2.4 – Вибір GPT-моделі як найкращого варіанту для розумної взаємодії чат-бота з користувачем

2. ІНТЕЛЕКТУАЛЬНІ ТЕХНОЛОГІЇ ТА ПРОЄКТУВАННЯ СИСТЕМИ

В останнє десятиліття спостерігається стрімке зростання інтересу до використання інтелектуальних технологій у різних галузях діяльності, зокрема в автоматизації обробки природної мови. Розробка програмних агентів, здатних вести осмислений діалог із користувачем, аналізувати зміст повідомлень, адаптувати відповіді до контексту — це вже не лише дослідницька область, а й практична реальність сучасного бізнесу [2, 7]. Особливо яскраво це проявляється у сфері обслуговування, техпідтримки, електронної комерції та транспорту, де швидкість та якість відповіді відіграють критичну роль.

Традиційні підходи до створення чат-ботів базуються на простій логіці правил: якщо повідомлення містить певне слово або фразу, виконується відповідна дія. Хоча подібні рішення є легкими у реалізації, вони не дозволяють повноцінно працювати з варіативністю людської мови [1, 7], де одна й та сама суть може бути передана десятками різних способів. У зв’язку з цим у розробці чат-ботів все частіше використовуються методи штучного інтелекту, зокрема штучні нейронні мережі, які дозволяють здійснювати семантичний аналіз, класифікацію намірів (intent classification), генерацію відповідей та навіть ведення повноцінного діалогу [2].

Завдяки розвитку обчислювальних потужностей та появі відкритих моделей, таких як GPT, стало можливим використання високоточних систем генерації мови навіть у невеликих застосунках. Платформи на кшталт OpenAI, Hugging Face, Google Cloud NLP, Amazon Comprehend зробили передові алгоритми доступними для інтеграції через API, що значно знизило вхідний поріг для впровадження ШІ у повсякденні розробки [2, 6]. Це дозволяє розробникам зосередитися на логіці інтеграції та адаптації моделі до конкретної задачі, а не на складному процесі навчання нейронної мережі з нуля.

2.1 Архітектура Telegram-бота

Функціонування Telegram-бота у межах дипломного проєкту базується на послідовній взаємодії між кількома ключовими компонентами. З метою ілюстрації цієї взаємодії нижче наведено діаграму послідовностей, яка демонструє процес обробки типового запиту користувача — від надсилання повідомлення до запису в базу даних і відповіді. Бот працює за наступним принципом:

1. Користувач надсилає повідомлення у Telegram.
2. Telegram API перенаправляє це повідомлення на сервер, де запущено Python-скрипт бота.
3. Python-бот (основний файл, наприклад main.py) передає повідомлення до модуля message_handler.py, у якому реалізовано логіку обробки.
4. У межах обробки перевіряється поточний етап (stage) взаємодії:
   - якщо потрібна дата — GPT використовується для розпізнавання запиту,
   - якщо вже відомі дані — виконується запит до бази даних у Notion.
5. Для аналізу повідомлення та формулювання відповіді або уточнення викликається GPT-модель через OpenAI API.
6. У випадку, якщо бот розпізнає, що користувач бажає записатися, здійснюється:
   - перевірка зайнятих місць у таблиці Notion,
   - додавання нового запису з відповідними позначками (наприклад, символами 🟢 або 🟣).
7. Після успішного запису користувач отримує підтвердження, а оператор інформується через окремий канал (наприклад, окреме повідомлення або запис у базі)

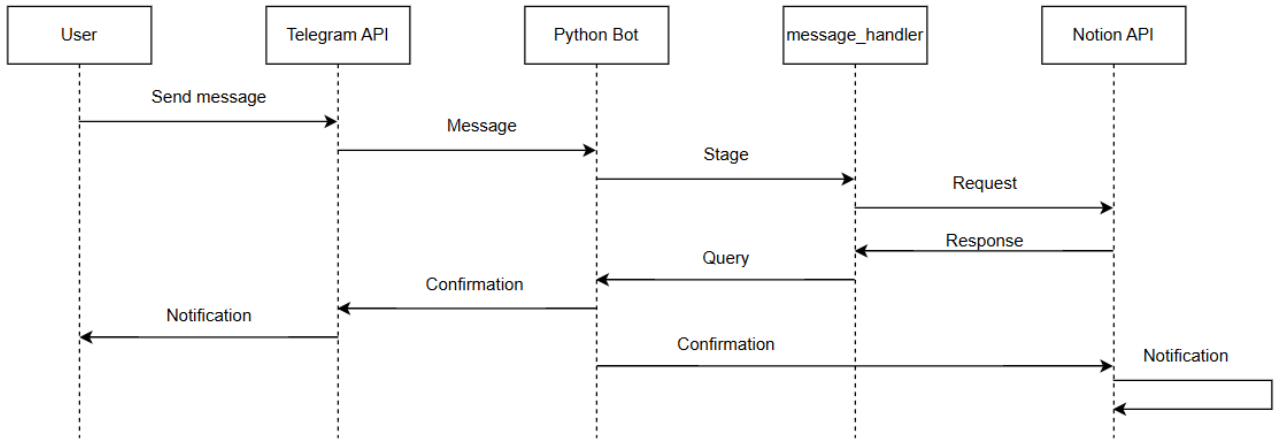

Рис. 2.1 – Діаграма послідовності обробки запиту (User – інтерфейс користувача)

Компоненти, що беруть участь у взаємодії:

- Користувач — надсилає повідомлення до бота у Telegram.
- Telegram API — передає повідомлення боту.
- Python-бот — основна програма, що отримує і маршрутизує повідомлення.
- message_handler.py — обробляє логіку сценаріїв взаємодії.
- GPT-модель — розпізнає природну мову, формулює відповіді.
- Notion база даних — зберігає дані про поїздки, пасажирів, вільні місця та надсилає повідомлення операторам.

2.2 Логіка обробки запитів і pipeline

У процесі взаємодії користувача з Telegram-ботом основну роль відіграє поетапна логіка, реалізована як pipeline всередині модуля message_handler.py. Уся обробка повідомлень побудована на відстеженні поточного стану користувача, що зберігається в контексті як значення поля stage. Це дозволяє послідовно керувати кожним етапом діалогу — від первинного звернення до остаточного запису в базу даних.

Після надсилання повідомлення користувачем, воно надходить через Telegram API до Python-бота, який викликає відповідну функцію обробки. Залежно від того, який етап зафіксовано в контексті, бот визначає, яку інформацію слід отримати далі. Якщо користувач тільки почав взаємодію, бот уточнює напрям поїздки. Наступним кроком, як правило, є розпізнавання дати — саме тут залучається GPT-модель. Модель отримує повідомлення користувача і повертає структуровану дату, навіть якщо вона була вказана у вільній формі — наприклад, «у п’ятницю» або «на наступний тиждень». Це дозволяє ботові працювати з природною мовою без складних регулярних виразів чи ручної перевірки.

Після успішного визначення дати бот переходить до етапу уточнення кількості місць. Весь час обробки зберігається контекст сесії, тому користувачеві не потрібно повторювати вже введену інформацію. Коли всі необхідні дані зібрано, бот ініціює взаємодію з базою даних, яка розташована в Notion. Через офіційний API здійснюється вибірка записів за вказаною датою, а також зчитуються назви вже існуючих записів, у яких зазначені зайняті місця. На основі цієї інформації бот ухвалює рішення про можливість запису нового пасажира.

У разі, якщо відповідна кількість місць доступна, бот формує новий запис у таблиці, додаючи в заголовок ім’я користувача разом із кількістю місць та спеціальною міткою, яка вказує, що запис створено ботом. Користувач одразу отримує підтвердження у Telegram, а оператору надходить повідомлення або відповідна позначка в базі. Завдяки такій поетапній логіці вдалося забезпечити контрольовану, стабільну та інтуїтивно зрозумілу взаємодію, у якій GPT використовується лише тоді, коли справді необхідно — для розпізнавання природної мови

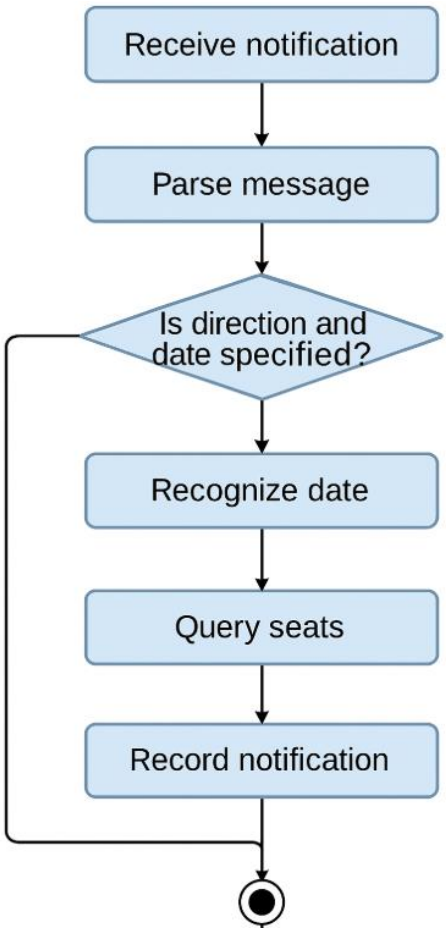

Рис 2.2 – Діаграма активності сценарію "Запит на бронювання місця"

Таким чином, логіка pipeline побудована так, що кожен наступний крок стає доступним лише після завершення попереднього. Це дозволяє виключити помилки користувача та забезпечити контроль за цілісністю запису.

2.3 Застосування GPT у реалізації

У межах реалізації дипломного проєкту GPT використовується як інструмент для обробки запитів природною мовою, що надходять від користувачів через Telegram. Основна мета інтеграції моделі полягає в тому, щоб розпізнавати контекст і смисл повідомлень, які не мають строгої структури, та перетворювати їх у формат, з яким може працювати програмна логіка бота.

Одним із ключових сценаріїв застосування GPT є розпізнавання дати поїздки. Користувач може вказати дату у довільній формі — наприклад, “у п’ятницю”, “на наступний тиждень” або “15 червня”. Модель інтерпретує таке повідомлення і повертає у відповідь стандартну дату у форматі YYYY-MM-DD, яку вже використовує Python-скрипт для запису або перевірки місць у базі. Це дозволяє досягти високої гнучкості без потреби реалізовувати вручну складні регулярні вирази або розгалуження в коді.

Крім цього, GPT бере участь у формуванні уточнень, якщо повідомлення користувача є неповним, суперечливим або незрозумілим. Наприклад, якщо користувач вказав дату, але не уточнив напрям або кількість місць, бот за допомогою моделі формулює уточнювальне запитання у зрозумілій формі. Також GPT використовується на етапі формування остаточного підтвердження — модель генерує повідомлення на зразок “Вас успішно записано на 2 місця на 14 червня з Бельгії до України”.

Окремо варто виділити ще одну важливу функцію — надання консультаційних відповідей. Якщо повідомлення користувача не стосується безпосередньо запису, а містить загальні питання про перевезення (наприклад: “Коли наступна поїздка з Німеччини?”, “Чи можна взяти тварину?” або “Скільки коштує квиток?”), бот звертається до GPT з іншим, спеціально підготовленим prompt-ом. Цей prompt зберігається у вигляді текстового файлу, який завантажується під час ініціалізації, і містить базову інформацію про правила перевезення, країни, графік руху, дозволені обмеження тощо. Таким чином, GPT виконує роль вбудованого консультанта, який формує інформативну та коректну відповідь на запитання, не потребуючи звернення до оператора.

У випадках, коли запит не підпадає під жоден з очікуваних шаблонів, GPT генерує ввічливу відповідь із проханням уточнити зміст або нагадує про основне призначення бота.

Завдяки такому підходу модель GPT не є лише допоміжним інструментом для аналізу дат, а стає повноцінним логічним компонентом, що забезпечує

“розумну” взаємодію з користувачем. Це дозволяє не лише оптимізувати сценарії запису, а й підвищити якість обслуговування через автоматизовані консультації.

2.4 Інтеграція з базою Notion

У межах реалізації Telegram-бота для автоматизації запису пасажирів основною системою зберігання даних обрано платформу Notion, яка надає зручний API для інтеграції з зовнішніми застосунками. Уся інформація про майбутні поїздки зберігається у вигляді таблиць — окремо для кожного напрямку: з Бельгії в Україну та з України в Бельгію. Кожна така таблиця містить стандартний набір полів, серед яких: ім’я або позначка пасажира (у заголовку), дата поїздки, адреса посадки, кількість місць, а також коментарі й номери телефонів. Саме ці таблиці використовуються як цільовий об’єкт для зчитування і запису даних.

Інтеграція реалізована за допомогою офіційної бібліотеки notion-client, яка дозволяє виконувати операції читання і створення сторінок у базі даних. При запиті користувача на запис, бот спочатку звертається до відповідної таблиці, фільтруючи записи за вказаною датою. Потім виконується зчитування заголовків усіх сторінок, що відповідають цій даті. В заголовках, згідно з домовленістю, вказуються вже зайняті місця (наприклад: “🟢 Іван (1 місце)”). Бот аналізує цю інформацію, підсумовує кількість зайнятих місць і порівнює її з максимально можливою для конкретної поїздки. Якщо вільні місця ще залишаються, формується новий запис.

Створення нового запису відбувається шляхом генерації JSON-структури із заповненими полями, після чого вона надсилається у базу через API. У заголовку нового запису вказується ім’я користувача разом із кількістю місць, а також додається спеціальний смайлик, який позначає, що запис зроблено саме ботом. Наприклад, символ 🟣 може слугувати для таких записів, тоді як 🟢 зарезервований для ручного введення. Це дозволяє оператору швидко відрізнити автоматичні бронювання від тих, що були внесені вручну.

Завдяки такій моделі інтеграції бот не лише зчитує структуру таблиці, а й динамічно працює з нею, забезпечуючи реальний облік місць та відсутність дублювань. Усі зміни у базі фіксуються миттєво, що дає змогу оператору бачити актуальну інформацію в реальному часі, не вдаючись до зовнішніх інструментів адміністрування.

Notion виступає не просто як база даних, а як повноцінна система управління бронюванням, з якою Telegram-бот веде двосторонній обмін — і для перевірки, і для внесення нових даних.

2.5 Середовище запуску і тестування

Розроблений Telegram-бот функціонує у локальному середовищі, що дозволяє швидко тестувати зміни та контролювати роботу всіх компонентів без потреби в зовнішньому хостингу чи серверних CI/CD-інструментах. Серверна частина реалізована на мові Python версії 3.x. Запуск бота здійснюється безпосередньо з командного рядка за допомогою інструкції python bot.py або python3 main.py, залежно від системного налаштування інтерпретатора.

До проєкту підключено кілька зовнішніх бібліотек, серед яких telebot для взаємодії з Telegram API, openai для підключення до GPT-моделі, а також notion_client для роботи з базою даних у Notion. Усі залежності визначено в окремому файлі requirements.txt, що забезпечує можливість швидкого розгортання проєкту на іншому пристрої або середовищі.

Тестування проводилося вручну, у режимі діалогу з ботом, де на кожному етапі перевірялося правильне реагування системи на дії користувача: від розпізнавання запиту до перевірки дати, перевірки зайнятих місць і створення нового запису. Тестові сценарії охоплювали як стандартні ситуації, так і граничні випадки — зокрема, неправильний формат дати, відсутність доступних місць, дублювання записів, а також запити, які не стосуються основної функції бота.

Тестування проводилося на двох операційних системах — Windows 10 та Ubuntu 22.04 LTS — що дало змогу переконатися у стабільності та кросплатформеності рішення. Підключення до GPT-моделі та API Notion здійснювалося через інтернет-з’єднання із захищеним зберіганням ключів у змінних середовища (файл .env), які автоматично зчитуються під час запуску.

Оскільки бот не передбачає складної інфраструктури, розгортання на хмарних сервісах або використання Docker наразі не впроваджувалося. Це рішення обґрунтовано обмеженням на обсяг завдання та характером внутрішнього використання системи. Проте, за потреби, проєкт може бути швидко адаптований під хмарне середовище або розгорнутий у вигляді вебсервісу з webhook-підключенням.

3. РЕАЛІЗАЦІЯ TELEGRAM-БОТА ДЛЯ КОНСУЛЬТУВАННЯ З ПЕРЕВЕЗЕНЬ

3.1 Створення Telegram-бота за допомогою BotFather

BotFather — це офіційний бот у Telegram, який дозволяє створювати нових ботів, керувати ними, змінювати опис, команду /start, зображення профілю, а також отримувати токен доступу до API [3]:

Покроковий процес створення:

1) Відкривається чат із ботом BotFather у Telegram (`@BotFather`).
2) Вводиться команда `/newbot` — команда для створення нового бота.
3) Далі BotFather пропонує ввести ім’я бота, яке буде відображатись у профілі (може містити пробіли, кирилицю, емодзі).
4) Після цього треба ввести юзернейм бота, який обов’язково має бути унікальним і завершуватись на `bot`, наприклад `PerevezennyaBot`.
5) У відповідь BotFather надсилає токен авторизації — це унікальний рядок, який використовується для підключення до Telegram Bot API. Його потрібно зберегти в `.env` файлі, щоб не допустити витоку.

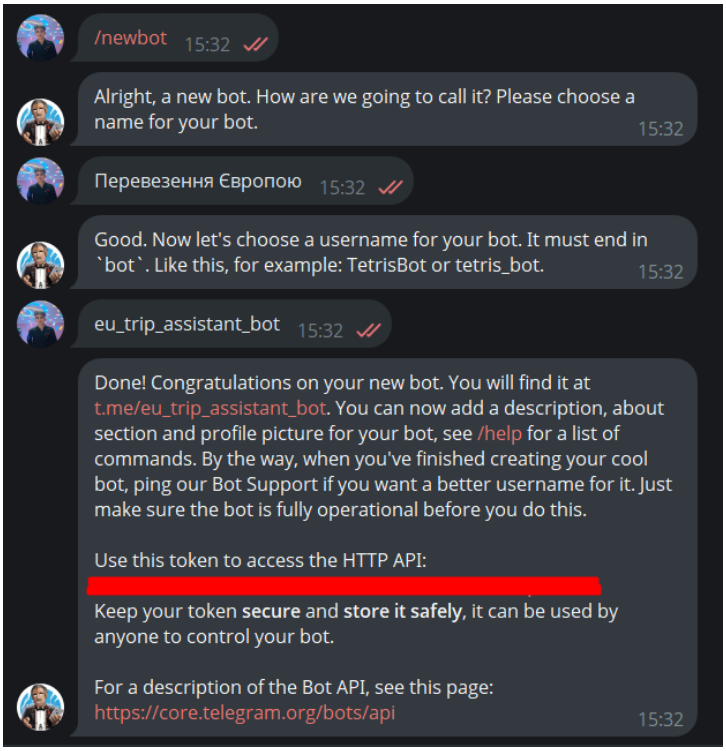

Рис. 3.1 – Діалог створення нового бота через BotFather

Додаткові команди BotFather, що були використані:

- /setdescription — описує, для чого призначений бот.
- /setabouttext — короткий текст, який відображається в профілі бота.
- /setuserpic — встановлення аватарки бота.
- /setcommands — список команд, які доступні користувачу, наприклад:
  - /start — запуск бота;
  - /help — інструкція;
  - /checkseats — перевірка місць;
  - /book — записатися.

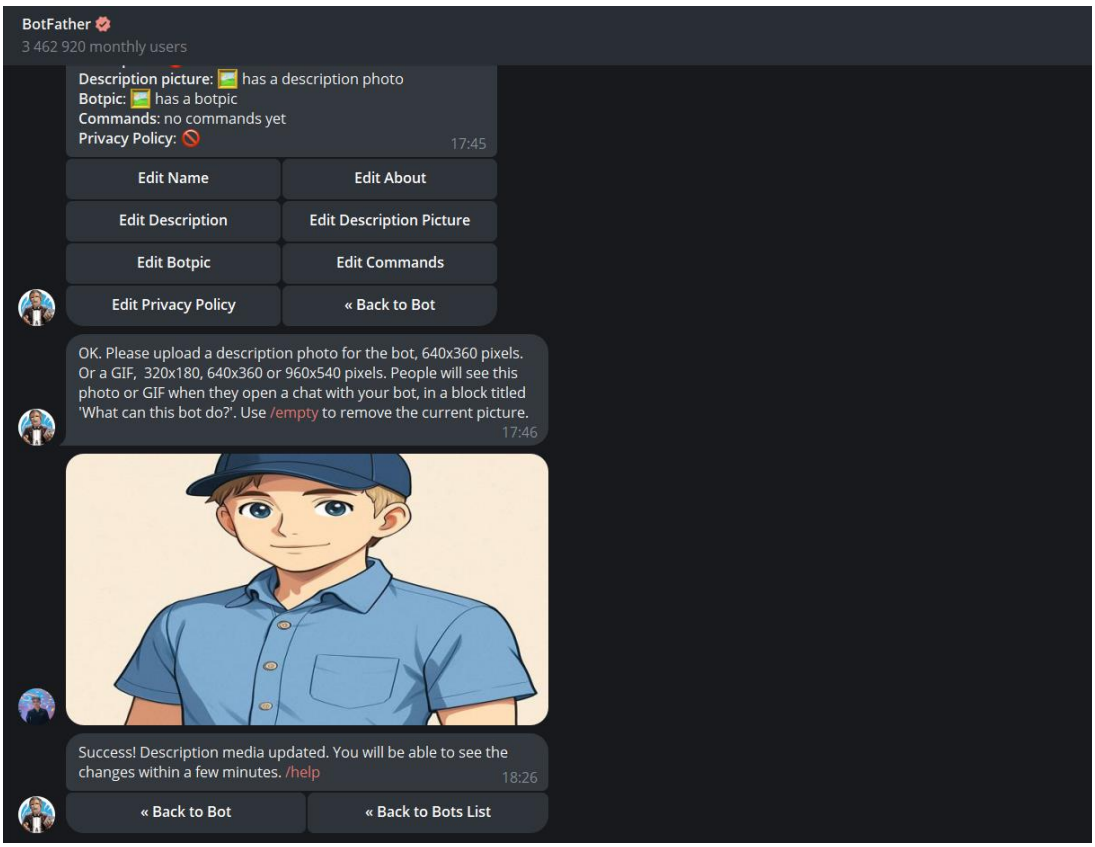

Рис. 3.2 – Приклад використання команд BotFather для додавання зображення бота.

На цьому етапі видно процес створення Telegram-бота — від команди /newbot до отримання токена авторизації. Вказані поля заповнювались згідно з логікою іменування проєкту та його призначення. Отриманий токен надалі використовувався у коді бота для з'єднання з Telegram API [8].

3.2 Організація баз даних у Notion

Telegram-бот працює у тісній зв'язці з двома базами даних, створеними у сервісі **Notion**, який обрано завдяки зручному API та можливості зберігати структуровану інформацію в табличному вигляді [8].

Створення таблиць:

1) У особистому робочому просторі Notion створено дві бази даних**:**

- «Україна → Бельгія»
- «Бельгія → Україна»

2) Для кожної таблиці додано поля:

- Ім’я пасажира (разом із номером місця),
- Дата виїзду,
- Адреса посадки,
- Два номери телефону,
- Коментарі / опис,
- Позначка про обробку оператором.

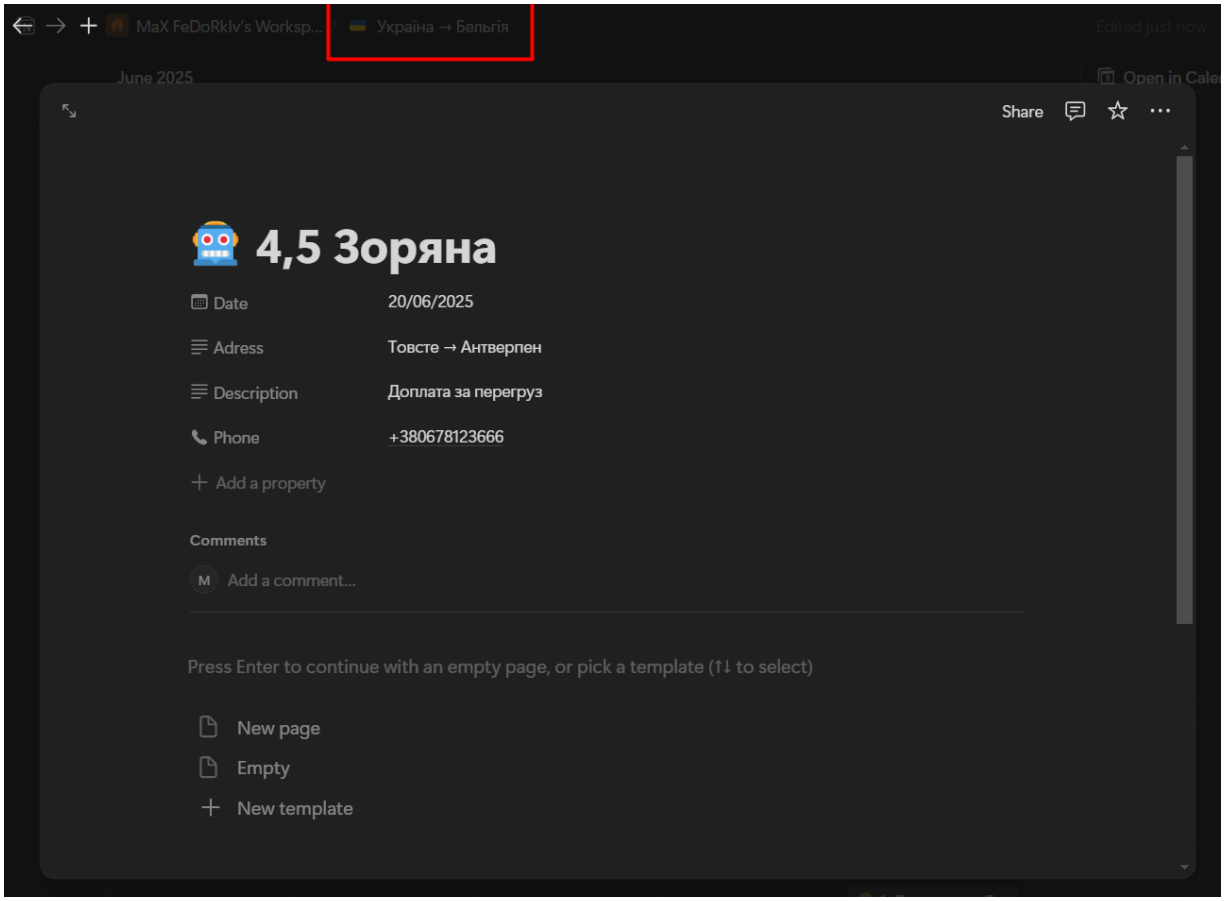

Рис. 3.3 – Структура бази даних у Notion для напрямку "Україна → Бельгія"

Важливе:

- Назва поля з номером місця була особливо важливою: саме з неї бот зчитує зайняті місця (наприклад, «Ім’я: Зоряна 🟢 4,5 (два місця)»).

- Для кожної таблиці в Notion отримано Database ID, який необхідний для API-запитів.
- Для доступу до API створено секретний інтеграційний токен в Notion, який має доступ лише до цих двох таблиць.

Для реалізації бази даних було обрано сервіс Notion завдяки його простоті, гнучкості та підтримці офіційного API. Однією з головних переваг стала можливість працювати з табличним форматом — це забезпечує зручне відображення записів, як для бота, так і для операторів [8]. Важливо також, що перевізники можуть у будь-який момент самостійно вносити або редагувати записи вручну, не вдаючись до програмування чи складних технічних засобів. Таким чином, Notion став оптимальним вибором для гібридного підходу: автоматизації через API та ручного керування.

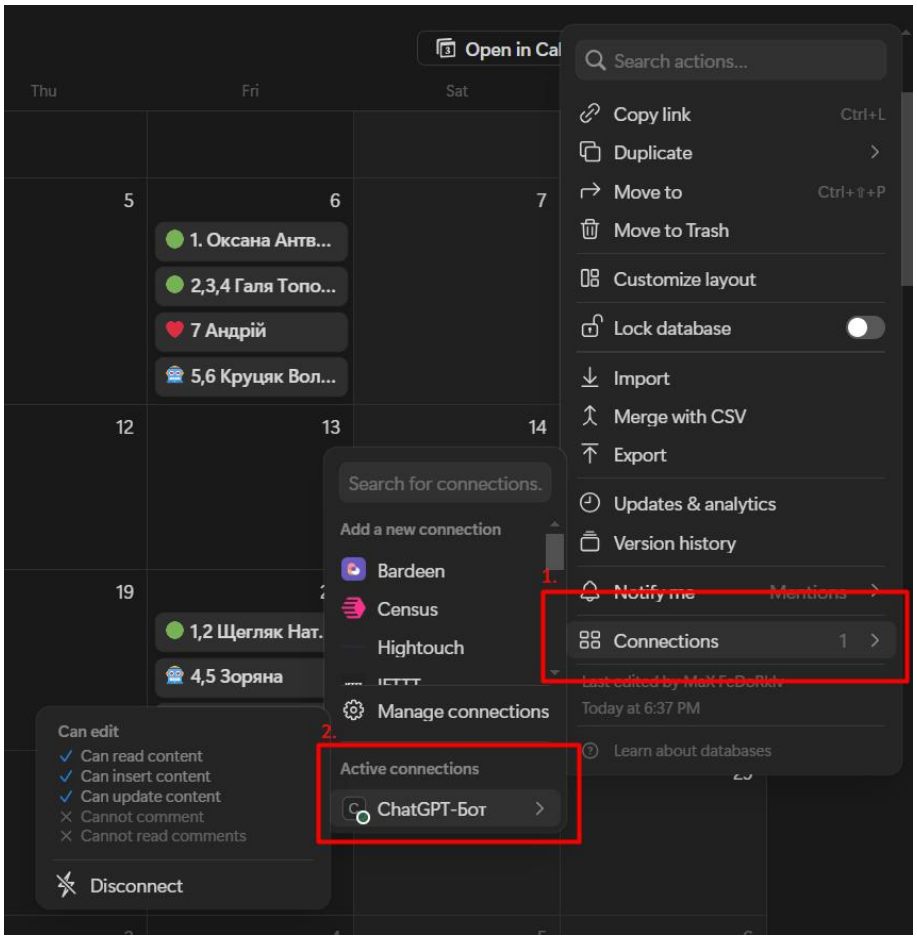

Рис. 3.4 – Приклад інтеграції до таблиці у Notion через API токен

Такий підхід до інтеграції дозволив забезпечити надійне з’єднання між Telegram-ботом і базою даних. Обмеження доступу лише до необхідних таблиць не тільки підвищує безпеку системи, а й спрощує її супровід. Усі запити виконуються через офіційне REST API Notion із зазначенням токена в заголовку Authorization, що відповідає сучасним вимогам безпечного обміну даними.

3.3 Підключення OpenAI API

Щоб бот міг не лише відповідати на команди, а й надавати живі консультації, було вирішено підключити модель штучного інтелекту на базі GPT від компанії OpenAI [5]. Цей інструмент забезпечує гнучке розуміння намірів користувача, дозволяє давати повні відповіді на запити, а також адаптується до нових ситуацій [5].

Завдяки інтеграції з GPT-моделлю реалізовано динамічне розпізнавання наміру користувача, що є одним з ключових аспектів людино-машинної взаємодії [14].

Причина вибору OpenAI: OpenAI API, зокрема модель GPT-3.5, було обрано через її здатність ефективно обробляти природну мову, генерувати логічні відповіді та підтримувати контекст у діалозі. Порівняно з іншими рішеннями, GPT-модель потребує мінімальної адаптації, що дозволило швидко інтегрувати її у вже існуючий проєкт. Завдяки цьому бот отримав можливість вести напівінтелектуальний діалог — не просто виконувати команди, а розуміти запитання у довільній формі, наприклад: *«Яка ціна поїздки в Німеччину наступної п’ятниці?»* або *«Коли наступний виїзд з Бельгії?»*

Покрокове підключення:

1) Перехід на офіційний сайт https://platform.openai.com, де після створення облікового запису було оформлено підписку на тарифний план Pay-As-You-Go
2) В особистому кабінеті згенеровано API ключ, який було скопійовано у .env файл проєкту у форматі: `OPENAI_API_KEY=sk-...`
3) У Python-проєкті було створено модуль openai_api.py, де здійснюється відправка тексту користувача до API, обробка відповіді та повернення її до Telegram-бота.

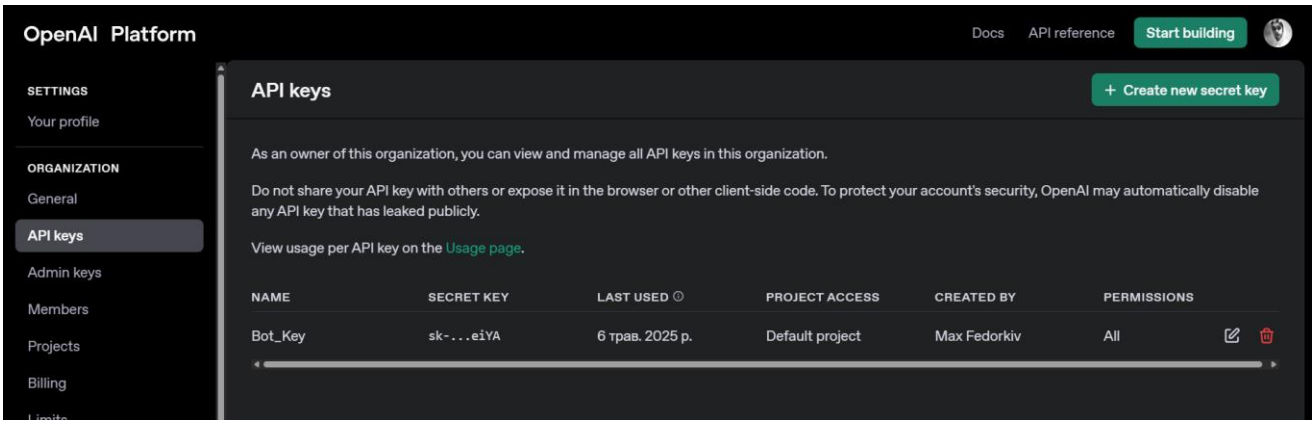

Рис. 3.5 – Інтерфейс генерації API-ключа на сайті OpenAI

Щоб уникнути ризику витоку ключа, змінні середовища .env не включаються у репозиторій, а файл openai_api.py використовує бібліотеку dotenv для безпечного читання токена.

Для запитів до OpenAI використовується метод chat/completions, де параметри включають:

- модель (gpt-3.5-turbo);
- історію повідомлень (messages);
- температуру генерації (temperature);
- кількість варіантів (n).

Формат відповіді налаштовується так, щоб відповідь бота була короткою, інформативною, а не надмірно творчою, адже мета — оперативна допомога пасажирам у стилі «запитання – відповідь» [5].

```python
import openai
import re
from config import OPENAI_API_KEY

openai.api_key = OPENAI_API_KEY

def get_chatgpt_response(user_message):
    with open("data/faq_context.txt", "r", encoding="utf-8") as f:
        system_context = f.read()

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",  #gpt-4-turbo
        messages=[
            {
                "role": "system",
                "content": system_context
            },
            {
                "role": "user",
                "content": user_message
            }
        ],
        max_tokens=500,
        temperature=0.5
    )


```

Рис. 3.6 – Кодова частина: приклад виклику OpenAI API з обробкою відповіді

Інтеграція GPT-моделі значно розширила функціональність Telegram-бота, зробивши його зручним помічником, який може не лише відповідати за шаблоном, а й пояснювати, уточнювати або задавати зустрічні питання. Такий рівень взаємодії формує відчуття, ніби користувач спілкується не з машиною, а з живим оператором.

3.4 Інтеграція компонентів і запуск логіки

Після створення Telegram-бота, організації баз даних у Notion та налаштування GPT API, було розпочато об’єднання всіх складових у функціональну систему [6]. Основна мета цього етапу — створення логіки обробки запитів, яка дозволяє боту реагувати на повідомлення користувача, визначати його намір, відповідно консультувати, перевіряти місця або здійснювати запис.

Архітектура Telegram-бота побудована на модулях, кожен з яких відповідає за свою ділянку роботи:

- message_handler.py — головний обробник усіх повідомлень від користувача;
- openai_api.py — модуль для взаємодії з GPT через OpenAI API;
- notion_api.py — запити до таблиць Notion: читання, додавання записів, перевірка зайнятих місць;
- intent_detector.py — визначення наміру користувача (наприклад, «записатися», «перевірити місця», «задати питання»);
- parse_date_from_text.py — розпізнавання дати з тексту типу «на 14 червня»;
- main.py — точка входу, яка ініціалізує запуск бота.

Усі ключі та конфіденційні дані зберігаються в .env файлі, а для доступу до них використовується бібліотека dotenv.

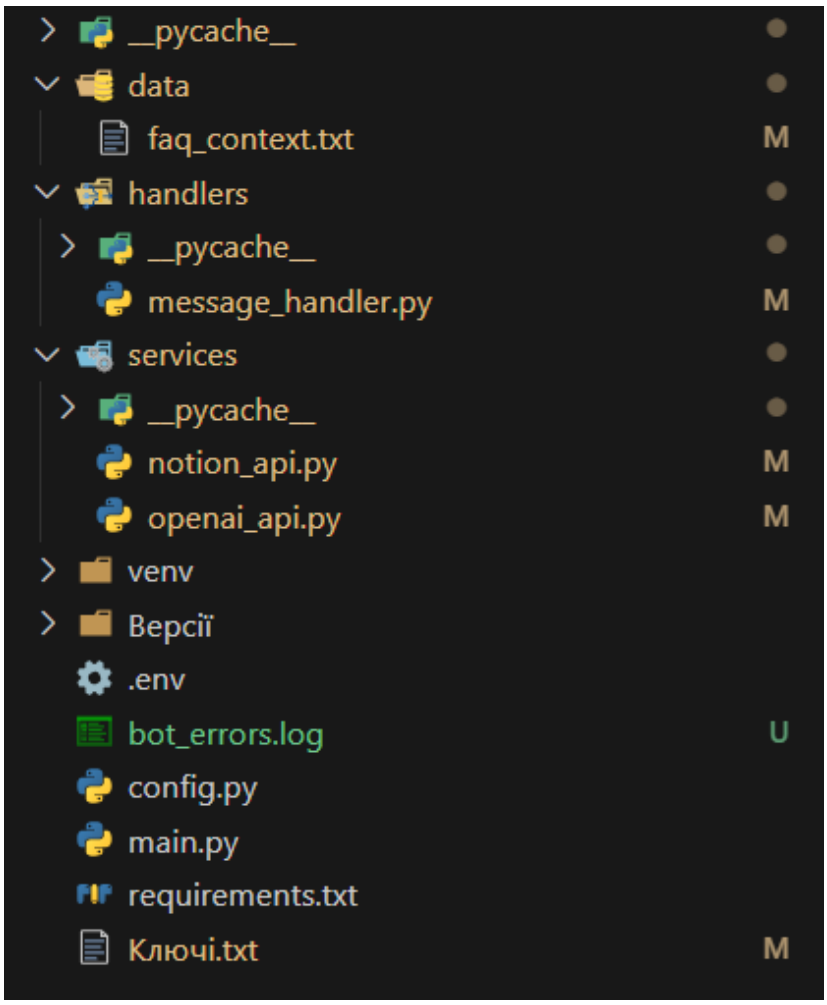

Рис. 3.7 – Структура файлів Telegram-бота у середовищі розробки.

Логіка обробки повідомлень наступна: Після отримання нового повідомлення бот передає його у модуль intent_detector.py, який визначає намір користувача — чи це питання, запит на бронювання або перевірка місць. Якщо повідомлення є загальним питанням, воно надсилається до GPT-модуля, який формує логічну відповідь у природній формі. У випадках, коли користувач хоче перевірити наявність місць або записатися, бот за допомогою модуля parse_date_from_text.py намагається розпізнати дату у тексті повідомлення. Якщо напрямок поїздки ще не уточнено, бот ставить додаткове запитання. Після цього викликається функція get_taken_seats_for_date() з модуля notion_api.py, яка повертає зайняті місця на обрану дату. У разі наявності вільних місць бот інформує користувача про кількість доступних та пропонує записатися, формуючи відповідний запис у базі даних.

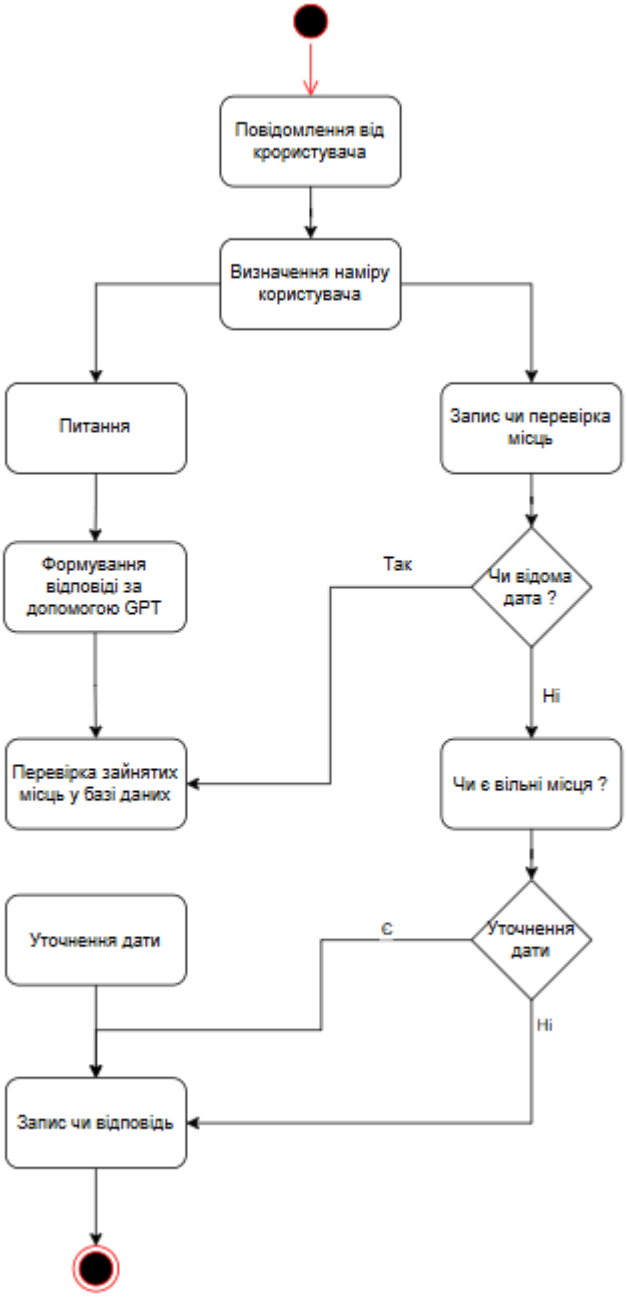

Рис. 3.8 – Діаграма діяльності схеми логіки обробки повідомлень користувача

Для запуску бота використовується Python-скрипт main.py, який ініціалізує токен Telegram, завантажує змінні середовища та запускає циклічну обробку подій [6]. Проєкт планується розгорнути на віддаленому сервері з постійною доступністю, що дозволить боту працювати 24/7.

3.5 Тестування та перевірка працездатності системи

Після завершення основних етапів розробки Telegram-бота було проведено ретельне тестування всіх ключових компонентів [9]: логіки діалогу, взаємодії з GPT, доступу до бази даних Notion, обробки запитів на запис та перевірку доступності місць. Метою тестування було переконатися, що система працює стабільно в різних ситуаціях і здатна правильно реагувати як на очікувані, так і на нетипові дії користувача.

Проведення тестування

Тестування охоплювало як ручну перевірку основної логіки, так і моделювання типових сценаріїв користувачів. Зокрема, перевірялись такі функції:

- генерація відповідей GPT на загальні запитання;
- уточнення напрямку поїздки;
- перевірка вільних місць на певну дату;
- запис нового користувача;
- обробка випадків, коли всі місця зайняті;
- ситуації з незрозумілими або помилковими запитами;
- реакція на несправність API або помилки у ключах;

Під час перевірки всі дії супроводжувались журналюванням — при виникненні виключень, помилки автоматично фіксувались у спеціальному лог-файлі bot_errors.log.

Приклад 1. Генерація відповіді GPT: Користувач написав повідомлення у довільній формі: *«Коли наступна поїздка?»*. Бот на основі GPT-моделі визначив зміст питання і сформував відповідь.

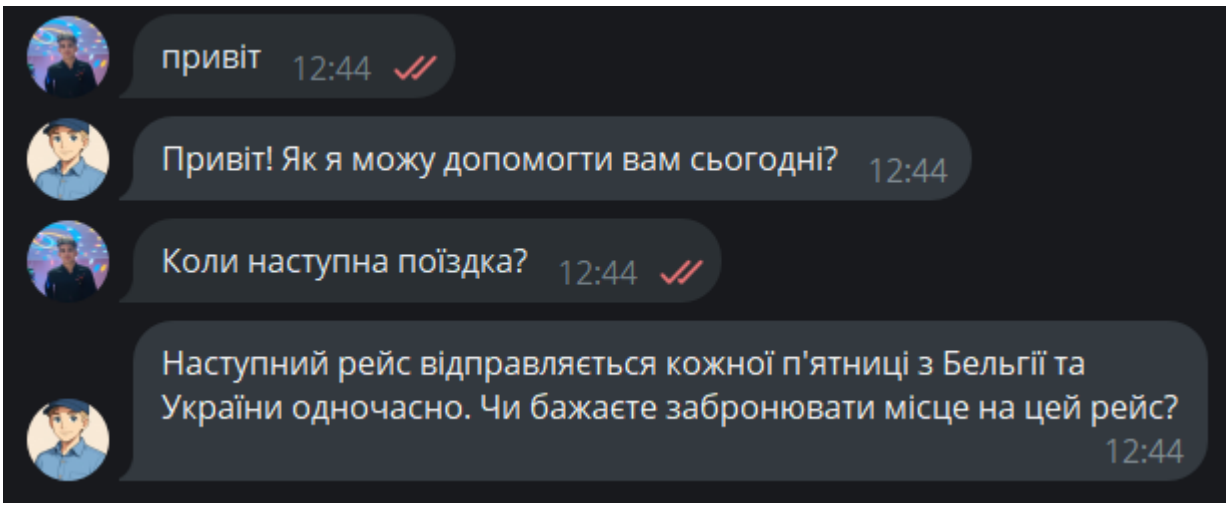

Рис. 3.9 – Відповідь GPT на запитання користувача про поїздку

Такий тип взаємодії підтверджує коректну роботу GPT-модуля у межах заданого сценарію, забезпечуючи гнучку та дружню взаємодію з користувачем [5].

Приклад 2. Перевірка наявності місць на конкретну дату: Користувач надіслав запит: *«Чи є місця на 21 червня?»*. Бот автоматично розпізнав дату, уточнив напрямок , і, здійснивши перевірку в базі Notion, повідомив про кількість доступних місць.

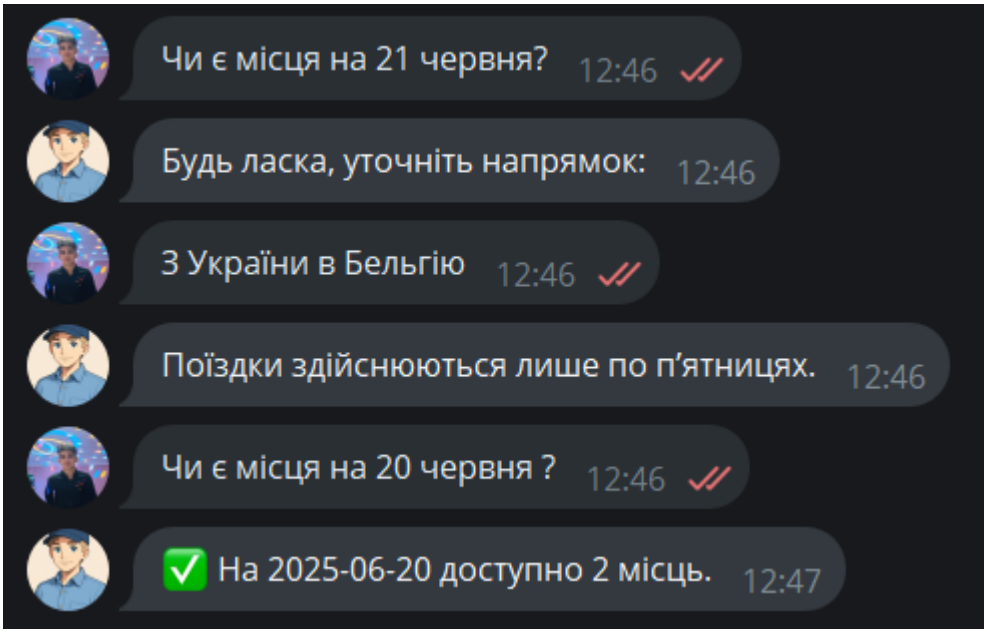

Рис. 3.10 – Результат перевірки доступних місць на обрану дату

Виконання цього тесту показало, що бот успішно взаємодіє з базою даних і вміє точно обробляти часові запити користувача.

Приклад 3. Запис користувача: Після підтвердження наявності місць користувач погодився записатись на поїздку. Бот сформував новий запис у відповідній таблиці Notion та додав маркування, що повідомляє оператора про те, що запис зроблено автоматично.

Рис. 3.11 – Повідомлення про успішний запис користувача

База даних оновилась без помилок, а оператор зможе легко побачити новий запис у списку клієнтів на конкретну дату.

Приклад 4. Обробка зайнятих місць та помилкових записів: У наступному сценарії користувач намагався зайняти місце, яке вже було зарезервоване. Бот проаналізував наявні записи, виявив конфлікт і коректно повідомив про

неможливість запису. У випадку незрозумілого формулювання дати або її відсутності, бот просив уточнити повідомлення.

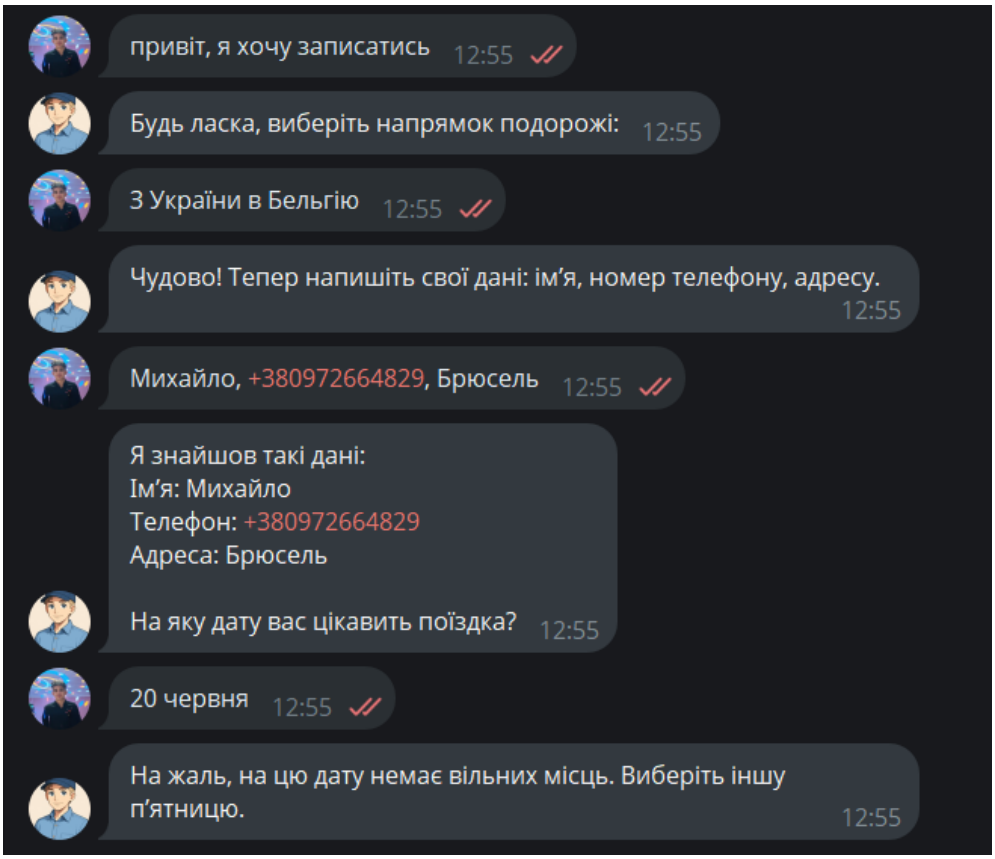

Рис. 3.12 – Повідомлення про відсутність вільних місць або помилкову дату

Це підтверджує наявність захисту від фантомного заповнення бази даних, що особливо важливо при роботі з клієнтами [8].

Виявлення та логування помилок: Окрему увагу було приділено контролю за стабільністю роботи системи. Для цього всі виняткові ситуації автоматично фіксуються у лог-файл bot_errors.log. Записи містять дату, час, тип помилки та контекст виконання, що дозволяє швидко діагностувати причини збоїв.

```
2025-06-03 21:48:12,746 ERROR ❌ Bot error: time data '32-06-2025'
does not match format '%Y-%m-%d'
Traceback (most recent call last):
  File "/app/handlers/message_handler.py", line 149, in
handle_message
    date_obj = datetime.strptime(date_str, "%Y-%m-%d")
ValueError: time data '32-06-2025' does not match format '%Y-%m-%d'
```

Лістинг 3.1 – Приклад запису помилки в журналі bot_errors.log

Цей механізм допоміг на етапі тестування виявити кілька нетипових ситуацій, наприклад помилку авторизації при втраті доступу до Notion або збій API GPT, і оперативно їх усунути [9].

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ.

4.1 Заходи пожежної безпеки

Пожежна безпека є важливою складовою загальної системи охорони праці, особливо в галузі інформаційних технологій, де повсякденна діяльність пов’язана з використанням великої кількості електронного обладнання. У межах дипломної роботи, присвяченої розробці інтелектуального чат-бота на базі штучного інтелекту, робоче середовище передбачає активне використання серверного обладнання, персональних комп’ютерів, хмарних інфраструктур та засобів зв’язку, що створює потенційно пожежонебезпечні умови [10, с. 132].

Офіс розробника, що виконує реалізацію інтелектуальної системи, зазвичай включає не лише робочі місця з ПК, а й серверні кімнати, точки безперебійного живлення, маршрутизатори, джерела збереження даних, зокрема зовнішні накопичувачі або NAS-системи. Усі ці елементи підключені до електромережі, що суттєво підвищує ризик короткого замикання, перегріву обладнання або виходу з ладу вентиляції — типових причин займань у середовищі ІТ-компаній [10, с. 134].

Згідно з Кодексом цивільного захисту України та Правилами пожежної безпеки в Україні, затвердженими наказами МВС № 141 від 01.03.2023 та № 474 від 11.07.2024, керівництво організації зобов’язане організувати систему запобігання пожежам, що включає інструктажі, позначення евакуаційних виходів, регулярне технічне обслуговування обладнання та встановлення засобів пожежогасіння [14, 15].

Зокрема, для офісів з високою концентрацією ІТ-обладнання рекомендовано:

- використовувати вуглекислотні вогнегасники (CO_2), які не пошкоджують електроніку;
- розміщувати схеми евакуації біля кожного виходу;
- встановлювати датчики диму та системи автоматичного оповіщення;
- забезпечити перевірку електропроводки відповідно до ПУЕ;

- обмежити використання побутових приладів та подовжувачів у серверних приміщеннях [10, с. 133].

Рис. 4.1 – План пожежної безпеки типового офісу для розробки програмного забезпечення.

Ще одним критичним елементом є збереження результатів інтелектуальної праці. У випадку пожежі можливе знищення локальних копій вихідного коду, тренованих моделей ШІ, структур баз даних або логів системи чат-бота. Тому на етапі розробки обов’язковим заходом є організація регулярного резервного копіювання: щоденне збереження змін на хмарних сервісах або дублювання інформації на фізичних носіях, що зберігаються у протипожежних сейфах [10, с. 135].

Також згідно з ДБН В.1.1-7:2016 «Пожежна безпека об’єктів будівництва», у офісах, де працюють розробники ПЗ, мають бути встановлені протипожежні двері,

що стримують поширення вогню. Це критично важливо для приміщень серверних кімнат, де зберігається як обладнання, так і інтелектуальна власність [16].

Крім технічних рішень, важливим є навчання персоналу. Співробітники, що беруть участь у створенні або підтримці програмного забезпечення, повинні розуміти, як діяти у випадку задимлення, спрацювання сигналізації або виявлення загоряння в мережевому обладнанні. Проведення періодичних тренувальних евакуацій та інструктажів з використання вогнегасників входить до переліку обов'язкових заходів згідно з чинним законодавством [10, с. 138].

Таким чином, створення безпечного офісного простору для розробки інформаційних систем, зокрема чат-ботів із використанням штучного інтелекту, потребує як технологічних рішень (вогнегасники, детектори диму, протипожежні двері), так і організаційних (інструктажі, план евакуації, резервне копіювання). Лише комплексний підхід дозволяє забезпечити безперервність розробки та захист інтелектуального продукту від незворотних втрат [10, с. 139].

4.2 Долікарська допомога при опіках

У сучасному офісному середовищі, особливо в галузі інформаційних технологій, де активно використовується електронне обладнання, існує ризик виникнення опіків у працівників. Це може статися внаслідок короткого замикання, перегріву пристроїв або порушення правил експлуатації. Тому знання правил домедичної допомоги є обов'язковою складовою забезпечення безпеки персоналу [10, с. 87].

Класифікація опіків: Залежно від глибини ураження, опіки поділяють на чотири ступені.

- І ступінь – почервоніння, невеликий набряк, біль
- ІІ ступінь – утворення пухирів, сильний біль.

- III ступінь – глибоке ураження шкіри, утворення струпу.
- IV ступінь – обвуглення тканин, ураження м’язів та кісток [10, с. 94].

За етіологією опіки класифікують як:

- термічні (вогонь, пара, кип’яток),
- хімічні (кислоти, луги),
- електричні (електричний струм),
- променеві (ультрафіолет, іонізуюче випромінювання) [10, с. 88].

Для оцінки площі ураження застосовується правило «дев’яток», яке дозволяє швидко визначити відсоток ушкодженої поверхні тіла [10, с. 95].

Алгоритм надання домедичної допомоги: Відповідно до Наказу МОЗ № 441 від 09.03.2022 року, основні дії при опіках включають:

- Припинення контакту з джерелом ураження.
- Охолодження зони ураження прохолодною (15–20°C) водою протягом 10–20 хвилин.
- Накладання стерильної сухої пов’язки.
- Забезпечення фізичного й емоційного спокою потерпілого.
- У разі важких опіків – негайне викликання швидкої допомоги [11].

Особливості допомоги при електричних опіках: Електричні опіки особливо небезпечні через внутрішні пошкодження тканин. У першу чергу необхідно знеструмити джерело. Далі – перевірити свідомість і дихання постраждалого. У разі необхідності слід розпочати серцево-легеневу реанімацію [10, с. 96].

Оснащення аптечками та профілактика: Наказ МОЗ України № 1254 від 21.06.2022 року встановлює перелік обов’язкових компонентів аптечки домедичної допомоги. Для випадків опіків аптечка повинна містити:

- стерильні серветки та бинти,
- антисептики,
- протиопікові гелі або спеціальні серветки з гідрогелем [12].

Рекомендується забезпечити працівників відповідним навчанням та проводити інструктажі не рідше ніж раз на три роки. Для працівників з підвищеним ризиком — щороку [13].

Навчання як елемент безпеки: Відповідно до Постанови КМУ № 1109 від 29.11.2017 (ред. 2023), роботодавець зобов’язаний організовувати навчання персоналу діям у надзвичайних ситуаціях, включаючи надання першої допомоги. У практиці ІТ-компаній такі навички є вкрай необхідними, оскільки часто працівники працюють із великою кількістю електроприладів [13].

Проведення тренінгів з долікарської допомоги дозволяє підвищити не лише рівень безпеки, але й загальну стресостійкість колективу, що позитивно позначається на якості роботи.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено інтелектуального Telegram-бота для автоматичної обробки текстових запитів клієнтів у сфері пасажирських перевезень між країнами Європи. В основі системи лежить використання мовної моделі GPT, яка здатна генерувати осмислені відповіді на довільно сформульовані повідомлення користувача, розпізнавати дати, наміри та контекст звернення. Розроблене рішення поєднує переваги гнучкого інтерфейсу Telegram, можливості сучасного штучного інтелекту та зручного для оператора інструменту керування у вигляді баз даних Notion.

У теоретичній частині проєкту було проведено аналіз сучасних підходів до обробки природної мови. Визначено, що rule-based системи, статистичні моделі та класичні нейронні мережі не забезпечують достатнього рівня адаптивності для реалізації відкритого діалогу. Натомість трансформерні моделі, зокрема GPT, продемонстрували здатність з високою точністю обробляти та генерувати природну мову, що й обумовило вибір цієї технології для проєкту.

Практична реалізація включала створення Telegram-бота за допомогою BotFather, організацію баз даних у Notion, підключення OpenAI API для інтеграції GPT, а також написання модулів логіки на мові Python. Архітектура проєкту модульна, що забезпечує її масштабованість, а також простоту супроводу. Було реалізовано логіку розпізнавання дати, напрямку поїздки, перевірки зайнятих місць та запису нових пасажирів з автоматичним формуванням записів у таблиці.

Результати тестування підтвердили працездатність системи в реальних сценаріях: бот успішно взаємодіє з клієнтами, відповідає на запити, перевіряє доступність поїздок та записує нових пасажирів без участі оператора. Такий підхід дозволяє значно знизити навантаження на персонал, підвищити швидкість обробки звернень і забезпечити цілодобову доступність сервісу для клієнтів. Крім того, впроваджено механізм логування, що дозволяє оперативно реагувати на помилки або нетипову поведінку.

Запропоноване рішення має високу прикладну цінність для транспортних компаній, що надають послуги перевезень у кількох країнах. Його можна легко адаптувати під інші сфери діяльності, де необхідна автоматизована комунікація з клієнтами. Подальший розвиток проєкту може включати розширення функціональності — додавання багатомовності, системи рейтингу клієнтів, можливості формування маршрутів або інтеграцію з платіжними сервісами

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaswani A., Shazeer N., Parmar N. et al. Attention is all you need // Advances in Neural Information Processing Systems. 2017. Vol. 30. URL: https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (дата звернення: 02.05.2025)
2. OpenAI. GPT models documentation [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs/guides/gpt (дата звернення: 06.05.2025).
3. Python Telegram Bot API documentation [Електронний ресурс]. – Режим доступу: https://docs.python-telegram-bot.org/ (дата звернення: 09.05.2025)
4. Notion API documentation [Електронний ресурс]. – Режим доступу: https://developers.notion.com/docs (дата звернення: 13.05.2025).
5. Власюк В. С. Штучний інтелект: підручник. Київ: Центр учбової літератури, 2020. 388 с.
6. Глушков В. М. Основи кібернетики: навч. посібник. Київ: Наукова думка, 2018. 312 с.
7. Python-dotenv documentation [Електронний ресурс]. – Режим доступу: https://saurabh-kumar.com/python-dotenv/ (дата звернення: 17.05.2025).
8. GitHub Repository – python-telegram-bot [Електронний ресурс]. – Режим доступу: https://github.com/python-telegram-bot/python-telegram-bot (дата звернення: 21.05.2025).
9. GitHub Repository – notion-sdk-py [Електронний ресурс]. – Режим доступу: https://github.com/ramnes/notion-sdk-py (дата звернення: 25.05.2025).
10. Безпека життєдіяльності : навчальний посібник / О. В. Березюк, М. С. Лемешев. – Вінниця : ВНТУ, 2011. – 204 с.
11. Про затвердження порядків надання домедичної допомоги особам при невідкладних станах: Наказ МОЗ України від 09.03.2022 № 441 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/go/z0356-22 (дата звернення: 28.05.2025)..

12. Про затвердження переліку обов'язкових компонентів аптечки домедичної допомоги: Наказ МОЗ України від 21.06.2022 № 1254 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0851-22 (дата звернення: 31.05.2025).
13. Про затвердження Порядку проведення навчань і перевірки знань з питань охорони праці: Постанова Кабінету Міністрів України від 29.11.2017 № 1109 (у редакції 2023 року) [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/1109-2017-п (дата звернення: 03.06.2025).
14. Stefanyshyn, V., Stefanyshyn, I., Pastukh, O., Yatsyshyn, V., Yakymenko. Accuracy of software and hardware of computer systems for human-machine interaction, I. CEUR Workshop Proceedings, 2024, 3842, pp. 178–183.
15. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329
16. Bryk O., Mudryk I., Holubovskyi M., Stoianov Y. Machine learning models and methods aspects of processing unstructured data // *Proceedings of the 1st International Workshop on Bioinformatics and Applied Information Technologies (BAIT 2024)*. Zboriv, Ukraine, 2024. – P. 64–74.
17. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

ДОДАТКИ

ДОДАТОК А

договірних потужностей, а також можливість переходу на тарифи, адаптовані до реального графіку споживання.

Зростаючий попит на енергоефективність стимулював розвиток програмного забезпечення для зручного управління та моніторингу споживання. Сучасне програмне забезпечення використовує передові технології, такі як штучний інтелект (ШІ), машинне навчання та Інтернет речей (IoT), для оптимізації споживання енергії [3].

Серед основних викликів — потреба у високому рівні кібербезпеки, особливо в умовах підключення критично важливого обладнання до мережі. Також важливо забезпечити навчання персоналу та правильну інтеграцію IoT-систем у існуючу інфраструктуру підприємства. У майбутньому очікується ще глибша інтеграція IoT з відновлюваними джерелами енергії, системами штучного інтелекту для прогнозування споживання.

Література

1. LU, Yang. The current status and developing trends of industry 4.0: A review. *Information Systems Frontiers*, 2025, 27.1: 215-234.

2. REICHARDT, Anna; MURAWSKI, Matthias; BICK, Markus. The use of the Internet of Things to increase energy efficiency in manufacturing industries. *International Journal of Energy Sector Management*, 2025, 19.2: 369-388.

3. SEGUN-FALADE, Osinachi Deborah, et al. Developing innovative software solutions for effective energy management systems in industry. Engineering Science & Technology Journal, 2024, 5.8: 2649-2669.

УДК 004.8

М. Федорків – ст. гр. СП-42, док. філософії. І. Мудрик
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗРОБКА ЧАТ-БОТА ДЛЯ КОНСУЛЬТУВАННЯ З ПЕРЕВЕЗЕНЬ ЄВРОПОЮ

M. Fedorkiv, Ph.D I. Mudryk

DEVELOPMENT OF A CHATBOT FOR TRANSPORT CONSULTATION ACROSS EUROPE

У роботі представлено створення Telegram-чат-бота для надання консультацій щодо перевезень між країнами Європи, зокрема з Бельгії до України та у зворотному напрямку. Основна функціональність полягає у наданні користувачам швидких відповідей на часті запитання про маршрути, країни транзиту (Польща, Німеччина, Нідерланди), документи, ціни та умови перевезення.

У якості бази знань використано платформу Notion, що дозволяє зручно структурувати інформацію про доступні поїздки та здійснювати оновлення в реальному часі без повторного розгортання бота. Telegram-бот реалізовано з використанням Python та бібліотеки python-telegram-bot, а також інтеграції з Notion API.

Завдяки автоматизації відповідей, система дозволяє оптимізувати процес комунікації з клієнтами, зменшити навантаження на операторів і підвищити оперативність обслуговування. Проєкт має перспективу масштабування, зокрема через підтримку багатомовності та розширення бази знань.

Література:

6. Telegram Bot API Documentation. URL: https://core.telegram.org/bots/api
7. Notion API Documentation. URL: https://developers.notion.com/
8. Brownlee J. Developing Intelligent Chatbots: Theory and Applications. Machine Learning Mastery, 2023.
9. OpenAI. GPT Integration in Customer Communication. URL: https://platform.openai.com

ДОДАТОК Б

Основні функції обробки повідомлень у Telegram-боті

```
async def handle_message(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    try:
        user_text = update.message.text.strip()
        user_lower = user_text.lower()
        intent = detect_user_intent(user_text)

        # Обробка вибору напрямку для перевірки
        if context.user_data.get("stage") ==
"awaiting_direction_for_check":
            if "україни" in user_lower:
                context.user_data["direction"] = "ua_to_be"
            elif "бельгії" in user_lower:
                context.user_data["direction"] = "be_to_ua"
            else:
                await update.message.reply_text("Будь ласка, оберіть
напрямок із кнопок.")
                return

            context.user_data["stage"] = None
            date_str = context.user_data.get("pending_query",
{}).get("date")
            if not date_str:
                await update.message.reply_text("На яку дату вас цікавить
поїздка?")
                return

            db_id = NOTION_DB_ID_UA_TO_BE if context.user_data["direction"]
== "ua_to_be" else NOTION_DB_ID_BE_TO_UA
            date_obj = datetime.strptime(date_str, "%Y-%m-%d")
            if date_obj.weekday() != 4:
                await update.message.reply_text("Поїздки здійснюються лише
по п’ятницях.")
                return

            taken = get_taken_seats_for_date(date_str, db_id)
            available = MAX_SEATS - len(taken)
            if available > 0:
                await update.message.reply_text(f"✅ На {date_str}
доступно {available} місць.")
            else:
                await update.message.reply_text(f"❌ На {date_str} немає
вільних місць.")
            context.user_data.pop("pending_query", None)
            return

        # Вибір напрямку для запису
        if not context.user_data.get("direction") and intent ==
"book_trip":
            keyboard = ReplyKeyboardMarkup(
                [["З України в Бельгію", "З Бельгії в Україну"]],
                one_time_keyboard=True,
                resize_keyboard=True
```

```
            )
            context.user_data["stage"] = "awaiting_direction"
            await update.message.reply_text("Будь ласка, виберіть напрямок
подорожі:", reply_markup=keyboard)
            return

        # Обробка вибору напрямку для запису
        if context.user_data.get("stage") == "awaiting_direction":
            if "україни" in user_lower:
                context.user_data["direction"] = "ua_to_be"
            elif "бельгії" in user_lower:
                context.user_data["direction"] = "be_to_ua"
            else:
                await update.message.reply_text("Будь ласка, оберіть
напрямок із кнопок.")
                return
            context.user_data["stage"] = None
            await update.message.reply_text("Чудово! Тепер напишіть свої
дані: ім’я, номер телефону, адресу.")
            return

        # Вибір бази даних
        db_id = NOTION_DB_ID_UA_TO_BE if context.user_data.get("direction")
== "ua_to_be" else NOTION_DB_ID_BE_TO_UA

        # Підтвердження
        if context.user_data.get("stage") == "confirm":
            if user_lower in ["так", "✅ так"]:
                data = context.user_data.get("passenger")
                taken = get_taken_seats_for_date(data["date"], db_id)
                free = find_free_seats(taken, data["seats"])
                if not free:
                    await update.message.reply_text("На жаль, недостатньо
вільних місць на цю дату.")
                    return
                success = add_passenger_to_notion(
                    name=data["name"],
                    date=data["date"],
                    address=data["address"],
                    comment=data.get("comment", ""),
                    phone=data["phone"],
                    seat_numbers=free,
                    db_id=db_id
                )
                if success:
                    await update.message.reply_text("✅ Ви успішно
записані на поїздку! Очікуйте дзвінка.")
                else:
                    await update.message.reply_text("⚠️ Сталася помилка
при записі. Спробуйте ще раз пізніше.")
                context.user_data.clear()
                return
            elif user_lower in ["ні", "❌ ні"]:
                await update.message.reply_text("Добре! Надішліть дані ще
раз, якщо хочете змінити.")
                context.user_data.clear()
                return
```

```
        # Очікування кількості місць
        if context.user_data.get("stage") == "awaiting_seats":
            match = re.search(r"\d+", user_text)
            if match:
                seats = int(match.group())
                if seats > 10:
                    await update.message.reply_text("Максимально можна
забронювати 7 місць.")
                    return
                context.user_data["passenger"]["seats"] = seats
                context.user_data["stage"] = "confirm"
                p = context.user_data["passenger"]
                keyboard = ReplyKeyboardMarkup([["✅ Так", "❌ Ні"]],
one_time_keyboard=True, resize_keyboard=True)
                await update.message.reply_text(
                    f"Підтвердіть, будь ласка:\n"
                    f"Ім’я: {p['name']}\nТелефон: {p['phone']}\nАдреса:
{p['address']}\n"
                    f"Дата: {p['date']}\nКількість місць: {seats}\n\nВсе
правильно?",
                    reply_markup=keyboard
                )
                return
            else:
                await update.message.reply_text("Скільки місць ви хочете
забронювати?")
                return

        # Очікування дати
        if context.user_data.get("stage") == "awaiting_date":
            date_str = parse_date_from_text(user_text)
            if date_str:
                date_obj = datetime.strptime(date_str, "%Y-%m-%d")
                today = datetime.today()
                if date_obj.date() <= today.date():
                    await update.message.reply_text("🚫 Ви не можете
записатись на сьогодні. Виберіть майбутню п’ятницю.")
                    return
                if date_obj.weekday() != 4:
                    await update.message.reply_text("🚫 Поїздки доступні
лише по п’ятницях. Вкажіть іншу дату.")
                    return
                taken = get_taken_seats_for_date(date_str, db_id)
                available = MAX_SEATS - len(taken)
                if available <= 0:
                    await update.message.reply_text("На жаль, на цю дату
немає вільних місць. Виберіть іншу п’ятницю.")
                    return
                context.user_data["passenger"]["date"] = date_str
                context.user_data["stage"] = "awaiting_seats"
                await update.message.reply_text(f"На {date_str} доступно
{available} місць. Скільки бажаєте забронювати?")
                return
            else:
                await update.message.reply_text("Не вдалося розпізнати
дату. Вкажіть ще раз.")
```

```
                return

        # Розпізнавання даних пасажира
        parsed = extract_passenger_data(user_text)
        if parsed and parsed.get("name") and parsed.get("phone") and
parsed.get("address"):
            context.user_data["passenger"] = parsed
            context.user_data["stage"] = "awaiting_date"
            await update.message.reply_text(
                f"Я знайшов такі дані:\nІм’я: {parsed['name']}\nТелефон:
{parsed['phone']}\n"
                f"Адреса: {parsed['address']}\n\nНа яку дату вас цікавить
поїздка?"
            )
            return

        # Перевірка наявності місць
        if intent == "check_availability":
            date_str = parse_date_from_text(user_text)

            if date_str:
                context.user_data["pending_query"] = {"date": date_str}

            if not context.user_data.get("direction"):
                keyboard = ReplyKeyboardMarkup(
                    [["З України в Бельгію", "З Бельгії в Україну"]],
                    one_time_keyboard=True,
                    resize_keyboard=True
                )
                context.user_data["stage"] = "awaiting_direction_for_check"
                await update.message.reply_text("Будь ласка, уточніть
напрямок:", reply_markup=keyboard)
                return

            db_id = NOTION_DB_ID_UA_TO_BE if context.user_data["direction"]
== "ua_to_be" else NOTION_DB_ID_BE_TO_UA
            date_str = date_str or context.user_data.get("pending_query",
{}).get("date")
            if not date_str:
                await update.message.reply_text("На яку дату вас цікавить
поїздка?")
                return

            date_obj = datetime.strptime(date_str, "%Y-%m-%d")
            if date_obj.weekday() != 4:
                await update.message.reply_text("Поїздки здійснюються лише
по п’ятницях.")
                return

            taken = get_taken_seats_for_date(date_str, db_id)
            available = MAX_SEATS - len(taken)
            if available > 0:
                await update.message.reply_text(f"✅ На {date_str}
доступно {available} місць.")
            else:
                await update.message.reply_text(f"❌ На {date_str} немає
вільних місць.")
            context.user_data.pop("pending_query", None)
```

```
            return

        # GPT fallback
        await
context.bot.send_chat_action(chat_id=update.effective_chat.id,
action="typing")
        reply = get_chatgpt_response(user_text)
        await update.message.reply_text(reply)

    except Exception as e:
        error_trace = traceback.format_exc()
        logging.error("Помилка під час обробки повідомлення:
%s\nКонтекст:\n%s", str(e), error_trace)
        await update.message.reply_text("⚠️ Сталася помилка. Вибач, щось
пішло не так.")
```

ДОДАТОК В

ДИСК