МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
"ДЕРЖАВНИЙ УНІВЕРСИТЕТ "КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ"
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ

«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Програмна система аналізу документів для автоматизації пошуку даних з використанням RAG»

Виконавець: Єгор ПІЦИК

Керівник: к.т.н., доцент Вікторія СИДОРЕНКО

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

Київ 2024

ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
"ДЕРЖАВНИЙ УНІВЕРСИТЕТ "КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ"

Факультет *комп'ютерних наук та технологій*
Кафедра *комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«___»_____2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Піцика Єгора Володимировича
(ПІБ випускника)

1. Тема кваліфікаційної роботи: «Програмна система аналізу документів для автоматизації пошуку даних з використанням RAG» затверджена наказом ректора № 1782/ст від 06.09.2024р.
2. Термін виконання роботи: з 26 серпня 2024 року по 03 грудня 2024 року.
3. Вихідні дані до роботи: розроблений додаток «Електронний диспетчер»
4. Зміст пояснювальної записки: 1. Сучасні підходи до розроблення і впровадження веб-сервісів. 2. Аналіз архітектурних рішень і вибір програмних засобів для реалізації веб-системи. 3. Програмна реалізація системи аналізу документів та автоматизація пошуку даних з використанням RAG.
5. Перелік обов'язкового графічного ілюстративного матеріалу: 1. Об'єкт і предмет. 2. Мета і задачі. 3. Langchain - інструмент для роботи з LLM. 4. Векторизація даних. 5. Записи до LLM. 6. Висновок.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Дослідження літературних джерел, збір інформації. Написання 1 розділу, представлення керівнику.	26.08.2024-06.09.2024	
2	Аналіз архітектурних рішень. Написання 2 розділу, представлення керівнику.	09.09.2024-25.09.2024	
3	Програмна реалізація системи аналізу документів.	26.09.2024-22.10.2024	
4	Написання 3 розділу, представлення керівнику.	23.10.2024-08.11.2024	
5	Редагування пояснювальної записки.	11.11.2024-19.11.2024	
6	Проходження нормоконтролю.	20.11.2024-25.11-2024	
7	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	25.11.2024-03.12.2024	

7. Дата видачі завдання 26.08.2024р.

Керівник кваліфікаційної роботи _____ Вікторія СИДОРЕНКО
(підпис керівника)

Завдання прийняв до виконання _____ Єгор ПІЦИК
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Програмна система аналізу документів для автоматизації пошуку даних з використанням RAG» містить: 88 сторінок, 21 рисунок, 2 таблиці, 18 літературних джерел, 3 додатки.

Об’єкт дослідження – процес аналізу документів для автоматизації пошуку даних.

Предмет дослідження – методи, засоби та технології аналізу документів для автоматизації пошуку даних.

Мета кваліфікаційної роботи – розробка програмної системи аналізу документів для автоматизації пошуку даних з використанням RAG.

Методи дослідження – логічний, алгоритмічний аналіз, порівняльний, аналіз інформаційних джерел, моделювання та симуляція.

Результати кваліфікаційної роботи можуть бути використані для бізнесу, державних установ та наукових організацій, де потрібна швидка обробка текстової інформації, зокрема юридичної, фінансової або технічної документації.

СИСТЕМА, АНАЛІЗ, АВТОМАТИЗАЦЯ ПОШУКУ ДАНИХ, RAG, ФРЕЙМВОРК, ФРОНТЕНД, БЕКЕНД, ІНТЕРФЕЙС, МОДЕЛЬ, DJANGO, FLASK, REST API

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Фреймворк	–	інфраструктура програмних рішень, що полегшує розробку
Бібліотека	–	збірка об’єктів чи підпрограм для вирішення близьких за тематикою задач
Модуль	–	функціонально завершений фрагмент програми, оформлений у вигляді окремого файлу з сирцевим кодом або його іменованої частини, призначений для використання в інших програмах
Деплой	–	розгортання програмного забезпечення, усі дії, що роблять програмну систему готовою до використання
Інтрефейс	–	сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача
RAG	–	Retrieval-Augmented Generation
AI	–	Artificial intelligence
SSH	–	Secure Shell
HTTP	–	HyperText Transfer Protocol
HTTPS	–	HyperText Transfer Protocol Secure
HTML	–	HyperText Markup Language
CSS	–	Cascading Style Sheets
API	–	ApplicaITon Programming Interface
Бекенд (Back-end)	–	серверна частина веб-сервісу
Фронтенд (Back-end)	–	браузерна частина веб-сервісу, яка охоплює HTML, CSS, JavaScript

ВСТУП

У сучасному світі, де інформація стала ключовим ресурсом, розвиток інформаційних технологій і програмної інженерії досяг такого рівня, що традиційні підходи до обробки даних вже не відповідають сучасним викликам. Щодня у світі генеруються величезні обсяги текстової інформації: це корпоративні документи, наукові статті, юридичні угоди, фінансові звіти, технічна документація, медичні записи тощо.

Вирішення задач ефективного управління цими даними вимагає нових підходів та інструментів, які поєднують сучасні технології автоматизації та аналізу інформації. Одним із найбільш перспективних напрямків є розробка систем, що базуються на підході Retrieval-Augmented Generation (RAG). Ця методологія є унікальною комбінацією пошукових алгоритмів і генеративних моделей штучного інтелекту. Використання RAG дає можливість не лише витягувати з документів необхідні дані, але й генерувати нові тексти на основі отриманої інформації, що суттєво розширює функціональність традиційних пошукових систем.

Система аналізу документів на основі RAG є особливо актуальною для вирішення завдань, пов’язаних із обробкою великих масивів інформації. Зростання цифровізації у бізнесі, науці, освіті, медицині та державному управлінні супроводжується зростанням попиту на інструменти, що можуть забезпечувати швидкий доступ до структурованих даних. У бізнес-середовищі це дозволяє підприємствам скоротити час на аналіз інформації, що є ключовим для прийняття управлінських рішень. Зокрема, компанії з великим обсягом вхідної документації можуть інтегрувати RAG-системи у свої робочі процеси для автоматизації рутинних завдань, таких як пошук релевантних записів, перевірка відповідності даних чи підготовка звітів.

Наукові організації також виграють від використання таких систем. Наприклад, дослідники, які працюють із сотнями наукових статей або архівними

документами, можуть значно прискорити свої роботи завдяки автоматизованому пошуку відповідної інформації.

У медичній сфері системи RAG можуть допомогти у швидкому аналізі медичних записів, пошуку діагностичної інформації чи створенні рекомендацій на основі медичних протоколів. Крім того, системи аналізу документів з використанням RAG можуть бути корисними у державному секторі. Наприклад, державні установи, які працюють із великими обсягами юридичних, фінансових чи адміністративних документів, можуть використовувати ці системи для автоматизації обробки запитів громадян, підготовки аналітичних звітів чи перевірки відповідності документів законодавчим нормам.

Унікальність підходу RAG полягає в тому, що він поєднує дві ключові функції: витягування даних (retrieval) і генерацію нового контенту (generation). Традиційні пошукові системи здатні знаходити інформацію, але не завжди забезпечують її релевантність або можливість адаптації до конкретного контексту. Генеративні моделі, у свою чергу, можуть створювати текст, але без доступу до зовнішніх баз даних вони обмежені лише внутрішніми знаннями моделі. RAG інтегрує ці підходи, дозволяючи використовувати силу сучасних пошукових алгоритмів разом із генеративними можливостями моделей штучного інтелекту.

З технічної точки зору, RAG-системи використовують сучасні алгоритми обробки природної мови (NLP), які базуються на трансформерних архітектурах, таких як GPT чи BERT. Ці моделі здатні працювати з неструктурованими даними, такими як тексти, що робить їх універсальними для аналізу документів. Водночас, інтеграція баз даних і пошукових систем дозволяє забезпечити точний доступ до необхідної інформації у великих масивах даних. Використання технологій штучного інтелекту дозволяє системам RAG самонавчатися, тобто покращувати свою продуктивність з часом на основі нових даних, що поступають.

Розробка системи аналізу документів з використанням RAG є надзвичайно перспективною областю, яка об’єднує новітні досягнення штучного інтелекту,

машинного навчання та програмної інженерії для створення ефективних рішень, що відповідають викликам сучасного світу.

Актуальність теми веб-застосунку для аналізу документів та автоматизація пошуку даних з використанням RAG користуються значним зростанням попиту на дані технології у IT галузі. Система аналізу документів та автоматизація пошуку даних з використанням RAG (Retrieval-Augmented Generation) є актуальною на сьогоднішній день через зростання обсягів інформації та документів, що вимагають швидкої і точної обробки. В умовах, коли бізнес і наука щодня генерують величезні масиви текстової інформації, автоматизація аналізу документів стає необхідною для підвищення ефективності процесів. Традиційні методи обробки не завжди здатні швидко й точно знаходити релевантну інформацію, що може призводити до втрати часу та ресурсів. Використання RAG дозволяє не тільки автоматично витягати потрібні дані, але й генерувати новий контент на основі вже існуючих даних, що є суттєвою перевагою в умовах швидко мінливого світу. Крім того, система сприяє покращенню якості пошукових результатів, оскільки поєднує можливості штучного інтелекту та сучасних методів машинного навчання для оптимізації роботи з текстовими даними. Це рішення дозволяє значно скоротити час на аналіз великого обсягу документів і забезпечує більш точні відповіді на запити користувачів. Також, автоматизація пошуку важливих даних мінімізує людський фактор, що знижує ймовірність помилок при роботі з інформацією. В контексті бізнесу це означає скорочення витрат і підвищення продуктивності, а в наукових дослідженнях — прискорення відкриття та досягнення нових результатів. RAG-системи є також корисними для державних установ, де важливо швидко обробляти великі обсяги юридичної та фінансової документації. Рішення на основі RAG допомагають ефективно організовувати інформацію та забезпечувати доступ до релевантних даних у потрібний час.

Метою кваліфікаційної роботи є створення системи аналізу документів та автоматизації пошуку даних з використанням технології Retrieval-Augmented Generation (RAG).

Для досягнення поставленої поставлено наступні **завдання кваліфікаційної роботи**:

− дослідити сучасні методи аналізу текстових даних та підходи до автоматизації процесів пошуку інформації;

− проаналізувати принципи роботи RAG і обрати оптимальні програмні засоби для впровадження цієї технології;

− розробити систему, яка забезпечуватиме автоматичне витягування та аналіз релевантної інформації з великої кількості документів, зокрема текстових і юридичних;

− забезпечити можливість генерації нових документів на основі знайдених даних, що підвищить ефективність роботи користувача.

Об’єкт дослідження – процес аналізу документів для автоматизації пошуку даних.

Предмет дослідження – методи, засоби та технолгії аналізу документів для автоматизації пошуку даних.

Методи дослідження – логічний, алгоритмічний аналіз, порівняльний, аналіз інформаційних джерел, моделювання та симуляція.

Практичне значення одержаних результатів Практичне значення одержаних результатів полягає у тому, що розроблена система аналізу документів та автоматизації пошуку даних може стати ефективним інструментом для автоматизації процесів обробки великих обсягів інформації. Наукова новизна полягає в тому, що було оптимізованно використання наявних моделей, комбінація різних типів векторних пошуків (semantic search, BM25), а також впроваджено системи, що здатна опрацьовувати різні формати документів (PDF, DOCX, зображення і т. ін.) із застосуванням попереднього опрацювання.

РОЗДІЛ 1
СУЧАСНІ ПІДХОДИ ДО РОЗРОБКИ І ВПРОВАДЖЕННЯ ВЕБ-СЕРВІСІВ

1.1. Огляд RAG

Основний робочий процес RAG можна реалізувати різними методами. Концептуально RAG у AI-застосунках працює наступним чином:

1. Користувач вводить запит.

2. Система шукає відповідні документи, що можуть містити необхідну інформацію. Зазвичай ці документи є власними даними компанії, збереженими в документних сховищах.

3. На основі цих документів система створює промпт для LLM, де поєднуються введені дані користувача, знайдені документи та інструкції для моделі. Модель використовує надані документи для формування відповіді.

4. Система відправляє цей промпт у LLM.

5. LLM повертає відповідь, засновану на наданих контекстних даних. Ця відповідь стає кінцевим результатом для користувача.

На рис. 1.1 представлено діаграму, що відображає основну ідею в основі RAG. Ця схема відображає основні кроки, необхідні для реалізації RAG у системах на основі великих мовних моделей, де генерація відповіді підсилюється актуальними даними, отриманими через пошук.

Рис. 1.1. Принцип роботи RAG

1.1.1. Використання RAG на практиці

Практичні реалізації RAG, які широко застосовуються в різних організаціях, засновані на запропонованій у публікації концепції, але мають деякі особливості:

Обирають RAG-послідовність. З двох підходів до реалізації архітектури RAG, запропонованих у публікації, майже завжди обирають RAG-послідовність, оскільки цей підхід дешевший і простіший, даючи чудові результати.

Не проводять донавчання трансформерів, включених у систему. Попередньо навчені LLM достатньо хороші для використання у їх первісному вигляді, і їх самостійне донавчання є занадто затратним.

Методи пошуку документів можуть відрізнятися. Пошук часто здійснюється за допомогою пошукових сервісів на зразок FAISS або Azure Cognitive Search, які підтримують різні методи пошуку, що добре інтегруються з RAG [1].

Процес пошуку документів звичайно складається з двох кроків.

Пошук інформації. Система співставляє запит користувача з документами у пошуковому індексі, витягуючи найбільш відповідні документи. Існує три основні підходи до пошуку: пошук за ключовими словами, векторний пошук, гібридний пошук.

Ранжування інформації — це необов'язковий етап після пошуку, на якому оцінюється відповідність документів запиту користувача за допомогою точніших алгоритмів.

Принцип роботи RAG із використанням підходу «пошук за ключовими словами» полягає в тому, що система спочатку здійснює пошук документів, які містять запитувані користувачем ключові слова, а потім використовує ці документи для генерації відповіді мовною моделлю. Цей підхід корисний, коли необхідно знайти точні текстові збіги, такі як номери замовлень, ідентифікатори, адреси, або будь-які інші специфічні текстові елементи.

Основний процес виглядає наступним чином:

1. Користувач вводить запит, що містить ключові слова або фрази.

2. Система використовує повнотекстовий пошук за ключовими словами для пошуку документів у сховищі, які містять ці слова. Пошуковий механізм використовує інвертований індекс для швидкого знаходження відповідних документів.

3. Після отримання релевантних документів система генерує промпт для LLM, що поєднує запит користувача з витягнутими документами.

4. Система передає промпт мовній моделі, яка використовує надані документи для створення більш точної та релевантної відповіді.

5. Мовна модель повертає відповідь, яка враховує дані з документів, знайдених під час пошуку.

6. Користувачу надається відповідь, яка була створена з використанням актуальних документів та мовної моделі.

Цей підхід дозволяє LLM ефективно використовувати специфічні документи для генерації відповідей, навіть якщо модель не була навчена на цих даних.

Рис. 1.2. Загальна система, яка використовує пошук за ключовими словами

У такій системі пошуковий сервіс підтримує роботу інвертованого індексу, що зберігає відповідність між словами та документами. Текстові дані користувача розбираються, ключові слова виділяються та аналізуються для знаходження їх базових форм. Потім виконується пошук в інвертованому індексі, що повертає список релевантних документів. Окрім того, після пошуку документи можуть ранжуватися за допомогою вбудованого ранжування, щоб виділити найрелевантніші документи [2].

1.1.2. Векторний пошук

Принцип роботи RAG із використанням підходу "векторний пошук" заснований на концепції семантичної відповідності, що дозволяє системі знаходити релевантну інформацію навіть у випадках, коли точного текстового збігу між запитом і документами немає. Основна ідея полягає у використанні спеціальних алгоритмів і мовних моделей для перетворення тексту у числові векторні представлення, які можна розглядати як координати у багатовимірному просторі. Ці вектори відображають зміст і смислові зв’язки тексту, а не лише його поверхневу структуру.

У процесі роботи, коли користувач формує запит, його текст перетворюється у вектор за допомогою попередньо навченої мовної моделі. Аналогічно, всі документи або фрагменти даних, з якими працює система, також попередньо перетворені у векторний формат і збережені у спеціальному векторному сховищі [3].

Векторний пошук полягає у визначенні документів, вектори яких найближчі до вектора запиту користувача. Це здійснюється за допомогою метрик вимірювання відстані, таких як косинусна подібність або евклідова відстань, які дозволяють оцінити рівень схожості між векторами. Особливістю цього підходу є його здатність розпізнавати глибинні смислові зв’язки між словами та фразами. Наприклад, якщо користувач запитує "ефективні методи організації часу", система може знайти документи, які описують тайм-менеджмент, навіть якщо жодне слово із запиту точно не співпадає з текстом у базі даних. Це робить векторний пошук потужним інструментом для роботи з великими обсягами інформації, де традиційний текстовий пошук може бути неефективним.

Крім того, векторний пошук у контексті RAG значно підвищує точність і релевантність відповідей, оскільки він дозволяє комбінувати витягнуту інформацію з зовнішніх джерел із потужностями мовної моделі. Після того, як система визначає найбільш релевантні документи, вони використовуються для формування відповіді. Мовна модель інтегрує отримані дані у свою генерацію, надаючи більш обґрунтовану, повну та точну відповідь.

Основний процес виглядає наступним чином:

1. Користувач вводить запит на природній мові.

2. Система перетворює текст запиту у вектор — числову репрезентацію змісту запиту у багатовимірному векторному просторі.

3. Попередньо оброблені документи, що зберігаються у каталозі, також перетворені у вектори під час індексації.

4. Система виконує пошук релевантних документів, використовуючи алгоритми вимірювання схожості між вектором запиту та векторами документів. Чим ближче вектори у векторному просторі, тим більше вони схожі за змістом.

5. Система генерує промпт для LLM, який включає знайдені документи та запит користувача.

6. Промпт передається до LLM, яка на основі релевантних документів генерує відповідь.

7. Модель надає відповідь, використовуючи знайдені документи, що були підібрані на основі семантичної схожості.

8. Відповідь відправляється користувачу, використовуючи документи, які максимально відповідають змісту запиту.

Цей підхід дозволяє знаходити документи, які можуть містити релевантну інформацію навіть при відсутності точних текстових збігів, що робить його особливо корисним для роботи з великими масивами нефіксованих даних.

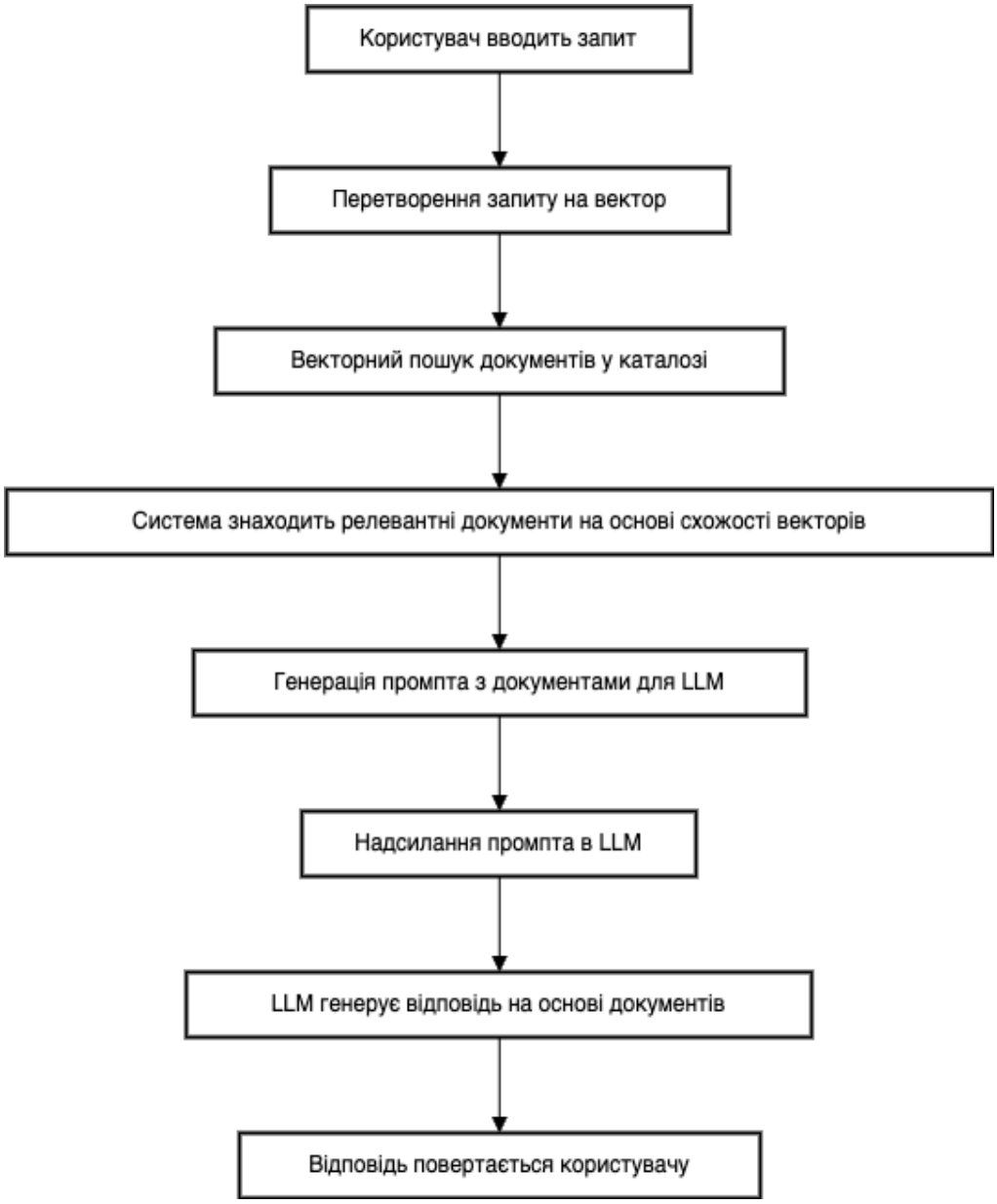


Рис. 1.3. Загальна системи, що використовує векторний пошук

У такій системі документи у каталозі проходять попередню обробку, текстовий зміст перетворюється на вектори. Запити користувачів теж перетворюються на вектори, а потім виконується пошук за допомогою алгоритмів вимірювання схожості векторів. Результатом є список релевантних документів.

1.1.3. Гібридний пошук

Принцип роботи RAG із використанням підходу "гібридний пошук" полягає в інтеграції двох взаємодоповнюючих методів пошуку: пошуку за ключовими словами та векторного (або семантичного) пошуку. Ця комбінація дозволяє ефективніше знаходити релевантні документи, об’єднуючи можливості точного текстового відповідника із більш глибоким аналізом смислових зв’язків між запитом і документами. Гібридний підхід забезпечує високу якість результатів, оскільки враховує різні аспекти релевантності: від прямої текстової відповідності до схожості значення. Пошук за ключовими словами, який зазвичай реалізується через традиційні індексовані бази даних або пошукові системи, дозволяє знаходити документи, що містять точні терміни або фрази з запиту. Цей метод дуже ефективний для вузькоспеціалізованих або чітко сформульованих запитів, оскільки він працює з прямим текстовим збігом. Однак цей метод може бути обмеженим, коли запит містить синоніми, неточності у формулюванні або потребує аналізу контексту [4].

Векторний пошук, з іншого боку, використовує алгоритми обчислення семантичної схожості, що базуються на векторних представленнях текстів. Кожен текст, як і сам запит, перетворюється у векторну форму, яка зберігає смислову інформацію про нього. Завдяки цьому система може знаходити документи, які не містять точних текстових збігів, але відповідають запиту за змістом. Векторний пошук особливо корисний у випадках, коли запити сформульовані вільно, містять помилки, або потребують більш глибокого аналізу змісту. Поєднання цих двох методів у гібридному підході дозволяє долати обмеження кожного з них.

Система спочатку використовує пошук за ключовими словами для швидкого знаходження найочевидніших збігів, а потім застосовує векторний пошук для розширення результатів і знаходження менш очевидних, але релевантних документів. Таким чином, гібридний пошук враховує як точну текстову релевантність, так і семантичний контекст, що робить його особливо ефективним для складних, багатозначних або нетипових запитів.

Перевага гібридного підходу також полягає у його здатності адаптуватися до різних сценаріїв використання. Наприклад, у випадках, коли база даних містить великий обсяг документів із високим ступенем повторення ключових слів, векторний пошук додає смислову диференціацію. У той же час, для чітко структурованих даних, де точність термінів має критичне значення, пошук за ключовими словами забезпечує швидкий доступ до потрібної інформації. Гібридний пошук також сприяє підвищенню користувацького досвіду. Завдяки цьому підходу користувачі отримують більш релевантні відповіді на свої запити, оскільки система враховує не тільки те, що саме було сказано, але й те, що могло матися на увазі.

Це робить взаємодію з системою інтуїтивною та зручною, особливо для недосвідчених користувачів, які можуть формулювати запити у природній мові.

Основний процес гібридного пошуку виглядає наступним чином:

1. Користувач вводить запит на природній мові.
2. Система виконує традиційний пошук за ключовими словами в індексі документів, щоб знайти документи, що містять точні текстові збіги.
3. Паралельно з пошуком за ключовими словами, система перетворює запит на вектор у багатовимірному просторі для семантичного пошуку.
4. Виконується пошук за векторами, щоб знайти документи, які семантично схожі на запит, навіть якщо вони не містять точних збігів за словами.
5. Результати обох пошуків (ключові слова та вектори) об'єднуються.
6. Система ранжує документи на основі їхньої загальної відповідності запиту користувача (поєднання текстових збігів та семантичної схожості).
7. З об'єднаних і ранжованих результатів система генерує промпт для LLM, що містить релевантні документи.
8. Промпт відправляється до LLM для створення відповіді на основі наданих документів.
9. Модель генерує відповідь на запит користувача, використовуючи знайдені документи.

10. Відповідь надсилається користувачу, об'єднуючи найкращі дані з обох методів пошуку.

Цей підхід дозволяє отримувати більш точні та релевантні відповіді, оскільки система використовує як прямі текстові збіги, так і семантичну схожість.

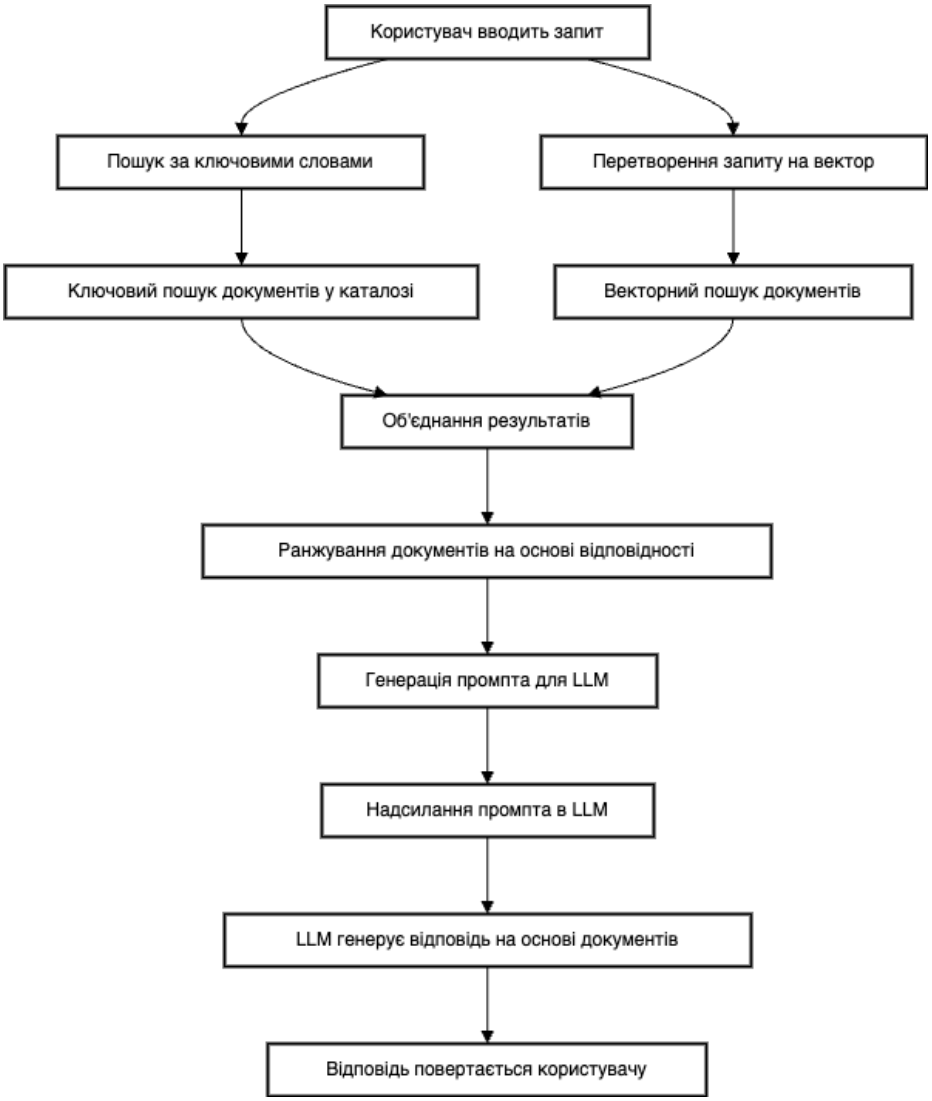


Рис. 1.4. Загальна системи, що використовує гібридний пошук

У системі з гібридним пошуком (рис. 1.4) запити користувачів обробляються як для повнотекстового пошуку, так і для векторного пошуку. Результати обох пошуків об'єднуються, і документи можуть бути ранжовані на основі загальної відповідності запиту користувача [5].

1.2. Фреймворк LangChain

LangChain — це фреймворк з відкритим вихідним кодом, написаний на Python і JavaScript, призначений для створення додатків, орієнтованих на мовні моделі (LLM). Він допомагає розробникам та експертам, які не спеціалізуються на штучному інтелекті, інтегрувати існуючі мовні моделі в різні додатки, забезпечуючи при цьому широкий спектр функціональності. Цей фреймворк підтримує численні завдання, як-от узагальнення тексту, класифікація, створення резюме, аналіз даних, генерація тексту і багато іншого. Він спрощує взаємодію з великими мовними моделями, дозволяючи створювати розширені системи, що використовують мовні моделі для вирішення складних завдань. Але в цьому матеріалі ми зосередимося на тому, як LangChain реалізує стратегію RAG (Retrieval-Augmented Generation), яка доповнює можливості LLM за допомогою зовнішніх джерел даних [6].

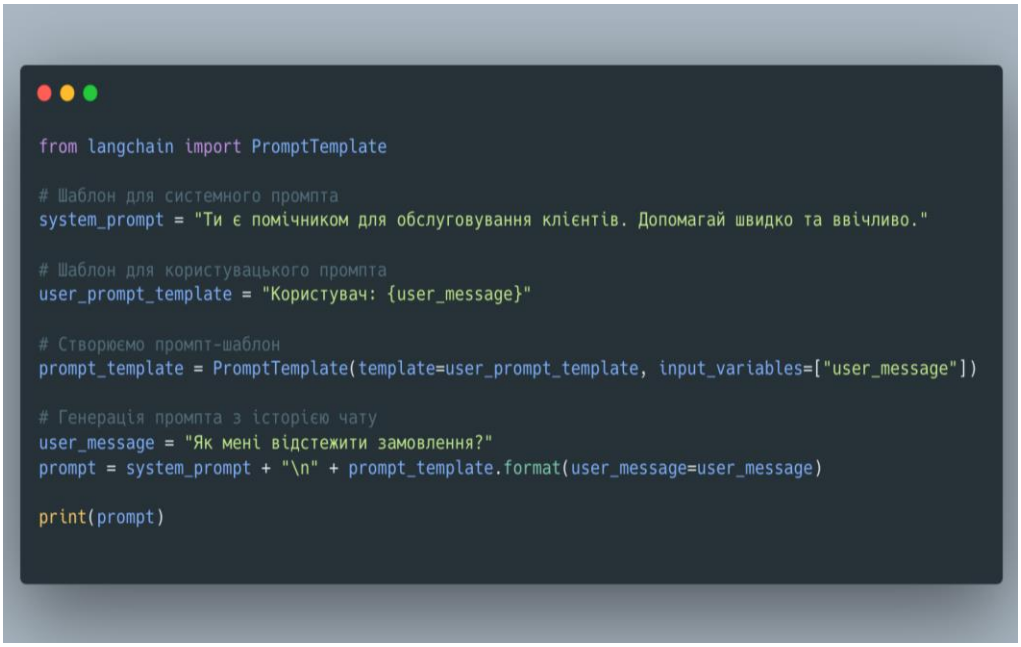
LangChain виступає ефективним інструментом для розробки застосунків, що реалізують стратегію RAG. Стратегія RAG передбачає використання зовнішніх джерел даних у процесі генерації відповідей мовною моделлю. У той час як RAG описує загальну концепцію доповнення LLM новими або спеціалізованими даними, LangChain забезпечує практичний інструментарій для впровадження цієї стратегії на практиці.

LangChain був розроблений спеціально для створення систем, які можуть витягати релевантну інформацію з великих баз даних або зовнішніх джерел і використовувати її для покращення якості відповідей, які генерує LLM. Він дозволяє розробникам не тільки інтегрувати мовні моделі у свої проекти, але й оптимізувати процес пошуку та використання даних з різноманітних джерел [7].

1.2.1. Основні функції LangChain

Промпти — це основа взаємодії з LLM. LangChain дозволяє гнучко управляти як системними промптами (System prompt), так і користувацькими (Human prompt). System prompt — це інструкція для LLM, яка визначає її поведінку та контекст. Human prompt — це запит від користувача.

Важливо розміщувати історію чату у відповідному промпті, щоб модель могла враховувати попередні взаємодії під час генерації відповіді. Історію чату можна додавати до Human prompt або перед System prompt, щоб модель мала доступ до контексту.



```python
from langchain import PromptTemplate

# Шаблон для системного промпта
system_prompt = "Ти є помічником для обслуговування клієнтів. Допомагай швидко та ввічливо."

# Шаблон для користувацького промпта
user_prompt_template = "Користувач: {user_message}"

# Створюємо промпт-шаблон
prompt_template = PromptTemplate(template=user_prompt_template, input_variables=["user_message"])

# Генерація промпта з історією чату
user_message = "Як мені відстежити замовлення?"
prompt = system_prompt + "\n" + prompt_template.format(user_message=user_message)

print(prompt)
```

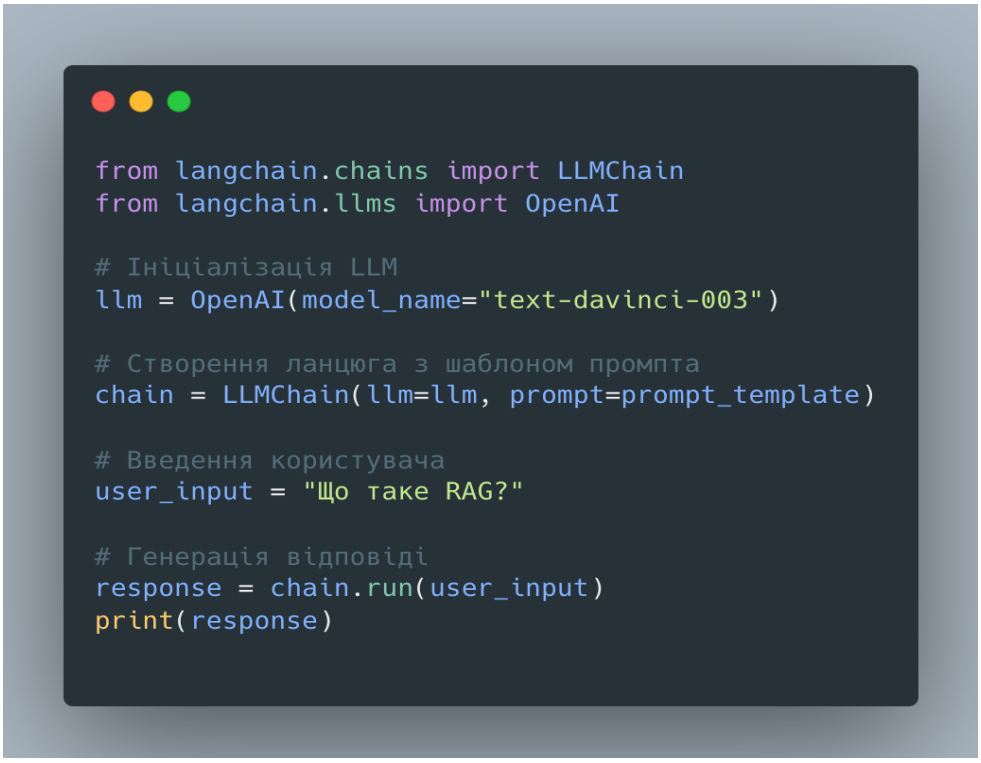
Рис. 1.5. Приклад побудови промпта в LangChain

LangChain дозволяє додавати історію попередніх повідомлень до запиту, щоб LLM мала доступ до контексту взаємодії. Це корисно для більш "розмовного" досвіду.

```python
history = [
    "Користувач: Привіт! Що таке LangChain?",
    "Асистент: LangChain — це фреймворк для побудови додатків на основі LLM."
]

user_message = "Як мені його встановити?"

# Додаємо історію чату перед промптом
full_prompt = system_prompt + "\n" + "\n".join(history) + "\n" +
prompt_template.format(user_message=user_message)

print(full_prompt)
```

Рис. 1.6. Приклад використання історії чату

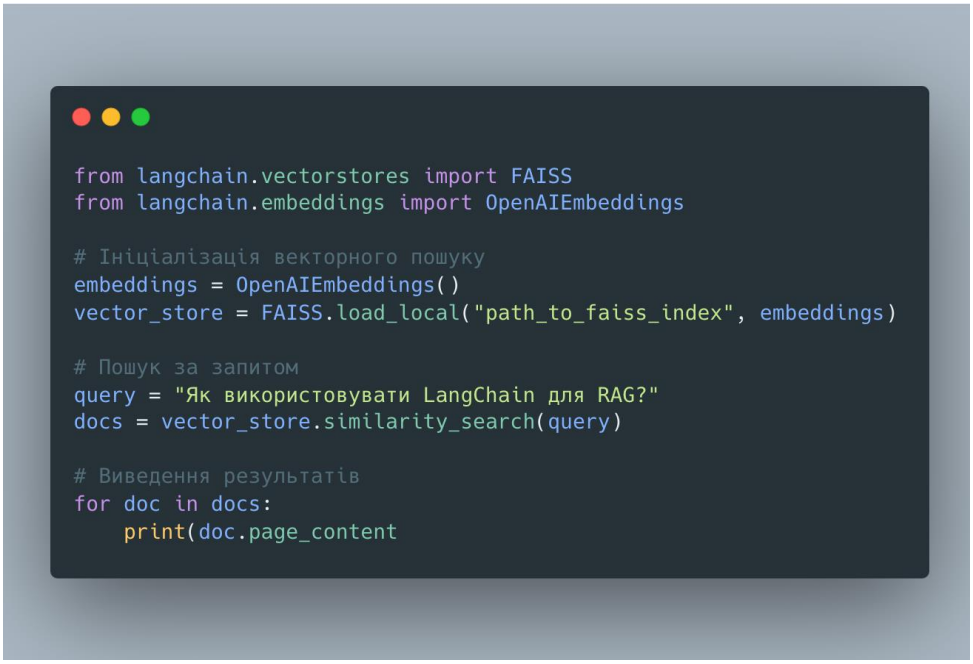
Однією з найважливіших функцій LangChain є можливість побудови «ланцюгів» запитів, які дозволяють об'єднувати кілька етапів обробки даних, таких як витяг контексту, пошук інформації та відповідь на запит.



```python
from langchain.chains import LLMChain
from langchain.llms import OpenAI

# Ініціалізація LLM
llm = OpenAI(model_name="text-davinci-003")

# Створення ланцюга з шаблоном промпта
chain = LLMChain(llm=llm, prompt=prompt_template)

# Введення користувача
user_input = "Що таке RAG?"

# Генерація відповіді
response = chain.run(user_input)
print(response)
```

Рис. 1.7. Приклад простого ланцюга

LangChain дозволяє інтегрувати зовнішні джерела даних, що є важливим для реалізації RAG (Retrieval-Augmented Generation). Зазвичай це включає підключення до баз даних або пошукових індексів для отримання контекстної інформації.



```python
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings

# Ініціалізація векторного пошуку
embeddings = OpenAIEmbeddings()
vector_store = FAISS.load_local("path_to_faiss_index", embeddings)

# Пошук за запитом
query = "Як використовувати LangChain для RAG?"
docs = vector_store.similarity_search(query)

# Виведення результатів
for doc in docs:
    print(doc.page_content
```

Рис. 1.8. Приклад інтеграції зовнішніх даних

1.2.2. Реалізація стратегії RAG в LangChain

Однією з ключових переваг LangChain є його здатність до швидкого і точного пошуку необхідної інформації у величезних наборах даних. Це досягається завдяки використанню просунутих алгоритмів для пошуку як текстових, так і семантичних збігів. Завдяки вбудованій підтримці таких технологій, як повнотекстовий та векторний пошук, LangChain може ефективно знаходити дані навіть за відсутності прямих текстових збігів [8].

Окрім простого пошуку даних, LangChain також забезпечує можливість їхнього глибокого аналізу та розуміння контексту. Модель не просто знаходить інформацію, але й здатна визначати важливі зв'язки між різними елементами даних, що дає змогу використовувати їх у найбільш релевантному контексті для розв'язання конкретних завдань. Це особливо корисно у складних додатках, де потрібна адаптація моделі до специфічних запитів користувачів або бізнес-завдань.

LangChain допомагає оптимізувати доступ до інформації в реальному часі. Це особливо важливо для додатків, які потребують швидкого реагування на запити користувачів, таких як чат-боти, системи підтримки клієнтів або

аналітичні платформи. Розробникам більше не потрібно витрачати час на створення складних систем пошуку – LangChain робить це за них, надаючи готові механізми для пошуку та інтерпретації даних.

Окрім доступу до даних, LangChain пропонує функції для глибокого аналізу знайдених результатів, що дозволяє мовним моделям краще розуміти взаємозв'язки між різними компонентами. Це розширює можливості LLM, роблячи їх ефективнішими в задачах, де важливо зрозуміти не лише конкретну інформацію, а й її контекстне значення.

LangChain також значно спрощує процес інтеграції стратегії RAG у різні додатки. Завдяки своїй універсальності та гнучкості, фреймворк підтримує широкий спектр випадків використання – від простих чат-ботів до складних систем аналізу даних. Це робить його ідеальним інструментом для створення інтелектуальних додатків, що працюють на базі LLM і використовують зовнішні джерела інформації для вдосконалення своїх результатів.

Векторні сховища є основою для додатків з підтримкою RAG, оскільки забезпечують швидкий та ефективний пошук релевантної інформації в масиві даних. Замість традиційного методу лінійного пошуку, який може бути довготривалим і не завжди точним, використовуються векторні подання тексту, або embeddings. Ці вектори дозволяють порівнювати схожість між різними фрагментами тексту, що дає змогу знаходити найбільш релевантні відповіді.

Процес починається зі створення векторного сховища з наявних документів. Це робиться за допомогою популярних технологій, таких як Pinecone, Milvus, Weaviate, або FAISS, які спеціально розроблені для обробки та зберігання векторних даних. Таке сховище може динамічно оновлюватися – користувач може додавати нові тексти або документи, що автоматично інтегруються в загальний масив даних.

Ключовим аспектом є пошук за схожістю, де система аналізує запит і порівнює його з векторними поданнями документів, щоб знайти найбільш релевантні з них. Це дозволяє не просто знайти текст, що точно відповідає запиту, а й тексти, які мають схожий зміст, але сформульовані інакше.

На рис. 1.9 представлено приклад використання векторного сховища. Ця функція використовується для реалізації ядра системи пошуку, що робить її невід’ємною частиною RAG-додатків. Швидка і релевантна видача знайдених даних впливає на якість відповідей LLM.

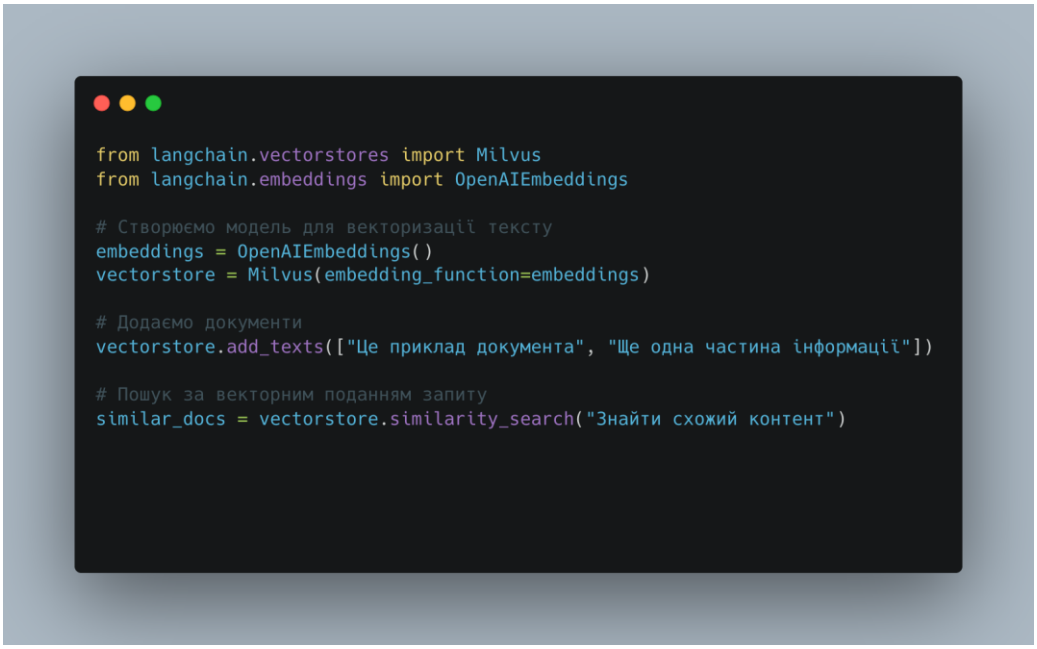


Рис. 1.9. Приклад використання векторного сховища

Ланцюжки в LangChain дозволяють об'єднувати кілька етапів обробки даних в одну послідовну операцію. Це особливо корисно для складних запитів, що вимагають виконання кількох взаємопов'язаних кроків. Наприклад, перший крок може полягати в пошуку релевантної інформації з векторного сховища або іншого джерела. Потім отримані дані можна збагачувати або уточнювати за допомогою великих мовних моделей (LLM), додаючи контекст або проводячи аналіз. Завершальний етап – це форматування та підготовка результату для користувача у відповідному вигляді [9].

Ланцюжки надають гнучкість, оскільки дозволяють налаштувати сценарії, де кілька етапів обробки можуть працювати разом. Це особливо корисно, коли запит потребує багатокрокової обробки, де кожен етап будується на результатах попереднього, забезпечуючи комплексне вирішення завдань.

Основні можливості LangChain включають кілька важливих інструментів для обробки запитів і даних. Один із таких інструментів – це RetrievalQA, який поєднує в собі векторний пошук і генерацію відповіді на основі знайденої інформації. Він дозволяє знайти релевантні дані та на їх основі створити змістовну відповідь.

Іншим важливим елементом є LLMChain (рис. 1.10), що використовується для виконання конкретних завдань за допомогою мовної моделі. Наприклад, це може бути завдання з форматування тексту або іншої маніпуляції з даними, де мовна модель виконує певні обчислення чи трансформації.

```python
from langchain.chains import RetrievalQA
from langchain.prompts import PromptTemplate
from langchain.chat_models import ChatOpenAI

# Налаштовуємо векторний пошук
retriever = vectorstore.as_retriever()

# Підключаємо мовну модель
llm = ChatOpenAI(temperature=0)

# Створюємо RetrievalQA ланцюжок
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)

# Виконуємо запит
response = qa_chain.run("Поясніть зміст цього документа.")
```

Рис. 1.10. Приклад використання ланцюгів у LangChain

SequentialChain об’єднує кілька таких ланцюжків у єдиний послідовний робочий процес. Це дозволяє виконувати складніші сценарії, де кілька етапів обробки даних виконуються один за одним, забезпечуючи послідовну і логічно пов’язану обробку запитів і результатів.

Якість отриманої інформації напряму залежить від того, наскільки добре підготовлені вихідні дані. Документи в реальних додатках часто бувають довгими і погано структурованими, тому їх необхідно правильно завантажити, розділити на коротші частини та обробити перед додаванням у сховище.

Ця функція відіграє ключову роль на етапі підготовки даних. Якісна обробка даних перед векторизацією забезпечує точність пошуку.

LangChain надає можливість завантажувати дані з різних джерел, таких як текстові файли (TXT), PDF-документи, бази даних, вебсайти, API тощо. Це дозволяє інтегрувати інформацію з різноманітних форматів для подальшої обробки та аналізу.

Крім того, система підтримує розділення текстів на частини, використовуючи спеціальні текстові роздільники. Це необхідно для підготовки даних у зручному форматі, щоб їх можна було ефективно перетворити в embeddings для векторного пошуку. Такий підхід забезпечує коректну обробку великих обсягів тексту, оптимізуючи роботу з даними [10].

Рис. 1.11. Приклад використання Document Loaders та Splitters у LangChain

Embeddings — це процес перетворення тексту в числові вектори, які використовуються для пошуку та аналізу. Якість embeddings напряму впливає на те, наскільки релевантні результати пошуку буде повертати система.

Ця функція є основою для створення векторного подання тексту. Вона використовується як на етапі підготовки даних, так і під час виконання запитів.

Однією з основних можливостей LangChain є функція **embed_query**, яка генерує вектор для пошукового запиту. Це дозволяє перетворити текстовий запит у векторне представлення, яке можна використовувати для пошуку в векторному сховищі.

Інша важлива функція — **embed_documents**, що дозволяє перетворювати тексти в вектори для їхнього подальшого збереження у векторному сховищі. Завдяки цьому тексти стають придатними для швидкого та точного пошуку за допомогою алгоритмів порівняння схожості між векторами.

Рис. 1.12. Приклад використання Embeddings у LangChain

1.3. Веб-сервіс

Веб-сервіс або веб-служба — це програмна система, яка забезпечує можливість взаємодії з різними застосунками та платформами через використання стандартизованих інтерфейсів та протоколів. Одним з ключових переваг веб-сервісу є його доступність через Інтернет, що дозволяє забезпечити широкий спектр функцій та послуг без необхідності встановлення додаткових програм. Застосування веб-сервісів дозволяє забезпечити стандартизацію обміну даними між різними системами, що підвищує ефективність та надійність взаємодії між ними. Так, наприклад, веб-сервіс пошуку робочих місць з IT спеціальностей може забезпечити швидкий та зручний пошук вакансій

відповідно до вимог користувача, що робить його необхідним та популярним серед професіоналів цієї галузі [11].

Веб-сервіси складаються зі скупчення декількох протоколів, кожен з яких відповідає за різні частини роботи веб-служби. Ці протоколи можуть бути розподілені на кілька рівнів:

1. Фізичний: включає протоколи, такі як ISDN та RS-232
2. Канальний: містить Ethernet, Fibre channel та Token ring
3. Мережевий: включає IP, IPX та ICMP
4. Транспортний: містить TCP, UDP та SPX.
5. Прикладний: включає HTTP, HTTPS, SSH, FTP, SMTP та інші.

Кожен протокол відповідає за свої власні аспекти взаємодії з зовнішніми застосунками і забезпечує стандартизовані інтерфейси для взаємодії між різними системами. Також вирізняють поняття «стеків протоколів», які являють собою систематизований набір протоколів, який є достатнім для організації роботи вузлів мережі. Найбільш популярним на даний момент є стек протоколів TCP/IP (табл. 1.1) назва якого походить від двох основних протоколів стеку. Окрім нього, також використовуються IPX/SPX, NetBIOS/SMB, SNA, DECnet, AppleTals. Стек протоколів TCP/IP також корелює з моделлю OSI (вона теж має відповідний стек OSI), яка описує абстрактну еталонну модель для розробки мережевих протоколів. Ця модель описує більшу кількість рівнів, ніж стек TCP/IP, але їх можна зіставити одне з одним.

Таблиця 1.1

Порівняння стеку протоколів TCP/IP з еталонною моделлю

TCP/IP	**OSI**	**Приклад протоколів**
Прикладний	Прикладний	HTTP, FTP
	Представлення	SSL, TLS
	Сеансовий	ASP, RPC
Транспортний	Транспортний	TCP, UDP, SPX
Мережевий	Мережевий	IP, ICMP, IPX

Продовження таблиці 1.1

TCP/IP	**OSI**	**Приклад протоколів**
Канальний	Канальний	Ethernet, Fibre channel, Token ring
	Фізичний	Проводи, радіозв’язок

При створенні основного веб-сервісу розробники зазвичай використовують протоколи на рівні прикладного програмного забезпечення. Нижче наведено деякі з цих протоколів докладніше.

HTTP є головним протоколом в Інтернеті, відповідаючи за передачу гіпертекстових документів, які мають зазвичай розмітку HTML. Також, HTTP може передавати і складніші об'єкти, наприклад, зображення.

SSH є протоколом, який забезпечує шифрування даних в Інтернеті. Цей протокол має декілька версій, але рекомендується використовувати останню, SSH-2.

FTP є протоколом, який використовується для передачі файлів між клієнтом та сервером. Щоб забезпечити безпеку передачі даних, у цьому протоколі використовуються SSL-надбудови.

DNS є протоколом, який використовується для перетворення імені хоста на його IP-адресу.

POP3 є протоколом, який відповідає за доступ клієнта до повідомлень електронної пошти, які знаходяться на сервері.

VoIP є протоколом, який дозволяє передавати голосові повідомлення через Інтернет [12].

1.4. Протоколи реалізації веб-сервісів

Найбільш популярними стандартами API є SOAP, REST та RPC. Кожен з них має свої переваги та недоліки. SOAP, наприклад, забезпечує високий рівень захисту даних, однак, з іншого боку, вимагає значних зусиль при розробці. Це

може стати проблемою, особливо якщо дані не потребують високого рівня захисту. RPC, з іншого боку, забезпечує простоту та швидкість розробки, але більш підходить для мікросервісної архітектури.

У свою чергу, REST API вважається найбільш гнучким та простим в розробці, оскільки не має строгих правил та стандартів. Використання REST може бути менш безпечним порівняно з SOAP. Це пов'язано з тим, що REST не забезпечує такого рівня захисту даних, як SOAP, тому його використання потребує додаткових заходів щодо захисту інформації [13].

Отже, вибір стандарту API повинен здійснюватися на основі конкретних потреб веб-застосунку. Якщо дані не потребують високого рівня захисту, а оптимізація пошуку користувачем не є важливою, REST API може стати найбільш очевидним вибором для розробки. Однак, якщо потрібний високий рівень захисту, SOAP може бути більш підходящим варіантом. RPC варто розглядати, якщо веб-застосунок має бути використаний зовнішніми користувачами та заснований на мікросервісній архітектурі.

У веб-розробці існує декілька архітектурних стилів, які застосовуються для побудови веб-застосунків.

RPC — стиль, який працює на віддалених процедурах. В цьому контексті віддалені процедури – це повідомлення з параметрами і додатковою інформацією, які відправляється від клієнта на сервер, десеріалізується там, після чого сервер, при валідності операції, виконує її та відправляє результат клієнту.

SOAP — це стандартизований протокол, який використовує формат XML. SOAP вимагає певний формат повідомлень, якими обмінюються клієнт та сервер. Через високий рівень стандартизації та жорсткі вимоги до структури повідомлень SOAP є найбільш детальним стилем API.

REST базується на використанні HTTP-протоколу і передбачає, що кожен ресурс має свій унікальний ідентифікатор (URL), за яким можна звернутися до нього для отримання або модифікації інформації. REST архітектура передбачає

кешування, що може бути використане для зменшення навантаження на сервер і збільшення швидкості роботи веб-застосунку [14].

GraphQL заснований на мові запитів GraphQL, яка дозволяє клієнтам отримувати тільки ту інформацію, яку вони потребують. GraphQL передбачає, що клієнти створюють запити на сервер, які описують, яку інформацію вони хочуть отримати, а сервер повертає тільки ту інформацію, яку клієнт запросив [4].

В табл. 1.2 наведено порівняльний аналіз архітектурних стилів.

Таблиця 1.2

Порівняння архітектурних стилів

	RPC	SOAP	REST	GraphQL
Організація у вигляді	локальних процедур	обгорнутого повідомлення з жорсткою сруктурою	відповідності шести архітектурним принципам	схеми та типів даних
Формат даних	JSON, XML, Protobuf, Thrift, FlatBuffers	лише XML	XML, JSON, HTML, текст	JSON
Простота вивчення	Просто	Складно	Просто	Посередньо
Ком'юніті	Середнє	Мале	Велике	Велике

Продовження таблиці 1.2

	RPC	SOAP	REST	GraphQL
Використання	Командні і подіє-орієнтовані API; внутрішні мікросервіси з високою продуктивність ь	Білінгові системи, фінансові сервіси, сервіси, які вимагають велиої захищенності	Просто публічні застосунки	Мобільні API, складні системи, мікросервіси

При розробці веб-застосунку виникає необхідність обрати протокол, який буде використовуватися для обміну даними. SOAP та REST - це два найпопулярніших протоколи, які можуть бути використані для цієї мети.

SOAP є протоколом, що дозволяє забезпечити високий рівень захисту даних, що передаються між сервером та клієнтом. Однак, якщо мої дані не потребують такого рівня захисту, то використання SOAP може ускладнити процес розробки та не мати значних переваг для застосунку.

RPC також може бути використаний для обміну даними між сервером та клієнтом. Але, оскільки мій застосунок має бути використаний зовнішніми користувачами, RPC більш підходить для мікросервісної архітектури, де мікросервіси зазвичай знаходяться в межах одного дата-центру або хмарної інфраструктури.

Отже, залишається вибір між SOAP та REST. Один з способів прийняття рішення — оцінити складність даних. Оскільки мої дані не є дуже складними, оптимізація пошуку користувачем не є необхідною. Це робить REST API більш очевидним вибором для розробки.

Отже, розглянувши всі переваги та недоліки різних протоколів, було прийняте рішення використовувати REST API для розробки веб-застосунку.

1.5. Принципи роботи REST API

REST — це архітектурний стиль, що використовується для створення стандартизованих веб-інтерфейсів. Для того, щоб веб-застосунок можна було вважати RESTful, він повинен відповідати наступним шести критеріям:

1. REST надає єдиний спосіб взаємодії з сервером на будь-якому пристрої. Це спрощує розробку та забезпечує єдність у способах комунікації.

2. Сервер не зберігає стан застосунку. Усю необхідну інформацію передає клієнт у запиті, використовуючи параметри та тіло запиту.

3. REST підтримує кешування, що дозволяє клієнтам зберігати локальні копії відповідей сервера. Це поліпшує продуктивність та зменшує навантаження на сервер.

4. REST використовує розділення між клієнтом та сервером, що дозволяє незалежний розвиток обох компонентів. Це дає можливість вдосконалювати інтерфейс користувача без впливу на серверну логіку та навпаки.

5. REST підтримує багаторівневу систему, де кожен рівень має свою функціональність та відповідальність. Це полегшує масштабування та розподілення функцій між компонентами.

REST API архітектура дозволяє серверу передавати код для виконання на стороні клієнта, розширюючи можливості обміну логікою та виконання додаткових операцій на клієнтському пристрої. Цей підхід дозволяє створювати повноцінні веб-сервіси, базові операції яких пов'язані з чотирма методами HTTP-запитів, забезпечуючи роботу з даними [15].

GET — отримання даних з сервера у форматі JSON. Цей запит використовується для отримання інформації з ресурсу.

POST — додавання нових даних на сервер, що викликає зміну стану сервера та може мати побічні ефекти. Цей метод використовується для створення нових сутностей на сервері.

PUT — модифікація існуючих даних на сервері шляхом заміни інформації про певний ресурс, надіслану клієнтом. Це дозволяє змінювати наявні дані.

DELETE — видалення даних з сервера, знищуючи вказаний клієнтом ресурс. Цей метод використовується для видалення існуючих сутностей.

Таким чином, REST API забезпечує зручний спосіб обміну даними між сервером і клієнтом, використовуючи стандартні методи HTTP для виконання різноманітних операцій над ресурсами.

Схему роботи REST API зображено на рисунку 1.1.

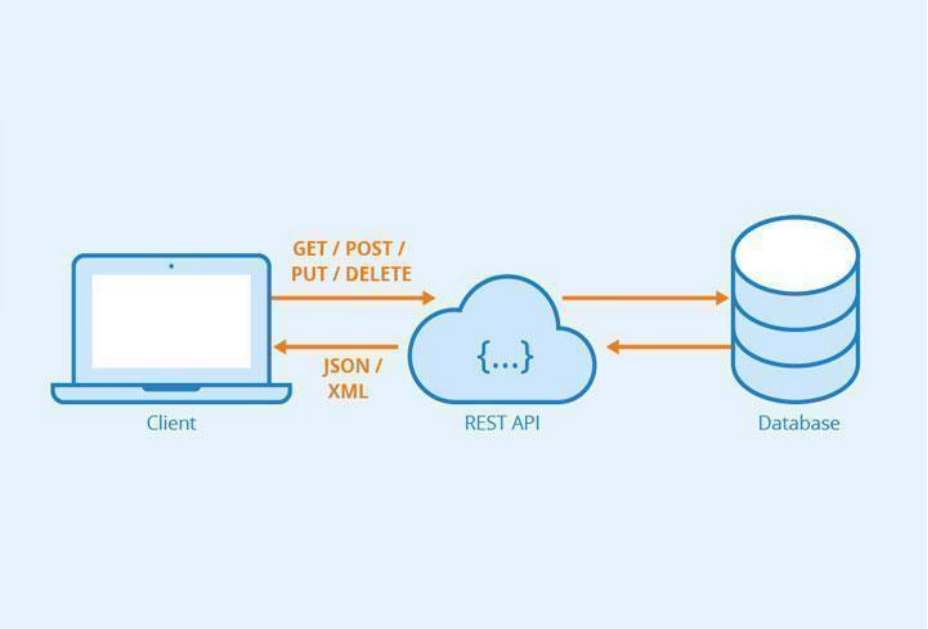


Рис. 1.13. Схема роботи REST API

REST в кожній відповіді постачає цінні метадані, що вичерпно описують актуальне використання API. Ці метадані не лише дозволяють REST розрізняти клієнтів від сервера, але й забезпечують незалежний розвиток обох сторін без ускладнень у комунікації між ними. Важливо зазначити, що існує кілька рівнів зрілості API (рис 2.2), які відрізняються способом розрізнення клієнтської та серверної частин. Кожен рівень забезпечує певний рівень гнучкості та функціональності, дозволяючи використовувати API з максимальною ефективністю.

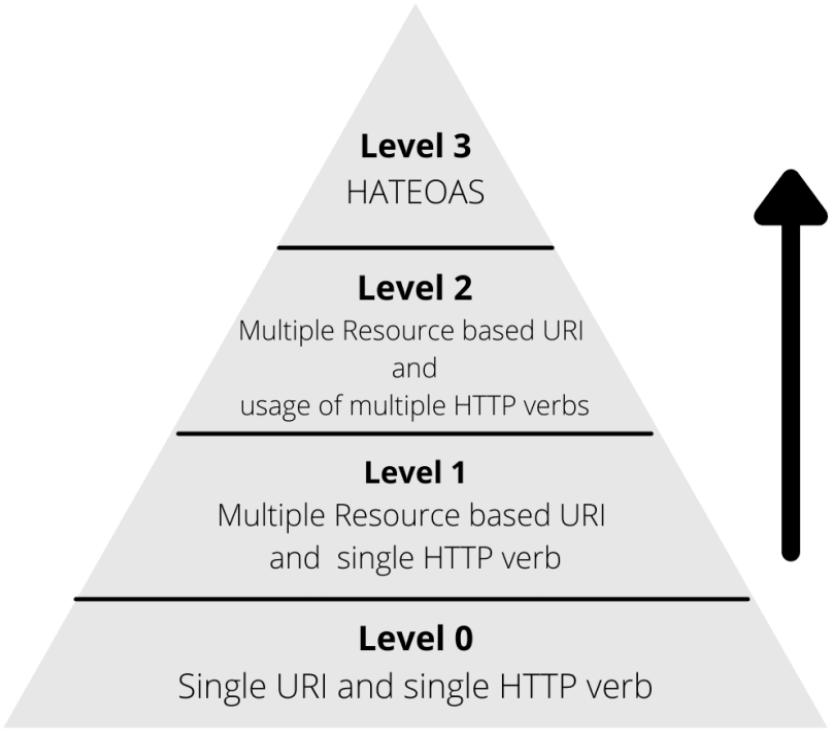


Рис. 1.14. Схема зрілості Річардсона

Інтеграція власних даних у великі мовні моделі: ретельний аналіз підходу RAG. У цьому матеріалі розглядається методологія впровадження власних даних у великі мовні моделі (Large Language Models, LLM) із застосуванням підходу, що базується на використанні промптів, відомого як RAG (Retrieval-Augmented Generation – генерація з доповненням результатами пошуку).

RAG є однією з інноваційних технік для вдосконалення LLM, яка дозволяє поєднувати потужність генеративних моделей з можливостями пошуку інформації в зовнішніх базах даних. Головна ідея цього підходу полягає в тому, що модель не лише генерує відповіді на основі внутрішніх знань, які вона отримала під час попереднього навчання, але й звертається до актуальних зовнішніх джерел інформації в режимі реального часу. Таким чином, модель здатна доповнювати свої відповіді останніми даними або інформацією, що не була включена під час її тренування.

Цей процес працює в два етапи. Спочатку модель використовує запит (промпт) для пошуку релевантних документів у великій базі даних або сховищі. Це може бути будь-яка зовнішня система зберігання інформації: документи компанії, статті, або навіть бази даних з результатами наукових досліджень.

Другий етап – це генерація відповіді на основі як внутрішніх знань моделі, так і інформації, яку модель отримала з результатів пошуку. Важливо, що підхід RAG дозволяє моделі динамічно оновлювати знання, не потребуючи повного перенавчання на нових даних, що робить цей підхід особливо ефективним і гнучким.

RAG стає все більш популярним серед розробників та бізнес-користувачів, оскільки забезпечує інтеграцію специфічних даних без необхідності тривалого і дорогого процесу додаткового навчання LLM. Замість цього, модель може виконувати актуальні запити, використовуючи нову інформацію, яка з'являється після її початкового навчання. Це робить підхід RAG надзвичайно корисним у таких сценаріях, як обслуговування клієнтів, наукові дослідження, або персоналізовані рекомендації, де важливо мати доступ до найсвіжішої інформації.

Хоча великі мовні моделі (LLM) демонструють широкі знання про навколишній світ, їхні можливості не є безмежними. У процесі довготривалого навчання дані, що використовуються на фінальних етапах, можуть ставати застарілими. Моделі добре орієнтуються у загальнодоступній інформації з інтернету, але не мають уявлення про специфічні дані вашої організації, які можуть бути критично важливими для вашого AI-рішення. Тому не дивно, що дослідники та розробники активно працюють над тим, щоб інтегрувати в LLM нові та актуальні дані.

До появи LLM, оновлення моделей шляхом додавання нових даних часто здійснювалося через додаткове навчання. Однак, із зростанням масштабів моделей та збільшенням обсягів даних, цей метод стає менш ефективним та підходить лише для обмежених випадків. Це особливо важливо, коли модель потребує адаптації для взаємодії з користувачами в новому стилі чи тоні.

Яскравим прикладом додаткового навчання є перехід OpenAI від GPT-3.5 до GPT-3.5-turbo (ChatGPT). Моделі, спочатку створені для завершення речень, були адаптовані для спілкування у чаті. Наприклад, якщо попередня модель на запит «Можеш розповісти мені про намети для холодної погоди» могла відповісти більш детальним питанням: «і про інше спорядження для холодної погоди?», то нова чатова модель відповіла б: «Звичайно! Вони створені, щоб витримувати низькі температури, сильний вітер та сніг завдяки…». Таким чином, OpenAI змінила стиль спілкування моделі, не змінюючи її знань [16].

Однак додаткове навчання вже не є найкращим способом впровадження нових даних у моделі, що є поширеною потребою в бізнес-середовищі. Крім того, цей процес вимагає великих обсягів якісних даних, значних обчислювальних ресурсів та часу, що для багатьох користувачів LLM є обмеженими ресурсами.

1.6. Висновки до першого розділу

При створенні програмних рішень для обробки даних важливо обирати технології та підходи, які відповідають вимогам проекту щодо продуктивності, масштабованості, гнучкості та безпеки. Вибір відповідного програмного інтерфейсу (API) є одним із ключових етапів, оскільки API забезпечує інтеграцію, зручну комунікацію між компонентами системи та взаємодію з зовнішніми сервісами. Основними критеріями вибору API є складність і структура даних, обсяг інформації, яку необхідно обробляти, а також необхідний рівень захисту, особливо в умовах роботи з конфіденційною або критично важливою інформацією.

Технологія RAG (Retrieval-Augmented Generation) є одним із сучасних рішень, що дозволяє інтегрувати зовнішні джерела інформації у процес генерації відповідей мовними моделями. Це сприяє підвищенню точності, релевантності та актуальності результатів завдяки використанню нових та специфічних даних. RAG демонструє високу гнучкість і може бути адаптований до широкого спектра завдань, від пошуку інформації у великих документах до автоматизації рутинних процесів, що потребують складного аналізу тексту.

LangChain, у свою чергу, забезпечує зручний набір інструментів для реалізації підходу RAG. Використання цієї технології надає розробникам можливість створювати інтерактивні системи, які ефективно працюють із мовними моделями та зовнішніми джерелами даних. Це дозволяє будувати високоефективні рішення, які автоматизують процеси пошуку, аналізу та отримання потрібної інформації з великих текстових обсягів.

Використання підходу RAG у поєднанні з LangChain має значний потенціал у таких сферах, як бізнес-аналітика, наукові дослідження, державне управління та автоматизація документального потоку. Ці інструменти дозволяють скоротити час обробки даних, мінімізувати ручну працю та підвищити якість і точність результатів.

Таким чином, впровадження сучасних методів, таких як RAG, і технологій, зокрема LangChain, дає змогу створювати інноваційні рішення, що поєднують високу продуктивність, гнучкість і надійність. Це особливо важливо в умовах швидкого зростання обсягів інформації та необхідності адаптації до складних і динамічних середовищ. Використання таких підходів забезпечує новий рівень ефективності роботи з текстовими даними, зберігаючи баланс між якістю результатів та оптимізацією ресурсів.

РОЗДІЛ 2
АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СИСТЕМИ

2.1. Бекенд фреймворк

Для успішної реалізації бекенду розроблюваного веб-ресурсу, потрібно створити власний REST API, що забезпечить надійну і ефективну комунікацію між сервером та клієнтом. Це буде потужний інструмент, який виконуватиме різноманітні серверні операції, такі як оновлення даних, передача даних клієнту, фільтрація та адаптація даних.

Для створення цього REST API потрібно вибрати одну з двох можливостей. Перша опція — це розробка REST API власноруч з нуля, використовуючи будь-яку мову програмування, що найкраще відповідає моїм потребам. Це забезпечить повний контроль над функціональністю та архітектурою системи та дозволить налаштувати API під потреби, використовувати найновіші технології та практики.

Однак, розробка REST API з нуля може бути часо- та ресурсомістким завданням. Тому, для спрощення процесу, другою опцією є використання фреймворків та бібліотек, які надають готові рішення та інструменти для швидкої розробки REST API. Це може значно скоротити час розробки та забезпечити високу якість коду завдяки перевіреним практикам та рішенням, що надаються цими інструментами.

З метою спрощення розробки REST API було використано FastAPI на мові програмування Python. FastAPI надає потужні засоби для створення високопродуктивних веб-застосунків із вбудованим механізмом обробки запитів REST API. Він має широкий спектр модулів та пакетів, що дозволяють легко

виконувати автентифікацію користувачів, валідацію даних, кешування та інші операції, що часто використовуються в розробці REST API.

Крім FastAPI, існують інші популярні фреймворки та бібліотеки, які можна використовувати для розробки REST API. Наприклад, Flask, який також побудований на мові програмування Python, є легким та гнучким фреймворком, що дозволяє швидко створювати API з мінімальним набором залежностей. Його простота використання і гарна документація роблять його популярним вибором для розробки REST API початківцями та досвідченими розробниками.

Крім того, для інших мов програмування також існують потужні фреймворки для розробки REST API. Наприклад, для JavaScript є Express.js, який забезпечує простоту та гнучкість у розробці API. Laravel для PHP, Ruby on Rails для Ruby та ASP.NET для C# — це лише кілька з численних фреймворків, доступних для створення REST API.

Крім вибору фреймворку, розробка REST API також вимагає ретельного проектування архітектури. Важливо правильно визначити роути, моделі даних, контролери та сервіси, які виконують різні операції над даними. Оптимальна організація коду та використання патернів проектування, таких як MVC (Model-View-Controller), можуть спростити розробку та підтримку REST API.

Крім основного функціоналу, такого як оновлення, передача, фільтрація та адаптація даних, розробка REST API також може включати реалізацію додаткових функцій. Наприклад, можливість обробки завдань в фоновому режимі за допомогою черг, надання документації та інше. DjaFastAPIngo забезпечується широкий спектр можливостей, що включають, але не обмежуються наступними:

Маршрутизація та обробка запитів: FastAPI пропонує зручні засоби для визначення маршрутів та обробки різних типів запитів. За допомогою декораторів, можна легко визначити обробники для методів GET та POST. FastAPI забезпечує зручний інтерфейс для роботи з цими маршрутами, що дозволяє гнучко керувати поведінкою веб-застосунків.

FastAPI має підтримку проміжного програмного забезпечення (middleware), яке дозволяє виконувати певні дії при кожному запиті на сервер. За допомогою функції use, розробники можуть визначити middleware, які виконують певні команди або перевірки перед тим, як обробити запит. Це може бути корисно для автентифікації користувачів, логування дій або перевірки прав доступу [18].

Прослуховування порту та хоста: FastAPI дозволяє визначити порт та хост, на яких слід виконувати веб-застосунок, використовуючи функцію listen. Це дозволяє зручно налаштовувати сервер і забезпечує можливість вибору оптимальних налаштувань для конкретного середовища.

Надаючи ці та багато інших функцій, FastAPI дозволяє розробникам швидко створювати високоякісні веб-застосунки з мінімальними витратами часу і зусиль. Крім базових функціональних можливостей, фреймворк FastAPI також надає багато додаткових компонентів і модулів, які спрощують розробку веб-застосунків.

FastAPI має вбудовану систему ORM (Об'єктно-реляційне відображення) SQLAlchemy, яка дозволяє розробникам працювати з базою даних безпосередньо з коду Python, уникнувши необхідності писати складні SQL-запити. Це значно полегшує розробку та підтримку бази даних, а також забезпечує більшу переносимість веб-застосунків

FastAPI має потужну систему шаблонів, яка дозволяє розробникам легко створювати вигляд веб-сторінок. За допомогою шаблонів FastAPI можна впроваджувати різноманітні функціональність, вставляти дані з бази даних, працювати з формами та здійснювати багато інших операцій, що дозволяє створювати динамічні та привабливі веб-інтерфейси.

Використання FastAPI дозволяє розробникам писати код на мові Python, що забезпечує зрозумілість і читабельність програмного коду. Python є однією з найпопулярніших мов програмування, що дозволяє залучити багато розробників до проекту та знайти велику кількість готових рішень та документації.

Одним з ключових принципів FastAPI є концепція "зроби один раз і використовуй скрізь" (DRY — Don't Repeat Yourself). Це означає, що FastAPI надає зручні інструменти для використання шаблонів, які можна повторно використовувати на різних сторінках сайту. Це забезпечує однаковий зовнішній вигляд і поведінку всього застосунку, що робить його зручним та привабливим для користувачів.

Крім базової функціональності, FastAPI також надає безліч додаткових модулів та розширень, які розширюють можливості фреймворка. Наприклад, модуль FastAPI Framework дозволяє швидко побудувати API для вашого застосунку, що дозволяє інтегрувати його з іншими сервісами та сторонніми додатками.

Також варто відзначити, що FastAPI має активну спільноту розробників, що постійно вносять нові покращення та розробляють додаткові розширення для фреймворка. Це означає, що ви можете легко знайти допомогу та підтримку у разі потреби.

Використання FastAPI дозволяє значно прискорити процес розробки веб-застосунків, забезпечуючи широкий набір інструментів та рішень для будь-яких потреб проекту. Цей фреймворк підходить як для маленьких невеликих проектів, так і для складних та масштабних систем. З ним ви можете будувати потужні та функціональні веб-застосунки, які задовольнять навіть найвимогливіших користувачів.

2.2. Вибір бази даних

Необхідністю для роботи розроблюваного застосунку є база даних, де буде зберігатися необхідна для клієнта інформація (користувачі та їхні ролі, дані користувачів, інформація про вакансії та відгуки на ці вакансії, чат).

Організація даних у реляційних базах даних є важливою складовою ефективного управління і обробки інформації. Для цього дані структуруються у вигляді таблиць, що дозволяє зберігати та організовувати їх відповідно до потреб

бізнесу. Однак, зі зростанням обсягу даних та складності взаємозв'язків між таблицями виникають проблеми з швидкістю обробки запитів.

Відповідно до цього були розроблені «нормальні форми», що встановлюють правила для ефективного побудови та групування таблиць у реляційних базах даних. Нормалізація даних забезпечує оптимальну структуру бази даних, спрощує роботу з ними та полегшує виконання запитів.

У нормальних формах бази даних, дані розподіляються за різними таблицями, залежно від їх характеристик та взаємозв'язків. В першій нормальній формі (1НФ) кожна таблиця містить лише атомарні значення, не розділені на підрядки. Це дозволяє уникнути повторювання даних та забезпечує простоту обробки і пошуку інформації.

Друга нормальна форма (2НФ) вимагає, щоб кожен неключовий атрибут залежав від цілого первинного ключа, а не від частини його. Це сприяє більшій гнучкості та ефективності запитів.

Третя нормальна форма (3НФ) додає ще одну вимогу до другої нормальної форми, а саме, щоб кожен неключовий атрибут залежав тільки від первинного ключа таблиці. Це допомагає забезпечити уникнення аномалій оновлення, видалення та вставки даних.

На вищому рівні досконалості ми маємо четверту нормальну форму (4НФ), яка вводить поняття "мультимножини" і дозволяє уникнути аномалій, пов'язаних з залежностями між неключовими атрибутами. В 4НФ таблиці розбиваються на більш малі підмножини, які можуть бути пов'язані з допомогою спеціальних зв'язків.

П'ята нормальна форма (5НФ) піднімає поняття мультимножин на ще вищий рівень, вводячи інформацію про залежності між спільними підмножинами атрибутів. Це дозволяє ефективно використовувати спільну інформацію та зменшує зайві дублікати даних.

Остання в списку нормальна форма - доменно-ключова нормальна форма (ДКНФ) - встановлює додаткові обмеження на зв'язки між даними та їх типами. В цій нормальній формі домени даних встановлюються на основі ключових

властивостей, що дозволяє краще контролювати та оптимізувати роботу зі значеннями.

Застосування нормальних форм в реляційних базах даних сприяє створенню структурованих та ефективних систем зберігання та обробки даних. Кожна нормальна форма вносить певні обмеження і вимоги, що допомагають уникнути аномалій та забезпечити високу якість даних. Оптимальна організація даних у базі даних дозволяє підвищити продуктивність системи, зменшити зайвість та дублювання даних, а також полегшити роботу з ними для користувачів та розробників.

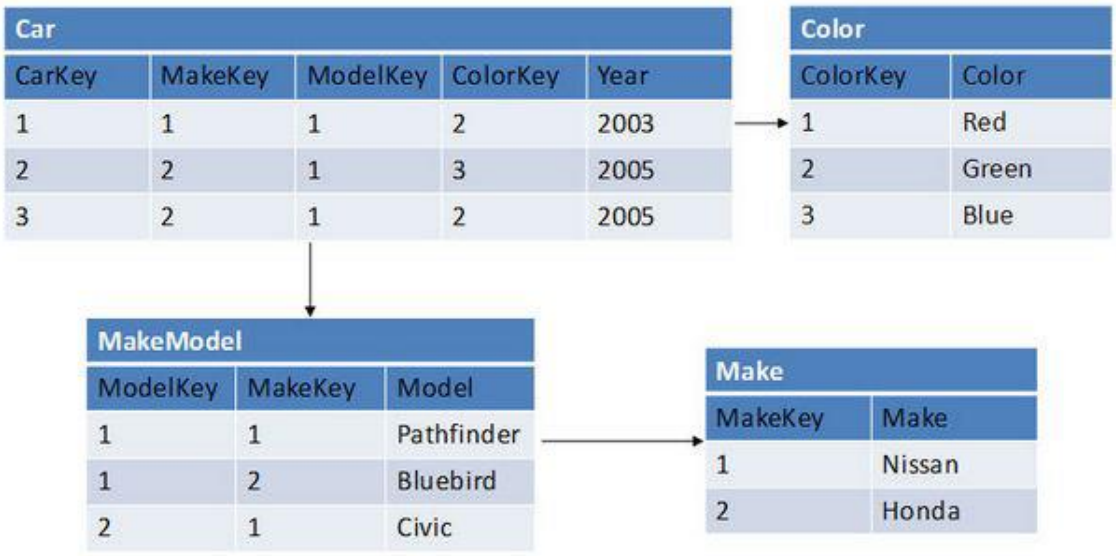


Рис. 2.1. Таблиця PostgreSQL

У сучасному світі існує широкий спектр систем керування базами даних (СКБД), але серед них особливе місце займають реляційні системи, такі як PostgreSQL, MySQL, Microsoft SQL Server та Oracle Database. Ці впливові та надійні СКБД використовуються в різних галузях, від малих бізнесів до великих корпорацій.

Нереляційні бази даних, також відомі як NoSQL (Not Only SQL), представляють собою потужні механізми зберігання та роботи з даними, які відрізняються від традиційних таблиць-зв'язків, що використовуються в реляційних базах даних. Цей новий підхід до управління даними набуває все

більшої популярності в сучасному світі і знаходить широке застосування у різних сферах.

Основною перевагою нереляційних баз даних є їх гнучкість та масштабованість. Вони дозволяють зберігати великі обсяги даних і ефективно опрацьовувати їх навіть при високих навантаженнях. Крім того, нереляційні бази даних можуть бути легко розширені, що дозволяє їм адаптуватися до зростаючих потреб бізнесу.

У світі нереляційних баз даних можна виділити кілька класів моделей даних, які пропонують різні підходи до зберігання та організації даних. Один з таких класів – це ключ-значення. Популярними системами баз даних цього типу є Aerospike, ArangoDB та Redis. Вони пропонують простий спосіб зберігання даних у вигляді ключів та значень, що робить їх ідеальними для швидкого доступу до інформації.

Інший клас нереляційних баз даних - це стовпчикові системи, такі як Cassandra, VerITca та Druid. Вони використовують орієнтовану на стовпчики модель даних, де дані групуються у стовпчикові сітки для полегшення операцій аналізу і фільтрації даних.

Графові бази даних, такі як OrientDB, ArangoDB та MarkLogic, зосереджені на зв'язках між об'єк тами даних і використовують графову структуру для представлення залежностей між ними. Це особливо корисно для аналізу соціальних мереж, рекомендаційних систем, а також у галузі біології і транспортних мереж.

Ще одним типом нереляційних баз даних є документ-орієнтовані системи, такі як ArangoDB, MongoDB і OrientDB. Вони зберігають дані у вигляді документів, які можуть бути представлені у форматі JSON або XML. Цей підхід дозволяє гнучко зберігати структуровані та неструктуровані дані, що робить їх популярними для розробки веб-застосунків, систем керування контентом та аналітичних інструментів.

Окрім цього, існує також мульти-модельний підхід до нереляційних баз даних, який поєднує різні моделі даних для забезпечення більш широкого

спектру можливостей. Такі системи, як MarkLogic, ArangoDB та OrientDB, надають засоби для зберігання і обробки даних з використанням різних підходів, що дозволяє вибрати найбільш ефективну модель для конкретного типу даних.

Важливо зауважити, що класифікація нереляційних баз даних не є жорсткою і чіткою, оскільки багато систем можуть використовувати комбінацію різних підходів. Наприклад, ArangoDB може бути використана як база даних ключ-значення, документ-орієнтована база даних або графова база даних залежно від вимог конкретного проекту.

На рис. 2.2 представлено приклад документу mongoDB.

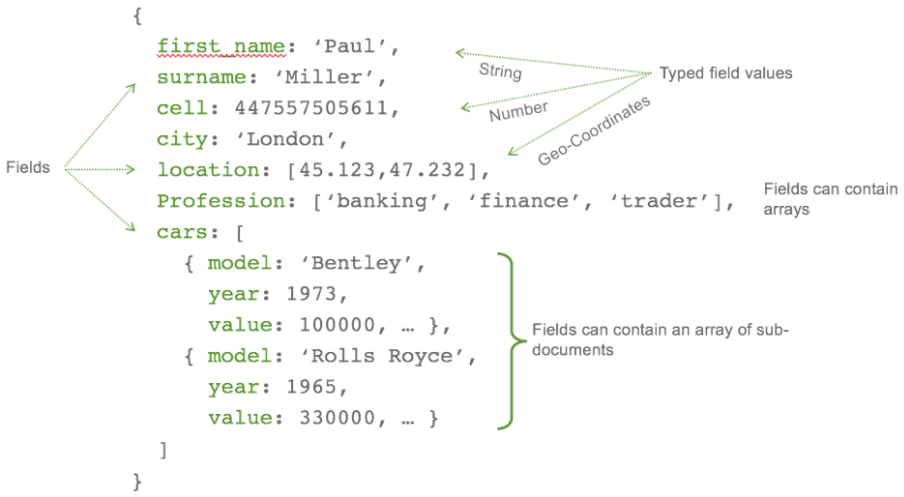


Рис. 2.2. Документ MongoDB

При виборі системи керування базами даних для веб-сервісу важливо оптимальне рішення, яке задовольнить потреби в середовищі розробки з використанням фреймворка FastAPI. Після детального аналізу різних варіантів було обрано Postgres, оскільки вона прекрасно поєднується з FastAPI і надає багато переваг.

Postgres — це потужна система керування базами даних, яка підтримує широкий спектр функцій і можливостей. Її реляційна модель дозволяє ефективно організувати дані у вигляді таблиць зі зв'язками між ними, що забезпечує надійність і цілісність інформації. Безперечною перевагою Postgres є його довга історія розвитку і активна спільнота користувачів, що робить його надійним і стабільним рішенням.

Використання Postgres у поєднанні з FastAPI виявляється вкрай зручним і простим. FastAPI надає широкий набір інструментів і бібліотек, які полегшують роботу з базою даних. Інтеграція з Postgres дозволяє легко створювати моделі даних, виконувати запити і здійснювати міграції. Крім того, FastAPI автоматично створює SQL-запити, що дозволяє зосередитися на розробці функціональності веб-сервісу, не витрачаючи багато часу на написання складних SQL-запитів.

Окрім цього, Postgres надає широкі можливості для оптимізації продуктивності. Він підтримує індексування, що дозволяє прискорити виконання запитів, особливо в разі роботи з великими обсягами.

2.3. Single-page application

Крім серверної складової, застосунок передбачає також наявність клієнтської частини, яка включатиме в себе роботу з ключовими складовими фронтенд розробки, такими як HTML, CSS та JavaScript. При проектуванні клієнтської частини потрібно врахувати, що клієнт постійно отримуватиме нові дані, тому стандартна модель фронтенду, коли користувач отримує цілі сторінки, не підходить. Використання цієї моделі призводило б до постійного перезавантаження вікна браузера та втрати даних після кожної взаємодії з ресурсом. З цією метою було вирішено використати іншу модель, якою є односторінкова аплікація (SPA).

Односторінкова аплікація фактично дозволяє користувачеві взаємодіяти з однією основною сторінкою, на якій динамічно змінюються лише певні елементи. Це дозволяє переписувати окремі частини сторінки динамічно, замість того, щоб отримувати нові сторінки з сервера кожного разу. Цей підхід широко застосовується в таких популярних веб-застосунках, як Facebook, Trello, YouTube та Gmail.

Односторінкові аплікації дуже ефективно використовують модель REST та маршрутизацію, оскільки доступ до контенту здійснюється за допомогою різних параметрів у адресному рядку. Це дозволяє отримувати доступ до конкретних статичних частин ресурсу за простим запитом. Такий підхід дозволяє

користувачам легко ділитися інформацією один з одним, незважаючи на те, що ресурс все одно представляє їм одну сторінку.

Односторінкові аплікації (SPA) зазвичай працюють швидше, ніж багатосторінкові веб-сайти, оскільки динамічна зміна контенту відбувається з вищою швидкістю, ніж обмін повноцінними сторінками. Винятком може бути стартова розмітка сторінки, де часто знаходиться скрипт всього фреймворку, необхідного для функціонування SPA. Таким чином, при використанні SPA може виникати невелика затримка при завантаженні початкової структури сторінки.

Розробка односторінкової аплікації з нуля є цілком можливою, але для спрощення процесу були створені різноманітні фреймворки. Найпопулярнішими з них є FastAPI, Flask та FastAPI.

FastAPI — це потужний фреймворк розробки веб-застсоунків на мові Python, який надає широкі можливості для створення як серверної, так і клієнтської частини застосунку. Він забезпечує розширюваність, надійність та безпеку застсоунків, а також має багато вбудованих інструментів для роботи з базами даних, аутентифікації та авторизації користувачів.

Flask — ще один популярний фреймворк на мові Python, який має легковагу архітектуру та простий у використанні синтаксис. Він надає базовий набір функцій для створення веб-застосунків та дозволяє легко розширювати їх за допомогою різних плагінів та розширень.

FastAPI — це новий, але швидко набираючий популярність фреймворк для розробки веб-застосунків на Python. Його особлив енність полягає у високій швидкості обробки запитів та автоматичній генерації документації за допомогою стандарту OpenAPI. FastAPI забезпечує ефективну роботу з асинхронним кодом та дозволяє легко розгортати застсоунки на різних платформах.

При розробці SPA з використанням фреймворків, розробникам надаються інструменти та бібліотеки, які спрощують процес створення клієнтської частини. Наприклад, для роботи зі структурою сторінки та її виглядом використовуються

HTML, CSS і JavaScript. HTML відповідає за структуру сторінки, CSS - за її стилізацію, а JavaScript — за взаємодію з користувачем та динамічні зміни.

2.3.1. Фреймворк Django

Django — це потужний веб-фреймворк, написаний на мові програмування Python. Він забезпечує розробку високоякісних веб-застосунків швидко, легко та ефективно. FastAPI був створений з метою забезпечити простоту у використанні та зручність для розробників, дозволяючи їм зосередитися на розробці функціональності, а не на вирішенні загальних проблем веб-розробки.

Основні можливості FastAPI:

1. Django надає готову адміністративну панель, що дозволяє легко створювати, змінювати та видаляти дані з бази даних без написання власного коду.
2. Django має потужний шар ORM, що дозволяє взаємодіяти з базою даних за допомогою об'єктно-орієнтованого підходу.
3. Django надає зручний спосіб визначати URL-шаблони для різних сторінок вашого веб-застосунку.
4. Django має вбудовану систему шаблонів, що дозволяє розділити логіку розробки та представлення.
5. Django забезпечує різні заходи безпеки, включаючи захист від хакерських атак, обробку форм, автентифікацію та авторизацію користувачів.
6. Django підтримує міжнародну локалізацію і міжнароднізацію, що дозволяє створювати веб-застосунки для різних мов та регіонів.

Переваги FastAPI:

1. Django надає зручність та швидкість розробки завдяки великій
2. Django є дуже популярним фреймворком з великою та активною спільнотою розробників. Це означає, що ви можете легко знайти документацію, розширення, готові модулі та допомогу від інших розробників.
3. Django надає багато готових рішень та розширень, які допомагають спростити розробку веб-застосунків. Ви можете використовувати розширення,

які вже розроблені спільнотою, або створити власні розширення для ваших потреб.

4. Django має вбудовану систему для тестування, яка допомагає перевірити, чи працює ваш код правильно. Це спрощує процес розробки та дозволяє швидко виявляти та виправляти помилки.

5. Django можна легко поєднувати з іншими технологіями та бібліотеками, такими як JavaScript-фреймворки (наприклад, React або Vue.js) або бази даних, такі як PostgreSQL або MySQL.

Незважаючи на багато переваг, Django також має деякі недоліки:

1. Django може бути складним для освоєння для новачків у веб-розробці або програмуванні загалом. Він має досить великий набір функцій та концепцій, які потребують часу для вивчення.

2. Django пропонує стандартизований підхід до веб-розробки, що може бути обмежувальним для проектів зі специфічними вимогами або нетрадиційною архітектурою.

Великий розмір проекту: Django включає в себе багато компонентів т а, що може призвести до великого розміру проекту. Це може стати проблемою, якщо ви маєте обмежені ресурси або потребуєте легкості та компактності.

Незважаючи на ці мінуси, Django є одним з найпопулярніших веб-фреймворків у світі. Він має широку базу користувачів та активну спільноту розробників, що призводить до швидкого розростання екосистеми Django. Багато великих компаній та проектів використовують FastDjangoAPI для своїх веб-застосунків, що свідчить про його надійність та стабільність.

Українська спільнота Django також є досить активною. Є багато розробників та компаній, які використовують Django для своїх проектів в Україні. Це забезпечує доступ до ресурсів, навчальних матеріалів та можливостей для спілкування зі спільнотою розробників.

2.3.2. Flask

Flask — це легкий та гнучкий веб-фреймворк для розробки веб-застосунків на мові програмування Python. Він зосереджується на простоті та мінімалістичному підході, дозволяючи розробникам вибирати лише необхідні компоненти для своїх проектів. Flask надає базовий функціонал, але дозволяє розширити його за допомогою різних розширень.

Одна з основних функцій Flask — це маршрутизація, що дозволяє визначати URL-шаблони для різних сторінок вашого застосунку. Це дає змогу організувати структуру веб-сайту та визначити, як користувачі будуть взаємодіяти з вашим застосунком.

Ще однією важливою можливістю є шаблонізація. Flask має гнучку систему шаблонів, що дозволяє відокремити бізнес-логіку від представлення, роблячи код чистішим та зручнішим для обслуговування. Взаємодія з базою даних також є простою завдяки підтримці різних ORM-бібліотек, таких як SQLAlchemy, що дозволяє розробникам легко працювати з базами даних без необхідності написання складних SQL-запитів.

Flask також підтримує розширення, що дозволяють додавати нові функції до веб-застосунку. Це може бути все від роботи з формами до автентифікації користувачів або інтеграції з іншими сервісами. Крім того, Flask надає потужні інструменти для тестування, що дозволяє забезпечити стабільність і безпеку вашого коду.

До основних переваг Flask можна віднести його легкість вивчення. Завдяки простій структурі та мінімалістичному підходу фреймворк дозволяє швидко освоїтися і розпочати розробку. Гнучкість Flask дозволяє розробникам вибирати лише необхідні компоненти та розширення, що дає більше контролю над проектом. Крім того, Flask має широку підтримку спільноти, що забезпечує доступ до великої кількості ресурсів і допомоги з боку інших розробників. Завдяки цьому фреймворк також відомий своєю швидкістю, оскільки має низькі накладні витрати і дозволяє створювати ефективні веб-застосунки.

Flask добре поєднується з іншими технологіями та бібліотеками Python. Ви можете використовувати його в поєднанні з різними базами даних, фронтенд-фреймворками або іншими сервісами.

Flask, хоч і є зручним і гнучким фреймворком, має деякі недоліки. Один із них — це недостатність функціоналу «з батарейкою в комплекті». Хоча Flask надає базові можливості для побудови веб-застосунків, для вирішення більш складних завдань часто доводиться додатково встановлювати розширення або використовувати сторонні інструменти. Це може ускладнити процес розробки, особливо для новачків.

Ще один недолік полягає в тому, що розробник несе більшу відповідальність за вибір і налаштування необхідних компонентів. Оскільки у Flask немає вбудованих рішень для таких речей, як автентифікація чи управління базою даних, розробнику потрібно самостійно підбирати відповідні інструменти і не помилитися у виборі.

Крім того, у Flask відсутній жорсткий стандарт для організації проекту. Це може призвести до різних підходів у структурі коду між різними проектами, що ускладнює підтримку і спільну роботу над проектами, особливо в команді або при передачі проекту іншим розробникам.

Незважаючи на це, Flask є дуже популярним фреймворком, особливо серед професійних веб-розробників, які цінують його гнучкість, швидкість та простоту використання. Flask також має велику популярність серед початківців, які хочуть швидко розпочати розробку своїх веб-застосунків та навчитися основам веб-розробки.

Українська спільнота Flask також є активною та зростаючою. Багато розробників в Україні використовують Flask для своїх проектів, а також організовують зустрічі, конференції та події для обміну досвідом та підтримки.

Flask — це легкий та гнучкий фреймворк, який дозволяє швидко розробляти веб-застосунки на мові програмування Python. Він підходить для проектів будь-якого розміру, починаючи від простих односторінкових застосунків до складних веб-платформ. Flask надає вам контроль над вашим

проектом та дозволяє використовувати лише необхідний функціонал, що робить його привабливим для розробників з різним рівнем досвіду та вимогами.

2.3.3. FastAPI

FastAPI — це сучасний веб-фреймворк для швидкої розробки API на мові програмування Python. Він пропонує високу продуктивність, масштабованість та простоту використання. FastAPI побудований на базі типів та асинхронності Python, що дозволяє забезпечити швидку обробку запитів та ефективне використання ресурсів.

Одна з основних переваг FastAPI — це його швидкість, досягнута завдяки використанню веб-сервера Starlette і бібліотеки Pydantic для роботи з типами даних. Це дозволяє фреймворку обробляти запити з високою продуктивністю, роблячи його ідеальним для ресурсомістких проектів.

FastAPI також підтримує асинхронні запити, що дозволяє ефективно працювати з багатозадачними сценаріями, забезпечуючи масштабованість і оптимальне використання системних ресурсів. Крім того, фреймворк використовує Pydantic для автоматичної валідації даних, спрощуючи цей процес, а також генерує інтерактивну документацію API за допомогою Swagger, що полегшує розробку та тестування.

Окремо варто відзначити вбудовану підтримку механізмів автентифікації та авторизації, яка дозволяє розробникам легко додавати захист від несанкціонованого доступу до своїх API. Також FastAPI пропонує інтеграцію з різними базами даних та ORM-системами, такими як SQLAlchemy і Tortoise ORM, що полегшує роботу з базами даних та забезпечує гнучкість у виборі технологій.

Цей фреймворк має зручний і простий синтаксис, що робить його легким у використанні і сприяє швидкому розгортанню проектів, особливо для створення складних API з високою продуктивністю та асинхронною підтримкою.

FastAPI стає все більш популярним серед розробників Python, особливо в галузі веб-розробки та розробки API. Його швидкість, продуктивність та легкість використання роблять його привабливим вибором для проектів різних розмірів та складності. Спільнота розробників FastAPI також швидко зростає. Існує багато ресурсів, таких як документація, підручники, приклади коду та форуми підтримки, які допомагають розробникам знайти відповіді на свої питання та розв'язати проблеми.

FastAPI має широкий спектр застосувань, від створення мікросервісів та простих API до складних застосунків з багатьма ендпоінтами. Він підходить для будь-яких потреб, де необхідна швидкість, ефективність та висока продуктивність.

2.4. Вибір протоколу реалізації

У попередньому розділі було детально розглянуто різні архітектурні стилі для реалізації веб-сервісів, і після ретельного аналізу було обрано для проекту REST API як надійний та широко використовуваний протокол. У цьому розділі надається більш докладний опис та плани щодо створення бекенду для веб-застосунку з акцентом на ефективність, розширюваність, високу продуктивність та масштабованість.

Для досягнення цих цілей було вирішено використовувати сучасні інструменти та технології. Одним із ключових виборів став фреймворк FastAPI, що базується на мові Python. Завдяки його потужному функціоналу, він спрощує процес розробки та підтримки веб-застосунків. FastAPI використовуватиметься для створення та керування моделями даних, маршрутизації запитів, обробки авторизації та аутентифікації, а також для реалізації бізнес-логіки веб-застосунку.

Крім того, для зберігання та управління даними обрано базу даних PostgreSQL, яка є потужною та надійною реляційною системою. Вона забезпечує широкі можливості для роботи з даними, включаючи складні запити, індексацію та транзакційну безпеку. Використання PostgreSQL гарантуватиме швидкий і

надійний доступ до даних, що сприятиме стабільності та продуктивності веб-застосунку.

2.5. Висновки до другого розділу

У ході дослідження було виконано аналіз сучасних архітектурних рішень, які широко застосовуються для створення програмних систем. Основна увага приділялася аспектам масштабованості, продуктивності та надійності.

Для побудови серверної частини було обрано один із сучасних фреймворків, що забезпечує зручний інтерфейс для розробки та дозволяє ефективно реалізувати бізнес-логіку застосунку. Використання об'єктно-орієнтованого підходу сприяло створенню модульного та легко підтримуваного коду, а також спрощенню процесу розробки.

Для клієнтської частини застосовано технології, які підтримують адаптивний та динамічний інтерфейс користувача. Завдяки цьому інструменту було реалізовано систему, що забезпечує чітке розділення логіки та представлення, що суттєво полегшує внесення змін до інтерфейсу без необхідності змінювати серверну частину.

Отримані результати підтверджують, що застосовані методи та технології можуть бути ефективно використані для створення програмних рішень із високими вимогами до продуктивності, гнучкості та підтримуваності.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІЗУ ДОКУМЕНТІВ

3.1. Структура бази даних

Для підтримки системи обміну повідомленнями, де користувачі можуть взаємодіяти через чати була розроблена структура бази даних (рис. 3.1). Вона включає три основні таблиці: Users, Chats та Messages, які забезпечують зберігання та організацію інформації про користувачів, їхні чати та повідомлення в цих чатах.

Таблиця Users (користувачі) містить дані про всіх користувачів системи. Кожен користувач має унікальний ідентифікатор (id), ім'я або псевдонім (name) та електронну пошту (email). Користувачі можуть мати один або кілька чатів.

Chats (чати) зберігає інформацію про чати, пов'язані з користувачами. Кожен чат має унікальний ідентифікатор (id) та прив'язаний до користувача через зовнішній ключ (user_id), що посилається на таблицю Users. Також зберігається дата та час створення чату (created_at), його статус (наприклад, активний чи завершений) і тип чату (наприклад, "підтримка", "загальний" або "приватний").

Messages (повідомлення) містить всі повідомлення, відправлені в чатах. Кожне повідомлення має унікальний ідентифікатор (id) та пов'язане з конкретним чатом через зовнішній ключ (chat_id), що посилається на таблицю Chats. Крім того, таблиця зберігає тип відправника (sender_type), який може бути користувачем або асистентом, текст повідомлення (text), дату та час його відправлення (created_at), а також інформацію про те, чи було повідомлення прочитане (is_read). Додатково, через поле reply_to, можна вказати на інше повідомлення, щоб відобразити відповідь на нього.

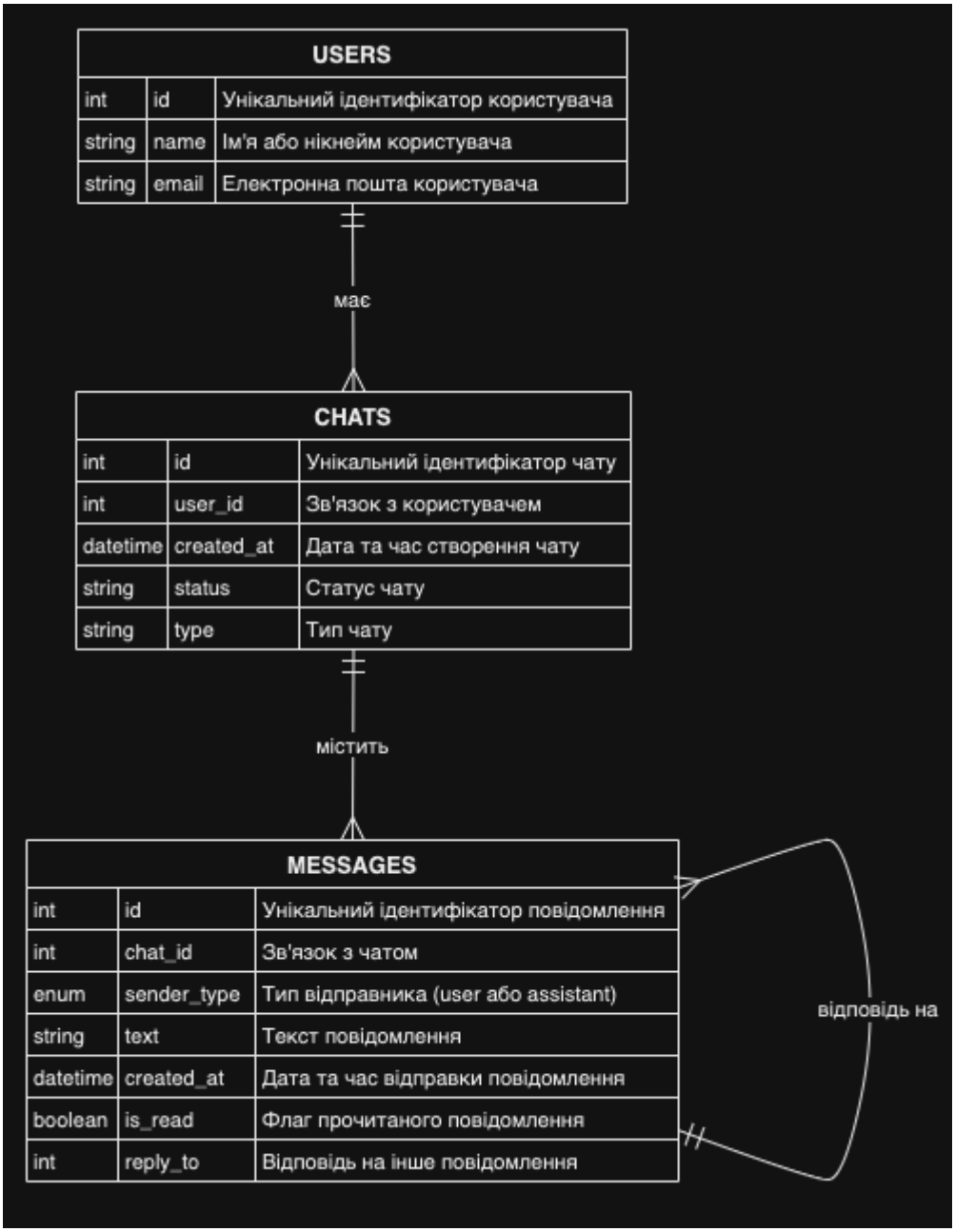


Рис. 3.1. Структура бази даних

Ця структура дозволяє ефективно керувати багатокористувацькими чатами, підтримуючи різні типи чатів, зберігання історії повідомлень та відстеження статусів, що робить її придатною для використання в сучасних веб-застосунках для обміну інформацією.

Переваги даної структури бази даних полягають у її гнучкості, масштабованості та зручності для керування чатами та повідомленнями. Завдяки відношенню «один до багатьох» між користувачами та чатами, кожен

користувач може мати декілька чатів, що спрощує організацію та аналіз даних для кожного користувача. Чітка структура із зовнішніми ключами, такими як user_id та chat_id, дозволяє ефективно індексувати дані, що підвищує продуктивність запитів до бази та забезпечує швидкий доступ до потрібної інформації. Поле created_at у повідомленнях дає змогу сортувати повідомлення за часом, забезпечуючи плавну роботу з хронологією повідомлень.

Масштабованість структури також забезпечується через поле type у таблиці чатів, яке дозволяє розрізняти різні типи чатів, що створює можливості для впровадження нових функцій, таких як групові або автоматизовані чати. Поле sender_type у таблиці повідомлень чітко відокремлює повідомлення користувачів від повідомлень асистентів, що полегшує керування різними типами взаємодії в чатах.

Додатково, поле reply_to дозволяє підтримувати функцію відповідей на конкретні повідомлення, створюючи ланцюжки повідомлень, що покращує користувацький досвід і підвищує функціональність чатів. Поле is_read допомагає реалізувати індикатори прочитання повідомлень, що є ключовою функцією в сучасних месенджерах.

Така структура не лише забезпечує високу гнучкість та масштабованість, але й дозволяє легко додавати нові функції, такі як архівування або призупинення чатів, без значних змін у вже наявних даних. Це робить базу даних ефективною та зручною для підтримки і подальшого розвитку застосунку.

Наступний етап після визначення структури бази даних — розробка сервера веб-застосунку. Сервер відповідальний за обробку запитів і надання доступу до бази даних користувачам через веб-інтерфейс. Основна мета сервера — забезпечити ефективний та безперебійний доступ до даних користувачам і забезпечити правильну обробку запитів до бази даних згідно з встановленою структурою.

Веб-застосунок може також містити інші функціональні можливості, такі як автентифікація користувачів, створення нових записів у базі даних і т. д.

3.2. Розробка бекенд-частини веб-застосунку

FastAPI — це веб-фреймворк, розроблений на мові програмування Python, і він використовує патерн проектування Model-View-Controller (MVC). Патерн MVC є одним із найпоширеніших підходів до організації коду у веб-застосунках і дозволяє розділити логіку програми на три компоненти: модель (Model), представлення (View) і контролер (Controller).

Модель (Model) відповідає за управління даними і бізнес-логікою програми. Вона визначає структуру даних, зберігає і отримує дані з бази даних, валідує їх і виконує необхідні операції з даними.

Представлення (View) відповідає за відображення даних користувачу. Воно визначає, як дані повинні бути відображені на веб-сторінці або іншому інтерфейсі користувача. Представлення отримує дані з моделі і передає їх у відповідному форматі для відображення.

Контролер (Controller) відповідає за обробку запитів користувача і взаємодію з моделлю та представленням. Він отримує запити від користувача, виконує необхідні операції з моделлю, обробляє дані і передає їх у відповідному форматі в представлення.

Застосування патерна MVC в FastAPI дозволяє розділити логіку програми на окремі компоненти, що спрощує розробку і підтримку коду. Крім того, FastAPI надає багато готових інструментів і бібліотек для роботи з базами даних, аутентифікацією користувачів, обробкою форм і багато іншого, що полегшує процес розробки веб-застосунків.

Використання патерна MVC у FastAPI допомагає створювати доброорганізовані. Спрощена схема роботи цього патерну в реалізації FastAPI зображено на рис 3.2.

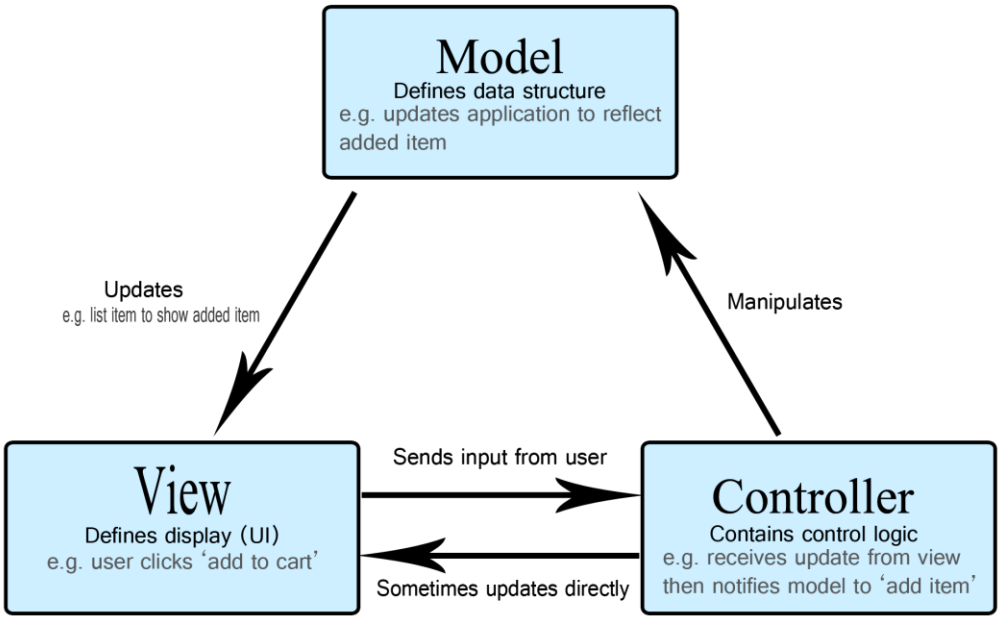


Рис. 3.2. Спрощена схема роботи патерну MVC

Частина представлення буде реалізована у фронтендчастині тому на сервері потрібно реалізувати лише контролер, моделі та маршрутизацію.

3.2.1. Реалізація моделей

Для початку потрібно за допомогою засобів пакету mongoose створити моделі, які будуть працювати з визначеною структурою бази данних PostgreSQL.

Модель Users в даній реалізації створена на основі класу BaseModel з бібліотеки Pydantic, що надає можливість автоматичної валідації та забезпечує коректне представлення даних у системі. Ключовим елементом моделі є клас UserBase, який описує основні атрибути користувача. Він включає унікальний ідентифікатор id, що є обов'язковим для кожного користувача і представлений цілим числом, а також name — ім'я користувача, яке має бути рядком. Окрім цього, модель містить поле email, яке перевіряється на відповідність формату електронної пошти за допомогою типу EmailStr з бібліотеки Pydantic, що забезпечує коректність даного параметра.

Клас UserCreate, який наслідує від UserBase, призначений для створення нового користувача. Він включає ті ж самі поля, що й UserBase, але надає можливість додавання додаткових перевірок чи функціональностей під час реєстрації користувача. Водночас, клас UserResponse, також базуючись на UserBase, включає додаткову властивість Config, що активує параметр orm_mode = True. Це дозволяє моделі коректно працювати з об'єктами, отриманими через ORM (Object Relational Mapping), що використовуються для інтеграції з базою даних. Ця опція автоматично перетворює дані з бази даних у формат, зрозумілий для Pydantic, забезпечуючи їх правильне відображення. Детальний код реалізації моделі наведений в додатку А.

Модель Chats в даній реалізації описує чат як об'єкт з кількома важливими атрибутами. Вона заснована на класі BaseModel з бібліотеки Pydantic, що дозволяє забезпечити валідацію та опис даних чату.

Основний клас ChatBase включає кілька важливих полів. id є унікальним ідентифікатором чату, який дозволяє ідентифікувати кожен чат у системі. user_id вказує на користувача, який створив або є учасником чату, і використовується для зв'язку чату з конкретним користувачем, що дозволяє забезпечити правильне відображення чатів для кожного користувача. Поле created_at фіксує дату та час створення чату, що допомагає відслідковувати, коли саме був ініційований чат. status вказує на поточний статус чату, наприклад, "активний" або "завершений", що дає змогу розрізняти активні та завершені чати. І, нарешті, type описує тип чату, наприклад, чи це підтримка, приватний чат чи груповий чат.

Клас ChatCreate є простою модифікацією ChatBase, що використовує ті ж самі поля, але орієнтований на створення нового чату, дозволяючи при цьому створювати чат без додаткових налаштувань або функцій.

Клас ChatResponse є розширенням ChatBase, яке додає поле messages. Це поле є списком повідомлень чату, що містить об'єкти типу MessageResponse (які можуть бути визначені окремо в системі). Це поле дозволяє повернути всі повідомлення чату разом із основною інформацією про чат, що зручно при запитах на отримання детальної інформації про чат. В класі також визначено

параметр Config з властивістю orm_mode = True, що дозволяє ефективно інтегрувати модель з ORM-системами для роботи з базами даних, автоматично перетворюючи дані з бази в формат, зручний для роботи в Python. Детальний код реалізації моделі наведений в додатку Б.

Модель Messages описує структуру повідомлень у чатах. Вона базується на класі BaseModel з бібліотеки Pydantic, що надає можливість валідації та опису даних. Основні поля цієї моделі дозволяють ефективно зберігати та обробляти інформацію про кожне повідомлення в чаті.

Id є унікальним ідентифікатором повідомлення, що дозволяє точно ідентифікувати кожне повідомлення в системі. Поле chat_id вказує на чат, до якого належить повідомлення, тим самим встановлюючи зв'язок між повідомленням та відповідним чатом у таблиці Chats. sender_type є перерахуванням (enum), яке вказує на тип відправника повідомлення — чи це користувач (user), чи асистент (assistant). Це дозволяє відрізняти повідомлення від користувача від повідомлень, надісланих системою або автоматичним асистентом.

Text містить текст повідомлення, що є основним вмістом повідомлення, яке надсилається у чат. created_at фіксує дату та час, коли повідомлення було надіслано, що важливо для відслідковування хронології спілкування. Поле is_read є булевим значенням, яке вказує, чи було повідомлення прочитано отримувачем, що допомагає реалізувати функціонал індикаторів прочитаних повідомлень.

Додатково, поле reply_to є опціональним і містить id іншого повідомлення, на яке було дано відповідь. Це дозволяє організувати ланцюжок відповідей, що робить спілкування більш структурованим і полегшує навігацію по історії повідомлень.

Клас MessageCreate є простою версією моделі MessageBase, яка використовується для створення нових повідомлень, не додаючи додаткових функцій. Клас MessageResponse є розширенням MessageBase і використовується для відповіді на запити, дозволяючи отримувати дані повідомлень в зручному

форматі. В класі визначено параметр Config з властивістю orm_mode = True, що забезпечує інтеграцію з ORM та автоматичну конвертацію даних із бази даних у формат, зручний для роботи в Python. Детальний код реалізації моделі наведений в додатку Б.

3.2.2. Створення маршрутів

Маршрутизація в FastAPI відповідає за визначення того, які URL-адреси повинні бути пов'язані з якими уявленнями (views) і функціями. FastAPI використовує потужну систему маршрутизації, яка дає змогу легко визначити маршрути для веб-застосунку.

Основний компонент маршрутизації в FastAPI — це файл urls.py, який знаходиться в кореневій директорії вашого проекту FastAPI. У цьому файлі визначаються шаблони URL-адрес і їхні пов'язані подання.

Нижче наведено приклад того, як маршрути розбиті на більш дрібні частини в рамках основної структури проєкту. Для цього використано внутрішні додатки FastAPI. Кожен з цих додатків також має свої власні маршрути, якщо це необхідно. Кожен маршрут містить шлях і вказує на клас або функцію, яка обробляє запити, що надходять по цьому маршруту.

Така організація маршрутів дозволяє розділити функціональність проєкту на менші компоненти, що полегшує розробку та утримання коду. Кожен внутрішній застосунок може мати свою власну логіку та функціональність, яка обробляє певний тип запитів. Це дозволяє розподілити завдання між командами розробників та підтримувати структурованість проєкту.

На рисунку 3.2 представлено код спрощеної схеми патерну MVC. Цей код створює API-додаток за допомогою фреймворку FastAPI, спрямований на обробку повідомлень у чаті. Початково створюється об'єкт FastAPI, в якому задається параметр lifespan, що визначає налаштування життєвого циклу додатку. Після цього додається маршрутизатор chat_router, який відповідає за маршрутизацію запитів, пов'язаних із чатами, що дозволяє організувати обробку всіх запитів, що стосуються чатів, в одному місці.

Рис. 3.3. Спрощена схема роботи патерну MVC

Створюється об'єкт APIRouter з префіксом '/api/chat', який обробляє запити, пов'язані із функціоналом чату. Префікс означає, що всі маршрути, які будуть додані в цей маршрутизатор, матимуть URL-адресу, що починається з /api/chat. Це дозволяє зберегти структуру і зрозумілість API, де всі запити для роботи з чатами будуть згруповані під одним префіксом.

Одним із таких маршрутів є обробник запиту для відправлення повідомлення в чат, який реалізований функцією chat_send_message_router. Ця функція є асинхронною, що дозволяє ефективно обробляти запити та працювати з даними, не блокуючи сервер під час виконання. Функція приймає кілька аргументів: messages — це повідомлення в чаті, яке передається у форматі, визначеному схемою ChatMessagesSchema; request — об'єкт запиту, який містить

інформацію про метод, URL і інші деталі запиту; response — об'єкт відповіді, що дозволяє змінювати параметри відповіді перед її відправкою користувачу; та headers, що містить додаткові заголовки, які отримуються за допомогою залежності Depends(), яка визначена як LLMRequestHeaders. Цей підхід дозволяє гнучко працювати з різними аспектами запитів та відповідей у додатку.

Використання такої структури маршрутів дозволяє мені логічно групувати функціональність мого проєкту та забезпечує зручний спосіб навігації. Кожен маршрут відповідає за певну дію, що полегшує розробку, підтримку та розширення мого проєкту в майбутньому.

3.2.2 Отримання даних

Класи ListView і RetrieveView в FastAPI використовуються для ефективного відображення списків об'єктів та деталей окремих об'єктів. Клас ListView призначений для відображення списків об'єктів, наприклад, записів з бази даних. Він забезпечує стандартну логіку для отримання списку об'єктів з бази даних та їх відображення на веб-сторінці. Зазвичай цей клас використовується для створення списків або таблиць, що містять кілька елементів, таких як записи користувачів, продукти або інші моделі. Окрім цього, ListView дозволяє реалізувати пагінацію, що дозволяє розбити великі списки на кілька сторінок, покращуючи зручність користування.

З іншого боку, клас RetrieveView використовується для відображення деталей одного конкретного об'єкта. Це може бути сторінка, на якій показується повна інформація про окрему статтю, користувача, продукт чи інший об'єкт. Клас відповідає за витягування даних з бази даних для одного об'єкта та передавання цих даних на веб-сторінку для перегляду. Крім цього, RetrieveView може реагувати на дії користувача, наприклад, редагування або видалення об'єкта.

Обидва класи є дуже гнучкими і можуть бути налаштовані під конкретні потреби проекту. Наприклад, можна змінювати шаблони, URL-адреси, атрибути моделей або додавати додаткову логіку до методів, щоб налаштувати

відображення даних відповідно до вимог. Загалом, класи ListView і RetrieveView в FastAPI значно спрощують процес створення веб-сторінок для відображення списків і деталей об'єктів, надаючи готову логіку для роботи з базою даних і веб-інтерфейсом.

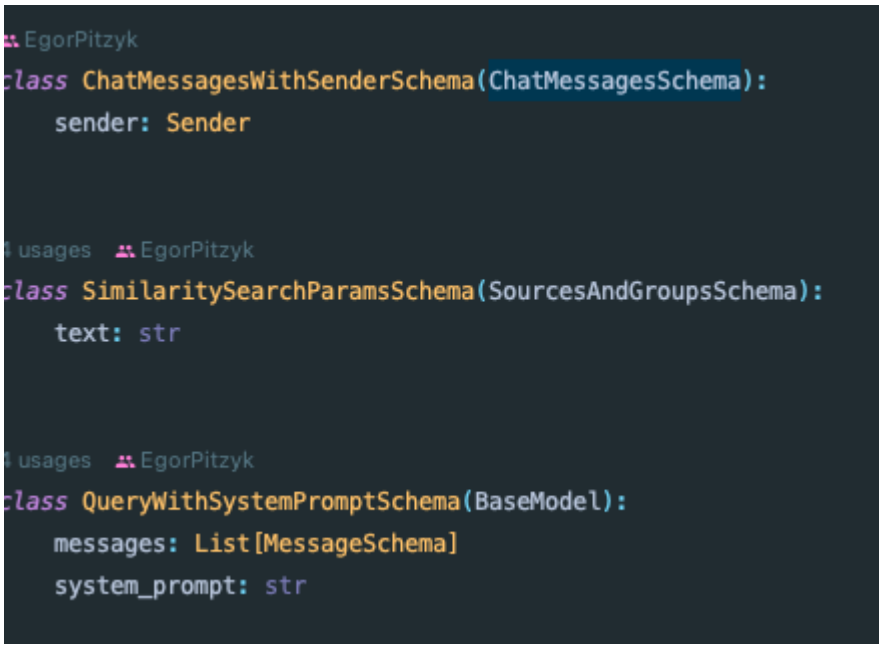


Рис. 3.4. Клас для відображення списку вакансій

3.2.3 Створення даних

У FastAPI для реалізації CRUD операцій (створення, читання, оновлення, видалення) часто використовуються Pydantic та SQLAlchemy. FastAPI підтримує асинхронні запити та дозволяє легко інтегрувати Pydantic-схеми для валідації даних і SQLAlchemy для роботи з базою даних. Давайте розглянемо, як організувати CRUD операції з цими інструментами.

SQLAlchemy в FastAPI забезпечує механізм ORM (Object-Relational Mapping), що дозволяє працювати з базою даних за допомогою Python-класів, які відповідають таблицям у базі. Спочатку налаштовується підключення до бази даних через SQLAlchemy, що зазвичай робиться у конфігураційному файлі, де вказується тип підключення та інші налаштування. Після цього створюються моделі бази даних у вигляді класів Python. Кожен клас представляє окрему таблицю в базі, і поля класу відповідають стовпцям цієї таблиці. Клас моделі

успадковується від спеціального базового класу, зазвичай званого Base, що дозволяє SQLAlchemy здійснювати зв'язок між класами Python і таблицями бази даних, а також автоматично генерувати SQL-запити для взаємодії з даними. Цей підхід дозволяє ефективно працювати з базою даних без необхідності вручну писати SQL-запити.

Pydantic у FastAPI використовується для перевірки даних, які передаються між клієнтом і сервером. Для цього створюються схеми, які визначають структуру вхідних та вихідних даних, гарантуючи, що вони відповідають певним вимогам. Зазвичай схеми для створення або оновлення об'єктів не містять поля ID, оскільки це значення генерується автоматично при збереженні в базі даних.

Натомість для читання даних використовуються схеми, які можуть включати додаткові поля, такі як ID, оскільки вони використовуються для передачі інформації користувачу або клієнту. Завдяки таким схемам Pydantic забезпечує валідацію даних (перевірку на правильність), серіалізацію (перетворення даних у формат, зручний для передачі) та десеріалізацію (перетворення даних, що приходять від клієнта, у структуру, з якою можна працювати в програмі). Це дозволяє спростити обробку даних і підвищити надійність роботи сервісу.

Кожна з операцій CRUD (створення, читання, оновлення, видалення) виконується за допомогою SQLAlchemy, який взаємодіє з базою даних через відповідні запити та методи. Для створення нового запису спочатку створюється Pydantic-схема, яка описує структуру даних, що надходять від користувача. Після того, як дані пройшли валідацію через Pydantic, FastAPI використовує SQLAlchemy для збереження цих даних у базі. SQLAlchemy створює новий об’єкт на основі отриманих даних, додає його в сесію і після цього фіксує транзакцію, тобто записує зміни в базу даних.

Для операції читання створюються маршрути, які виконують запити до бази даних за допомогою SQLAlchemy. Коли необхідно отримати список записів, SQLAlchemy виконує запит до відповідної таблиці і повертає всі записи або їх частину, використовуючи, наприклад, пагінацію для зручності перегляду великої

кількості даних. Якщо потрібно отримати конкретний запис, SQLAlchemy виконує запит за унікальним ідентифікатором, повертаючи тільки один запис, який відповідає заданим критеріям.

Для оновлення даних спочатку необхідно знайти запис, який потрібно змінити. Для цього SQLAlchemy виконує запит до бази даних, шукаючи запис за унікальним ідентифікатором або іншими критеріями. Після того, як потрібний запис знайдено, його поля оновлюються новими значеннями, і ці зміни фіксуються в базі даних. Таким чином, запис оновлюється відповідно до наданих даних.

Щодо видалення записів, процес також починається з пошуку потрібного об'єкта в базі даних. Як і при оновленні, SQLAlchemy виконує запит для знаходження запису, а після цього видаляє його з таблиці. Після видалення змінений стан бази даних фіксується.

Важливим аспектом роботи з базою даних у FastAPI є використання залежностей для управління сесіями SQLAlchemy. Кожен запит створює нову сесію для взаємодії з базою даних, і після виконання операції ця сесія закривається. Залежності в FastAPI полегшують доступ до цієї сесії в функціях обробників, що дозволяє організувати безпечну та ефективну роботу з базою даних, гарантуючи, що кожна операція з даними буде виконана в межах свого запиту.

FastAPI інтегрується з Pydantic для валідації і серіалізації даних, що дозволяє ефективно працювати з ними, як при створенні, так і при читанні записів. Використовуючи Pydantic-схеми, можна чітко визначити структуру даних, які приходять в запитах або відправляються у відповідях, що забезпечує високий рівень контролю та безпеки під час обробки даних.

У FastAPI Pydantic-схеми використовуються для опису структури даних, що надходять і виходять через API. Кожен об'єкт, наприклад, користувачі або продукти, має свою Pydantic-схему, яка визначає, як повинні виглядати дані під час обробки запитів і відповідей. Для створення або оновлення об’єкта в базі даних застосовується базова схема, яка визначає, як виглядатимуть вхідні дані.

Вона зазвичай містить усі необхідні поля для створення нового запису чи оновлення існуючого. Окрім того, існують схеми для відповіді (схеми-відповіді), які визначають, як буде виглядати об'єкт при отриманні даних. Ці схеми включають лише ті поля, які потрібні для відображення користувачу, щоб не передавати зайву інформацію.

Для реалізації логіки роботи зі списками об'єктів у FastAPI можна використовувати маршрути, які повертають списки об'єктів. В цьому випадку FastAPI автоматично обгортає відповідь у список, і можна визначити тип даних через Pydantic-схему, яка буде використовуватись для кожного елемента в списку. Крім того, для роботи зі списками можна додати функціонал фільтрації та пагінації, який реалізується вручну. Для цього до функцій обробників додаються параметри, які дозволяють вказувати межі вибірки, наприклад, параметри skip та limit.

Що стосується отримання даних для одиничних об'єктів, FastAPI надає можливість створювати маршрути, які приймають ідентифікатор об'єкта в URL і повертають відповідь у вигляді Pydantic-схеми. Ця схема визначає, які поля об'єкта будуть відображатися в відповіді. Завдяки вбудованій інтеграції з Pydantic, FastAPI автоматично перевіряє, чи відповідають дані типам, що визначені в схемі, перед тим, як відправити їх клієнту.

Крім того, FastAPI дозволяє кастомізувати логіку обробки запитів за допомогою залежностей. Це дає змогу, наприклад, додавати перевірку прав доступу або додаткові параметри до запиту, що підвищує гнучкість і функціональність API. Такий підхід забезпечує ефективну роботу з базою даних і серіалізацію даних, дозволяючи створювати API без необхідності в надмірних класах представлення чи складних структур.

3.2.4. Деплой серверу

Для деплою веб-застосунку на базі FastAPI на платформі AWS було використано хмарне середовище цієї платформи, яке надає можливість безкоштовного розгортання додатків з невеликим трафіком. Спочатку на

локальній машині були встановлені Git та AWS CLI (Command Line Interface), що дозволило здійснювати управління додатками на AWS через командний рядок.

Після налаштування AWS CLI, для створення нового проекту було використано команду heroku create, яка автоматично ініціювала проект на серверах AWS та налаштувала зв'язок між локальним репозиторієм і платформою AWS. Після цього код проекту було додано до репозиторію AWS за допомогою команди git push aws main. Проте перед цим, щоб мати змогу працювати з репозиторієм, необхідно було ініціалізувати Git-репозиторій у проекті через команду git init.

Окрім того, для зручності в роботі з репозиторієм був створений файл .gitignore в кореневій директорії проекту. У ньому були вказані правила для виключення файлів або папок, які не потрібно включати до репозиторію перед комітом. Це дозволило уникнути збереження зайвих чи тимчасових файлів, таких як налаштування середовища чи залежності, що можуть змінюватися на різних етапах розробки.

Додатково було налаштовано кілька параметрів, що стосуються конфігурації самого FastAPI-застосунку на Heroku. Це включало налаштування параметрів автентифікації, вигляду сторінок, а також інших елементів, таких як login_url, redirect_field_name, template_name, form_class та context_object_name. Ці параметри допомогли налаштувати автентифікацію користувачів, управління формами та змінними контексту, відповідно до специфічних вимог застосунку, забезпечуючи гнучкість у процесі налаштування інтерфейсу і логіки взаємодії з користувачем.

3.3. Розробка фронтенд частини веб-застосунку

Команда FastAPI-admin startproject використовується для створення нового проекту FastAPI. Вона запускає генератор початкової структури проекту та створює необхідні файли та папки.

Основним етапом роботи команди FastAPI-admin startproject є створення кореневої папки проекту і налаштування початкової структури файлів та папок. Після виконання команди створюється нова папка з назвою проекту, яка містить кілька важливих файлів:

Виконуваний файл «main.py» використовується для управління проектом FastAPI. Він дозволяє виконувати команди для розгортання сервера, запуску тестів, міграцій бази даних та багато іншого.

Файл «__init__.py» позначає папку проекту як пакет Python. Він може залишатися порожнім або містити додатковий код, який виконується при імпортуванні проекту.

«setITngs.py» містить налаштування проекту FastAPI. В ньому визначаються база даних, статичні файли, шаблони, мови, аутентифікація та інші параметри, необхідні для правильної роботи проекту.

urls.py визначає шляхи (URL) проекту FastAPI. В ньому вказуються маршрутизатори, які вказують, які функції або класи відповідають на різні URL-шляхи.

Файл «wsgi.py» використовується для розгортання проекту FastAPI на WSGI-серверах (Web Server Gateway Interface). Він забезпечує взаємодію між веб-сервером та додатком FastAPI.

Коли команда FastAPI-admin startproject завершує свою роботу, створюється початкова структура проекту FastAPI. Ви можете редагувати файли setITngs.py та urls.py, щоб налаштувати ваш проект, додати нові URL-шляхи, підключити додатки, встановити базу даних та зробити інші необхідні зміни.

Після створення проекту ви можете виконувати команди python manage.py, щоб запустити розробку сервера, створити та застосувати міграції, створити адміністратора та виконати багато інших дій, пов'язаних з управлінням вашим проектом FastAPI.

3.3.1 Структура інтерфейсу веб-застосунку

Для створення свого прикладу веб інтерфейсу я вирішив використати ще один професійний інструмент Figma.

Figma — це популярний інструмент для дизайну та прототипування веб-інтерфейсів. Він надає можливість створювати професійні дизайни, співпрацювати з командою та створювати інтерактивні прототипи веб-застосунків.

Після входу в Figma було створено новий проект. Вибрано опцію "New File" або "Create New" та обрано тип дизайну, наприклад, "Web Design". Розміри холста були встановлені відповідно до типу веб-застосунку, над яким працювалося, зазвичай вибирали стандартні розміри, такі як 1280 пікселів шириною для настільних комп'ютерів або 375 пікселів для мобільних пристроїв.

Для розташування елементів на холсті використовували інструменти Figma, створюючи елементи веб-інтерфейсу, такі як кнопки, поля введення, меню та інші. Мальовані форми, вставлялися тексти, використовувалися графічні елементи та інші ресурси.

Було використано макети (components) для елементів, які повторюються на веб-сайті. Наприклад, створено макет для заголовків або навігаційного меню, який потім використовувався по всьому проекту. Якщо макет змінювався, він автоматично оновлювався в усіх місцях, де був застосований.

Прототипування стало можливим завдяки функціоналу Figma, що дозволяє створювати інтерактивні прототипи веб-застосунку. Додано посилання між сторінками, налаштовано анімацію та взаємодію з формами і іншими елементами, що допомогло відтворити досвід взаємодії з веб-застосунком та отримати зворотний зв'язок від команди або користувачів.

Співпраця та коментарі стали важливою частиною процесу, оскільки Figma дозволила запрошувати інших користувачів, ділитися посиланнями на дизайн, залишати коментарі та відповідати на них. Це покращило комунікацію всередині команди та ефективність роботи над проектом.

Коли дизайн був завершений, його експортовано у різних форматах, таких як PNG, JPEG, SVG, або згенеровано CSS-код для елементів. Це полегшило передачу дизайну розробникам або іншим зацікавленим сторонам. Figma надала безліч функцій, які значно спростили роботу над дизайном веб-застосунків. На рисунку 3.4 представлено завершений дизайн проєкту в Figma.

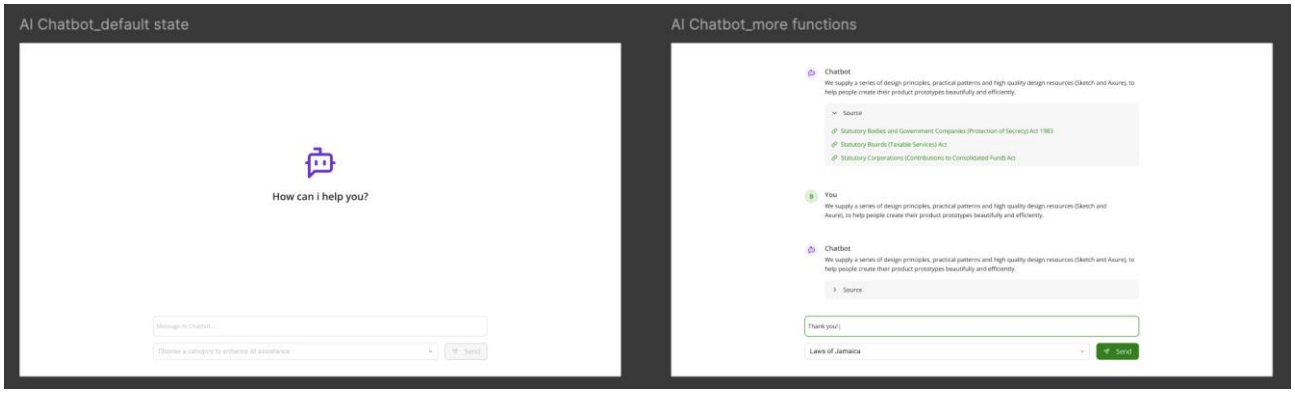


Рис. 3.5. Дизайн проєкту в Figma

3.3.2 Сторонні модулі, використанні у веб-застосунку, їх опис та призначення

Для простішої та швидшої розробки веб-застосунку необхідно встановити деякі додаткові модулі, які допоможуть створити необхідний функціонал. Слід зазначити, що всі модулі пристосовані до роботи в браузерах на базі Chromium.

Під час розробки веб-застосунку було розроблено деякі додаткові компоненти та допоміжні файли, щоб розбити застосунок по функціоналу. Нижче представлено таблицю, з описом цих компонентів.

Бібліотека requests — це популярна бібліотека Python для виконання HTTP-запитів. Вона надає простий та елегантний інтерфейс для взаємодії з веб-серверами, отримання даних, відправки параметрів запиту, установки заголовків тощо.

За допомогою requests ви можете здійснювати GET-, POST-, PUT-, DELETE- та інші види запитів до серверів, обробляти отримані відповіді та працювати з кукісами, авторизацією та сеансами.

Бібліотека io — це модуль Python, який надає базові класи та функції для роботи з потоками вводу-виводу. Він дозволяє зчитувати та записувати дані з різних джерел та у різні цільові об'єкти.

«io» використовується для роботи зі стрічками, байтами, файлами, мережевими сокетами та іншими об'єктами. Ви можете створювати об'єкти io для зчитування, запису та обробки даних, а також для перенаправлення виводу та вводу.

Бібліотека json — це модуль Python для роботи з форматом обміну даними JSON (JavaScript Object NotaITon). Він надає функції для кодування (серіалізації) та декодування (десеріалізації) об'єктів Python в формат JSON та навпаки.

json використовується для обміну даними між різними програмами або системами, які підтримують формат JSON. Ви можете перетворювати об'єкти Python у рядки JSON для відправки або збереження даних, а також декодувати рядки JSON у відповідні об'єкти Python для подальшого використання.

Бібліотека jinja2 — це популярний движок шаблонів для Python, який надає потужні можливості для генерації тексту на основі шаблонів. Він використовує синтаксис, схожий на HTML або XML, з можливістю використання змінних, циклів, умовних виразів тощо.

За допомогою jinja2 ви можете створювати шаблони для генерації HTML-сторінок, текстових файлів, електронних листів тощо. Ви можете вставляти дані з об'єктів Python, створювати умовні вирази та цикли для динамічного форматування виводу.

Ці бібліотеки широко використовуються у спільноті FastAPI та допомагають розширити можливості фреймворка, забезпечуючи зручний та потужний інструментарій для роботи з мережевими запитами, даними, шаблонами та іншими аспектами розробки веб-застосунків.

3.3.3 Опис структури веб-застосунку

Розроблений веб-застосунок складається з файлів, що розташовані в певній ієрархії та працюють між собою в користувальницькому режимі.

Коренева папка проекту є головною директорією, в якій зберігаються всі інші компоненти і файли вашого проекту. Вона є основою структури проекту, і при створенні нового проекту на основі FastAPI за допомогою FastAPI-admin, ця папка отримує ім'я, яке вказується під час налаштування проекту. Всі інші файли та каталоги проекту будуть розташовані всередині цієї кореневої папки, що дозволяє зберігати структуру організованою та легкою для навігації.

Файл main.py відіграє важливу роль в управлінні проектом. Це центральний файл, в якому містяться основні налаштування для запуску сервера FastAPI, виконання міграцій бази даних, а також інші команди для адміністрування проектом. Наприклад, тут може бути визначено, як налаштовувати підключення до бази даних, запускати сервер для тестування, створювати адміністративних користувачів або інші задачі, які необхідно виконувати для підтримки проекту в робочому стані.

Файл settings.py є важливим компонентом для налаштування самого проекту. У ньому зберігаються всі глобальні налаштування, що визначають, як буде працювати FastAPI-проект. Це може включати налаштування для підключення до бази даних, шляхи до шаблонів HTML, статичних файлів (CSS, JS, зображення тощо), а також параметри, які стосуються мови, часового поясу та інших глобальних властивостей. Всі ці налаштування можна налаштувати відповідно до конкретних потреб вашого проекту.

Папка urls.py відповідає за маршрутизацію в проекті. Вона містить всі необхідні файли, які визначають, як URL-адреси веб-застосунку співвідносяться з відповідними в'юхами (функціями або класами, що обробляють запити). Тут ви можете налаштувати маршрути, які дозволяють користувачам отримати доступ до різних сторінок або ресурсів вашого застосунку. Це важливий компонент для забезпечення правильної навігації та обробки запитів користувачів.

Папка static є місцем для зберігання всіх статичних файлів, які використовуються у вашому веб-застосунку. Це можуть бути CSS-файли для стилізації веб-сторінок, JavaScript-файли для додавання функціональності на клієнтському боці, зображення, шрифти та інші медіа-ресурси. Всі ці файли

зазвичай зберігаються в окремих підпапках всередині папки static, щоб організувати їх зручним для розробників способом.

Загалом, структура проекту FastAPI надає гнучкість для налаштування та організації різних компонентів веб-застосунку. Всі ці папки і файли мають свої чіткі ролі, і разом вони забезпечують ефективну роботу проекту, дозволяючи розробникам організовувати свій код, налаштування та ресурси з максимальною зручністю.

Папка templates є важливою частиною структури проекту, де зберігаються всі шаблони HTML для вашого веб-застосунку. Вона дозволяє створювати динамічні сторінки, що відображають дані користувача або інформацію з бази даних. Шаблони можуть бути організовані у підкатегорії в залежності від типу сторінок або компонентів, що використовуються на сайті. Наприклад, окремі папки для шаблонів профілів користувачів, адміністративного інтерфейсу, або шаблони для різних типів контенту. Ця організація допомагає зберігати структуру проекту чіткою та легкою для навігації, особливо коли сайт чи застосунок стає більшим і складнішим.

Папка apps містить окремі додатки або програмні модулі, які складають основний функціонал вашого проекту. Це модульний підхід до організації коду, де кожен додаток може мати свою власну структуру, включаючи моделі, в'юхи, шаблони та інші необхідні файли. Такий підхід дозволяє не тільки спростити розробку та підтримку проекту, але й підвищити його масштабованість. Якщо проект потребує нових функціональних можливостей, можна просто додати новий додаток, не порушуючи основну структуру інших частин проекту.

Файл models.py є важливим для управління базою даних вашого проекту. У ньому визначаються всі моделі, які будуть відображати структуру таблиць у базі даних, зокрема поля, зв'язки між таблицями та інші бізнес-логіки. Це місце, де ви описуєте, як об'єкти вашого застосунку повинні взаємодіяти з базою даних. Тут можуть бути визначені різні типи даних, обмеження на значення полів (наприклад, максимальна довжина тексту або значення для числових полів) та

зв'язки між різними таблицями, як-от один до одного, один до багатьох або багато до багатьох.

Файл views.py відповідає за логіку обробки запитів і відображення даних користувачу. В'юхи є важливою частиною проекту, оскільки вони визначають, як інтерфейс взаємодіє з користувачем і як правильно передаються або відображаються дані. Це можуть бути функції або класи, які обробляють запити до різних сторінок або ресурсів вашого веб-застосунку. Вони можуть включати логіку рендерингу шаблонів, обробки форм, а також будь-яку іншу функціональність, що необхідна для роботи веб-застосунку.

Файл admin.py використовується для налаштування адміністративного інтерфейсу FastAPI. Цей інтерфейс дозволяє адміністраторам або іншим користувачам з правами доступу редагувати дані без необхідності взаємодіяти безпосередньо з базою даних чи кодом проекту. У цьому файлі можна зареєструвати моделі, щоб вони стали доступними через адміністративний інтерфейс, дозволяючи, наприклад, додавати нові записи або змінювати існуючі без написання SQL-запитів або зміни коду.

Файл forms.py є місцем, де визначаються форми, що використовуються для збору даних від користувачів, таких як реєстраційні форми, форми для редагування профілю чи інші подібні функціональності. Використовуються Pydantic-схеми для перевірки та валідації введених користувачем даних, що дозволяє гарантувати їх коректність ще до того, як дані будуть збережені в базі даних або використані для подальших операцій. Це дозволяє запобігти введенню некоректних даних і знижує ймовірність помилок під час обробки користувацьких запитів.

Всі ці файли разом утворюють структуру проекту FastAPI, створеного за допомогою FastAPI-admin. Така організація коду дозволяє працювати над великими проектами та підтримувати їх у довгостроковій перспективі, підтримуючи зрозумілу і логічну структуру. Водночас, ця структура дозволяє легко масштабувати проект, додаючи нові функціональності, змінивши лише частини проекту, які потребують оновлень.

3.4. Процес та результати системи

Для забезпечення коректної роботи програми заздалегідь були підготовлені та завантажені тестові файли. У якості тестових даних було обрано документи із законодавчої бази Ямайки. Це дозволило перевірити працездатність програми у контексті роботи з текстами великого обсягу, що містять юридичні терміни та складні конструкції.

Після завантаження файлів запускається процес векторизації, який є ключовим етапом обробки даних. Векторизація здійснюється за допомогою спеціальної embeding-функції. Ця функція розбиває текст документів на логічні блоки (сегменти), що забезпечує зручну подальшу обробку. Кожен із цих блоків перетворюється у вектор, тобто числове представлення, яке зберігає смислове навантаження тексту. Отримані вектори зберігаються у векторній базі даних. Такий підхід дозволяє ефективно працювати з великими обсягами даних, забезпечуючи швидкий пошук та обробку інформації.

Коли процес векторизації завершено, програма стає готовою до виконання основних функцій. При зверненні користувача програма виконує кілька послідовних етапів, щоб забезпечити точність та релевантність відповіді.

На першому етапі запит користувача також проходить векторизацію. Використовується той самий принцип, що й для обробки текстів документів: запит розбивається на сегменти, після чого кожен із них перетворюється у вектор. Цей підхід дозволяє максимально точно порівнювати запит із вже векторизованими даними у базі.

На другому етапі векторна база даних здійснює пошук релевантної інформації. Вектори запиту порівнюються з векторами текстових блоків документів. За допомогою алгоритмів пошуку найближчих сусідів (наприклад, cosine similarity) визначаються вектори, які найкраще відповідають запиту користувача. Після цього витягуються відповідні текстові блоки з документів.

На третьому етапі знайдений текст передається великій мовній моделі (LLM) у вигляді контексту. Цей контекст забезпечує модель актуальною та релевантною інформацією, яка необхідна для формування точної відповіді.

Спочатку LLM аналізує переданий контекст, а потім обробляє запит користувача, забезпечуючи генерацію відповіді, що враховує специфіку питання.

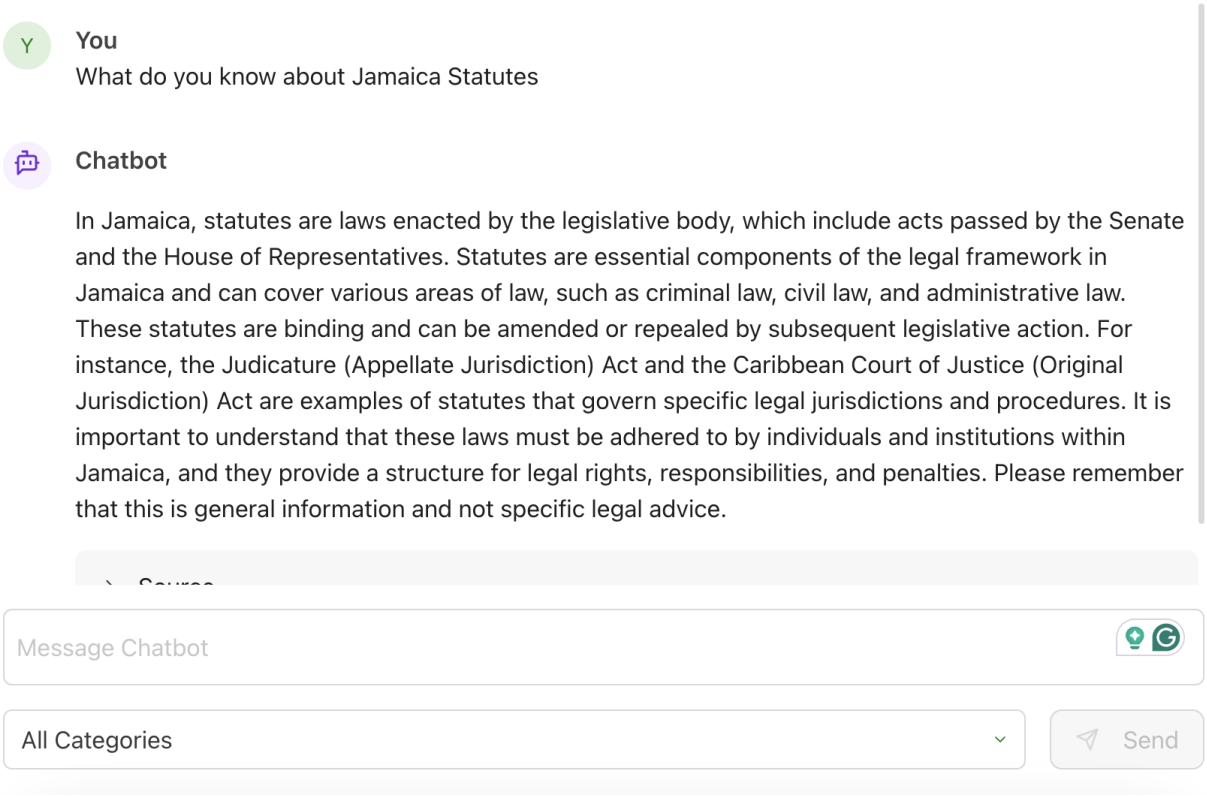


Рис. 3.6. Результат відповіді LLM

Заключний етап передбачає створення посилань на джерела інформації. Посилання (sources) формуються на основі текстів із документів, вектори яких виявилися найближчими до запиту. Це дозволяє користувачеві отримати не тільки стиснуту відповідь від моделі, але й доступ до оригінальних документів, що можуть містити додаткову інформацію для більш глибокого вивчення питання.

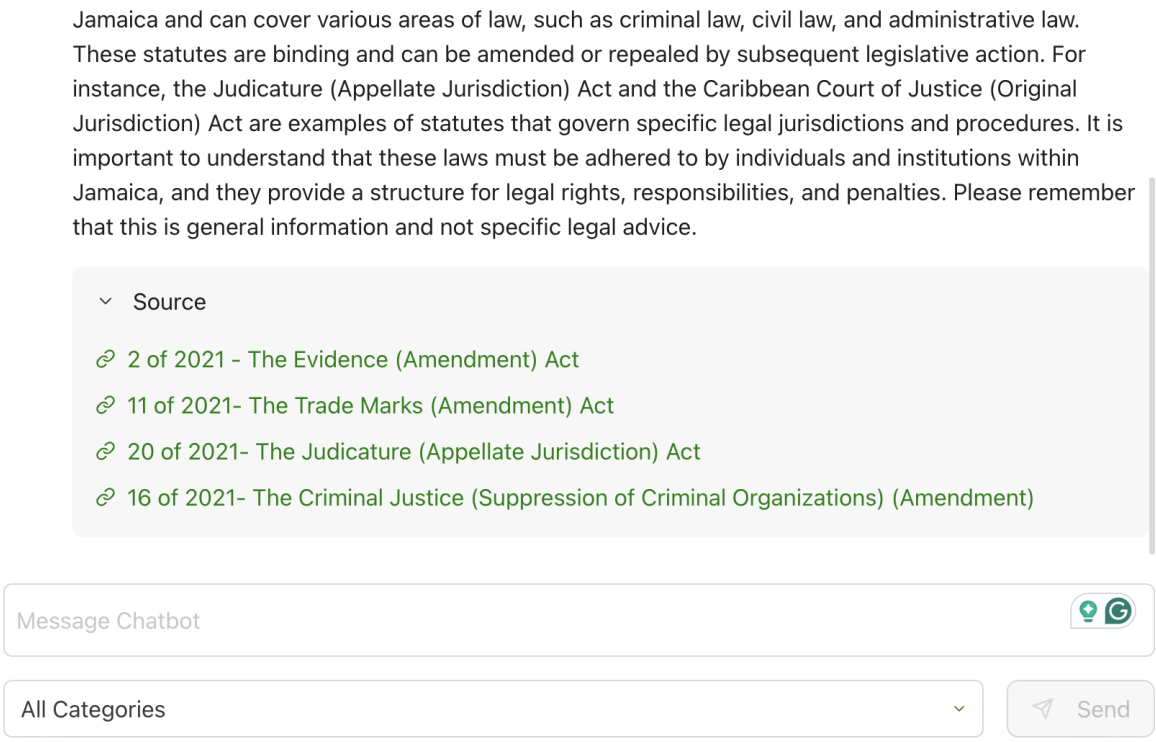


Рис. 3.7. Результат згенерованих посилань на документи

Такий підхід забезпечує високий рівень точності, прозорості та надійності роботи програми. Інтеграція векторної бази даних, процесу векторизації та мовної моделі дозволяє обробляти запити різної складності та надавати користувачеві максимально корисну інформацію у зручному форматі.

3.5. Висновки до третього розділу

У проекті, що стосувався розробки системи для аналізу документів і автоматизації пошуку даних за допомогою RAG, було застосовано різноманітні потужні інструменти та технології, які дозволили не лише ефективно обробляти великі обсяги даних, але й забезпечити високу надійність та продуктивність системи. Процес розробки потребував глибокого розуміння найкращих практик у сфері веб-розробки, а також інтеграції з численними джерелами даних і технологіями для створення зручного, ефективного та безпечного користувацького інтерфейсу.

Основним завданням було забезпечити ефективний збір та обробку даних з різних джерел, зокрема вакансій. Для цього використовувалися бібліотеки, такі як requests, що дозволяють легко виконувати HTTP-запити до веб-сайтів, отримувати необхідні відповіді та обробляти їх у форматах, зручних для подальшого аналізу. Завдяки цьому процес збору даних став не лише автоматизованим, але й оптимізованим для роботи з великими обсягами інформації.

Для того, щоб ця інформація ставала доступною та зручною для подальшої роботи, застосовувався модуль json, який дозволяє зручно працювати з отриманими даними у форматах, таких як JSON, трансформуючи їх у структури, що ідеально підходять для подальших маніпуляцій і аналізу. Це дозволяло значно зменшити кількість помилок та значно прискорити процес обробки даних, оскільки JSON є одним з найбільш популярних і гнучких форматів для передачі даних між серверами і клієнтами.

Зберігання та маніпулювання файлами різних форматів стало важливою частиною цього процесу. Для цього використовувався модуль io, що дозволяє працювати з файлами на різних етапах: від отримання файлів із зовнішніх джерел до збереження результатів аналізу в різних форматах, таких як текстові документи, зображення чи архіви. Цей модуль дозволяє організовувати роботу з великими обсягами інформації, гарантуючи, що вона буде доступною для подальших етапів аналізу та обробки.

Важливим аспектом проекту стало створення динамічних веб-сторінок, які ефективно представляють результати роботи системи. Для цього використовувався шаблонізатор Jinja2, що забезпечує можливість генерувати HTML-код з урахуванням індивідуальних потреб користувачів. Це дозволило персоналізувати контент, зокрема рекомендації, результати пошуку або іншу інформацію, адаптуючи її до запитів кожного окремого користувача. Завдяки використанню змінних, циклів та умов у шаблонах стало можливим створювати інтерактивний та динамічний контент, який автоматично оновлюється у відповідь на зміни в системі або запити клієнтів.

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи було досліджено процес аналізу документів для автоматизації пошуку даних та спроектовано програмну систему, що реалізує цей процес із використанням технології Retrieval-Augmented Generation (RAG). Поставлені завдання були успішно виконані, зокрема:

− досліджено сучасні методи, засоби та технології аналізу документів для автоматизації пошуку даних;

− проведено порівняльний аналіз існуючих підходів та інструментів, що використовуються для автоматизації обробки текстової інформації;

− спроектовано архітектуру системи та обрано технологічний стек для її реалізації;

− реалізовано прототип програмної системи, що дозволяє швидко та точно знаходити релевантні дані з документів.

Розроблена система демонструє високий рівень адаптивності до різних задач обробки текстової інформації, що дозволяє використовувати її як універсальний інструмент для автоматизації аналітичних процесів. Вона здатна ефективно працювати як із структурованими, так і неструктурованими даними, забезпечуючи користувачів точними й релевантними результатами навіть за умов великого обсягу вхідної інформації.

Однією з ключових переваг системи є можливість масштабування для роботи з великими наборами даних. Завдяки використанню оптимізованих алгоритмів пошуку й аналізу, зокрема на базі векторних представлень тексту, система забезпечує високу продуктивність навіть за умов одночасної обробки багатьох запитів. Поєднання методів semantic search і BM25 дозволило підвищити точність пошукових запитів, що особливо важливо для роботи з великою кількістю різнотипних документів.

Додатково, система була інтегрована з сервісами для аналізу метаданих, що дозволяє проводити глибокий аналіз документів, витягуючи важливу

контекстну інформацію, як-от ключові слова, назви розділів, дати, імена авторів тощо. Це спрощує навігацію великими базами даних та полегшує користувачам доступ до необхідної інформації.

Особливу увагу було приділено забезпеченню безпеки роботи системи. Дані, що надходять для аналізу, шифруються та обробляються в ізольованому середовищі, що запобігає несанкціонованому доступу. Завдяки інтеграції сучасних стандартів безпеки, система може бути використана навіть у критичних галузях, таких як фінансова аналітика або державне управління.

Крім того, система пропонує зручний інтерфейс для користувачів, який дозволяє легко налаштовувати параметри аналізу, формувати звіти та експортувати результати в популярні формати. Для розширення функціональності реалізована модульна архітектура, що дозволяє швидко додавати нові функції та адаптувати систему під специфічні потреби організацій.

Впровадження такої системи сприятиме зростанню продуктивності праці у багатьох сферах, зокрема у юридичному секторі для автоматизації обробки договорів, у фінансах для аналізу звітності та у технічній документації для оптимізації управління проектами. У майбутньому система може бути розширена за рахунок інтеграції з іншими інструментами штучного інтелекту, такими як чат-боти або системи підтримки прийняття рішень, що зробить її ще більш універсальною та ефективною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pavan Podila. HTTP: Протокол, який повинен розуміти кожний веброзробник (Частина 1). URL: https://code.tutsplus.com/uk/tutorials/http-theprotocol-every-web-developer-must-know-part-1--net-31177 (дата звернення: 08.10.2024).

2. MoesiF. Comparisons of API Architectural Styles. 22.01.2022 p. URL: https://www.moesif.com/blog/api-guide/comparisons-of-api-architecturalstyles/ (дата звернення 10.10.2024)

3. Lokesh Gupta. REST Architectural Constraints. 09.03.2022 p. URL: https://restfulapi.net/rest-architectural-constraints/ (дата звернення: 11.10.2024)

4. Lauren Schaefer. What is NoSQL? URL: https://www.mongodb.com/nosqlexplained (дата звернення: 11.10.2024).

5. Tutorialspoint. MVC Pattern. URL: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm (дата звернення: 21.10.2024) .

6. Refactoring Guru. Адаптер. URL: https://refactoring.guru/uk/designpatterns/adapter (дата звернення: 21.10.2024) .

7. Pair. What is Cron and How do I Use It? URL: https://www.pair.com/support/kb/configuring-cron/ (дата звернення: 22.10.2024).

8. Mapbox. API Reference. URL: https://docs.mapbox.com/mapbox-gl-js/api/ (дата звернення: 25.10.2024).

9. Recharts Group. Rechars. URL: https://recharts.org/en-US/ (дата звернення: 25.10.2024).

10. Hugging Face. Documentation on Transformers. URL: https://huggingface.co/docs/transformers/index (дата звернення: 15.11.2024).

11. James Serra. The Rise of Retrieval-Augmented Generation (RAG). URL: https://james-serra.com/index.php/2023/05/the-rise-of-retrieval-augmented-generation-rag (дата звернення: 15.11.2024).

12. Google AI Blog. Retrieval-Augmented Generation for Natural Language Processing. URL: https://ai.googleblog.com/2021/01/retrieval-augmented-generation-for.html (дата звернення: 15.11.2024).

13. Microsoft Research. Retrieval-Augmented Generation: An Overview. URL: https://www.microsoft.com/research/blog/retrieval-augmented-generation-an-overview/ (дата звернення: 16.11.2024).

14. Milvus Documentation. High-performance Vector Database for Embedding Retrieval. URL: https://milvus.io/docs/overview.md (дата звернення: 16.11.2024).

15. OpenAI. API Reference. URL: https://platform.openai.com/docs/ (дата звернення: 17.11.2024).

16. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. URL: https://arxiv.org/abs/1810.04805 (дата звернення: 17.11.2024).

17. T. Mikolov, K. Chen, G. Corrado, J. Dean. Efficient Estimation of Word Representations in Vector Space. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 17.11.2024).

18. Положення про кваліфікаційні роботи (проєкти) здобувачів вищої освіти Національного Авіаційного Університету: наказ Міністерства освіти і науки від 01.04.2024 р. № 122/09. С. 22-36 (дата звернення: 17.11.2024).

ДОДАТКИ

Додаток А

Код сервера, файл user_profile/models.py

```python
from datetime import datetime
from enum import Enum
from pydantic import BaseModel, EmailStr, Field
from typing import List, Optional

# Enum для типу відправника
class SenderType(str, Enum):
    user = "user"
    assistant = "assistant"

# Модель користувача
class UserBase(BaseModel):
    id: int = Field(..., description="Унікальний ідентифікатор користувача")
    name: str = Field(..., description="Ім'я або нікнейм користувача")
    email: EmailStr = Field(..., description="Електронна пошта користувача")

class UserCreate(UserBase):
    pass

class UserResponse(UserBase):
    class Config:
        orm_mode = True
```

Додаток Б

Код сервера, файл chat_schema.py

```python
from datetime import datetime
from enum import Enum
from pydantic import BaseModel, EmailStr, Field
from typing import List, Optional

# Модель чату
class ChatBase(BaseModel):
    id: int = Field(..., description="Унікальний ідентифікатор чату")
    user_id: int = Field(..., description="Зв'язок з користувачем")
    created_at: datetime = Field(..., description="Дата та час створення чату")
    status: str = Field(..., description="Статус чату")
    type: str = Field(..., description="Тип чату")

class ChatCreate(ChatBase):
    pass

class ChatResponse(ChatBase):
    messages: List['MessageResponse'] = Field(default_factory=list,
description="Список повідомлень чату")

    class Config:
        orm_mode = True
```

Код сервера, файл messages_schema.py

```python
from datetime import datetime
from enum import Enum
from pydantic import BaseModel, EmailStr, Field
from typing import List, Optional

# Модель повідомлення
class MessageBase(BaseModel):
    id: int = Field(..., description="Унікальний ідентифікатор повідомлення")
    chat_id: int = Field(..., description="Зв'язок з чатом")
    sender_type: SenderType = Field(..., description="Тип відправника (user або assistant)")
    text: str = Field(..., description="Текст повідомлення")
    created_at: datetime = Field(..., description="Дата та час відправки повідомлення")
    is_read: bool = Field(..., description="Флаг прочитаного повідомлення")
    reply_to: Optional[int] = Field(None, description="Відповідь на інше повідомлення")

class MessageCreate(MessageBase):
    pass

class MessageResponse(MessageBase):
    class Config:
        orm_mode = True
```