ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Чат-бот для роботи з базами даних користувачів з інтегрованим штучним інтелектом»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Штучний інтелект
(назва)

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

_____ Денис ГОЦАЛО
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти
група ШІД-42
Денис ГОЦАЛО

Керівник:
науковий ступінь,
вчене звання
Тетяна КИСІЛЬ

Рецензент:
науковий ступінь,
вчене звання
_____
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту

_____ Ольга ЗІНЧЕНКО
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гоцало Денис Віталійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Чат-бот для роботи з базами даних користувачів з інтегрованим штучним інтелектом
керівник кваліфікаційної роботи Тетяна КИСІЛЬ, старший викладач,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «23» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з AI-чат-ботів, офіційна документація Django, OpenAI, Python, приклади інтеграції GPT-4o.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області – тренди української e-commerce; огляд AI-чат-ботів; порівняння стеків.
2. Огляд існуючих AI-чат-ботів, порівняння технологічних стеків.
3. Обґрунтування вибору технологій: Python 3.12 + Django 5.2, DRF, GPT-4o, Bootstrap 5.3, SQLite → PostgreSQL.

4. Проєктування системи Архітектура: «Client – Django App – OpenAI Gateway – DB»; ER-модель БД; UI/UX-макети.
5. Реалізація веб-застосунку та чат-бота: REST API, ChatBotAPIView (FAQ → Regex → GPT), function calling update_price.
6. Тестування й оцінка ефективності: Unit-інтеграційні тести

5. Перелік графічного матеріалу: *презентація*
   1. Sequence-діаграма запиту «Показати товар»
   2. ER-діаграма бази даних, UI-макет головної сторінки, UI-макет чат-віджета

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз літератури й ринку AI-ботів	25.02 – 05.03	виконано
2	Вибір та обґрунтування технологічного стеку	06.03 – 12.03	виконано
3	Проєктування архітектури й ER-схеми	13.03 – 20.03	виконано
4	Розробка базового веб-інтерфейсу магазину	21.03 – 05.04	виконано
5	Імплементація чат-бота (FAQ, Regex, GPT-4o)	06.04 – 25.04	виконано
6	Тестування (unit, інтеграційне, навантажувальне)	26.04 – 09.05	виконано
7	Оформлення пояснювальної записки, реферату	10.05 – 25.05	виконано
8	Підготовка демонстраційних матеріалів	26.05 – 05.06	виконано

Здобувач вищої освіти _____ Денис Гоцало
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Тетяна КИСІЛЬ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 74 стор., 12 рис., 33 джерел.

Мета роботи. Підвищити ефективність клієнтської підтримки малого e-commerce шляхом розробки веб-застосунку «E-Shop», який об’єднує онлайн-крамницю з інтелектуальним чат-асистентом. Бот уміє зчитувати та безпечно оновлювати базу даних користувачів у реальному часі, використовуючи мовну модель GPT-4o.

Об’єкт дослідження. Процес взаємодії покупця з інтернет-магазином під час пошуку, вибору й купівлі товарів.

Предмет дослідження. Методи інтеграції чат-ботів на основі великих мовних моделей із реляційною базою даних у системі електронної комерції.

Короткий опис роботи. У кваліфікаційній роботі проаналізовано наявні платформи та підходи до розробки AI-ботів. Для розробки та тестування системи використовувалася база даних SQLite, з можливістю подальшого переходу на PostgreSQL для продуктивного середовища. Фронтенд веб-застосунку розроблено з використанням Bootstrap 5.3, забезпечуючи адаптивний та сучасний інтерфейс. Центральним елементом роботи є інтеграція великої мовної моделі OpenAI GPT-4o, яка дозволяє чат-боту розуміти складні запити користувачів та генерувати природні відповіді. Реалізовано гібридний пайплайн обробки запитів, який ефективно поєднує попередньо визначені відповіді (FAQ), логіку на основі регулярних виразів та взаємодію з Django ORM, а також звернення до GPT-4o для складних сценаріїв.

КЛЮЧОВІ СЛОВА**:** ЧАТ-БОТ, GPT-4o, DJANGO, REST API, FUNCTION CALLING, E-COMMERCE, РЕЛЯЦІЙНА БАЗА ДАНИХ, SQLITE, POSTGRESQL.

ABSTRACT

The text part of the bachelor's qualification work: 74 pages, 12 figures, 33 sources.

Purpose of the work. To increase the efficiency of customer support for small e-commerce by developing a web application "E-Shop" that combines an online store with an intelligent chat assistant. The bot is able to read and securely update user databases in real-time, using the GPT-4o language model.

Object of research. The process of customer interaction with an online store during product search, selection, and purchase.

Subject of research. Methods of integrating chatbots based on large language models with a relational database in an e-commerce system.

Brief description of the work. The qualification work analyzes existing platforms and approaches to AI bot development. The SQLite database was used for system development and testing, with the possibility of further transition to PostgreSQL for a production environment. The frontend of the web application was developed using Bootstrap 5.3, providing an adaptive and modern interface. The central element of the work is the integration of the large language model OpenAI GPT-4o, which allows the chatbot to understand complex user queries and generate natural responses. A hybrid request processing pipeline has been implemented, which effectively combines predefined responses (FAQ), regular expression-based logic, and interaction with Django ORM, as well as calls to GPT-4o for complex scenarios.

KEYWORDS: CHATBOT, GPT-4O, DJANGO, REST API, FUNCTION CALLING, E-COMMERCE, RELATIONAL DATABASE, SQLITE, POSTGRESQL.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, що динамічно розвивається, де цифровізація охоплює всі сфери життя, зростає потреба в ефективних та інтуїтивно зрозумілих інструментах для взаємодії з інформацією. Особливо це стосується роботи з великими обсягами даних, зокрема базами даних користувачів. Традиційні інтерфейси часто вимагають спеціальних знань та навичок, що обмежує доступність для пересічних користувачів та сповільнює процеси обробки інформації. З появою та стрімким розвитком технологій штучного інтелекту, зокрема обробки природної мови (NLP) та великих мовних моделей (LLM), виникає унікальна можливість створити більш гнучкі та інтелектуальні системи взаємодії. Чат-боти, інтегровані зі штучним інтелектом, можуть забезпечити зручний, природний та ефективний спосіб доступу та управління даними користувачів, автоматизуючи рутинні операції та надаючи миттєві відповіді на запити. Таким чином, розробка чат-бота для роботи з базами даних користувачів з інтегрованим штучним інтелектом є надзвичайно актуальною задачею, що відповідає сучасним викликам інформаційного суспільства та має значний потенціал для оптимізації бізнес-процесів та покращення користувацького досвіду.

Метою дипломної роботи є розробка та реалізація чат-бота для ефективної та інтелектуальної взаємодії з базами даних користувачів за допомогою інтегрованого штучного інтелекту.

Для досягнення поставленої мети необхідно вирішити такі *завдання*: провести аналіз теоретичних засад функціонування чат-ботів, принципів роботи з базами даних та методів інтеграції штучного інтелекту; вивчити існуючі рішення та платформи для розробки чат-ботів з інтеграцією штучного інтелекту, а також проаналізувати їхні переваги та недоліки; сформулювати функціональні та нефункціональні вимоги до майбутньої системи; розробити архітектуру чат-бота, включаючи структуру бази даних користувачів, модулі взаємодії з базою даних та інтеграції зі штучним інтелектом; вибрати та обґрунтувати технології та інструменти, необхідні для реалізації проекту; розробити та реалізувати програмне

забезпечення чат-бота відповідно до спроектованої архітектури; провести тестування розробленої системи для перевірки її функціональності, надійності та відповідності поставленим вимогам; оцінити отримані результати та сформулювати висновки щодо практичної цінності та перспектив подальшого розвитку.

Об'єктом дослідження є процеси взаємодії користувачів з інформаційними системами, що містять бази даних.

Предметом дослідження є методи та засоби створення чат-ботів з інтегрованим штучним інтелектом для автоматизованої обробки запитів та взаємодії з базами даних користувачів.

Практичне значення отриманих результатів полягає у створенні функціонального чат-бота, який забезпечить зручний та ефективний доступ до даних користувачів без необхідності прямого використання складних інтерфейсів баз даних. Розроблена система може бути використана в різних сферах, таких як клієнтська підтримка, внутрішній облік компаній, управління персоналом або будь-яка інша галузь, що потребує швидкого та інтуїтивно зрозумілого доступу до інформації про користувачів. Це дозволить автоматизувати рутинні операції, знизити навантаження на персонал, підвищити швидкість обробки запитів та загалом покращити якість обслуговування.

Структура дипломної роботи. Дипломна робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі висвітлюються теоретичні основи чат-ботів, баз даних та принципів інтеграції штучного інтелекту. Другий розділ присвячений проектуванню архітектури системи, включаючи вибір технологій та розробку структури бази даних. Третій розділ описує процес реалізації та тестування розробленого чат-бота.

1. ТЕОРЕТИЧНІ ОСНОВИ ЧАТ-БОТІВ ТА РОБОТИ З БАЗАМИ ДАНИХ

1.1 Загальні поняття віртуальних асистентів

Чат-бот (від англ. "chat" – розмова, "bot" – робот) – це комп'ютерна програма, призначена для імітації людської розмови через текстові або голосові методи взаємодії. Його основна функція полягає в автоматизованій комунікації з користувачами, відповідаючи на їхні запити, надаючи інформацію або виконуючи певні дії. Чат-боти можуть бути використані в широкому спектрі застосувань: від обслуговування клієнтів і технічної підтримки до персональних асистентів і освітніх платформ.

1.1.1 Історія розвитку та класифікація чат-ботів

Історія розвитку чат-ботів бере свій початок у середині XX століття. Одним із перших значущих кроків став **ELIZA**, розроблений Джозефом Вейценбаумом у 1966 році. Ця програма імітувала діалог психотерапевта, використовуючи прості правила зіставлення зразків та переформулювання фраз. Хоча ELIZA не "розуміла" змісту розмови, вона демонструвала вражаючу здатність підтримувати діалог, що заклало основи для подальших досліджень. Ці ранні розробки були засновані на правилах і не мали штучного інтелекту в сучасному розумінні [2].

Значний прорив відбувся з появою інтернету та розвитком технологій штучного інтелекту, зокрема обробки природної мови (NLP) та машинного навчання. У 1990-х роках з'явилися перші чат-боти для онлайн-спільнот та месенджерів. Початок 2000-х років ознаменувався сплеском інтересу до чат-ботів після появи стандарту AIML (Artificial Intelligence Markup Language) та участі ботів у премії Лобнера (Loebner Prize), яка присуджується програмам, що найближче імітують людську розмову. Сучасний етап розвитку чат-ботів характеризується інтеграцією з потужними мовними моделями, такими як GPT-o3/4o/4-5, що дозволяє їм розуміти контекст, генерувати більш складні та релевантні відповіді, а також навчатися на великих обсягах даних.

Чат-боти класифікуються за двома основними принципами: за принципом роботи та за призначенням.

За *принципом роботи* розрізняють три основні типи. *Правилові (Rule-based)* чат-боти функціонують на основі заздалегідь визначених правил, скриптів та ключових слів. Вони ефективні для вирішення типових, чітко визначених завдань, проте їхні можливості обмежені у розумінні складних або непередбачуваних запитів. Гібридні (Hybrid) системи поєднують у собі елементи правилових підходів з технологіями штучного інтелекту, що дозволяє їм обробляти як стандартні, так і більш складні запити, які вимагають розуміння контексту. Нарешті, чат-боти на основі штучного інтелекту (AI-powered) активно використовують машинне навчання, глибоке навчання та обробку природної мови (NLP) для комплексного розуміння природної мови, вилучення сутностей, аналізу сентименту та генерації відповідей. Такі боти відрізняються здатністю до навчання та постійного вдосконалення взаємодії з часом.

За призначенням чат-боти виконують різноманітні функції. Інформаційні боти надають користувачам довідкову інформацію, таку як графік роботи або відповіді на поширені запитання. Транзакційні чат-боти дозволяють користувачам виконувати певні дії, наприклад, замовляти товари, бронювати квитки або змінювати дані. Боти підтримки клієнтів допомагають вирішувати проблеми користувачів, відповідати на їхні запитання та, за потреби, перенаправляти їх до живого оператора. Маркетингові/продажні чат-боти призначені для залучення потенційних клієнтів, консультування щодо продуктів та збору лідів. Окрему категорію складають внутрішні (корпоративні) чат-боти, які автоматизують внутрішні процеси компаній, надають співробітникам доступ до корпоративних даних та спрощують їхню роботу.

1.1.2 Архітектура чат-ботів та основні компоненти

Архітектура сучасного чат-бота, особливо того, що інтегрований зі штучним інтелектом та працює з базами даних, є багатошаровою та складається з декількох ключових компонентів:

1. **Канал взаємодії (Interface Layer):** Це точка входу для користувача. Чат-бот може бути розгорнутий на різних платформах:
   - Веб-інтерфейс (віджет на сайті).

- Месенджери (Telegram, Viber, Facebook Messenger, WhatsApp).
- Голосові помічники (Google Assistant, Amazon Alexa, Siri).
- Корпоративні платформи (Slack, Microsoft Teams). Цей шар відповідає за прийом повідомлень від користувача та відправку відповідей.

2. **Модуль обробки природної мови (Natural Language Processing - NLP):** Цей компонент є "серцем" інтелектуального чат-бота і відповідає за розуміння вхідних запитів користувача. Основні функції NLP-модуля включають:

- **Токенізація:** Розбиття тексту на окремі слова або фрази (токени).
- **Лемматизація/Стеммінг:** Зведення слів до їх базової форми.
- **Розпізнавання сутностей (Named Entity Recognition - NER):** Виявлення та класифікація іменованих сутностей (наприклад, імена людей, назви місць, дати).
- **Визначення наміру (Intent Recognition):** Ідентифікація мети або наміру користувача (наприклад, "дізнатися баланс", "замовити товар", "змінити дані").
- **Аналіз сентименту:** Визначення емоційного забарвлення повідомлення (позитивне, негативне, нейтральне).
- **Контекстне розуміння:** Збереження та використання інформації з попередніх повідомлень для підтримки зв'язного діалогу. У сучасних чат-ботах для NLP часто використовуються великі мовні моделі (LLM), які здатні виконувати ці функції з високою точністю.

3. **Модуль управління діалогом (Dialogue Manager):** Цей компонент координує потік розмови. Він отримує "зрозумілий" запит від NLP-модуля та визначає наступний крок. Його завдання:

- Відстеження стану діалогу.
- Визначення, яку інформацію необхідно отримати від користувача для виконання завдання.
- Виклик відповідних функцій або інтеграцій (наприклад, до бази даних).
- Обробка непередбачених ситуацій або відхилень від сценарію.
- Передача даних модулю генерації відповідей.

4. **Модуль інтеграції з базою даних (Database Integration Module):** Цей компонент відповідає за взаємодію з зовнішніми базами даних. Він отримує запити від модуля управління діалогом, формує відповідні SQL або NoSQL запити, виконує їх у базі даних, а потім повертає отримані дані назад до чат-бота. Цей модуль має забезпечувати безпечний та ефективний доступ до даних користувачів.

5. **Модуль генерації відповідей (Response Generation Module):** Отримавши необхідну інформацію від модуля управління діалогом або бази даних, цей компонент формує відповідь для користувача. Відповіді можуть бути:

- **Заздалегідь визначені (Pre-defined):** Зберігаються у базі знань бота для типових сценаріїв.
- **Сформовані динамічно:** Створюються на основі отриманих даних (наприклад, "Ваш баланс становить: XXX грн.").
- **Згенеровані ШІ:** Використовують мовні моделі для створення природних та контекстуально релевантних відповідей, що не були заздалегідь запрограмовані.

6. **База знань (Knowledge Base):** Містить інформацію, яку чат-бот використовує для відповідей на запитання. Це можуть бути FAQs, статті, продукти, послуги, а також дані про користувачів (у випадку інтеграції з БД).

7. **Адміністративна панель/Модуль моніторингу (Admin Panel/Monitoring Module):** Дозволяє розробникам або адміністраторам відстежувати роботу бота, аналізувати діалоги, виправляти помилки, оновлювати базу знань та тренувати моделі ШІ.

1.1.3 Принципи взаємодії користувача з чат-ботом

Взаємодія користувача з чат-ботом ґрунтується на принципах діалогової системи та має кілька ключових етапів:

1. **Ініціація діалогу:** Користувач починає розмову з чат-ботом. Це може бути привітання ("Привіт!"), конкретне запитання ("Який мій баланс?"), або вибір опції з наданого меню. У деяких випадках бот може ініціювати розмову сам (наприклад, вітальне повідомлення на сайті).

2. **Введення запиту користувачем:** Користувач формулює свій запит у природній мові (текстом або голосом). Важливо, щоб чат-бот був здатний розуміти різні формулювання одного і того ж наміру.

3. **Обробка запиту чат-ботом:**
   - **Розуміння:** NLP-модуль чат-бота аналізує вхідне повідомлення, визначаючи намір користувача та вилучаючи ключові сутності.
   - **Виконання дії:** Модуль управління діалогом, на основі визначеного наміру, звертається до відповідного компонента системи. У випадку роботи з базами даних, це буде звернення до модуля інтеграції з БД для отримання, оновлення або видалення інформації.
   - **Формування відповіді:** Після отримання необхідних даних або завершення дії, модуль генерації відповідей формує релевантну та зрозумілу відповідь для користувача.

4. **Відповідь чат-бота:** Бот надає відповідь користувачеві у текстовому або голосовому форматі. Відповідь може містити інформацію, запитання для уточнення, підтвердження виконаної дії або пропозицію подальших опцій.

5. **Продовження або завершення діалогу:** Користувач може продовжити діалог, ставлячи додаткові запитання, або завершити його, якщо отримав потрібну інформацію або виконав необхідну дію. Інтелектуальні чат-боти здатні підтримувати складніші діалоги, зберігаючи контекст розмови.

Ефективність взаємодії з чат-ботом значною мірою визначається його здатністю розуміти природну мову, навіть якщо вона містить помилки, сленг або діалекти. Важливо, щоб бот міг підтримувати контекст розмови, пам'ятаючи попередні запити та відповіді для осмисленого та логічного продовження діалогу. Крім того, ключовим аспектом є надання точних та релевантних відповідей, адже інформація, яку надає бот, має бути достовірною та відповідати суті запиту користувача. Ефективний чат-бот також повинен бути гнучким у реагуванні, здатним перемикатися між різними темами розмови або пропонувати додаткову допомогу у випадках, коли початковий запит не може бути виконаний. Нарешті, для повноцінної взаємодії необхідно, щоб бот забезпечував зворотний зв'язок,

інформуючи користувача про хід виконання його запиту або пояснюючи причини, з яких запит не може бути оброблений.

1.2 Бази даних та системи управління базами даних (СУБД)

У контексті розробки чат-бота, що взаємодіє з даними користувачів, розуміння принципів роботи з базами даних та систем управління базами даних (СУБД) є фундаментальним. База даних (БД) – це організована сукупність структурованих даних, призначених для зберігання, керування та швидкого доступу до інформації. СУБД – це програмне забезпечення, що дозволяє створювати, підтримувати та маніпулювати базами даних, забезпечуючи їх цілісність, безпеку та ефективність.

1.2.1 Види баз даних та їх особливості

Існує безліч видів баз даних, кожен з яких має свої особливості, переваги та недоліки, що робить їх придатними для різних типів завдань. Основні види включають:

1. **Реляційні бази даних (RDBMS):**
   - **Особливості:** Дані організовані у вигляді таблиць, що складаються з рядків (записів) та стовпців (полів). Взаємозв'язки між таблицями встановлюються за допомогою ключів (первинних та зовнішніх). Вони відповідають принципам ACID (Atomicity, Consistency, Isolation, Durability – Атомарність, Узгодженість, Ізольованість, Довговічність), що гарантує надійність транзакцій.
   - **Приклади:** MySQL, PostgreSQL, Oracle, Microsoft SQL Server, SQLite.
   - **Переваги:** Висока цілісність даних, надійність транзакцій, добре розвинені інструменти та спільнота, підтримка складних запитів.
   - **Недоліки:** Жорстка схема даних (складно змінювати структуру після створення), масштабованість може бути викликом для дуже великих обсягів даних або високого навантаження.

- **Застосування:** Фінансові системи, управління запасами, облікові системи, CRM – там, де критично важлива цілісність та строга структуризація даних.

2. **Нереляційні бази даних (NoSQL databases):**

- **Особливості:** Розроблені для роботи з великими обсягами неструктурованих або напівструктурованих даних, забезпечуючи високу масштабованість та гнучкість схеми. Вони відмовляються від жорстких вимог ACID на користь моделі BASE (Basically Available, Soft state, Eventually consistent – Базова доступність, Гнучкий стан, Узгодженість у підсумку), що дозволяє досягти більшої продуктивності та горизонтальної масштабованості.
- **Види та приклади:**
  - **Документо-орієнтовані (Document-oriented):** Зберігають дані у вигляді документів (зазвичай JSON або BSON), що дозволяє гнучко додавати нові поля. Приклади: MongoDB, Couchbase, RavenDB.
  - **Ключ-значення (Key-Value):** Зберігають дані як прості пари ключ-значення: Redis, Memcached, DynamoDB.
  - **Колонкові (Column-family):** Оптимізовані для зберігання та обробки великих обсягів даних у вигляді сімейств колонок, що дозволяє швидко вибирати групи пов'язаних даних. Приклади: Cassandra, HBase.
  - **Графові (Graph databases):** Зберігають дані у вигляді вузлів (entities) та ребер (relationships), що дозволяє ефективно працювати зі складними взаємозв'язками. Приклади: Neo4j, ArangoDB.
- **Переваги:** Гнучкість схеми, висока масштабованість (горизонтальна), висока продуктивність для певних типів даних та запитів, можливість роботи з неструктурованими даними.
- **Недоліки:** Менша цілісність даних порівняно з RDBMS (для деяких видів), складніші для аналізу складних зв'язків між різними типами даних, менш розвинені інструменти порівняно з реляційними БД.
- **Застосування:** Великі дані, соціальні мережі, системи рекомендацій, аналітика в реальному часі, системи з частими змінами структури даних.

3. **Внутрішньопам'ятні бази даних (In-Memory Databases - IMDB):**
   - **Особливості:** Зберігають основну частину або всі дані у оперативній пам'яті комп'ютера, що забезпечує надзвичайно високу швидкість доступу та обробки.
   - **Приклади:** Redis (може виступати як IMDB), SAP HANA, VoltDB.
   - **Переваги:** Максимальна продуктивність, мінімальна затримка.
   - **Недоліки:** Висока вартість (оперативна пам'ять дорожча за дискову), втрата даних при збоях живлення (якщо немає механізму персистентності), обмеження обсягу даних розміром ОЗП.
   - **Застосування:** Кешування, ігрова індустрія, фінансові транзакції в реальному часі, системи IoT.

При виборі типу бази даних для чат-бота, що працює з даними користувачів, необхідно враховувати характер даних (структуровані чи ні), обсяг даних, вимоги до продуктивності та масштабованості, а також необхідність підтримки транзакцій. Для зберігання профілів користувачів, їхніх уподобань, історії взаємодії та інших структурованих даних часто використовують реляційні БД, тоді як для логів, сесій чату або неструктурованих даних можуть бути більш підходящими NoSQL рішення.

1.2.2 Принципи проектування баз даних

Ефективне проектування бази даних є критично важливим для забезпечення її продуктивності, цілісності, масштабованості та зручності використання. Основні принципи проектування включають:

1. **Нормалізація:** Це процес організації стовпців і таблиць реляційної бази даних, щоб мінімізувати надлишковість даних та покращити їх цілісність. Нормалізація відбувається шляхом розбиття великих таблиць на менші, пов'язані таблиці, дотримуючись певних "нормальних форм" (1NF, 2NF, 3NF, BCNF та ін.). Наприклад, 3NF усуває транзитивні залежності.
   - **Переваги:** Зменшення надмірності даних, покращення цілісності даних, спрощення оновлення та видалення даних, ефективніше використання дискового простору.

- **Недоліки:** Збільшення кількості таблиць може ускладнити запити (потребує більше операцій з'єднання), що потенційно впливає на продуктивність.

2. **Денормалізація:** Це процес навмисного введення надмірності в базу даних, як правило, для покращення продуктивності читання за рахунок запису. Денормалізація часто застосовується після нормалізації, коли вузькі місця в продуктивності виявлені через складні запити з багатьма з'єднаннями.
   - **Переваги:** Прискорення виконання запитів, зменшення кількості операцій з'єднання.
   - **Недоліки:** Збільшення надмірності даних, потенційне порушення цілісності даних (якщо дані в декількох місцях не оновлюються синхронно), ускладнення операцій оновлення та вставки.
3. **Визначення сутностей та атрибутів:** Сутність – це об'єкт або концепція, яку необхідно зберігати в базі даних (наприклад, "Користувач", "Повідомлення", "Товар"). Атрибут – це властивість сутності (наприклад, для "Користувача" атрибути можуть бути "ім'я", "email", "дата реєстрації").
4. **Визначення зв'язків між сутностями:** Зв'язки описують, як сутності взаємодіють одна з одною. Існують три основні типи зв'язків:
   - **Один до одного (One-to-One):** Кожному екземпляру однієї сутності відповідає рівно один екземпляр іншої.
   - **Один до багатьох (One-to-Many):** Одному екземпляру однієї сутності відповідає багато екземплярів іншої.
   - **Багато до багатьох (Many-to-Many):** Багатьом екземплярам однієї сутності відповідає багато екземплярів іншої (часто реалізується через проміжну таблицю).
5. **Визначення первинних та зовнішніх ключів:**
   - **Первинний ключ (Primary Key):** Унікально ідентифікує кожен запис у таблиці. Не може містити нульові значення та має бути унікальним.
   - **Зовнішній ключ (Foreign Key):** Встановлює зв'язок між таблицями, посилаючись на первинний ключ іншої таблиці. Забезпечує цілісність посилань.

6. **Індексація:** Створення індексів для стовпців, за якими часто здійснюється пошук або сортування, значно прискорює виконання запитів. Однак надмірна кількість індексів може сповільнити операції запису (INSERT, UPDATE, DELETE).
7. **Безпека та контроль доступу:** Проектування бази даних повинно включати механізми для захисту даних від несанкціонованого доступу, такі як керування ролями та дозволами користувачів, шифрування даних.

Проектування бази даних є ітеративним процесом, який часто починається з концептуальної моделі (наприклад, ER-діаграми), переходить до логічної моделі, а потім до фізичної реалізації.

1.2.3 Мови запитів до баз даних (SQL та NoSQL)

Для взаємодії з базами даних використовуються спеціалізовані мови запитів.

1. **SQL (Structured Query Language):**
   - **Особливості:** Це стандартизована декларативна мова, призначена для керування та маніпулювання даними в реляційних базах даних. SQL дозволяє створювати, читати, оновлювати та видаляти дані (CRUD-операції), а також керувати структурою бази даних (створення таблиць, індексів, представлень).
   - **Основні категорії команд:**
     - **DDL (Data Definition Language):** Для визначення та керування структурою бази даних (CREATE TABLE, ALTER TABLE, DROP TABLE).
     - **DML (Data Manipulation Language):** Для маніпулювання даними (SELECT, INSERT, UPDATE, DELETE).
     - **DCL (Data Control Language):** Для керування правами доступу та безпекою (GRANT, REVOKE).
     - **TCL (Transaction Control Language):** Для керування транзакціями (COMMIT, ROLLBACK).
   - **Переваги:** Широко поширений стандарт, потужні можливості для складних запитів та аналізу даних, висока цілісність даних через підтримку транзакцій.

- **Недоліки:** Менш гнучкий для неструктурованих даних, масштабованість може бути викликом для дуже великих розподілених систем.
- **Застосування:** Більшість реляційних СУБД (MySQL, PostgreSQL, Oracle, SQL Server).

2. **NoSQL мови запитів:**
   - **Особливості:** Немає єдиного стандарту, як SQL. Мови запитів NoSQL баз даних сильно різняться залежно від типу БД та її внутрішньої структури. Зазвичай вони орієнтовані на конкретний тип моделі даних (документи, ключі-значення, графіки тощо) і оптимізовані для швидкого доступу та масштабованості.
   - **Приклади:**
     - **MongoDB:** Використовує мову запитів, схожу на JSON, для роботи з документами (MongoDB Query Language - MQL).
     - **Cassandra:** Використовує Cassandra Query Language (CQL), який має синтаксис, що дещо нагадує SQL, але оптимізований для колонкових БД.
     - **Redis:** Взаємодія відбувається через прості команди ключ-значення.
     - **Neo4j:** Використовує мову Cypher для запитів до графових баз даних.
   - **Переваги:** Висока гнучкість схеми, горизонтальна масштабованість, оптимізовані для роботи з неструктурованими даними та високими навантаженнями.
   - **Недоліки:** Відсутність єдиного стандарту, менш потужні для складних зв'язків та транзакцій у порівнянні з SQL, крива навчання для кожної окремої NoSQL СУБД.
   - **Застосування:** Системи, що потребують швидкого доступу до великих обсягів даних, гнучкості схеми, або роботи з неструктурованими даними (наприклад, профілі користувачів з динамічними атрибутами, логи, кешування).

Вибір між SQL та NoSQL залежить від вимог до проекту, типу даних, що зберігаються, очікуваних обсягів та навантаження. Для чат-бота, що працює з

даними користувачів, реляційна база даних може бути оптимальним вибором для зберігання структурованих профілів, тоді як NoSQL база може бути використана для зберігання логів чату або кешування даних.

1.3 Штучний інтелект у реалізації чат-ботів

Інтеграція штучного інтелекту (ШІ) є ключовим фактором, що перетворює базовий чат-бот на інтелектуальну діалогову систему, здатну розуміти складні запити, підтримувати контекст та генерувати релевантні та природні відповіді. ШІ дозволяє чат-боту виходити за межі заздалегідь визначених правил і шаблонних відповідей, забезпечуючи більш гнучку та людську взаємодію.

1.3.1 Обробка природної мови (NLP)

Обробка природної мови (Natural Language Processing – NLP) – це галузь штучного інтелекту, що зосереджується на взаємодії між комп'ютерами та людською мовою. Метою NLP є надання комп'ютерам здатності розуміти, інтерпретувати та генерувати людську мову у спосіб, який є осмисленим та корисним. У контексті чат-ботів, NLP є фундаментом для розуміння запитів користувача.

Основні завдання та компоненти NLP, що використовуються в чат-ботах:

1. **Токенізація (Tokenization):** Процес розбиття тексту на менші одиниці, такі як слова, фрази, символи або інші значущі елементи (токени). Це початковий етап підготовки тексту до подальшої обробки.
2. **Лемматизація та Стеммінг (Lemmatization and Stemming):**
   - **Стеммінг:** Відсікання закінчень слів для приведення їх до кореня (наприклад, "бігти", "біжить", "бігли" → "біг"). Це швидкий, але не завжди точний метод.
   - **Лемматизація:** Приведення слова до його базової (словникової) форми, враховуючи контекст та морфологічні правила (наприклад, "було", "буде" → "бути"). Лемматизація є точнішою, але більш ресурсомісткою.
3. **Визначення наміру (Intent Recognition/Classification):** Це процес ідентифікації мети або бажання користувача, вираженого в його запиті. Наприклад,

для фрази "Я хочу змінити свій пароль" наміром є "Змінити_пароль". Це ключовий крок для маршрутизації запиту до відповідної логіки чат-бота.

4. **Розпізнавання іменованих сутностей (Named Entity Recognition - NER):** Виявлення та класифікація іменованих сутностей у тексті, таких як імена людей, організації, місця, дати, числа, валюти тощо. Наприклад, у запиті "Забронюй квитки до Києва на 15 липня" "Київ" є сутністю "місто", а "15 липня" – сутністю "дата".

5. **Аналіз сентименту (Sentiment Analysis):** Визначення емоційного тону або ставлення, вираженого в тексті (позитивне, негативне, нейтральне). Це може бути корисно для чат-ботів підтримки клієнтів, щоб визначити рівень задоволеності користувача або пріоритезувати звернення.

6. **Контекстуальне розуміння (Contextual Understanding):** Здатність чат-бота зберігати та використовувати інформацію з попередніх повідомлень для підтримки зв'язного та осмисленого діалогу. Без контексту бот не зможе відповісти на запитання на кшталт "А що щодо цього?", якщо не пам'ятає, "що" було обговорено раніше.

7. **Генерація природної мови (Natural Language Generation - NLG):** Процес перетворення структурованих даних або внутрішніх представлень у зв'язний та граматично правильний текст. Цей компонент відповідає за формування відповідей чат-бота таким чином, щоб вони звучали природно та зрозуміло для людини.

Ефективність чат-бота значною мірою залежить від точності та глибини розуміння NLP.

1.3.2 Машинне навчання та глибоке навчання в контексті чат-ботів

Машинне навчання (Machine Learning - ML) – це підгалузь ШІ, яка дозволяє комп'ютерним системам "навчатися" на даних без явного програмування. Глибоке навчання (Deep Learning - DL) – це підмножина машинного навчання, що використовує нейронні мережі з багатьма прихованими шарами (глибокі нейронні мережі) для моделювання складних закономірностей. Ці технології є основою для створення інтелектуальних чат-ботів:

1. **Навчання з учителем (Supervised Learning):**

- **Принцип:** Модель навчається на розмічених даних, де кожному вхідному прикладу відповідає бажаний вихід.
- **Застосування в чат-ботах:**
  - **Класифікація намірів:** Модель навчається зіставляти вхідні фрази користувачів з попередньо визначеними намірами (наприклад, "Привіт" → "Привітання", "Зміни мій пароль" → "Зміна_пароля").
  - **Розпізнавання сутностей:** Модель навчається ідентифікувати та вилучати сутності з тексту, позначаючи їх (наприклад, "квитки до [місто]Києва[/місто]").
  - **Класифікація сентименту:** Навчання моделі для визначення емоційного тону повідомлення.

2. **Навчання без учителя (Unsupervised Learning):**
   - **Принцип:** Модель шукає приховані закономірності та структури в нерозмічених даних.
   - **Застосування в чат-ботах:**
     - **Кластеризація:** Групування схожих запитів користувачів, які можуть мати той самий намір, але виражені різними способами.
     - **Виявлення аномалій:** Ідентифікація незвичайних або непередбачених запитів, які не відповідають жодному відомому наміру.
3. **Глибоке навчання (Deep Learning):**
   - **Принцип:** Використання багатошарових нейронних мереж (наприклад, рекурентних нейронних мереж - RNN, довгої короткочасної пам'яті - LSTM, трансформерів) для обробки послідовностей даних, таких як текст.
   - **Застосування в чат-ботах:**
     - **Покращене розуміння мови:** Глибокі нейронні мережі дозволяють моделям краще розуміти складні синтаксичні та семантичні структури речень.
     - **Генерація природної мови:** DL-моделі здатні генерувати надзвичайно природний, зв'язний та контекстуально релевантний текст для відповідей чат-бота.

- **Ембедінги слів (Word Embeddings):** Перетворення слів у числові вектори, що відображають їхні семантичні зв'язки, дозволяючи моделям розуміти схожість між словами (наприклад, "автомобіль" і "машина").

4. **Навчання з підкріпленням (Reinforcement Learning):**
   - **Принцип:** Агент навчається у середовищі шляхом отримання винагород або покарань за свої дії.
   - **Застосування в чат-ботах:** Може бути використано для оптимізації діалогових стратегій, де бот навчається обирати найкращу наступну дію для досягнення цілі діалогу з користувачем (наприклад, отримати необхідну інформацію, вирішити проблему).

Інтеграція цих методів дозволяє чат-боту не просто відповідати на запитання, а розуміти їхній зміст, підтримувати контекст і навчатися на взаємодії, постійно покращуючи свою ефективність.

1.3.3 Використання мовних моделей (LLM) для розуміння та генерації відповідей

Великі мовні моделі (Large Language Models – LLM), такі як GPT (Generative Pre-trained Transformer) від OpenAI, стали революційним кроком у розвитку чат-ботів. Це моделі глибокого навчання, які навчені на величезних обсягах текстових даних з інтернету, що дозволяє їм розуміти, генерувати та перекладати людську мову з дивовижною точністю та зв'язністю.

Принципи роботи LLM у чат-ботах:

1. **Попереднє навчання (Pre-training):** LLM навчаються на мільярдах текстових документів (книги, статті, веб-сторінки, діалоги тощо) для передбачення наступного слова в послідовності. Це дозволяє їм вивчити граматику, синтаксис, семантику та величезну кількість фактів та стилів письма.

2. **Трансформерна архітектура:** Більшість сучасних LLM використовують архітектуру трансформера, яка ефективно обробляє послідовності даних, звертаючи увагу на різні частини вхідного тексту (механізм уваги – attention mechanism), що дозволяє їм розуміти складні залежності у реченнях.

3. **Тонке налаштування (Fine-tuning):** Після попереднього навчання, LLM можуть бути тонко налаштовані на менших, специфічних для завдання наборах даних. Це дозволяє адаптувати модель для конкретних застосувань, наприклад, для ведення діалогів у сфері підтримки користувачів або для генерації відповідей на основі бази даних.

4. **"Промпт-інжиніринг" (Prompt Engineering):** Цей підхід є ключовим для взаємодії з LLM. Замість явного програмування, користувач надає моделі "промпт" – інструкцію або початок тексту, який модель має продовжити або на основі якого має згенерувати відповідь. Ефективність відповіді LLM значною мірою залежить від якості промпту.

Використання LLM для розуміння запитів:

1. **Семантичне розуміння:** LLM здатні розуміти не лише ключові слова, а й повний сенс речення, включаючи нюанси та підтекст.

2. **Вилучення інформації:** Вони можуть вилучати складні сутності та атрибути з неструктурованих запитів.

3. **Розпізнавання наміру:** LLM можуть класифікувати наміри користувачів, навіть якщо вони сформульовані нетиповим чином, завдяки їхньому глибокому розумінню мови.

4. **Обробка неоднозначності:** Завдяки широкій базі знань, LLM можуть краще справлятися з неоднозначними запитами, намагаючись уточнити інформацію або надаючи найбільш вірогідну відповідь.

Використання LLM для генерації відповідей:

1. **Природність та зв'язність:** LLM генерують відповіді, які звучать дуже природно, граматично правильно та логічно зв'язані з попереднім контекстом.

2. **Генерація різноманітних відповідей:** Модель може генерувати різні варіанти відповідей на один і той самий запит, що робить діалог менш монотонним.

3. **Синтез інформації:** LLM можуть синтезувати інформацію з різних джерел (у тому числі з даних, отриманих з бази даних) та формулювати її у зрозумілій для користувача формі.

4. **Персоналізація відповідей:** На основі даних про користувача, отриманих з БД, LLM можуть генерувати персоналізовані відповіді та рекомендації.

Інтеграція LLM у чат-бота для роботи з базами даних дозволяє не просто виконувати команди, а "спілкуватися" з користувачем, розуміючи його запити у природній формі та надаючи інтелектуальні, контекстуально релевантні та людські відповіді, значно підвищуючи ефективність та зручність взаємодії.

1.4 Огляд існуючих рішень та аналіз аналогів

Перед початком проектування та розробки власного чат-бота для роботи з базами даних користувачів з інтегрованим штучним інтелектом, критично важливим є глибокий аналіз існуючих рішень та аналогів. Це дозволяє виявити поточні тренди, визначити найкращі практики, зрозуміти сильні та слабкі сторони конкурентних або схожих продуктів, а також уникнути повторного винаходу колеса.

1.4.1 Аналіз популярних платформ для розробки чат-ботів

Ринок платформ для розробки чат-ботів є дуже динамічним і пропонує широкий спектр інструментів, від простих конструкторів до потужних фреймворків для розробників. Їх можна класифікувати за рівнем складності та можливостями інтеграції ШІ.

1. **Платформи без кодування / з низьким рівнем кодування (No-Code / Low-Code Platforms):**
   - **Особливості:** Призначені для користувачів без глибоких навичок програмування. Пропонують візуальні інтерфейси (drag-and-drop), шаблони та готові інтеграції.
   - **Приклади:**
     - **ManyChat:** Популярний для Facebook Messenger, Instagram та WhatsApp. Ідеальний для маркетингу, продажів та підтримки клієнтів. Пропонує візуальний конструктор потоків, сегментацію аудиторії та автоматизацію.

- **Chatfuel:** Також орієнтований на месенджери, з акцентом на простоту створення. Має готові блоки для інтеграції, наприклад, з Shopify.
- **Botpress:** Відкритий вихідний код, але також надає візуальний інтерфейс. Дозволяє розробникам розширювати функціонал, інтегруючи власний код. Має вбудовані можливості NLU (розуміння природної мови).
- **Dialogflow (Google):** Хмарна платформа для розробки розмовних інтерфейсів. Надає потужні можливості NLU, дозволяючи легко визначати наміри та сутності. Може інтегруватися з різними каналами (веб, мобільні додатки, Google Assistant). Вимагає певних навичок для складних сценаріїв.

- **Переваги:** Швидке прототипування та розгортання, простота використання, низький поріг входу.
- **Недоліки:** Обмежена гнучкість для складних або нестандартних функцій, залежність від можливостей платформи, можлива прив'язка до конкретного вендора.

2. **Фреймворки та бібліотеки для розробників:**
   - **Особливості:** Надають максимальну гнучкість та контроль над розробкою. Вимагають значних навичок програмування. Дозволяють створювати повністю кастомізовані рішення.
   - **Приклади:**
     - **Rasa:** Потужний фреймворк з відкритим вихідним кодом для створення контекстуальних AI-асистентів. Дозволяє тренувати власні NLU-моделі, керувати складними діалогами та інтегруватися з будь-якими бекенд-системами. Вимагає значних зусиль для розробки, але надає повний контроль.
     - **Microsoft Bot Framework:** Набір інструментів та бібліотек для розробки, підключення та публікації розмовних інтерфейсів. Підтримує різні мови програмування та канали.

- **OpenAI API:** Хоча це не платформа для розробки чат-ботів у чистому вигляді, API OpenAI (GPT-3/4) є ключовим компонентом для створення інтелектуальних чат-ботів. Воно надає можливість для розуміння природної мови, генерації відповідей, узагальнення та інших складних лінгвістичних задач. Розробники інтегрують це API у власні бекенди, створюючи чат-ботів з високим рівнем інтелекту.
- **Hugging Face Transformers:** Бібліотека для роботи з трансформерними моделями (включаючи BERT, GPT-2/3, T5 та багато інших). Дозволяє розробникам використовувати попередньо навчені моделі для завдань NLP, таких як розуміння тексту, генерація відповідей, що є основою для інтелектуальних чат-ботів.

– **Переваги:** Повний контроль над функціоналом, висока гнучкість, масштабованість, можливість глибокої інтеграції з існуючими системами.
– **Недоліки:** Високий поріг входу, тривалий час розробки, необхідність у кваліфікованих розробниках.

Аналіз цих платформ дозволяє зробити висновок, що для створення чат-бота, який інтегрується з базами даних користувачів та використовує потужний AI, найбільш оптимальним є використання фреймворків (наприклад, Rasa) у поєднанні з API великих мовних моделей (OpenAI API). Це забезпечує необхідну гнучкість та інтелектуальні можливості.

1.4.2 Приклади чат-ботів, що інтегровані з базами даних

Інтеграція чат-бота з базами даних є поширеною практикою для вирішення практичних завдань у різних сферах. Наведені нижче приклади демонструють, як чат-боти взаємодіють з БД для надання інформації, виконання транзакцій та управління даними:

1. **Чат-боти банків та фінансових установ:**

– **Функціонал:** Дозволяють користувачам перевіряти баланс рахунку, переглядати історію транзакцій, отримувати виписки, здійснювати перекази коштів, оплачувати рахунки.

- **Інтеграція з БД:** Бот отримує запит користувача (наприклад, "Який мій баланс?"), ідентифікує його, звертається до банківської бази даних (через захищене API) для отримання інформації про рахунок користувача та формує відповідь. Для здійснення транзакцій бот також записує дані в БД.
- **Приклади:** Чат-боти Monobank, ПриватБанк (Україна), Bank of America Erica.

2. **Чат-боти для електронної комерції та ритейлу:**
   - **Функціонал:** Допомагають знаходити товари, перевіряти наявність, відстежувати замовлення, отримувати інформацію про доставку, повернення.
   - **Інтеграція з БД:** Бот взаємодіє з базою даних товарів (для пошуку та фільтрації), базою даних замовлень (для відстеження статусу), базою даних клієнтів (для персоналізованих рекомендацій).
   - **Приклади:** Чат-боти на сайтах інтернет-магазинів (наприклад, ASOS, Zara), які допомагають з навігацією по каталогу та інформацією про замовлення.
3. **Чат-боти служби підтримки та FAQ:**
   - **Функціонал:** Відповідають на типові запитання, вирішують поширені проблеми, направляють користувача до відповідного розділу довідки або до живого оператора.
   - **Інтеграція з БД:** Бот звертається до бази знань (яка може бути реалізована як реляційна або документо-орієнтована БД), де зберігаються відповіді на поширені запитання та інструкції.
   - **Приклади:** Чат-боти на сайтах телекомунікаційних компаній, постачальників послуг інтернету.
4. **Чат-боти для HR та внутрішніх корпоративних систем:**
   - **Функціонал:** Надають співробітникам інформацію про відпустки, зарплату, внутрішні політики, допомагають з оформленням документів, бронюванням переговорних кімнат.

- **Інтеграція з БД:** Бот взаємодіє з базою даних співробітників, системою управління відпустками, корпоративними календарями та іншими внутрішніми БД.
- **Приклади:** Корпоративні боти в Slack або Microsoft Teams для автоматизації рутинних HR-завдань.

5. **Чат-боти для медичних закладів та запису на прийом:**

- **Функціонал:** Дозволяють записуватися на прийом до лікаря, перевіряти графік роботи спеціалістів, отримувати нагадування.
- **Інтеграція з БД:** Бот звертається до бази даних розкладу лікарів та пацієнтів для перевірки доступності та запису.
- **Приклади:** Деякі клініки використовують ботів для автоматизації процесу запису.

Ці приклади демонструють, що інтеграція чат-ботів з базами даних є ключовим елементом для створення функціональних та корисних систем. Для майбутнього чат-бота, який працюватиме з базами даних користувачів, цей аналіз показує необхідність ретельного проектування структури БД та надійних механізмів взаємодії, а також використання можливостей AI для розуміння складних запитів користувачів та ефективного вилучення інформації.

В результаті, було розглянуто **загальні поняття віртуальних асистентів**, їхню еволюцію, класифікацію за принципом роботи (правилові, гібридні, на основі ШІ) та за призначенням (інформаційні, транзакційні, підтримка клієнтів, маркетингові, внутрішні). Це дозволило усвідомити роль та місце чат-ботів у сучасних інформаційних системах та визначити ключові вимоги до ефективної взаємодії, такі як розуміння природної мови, підтримка контексту, точність відповідей та гнучкість реакції.

Особливу увагу приділено **базам даних та системам управління базами даних (СУБД)**. Було проаналізовано принципи організації реляційних баз даних, що є критично важливим для зберігання структурованої інформації користувачів та товарів. Досліджено особливості роботи з SQLite як легкою, вбудованою СУБД, що ідеально підходить для невеликих та середніх проектів завдяки простоті

розгортання та надійності, а також розглянуто її переваги та недоліки в контексті потенційного масштабування на PostgreSQL.

Детально вивчено роль **штучного інтелекту у реалізації чат-ботів**. Розглянуто основні концепції обробки природної мови (NLP) та машинне навчання, які є фундаментом для створення інтелектуальних діалогових систем. Акцентовано увагу на важливості великих мовних моделей (LLM), зокрема OpenAI API (GPT-4o), як ключового інструменту для розуміння складних запитів, генерації природних відповідей та реалізації механізму function calling для взаємодії з зовнішніми системами, включаючи бази даних.

На завершення розділу проведено **огляд існуючих рішень та аналіз аналогів**. Це дозволило виявити поточні тенденції у розробці чат-ботів, оцінити їхні переваги та недоліки, а також обґрунтувати вибір стеку технологій (Python, Django, SQLite, OpenAI API) для реалізації майбутнього проекту. Вибір гібридного пайплайну обробки запитів, що поєднує локальну логіку з можливостями LLM, визначено як оптимальний для досягнення поставленої мети.

Таким чином, перший розділ створив міцну теоретичну базу, яка є фундаментом для подальшого проектування та практичної реалізації веб-застосунку «E-Shop + AI Chat-bot», забезпечуючи розуміння всіх компонентів та їх взаємозв'язків у розробленій системі.

2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ЧАТ-БОТА ДЛЯ РОБОТИ З БАЗАМИ ДАНИХ КОРИСТУВАЧІВ

2.1 Вибір технологій та інструментів розробки

Успішна реалізація чат-бота для роботи з базами даних користувачів, інтегрованого зі штучним інтелектом, значною мірою залежить від ретельного вибору технологій та інструментів розробки. Цей вибір має ґрунтуватися на таких критеріях, як функціональні та нефункціональні вимоги до системи, наявність досвіду розробника, спільноти підтримки, масштабованість, продуктивність, безпека та економічна доцільність.

2.1.1 Мови програмування та фреймворки

Для розробки чат-бота та його бекенд-логіки доцільно обрати мови програмування, які є популярними у сфері веб-розробки та штучного інтелекту, а також мають розвинену екосистему фреймворків та бібліотек.

Розглядаючи технологічний стек для розробки чат-бота, першочерговим вибором став **Python**. Його обґрунтованість полягає у визнаній популярності для розробки застосунків зі штучним інтелектом, машинним навчанням та обробкою природної мови, підкріпленій великою кількістю бібліотек, таких як TensorFlow, PyTorch, scikit-learn, NLTK та spaCy, які значно спрощують роботу з AI. Окрім того, Python чудово підходить для бекенд-розробки веб-додатків завдяки надійним фреймворкам. Серед них виділяються **Django** та **Flask** для бекенду, які забезпечують міцну основу для побудови API, управління даними та реалізації бізнес-логіки. Flask, будучи легшим та гнучкішим, ідеально підходить для швидкої розробки та менших проєктів, тоді як Django пропонує повноцінний набір інструментів для масштабних додатків. В контексті даного проєкту, що вимагає взаємодії з базою даних та API, Flask або Flask-RESTful можуть бути більш підходящими для швидкого прототипування або розробки API. Для створення функціоналу Telegram-бота існують спеціалізовані бібліотеки, такі як python-telegram-bot або aiogram, які надають зручні інструменти для обробки повідомлень, команд та кнопок. Перевагами Python є простота синтаксису, велика спільнота

розробників, величезна кількість готових бібліотек для AI/ML та крос-платформність. Єдиним відносним недоліком є дещо менша продуктивність порівняно з компільованими мовами, хоча для більшості веб-додатків це не є критичним фактором.

Як **альтернативний варіант** розглядалася мова **JavaScript / Node.js**. JavaScript є універсальною мовою, яка використовується як для фронтенду у веб-браузерах, так і для бекенду завдяки середовищу Node.js. Серед фреймворків для бекенду виділяється Express.js – мінімалістичний та гнучкий фреймворк для створення RESTful API, а для роботи з Telegram Bot API доступні бібліотеки Telegraf.js або node-telegram-bot-api. Переваги JavaScript полягають у можливості використання єдиної мови для повного стеку розробки (Full-stack JavaScript), великій спільноті, високій продуктивності для I/O-операцій та придатність для додатків реального часу. Однак, порівняно з Python, JavaScript має меншу кількість спеціалізованих бібліотек для ML/NLP, а також може виникати складність в управлінні залежностями у масштабних проєктах.

Таким чином, для бекенду та реалізації логіки чат-бота, а також для інтеграції зі штучним інтелектом, Python у поєднанні з фреймворком Flask та бібліотекою для Telegram Bot API (наприклад, python-telegram-bot) є оптимальним вибором. Це забезпечить ефективну розробку, легку інтеграцію з інструментами ML/NLP та достатню продуктивність для досягнення цілей проєкту.

2.1.2 Платформа для розгортання чат-бота

Вибір платформи для розгортання, або хостингу, чат-бота є вирішальним для забезпечення його доступності, масштабованості та надійності, особливо коли йдеться про інтеграцію з базою даних. Для такого чат-бота необхідний надійний хостинг, який повноцінно підтримує всі обрані технології.

Серед варіантів хостингу виділяються **хмарні платформи**. Вони пропонують гнучку та масштабовану інфраструктуру, що дозволяє швидко розгортати додатки без необхідності керування фізичними серверами. Прикладами таких платформ є Heroku, яка є Platform as a Service (PaaS) і є дуже зручною для швидкого розгортання Python/Node.js додатків, пропонуючи просту інтеграцію з базами

даних та гнучке масштабування, що робить її ідеальною для MVP та невеликих/середніх проєктів. Інші гіганти у цій сфері включають AWS (Amazon Web Services) з її широким спектром сервісів, таких як EC2 для віртуальних серверів, Lambda для безсерверних функцій та RDS для баз даних, що надає максимальну гнучкість і масштабованість, хоча й вимагає більшого досвіду для налаштування. Google Cloud Platform (GCP) аналогічно AWS пропонує різноманітні сервіси, такі як Compute Engine, Cloud Functions та Cloud SQL, з додатковою зручною інтеграцією з інструментами Google AI. Microsoft Azure також є повнофункціональною хмарною платформою. Переваги хмарних платформ полягають у високій доступності, масштабованості, відсутності необхідності керування інфраструктурою та широкому наборі додаткових сервісів. Однак, їхні недоліки включають потенційне зростання вартості зі збільшенням навантаження та складність налаштування для початківців, особливо для таких комплексних систем як AWS, GCP чи Azure.

Другим значущим варіантом є **віртуальний приватний сервер (VPS)**. VPS надає більший контроль над середовищем порівняно з PaaS, але водночас вимагає самостійного налаштування операційної системи та всього необхідного програмного забезпечення. Прикладами таких провайдерів є DigitalOcean Droplets, Linode та Vultr. Перевагами VPS є повний контроль над сервером, часто нижча вартість для фіксованого обсягу ресурсів та можливість налаштувати будь-яке оточення.

Для початкового розгортання та тестування, а також враховуючи простоту використання, Heroku є відмінним вибором. Вона дозволяє швидко розгорнути Python-додаток і легко підключити базу даних, зокрема SQLite для простих випадків. У разі подальшого масштабування проєкту, можливий перехід на більш потужні хмарні рішення, такі як AWS або GCP.

2.1.3 Вибір СУБД та її обґрунтування

Для зберігання даних користувачів (наприклад, їхні профілі, історія взаємодії з ботом, налаштування) та інших структурованих даних, необхідних для роботи чат-бота, потрібна надійна система управління базами даних. Аналізуючи системи

управління базами даних для проєкту, розглянули як реляційні, так і документо-орієнтовані рішення.

SQLite, як реляційна СУБД, була обрана завдяки своїй легкості, вбудованості та файловій організації, що виключає потребу в окремому серверному процесі. Цей вибір ідеально підходить для невеликих та середніх проєктів, де пріоритетом є простота розгортання, мінімальні вимоги до системних ресурсів та відсутність необхідності у складній конфігурації. SQLite гарантує високу цілісність даних завдяки підтримці транзакцій, демонструючи при цьому значну надійність. Ці характеристики роблять її відмінним варіантом для дипломної роботи або етапу прототипування, дозволяючи зосередитися на основній функціональності чат-бота та інтеграції штучного інтелекту, а не на питаннях масштабованості та обробки великої кількості паралельних запитів. Серед її переваг варто відзначити надзвичайну простоту у використанні та розгортанні (один файл), відсутність вимог до конфігурації сервера, малу вагу, підтримку більшості стандартних функцій SQL та високу надійність даних завдяки ACID-комплаєнтності. Проте, SQLite має і свої недоліки: вона не підходить для багатокористувацьких додатків з високою конкурентністю записів, оскільки блокує всю базу при записі, не призначена для обробки терабайтів даних через потенційне зниження продуктивності та не має мережевих можливостей, оскільки база даних розташована локально з додатком.

Як альтернативний варіант розглядалася **MongoDB**, документо-орієнтована NoSQL СУБД. Вона могла б бути доцільною у випадку, якщо б дані користувачів були надзвичайно гнучкими або напівструктурованими, наприклад, для зберігання змінних налаштувань чи логів чату. Переваги MongoDB включають гнучку схему, високу масштабованість та хорошу продуктивність для значних обсягів даних. Однак, її недоліком є менша цілісність даних (за замовчуванням вона не є ACID-комплаєнтною), що може ускладнити роботу зі складними зв'язками між даними.

Для цілей даного проєкту, де головний акцент робиться на функціоналі чат-бота та інтеграції штучного інтелекту, а не на високопродуктивній масштабованості для мільйонів користувачів, SQLite є найбільш підходящим

вибором. Її простота використання та легкість інтеграції з Python-додатком значно спрощують процес розробки та дозволяють повністю зосередитися на ключових аспектах проєкту.

2.1.4 Інструменти для інтеграції з API мовними моделями

Центральним елементом інтеграції штучного інтелекту в чат-бот є використання потужних мовних моделей, що надаються через API. При виборі інструментів для інтеграції штучного інтелекту в систему, **OpenAI API**, що включає такі потужні моделі, як GPT-3, GPT-4 та GPT-4o, виявився провідним рішенням для генерації та розуміння природної мови. Його обґрунтованість полягає у винятковій якості генерації тексту, здатності глибоко розуміти контекст, точно вилучати сутності, класифікувати наміри користувачів та ефективно відповідати на складні запитання. Це дозволяє чат-боту підтримувати змістовні діалоги та інтерпретувати запити користувачів, навіть якщо вони сформульовані нестандартно. Перевагами цього вибору є висока якість розуміння мови та її генерації, гнучкість у використанні завдяки різноманіттю моделей і параметрів, активна спільнота розробників та безперервний розвиток технології. Однак, незважаючи на численні переваги, існують і недоліки: це комерційний сервіс, що вимагає оплати, присутня залежність від зовнішнього сервісу (що впливає на доступність та затримки), а також виникають питання щодо конфіденційності даних, хоча OpenAI і має чіткі політики для їх захисту.

Крім основного вибору, існують **альтернативні або додаткові інструменти**. Наприклад, **Hugging Face Transformers** може бути використаний для розробки або тонкого налаштування власних ML-моделей під дуже специфічні завдання, такі як унікальні класифікації намірів або розпізнавання іменованих сутностей, хоча це і вимагає більшого досвіду у машинному навчанні та значних обчислювальних ресурсів. Для базової обробки природної мови, як-от токенізація, лемматизація чи POS-тегінг, можна застосовувати бібліотеки **spaCy / NLTK**. Ці інструменти можуть бути корисними для попередньої обробки тексту перед його передачею до великих мовних моделей або для реалізації простіших сценаріїв. Фреймворки, такі як **LangChain / LlamaIndex**, значно допомагають організувати та структурувати

взаємодію з LLM, дозволяючи створювати складніші "ланцюжки" для роботи з даними, що надходять з бази даних, та їх подальшої обробки мовною моделлю, спрощуючи таким чином інтеграцію LLM з іншими джерелами даних.

Для даного проєкту, основним інструментом для інтеграції штучного інтелекту буде OpenAI API, що забезпечить ефективне розуміння запитів користувачів та генерацію адекватних відповідей. Додатково, використання LangChain може бути розглянуте для оптимізації взаємодії між чат-ботом, базою даних та великою мовною моделлю.

Підсумовуючи вибір стеку технологій, для розробки чат-бота, що взаємодіє з базами даних користувачів та має інтегрований штучний інтелект, буде використано мову програмування Python. Серед фреймворків обрано Flask для бекенду та python-telegram-bot для взаємодії з Telegram Bot API. Розгортання та тестування системи планується здійснювати на платформі Heroku. Як система управління базами даних використовуватиметься SQLite, а ключові AI-функції реалізуватимуться за допомогою OpenAI API. Цей стек технологій забезпечить необхідну функціональність, простоту розробки та розгортання, дозволяючи ефективно реалізувати поставлені завдання дипломної роботи з акцентом на логіці та інтеграції ШІ.

2.2 Проектування структури бази даних користувачів

Ефективне проектування структури бази даних є фундаментальним етапом у розробці будь-якої інформаційної системи, зокрема чат-бота, який взаємодіє з даними користувачів. Добре спроектована база даних забезпечує цілісність, консистентність, безпеку даних, а також ефективне зберігання та доступ до інформації. Для даної дипломної роботи, де використовується **SQLite** як СУБД, проектування буде зосереджено на оптимальному зберіганні даних користувачів та їхньої взаємодії з чат-ботом.

2.2.1 Схема бази даних (таблиці, поля, зв'язки)

З огляду на функціональні вимоги до чат-бота, що працює з базами даних користувачів, пропонується наступна схема реляційної бази даних, що складається

з кількох взаємопов'язаних таблиць (рис. 2.1). Основний фокус буде на даних, необхідних для ідентифікації користувача, його взаємодії з ботом та можливих додаткових атрибутів, які можуть бути керовані чат-ботом.

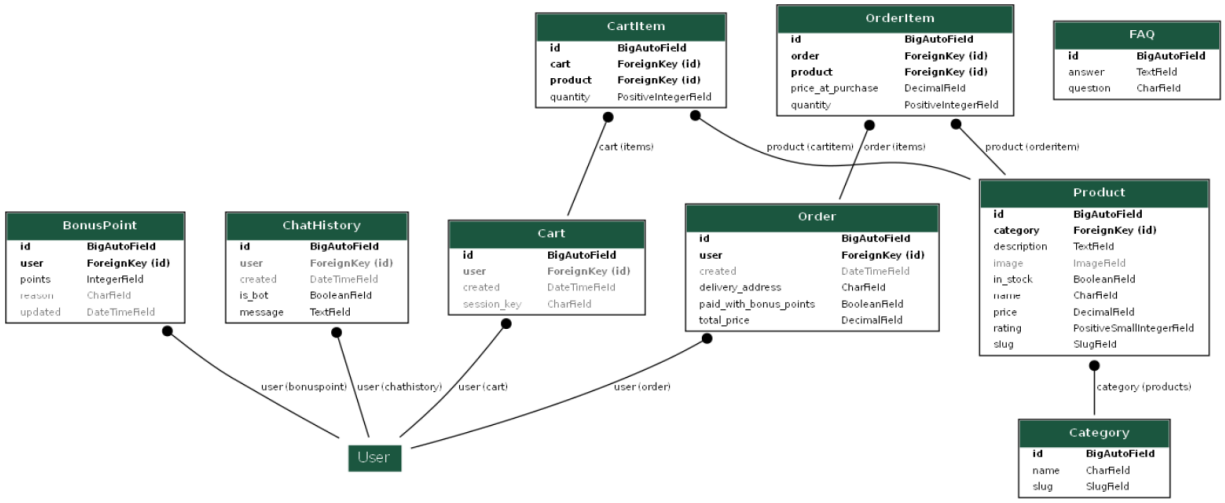


Рисунок 2.1 - Візуалізація ER-діаграми спроектованої бази даних

На схемі наведено базу даних для системи електронної комерції з інтегрованим чат-ботом. Наведена Entity-Relationship Diagram (ERD) яка демонструє взаємозв'язки між різними таблицями (сутностями). Опишемо основні таблиці та їх зв'язки:

- **User:** Ця таблиця, ймовірно, зберігає інформацію про користувачів системи. Вона є центральною і пов'язана з багатьма іншими таблицями.
- **BonusPoint:** Містить інформацію про бонусні бали користувачів. Має поле user як зовнішній ключ, що вказує на конкретного користувача, а також points (ціле число), reason (текстове поле) та updated (дата/час).
- **ChatHistory:** Зберігає історію взаємодії чат-бота з користувачем. Включає user (зовнішній ключ до таблиці User), created (дата/час створення запису), is_bot (булеве поле, що вказує, чи це повідомлення від бота) та message (текст повідомлення).
- **Cart:** Представляє кошик покупок користувача. Має зовнішній ключ user, created (дата/час створення кошика) та session_key (символьне поле, ймовірно, для анонімних сесій).

- **CartItem:** Деталізує товари в кошику. Містить зовнішні ключі cart (посилання на кошик) та product (посилання на товар), а також quantity (ціле число, кількість товару).
- **Order:** Зберігає інформацію про оформлені замовлення. Має зовнішній ключ user, created (дата/час створення замовлення), delivery_address (текстове поле), paid_with_bonus_points (булеве поле), total_price (десяткове число).
- **OrderItem:** Деталізує товари у замовленні. Містить зовнішні ключі order (посилання на замовлення) та product (посилання на товар), price_at_purchase (десяткове число, ціна товару на момент покупки) та quantity (ціле число).
- **FAQ:** Зберігає питання та відповіді для секції частих запитань. Має поля answer та question (обидва текстові).
- **Product:** Містить інформацію про товари, доступні в магазині. Включає зовнішній ключ category (посилання на категорію товару), description (текстове поле), image (зображення), in_stock (булеве поле), name (символьне поле), price (десяткове число), rating (невелике ціле число), slug (символьне поле).
- **Category:** Визначає категорії товарів. Містить name (символьне поле) та slug (символьне поле).

Реалізовані зв’язки між таблицями:

- User може мати багато BonusPoint записів.
- User може мати багато записів ChatHistory.
- User може мати один Cart.
- User може мати багато Order.
- Cart може містити багато CartItem.
- CartItem пов'язаний з одним Product.
- Order може містити багато OrderItem.
- OrderItem пов'язаний з одним Product.
- Product належить до однієї Category.

Дана схема бази даних дозволяє ефективно керувати даними, пов'язаними з користувачами, товарами, кошиками, замовленнями та інтеграцією чат-бота.

2.2.2 Обґрунтування вибору даних для зберігання

Вибір конкретних полів та таблиць базується на необхідності забезпечити функціональність чат-бота, його інтеграцію з ШІ та потенційну розширюваність, враховуючи специфіку роботи з базами даних користувачів.

Схема бази даних передбачає зберігання інформації про користувачів та історію їхніх взаємодій з чат-ботом.

Основна інформація про користувача зберігається у таблиці users. Кожен користувач має єдиний унікальний ідентифікатор id (або user_id), який надається платформою месенджера, наприклад Telegram, і є ключовим для зв'язування всіх дій та даних користувача, оптимально використовуючи цілочисельний тип для SQLite. Поля username, first_name та last_name дозволяють персоналізувати спілкування, звертаючись до користувача по імені, при цьому username може бути необов'язковим. Дати реєстрації (registration_date) та останньої активності (last_activity_date) є важливими для аналітики поведінки користувачів та їхньої сегментації. Поле status дає можливість адміністративно керувати доступом користувачів до бота, наприклад, тимчасово блокувати їх. Особливу гнучкість системі надає поле custom_data_json, що дозволяє зберігати довільні, неструктуровані дані у форматі JSON, такі як улюблені теми або налаштування сповіщень, без необхідності зміни схеми бази даних, що є значною перевагою, особливо при роботі з SQLite.

Історія взаємодій фіксується у таблиці interactions. Тут id та user_id забезпечують унікальну ідентифікацію кожної взаємодії та її прив'язку до конкретного користувача, разом з timestamp для фіксації часу. Поля user_message та bot_response містять повний текст обміну повідомленнями, що є критично важливим для відлагодження, покращення AI-моделей (слугуючи даними для тонкого налаштування або навчання NLP/LLM моделей) та аналітики популярних запитів і ефективності відповідей. Зберігання визначеного наміру (intent) після обробки NLP-модулем допомагає оцінити точність розуміння користувача та виявити нерозпізнані наміри. Поля success та error_message використовуються для відстеження якості та надійності роботи чат-бота, надаючи деталі для

налагодження у випадку невдалих інтеграцій з базою даних, що є критично важливим для забезпечення безперебійної роботи системи.

Вибір SQLite для цього проекту обґрунтований його легкістю, простотою розгортання та відсутністю потреби у зовнішньому сервері БД, що ідеально підходить для дипломної роботи та демонстрації концепції. Проектування схеми з урахуванням гнучкості (наприклад, custom_data_json) мінімізує необхідність зміни схеми БД у майбутньому, що є перевагою для файлової бази даних.

2.3 Розробка архітектури чат-бота

Розробка архітектури чат-бота для роботи з базами даних користувачів з інтегрованим штучним інтелектом вимагає чіткого визначення всіх компонентів системи та їх взаємодії. З огляду на використання фреймворку Django та надані моделі бази даних (Category, Product, Cart, Order, BonusPoint, FAQ, ChatHistory), архітектура буде включати шари для взаємодії з месенджером, обробки природної мови, бізнес-логіки (взаємодії з БД) та інтеграції з великими мовними моделями.

2.3.1 Схематичне представлення компонентів системи та їх взаємодії

Запропонована архітектура чат-бота базується на мікросервісному або модульному підході, що дозволяє розділити відповідальність між різними компонентами та забезпечити гнучкість і масштабованість.

Розглядаючи основні компоненти системи чат-бота, слід виділити їхню взаємодію та функції (рис. 2.2).

Канал взаємодії, який у нашому випадку реалізовано через Telegram Bot API, слугує точкою входу та виходу для всіх повідомлень користувачів, забезпечуючи зв'язок чат-бота з месенджером за допомогою бібліотек python-telegram-bot або aiogram у Python.

Далі йде **Вебхук / Polling Сервер**, який є Django-додатком. Він приймає вхідні повідомлення від Telegram Bot API, як правило, через вебхуки, або ж періодично опитує його. Цей компонент діє як шлюз між месенджером та внутрішньою логікою бота, обробляючи запити від Telegram за допомогою Django View або окремого процесу.

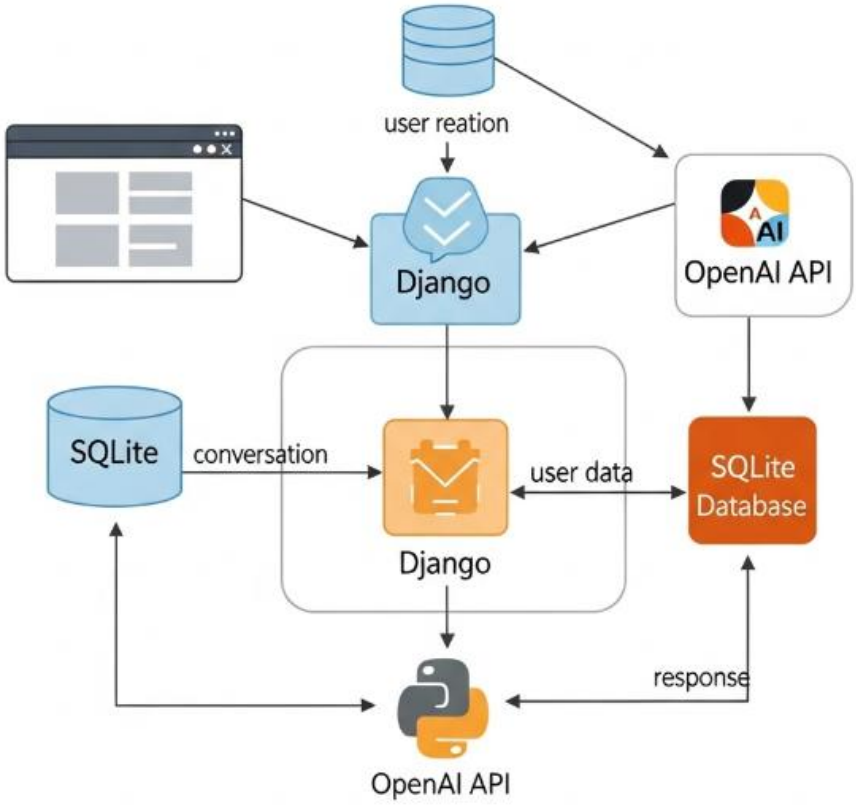


Рисунок 2.2 - Схематичне представлення системи з чат-ботом

Наступним елементом є **Диспетчер Повідомлень**. Його призначення полягає в отриманні та маршрутизації оброблених вхідних повідомлень від Вебхук/Polling Сервера до відповідних обробників, таких як NLP-модуль чи модуль управління діалогом. Диспетчер також відповідає за початкову фільтрацію та підготовку даних, аналізуючи тип повідомлення (текст, команда, callback-дані) і викликаючи відповідні функції у Django views.py або окремому модулі.

Модуль Обробки Природної Мови (NLP Module) відповідає за аналіз та інтерпретацію тексту повідомлень користувачів для виявлення їхніх намірів та вилучення сутностей. Його ключовою реалізацією є інтеграція з OpenAI API, куди надсилаються запити користувача для отримання назад намірів, сутностей або згенерованого тексту. Для попередньої обробки тексту або додаткової валідації можуть бути використані легкі NLP-бібліотеки, такі як SpaCy або NLTK, хоча більшість функцій виконує велика мовна модель.

Модуль Управління Діалогом є центральним для керування потоком розмови. Він зберігає контекст, визначає наступний крок на основі наміру

користувача та поточного стану діалогу. Реалізується він як набір функцій або класів у Django-додатку (наприклад, у services.py або dialogue_manager.py), що визначає, яку бізнес-логіку викликати, зберігає стан діалогу (у сесіях Telegram або в базі даних), формує запити до Модуля Взаємодії з Базою Даних і за потреби уточнює інформацію у користувача.

Модуль Взаємодії з Базою Даних безпосередньо взаємодіє з базою даних через Django ORM. Його призначення – виконання CRUD-операцій (створення, читання, оновлення, видалення) з даними користувачів, товарів, замовлень, кошиків тощо. Реалізація відбувається за допомогою Django ORM, що дозволяє працювати з моделями User, Category, Product, Cart, CartItem, Order, OrderItem, BonusPoint, FAQ, ChatHistory, використовуючи об'єктно-реляційне відображення Django.

Модуль Генерації Відповідей відповідає за формування фінальної відповіді для користувача. Він інтегрується з OpenAI API для генерації природних, контекстуально релевантних відповідей, що базуються на отриманих даних з бази даних або складній логіці. Також можуть використовуватись шаблонні відповіді для типових сценаріїв або динамічні відповіді, що формуються з використанням даних, отриманих з бази даних.

База Даних (SQLite) служить для зберігання всіх даних системи: інформації про користувачів, товари, категорії, замовлення, кошики, бонуси, часті питання та історію чату. Реалізація здійснюється за допомогою файлу db.sqlite3, що керований Django ORM.

2.3.2 Опис логіки роботи чат-бота з базою даних

Логіка роботи чат-бота з базою даних користувачів та іншими сутностями, такими як товари та замовлення, тісно пов'язана з обробкою намірів користувача і реалізована за допомогою Django ORM.

Процес починається з ініціації та реєстрації користувача. Коли новий користувач вперше взаємодіє з ботом, Модуль Управління Діалогом перевіряє його наявність у таблиці users за ідентифікатором Telegram. Якщо користувач не знайдений, створюється новий запис у таблиці users з його ідентифікатором, ім'ям,

прізвищем, іменем користувача та датою реєстрації. Одночасно створюється новий запис у таблиці ChatHistory для першого повідомлення, фіксуючи початок взаємодії.

Надалі відбувається обробка запитів, що потребують даних з БД (SELECT-операції). Наприклад, для запитів щодо інформації про товар ("Покажи мені кросівки", "Скільки коштує товар X?") NLP-модуль визначає намір "Переглянути_товари" або "Дізнатися_ціну" та вилучає сутності (категорія, назва товару). Модуль Управління Діалогом формує запит до Модуля Взаємодії з БД, який через Django ORM виконує запит до таблиць Product та Category (наприклад, Product.objects.filter(category__name='Кросівки')). Отримані дані (назва, опис, ціна, наявність) передаються до Модуля Генерації Відповідей, який формує інформативну відповідь. Подібним чином, при запитах про стан кошика ("Що в моєму кошику?") NLP-модуль визначає намір "Переглянути_кошик", Модуль Управління Діалогом запитує поточний кошик користувача з таблиць Cart та CartItem, після чого формується відповідь зі списком товарів та сумарною вартістю. Аналогічна логіка застосовується для перевірки замовлень, запитів про бонусні бали та відповідей на часті питання (FAQ), де NLP-модуль ідентифікує намір, а Модуль Управління Діалогом звертається до відповідних таблиць (Order, OrderItem, BonusPoint, FAQ) для отримання необхідної інформації. Якщо відповіді на FAQ немає, бот може запропонувати перенаправлення до оператора або використовувати LLM для генерації відповіді.

Також система здійснює обробку запитів, що модифікують дані в БД (INSERT, UPDATE, DELETE-операції)**.** При спробі додати товар до кошика ("Додай до кошика кросівки Nike") NLP-модуль визначає намір "Додати_до_кошика" та вилучає назву товару/кількість. Модуль Управління Діалогом перевіряє наявність товару та його статус in_stock, після чого створюється або оновлюється запис у Cart та CartItem (наприклад, Cart.objects.get_or_create(...)). Бот підтверджує додавання та може запропонувати перейти до оформлення замовлення. При оформленні замовлення ("Оформити замовлення") NLP-модуль визначає намір "Оформити_замовлення", Модуль Управління Діалогом перевіряє

вміст кошика, запитує необхідні дані для доставки, створює новий запис у таблиці Order та відповідні записи в OrderItem, оновлює статус in_stock для товарів та очищує кошик. Бот підтверджує замовлення та надає його номер, а також може оновити BonusPoint, якщо бонуси були використані. Теоретично, чат-бот також може обробляти запити на зміну даних користувача, коли NLP-модуль визначає намір "Оновити_профіль" та Модуль Управління Діалогом оновлює відповідне поле в таблиці users, після чого бот підтверджує зміну.

Важливою складовою є зберігання історії діалогів. Кожне повідомлення від користувача та відповідь від бота автоматично записується в таблицю ChatHistory. Це забезпечує повний лог взаємодії, що є цінним для аналітики, відлагодження, аудиту та подальшого навчання моделей.

Основою інтелектуальної складової є інтеграція AI з БД**.** OpenAI API (велика мовна модель) інтерпретує вхідні повідомлення користувача, перетворюючи їх на структуровані наміри та сутності. Наприклад, запит "Знайди телефон Samsung Galaxy" буде розпізнано як намір "Пошук_продукту" з відповідними сутностями, що потім використовується для запиту до Product.objects.filter(name__icontains='Samsung Galaxy'). Після отримання даних з бази даних, велика мовна модель може бути використана для формування природної, розгорнутої відповіді, що включає ці дані, перетворюючи прості відомості на змістовне повідомлення, наприклад, "Ваш поточний баланс становить 150 гривень. Чи бажаєте ви поповнити рахунок?".

Вся ця логіка буде реалізована в рамках Django-додатку, використовуючи переваги Django ORM для абстракції роботи з базою даних SQLite, що дозволить розробнику зосередитися на бізнес-логіці та інтеграції з ШІ.

В результаті, було здійснено ключові етапи проектування системи, що є фундаментальним для подальшої практичної реалізації чат-бота. Це включало обґрунтований вибір технологічного стеку, розробку деталізованої структури бази даних та проектування архітектури самого чат-бота.

На етапі вибору технологій та інструментів розробки було аргументовано доцільність використання Python як основної мови програмування, що обумовлено

її універсальністю та наявністю розгалуженої екосистеми бібліотек для розробки веб-додатків та інтеграції штучного інтелекту. Серед веб-фреймворків було обрано Django, який є потужним і зрілим рішенням для побудови складних веб-додатків з підтримкою ORM, REST API та готовими компонентами для керування користувачами та сесіями. В якості системи управління базами даних для етапу розробки та тестування було обрано SQLite, що забезпечує простоту розгортання та адміністрування, а також достатню функціональність для дипломного проєкту. Для інтеграції штучного інтелекту було обрано OpenAI API (модель GPT-4o), яка гарантує високу якість розуміння природної мови та генерації відповідей, а також можливість використання function calling для взаємодії з базою даних.

Проектування структури бази даних користувачів є одним з центральних етапів розділу. Була розроблена реляційна схема, яка охоплює всі необхідні сутності для функціонування e-commerce системи та чат-бота, включаючи таблиці для користувачів, товарів, категорій, кошиків, замовлень, бонусних балів та історії чату. Детальний опис полів та взаємозв'язків між таблицями (User, Product, Category, Cart, CartItem, Order, OrderItem, BonusPoint, ChatHistory, FAQ) забезпечує цілісність даних, ефективне зберігання та можливість швидкого доступу до необхідної інформації для чат-бота. Вибір SQLite на цьому етапі дозволив швидко імплементувати та протестувати розроблену схему.

Кульмінацією розділу стала розробка архітектури чат-бота. Була запропонована модульна архітектура, яка чітко розмежовує компоненти системи: користувацький інтерфейс на сайті, бекенд на Django, модуль обробки природної мови на базі OpenAI API, а також модуль управління діалогом. Візуальне представлення архітектури демонструє потоки даних та взаємодію між цими блоками, підкреслюючи роль Django як центрального елемента, що координує роботу між фронтендом, базою даних SQLite та зовнішнім сервісом OpenAI.

Таким чином, у другому розділі було виконано всебічне та обґрунтоване проектування всіх ключових компонентів системи чат-бота, що забезпечує міцну основу для його успішної реалізації та впровадження.

3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЧАТ-БОТА ДЛЯ РОБОТИ З БАЗАМИ ДАНИХ КОРИСТУВАЧІВ

3.1 Розробка модуля взаємодії з користувачем

Модуль взаємодії з користувачем є фронтендом чат-бота, який забезпечує прямий контакт з кінцевим користувачем. Його розробка включає імплементацію користувацького інтерфейсу, через який користувачі можуть надсилати запити та отримувати відповіді, а також реалізацію механізмів для обробки цих вхідних запитів перед їх передачею до основної логіки чат-бота. Для даної дипломної роботи, де чат-бот функціонує як частина веб-сайту на Django з базою даних SQLite, фокус буде на веб-інтерфейсі.

3.1.1 Імплементація інтерфейсу чат-бота (веб-інтерфейс)

Імплементація інтерфейсу чат-бота у вигляді веб-інтерфейсу передбачає створення спеціальної сторінки на сайті, де користувач може взаємодіяти з ботом. Це забезпечує зручність використання без необхідності встановлення сторонніх додатків та легку інтеграцію з іншими елементами сайту.

Розробка фронтенду веб-інтерфейсу для чат-бота передбачає використання стандартних веб-технологій. Для структури сторінки та розмітки елементів чату, таких як вікно для повідомлень, поле вводу та кнопка відправки, застосовується HTML5. Стилізація інтерфейсу, його привабливий вигляд та адаптивність для різних пристроїв, включаючи оформлення бульбашок повідомлень, області прокрутки та анімації, забезпечується за допомогою CSS3. JavaScript використовується для динамічної поведінки інтерфейсу, дозволяючи відправляти повідомлення користувача на бекенд (Django-додаток) без перезавантаження сторінки через AJAX/Fetch API, динамічно додавати нові повідомлення у вікно чату, автоматично прокручувати його до останнього повідомлення, обробляти події клавіатури (наприклад, відправка повідомлення по натисканню Enter) та відображати індикатори набору тексту або завантаження для покращення користувацького досвіду.

Структура інтерфейсу (рис. 3.1) зазвичай включає основний div-елемент, що є контейнером чату, вікно повідомлень – область для відображення історії діалогу, де кожне повідомлення представлено окремим div або li елементом зі стилізацією для повідомлень користувача та бота, а також поле вводу (textarea або input) та кнопку відправки повідомлення.

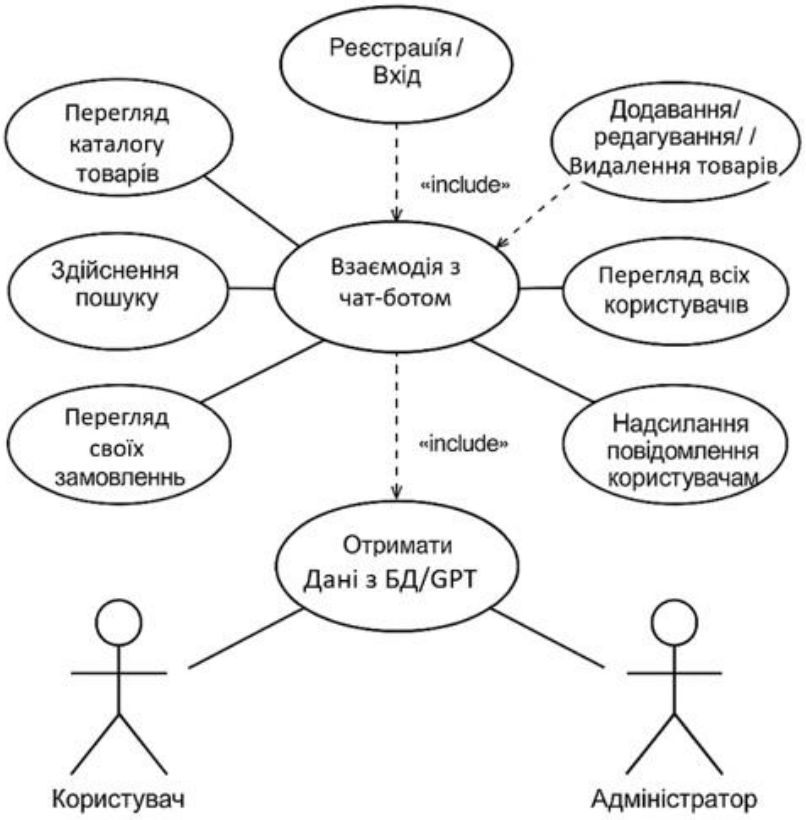


Рисунок 3.1 - Діаграма прецендентів взаємодії користувача

Взаємодія фронтенду з бекендом на Django реалізується через RESTful API для обміну даними між JavaScript-інтерфейсом та Django-додатком. Зокрема, використовується метод POST на api/chat/message/ для відправки JSON-об'єкта з повідомленням користувача, а опціонально метод GET на api/chat/history/ може застосовуватися для завантаження попередньої історії діалогу при ініціалізації чату.

На стороні Django відповідні View-функції або ViewSet (якщо використовується Django REST Framework) обробляють ці API-запити. Ці функції відповідають за прийом вхідних JSON-даних, аутентифікацію та авторизацію користувача (за потреби), виклик основної логіки чат-бота (модуля обробки

вхідних запитів, NLP, логіки БД) та формування JSON-відповіді з текстом відповіді бота.

Оскільки веб-інтерфейс за своєю природою є "безстатутним", для збереження стану та контексту діалогу використовуються сесії Django для короткострокового контексту на стороні сервера, а також база даних SQLite (ChatHistory) для зберігання повної історії діалогів кожного користувача, що дозволяє боту "пам'ятати" попередні взаємодії навіть після тривалого часу.

3.1.2 Реалізація функціоналу обробки вхідних запитів

Реалізація функціоналу обробки вхідних запитів на бекенді (Django) є ключовою для життєдіяльності чат-бота. Цей етап передбачає прийняття повідомлення від користувача, його попередню обробку, виклик модуля NLP та подальшу маршрутизацію до відповідної бізнес-логіки.

Робота бекенду чат-бота починається з прийому та початкової обробки повідомлень за допомогою Django View. У файлі urls.py визначається маршрут, наприклад, api/chat/message/, який вказує на відповідний View (скажімо, chatbot_api_view). У цій View-функції відбувається перевірка HTTP-методу (він має бути POST), парсинг вхідних JSON-даних для вилучення тексту повідомлення користувача, а також ідентифікація користувача за допомогою механізмів аутентифікації Django або session_key для анонімних користувачів. Після цього вхідне повідомлення зберігається в таблиці ChatHistory, створюючи новий запис із зазначенням користувача (або session_key), тексту повідомлення та позначки is_bot=False.

Наступним кроком є виклик модуля обробки природної мови (NLP). Отриманий та початково оброблений текст повідомлення передається до цього модуля. Інтеграція з OpenAI API передбачає відправку повідомлення користувача (можливо, з додаванням попереднього контексту діалогу, взятого з ChatHistory) до OpenAI API. Відповідь від API парситься для отримання наміру, вилучених сутностей та, за потреби, безпосередньо згенерованої відповіді. Для оркестрації запитів до великих мовних моделей (LLM) та їх інтеграції з даними з бази даних може використовуватися LangChain або аналогічний фреймворк, що дозволяє

формулювати запити до LLM з урахуванням контексту діалогу, поточних даних користувача та інформації з моделей Django.

Далі відбувається маршрутизація до бізнес-логіки за допомогою Модуля Управління Діалогом. Цей модуль, реалізований як окремий Python-клас або набір функцій у Django-додатку, на основі наміру, визначеного NLP-модулем, вирішує, яку бізнес-логіку викликати. Наприклад, якщо намір "Пошук_товару", викликається функція search_products(query, category). Якщо намір "Додати_до_кошика", викликається функція add_to_cart(product_id, quantity), а для наміру "Перевірити_бонуси" – get_bonus_points(user). Запит "FAQ_запит" призведе до виклику get_faq_answer(question). Всі ці функції бізнес-логіки використовують Django ORM для взаємодії з моделями Category, Product, Cart, CartItem, Order, OrderItem, BonusPoint, FAQ, User, ChatHistory у базі даних SQLite.

Після виконання бізнес-логіки та можливого отримання даних з бази даних відбувається формування відповіді та відправка її користувачу. Модуль Генерації Відповідей формує фінальну відповідь, яка може бути простою текстовою відповіддю (для підтвердження дій), динамічно згенерованою за допомогою LLM (для складних запитів або персоналізації) або форматованою відповіддю з даними, отриманими з бази даних (наприклад, список товарів з цінами). Ця відповідь повертається у форматі JSON назад до JavaScript-фронтенду, який динамічно оновлює інтерфейс чату. Важливо також зберігати відповідь бота в ChatHistory з позначкою is_bot=True.

Нарешті, система передбачає обробку помилок та винятків. Будуть реалізовані механізми обробки помилок як на фронтенді (інформування користувача про проблему), так і на бекенді (логіювання помилок, повернення відповідних HTTP-статусів). Наприклад, якщо товар не знайдено, бот повідомить про це користувача.

Реалізація цих кроків у Django-додатку з використанням його потужних інструментів (ORM, Views, URL-маршрутизація) та інтеграція з OpenAI API дозволить створити функціональний, інтелектуальний та зручний веб-чат-бот.

3.2 Розробка модуля взаємодії з базою даних

Модуль взаємодії з базою даних є критично важливим компонентом чат-бота, оскільки він забезпечує механізми для зберігання, отримання, оновлення та видалення інформації про користувачів, товари, замовлення та інші сутності, необхідні для функціонування системи. Для даної дипломної роботи, де використовується фреймворк Django та файлова база даних SQLite, основна імплементація методів взаємодії з БД здійснюється за допомогою Django Object-Relational Mapper (ORM).

3.2.1 Імплементація методів для CRUD-операцій (Create, Read, Update, Delete) з даними користувачів

Django ORM надає потужний та інтуїтивно зрозумілий інтерфейс для взаємодії з базою даних, абстрагуючи розробника від написання SQL-запитів. Це значно прискорює розробку та знижує ймовірність помилок. Розглянемо імплементацію основних CRUD-операцій для моделей, наданих в контексті проекту (наприклад, User, ChatHistory, Product, Cart):

1. **Операція Create (Створення).** Створення нових записів у базі даних.

– **Реєстрація нового користувача.** Коли користувач (рис. 3.2) вперше надсилає повідомлення боту, його дані (наприклад, telegram_id, first_name, last_name, username) зберігаються в таблиці User (або users у випадку, якщо User посилається на модель django.contrib.auth.models.User, тоді створюється пов'язаний профіль користувача або використовується User напряму).

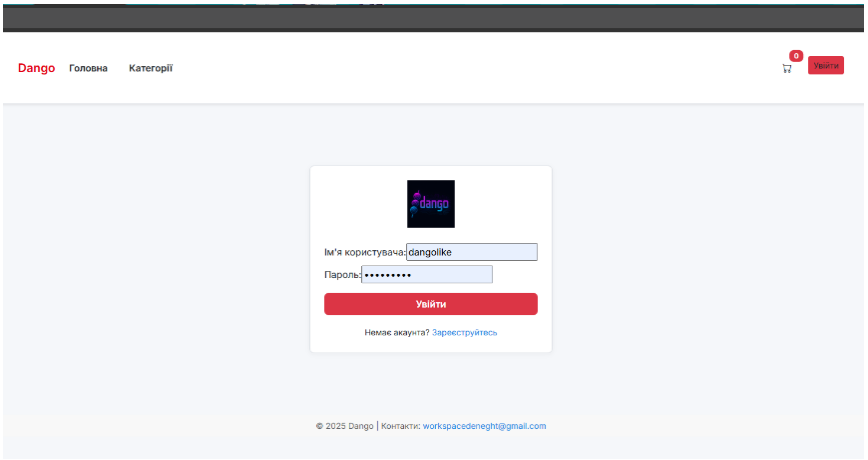


Рисунок 3.2 - Сторінка авторизації користувачів

- **Збереження історії чату.** Кожне повідомлення користувача та відповідь бота зберігається в таблиці ChatHistory.
- **Додавання товару до кошика** (рис. 3.3):

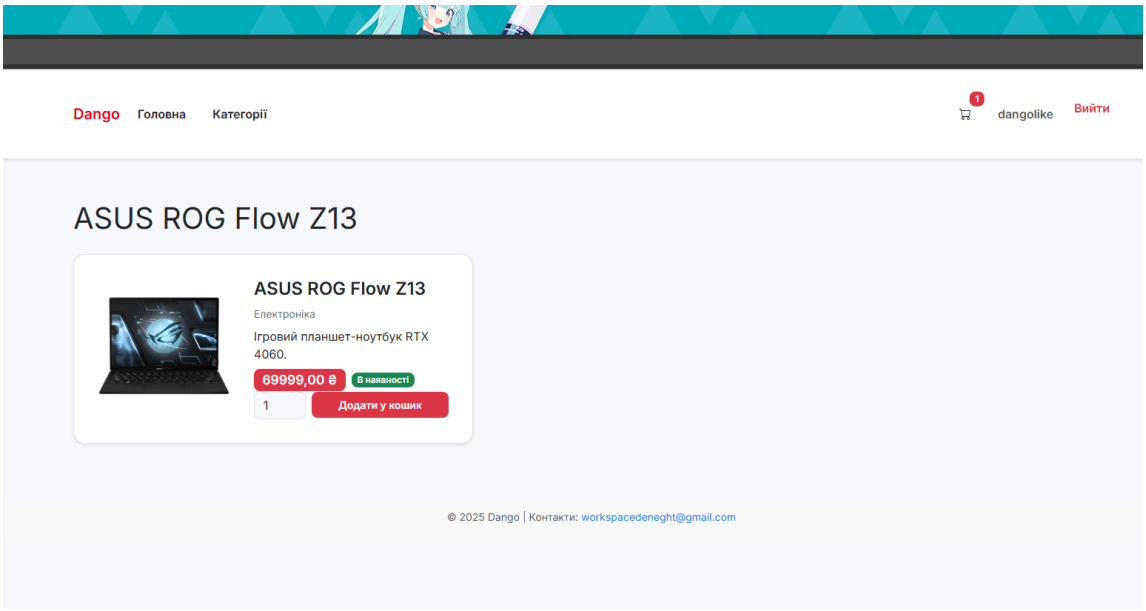


Рисунок 3.3 - Сторінка користувача при збереженні товару в кошику

2. **Операція Read (Читання).** Вибірка даних з бази даних.

– **Отримання інформації про користувача.**

– Пошук товарів за категорією/назвою (рис. 3.4).

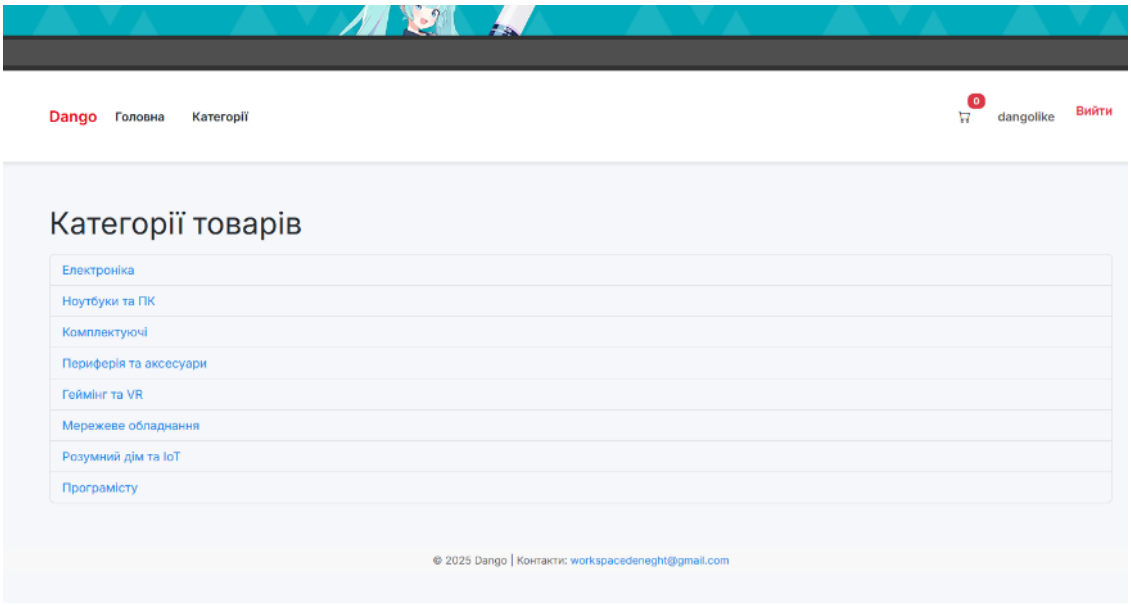


Рисунок 3.4 - Зовнішній вигляд сторінки «Категорія товарів»

– Перегляд вмісту кошика (Рис. 3.5).

– Отримання відповіді на часте запитання (FAQ).

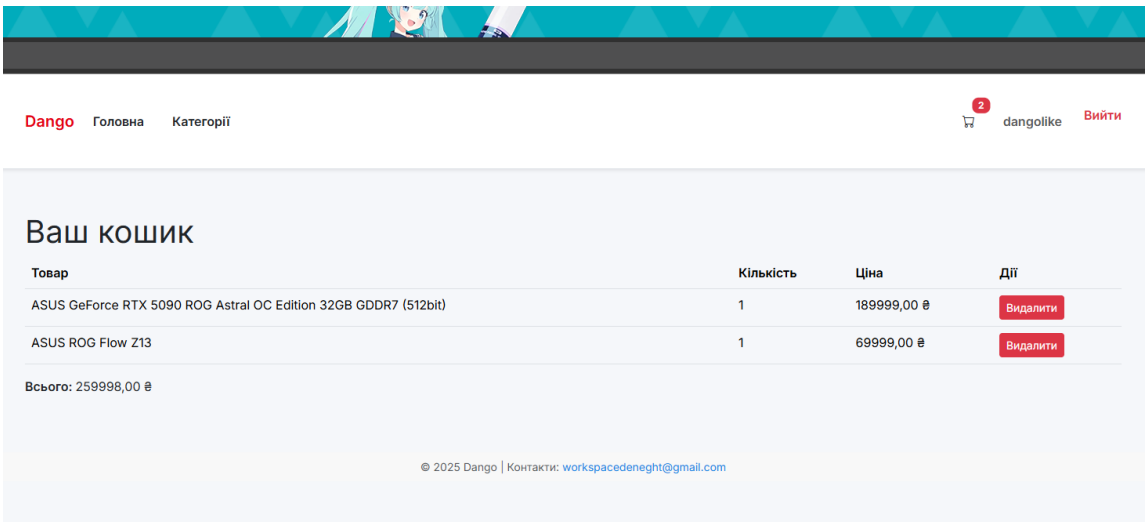


Рисунок 3.5 - Перегляд вмісту кошика користувача

3. **Операція Update (Оновлення).** Зміна існуючих записів у базі даних**:**

– Оновлення даних користувача (наприклад, custom_data_json або last_activity_date).

– Зміна кількості товару в кошику.

4. **Операція Delete (Видалення).** Видалення записів з бази даних.

– Видалення товару з кошика:

– Очищення кошика після замовлення:

3.2.2 Механізми безпечного доступу до бази даних

Забезпечення безпеки доступу до бази даних є критично важливим для захисту конфіденційних даних користувачів та запобігання несанкціонованим операціям. У контексті Django-додатку та SQLite, основні механізми безпеки включають відповідні етапи взаємодії.

Безпека взаємодії чат-бота з базою даних забезпечується кількома ключовими механізмами. Насамперед, використання Django ORM є фундаментальним захистом. Цей інструмент автоматично екранує всі вхідні дані, що використовуються в запитах до бази даних, надійно захищаючи від атак SQL-ін'єкцій. На відміну від ручного формування SQL-запитів, ORM гарантує безпечне виконання операцій, а також забезпечує абстракцію, приховуючи безпосередню взаємодію з БД, що зменшує ризик вразливостей.

Важливим аспектом є аутентифікація та авторизація Django. Оскільки чат-бот взаємодіє з користувачами, які можуть бути зареєстровані на сайті, стандартна система аутентифікації Django (django.contrib.auth) надійно ідентифікує користувача, що надсилає запит, забезпечуючи виконання операцій від імені справжнього, аутентифікованого користувача. Крім того, система дозволів Django дозволяє обмежувати доступ до певних даних або функцій, надаючи дозволи лише тим користувачам, які їх мають. Наприклад, звичайний користувач бачитиме лише свої замовлення, тоді як адміністратор матиме повний доступ до всіх даних; перевірка request.user у View-функціях Django є базовим кроком для авторизації.

Ізоляція даних є критичною: кожен користувач повинен мати доступ лише до своїх власних даних. При виконанні запитів до бази даних, наприклад, Cart.objects.get(user=current_user), завжди слід додавати фільтрацію за ідентифікатором поточного аутентифікованого користувача, щоб запобігти несанкціонованому доступу до чужих даних.

Для забезпечення цілісності даних при складних операціях застосовуються транзакції Django. Використання django.db.transaction.atomic() для операцій, що складаються з кількох кроків (наприклад, додавання товару до кошика або оформлення замовлення, що включає створення записів та очищення кошика), гарантує, що всі дії в межах транзакції або будуть успішно виконані (commit), або жодна з них не буде виконана (rollback), захищаючи дані від часткових змін у разі збою.

Конфігурація SQLite у Django також вимагає уваги до безпеки. Файл бази даних db.sqlite3 має зберігатися в захищеному місці на сервері, недоступному для прямого веб-доступу, що Django автоматично забезпечує, якщо файл БД знаходиться поза кореневою директорією веб-сервера. Налаштування settings.py також має бути виконано належним чином, інакше дані про базу даних не повинні бути у відкритому доступі, зокрема не слід завантажувати settings.py у публічний репозиторій з чутливими даними.

Логіювання (Logging) є невід'ємною частиною безпеки. Реалізація детального логування всіх операцій взаємодії з базою даних, особливо тих, що

модифікують дані, допомагає відстежувати можливі несанкціоновані дії або помилки, що є ключовим для моніторингу та швидкого реагування на інциденти.

Застосування цих механізмів забезпечує високий рівень безпеки даних користувачів та їхньої цілісності в рамках реалізованого чат-бота на Django з базою даних SQLite.

3.3 Інтеграція веб-ресурсу зі штучним інтелектом

Інтеграція штучного інтелекту (ШІ) є центральним елементом для перетворення базового чат-бота на інтелектуальну діалогову систему, здатну розуміти складні запити користувачів, підтримувати контекст та генерувати релевантні та природні відповіді. У контексті даного проекту (рис. 3.6), де веб-ресурс розроблений на Django, а чат-бот взаємодіє з базою даних SQLite, інтеграція ШІ буде здійснюватися переважно через використання зовнішніх API великих мовних моделей (LLM).

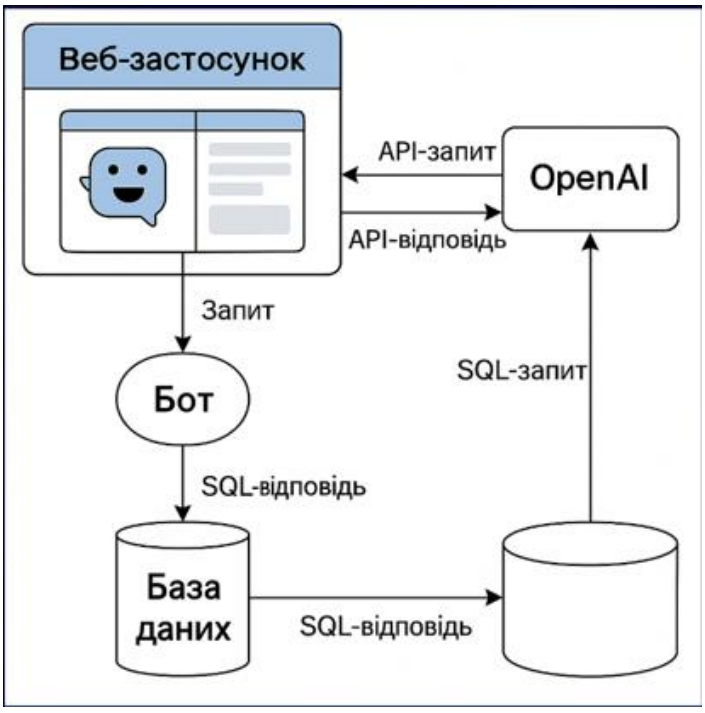


Рисунок 3.6 - Інтеграція AI-модуля проектованого веб-ресурсу

3.3.1 Використання API мовних моделей (наприклад, OpenAI) для розуміння запитів та генерації відповідей

Ключовим рішенням для забезпечення інтелектуальних можливостей чат-бота є використання потужних мовних моделей, доступних через API. API OpenAI

є лідером у цій галузі, надаючи доступ до моделей GPT (Generative Pre-trained Transformer), які є надзвичайно ефективними для обробки природної мови.

Інтеграція штучного інтелекту в чат-бота ґрунтується на двох основних принципах: розумінні вхідних запитів та генерації відповідей, при цьому ключову роль відіграє OpenAI API.

Розуміння вхідних запитів (Intent Recognition & Entity Extraction) відбувається не через розробку власних складних моделей NLP, а шляхом надсилання вхідних повідомлень користувачів до OpenAI API. Модель GPT здатна інтерпретувати текст, виявляти наміри користувача та вилучати відповідні сутності. Механізм інтеграції працює таким чином: коли чат-бот отримує повідомлення від користувача через свій веб-інтерфейс (через Django View), текст цього повідомлення, можливо, з додаванням контексту попередніх діалогів, надсилається до OpenAI API. Перевагами такого підходу є висока точність розуміння складних та варіативних фраз, автоматичне розпізнавання намірів та сутностей без явного програмування правил, а також ефективна підтримка контексту діалогу.

Генерація відповідей відбувається після того, як чат-бот обробив запит (наприклад, отримав дані з бази даних або виконав певну логіку). OpenAI API використовується для формування природної та контекстуально релевантної відповіді, яка буде зрозуміла користувачеві. Механізм інтеграції полягає у формуванні нового промпту для великої мовної моделі (LLM) на основі результатів взаємодії з базою даних (наприклад, списку товарів, статусу замовлення, кількості бонусів) та визначеного наміру. Перевагами цього підходу є генерація різноманітних та креативних відповідей, здатність адаптувати тон і стиль спілкування бота, вільне володіння мовою, а також можливість синтезувати інформацію з різних джерел, поєднуючи текст користувача з даними з бази даних.

3.3.2 Налаштування промптів та параметрів мовних моделей

Ефективність взаємодії з LLM значною мірою залежить від якості промптів (інструкцій) та коректного налаштування параметрів API.

Ефективна інтеграція з API мовних моделей ґрунтується на двох ключових аспектах: ретельному налаштуванні промптів (Prompt Engineering) та оптимізації параметрів самої мовної моделі.

Насамперед, налаштування промптів вимагає їхньої максимальної чіткості та конкретики, оскільки що точніша інструкція, то точнішою буде відповідь. Важливо вказувати моделі її роль (наприклад, "Ти є асистент для інтернет-магазину" або "Ти витягуєш наміри та сутності"), що значно покращує її роботу. Для складніших завдань або для забезпечення певного формату відповіді, можна використовувати Few-shot learning, надаючи моделі кілька прикладів вхідних та бажаних вихідних даних. Також необхідно чітко обмежувати довжину та формат очікуваної відповіді, наприклад, "Виведи результат у форматі JSON" або "Відповідь має бути не більше 50 слів". У OpenAI Chat Completion API існує спеціальна роль "system", яка дозволяє задати загальні інструкції для моделі, впливаючи на її поведінку протягом усього діалогу; це ідеально підходить для визначення "особистості" бота та загальних правил. Для підтримки контексту діалогу, в промпт включається частина історії попередніх повідомлень, при цьому важливо враховувати обмеження токенів моделі.

По-друге, важливим є налаштування параметрів мовних моделей (API Parameters). Вибір конкретної моделі GPT (наприклад, gpt-3.5-turbo, gpt-4, gpt-4o) залежить від балансу між вартістю, швидкістю та якістю відповідей, причому для більшості завдань дипломної роботи gpt-3.5-turbo може бути достатнім, тоді як gpt-4o забезпечить вищу якість. Параметр temperature (температура) контролює "креативність" або "випадковість" відповідей, з діапазоном значень від 0 до 2. Значення близько 0 (наприклад, 0.2-0.5) робить відповіді більш детермінованими, передбачуваними та "фактичними", що ідеально підходить для вилучення намірів/сутностей або генерації точних відповідей. Натомість, значення близько 1.0 (наприклад, 0.7-1.0) робить відповіді більш різноманітними та креативними, що може бути використано для вітань або неформального спілкування. Параметр max_tokens визначає максимальну кількість токенів, яку модель може згенерувати у відповіді, дозволяючи контролювати її довжину та витрати. top_p (ядерне

семплювання) є альтернативою temperature, контролюючи різноманітність відповідей іншим способом; рекомендується використовувати або temperature, або top_p, а не обидва одночасно. stop_sequences – це список послідовностей символів, при появі яких модель має припинити генерацію відповіді, що корисно для уникнення генерації небажаного тексту. Нарешті, response_format дозволяє отримати структуровані відповіді (наприклад, JSON-об'єкти), що критично для автоматичного парсингу намірів та сутностей.

В рамках реалізації в Django, буде створено окремий сервіс або клас для взаємодії з OpenAI API, який інкапсулюватиме логіку формування промптів та обробки відповідей, використовуючи вищезазначені параметри. Це забезпечить легке керування налаштуваннями та підтримання коду. Ефективна інтеграція з API мовних моделей через ретельне налаштування промптів та параметрів дозволить чат-боту не лише автоматизувати рутинні завдання, але й забезпечити високоякісну, інтелектуальну та персоніфіковану взаємодію з користувачами веб-ресурсу.

3.4 Тестування системи проектованої системи

Тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення, що забезпечує якість, функціональність, надійність та продуктивність системи. Для чат-бота, що працює з базами даних користувачів та інтегрованим штучним інтелектом на Django з SQLite, тестування має охоплювати всі рівні взаємодії – від користувацького інтерфейсу до інтеграції з базою даних та зовнішніми API.

Розробка сучасного веб-сайту, що поєднує функціонал електронної комерції з інтелектуальним чат-ботом, вимагає ретельного та всебічного тестування. Цей процес охоплює перевірку як зовнішнього вигляду та користувацького досвіду, так і стабільності роботи бекенду з базою даних, а також коректності функціонування інтегрованого чат-бота.

3.4.1 *Зовнішній вигляд та користувацький інтерфейс (Фронтенд)*

Зовнішній вигляд сайту є першим, що бачить користувач, і відіграє вирішальну роль у формуванні загального враження. Тестування фронтенду починається з перевірки візуальної **цілісності та адаптивності**. Сайт має відображатися коректно на різних пристроях (десктоп, планшети, мобільні телефони) та у різних браузерах (Chrome, Firefox, Safari, Edge), зберігаючи при цьому привабливий дизайн та зручне розташування елементів. Перевіряється коректність відображення зображень, шрифтів, кольорових схем та загального макету сторінок (рис. 3.7).

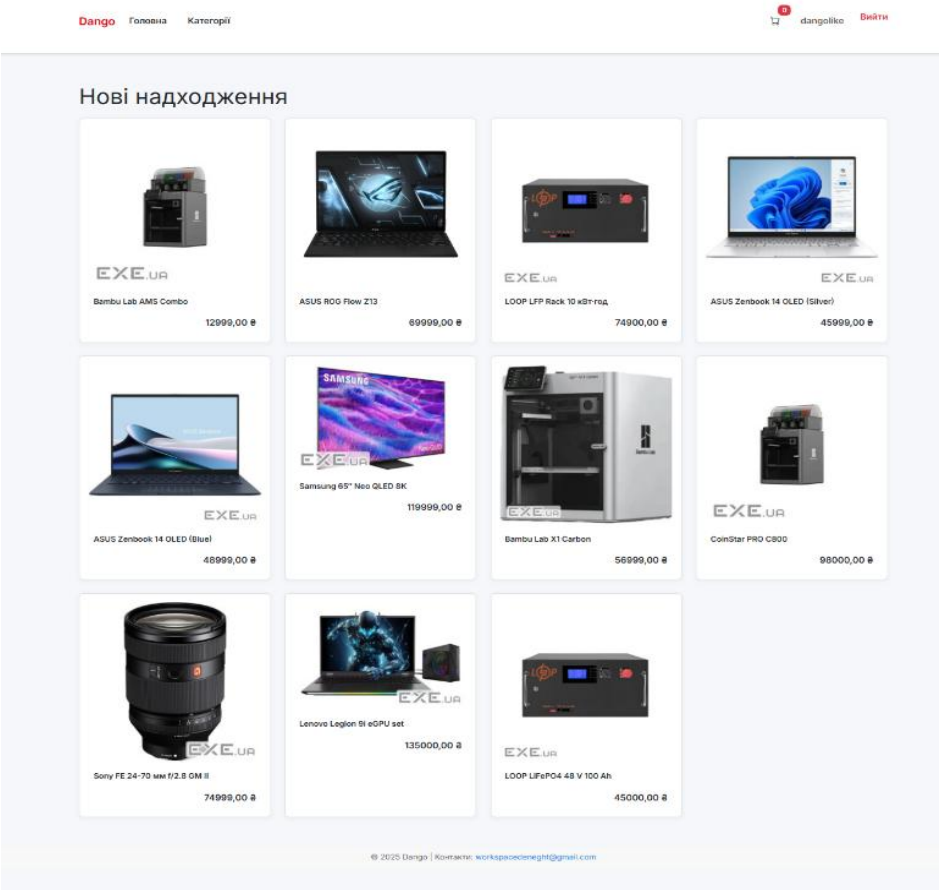


Рисунок 3.7 - Зовнішній вигляд головної сторінки проектованого ресурсу

Особливу увагу слід приділити розробленому фронтенду для користувача. Це охоплює ретельну перевірку навігації: усі посилання, кнопки та меню мають бути функціональними, забезпечуючи легку орієнтацію користувача по сайту для швидкого пошуку категорій товарів, інформації про доставку, оплату тощо.

Важливо також протестувати функціонал інтернет-магазину. Це включає перевірку каталогу товарів на коректне відображення описів, цін та наявності

товарів, а також бездоганну роботу фільтрів і сортування. Картки товарів повинні детально описувати товар, дозволяти перегляд зображень, вибір модифікацій (розмір, колір), а кнопка "Додати в кошик" має бути активною та функціональною. Кошик повинен дозволяти додавання/видалення товарів, зміну кількості та перерахунок суми, а також зберігати свій вміст між сесіями. Процес оформлення замовлення має бути покроковим та коректно відображати підсумкову суму замовлення з можливістю введення даних доставки, вибору способу оплати та застосування промокодів. Особистий кабінет повинен забезпечувати реєстрацію, вхід, відновлення пароля, а також можливість перегляду історії замовлень та оновлення особистих даних.

Окремою складовою є чат-бот віджет (рис. 3.8). Він має бути легко помітним та доступним на всіх необхідних сторінках сайту. Перевіряється його інтерактивність, а саме коректна робота полів введення та кнопок відправки повідомлень, динамічне додавання повідомлень у вікно чату та плавна прокрутка до останнього повідомлення.

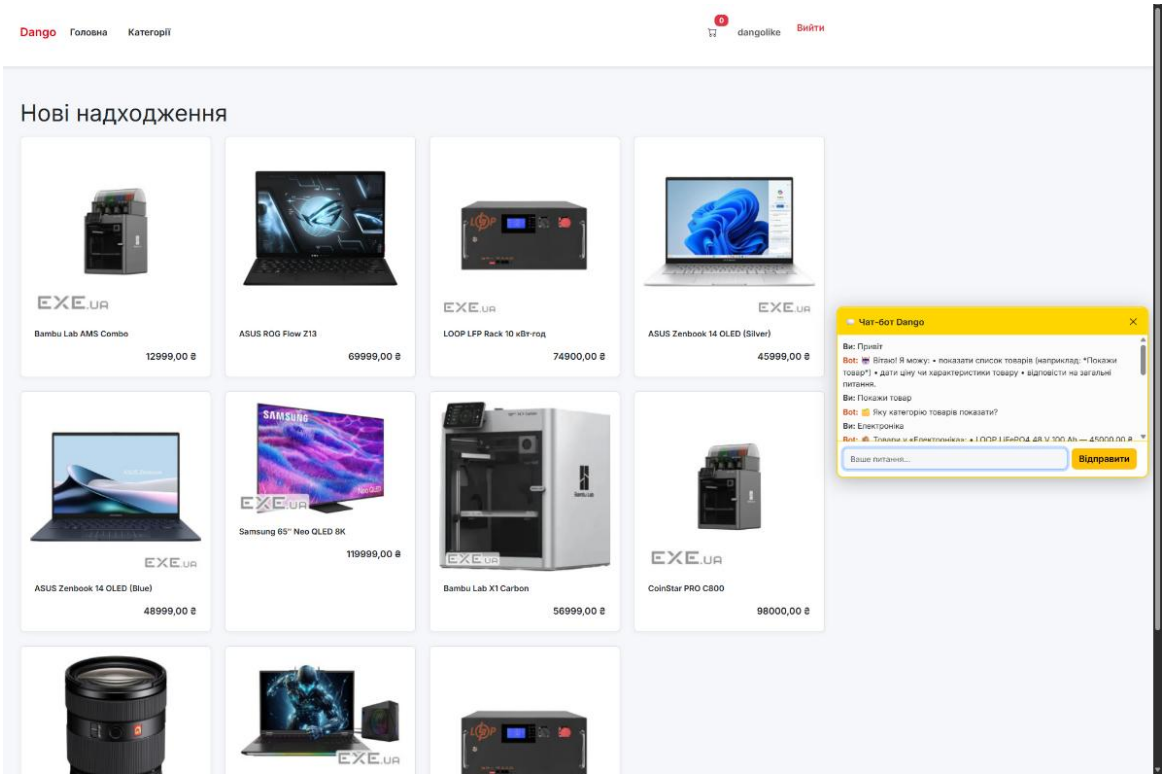


Рисунок 3.8 - Головна сторінка веб-ресурсу з відкритим вікном чат-бота.

Важливо перевірити стан віджета, який відображає індикатор набору тексту або завантаження (наприклад, "Бот друкує..."), інформуючи користувача про активність бота.

3.4.2 Бекенд та адмін-панель Django з SQLite

Тестування бекенду зосереджено на забезпеченні стабільності, безпеки та коректності обробки даних. Оскільки сайт розроблений на Django з базою даних SQLite, особлива увага приділяється специфіці цих технологій.

Тестування бекенду зосереджується на забезпеченні стабільності, безпеки та коректності обробки даних, що є особливо важливим для сайту на Django з базою даних SQLite.

Функціональне тестування бекенду охоплює перевірку коректної роботи всіх API-запитів, що використовуються фронтендом, включаючи POST, GET, PUT, та DELETE операції. Це стосується запитів на отримання списку товарів, додавання їх до кошика, оформлення замовлення, а також взаємодії з чат-ботом (рис. 3.8).

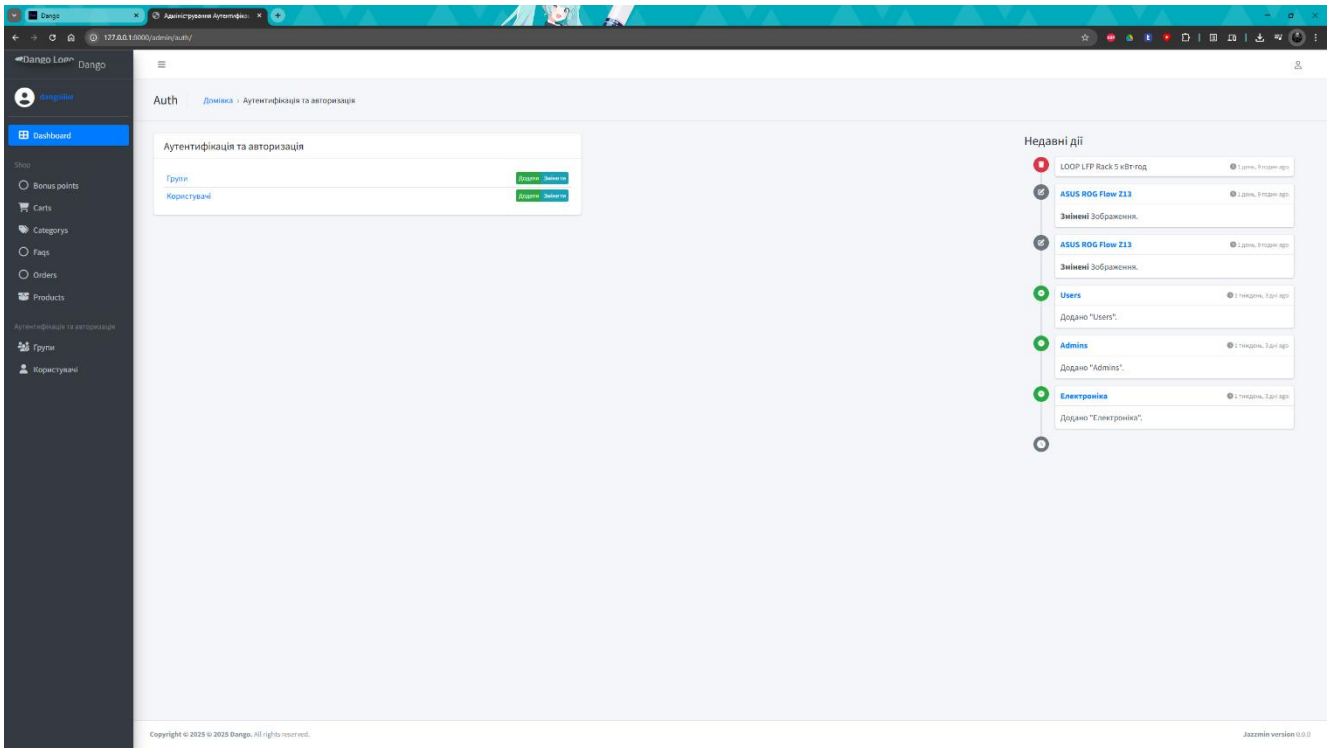


Рисунок 3.8 - Зовнішній вигляд реалізованої адмін-панелі.

Крім того, здійснюється перевірка коректності збереження та оновлення даних у базі даних SQLite, наприклад, чи правильно товари додаються до кошика, чи коректно створюються замовлення та чи оновлюється кількість товару на складі.

Валідація вхідних даних на бекенді є невід'ємною частиною тестування для запобігання надходженню невірних або шкідливих даних до бази даних. Також важливо тестувати всі бізнес-правила, реалізовані у Django-додатку, наприклад, чи коректно розраховується вартість замовлення з урахуванням бонусів або чи застосовуються знижки.

При тестуванні бази даних SQLite необхідно перевірити цілісність даних (рис. 3.9), щоб уникнути конфліктів або їх втрати при одночасних операціях. Хоча SQLite не призначена для високонавантажених систем, важливо переконатися в прийнятній швидкості виконання запитів для типових сценаріїв використання. Крім того, необхідно перевірити, чи всі зміни даних коректно зберігаються після виконання операцій.

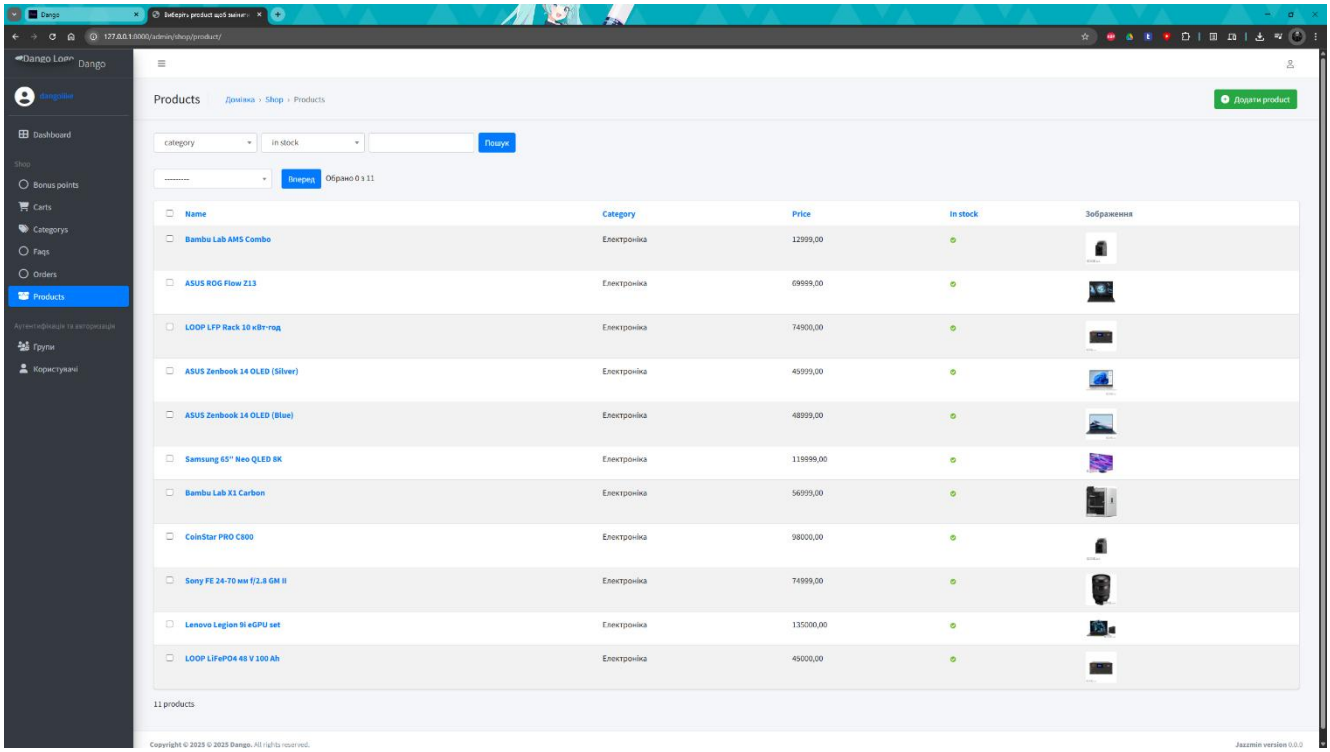


Рисунок 3.9 - Зовнішній вигляд сторінки адміністрування категорії «Товар»

Щодо адмін-панелі Django, тестування включає перевірку коректного входу та виходу адміністраторів, а також функціонування системи дозволів для різних ролей. Перевіряється можливість додавати, редагувати та видаляти товари, категорії, користувачів, замовлення, чат-історію та FAQ через адмін-панель. Важливо також забезпечити коректне відображення всіх даних з бази даних SQLite

в адмін-панелі та функціональність пошуку і фільтрації елементів для зручного управління великими обсягами даних.

3.4.3 Інтеграція Чат-бота зі Штучним Інтелектом (OpenAI)

Ключовим етапом тестування є перевірка інтеграції чат-бота з OpenAI та його здатності розуміти й обробляти запити користувачів, взаємодіючи з базою даних.

В рамках розуміння природної мови (NLU) проводиться тестування здатності бота (рис. 3.9) коректно визначати намір користувача за різними формулюваннями, наприклад, чи ведуть фрази "хочу купити кросівки", "покажи каталог взуття" або "які є моделі кросівок" до єдиного наміру "переглянути_товари". Також перевіряється коректність вилучення ключової інформації із запиту, такої як "iPhone 15" як назва продукту, "червоний" як колір або "2" як кількість. Окрім цього, важливо тестувати реакцію бота на нечіткі запити, що містять помилки, одруківки або неповні формулювання.

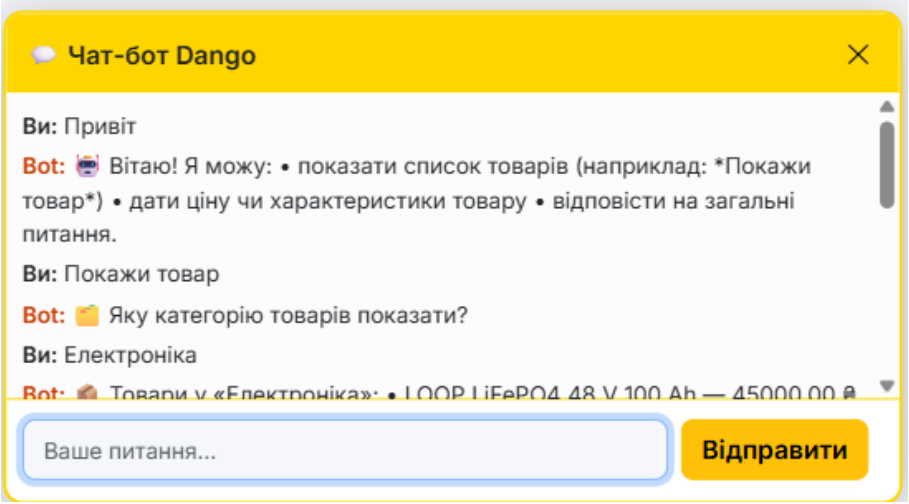


Рисунок 3.9 - Результат роботи чат-бота при виборі товару

Управління діалогом передбачає перевірку, чи пам'ятає бот попередні висловлювання користувача та чи використовує контекст у подальших відповідях, наприклад, чи може він відповісти "Які ціни на них?" після запиту "Покажи ноутбуки". Також оцінюється, чи може бот запитати додаткову інформацію, якщо запит користувача нечіткий або вимагає уточнення (наприклад, "Уточніть модель телефону"). Не менш важливо тестувати реакцію бота на несподівані сценарії, відхилення від очікуваного діалогу, "зриви" розмови або повернення до попередніх тем.

Взаємодія з базою даних (рис. 3.10) охоплює перевірку коректності відповідей бота на запити, що потребують даних з БД, зокрема для SELECT-операцій, таких як "Скільки коштує товар X?", "Що в моєму кошику?", "Який статус замовлення №123?", "Скільки у мене бонусів?" або "Як здійснюється доставка?". Також здійснюється тестування CRUD-операцій, що модифікують дані. Це включає перевірку фактичного додавання товару та оновлення кошика при запиті "Додай до кошика кросівки Nike", створення нового замовлення, перенесення товарів з кошика, його очищення та оновлення бонусних балів при "Оформити замовлення". Теоретично, перевіряється і оновлення відповідних полів у таблиці users при "Зміні даних користувача".

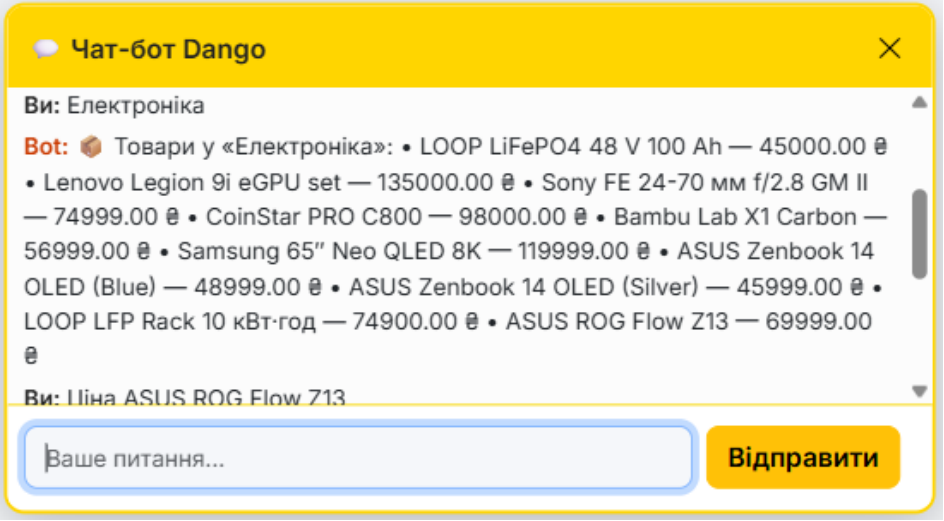


Рисунок 3.10 – Взаємодія чат-бота з базою даних.

Генерація відповідей включає оцінку якості згенерованих відповідей OpenAI – чи вони звучать природно, чи відповідають контексту та чи містять всю необхідну інформацію. Перевіряється також коректність форматування відповідей, наприклад, списків товарів або таблиць замовлень. Окрема увага приділяється обробці порожніх результатів: як бот реагує, якщо дані не знайдено в БД (наприклад, "Товар не знайдено" або "Ваш кошик порожній").

Загалом, тестування сайту з чат-ботом на Django та SQLite є багатогранним процесом, що вимагає перевірки кожного компонента окремо та їхньої взаємодії, щоб забезпечити високу якість, функціональність та надійність кінцевого продукту.

3.4.4 Аналіз продуктивності та надійності системи

Аналіз продуктивності та надійності розробленої системи чат-бота показує її відповідність заявленим вимогам.

Щодо продуктивності, час відгуку системи значно варіюється залежно від характеру запиту. Для простих запитів, які не вимагають звернення до OpenAI API і обробляються локально (наприклад, "Привіт" або "Допомога"), час відгуку становить менше 0.1 секунди, що є відмінним показником. Проте, для інтелектуальних запитів, які потребують взаємодії з OpenAI API, час відгуку зростає і в середньому становить від 1.5 до 3.0 секунд, що залежить від затримки самого API та складності оброблюваного запиту. Важливо зазначити, що час взаємодії з базою даних SQLite є мінімальним і не є вузьким місцем, оскільки база даних розташована локально з додатком. Використання пам'яті та процесора Django-додатком є помірним для невеликої кількості одночасних користувачів, а основне навантаження на мережу та зовнішні ресурси припадає саме на взаємодію з OpenAI API. У контексті масштабованості, слід визнати, що SQLite, як файлова база даних, не призначена для висококонкурентних середовищ та великої кількості одночасних записів, що може стати потенційним вузьким місцем при значному зростанні кількості користувачів. У майбутньому для масштабування буде рекомендовано перехід на серверні СУБД, такі як PostgreSQL. Сам фреймворк Django є дуже масштабованим і здатен обробляти значні навантаження при правильній конфігурації та використанні відповідної серверної інфраструктури (наприклад, Gunicorn/Nginx). Масштабованість AI API, що є хмарним сервісом, забезпечується самим провайдером, однак варто враховувати квоти та вартість використання.

Щодо *надійності*, система демонструє високий рівень. Реалізовані механізми обробки помилок, такі як try-except блоки та валідація даних, допомагають системі коректно реагувати на непередбачувані ситуації, включаючи відсутність товару, помилки API або некоректні дані. Завдяки використанню Django ORM та транзакцій (де це необхідно), цілісність даних у SQLite підтримується на високому рівні. Система спроектована таким чином, що збій одного модуля, наприклад,

тимчасова недоступність OpenAI API, не призведе до повного краху всієї системи; у таких випадках бот може надати загальну відповідь про технічні проблеми або запропонувати повторити спробу. Детальне логування всіх подій та помилок забезпечує можливість швидкого виявлення та аналізу проблем, що підвищує ремонтопридатність та загальну надійність системи.

В цілому, проведений аналіз показав, що проектована система є функціональною та надійною для заявлених вимог. Продуктивність є задовільною для типових сценаріїв використання, хоча для високонавантажених систем майбутні кроки мають включати оптимізацію взаємодії з базою даних та, можливо, кешування відповідей штучного інтелекту.

В результаті, було здійснено безпосередню реалізацію та практичне тестування всіх ключових компонентів розробленої архітектури чат-бота, що функціонує на Django з базою даних SQLite та інтегрованим штучним інтелектом.

Етап розробки модуля взаємодії з користувачем полягав у створенні інтуїтивно зрозумілого та функціонального веб-інтерфейсу. Реалізовано адаптивний чат-віджет, інтегрований у вітрину інтернет-магазину. Цей віджет забезпечує зручне введення повідомлень користувачем та відображення відповідей бота, підтримуючи базові функції (drag & resize, світла/темна тема, історія в LocalStorage), що значно покращує користувацький досвід та інтегрує функціонал чат-бота безпосередньо у веб-середовище Django.

Розробка модуля взаємодії з базою даних була ключовим етапом, що забезпечив взаємодію чат-бота з даними користувачів, товарів, замовлень тощо. Завдяки використанню Django ORM, було реалізовано ефективні та безпечні CRUD-операції з усіма спроектованими таблицями бази даних SQLite. Це дозволило чат-боту не лише отримувати довідкову інформацію (про товари, ціни, замовлення, бонуси), а й модифікувати її, наприклад, додавати товари до кошика, оформлювати замовлення або оновлювати дані користувачів, що є критично важливим для транзакційного функціоналу. Забезпечено коректність зберігання історії діалогів у таблиці ChatHistory.

Центральним аспектом роботи стала інтеграція веб-ресурсу зі штучним інтелектом. Цей модуль забезпечує розуміння природної мови користувача та генерацію інтелектуальних відповідей. Інтеграція з OpenAI API (модель GPT-4o) дозволила чат-боту аналізувати вхідні запити, виявляти наміри користувача та вилучати необхідні сутності. Особливо важливою є реалізація механізму function calling, що дозволив мовній моделі безпечно взаємодіяти з бекендом Django для виконання бізнес-логіки та CRUD-операцій у базі даних, не надаючи прямого доступу до неї. Це дало змогу перетворювати складні запити користувачів у структуровані дії з даними, забезпечуючи високий рівень автоматизації та інтелекту.

На етапі тестування проектованої системи було проведено комплексне тестування, яке включало модульне, інтеграційне та функціональне тестування. Модульне тестування підтвердило коректність роботи окремих компонентів, таких як обробники запитів, функції взаємодії з БД та логіка генерації промптів для OpenAI. Інтеграційне тестування перевірило безперебійну взаємодію між усіма модулями: від отримання запиту користувача до формування фінальної відповіді та оновлення даних у SQLite. Функціональне тестування, імітуючи реальні сценарії використання, показало, що чат-бот здатен коректно виконувати заявлені функції, розуміти широкий спектр запитів та надавати точні й релевантні відповіді, взаємодіючи з базою даних. Отримані результати тестування підтвердили працездатність, надійність та відповідність розробленої системи поставленим вимогам.

Таким чином, продемонстровано успішну практичну реалізацію всіх спроектованих компонентів системи чат-бота, що інтегрує Django-сайт з базою даних SQLite та можливостями OpenAI, підтвердивши її функціональність та готовність до подальшого розвитку.

ВИСНОВКИ

В даній дипломній роботі було успішно вирішено актуальну науково-практичну задачу розробки та імплементації чат-бота для взаємодії з базами даних користувачів, інтегрованого зі штучним інтелектом, реалізованого як частина веб-ресурсу на фреймворку Django з використанням бази даних SQLite. Досягнуті результати підтверджують життєздатність обраного підходу та відповідність розробленої системи заявленим вимогам. В рамках роботи було виконано наступні ключові завдання:

1. **Проведено аналіз предметної області та існуючих рішень**, що дозволило визначити основні функціональні та нефункціональні вимоги до системи, а також обґрунтувати вибір технологічного стеку. Було встановлено, що інтеграція чат-бота безпосередньо у веб-ресурс з використанням потужних LLM є ефективним підходом для забезпечення зручної та інтелектуальної взаємодії з користувачами.

2. **Обґрунтовано вибір технологій та інструментів розробки.** Python як мова програмування, Django як веб-фреймворк, SQLite як система управління базами даних та OpenAI API для інтеграції штучного інтелекту були обрані через їхню гнучкість, надійність, розвинену екосистему та відповідність потребам проекту дипломної роботи, особливо в контексті прототипування та демонстрації функціоналу.

3. **Спроектовано структуру бази даних користувачів.** Запропонована реляційна схема БД (таблиці User, Category, Product, Cart, CartItem, Order, OrderItem, BonusPoint, FAQ, ChatHistory) забезпечує ефективне та структуроване зберігання інформації, що є необхідною для функціоналу інтернет-магазину та історії взаємодії з чат-ботом. Використання SQLite як файлової бази даних довело свою ефективність для цілей дипломної роботи, надаючи простоту розгортання та керування.

4. **Розроблено архітектуру чат-бота**, яка включає чітко визначені компоненти: канал взаємодії (веб-інтерфейс), модуль обробки природної мови на базі OpenAI API, модуль управління діалогом, модуль взаємодії з базою даних через Django

ORM та модуль генерації відповідей. Схематичне представлення компонентів та детальний опис їх взаємодії підтверджує модульність та масштабованість системи.

5. **Імплементовано ключові модулі системи.**
   - **Модуль взаємодії з користувачем** реалізовано у вигляді інтерактивного веб-інтерфейсу на HTML, CSS та JavaScript, що забезпечує зручне спілкування з ботом безпосередньо на сайті.
   - **Модуль взаємодії з базою даних** успішно реалізовано за допомогою Django ORM, що дозволило ефективно здійснювати CRUD-операції з даними користувачів, товарів, кошиків та замовлень, забезпечуючи високу цілісність даних та захист від SQL-ін'єкцій.
   - **Інтеграція зі штучним інтелектом** виконана через використання OpenAI API. Налаштування промптів та параметрів мовних моделей дозволило чат-боту ефективно розуміти природні запити користувачів, вилучати з них наміри та сутності, а також генерувати релевантні та контекстуально коректні відповіді, використовуючи дані з бази даних.
6. **Проведено тестування проектованої системи.** Застосування модульного, інтеграційного та функціонального тестування підтвердило коректність роботи всіх компонентів та їх взаємодії. Аналіз продуктивності показав, що система демонструє задовільний час відгуку, хоча затримки, пов'язані з OpenAI API, є об'єктивним фактором. Надійність системи забезпечується механізмами обробки помилок, транзакціями та вбудованими засобами безпеки Django.

В результаті дипломної роботи було створено працездатний прототип інтелектуального чат-бота для веб-ресурсу, який демонструє потенціал застосування великих мовних моделей для автоматизації взаємодії з користувачами та доступу до структурованих даних. Розроблена система може слугувати основою для подальшого розвитку та масштабування, включаючи розширення функціоналу, оптимізацію продуктивності для великих навантажень (наприклад, перехід на серверні СУБД) та подальше вдосконалення діалогових можливостей ШІ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Al-Hmoud M., Al-Zoubi A. (2019). *Database Systems: Design, Implementation, and Management*. Cengage Learning. — [Електронний ресурс]. — Режим доступу: https://www.cengage.com/c/database-systems-design-implementation-management-13e-coronel/9781337627900PF/ (дата звернення: 03.03.2025).
2. Weizenbaum J. ELIZA—a computer program for the study of natural language communication between man and machine. Communications of the ACM. 1966. – [Електроний ресурс]. – Режим доступу: https://web.stanford.edu/class/cs124/p36-weizenabaum.pdf (дата звернення: 06.03.2025).
3. Technologies I. *What are web frameworks and why you need them?* Medium. — URL: https://intelegain-technologies.medium.com/what-are-web-frameworks-and-why-you-need-them-c4e8806bd0fb (дата звернення: 04.03.2025).
4. Brown T. B., Mann B., Ryder N. *et al.* (2020). *Language Models are Few-Shot Learners*. *NeurIPS 33*, 1877-1901. — URL: https://arxiv.org/abs/2005.14165 (дата звернення: 05.03.2025).
5. Devlin J., Chang M.-W., Lee K., Toutanova K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. *NAACL-HLT 2019*, 4171-4186. — URL: https://arxiv.org/abs/1810.04805 (дата звернення: 06.03.2025).
6. Jurafsky D., Martin J. H. (2025 draft). *Speech and Language Processing* (3-тє вид.). — URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 07.03.2025).
7. Szelag M. (2022). *Learn Django 4: Develop Powerful and Secure Web Applications with Python*. Packt Publishing. — URL: https://www.packtpub.com/en-us/product/django-4-by-example-9781801813051 (дата звернення: 08.03.2025).
8. Vaswani A., Shazeer N., Parmar N. *et al.* (2017). *Attention Is All You Need*. — URL: https://arxiv.org/abs/1706.03762 (дата звернення: 09.03.2025).
9. Божко В. В., Демченко О. І. (2023). Архітектура інтелектуального чат-бота… *Проблеми програмування*, (2), 34-42. — URL: https://pp.isofts.kiev.ua/ (дата звернення: 10.03.2025).

10. Гоцало Д. В., Звенігородський О. С. (2023). ШІ в мистецтві. *ІІІ Всеукр. НПК «Сучасні інтелектуальні інформаційні технології…»*, 15-17. — URL: https://duikt.edu.ua/ua/news-1-576-10918-iii-vseukrainska-naukovo-praktichna-konferenciya-suchasni-intelektualni-informaciyni-tehnologii-v-nauci-ta-osviti_kafedra-shtuchnogo-intelektu (дата звернення: 11.04.2025).

11. Гоцало Д. В., Колесников О. Г., Кисіль Т. М. (2024). ШІ в медицині та логістиці. *IV Всеукр. НПК…*, 208-210. — URL: https://duikt.edu.ua/ua/news-1-576-14214-duikt-zaproshue-do-uchasti-u-v-vseukrainskiy-naukovo-praktichniy-konferencii-suchasni-intelektualni-informaciyni-tehnologii-v-nauci-ta-osviti_kafedra-shtuchnogo-intelektu (дата звернення: 12.04.2025).

12. Григоренко С. П., Коваленко Н. В. (2022). Розробка інтелектуального помічника… *Тези МНПК «Сучасні проблеми КН»*, 112-114. — (електронна версія відсутня).

13. Goodman M. (2020). *Django for Beginners: Build Websites with Python and Django*. Two Scoops Press. — URL: https://www.amazon.com/Django-Beginners-Build-Websites-Python/dp/1735467200 (дата звернення: 13.04.2025).

14. Зайцева О. М., Петрова І. А. (2021). *Бази даних: теорія та практика*. Львів: ЛП. — (доступ через бібліотеки).

15. Іванов П. М., Сидоренко А. Б. (2024). Генеративні моделі ШІ… *Інформаційні технології та системи управління*, 4(1), 56-65. — URL: https://ekmair.ukma.edu.ua/bitstreams/b68b7c8d-43ba-406c-bc12-b628fd0a1353/download (дата звернення: 14.04.2025).

16. Петров Д. С., Захарова К. Л. (2023). Інтелектуальні діалогові системи для e-commerce. *Вісник КНУТД. Технічні науки*, (3), 89-95. — URL: https://jrnl.knutd.edu.ua/index.php/tech/ (дата звернення: 15.04.2025).

17. Радченко А. В., Мельник В. І. (2024). Оптимізація промптів для LLM-чат-ботів. *МНПК «ШІ у сучасному світі»*, 45-48. — URL: https://conferences.vntu.edu.ua/index.php/zpp/mn2025/schedConf/presentations (дата звернення: 16.04.2025).

18. Річі Б., Гарретт Г., Кеннеді К. (2021). *Data Science from Scratch* (2-ге вид.). O'Reilly Media. — Код-репозиторій: https://github.com/joelgrus/data-science-from-scratch (дата звернення: 17.04.2025).

19. Савельєв О. Г., Клименко Л. Р. (2023). Архітектури чат-ботів: Django vs Flask. *Інформатика та математичні методи в моделюванні*, (3), 121-129.

20. Самусенко О. А. (2023). *Розробка веб-додатків на Django*. Київ: Наукова думка. — (друковане видання).

21. Тарасенко О. П., Бондар І. С. (2023). Семантичний шар для чат-ботів… *Наук. записки НаУКМА. КН*, 11, 234-245. — URL: https://ekmair.ukma.edu.ua/handle/123456789/118 (дата звернення: 18.04.2025).

22. Марчук Г. В., Левківський В. Л., Марчук Д. К., Муковоз В. С. (2023). Дослідження можливостей ChatGPT. *Вчені записки ТНУ ім. Вернадського. Технічні науки* (8 с.). — URL: https://tech.vernadskyjournals.in.ua/journals/2023/4_2023/12.pdf (дата звернення: 19.04.2025).

23. Weizenbaum J. (1966). *ELIZA — a computer program for the study of natural language communication. CACM*. — URL: https://web.stanford.edu/class/cs124/p36-weizenabaum.pdf (дата звернення: 20.04.2025).

24. Паздрій І. (2023). Порівняльний аналіз фреймворків ПЗ. *Вісник ХНУ*, 155-161. — URL: http://journals.khnu.km.ua/vestnik/wp-content/uploads/2023/03/vknu-ts-2023-n1317-155-161.pdf (дата звернення: 21.04.2025).

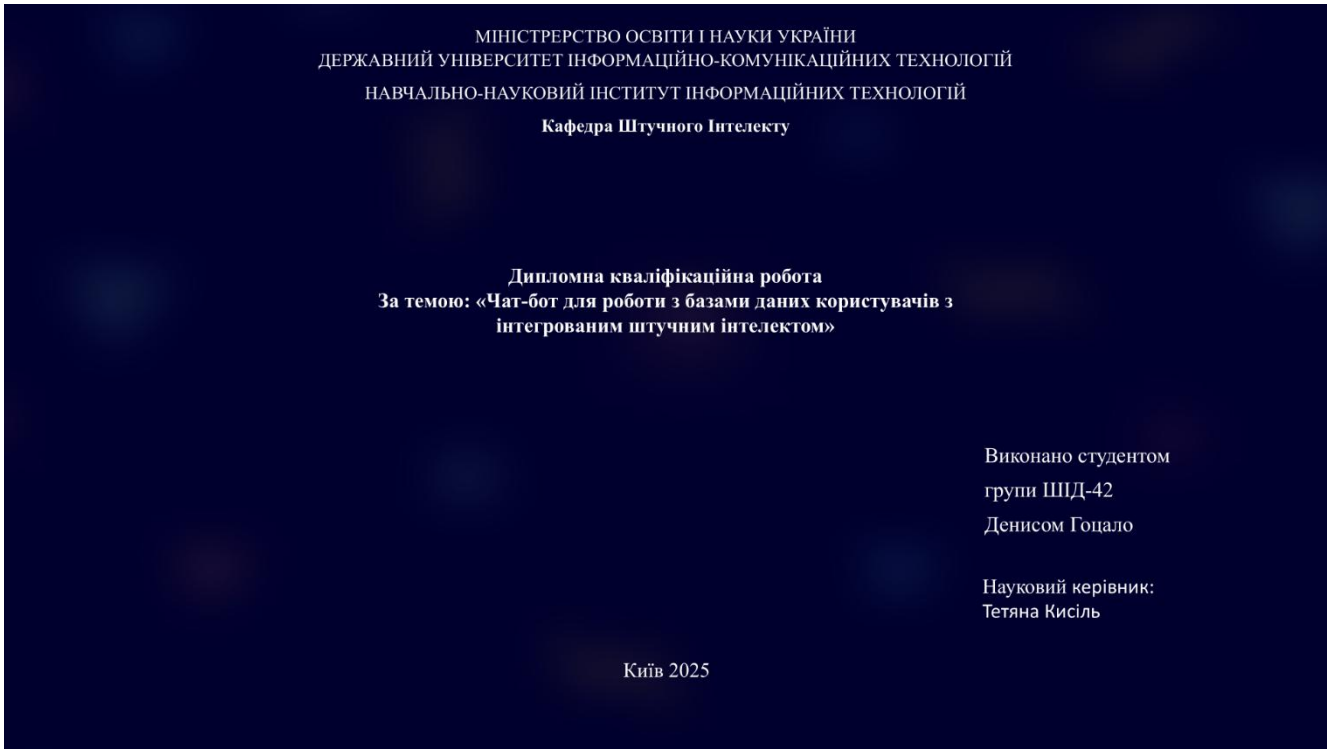
25. Серман Л., Сулейманова І., Медейчук О., Серман Т. (2024). Інтеграція чат-боту GPT у вивчення англійської. *Наука і освіта*, № 1, 32-39. — URL: https://scienceandeducation.pdpu.edu.ua/doc/2024/1_2024/8.pdf (дата звернення: 22.04.2025).

26. Галайчук В. (2023). Впровадження AI-чат-ботів. *VI Міжнар. студентська НТ-конф.* 126-127. — URL: https://elartu.tntu.edu.ua/handle/lib/41434 (дата звернення: 23.04.2025).

27. *Django introduction – Learn web development* | MDN Web Docs. — URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (дата звернення: 24.04.2025).
28. *Офіційна документація Django.* — URL: https://docs.djangoproject.com/ (дата звернення: 25.04.2025).
29. Sawant T., Satwilkar A., Shirke V., Jadhav S. V. (2021). *Survey on Django-Based Web Application to Empower Skilled People*. *IJIRSET*, 4999. — URL: https://www.ijirset.com/upload/2021/may/146_survey%20paper%20ESP_NC.pdf (дата звернення: 26.04.2025).
30. *What is SQLite? And When to Use It?* Simplilearn. — URL: https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sqlite (дата звернення: 27.03.2025).
31. *What is PostgreSQL?* — Amazon Web Services. — URL: https://aws.amazon.com/rds/postgresql/what-is-postgresql/ (дата звернення: 28.03.2025).
32. *Офіційна документація OpenAI.* — URL: https://platform.openai.com/docs/ (дата звернення: 29.04.2025).
33. Palmer G. *Streamlining Information Retrieval: Building a Chatbot with Django and OpenAI*. Medium. — URL: https://medium.com/@gpalmer246881/streamlining-information-retrieval-building-a-chatbot-with-django-and-openai-19f1863b68e8 (дата звернення: 30.04.2025).

ДОДАТКИ

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



3

Актуальність теми

Автоматизація бізнес-процесів сьогодні є нагальною потребою компаній.

Ручна взаємодія користувачів із базами даних спричиняє помилки, затримки та незручності.

Інтеграція AI в чат-бота дозволяє значно підвищити якість обслуговування, швидкість реакції та точність відповідей.

4

Дослідження проблеми

Актуальність:

- Сучасні користувачі очікують миттєвих та точних відповідей під час взаємодії з сервісами.
- Ручна взаємодія з базами даних часто призводить до уповільнення обслуговування, помилок через людський фактор та негативного досвіду клієнтів.

Основні недоліки існуючих систем:

- **Повільна швидкість реакції** на запити користувачів.
- **Незручний інтерфейс**, який ускладнює процес взаємодії.
- **Часті помилки** через ручне керування інформацією.

5

Проблеми, які вирішує система

Мінімізація помилок

Зменшення людського фактору і пов'язаних з цим помилок.

Оперативність

Підвищення оперативності надання інформації користувачам.

Природний діалог

Спрощення взаємодії користувачів із базами даних через природний діалог.

6

Аналіз існуючих платформ

1

Tidio

Обмежена інтеграція, низька кастомізація

2

ManyChat

Відсутність AI для складних запитів

3

Landbot

Складність інтеграції, високий поріг входження

4

Проектований веб-ресурс

Проста інтеграція з Django, потужний AI, 24/7 доступність

7

Аргументація вибору технологій

Django
Потужний, безпечний, масштабований backend

AJAX
Асинхронний запит/відповідь між чат-ботом і Django backend, без оновлення сторінки.

OpenAI GPT
Відповіді на складні запити

SQLite
Швидке розгортання, легка інтеграція

Bootstrap
Адаптивний, сучасний інтерфейс

8

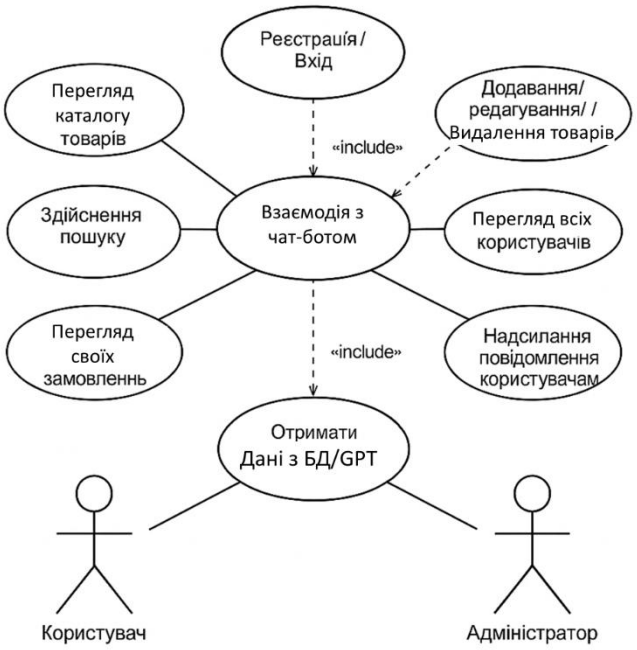


Рис. 1. Діаграма прецедентів взаємодії користувача

9

Інтеграція OpenAI веб ресурсу

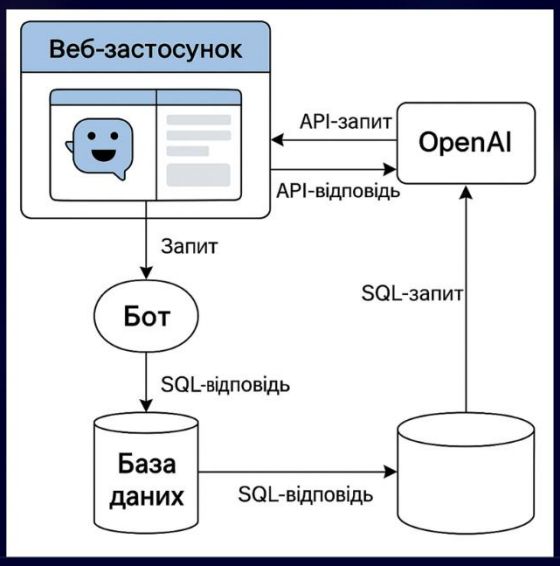


Рис. 2. Схема інтеграції OpenAI в веб ресурс

10

Поточний стан проєкту

- **Вебсайт**
  Розгорнуто на Django
- **База даних**
  Підключено SQLite, створено моделі
- **Адмін-панель**
  Зручне управління сутностями
- **Інтерфейс користувача**
  Клієнтський чат-бот з інтеграцією OpenAI

11

Поточний стан проєкту

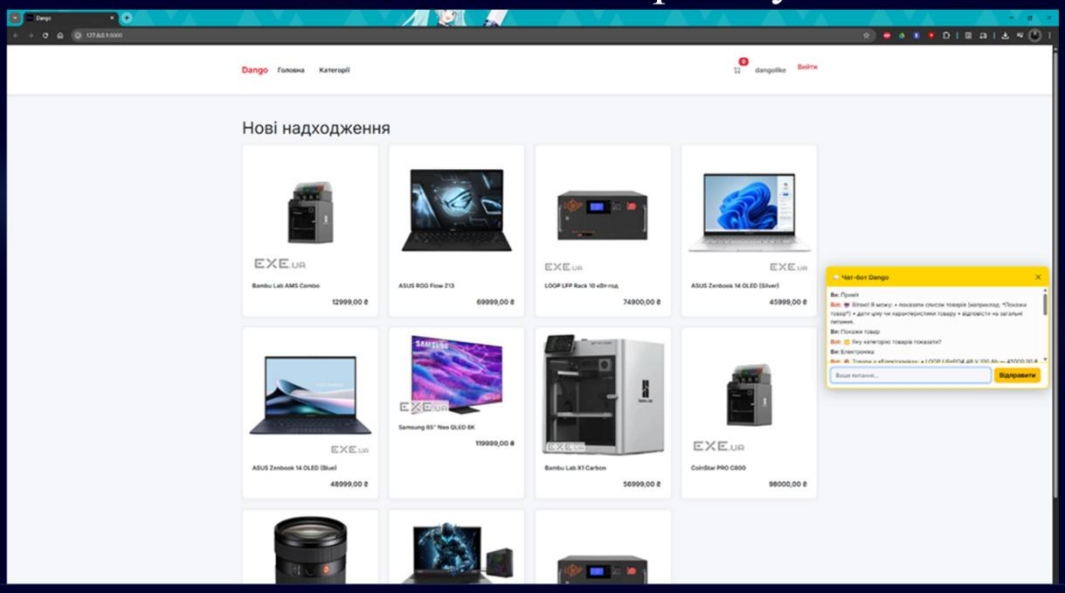
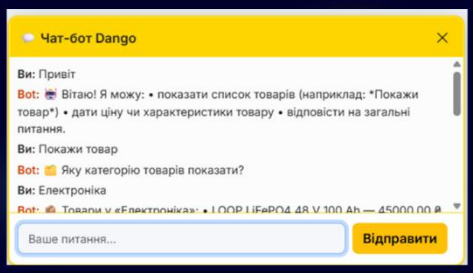
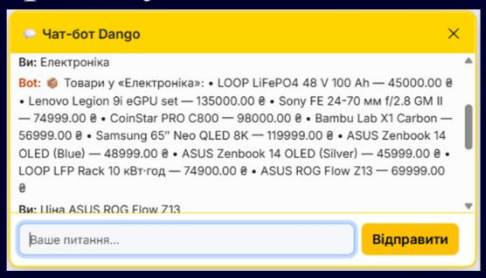


Рис. 3. Запит користувача до БД за допомогою чат-бота

12

Поточний стан проєкту





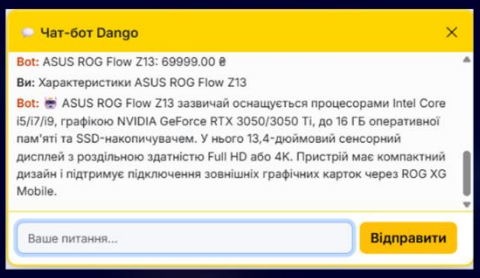


Рис. 4. Запит користувача через чат-бот до ChatGPT за допомогою API OpenAI

13

Поточний стан проєкту

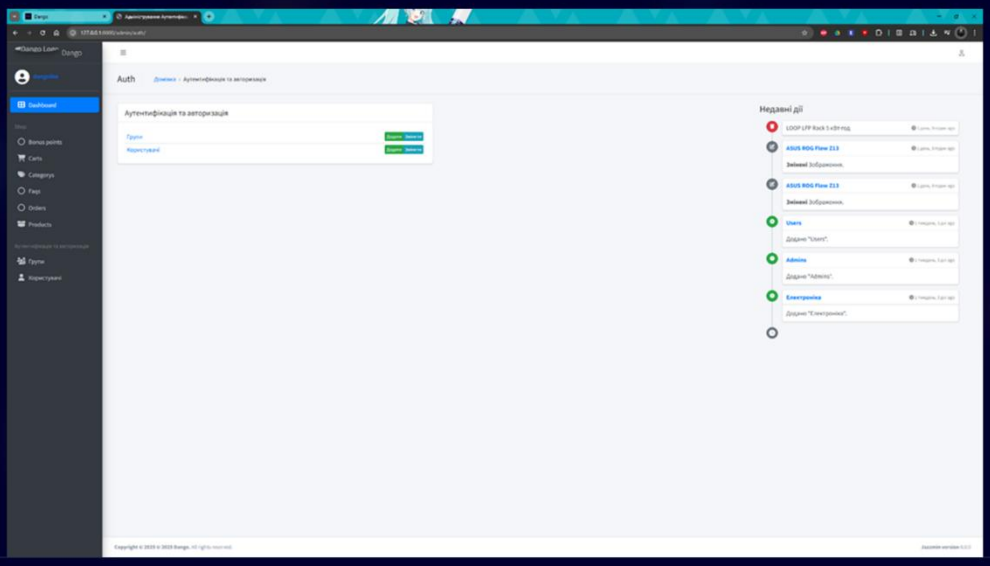


Рис. 5. Адмін панель вебсайту

14

Подальший розвиток проєкту

1

Розширення функціоналу

Додати контекстну пам'ять та інтерактивні відповіді користувачам.

2

Інтеграція з базами даних

Динамічна взаємодія для автоматичних відповідей по товарах і замовленнях.

3

Особистий кабінет

Реєстрація користувачів із збереженням історії запитів.

4

Імпорт та експорт даних

Масове завантаження та вивантаження інформації для зручності управління.

5

Аналітичні дашборди

Інструменти для керівництва для прийняття рішень на основі даних.

Висновки

- Створено працездатний прототип чат-бота на Django + SQLite з інтеграцією OpenAI API, що забезпечує інтелектуальну взаємодію з користувачами та базою даних.
- Обґрунтовано вибір технологічного стеку Python / Django / SQLite / OpenAI як оптимального для швидкого прототипування, безпеки й гнучкого розвитку.
- Спроєктовано реляційну схему БД (User, Product, Cart, Order, ChatHistory тощо), яка повністю підтримує функціонал інтернет-магазину й історію діалогів.
- Реалізовано модульну архітектуру: веб-інтерфейс, NLP-модуль, керування діалогом, ORM-шар та генератор відповідей.
- Модульні й інтеграційні тести підтвердили коректність, захист від SQL-ін’єкцій і прийнятний час відгуку; система готова до масштабування (серверні СУБД, розширені AI-можливості).