МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ЧАТБОТУ ДЛЯ АВТОМАТИЗАЦІЇ ПІДТРИМКИ КЛІЄНТІВ З ВИКОРИСТАННЯМ REACT, PYTHON ТА POSTGRESQL»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.А. Мкртчян
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Мкртчяну Артуру Артаковичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка чатботу для автоматизації підтримки клієнтів з використанням React, Python та PostgreSQL

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі та планування проєкту.
2. Проєктування та аналіз вимог.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	13.01.2025	
2.	Збір вихідних даних.	10.02.2025	
3.	Обробка методичних та теоретичних	10.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	14.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2025	

Студент _____ А.А. Мкртчян
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка чатботу для автоматизації підтримки клієнтів з використанням React, Python та PostgreSQL»: 66 с., 19 рис., 2 табл., 18 джерел.

АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ПІДТРИМКА КЛІЄНТІВ, ЧАТБОТ, POSTGRESQL, PYTHON, REACT.

Об’єкт дослідження – процес створення чатботів для автоматизованої взаємодії з користувачами.

Предмет дослідження – розробка чатботу для автоматизації підтримки клієнтів із використанням React, Python та PostgreSQL.

Мета роботи – розробка чатботу для автоматизації клієнтської підтримки.

Методи дослідження – методи об’єктно-орієнтованого програмування, інструменти та практики програмної інженерії.

У роботі представлено етапи розробки чатботу, який призначений для автоматизації обробки запитів користувачів у режимі реального часу. Наведено аналіз предметної області, зокрема особливостей організації клієнтської підтримки. Описано технічні аспекти реалізації чатботу з використанням сучасного стеку технологій: React для фронтенду, Python для бекенду та PostgreSQL як системи керування базами даних. Розглянуто ключові виклики під час збору та аналізу вимог, проєктування архітектури та реалізації бізнес-логіки чатботу. Зростання попиту на автоматизовану взаємодію з користувачами робить тему розробки чатботів особливо актуальною.

SUMMARY

Bachelor’s qualifying paper “Development of a Chatbot for Customer Support Automation Using React, Python and PostgreSQL”: 66 pages, 19 figures, 2 tables, 18 references.

AUTOMATION, CHATBOT, CUSTOMER SUPPORT, DATABASE, POSTGRESQL, PYTHON, REACT, WEB APPLICATION.

Object of study – the process of creating chatbots for automated user interaction.

Subject of study – development of a chatbot for automating customer support using React, Python, and PostgreSQL.

Objective of the thesis – development of a chatbot for automating customer support.

Research methods – object-oriented programming methods, tools and practices of software engineering.

The thesis presents the stages of developing a chatbot designed to automate the processing of user requests in real time. The domain area is analyzed, with a focus on the specifics of organizing customer support. The technical aspects of implementing the chatbot using a modern technology stack are described: React for the frontend, Python for the backend, and PostgreSQL as the database management system. Key challenges in requirements gathering, architecture design, and business logic implementation are discussed. The growing demand for automated user interaction makes chatbot development a highly relevant topic.

ЗМІСТ

Завдання на кваліфікаційну роботу……..……..2
Реферат 4
Summary 5
Вступ..... 7
1 Огляд теоретичних властивостей та планування проєкту 9
1.1 Огляд сучасних систем автоматизації клієнтської підтримки 9
1.2 Аналіз фреймворку React та його можливостей 13
1.3 Все про Python (FastAPI) його перваги та недоліки..... 16
1.4 Все про PostgreSQL 19
1.5 Порівняння з альтернативними технологіями та платформами..... 21
1.6 Висновки до розділу 1 22
2 Проєктування та аналіз вимог до онлайн чату..... 24
2.1 Опис предметної області 24
2.2 Функціональні вимоги та нефункціональні вимоги 25
2.3 Інтерфейсні вимоги 27
2.4 Проєктування фронтендної частини 30
2.5 Налаштування серверної частини та запитів АПІ 34
2.6 Інтеграція бази даних..... 37
2.7 Висновки до розділу 2 41
3 Реалізація та тестування 43
3.1 Побудова API серверної частини на Python (FastAPI) 43
3.2 Налаштування бази даних PostgreSQL та зберігання діалогів 48
3.3 Реалізація клієнтського інтерфейсу чатботу на React 51
3.4 Тестування функціоналу та обробка типових сценаріїв 56
3.5 Висновки до розділу 3 62
Висновки 63
Перелік посилань..... 65

ВСТУП

Розвиток цифрових технологій суттєво змінює способи взаємодії між бізнесом та клієнтами. В умовах високої конкуренції та зростаючих очікувань користувачів важливо забезпечити миттєву, ефективну й персоналізовану підтримку. Одним із найперспективніших рішень у цій сфері є впровадження чатботів – автоматизованих систем для комунікації з клієнтами, здатних відповідати на запити в режимі реального часу.

Сучасні вебсайти відіграють ключову роль у взаємодії компаній з клієнтами. Проте для підвищення ефективності цієї взаємодії та зменшення навантаження на операторів підтримки дедалі частіше використовуються чатботи. Вони дозволяють забезпечити цілодобову підтримку, знизити витрати на обслуговування клієнтів та покращити загальний користувацький досвід.

Цей проєкт присвячений розробці чатботу для автоматизації підтримки клієнтів із використанням сучасних технологій – React, Python та PostgreSQL. React буде використано для створення динамічного та зручного клієнтського інтерфейсу. Python (зокрема фреймворк FastAPI або Django) відповідатиме за обробку логіки спілкування чатбота з користувачами, а PostgreSQL – за надійне зберігання даних, включаючи історію діалогів, відповіді та сценарії обробки запитів.

У межах проєкту буде проаналізовано предметну область, визначено ключові вимоги до функціоналу чатбота, розроблено архітектуру системи та реалізовано її основні компоненти. Особлива увага приділятиметься питанням обробки природної мови, інтеграції з базами знань та забезпеченню масштабованості рішення.

Також важливим аспектом стане безпека обробки даних, адже чатботи часто працюють із персональною або конфіденційною інформацією користувачів. Буде реалізовано методи шифрування, автентифікації та контроль доступу, щоб забезпечити захищене середовище взаємодії.

Результатом роботи стане функціональний чатбот, який може інтегруватися з корпоративними вебресурсами, забезпечуючи швидку, ефективну та безперебійну підтримку клієнтів. Це дозволить підвищити рівень задоволеності користувачів, скоротити час обробки запитів та оптимізувати витрати бізнесу на обслуговування.

Таким чином, розробка чатботу з використанням React, Python та PostgreSQL є актуальним і практично значущим завданням, яке відповідає сучасним викликам цифрової трансформації та орієнтоване на покращення якості клієнтського сервісу.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВЛАСТИВОСТЕЙ ТА ПЛАНУВАННЯ ПРОЄКТУ

1.1 Огляд сучасних систем автоматизації клієнтської підтримки

У сучасному бізнес-середовищі автоматизація клієнтської підтримки є однією з ключових складових успішного обслуговування користувачів. З розвитком цифрових технологій компанії все частіше звертаються до використання систем автоматизації, які допомагають знизити витрати на підтримку, підвищити якість обслуговування та оперативність реагування на запити клієнтів [3].

Основні види систем автоматизації підтримки

– чатботи;
– системи керування зверненнями (Helpdesk);
– автоматизовані голосові системи (IVR);
– системи управління знаннями (Knowledge Base).

Розглянемо кожен з них більш детально, та проведемо аналіз їх переваг, щоб отримати загальну картину.

Чатботи – це програмні агенти, які використовують штучний інтелект та алгоритми обробки природної мови для автоматичного спілкування з користувачами у текстовому або голосовому форматі. Вони стали одним із найпопулярніших інструментів автоматизації клієнтської підтримки, оскільки дозволяють значно покращити взаємодію між компанією та клієнтами [4].

Основні функції чатботів у підтримці клієнтів включають відповіді на типові запитання, обробку замовлень і бронювань, персоналізовані консультації, а також передачу складних запитів живим операторам. Це допомагає значно знизити навантаження на службу підтримки та підвищити швидкість обслуговування [12].

Переваги використання чатботів у клієнтській підтримці:

– економія ресурсів за рахунок автоматизації рутинних процесів;

– доступність 24/7, що дозволяє клієнтам отримувати допомогу у будь-який час;
– швидкість реакції на запити, що підвищує задоволеність користувачів;
– масштабованість, що забезпечує одночасне обслуговування великої кількості користувачів.

При розробці чатботів важливо враховувати виклики, пов’язані з якісним розпізнаванням намірів користувачів, підтримкою природної взаємодії, інтеграцією з існуючими системами компанії, а також забезпеченням безпеки і конфіденційності даних [6].

Системи керування зверненнями, або Helpdesk, є важливим інструментом автоматизації клієнтської підтримки. Вони призначені для організації, відстеження та обробки звернень клієнтів, що надходять через різні канали комунікації – електронну пошту, телефон, соціальні мережі, месенджери тощо. Такі системи допомагають централізувати інформацію, контролювати процес вирішення запитів і підвищувати якість обслуговування.

Основні можливості систем Helpdesk включають автоматичне розподілення звернень між відповідальними співробітниками, ведення історії спілкування з клієнтом, а також формування звітності для аналізу ефективності підтримки [7].

Переваги використання систем керування зверненнями:

– покращення організації роботи служби підтримки та контроль за статусом запитів;
– зниження часу відповіді завдяки автоматичному розподілу запитів;
– можливість відстежувати та аналізувати якість обслуговування за допомогою звітів;
– підвищення прозорості процесу взаємодії з клієнтами.

Ефективна інтеграція Helpdesk-систем з іншими інструментами, такими як CRM, чатботи та бази знань, дозволяє побудувати комплексну систему підтримки клієнтів, яка підвищує задоволеність користувачів і оптимізує роботу компанії [13].

IVR (Interactive Voice Response) – це технологія, яка дозволяє абоненту взаємодіяти з телефонною системою за допомогою голосових команд або натискань клавіш на телефоні. Такі системи широко застосовуються у сфері обслуговування клієнтів для попереднього фільтрування запитів, надання інформації без участі оператора або маршрутизації дзвінків до відповідного відділу.

Голосові меню IVR допомагають автоматизувати рутинні процеси, такі як перевірка балансу, стану замовлення, розкладу або надання довідкової інформації. Це зменшує навантаження на операторів і дозволяє обслуговувати більшу кількість дзвінків одночасно.

Основні переваги автоматизованих голосових систем:

– зменшення часу очікування завдяки попередній обробці запитів;
– доступність інформації для клієнтів 24/7 без участі персоналу;
– ефективне перенаправлення дзвінків до відповідного спеціаліста;
– можливість обробки великої кількості дзвінків одночасно.

Сучасні IVR-системи можуть бути інтегровані з базами даних, CRM та іншими службами, що дозволяє персоналізувати відповіді залежно від номера телефону абонента або його історії звернень. Розвиток технологій розпізнавання голосу та ШІ сприяє поступовому переходу від простих IVR-меню до інтелектуальних голосових асистентів [6].

Системи управління знаннями (англ. Knowledge Base, КВ) – це спеціалізовані інформаційні ресурси, призначені для зберігання, структуризації та швидкого надання користувачам відповідей на найпоширеніші запитання. Такі системи часто використовуються як окремий інструмент самообслуговування або інтегруються з іншими компонентами підтримки – наприклад, чатботами, Helpdesk або IVR.

Головна мета КВ – зменшити кількість звернень до служби підтримки, надаючи клієнтам можливість самостійно знаходити відповіді на свої запитання. Добре організована база знань підвищує ефективність роботи компанії, економить час користувачів та знижує навантаження на персонал.

Переваги використання систем управління знаннями:

– доступ до інформації 24/7 без залучення фахівців;
– швидке вирішення типових проблем клієнтами самостійно;
– зменшення кількості повторних звернень до служби підтримки;
– підвищення якості обслуговування за рахунок єдиних стандартних відповідей.

Системи управління знаннями можуть бути інтегровані з аналітичними модулями для визначення найчастіших запитів, а також з інструментами машинного навчання для покращення пошуку й рекомендацій. У поєднанні з іншими засобами підтримки КВ дозволяють створити цілісну екосистему обслуговування клієнтів [9].

Серед розглянутих рішень особливу увагу було приділено чатботам, які поєднують у собі можливості швидкої відповіді, персоналізації та масштабованості. Вони демонструють високу ефективність у випадках, коли необхідна інтерактивна взаємодія з користувачем через вебінтерфейс або месенджер [11].

У межах проєкту було обрано реалізацію системи автоматизації клієнтської підтримки у вигляді чатбота, оскільки це забезпечує високу швидкість реагування, цілодобову доступність та просту інтеграцію з іншими сервісами.

Для реалізації рішення використовується сучасний стек технологій:

– React – для створення динамічного та зручного інтерфейсу користувача;
– Python – як основна мова серверної частини, що дозволяє гнучко реалізувати логіку роботи чатбота та взаємодію з бібліотеками обробки природної мови;
– PostgreSQL – як надійна база даних для зберігання інформації про користувачів, повідомлення та сценарії діалогів.

Цей вибір забезпечує баланс між продуктивністю, масштабованістю та зручністю розробки, що є ключовим для успішної реалізації системи автоматизованої клієнтської підтримки [13].

1.2 Аналіз фреймворку React та його можливостей

React.js – це бібліотека для JavaScript з відкритим вихідним кодом для розробки користувацьких інтерфейсів. Вона допомагає швидко і легко реалізувати реактивність – явище, коли у відповідь на зміну одного елемента змінюється все інше [1].

React використовують фронтенд-розробники. У моделі MVC, де Model-View-Controller – це назва, що розшифровується як Model-View-Controller, інтерфейс – це View, представлення, з яким взаємодіє користувач, та частина вебсайту або додатка, яка видна людині. Фахівці, які з нею працюють, зокрема, використовують React.js. Також з React можуть працювати верстальники, тестувальники та інші спеціалісти, задіяні у створенні вебінтерфейсів [1].

Основною функцією React є створення односторінкових і багатосторінкових додатків, розробки великих сайтів. Наприклад, за його допомогою написаний сервіс стрімінгу Netflix і реалізовані новинні стрічки найбільших соціальних мереж. Бібліотека призначена для:

- створення функціональних інтерактивних вебінтерфейсів, з якими працюють, не потрібно постійно оновлювати сторінку;
- швидкої та зручної реалізації окремих компонентів та сторінок в цілому – елементи в React легко використовувати повторно;
- легкої розробки складних програмних структур – їх просто описувати, якщо використовувати реалізований в React підхід;
- доопрацювання нової функціональності з будь-яким початковим набором технологій: бібліотека не залежить від решти інструментарію і працюватиме добре на будь-якому коді;
- розробки односторінкових і багатосторінкових додатків (SPA і PWA);
- роботи з серверною частиною сайту або розробки мобільних додатків – таких випадках React використовують разом із засобами, які адаптовують вебтехнології для інших цілей.

React має ряд особливостей, які роблять його гнучким та потужним інструментом. Ми розглянемо деякі з них [7].

React не є повноцінним фреймворком, а лише бібліотекою для створення користувацьких інтерфейсів. Важливо розуміти різницю між фреймворками та бібліотеками.

Таким чином, React надає інструменти для створення компонентів користувацького інтерфейсу, керування станом та взаємодії з DOM, але не нав’язує конкретну архітектуру додатка. Розробники можуть використовувати React у поєднанні з іншими бібліотеками та інструментами на свій розсуд, що робить його гнучким і дозволяє створювати різноманітні типи вебдодатків [6].

Реакт базується на компонентах – окремих елементах вебінтерфейсу. Компоненти інкапсульовані, що означає, що в кожному з них розміщені всі необхідні методи та дані. Детальніше про інкапсуляцію можна прочитати в нашій статті про ООП: це один з ключових критеріїв об’єктно-орієнтованого підходу. У випадку з компонентами Реакт інкапсуляція також означає, що стан елемента зберігається в ньому самому, а не в окремому об’єкті [2].

Компоненти можуть передавати свої властивості та дані один одному, але тільки в одному напрямку – від «батьківських» до «дочірніх». Це допомагає реалізувати чітку ієрархію та спрощує відлагодження. Однонапрямковий потік даних означає, що програміст завжди може зрозуміти, звідки саме до елементу надійшли дані [2].

Ще одна особливість потоку даних – відсутність можливості змінювати властивості безпосередньо. Для цього існують спеціальні зворотні виклики (callback-функції). Так називаються частини виконуваного коду, які передаються від одного компонента до іншого у вигляді параметра функції. Це дозволяє підтримувати стабільність: властивості залишаються незмінними після створення, а callback-функція, мовляв, перестворює їх. В результаті використання колбеків потік даних виглядає так: властивості та інформація передаються зверху вниз, а події відбуваються знизу вгору [2].

У старій версії React, керування станами зазвичай відбувалося за допомогою класів – спеціальних конструкцій, про які можна дізнатися більше

в статті про ООП. Зараз в React.js існує підтримка хуків – так називаються спеціальні функції-«крючки», які «підключаються» до стану елемента або до методу. Зміна стану або виклик методу «тягне» за собою ці функції, і вони автоматично виконуються – це допомагає уникнути використання класів, полегшує і спрощує написання коду [6].

У React існують вбудовані хуки, але розробник може створювати й власні – це допомагає повторно використовувати код і зменшити його кількість [8].

React Developer Tools є незамінним інструментом для фронтенд-розробників, оскільки вони дозволяють ефективно відлагоджувати та аналізувати React-додатки безпосередньо в браузері.

Розглянемо переваги React.js:

– швидкість рендерингу;
– компонентна архітектура;
– односторонній потік даних;
– велика спільнота та екосистема;
– декларативний синтаксис.

Не зважаючи на всі переваги, React.js має свої недоліки:

– великий поріг входження;
– незручність при роботі з формами;
– необов’язкове використання JSX;
– потреба в додаткових бібліотеках для повноцінного SPA;
– проблеми з SEO.

React.js є потужним та гнучким інструментом для створення сучасних вебінтерфейсів. Завдяки компонентному підходу, підтримці хуків, зручному керуванню станом і активній спільноті, ця бібліотека дозволяє ефективно будувати як прості односторінкові додатки, так і складні багатокомпонентні системи [3].

Незважаючи на деякі недоліки, такі як складність початкового навчання чи потреба у сторонніх бібліотеках, React залишається одним із найпопулярніших рішень для фронтенд-розробки. Саме тому він був обраний

як ключовий інструмент для реалізації клієнтської частини у межах нашого проєкту чатбота, що забезпечує інтерактивну взаємодію з користувачем у режимі реального часу [3].

1.3 Все про Python (FastAPI) його переваги та недоліки

Python – це одна з найпопулярніших мов програмування у світі, яка вирізняється простотою, лаконічністю синтаксису та великим обсягом доступних бібліотек і фреймворків. Його популярність зумовлена легкістю навчання, гнучкістю та активною спільнотою. Python використовується у багатьох галузях: від веброзробки й автоматизації до штучного інтелекту, обробки природної мови (NLP) та роботи з великими даними [4].

Особливістю Python є його динамічна типізація, підтримка об’єктно-орієнтованого, процедурного та функціонального стилів програмування, а також велика кількість пакетів, що полегшують вирішення типових задач (наприклад, requests, sqlalchemy, numpy, pandas, tensorflow, nltk тощо). Завдяки підтримці багатьох парадигм Python дозволяє швидко створювати прототипи, а потім масштабувати їх до повноцінних додатків [4].

FastAPI, у свою чергу, є легким, але потужним вебфреймворком нового покоління, розробленим спеціально для побудови високопродуктивних API. Він спроєктований з урахуванням останніх можливостей Python 3.6+ – зокрема, системи типізацій, асинхронного виконання коду та інтеграції з бібліотеками, такими як Pydantic для перевірки даних і Uvicorn для запуску серверів [5].

Ключові особливості FastAPI, які виділяють його серед інших вебфреймворків:

– строга типізація: використання анотацій типів (type hints) дозволяє інтегрувати автогенерацію документації, статичну перевірку коду, підвищити надійність і зручність розробки;

- Pydantic-моделі: FastAPI використовує цю бібліотеку для перевірки, серіалізації та десеріалізації вхідних і вихідних даних (це зменшує ризик помилок і полегшує читання коду);
- асинхронна обробка: підтримка async/await забезпечує неблокуюче виконання запитів, що критично важливо для масштабованих сервісів і мікросервісної архітектури;
- підтримка OAuth2, JWT та інших механізмів авторизації: дозволяє створювати безпечні API без надмірної складності;
- готові OpenAPI-специфікації: автоматичне створення Swagger-документації, що ідеально підходить для командної роботи, інтеграцій та тестування API.

FastAPI зберігає чисту архітектуру, а також дозволяє легко поєднуватися з іншими компонентами Python-екосистеми: ORM (наприклад, SQLAlchemy), бібліотеками для обробки черг (Celery), кешування (Redis), логування (Loguru) та іншими [10].

У поєднанні з PostgreSQL як основною базою даних FastAPI забезпечує надійну, масштабовану та високопродуктивну серверну частину, яка здатна обслуговувати велику кількість клієнтів одночасно – саме те, що необхідно для ефективної роботи чатбота у режимі реального часу [6].

Для реалізації серверної частини чатбота в межах проєкту було обрано FastAPI – сучасний асинхронний вебфреймворк для створення API на Python. Він був розроблений для швидкої побудови високопродуктивних REST-сервісів, з урахуванням найкращих практик сучасної розробки. FastAPI базується на стандарті OpenAPI та використовує типізацію з мови Python 3.6+, що дозволяє автоматично генерувати документацію до API та забезпечує високу зручність у підтримці коду [18].

Основні переваги використання FastAPI:

- висока швидкодія: FastAPI побудований на основі бібліотеки Starlette і асинхронної моделі виконання, що дозволяє обробляти велику кількість запитів паралельно; за швидкістю він порівнюваний із Node.js і Go;

- автоматична генерація документації: завдяки інтеграції з OpenAPI, документація до API створюється автоматично та доступна за спеціальними маршрутами (/docs, /redoc);
- підтримка асинхронного програмування: FastAPI дозволяє використовувати async/await, що особливо важливо при роботі з базами даних та зовнішніми API;
- інтеграція з Pydantic: схеми запитів і відповідей створюються на основі типізованих моделей Pydantic, що дозволяє валідувати дані автоматично та зручно;
- швидка розробка та підтримка: завдяки простому синтаксису та автоматизованим перевіркам даних знижується кількість помилок і підвищується продуктивність розробника;
- гнучкість: підходить як для невеликих проєктів, так і для складних мікросервісних архітектур.

Не зважаючи на свої переваги, FastAPI має такі недоліки:

- менш розвинена екосистема у порівнянні з класичними фреймворками, такими як Django або Flask (деякі бібліотеки або рішення потребують додаткової інтеграції);
- вища складність для новачків у порівнянні з Flask, особливо через використання асинхронності та типізацій;
- обмежена кількість матеріалів та туторіалів українською мовами, порівняно з іншими популярними інструментами.

FastAPI активно розвивається та вже здобув популярність серед розробників завдяки своїй продуктивності, простоті та сучасному підходу до створення API.

У межах нашого проєкту цей фреймворк дозволяє ефективно реалізувати серверну частину чатбота, обробляти запити користувачів, взаємодіяти з базою даних та забезпечувати швидкий обмін даними з фронтендом, створеним на React [18].

1.4 Все про PostgreSQL

PostgreSQL – це потужна об’єктно-реляційна система керування базами даних (СУБД) з відкритим кодом, яка широко застосовується у різних сферах. Вона підтримує більшість стандартів SQL і забезпечує гнучку роботу з великими обсягами даних, складними запитами та багаторівневими транзакціями [7].

Ця СУБД відрізняється високою надійністю та стійкістю до збоїв завдяки підтримці транзакцій за стандартом ACID та механізму журналювання змін (Write Ahead Logging, WAL). PostgreSQL також є розширюваною системою, що дозволяє користувачам додавати власні типи даних, функції та оператори. Серед численних розширень варто виділити PostGIS для роботи з геопросторовими даними, pg_trgm для пошуку за схожістю, а також інші модулі, які значно розширюють функціональність бази [7].

Однією з важливих переваг PostgreSQL є підтримка роботи з JSON та іншими типами напівструктурованих даних, що дозволяє поєднувати можливості реляційних і NoSQL баз даних у межах одного проєкту. Також СУБД забезпечує широкі можливості масштабування, включно з реплікацією (streaming replication, logical replication) та паралельною обробкою запитів.

У нашому проєкті чатбота PostgreSQL обрано як основну систему зберігання даних через її високу продуктивність, надійність, гнучкість у роботі зі складними зв’язками між таблицями та можливість зберігати як структуровані, так і напівструктуровані дані. Вона також забезпечує безпеку й ефективне адміністрування, що є критичним для роботи з конфіденційною інформацією користувачів [8].

В даний час промислові виробники також використовують PostgreSQL щоб прискорити їхній загальний бізнес-процес. Це також допомагає їм оптимізувати продуктивність ланцюга постачання, використовуючи цю СУБД з відкритим вихідним кодом як систему зберігання. Це дозволяє компаніям знизити витрати на функціонування свого бізнесу [9].

PostgreSQL пропонує потужну ГІС, яка називається “PostGIS”. Це розширення надає сотні функцій для обробки геометричних даних у різних форматах. PostGIS повністю відповідає стандартам. Крім того, використовуючи як QGIS, так і GeoServer, спільнота Open Source забезпечує найпростіший спосіб обробки геоданих [13].

Переваги PostgreSQL:

- відкрите програмне забезпечення без ліцензійних обмежень;
- висока надійність та стійкість до збоїв;
- розширюваність: можливість додавання власних типів даних та функцій;
- підтримка роботи з JSON та іншими напівструктурованими даними;
- масштабованість завдяки реплікації та паралельній обробці запитів;
- розвинена система безпеки та контролю доступу;
- широка підтримка стандарту SQL.

Недоліки PostgreSQL:

- відносна складність налаштування та адміністрування;
- порівняно менше графічних інтерфейсів у порівнянні з іншими СУБД;
- можлива надмірність функціоналу для дуже простих проєктів;
- потреба в тонкому налаштуванні під великі навантаження.

На завершення, PostgreSQL є потужною та надійною системою керування базами даних, яка ідеально підходить для реалізації складних та масштабованих проєктів, таких як наш чатбот для автоматизації клієнтської підтримки. Завдяки широкому функціоналу, гнучкості та стабільності, ця СУБД забезпечує ефективне зберігання і обробку даних, що є ключовими факторами для якісної роботи системи. Вибір PostgreSQL дозволяє гарантувати безпеку, масштабованість і зручність подальшого розвитку проєкту [18].

1.5 Порівняння з альтернативними технологіями та платформами

Порівняння FastAPI, PostgreSQL та React з альтернативними технологіями дозволяє краще оцінити їхні переваги та обмеження в контексті різних типів проєктів [17].

FastAPI вирізняється серед Python-фреймворків завдяки своїй продуктивності та сучасному підходу до створення API. Він підтримує асинхронне програмування (async/await), автоматично генерує документацію через OpenAPI та використовує типізацію Python для валідації даних, що підвищує надійність і швидкість розробки. У порівнянні з ним, Flask є легким мікрофреймворком, зручним для невеликих додатків, але без вбудованої асинхронності та менш оптимізований для високих навантажень. Django, навпаки, пропонує повноцінну екосистему для складних проєктів, однак може бути надмірним для побудови простих API і не настільки ефективним у контексті асинхронності, як FastAPI [14].

У сфері баз даних PostgreSQL демонструє перевагу над MySQL завдяки підтримці складних запитів, транзакційності (ACID), розширюваності та зручній роботі з JSON-даними. Вона ідеально підходить для аналітичних задач і великих обсягів інформації. MySQL, своєю чергою, частіше використовується в проєктах, де важлива швидкість при простих запитах і менше значення мають складні аналітичні операції [16].

Порівнюючи React із Angular та Vue.js, можна сказати, що React є бібліотекою, яка надає гнучкий компонентний підхід до створення UI, має велику екосистему і є відносно легкою для вивчення. Проте для створення великих додатків може знадобитися додаткове налаштування. Angular – це повноцінний фреймворк із підтримкою всього життєвого циклу застосунку, побудований на TypeScript, але він складніший у засвоєнні. Vue.js займає проміжну позицію – простий у використанні, зручний для швидкого старту та має вбудовані інструменти, хоч і менш розвинену екосистему порівняно з React та Angular [10].

У порівнянні з Node.js (Express.js), FastAPI виграє завдяки вбудованій типізації, підтримці асинхронності та автоматичній генерації документації. Express.js, хоч і є популярною платформою на JavaScript, не має вбудованих засобів для валідації даних або створення документації, що потребує додаткових рішень [15].

Щодо зберігання даних, PostgreSQL перевершує NoSQL-рішення, як-от MongoDB, коли мова йде про реляційні структури, складні транзакції та запити. Водночас MongoDB є чудовим вибором для додатків, які працюють з неструктурованими або змінними даними, завдяки гнучкості документно-орієнтованої моделі та хорошій масштабованості.

Загалом вибір між цими технологіями залежить від цілей проєкту. FastAPI краще підходить для створення високопродуктивних API, які потребують асинхронності та автоматизації документації. PostgreSQL ідеально підходить для проєктів із великими обсягами структурованих даних і високими вимогами до надійності. React забезпечує гнучкість у побудові користувацького інтерфейсу, тоді як Angular і Vue.js більше підходять для великих, структурованих клієнтських додатків [14].

1.6 Висновки до розділу 1

У першому розділі проведено аналіз сучасних систем автоматизації клієнтської підтримки, зокрема таких як чатботи, системи керування зверненнями (Helpdesk), автоматизовані голосові системи (IVR) та системи управління знаннями (Knowledge Base). Вивчено їхні основні функції, переваги та недоліки, що дозволило окреслити актуальні вимоги до автоматизованих сервісів підтримки.

Обґрунтовано вибір технологічного стеку для розробки чатбота: React як інструмент для створення динамічного та зручного користувацького інтерфейсу, Python (зокрема FastAPI) для реалізації серверної логіки і обробки

запитів, а також PostgreSQL як надійну та масштабовану систему керування базою даних. Цей стек забезпечує високу продуктивність, гнучкість у розробці та можливість ефективної інтеграції з різними сервісами.

Також розглянуто ключові характеристики обраних технологій, їхні сильні та слабкі сторони, що дозволило створити комплексне уявлення про можливості і обмеження обраного підходу. Це стало підґрунтям для подальшої розробки чатбота, орієнтованого на швидке реагування, доступність 24/7 і зручність використання для кінцевих користувачів.

2 ПРОЄКТУВАННЯ ТА АНАЛІЗ ВИМОГ ДО ОНЛАЙН ЧАТУ

2.1 Опис предметної області

Опис поставленої задачі полягає в розробці вебдодатку, який реалізує певні функціональні можливості для користувачів, зокрема, забезпечує інтерактивний інтерфейс для взаємодії з даними, візуалізації, а також взаємодії з серверною частиною через API.

Основною метою є створення високопродуктивної та зручної системи для ефективного управління даними та забезпечення безпеки при роботі з ними.

Задача передбачає використання таких технологій, як React для створення інтерактивного користувацького інтерфейсу, FastAPI для розробки серверної частини з високою продуктивністю, а також PostgreSQL для збереження та обробки даних.

Детальніше, основні цілі та функціональні вимоги задачі включають:

- **розробка вебінтерфейсу**: створення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів, який дозволяє взаємодіяти з даними через різноманітні елементи управління (форми введення, списки, таблиці, графіки, фільтри тощо);
- **API для взаємодії з базою даних**: розробка серверної частини з використанням FastAPI для обробки запитів до бази даних, включаючи операції читання, запису, оновлення та видалення даних;
- **забезпечення масштабованості та безпеки**: створення системи, яка здатна масштабуватися при збільшенні навантаження, а також має захищений доступ до даних (автентифікація, шифрування тощо);
- **робота з даними в PostgreSQL**: зберігання та обробка даних у реляційній базі даних PostgreSQL, з урахуванням нормалізації даних, підтримки транзакцій та складних запитів;

– **інтеграція з іншими системами**: можливість інтеграції з зовнішніми сервісами через API для отримання додаткових даних або для виконання специфічних операцій.

Завдання також передбачає забезпечення високої ефективності додатку, зокрема через використання асинхронних операцій у FastAPI, що дозволяє обробляти велику кількість запитів одночасно [13].

2.2 Функціональні вимоги та нефункціональні вимоги

Перед початком розробки програмного продукту, проаналізувавши переваги та недоліки обраного інструментарі, доцільно провести детальний аналіз вимог до майбутньої системи. Це дозволяє чітко окреслити межі й функціональні можливості чатбота, сформулювати цілі проєкту, а також встановити критерії успішності його впровадження. Такий підхід забезпечує відповідність кінцевого рішення очікуванням користувачів, знижує ризики помилок на етапі реалізації й створює стабільне підґрунтя для подальшого технічного проєктування.

Функціональні вимоги визначають перелік основних дій, які має виконувати система чатбота у процесі роботи. У рамках нашого проєкту функціональність включає:

– авторизацію та реєстрацію користувачів з урахуванням ролей (адміністратор, оператор, користувач);
– обробку текстових запитів користувача з використанням Natural Language Processing (NLP);
– надання відповідей чатботом згідно зі сценаріями або базою знань;
– можливість переадресації діалогу оператору за потреби;
– збереження історії розмов для подальшого аналізу;
– підтримку інтеграцій із зовнішніми системами (наприклад, CRM або месенджери);
– підтримку push-сповіщень про нові відповіді або події;

– модуль адміністрування для керування сценаріями, діалогами та користувачами;
– ведення статистики щодо звернень, активності користувачів та ефективності відповідей;
– підтримку багатомовності чатбота.

Система повинна дозволяти створювати гнучкі сценарії обробки запитів, забезпечувати адаптацію до різних типів користувацьких звернень і легко інтегруватися з наявними сервісами компанії [12].

Нефункціональні вимоги описують якісні характеристики, що визначають надійність, продуктивність, зручність та безпеку роботи системи:

– продуктивність: забезпечення швидкої та ефективної обробки запитів користувача чатботом;
– масштабованість: можливість обробки зростаючої кількості діалогів без втрати продуктивності;
– надійність: забезпечення доступності сервісу на рівні не нижче 99.5%;
– безпека: шифрування даних, аутентифікація користувачів, журналювання дій;
– адаптивність: коректна робота на мобільних пристроях, планшетах і десктопах;
– зручність: інтуїтивно зрозумілий інтерфейс для користувача та адміністратора;
– відновлюваність: резервне копіювання даних та механізми їх відновлення;
– інтернаціоналізація: підтримка кількох мов інтерфейсу та діалогу.

Сучасний чатбот повинен не лише ефективно відповідати на запити, а й бути надійним та безпечним інструментом комунікації.

Дотримання наведених вимог гарантує високу якість кінцевого продукту, його стабільну роботу у реальних умовах і готовність до масштабування відповідно до зростання навантаження або нових бізнес-

потреб. Отриманий список вимог є основою для побудови архітектури системи та її реалізації на наступних етапах розробки [11].

Інтерфейс користувача має відповідати сучасним стандартам UX/UI-дизайну, забезпечуючи комфортну взаємодію як для кінцевих користувачів, що взаємодіють з чатботом через мобільні пристрої чи браузер, так і для операторів або адміністраторів, які керують сценаріями чи переглядають історію діалогів. Простота у користуванні, зрозуміла навігація та адаптивність інтерфейсу – ключові чинники для досягнення високої якості користувацького досвіду.

Таким чином, проведений аналіз функціональних та нефункціональних вимог до системи чатбота сформував чітку основу для подальшого етапу – детального проєктування та реалізації архітектури програмного продукту, яка буде відповідати очікуванням цільової аудиторії, вимогам безпеки та масштабованості.

2.3 Інтерфейсні вимоги

Інтерфейсні вимоги до програмного продукту «Розробка чатботу для автоматизації підтримки клієнтів з використанням React, Python та PostgreSQL» визначають загальні принципи, яких слід дотримуватись при створенні користувацького інтерфейсу, щоб забезпечити зручну, зрозумілу та ефективну взаємодію між користувачем і системою.

Чатбот є сервісом, який має працювати в режимі реального часу та підтримувати швидку реакцію на запити користувачів. Тому інтерфейс має бути інтуїтивно зрозумілим, адаптивним та мінімалістичним, не перевантаженим зайвими елементами.

Основні інтерфейсні вимоги до клієнтської частини включають:

– підтримка адаптивного дизайну – забезпечення коректного відображення інтерфейсу на ПК, планшетах та смартфонах;

– чітка структура вікна чату – відображення повідомлень, кнопок швидкої відповіді, завантажених файлів та індикатора активності бота;
– наявність форм авторизації та реєстрації – мінімум полів для зручності, валідація введених даних, повідомлення про помилки;
– система навігації – зручне перемикання між основними сторінками (чат, профіль, підтримка, адміністрування);
– механізми сповіщень – візуальні або звукові повідомлення про нові відповіді від чатбота;
– доступність інтерфейсу – підтримка контрастного режиму, масштабування шрифтів, навігація з клавіатури для користувачів з обмеженими можливостями;
– локалізація – підтримка щонайменше двох мов інтерфейсу (наприклад, українська та англійська) з можливістю легкої зміни мови користувачем;
– сучасний дизайн – дотримання принципів UX/UI-дизайну з використанням анімованих переходів, іконок та зрозумілих підказок.

Для адміністративного інтерфейсу передбачені окремі вимоги:

– панель управління чатботом – з можливістю перегляду журналу звернень, аналітики, керування шаблонними відповідями;
– модуль сценаріїв – створення та редагування послідовностей дій бота у вигляді блок-схем або таблиць;
– управління користувачами – призначення ролей, блокування або активація акаунтів.

Окрему увагу слід приділити простоті взаємодії, особливо у випадках, коли користувач вперше взаємодіє з ботом. Повідомлення повинні мати логічну структуру, а всі елементи – зрозумілі підписи та дії. Важливо також дотримуватись єдиного стилю оформлення компонентів інтерфейсу (кольорова палітра, шрифти, кнопки), що сприяє створенню позитивного досвіду користувача. У таблиці 2.1 проаналізуємо додаткові деталі нашого інтерфейсу, щоб створити повну картину.

Таблиця 2.1 – Основні екрани та функціональність інтерфейсу чатбот

Назва сторінки	**Опис функціональності**
Головна сторінка	Вітання користувача, доступ до чату, коротка інструкція або підказки
Чат (ChatWindow)	Основний блок спілкування з ботом: історія діалогу, введення запитів, кнопки дій
Авторизація / Реєстрація	Вхід до системи через email/пароль, реєстрація нового користувача, обробка помилок вводу
Профіль користувача	Налаштування особистих даних, вибір мови інтерфейсу, перегляд історії звернень
Адмін-панель	Інструменти керування ботом, сценаріями, шаблонами відповідей, аналітика звернень
Сторінка помилки (404 / 500)	Повідомлення про відсутність сторінки або помилку сервера з рекомендаціями дій

Взаємодія між користувачем та інтерфейсом має бути логічною та передбачуваною. Усі події – від введення тексту до надсилання повідомлення – мають супроводжуватись візуальним підтвердженням (анімації, індикатори завантаження, спливаючі повідомлення).

Особливу увагу приділено реакції інтерфейсу на різні ситуації:

– якщо бот не може надати відповідь – виводиться ввічливе повідомлення із пропозицією звернутись до оператора;
– у разі втрати з’єднання – показується сповіщення з автоматичними спробами повторного підключення;
– при довготривалому очікуванні – відображається анімація або статус очікування.

Успішна реалізація інтерфейсних вимог є основою для забезпечення зручної взаємодії користувача з чатботом. Чітка структура, адаптивний дизайн, підтримка декількох мов, інтуїтивне управління та швидкий відгук інтерфейсу – це ключові характеристики, що безпосередньо впливають на

рівень задоволеності користувача та ефективність використання системи підтримки.

Правильне проєктування інтерфейсу дозволяє мінімізувати навчальну криву, скоротити час вирішення проблем клієнтів та підвищити якість обслуговування загалом.

2.4 Проєктування фронтендної частини

Проєктування фронтендної частини чатбота є ключовим етапом розробки, адже саме від неї залежить зручність та ефективність взаємодії користувачів із системою. Основною метою фронтендної частини є створення інтуїтивно зрозумілого та адаптивного інтерфейсу, який забезпечить швидкий та комфортний доступ до функціоналу чатбота як із десктопних, так і з мобільних пристроїв.

Для реалізації інтерфейсу було обрано React – сучасну JavaScript-бібліотеку, що дозволяє ефективно створювати компоненти та управляти станом додатка. Компонентний підхід React сприяє модульності, полегшує підтримку і масштабування проєкту. Основні компоненти фронтенду включатимуть:

- вікно чату – основний інтерфейс для спілкування користувача з чатботом, що підтримує відображення текстових повідомлень, кнопок та мультимедійних елементів;
- система авторизації та реєстрації – модулі, які забезпечують безпечний вхід користувачів у систему;
- панель налаштувань – для керування особистими даними користувача та налаштувань взаємодії з чатботом;
- адміністративна панель – для операторів та адміністраторів, де вони можуть переглядати статистику, керувати сценаріями та переглядати історію звернень.

Особлива увага при розробці фронтенду приділяється адаптивності інтерфейсу. Це означає, що всі елементи коректно відображатимуться та будуть зручними у використанні на різних пристроях – від великих екранів ПК до смартфонів і планшетів.

Для забезпечення швидкої реакції додатка використовується механізм управління станом (наприклад, React Context або Redux), що дозволяє оптимізувати оновлення інтерфейсу та уникнути зайвих перерендерів.

На рисунку 2.1 зображено детальну структуру клієнтської частини та її основні компоненти.

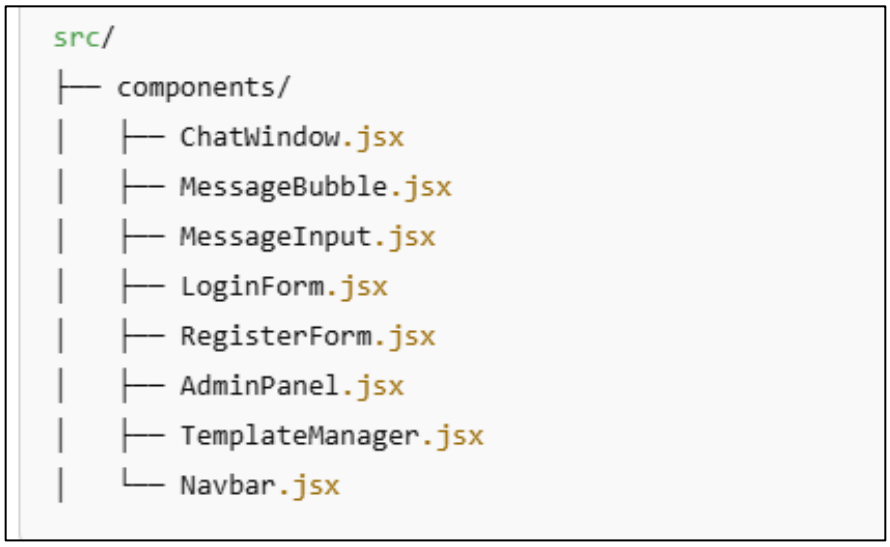


Рисунок 2.1 – Структура клієнтської частини

Розберемо основні компоненти:

– ChatWindow – відображає діалог між користувачем і ботом;
– MessageBubble – представлення окремого повідомлення;
– MessageInput – поле для вводу тексту, кнопка «Надіслати»;
– LoginForm / RegisterForm – форма авторизації та реєстрації;
– AdminPanel – доступна лише адміністратору, керування шаблонами та перегляд логів;
– Navbar – меню навігації між сторінками.

Також для біль детального розуміння на рисунку 2.2 представлена діаграма класів, щоб побачити взаємодії компонентів між собою.

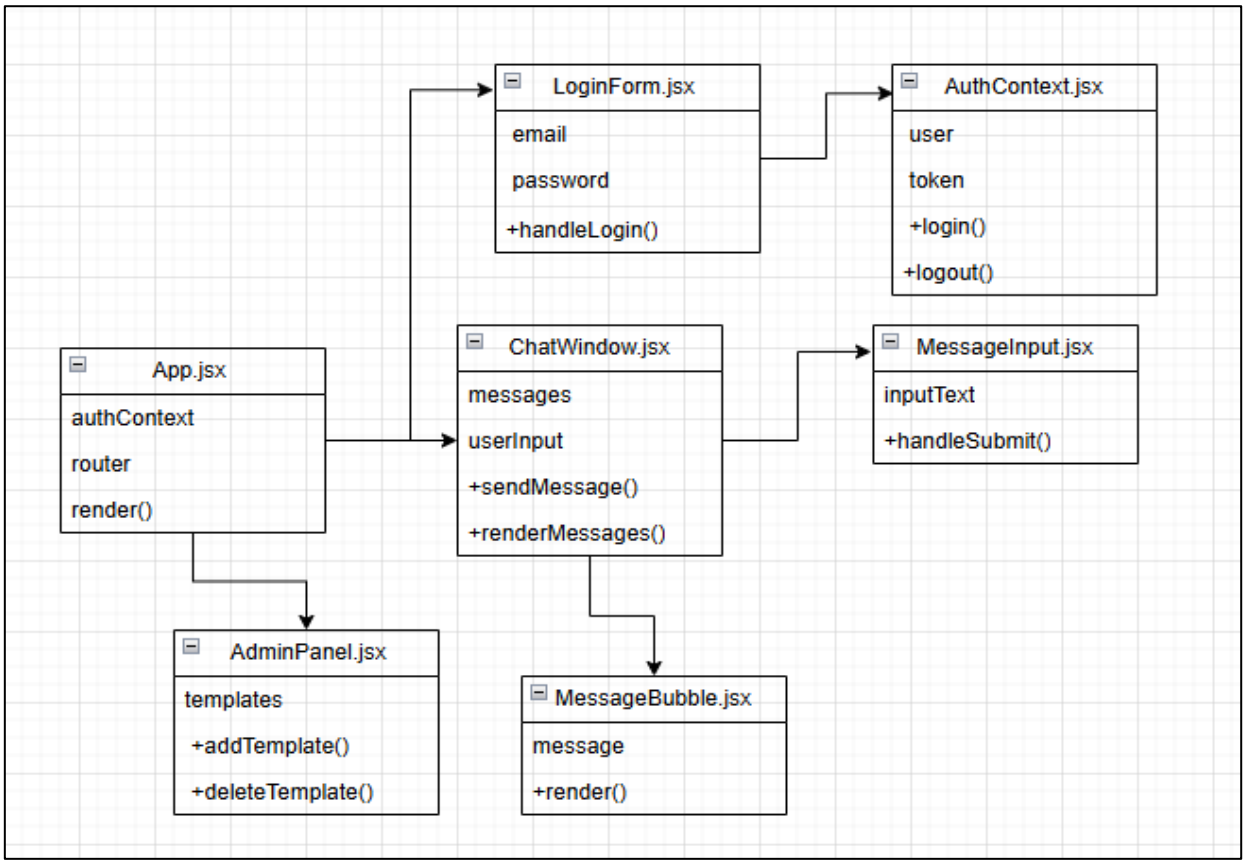


Рисунок 2.2 – Діаграма класів фронтендної частини

Розберемо логіку взаємодії між користувачем та чатботом:

– перевіряється токен у localStorage;
– якщо токен є – користувач автоматично входить;
– користувач вводить текст → компонент MessageInput передає його до ChatWindow;
– ChatWindow викликає sendMessage() із api.js, надсилає запит на бекенд;
– отримана відповідь додається до масиву messages, і інтерфейс оновлюється;
– через форми LoginForm та RegisterForm;
– токен автентифікації зберігається у localStorage;
– через роль, збережену в токені, користувач перенаправляється до AdminPage;
– через TemplateManager адміністратор керує шаблонними відповідями.

Крім основних візуальних компонентів, важливу роль у фронтендній архітектурі відіграють служби для взаємодії з API. Наприклад, модуль api.js містить усі методи HTTP-запитів до бекенд-сервера (отримання відповідей від бота, автентифікація, реєстрація, збереження повідомлень, доступ до шаблонів відповідей тощо). Це дозволяє централізовано керувати всіма точками зв’язку між клієнтом і сервером, що покращує масштабованість та тестованість коду.

Для управління маршрутизацією між сторінками використовується бібліотека React Router, яка забезпечує плавну навігацію між такими маршрутами, як /chat, /login, /register, /admin. Кожен маршрут обробляє відповідний компонент, перевіряючи наявність авторизації, ролі користувача та інші умови доступу.

Також застосовується система контролю доступу на клієнтському рівні, яка перевіряє роль користувача (адміністратор чи звичайний клієнт) перед завантаженням певних компонентів. Це підвищує безпеку та забезпечує правильний розподіл функціоналу між користувачами.

Ще один важливий аспект – локалізація. У системі передбачена можливість зміни мови інтерфейсу. Для цього використовується бібліотека i18next, що дозволяє легко додавати нові мовні файли без зміни коду інтерфейсу.

Вся текстова інформація винесена у словники, що спрощує супровід багатомовного інтерфейсу.

У якості інструментів розробки й автоматизації процесів використовуються:

- Vite або Webpack – для збірки та оптимізації проєкту;
- ESLint та Prettier – для забезпечення єдиного стилю коду;
- Jest / React Testing Library – для модульного тестування основних компонентів;
- Axios – як клієнт HTTP-запитів;
- Tailwind CSS або Material UI – для швидкої верстки адаптивного та сучасного інтерфейсу.

Таким чином, фронтендна частина чатбота реалізована за сучасними стандартами веброзробки, із дотриманням принципів модульності, масштабованості та зручності використання.

Такий підхід дозволяє забезпечити якісну взаємодію кінцевих користувачів із системою, гнучкість у подальшому розвитку проєкту та швидку адаптацію до нових вимог бізнесу.

2.5 Налаштування серверної частини та запитів АПІ

Серверна частина реалізована за допомогою мови програмування Python з використанням фреймворку FastAPI (або Flask, залежно від вибору). Вона виконує роль основного посередника між клієнтом (React-додатком) та базою даних (PostgreSQL), а також забезпечує логіку обробки повідомлень чатбота.

Основні функції серверної частини:

– обробка REST API-запитів із фронтенду;
– реєстрація та автентифікація користувачів (JWT);
– збереження та отримання історії діалогів;
– надання відповідей чатбота (через шаблони або NLP-обробку);
– розмежування доступу для ролей (користувач / адміністратор);
– керування шаблонами відповідей (CRUD);
– логування дій для адміністрування.

На рисунку 2.3 зображено детальну структуру серверної частини та її основні компоненти.

Основні компоненти бекенду:

– користувач надсилає логін/пароль → отримає токен (JWT);
– токен додається до заголовків запитів;
– захищені маршрути перевіряють роль та ідентифікацію;
– користувач надсилає текст → API аналізує його;
– якщо ключові слова – повертає шаблонну відповідь;
– інакше – (опціонально) передає на NLP-модуль.

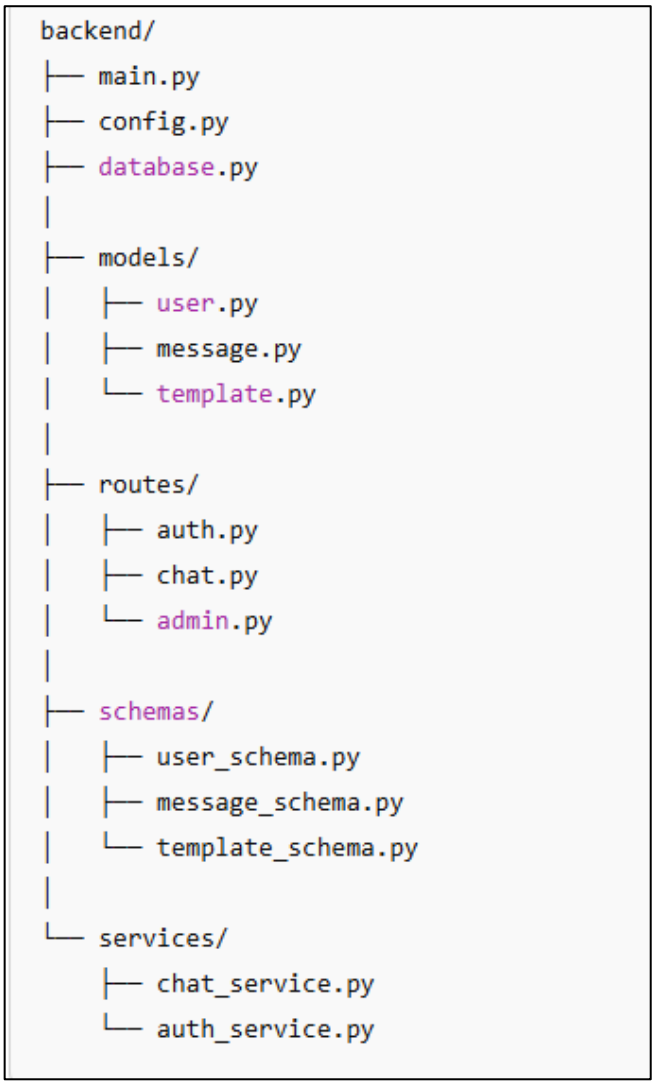


Рисунок 2.3 – Структура серверної частини

Серверна частина реалізована за принципами REST-архітектури. У таблиці 2.2 зображені функціональні можливості нашої архітектури, які представлені у вигляді набору ендпоінтів, згрупованих за логікою взаємодії.

Таблиця 2.2 – Основні API-ендпоінти

Метод	**Шлях**	**Опис функціональності**	**Роль доступу**
POST	/auth/register	Реєстрація нового користувача	Публічний
POST	/auth/login	Авторизація, видача JWT-токена	Публічний
GET	/chat/history	Отримання історії повідомлень	Авторизований користувач

Продовження таблиці 2.2

Метод	**Шлях**	**Опис функціональності**	**Роль доступу**
POST	/chat/message	Надсилання повідомлення боту	Авторизований користувач
GET	/admin/templates	Перегляд шаблонів відповідей	Адміністратор
POST	/admin/templates	Додавання нового шаблону	Адміністратор
PUT	/admin/templates/{id}	Оновлення шаблону	Адміністратор
DELETE	/admin/templates/{id}	Видалення шаблону	Адміністратор
GET	/admin/logs	Перегляд журналу подій	Адміністратор

Окрім вищезазначених компонентів, серверна частина, реалізована за допомогою FastAPI, має низку додаткових переваг, що впливають на якість та продуктивність системи:

- висока швидкодія – завдяки ASGI FastAPI ефективно обробляє велику кількість одночасних запитів, що критично для чатботів;
- автоматична документація – підтримка OpenAPI дозволяє генерувати Swagger UI / ReDoc без додаткових налаштувань;
- типізація та валідація – бібліотека Pydantic забезпечує перевірку даних і зменшує кількість помилок;
- інтеграція з Python-екосистемою – легко працює з SQLAlchemy, OAuth2, NLP-модулями тощо;
- модульна структура – логічний поділ на auth, chat, admin, database підвищує підтримуваність і масштабованість.

FastAPI є ефективним інструментом для побудови сучасного високопродуктивного вебсервісу, який легко масштабується, має потужну систему валідації, документації та підтримує асинхронну обробку запитів. Завдяки модульному підходу серверна частина чатбота легко підтримується, розширюється та забезпечує необхідну безпеку й стабільність у роботі з клієнтським застосунком та базою даних PostgreSQL.

2.6 Інтеграція бази даних

У проєкті використовується реляційна система управління базами даних PostgreSQL, яка забезпечує збереження, обробку та доступ до даних, пов’язаних із користувачами, повідомленнями, шаблонами відповідей та логами дій.

База даних є критично важливою частиною проєкту, оскільки вона виконує наступні функції:

– зберігає інформацію про користувачів (реєстраційні дані, ролі);
– фіксує кожне повідомлення та відповідь чатбота;
– містить шаблонні відповіді, які використовує чатбот;
– реєструє дії користувачів та адміністраторів (логи).

Для взаємодії з базою даних використовується ORM-бібліотека SQLAlchemy, яка дозволяє визначати структури таблиць через класи Python, а також легко виконувати CRUD-операції. Підключення до бази здійснюється за допомогою асинхронного драйвера asyncpg, що дозволяє інтегрувати PostgreSQL із FastAPI у повністю асинхронному режимі.

На рисунку 2.4 зображено ER-діаграма бази даних та з’язки між компонентами.

Опис структури бази даних чатбот-системи охоплює чотири основні таблиці, кожна з яких відповідає за окремий аспект функціонування застосунку.

Таблиця users (користувачі) зберігає інформацію про всіх зареєстрованих користувачів, включно з адміністраторами. Вона містить такі поля: id – унікальний ідентифікатор користувача (автоінкремент), name – ім’я користувача (до 100 символів), email – унікальна електронна адреса (до 100 символів), password_hash – хешований пароль, role – роль користувача (“user” або “admin”), та created_at – дата реєстрації.

Таблиця messages (повідомлення) відповідає за збереження усіх повідомлень, які надсилають користувачі, а також відповідей, сформованих

чатботом. До її складу входять: id, user_id (зовнішній ключ на таблицю users), text (вхідне повідомлення), response (відповідь чатбота) та timestamp.

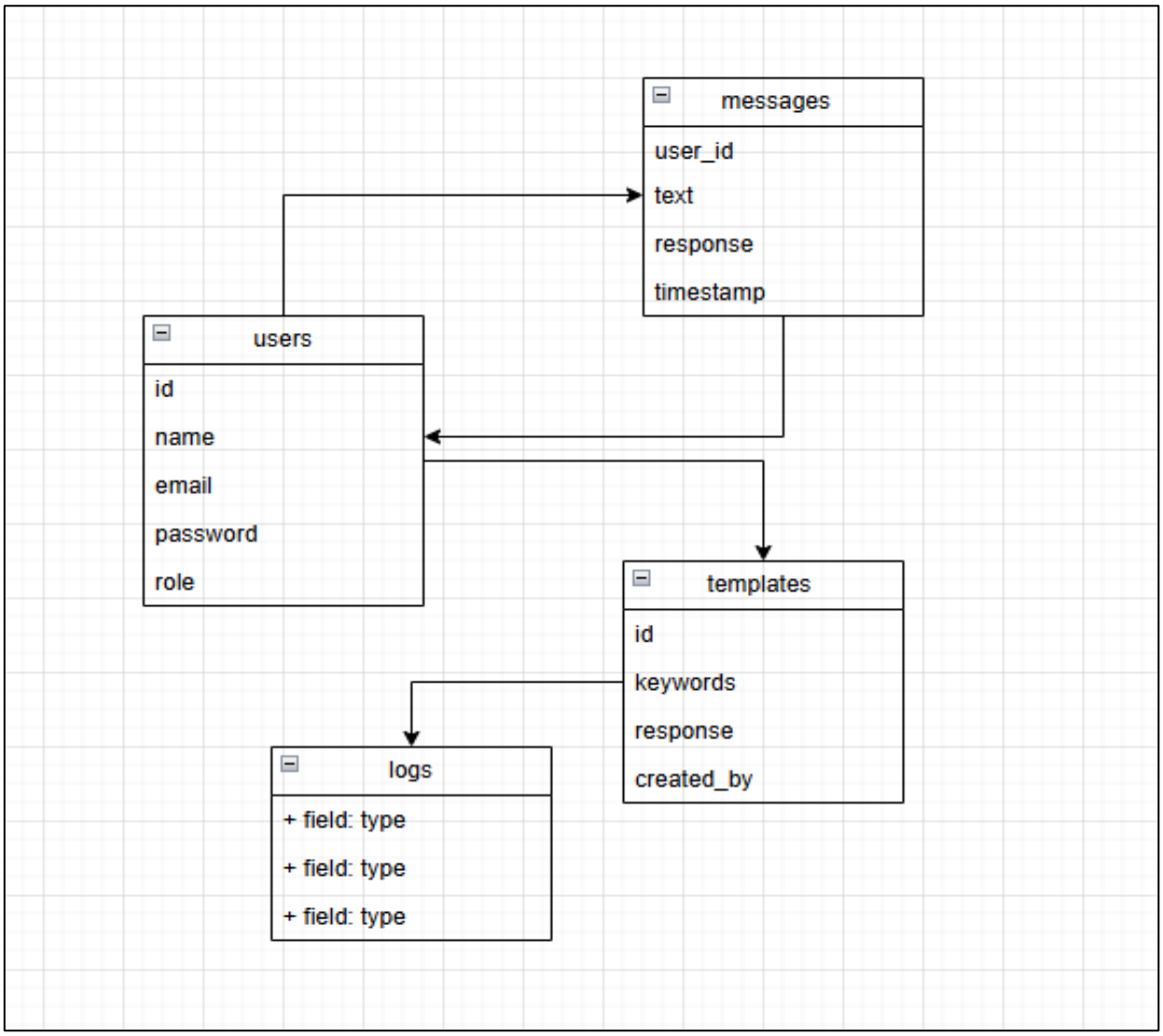


Рисунок 2.4 – ER-діаграма бази даних

Таблиця templates (шаблонні відповіді) використовується для зберігання шаблонів, які чатбот може автоматично повертати при виявленні певних ключових слів. Вона включає такі поля: id, keywords (масив ключових слів), response_text (відповідь), created_by (зовнішній ключ на users) і created_at.

Таблиця logs (журнал подій) веде облік усіх ключових дій користувачів: авторизації, надсилання повідомлень, змін шаблонів тощо. Вона містить: id, user_id, action (опис дії) і timestamp.

На рисунку 2.5 подано приклад створення таблиць у SQL-синтаксисі.

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role VARCHAR(10) DEFAULT 'user',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE messages (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    text TEXT NOT NULL,
    response TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE templates (
    id SERIAL PRIMARY KEY,
    keywords TEXT[],
    response_text TEXT NOT NULL,
    created_by INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 2.5 – SQL-приклад створення таблиць

Додаткові функції бази даних:

– індекси: для поліпшення швидкодії додано індекси на поля email, user_id, timestamp;
– цілісність даних: зовнішні ключі забезпечують зв’язність даних і запобігають їх неконсистентності;
– бекапи: реалізовано автоматичне щоденне резервне копіювання за допомогою планувальника cron (у разі хостингу в AWS – через RDS snapshots);
– безпека: доступ до бази обмежується лише через бекенд, паролі не зберігаються у відкритому вигляді (тільки хеші).

Щоб забезпечити контрольоване та послідовне оновлення схеми БД, використовується механізм міграцій – набір інструкцій, що дозволяє автоматично застосовувати зміни до бази даних у різних середовищах (розробка, тестування, продакшн).

Для реалізації міграцій у проєкті застосовується утиліта Alembic – офіційний інструмент для керування міграціями в SQLAlchemy (ORM, який використовується разом із FastAPI).

Основними перевагами Alembic:

– автоматичне виявлення змін у моделях;
– генерація скриптів міграцій;
– можливість відкату змін (rollback);
– збереження історії версій у таблиці alembic_version.

Представимо приклад міграції на рисунку 2.6, яка демонструє процес створення нової таблиці books за допомогою інструменту Alembic. Такий приклад допоможе краще зрозуміти механізм роботи міграцій у PostgreSQL та способи внесення змін до структури бази даних без необхідності ручного редагування SQL-скриптів.

```
# Унікальні ідентифікатори міграції
revision = '20240521_create_books_table'
down_revision = None
branch_labels = None
depends_on = None

def upgrade():
    op.create_table(
        'books',
        sa.Column('id', sa.Integer, primary_key=True),
        sa.Column('title', sa.String(100), nullable=False),
        sa.Column('author', sa.String(100), nullable=False),
        sa.Column('year', sa.Integer)
    )

def downgrade():
    op.drop_table('books')
```

Рисунок 2.6 – Приклад міграцій

У даному прикладі функція upgrade() створює нову таблицю books із чотирма колонками: id, title, author, year. Якщо пізніше потрібно відкотити зміни, функція downgrade() видалить цю таблицю з бази даних.

Пояснення етапів міграції:

– створення моделі – опис таблиці на Python з використанням SQLAlchemy;
– автоматичне створення файлу міграції – команда alembic revision -autogenerate;
– редагування файлу міграції – за потреби, внесення кастомних змін;
– застосування міграції – команда alembic upgrade head виконує SQL-команди для зміни структури бази;
– відкат змін – команда alembic downgrade -1 повертає попередній стан БД.

Цей механізм дозволяє безпечно й системно оновлювати структуру бази даних у будь-який момент часу, що особливо важливо для проєктів із постійною розробкою, тестуванням та командною співпрацею.

У фінальній версії проєкту міграції можуть запускатися автоматично під час деплою (через GitHub Actions або AWS CodePipeline), що забезпечує безперервну інтеграцію змін у базу даних.

2.7 Висновки до розділу 2

У другому розділі дипломного проєкту, присвяченому проєктуванню чатботу для автоматизації підтримки клієнтів із використанням React, Python та PostgreSQL, було проведено всебічне моделювання архітектури системи та визначено її ключові складові. Зокрема, сформульовано функціональні й нефункціональні вимоги до застосунку, що дало змогу чітко окреслити очікувану поведінку системи та технічні критерії якості.

Було розроблено структурну модель інтерфейсу користувача, яка забезпечує інтуїтивну взаємодію з чатботом, а також описано інтерфейсні вимоги, які відображають особливості взаємодії між frontend-частиною на React та backend-сервером на FastAPI. Серверну логіку реалізовано на Python

із використанням FastAPI, що забезпечує високопродуктивну обробку запитів, підтримку JWT-автентифікації, розмежування прав доступу та можливість керування шаблонними відповідями.

Окрему увагу приділено інтеграції бази даних PostgreSQL: розроблено ER-діаграму, описано структуру основних таблиць (users, messages, templates, logs), а також надано приклад SQL-міграції, що формує базову схему з підтримкою історії змін. Завдяки чіткому структуруванню логіки системи та даних, створено надійний фундамент для реалізації чатбота, який буде масштабованим, безпечним і зручним у використанні.

Таким чином, результати другого розділу формують концептуальну, логічну та технічну основу системи, що дозволяє перейти до наступного етапу – безпосередньої реалізації та тестування програмного продукту.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Побудова API серверної частини на Python (FastAPI)

У рамках розробки чатботу для автоматизації підтримки клієнтів було прийнято рішення реалізувати серверну логіку за допомогою фреймворку FastAPI на Python, який забезпечує високу продуктивність та асинхронну обробку запитів без необхідності адміністрування складних серверних інфраструктур.

Клієнтська частина, створена з використанням React, взаємодіє із сервером через REST API, що реалізований на FastAPI. API обробляє HTTP-запити від фронтенду, виконує бізнес-логіку, взаємодіє з базою даних PostgreSQL і повертає необхідні дані чи відповіді чатбота.

Схема взаємодії виглядає наступним чином:

– користувач через React-інтерфейс надсилає повідомлення або інші запити;
– запит надходить на сервер FastAPI;
– серверна частина обробляє запит, взаємодіє з базою даних, виконує логіку чатбота (пошук шаблонів відповідей або NLP-аналіз);
– отримана відповідь передається назад фронтенду;
– React-додаток відображає відповідь користувачу.

Розглянемо приклади основних API-ендпоінтів, що реалізують ключові функції чатбота: автентифікацію, обробку повідомлень, управління шаблонами відповідей та логування дій користувачів. Такий підхід дозволяє забезпечити ефективну, масштабовану та підтримувану архітектуру системи підтримки клієнтів.

На рисунку 3.1 наведено приклад функції на Python, яка відповідає за отримання даних із бази даних PostgreSQL. Вона виконує роль частини серверної логіки чатботу, зокрема отримання списку користувачів та їхніх повідомлень для подальшої обробки або відображення у клієнтській частині.

```python
app = FastAPI()

# Модель відповіді для повідомлення
class Message(BaseModel):
    user_id: int
    text: str
    timestamp: str

# Конфігурація підключення до бази
DB_HOST = os.getenv("DB_HOST", "localhost")
DB_PORT = os.getenv("DB_PORT", "5432")
DB_NAME = os.getenv("DB_NAME", "chatbot_db")
DB_USER = os.getenv("DB_USER", "user")
DB_PASSWORD = os.getenv("DB_PASSWORD", "password")

def get_db_connection():
    return psycopg2.connect(
        host=DB_HOST,
        port=DB_PORT,
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASSWORD
    )

@app.get("/messages/", response_model=list[Message])
async def get_messages():
    try:
        conn = get_db_connection()
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        cursor.execute("SELECT user_id, text, timestamp FROM messages ORDER BY timestamp DESC;")
        messages = cursor.fetchall()
        cursor.close()
        conn.close()

        if not messages:
            raise HTTPException(status_code=404, detail="Повідомлень не знайдено")

        # Конвертуємо datetime у строку ISO для JSON
        for msg in messages:
            msg['timestamp'] = msg['timestamp'].isoformat()

        return messages

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Виникла помилка: {str(e)}")
```

Рисунок 3.1 – Функція для отримання даних з БД

Функція виконує наступні кроки:

– отримує конфіденційні параметри підключення до бази даних (логін, пароль, хост, порт, назву бази) із захищеного сховища;
– встановлює підключення до PostgreSQL за допомогою отриманих параметрів;
– виконує SQL-запит для отримання необхідних даних, наприклад, список повідомлень користувачів або шаблонів;

– формує HTTP-відповідь у форматі JSON із даними або повідомленням про відсутність записів.

Якщо дані відсутні, функція повертає статус 404 із відповідним повідомленням, у разі успішного запиту – статус 200 та самі дані.

Даний код з рисунку 3.1 реалізує серверний ендпоінт чатботу на базі FastAPI, що відповідає за отримання повідомлень із бази даних PostgreSQL.

Основні особливості функції:

– встановлення з'єднання з базою даних за допомогою параметрів підключення, які завантажуються із змінних середовища (DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD);
– виконання SQL-запиту для отримання списку повідомлень користувачів (user_id, text, timestamp), відсортованих за датою від найновіших до найстаріших;
– обробка результату запиту: якщо повідомлень немає, повертається HTTP-помилка 404 з відповідним повідомленням;
– конвертація об'єктів дати і часу у формат ISO-8601 для коректного відображення у JSON-відповіді;
– обробка потенційних помилок;
– у разі виникнення будь-якої помилки сервер повертає HTTP-помилку 500 з описом проблеми.

Цей підхід забезпечує надійну та безпечну взаємодію між клієнтською частиною (React-додатком) та сервером чатбота, гарантуючи, що користувач отримає коректні дані або зрозуміле повідомлення про помилку.

На рисунку 3.2 наведено реалізацію серверного ендпоінта на FastAPI для отримання переліку ролей користувачів із бази даних PostgreSQL. Цей ендпоінт є частиною бекенд-системи чатботу для автоматизації підтримки клієнтів. Його основне завдання – обробляти запити клієнтської частини (React-додатка) і повертати список ролей користувачів (наприклад, "user", "admin", "support") у форматі JSON.

Функція виконує такі кроки:

– встановлює з’єднання з базою даних PostgreSQL;
– виконує SQL-запит для отримання усіх ролей із таблиці roles;
– формує відповідь із переліком ролей;
– у разі відсутності ролей повертає HTTP-помилку 404;
– обробляє можливі помилки під час підключення або запиту.

```python
# Конфігурація підключення до бази даних
DB_HOST = os.getenv("DB_HOST", "localhost")
DB_PORT = os.getenv("DB_PORT", "5432")
DB_NAME = os.getenv("DB_NAME", "chatbot_db")
DB_USER = os.getenv("DB_USER", "user")
DB_PASSWORD = os.getenv("DB_PASSWORD", "password")

def get_db_connection():
    return psycopg2.connect(
        host=DB_HOST,
        port=DB_PORT,
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASSWORD
    )

@app.get("/roles/", response_model=List[str])
async def get_roles():
    try:
        conn = get_db_connection()
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        cursor.execute("SELECT role FROM roles;")
        rows = cursor.fetchall()
        cursor.close()
        conn.close()

        if not rows:
            raise HTTPException(status_code=404, detail="Ролі не знайдено")

        roles = [row['role'] for row in rows]
        return roles

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Виникла помилка: {str(e)}")
```

Рисунок 3.2 – Функція для отримання ролей

На рисунку 3.3 наведено реалізацію серверного ендпоінта на FastAPI для отримання історії повідомлень конкретного користувача з бази даних PostgreSQL. Цей ендпоінт є важливою складовою бекенд-системи чатботу для автоматизації підтримки клієнтів, оскільки дозволяє клієнтській частині отримувати повну історію спілкування користувача з чатботом.

```python
@app.get("/messages/{user_id}", response_model=list[Message])
async def get_user_messages(user_id: int, limit: int = Query(20, ge=1, le=100)):
    try:
        conn = get_db_connection()
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        cursor.execute(
            "SELECT user_id, text, timestamp FROM messages WHERE user_id = %s ORDER BY timestamp DESC LIMIT %s;",
            (user_id, limit)
        )
        messages = cursor.fetchall()
        cursor.close()
        conn.close()

        if not messages:
            raise HTTPException(status_code=404, detail="Повідомлень не знайдено")

        # Конвертуємо datetime у ISO-формат для JSON
        for msg in messages:
            msg['timestamp'] = msg['timestamp'].isoformat()

        return messages

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Виникла помилка: {str(e)}")
```

Рисунок 3.3 – Функція для отримання історії користувачів

Основним завданням цього ендпоінта є обробка HTTP-запитів від React-додатка, у яких передається ідентифікатор користувача, та формування у відповідь списку повідомлень, які цей користувач надіслав або отримав. Повідомлення повертаються у форматі JSON, що забезпечує просту інтеграцію з фронтендом.

У процесі роботи функція спочатку встановлює безпечне з’єднання з базою даних PostgreSQL, використовуючи параметри підключення, які можуть бути налаштовані через змінні середовища. Після цього виконується SQL-запит, який вибирає всі записи з таблиці messages, пов’язані з заданим користувачем, і сортує їх за датою від нових до старих.

Якщо для вказаного користувача в базі даних не знайдено жодного повідомлення, сервер повертає HTTP-помилку 404 із відповідним повідомленням. Це дозволяє фронтенду коректно обробляти випадки, коли історія відсутня.

Крім того, реалізована обробка можливих помилок, таких як проблеми з підключенням до бази даних або помилки під час виконання SQL-запиту. У разі виникнення таких помилок сервер повертає відповідний HTTP-код 500 із

текстом помилки, що сприяє більш надійній роботі системи і полегшує відлагодження.

Таким чином, цей ендпоінт є ключовим для підтримки функціональності чатботу, що дозволяє відстежувати та аналізувати діалоги користувачів, підвищуючи якість автоматизованої підтримки клієнтів.

3.2 Налаштування бази даних PostgreSQL та зберігання діалогів

Для ефективної роботи чатботу та збереження історії спілкування користувачів була обрана реляційна база даних PostgreSQL. Ця система управління базами даних забезпечує надійне зберігання, масштабованість та гнучкість у роботі з даними.

Було розроблено структуру таблиць, які зберігають інформацію про користувачів, їх ролі, повідомлення та шаблони відповідей. Особливу увагу приділено таблиці messages, де фіксуються всі діалоги користувачів із чатботом. Це дозволяє аналізувати історію запитів і покращувати якість відповідей.

На рисунку 3.4 наведено SQL-схеми бази даних PostgreSQL, яка реалізує основні таблиці для зберігання інформації про користувачів, їх ролі, повідомлення чатботу, шаблони відповідей та журнал дій.

Ця структура забезпечує ефективне збереження та організацію даних, що використовуються серверною частиною для обробки запитів і генерації відповідей.

Схема включає такі таблиці:

– users – зберігає дані про користувачів системи, включаючи їх логіни, ролі та дату створення облікового запису;
– messages – фіксує всі повідомлення від користувачів і відповіді чатбота з часовими позначками, що дозволяє зберігати повну історію діалогів;

– templates – містить заздалегідь визначені шаблони відповідей, які чатбот використовує при обробці певних ключових слів або фраз;
– logs – веде журнал подій, що відслідковує дії користувачів і адміністраторів, наприклад, авторизацію, зміну шаблонів тощо.

schema.sql

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('user', 'admin', 'support')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE messages (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    text TEXT NOT NULL,
    response TEXT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE templates (
    id SERIAL PRIMARY KEY,
    keywords TEXT[] NOT NULL,
    response_text TEXT NOT NULL,
    created_by INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE logs (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    action TEXT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 3.4 – SQL-схеми бази даних

Такий підхід дозволяє підтримувати цілісність даних, розмежовувати права доступу, а також забезпечує масштабованість і зручність подальшого розвитку системи.

Також основним моментом є створення таблиці messages, яка виконує роль журналу діалогів між користувачем і чатботом. Завдяки чіткій структурі таблиці, що включає ідентифікатори користувачів, текст повідомлень, відповіді бота та часові позначки, система здатна зберігати повну історію спілкування.

Це дозволяє відновлювати контекст розмови при повторних зверненнях користувача, що значно підвищує якість та релевантність відповідей чатбота. Завдяки збереженню історії діалогів, система може краще розуміти потреби користувача і надавати більш персоналізовану підтримку.

Крім того, накопичення даних про типові запити і шаблони поведінки користувачів дає змогу проводити детальний аналіз, що є важливим для подальшого удосконалення моделей відповіді. Це допомагає виявляти найпоширеніші проблеми і швидше адаптувати чатбот до реальних сценаріїв використання.

Також збереження історії взаємодій забезпечує аудит та контроль дій користувачів, що є ключовим елементом безпеки та підтримки системи. Це дозволяє відстежувати потенційно небажані дії і вчасно реагувати на можливі загрози.

Саме підключення додатку до бази даних є важливим етапом для забезпечення коректної роботи чатботу. У проєкті використовується бібліотека psycopg2 – одна з найпопулярніших і надійних Python-бібліотек для роботи з PostgreSQL. Вона забезпечує швидке й стабільне підключення до бази даних, а також підтримує виконання SQL-запитів.

Конфігурація підключення здійснюється через змінні середовища (environment variables). Це дозволяє:

– захистити конфіденційну інформацію (логіни, паролі, адреси серверів) від прямого потрапляння в код;
– гнучко змінювати параметри підключення без необхідності змінювати сам код програми (наприклад, при переході з тестового середовища на продуктивне);
– спрощує деплой і масштабування, оскільки параметри можна задавати окремо для кожного середовища.

На рисунку 3.5 наведено реалізацію підключення додатку до бази даних. Конфігурація підключення здійснюється за допомогою змінних середовища, що дозволяє зберігати конфіденційну інформацію окремо від коду та легко

змінювати параметри без необхідності змінювати сам додаток. Це підвищує безпеку і гнучкість розгортання системи у різних середовищах: локальному, тестовому або продуктивному.

```
DB_HOST = os.getenv("DB_HOST", "localhost")
DB_PORT = os.getenv("DB_PORT", "5432")
DB_NAME = os.getenv("DB_NAME", "chatbot_db")
DB_USER = os.getenv("DB_USER", "user")
DB_PASSWORD = os.getenv("DB_PASSWORD", "password")
```

Рисунок 3.5 – Підключення застосунку до БД

Для встановлення з’єднання з базою даних використовується бібліотека psycopg2, яка забезпечує стабільний і ефективний інтерфейс для роботи з PostgreSQL.

3.3 Реалізація клієнтського інтерфейсу чатботу на React

Реалізація клієнтської частини чатботу для автоматизації підтримки клієнтів базується на використанні React, що забезпечує зручний і динамічний інтерфейс користувача. Основним завданням є організація взаємодії користувача з чатботом через REST API, реалізоване на сервері за допомогою Python і бази даних PostgreSQL.

Клієнтська частина відповідає за відправлення повідомлень користувача, отримання відповідей чатбота, а також відображення історії діалогів у реальному часі. Для забезпечення надійного зв’язку з сервером використовується асинхронна робота з API.

На рисунку 3.6 наведено компонента React, який реалізує форму для відправлення повідомлень та отримання відповідей чатбота, забезпечуючи інтерактивну комунікацію між користувачем і системою підтримки.

Компонент Chatbot відповідає за відображення інтерфейсу для спілкування користувача з чатботом у додатку, розробленому на React. Він

реалізує логіку отримання історії повідомлень, відправки нових повідомлень та обробку станів завантаження і помилок.

```jsx
import React, { useState, useEffect } from "react";

function Chatbot() {
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState("");
  const [loading, setLoading] = useState(false);

  const fetchMessages = async () => {
    try {
      const response = await fetch("/api/messages");
      if (!response.ok) throw new Error("Помилка завантаження повідомлень");
      const data = await response.json();
      setMessages(data);
    } catch (error) {
      console.error(error);
    }
  };

  useEffect(() => {
    fetchMessages();
  }, []);

  const sendMessage = async () => {
    if (!input.trim()) return;
    setLoading(true);

    try {
      const response = await fetch("/api/messages", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ text: input }),
      });
      if (!response.ok) throw new Error("Помилка відправки повідомлення");
      const newMessage = await response.json();

      setMessages((prev) => [newMessage, ...prev]);
      setInput("");
    } catch (error) {
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
```

Рисунок 3.6 – Компонент форми відображення повідомлення

Розглянемо основні етапи роботи компонента. Перший етап – ініціалізація стану та завантаження повідомлень:

– використовуються React-хуки useState для збереження списку повідомлень (messages), тексту поточного введення (input) та стану завантаження (loading) (це дозволяє контролювати вміст форми та інтерфейсу користувача);

– за допомогою хука useEffect виконується виклик функції fetchMessages при першому рендері компонента, яка робить

асинхронний запит на сервер (HTTP GET) для отримання існуючих повідомлень чатбота;

– якщо відповідь від сервера успішна, список повідомлень зберігається в стані messages (у разі помилки вона фіксується в консолі).

Другий етап – обробка надсилання повідомлення:

– при введенні тексту користувачем значення поля input синхронізується зі станом через подію onChange;
– коли користувач натискає кнопку «Відправити», виконується функція sendMessage;
– у цій функції спочатку перевіряється, чи не є поле порожнім (видаляються зайві пробіли);
– якщо поле заповнене, встановлюється стан loading, щоб заборонити повторні відправлення та показати, що триває процес;
– потім відправляється POST-запит на сервер із текстом повідомлення в тілі запиту у форматі JSON;
– якщо сервер повертає успішну відповідь, нове повідомлення додається у список messages, а поле введення очищується;
– у разі помилки вона відображається у консолі;
– в будь-якому випадку по завершенню запиту стан loading скидається.

Таким чином, компонент забезпечує повноцінний цикл взаємодії з чатботом – від отримання та відображення історії діалогу до надсилання нових повідомлень.

Використання React-хуків для керування станом та асинхронних запитів дозволяє зробити код зрозумілим, гнучким та зручним для подальшого розширення. Це забезпечує зручний інтерфейс для користувачів, а також легко інтегрується з серверною частиною на Python та базою даних PostgreSQL.

На рисунку 3.7 наведено компонент Register для реєстрації нових користувачів у React-додатку чатботу. Цей компонент реалізує форму для введення необхідних даних (ім’я, email, пароль) з базовою валідацією та взаємодією з бекендом через API.

```jsx
import React, { useState } from "react";

function Register() {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    password: "",
  });
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const validateEmail = (email) => {
    return /\S+@\S+\.\S+/.test(email);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");

    if (!formData.name.trim() || !formData.email.trim() || !formData.password.trim()) {
      setError("Будь ласка, заповніть усі поля.");
      return;
    }

    if (!validateEmail(formData.email)) {
      setError("Невірний формат email.");
      return;
    }

    setLoading(true);

    try {
      const response = await fetch("/api/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(formData),
      });

      if (!response.ok) {
        const data = await response.json();
        throw new Error(data.message || "Помилка реєстрації.");
      }
```

Рисунок 3.7 – Компонент для реєстрації

Основні етапи роботи компонента:

– створення локального стану за допомогою React-хука useState для зберігання введених користувачем даних, повідомлень про помилки та стану завантаження;
– обробка введення користувача у відповідні поля форми через керовані компоненти;
– валідація даних перед відправкою перевірка на порожні поля та правильність формату email;

– відправка POST-запиту на сервер для реєстрації користувача;
– обробка відповіді сервера: у разі успіху – перенаправлення на сторінку входу; у разі помилки – виведення повідомлення.

Цей компонент є базовим для створення користувачів, які зможуть взаємодіяти з чатботом, отримуючи персоналізовану підтримку.

На рисунку 3.8 наведено компонент ChatHistory, який відповідає за відображення історії повідомлень між користувачем і чатботом. Цей компонент є окремим елементом клієнтської частини застосунку, розробленої з використанням React. Його основна мета – надати користувачеві можливість переглядати попередні сесії спілкування та зручно орієнтуватися в діалогах.

```
import React, { useEffect, useState } from "react";

function ChatHistory() {
  const [messages, setMessages] = useState([]);

  const fetchHistory = async () => {
    try {
      const response = await fetch("/api/messages/history");
      if (!response.ok) throw new Error("Не вдалося завантажити історію");
      const data = await response.json();
      setMessages(data);
    } catch (error) {
      console.error("Помилка завантаження історії:", error);
    }
  };

  useEffect(() => {
    fetchHistory();
  }, []);

  const groupByDate = (messages) => {
    return messages.reduce((acc, msg) => {
      const date = new Date(msg.timestamp).toLocaleDateString();
      acc[date] = acc[date] || [];
      acc[date].push(msg);
      return acc;
    }, {});
  };

  const groupedMessages = groupByDate(messages);

```

Рисунок 3.8 – Компонент історії повідомлень

Компонент ChatHistory реалізує завантаження збережених повідомлень із серверної частини (через API-запит до бази даних PostgreSQL) та їх

виведення у форматі, що групується за датами або сесіями. Це дозволяє зберігати контекст розмов, аналізувати попередні відповіді та полегшує користувачу навігацію в інтерфейсі. Додатково реалізовано візуальне розділення повідомлень за датами, що покращує зручність інтерфейсу.

Крім того, передбачено механізм оновлення історії у режимі реального часу після надсилання нового повідомлення, що забезпечує безперервність діалогу. Завдяки використанню useEffect та асинхронних запитів до API, компонент забезпечує стабільну роботу навіть при великій кількості повідомлень.

3.4 Тестування функціоналу та обробка типових сценаріїв

У цьому розділі буде розглянуто процес тестування функціоналу та обробки типових сценаріїв вебзастосунку чатбота для автоматизації підтримки клієнтів, розробленого з використанням React, Python (FastAPI) та PostgreSQL. Основною метою тестування є перевірка стабільності, надійності та передбачуваності роботи системи, а також своєчасне виявлення і усунення потенційних помилок до впровадження в реальне середовище.

Особлива увага приділяється перевірці ключових компонентів: форми автентифікації, логіки обробки повідомлень чатбота, взаємодії з базою даних та механізмів збереження історії повідомлень.

На рисунку 3.9 представлено реалізований інтерфейс чатботу для автоматизації клієнтської підтримки. Основний акцент зроблено на зручності спілкування користувача з ботом у режимі реального часу. Центральна частина екрана відведена під вікно діалогу, де виводяться повідомлення користувача та відповіді системи.

Праворуч розташовано панель з додатковими функціями: доступ до історії звернень, інформації про активну сесію та короткі підказки щодо використання чатбота. У верхній частині інтерфейсу розміщено меню доступу

до налаштувань облікового запису, аналітики звернень та адміністративної панелі (для користувачів із відповідними правами доступу).

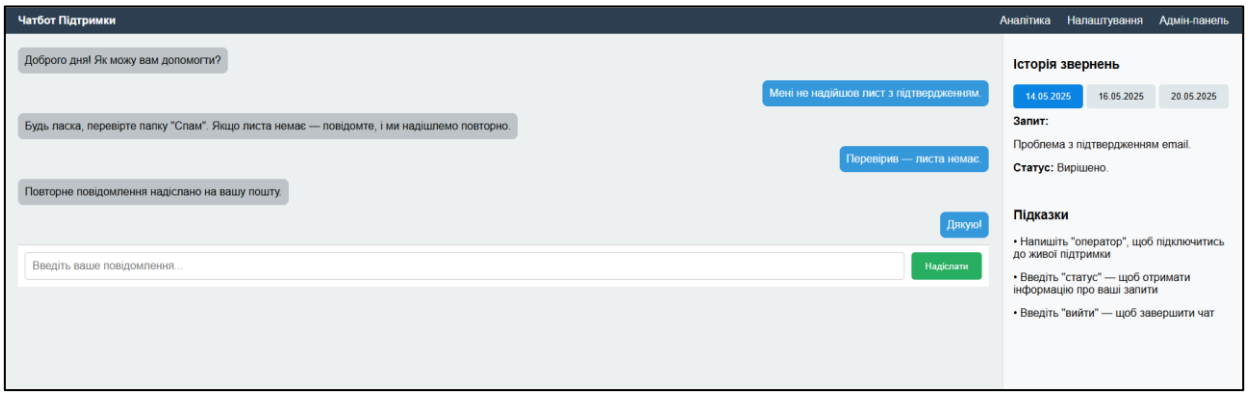


Рисунок 3.9 – Інтерфейс чатбота

Таке компонування забезпечує інтуїтивно зрозумілу навігацію та дозволяє користувачеві зосередитися на основному – швидкому та ефективному спілкуванні з чатботом.

На рисунку 3.10 представлено реалізований інтерфейс сторінки реєстрації нового користувача.

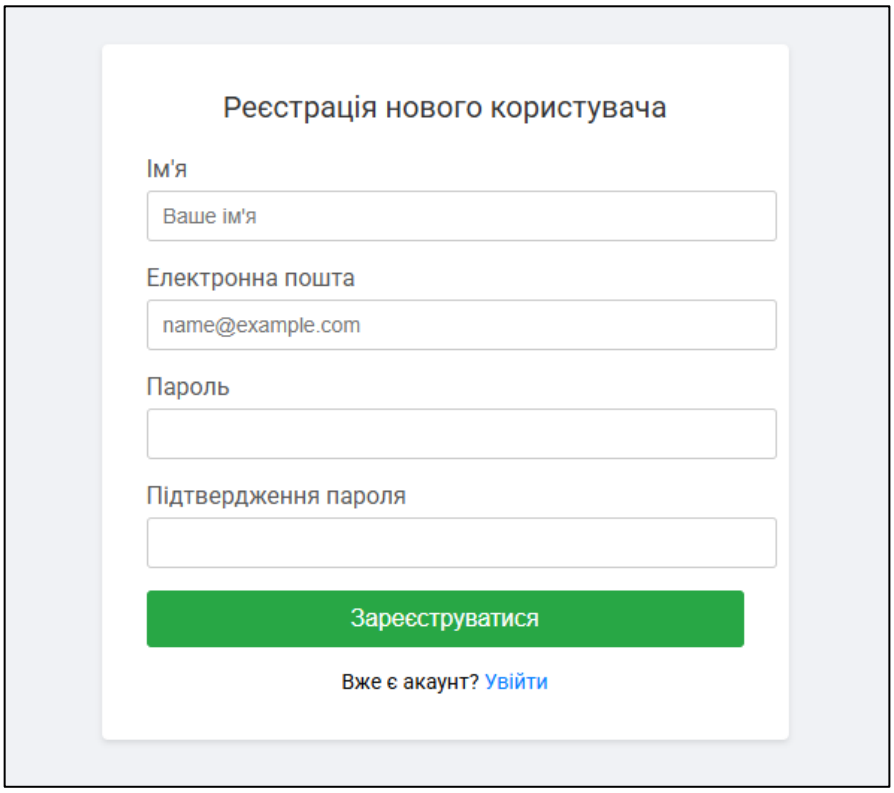


Рисунок 3.10 – Форма реєстрації користувача

Основну увагу приділено зручності та мінімалізму – інтерфейс є інтуїтивно зрозумілим і не перевантажений зайвими елементами.

У верхній частині екрана розташовано заголовок із назвою системи «Чатбот підтримки», який виконує навігаційну та інформаційну функцію.

Центральна частина сторінки займає компактна панель реєстрації, розміщена у центрі екрану з достатніми відступами та тінню для візуального відокремлення від фону.

У панелі послідовно розміщені поля для введення імені, електронної пошти, пароля та підтвердження пароля. Кожне поле супроводжується відповідним підписом, що сприяє легкості заповнення форми навіть для недосвідчених користувачів.

Кнопка «Зареєструватися» стилізована у фірмових кольорах системи та займає всю ширину форми, що дозволяє зручно натискати її як на комп’ютері, так і на мобільному пристрої.

Нижче розміщено посилання для переходу до форми авторизації, якщо користувач уже має обліковий запис.

Такий дизайн забезпечує зрозумілу та приємну взаємодію з системою, а також підтримує єдину візуальну концепцію усієї платформи, включаючи інтерфейс чатбота, сторінки авторизації та панелі керування.

На рисунку 3.11 зображено користувацький інтерфейс чатбота з прикладом обробки помилки підключення до сервера електронної пошти. Як видно зі скріншота, система реалізує багаторівневу візуалізацію помилок для покращення користувацького досвіду.

Якщо аналізувати конкретно про додаткову систему тестування, то доцільно реалізувати юніт-тести не лише для модуля автентифікації, а й для інших ключових частин серверної логіки чатбота, таких як обробка повідомлень, підключення до бази даних PostgreSQL, а також взаємодія з зовнішніми API.

Це дозволяє на ранніх етапах виявити логічні помилки, забезпечити стабільність системи та пришвидшити відлагодження.

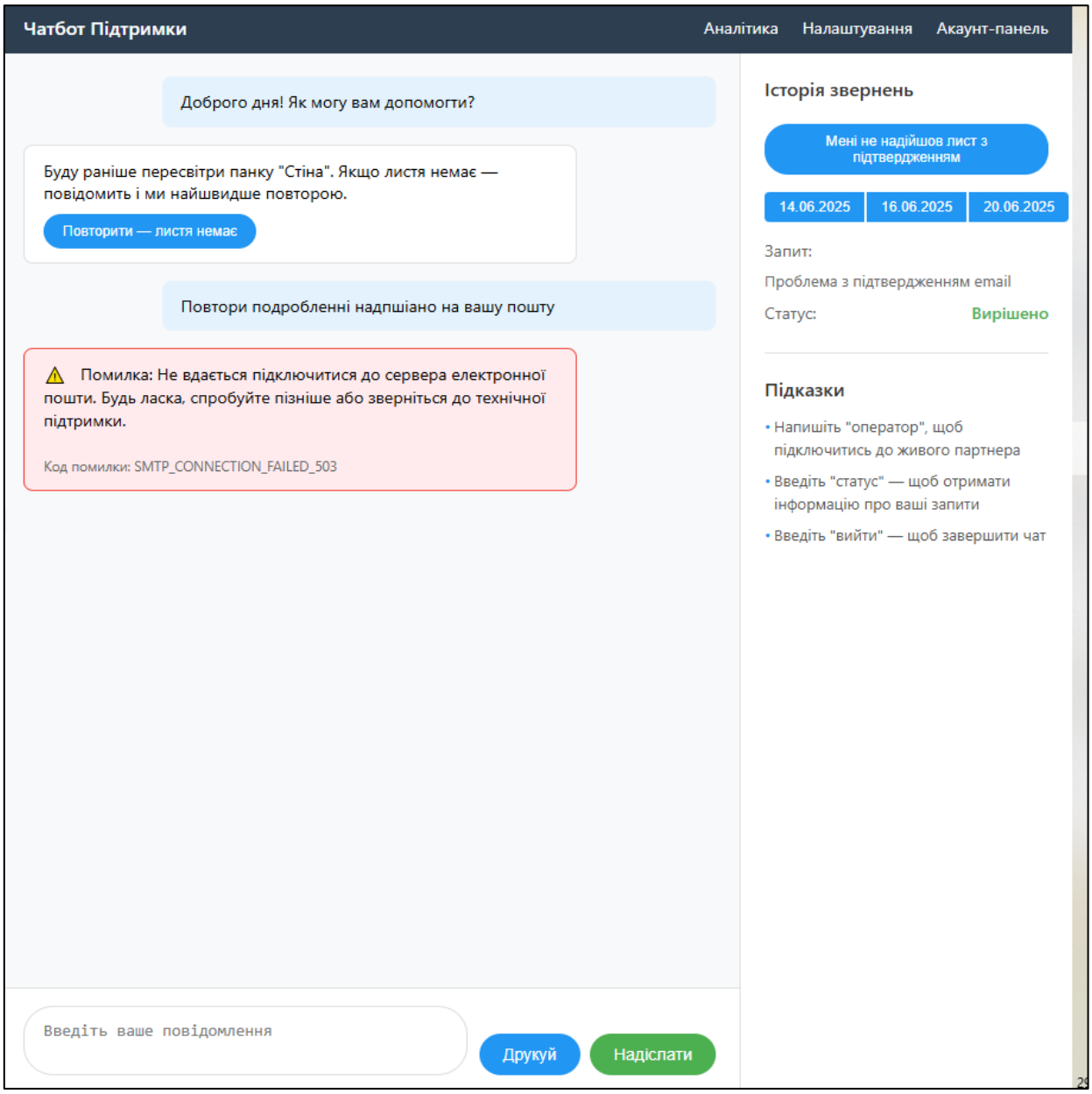


Рисунок 3.11 – Обробка помилок чатбота

На рисунку 3.12 наведено юніт-тестування функціоналу реєстрації користувача у чатботі, реалізованого за допомогою фреймворку Pytest.

Метою тесту є перевірка коректності логіки функції register_user(). Зокрема, тестування охоплює правильність обробки вхідних даних, таких як електронна пошта та пароль, а також перевірку на дублювання користувача в базі даних. Важливо також переконатися, що новий користувач успішно зберігається до бази даних PostgreSQL за допомогою SQLAlchemy. Окрему увагу приділено обробці типових помилок, наприклад, порушення унікальності електронної пошти.

```python
def test_register_user_success(session):
    test_data = {
        "email": "newuser@example.com",
        "password": "Password2024!"
    }

    result = register_user(session, test_data)

    assert result["status"] == "success"
    assert "user_id" in result

    created_user = session.query(User).filter_by(email="newuser@example.com").first()
    assert created_user is not None
    assert created_user.email == "newuser@example.com"
```

Рисунок 3.12 – Юніт-тестування реєстрації

Завдяки такому підходу юніт-тестування забезпечує стабільність та передбачуваність роботи реєстраційного модуля. Це дозволяє швидко виявляти помилки при зміні логіки або структури моделі користувача. Крім того, тест гарантує, що користувач не зможе зареєструватися повторно з уже існуючою електронною адресою, що є критично важливим для цілісності та безпеки системи.

На рисунку 3.13 наведено реалізацію unit-тесту, з використанням Pytest для перевірки функціоналу авторизації користувача у чатботі. Цей тест охоплює ключові етапи логіки: обробку введених даних, перевірку облікових записів у базі даних, а також генерацію токена після успішного входу.

Метою тестування є забезпечення коректної роботи модуля авторизації, який реалізований на Python з використанням бібліотек bcrypt, jwt та psycopg2 для взаємодії з PostgreSQL.

Основний функціонал, що перевіряється:

– валідна авторизація: введено правильні email та пароль – система має повернути токен доступу;
– невірний пароль: якщо пароль неправильний, повертається помилка з відповідним повідомленням;
– неіснуючий користувач: перевірка ситуації, коли в базі даних немає email-адреси;

– відсутність даних: якщо тіло запиту порожнє або некоректне, обробник повинен повернути повідомлення про помилку;
– обробка винятків: наприклад, якщо не вдалося підключитися до бази даних – тест має переконатися, що відповідь не призведе до краху застосунку.

```python
def test_login_success(monkeypatch):
    def mock_get_user_by_email(email):
        return {
            "email": "user@example.com",
            "password_hash": b"$2b$12$testhash",
            "id": 1
        }

    def mock_check_password(password, hash_):
        return True

    def mock_generate_token(user_id):
        return "mocked.jwt.token"

    monkeypatch.setattr("auth.get_user_by_email", mock_get_user_by_email)
    monkeypatch.setattr("auth.check_password", mock_check_password)
    monkeypatch.setattr("auth.generate_token", mock_generate_token)

    result = login_user("user@example.com", "correctpass")
    assert result["token"] == "mocked.jwt.token"

def test_login_wrong_password(monkeypatch):
    def mock_get_user_by_email(email):
        return {
            "email": "user@example.com",
            "password_hash": b"$2b$12$testhash",
            "id": 1
        }

    def mock_check_password(password, hash_):
        return False

    monkeypatch.setattr("auth.get_user_by_email", mock_get_user_by_email)
    monkeypatch.setattr("auth.check_password", mock_check_password)

    with pytest.raises(ValueError, match="Невірний пароль"):
        login_user("user@example.com", "wrongpass")

def test_login_user_not_found(monkeypatch):
    def mock_get_user_by_email(email):
        return None

    monkeypatch.setattr("auth.get_user_by_email", mock_get_user_by_email)

    with pytest.raises(ValueError, match="Користувача не знайдено"):
        login_user("notfound@example.com", "any")
```

Рисунок 3.13 – Юніт-тестування авторизації

3.5 Висновки до розділу 3

У третьому розділі було розглянуто процес реалізації та тестування основних компонентів чатбот-системи для автоматизації клієнтської підтримки, розробленої із застосуванням React для клієнтської частини, Python (FastAPI) – для серверної логіки, та PostgreSQL – як системи керування базами даних. Основну увагу приділено інтеграції фронтенду з бекендом, організації обміну повідомленнями між користувачем і чатботом, а також реалізації механізмів зберігання історії діалогів у базі даних. Серверна частина забезпечує обробку запитів, авторизацію користувачів, маршрутизацію повідомлень і формування відповідей, а клієнтський інтерфейс на React дозволяє здійснювати взаємодію в реальному часі.

У розділі також охарактеризовано застосовані типи тестування, включаючи модульне, інтеграційне та системне тестування, проведені для перевірки функціональності як окремих модулів, так і всієї системи в цілому. Для перевірки надійності та стабільності роботи чатботу використовувалися такі інструменти, як Jest (для клієнтської частини), Pytest (для серверної логіки), а також Postman для тестування REST API.

Окремо розглянуто тестування функціоналу автентифікації та реєстрації користувачів, що є критично важливими для забезпечення безпеки доступу до сервісу. Було реалізовано низку юніт і інтеграційних тестів для перевірки обробки коректних та некоректних запитів, перевірки збереження токенів доступу (JWT) та контролю захищених маршрутів.

Таким чином, проведена розробка й тестування підтверджують, що реалізована чатбот-система є ефективною, масштабованою та готовою до впровадження в реальному середовищі. Її гнучка архітектура дозволяє легко інтегрувати нові функції, адаптувати до різних сценаріїв підтримки клієнтів і використовувати як основу для подальшого розвитку систем автоматизованої взаємодії з користувачами.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто основні етапи проєктування та реалізації чатботу для автоматизації підтримки клієнтів, розробленого з використанням фреймворку React для клієнтської частини, мови програмування Python для серверної логіки та PostgreSQL як системи керування базами даних.

Під час роботи було реалізовано такі задачі:

- проаналізовано сучасні рішення в сфері автоматизованої клієнтської підтримки та відповідні технології розробки;
- сформульовано функціональні та нефункціональні вимоги до чатбот-системи;
- спроєктовано архітектуру клієнтської та серверної частин, а також структуру бази даних;
- реалізовано компоненти інтерфейсу користувача на базі React;
- розроблено серверну логіку обробки повідомлень і взаємодії з базою даних за допомогою Python;
- впроваджено систему автентифікації та реєстрації користувачів;
- проведено тестування клієнтської та серверної частин, зокрема модульне та інтеграційне тестування.

Робота поділена на три основні розділи. У першому розділі виконано аналіз існуючих чатбот-систем (наприклад, Dialogflow, ManyChat, Tidio), обґрунтовано вибір стеку технологій: React – як зручного інструменту для побудови інтерактивного UI, Python – як ефективного засобу для обробки запитів на сервері, та PostgreSQL – як надійної реляційної СУБД.

У другому розділі представлено процес проєктування на концептуальному, логічному та фізичному рівнях. У третьому розділі наведено реалізацію ключових компонентів клієнтської частини на React (вхід, реєстрація, інтерфейс чату), серверної логіки на Python (обробка повідомлень,

відповіді бот-системи) та взаємодії з базою даних PostgreSQL. Також описано процес авторизації користувачів і наведено приклади тестування застосунку.

Результати виконання роботи можуть бути використані як основа для створення гнучких, масштабованих і безпечних систем клієнтської підтримки у різних сферах, зокрема електронній комерції, сервісному обслуговуванні та освітніх платформах. Отримані рішення можуть слугувати базою для подальших досліджень у сфері інтеграції чатботів зі штучним інтелектом та самообслуговуванням клієнтів.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 09.02.2025).

2. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 23.02.2025).

3. React Router Documentation. URL: https://reactrouter.com/ (дата звернення: 26.02.2025).

4. Axios Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 26.02.2025).

5. Python Official Documentation. URL: https://docs.python.org/3/ (дата звернення: 05.03.2025).

6. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 06.03.2025).

7. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 07.03.2025).

8. SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org/ (дата звернення: 10.03.2025).

9. WebSocket API Documentation. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 15.03.2025).

10. Python WebSockets Library Documentation. URL: https://websockets.readthedocs.io/ (дата звернення: 20.03.2025).

11. How Universities are Using Education Chatbots to Enhance the System. URL: https://www.aivo.co/blog/how-universities-are-using-education-chatbots-toenhance-the-system (дата звернення: 01.04.2025).

12. How Universities are Using Chatbots to Help Improve Student Experience. URL: https://www.kommunicate.io/blog/chatbots-in-universities/ (дата звернення: 01.04.2025).

13. Why Universities are Using Chatbots: The Future of Higher Education. URL: https://blog.ubisend.com/discover-chatbots/universities-using-chatbots (дата звернення: 01.04.2025).

14. ESLint Documentation. URL: https://eslint.org/docs/user-guide/getting-started (дата звернення: 05.04.2025).

15. GitHub Actions Documentation. URL: https://docs.github.com/en/actions (дата звернення: 09.04.2025).

16. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 12.04.2025).

17. JWT.io Documentation. URL: https://jwt.io/introduction (дата звернення: 14.04.2025).

18. OpenAI API Documentation. URL: https://platform.openai.com/docs (дата звернення: 14.04.2025).