

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ІНФОРМАЦІЙНА СИСТЕМА ОБРОБКИ ЗАМОВЛЕНЬ
РЕСТОРАННОГО КОМПЛЕКСУ

Виконав: студент 2 курсу, групи КНТ-613м
спеціальності

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Спеціалізовані комп'ютерні системи

БОРИСОВ О.Є.

(ПРИЗВИЩЕ та ініціали)

Керівник КУДЕРМЕТОВ Р.К.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОЗИНА Г.Л.

(ПРИЗВИЩЕ та ініціали)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	КУДЕРМЕТОВ Р. К., к.т.н., зав.каф.		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 01.10.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних месенджерів та їх переваг, перегляд доступності API-інтерфейсів для месенджерів, перегляд отриманих результатів	11.10.2024 р.	виконано
2	Порівняння та вибір програмного забезпечення	16.10.2024 р.	виконано
3	Порівняння та вибір графічних бібліотек	21.10.2024 р.	виконано
4	Реалізація системи	06.10.2024 р.	виконано
5	Тестування роботи системи	21.11.2024 р.	виконано
6	Оформлення отриманих результатів у ПЗ	26.11.2024 р.	виконано
7	Оформлення графічного матеріалу	02.12.2024 р.	виконано
8	Оформлення допоміжного матеріалу	10.12.2024 р.	виконано

Студент Олександр БОРИСОВ
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) Равіль КУДЕРМЕТОВ
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 108 с., 42 рис., 26 джерел.

АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ, ПРОГРАМНЕ ЗАПЕЗПЕЧЕННЯ, АВТОМАТИЗАЦІЯ, РЕСТОРАН, ЧАТ-БОТ, PYTHON, ШТУЧНИЙ ІНТЕЛЕКТ

Об'єкт дослідження – сервіси автоматичного обслуговування клієнтів у ресторанах.

Предмет дослідження – програмні засоби для автоматичного приймання та оброблення замовлень у ресторанах.

Мета роботи – поліпшення оперативності обслуговування клієнтів та оптимізація робочих процесів за рахунок розробки ефективного автоматичного сервісу приймання та оброблення замовлень у ресторанах через Telegram-бот.

У першій частині роботи проведено аналіз предметної області, розглянуто сучасні підходи до впровадження чат-ботів у ресторанному бізнесі, проаналізовано їхні переваги та недоліки. Також досліджено специфіку роботи Telegram-ботів та їхній потенціал у сфері обслуговування клієнтів.

У другій частині – визначено основні вимоги до системи та проведено обґрунтований вибір програмних інструментів та технологій для розробки системи.

У третій частині – описано проектування та реалізацію програмного забезпечення, яке включає Telegram-бота та програму-менеджер. Деталізовано реалізацію алгоритмів роботи системи.

У четвертій частині проведено тестування програмного забезпечення, результати якого підтвердили відповідність створеної системи поставленим вимогам. Також розглянуто можливості масштабування системи для підтримки роботи в умовах значного збільшення кількості замовлень.

ABSTRACT

Explanatory note to the master's work: 108 pages, 42 figures, 26 sources.

HARDWARE, SOFTWARE, AUTOMATION, RESTAURANT, CHATBOT,
PYTHON, ARTIFICIAL INTELLIGENCE

Object of research: services for automated customer service in restaurants.

Subject of research: software tools for automated order reception and processing in restaurants.

Objective: to improve the efficiency of customer service and optimize workflows by developing an effective automated service for order reception and processing in restaurants using a Telegram bot.

In the first part of the work, an analysis of the subject area is conducted, modern approaches to the implementation of chatbots in the restaurant business are reviewed, and their advantages and disadvantages are analyzed. Additionally, the specifics of Telegram bots and their potential in customer service are studied.

In the second part, the main system requirements are defined, and a justified selection of software tools and technologies for system development is provided.

In the third part, the design and implementation of the software are described, including the Telegram bot and a manager application. The implementation of system operation algorithms is detailed.

In the fourth part, the software testing results are presented, confirming the compliance of the developed system with the specified requirements. Additionally, possibilities for system scalability to handle a significant increase in order volume are discussed.

ЗМІСТ

Скорочення та умовні позначки	7
Вступ	8
1 Аналіз предметної області	10
1.1 Історія розвитку чат-ботів	11
1.2 Підходи до розробки чат-ботів в месенджерах	16
1.3 Переваги чат-ботів для бізнесу	18
1.4 Популярні месенджери в Україні та їхнє значення для бізнесу	19
1.5 Доступність API-інтерфейсів для месенджерів	21
1.6 Постановка завдання	23
2 Вибір програмних засобів для розробки	24
2.1 Вимоги до розроблюваного програмного забезпечення	25
2.2 Вибір месенджера для чат-бота	27
2.3 Порівняння існуючих API для обраного месенджера	28
2.4 Вибір LLM-моделі для чат-бота	34
2.5 Вибір мови програмування для розробки програми-менеджера	39
2.6 Порівняння графічних бібліотек для UI на Python	40
2.7 Вибір графічної бібліотеки для програми-менеджера	48
2.8 Вибір програмного забезпечення баз даних	50
3 Реалізація системи	52
3.1 Підготовка середовища розробки	52
3.2 Розробка програми-менеджера для обробки замовлень	61
3.3 Розробка бази даних	73
3.4 Розробка telegram-бота	77
4 Тестування роботи системи	86
Висновки	91
Перелік джерел посилання	92
Додаток А Лістинги програм	95

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	– База даних
СУБД	– Система управління базами даних
ШІ	– Штучний інтелект
API	– Application Programming Interface, Інтерфейс програмування застосунків
CV	– Computer vision, Комп'ютерний зір
LLMA	– Large Language Model AI, Архітектури великих мовних моделей
NPL	– Natural language processing, Обробка природньої мови
HTTP	– Hypertext Transfer Protocol, Протокол передачі гіпертекстових документів
SQL	– Structured query language, Мова структурованих запитів
UI	– User interface, Інтерфейс користувача

ВСТУП

В епоху стрімкого технологічного прогресу, з розвитком інтернет-технологій і передачею великих обсягів інформації за секунди, ми стали свідками глибоких змін у суспільстві. Сучасні інтернет-технології, такі як широкосмуговий доступ до інтернету, мобільний зв'язок і хмарні сервіси, радикально трансформували наше повсякденне життя. Вони не тільки спростили доступ до інформації, а й створили нові можливості для комунікації, навчання, роботи та розваг. Інтернет-технології сприяли появі нових форм бізнесу, полегшили доступ до освітніх і культурних ресурсів, значно поліпшили рівень життя і підвищили глобальну взаємодію людей.

Серед ключових інновацій, які справили істотний вплив є месенджери. Останніми роками месенджери стали одним із найпопулярніших засобів комунікації, надаючи користувачеві миттєвий і зручний доступ до спілкування без кордонів. Наприклад, станом на 2024 рік, кількість активних користувачів месенджера WhatsApp перевищує два мільярди осіб по всьому світу. Це можна порівняти з традиційними методами зв'язку, такими як електронна пошта та дзвінки, але месенджери пропонують більшу швидкість і гнучкість в обміні повідомленнями.

У цьому контексті месенджер-боти стали невід'ємною частиною екосистеми сучасних комунікацій. Ці автоматизовані застосунки, що функціонують через інтерфейс месенджера, вже міцно увійшли в наше повсякденне життя. Вони виконують широкий спектр завдань: від надсилання повідомлень і новин до складних операцій, таких як опрацювання замовлень або управління бізнес-процесами.

У дипломній роботі особливу увагу приділено використанню можливостей реалізації месенджер-бота для автоматизації приймання та обробки замовлень у ресторанах. Ця автоматизація значно скорочує час обробки замовлень, що дає змогу оптимізувати робочі процеси в закладі. Крім того, використання

месенджер-бота відкриває нові можливості для взаємодії з клієнтами, надаючи їм зручний і сучасний спосіб оформлення замовлень віддалено або в ресторані замість карти страв з офіціантом. Подібна інтеграція технології в ресторанний бізнес сприяє підвищенню операційної ефективності та поліпшенню клієнтського досвіду. Таким чином, обрана тематика дипломної роботи магістра є актуальною.

Мета роботи – поліпшення оперативності обслуговування клієнтів та оптимізація робочих процесів за рахунок розробки ефективного автоматичного сервісу приймання та оброблення замовлень у ресторанах через Telegram-бот.

Для реалізації цієї мети належить вирішити такі проблеми:

- провести аналіз предметної області та визначити вимоги до розроблюваної системи;
- провести обґрунтований вибір програмних засобів, необхідних для розробки системи, включаючи вибір відповідних інструментів для розробки зручного і функціонального користувальницького інтерфейсу;
- спроектувати систему, враховуючи вимоги легкості керування замовленнями;
- реалізувати систему на основі обраних обраних технологій та інструментів;
- провести тестування системи та аналіз отриманих результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цій частині буде наведено ретельний аналіз взаємозв'язку між розвитком технологій чат-ботів і месенджерів та їх застосуванням у бізнесі, особливо в ресторанній індустрії. Дослідження починається з вивчення історичного розвитку чат-ботів, що відображає їхню еволюцію від простих автоматизованих відповідачів до складних систем, здатних до ведення осмисленого діалогу та виконання складних завдань.

Далі увага приділяється різним підходам до розроблення чат-ботів. Аналізуються як традиційні методи, так і новітні технології глибокого навчання, що дають змогу створювати більш інтелектуальні та адаптивні боти. Особлива увага приділяється застосуванню цих технологій для поліпшення взаємодії з клієнтами та оптимізації бізнес-процесів. Вони виконують широкий спектр завдань: від надсилання повідомлень і новин до складних операцій.

Переваги чат-ботів для бізнесу розглядаються через призму їхньої здатності покращувати обслуговування клієнтів та ефективність операцій. Обговорюються питання збільшення продуктивності, зниження витрат і можливості надання більш персоналізованого сервісу.

Дослідження також включає аналіз популярних месенджерів в Україні, таких як Telegram, WhatsApp та інших, і їхньої ролі в комерційному середовищі. Розглядається, як ці платформи використовуються для поліпшення взаємодії з клієнтами, ведення маркетингових кампаній і полегшення процесу замовлення в ресторанах. Подібна інтеграція технологій в ресторанний бізнес сприяє підвищенню операційної ефективності та поліпшенню клієнтського досвіду. Таким чином, обрана тематика дипломної роботи магістра є актуальною.

Далі наводиться огляд доступності API-інтерфейсів для месенджерів підкреслює їхню значимість для інтеграції з різними бізнес-системами.

Завершуються перша частина дипломної роботи формуванням цілей та завдання дослідження.

1.1 Історія розвитку чат-ботів

Чат-боти являють собою програмне забезпечення, яке здатне імітувати людське спілкування, зіграли революційну роль у сфері цифрових технологій. Їхній розвиток тісно пов'язаний із прогресом у галузі ШІ і машинного навчання, що дало змогу значно розширити їхню функціональність і сфери застосування.

Від самого початку їхньої появи чат-боти призначалися для автоматизації взаємодій між людьми та комп'ютерними системами, лінія часу появи-чат ботів представлена нижче на рисунку 1.1. Вони пропонують унікальні можливості для бізнесу, освіти, охорони здоров'я та багатьох інших сфер, де потрібне швидке спілкування. У бізнес-секторі чат-боти використовуються для обслуговування клієнтів, автоматизації продажів і маркетингових взаємодій. В освіті, наприклад, вони можуть слугувати персональними помічниками, надаючи інформацію та навчальні матеріали студентам.

Особливістю чат-ботів на сьогодні є їхня здатність до навчання та адаптації. Сучасні чат-боти, засновані на нейромережах і алгоритмах машинного навчання, здатні аналізувати великі обсяги даних, покращуючи свою здатність до розуміння й обробки природної мови. Це дає їм змогу вести більш природний і людиноорієнтований діалог, значно підвищуючи зручність і ефективність їхнього використання.

Першим проривом у розглянутій галузі став чат-бот ELIZA, розроблений 1965 року в Массачусетському технологічному інституті. ELIZA була однією з перших спроб створення програми, здатної імітувати людський діалог. Хоча її можливості були обмежені, ELIZA заклала основу для майбутнього розвитку чат-ботів, демонструючи потенціал штучного інтелекту в галузі обробки природної мови [2].

Чат-бот ELIZA створена Джозефом Вайценбаумом у 1966 року, стоїть на перетині досліджень у царині ШІ та обробки природної мови. Ця програма використовувала методологію зіставлення зі зразком і заміни для імітації діалогу,

що було новаторським підходом на той час. Незважаючи на свою простоту, ELIZA сприяла значним змінам у розумінні можливостей комп'ютерів імітувати людську мову на рисунку 1.2 представлено вигляд переписки з ELIZA [3].

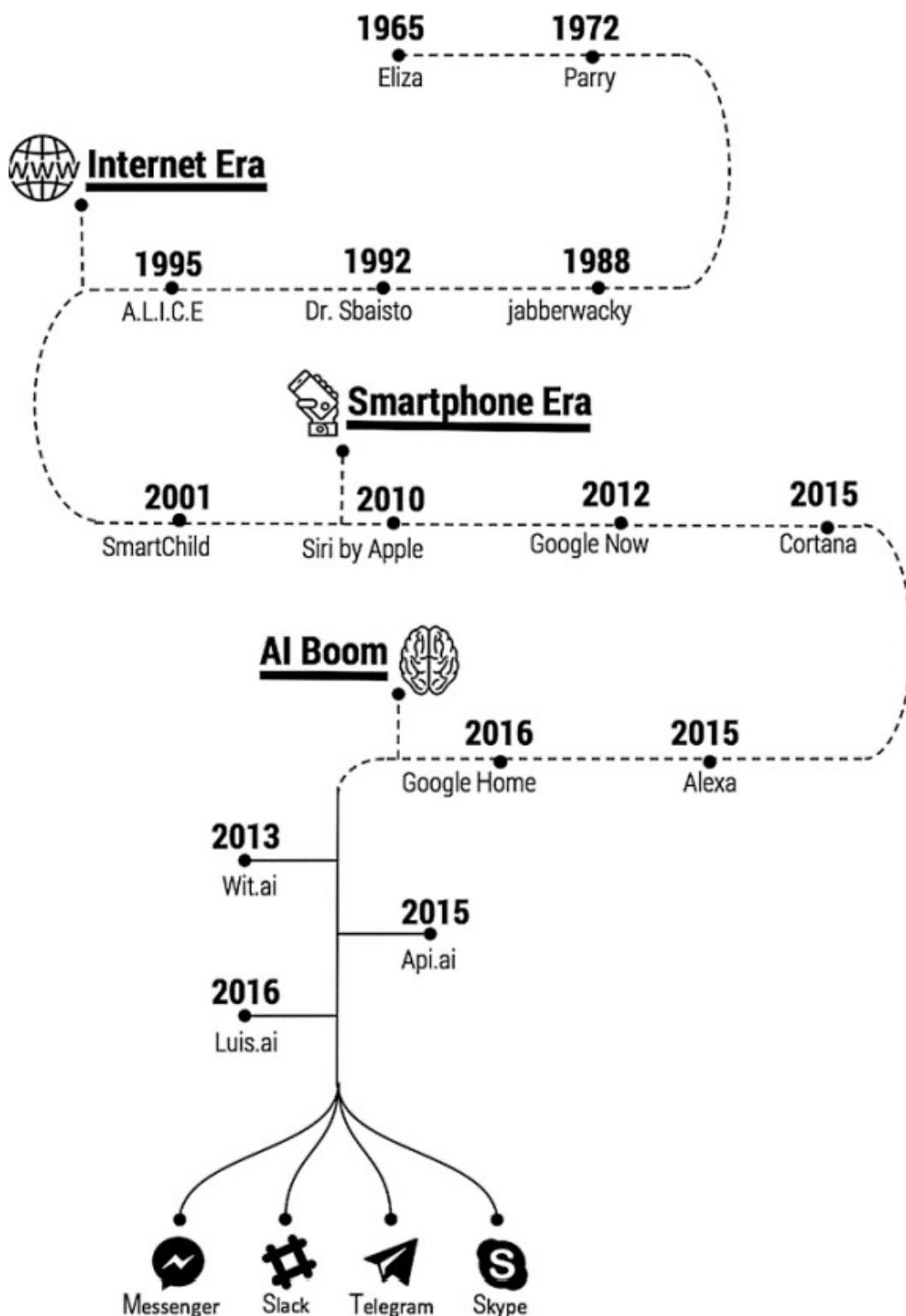


Рисунок 1.1 – Лінія часу появи чат-ботів

```

Welcome to

EEEEEE LL      IIII  ZZZZZZ  AAAAA
EE      LL      II    ZZ      AA  AA
EEEEEE LL      II    ZZZ      AAAAAA
EE      LL      II    ZZ      AA  AA
EEEEEE LLLLLL  IIII  ZZZZZZ  AA  AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:

```

Рисунок 1.2 – Скріншот процесу листування з чат-ботом ELIZA

Одним із найвідоміших сценаріїв ELIZA був «Доктор», який імітував поведінку психотерапевта, часто повторюючи фрази користувача, щоб підтримувати діалог. Цей підхід дозволив ELIZA створити ілюзію розуміння та емпатії, хоча насправді програма не мала ні розуміння, ні інтелекту.

Згідно з історією, Вайценбаум був здивований і навіть занепокоєний тим фактом, що користувачі почали довіряти ELIZA, розповідаючи їй свої особисті проблеми і секрети. Ця реакція засвідчила не лише можливості програмного забезпечення, а й порушила важливі питання про етику та сприйняття штучного інтелекту.

Вайценбаум активно відкидав ідею про те, що машини можуть замінити людський інтелект або розуміння. Він стверджував, що комп'ютери - це лише інструменти, які розширюють людські можливості, але не замінюють їх. Ця позиція стала основоположною для подальшого розвитку галузі чат-ботів і досліджень у сфері штучного інтелекту, а також інтерфейсу взаємодії людина-машина.

Створення ELIZA справило значний вплив на наступні покоління чат-ботів. Багато розробників і дослідників спиралися на модель Вайценбаума, прагнучи до

створення більш вдосконалених і людиноподібних взаємодій. Це прагнення природно привело до зацікавленості в тесті Тюрінга – методі оцінки здатності комп'ютерної програми вести діалог, який не відрізняється від людського. З часом проходження цього тесту стало спільною метою для багатьох розробників чат-ботів того часу.

ELIZA, незважаючи на свої обмеження, заклала основу для майбутніх досліджень у сфері ШІ та обробки природної мови. Цей проєкт показував, що навіть з обмеженими засобами і простими методами можна створити взаємодію, яка сприймається як переконлива і значуща.

Наступним ботом на який було звернено увагу під час історичного дослідження став чат-бот Jabberwacky, створений британським програмістом Ролло Карпентером 1988 року. Jabberwacky був розроблений з метою імітації природного людського спілкування, але вже в більш розважальній і гумористичній формі, що відрізняло його від більш ранніх моделей.

Особливістю Jabberwacky було використання техніки штучного інтелекту, відомої як «зіставлення контекстного шаблону». Ця техніка давала змогу боту адаптуватися до контексту розмови, роблячи взаємодію з користувачем більш природною і непередбачуваною. У результаті, Jabberwacky зміг запропонувати новий рівень взаємодії, який не тільки розважав користувачів, а й продемонстрував потенціал ШІ у створенні більш складних і людських діалогів.

Jabberwacky привернув увагу не тільки у сфері розваг, а й в академічних колах, де його використовували для вивчення взаємодії людини і машини. Цей бот також зробив внесок у розвиток розуміння штучного інтелекту, надаючи унікальну платформу для вивчення людської поведінки та комунікації.

На відміну від своїх попередників, Jabberwacky не був створений для виконання конкретних практичних завдань, таких як обслуговування клієнтів або автоматизація роботи. Його основна мета – імітація людського спілкування і створення інтерактивної розваги. Цей підхід відкрив нові перспективи у використанні ШІ, демонструючи, що штучний інтелект може бути не тільки функціональним, а й захопливим.

За роки свого існування Jabberwacky продовжив розвиватися, а його технології почали використовувати в різних сферах, включно з продажами та маркетингом. Його творець прагнув перевести програму від текстової взаємодії до повністю голосового управління, розширюючи можливості навчання ШІ на основі звуку та інших сенсорних сигналів.

Jabberwacky уособлює значний зсув у розвитку чат-ботів і штучного інтелекту, показуючи, що ШІ може бути використаний як засіб для створення унікальних, розважальних і освітніх дослідів. Його внесок у сферу чат-ботів і ШІ залишається значним, оскільки він показує, як технології можуть збагатити людське спілкування і взаємодію.

Продовжуючи огляд історії чат-ботів, варто згадати Dr. Sbaitso, створеного Creative Labs для MS-DOS у 1992 році (рисунок 1.3). Цей чат-бот являв собою одну з перших спроб впровадження штучного інтелекту в програму чату з повним голосовим керуванням. Dr. Sbaitso був розроблений, щоб вести діалог з користувачем, імітуючи роль психолога. Його відповіді, хоча й були відносно простими, наприклад (з англ.), «Чому ви так почуваетесь?», ознаменували важливий етап у розвитку чат-ботів.

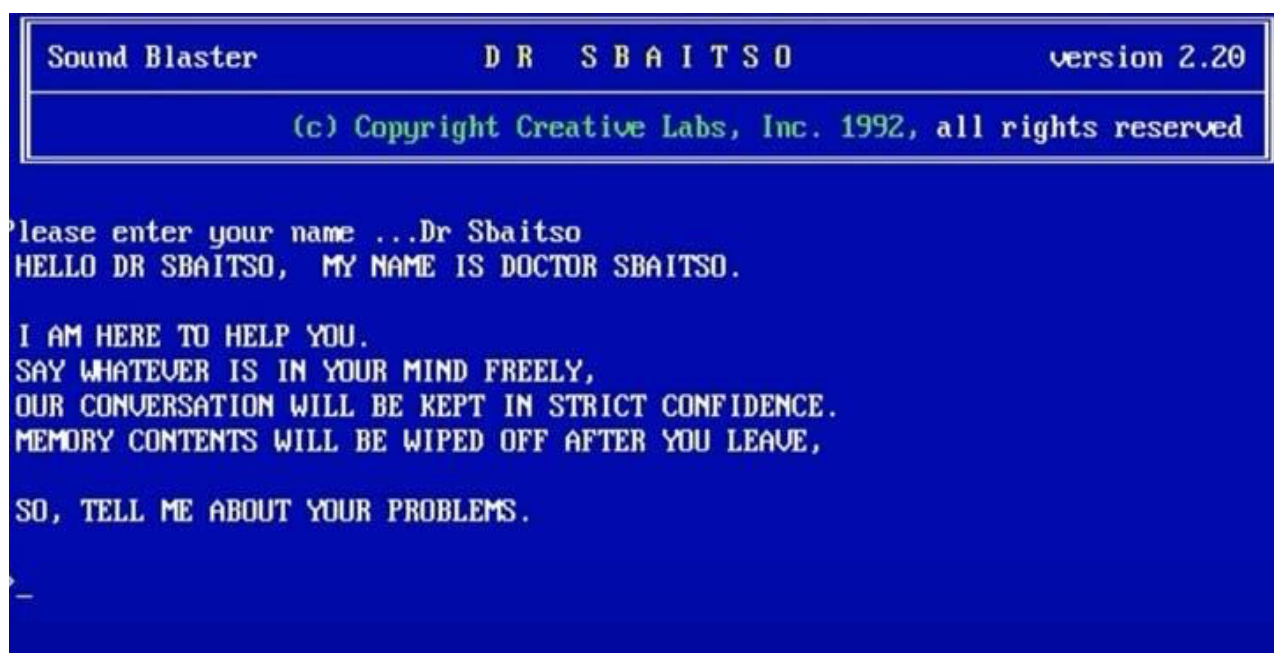


Рисунок 1.3 – Скріншот роботи програми чат-бота Dr. Sbaitso

Наступним значущим кроком у цьому напрямку стали системи відомі як LLMA(Large Language Model Architecture), архітектури великих мовних моделей до них можна віднести такий програмний від продукт Apple як ALICE, Siri від а також Gemini від Google, LLMA3.1 Meta, Microsoft's Cortana, Amazon's Alexa, та GPT від OpenAI. Ці системи значно просунулися в «розумінні» природної мови і здатності вести складні діалоги, демонструючи величезні поліпшення завдяки розвитку нейромережових алгоритмів трансформеру з комбінацією іншими видами нейрони мереж, що у композиції дає штучний інтелект такий, який він відомий зараз на 2024 р.

Чат-боти швидко стали незамінними в комерційному та особистому використанні. Вони знайшли своє застосування в обслуговуванні клієнтів, наданні персоналізованих рекомендацій, підтримці ухвалення рішень і багато іншого. Ця технологія стала ключовим елементом у багатьох сферах, від бізнесу до повсякденного життя.

Сучасний стан чат-ботів характеризується їхньою інтеграцією не тільки з месенджерами, а й соціальними мережами, водночас вони стають дедалі інтелектуальнішими та багатофункціональними. Розвиток алгоритмів штучного інтелекту дає змогу чат-ботам навчатися на основі великих даних і надавати точніші та корисніші відповіді.

Від ELIZA до LLMA, чат-боти пройшли довгий шлях і мали значний вплив на розвиток сучасних технологій. Вони перетворили способи взаємодії людини з машинами, і ми можемо очікувати, що вони все більше будуть інтегруватися у повсякденне життя та бізнес-ідеї.

1.2 Підходи до розробки чат-ботів в месенджерах

Існує багато методів, які використовуються при розробці чат-ботів в месенджерах, і всі вони мають ті чи інші відмінні риси, а їхня робота може бути

застосована в певних галузях. Тип реалізації відіграє важливу роль у визначенні функціональності та ефективності бота щодо виконання поставлених завдань [1].

Основні методи включають:

- кінцеві автомати;
- глибоке навчання;
- правила та шаблони;
- гібридні підходи.

Для досягнення цілей дипломної роботи з розроблення чат-бота для опрацювання ресторанних замовлень було вирішено сфокусуватися на двох підходах – глибокому навчанні та кінцевих автоматах. Реалізація чат-бота з використанням глибокого навчання надасть змогу адаптуватися до різних сценаріїв взаємодії, враховуючи контекст спілкування. Де у реалізації системи отримується можливість точніше «розуміти» наміри користувача та реагувати на складні запити. Водночас, для виконання простих завдань, таких як прийняття та підтвердження замовлень, кінцеві автомати виявляються більш ефективними завдяки своїй структурованій і передбачуваній природі. Вони дають змогу розробити бота з чітко визначеними сценаріями, що спрощує його реалізацію, тому що кінцеві автомати являють собою один з найбільш ранніх і базових підходів до розробки чат-ботів. Цей метод базується на використанні зумовлених шаблонів і правил, що робить його оптимальним для обмежених завдань із чітко окресленими сценаріями. Зокрема, їх ефективно застосовують у системах інтерактивних голосових відповідей (IVR) і для опрацювання вузькоспеціалізованих запитів, таких як запис на прийом або відповіді на поширені запитання.

Щодо методів глибокого навчання, для реалізації чат боту, то вони являють собою більш просунутий підхід, що дає змогу ботам аналізувати комплексні запити і генерувати більш релевантні відповіді. Цей підхід передбачає використання нейронних мереж і забезпечує адаптацію ботів до контексту, що робить його незамінним для опрацювання багатодомених запитів і складніших взаємодій із користувачами. Наприклад, чат-боти, засновані на глибокому навчанні, здатні

адаптуватися до різних запитів клієнтів, пропонуючи персоналізовані рекомендації та відповіді.

Вибір між кінченими автоматами і методами глибокого навчання детермінується цілями і завданнями, які повинен вирішувати чат-бот. Для простих і передбачуваних завдань, таких як приймання замовлень у ресторані, кінцеві автомати можуть бути достатніми. Однак, для реалізації поставлених завдань дипломної роботи системі потрібно буде обробляти і складніші запити, наприклад, адаптуватися до запитів клієнтів у реальному часі або керувати кількома сервісами, використання глибокого навчання буде більш ефективним.

Виходячи з вищевикладеного, доцільно провести подальші дослідження в напрямку реалізації чат-бота на основі методів глибокого навчання з мінімальною інтеграцією кінцевого автомата для оптимізації певних аспектів функціонування.

1.3 Переваги чат-ботів для бізнесу

Чат-боти роблять значний внесок у бізнес-стратегії сучасних компаній, пропонуючи безліч переваг, від поліпшення обслуговування клієнтів до оптимізації внутрішніх процесів.

Особистість чат-бота відіграє ключову роль у сприйнятті бренду клієнтами. Індивідуальний підхід у розробці чат-бота, включно з вибором імені, зображення і стилю спілкування, може істотно поліпшити користувацький досвід. Це надає унікальності бренду і зміцнює його сприйняття. Такі аспекти, як мова бота і спосіб взаємодії, повинні відповідати образу і цінностям компанії, а також уподобанням цільової аудиторії.

Чат-боти забезпечують цілодобове обслуговування клієнтів, що є значною перевагою порівняно з традиційними методами підтримки. Вони здатні обробляти безліч запитів одночасно без збільшення робочого навантаження, що значно покращує ефективність обслуговування клієнтів і зменшує час очікування. У разі

складних запитів, які чат-бот не може обробити, він може перенаправити користувача до живого оператора. Використання чат-ботів також може збільшити продажі. Вони надають потенційним клієнтам потрібну інформацію та пропозиції в критичні моменти, що може значно підвищити ймовірність здійснення покупки. Чат-боти можуть активно допомагати користувачам на сайті, направляючи їх через воронку продажів, надаючи консультації щодо продуктів і допомагаючи з вибором.

У сфері діалогового маркетингу чат-боти пропонують нові можливості для інтерактивного спілкування з клієнтами. Вони дають змогу компаніям вести бесіду з клієнтами на рівних, пропонуючи індивідуальні рішення і відповіді на їхні запити. Це сприяє зміцненню стосунків з клієнтами і поліпшенню сприйняття бренду.

Крім цих переваг, чат-боти надають цінні дані про поведінку клієнтів. Аналізуючи питання і проблеми, обговорювані в розмовах, компанії можуть оптимізувати свою стратегію контенту, адаптувати продуктове портфоліо і поліпшити загальну бізнес-стратегію. Виходячи з джерела можна сказати, що чат-боти пропонують широкий спектр застосування в різних галузях і сценаріях використання, від служби підтримки клієнтів до маркетингу і продажів [4]. Їх можна використовувати для автоматизації та поліпшення процесів в IT-підтримці, HR, управлінні персоналом і багатьох інших галузях, що робить їх потужним інструментом для будь-якого бізнесу.

1.4 Популярні месенджери в Україні та їхнє значення для бізнесу

Інформація про активність користувачів від різних месенджерів є конфіденційною інформацією компанії, і її часто складно отримати, поки компанія сама не опублікує дець інформацію. Тому, під час дослідження аналізу популярності месенджерів в Україні, було вирішено використовувати непрямі методи. Ці методи включають використання даних, таких як перевірка публічно доступних досліджень і статистичних даних від незалежних організацій або людей,

які аналізують популярність месенджерів. Також оцінка мобільних застосунків через вивчення рейтингів і відгуків користувачів в App Store і Google Play допомагає зрозуміти, які месенджери найбільш популярні в Україні. Аналіз обговорень у соціальних медіа та на форумах може дати уявлення про популярні месенджери серед користувачів. Крім того, пошук опитувань і проведення анкетування серед користувачів також сприяють отриманню інформації про їхні вподобання у сфері месенджерів.

У процесі аналізу питання про популярні месенджери в Україні на період 2021-2023 роки за вільно доступними джерелами, знайдені через пошукову систему Google. Одне з перших вільно доступних джерел, що надають потрібну інформацію [5], є опитування, проведене компанією КМІС у вересні, де учасники могли обрати кілька варіантів відповіді (рисунок 1.4).

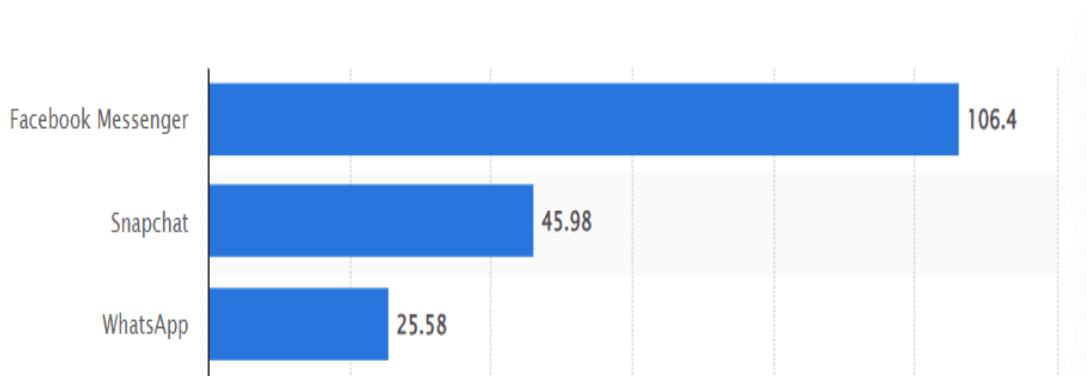


Рисунок 1.4 – Результат опитування компанією КМІС

Ще одним джерелом для отримання даних про популярність месенджерів в Україні є платформа Similarweb [6]. Цей сервіс займається аналізом веб-трафіку, надаючи деталізовані дані та аналітику про відвідуваність сайтів і використання мобільних застосунків. Основне завдання Similarweb полягає в тому, щоб допомогти бізнесам і маркетологам відстежувати активність користувачів, оцінювати конкурентні позиції і знаходити нові можливості для просування і зростання.

На підставі даних платформи на 2024 рік, Telegram займає лідируючі позиції серед месенджерів в Україні. Також серед популярних застосунків відзначаються

Viber, WhatsApp, Facebook Messenger та Instagram Direct. Telegram надає можливості для передачі медіафайлів і текстових повідомлень і вважається непоганим інструментом для бізнесу завдяки інтеграції чат-ботів, які можуть автоматизувати рекламу, вести спілкування з клієнтами та керувати публічними сторінками. Месенджер дає змогу створювати маркетингові стратегії з більшою гнучкістю завдяки відсутності суворих обмежень на рекламні розсилки.

Viber, популярний серед старших користувачів, також широко використовується завдяки інтуїтивно зрозумілому інтерфейсу і широкому охопленню аудиторії. Бізнес-акаунти та публічні чати у Viber дають змогу компаніям безпосередньо взаємодіяти з клієнтами. Інтеграція з CRM-системами спрощує управління замовленнями і клієнтськими даними.

Facebook Messenger, який вимагає наявності облікового запису в соціальній мережі, також залишається популярним інструментом завдяки своєму функціоналу для бізнесу, що включає чат-боти та інтеграцію з сайтами. Однак згідно аналізу відгуків на playmarket та apple store його інтерфейс вважається складнішим порівняно з іншими месенджерами.

WhatsApp, незважаючи на його простоту і популярність, особливо зручний для текстових чатів і обміну мультимедіа. Хоча він не підходить для масових рекламних розсилок, WhatsApp активно використовується для підтримки зв'язку з клієнтами та інформування їх про статус замовлень.

1.5 Доступність API-інтерфейсів для месенджерів

Як уже зазначалося, в епоху цифрового зв'язку месенджери суттєво змінили способи взаємодії як між людьми, так і між підприємствами. месенджери пропонують широкий спектр функцій – від текстових повідомлень і обміну медіафайлами до голосових і відеодзвінків, що забезпечує зручні та різноманітні

способи комунікації. На сьогодні майже всі месенджери безкоштовно надають розробникам API-інтерфейси, що дають змогу інтегрувати їхній функціонал у бізнес-процеси, автоматизувати завдання і розширювати можливості взаємодії з клієнтами. Однак, при всіх їхніх можливостях, API різних месенджерів має свої обмеження, які впливають на те, як їх можна використовувати в бізнесі.

WhatsApp, що належить корпорації Meta, є одним із найпопулярніших месенджерів згідно інформації з сервісу Similarweb використанню трафіка України посягає третя місце, можна припустити, що він активно використовуються для ділових комунікацій України. WhatsApp Business API надає компаніям можливість взаємодіяти з клієнтами, відправляти повідомлення і медіафайли, а також здійснювати дзвінки. Але варто зазначити, що доступ до цього API не є відкритим для всіх охочих - компанії повинні пройти процес схвалення, перш ніж зможуть його використовувати. Це обмеження може впливати на можливість невеликих підприємств або стартапів швидко впровадити цю платформу у свої процеси. Важливим аспектом також є система оплати за повідомлення, яка залежить від типу повідомлення та часу його надсилання. Наприклад, повідомлення, відправлені поза 24-годинним вікном з моменту останньої взаємодії клієнта, оплачуються окремо, що може призвести до значних витрат для компаній, які активно використовують цю платформу для масових розсилок [7]. Крім цього, правила WhatsApp забороняють масові рекламні розсилки без згоди користувача, що обмежує використання API для широких маркетингових кампаній.

Telegram пропонує більш гнучкі можливості для інтеграції в бізнес завдяки двом типам API: Bot API, який дає змогу створювати автоматизовані боти для взаємодії з клієнтами, і Telegram API, який використовується для розробки користувацьких додатків. Особливістю Telegram є те, що його API більш доступний і не вимагає проходження довгого процесу схвалення. Це робить Telegram привабливим вибором для компаній, які прагнуть швидкої інтеграції та широкого охоплення. Однак Telegram також накладає свої обмеження, зокрема на масові розсилки. Хоча правила щодо рекламних повідомлень менш жорсткі, ніж у WhatsApp, Telegram все ж таки регулює обсяги розсилок і може блокувати боти, які

відправляють спам. Також варто враховувати, що Bot API обмежений у частоті запитів - не більше ніж 30 повідомлень на секунду, що може створювати перешкоди для високоактивних ботів або великих компаній з великою кількістю клієнтів [8]. Проте відсутність плати за використання API робить його привабливим варіантом для малого та середнього бізнесу.

Viber - ще один популярний месенджер в Україні, який активно використовується в Україні та інших країнах для ділових цілей. API Viber дає змогу бізнесу надсилати повідомлення, відеоповідомлення та отримувати дані про користувачів. Однак використання API Viber також пов'язане з низкою обмежень, особливо коли йдеться про масові розсилки. Рекламні повідомлення можуть бути відправлені тільки користувачам, які дали свою згоду на отримання таких повідомлень, що обмежує можливість для агресивних маркетингових кампаній. На додаток до цього, як і WhatsApp, Viber використовує модель оплати за повідомлення, де вартість залежить від обсягу та типу розсилки. Компанії, що використовують Viber для взаємодії з великою кількістю клієнтів, повинні враховувати ці витрати під час планування своїх маркетингових і комунікаційних стратегій. Важливо відзначити, що API Viber також обмежений за кількістю запитів, які можуть бути відправлені за певний період часу [9].

Інтеграція API месенджерів у бізнес-процеси пропонує компанії безліч можливостей, але ці можливості залежать від багатьох чинників.

1.6 Постановка завдання

На основі проведеного дослідження та аналізу можливостей месенджерів і API-інтерфейсів, акцентується увага на розробці автоматизованого рішення для поліпшення бізнес-процесів у сфері ресторанного бізнесу. Мета цього дослідження лежить у впровадженні інноваційних технологій для раціоналізації процесу

приймання та обробки замовлень. Це включає створення месенджер-бота, здатного автоматизувати взаємодію з клієнтами, і програмного менеджера для управління потоком замовлень і адаптації меню. Такий підхід передбачає значне поліпшення ефективності операційної діяльності ресторанів, зміцнення зв'язку з клієнтами та підвищення рівня задоволеності обслуговуванням.

Для досягнення поставленої мети було передбачено такі завдання дослідження:

- дослідження та порівняння наявних програмних продуктів, які можуть бути використані для створення проекту дипломної роботи. Це включає аналіз доступних інструментів і технологій, здатних забезпечити створення функціонального та надійного месенджер-бота для ресторанного бізнесу.

- розробка програмної складової проекту, що включає визначення програмної архітектури. Це завдання передбачає вибір відповідних мов програмування, фреймворків і бібліотек, а також розробку алгоритмів і логіки роботи бота і менеджерської програми.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ

У приведеному розділі подано покроковий опис дослідження та аналіз основних передумов, які можуть бути важливими для побудови програмного забезпечення для автоматизації роботи ресторанів. Він починається з аналізу функціональних і нефункціональних вимог до програмного забезпечення, яке буде досліджуватися. Це включає оцінку можливостей чат-бота, реалізацію основних та додаткових функцій програми-менеджера, а також визначення критеріїв, яким повинна відповідати система, стабільність та інформаційна безпека.

Згідно з аналізом, наведеним у пунктах 1.4 та 1.5, здійснюється вибір месенджера відповідно до вимог системи. Це рішення приймається з урахуванням функціональності месенджера та його сумісності з іншими системами, а також

загального охоплення цільової аудиторії. Крім того, враховуються технічні параметри API месенджерів і підтримувані мови програмування, що, в свою чергу, дозволяє вибрати найбільш підходящу платформу для реалізації чат-бота з урахуванням бізнес-завдань і специфіки ресторану.

Наступним кроком є оцінка та порівняння графічних бібліотек, які будуть використовуватися для розробки користувацького інтерфейсу програми менеджера. Також на цьому етапі обираються потенційні рішення залежно від їхньої взаємодії, сумісності, експлуатаційних та необхідних аспектів стосовно конкретного підприємства. Таким чином, на основі зібраних даних приймається рішення про вибір найбільш підходящої графічної бібліотеки.

Завершальним етапом дослідження є аналіз програмного забезпечення для організації бази даних. Тут розглядаються різні варіанти з погляду їхньої продуктивності, масштабованості, безпеки та здатності інтегруватися з іншими компонентами системи з ціллю знайти оптимальне рішення, яке якнайкраще відповідає встановленим критеріям.

Поточний розділ починається з технічного дослідження і спрямований на вибір ключових компонентів для розроблення програмного комплексу, що забезпечує автоматизацію роботи ресторанів або ресторану.

2.1 Вимоги до розроблюваного програмного забезпечення

Під час проектування комплексу програмного забезпечення, що включає месенджер-бота для автоматичного приймання замовлень у ресторанах, а також програми-менеджера, яка оброблятиме замовлення, що надходитимуть, та даватиме змогу вносити зміни в меню страв, ключовим етапом є визначення вимог до системи.

Вимоги визначають основні параметри та характеристики, якими має володіти програмне забезпечення, щоб ефективно виконувати свою функцію в заданих умовах.

На першому етапі необхідно визначити ключові функціональні можливості програмного забезпечення. Для месенджер-бота це охоплює можливість приймання замовлень, опрацювання запитів клієнтів, а також інтеграцію із системами управління рестораном для оновлення статусів замовлень.

Програма-менеджер повинна надавати інтерфейс для редагування меню, обробки замовлень і аналізу звітів про продажі. Нефункціональні вимоги стосуються продуктивності, надійності, сумісності та зручності використання програмного забезпечення.

Важливо, щоб система була достатньо масштабованою, щоб справлятися з високими навантаженнями в години пік і забезпечувала безперервність роботи навіть при високому навантаженні.

Особливу увагу слід приділити безпеці даних. Необхідно забезпечити захист особистих даних клієнтів і фінансової інформації. Це охоплює шифрування даних, управління доступом і механізми аутентифікації.

Програмне забезпечення має бути здатне адаптуватися до збільшення обсягу замовлень і розширення функціоналу. Це вимагає гнучкої архітектури та можливості легкого оновлення системи.

Важливим аспектом є розробка ефективних методів тестування програмного забезпечення, щоб забезпечити його надійну роботу. Необхідно провести тестування продуктивності, тестування навантаження та тестування безпеки.

Ці вимоги слугують основою для подальших етапів розробки, починаючи з вибору месенджера для чат-бота.

До наступних кроків відносяться дослідження та порівняння існуючих API для обраного месенджера, вибір мови програмування для розробки програми-менеджера, порівняння графічних бібліотек для призначеного для користувацького інтерфейсу, порівняння програмного забезпечення баз даних, огляд сучасних

технологій машинного навчання та їхнього застосування в чат-ботах, а також аналіз найкращих практик у дизайні інтерфейсів чат-ботів.

Усе це забезпечує чітке розуміння цілей та обмежень проєкту, що є ключовим для успішного проєктування та реалізації системи.

2.2 Вибір месенджера для чат-бота

Виходячи з проведеного дослідження в першій частині дипломної роботи, було прийнято рішення розробляти чат-бота для месенджера Telegram. Цей вибір зумовлений кількома ключовими факторами, які роблять Telegram оптимальною платформою для реалізації проєкту в умовах українського ринку.

По-перше, Telegram демонструє все більше і більше зростання популярності в Україні.

По-друге, Telegram пропонує широкий спектр функціональних можливостей, придатних для комерційного використання. Можливість передачі фото, відео та текстових файлів, налаштування і розсилки реклами, ведення комунікації з клієнтами та збору даних створюють сприятливе середовище для інтеграції чат-ботів. Крім того, відсутність суворих обмежень на розсилку рекламного контенту в Telegram надає бізнесу більшу свободу для маркетингових стратегій.

Третім важливим аспектом є технічна підтримка та стабільність роботи платформи. Telegram відомий своєю надійністю і гнучкістю в інтеграції з різними зовнішніми сервісами, включно з CRM-системами. Це дає змогу більш ефективно обробляти замовлення та управляти клієнтськими даними.

По-четверте, наявність великої та активної бази користувачів в Україні робить Telegram привабливою платформою автоматизації прийому замовлень для місцевих ресторанів. Ба більше, можливість створення публічних сторінок і груп у Telegram спрощує процес взаємодії з клієнтами та поширення інформації про ресторани, спеціальні пропозиції та акції.

На основі цих суджень було ухвалено рішення вибрати месенджер Telegram для розробки чат-бота, призначеного для автоматизації процесу приймання та обробки замовлень у ресторанах. Його технологічні можливості, широке охоплення аудиторії в Україні й в більшості країн Європи його гнучкість в інтеграції з іншими системами та програмними продуктами забезпечують необхідні умови для успішної реалізації проєкту.

2.3 Порівняння існуючих API для обраного месенджера

У процесі розробки чат-бота для месенджера Telegram, ключовим аспектом є вибір відповідного API, який визначає можливості взаємодії бота з призначеним для користувача інтерфейсом і функціональністю месенджера. У межах цього дослідження було проаналізовано різні API і бібліотеки, запропоновані спільнотою розробників Telegram, оцінюючи їх за такими критеріями, як гнучкість, масштабованість, підтримка мов програмування і легкість інтеграції.

Першим об'єктом для дослідження в контексті розроблення програмних продуктів є фреймворк Nutgram, він представляє і себе популярне рішення створення чат-ботів на PHP. Логотип Nutgram, представлений на рисунку 2.1, символізує його простоту. Nutgram пропонує розробникам широкий спектр можливостей для створення чат-ботів, вирізняючись гнучкістю, масштабованістю, підтримкою мов програмування і легкістю інтеграції.

Гнучкість Nutgram полягає в його здатності підтримувати розробку як простих, так і складних чат-ботів. Це досягається за рахунок розширеного набору функцій, що надаються фреймворком, які дають змогу адаптувати продукт під різноманітні вимоги та умови використання. Адаптивність Nutgram є його сильною стороною, оскільки вона дає розробникам свободу у створенні чат-ботів, що повністю відповідають індивідуальним потребам [10]. Масштабованість Nutgram проявляється в його здатності ефективно справлятися з високими навантаженнями,

що особливо важливо для чат-ботів, призначених для обробки великого обсягу взаємодій.



Рисунок 2.1 – Логотип Telegram-бот фреймворку Nutgram

Використання останніх можливостей РНР 8.2 забезпечує зарекомендовану продуктивність й упевненість у тому, що фреймворк буде стабільно функціонувати як у невеликих, так і у великих проектах з інтенсивним трафіком.

Підтримка мов програмування є ключовим аспектом Nutgram, фреймворка, який повністю базується на РНР. Це робить його ідеальним вибором для розробників, які віддають перевагу цій мові, особливо в тих випадках, коли розробка програмного забезпечення заснована на РНР-рішеннях. Спеціалізація на РНР дає змогу розробникам уникати необхідності використання обгортки і додаткових сполучних застосунків, надаючи можливість працювати безпосередньо в єдиній екосистемі РНР. Завдяки цьому розробка та архітектура програмного забезпечення не вимагає інтеграції проміжних бібліотек або сервісів, які могли б ускладнити взаємодію між компонентами, що дає змогу мінімізувати додаткові залежності та реалізувати прямишу й ефективнішу взаємодію всередині системи.

Простота інтеграції Nutgram проявляється в його сумісності з іншими популярними РНР-фреймворками, такими як Laravel і Symfony. Це спрощує процес впровадження Nutgram у вже наявні системи та проекти, а також підтверджує його надійність у реальних виробничих умовах.

Як і всі сучасні програмні продукти, Nutgram містить у собі пакет тестування, який значно спрощує процес перевірки чат-ботів на помилки та надійність. Ця підвищує рівень довіри до продукту і сприяє створенню ефективних чат-ботів, що стабільно працюють.

На закінчення, Nutgram являє собою надійний, гнучкий і масштабований інструмент для розробки чат-ботів у середовищі Telegram, особливо для тих, хто має досвід роботи з РНР. Його переваги в царині гнучкості, масштабованості. Однак Nutgram є свої обмеження, які необхідно враховувати під час вибору фреймворка для розробки чат-боту. Основним недоліком є його залежність від РНР, що може бути обмежувальним фактором для розробки і реалізації ідеї проєкту. Це робить Nutgram менш універсальним порівняно з деякими іншими інструментами, які підтримують ширший діапазон мов програмування.

Крім того, незважаючи на значні поліпшення в продуктивності та масштабованості, представлені в РНР 8.2, використання фреймворку може надавати додаткові зусилля для того, щоб відповідати вимогам, оголошеними у підрозділі 2.1, і створювати додаткові труднощі під час реалізації проєктів, особливо якщо необхідно вдаватися до сторонніх сервісів і систем для розгортання. Це особливо актуально під час розроблення адміністративних додатків для персоналу, де можуть виникнути складнощі з інтеграцією та забезпеченням необхідного рівня взаємодії між різними компонентами системи.

Наступний на дослідження став фреймворк Telegram.Bot для .NET. Це спеціалізований інструмент для розробки ботів Telegram на основі платформи .NET. Цей фреймворк надає розробникам набір функцій, що забезпечують гнучкість, масштабованість, зручність використання і простоту інтеграції, що робить його популярним вибором серед .NET розробників. Гнучкість Telegram.Bot полягає в його здатності підтримувати як прості, так і більш складні проєкти.

Фреймворк включає функції, такі як робота з кількома ботами в одному застосунку, що дає змогу створювати різноманітні рішення, включно з ботами для ігор та інших сценаріїв із різним функціоналом. Це дає розробникам можливість адаптувати ботів під конкретні вимоги та завдання. Масштабованість Telegram.Bot забезпечується підтримкою стандартів .NET Standard 2.0 і .NET 6, що робить його сумісним із сучасними середовищами розробки та стабільним під час роботи в різних умовах експлуатації, починаючи з невеликих проєктів і закінчуючи великими системами з високими вимогами до навантаження.

Основна мова програмування фреймворка - C#, що робить його зручним для розробників, уже знайомих з екосистемою .NET. Це спрощує інтеграцію Telegram.Bot з іншими проєктами на базі .NET і використання наявних бібліотек та інструментів.

Процес інтеграції Telegram.Bot підтримується за рахунок сумісності з широким діапазоном .NET проєктів. Пакети розширень фреймворка полегшують розробку додаткових функцій, а підтримка Docker для веб-хуків спрощує розгортання та інтеграцію в різні середовища.

Telegram.Bot піддається регулярним тестуванням, включно з модульними тестами і тестами системної інтеграції, що забезпечує стабільність і надійність. Приклади тестів також слугують зразками для нових розробників, полегшуючи процес вивчення і застосування фреймворка на практиці [11].

Наступним на розгляді був фреймворк grammY, призначений для створення ботів Telegram рисунок 2.2. Цей фреймворк, орієнтований на використання TypeScript і JavaScript, є сучасним рішенням для розроблення в галузі чат-ботів, поєднуючи в собі гнучкість, масштабованість, мовну підтримку і простоту інтеграції. Гнучкість фреймворка grammY проявляється в його зручності використання, що робить його доступним як для початківців, так і для досвідчених розробників. Відкрита архітектура та можливість розширення через плагіни забезпечують значні можливості для налаштування та адаптації під специфічні вимоги проєкту. Це дає змогу створювати Telegram-боти з різноманітними функціями та рівнями складності, від простих інформаційних помічників до

складних інтегрованих систем. За з'явленням розробників масштабованість є ключовим аспектом у grammY. Фреймворк здатний ефективно працювати при збільшенні навантаження, що важливо для ботів, які стикаються зі зростаючим числом користувачів і взаємодій. Стійкість до масштабування гарантує, що боти, розроблені за допомогою grammY, продовжуватимуть функціонувати ефективно навіть за значного зростання активності [12].



Рисунок 2.2 – Логотип Telegram-бот фреймворку grammY

Підтримка мов програмування в grammY орієнтована на TypeScript і JavaScript, що працюють на платформах Node.js або Deno. Переваги TypeScript, включно зі зручними інструментами редактора і легкістю масштабування кодової бази, роблять його ідеальним вибором для розроблення інтерактивних і чуйних ботів. Орієнтація на ці мови робить фреймворк привабливим для розробників, які вже знайомі з ними, або бажають використовувати їхні можливості в розробці ботів.

Простота інтеграції - ще одна важлива характеристика grammY. Фреймворк забезпечує безшовну інтеграцію з веб-фреймворками та базами даних, що спрощує розробку. Підтримка Deno і документація, орієнтована на Node.js, відображають гнучкість фреймворка і його адаптивність до різних середовищ розробки.

Сумісність JavaScript-пакетів `grammY` з браузерами розширює можливості використання фреймворка, включно з розробкою на таких платформах, як Cloudflare Workers.

Останнім програмним забезпеченням, яке було розглянуто, є бібліотека `PyTelegramBotAPI`, призначена для створення ботів Telegram мовою Python. Вона вирізняється гнучкістю, масштабованістю та зручністю інтеграції, що робить її широко застосовуваною серед Python-розробників.

Гнучкість `PyTelegramBotAPI` полягає в підтримці як синхронних, так і асинхронних операцій. Це дає змогу розробникам ефективно керувати відгуком бота і створювати більш плавну взаємодію з користувачем. Важливою особливістю бібліотеки є асинхронна реалізація `Telebot (AsyncTeleBot)`, яка дає змогу виконувати запити до Telegram API в рамках окремих асинхронних завдань. Така архітектура особливо корисна для обробки великої кількості одночасних запитів, що підвищує загальну продуктивність бота при високих навантаженнях [13].

В аспекті масштабованості `PyTelegramBotAPI` підтримує поділ довгих повідомлень на кілька частин, що допомагає долати обмеження Telegram API щодо довжини текстів. Крім того, бібліотека надає можливість контролю кількості потоків, задіяних `TeleBot`. Це дає розробникам додаткові інструменти для управління ресурсами, що позитивно позначається на масштабованості під час роботи з великими обсягами повідомлень.

Програмне забезпечення `PyTelegramBotAPI` повністю орієнтовано на Python і підтримує версії Python 3.8-3.11, а також Pyru 3. Спеціалізація бібліотеки на Python робить її зручним вибором для розробників, які працюють у цій екосистемі, забезпечуючи сумісність із широким спектром інструментів і бібліотек мови.

У плані інтеграції, `PyTelegramBotAPI` характеризується простотою в освоєнні та використанні. Вона підтримує логування, що спрощує процес налагодження та моніторингу. Можливість використання проксі-серверів і гнучкі налаштування взаємодії з сервером Telegram додатково сприяють зручній інтеграції бота в різні умови та середовища.

Незважаючи на те, що PyTelegramBotAPI обмежена використанням тільки Python, її функціональність, підтримка асинхронності та зручність інтеграції роблять її потужним інструментом для створення ботів Telegram. Розробники, які віддають перевагу Python, знайдуть у цій бібліотеці гнучке й ефективне рішення для своїх проєктів. Виходячи з аналізу перерахованих вище програмних рішень для розробки ботів Telegram, можна зробити висновок, що кожна з них пропонує практично ідентичні функціональні рішення, але в рамках своєї екосистеми і мови програмування. Замість того щоб акцентувати увагу на порівнянні їхніх переваг або недоліків, розробникам рекомендується зосередитися на виборі екосистеми, що відповідає їхньому проєкту.

Незалежно від того, чи буде це бібліотека для Python або .NET, основна функціональність залишається незмінною, що дає змогу ухвалити рішення на основі інших чинників, таких як сумісність з наявними інструментами або мовними вподобаннями, без втрати в можливостях або продуктивності.

2.4 Вибір LLMA моделі для чат-бота

Існує кілька методів (розділ 1.2), які можуть бути використані при розробці чат-бота, і вони різняться залежно від конкретних характеристик, яких має досягти та чи інша система. Щодо вибору моделі для цього проєкту, то варто відзначити пріоритетність підходу, заснованого на глибокому навчанні, оскільки він пропонує більш глибоку обробку природної мови та кращий контроль поточного контексту діалогу. У порівнянні зі скінченними автоматами та методі правил та шаблонів, моделі глибокого навчання здатні реагувати на кілька раундів діалогу, включаючи складні запити; крім того, вони здатні автономно навчатися на величезній кількості інформації. Тому вибір моделі LLMA має велике значення для реалізації поставлених вимог дипломного проєкту – чат-бота, який оброблятиме замовлення в ресторані. Подібний бот може не лише приймати замовлення, обробляти

додаткові запити від користувача, вести активний діалог та швидко інформувати про статус замовлення. Для цього потрібні моделі, які можуть видавати подібні контекстно-адекватні відповіді з високою швидкістю та масштабованістю. Згідно з основним критерієм щодо вибору моделі, необхідно, щоб модель була сумісною з іншими компонентами системи та ефективно працювала в умовах високого рівня навантаження.

Наразі одним з найкращих сайтів для завантаження, вивантаження та навчання нейронних мереж є Hugging Face. Hugging Face – це відкрита платформа та екосистема в галузі машинного навчання, де люди можуть отримати доступ до тисяч попередньо навчених моделей для таких завдань, як NLP, CV, генерація тексту, переклад, узагальнення тексту та ін. Платформа призначена для побудови моделей на основі сучасної архітектури, такої як Transformer, і включає в себе зручні інструменти для їх реалізації в різних проектах.

Hugging Face має API та бібліотеки, які допомагають у створенні та оцінці моделей, що робить платформу придатною для авторів нейромережових рішень, дослідників, розробників та компаній.

На платформі Hugging Face можна знайти нейромережові мовні моделі, побудовані на таких архітектурах:

- Mistral;
- LLaMA 3.1;
- GEMINI;
- MPT (Mosaic Pretrained Transformer) ;
- Phi-3;
- Qwen-2;
- GPT-2/GPT-Neo/GPT-J;
- BERT (Bidirectional Encoder Representations from Transformers);
- T5 (Text-to-Text Transfer Transformer).

Ці моделі різняться за своїм устроєм і сферами застосування, проте їх об'єднує використання архітектури трансформера. Зокрема, такі моделі, як GPT-2/GPT-Neo/GPT-J, ґрунтуються на архітектурі decoder-only трансформера, що

оптимізована для генеративних завдань, як-от синтез тексту, автодоповнення та діалогові системи. З іншого боку, моделі типу BERT використовують архітектуру encoder-only трансформера, орієнтовану на розуміння тексту, включно із завданнями класифікації, вилучення сутностей і аналізу семантики. Моделі T5 поєднують переваги encoder-decoder трансформерів, що робить їх особливо ефективними для завдань перетворення тексту, як-от машинний переклад або текстова інтерпретація.

Більш сучасні архітектури, такі як Mistral, LLaMA, GEMINI, MPT, Phi-3 і Qwen-2, включають оптимізації, спрямовані на поліпшення продуктивності і зниження обчислювальних витрат. Наприклад, Mistral і LLaMA прагнуть зменшити кількість параметрів без шкоди для якості, що робить їх придатними для застосування на обмежених ресурсах. GEMINI і Phi-3 реалізують додаткові механізми самонавчання й адаптації, що підвищує їхню універсальність для різних типів завдань. Кожна з наведених моделей являє собою свій крок уперед у розробленні трансформерних архітектур, пропонуючи поліпшення для конкретних галузей машинного навчання[14].

Більшість перерахованих моделей мають закриту інформацію, однак щодо продукту LLaMA 3.1 можна знайти значну кількість відкритої документації, включно з описом архітектури, кодом і підходами до реалізації.

Основна структура трансформера в LLaMA 3.1 складається з двох ключових елементів: механізму уваги (Attention) і багатошарової нейронної мережі (Feed Forward Network, FFN), що показано на рисунку 2.3. Ці два компоненти виконують різні завдання: увага опрацьовує зв'язки між токенами у вхідній послідовності, а FFN виконує нелінійні перетворення, забезпечуючи більш складне представлення даних.

У дослідженнях [15, 16] наведено, що механізм уваги включає обчислення матриці уваги з використанням скалярного добутку ключів і запитів, масштабованого на корінь із розмірності ключів (рис. 2.4). Це дає змогу уникнути ефекту зникаючих градієнтів і зберігає стабільність навчання.

Використання багатоголової уваги (Multi-Head Attention) дає можливість одночасно обробляти кілька підпросторів уявлень, що забезпечує більш детальне вилучення інформації з вхідних даних. Кожна «голова» виконує лінійну проекцію ключів, запитів і значень, а результати об'єднуються для формування остаточного уявлення.

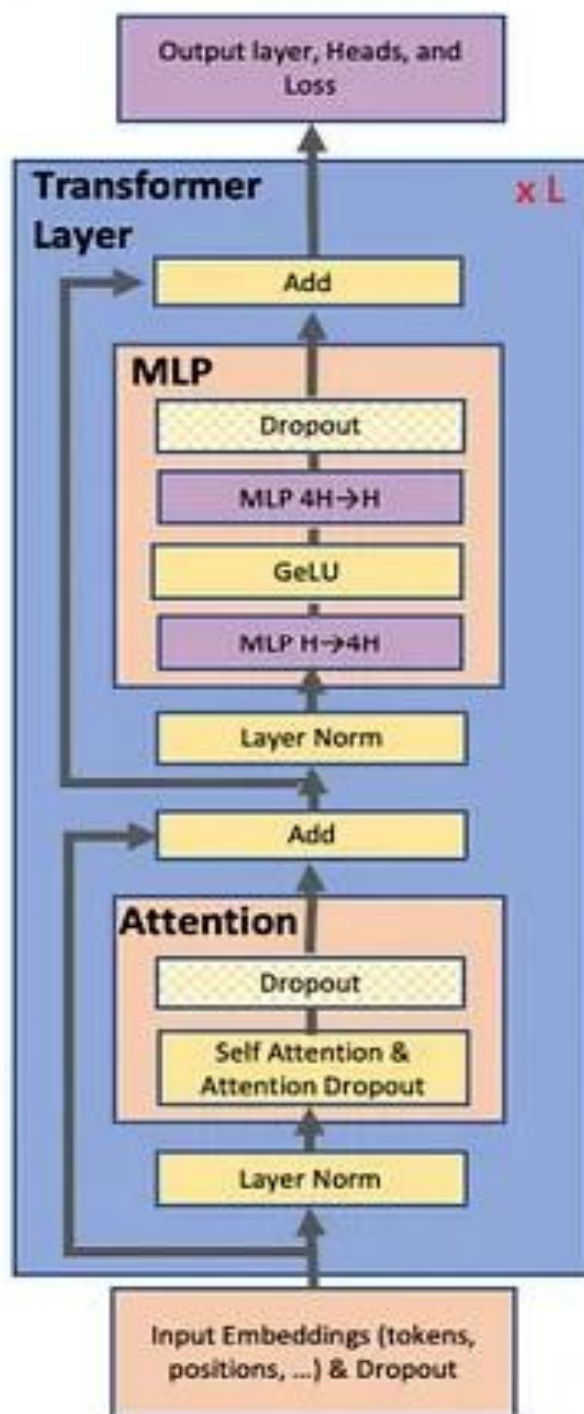
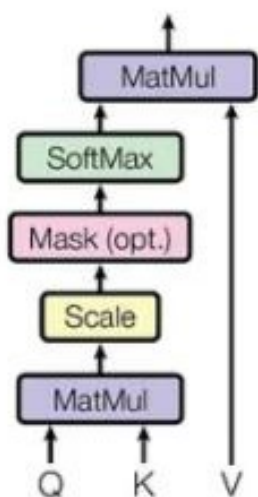


Рисунок 2.3 – Трансформаторна архітектура LLaMA 3.1

На наступному рівні модель реалізує токенизацію вхідних даних з використанням компонента `VocabParallelEmbedding`, який розподіляє обчислення між графічними процесорами. Токени перетворюються у векторний простір, після чого вони передаються в блоки трансформера (`TransformerBlock`). Кожен блок включає нормалізацію (`RMSNorm`), модуль уваги і FFN. Застосування нормалізації на рівні блоку підвищує стабільність градієнтів і прискорює навчання.

Модель LLaMA 3.1 оптимізована для розподіленого навчання та роботи в паралельних середовищах. Це досягається завдяки реалізації таких компонентів, як `ColumnParallelLinear` і `RowParallelLinear`, які розподіляють матричні операції за кількома графічними процесорами. Наприклад, операції множення матриць у шарі уваги або FFN розділяються по рядках або стовпчиках, як це показано на Рисунку 2.4. Така архітектура забезпечує значне прискорення роботи та зниження вимог до пам'яті.

Scaled Dot-Product Attention



Multi-Head Attention

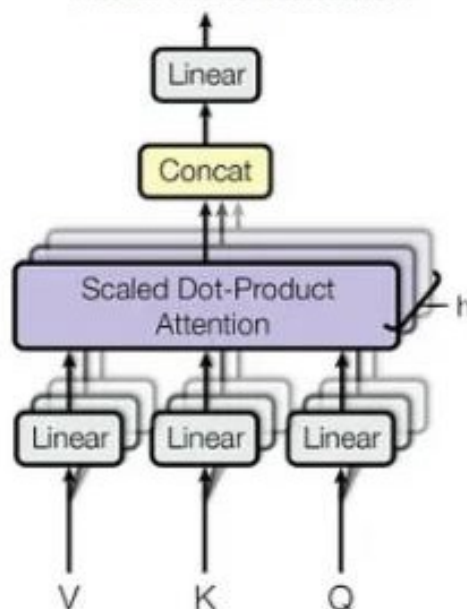


Рисунок 2.4 – Механізм уваги

На етапі виконання модель використовує кешування ключів і значень у механізмі уваги для зниження обчислювальних витрат. Це дає змогу зберігати проміжні результати й уникати їхнього повторного обчислення, що особливо

корисно під час опрацювання довгих послідовностей. Модель ефективно використовує розподілені обчислення, розділяючи словник та інші параметри між графічними процесорами. Наприклад, загальний розмір словника в LLaMA 3.1 становить 128256 токенів, які діляться порівну між GPU, що зменшує навантаження на кожний пристрій і прискорює процес навчання.

Виходячи з аналізу, можна зробити висновок, що архітектура LLaMA 3.1 демонструє поєднання сучасних підходів до роботи з трансформерами, оптимізацій для розподілених систем і високого ступеня масштабованості. Ці характеристики роблять модель універсальною і придатною для інтеграції в дипломний проєкт. З огляду на це модель LLaMA 3.1 було обрано в якості модкелі для чат-бота обробки запитів і взаємодії в месенджері.

2.5 Вибір мови програмування для розробки програми-менеджера

Провівши дослідження доступних бібліотек для месенджера Telegram, було вирішено вибрати Python як основну мову програмування для створення програми-менеджера, виходячи з вимог, висунутих у розділі 2.1. Цей вибір був обумовлений кількома важливими факторами, включно з гнучкістю, масштабованістю, безпекою даних і зручністю розробки.

Мова програмування Python вирізняється своєю потужною екосистемою і великою документацією, що робить його особливо привабливим для розроблення складних програмних систем, таких як програма-менеджер для ресторану. Під час аналізу доступних фреймворків для роботи з Telegram, Python виділяється завдяки таким бібліотекам, як PyTelegramBotAPI, що надають гнучкі та масштабовані рішення для створення месенджер-ботів.

Важливо зазначити, що розробка програми-менеджера вимагає не лише опрацювання замовлень і взаємодії з клієнтами, а й розробки графічного інтерфейсу. Тут Python також має значну перевагу завдяки своїй популярності та

безлічі доступних бібліотек для створення кросплатформного графічного інтерфейсу. Фреймворки, такі як Tkinter, PyQt, які розглядаються нижче, надають різноманітні можливості для розробки інтуїтивно зрозумілих і функціональних користувацьких інтерфейсів.

Крім цього, Python відомий своєю здатністю ефективно працювати з великими обсягами даних, що важливо для аналізу звітів про продажі та обробки замовлень. Його бібліотеки для роботи з базами даних, як-от SQLAlchemy і Django ORM, спрощують інтеграцію і забезпечують надійне управління даними.

У контексті нефункціональних вимог, як-от продуктивність, надійність і безпека, Python пропонує різні інструменти та бібліотеки для шифрування даних, управління доступом і аутентифікації, що є критично важливим для захисту особистих даних клієнтів і фінансової інформації. Крім того, мова має високу масштабованість, що дає змогу системі ефективно справлятися з високими навантаженнями і забезпечувати безперервність роботи.

Проте слід враховувати, що Python може бути не найкращим вибором у ситуаціях, що вимагають крайньої продуктивності або ресурсоемних обчислень. Однак його переваги в гнучкості, масштабованості та багатому наборі функцій переважають ці обмеження для поточного проєкту.

Отже, з огляду на всі аспекти, Python є оптимальним вибором для розробки програми-менеджера, поєднуючи в собі необхідні технічні характеристики та зручність розробки. Він забезпечує гнучкість, необхідну для адаптації до вимог, що змінюються, і розширення функціоналу, а також пропонує ефективні методи тестування для гарантії надійності та безпеки програмного забезпечення.

2.6 Порівняння графічних бібліотек для UI на Python

Для розроблення графічного інтерфейсу програми-менеджера було проведено порівняльний аналіз графічних бібліотек для створення

користувачького інтерфейсу (UI) на Python, розглядаючи їхню сумісність з обраною мовою програмування, зручність використання та адаптивність до різних проєктних вимог. Це дослідження особливо важливе, оскільки ефективність і зручність користувачького інтерфейсу безпосередньо впливають на сприйняття програмного продукту кінцевими користувачами. Вибір відповідної бібліотеки для UI є ключовим фактором, що визначає не тільки зовнішній вигляд і відчуття застосунку, а й його функціональність, розширюваність і можливості інтеграції.

Для цього було проведено аналіз низки популярних бібліотек, таких як Tkinter, PyQt, Kivy та інші, оцінюючи їх за кількома поставленими в розділі 2.1 параметрами. По-перше, особливу увагу було приділено сумісності цих бібліотек з Python, враховуючи, що ефективна інтеграція з обраною мовою програмування є основоположним критерієм. Це включає в себе оцінку легкості інтеграції бібліотеки в наявну кодову базу, а також підтримку мовних особливостей і структур Python.

Укупі з цим було проведено аналіз кожної бібліотеки з погляду її функціональних можливостей, включно з наявністю попередньо створених елементів управління, гнучкості налаштування дизайну і можливості створення динамічних, інтерактивних користувачьких інтерфейсів. Важливо враховувати не тільки багатство вбудованих функцій, а й ступінь кастомізації, яку пропонують ці інструменти. Також було приділено увагу зручності у використанні кожної бібліотеки, оцінюючи легкість вивчення і застосування, а також доступність документації та спільноти підтримки. Зручність використання важлива в розробці, оскільки вона потенційно може значно скоротити час розробки і підвищити ефективність робочого процесу.

Першою в розгляді стала бібліотека Tkinter, що заслуговує на особливу увагу в контексті розробки графічних інтерфейсів для Python, приклад інтерфейсу можна побачити на рисунку 2.5. Ця бібліотека, що виникла з мови програмування Tool Command Language (Tcl) і створена Джоном Оустерманом наприкінці 1980-х років, є прикладом класичного підходу до розроблення GUI. Її походження з академічного та інженерного середовища, а також її коріння в Unix-системах,

значною мірою визначають її функціональність і дизайн, що зберігають простоту і практичність досі [17].

Бібліотека Tkinter, включена до стандартної бібліотеки Python з випуском Python версії 1.1 1994 року, по праву вважається де-факто бібліотекою GUI для Python. Це підкреслює його довгострокову стабільність і надійність як інструменту для розробки користувацького інтерфейсу. Наявність докладної документації, доступної на офіційному сайті Python, робить його ще більш доступним для розробників усіх рівнів.

Однією зі значних переваг Tkinter є його включення до стандартної бібліотеки Python, що забезпечує його доступність практично скрізь, де встановлено Python. Це позбавляє від необхідності встановлення додаткових пакетів або налаштування складних середовищ розроблення, що особливо цінно для швидкого прототипування і розроблення простих проєктів. Ба більше, стабільність API Tkinter і його еволюційний розвиток роблять його надійним вибором для довгострокових проєктів.

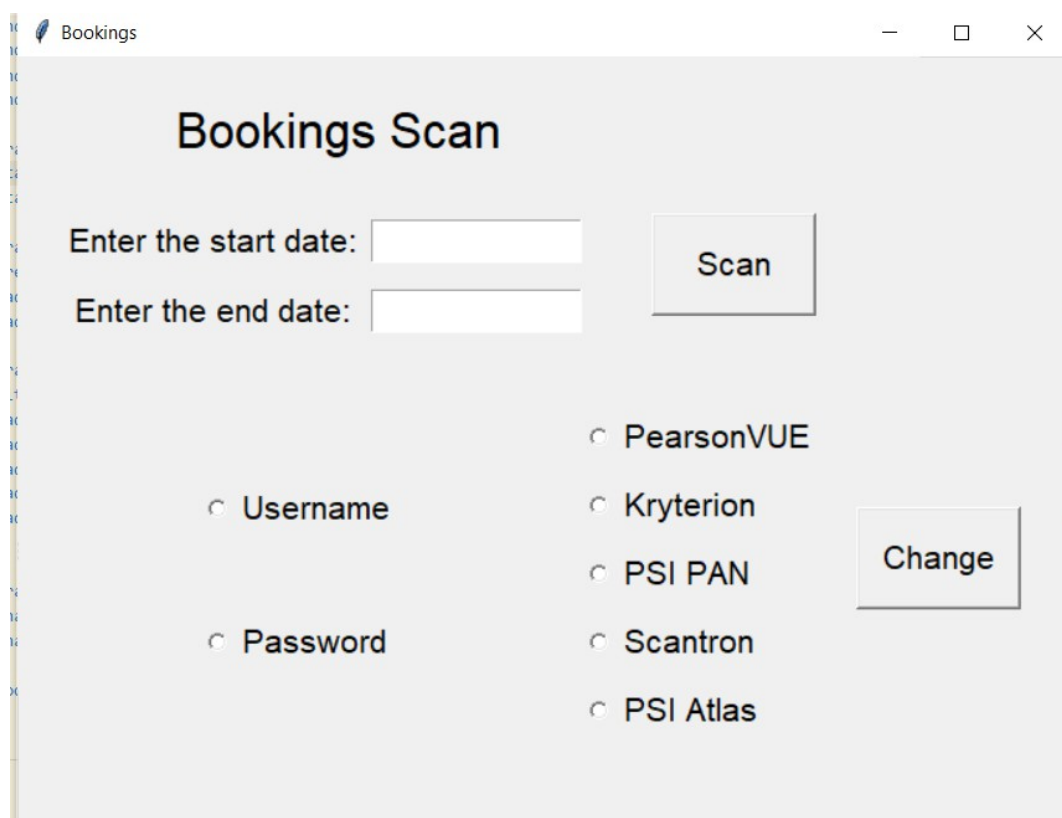


Рисунок 2.5 – Приклад реалізації графічного інтерфейсу за допомогою Tkinter

Бібліотека Tkinter надає функціональний і об'єктно-орієнтований підхід до створення GUI, що робить його простим і зрозумілим для програмістів, які звикли до стандартів класичного програмування. На відміну від деяких інших бібліотек, він не вимагає вивчення додаткових мов або парадигм, що знижує поріг входу і прискорює процес розробки.

Однак, незважаючи на ці переваги, Tkinter має низку недоліків. Зокрема, його зовнішній вигляд часто критикують за застарілі елементи дизайну, характерні для Unix-систем 1990-х років. Незважаючи на поліпшення, внесені останніми роками, включно з додаванням тематичних віджетів, Tkinter все ще може залишатися поза межами вибору для проєктів, що вимагають сучасного і стильного інтерфейсу. Також варто відзначити відсутність у Tkinter складних віджетів, наприклад інструменти для роботи з форматованим текстом або HTML-рендерингом, що може обмежувати його застосування в певних типах проєктів.

Вибираючи Tkinter для створення графічного інтерфейсу в Python, можна обґрунтувати його простоту, стабільність і доступність у стандартній бібліотеці Python. Він підходить для широкого спектра додатків, включно з утилітами, конфігураційними діалогами та іншими інструментами бізнес-логіки. Однак для проєктів, які потребують складнішого та сучаснішого дизайну, Tkinter пропонує розширення CustomTkinter. Це розширення призначене для того, щоб надати розробникам більшу гнучкість і поліпшені візуальні можливості під час роботи з Tkinter. CustomTkinter підсилює стандартний функціонал Tkinter, додаючи поліпшені елементи стилю та налаштування, що дає змогу створювати привабливіші та інтуїтивно зрозуміліші користувацькі інтерфейси, порівняння вигляду інтерфейсів можна побачити на рисунку 2.6 [18].

Розширення CustomTkinter приділяє особливу увагу сучасному дизайну та користувацькому досвіду, пропонуючи широкий вибір тематичних віджетів і стилів, які можна легко інтегрувати в наявні проєкти. Це робить його особливо привабливим для тих проєктів, у яких зовнішній вигляд і відчуття додатка відіграють ключову роль. Використання CustomTkinter дає змогу розробникам

подолати деякі обмеження класичного Tkinter, як-от його застарілий зовнішній вигляд, надаючи сучасні інтерфейсні компоненти, які можна адаптувати.

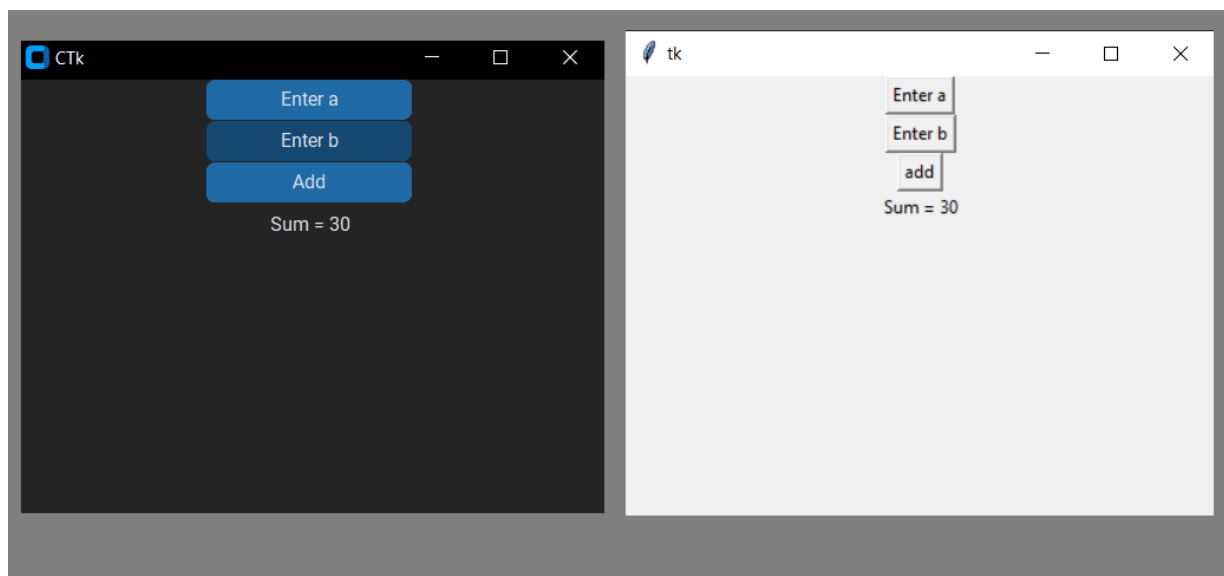


Рисунок 2.6 – Інтерфейс thinker в порівнянні з його розширенням CustomTkinter

Наступна популярна бібліотека для створення користувацького інтерфейсу на Python - PyQt. PyQt являє собою сполучну ланку, яка гармонійно поєднує в собі міць кросплатформенного середовища Qt C++ і гнучкість Python, виступаючи насамперед як ефективний модуль для створення графічного інтерфейсу (рисунок 2.7).

Середовище Qt представляє з себе не просто набір інструментів для графічного інтерфейсу, а комплексне рішення, що охоплює широкий спектр функціональних можливостей. У його арсеналі - абстракції для роботи з такими елементами, як мережеві сокети, потоки, Unicode, SQL, бази даних, SVG, OpenGL і XML, а також вбудований веб-браузер і просунута система обслуговування, підтримувана великою колекцією віджетів графічного інтерфейсу.

Однією з ключових особливостей Qt є його механізм сигналів і слотів, який відіграє важливу роль в організації взаємодії між об'єктами, полегшуючи створення компонентів програми, які повторно використовуються.

Набір розширень PyQt підсилює Qt за допомогою інтуїтивно зрозумілого інструменту для створення графічних інтерфейсів - Qt Designer. Поєднання PyQt і

Qt Designer унікальне тим, що дає змогу легко перетворювати проекти, створені в Qt Designer, на код Python, розширюючи можливості впровадження нових елементів керування графічним інтерфейсом і покращуючи процес розроблення.

Архітектура PyQt включає в себе кілька модулів, кожен з яких призначений для вирішення певних завдань. Ключові модулі включають QtCore для управління часом, файлами і процесами; QtGui для візуальних ефектів і графічних компонентів; QtNetwork для мережевого програмування; QtXml для роботи з XML-файлами; QtSvg для рендерингу SVG-графіки; QtOpenGL для рендерингу 3D і 2D графіки; і QSql для взаємодії з базами даних. Ці модулі забезпечують гнучкість і потужність PyQt, роблячи його одним із найкращих виборів для розроблення складних і функціонально насичених додатків [19].

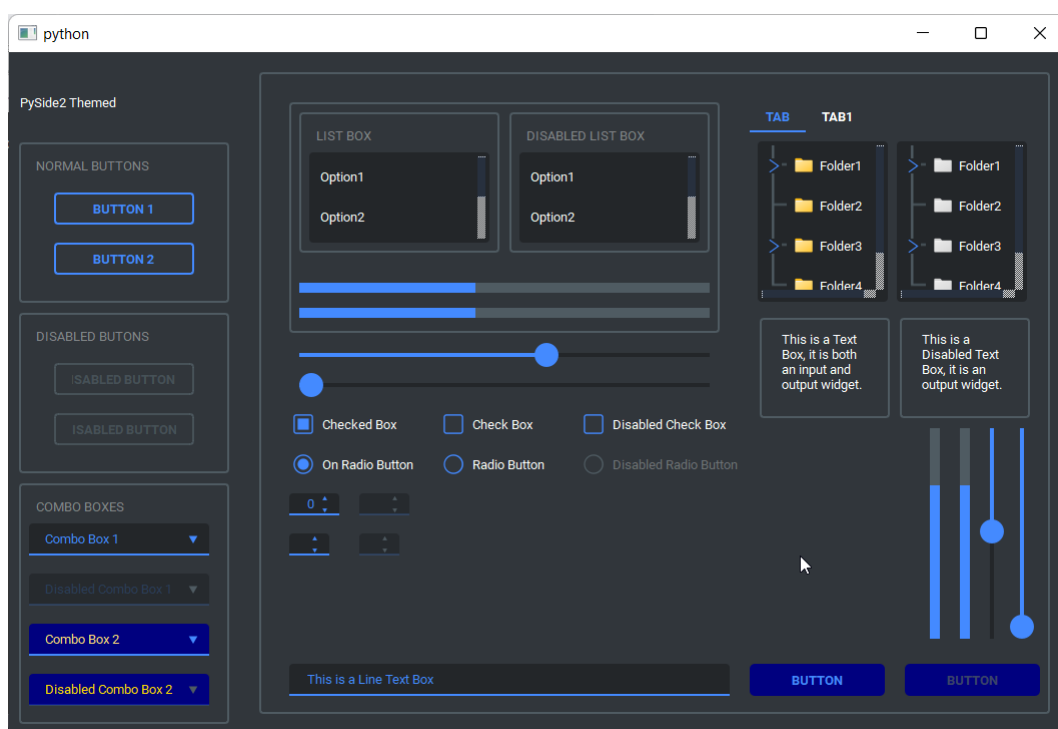


Рисунок 2.7 – Приклад графічного інтерфейсу реалізованого за допомогою PyQt

Набір розширень PyQt, зі своїми можливостями та модульністю, стає гарним вибором для тих проектів, де потрібен складніший і динамічніший користувацький інтерфейс, а також для ситуацій, де важливо інтегрувати різні технології та формати даних. Він підходить як для створення бізнес-додатків із багатим

функціоналом, так і для розроблення програм, що вимагають складних користувацьких інтерфейсів з інтенсивною взаємодією користувача.

У процесі пошуку, аналізу та дослідження графічних бібліотек було розглянуто ще одну бібліотеку wxPython (рисунок 2.8).

Цей інструментарій є кросплатформеним набором інструментів для створення настільних додатків із графічним інтерфейсом. Основне його завдання - надати розробникам можливість створювати додатки для операційних систем Windows, Mac і різних версій Unix. Особливість wxPython полягає в тому, що він є оболонкою для wxWidgets, зрілої кросплатформної бібліотеки C++.

Набір інструментів wxPython складається з п'яти основних модулів, кожен з яких виконує свої специфічні функції.

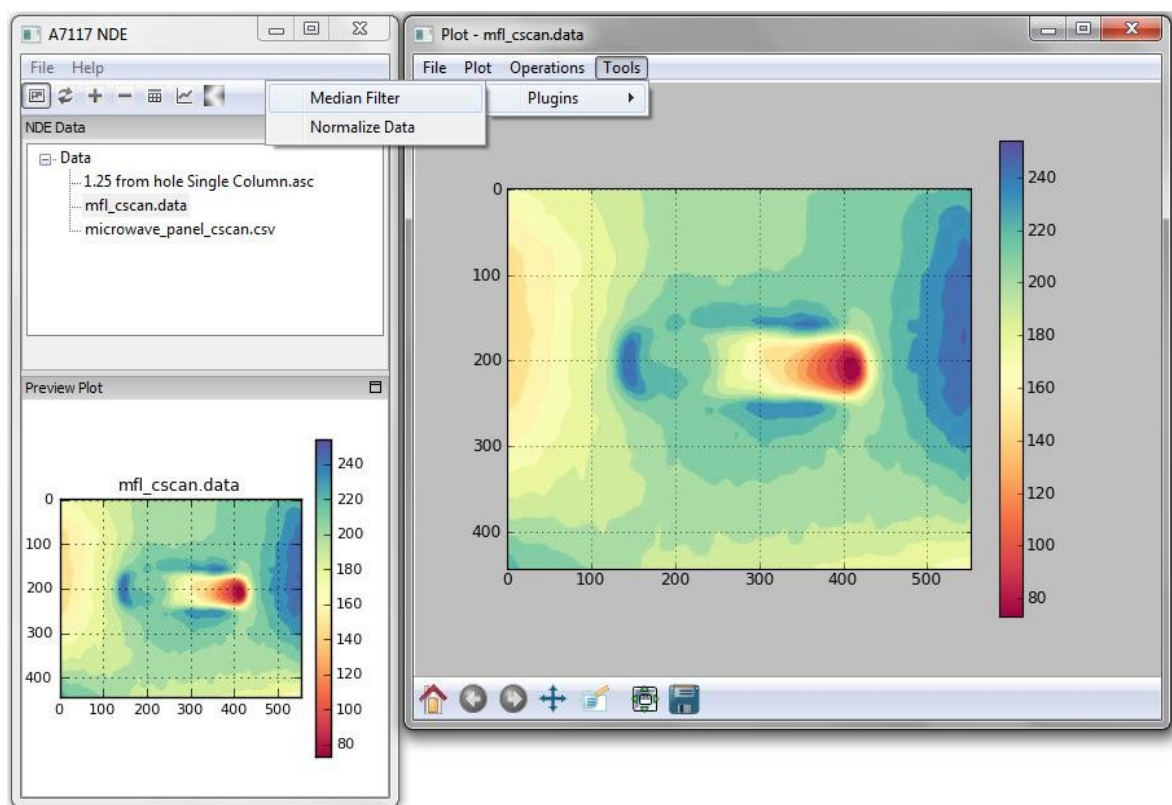


Рисунок 2.8 – Приклад графічного інтерфейсу реалізованого за допомогою wxPython

Модуль керування, наприклад, містить загальні віджети, які часто використовуються в графічних додатках, як-от кнопки, панелі інструментів або

блокноти. Важливо зазначити, що ці віджети в операційній системі Windows називаються елементами управління.

Далі йде модуль Core, який складається з фундаментальних класів, що використовуються в процесі розробки. Ці класи включають у себе клас Object, що є основою всіх класів, Sizers для макетування віджетів, Events і базові класи геометрії, такі як Point і Rectangle. Також варто виділити модуль GDI (Інтерфейс графічного пристрою), який містить класи для малювання на віджетах, керування шрифтами, кольорами, пензлями, пір'ям і зображеннями.

Модуль Misc являє собою збірку різних класів і функцій, які використовуються для логування, налаштування програми та системи, роботи з дисплеєм або джойстиком. Нарешті, модуль Windows включає в себе різні вікна, які формують додаток, такі як панелі, діалогові вікна, фрейми або прокручувані вікна [20].

Переходячи до wxPython API, варто зазначити, що він являє собою набір методів і об'єктів, серед яких особливу важливість мають віджети. Вони є основними будівельними блоками для будь-якої програми з графічним інтерфейсом. Віджети в wxPython можна поділити на кілька логічних груп, таких як базові віджети, віджети верхнього рівня, контейнери, динамічні та статичні віджети, а також інші віджети, які реалізують елементи інтерфейсу, такі як рядок стану, панель інструментів і рядок меню.

Особливо важливим є розуміння принципу успадкування в wxPython. Наприклад, віджет кнопки успадковується від чотирьох різних базових класів, включно з wx.Control і wx.Window. Це забезпечує гнучкість і міць у створенні користувацьких інтерфейсів. Успадкування в wxPython відіграє ключову роль, оскільки воно дає змогу віджетам успадковувати функціональність і властивості від своїх предків, що значно спрощує процес розробки.

Наприклад, віджет кнопки, успадковуючи від класу wx.Control, отримує всі необхідні атрибути керування, включно з подіями, стилями та поведінкою. Одночасно, завдяки успадкуванню від класу wx.Window, віджет має всі характеристики віконного елемента, як-от розмір, позиціонування та відтворення.

У рамках wxPython, успадкування також важливе для реалізації подієвої моделі. Кожен віджет може реагувати на різні події, такі як кліки миші, натискання клавіш клавіатури або зміни стану. Це досягається завдяки успадкуванню від класу `wx.EvtHandler`, який надає механізми для оброблення та передавання подій.

Таким чином, віджет кнопки, реагуючи на подію `wx.EVT_COMMAND_BUTTON_CLICKED`, дає змогу розробникам легко налаштувати відповідні дії під час взаємодії користувача з інтерфейсом.

Я вже зазначалось програмний продукт є кросплатформним. Бібліотека забезпечує однаковість розробки на різних операційних системах, що дає змогу розробникам писати код, який працюватиме на Windows, macOS і Unix-подібних системах без значних змін. Це досягається за рахунок того, що wxPython слугує обгорткою для нативних елементів керування кожної операційної системи, надаючи єдиний API для розробників.

Завершуючи аналіз бібліотеки можна сказати, що вона являє собою потужний і гнучкий інструмент для створення настільних додатків з графічним інтерфейсом. Її модульна структура, великий набір віджетів, потужні механізми успадкування і кросплатформеність роблять привабливим вибором для розробки сучасних додатків.

Завдяки цим особливостям, wxPython залишається одним із бажаних інструментів серед розробників, які прагнуть створити надійні та функціонально багаті додатки на різних платформах.

2.7 Вибір графічної бібліотеки для програми-менеджера

Виходячи з основних вимог, описаних у пунктах 2.1 і 2.5, і з огляду на попередній вибір мови програмування Python для проєкту, було ухвалено рішення використати бібліотеку CustomTkinter для розроблення користувацького інтерфейсу програми-менеджера. Цей вибір підкріплюється кількома ключовими

аспектами, які гармонійно поєднуються із заявленими критеріями та особливостями Python.

Бібліотека CustomTkinter, як розширення класичної бібліотеки Tkinter, чудово інтегрується з Python, забезпечуючи гладку взаємодію і повну сумісність з мовою. Це забезпечує безпроблемну роботу бібліотеки в рамках Python-проектів, виключаючи можливі конфлікти і підтримуючи типові для Python структури і парадигми програмування.

Можливості CustomTkinter у контексті створення функціональних і візуально привабливих користувацьких інтерфейсів виходять на передній план. Бібліотека пропонує різноманітні елементи керування і підтримує створення інтерактивних UI, що робить її ідеальним інструментом для реалізації як стандартних, так і складніших користувацьких сценаріїв.

Гнучкість налаштування і кастомізації в CustomTkinter дає змогу детально налаштовувати зовнішній вигляд і поведінку елементів керування. Це важливо для створення унікального користувацького інтерфейсу, який буде точно відповідати вимогам і перевагам проєкту, покращуючи враження користувачів від роботи з додатком.

Простота використання бібліотеки - ще одна важлива перевага. CustomTkinter вирізняється інтуїтивно зрозумілим інтерфейсом і легкістю в опануванні, що значно скорочує час на навчання та інтеграцію в проєкт. Підтримка з боку активної спільноти і наявність докладної документації додатково спрощують процес роботи з бібліотекою.

Вибір CustomTkinter сприяє оптимізації процесу розробки. Завдяки наявності готових компонентів і простому API, розробники можуть сфокусуватися на важливіших аспектах проєкту, як-от розробка бізнес-логіки, замість витратити час на створення елементів інтерфейсу з нуля. Це збільшує продуктивність і прискорює реалізацію проєкту.

Таким чином, вибір CustomTkinter для розроблення користувацького інтерфейсу програма менеджера є логічним та обґрунтованим у контексті використання Python як основної мови програмування, з огляду на її сумісність,

функціональні можливості, гнучкість, зручність використання та внесок у пришвидшення процесу розробки.

2.8 Вибір програмного забезпечення баз даних

У процесі вибору програмного забезпечення для баз даних у межах розроблення дипломного проекту, ґрунтуючись на аналізі наявних технологій і вимогах до створення ПЗ, важливо враховувати кілька ключових чинників.

Перший фактор - сумісність із Python. База даних повинна легко інтегруватися з Python, оскільки це основна мова програмування розроблюваного проекту. Це забезпечить безперебійну роботу між розробленими програмами і базою даних.

Другий фактор - підтримка роботи з Telegram Bot API. Оскільки проект включає Telegram-бота, то база даних має бути здатна ефективно обробляти запити і відповіді, що виникають у процесі взаємодії бота з користувачами.

Третій фактор - масштабованість і продуктивність. База даних має бути масштабованою, щоб справлятися з можливим збільшенням обсягу замовлень і користувачів, а також забезпечувати високу продуктивність у години пік.

Четвертий фактор - безпека. Важливим аспектом є захист даних клієнтів і фінансової інформації. Обрана база даних повинна забезпечувати шифрування даних, управління доступом і надійні механізми аутентифікації.

П'ятий фактор - гнучкість і можливість оновлення. База даних має бути гнучкою, щоб підтримувати розширення функціоналу та легко інтегруватися з новими компонентами системи.

Враховуючи це було розглянуто кілька потенційних варіантів баз даних. PostgreSQL являє собою потужну і масштабовану систему управління базами даних. Ця система добре інтегрується з Python, пропонує сильні функції безпеки та вирізняється високою продуктивністю.

Інший варіант - MySQL, популярна реляційна база даних, широко використовувана у веб-додатках. Вона сумісна з Python і пропонує хороші можливості масштабування і безпеки, хоча в деяких аспектах може бути менш гнучкою порівняно з PostgreSQL.

База даних MongoDB може бути хорошим вибором. Коли застосунок вимагає гнучкості в структурі даних або ви працюєте з великими обсягами неструктурованих даних. Це документо-орієнтована NoSQL база даних, що забезпечує високу продуктивність і легкість масштабування.

Ще однією популярною базою даних є SQLite: для маленьких проєктів або ситуацій, коли потрібна вбудована база даних, SQLite стає зручним вибором. Він легко інтегрується з Python і чудово підходить для додатків із низькими вимогами до масштабованості та складності бази даних.

Грунтуючись на описаних вище вимогах для проєкту, який включає месенджер-бота для автоматичного приймання замовлень у ресторанах і програму-менеджера для обробки цих замовлень та управління редагуванням меню їжі, найвідповіднішим варіантом бази даних є PostgreSQL. Це зумовлено низкою ключових чинників, які роблять PostgreSQL ідеальним рішенням для цього проєкту.

Перш за все, PostgreSQL пропонує великі можливості для роботи з різноманітними типами даних, що є важливим аспектом для системи, яка обробляє замовлення в ресторані. Ця гнучкість особливо корисна під час опрацювання різноманітних даних, включно з інформацією про замовлення, даними клієнтів, а також деталями та змінами в меню. PostgreSQL підтримує як традиційні структури даних, так і JSON-подібні неструктуровані формати, що дає змогу легко інтегрувати й обробляти дані, які надходять як із месенджер-бота, так і з програми-менеджера.

Далі, важливим аспектом є продуктивність і масштабованість PostgreSQL. Як система управління базами даних, PostgreSQL відома своєю здатністю ефективно обробляти великі обсяги даних і високі навантаження, що критично важливо для системи, призначеної для роботи в умовах ресторанної індустрії, де обсяги

замовлень і дані клієнтів можуть різко зростати в певні періоди. PostgreSQL забезпечує високу продуктивність навіть в умовах інтенсивного потоку даних і запитів, що гарантує стабільність і надійність системи в години пік. Безпека є ще одним критично важливим фактором під час вибору бази даних. PostgreSQL пропонує просунуті функції безпеки, включно з потужними засобами шифрування даних і керування доступом. Це забезпечує захист конфіденційної інформації, такої як особисті дані клієнтів і фінансові транзакції. У контексті месенджер-ботів і систем обробки замовлень, де безпека даних має першорядне значення, PostgreSQL надає необхідні інструменти для забезпечення цілісності та конфіденційності даних.

Ще, слід зазначити, що PostgreSQL має широке співтовариство і велику документацію, що значно полегшує розробку і підтримку. Наявність активного співтовариства розробників і великої бази знань дає змогу швидко знаходити розв'язання проблем, що виникають, і полегшує процес інтеграції бази даних з іншими компонентами системи, як-от месенджер-боти та користувацькі інтерфейси. Таким чином, вибір PostgreSQL як бази даних для проєкту, що містить месенджер-бота для автоматичного приймання замовлень у ресторанах і програму-менеджера для їхнього опрацювання та управління меню, є обґрунтованим і доцільним рішенням. Цей вибір підкріплюється вищеописаними ключовими факторами, які роблять PostgreSQL придатною для реалізації дипломного проєкту.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Підготовка середовища розробки

3.1.1 Встановлення Python і необхідних бібліотек

З огляду на аналіз фреймворків, описаних у другій частині дипломної роботи, як основу для розробки системи обрано інтерпретатор Python. Процес

встановлення та налаштування Python буде описано нижче на прикладі операційної системи Windows 11, оскільки вона використовується для реалізації дипломного проєкту. Водночас надані інструкції та код є універсальними і можуть бути застосовані на інших операційних системах, таких як Linux і macOS.

Для встановлення Python необхідно відвідати офіційний сайт Python (python.org) і завантажити актуальну версію інтерпретатора, сумісну з операційною системою. Після завантаження інсталяційного файлу запуск файлу ініціює процес встановлення (рисунк 3.1). Під час інсталяції було активовано опцію «Add Python to PATH», для забезпечення зручного доступу до Python з командного рядка Windows.



Рисунок 3.1 – Головне вікно інсталяції Python

Наступним етапом є встановлення бібліотек, необхідних для роботи проєкту. Для цього була створена робоча папка проєкту Program_RestoOrderManager (рисунк 3.2). Потім у ній було створено файл requirements.txt, у якому будуть перелічені всі залежності, необхідні для розробки та виконання програми. Цей файл дозволить встановити всі потрібні бібліотеки за допомогою однієї команди.

Крім файлу залежностей, у папці проєкту було створено дві основні директорії. Перша – `orderManager`, призначена для розроблення програми обробки замовлень. У цій директорії розміщуються файли і модулі, пов'язані з функціональністю управління замовленнями, включно із взаємодією з базою даних і обробкою логіки замовлень.

Друга директорія – `tg-bot`, де будуть розташовуватися файли програми Telegram-бота.

Для ізоляції залежностей необхідно виконати таку команду:

```
C:\> python -m venv envRestoOrderManager
```

Це створить папку `envRestoOrderManager` всередині папки проєкту, яка міститиме локальне оточення Python з ізольованими бібліотеками. Таке оточення дасть змогу встановити і використовувати залежності, необхідні тільки для цього проєкту, без впливу на глобальну систему.

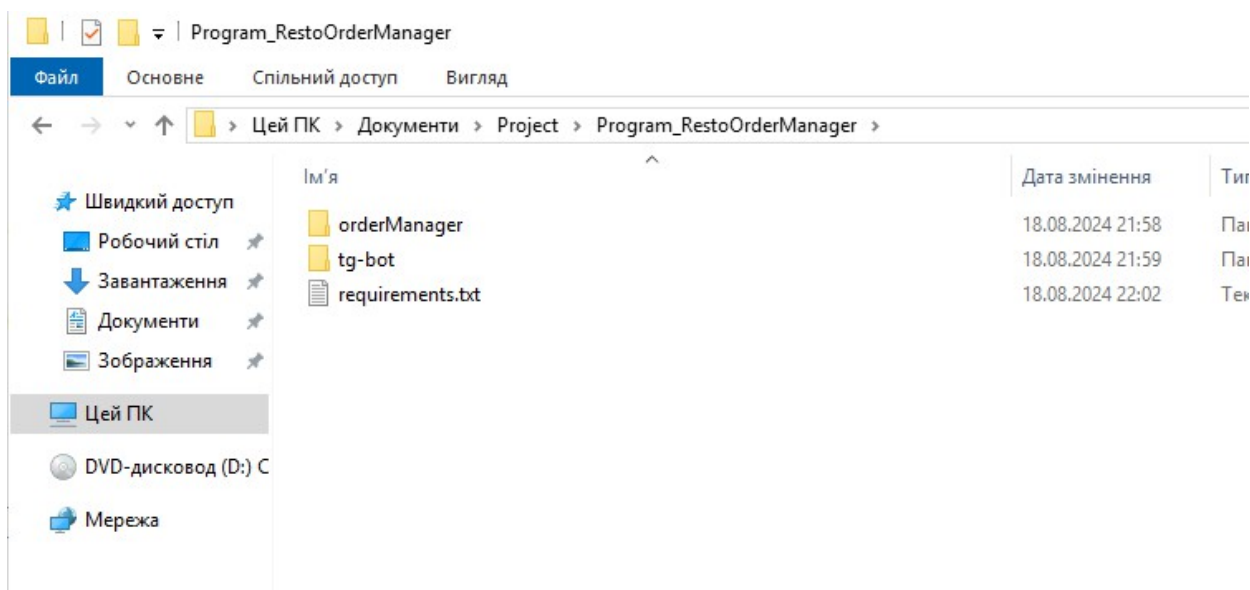


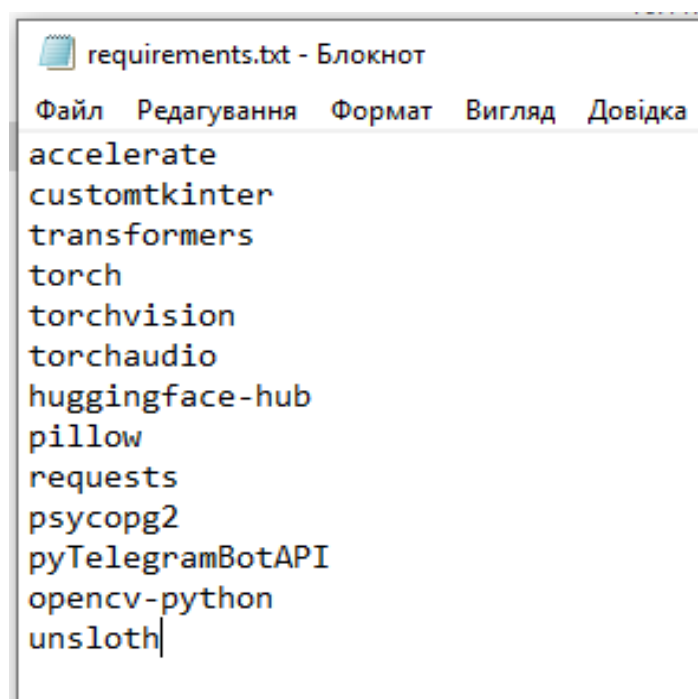
Рисунок 3.2 – Робоча папка проєкту

Список бібліотек та їх призначення:

- `accelerate` – використовується для оптимізації виконання нейронних моделей і роботи з прискорювачами, такими як GPU;
- `customtkinter` – бібліотека для створення графічного інтерфейсу програми;

- transformers – необхідна для роботи з попередньо навчальними мовними моделями, такими як GPT, для обробки текстових даних;
- torch, torchvision, torchaudio – основні бібліотеки для роботи з моделями глибокого навчання, включаючи обробку зображень та аудіо;
- huggingface-hub – забезпечує інтеграцію з платформою Hugging Face для завантаження та використання готових моделей і даних;
- pillow – використовується для обробки зображень (завантаження, перетворення, збереження);
- requests – бібліотека для виконання HTTP-запитів і взаємодії з API;
- psycopg2 – бібліотека для роботи з базою даних PostgreSQL;
- pyTelegramBotAPI – бібліотека для створення Telegram-бота;
- opencv-python – використовується для обробки зображень і відео;
- unsloth – допоміжна бібліотека для навчання нейронних мереж;

На рисунку 3.3 представлено вміст файлу requirements.txt, в якому написано через нову строку перелік усіх бібліотек, необхідних для реалізації системи. Цей файл спрощує процес налаштування середовища розробки та дає змогу швидко і централізовано встановити всі залежності.



```
requirements.txt - Блокнот
Файл  Редагування  Формат  Вигляд  Довідка
accelerate
customtkinter
transformers
torch
torchvision
torchaudio
huggingface-hub
pillow
requests
psycopg2
pyTelegramBotAPI
opencv-python
unsloth|
```

Рисунок 3.3 – Список усіх бібліотек

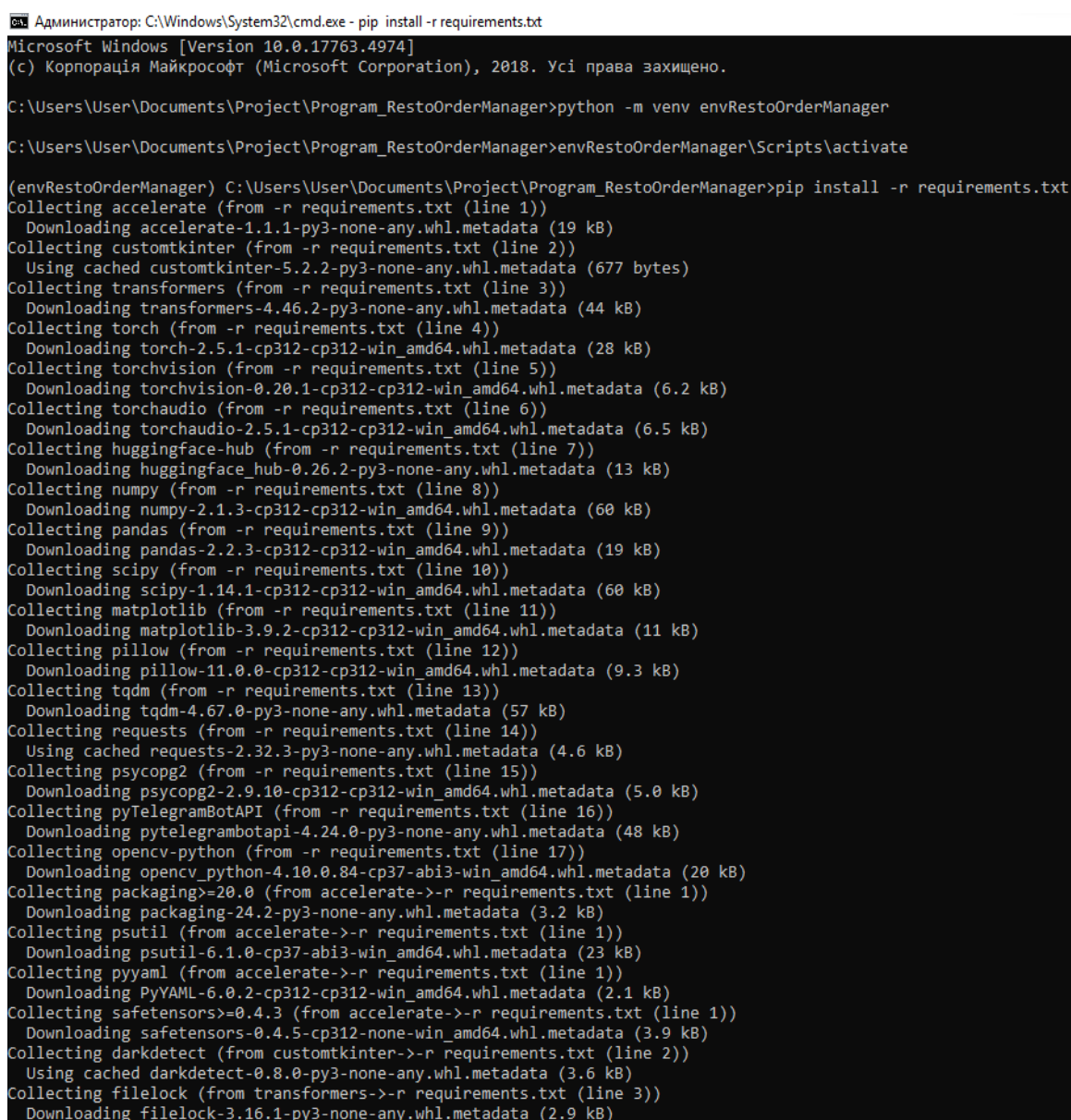
Для того щоб встановити перераховані вище бібліотеки необхідно спочатку активувати віртуальне оточення за допомогою команди:

```
C:\> RestoOrderManager>envRestoOrderManager\Scripts\activate
```

А потім встановити бібліотеки за допомогою наступної команди:

```
C:\> pip install -r requirements.txt
```

Результат виконання послідовності зазначених команд представлено на рисунку 3.4, де зображено як сам процес, так і процес початку встановлення необхідних бібліотек.



```
Администратор: C:\Windows\System32\cmd.exe - pip install -r requirements.txt
Microsoft Windows [Version 10.0.17763.4974]
(c) Корпорація Майкрософт (Microsoft Corporation), 2018. Усі права захищено.

C:\Users\User\Documents\Project\Program_RestoOrderManager>python -m venv envRestoOrderManager

C:\Users\User\Documents\Project\Program_RestoOrderManager>envRestoOrderManager\Scripts\activate

(envRestoOrderManager) C:\Users\User\Documents\Project\Program_RestoOrderManager>pip install -r requirements.txt
Collecting accelerate (from -r requirements.txt (line 1))
  Downloading accelerate-1.1.1-py3-none-any.whl.metadata (19 kB)
Collecting customtkinter (from -r requirements.txt (line 2))
  Using cached customtkinter-5.2.2-py3-none-any.whl.metadata (677 bytes)
Collecting transformers (from -r requirements.txt (line 3))
  Downloading transformers-4.46.2-py3-none-any.whl.metadata (44 kB)
Collecting torch (from -r requirements.txt (line 4))
  Downloading torch-2.5.1-cp312-cp312-win_amd64.whl.metadata (28 kB)
Collecting torchvision (from -r requirements.txt (line 5))
  Downloading torchvision-0.20.1-cp312-cp312-win_amd64.whl.metadata (6.2 kB)
Collecting torchaudio (from -r requirements.txt (line 6))
  Downloading torchaudio-2.5.1-cp312-cp312-win_amd64.whl.metadata (6.5 kB)
Collecting huggingface-hub (from -r requirements.txt (line 7))
  Downloading huggingface_hub-0.26.2-py3-none-any.whl.metadata (13 kB)
Collecting numpy (from -r requirements.txt (line 8))
  Downloading numpy-2.1.3-cp312-cp312-win_amd64.whl.metadata (60 kB)
Collecting pandas (from -r requirements.txt (line 9))
  Downloading pandas-2.2.3-cp312-cp312-win_amd64.whl.metadata (19 kB)
Collecting scipy (from -r requirements.txt (line 10))
  Downloading scipy-1.14.1-cp312-cp312-win_amd64.whl.metadata (60 kB)
Collecting matplotlib (from -r requirements.txt (line 11))
  Downloading matplotlib-3.9.2-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting pillow (from -r requirements.txt (line 12))
  Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl.metadata (9.3 kB)
Collecting tqdm (from -r requirements.txt (line 13))
  Downloading tqdm-4.67.0-py3-none-any.whl.metadata (57 kB)
Collecting requests (from -r requirements.txt (line 14))
  Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting psycpg2 (from -r requirements.txt (line 15))
  Downloading psycpg2-2.9.10-cp312-cp312-win_amd64.whl.metadata (5.0 kB)
Collecting pyTelegramBotAPI (from -r requirements.txt (line 16))
  Downloading pytelegrambotapi-4.24.0-py3-none-any.whl.metadata (48 kB)
Collecting opencv-python (from -r requirements.txt (line 17))
  Downloading opencv-python-4.10.0.84-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting packaging>=20.0 (from accelerate->-r requirements.txt (line 1))
  Downloading packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting psutil (from accelerate->-r requirements.txt (line 1))
  Downloading psutil-6.1.0-cp37-abi3-win_amd64.whl.metadata (23 kB)
Collecting pyyaml (from accelerate->-r requirements.txt (line 1))
  Downloading PyYAML-6.0.2-cp312-cp312-win_amd64.whl.metadata (2.1 kB)
Collecting safetensors>=0.4.3 (from accelerate->-r requirements.txt (line 1))
  Downloading safetensors-0.4.5-cp312-none-win_amd64.whl.metadata (3.9 kB)
Collecting darkdetect (from customtkinter->-r requirements.txt (line 2))
  Using cached darkdetect-0.8.0-py3-none-any.whl.metadata (3.6 kB)
Collecting filelock (from transformers->-r requirements.txt (line 3))
  Downloading filelock-3.16.1-py3-none-any.whl.metadata (2.9 kB)
```

Рисунок 3.4 – Процес створення віртуального оточення та встановлення залежностей

3.1.2 Встановлення і налаштування PostgreSQL

Для встановлення СУБД PostgreSQL було завантажено інсталяційний файл з офіційного сайту PostgreSQL (www.postgresql.org). Процес установки включав кілька етапів, зокрема вибір директорії для збереження файлів програми та даних бази, налаштування порту доступу до бази даних, а також визначення облікового запису суперкористувача, що матиме повний доступ до системи. Для роботи було обрано стандартний порт 5432, що забезпечує доступ до сервера бази даних.

Під час налаштування була вказана спеціально відведене директорія для збереження БД. Було також визначено каталог для встановлення утиліт командного рядка та інструменту адміністрування pgAdmin.

На рисунку 3.5 відображено результати завершального етапу установки PostgreSQL, зокрема вибрані каталоги для файлів програми, даних бази, а також інформація про службу бази даних та її конфігурацію.

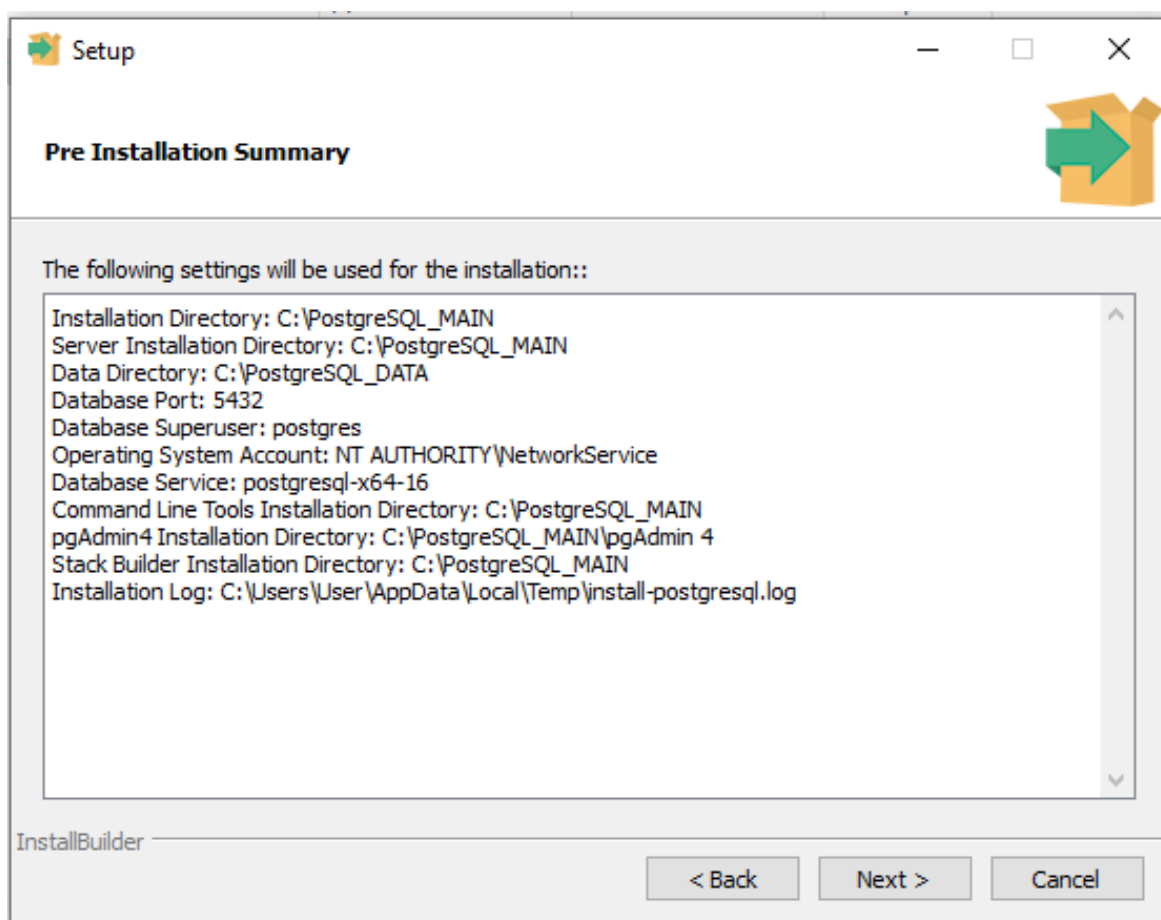


Рисунок 3.5 – Підсумкове налаштування параметрів встановлення PostgreSQL

У процесі тестування системи, після запуску SQL Shell і виконання авторизації за допомогою введення пароля суперкористувача postgres, у консолі з'явилося попередження. Це попередження стосувалося проблеми з кодуванням консолі Windows (1251) і можливих помилок у відображенні символів. У тексті попередження вказувалося, що через невідповідність кодування деякі символи можуть відображатися некоректно.

Для перевірки створених баз даних було введено команду \l, яка відображає список усіх баз даних. Однак отримана таблиця мала некоректне відображення символів, з текстовою інформацією української мови (рисунк 3.6). Це пов'язано з тим, що кодування, налаштоване в PostgreSQL, і кодування консолі Windows не узгоджуються, що призводить до некоректного відображення інформації.

Незважаючи на цю проблему, серед відображених баз даних було на рисунку 3.6 видно створену тестову базу даних test і вивід її вмісту, що підтверджує успішне налаштування і роботу системи.

Для вирішення проблеми з некоректним відображенням символів у SQL Shell необхідно внести зміни у параметри запуску консолі. Це дозволить налаштувати відповідне кодування, що забезпечить коректне відображення тексту.

```

SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль користувача postgres:
psql (16.4)
УВАГА: Кодова сторінка консолі (866) відрізняється від кодової сторінки Windows (1251)
8-бітові символи можуть працювати неправильно. Детальніше у розділі?
"Нотатки для користувачів Windows" у документації psql.
Введіть "help", щоб отримати допомогу.

postgres=# \l

```

ім'я бази	тип	кодування	бібліотека	хазард	хазард	хазард	хазард	хазард	хазард
postgres	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251				
template0	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251				
template1	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251				
test	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251				

```

(4 рядків)

postgres=# \c test
postgres=# \c test
Ви тепер знаходитесь у базі даних "test" як користувач "postgres".
test=# \dt

```

схема	ім'я таблиці	тип	хазард
public	test_table	table	postgres

```

(1 рядок)

test=#

```

Рисунок 3.6 – Відображення таблиці баз даних у SQL Shell з некоректним відображенням

Першим кроком є відкриття ярлика SQL Shell і перехід до його властивостей. Як видно на рисунку 3.7, у розділі «Об’єкт» вказано шлях до виконуваного файлу runpsql.bat, який відповідає за запуск консолі. Для подальшого налаштування потрібно відкрити цей файл у текстовому редакторі.

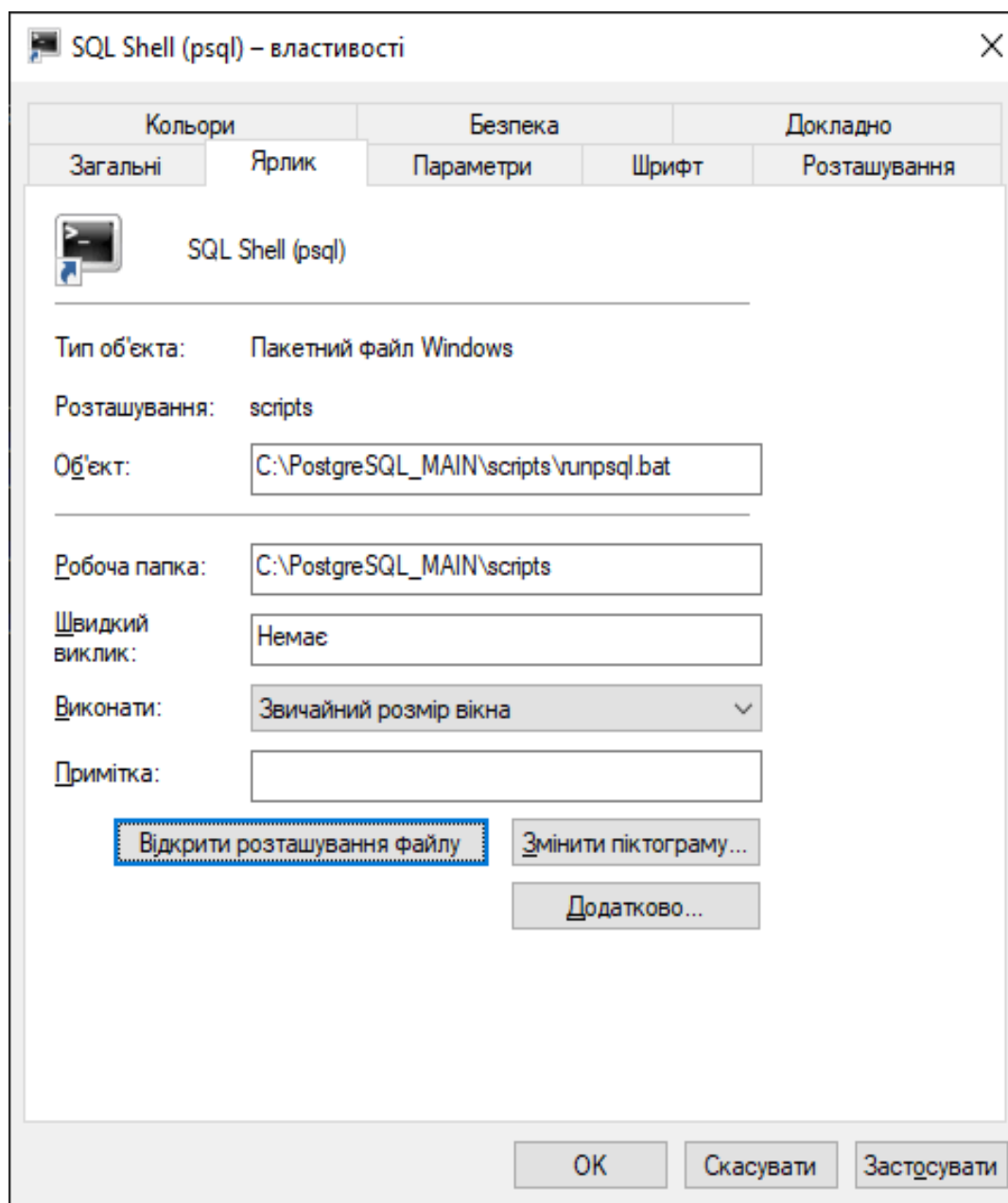
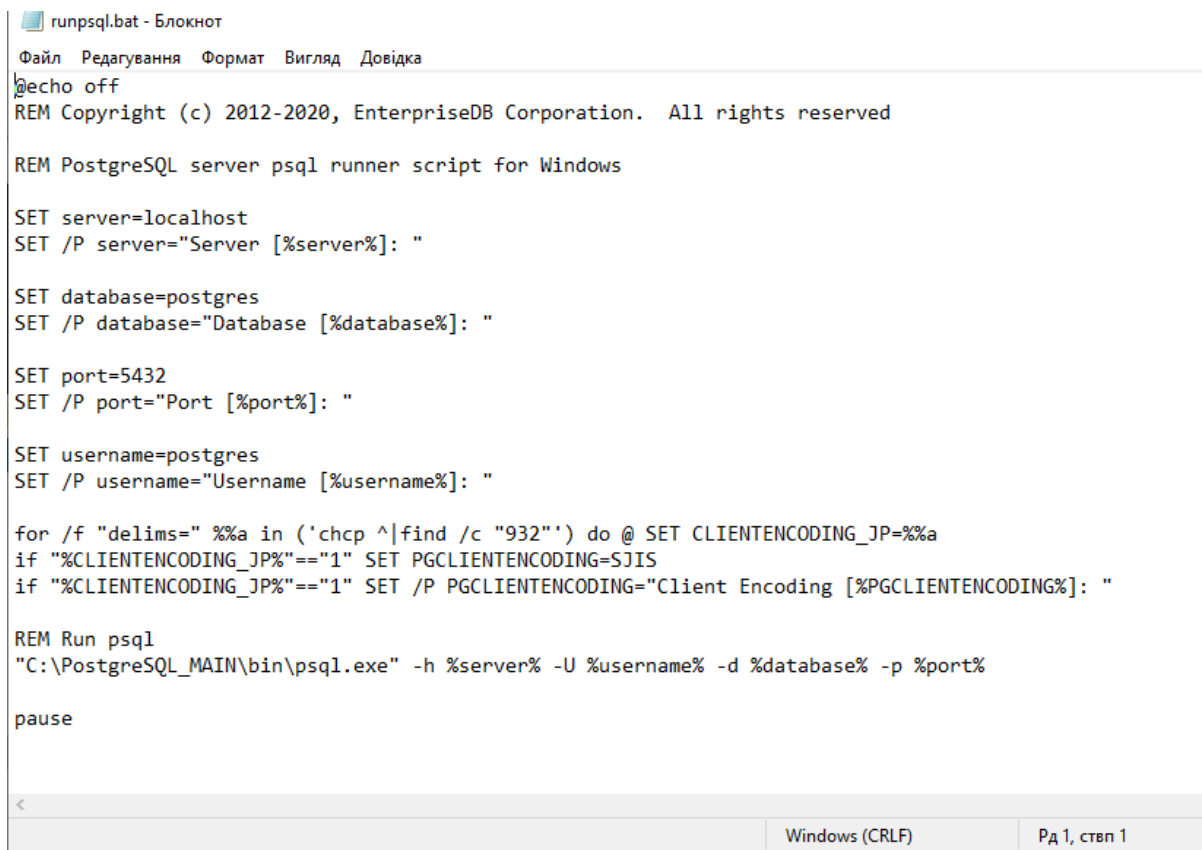


Рисунок 3.7 – Властивості ярлика SQL Shell

На рисунку 3.8 зображено вміст файлу runpsql.bat до внесення змін. У цьому файлі визначено параметри запуску SQL Shell, включно з налаштуваннями сервера, бази даних, порту та імені користувача.



```

runpsql.bat - Блокнот
Файл  Редагування  Формат  Вигляд  Довідка
@echo off
REM Copyright (c) 2012-2020, EnterpriseDB Corporation. All rights reserved

REM PostgreSQL server psql runner script for Windows

SET server=localhost
SET /P server="Server [%server%]: "

SET database=postgres
SET /P database="Database [%database%]: "

SET port=5432
SET /P port="Port [%port%]: "

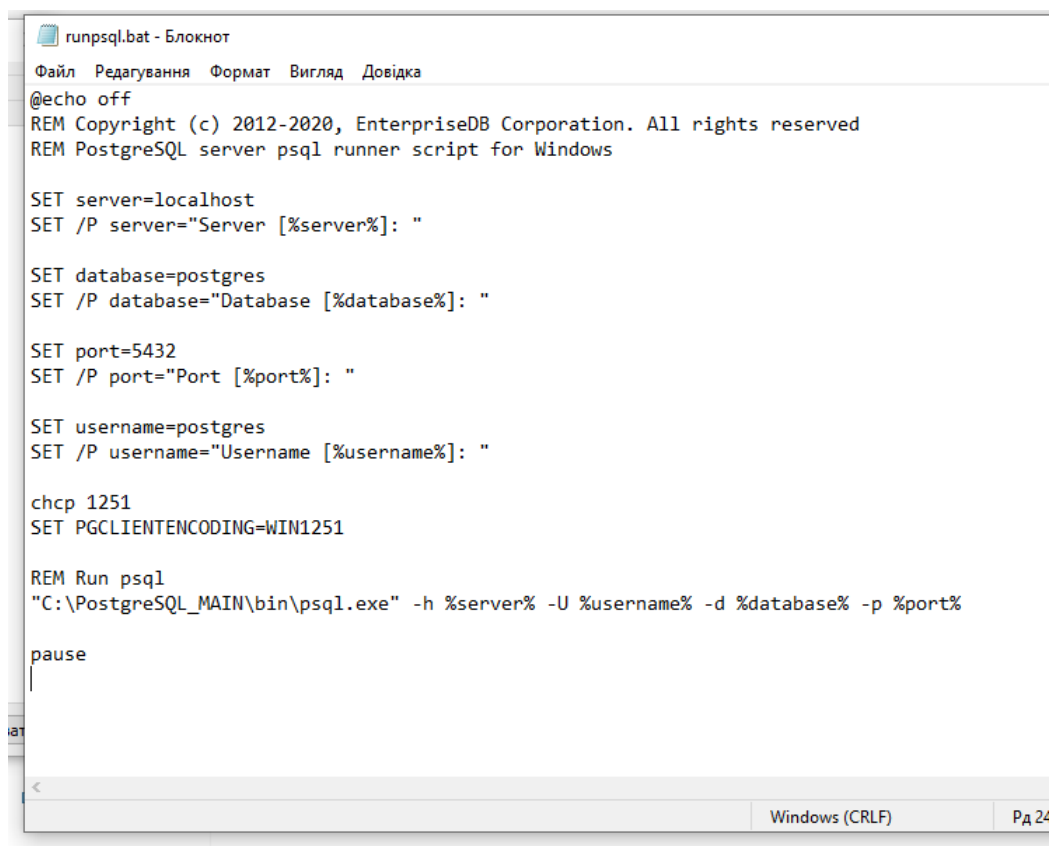
SET username=postgres
SET /P username="Username [%username%]: "

for /f "delims=" %%a in ('chcp ^|find /c "932"') do @ SET CLIENTENCODING_JP=%%a
if "%CLIENTENCODING_JP%"=="1" SET PGCLIENTENCODING=SJIS
if "%CLIENTENCODING_JP%"=="1" SET /P PGCLIENTENCODING="Client Encoding [%PGCLIENTENCODING%]: "

REM Run psql
"C:\PostgreSQL_MAIN\bin\psql.exe" -h %server% -U %username% -d %database% -p %port%

pause
  
```

Рисунок 3.8 – Вміст файлу runpsql.bat до внесення змін



```

runpsql.bat - Блокнот
Файл  Редагування  Формат  Вигляд  Довідка
@echo off
REM Copyright (c) 2012-2020, EnterpriseDB Corporation. All rights reserved
REM PostgreSQL server psql runner script for Windows

SET server=localhost
SET /P server="Server [%server%]: "

SET database=postgres
SET /P database="Database [%database%]: "

SET port=5432
SET /P port="Port [%port%]: "

SET username=postgres
SET /P username="Username [%username%]: "

chcp 1251
SET PGCLIENTENCODING=WIN1251

REM Run psql
"C:\PostgreSQL_MAIN\bin\psql.exe" -h %server% -U %username% -d %database% -p %port%

pause
  
```

Рисунок 3.9 – Модифікований вміст файлу runpsql.bat

На рисунку 3.9 показано модифікований вміст файлу runpsql.bat, у якому було додано налаштування для усунення проблеми з некоректним відображенням символів у SQL Shell згідно документації [21]. Основні зміни включають встановлення відповідного кодування як для консолі, так і для клієнта PostgreSQL. На початку файлу додано команду chcp 1251, яка змінює кодування консолі Windows на Windows-1251, для відображення українських символів.

Після внесення змін у файл запуску необхідно виконати перезапуск SQL Shell (psql). Під час перезапуску виконуються інструкції налаштування з потрібним режимом роботи консолі. Результат з оновленими параметрами запуску консолі можна побачити на рисунку 3.10.

```

SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Текущая кодовая страница: 1251
Пароль пользователя postgres:
psql (16.4)
Введите "help", щоб отримати допомогу.

postgres=# \l

```

Назва	Власник	Кодування	Постачальник локалі	Порядок сортування	Ctype	Локалізація ICU	Правила ICU	Права доступу
postgres	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251			=c/postgres +
template0	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251			postgres=CtC/postgres +
template1	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251			=c/postgres +
test	postgres	UTF8	libc	Ukrainian_Ukraine.1251	Ukrainian_Ukraine.1251			postgres=CtC/postgres

```

(4 рядки)

postgres=# \c test
Ви тепер під'єднані до бази даних "test" як користувач "postgres".
test=# \td
Невірна команда \td
Спробуйте \? для отримання довідки.
test=# \dt

```

Схема	Назва	Тип	Власник
public	test_table	таблиця	postgres

```

(1 рядок)

test=#

```

Рисунок 3.10 – Результат виконання з оновленими параметрами запуску

3.2 Розробка програми-менеджера для обробки замовлень

Після встановлення та налаштування всіх необхідних компонентів для розроблення програмного забезпечення насамперед було створено макет програми-менеджера. Це дозволило абстрактно-логічно визначити, яким чином будуть представлені основні елементи інтерфейсу та функціональність програми. Макет

було створено для візуалізації логіки роботи системи, з акцентом на зручність використання і функціональність, необхідну для автоматизації обробки замовлень.

Програма-менеджер є центральним компонентом системи автоматизації й обробки замовлень, який забезпечує централізоване управління всіма процесами, як показано на рисунку 3.11.

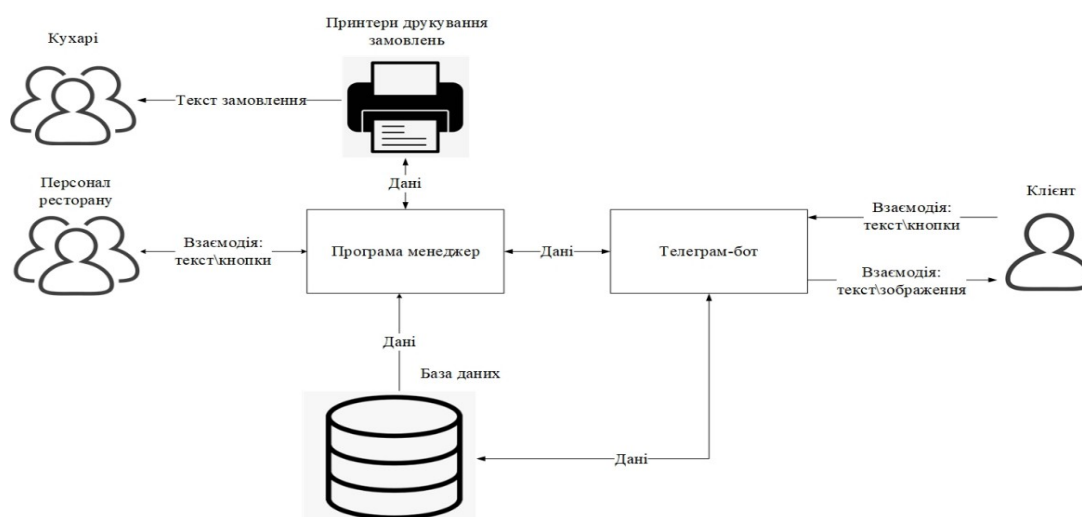


Рисунок 3.11 – Загальна схема системи

Вона обробляє замовлення, керує меню, відстежує поточний статус замовлень і забезпечує взаємодію між клієнтами та персоналом ресторану. Програма отримує дані від Telegram-бота, який служить точкою взаємодії з клієнтами. Клієнт надсилає замовлення через Telegram-бота у вигляді тексту або кнопок. Telegram-бот передає ці дані до програми-менеджера, яка обробляє їх та за необхідністю може звертатися до БД.

БД зберігає всі замовлення та іншу пов'язану інформацію, забезпечуючи її доступність для подальшої обробки. Персонал ресторану має можливість взаємодіяти з програмою-менеджером через графічний інтерфейс, у разі прийняття замовлення безпосередньо біля каси. У графічному інтерфейсі відображаються всі поточні замовлення, а також доступні функції для їх редагування, видалення або оновлення статусу. Інформація про замовлення автоматично передається до кухарів у вигляді друкованих замовлень за допомогою спеціально відведеного принтеру.

Таким чином на етапі розробки макета програми-менеджера (рисунок 3.12)

були передбачені ключові функціональні елементи, такі як картки замовлень, блок управління меню, а також набір інструментів для взаємодії з клієнтами. Картки замовлень розташовані в лівій частині інтерфейсу і дають змогу швидко переглядати і редагувати інформацію про замовлення, включно з ім'ям клієнта, часом замовлення і складом страв.

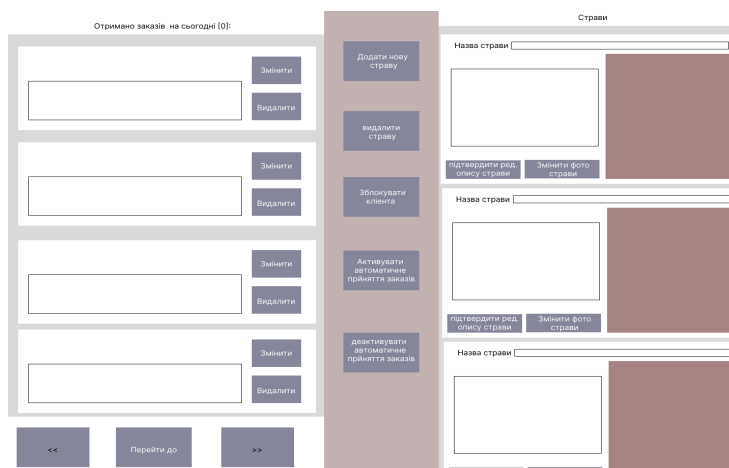


Рисунок 3.12 – Макет інтерфейсу програми-менеджера

У центральній частині інтерфейсу розміщені кнопки для додавання, зміни та видалення замовлень, забезпечуючи гнучкість у роботі персоналу. Права частина інтерфейсу призначена для відображення меню ресторану, включно з назвами страв, описами та зображеннями, які можна редагувати через відповідні функції. Для забезпечення зручності роботи було додано елементи навігації, так як кнопки для гортання карток замовлень та інструменти для фільтрації даних. Також передбачено функцію активації або деактивації автоматичного приймання замовлень, що робить програму адаптивною до різних сценаріїв використання. Такий підхід дозволив закласти основу для побудови програми, яка враховує базові потреби персоналу ресторану. Після формування макета програми-менеджера і створення ключових елементів інтерфейсу було розроблено загальний алгоритм роботи системи, який деталізує взаємодію всіх компонентів. На рисунку 3.13 представлено загальний алгоритм функціонування системи, що демонструє послідовність дій і розподіл завдань між модулями. Алгоритм охоплює основні етапи функціонування роботи системи.

Першим кроком роботи системи є ініціалізація системи управління базами даних (СУБД). База даних виступає центральною ланкою, в якій зберігається інформація про замовлення, клієнтів, доступні столики та меню ресторану. На

цьому етапі база даних готується до виконання запитів на запис, читання та оновлення даних. Це дає змогу забезпечити надійну взаємодію між усіма компонентами системи.

Наступним етапом є запуск програми-менеджера, яка виконує функцію центрального керуючого елемента. Програма-менеджер приймає дані від Telegram-бота, виконує запити до бази даних і обробляє інформацію для подальшого виведення на графічний інтерфейс або передачі на принтер для кухарів. Вона також відповідає за обробку запитів, що надходять від касової системи.

Після запуску програми-менеджера активується Telegram-бот, який надає користувачам зручний інтерфейс взаємодії. Бот приймає команди та текстові повідомлення, обробляє їх і передає інформацію програмі-менеджеру для подальшого опрацювання.

Telegram-бот також відповідає за надсилання відповідей клієнтам, включно з підтвердженням замовлень, повідомленнями про доступність столиків і їх бронювання. Ця частина системи забезпечує інтерактивність і простоту взаємодії для клієнтів, які використовують дистанційний формат оформлення замовлень.

Коли всі основні компоненти системи запуснено, вона переходить у режим очікування запитів від користувачів. У цьому стані система готова до обробки вхідних даних, що надходять через Telegram-бот або безпосередньо через графічний інтерфейс на касі. Такий підхід забезпечує гнучкість системи, дозволяючи клієнтам використовувати як онлайн, так і офлайн методи взаємодії.

Обробка запиту починається з перевірки його джерела. Якщо запит надходить від Telegram-бота, система активує процес взаємодії з клієнтом, пропонуючи різні команди та варіанти дій. У разі, якщо запит надходить через касову систему, його обробка виконується безпосередньо через програму-менеджер, що спрощує управління замовленнями в ресторані.

Коли користувач взаємодіє з Telegram-ботом, той обробляє запит, аналізує його зміст і звертається до програми-менеджера для отримання актуальної інформації.

Наприклад, Telegram-бот може запросити дані про доступні столики, статус замовлення або склад меню. Після отримання даних бот формує відповідну відповідь і надсилає її клієнту. У разі підтвердження замовлення інформація

передається в програму-менеджер для виконання подальших дій.

Програма-менеджер, отримавши дані від Telegram-бота або касової системи, аналізує запит і оновлює базу даних. За необхідності формується текст замовлення, який відправляється на принтер для передачі кухарям. Одночасно програма-менеджер перевіряє коректність виконання операцій, записує звіти про можливі помилки та оновлює стан замовлення в базі даних.

Після обробки замовлення дані в базі даних актуалізуються. Це включає оновлення статусів замовлень, зміну інформації про бронювання столиків та інші дії, необхідні для підтримки актуального стану системи. Оновлення даних у реальному часі дає змогу уникнути неузгодженості інформації між компонентами системи. Цикл завершується поверненням системи в стан очікування нових запитів.

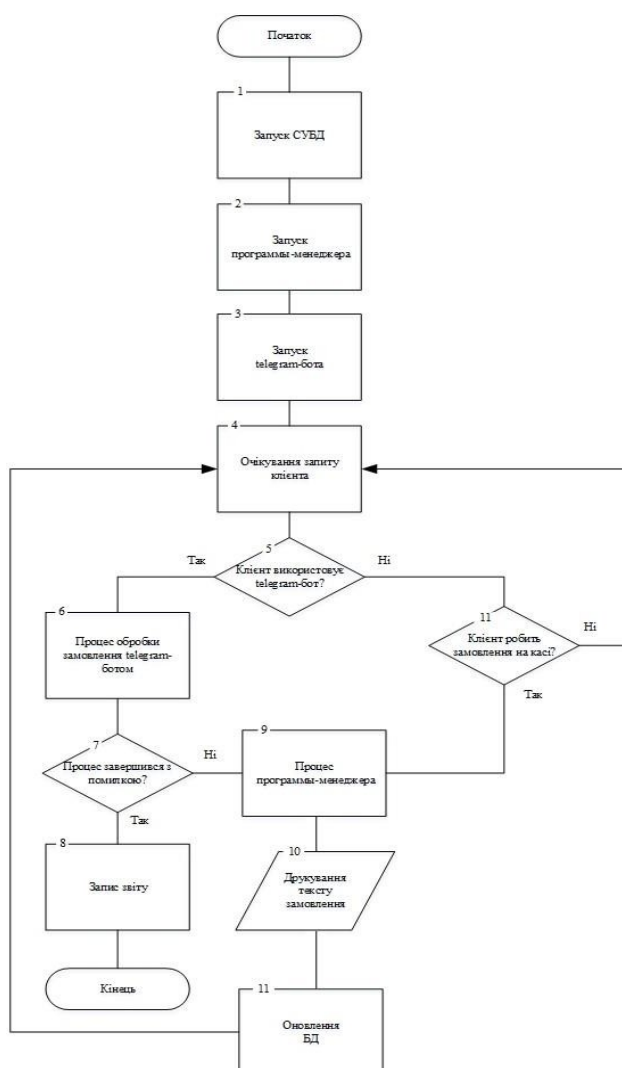


Рисунок 3.13 – Загальний алгоритм роботи системи

Представлений алгоритм дає змогу синхронізувати роботу всіх компонентів системи та ефективно управляти процесами в ресторані. Його реалізація спрямована

на автоматизацію обробки замовлень, зниження помилок під час взаємодії та підвищення загальної ефективності обслуговування клієнтів.

3.2.1 Апаратне забезпечення системи

Апаратна складова включає серверне обладнання (хмарний сервер), мережеву інфраструктуру (точки Wi-Fi), робочі станції персоналу (ПК менеджера, планшети офіціантів), кухонне обладнання (термопринтер чеків, планшет для кухаря).

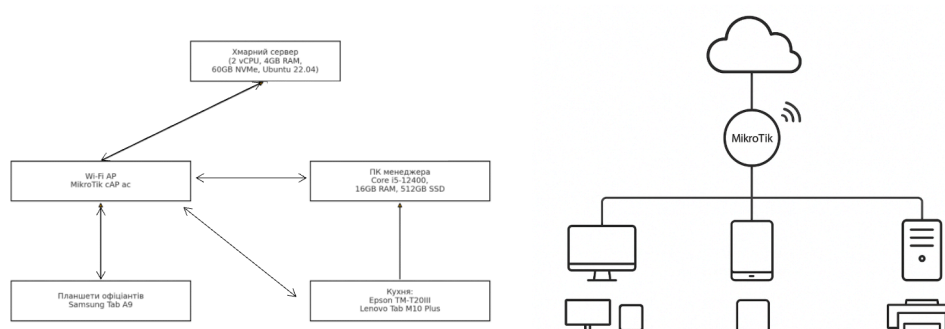


Рисунок 3.14 – Апаратне забезпечення системи

Для забезпечення роботи системи обрано таку апаратну конфігурацію:

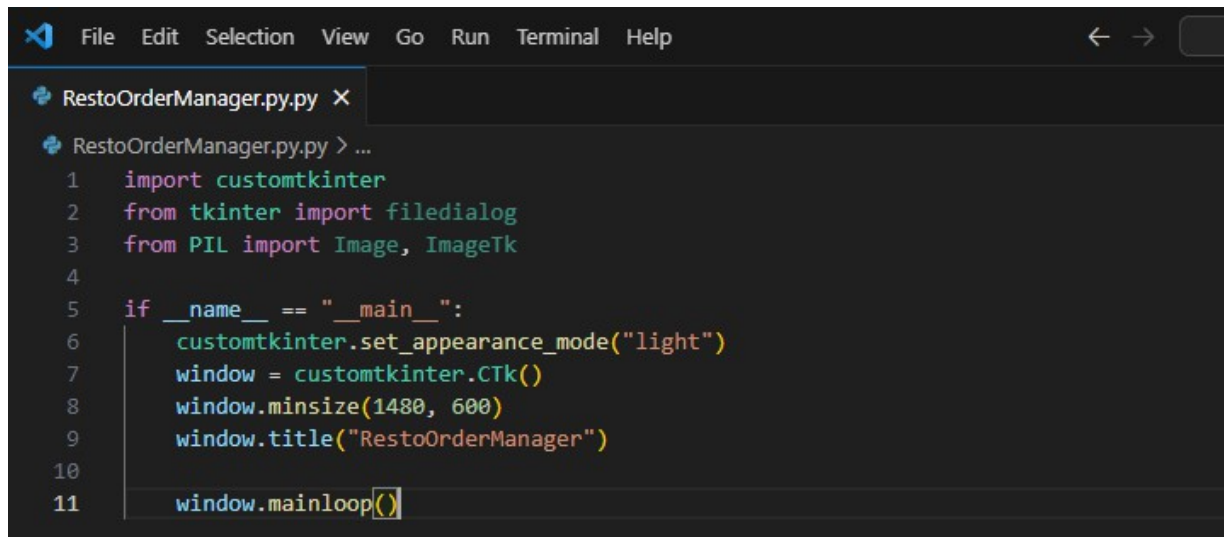
- хмарний сервер: 2 vCPU, 4 GB RAM, 60 GB NVMe SSD, Ubuntu Server 22.04.
- Wi-Fi інфраструктура: точки доступу MikroTik sAP ac (стандарти мереж - b (до 11 Мбіт/с), g (до 54 Мбіт/с), n (до 300 Мбіт/с), ac (до 867 Мбіт/с), робоча частота - 2.4 / 5 ГГц, тип антени - вбудована/незнімна, інтерфейси - LAN 10/100/1000 Мбіт/с);
- ПК менеджера: Intel Core i5-12400, 16 GB RAM, 512 GB SSD, монітор 24".
- планшети офіціантів: Samsung Galaxy Tab A9 (8.7", 4/64 GB).
- кухонне обладнання: термопринтер Epson TM-T20III та планшет Lenovo Tab M10 Plus для відображення замовлень.

3.2.1 Програмування інтерфейсу з за допомогою CustomTkinter

На етапі розроблення користувацького інтерфейсу було використано бібліотеку CustomTkinter. Повна реалізація програми представлена в лістингу А.1. На початку коду виконано імпорт необхідних модулів: customTkinter – для створення елементів користувацького інтерфейсу, filedialog з модуля tkinter – для роботи з файловою системою, а також Image і ImageTk з бібліотеки PIL (Pillow) – для обробки графічних об'єктів.

Наведений набір модулів забезпечує функціональність, необхідну для побудови інтерфейсу з використанням графічних елементів та інтеграції зображень.

На Рисунку 3.14 представлено мінімальну реалізацію класу RestoOrderManager, що демонструє базовий підхід до створення інтерфейсу. Цей варіант заклав основу для подальшого розвитку функціональних можливостей програми.



```

File Edit Selection View Go Run Terminal Help
RestoOrderManager.py.py X
RestoOrderManager.py.py > ...
1  import customtkinter
2  from tkinter import filedialog
3  from PIL import Image, ImageTk
4
5  if __name__ == "__main__":
6      customtkinter.set_appearance_mode("light")
7      window = customtkinter.CTk()
8      window.minsize(1480, 600)
9      window.title("RestoOrderManager")
10
11     window.mainloop()

```

Рисунок 3.14 – Ініціалізація Основного Інтерфейсу RestoOrderManager

В коді задається режим зовнішнього вигляду інтерфейсу за допомогою функції `set_appearance_mode("light")`, яка активує світлу тему інтерфейсу.

Далі виконання коду переходить до блоку коду під умовою `if __name__ == "__main__":`. Такий підхід надає змогу розмежити виконавчого коду програми від

її модульних компонентів, таке рішення було прийняте для забезпечення гнучкості та модульності в проєкті. У контексті розробки `RestoOrderManager.py`, використання цієї конструкції дає змогу визначити блок коду, який виконуватиметься тільки під час прямого запуску файлу, запобігаючи небажаним побічним ефектам під час імпортування модуля в інші частини проєкту та спрощуючи його інтеграцію в більшу систему.

В умові створення основного вікна програми відбувається через виклик `CTk()`, який є частиною бібліотеки `CustomTkinter` і являє собою покращений варіант стандартного вікна `Tkinter`. Встановлення мінімального розміру вікна за допомогою `minsize(1480, 600)` було визначено експериментально, при цьому така роздільна здатність є мінімальною для представлення інформації в картках замовлення та в картках меню страв. Виклик цього методу гарантує мінімальний розмір вікна для розміщення всіх елементів управління та інформації, необхідних для ефективного подання інформації. Це рішення ґрунтується на експериментальній взаємодії з інтерфейсом за допомогою макета програмного інтерфейсу в інструменті для розробки дизайну `Figma`, використовуючи прив'язки.

Наприкінці умови виклик методу `mainloop()` запускає основний цикл опрацювання подій інтерфейсу, що є стандартною практикою для додатків, розроблених на основі `Tkinter` і його похідних[22]. Цей метод підтримує активність інтерфейсу, обробляючи взаємодії користувача і системні події.

Далі, в основному умовному блоці головного виконання програми, було додано всі необхідні об'єкти, реалізовані в макеті для створення інтерфейсу. Це включало спеціальні об'єкти для реалізації карток, такі як фрейми, поміщені в `ScrollableFrame`, які забезпечують динамічне та гнучке відображення інформації. Кожна картка є окремим елементом інтерфейсу, що містить ключову інформацію та функціональні кнопки для взаємодії користувача з програмою.

Кнопки і фрейми являють собою картки, були спроектовані як варіант програмної реалізації макета користувацького інтерфейсу. `Scroll-бари`, вбудовані в кожен `ScrollableFrame`, дають змогу зручно переглядати велику кількість карток під

час роботи з великим обсягом замовлень. У кодї також було визначено місця розташування і розміри всіх елементів інтерфейсу за допомогою методів `grid` і `pack`.

Дотримуючись задуму макета інтерфейсу, було розміщено кнопки управління, такі як кнопки для додавання, зміни та видалення страв, а також кнопки для блокування клієнтів і управління автоматичним прийняттям замовлень. Ці елементи були розміщені в окремому фреймі, що програмно забезпечує помістити його в сітку і прив'язати його місце до динамічного розширення дисплея в червоному дисплеї [23].

Функція `loadFood ()`, що викликається наприкінці блоку, відповідає за завантаження даних про страви з бази даних, потім інформація інтерпретується на запрограмоване відображення в інтерфейсі. запуск цієї функції на початку забезпечує довантаження інформації з бази даних під час увімкнення програми менеджера під час запуску програми менеджера. Цей аспект програми забезпечує актуальність і повноту інформації, доступної користувачеві. Цей метод дає змогу легко оновлювати і розширювати список страв без необхідності вносити зміни в основний код програми або запускати цю програму.

Вікно програми запрограмованою таким чином, що забезпечує автоматичне масштабування його елементів, з розрахунком на адаптацію інтерфейсу під різні розміри екранів та умови використання.

Використання методів `grid_rowconfigure` і `grid_columnconfigure` в Tkinter дозволяє елементам інтерфейсу гнучко адаптуватися до змін розміру вікна[24]. Зокрема, виклики `window.grid_rowconfigure (1, weight=1)` та `window.grid_columnconfigure (5, weight=1)` задають, що певні рядки та стовпчики в сітці розташування елементів масштабуватимуться пропорційно зміні розміру вікна, результат роботи представлено на рисунках 3.15 (мінімум) та 3.16 (ростягнуте вікно).

Це забезпечує, що ключові елементи інтерфейсу, як-от списки замовлень та інформація про страви, залишаються видимими та доступними, незалежно від розміру вікна програми. Функція завантаження зображень у програмі створена для завантаження контексту відображення візуальної інформації про страву на картці.

Метод `getImage`, що викликається при натисканні на кнопку «Змінити зображення», дає змогу користувачеві вибрати зображення з файлової системи з використанням стандартного діалогового вікна.

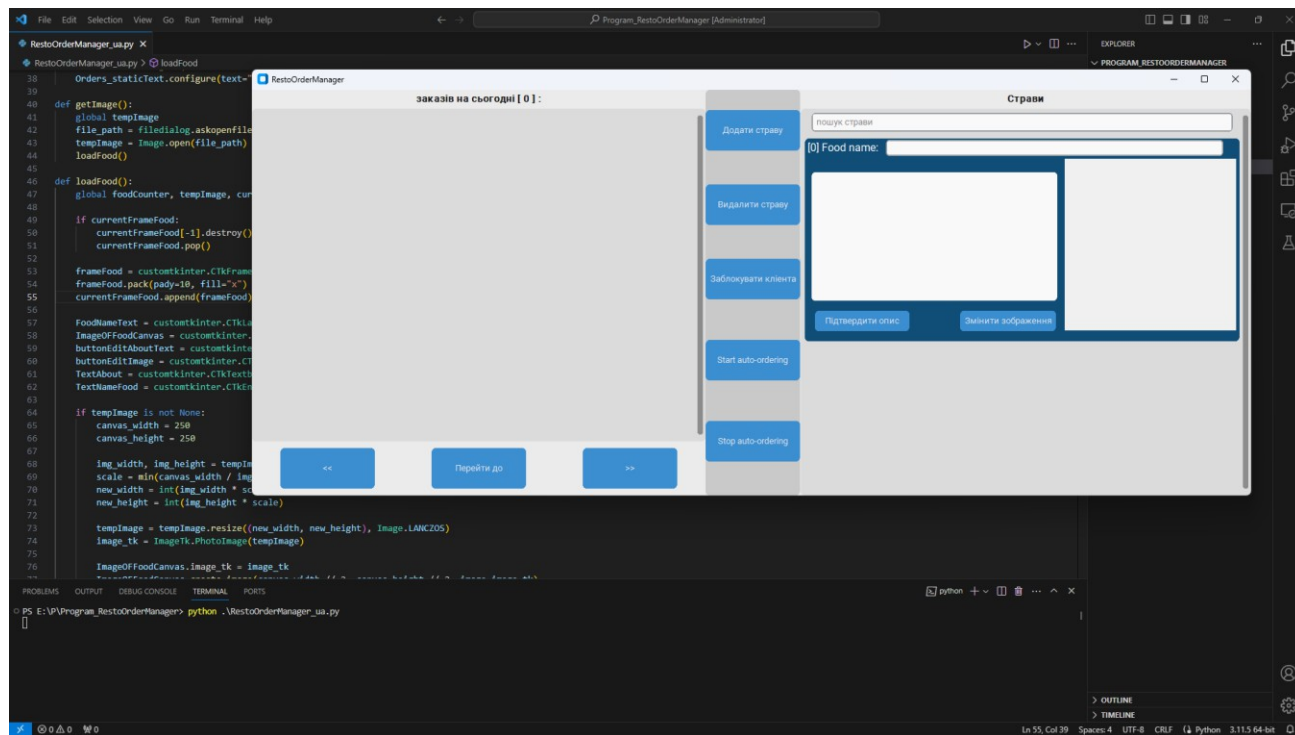


Рисунок 3.15 – Мінімальний розмір інтерфейсу RestoOrderManager

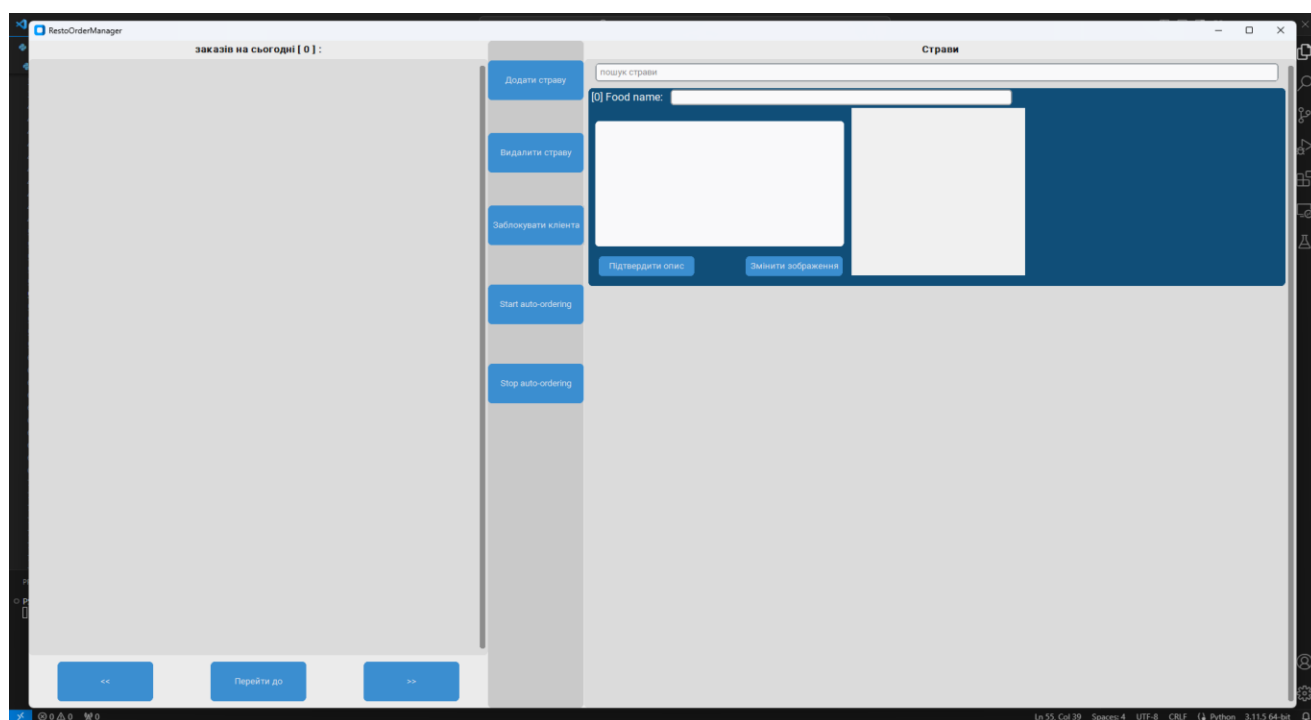


Рисунок 3.16 – Розтягнуте вікно інтерфейсу RestoOrderManager

Після вибору файлу зображення завантажується і обробляється за допомогою бібліотеки PIL. Обробка зображення включає в себе його масштабування для відповідності заданим розмірам елемента інтерфейсу, що забезпечує його коректне відображення без спотворень. На рисунку 3.17 показано початкове розширення зображення та на рисунку 3.18 показана результат відпрацювання розробленої функції loadFood() та getImage().

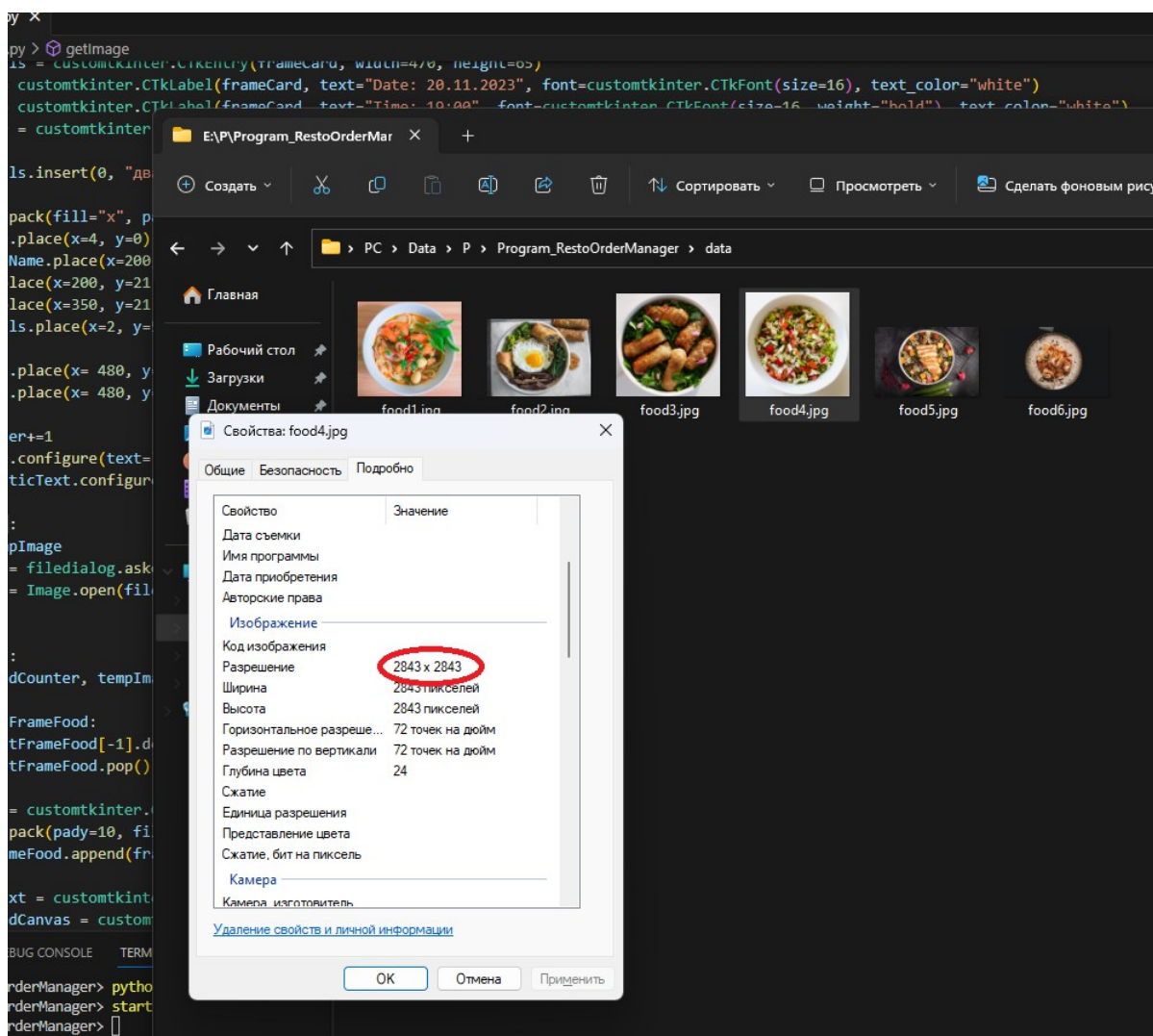


Рисунок 3.17 – Розмір вхідного зображення фотографії страви

Функція loadFood, що викликається під час додавання нової страви, відповідає за динамічне створення і відображення інформації про страви в інтерфейсі. Як уже зазначалося, кожну страву представляють як окремий фрейм

усередині ScrollableFrame, що дає змогу переглядати великий список страв прокручуванням донизу та догори.

У кожному такому фреймі відображається назва страви, її зображення, а також надаються елементи керування для редагування інформації про страву. Це включає в себе текстові поля для введення назви та опису страви, а також кнопки для редагування зображення і підтвердження внесених змін.

Такий підхід забезпечує непоганий ступінь інтерактивності та зручності використання програмного забезпечення, призначеного для використання в динамічному середовищі ресторанного бізнесу.

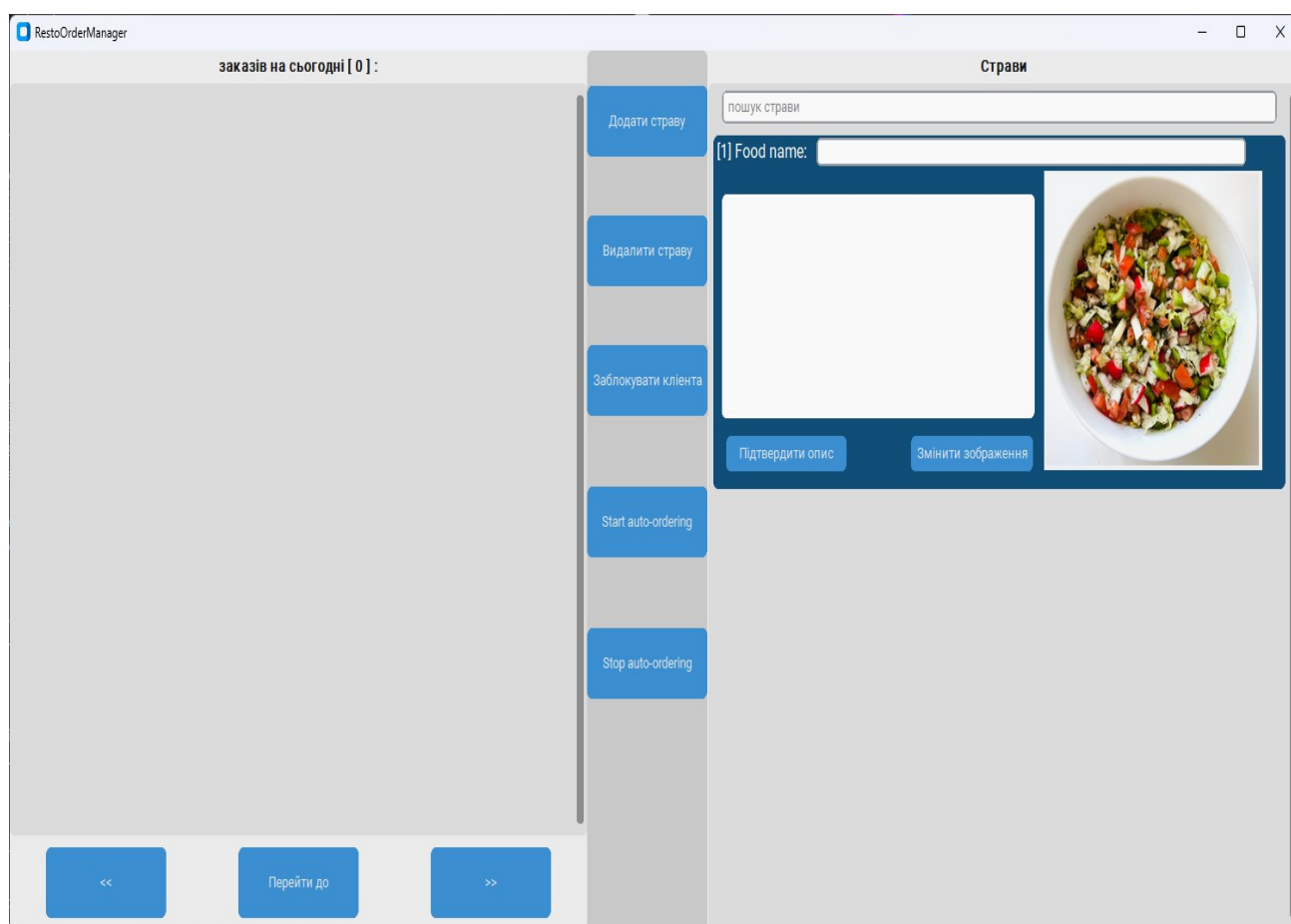


Рисунок 3.18 – Результат відпрацювання розроблених функції loadFood() та getImage()

Також, варто зазначити, що розроблений інтерфейс поєднує в собі простоту і функціональну гнучкість. Використання бібліотеки CustomTkinter надало змогу

створити сучасний і візуально привабливий інтерфейс, який водночас забезпечує ефективну та інтуїтивно зрозумілу взаємодію з користувачем.

3.3 Розробка бази даних

Розроблення БД бере свій початок з аналізу загальної схеми системи, представленої раніше на рисунку 3.11. Основною метою аналізу було виділення ключових сутностей, їхніх атрибутів і взаємозв'язків, необхідних для реалізації логіки взаємодії компонентів системи. На підставі цієї схеми було сформульовано концептуальну схему бази даних, яка відображає структуру даних і зв'язки між елементами системи. Ця концепція стала основою для створення бази даних, яка забезпечуватиме надійне зберігання інформації, зручний доступ до даних та їх оброблення.

Схема бази даних, представлена на рисунку 3.19, включає основні сутності, які забезпечують функціонування системи. Однією з ключових сутностей є таблиця «Клієнти» (Customers), яка зберігає дані про користувачів системи. Ця таблиця включає такі атрибути, як унікальний ідентифікатор клієнта (CustomerID), ім'я клієнта, номер телефону, TelegramID для ідентифікації клієнта в месенджері, дату реєстрації клієнта в системі та поточний баланс бонусних балів. Ці дані необхідні для ідентифікації клієнта, відстеження його активності та роботи програми лояльності.

Таблиця «Меню» (Menu) містить інформацію про доступні страви та їхні характеристики. Серед атрибутів цієї таблиці можна виділити унікальний ідентифікатор страви (MenuItemID), назву страви, її опис, вартість і посилання на зображення. Ця інформація дає змогу системі надавати клієнтам дані про доступні страви та формувати замовлення.

Таблиця «Замовлення» (Orders) відіграє центральну роль у системі та зберігає інформацію про замовлення клієнтів. У ній представлені такі атрибути, як

унікальний ідентифікатор замовлення (OrderID), ідентифікатор клієнта (CustomerID), дата і час оформлення замовлення, загальна сума замовлення (TotalAmount) та ідентифікатор статусу замовлення (StatusID). Ці дані використовуються для управління замовленнями та їх обробки.

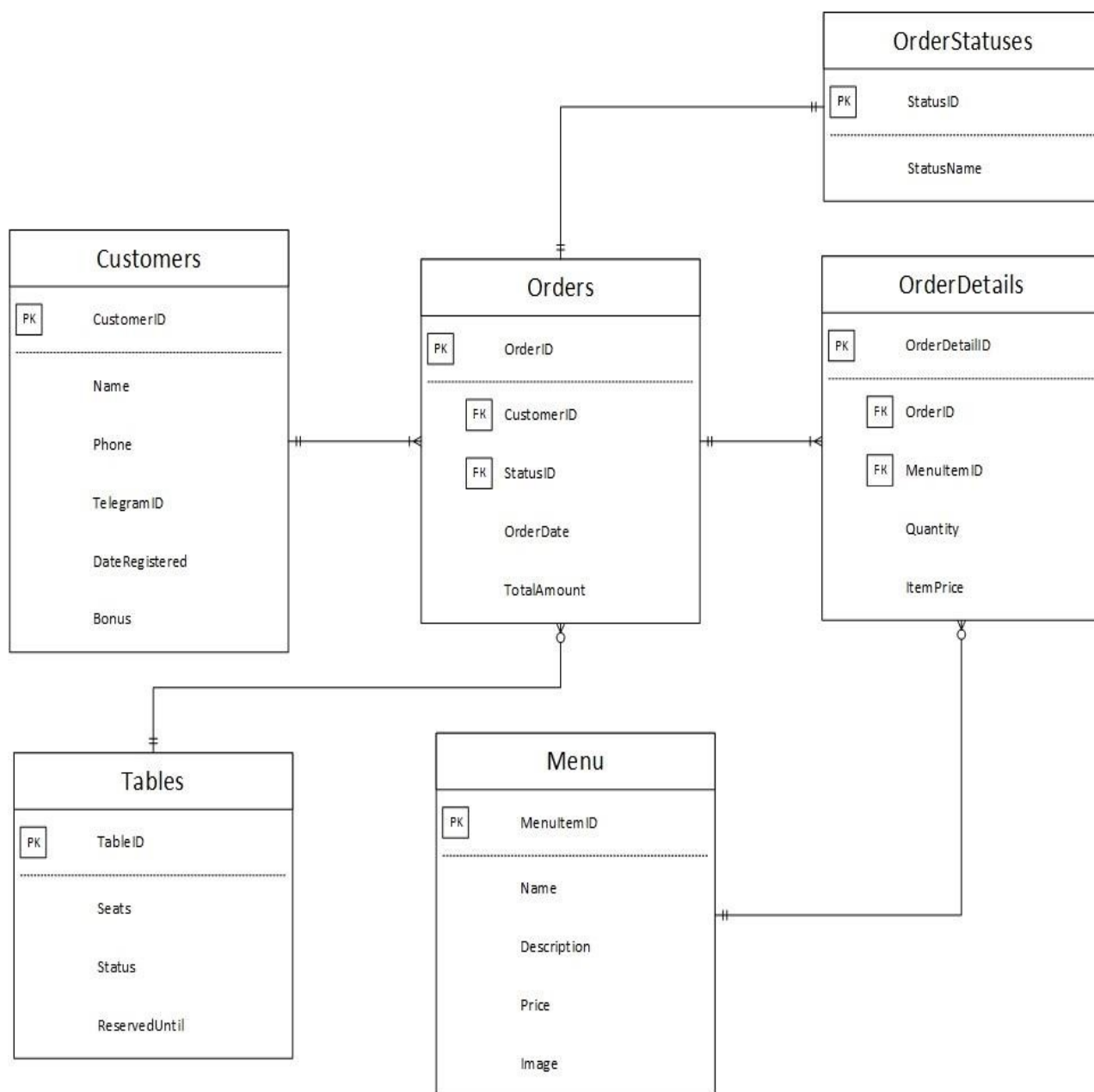


Рисунок 3.19 – Схема бази даних системи

Для деталізації складу замовлення використовується таблиця OrderDetails. Ця таблиця пов'язує замовлення і страви, вказуючи, які саме страви і в якій кількості були включені в конкретне замовлення. Основні атрибути таблиці

включають унікальний ідентифікатор запису (OrderDetailID), ідентифікатор замовлення (OrderID), ідентифікатор страви (MenuItemID), кількість страви (Quantity) та її ціну за одиницю (ItemPrice). Така структура дає змогу детально фіксувати склад кожного замовлення.

Довідник таблиця OrderStatuses містить інформацію про можливі статуси замовлення, такі як «Створено», «Готується», «Видано». Кожен статус представлений унікальним ідентифікатором (StatusID) і назвою статусу (StatusName). Ця таблиця використовується для відстеження поточного стану замовлень.

Таблиця столиків Tables призначена для обліку столиків у ресторані. Вона містить інформацію про кількість місць за столиком (Seats), поточний статус столика (наприклад, «Вільний», «Зайнятий», «Зарезервований») і час, до якого столик зарезервований (ReservedUntil). Така структура дає змогу системі керувати доступністю столиків і надавати інформацію клієнтам про можливості їхнього бронювання.

Для успішного створення бази даних важливо дотримуватися певного порядку, особливо у випадках, коли між таблицями встановлюються зовнішні зв'язки. У рамках розробленої бази даних процес почався зі створення таблиці «Статуси замовлень» (OrderStatuses). Ця таблиця є довідником, що містить можливі статуси замовлень, як-от «Створено», «Готується» і «Видано». Оскільки на неї посилаються інші таблиці, зокрема таблиця «Замовлення» (Orders), її створення було виконано насамперед.

Наступною було створено таблицю «Клієнти» (Customers), призначену для зберігання інформації про користувачів системи, включно з їхніми ідентифікаторами, іменами, контактними даними та балансом бонусних балів. Ця таблиця не залежить від інших таблиць, але використовується в таблиці «Замовлення» як зовнішній ключ. Тому вона також створювалася до залежних сутностей.

Після цього було створено таблицю «Меню» (Menu), у якій зберігаються дані про доступні страви. Таблиця «Меню» не пов'язана з іншими таблицями

безпосередньо, але використовується в таблиці «Склад замовлення» (OrderDetails), що вимагає її попереднього створення.

На наступному етапі було створено таблицю «Замовлення» (Orders), яка посиляється на таблиці «Клієнти» і «Статуси замовлень». Оскільки обидві ці таблиці були вже створені, створення таблиці «Замовлення» пройде без помилок. Таблиця містить такі атрибути, як унікальний ідентифікатор замовлення, посилання на клієнта, посилання на статус замовлення, дата оформлення і загальна сума.

Після таблиці «Замовлення» було створено таблицю «Склад замовлення» (OrderDetails). Ця таблиця відіграє важливу роль, пов'язуючи конкретне замовлення зі стравами з таблиці «Меню». Вона містить дані про кількість і вартість страв, що входять до замовлення. Останньою була створена таблиця «Столики» (Tables), яка призначена для обліку столиків у ресторані. Ця таблиця містить інформацію про кількість місць за кожним столиком, його поточний статус (наприклад, «Вільний» або «Зайнятий») і час, до якого він зарезервований. Оскільки ця таблиця не має зовнішніх зв'язків з іншими, вона могла бути створена на будь-якому етапі. Однак логічно виконувати її створення після всіх ключових таблиць, пов'язаних із замовленнями. Таким чином, дотримання зазначеного порядку створення таблиць дало змогу уникнути помилок, пов'язаних із відсутністю посилальних таблиць, і забезпечити цілісність структури бази даних.

Цей підхід забезпечує надійну основу для подальшого наповнення бази даних і тестування її функціональності. Список успішно створених усіх таблиць представлений на рисунку 3.20.

Список відношень							
Схема	Назва	Тип	Власник	Стійкість	Метод доступу	Розмір	Опис
public	customers	таблиця	postgres	постійна	hear	0 bytes	
public	menu	таблиця	postgres	постійна	hear	8192 bytes	
public	orderdetails	таблиця	postgres	постійна	hear	0 bytes	
public	orders	таблиця	postgres	постійна	hear	0 bytes	
public	orderstatuses	таблиця	postgres	постійна	hear	0 bytes	
public	tables	таблиця	postgres	постійна	hear	0 bytes	

Рисунок 3.20 – Список таблиць

3.4 Розробка Telegram -бота

3.4.1 Підключення мовної моделі та базова інтеграція

Початковим етапом розробки системи було підключення мовної моделі Llama 3.1 Instruct 8B, її запуск для тестування розуміння української мови, а також аналіз ліцензійної угоди, наданої компанією Meta. Ліцензійна угода дозволяє використовувати модель у дослідницьких і комерційних цілях, включно з інтеграцією в локальні програмні рішення, такі як Telegram-боти. Це стало основою для подальшого розроблення системи.

Умови угоди дають змогу обробляти текстові запити підтримуваними мовами, включно з українською, що відповідає завданням автоматизації приймання та обробки замовлень. Ліцензійні вимоги дозволяють інтеграцію моделі в програмні інтерфейси, такі як Telegram-боти, за умови дотримання політики допустимого використання. Це передбачає, що розробники, які використовують продукт, мають вказати атрибуцію «Built with Llama 3.1» у користувацькому інтерфейсі та супровідній документації.

Для запуску моделі було написано базовий код на Python, що забезпечує функціональність моделі. Основні компоненти системи включають мовну модель Llama 3.1 Instruct 8B, що відповідає за генерацію текстових відповідей, токенизатор, що використовується для обробки вхідних запитів користувачів, і логічний модуль, що керує взаємодією моделі з системою.

У процесі роботи використовувалися бібліотеки:

- transformers, яка надала доступ до моделі та токенизатора, а також забезпечила генерацію текстових відповідей;
- torch, яка використовувалася для управління обчисленнями і роботи з тензорами;
- safetensors, яка використовувалася для завантаження і зберігання моделі в безпечному форматі, що збільшило швидкість роботи системи.

Для запуску моделі було налаштовано роботу в режимі CPU та максимальну довжину генерованого тексту в 515 токенів, що забезпечило оптимальну продуктивність.

Особливу увагу було приділено модифікації системного запиту, щоб адаптувати модель під завдання Telegram-бота. Запит був оптимізований для чіткого формулювання запитів користувачів і генерації відповідей у чітко визначеному форматі. Це передбачало опрацювання неточних формулювань, помилок і універсалізацію підходу, що зробило модель надійнішою в роботі з різними сценаріями. Для перевірки працездатності системи було проведено тестовий запуск. На запит українською мовою «Привіт, розумієш українську?» модель згенерувала відповідь: «Так, я розумію українську мову. Привіт! Як можу допомогти сьогодні?». Цей результат підтверджує здатність моделі коректно розуміти запити та генерувати адекватні відповіді.

На рисунку 3.21 представлено процес тестування системи. На зображенні видно, як із використанням бібліотеки transformers було під'єднано токенизатор і модель, опрацьовано тестовий запит, і результат виведення в терміналі демонструє успішне виконання завдання. Код, представлений на рисунку, включає базову реалізацію виклику моделі та відображення результату користувачеві.

The screenshot shows a Visual Studio Code window titled 'LLM_GGUF_TEST [Administrator]'. The editor displays a file named 'main.py' with the following Python code:

```

1 import torch
2 from transformers import AutoModelForCausalLM, AutoTokenizer
3
4 local_model_path = "./llama3.1_8b"
5
6 tokenizer = AutoTokenizer.from_pretrained(local_model_path)
7
8 model_lm = AutoModelForCausalLM.from_pretrained(local_model_path, torch_dtype=torch.float32, device_map="cpu")
9
10 prompt = "Привіт розумієш українську"
11
12 inputs = tokenizer(prompt, return_tensors="pt").to("cpu")
13
14 output = model_lm.generate(inputs["input_ids"], max_length=515)
15
16 print("#####\n")
17
18 print(tokenizer.decode(output[0], skip_special_tokens=True))
19 input()

```

The terminal at the bottom shows the execution of the script:

```

PS F:\LLM_GGUF_TEST> python .\main.py
Loading checkpoint shards: 100%| 4/4 [00:14:00:00, 3.60s/it]
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's "attention_mask" to obtain reliable results.
Starting from v4.46, the "logits" model output will have the same type as the model (except at train time, where it will always be FP32)
#####
Привіт розумієш українську мову?
Так, я розумію українську мову. Привіт! Як можу допомогти сьогодні?
PS F:\LLM_GGUF_TEST>

```

The Explorer panel on the right shows the project structure for 'LLM_GGUF_TEST', including a 'data' folder and a 'llama3.1_8b' folder containing various configuration and model files.

Рисунок 3.21 – Запуск та перевірка мовної моделі

Виходячи з результатів роботи системи, було ухвалено рішення перейти до наступного етапу розробки - створення Telegram-бота. На цьому етапі планується інтеграція мовної моделі з використанням спеціальних запитів для отримання необхідного результату.

3.4.1 Розробка Telegram-бота з використанням мовної моделі

Telegram-бот виступає одним з інтерфейсів взаємодії між клієнтами і системою (рисунок 3.11). Крім можливості оформлення замовлень через касу або особистого звернення до ресторану, бот являє собою зручний інструмент для дистанційної взаємодії з системою. Він спрощує процес оформлення замовлень, надає інформацію про доступність столиків, бронювання і дає змогу клієнтам взаємодіяти з меню ресторану в зручній текстовій формі, використовуючи команди і кнопки. Для проектування і тестування системи було використано концептуальний макет ресторану, створений на основі фотографій рендеру з 3D-моделі [25]. Двоповерховий заклад, представлений на рисунках 3.22 і 3.23, призначений для комфортного обслуговування відвідувачів. Цей підхід дав змогу змодельовати реалістичні сценарії використання системи, включно із взаємодією з меню та бронюванням столиків.

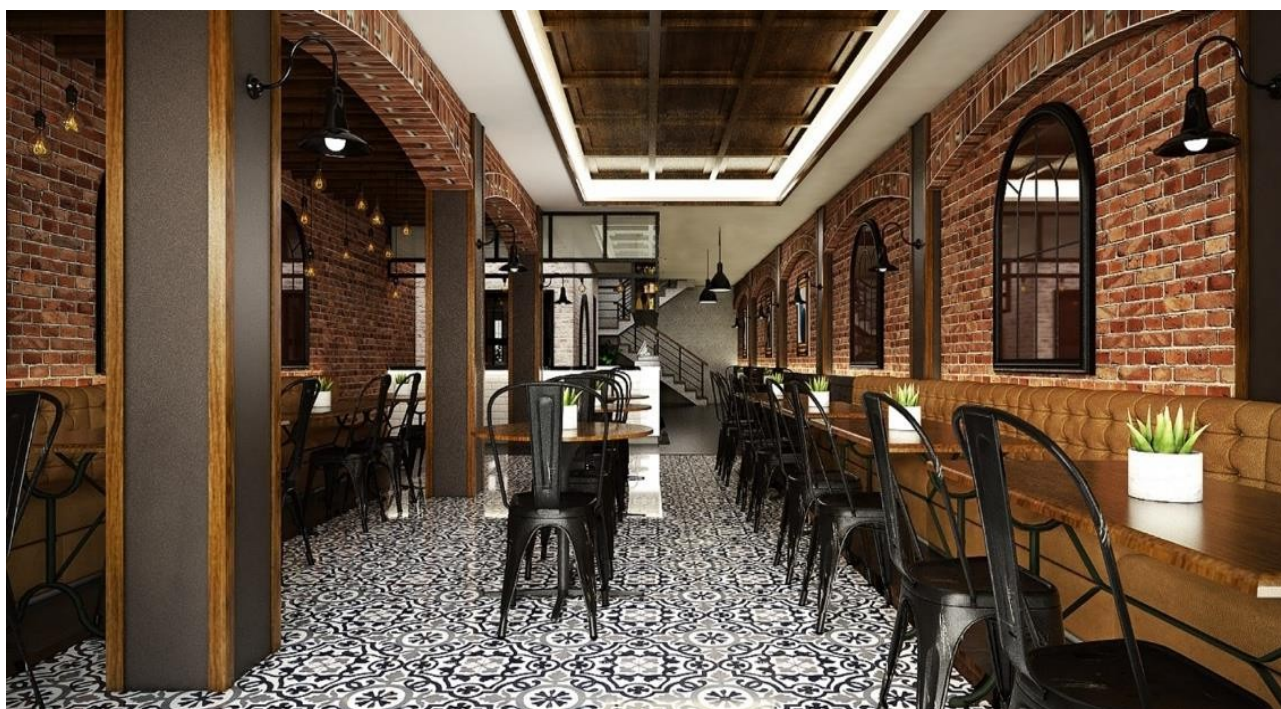


Рисунок 3.22 – Внутрішній інтер'єр ресторану (інший вид із першого поверху)



Рисунок 3.23– Внутрішній інтер'єр ресторану (вид із другого поверху)

Під час розробки Telegram-бота ця концепція використовували як основа для тестування і демонстрації його можливостей. Це дало змогу опрацювати сценарії взаємодії, такі як бронювання столиків, уточнення даних про доступні зони та вибір страв із меню.

Програмний код Telegram-бота реалізовано з використанням бібліотек `telebot`, `psycopg2`, `torch` і `PIL`.

Повний програмний код представлено в лістингу А.2. Ці бібліотеки забезпечують основні функції взаємодії з Telegram API, під'єднання до СУБД PostgreSQL, обробку зображень та інтеграцію з мовною моделлю для аналізу текстових запитів користувачів. Telegram-бот функціонує в автоматичному режимі, надаючи користувачам доступ до ключових сервісів ресторану.

Основний інтерфейс бота включає дві базові кнопки: «Показати меню» і «Замовити столик» (рисунок 3.24).

Ці кнопки формуються за допомогою методу `ReplyKeyboardMarkup` бібліотеки `telebot`, який створює інтерактивну клавіатуру. Кнопки додаються за допомогою об'єктів `KeyboardButton`, що забезпечує зручну навігацію для користувачів. Ці елементи залишаються доступними на всіх етапах взаємодії, що забезпечує консистентність інтерфейсу.



Рисунок 3.24 – Основний інтерфейс Telegram-бота

Під час натискання кнопки «Показати меню» бот активує відповідний обробник у коді, пов'язаний із текстом кнопки. Для обробки цієї дії використовується декоратор `@bot.message_handler`. Усередині обробника завантажується зображення меню (рисунок 3.25) за допомогою модуля `PIL`, а потім це зображення надсилається користувачеві у форматі файлу (рисунок 3.26) через метод `send_document`. Формат файлу було обрано, щоб уникнути компресії зображення, що дає змогу користувачеві переглядати меню з високою роздільною здатністю.

Після надсилання зображення меню інтерфейс автоматично розширюється, що продемонстровано на рисунку 3.26, за рахунок додавання трьох додаткових кнопок: «Доставити за адресою», «Планую їсти в ресторани» і «Заберу з собою».



Рисунок 3.25 – Зображення меню

Ці кнопки формуються за допомогою аналогічного методу ReplyKeyboardMarkup, який замінює поточну клавіатуру новими опціями. Додавання кнопок реалізовано таким чином, що кожна з нових кнопок прив'язана до власного обробника, який ініціюється при виборі відповідного варіанта [26]. Наприклад, при натисканні кнопки «Доставити за адресою» бот запитує в

користувача адресу та страву для доставки. Обробник опрацьовує текстові команди і знову перемикає клавіатуру на базові кнопки після завершення введення.

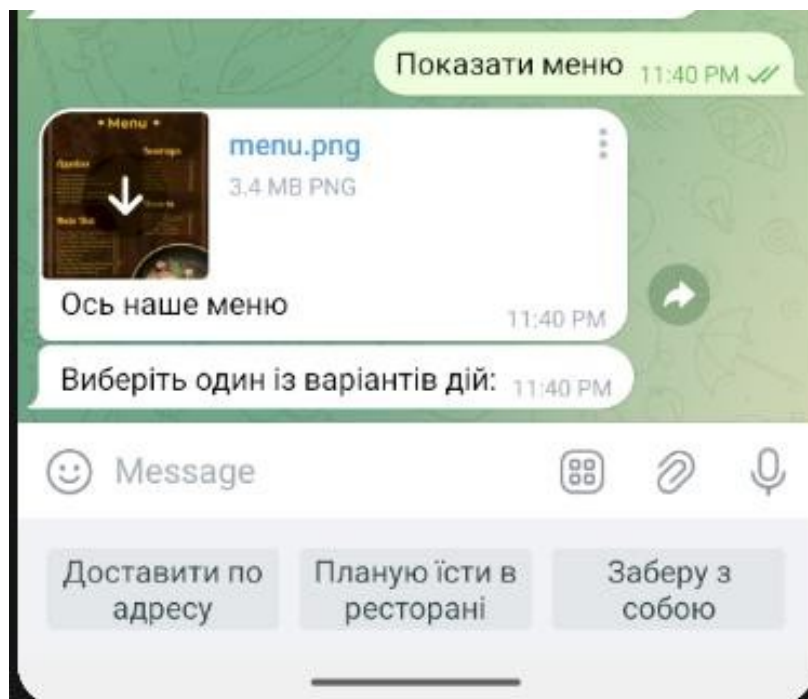


Рисунок 3.26 – Результат надсилання картинки у форматі файлу

Цей механізм динамічної зміни кнопок і обробників забезпечує гнучкість взаємодії, даючи змогу адаптувати інтерфейс бота під різні сценарії.

Для реалізації повернення до вихідних кнопок після завершення взаємодії використовується метод `reply_markup=main_menu()`. Це дає змогу скоротити кількість дій, необхідних користувачеві для подальшої взаємодії з ботом, і зберігає консистентність інтерфейсу.

При виборі «Замовити столик» бот відправляє користувачеві галерею зображень інтер'єру ресторану (рисунок 3.27). Для реалізації цього функціоналу використовується метод `send_photo`, який відправляє зображення зі списку `galleryPaths`. Цей список заздалегідь визначений у коді та містить шляхи до файлів зображень. Блок коду, що відповідає за надсилання галереї функціонує наступним чином: якщо користувач вже існує в базі даних (ідентифікується за `TelegramID`) у БД, галерея зображень не надсилається. Замість цього надсилається схема ресторану (рисунок 3.28), на якій столики динамічно відмічаються рамкою з

відповідним кольором в залежності від їхнього стану, отриманого з бази даних. Для цього використовується таблиця Tables у базі даних, де зберігаються дані про статус кожного столика.

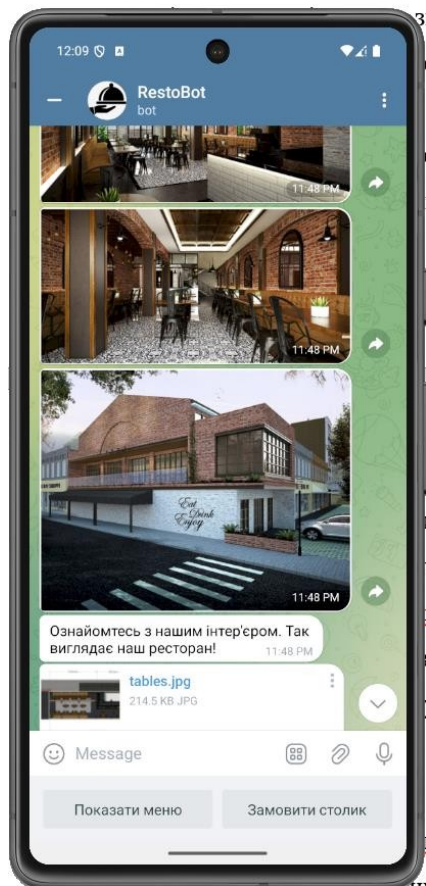


Рисунок 3.27 – Галерея зображень інтер'єру



Рисунок 3.28 – Схема вільних столиків

Систему візуалізації схеми ресторану реалізовано таким чином: спочатку Telegram-бот робить запит до бази даних за допомогою бібліотеки psycopg2. Для

отримання статусів столиків використовується SQL-запит, який повертає інформацію про ідентифікатори столиків та їхні статуси (вільний або зайнятий). Ці дані використовуються для побудови візуального представлення доступності столиків на схемі ресторану.

Після отримання інформації з бази даних дані обробляються для позначення координат столиків на схемі ресторану. Для візуалізації столиків використовуються зелені рамки (вільні столики) і червоні рамки (зайняті столики). Координати для відтворення рамок задаються в масиві `table_coordinates_first_floor`, який містить початкові та кінцеві координати для кожного столика на зображенні схеми.

Для динамічного малювання рамок поверх зображення використовується бібліотека PIL. У процесі візуалізації створюється новий шар поверх вихідного зображення, де зеленим або червоним кольором позначаються вільні та зайняті столики відповідно. Після опрацювання зображення схема ресторану з оновленою інформацією про статус столиків зберігається і надсилається користувачеві у форматі файлу.

Після надсилання схеми бот пропонує користувачеві вибрати столик, який його цікавить. Користувач може надіслати текстовий запит із номером столика. Бот обробляє цей запит, перевіряючи доступність зазначеного столика в базі даних. У разі, якщо столик вільний, бот запитує в користувача час бронювання й уточнює, чи потрібно заздалегідь приготувати страви.

Якщо користувач надсилає складний текстовий запит, який не відповідає заздалегідь заданим командам, бот передає повідомлення в мовну модель для аналізу.

Лінгвістична модель обробляє запит, повертає структурований результат, який спеціально написаним програмним обробником інтегрується в базу даних через SQL-запити. Для цього використовується бібліотека `transformers`, яка забезпечує роботу моделі. Вхідні дані передаються разом із контекстом меню ресторану, а вихідні дані містять строго структурований список обраних користувачем позицій.

Наприклад, запит «на десерт вафлі і сендвіч bbq й соду» буде опрацьовано мовною моделлю, яка поверне відповідь: «<Bacon Waffles><Smoky BBQ Sandwich><Lime Soda>». Бот перехоплює цю відповідь, формує замовлення і надає користувачеві інформацію про його зміст із пропозицією обрати спосіб оплати.

Таким чином, система забезпечує інтеграцію візуального та текстового інтерфейсів, синхронізацію з базою даних і динамічне оновлення інформації. Це робить процес бронювання столиків зручним, ефективним та інтуїтивно зрозумілим для користувачів.

4 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

Тестування системи є важливим етапом розробки, метою якого є перевірка коректності роботи програми Telegram-бота, інтегрованого з мовною моделлю Llama 3.1 Instruct 8B.

Telegram-бот використовується як інтерфейс для взаємодії з користувачем і перевірки здатності моделі обробляти різні запити. Основним завданням тестування є оцінювання точності роботи запитів, які забезпечують правильне формування контексту та генерацію відповідей, водночас виключаючи необхідність застосування файнтюнінгу.

Тестування спрямоване на визначення рівня якості відповідей, які відповідають очікуваним сценаріям взаємодії з користувачем. Додатково, воно включає перевірку функціональності інтерфейсу, пов'язаного з графічними елементами і обробкою файлів.

Метод тестування ґрунтується на послідовній взаємодії з Telegram-ботом. Процес тестування включає надсилання різних запитів і аналіз відповідей на основі заздалегідь підготовлених запитів.

Основна увага приділяється ключовим сценаріям, серед яких запити про меню ресторану, оформлення замовлення та обробка некоректних або неповних

запитів. Наприклад, у сценаріях оформлення замовлень перевіряється, наскільки бот підтримує послідовність дій - від вибору страви до підтвердження замовлення.

Особлива увага приділяється стійкості системи до невизначених або некоректних даних, таких як запити з друкарськими помилками, недостатньою інформацією або складними формулюваннями. Такий підхід дає змогу виявити здатність моделі коректно інтерпретувати контекст і видавати логічні відповіді в умовах невизначеності.

Тестування системи проводилося на основі різних сценаріїв, один з яких включає взаємодію користувача з функціональністю меню страв. В одному сценарії користувач одразу починає ставити запитання з неповними або неоднозначними формулюваннями, а в іншому – надсилає заздалегідь підготовлене повідомлення, скопійоване з листування в минулому.

Нижче описано процес тестування на основі сценарію, в якому користувач обирає пункт «Показати меню». Після цього бот надає список доступних страв.

На наступному етапі користувачеві пропонується зробити вибір: скористатися готовими кнопками для оформлення замовлення або скористатися можливістю введення інформації вручну.

Користувач вирішив не використовувати готові відповіді, представлені кнопками, і замість цього написати інформацію вручну, вказавши всі деталі після перегляду меню рисунок 4.1.

Це повідомлення не збігається а програмному коді Telegram-бота з жодними попередньо прописаними шаблонами і, в такому разі, передається на обробку мовною моделлю. Мовна модель, використовуючи спеціальний запит (лістинг А.3.), обробляє запит користувача і формулює вихідний запит. Вихідний запит структурується з використанням спеціальних тегів, укладених у кутові дужки, і даних, наданих користувачем, оформлених у квадратних дужках. Така структура дає змогу програмному забезпеченню легко аналізувати й обробляти інформацію. Дані про час, наміри або локацію, зазначені користувачем, передаються таким чином:

- timer: зворотний відлік часу до виконання дії, вказаний у хвилинах;

- h: час, у який користувач планує прибуття;
- location: текст адреси, яку надав користувач. Приклад: location: [проспект 117А, другий поверх, 76 кімната].

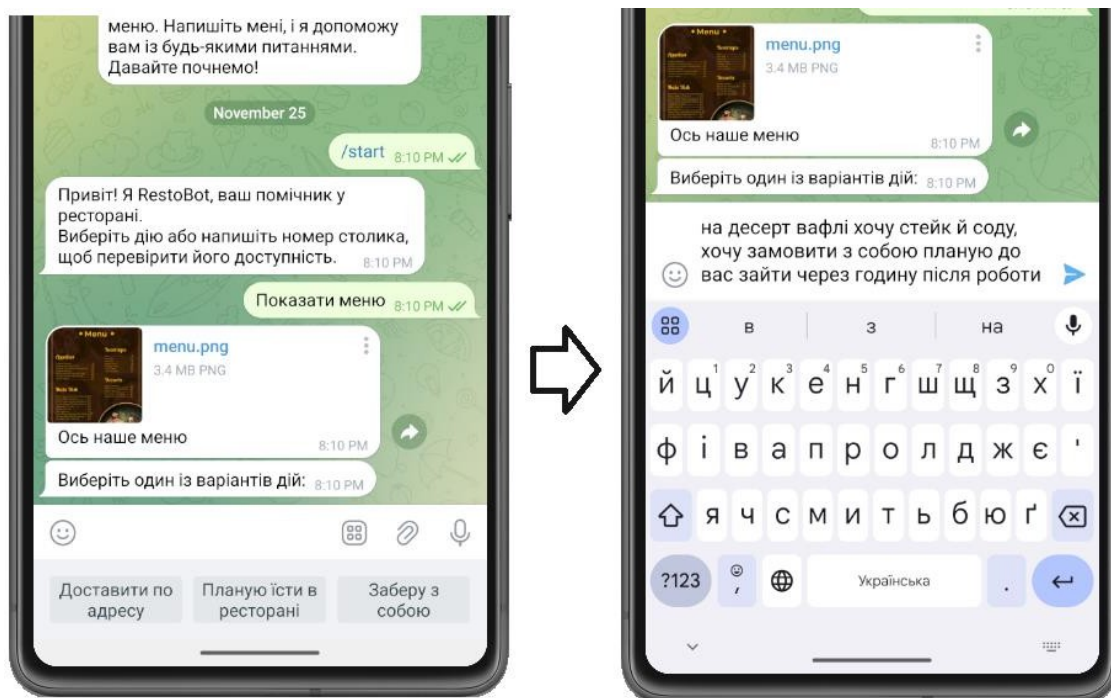


Рисунок 4.1 – Процес вибору оформлення замовлення вручну

Список варіантів намірів користувача відповідає кнопкам на рис. 4.1.

- 1 доставити за адресою.;
- 2 планує їсти в ресторані;
- 3 заберу із собою.

Модель успішно проаналізувала й інтерпретувала надані дані, перетворивши їх на потрібний формат для подальшого опрацювання. На основі повідомлення користувача: «на десерт вафлі хочу стейк й соду, хочу замовити з собою планую до вас зайти через годину після роботи»

Модель сформувала структурований вихідний запит:

<Bacon Waffles><Garlic Butter Steak Medallions><Lime Soda>[3][timer: 60].

Повідомлення, повернуте мовною моделлю, було оброблено алгоритмами Telegram-бота. Це дало змогу витягти ключові дані: обрані страви, намір користувача («заберу з собою») і додаткову інформацію, таку як таймер часу

прибуття («через 60 хвилин»). На основі отриманих даних Telegram-бот сформував SQL-запити, які були успішно виконані для оновлення відповідних таблиць бази даних, показані на рисунку 4.2.

```
postgres=# SELECT * FROM Customers;
customerid | name      | phone | telegramid | dateregistered | bonuspoints
-----+-----+-----+-----+-----+-----
1 | tg_bottest |      | 83475435345 | 2024-11-24 00:00:00 | 0
(1 рядок)
```

```
postgres=# SELECT * FROM Menu;
menuitemid | name | description | price | image
-----+-----+-----+-----+-----
1 | Bacon Waffles | Delicious bacon-infused waffles | 8.50 | /images/bacon_waffles.jpg
2 | Garlic Butter Steak Medallions | Juicy steak medallions with garlic butter sauce | 15.99 | /images/steak_medallions.jpg
3 | Lime Soda | Refreshing lime-flavored soda | 2.99 | /images/lime_soda.jpg
4 | Bacon Waffles | Delicious bacon-infused waffles | 8.50 | /images/bacon_waffles.jpg
5 | Garlic Butter Steak Medallions | Juicy steak medallions with garlic butter sauce | 15.99 | /images/steak_medallions.jpg
6 | Lime Soda | Refreshing lime-flavored soda | 2.99 | /images/lime_soda.jpg
(6 рядків)
```

```
postgres=# SELECT * FROM Orders;
orderid | customerid | statusid | orderdate | totalamount
-----+-----+-----+-----+-----
2 | 1 | 1 | 2024-11-24 00:00:00 | 27.48
(1 рядок)
```

```
postgres=# SELECT * FROM OrderDetails;
orderdetailid | orderid | menuitemid | quantity | itemprice
-----+-----+-----+-----+-----
(0 рядків)
```

```
postgres=# SELECT * FROM OrderStatuses;
statusid | statusname
-----+-----
1 | Нове замовлення
2 | В обробці
3 | Готове до видачі
4 | Завершено
(4 рядки)
```

```
postgres=# SELECT * FROM Tables;
tableid | seats | status | reserveduntil
-----+-----+-----+-----
1 | 4 | Зайнято | 2023-11-25 20:00:00
2 | 2 | Вільно |
(2 рядки)
```

```
postgres=#
```

Рисунок 4.2 – Результат виконання SQL-запитів

У таблицю Customers було занесено інформацію про користувача, включно з його унікальним ідентифікатором Telegram, ім'ям і датою реєстрації. Ці дані необхідні для ідентифікації клієнта і подальшої взаємодії з ним.

У таблицю Orders було додано запис про нове замовлення. Він містить відомості про клієнта, загальний статус замовлення і загальну суму, що дає змогу відстежувати етапи виконання замовлення і вести облік фінансових операцій.

У таблицю OrderDetails було внесено позиції меню, обрані користувачем. Ця інформація включає найменування страви, її кількість і ціну, що дає змогу точно сформувати замовлення для подальшої обробки.

У таблицю Tables було додано інформацію про статус столиків. Це дало змогу актуалізувати дані про резервування або звільнення столиків, що важливо для управління простором ресторану. Додатково, статус замовлення було оновлено в таблиці OrderStatuses. Це забезпечило синхронізацію поточного етапу виконання замовлення з системою, що дає змогу оперативно відстежувати його стан.

Такий процес автоматизації забезпечує високу точність і структурованість роботи системи. Використання SQL-запитів у зв'язці з Telegram-ботом спрощує управління даними, надаючи зручний та інтуїтивно зрозумілий інтерфейс як для користувачів, так і для адміністраторів ресторану.

Таким чином, модель завершила своє завдання, надавши дані в структурованому вигляді, готовому для подальшої автоматичної обробки.

ВИСНОВКИ

У магістерській роботі було розроблено ситему приймання та оброблення замовлень у ресторанах через Telegram-бот. У результаті проведених досліджень і виконаної роботи були досягнуті такі результати.

У процесі роботи було проведено аналіз предметної області, який дозволив визначити основні недоліки та проблеми існуючих підходів, а також знайти шляхи їх подолання. Це стало основою для формування вимог до створюваної системи.

Розроблено концепцію системи, яка базується на сучасних інформаційних технологіях та забезпечує ефективність і зручність у використанні. Проектування системи здійснювалося з урахуванням сучасних стандартів, що дозволило забезпечити її масштабованість та інтеграцію з іншими рішеннями.

На основі розробленої концепції створено програмне забезпечення, яке реалізує автоматизацію ключових процесів. Впровадження цієї системи дозволяє скоротити витрати часу на виконання завдань, підвищити точність даних та мінімізувати людський фактор.

Ефективність запропонованого рішення підтверджено в ході тестування, результати якого засвідчили відповідність системи поставленим вимогам. Система є стійкою, зручною у використанні та відповідає потребам користувачів.

Таким чином, результати магістерської роботи мають практичну цінність і можуть бути використані для подальшого вдосконалення досліджуваної системи та її адаптації до змінюваних умов експлуатації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Petruk M. Rule-Based vs AI chatbot: what is the difference?. WeSoftYou. URL: <https://wesoftyou.com/ai/rule-based-vs-ai-chatbot-what-is-the-difference/> (date of access: 10.07.2024).
2. Rajaraman V. From ELIZA to ChatGPT. Resonance. 2023. Vol. 28, no. 6. P. 889–905. URL: <https://doi.org/10.1007/s12045-023-1620-6> (date of access: 15.12.2024).
3. Khan R. Build better chatbots: a complete guide to getting started with chatbots : навчальний посібник. Apress, 2017. 169 p.
4. Maderis G. Top 22 benefits of chatbots for businesses and customers. Zendesk. URL: <https://www.zendesk.com/blog/5-benefits-using-ai-bots-customer-service/> (date of access: 15.12.2024).
5. Епоха чатів: месенджери для інтернет-торгівлі - KeyCRM Blog. KeyCRM Blog. URL: <https://blog.keycrm.app/uk/epoha-chativ-mesendzheri-dlya-internet-torgivli/> (дата звернення: 15.12.2024).
6. Similarweb digital intelligence: unlock your digital growth. Similarweb. URL: <https://www.similarweb.com/> (date of access: 15.12.2024).
7. Kozinskiy A. WhatsApp Business API: overview, limits and pricing [2024]. Elfsight. URL: <https://elfsight.com/blog/whatsapp-business-api-overview/> (date of access: 15.12.2024).
8. Bots FAQ. Telegram APIs. URL: <https://core.telegram.org/bots/faq#> (date of access: 15.12.2024).
9. Viber REST API. Viber Developers Hub. URL: <https://developers.viber.com/docs/api/rest-bot-api/> (date of access: 15.12.2024).
10. Welcome | nutgram. Welcome | Nutgram. URL: <https://nutgram.dev/> (date of access: 15.12.2024).

11. GitHub - telegrambots/telegram.bot: .NET client for telegram bot API. GitHub. URL: <https://github.com/TelegramBots/Telegram.Bot?tab=readme-ov-file> (date of access: 15.12.2024).
12. Overview | grammY. grammY. URL: <https://grammy.dev/guide/> (date of access: 15.12.2024).
13. PyTelegramBotAPI 4.25.0 documentation. pyTelegramBotAPI 4.25.0 documentation. URL: <https://pytba.readthedocs.io/en/latest/index.html> (date of access: 15.12.2024).
14. Best open source llms of 2024 – klu. Design, Deploy, and Optimize LLM Apps with Klu – Klu.ai. URL: <https://klu.ai/blog/open-source-llm-models> (date of access: 15.12.2024).
15. Sam K. Llama 3.1: an in-depth analysis of the next generation large language model. ResearchGate. 2024. URL: <https://doi.org/10.13140/RG.2.2.10628.74882> (date of access: 15.12.2024).
16. Vignesh yaadav. Exploring and building the llama 3 architecture : a deep dive into components, coding, and... Medium. URL: https://medium.com/@vi.ai_/exploring-and-building-the-llama-3-architecture-a-deep-dive-into-components-coding-and-43d4097cfbbb (date of access: 15.12.2024).
17. Tkinter Python interface to Tcl/Tk. Python documentation. URL: <https://docs.python.org/uk/3/library/tkinter.html> (date of access: 15.12.2024).
18. Official documentation and tutorial | customtkinter. Official Documentation And Tutorial | CustomTkinter. URL: <https://customtkinter.tomschimansky.com/> (date of access: 15.12.2024).
19. PyQt- introduction. URL: https://www.tutorialspoint.com/pyqt/pyqt_introduction.htm (date of access: 15.12.2024).
20. WxWidgets: Cross-Platform GUI Library. wxWidgets: Cross-Platform GUI Library. URL: <https://wxwidgets.org/> (date of access: 15.12.2024).
21. Set. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/current/sql-set.html> (date of access: 15.12.2024).

22. Amos D. Python GUI programming with tkinter – real python. Python Tutorials – Real Python. URL: <https://realpython.com/python-gui-tkinter/> (date of access: 15.12.2024).
23. Grid System | CustomTkinter. Official Documentation And Tutorial | CustomTkinter. URL: <https://customtkinter.tomschimansky.com/tutorial/grid-system/> (date of access: 15.12.2024).
24. Auto resize window · TomSchimansky/CustomTkinter · Discussion #435. GitHub. URL: <https://github.com/TomSchimansky/CustomTkinter/discussions/435> (date of access: 15.12.2024).
25. Vivame Design Design. Industrial coffee shop. Turbosquid. URL: <https://www.turbosquid.com/3d-models/industrial-coffee-shop-1364959> (date of access: 16.12.2024).
26. Борисов О., Тягунова М. Підхід до автоматизації замовлень у ресторані з використанням ІТ-технологій. Інформаційні технології та суспільство. – Київ. – 2024. – Вип. № 4 (15) - прийнято до друку.

ДОДАТОК А

ЛІСТИНГИ ПРОГРАМ

Лістинг А.1 – RestoOrderManager.py

```
import customtkinter
from tkinter import filedialog
from PIL import Image, ImageTk

BUTTONS_HEIGHT = 60
orderCounter = 0
foodCounter = 0
tempImage = None
modeImageChange = None
currentFrameFood = []

def addCardOrder():
    global orderCounter

    frameCard = customtkinter.CTkFrame(OrderScrollableFrame,
height=120, fg_color="#104e78")

    buttonEdit = customtkinter.CTkButton(frameCard, text="Edit",
height=50)

    buttonDell = customtkinter.CTkButton(frameCard, text="Dell",
height=50)

    TextClientName = customtkinter.CTkLabel(frameCard,
text="Customer: Ivan Ivanov", font=customtkinter.CTkFont(size=16),
text_color="white")

    OrderDetails = customtkinter.CTkEntry(frameCard, width=470,
height=65)

    TextDate = customtkinter.CTkLabel(frameCard, text="Date:
20.11.2023", font=customtkinter.CTkFont(size=16), text_color="white")
```

```

    TextTime = customtkinter.CTkLabel(frameCard, text="Time: 19:00",
font=customtkinter.CTkFont(size=16, weight="bold"),
text_color="white")

    CardNumber = customtkinter.CTkLabel(frameCard, text="--1--",
font=customtkinter.CTkFont(size=16), text_color="white")

    OrderDetails.insert(0, "дві людини, салат: цезар, напої: томатний
сік")

    frameCard.pack(fill="x", padx=10, pady = 5)
    CardNumber.place(x=4, y=0)
    TextClientName.place(x=200, y=0)
    TextDate.place(x=200, y=21)
    TextTime.place(x=350, y=21)
    OrderDetails.place(x=2, y=50)

    buttonEdit.place(x= 480, y=5)
    buttonDell.place(x= 480, y= 60)

    orderCounter+=1
    CardNumber.configure(text="--"+ str(orderCounter) +"--")
    Orders_staticText.configure(text="Заказів на сьогодні[ "+
str(orderCounter) +" ] :")

def getImage():
    global tempImage
    file_path = filedialog.askopenfilename()
    tempImage = Image.open(file_path)
    loadFood()

def loadFood():

```

```

global foodCounter, tempImage, currentFrameFood

if currentFrameFood:
    currentFrameFood[-1].destroy()
    currentFrameFood.pop()

    frameFood = customtkinter.CTkFrame(FoodScrollableFrame,
height=300, fg_color="#104e78")
    frameFood.pack(pady=10, fill="x")
    currentFrameFood.append(frameFood)

    FoodNameText = customtkinter.CTkLabel(frameFood, text="["+
str(foodCounter) +"] Food name:",
font=customtkinter.CTkFont(size=16), text_color="white")

    ImageOFFoodCanvas = customtkinter.CTkCanvas(frameFood, width=250,
height=250)

    buttonEditAboutText = customtkinter.CTkButton(frameFood,
text="Підтвердити опис", height=30)

    buttonEditImage = customtkinter.CTkButton(frameFood,
text="Змінити зображення", height=30, command=getImage)

    TextAbout = customtkinter.CTkTextbox(frameFood, width=365,
height=190)

    TextNameFood = customtkinter.CTkEntry(frameFood, width=500,
height=25)

if tempImage is not None:
    canvas_width = 250
    canvas_height = 250

    img_width, img_height = tempImage.size
    scale = min(canvas_width / img_width, canvas_height /
img_height)

```

```

        new_width = int(img_width * scale)
        new_height = int(img_height * scale)

        tempImage = tempImage.resize((new_width, new_height),
Image.LANCZOS)

        image_tk = ImageTk.PhotoImage(tempImage)

        ImageOFFoodCanvas.image_tk = image_tk

        ImageOFFoodCanvas.create_image(canvas_width // 2,
canvas_height // 2, image=image_tk)

        tempImage = None

FoodNameText.place(x=4, y=0)
TextNameFood.place(x=120, y=2)
ImageOFFoodCanvas.place(x=385, y=30)
buttonEditAboutText.place(x=15, y=255)
buttonEditImage.place(x=230, y=255)
TextAbout.place(x=10, y=50)

foodCounter += 1

def addFood():
    loadFood()

if __name__ == "__main__":
    customtkinter.set_appearance_mode("light")

```

```

window = customtkinter.CTk()
window.minsize(1480, 600)
#window.geometry("1920x1080")
window.title("RestoOrderManager")

window.grid_rowconfigure(1, weight=1)
##window.grid_columnconfigure(0, weight=1)
window.grid_columnconfigure(5, weight=1)

Orders_staticText = customtkinter.CTkLabel(window, text="заказів  
на сьогодні [ 0 ] :", font=customtkinter.CTkFont(size=14,  
weight="bold"))

OrderScrollableFrame = customtkinter.CTkScrollableFrame(window,  
width=650, height=500)

Food_staticText = customtkinter.CTkLabel(window, text="Страви",  
font=customtkinter.CTkFont(size=14, weight="bold"))

FoodScrollableFrame = customtkinter.CTkScrollableFrame(window,  
width=650, height=500)

buttonLeft = customtkinter.CTkButton(window, text="<<",  
height=BUTTONS_HEIGHT)

SomeButton = customtkinter.CTkButton(window, text="Перейти до",  
height=BUTTONS_HEIGHT, command=addCardOrder)

buttonRight = customtkinter.CTkButton(window, text=">>",  
height=BUTTONS_HEIGHT)

buttons_frame_admin = customtkinter.CTkFrame(window,  
fg_color="#C9C9C9")

buttonAddFood = customtkinter.CTkButton(buttons_frame_admin,  
text="Додати страву", height=BUTTONS_HEIGHT, command=addFood)

buttonChangeFood = customtkinter.CTkButton(buttons_frame_admin,  
text="Змінити страву", height=BUTTONS_HEIGHT)

```

```

        buttonDellFood = customtkinter.CTkButton(buttons_frame_admin,
text="Видалити страву", height=BUTTONS_HEIGHT)

        buttonLock_out_customer =
customtkinter.CTkButton(buttons_frame_admin, text="Заблокувати
клієнта", height=BUTTONS_HEIGHT)

        buttonStopAutoOrdering =
customtkinter.CTkButton(buttons_frame_admin, text="Stop auto-
ordering", height=BUTTONS_HEIGHT)

        buttonStartAutoOrdering =
customtkinter.CTkButton(buttons_frame_admin, text="Start auto-
ordering", height=BUTTONS_HEIGHT)


        entry_foodSearch = customtkinter.CTkEntry(FoodScrollableFrame,
placeholder_text="пошук страви")


        Orders_staticText.grid(row=0, column=0, columnspan=3)
        OrderScrollableFrame.grid(row=1, column=0, columnspan=3,
sticky="ns")

        buttonLeft.grid(pady= 10, row=2, column=0)
        SomeButton.grid(pady= 10, row=2, column=1)
        buttonRight.grid(pady= 10, row=2, column=2)


        buttons_frame_admin.grid(row=0, column=4, rowspan=5, sticky="ns")
        FoodScrollableFrame.grid(row=1, column=5, rowspan=5,
sticky="nsew")


        buttonAddFood.grid(pady= 30, row=0, column=0)
        buttonDellFood.grid(pady= 20, row=1, column=0)
        buttonDellFood.grid(pady= 20, row=2, column=0)
        buttonLock_out_customer.grid(pady= 30, row=3, column=0)
        buttonStartAutoOrdering.grid(pady= 30, row=4, column=0)
        buttonStopAutoOrdering.grid(pady= 30, row=5, column=0)

```

```

    Food_staticText.grid(row=0, column=5)
    FoodScrollableFrame.grid(row=1, column=5, rowspan=5,
sticky="nsew")
    entry_foodSearch.pack(fill="x", padx=10)

    loadFood()

    window.mainloop()

```

Лістинг A.2 – tgBot.py

```

import os
import time
import telebot
import torch
import psycopg2
import PIL

from telebot.types import ReplyKeyboardMarkup, KeyboardButton
from transformers import AutoModelForCausalLM, AutoTokenizer

menuPath = "./data/menu.png"
galleryPaths = ["./data/preview1.jpg", "./data/preview2.jpg",
"./data/facaderesto.jpg"]
tableSchemePath = "./data/tables.jpg"

table_coordinates_first_floor = {
    1: {"top_left": (100, 200), "bottom_right": (150, 250)},
    2: {"top_left": (160, 200), "bottom_right": (210, 250)},
    3: {"top_left": (220, 200), "bottom_right": (270, 250)},
    4: {"top_left": (280, 200), "bottom_right": (330, 250)},

```

```

5: {"top_left": (340, 200), "bottom_right": (390, 250)},
6: {"top_left": (100, 260), "bottom_right": (150, 310)},
7: {"top_left": (160, 260), "bottom_right": (210, 310)},
8: {"top_left": (220, 260), "bottom_right": (270, 310)},
9: {"top_left": (280, 260), "bottom_right": (330, 310)},
10: {"top_left": (340, 260), "bottom_right": (390, 310)},
11: {"top_left": (400, 200), "bottom_right": (450, 250)},
12: {"top_left": (460, 200), "bottom_right": (510, 250)},
13: {"top_left": (520, 200), "bottom_right": (570, 250)}
}

connection = psycopg2.connect(
    dbname="restodb", user="postgres", password="password",
    host="localhost"
)

cursor = connection.cursor()

tokenizer = AutoTokenizer.from_pretrained("distilgpt2")
model = AutoModelForCausalLM.from_pretrained("distilgpt2")

TOKEN = "TOKEN"
bot = telebot.TeleBot(TOKEN)

def main_menu():
    markup = ReplyKeyboardMarkup(resize_keyboard=True)
    button_menu = KeyboardButton("Показати меню")
    button_table = KeyboardButton("Замовити столик")
    markup.add(button_menu, button_table)
    return markup

def menu_options():

```

```

markup = ReplyKeyboardMarkup(resize_keyboard=True)
button_delivery = KeyboardButton("Доставити по адресу")
button_eat_here = KeyboardButton("Планую їсти в ресторані")
button_takeaway = KeyboardButton("Заберу з собою")
markup.add(button_delivery, button_eat_here, button_takeaway)
return markup

@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.send_message(
        message.chat.id,
        "Привіт! Я RestoBot, ваш помічник у ресторані.\n"
        "Виберіть дію або напишіть номер столика, щоб перевірити його"
        "доступність.",
        reply_markup=main_menu()
    )

@bot.message_handler(func=lambda message: message.text == "Показати"
меню")
def show_menu(message):
    with open(menuPath, "rb") as menu_image:
        bot.send_document(message.chat.id, menu_image, caption="Ось"
наше меню")
    bot.send_message(
        message.chat.id,
        "Виберіть один із варіантів дій:",
        reply_markup=menu_options()
    )

@bot.message_handler(func=lambda message: message.text == "Замовити"
столок")

```

```

def reserve_table(message):
    for photo_path in galleryPaths:
        with open(photo_path, "rb") as photo:
            bot.send_photo(message.chat.id, photo)

    bot.send_message(
        message.chat.id,
        "Ознайомтесь з нашим інтер'єром. Так виглядає наш ресторан!"
    )

    with open(tableSchemePath, "rb") as table_scheme:
        bot.send_document(message.chat.id, table_scheme, caption="Ось
схема наших столиків. Оберіть один, вкажіть дату та час для
бронювання.")

@bot.message_handler(func=lambda message: message.text.isdigit())
def check_table_availability(message):
    table_number = int(message.text)
    cursor.execute("SELECT status FROM tables WHERE tableid = %s",
(table_number,))
    result = cursor.fetchone()

    if result and result[0] == "вільний":
        bot.send_message(
            message.chat.id,
            f"Столик {table_number} вільний. Якщо хочете його
зарезервувати, напишіть час, наприклад: 'на 15:30'."
        )
    elif result:
        bot.send_message(
            message.chat.id,

```

```

        f"На жаль, столик {table_number} зараз зайнятий.  

        Спробуйте обрати інший."
    )
    else:
        bot.send_message(
            message.chat.id,
            f"Столик номер {table_number} не існує. Будь ласка,  

            виберіть столик від 1 до 29."
        )

@bot.message_handler(func=lambda message: "на" in message.text and
": " in message.text)
def reserve_time(message):
    if message.chat.id in user_states and "table" in
user_states[message.chat.id]:
        reservation_time = message.text.split("на")[1].strip()
        table_number = user_states[message.chat.id]["table"]
        cursor.execute("UPDATE tables SET status = 'Зарезервирован'  

WHERE tableid = %s", (table_number,))
        connection.commit()
        bot.send_message(
            message.chat.id,
            f"Столик {table_number} зарезервований на  

{reservation_time}. Дякуємо за бронювання!"
        )
    else:
        bot.send_message(message.chat.id, "Спочатку виберіть столик,  

перед тим як вказати час.")

print("Бот запущен і готовий приймати команди...")
bot.infinity_polling()

```

Лістинг А.3 – Файл prompt.txt

Ви - штучний інтелект, який приймає запити користувача на основі наступного меню і повертає назви обраних позицій у строгому форматі <Назва позиції з меню>, а також додаткові дані у відповідному форматі.

Меню:

Appetizer:

Crispy Calamari

Spinach Artichoke Dip

Stuffed Portobello Mushrooms

Lobster Bisque

Mediterranean Quinoa Salad

Creamy Tomato Basil Soup

Caesar Salad

Main Dish:

Rosemary Grilled Lamb Chops

Smoky BBQ Sandwich

Garlic Butter Steak Medallions

Honey Glazed Chicken Breast

Vegetable Risotto

Grilled Salmon

Filet Mignon

Spicy Cajun Sausage Jambalaya

Lemon Herb Baked Salmon

Thai Red Curry Shrimp

Beverages:

Beer

Milk Tea

Iced Tea

Apple Juice

Orange Juice

Lime Soda

Lemon Soda

Desserts:

Bacon Waffles

New York Cheesecake

Crème Brûlée

Ice Cream Sundae

Apple Pie à la Mode

Правила роботи:

1. Ви аналізуєте запит користувача, щоб знайти відповідні позиції з меню.
2. Якщо користувач вказує спосіб отримання замовлення:
 - "заберу з собою" – додати позначку [3].
 - "планую їсти в ресторані" – додати позначку [2].
 - "доставити за адресою" – додати позначку [1].
3. Якщо користувач вказує час:
 - Додати позначку [timer: X] (час в хвилинах або годинах).
4. Якщо користувач вказує адресу або місце, де планує бути:
 - Додати позначку [location: текст адреса].
5. Формат відповіді:
 - <Назва позиції><Назва позиції>...[X][додаткові дані]
6. Якщо запит не містить ключових деталей, ви намагаєтеся інтерпретувати його максимально точно.
7. Ігноруйте друкарські помилки і неточності.

Приклади запитів і відповідей:

Приклад 1:

Введення користувача: "на десерт вафлі і сендвіч bbq і соду я прийду через 30 хвилин заберу з собою"

Ваша відповідь: <Bacon Waffles><Smoky BBQ Sandwich><Lime Soda>[3][timer: 30]

Приклад 2:

Введення користувача: "ваші салати і сік, принесіть у кімнату 76, проспект 117А"

Ваша відповідь: <Mediterranean Quinoa Salad><Apple Juice>[1][location: проспект 117А кімната 76]

Приклад 3:

Введення користувача: "куряче філе, чизкейк і лимонад, буду в ресторані через годину"

Ваша відповідь: <Honey Glazed Chicken Breast><New York Cheesecake><Lemon Soda>[2][timer: 60]

Додаткові вказівки:

1. Якщо користувач вводить неточні або часткові назви (наприклад, «чизкейк» замість «New York Cheesecake»), намагайтеся знайти максимально близьку відповідність.
2. Якщо користувач не вказує спосіб отримання, але зазначає час або місце, ви робите висновок на основі контексту.
3. Усі відповіді формуються строго у вказаному форматі.