

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ПРИРОДОКОРИСТУВАННЯ**

**ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ**

**ТЕХНОЛОГІЙ**

**КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

# **КВАЛІФІКАЦІЙНА РОБОТА**

другого (магістерського) рівня вищої освіти

на тему: «Створення веб-додатку для генерування коду на основі текстового запиту із використанням OpenAI та Google API»

Виконав: здобувач 6 курсу групи Іт-62

Спеціальності 126 «Інформаційні системи та технології»

(шифр і назва)

Мартиняк Максим Андрійович

(Прізвище та ініціали)

Керівник: Падюка Р. І.

(Прізвище та ініціали)

Рецензент: \_\_\_\_\_

(Прізвище та ініціали)

**ДУБЛЯНИ-2024**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ЛЬВІВСЬКИЙ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ПРИРОДОКОРИСТУВАННЯ  
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ  
ТЕХНОЛОГІЙ  
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти  
Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри \_\_\_\_\_

д.т.н., проф. А. М. Тригуба \_\_\_\_\_

« \_\_\_\_\_ » \_\_\_\_\_ 2024 р.

### ЗАВДАННЯ

на кваліфікаційну роботу здобувачу

Мартиняку Максиму Андрійовичу

1. Тема роботи: «Створення веб-додатку для генерування коду на основі

текстового запиту із використанням OpenAI та Google API»

Керівник роботи Падюка Роман Іванович, к.т.н., в.о. доцента.

затверджені наказом по університету від 28 квітня 2023 року №133/к-с.

2. Строк подання здобувачем роботи 10.01.2024р.

3. Вихідні дані до роботи: база даних з інструкціями для створення асистентів та їх ідентифікаторами в OpenAI API; розроблений бекенд та фронтенд для веб-додатку; методика створення просунутих інструкцій для створення асистентів у OpenAI API.

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

1. Аналіз стану питання в теорії та практиці

2. Обґрунтування, вибір та реалізація інструментарію вирішення задачі

3. Результати вирішення задачі

4. Охорона праці та безпека у надзвичайних ситуаціях

5. Визначення ефективності розробки веб-додатку

Висновки

Список використаних джерел

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових схем та моделей): Тема, автор, керівник магістерської роботи; Мета, завдання, об'єкт, предмет дослідження; Огляд моделей OpenAI; Огляд стеку розробленого веб-додатку; Огляд архітектури веб-додатку (рис. 2.3 і 2.4);

Інтеграція з OpenAI та Google API; Відео демонстрація роботи веб-додатку; Порівняння часу відгуку різних систем (рис. 5.3); Огляд активності використання веб-додатку (рис. 5.20 та 5.21); Огляд цінової політики створеного веб-додатку (рис. 5.22); Висновки.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5	<i>Падюка Р.І., в.о. доцента кафедри інформаційних технологій</i>		
4	<i>Городецький І.М., доцент кафедри фізики, інженерної механіки та безпеки виробництва</i>		

7. Дата видачі завдання 01 травня 2023 р.

#### Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Написання першого розділу та означення головних завдань роботи	01.05.-30.05.23	
2	Виконання другого розділу та формування головних показників для розрахунків	01.06.-30.06.23	
3.	Виконання третього розділу та формування початкових даних	01.09.-30.09.23	
4.	Виконання четвертого розділу та узагальнення отриманих результатів магістерської роботи	01.10.-30.10.23	
6.	Вартісне оцінення ефективності пропозицій роботи	01.11.-30.11.23	
7.	Завершення оформлення розрахунково-пояснювальної записки та аркушів графічної частини	01.12.-30.12.23	
8.	Завершення роботи в цілому	01.01.-10.01.24	

Здобувач \_\_\_\_\_ Мартиняк М. А.  
(підпис)

Керівник роботи \_\_\_\_\_ Падюка Р. І.  
(підпис)

УДК 004.9 : 631.1

Створення веб-додатку для генерування коду на основі текстового запиту із використанням OpenAI та Google API  
Мартиняк М.А. Кафедра ІТ – Дубляни, Львівський НАУ, 2024.  
Кваліфікаційна робота: 102 с. текст. част. 63 рис., 12 табл., 11 арк. ілюстраційного матеріалу, 40 джерел.

Проведено огляд інформаційних систем аналогів. Наведено різні підходи, для вирішення задачі генерації коду. Сформульовано завдання кваліфікаційної роботи.

Проведено аналіз різних підходів побудови веб-додатку. Оглянуто ряд пропозицій, які надають хмарні сервіси, для розгортання веб-додатку.

Окреслено задачі розробки веб-додатку для генерації коду. Вибрано засоби реалізації. Проведено проектування веб-додатку на різних рівнях, а саме: інфраструктурному, алгоритмічному та системному.

Подано особливості реалізації даного веб-додатку зі включенням серверної та користувацької частини.

Розроблено заходи із охорони праці та безпеки у надзвичайних ситуаціях під час використання веб-додатку.

Визначено показники ефективності розробленого веб-додатку.

**Ключові слова:** веб-додаток, генерація коду, машинне навчання, мовна модель, OpenAI API.

## ЗМІСТ

<u>ЗМІСТ .....</u>	<u>5</u>
<u>ВСТУП.....</u>	<u>7</u>
<u>РОЗДІЛ 1. АНАЛІЗ СТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ.....</u>	<u>9</u>
1.1. АНАЛІЗ ІСТОРИЧНОГО КОНТЕКСТУ ГЕНЕРАЦІЇ КОДУ .....	9
1.2 Огляд сучасних методів генерації коду .....	10
1.3 Огляд можливостей API OpenAI та Google для генерації коду .....	12
1.4 Визначення основних завдань дослідження .....	14
<u>РОЗДІЛ 2. ОБГРУНТУВАННЯ, ВИБІР ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТАРІЮ</u>	
<u>ВИРІШЕННЯ ЗАДАЧІ.....</u>	<u>17</u>
2.1 Огляд доступних інструментів та платформ для розробки веб-додатків.....	17
2.2 Критерії та обґрунтування вибору інструментарію для розробки веб-додатку .....	24
2.3 Планування архітектури додатку. ....	31
2.4 Аналіз методів інтеграції веб-додатку з API OpenAI. ....	34
<u>РОЗДІЛ 3. РЕЗУЛЬТАТИ ВИРІШЕННЯ ЗАДАЧІ .....</u>	<u>39</u>
3.1 Демонстрація роботи веб-додатку.....	39
3.2 Огляд інтеграцій з OpenAI API для створення асистентів та Google API для аналізу відгуків .....	55
3.3 Можливі шляхи для подальшого вдосконалення веб-додатку .....	64
<u>РОЗДІЛ 4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ .</u>	<u>69</u>

4.1 АНАЛІЗ НЕБЕЗПЕЧНИХ І ШКІДЛИВИХ ВИРОБНИЧИХ ЧИННИКІВ ТА РОЗРОБКА ЗАХОДІВ ЩОДО ПОКРАЩЕННЯ УМОВ ПРАЦІ .....	69
4.2 РОЗРОБКА ЛОГІКО-ІМІТАЦІЙНОЇ МОДЕЛІ ВИНИКНЕННЯ ТРАВМ І АВАРІЙ .....	69
4.3 РОЗРОБКА ЗАХОДІВ ЩОДО БЕЗПЕКИ У НАДЗВИЧАЙНИХ СИТУАЦІЯХ .....	72
<b><u>РОЗДІЛ 5. ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ РОЗРОБКИ ВЕБ-ДОДАТКУ .....</u></b>	<b>74</b>
5.1 МЕТОДИ ОЦІНКИ ЕФЕКТИВНОСТІ ВЕБ-ДОДАТКУ .....	74
5.2. РЕЗУЛЬТАТИ ОЦІНКИ ЕФЕКТИВНОСТІ РОБОТИ ВЕБ-ДОДАТКУ. ....	80
<b><u>ЗАГАЛЬНІ ВИСНОВКИ .....</u></b>	<b>100</b>
<b><u>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</u></b>	<b>101</b>

## ВСТУП

Розробка інформаційних систем (ІС) сьогодні переживає своєрідний ренесанс, адаптуючись до стрімких змін технологій і потреб ринку. Традиційний процес, що охоплює аналіз, проектування та впровадження ІС, еволюціонував, включаючи нові парадигми та інструментарій. Сьогодні центральну роль у цьому процесі відіграють автоматизовані системи, які використовують передові досягнення штучного інтелекту та обробки природної мови.

Практична необхідність виникає у розробці ІС, здатних оперативно адаптуватися до змін, що є особливо актуальним у сферах з високою динамікою змін та великою кількістю даних. Розробка такої системи вимагає не лише знання сучасних технологій, але й розуміння специфіки застосування алгоритмів машинного навчання, що враховують унікальність кожного конкретного випадку.

Однією з критичних задач є створення веб-додатку для генерації коду на основі запитів користувачів з використанням можливостей API, які надають OpenAI та Google. Такий підхід дозволить забезпечити більшу гнучкість, ефективність і точність у розробці програмного забезпечення, відповідаючи на потреби сучасного програмування.

Ця магістерська робота спрямована на розробку та аналіз веб-додатку, який за допомогою інтеграції сучасних API і технік prompt engineering зможе оптимізувати процес програмування. Важливість цього дослідження впливає з постійного зростання вимог до швидкості та якості розробки програмного забезпечення, а також з потреби упровадження інноваційних рішень у сфері ІТ.

Виклик полягає у створенні системи, що здатна не тільки виконувати запити користувачів, але й самонавчатися та адаптуватися до постійно змінюваного кодового середовища. Використання API OpenAI та Google як основи для розробки такого додатку є принципово новим підходом, який відкриває широкі можливості для оптимізації розробки ІС.

Актуальність роботи обумовлена стрімким розвитком галузі програмування та постійним попитом на інноваційні рішення. Таким чином, ця магістерська робота не

лише вносить вклад у теоретичне дослідження в області ІС, але й має велике прикладне значення для практики розробки програмного забезпечення.

**Об'єкт дослідження** - це процеси розробки інформаційних систем з акцентом на автоматизацію програмування та використання сучасних інформаційних технологій для підвищення ефективності програмних розробок. В контексті даної роботи, об'єкт дослідження зосереджується на використанні API, наданих OpenAI та Google, для створення систем, які здатні обробляти та оптимізувати запити користувачів з метою генерації високоякісного програмного коду.

**Предметом дослідження** є застосування технік prompt engineering для покращення інтерактивної взаємодії з користувачами та оптимізації процесів програмування в межах розробленої системи. Предмет дослідження включає аналіз методів інтеграції API, процесів адаптації до змін у програмних бібліотеках та розробку асистентів для підтримки різноманітних програмних мов та фреймворків.

**Мета роботи** полягає в розробці та дослідженні ефективності веб-додатку для генерації коду, який використовує інтеграцію сучасних API, зокрема OpenAI та Google, для оптимізації програмувальних запитів через техніки prompt engineering. Очікується, що додаток забезпечить розширені можливості для точного та ефективного програмування, підтримуватиме актуалізацію інформації про існуючі бібліотеки та їх кодову базу, а також дозволить користувачам отримувати відповіді, що відповідають сучасним вимогам до швидкості та якості розробки програмного забезпечення. Основною теоретичною цінністю роботи є аналіз можливостей штучного інтелекту у контексті генерації коду, тоді як прикладне значення полягає у створенні робочого прототипу зі вбудованою системою оцінки ефективності його використання.



## РОЗДІЛ 1. АНАЛІЗ СТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ

### 1.1. Аналіз історичного контексту генерації коду

Історія генерації коду має глибокі корені, що сягають ранньої епохи комп'ютерної науки. Спочатку програмування виконувалося на низькому рівні, з безпосереднім використанням машинного коду, що було складним і часомістким процесом. Поява мов високого рівня, як-от FORTRAN у 1950-х роках (рис. 1.1), позначила перші спроби абстрагування від машинної логіки, спрощуючи процес програмування [1]. Такі інновації сприяли переходу від ручного кодування до більш автоматизованих форм створення програмного забезпечення.

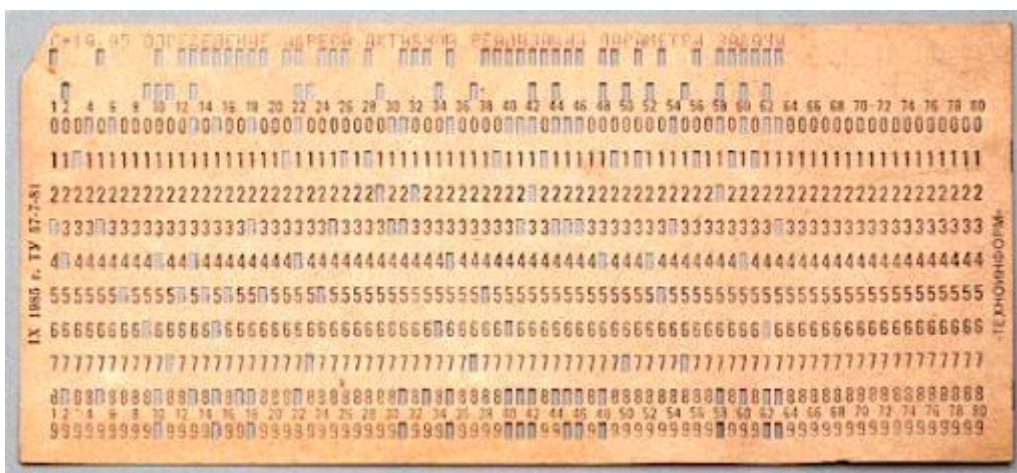


Рисунок 1.1 – FORTRAN у 1950-х роках

У 1960-1970-х роках з розвитком структурного програмування було запроваджено нові парадигми, що дозволили програмістам більш ефективно управляти складністю програм. Ці підходи лягли в основу сучасних методологій проектування та генерації коду [2].

З приходом об'єктно-орієнтованого програмування в 1980-х роках виникли нові можливості для генерації коду. Використання класів, наслідування та інкапсуляції сприяло створенню гнучкіших і більш масштабованих систем [3].

У 1990-х роках з появою перших інтегрованих розробницьких середовищ (IDE), які надавали інструменти для автоматизації створення коду, програмісти отримали

змогу значно прискорити розробку продуктів. Об'єктно-орієнтоване програмування та паттерни проектування стали стандартом у створенні програмного забезпечення [4].

Останні десятиліття принесли революцію в генерації коду через розвиток штучного інтелекту. Алгоритми машинного навчання та нейронні мережі відкрили двері до автоматичного створення коду, що відповідає складним вимогам і може адаптуватися до змін у процесі розробки [5]. Це стало можливим завдяки прогресу в обробці природної мови, що дозволяє перетворювати людське спілкування в код, як це робить OpenAI Codex [6].

Сучасна генерація коду нерозривно пов'язана з використанням різноманітних API, що розширюють можливості програмного забезпечення, інтегруючи його з великою кількістю онлайн сервісів. Google API — це один із таких інструментів, що надає доступ до широкого спектру сервісів від аналітики до машинного перекладу, які можуть бути інтегровані при розробці додатків [7].

Таким чином, аналіз історичного контексту генерації коду показує, що цей процес є відображенням загального розвитку програмної інженерії. Від ранніх автоматизованих систем до сучасних рішень з використанням штучного інтелекту, генерація коду продовжує розвиватись, пропонуючи все більш складні та інноваційні підходи.

## **1.2 Огляд сучасних методів генерації коду**

Сучасні технології генерації коду зазнали значних змін завдяки розвитку штучного інтелекту, зокрема, технологій обробки природної мови (NLP). Останнім часом особливу увагу привертає інструмент GitHub Copilot, розроблений на базі OpenAI Codex, який є одним із найпередовіших засобів у цій галузі. Він здатен не просто перетворювати текстові описи на код, але й надавати рекомендації та варіанти вдосконалення вже існуючого коду, враховуючи контекст та задані параметри [8].

Інтегровані середовища розробки (IDE) також включають в себе все більше інтелектуальних функцій, які значно спрощують розробку програмного забезпечення. Сучасні IDE надають розширені можливості для автоматичного аналізу коду,

рефакторингу, та інших аспектів, що вимагають глибокого розуміння структури програми [9].

API, такі як Google Cloud Vision або Natural Language API, відкривають нові можливості для розробників, дозволяючи легко інтегрувати складні функції штучного інтелекту в їх програми, як це зображено на рисунку 1.2. Використання цих API може зменшити кількість ручного кодування, автоматизувати тести та документацію, а також покращити взаємодію з користувачем [10].

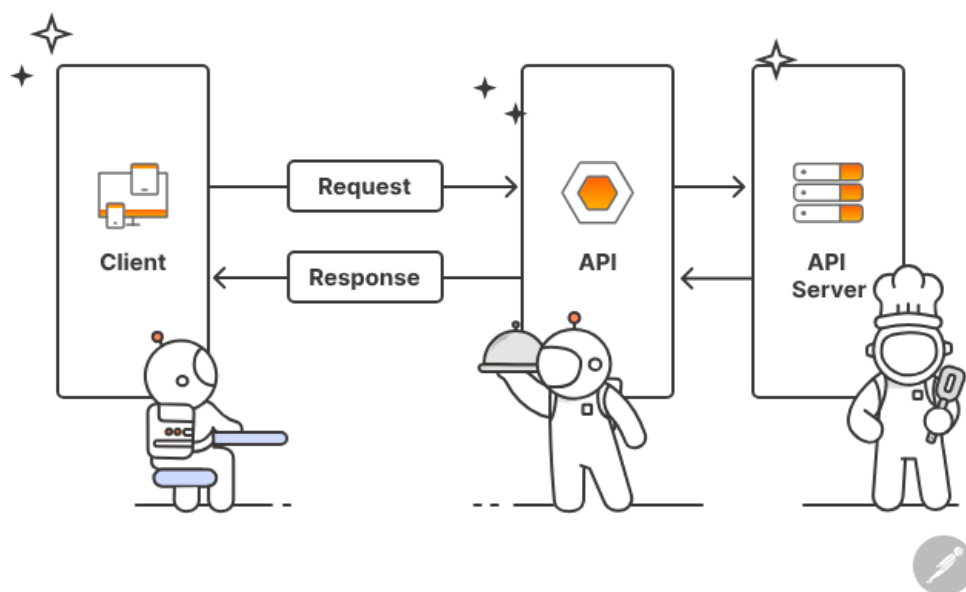


Рисунок 1.2 – Схематичне зображення роботи API

Застосування машинного навчання для оптимізації та генерації коду постійно розвивається. На додаток до навчання з підкріпленням, алгоритми навчання з перенесенням (transfer learning) дозволяють створювати моделі, здатні адаптувати знання, отримані при роботі над одним проектом, для використання в інших [11].

Важливо відзначити, що співпраця в галузі open source значно підвищує якість і швидкість розвитку інструментів генерації коду. Спільноти розробників навколо платформ як GitHub сприяють вдосконаленню технік генерації коду, ділячись своїми програмами, патчами та плагінами [12].

Для розуміння впливу і потенціалу сучасних методів генерації коду, важливо вивчати академічні дослідження, технічні документації та індустріальні випадки їх застосування. Аналіз цих матеріалів вказує на швидкий прогрес у цій області та

нарощування потенціалу штучного інтелекту, який може стати основою для наступних поколінь систем автоматизації розробки.

### 1.3 Огляд можливостей API OpenAI та Google для генерації коду

Розвиток інструментів для автоматизованої генерації коду суттєво вплинув на індустрію програмування. Серед лідерів у цій області — API від OpenAI та Google, які надають розробникам розширені можливості для створення та оптимізації коду.

OpenAI Codex, зокрема, є вражаючим прикладом досягнення у цій сфері. Використовуючи передові техніки глибокого навчання, Codex може перекладати інструкції, виражені природною мовою, у робочий код, що відкриває нові горизонти для програмістів та розробників. Цей API не тільки підвищує продуктивність, але й сприяє навчанню користувачів, демонструючи рекомендовані практики та можливі рішення програмних завдань (рис. 1.3)[13].

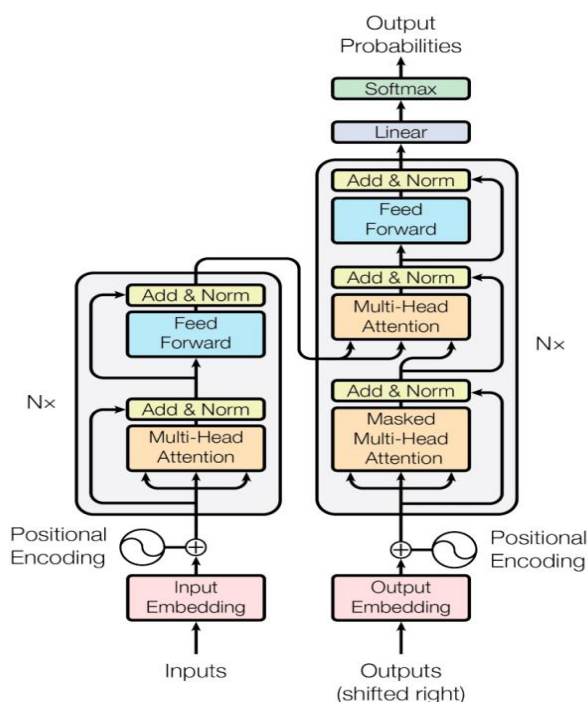


Рисунок 1.3 – Схематичне зображення OpenAI Codex

Така технологія особливо корисна у великих проектах, де потрібно швидко генерувати бутстреп-код для нових компонентів або модулів. Вона також може

слугувати підтримкою для новачків у програмуванні, надаючи їм керівництво та відгуки в процесі навчання [14].

Google API пропонує широкий спектр можливостей (рис. 1.4) для розробки веб-додатків, включаючи інтеграцію з хмарними сервісами, аналітикою, інтелектуальним пошуком та іншими. Ці інструменти можуть бути використані для підвищення функціональності та інтерактивності додатків, забезпечуючи їх інтеграцію з передовими технологіями обробки даних та штучного інтелекту [15].

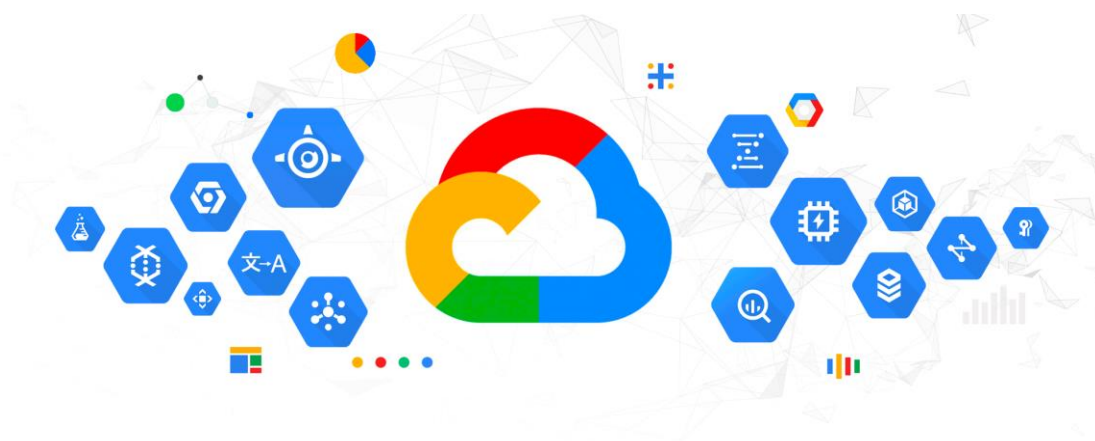


Рисунок 1.4 – Візуалізація сервісів Google API

Крім того, Google API дозволяють розробникам використовувати машинне навчання для аналізу поведінки користувачів та оптимізації UX/UI, що може значно підвищити якість кінцевих продуктів. Це також сприяє створенню адаптивного дизайну, який реагує на потреби користувачів в реальному часі [16].

Використання цих API у сукупності може значно скоротити час розробки, зменшити кількість помилок та підвищити якість програмного коду. Це, у свою чергу, може сприяти швидкому прототипуванню та ітеративному розробленню, дозволяючи розробникам швидко тестувати та адаптувати свої продукти до змінюваних потреб ринку [17].

На закінчення, важливо підкреслити, що постійне оновлення і поліпшення API від OpenAI та Google, залучення розробницької спільноти та відгуки користувачів є критичними для еволюції інструментів генерації коду. Таке динамічне середовище

сприяє інноваціям та розвитку нових методологій програмування, які можуть змінити майбутнє програмної інженерії [18].

#### **1.4 Визначення основних завдань дослідження**

У рамках магістерського дослідження, які впливають з теми дослідження "Створення веб-додатку для генерування коду на основі текстового запиту із використанням Open AI та Google API", визначаються наступні основні завдання:

Перше завдання: Аналіз існуючих технологій та підходів до генерації коду, включаючи дослідження сучасних засобів NLP (Natural Language Processing) і їх застосування в програмуванні. Це передбачає вивчення робіт, опублікованих у наукових журналах та конференціях, а також аналіз документації розробників від OpenAI та Google.

Друге завдання: Розробка теоретичної моделі для системи генерації коду, яка буде інтегрувати NLP можливості OpenAI для інтерпретації текстових запитів та Google API для додавання додаткової функціональності, як-от синтаксичний аналіз або семантичне розуміння.

Третє завдання: Визначення архітектурних вимог до веб-додатку. Це включає визначення необхідних технологій стеку, вибір програмних мов та фреймворків, а також планування інтерфейсу користувача з урахуванням UX/UI кращих практик.

Четверте завдання: Розробка методології для оцінки ефективності генерації коду. Оцінка має базуватися на кількісних та якісних метриках, включаючи точність коду, продуктивність, масштабованість, і час відгуку.

П'яте завдання: Реалізація прототипу веб-додатку. Це завдання передбачає практичну розробку прототипу з урахуванням раніше визначеної архітектури та методології. Прототип повинен демонструвати основні можливості системи генерації коду і бути здатним обробляти реальні запити користувачів.

Шосте завдання: Проведення експериментального тестування прототипу. Експерименти повинні включати різні сценарії використання, щоб визначити межі системи, її надійність та ефективність в різних умовах.

Сьоме завдання: Аналіз отриманих результатів і порівняння з існуючими рішеннями. Цей аналіз повинен виявити сильні та слабкі сторони прототипу в контексті існуючих рішень і визначити шляхи для подальшого вдосконалення.

Восьме завдання: Розробка детальної документації для веб-додатку, яка охоплює технічні аспекти, керівництва користувача, а також опис алгоритмів та API використання.

Дев'яте завдання: Формулювання перспектив для майбутніх досліджень на базі виявлених тенденцій, проблем та можливостей, що виникають у процесі дослідження.

Десяте завдання: Підготовка рекомендацій для розробників програмного забезпечення щодо застосування веб-додатку для генерації коду та можливих напрямків використання таких систем у різних областях інформатики.

Ці завдання визначають структуру дослідницького процесу та будуть використані для розробки веб-додатку, який може автоматизувати процес програмування та надати нові можливості для розробників програмного забезпечення.

## Висновки до розділу 1

1. Проаналізовано історичний контекст генерації коду. Аналіз історичного контексту генерації коду показує, що цей процес є відображенням загального розвитку програмної інженерії. Від ранніх автоматизованих систем до сучасних рішень з використанням штучного інтелекту, генерація коду продовжує розвиватись, пропонуючи все більш складні та інноваційні підходи.

2. Оглянуто сучасні методи генерації коду. Визначено та розглянуто найпопулярніші сучасні інструменти для цього. Досліджено, як дані інструменти можна інтегрувати у інтегроване середовище розробки.

3. Розглянуто можливості OpenAI та Google API. Як результат було виділено основні інструменти, для подальшого використання.

4. Визначено основні завдання дослідження. На основі цих завдань буде сформована дана кваліфікаційна робота. Можна виділити декілька ключових, а саме: аналіз OpenAI та Google API, розробка архітектури веб-додатку (інфраструктурна,

схема бази даних, серверна), розробка прототипу веб-додатку, аналіз ефективності розробленого веб-додатку, визначення сильних та слабких сторін.



## РОЗДІЛ 2.

### ОБГРУНТУВАННЯ, ВИБІР ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТАРІЮ ВИРІШЕННЯ ЗАДАЧІ

У другому розділі цієї магістерської роботи зосереджується увага на критичному аспекті розробки веб-додатку - виборі та обґрунтуванні необхідного інструментарію та технологічної платформи. Цей вибір визначає технічні можливості проекту, його майбутню гнучкість, масштабованість та ефективність.

Перший розділ вже надав теоретичний фундамент та історичну перспективу генерації коду, а також сучасні підходи та інструменти у цій сфері. Тепер настає завдання дослідити та вибрати конкретні технології та фреймворки, які оптимально відповідають вимогам та цілям розроблюваного веб-додатку. Огляду підлягають доступні інструменти - від фреймворків для розробки фронтенду та бекенду, як-то React та Django, до інтеграції з API, зокрема OpenAI та Google.

Цей розділ включає критичний огляд та порівняльний аналіз різних технологічних рішень, їхніх переваг та недоліків, а також розгляд критеріїв для вибору найбільш підходящих інструментів. Важливим є розробка чіткої архітектури веб-додатку, яка передбачає визначення основних компонентів та їх взаємодій. Паралельно з технічними аспектами, акцент робиться на впливі цих рішень на продуктивність, безпеку та здатність додатку адаптуватися до майбутніх вимог.

Таким чином, цей розділ має на меті створити міцний фундамент для подальшої розробки та демонстрації роботи веб-додатку, забезпечуючи відповідність вибраних інструментів та технологій високим стандартам ефективності, гнучкості та інноваційності.

#### **2.1 Огляд доступних інструментів та платформ для розробки веб-додатків.**

У сучасному світі, де цифрові технології розвиваються з неймовірною швидкістю, веб-додатки стають невід'ємною частиною нашого повсякденного життя. Вони служать різноманітним цілям - від спрощення бізнес-процесів до забезпечення розваг. Розвиток таких додатків вимагає від розробників глибоких знань у

використанні сучасних технологій та інструментів. В даному розділі буде розглянуто ключові інструменти та підходи, які є необхідними для ефективного створення веб-додатків [19].

Перш за все, важливо розуміти, що розробка веб-додатків включає як фронтенд (клієнтська частина), так і бекенд (серверна частина). Для фронтенду часто використовуються технології, такі як HTML, CSS, та JavaScript, а також сучасні фреймворки та бібліотеки, як-от React або Angular. Ці інструменти дозволяють створювати інтерактивні та користувацькі інтерфейси, які є привабливими та зручними для користувачів.

З іншого боку, бекенд вимагає використання серверних мов програмування та фреймворків, таких як Python з Django, Ruby on Rails, або Node.js. Ці технології дозволяють ефективно обробляти запити користувачів, управляти базами даних та інтегрувати різні веб-сервіси.

Однією з ключових особливостей сучасної веб-розробки є інтеграція з зовнішніми API, в даній роботі це OpenAI. Цей сервіс надає розробникам доступ до потужних інструментів, таких як машинне навчання, обробка мови та інші, що значно розширює можливості створюваних додатків.

Фронтенд розробка - це ключовий аспект у створенні веб-додатків, який безпосередньо впливає на взаємодію користувачів з додатком. Вона включає в себе розробку того, що користувачі бачать та з чим взаємодіють у своєму браузері, таких як веб-сторінки, користувацькі інтерфейси, та анімації. Основні технології, які використовуються у фронтенд розробці, включають HTML для структури веб-сторінок, CSS для стилізації, та JavaScript для додання інтерактивності.

Сучасні підходи до фронтенд розробки включають використання фреймворків і бібліотек, таких як React, Angular або Vue.js. Ці інструменти спрощують процес розробки, надаючи готові компоненти та патерни для повторного використання коду, що підвищує ефективність розробки та забезпечує більшу консистентність інтерфейсу (рис. 2.1). Також, ці фреймворки пропонують реактивне програмування та компонентний підхід, це дозволяє легше управляти станом додатку та покращує взаємодію між компонентами.

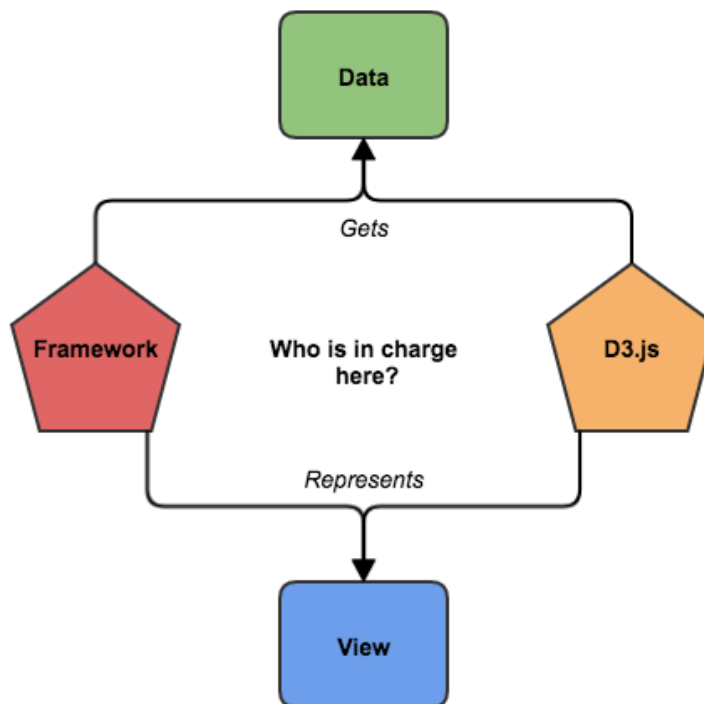


Рисунок 2.1 – Інтеграція Angular/Vue.js

Респонсивний дизайн є ще одним важливим аспектом фронтенд розробки, який забезпечує оптимальний перегляд та взаємодію з веб-сторінкою на різних пристроях, від мобільних телефонів до комп'ютерів. Використання CSS та гнучких сіток дозволяє створювати дизайни, які легко адаптуються до різних розмірів екрану.

Зрештою, фронтенд розробка сьогодні також орієнтується на поліпшення користувацького досвіду (UX) та інтерфейсу (UI), забезпечуючи інтуїтивне та зручне використання додатків. Це включає в себе все, від архітектури інформації до візуального дизайну, забезпечуючи, щоб додаток був не тільки функціональним, але й привабливим та легким у використанні. Далі, буде розглянуто три основні, найпопулярніші фреймворки.

React – розроблений Facebook, є одним з найпопулярніших JavaScript фреймворків для створення UI. Він базується на компонентному підході, дозволяючи розробникам будувати великі веб-додатки, які можуть змінювати дані, не перезавантажуючи сторінку. React значно спрощує процес створення інтерактивних компонентів, надаючи їм реактивні та ефективні оновлення. Його екосистема включає

такі інструменти, як Redux для управління станом додатку та React Router для навігації [20].

Angular – розроблений Google, є платформою та фреймворком для створення клієнтських додатків з використанням HTML та TypeScript. Він надає розробникам набір інструментів для побудови додатків, які працюють на різних платформах, від веб-браузерів до мобільних пристроїв. Angular пропонує широкі можливості з двосторонньою прив'язкою даних, інтеграцією залежностей, а також набором потужних можливостей для управління маршрутизацією, формами та багато іншого [21].

Vue.js - це прогресивний JavaScript фреймворк, який використовується для побудови інтерфейсів користувача. Його легко інтегрувати з іншими проектами та бібліотеками, що робить його ідеальним вибором для розробки складних односторінкових застосунків (SPA). Vue.js вирізняється своєю гнучкістю та простотою, пропонуючи розробникам інтуїтивно зрозумілу архітектуру та реактивні компоненти. Він також підтримує інструменти, такі як Vuex для управління станом та Vue Router для маршрутизації, роблячи його комплексним рішенням для фронтенд розробки [22].

Бекенд розробка є фундаментальною складовою веб-додатків, що забезпечує їх серверну функціональність. Це включає в себе серверну логіку, управління базами даних, інтеграцію з системами третіх сторін, а також обробку та зберігання даних.

У бекенд розробці широко використовуються такі мови програмування, як Python, Ruby, Java, та JavaScript (Node.js), кожна з яких має свої фреймворки та інструменти. Фреймворки, як Django для Python або Express для Node.js, надають розробникам структуровані шаблони і інструменти для ефективного створення безпечного та масштабованого серверного коду. Бекенд розробка також включає роботу з системами управління базами даних, такими як MySQL, PostgreSQL, або MongoDB, для забезпечення ефективного зберігання та маніпулювання даними.

Зрештою, аспекти безпеки та оптимізації продуктивності є важливими у бекенд розробці, оскільки вони забезпечують надійну та стабільну роботу веб-додатків. Це

включає в себе захист від загроз, таких як SQL ін'єкції, кросс-сайт скриптинг (XSS), та інших видів атак.

Python – відомий своєю читабельністю та ефективністю, є вибором багатьох розробників для бекенд-розробки. У поєднанні з Django, високорівневим фреймворком, він стає потужним інструментом для швидкого розвитку веб-додатків. Django славиться своїм принципом "батарейки включені", що означає, що він надає вбудовані рішення для багатьох типових завдань веб-розробки, таких як аутентифікація, сесії, адміністрування, та багато іншого. Це робить Django ідеальним для створення безпечних та масштабованих веб-додатків, зі значною економією часу розробника [23].

Node.js – використовує JavaScript на стороні сервера, дозволяє розробникам використовувати єдину мову програмування для написання клієнтського та серверного коду. Це не тільки спрощує процес розробки, але й підвищує його ефективність. Node.js відрізняється своєю подієво-орієнтованою архітектурою та неблокуючим введенням-виведенням, що робить його особливо підходящим для розробки масштабованих мережевих додатків, таких як веб-сервери, додатки в реальному часі, та інші. Node.js також має величезний екосистему модулів через NPM, що робить його гнучким та зручним у використанні [24].

Java – одна з найпопулярніших мов програмування, відома своєю стабільністю, масштабованістю, та безпекою, є широко використовуваною мовою для бекенд-розробки в корпоративних середовищах. Завдяки своїй міцній архітектурі та великій кількості бібліотек та фреймворків, таких як Spring та Hibernate, Java дозволяє розробникам створювати високопродуктивні та надійні серверні додатки. Її віртуальна машина (JVM) забезпечує високий рівень портативності та ефективності, що робить Java вибором для різноманітних веб-додатків, від маленьких стартапів до великих корпоративних систем [25].

Хмарні технології революціонізували спосіб, яким компанії розгортають та управляють своїми ІТ-ресурсами. Вони дозволяють зберігати дані та запускати додатки на віддалених серверах у хмарі, забезпечуючи гнучкість, масштабованість та ефективність витрат (рис. 2.2)[26].

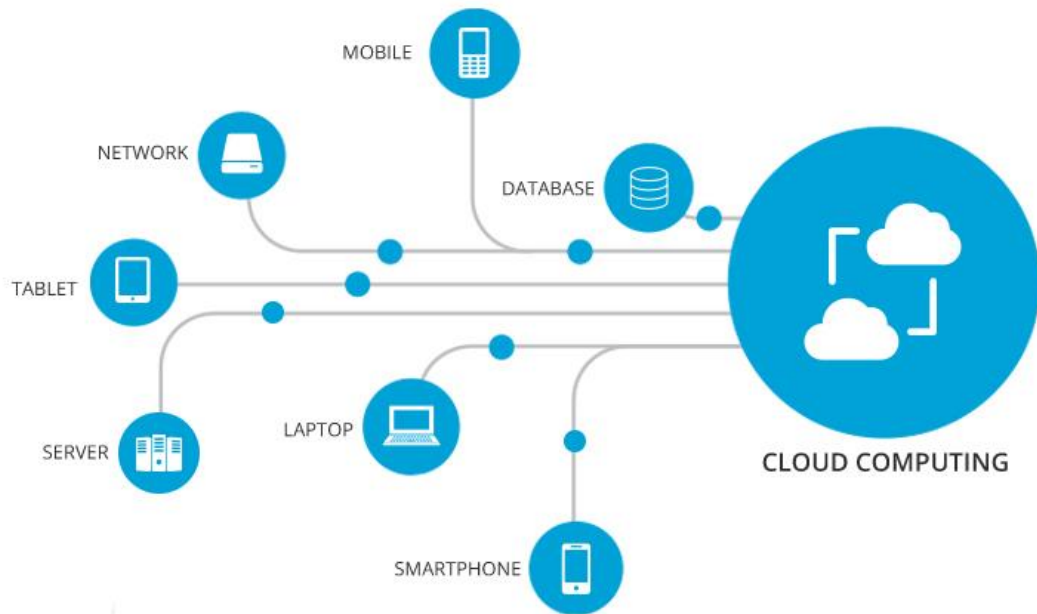


Рисунок 2.2 – Управління ресурсами за допомогою хмарних технологій

Amazon Web Services (AWS) від Amazon є одним з провідних провайдерів хмарних послуг, який пропонує широкий спектр продуктів та послуг. Це включає обчислювальні потужності (наприклад, EC2), зберігання даних (S3), бази даних (RDS), машинне навчання, інструменти для Інтернету речей, та багато іншого. AWS забезпечує високу надійність та безпеку, що робить його популярним вибором для бізнесу різного розміру - від стартапів до великих корпорацій. Його послуги дозволяють розробникам легко масштабувати та управляти інфраструктурою з мінімальними зусиллями [27].

Google Cloud є ще одним значним гравцем у світі хмарних технологій, який пропонує різноманітні послуги, включаючи Google Compute Engine, Google Cloud Storage, та BigQuery. Він відомий своїми потужними аналітичними та машинними навчальними можливостями, а також інтеграцією з різними Google сервісами. Google Cloud надає розробникам інструменти для створення, тестування та розгортання додатків на власній інфраструктурі Google, забезпечуючи високу продуктивність та надійність [28].

Обидва ці хмарні провайдери дозволяють компаніям оптимізувати витрати та ресурси, надаючи послуги "плати за використання", що робить їх ідеальними для

різноманітних застосувань, від хостингу веб-сайтів до складних обчислювальних задач.

CI/CD, що розшифровується як Continuous Integration та Continuous Delivery/Deployment, є ключовими практиками в сучасній розробці програмного забезпечення, спрямованими на підвищення ефективності та якості коду через автоматизацію процесів розробки.

Continuous Integration (CI) полягає в автоматичному об'єднанні коду, який розробники часто (зазвичай щодня) додають у спільний репозиторій. Це дозволяє швидко виявляти та виправляти конфлікти, забезпечуючи стабільність кодової бази. CI часто включає в себе автоматизоване тестування, що гарантує, що новий код не порушить існуючу функціональність.

Continuous Delivery є розширенням CI, яке передбачає, що код, який проходить через CI пайплайни, автоматично розгортається в тестове або продакшн середовище. Це забезпечує готовність програмного забезпечення до релізу в будь-який момент, скорочуючи часові затримки між написанням коду та його використанням кінцевими користувачами.

Continuous Deployment відноситься до автоматизації наступного етапу процесу, де кожна зміна, яка проходить всі етапи CI/CD пайплайна, автоматично розгортається в продакшн середовищі. Це забезпечує постійний потік оновлень до кінцевих користувачів, зменшуючи час та зусилля, необхідні для ручного розгортання.

CI/CD дозволяє командам розробників підтримувати високу частоту релізів з високою надійністю, водночас забезпечуючи ефективність процесів розробки та зниження ризиків пов'язаних із ручним розгортанням. Цей огляд інструментів та технологій відіграє ключову роль у формуванні архітектури та стратегії розробки веб-додатку, впливаючи на його функціональність, продуктивність та здатність задовольняти потреби користувачів [29].

## 2.2 Критерії та обґрунтування вибору інструментарію для розробки веб-додатку

Вибір правильних інструментів та технологій є критично важливим для успіху будь-якого проекту розробки програмного забезпечення. Цей вибір залежить від ряду чинників, включаючи специфічні вимоги проекту, командні навички, бюджет, та вимоги до часу виведення продукту на ринок.

Одним з найголовніших критеріїв підбору інструментарію є сумісність з OpenAI API. Це є ключовим через те, що основний функціонал, а саме – створення асистентів з надання коду та порад щодо нього, буде будуватись за рахунок інтеграції з даним API. Інтеграція на основі офіційних бібліотек забезпечить плавність роботи додатку та можливість використання передових функцій штучного інтелекту та машинного навчання, які надає OpenAI API. Представлено дві бібліотеки від OpenAI для взаємодії з їхнім API, серед яких одна реалізована на Python, а інша на Node.js, обидві мають однакову назву – `openai` [30].

Окрім цього стек розробки має бути підбрано так, щоб він був не лише продуктивний та ефективний, але й гнучкий. Код написаний на основі цього стеку повинен легко адаптуватись до змін у OpenAI API, так як воно стрімко розвивається і його функціонал часто змінюється та додається новий. Гарним прикладом є функціонал асистентів, зараз він перебуває у бета розробці, і є велика ймовірність, що зовсім скоро його буде переведено у продакшн, як результат імплементація асистентів у рамках бібліотеки `openai`, може суттєво відрізнятись від того, що можна побачити там зараз. Це означає, що інструментарій повинен бути модульним та здатним до швидкого оновлення без втрати загальної функціональності додатку.

Також, важливим критерієм є швидкість розробки. Фреймворки та мови програмування, які максимізують швидкість розробки, критично важливі для дотримання чітких термінів виконання проекту. Це включає інструменти, що підтримують швидке прототипування та ітеративну розробку. Вибір технологій, які забезпечують високий рівень повторного використання коду та легко впроваджуються у різних частинах проекту, також є важливим.



Додаток має бути продуктивним, це означає що необхідно забезпечити, щоб обрані інструменти пропонували оптимальну продуктивність кінцевого продукту. Це включає швидкість відгуку додатку, ефективну обробку даних та мінімальні затримки. Зосередження на оптимізації продуктивності на всіх етапах розробки - від запитів до бази даних, до рендерингу фронтенду - є фундаментальним для забезпечення високої якості користувацького досвіду.

Беручи до уваги всі вище наведені потреби та критерії, для серверної частини слід обирати між двома мовами, це Python та Node.js, саме вони мають існуючу імплементацію бібліотеки для взаємодії з OpenAI API. В рамках реалізації веб-додатку було обрано Python [30].

Python вирізняється своєю підтримкою об'єктно-орієнтованого програмування (ООП), що дозволяє розробникам будувати модульний і легко підтримуваний код на основі класів. Ця парадигма програмування, зосереджена на об'єктах та класах, сприяє ефективному управлінню та організації коду, забезпечуючи високу гнучкість та зрозумілість. Крім того, Python відомий своєю величезною базою фреймворків та пакетів, яка охоплює широкий спектр програмних потреб - від веб-розробки до наукових досліджень. Це забезпечує майже необмежені можливості для реалізації різних проектів. Щодо швидкості, хоча Python може не бути найшвидшою мовою програмування у термінах виконання коду, його продуктивність і швидкість розробки часто вважаються оптимальними, особливо завдяки зрозумілому синтаксису та великому співтовариству. Нарешті, Python вважається однією з найкращих мов для роботи з текстами, завдяки потужним бібліотекам для обробки природної мови та аналізу даних, що робить його ідеальним вибором для проектів, пов'язаних з машинним навчанням, обробкою мови та аналітикою даних.

У якості фреймворку було обрано Django та Django REST framework. Django є потужним фреймворком для веб-розробки на Python, який славиться своєю ефективністю та гнучкістю. Він дозволяє розробникам швидко створювати безпечні та масштабовані веб-додатки, завдяки своїй архітектурі, заснованій на принципах "батарейки включені". Django Admin Panel, одна з ключових особливостей фреймворку, дозволяє легко керувати даними у базі даних через інтуїтивно

зрозумілий графічний інтерфейс, без потреби у додатковому кодуванні. Ще однією важливою особливістю Django є його Object-Relational Mapper (ORM), який дозволяє розробникам взаємодіяти з базою даних за допомогою високорівневого абстрактного API. Django ORM значно спрощує процес роботи з базами даних, дозволяючи розробникам виконувати складні запити та маніпуляції з даними без необхідності писати чистий SQL код. Це робить роботу з базами даних більш інтуїтивною та безпечною, зменшуючи ризик помилок та вразливостей. Django REST framework (DRF) є доповненням до Django, яке забезпечує потужні і гнучкі інструменти для будівництва веб-API. DRF відзначається своєю здатністю до масштабування та підтримкою різних стилів API. Його серіалайзери відповідають за перетворення даних між складними типами, такими як модельні інстанції, та форматами, придатними для HTTP передачі, як JSON чи XML, спрощуючи обробку даних між клієнтами і сервером. Разом, Django та Django REST framework створюють сильний набір інструментів для сучасної веб-розробки, забезпечуючи розробників інструментами для швидкого та ефективного створення як веб-сайтів, так і API. Їх широка екосистема, ретельно продумана документація та активне співтовариство роблять їх одними з найбільш популярних виборів для розробників у всьому світі.

PostgreSQL був обраний як система управління базами даних для цього проекту, що є стратегічним вибором, враховуючи його сумісність та інтеграцію з Django. PostgreSQL є однією з найпопулярніших відкритих систем управління реляційними базами даних, відомою своєю надійністю, масштабованістю та потужними можливостями. Вона підтримує складні запити, транзакції, міцну інтеграцію з різними мовами програмування і є однією з найбільш розширюваних та стандартизованих баз даних на ринку. Однією з ключових переваг використання PostgreSQL разом з Django є легкість інтеграції між двома системами. Django ORM (Object-Relational Mapper) працює надзвичайно ефективно з PostgreSQL, забезпечуючи безшовну взаємодію між веб-додатком і базою даних. Ця комбінація дозволяє розробникам використовувати складні можливості PostgreSQL, такі як повнотекстовий пошук, транзакції на рівні бази даних та підтримку різноманітних типів даних, не втрачаючи при цьому простоту та ефективність, які надає Django

ORM. Використання PostgreSQL у парі з Django та Django REST Framework створює сильну, надійну та масштабовану платформу для будівництва веб-додатків. Це поєднання технологій гарантує, що база даних може ефективно обробляти великі обсяги даних та складні запити, в той же час надаючи потужні інструменти для розробки бекенду та фронтенду.

React.js був обраний для написання фронтенд-частини, оскільки це один з найбільш популярних та потужних JavaScript фреймворків для створення інтерактивних веб-інтерфейсів. React.js, розроблений Facebook, відомий своєю ефективністю та гнучкістю, що дозволяє створювати динамічні та відгуківі користувацькі інтерфейси. Основною концепцією React є компонентний підхід, який сприяє створенню перевикористовуваних та добре організованих блоків коду, що робить процес розробки більш ефективним та структурованим. React.js часто використовується разом з Django та Django REST Framework, оскільки це поєднання пропонує повний стек технологій для розробки сучасних веб-додатків. Django використовується як потужний бекенд-фреймворк, що забезпечує стабільну, безпечну та масштабовану архітектуру серверної частини, в той час як Django REST Framework дозволяє легко створювати API для обміну даними між бекендом та фронтендом. React.js, з іншого боку, використовується для створення динамічного та інтерактивного фронтенду, що взаємодіє з API, створеним за допомогою DRF. Ця інтеграція дозволяє розробникам використовувати сильні сторони обох технологій: надійність, безпеку та масштабованість Django для бекенду, а також гнучкість та відгуківість React для фронтенду. Такий підхід забезпечує високий рівень взаємодії між користувачем та веб-додатком, поліпшуючи загальний досвід користувача та ефективність розробки.

Після визначення основного стеку технологій для нашого веб-додатку, важливим кроком є вибір архітектурного рішення. В нашому випадку було вирішено прийняти мікросервісну архітектуру, де Django та React.js виконують ролі окремих мікросервісів. Такий підхід дозволяє нам розділити бекенд (Django) та фронтенд (React.js) частини нашого додатку на різні незалежні компоненти, кожен з яких може

бути розроблений, розгорнутий та масштабований окремо. Це забезпечує більшу гнучкість у розробці та управлінні, а також полегшує впровадження змін і оновлень.

Щодо хмарних рішень та інфраструктури, вони відіграють ключову роль у підтримці та масштабуванні нашого веб-додатку. Використання хмарних технологій, таких як Amazon Web Services (AWS), Google Cloud Platform (GCP) або Microsoft Azure, надає ряд переваг. Ці платформи пропонують гнучкі та масштабовані рішення для хостингу, зберігання даних, управління базами даних, безпеки та інших аспектів інфраструктури. Хмарні сервіси дозволяють легко масштабувати ресурси вгору або вниз відповідно до потреб додатку, що є важливим для забезпечення стабільної роботи під час піків навантаження. Хмарні платформи забезпечують високу доступність та надійність, розподіляючи ресурси через різні географічні регіони. Вони дозволяють більш гнучко управляти ресурсами, зменшуючи потребу в ручному управлінні інфраструктурою. Хмарні провайдери пропонують розширені можливості захисту, що є критично важливим для забезпечення безпеки даних та додатків. Таким чином, інтеграція мікросервісної архітектури з використанням хмарних рішень та інфраструктури надає нам велику гнучкість, масштабованість та надійність, що є вирішальним для успіху сучасного веб-додатку.

Вибір Amazon Web Services (AWS) як нашої хмарної платформи для реалізації веб-додатку був зумовлений рядом переконливих переваг, особливо важливих для стадії закритого бета-тестування. AWS вирізняється своєю легкою та зрозумілою інтеграцією між різними сервісами, що робить процес розгортання та управління ресурсами більш зручним та ефективним. Однією з ключових переваг є можливість використання безкоштовних EC2 інстансів, які AWS надає, що є ідеальним рішенням для розгортання бекенду та фронтенду нашого веб-додатку. Ці інстанси дозволяють нам ефективно тестувати та розвивати наш додаток, забезпечуючи необхідні ресурси без додаткових витрат. Крім того, використання RDS для управління базами даних дозволяє нам вигідно скористатися перевагами високоефективного управління даними з хмарою, забезпечуючи високий рівень доступності, безпеки та легкості масштабування. Інтеграція між EC2, RDS та іншими сервісами AWS є плавною, що дозволяє нам зосередитися на розробці та оптимізації додатку, замість того, щоб

витрачати час на керування інфраструктурою. У цілому, використання AWS у нашому проекті забезпечує не тільки економію коштів завдяки безкоштовним інстансам, але й високий рівень гнучкості та масштабованості, що є критично важливим для успішного бета-тестування та подальшого розвитку веб-додатку [31 - 32].

Впровадження практик Continuous Integration (CI) та Continuous Deployment/Delivery (CD) є ключовим аспектом в розвитку сучасних веб-додатків. CI/CD сприяє автоматизації процесів тестування та розгортання, що підвищує ефективність розробки, знижує ризик помилок та покращує якість кінцевого продукту. CI (Continuous Integration) включає в себе автоматичне тестування коду при кожному його оновленні, що гарантує, що нові зміни не порушують функціональності існуючої системи. Такий підхід дозволяє швидко виявляти та виправляти помилки, значно зменшуючи час, необхідний для розробки та тестування. CD (Continuous Deployment/Delivery) забезпечує автоматичне розгортання змін у продакшн-оточення після успішного проходження тестів. Це гарантує безперервне оновлення додатку з новими функціями та виправленнями, що сприяє оперативності та гнучкості бізнес-процесів. Різні інструменти CI/CD, такі як GitLab CI/CD, GitHub Actions та Jenkins, надають широкий спектр можливостей для автоматизації цих процесів. GitLab CI/CD є вбудованим рішенням у GitLab, що дозволяє легко налаштовувати робочі процеси CI/CD без необхідності використання сторонніх сервісів. GitHub Actions пропонує схожі можливості безпосередньо в рамках GitHub, дозволяючи створювати робочі процеси для автоматизації тестування, збірки та розгортання додатків. Jenkins є відкритим інструментом, який дозволяє гнучко налаштовувати процеси CI/CD з використанням плагінів та скриптів. Інтеграція цих інструментів з хмарними рішеннями, такими як AWS, GCP або Azure, додатково підвищує ефективність та масштабованість процесу розробки. Хмарні платформи можуть надавати послуги для автоматизованого розгортання, моніторингу, управління ресурсами та забезпечення безпеки, що синергічно працює з CI/CD процесами. Така інтеграція гарантує більш плавний та ефективний розвиток веб-додатків, забезпечуючи швидке впровадження нових функцій та високий рівень безпеки та стабільності додатку.

Вибір GitHub Actions для CI/CD у нашому проекті, нарівні з використанням Amazon Web Services (AWS) у якості хмарної платформи, створює сильну синергію, підсилюючи гнучкість та ефективність нашої розробки. AWS, як провідний хмарний сервіс, пропонує широкий спектр інструментів та послуг, що ідеально підходять для сучасних веб-додатків, включаючи рішення для зберігання даних, обчислень, безпеки та багато іншого [33]. Включення AWS у нашу інфраструктуру разом з GitHub Actions надає нам такі переваги:

- Інтеграція з AWS Сервісами: GitHub Actions може бути легко інтегрований з різними AWS сервісами, такими як EC2 для віртуальних серверів, S3 для зберігання, RDS для баз даних тощо. Це дозволяє автоматизувати процеси розгортання та управління інфраструктурою безпосередньо з GitHub.
- Автоматизація розгортання: З GitHub Actions можна автоматизувати розгортання додатків у різних середовищах AWS, що полегшує управління виробничими та тестовими середовищами.
- Масштабованість: AWS надає високу масштабованість, що дозволяє легко адаптувати ресурси до потреб проекту. Використання GitHub Actions для автоматизації цих процесів гарантує плавне масштабування без потреби в ручному втручанні.
- Безпека: AWS пропонує розширені можливості безпеки, які можуть бути інтегровані з CI/CD процесами в GitHub Actions для забезпечення безпеки коду та додатків.
- Ефективність витрат: Використання AWS разом з GitHub Actions може оптимізувати витрати, забезпечуючи ефективне використання ресурсів та платіж за використання.

Це поєднання GitHub Actions та AWS створює потужну інфраструктуру для автоматизації та управління процесами розробки та розгортання, забезпечуючи ефективність, масштабованість та високий рівень безпеки нашого веб-додатку [34].

### 2.3 Планування архітектури додатку.

Важливою складовою архітектури є інтеграція фронтенду та бекенду, реалізована за допомогою React та Django відповідно. Весь додаток задеплойований на AWS EC2, що забезпечує високу доступність та масштабованість. Для зберігання даних використовується база даних RDS PostgreSQL, що гарантує ефективне управління даними і високу продуктивність.

Значною частиною процесу розробки є впровадження CI/CD, яке здійснюється через GitHub. Ключовим елементом є те, що коли код потрапляє до основної гілки (main) у репозиторії GitHub, автоматично запускається процес CI/CD. Цей процес включає автоматичне тестування коду, підтвердження його стабільності та надійності, а після успішного проходження тестів - деплоймент нової версії коду на EC2 інстанси. Це забезпечує безперервне оновлення додатку з мінімальними перервами у роботі, забезпечуючи високу продуктивність та стабільність сервісу.

Таким чином, використання комбінації AWS EC2, RDS PostgreSQL та інтегрованого процесу CI/CD дозволяє створити міцну та ефективну архітектуру для веб-додатку, що забезпечує гнучкість, масштабованість та легкість управління, як зображено на рисунку 2.3.

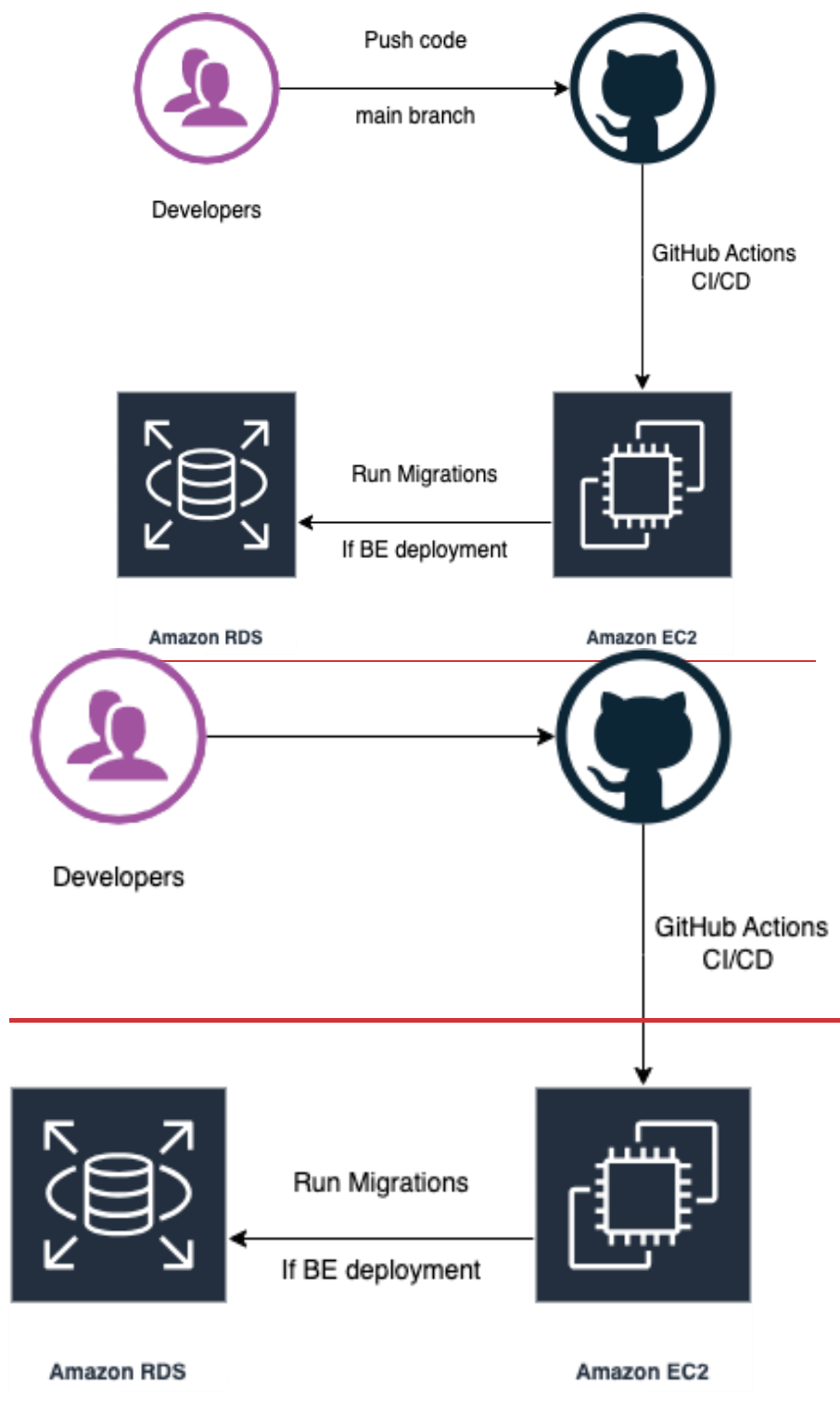


Рисунок 2.3 – Діаграма інфраструктурної архітектури.

Архітектура веб-додатку, яка використовує React для фронтенду, Django у поєднанні з Django ORM для бекенду, PostgreSQL як систему управління базами даних та OpenAI API як зовнішній сервіс, є прикладом сучасного, масштабованого та гнучкого програмного рішення.



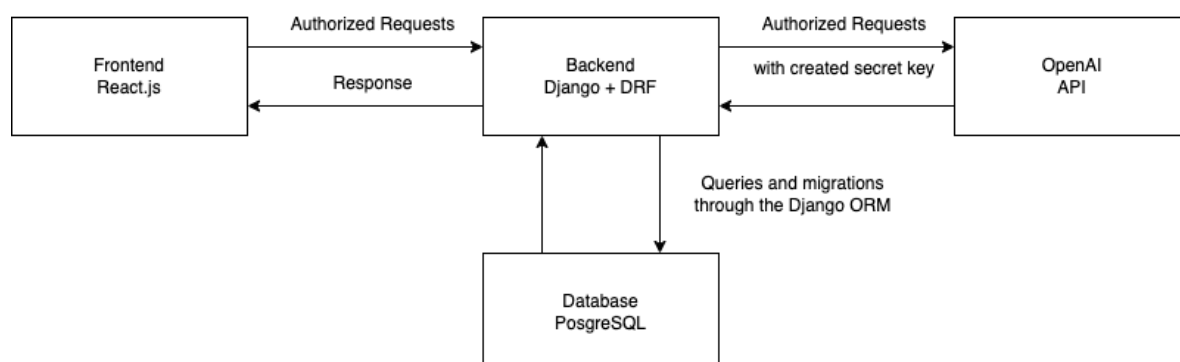
Фронтенд, розроблений з використанням бібліотеки React, забезпечує інтерактивний та високопродуктивний інтерфейс користувача. Він динамічно взаємодіє з бекендом через асинхронні запити HTTP, отримуючи дані у форматі JSON, що дозволяє створювати плавний та реактивний досвід для користувачів.

Бекенд, створений на основі Django, є міцним та гнучким рішенням, що дозволяє швидко розвивати бізнес-логіку з високим рівнем абстракції. Django ORM, потужний інструмент для взаємодії з базою даних PostgreSQL, забезпечує розробникам зручний інтерфейс для виконання запитів та маніпуляцій з даними без прямої роботи з SQL. Це сприяє швидкості розробки та зменшує ймовірність помилок.

PostgreSQL використовується як надійна, масштабована та ефективна система управління базами даних, що підтримує складні запити та гарантує цілісність даних. Це вибір для рішень, де потрібна висока продуктивність та гнучкість у роботі з великими обсягами даних.

OpenAI API інтегрується як зовнішній сервіс, який використовується для розширення функціоналу додатку штучним інтелектом. Воно дає змогу виконувати складні операції з обробки мови та генерації тексту, забезпечуючи користувачам новітні можливості, такі як автоматичне створення коду або генерація вмісту. Інтеграція з OpenAI API відбувається через безпечні HTTP запити, які виконуються бекендом, що дозволяє підтримувати контроль над використанням API та обробкою отриманих даних.

Загалом, така архітектура є ефективною комбінацією надійності, продуктивності та інноваційності, вона втілює в собі передові практики розробки сучасних веб-додатків (рис. 2.4).



## Рисунок 2.4 – Архітектура взаємодії фронтенду, бекенду, бази даних та OpenAI API

Також, одним із ключових елементів створення архітектури проекту, є проектування схеми бази даних, яка зображена на рисунку 2.5.

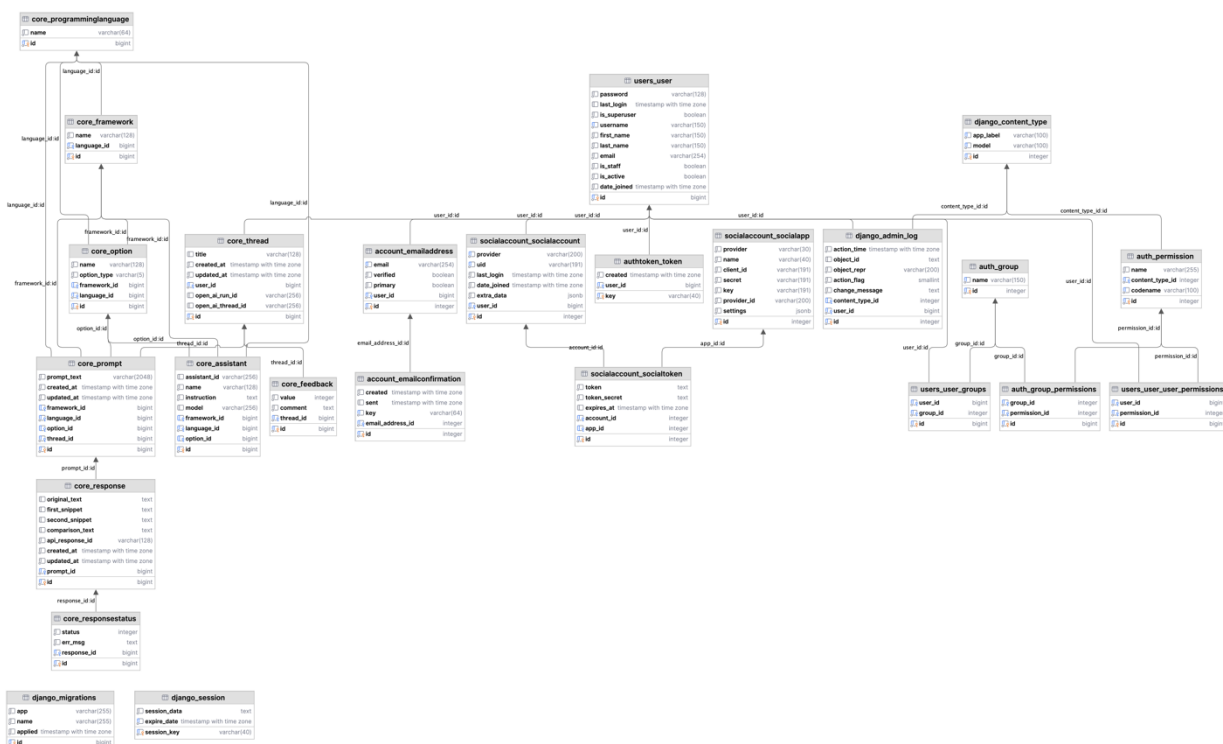


Рис. 2.5 Архітектура схеми бази даних

На основі обраних архітектурних рішень було реалізовано веб-додаток на тему магістерської роботи.

### 2.4 Аналіз методів інтеграції веб-додатку з API OpenAI.

Аналіз методів інтеграції веб-додатку з OpenAI API вимагає зосередження уваги на двох популярних мовах програмування: Python і Node.js, які є широко використовуваними при створенні веб-додатків.

Щоб інтегрувати OpenAI API, перш за все, необхідно створити API-ключ в інтерфейсі розробника OpenAI. Цей ключ є унікальним ідентифікатором, який забезпечує авторизований доступ до сервісів OpenAI і має бути збережений у безпечному місці. При інтеграції важливо дотримуватись принципів безпеки,

зокрема, не зберігати ключі у відкритому доступі чи в коді, який може бути доступний стороннім особам.

Після отримання ключа можна використовувати відповідні бібліотеки для Python або Node.js для роботи з API. Python-бібліотека 'openai' та Node.js пакет 'openai-api' надають інтуїтивно зрозумілі методи для виклику різноманітних функцій API, включно із здатністю використовувати різні моделі GPT для генерації тексту.

Функціонал, який надає OpenAI API, є обширним і включає можливості зі створення звичайного чату, де можна взаємодіяти з моделями мови в стилі запит-відповідь, та асистентів, які можуть виконувати більш складні завдання, такі як програмування чи написання текстів (таблиця 2.1) [35].

Таблиця 2.1 Огляд моделей, які надає OpenAI API

Модель	Опис
GPT-4 і GPT-4 Turbo	Набір моделей, які покращують GPT-3.5 і можуть розуміти, а також генерувати природну мову чи код.
GPT-3.5	Набір моделей, які покращують GPT-3 і можуть розуміти, а також генерувати природну мову чи код.
DALL·E	Модель, яка може створювати та редагувати зображення за допомогою природної мови.

Модель	Опис
TTS	Набір моделей, які можуть перетворювати це на природне звукове мовлення.
Whisper	Модель, яка може перетворювати аудіо в текст.
Embeddings	Набір моделей, які можуть перетворювати текст у числову форму.
Moderation	Тонко налаштована модель, яка може визначити, чи може текст бути конфіденційним чи небезпечним.
GPT base	Набір моделей без інструкцій, які можуть розуміти, а також генерувати природну мову чи код.
Moderation	Тонко налаштована модель, яка може визначити, чи може текст бути конфіденційним чи небезпечним.

Продовження таблиці 2.1

Більш детально буде розглянуто асистентів, які не представлені в рамках моделей, проте вони використовуються у веб-додатку розробленому в рамках цієї магістерської роботи.

API асистентів OpenAI надає розробникам змогу створювати асистентів на основі штучного інтелекту у власних додатках. Такі асистенти можуть використовувати набір інструментів для реагування на запити, включаючи інтерпретатор коду для виконання Python, пошук знань для отримання зовнішньої інформації та виклик функцій для більш складних операцій. Вони забезпечують адаптивність у особистості та можливостях через інструкції до моделей OpenAI і можуть управляти складними завданнями, використовуючи постійні потоки та файли в різних форматах. Задумані для гнучкості, API асистентів також дозволяє

розробникам інтегрувати власні інструменти, підвищуючи функціональність ШІ взаємодіях з користувачами. Ця інфраструктура покликана спростити розробку додатків на основі ШІ та активізувати динамічні функції асистентів.

Однак, суттєвою частиною успішної інтеграції є *prompt engineering*, тобто вміння формулювати запит таким чином, щоби максимально точно та ефективно використовувати можливості штучного інтелекту. Коректно сформульовані промпти можуть значно підвищити якість та релевантність відповідей, що отримує користувач. Це мистецтво і наука одночасно, що вимагає глибокого розуміння як можливостей API, так і предметної області, до якої стосується запит. Більш детально структуру для написання правильного *prompt* описано нижче.

*Prompt engineering* – це технічне та творче мистецтво формулювання запитів до моделей штучного інтелекту, з метою отримання максимально точних та релевантних відповідей. У технічній частині *prompt engineering* важливі архітектура моделі, тренувальні дані та токенізація, параметри моделі, температура та Top-k семплінг, функції втрат та градієнти. Творчий процес формування промптів передбачає врахування таких ключових елементів, як інструкція, контекст, вхідні дані та індикатор виводу. Наприклад, рольова гра дозволяє отримати відповіді, притаманні певному образу чи професії, а ітеративне уточнення та зворотній зв'язок сприяють підвищенню точності та релевантності відповідей.

Застосування просунутих технік, таких як *zero-shot prompting*, яке перевіряє здатність моделі генералізувати завдання, що не були присутні під час її тренування, та *few-shot prompting* або *in-context learning*, де модель отримує декілька прикладів для кращого розуміння задачі, є критично важливими для підвищення ефективності промптів. *Chain-of-Thought (CoT)* є ще однією просунутою технікою, яка включає в себе керівництво моделі через серію логічних кроків, що дозволяє досягти кращого розуміння мови та точніших результатів, подібно до того, як ви б пояснили комусь складну математичну задачу крок за кроком.

У випадку з асистентами, підходи розширюються та видозмінюються. При створенні асистентів в рамках веб-додатку, у загальний *prompt* включається інформація з посиланням на різні документи. У цих документах можуть міститися

посилання або інформація, яка буде використовуватись, як джерело знань, або ж шаблон відповіді.

## Висновки до розділу 2

1. Детально оглянуто доступні інструменти та платформи для розробки веб-додатків. Досліджено їх переваги та недоліки, так само як тенденції на ринку. На основі цього було сформовано базу знань і попередні вимоги до технологій, які будуть використовуватись при розробці веб-додатку.

2. Визначено стек для побудови веб-додатку на основі ряду факторів, найосновнішим з яких є легка інтеграція з OpenAI API. На основі цього було обрано наступні технології. Для бекенд розробки: Python, Django, Django REST framework, dj-rest-auth, openai. Для фронтенд розробки: React.js та MaterialUI. Для інфраструктурної реалізації, було обрано хмарне середовище Amazon Web Services, а саме: EC2 та RDS.

3. Досліджено підходи до побудови різних архітектур, на основі обраного стеку технологій. Розроблено архітектуру додатку, починаючи від схеми бази даних і закінчуючи інфраструктурною архітектурою.

4. Проаналізовано можливі підходи інтеграції з OpenAI API. Досліджено що таке prompt engineering, та як це можна імплементувати в систему для тренування моделей асистентів. Проаналізовано Python бібліотеку openai та визначено ряд функціоналу для подальшого використання при розробці серверної логіки веб-додатку.

## РОЗДІЛ 3. РЕЗУЛЬТАТИ ВИРІШЕННЯ ЗАДАЧІ

### 3.1 Демонстрація роботи веб-додатку

Цей розділ магістерської роботи присвячено детальному огляду веб-додатку, розробленого для генерації коду на основі текстового запиту з використанням API OpenAI та Google. Основна мета додатку — надання користувачам ефективного інструменту для швидкого та точного формулювання запитів коду, а також збір зворотного зв'язку для подальших поліпшень.

Першим кроком у використанні веб-додатку є процес авторизації. Сторінка логуювання (рис. 3.1) має зрозумілий та безпечний інтерфейс, забезпечуючи надійний захист даних користувачів. Простота і інтуїтивність процесу реєстрації та входу в систему забезпечує користувачам легкий доступ до основного функціоналу додатку.

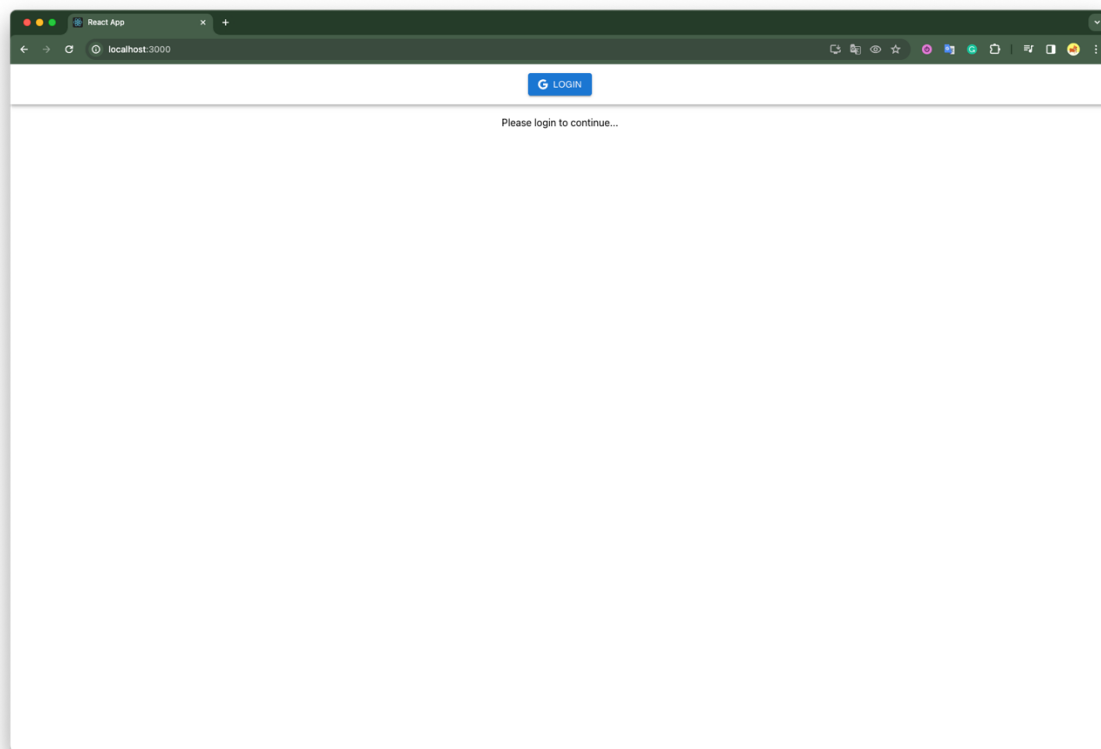


Рисунок 3.1 – Сторінка авторизації у веб-додаток

Після того, як користувач натисне кнопку Login відбувається авторизація за допомогою GoogleAPI, таким чином, користувачу не потрібно створювати окремий акаунт у системі (рис 3.2).

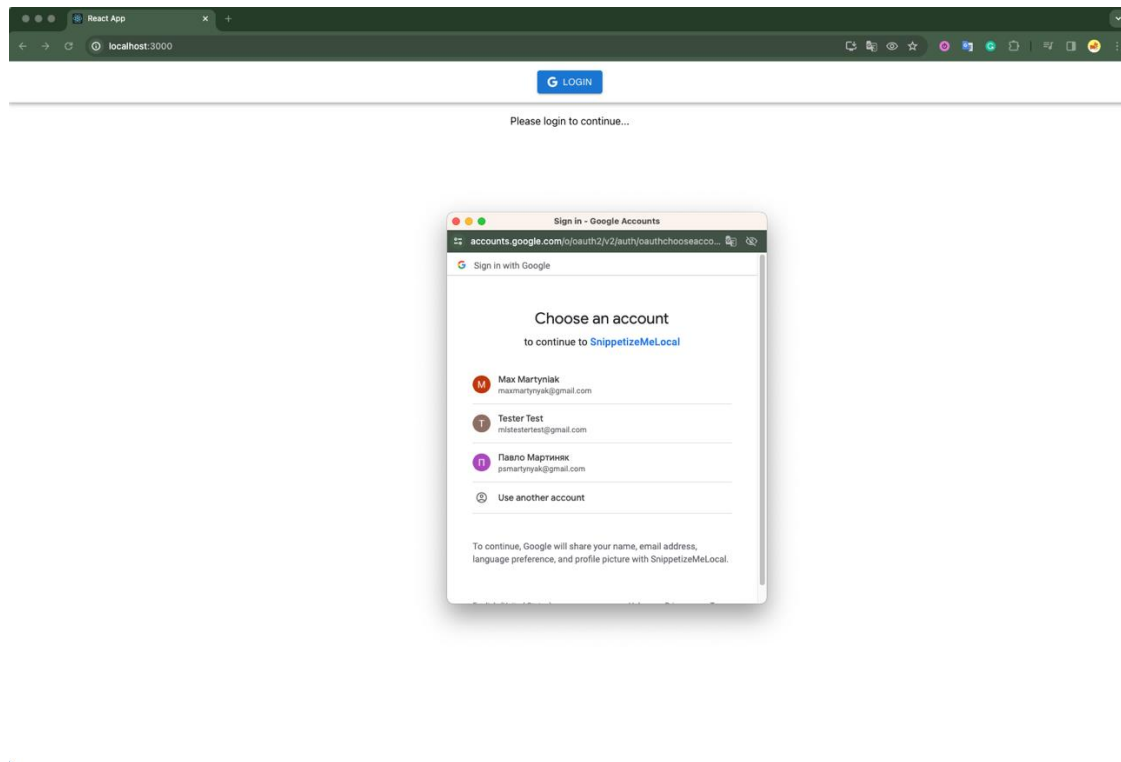


Рисунок 3.2 – Вибір акаунту для авторизації у веб-додаток

Це було реалізовано за допомогою ряду бібліотек, які інтегруються з Django, а саме: `dj_rest_auth` та `allauth`, яка використовує провайдер Google. Імплементация відбувається в файлі налаштувань додатку (рис. 3.3) і за рахунок імплементации додаткового ендпоінта (рис. 3.4).



```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.google',
    'dj_rest_auth',
    'dj_rest_auth.registration',
    'corsheaders',
    # Added apps
    'users',
    'core',
]

```

Рисунок 3.3 – Пакети для реалізації авторизації у додаток за допомогою Google (файл settings.py)

```

users/views.py × communicators.py settings.py controllers.py core/views.py
from allauth.socialaccount.providers.google.views import GoogleOAuth2Adapter
from allauth.socialaccount.providers.oauth2.client import OAuth2Client
from dj_rest_auth.registration.views import SocialLoginView

2 usages ± mmartyniak *
class GoogleLogin(SocialLoginView):
    adapter_class = GoogleOAuth2Adapter
    callback_url = 'http://localhost:3000'
    client_class = OAuth2Client

```

Рисунок 3.4 – Клас, який містить логіку ендпоінта, для реалізації авторизації (файл users/views.py)

Після успішного логування користувач потрапляє на основну сторінку веб-додатку (рис. 3.5).

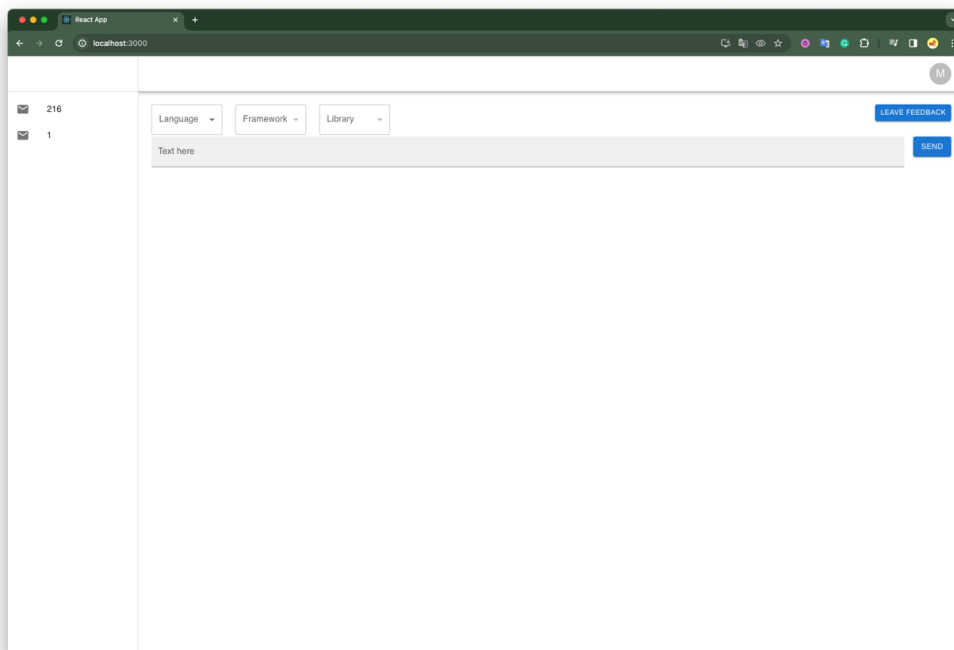


Рисунок 3.5 – Основна сторінка веб-додатку

На основній сторінці представлено список існуючих тредів користувача, які користувач може відкривати в будь-який момент часу, коли це буде йому необхідно (рис. 3.6).

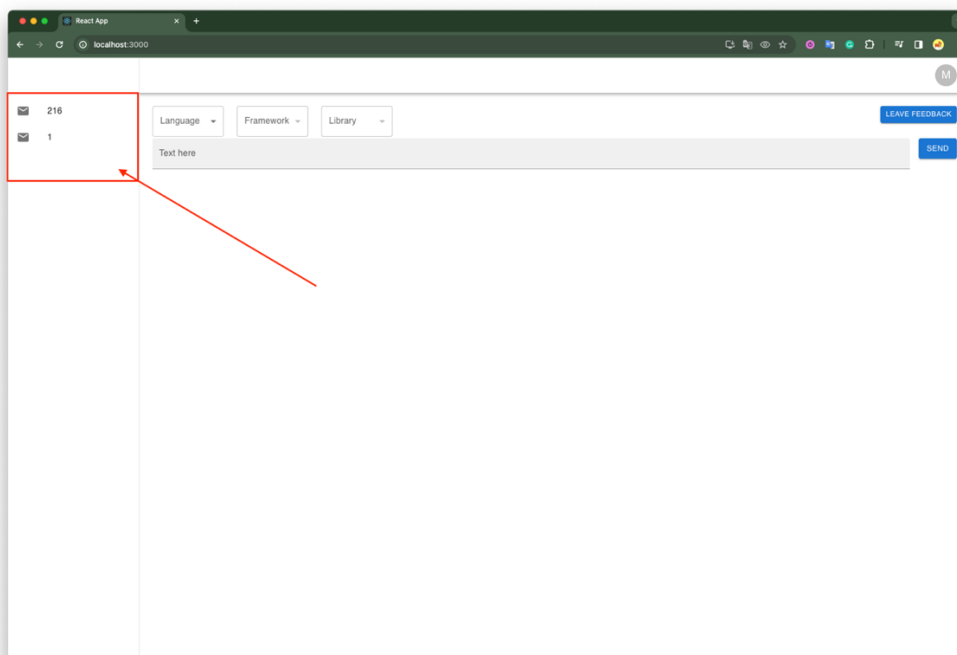


Рисунок 3.6 – Список тредів користувача

Після того як користувач натисне на ідентифікатор треду, його буде загружено вміст обраного треду (рис. 3.7).

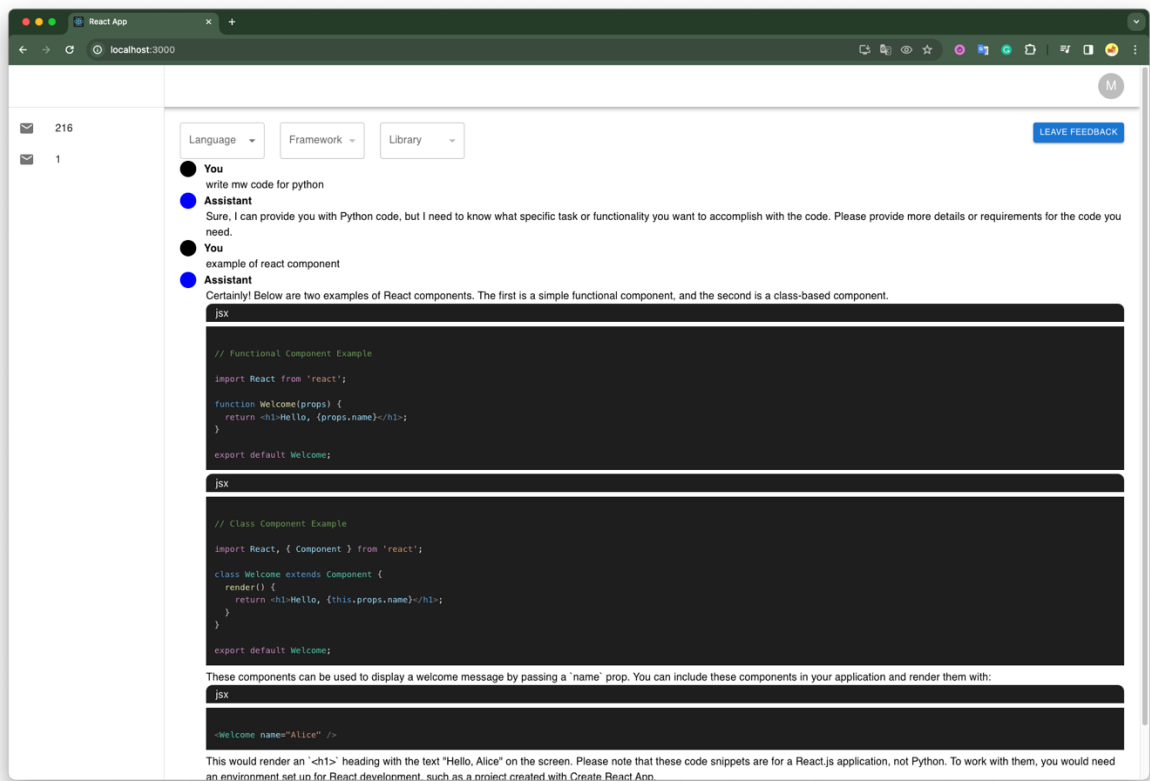


Рисунок 3.7 – Обраний користувачем тред

З серверної сторони, для реалізації цього функціоналу було створено дві кінцеві точки, зображені на рисунку 3.8.

```
path('thread-messages/', ListThreadMessagesView.as_view(),
     name='thread-messages-view'),
path('threads-list/', ThreadsListView.as_view(), name='threads-list-view'),
```

Рисунок 3.8 – Дві кінцеві точки для отримання даних по тредам

Ендпоінт для отримання всіх тредів користувача 'prompt/threads-list/', реалізований за допомогою класу ThreadsListView, його імплементація зображена на рисунку 3.9.

```

2 usages  ± mmartyniak
121 class ThreadsListView(ListAPIView):
122     permission_classes = [IsAuthenticated]
123     serializer_class = ThreadSerializer
124
125     ± mmartyniak
126     def get_queryset(self):
127         user = self.request.user
128         return Thread.objects.filter(user=user).order_by('-updated_at')

```

Рисунок 3.9 – Імплементация класу ThreadsListView

Для отримання всіх повідомлень з треду, який обрав користувач використовується кінцева точка 'prompt/thread-messages/', яка була реалізована за допомогою класу ListThreadMessagesView, проте на відміну від попереднього ендпоінту, цей вимагає ідентифікатор треду, який фронтенд надає як квері параметр (рис. 3.10).

```

2 usages  ± mmartyniak
105 class ListThreadMessagesView(APIView):
106     ± mmartyniak
107     def get(self, request, format=None):
108         thread_id = request.query_params.get('thread_id')
109         thread = Thread.objects.get(id=thread_id)
110         thread_msgs = thread.get_msgs(client)
111
112         # Serialize the data
113         serializer = ThreadMessageSerializer()
114         processed_data = serializer.parse_data(thread_msgs)
115         serializer = ThreadMessageSerializer(data=processed_data, many=True)
116         if serializer.is_valid():
117             return Response(serializer.data)
118         else:
119             return Response(serializer.errors, status=400)

```

Рисунок 3.10 – Імплементация класу ListThreadMessagesView

В класі, зображеному на рисунку 3.10, можна побачити те що тред, який був відфільтрований по ідентифікатору, згодом викликає метод `get_msgs(client)`, який реалізовано в класі на рисунку 3.11. Саме в цьому методі ми надсилаємо запит до клієнта OpenAI API, та витягуємо всі повідомлення з треду.

```

17 usages  ± mmartyniak *
class Thread(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=128, null=True, blank=True)
    language = models.ForeignKey(ProgrammingLanguage, on_delete=models.CASCADE)
    framework = models.ForeignKey(
        Framework, null=True, blank=True, on_delete=models.CASCADE
    )
    option = models.ForeignKey(
        Option, null=True, blank=True, on_delete=models.CASCADE
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    open_ai_thread_id = models.CharField(max_length=256, null=True, blank=True)
    open_ai_run_id = models.CharField(max_length=256, null=True, blank=True)

1 usage (1 dynamic)  ± mmartyniak
def get_status(self, client: OpenAI):
    run = client.beta.threads.runs.retrieve(
        thread_id=self.open_ai_thread_id,
        run_id=self.open_ai_run_id,
    )
    return run.status

1 usage (1 dynamic)  ± mmartyniak
def get_msgs(self, client: OpenAI):
    messages = client.beta.threads.messages.list(
        thread_id=self.open_ai_thread_id
    )
    return messages

```

Рисунок 3.11 – Модель Thread з імплементацією методу `get_msgs`

Після того як ми витягли список повідомлень, нам необхідно сформувати об'єкт, який ми будемо повертати, як відповідь фронтенду. Для цього у класі серіалізері моделі Thread, було імплементовано метод `parse_data`, який витягує дані зі списку повідомлень і формує їх у необхідну нам структуру (рис. 3.12).

```

3 usages  ± mmartyniak
class ThreadMessageSerializer(Serializer):
    role = CharField()
    content = CharField()

1 usage  ± mmartyniak
@staticmethod
def parse_data(thread_messages) -> list[dict]:
    parsed_msgs = []
    for msg in thread_messages.data:
        parsed_msgs.append({
            "role": msg.role,
            "content": msg.content[0].text.value
        })
    parsed_msgs.reverse()
    return parsed_msgs

```

Рисунок 3.12 – Серіалайзер для обробки повідомлень треду, з імплементацією методу для розпаршування даних

На основній сторінці також розташовано набір дропдаунів для вибору відповідних параметрів та опцій (рис. 3.14). Ці інструменти дозволяють користувачам налаштовувати запити відповідно до їхніх потреб, що значно підвищує точність та релевантність генерованого коду та відповідей. Вибір опцій є дуже важливою частиною, через те що на їх основі в подальшому буде обраний відповідний асистент, який було створено спеціально під ту чи іншу опцію.

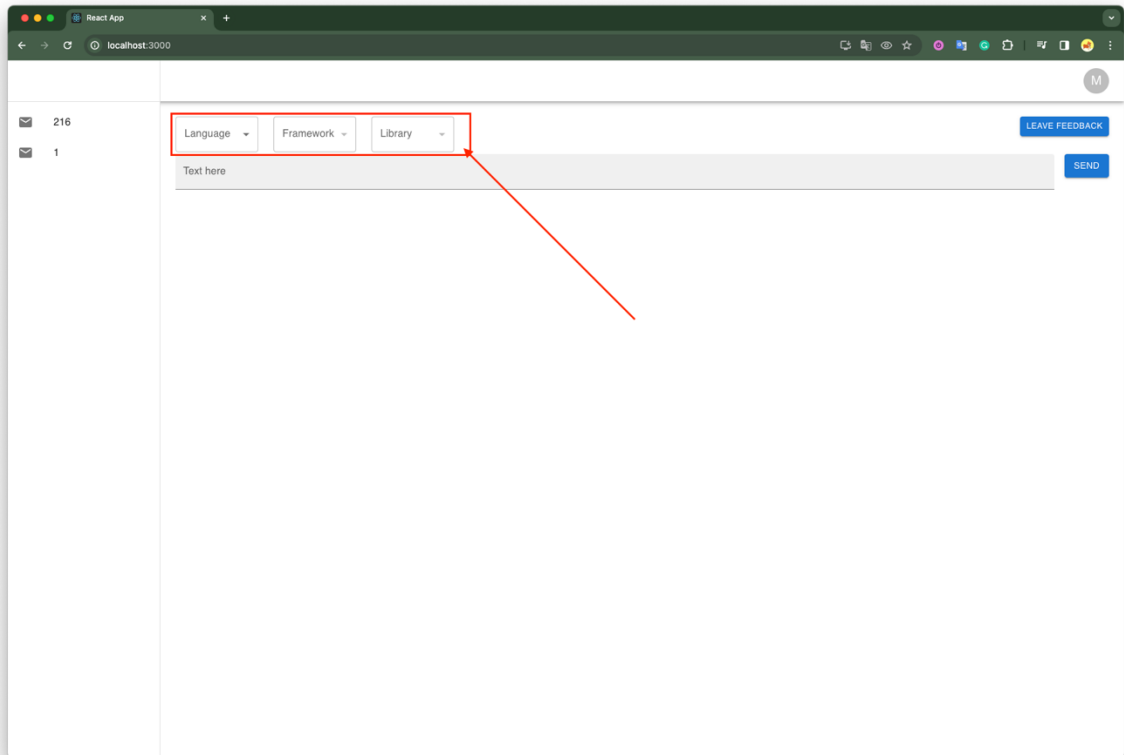


Рисунок 3.14 – Набір дропдаунів, для визначення асистента

Фронтенд отримує всі доступні опції за допомогою кінцевої точки 'prompt/options/' (рис. 3.15). UI частина, надсилає цей запит, як тільки користувач потрапляє на основну сторінку і зберігає всі ці дані у своїй пам'яті.

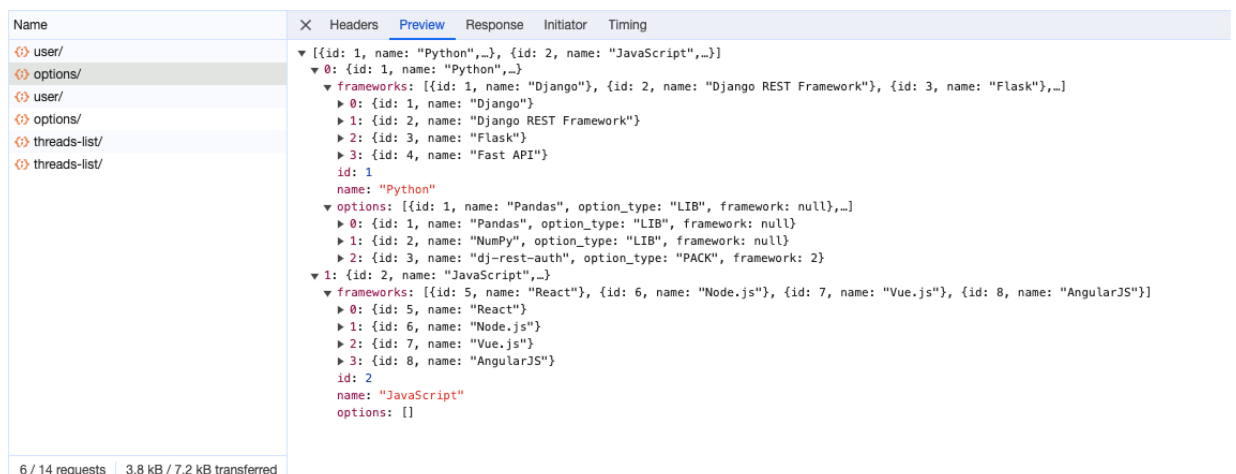


Рисунок 3.15 – Відповідь сервера на запит про всі можливі опції

На стороні сервера, це було реалізовано за допомогою класу (рис. 3.16) та серіалізаторів зображених на рисунку 3.17.

```

2 usages  ± mmartyniak
class PromptOptionsListView(ListAPIView):
    serializer_class = ProgrammingLanguageSerializer
    queryset = ProgrammingLanguage.objects.all()

```

Рисунок 3.16 – Клас з логікою кінцевої точки, для того щоб відати всі опції



```

controllers.py  core/views.py  serializers.py ×  analyze_feedbacks.py  assistant.py
from rest_framework.serializers import Serializer, ModelSerializer, CharField
from .models import (ProgrammingLanguage, Framework, Option, Thread)

1 usage  ± mmartyniak
class FrameworkSerializer(ModelSerializer):
    ± mmartyniak
    class Meta:
        model = Framework
        fields = ('id', 'name')

1 usage  ± mmartyniak
class OptionSerializer(ModelSerializer):
    ± mmartyniak
    class Meta:
        model = Option
        fields = ('id', 'name', 'option_type', 'framework')

2 usages  ± mmartyniak
class ProgrammingLanguageSerializer(ModelSerializer):
    frameworks = FrameworkSerializer(source='framework_set', many=True)
    options = OptionSerializer(source='option_set', many=True)

    ± mmartyniak
    class Meta:
        model = ProgrammingLanguage
        fields = ('id', 'name', 'frameworks', 'options')

```

Рисунок 3.17 – Класи серіалайзерів, які формують відповідь

Центральним елементом основної сторінки є текстове поле (рис. 3.18), де користувачі можуть вводити свої запити. Цей інтерфейс є простим для використання, але водночас надзвичайно потужним інструментом, оскільки він взаємодіє безпосередньо з API для генерації коду. Користувачі можуть вводити свої запити природною мовою, і додаток перетворює їх на детальні та оптимізовані інструкції коду.



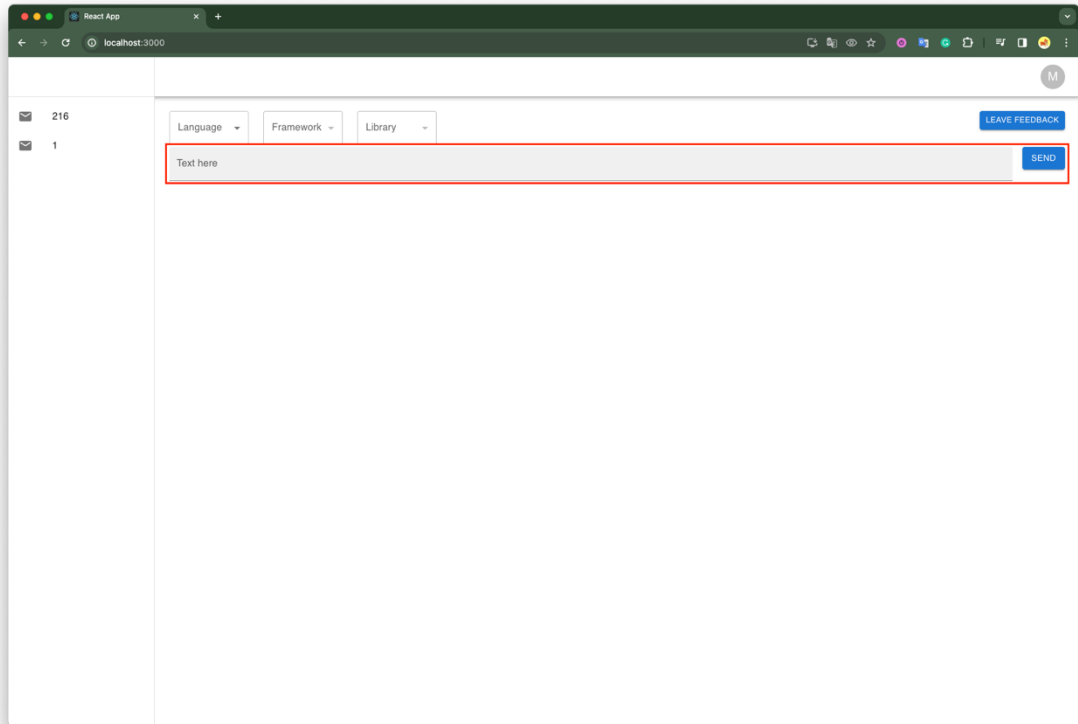


Рисунок 3.18 – Текстове поле, в якому користувачі прописують свої запити

Після введення користувачем текстового запиту та натискання кнопки на відправку, фронтенд надсилає запит до бекенду, де відбувається створення нового треду, для визначеного асистента на основі обраних опцій у дропдаунах. Це відбувається за допомогою ендпоінту 'prompt/post-prompt/' (рис. 3.19).

```

2 usages  ± mmartyniak
class PostPromptView(APIView):
    permission_classes = [IsAuthenticated]

    ± mmartyniak
    def post(self, request, format=None):
        # Extract input parameters from request.data
        thread_id = request.data.get('thread_id')
        programming_language_id = request.data.get('programming_language_id')
        framework_id = request.data.get('framework_id')
        option_id = request.data.get('option_id')
        prompt = request.data.get('prompt')

        # Extract query parameter
        res_type = request.query_params.get('res_type')
        thread = self.invoke_thread(thread_id)
        run = self.invoke_run(
            thread, prompt, programming_language_id, framework_id,
            option_id
        )
        # Return response
        return Response({
            "message": f"Your prompt has status '{run.status}'. "
                f"It will be processed as soon as possible.",
            "thread_id": thread.id
        })

1 usage  ± mmartyniak
def invoke_thread(self, thread_id: Optional[int]) -> Thread: ...

1 usage  ± mmartyniak
@staticmethod
def invoke_run(
    thread: Thread,
    prompt: str,
    language_id: int,
    framework_id: Optional[int],
    option_id: Optional[int]
): ...

```

Рисунок 3.19 – Клас PostPromptView, який реалізує логіку створення або оновлення треду новими повідомленнями користувача

На рисунку 3.19, можна побачити, що цей запит можуть робити лише авторизовані користувачі. Окрім головного методу post, у класі реалізовано два додаткові методи invoke\_thread та invoke\_run, які зображено на рисунку 3.20.

```

1 usage ± mmartyniak
49 def invoke_thread(self, thread_id: Optional[int]) -> Thread:
50     if thread_id is None:
51         open_ai_thread = client.beta.threads.create()
52         thread = Thread(
53             user=self.request.user,
54             open_ai_thread_id=open_ai_thread.id,
55         )
56         thread.save()
57     else:
58         thread = Thread.objects.get(id=thread_id)
59     return thread
60
1 usage ± mmartyniak
61 @staticmethod
62 def invoke_run(
63     thread: Thread,
64     prompt: str,
65     language_id: int,
66     framework_id: Optional[int],
67     option_id: Optional[int]
68 ):
69     message = client.beta.threads.messages.create(
70         thread_id=thread.open_ai_thread_id,
71         role='user',
72         content=prompt,
73     )
74     assistant = Assistant.objects.get(
75         language_id=language_id,
76         framework_id=framework_id,
77         option_id=option_id,
78     )
79     run = client.beta.threads.runs.create(
80         thread_id=thread.open_ai_thread_id,
81         assistant_id=assistant.assistant_id
82     )
83     thread.open_ai_run_id = run.id
84     thread.save()
85     return run
86

```

Рисунок 3.20 – Імплементация додаткових методів класу PostPromptView

З коду наведеного на рисунку 3.20 можна побачити, що метод `invoke_thread`, містить функціонал не лише на створення, але і на отримання вже існуючого треду. Таким чином, користувачі мають змогу розширювати вже існуючі сутності, які були створені раніше. За рахунок цього, система не обмежує користувача у написанні декількох запитів в рамках одного треду.

Відправка запиту користувача відбувається у методі `invoke_run`. У коді цього методу (рис. 3.20) видно, що спершу створюється повідомлення у треді OpenAI API. Після цього ми дістаємо сутність асистента, на основі вибраних користувачем опцій у дропдаунах, які UI передає серверу. З отриманого асистента ми витягуємо його OpenAI ідентифікатор, та запускаємо процес обробки треду цим асистентом.

Слід зауважити, що ця кінцева точка не видає відповідь згенеровану OpenAI API, тому що для обробки запиту потрібно почекати поки API, обробить його. Через це, було створено додаткову кінцеву точку, яка надає інформацію про статус треду. Запит вважається обробленим, коли статус треду має значення `completed`. На основі цього на стороні фронтенду, після надсилання запиту користувача, відбувається ряд запитів, які перевіряють статус треду, поки його значення не буде `completed` (рис. 3.21).

```

2 usages  ± Maksym Martyniak
42  const checkThreadStatus = async (id) : Promise<void> => {
43    setThreadId(id);
44    getThreadStatus(id)
45      .then((res : AxiosResponse<any> ) : void => {
46        if (res.data.status === "in_progress") {
47          setTimeout( callback: () : void => {
48            checkThreadStatus(id);
49          }, ms: 5000);
50        } else {
51          setLoading( value: false);
52          getThreadMessages(id) Promise<AxiosResponse<...>>
53            .then((data : AxiosResponse<any> ) : void => setMesages(data.data)) Promise<void>
54            .catch((err) => console.log(err));
55          setUserRequestMessage( value: "");
56        }
57      })
58    .catch((err) => console.log(err));
59  };

```

Рисунок 3.21 – Імплементація методу `checkThreadStatus`, для перевірки статусу треду

Зі сторони бекенду це реалізовано за допомогою класу `GetThreadStatusView`, зображеного на рисунку 3.22.

```

2 usages  ± mmartyniak
88  class GetThreadStatusView(APIView):
89    permission_classes = [IsAuthenticated]
90
91    ± mmartyniak
92    def get(self, request, format=None):
93      # Extract query parameters
94      thread_id = request.query_params.get('thread_id')
95
96      try:
97        thread = Thread.objects.get(id=thread_id)
98      except Thread.DoesNotExist as exc:
99        return Response({
100          "exc": exc, "user": self.request.user.id,
101          "thread_id": thread_id
102        })
103      return Response({"status": thread.get_status(client)})

```

Рисунок 3.22 – Імплементация класу `GetThreadStatusView`

На рисунку вище видно що отримання статусу відбувається за рахунок методу `Thread.get_status()`. Цей метод можна побачити на рисунку 3.11, з функціонало цього методу видно, що ми відсилаємо запит до OpenAI API з ідентифікатором треду, для отримання його статусу, після чого повертаємо його у відповіді серверу.

Після того, як фронтенд отримує інформацію про те що статус треду має значення `completed`, він відсилає запит на отримання інформації про повідомлення, які містяться в цьому треді. Це було реалізовано в класі `ListThreadMessagesView`, який було детально оглянуто вище (рис. 3.10).

Нарешті, кнопка для відправлення відгуку (рис. 3.23) та модальне вікно (рис. 3.24) в якому можна залишити фідбек є ключовою частиною інтерфейсу. Це дозволяє користувачам з легкістю ділитися своїми думками та пропозиціями щодо поліпшення додатку. Зібрані дані фідбеку використовуються для аналізу користувацького досвіду та вдосконалення функціоналу веб-додатку.

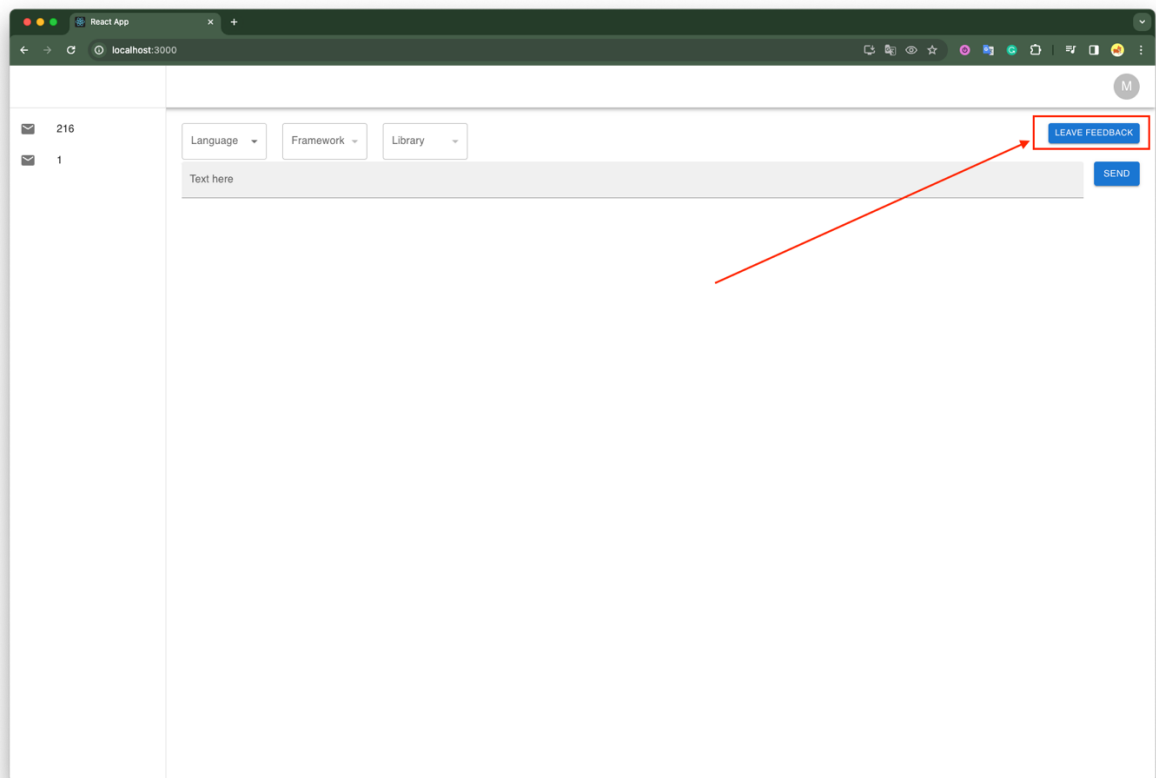


Рисунок 3.23 – Кнопка для відкриття модального вікна, яке дозволяє залишити відгук

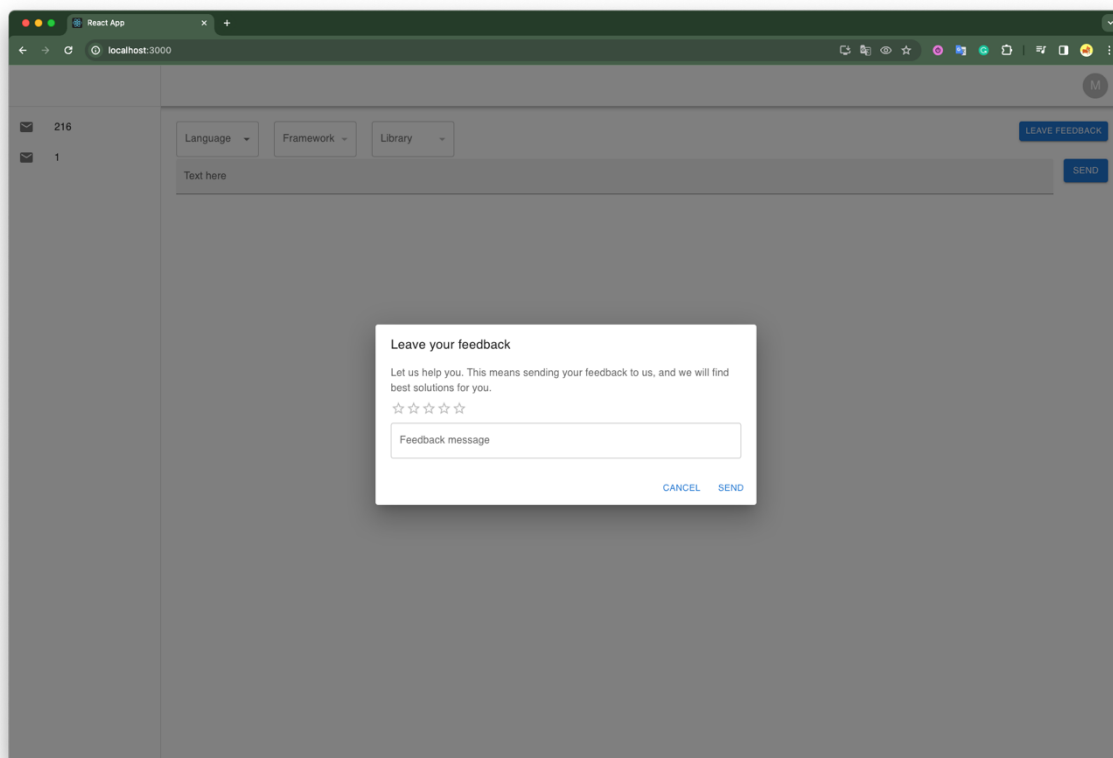


Рисунок 3.24 – Модальне вікно для відгуків

Для того, щоб відправити відгук, було створено клас `CreateFeedback`, який доступний за кінцевою точкою `'prompt/post-feedback/'` (рис. 3.25).

```

2 usages  ± mmartyniak
130 class CreateFeedback(APIView):
131     permission_classes = [IsAuthenticated]
132
133     ± mmartyniak
134     def post(self, request, format=None):
135         # Extract input parameters from request.data
136         thread_id = request.data.get('thread_id')
137         value = request.data.get('value')
138         comment = request.data.get('comment')
139         feedback = Feedback(
140             thread_id=thread_id,
141             value=value,
142             comment=comment,
143         )
144         feedback.save()
145         # Return response
146         return Response({
147             "message": f"Feedback successfully created.",
148             "feedback_id": feedback.id
149         })

```

Рисунок 3.25 – Реалізація класу CreateFeedback

Цей огляд веб-додатку надає всебічне розуміння його ключових компонентів та їх ролей у забезпеченні ефективного, інтуїтивного та корисного досвіду користувача. Включення зворотного зв'язку від користувачів гарантує, що додаток продовжуватиме розвиватися та адаптуватися до змінюваних потреб і вимог користувачів.

### 3.2 Огляд інтеграцій з OpenAI API для створення асистентів та Google API для аналізу відгуків

Основна логіка розробленого в рамках магістерської роботи веб-додатку базується на використанні API від OpenAI, а саме його імплементації в рамках бібліотеки openai написаної на Python. Це дозволяє створювати спеціалізовані асистенти, які можуть надавати користувачам не тільки конкретний код, але й корисні поради та рекомендації, залежно від їхніх запитів та вибраних опцій.

Для інтеграції OpenAI API в додаток було розроблено окремий клас, який відповідає за комунікацію між API та додатком. Цей клас є ключовим елементом для забезпечення взаємодії між веб-додатком та OpenAI, дозволяючи ініціалізувати

клієнта, який в подальшому буде виконувати всю роботу. Цей клас є дочірнім класом, який наслідується від абстрактного класу, що дає можливість створення сінгелтону. Singleton є популярним шаблоном проектування в Python, який використовується для забезпечення створення лише одного екземпляра об'єкта класу. Основна перевага Singleton полягає у тому, що він гарантує глобальний та єдиний доступ до ресурсу чи сервісу, що є критично важливим для сценаріїв, де потрібно уникнути неконтрольованого створення багатьох екземплярів, наприклад, при доступі до бази даних або конфігураційних файлів. Це допомагає уникнути конфліктів і забезпечує централізований контроль над спільними ресурсами. Використання Singleton сприяє чистоті коду, оскільки воно зменшує кількість глобальних змінних і забезпечує більш організоване управління ресурсами. Таким чином, в нас буде один екземпляр класу, який буде комунікувати з OpenAI API. Код реалізації класів зображено на рисунку 3.26.



```

1 from openai import OpenAI
2
3 from typing import Optional
4 from abc import ABC, abstractmethod
5
6 from snippetize_me.settings import config
7
8
9 SECRET_KEY = config.get('open-ai-api').get('secret-key')
10
11
12 usage  ± mmartyniak
13 @. class BaseCommunicator(ABC):
14     """
15     The base class for generating a communicator between API and controllers
16     """
17     _instance = None
18
19     ± mmartyniak
20 def __new__(cls, *args, **kwargs):
21     if cls._instance is None:
22         cls._instance = super().__new__(cls)
23         cls._is_initialized = False
24     return cls._instance
25
26     ± mmartyniak
27 def __init__(self, secret_key: Optional[str] = SECRET_KEY):
28     if not self._is_initialized:
29         self.api_client = self._initialize_api_client(secret_key)
30         self._is_initialized = True
31
32     1 usage  ± mmartyniak
33 @abstractmethod
34 def _initialize_api_client(self, secret_key):
35     """Method for api initialization with secret key"""
36     raise NotImplementedError
37
38     8 usages  ± mmartyniak
39 class OpenAICommunicator(BaseCommunicator):
40     ± mmartyniak
41     def _initialize_api_client(self, secret_key: str) -> OpenAI:
42         return OpenAI(api_key=secret_key)
43
44

```

Рисунок 3.26 – Файл з класами для взаємодії з OpenAI API

Як можна побачити на рисунку, базовий клас має атрибут `api_client`. За рахунок цього атрибута, нам відкривається можливість взаємодіяти з OpenAI API, це можна буде побачити на наступних рисунках.

Особливість реалізації полягає у створенні асистентів через панель адміністратора. Під час створення асистента використовується поле інструкцій, для якого було перевизначено дефолтний метод збереження `save` (рис. 3.27). Це дозволяє

адміністратору додатку налаштувати асистентів з конкретними інструкціями, що забезпечує більш цілеспрямоване та ефективне використання OpenAI API в контексті веб-додатку.

```

4 usages  ± mmartyniak *
class Assistant(models.Model):
    assistant_id = models.CharField(max_length=256, blank=True, null=True) # open ai api id
    name = models.CharField(max_length=128)
    language = models.ForeignKey(ProgrammingLanguage, on_delete=models.CASCADE)
    framework = models.ForeignKey(
        Framework, null=True, blank=True, on_delete=models.CASCADE
    )
    option = models.ForeignKey(
        Option, null=True, blank=True, on_delete=models.CASCADE
    )
    instruction = models.TextField()
    model = models.CharField(max_length=256, blank=True, null=True)

± mmartyniak *
def save(self, *args, **kwargs):
    if self._state.adding:
        communicator = OpenAICommunicator()
        client: OpenAI = communicator.api_client
        assistant = client.beta.assistants.create(
            # Customize the name and instructions as needed
            name=f"{self.name} Assistant",
            instructions=self.instruction,
            model='gpt-4-1106-preview',
            tools=[{"type": "code_interpreter"}],
        )
        self.assistant_id = assistant.id

    super().save(*args, **kwargs)

± mmartyniak
def __str__(self):
    return self.name

```

Рисунок 3.27 – Модель асистента, з перевизначенням методу збереження

На рисунку 3.27 можна побачити схему таблиці асистента, в якій визначенні всі необхідні поля та атрибути. Окрім, полів можна побачити метод збереження запису моделі. Цей метод перевизначає дефолтну логіку збереження, а саме, окрім, простого збереження запису зі всіма даними, викликається клієнт OpenAI, за допомогою класу, який було зображено на рисунку 3.27, та створюється новий асистент на основі інструкцій, наданих адміністратором.

На рисунку 3.28 зображено панель адміністратора. З цієї панелі можна створювати, аналізувати, оновлювати та видаляти всі існуючі сутності в базі даних.

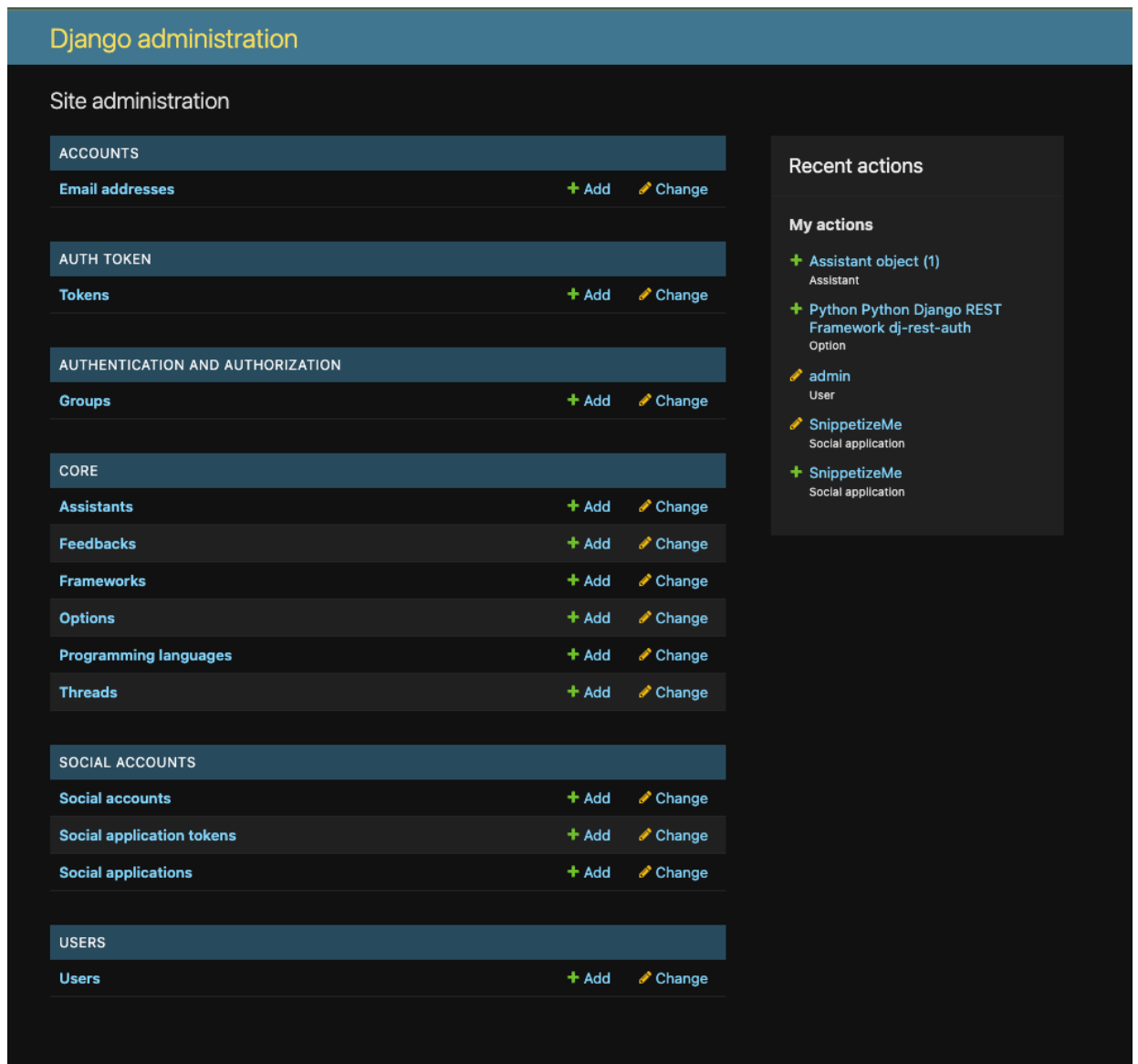


Рисунок 3.28 – Панель адміністратора веб-додатку

Для того щоб створити нового асистента можна просто натиснути кнопку Add навпроти відповідної моделі у списку табличок в рамках таблички "Core". Після цього відбудеться редірект на сторінку асистента (рис. 3.29), де можна вибрати всі необхідні параметри та прописати інструкцію.

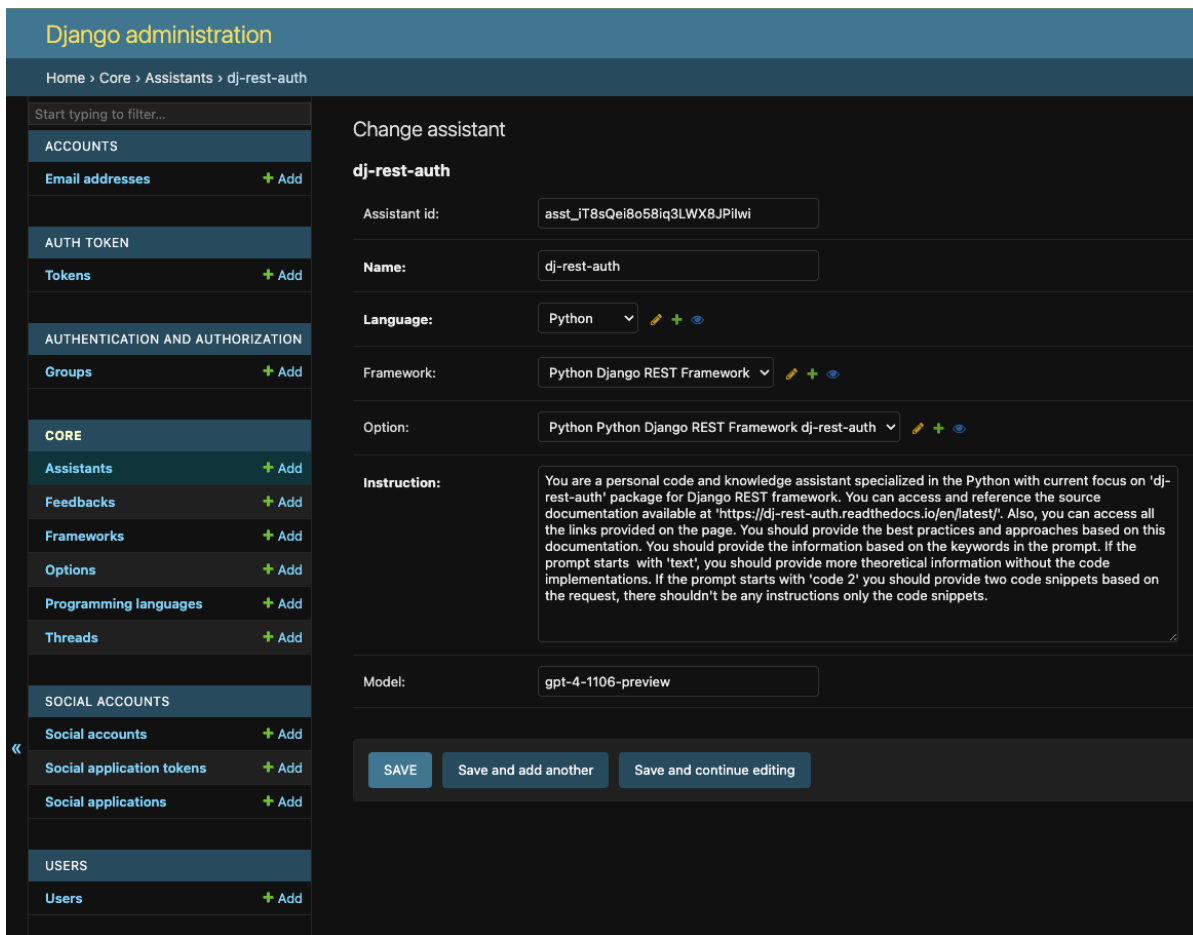


Рисунок 3.29 – Сторінка створення асистента

Після того, як адміністратор натискає кнопку **SAVE** викликається однойменний метод у моделі асистента, яку було зображено вище (рис. 3.29) разом із цим методом. Після цього відбувається декілька кроків, а саме:

- 1) створення асистента в рамках нашої бази даних.
- 2) створення асистента на стороні OpenAI API.
- 3) оновлення існуючого запису асистента, зокрема поля `assistant_id`, в яке буде встановлено значення ідентифікатора асистента на стороні OpenAI.

В майбутньому асистенти будуть використовуватись вже безпосередньо під час виконання запитів для генерації коду та покращених відповідей. Таким чином, інтеграція OpenAI API в проект відкриває широкі можливості для автоматизації генерації коду, пропонуючи гнучке та масштабоване рішення для різних задач програмування та розвитку програмного забезпечення, за рахунок того, що під кожен із можливих опцій створюється окремий детально налаштований асистент.

Збір відгуків від користувачів є критично важливим для розвитку та вдосконалення веб-додатків. Явні (explicit) відгуки надаються користувачами безпосередньо та свідомо: вони включають оцінювання зірочками, коментарі та відповіді на опитування. Цей тип відгуку дає прямий зв'язок від користувачів та допомагає визначити їх задоволеність продуктом та ідентифікувати конкретні аспекти, які потребують уваги.

Неявні (implicit) відгуки, в свою чергу, вимагають інтерпретації дій користувачів, таких як час, проведений на певній сторінці, частота використання додатку або покинуті кошики покупок. Хоча вони не надають безпосереднього зворотного зв'язку, неявні відгуки можуть вказувати на поведінкові тенденції та несвідомі переваги користувачів.

Аналіз відгуків включає в себе як кількісні, так і якісні методи. Кількісні методи можуть включати оцінювання середньої кількості зірочок або аналіз частоти згадувань певних тем у коментарях. Якісний аналіз може включати глибинний аналіз коментарів для виявлення настроїв, виявлення повторюваних проблем чи вивчення пропозицій користувачів.

В рамках веб-додатку, який було розроблено, було додано можливість залишати відгуки на тред. Це було реалізовано за допомогою створення вікна, в якому можна проставити оцінку та залишити коментар. Огляд модального вікна, яке дозволяє залишити відгук було зображено на рисунку 3.24.

У випадку розглядуваної системи, вікно відгуків дозволяє користувачам залишати явні відгуки, які можуть бути детально проаналізовані за допомогою створеної команд Django, реалізованих як заплановані задачі (крони). Ця команда сортує відгуки за заданими критеріями та передає їх до Google Bard API для подальшого аналізу. Нижче наведено рисунок 3.30, на якому зображено алгоритм команди для аналізу відгуків.

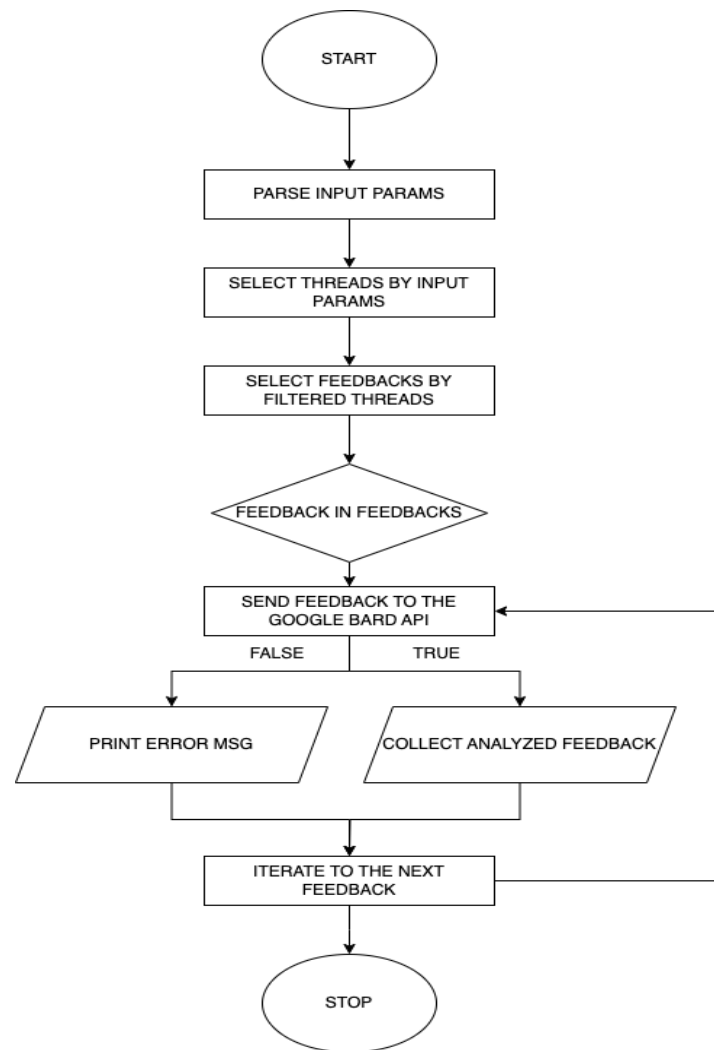


Рис. 3.30 Алгоритм роботи команди для аналізу відгуків

На основі цього алгоритму було зроблено Django команду, код для цієї команди представлено на рисунку 3.31.

```

1 from django.core.management.base import BaseCommand, CommandError
2 from core.models import Thread, Feedback
3 import requests
4
5
6 new *
7 class Command(BaseCommand):
8     help = 'Analyzes feedbacks for specific threads'
9
10 new *
11 def add_arguments(self, parser):
12     parser.add_argument('programming_language_id', type=int)
13     parser.add_argument('framework_id', type=int)
14     parser.add_argument('option_id', type=int)
15
16 new *
17 def handle(self, *args, **options):
18     programming_language_id = options['programming_language_id']
19     framework_id = options['framework_id']
20     option_id = options['option_id']
21
22     threads = Thread.objects.filter(
23         language_id=programming_language_id,
24         framework_id=framework_id,
25         option_id=option_id
26     )
27
28     feedbacks = Feedback.objects.filter(thread__in=threads)
29
30     for feedback in feedbacks:
31         data = {
32             'value': feedback.value,
33             'comment': feedback.comment
34         }
35         response = requests.post(
36             url='https://api.googlebard.com/analyze', json=data
37         )
38         if response.status_code == 200:
39             analysis_result = response.json()
40             print(analysis_result)
41         else:
42             print('error while analyzing feedback:', feedback.id)
43
44     self.stdout.write(self.style.SUCCESS('feedback review finished!'))

```

Рисунок 3.31 – Реалізована Django команда для аналізу відгуків

Завдяки цьому, аналізовані дані надають дані, які допомагають виявити та вирішити конкретні проблеми асистента, тим самим підвищуючи задоволеність користувачів та ефективність додатку.

### 3.3 Можливі шляхи для подальшого вдосконалення веб-додатку

Для подальшого вдосконалення веб-додатку, наразі важливо звернути увагу на його стан у рамках закритого бета-тестування. У цій фазі, доступ до додатку обмежений: користувачі можуть увійти через Google Login лише тоді, коли їхні акаунти були заздалегідь додані до спеціально створеного Google App. Такий підхід дозволяє забезпечити контрольований доступ до додатку, що є критично важливим для збору зворотного зв'язку та виявлення можливих проблем перед його повним запуском. Наступним кроком є публікація додатку, яка потребує проходження верифікації від Google. Цей процес є суттєвим, оскільки вимагає від розробників підтвердження відповідності додатку стандартам і політикам Google, забезпечуючи таким чином якість і безпеку для широкого кола користувачів. Верифікація від Google не тільки збільшить довіру до додатку, але й розширить його потенційну аудиторію, дозволяючи залучити більше користувачів та зібрати більше даних для подальшого аналізу та удосконалення.

У рамках розвитку веб-додатку, важливим досягненням є реалізація критичного функціоналу, що вже було здійснено. Це створює міцний фундамент для подальшого розширення можливостей додатку. Однією з перспективних ідей для розширення функціоналу є введення системи токенів. Ці токени дозволять користувачам здійснювати запити до додатку. У початковій фазі, користувачам буде надано певну кількість токенів, що дасть їм змогу випробувати функціонал і визначити, наскільки додаток відповідає їхнім потребам та чи є вони готові платити за його подальше використання. Цей підхід не тільки стимулює початкове використання додатку, але й забезпечує зворотний зв'язок від реальних користувачів, що є важливим для вдосконалення продукту. Також, використання токенів може сприяти розвитку монетизаційної стратегії додатку, визначаючи баланс між безкоштовним доступом для спроби та платним доступом для розширених можливостей.

Стратегія монетизації для нашого веб-додатку повинна бути детально продуманою, щоб забезпечити сталість та успішність проекту. Одним з ключових елементів є встановлення адекватної ціни на токени, які користувачі використовують



для запитів до асистентів. Ціноутворення на токени має базуватися на цінній політиці використання асистентів, враховуючи вартість ресурсів та забезпечуючи конкурентоспроможність на ринку.

Крім того, планується впровадження опції преміум-підписки, яка надасть користувачам більшу гнучкість та персоналізацію. Підписники зможуть самостійно додавати опції, формуючи асистента під свої індивідуальні потреби та використовувати бібліотеки чи пакети. Ця функція робить додаток значно більш привабливим для професіоналів, які шукають специфічні рішення. Процес налаштування асистента спрощено завдяки вже розробленій інструкції для створення асистента, де користувачу потрібно лише вказати необхідну документацію для бажаного пакету або бібліотеки.

Ці кроки спрямовані на створення стійкого фінансового фундаменту для додатку, одночасно забезпечуючи цінність та задоволення потреб користувачів. Важливо, щоб стратегія монетизації була гнучкою і адаптивною, аби вона могла ефективно реагувати на зміни в ринкових умовах та потребах користувачів.

Критичним аспектом розвитку нашого веб-додатку є створення базових асистентів, які підтримують популярні мови програмування, фреймворки та бібліотеки. Це стане ключовим фактором, який дозволить рядовим користувачам ефективно покривати свої потреби в програмуванні та розробці. На даний момент, в рамках бета-тестування, було створено лише кілька асистентів для демонстрації можливостей і отримання зворотного зв'язку. Проте, для повноцінного запуску та задоволення потреб широкого кола користувачів, необхідно значно розширити цей спектр.

Залучення базових асистентів для найпопулярніших технологій відкриє нові можливості для користувачів, особливо для тих, хто може не мати глибоких знань у специфічних областях. Це також сприятиме більш широкому використанню додатку, оскільки користувачі зможуть швидко отримати допомогу та рекомендації, адаптовані під конкретні мови програмування та технічні стеки.

Розширення бази асистентів підвищить цінність додатку, роблячи його більш привабливим для широкої аудиторії, від початківців до досвідчених розробників.

Таке оновлення дозволить додатку бути більш комплексним рішенням для програмування та розробки, забезпечуючи потужну підтримку у вирішенні різноманітних технічних задач.

Розвиток веб-додатку також вимагає розробки функціоналу, який встановлює ліміти на використання ресурсів користувачами. Це є критичним аспектом, оскільки ефективний контроль за використанням ресурсів безпосередньо впливає на стабільність та продуктивність додатку. Ліміти дозволять забезпечити справедливе використання ресурсів серед усіх користувачів, запобігаючи надмірному навантаженню на систему.

Встановлення лімітів може включати обмеження кількості запитів на одиницю часу, обмеження використання певних функцій або ресурсів, а також встановлення максимального часу виконання окремих операцій. Ці міри допоможуть запобігти перевантаженню серверів та іншої інфраструктури, забезпечуючи більш плавну та стабільну роботу додатку для всіх користувачів.

Крім того, такий підхід може стати частиною монетизаційної стратегії. Наприклад, базові користувачі можуть мати певний безкоштовний ліміт використання, тоді як преміум користувачі можуть отримати розширений доступ або вищі ліміти. Це не тільки сприяє більш ефективному використанню ресурсів, але й стимулює користувачів до оновлення своїх підписок для отримання додаткових можливостей.

Важливо також забезпечити прозорість цих лімітів для користувачів, інформуючи їх про встановлені обмеження та їхнє поточне використання ресурсів. Це допоможе уникнути непорозумінь та негативних вражень від користування додатком.

Розширення інфраструктури веб-додатку є ключовим кроком для підтримки його стабільності та розвитку. Одним із важливих аспектів цього процесу є створення додаткових середовищ на основі Amazon Web Services (AWS) EC2. Це дозволить створити чітке розділення між робочим середовищем, де користувачі взаємодіють із протестованим та стабільним кодом, та розробничим середовищем, призначеним для розробників для впровадження та тестування нового функціоналу. Створення окремих середовищ має ряд переваг:

- Стабільність для Користувачів: Забезпечує, що кінцеві користувачі мають доступ до надійного та стабільного середовища, з мінімальним ризиком помилок або проблем, пов'язаних із оновленнями.
- Безпека Розробки: Дозволяє розробникам експериментувати та тестувати нові функції без ризику вплинути на робоче середовище, зменшуючи ймовірність непередбачуваних помилок.
- Ефективність Оновлень: Спрощує процес розгортання оновлень, дозволяючи розробникам спочатку тестувати зміни в окремому середовищі перед тим, як впровадити їх у робоче середовище.
- Масштабування: AWS EC2 надає можливості гнучкого масштабування, що є важливим для адаптації до зростаючого числа користувачів та збільшення вимог до ресурсів.
- Контроль Версій та Розгортання: Це також допоможе в контролі версій та автоматизації процесу розгортання, забезпечуючи більш гладке та ефективне впровадження оновлень.

Реалізація такого розділення середовищ на AWS EC2 вимагатиме ретельного планування та управління, щоб забезпечити синхронізацію між середовищами та ефективно управління ресурсами. Однак, ці кроки є важливими для підтримки якості та надійності веб-додатку в довгостроковій перспективі.

### **Висновки до розділу 3**

1. Продемонстровано роботу розробленого веб-додатку з серверної та користувацької сторони. Детально розглянуто взаємодію серверних та користувацьких елементів. Оглянуто кодову базу бекенду.

2. Оглянуто реалізацію інтеграції з OpenAI та Google API. А саме: створення та роботу асистентів для генерації коду, команду для аналізу відгуків користувачів. Розглянуто різні типи відгуків, та продемонстровано, як відгуки реалізовані у користувацькій частині.

3. Визначено ряд подальших векторів розвитку продукту. Детально розглянуто важливість розробки монетизаційної стратегії, а також впровадження

відповідного функціоналу у веб-додаток. Визначено ряд елементів, які необхідно реалізувати для надання користувачеві можливостей для персоналізації додатку.

## РОЗДІЛ 4.

### ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

#### 4.1 Аналіз небезпечних і шкідливих виробничих чинників та розробка заходів щодо покращення умов праці

Потенційно шкідливим чинником кабінету особи, яка приймає управлінські рішення, вважається небезпека враження людини електричним струмом. Важливим, але менш ймовірним чинником являється пожежна небезпека під час аварійної ситуації. Хімічні та біологічні джерела практично не мають впливу.

Перелік небезпечних та шкідливих виробничих чинників наведено у таблиці 4.1.

Таблиця 4.1 Небезпечні та шкідливі виробничі чинники

Фізичні	Електробезпека, пожежа, шум, мікроклімат
Хімічні	Відсутні
Біологічні	Відсутні
Психофізіологічні	Відсутні

В приміщенні кабінету особи, яка приймає управлінські рішення, присутні небезпечні чинники, та за умов дотримання заходів безпеки, вони не є критичним.

#### 4.2 Розробка логіко-імітаційної моделі виникнення травм і аварій

Методикою оцінки рівня безпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об'єктивного критерію рівня безпеки для конкретного об'єкта. Таким показником вибрана ймовірність виникнення аварії, травми залежно від явища, що досліджується.

Для побудови логіко-імітаційної моделі процесу, формування і виникнення аварії та травми в процесі створення мікрокліматичних умов у приміщенні оцінюють відповідні небезпечні події. Кожній із них присвоїмо ймовірність виникнення у таблиці 4.2.

Таблиця 4.2 Таблиця з ймовірностями виникнення небезпечних подій

Шифр	Назва події	Ймовірність
P <sub>1</sub>	Відсутність захисного заземлення	0,02
P <sub>2</sub>	Пошкодження захисного заземлення	0,04
P <sub>3</sub>	Спрацювання складових захисту	0,1
P <sub>4</sub>	Неправильна експлуатація захисту	0,02
P <sub>5</sub>	Відсутність профілактичних заходів	0,2
P <sub>6</sub>	Відсутність захисного щита	0,12
P <sub>7</sub>	Недотримання правил вибору взуття	0,15
P <sub>8</sub>	Незнання правил техніки безпеки	0,1
P <sub>9</sub>	Відсутність засобів індивідуального захисту	0,2
P <sub>10</sub>	Легковажність	0,08

На основі наведених подій будемо матрицю логічних взаємозв'язків між окремими пунктами, графічна інтерпретація якої зображено на рис. 5.1.

Розрахуємо ймовірності виникнення подій, що формують логіко-імітаційну модель процесів створення мікрокліматичних умов. Розглянемо травмонебезпечну ситуацію, що виникає за умови роботи працівників із електронебезпекою.

Підставивши дані ймовірностей базових подій у формулу, отримаємо ймовірність події 13:  $P_{13} = 0,2 + 0,4 - 0,2 \cdot 0,4 = 0,0592$

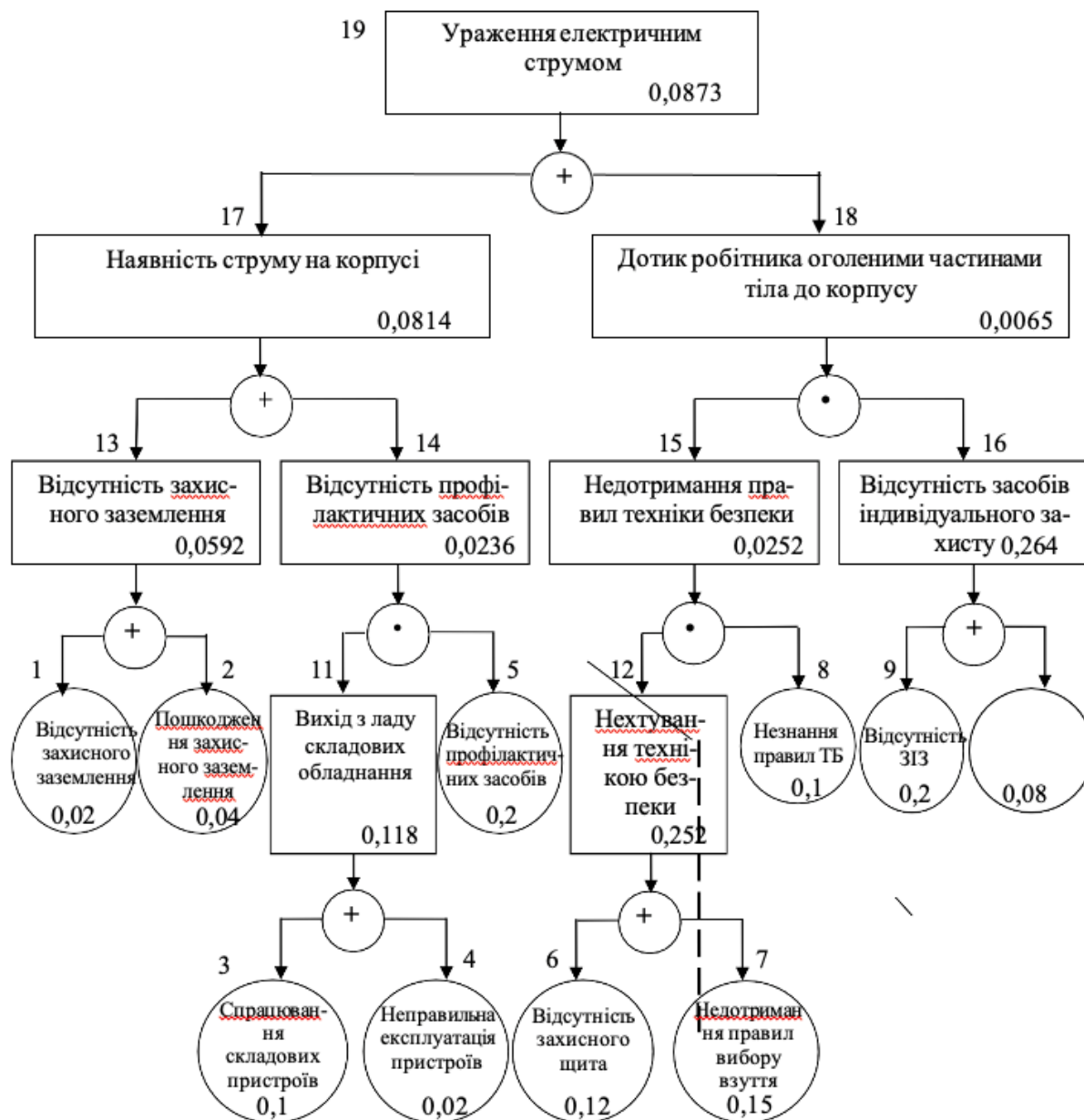


Рисунок 5.1 – Матриця логічних взаємозв'язків між окремими подіями травмонебезпечної ситуації

Аналогічно визначаємо ймовірність інших подій:

$$P_{11} = P_4 + P_5 - P_4P_5 = 0,3 + 0,4 - 0,3 \cdot 0,4 = 0,118.$$

$$P_{12} = P_6 + P_7 - P_6P_7 = 0,3 + 0,5 - 0,3 \cdot 0,5 = 0,252.$$

$$P_{16} = P_9 + P_{10} - P_9P_{10} = 0,2 + 0,15 - 0,2 \cdot 0,15 = 0,264.$$

$$P_{14} = P_{11} \cdot P_5 = 0,118 \cdot 0,2 = 0,0236.$$

$$P_{15} = P_{12} \cdot P_8 = 0,252 \cdot 0,1 = 0,0252.$$

$$P_{17} = P_{13} + P_{14} - P_{13} \cdot P_{14} = 0,592 + 0,0236 - 0,0592 \cdot 0,0236 = 0,0814.$$

$$P_{18} = P_{15} \cdot P_{16} = 0,264 \cdot 0,0252 = 0,0065.$$

$$P_{19} = P_{17} + P_{18} - P_{17} \cdot P_{18} = 0,0065 + 0,0814 - 0,0065 \cdot 0,0814 = 0,0873.$$

Таким чином, ймовірність перекидання машини та наслідкового виникнення травми працівника є досить мала і становить –  $P_{19} = 0,0873$ .

### 4.3 Розробка заходів щодо безпеки у надзвичайних ситуаціях

Науково-технічний прогрес радикально змінив світ, породивши нові загрози для цивілізації. У житті сучасної людини все більше місце займають турботи, пов'язані з подоланням різних кризових явищ, що виникають в процесі розвитку земної цивілізації. В Україні, як і в усьому світі, в останні роки спостерігається зростання числа військових дій, катастроф природного та техногенного характеру. Це обумовлено, перш за все, прогресуючої урбанізацією територій, збільшенням щільності населення Землі, і, як наслідок, збільшенням антропогенного навантаження на навколишнє середовище. Захист природних систем і населення від надзвичайних ситуацій різного характеру сформувалася в останні роки як нагальна і об'єктивна потреба суспільства і держави.

Заходи щодо захисту цивільного населення плануються проводяться по населених пунктах де розміщені підприємства і охоплюють населення навколишніх сіл. Водночас характер та зміст захисних засобів встановлюються від ступеня загрози, місцевих умов з урахуванням важливості виробництва для безпеки населення і інших економічних і соціальних чинників.

Основні заходи щодо захисту населення плануються та здійснюються завчасно і мають випереджувальний характер, це стосується насамперед підготовки, підтримання у постійній готовності індивідуальних та колективних засобів захисту, їх накопичення, а також підготовки до проведення евакуації населення із зон підвищеного ризику.

Також раз в три роки проводяться навчання по підготовці близьких до військових дій, що в разі небезпеки могло би не дістати людину зненацька. Керівництво докладає



максимум зусиль, щоб працівники підприємств були хоча би мінімально захищені в разі будь-якої небезпеки пов'язаної з тими чи іншими обставинами.

#### **Висновки до розділу 4**

1. Проаналізовано небезпечні і шкідливі виробничі чинники, які можуть повпливати на робочий процес. Також, розроблено заходи для покращення умов праці.

2. Проведено аналіз методики оцінка рівня небезпеки робочих місць. Розроблено логіко-імітаційну модель виникнення травм і аварій. Проведено розрахунки ймовірностей виникнення цих подій.

3. Досліджено впливу урбанізації на підвищення зростання катастроф природного та технічного характеру. Розроблено ряд заходів щодо безпеки у надзвичайних ситуаціях.

## РОЗДІЛ 5.

### ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ РОЗРОБКИ ВЕБ-ДОДАТКУ

#### 5.1 Методи оцінки ефективності веб-додатку

Визначення ефективності розробленого в рамках магістерської роботи веб-додатку є фундаментальною частиною оцінки його загальної цінності та придатності для кінцевих користувачів. Оцінювання ефективності не обмежується лише одним аспектом, а включає в себе цілий спектр критеріїв, від технічної продуктивності до користувацького досвіду.

Слід розпочати з технічної ефективності, яка зосереджується на таких аспектах, як швидкість роботи додатку, час відгуку системи, точність генерації коду, та загальна надійність. Це оцінюється через різноманітні тестування, такі як тестування навантаження та функціональне тестування, а також через аналіз продуктивності.

Користувацька ефективність, з іншого боку, вимірюється через аналіз UX/UI, збір відгуків користувачів, та дослідження поведінки користувачів на сайті. Важливо зрозуміти, наскільки інтуїтивно зрозумілим і зручним є додаток для тих, хто ним користується.

Економічна ефективність також є ключовою, адже вона включає в себе аналіз вартості розробки та експлуатації додатку в порівнянні з отриманими перевагами. Це допомагає оцінити, чи виправдовують досягнуті вигоди вкладені ресурси. Для цього потрібно зробити огляд тарифних планів сторонніх API, які використовуються у додатку, зокрема OpenAI API та Google API.

Масштабованість та адаптивність додатку також мають важливе значення. Вони визначають, наскільки легко можна адаптувати додаток до зростаючих потреб або змін у технічних вимогах без значного збільшення витрат. Масштабованість та адаптивність розробленого веб-додатку набувають особливої ваги в контексті його розгортання на хмарних сервісах, зокрема на платформі Amazon Web Services (AWS). Використання AWS як платформи для хостингу додатку надає нам значних переваг у гнучкості та ефективності. AWS надає широкі можливості для масштабування та адаптування сервісів згідно з потребами проекту. Наприклад, з сервісами як Amazon

EC2 для обчислювальних потужностей та Amazon RDS для баз даних, веб-додаток має змогу легко збільшувати або зменшувати ресурси відповідно до актуального навантаження. Це означає, що при збільшенні кількості користувачів чи даних, додаток може автоматично адаптуватися, надаючи додаткові ресурси без потреби в ручному перенастроюванні або додаткових капіталовкладеннях. AWS також пропонує гнучкі цінові політики, які дозволяють розробникам оптимізувати витрати. За допомогою моделі оплати "за використання" можна платити лише за ті ресурси, які дійсно використовуються. Це особливо ефективно для стартапів та проектів зі змінним навантаженням, оскільки воно забезпечує високу економічну ефективність та знижує ризики, пов'язані з необхідністю вгадування майбутніх потреб у ресурсах. Щодо цінових політик AWS EC2 та AWS RDS, вони забезпечують широкий спектр опцій, від базових інстансів для невеликих проектів до розширених конфігурацій для великих корпоративних додатків. Кожен з цих сервісів пропонує різні типи інстансів, які оптимізовані під конкретні завдання та вимоги до продуктивності. Такий підхід не лише гарантує технічну адаптивність додатку, але й дозволяє тонко налаштовувати витрати відповідно до бюджету проекту.

Цінова політика та тарифні плани для веб-додатку мають велике значення для його комерційного успіху. Тут важливо врахувати вартість розробки та підтримки, яка залежить від обраних технологій, таких як React, Python, Django, Django REST framework, та OpenAI API. Вибір технологій визначає не тільки вартість, але й функціональні можливості, які можуть бути запропоновані користувачам, і тим самим впливає на установку цін та розробку тарифних планів.

Важливо, щоб цінова політика була гнучкою та адаптованою до різних категорій користувачів. Це дозволяє відображати реальну вартість, яку додаток приносить, забезпечуючи при цьому конкурентоспроможність та доступність для цільового ринку. Ретельно продумана цінова стратегія не тільки забезпечує доходи, але й відображає цінність додатку для його користувачів. Для цього слід розглянути цінові політики використовуваних сервісів, зокрема OpenAI API та AWS [36-39].

OpenAI API має ряд різних моделей і ціна їх використання варіюється від моделі до моделі. Нижче наведено таблиці, з описом цінових політик OpenAI API, для

кожної категорії моделей. Потрібно взяти до уваги що нижче вказано ціна за 1000 токенів. Токени можна розглядати, як частини слів, де 1000 токенів — це приблизно 750 слів [40].

GPT-4 Turbo – завдяки контексту 128 КБ, новішим знанням і найширшому набору можливостей GPT-4 Turbo потужніший за GPT-4 і пропонується за нижчою ціною. Огляд цінових політик для цієї моделі представлено у таблиці 5.1.

Таблиця 5.1 Цінові політики для моделі GPT-4 Turbo

Назва Моделі	Вхідний текст	Вихідний текст
gpt-4-1106-preview	\$0.01 / тис. токенів	\$0.03 / тис. токенів
gpt-4-1106-vision-preview	\$0.01 / тис. токенів	\$0.03 / тис. токенів

GPT-4 – завдяки широким загальним знанням і досвіду в галузі GPT-4 може виконувати складні інструкції природною мовою та точно вирішувати складні проблеми. Огляд цінових політик для цієї моделі представлено у таблиці 5.2.

Таблиця 5.2 Цінові політики для моделі GPT-4

Назва Моделі	Вхідний текст	Вихідний текст
gpt-4	\$0.03 / тис. токенів	\$0.06 / тис. токенів
gpt-4-32k	\$0.06 / тис. токенів	\$0.12 / тис. токенів

GPT-3.5 Turbo – моделі GPT-3.5 Turbo потужні та економічно ефективні. gpt-3.5-turbo-1106 є флагманською моделлю цього сімейства, підтримує контекстне вікно 16К і оптимізовано для діалогу. gpt-3.5-turbo-instruct є моделлю Instruct і підтримує лише контекстне вікно 4К. Огляд цінових політик для цієї моделі представлено у таблиці 5.3.

Таблиця 5.3 Цінові політики для моделі GPT-3.5 Turbo

Назва Моделі	Вхідний текст	Вихідний текст
gpt-3.5-turbo-1106	\$0.0010 / тис. токенів	\$0.0020 / тис. токенів
gpt-3.5-turbo-instruct	\$0.0015 / тис. токенів	\$0.0020 / тис. токенів

Assistants API – API помічників та інструменти (пошук, інтерпретатор коду) полегшують розробникам створення помічників ШІ у власних програмах. Кожен помічник несе власну плату за зберігання файлів пошуку на основі файлів, переданих цьому помічнику. Інструмент пошуку фрагментує та індексує вміст файлів у векторній базі даних OpenAI. Токени, які використовуються для API помічника, виставляються за ставками введення/виведення вибраної мовної моделі за токен, і помічник розумно вибирає, який контекст із потоку включити під час виклику моделі. Огляд цінових політик для цієї моделі представлено у таблиці 5.4.

Таблиця 5.4 Цінові політики для моделі асистентів

Інструмент	Вхідні дані
Інтерпретатор коду	\$0,03 / сесія
Отримання	\$0,20 / Гб / помічник / день (безкоштовно до 01.02.2024)

Моделі тонкого налаштування – надають можливість створення власних користувацьких моделей, які можна точно налаштувати на основі базових моделей з користувацькими навчальними даними. Після точного налаштування моделі користувачеві буде виставлено рахунок лише за токени, які він використовує в запитах до цієї моделі. Огляд цінових політик для цієї моделі представлено у таблиці 5.5.

Таблиця 5.5 Цінові політики для моделі тонкого налаштування

Модель	Тренування	Вхідне використання	Вихідне використання
gpt-3.5-turbo	\$0.0080 / тис. токенів	\$0.0030 / тис. токенів	\$0.0060 / тис. токенів
davinci-002	\$0.0060 / тис. токенів	\$0.0120 / тис. токенів	\$0.0120 / тис. токенів
babbage-002	\$0.0004 / тис. токенів	\$0.0016 / тис. токенів	\$0.0016 / тис. токенів

Моделі вбудовування – надають можливість створення розширеного пошуку, кластеризації, моделювання тем і класифікацію за допомогою пропозицій щодо вбудовування. Огляд цінових політик для цієї моделі представлено у таблиці 5.6.

Таблиця 5.6 Цінові політики для моделі вбудовування

Модель	Використання
ada v2	\$0.0001 / тис. токенів

Базові моделі – моделі GPT не оптимізовані для виконання інструкцій і мають менші можливості, але вони можуть бути ефективними, якщо їх точно налаштувати для вузьких завдань. Огляд цінових політик для цієї моделі представлено у таблиці 5.6.

Таблиця 5.7 Цінові політики для базових моделей

Модель	Використання
davinci-002	\$0.0020 / тис. токенів
bebbage-002	\$0.0004 / тис. токенів

Нижче детальніше представлено цінові політики та тарифні плани AWS EC2 (таблиця 5.8) та AWS RDS (таблиця 5.9), які можуть бути використані для підтримки різних рівнів навантаження та продуктивності нашого веб-додатку. Ці таблиці демонструють, як можна збалансувати вартість та продуктивність, виходячи з конкретних потреб додатку та його користувачів. Для AWS EC2 представлено таблицю з найдешевшими сутностями, так як їх загальна кількість сягає 695 штук і буде не раціонально розглядати їх всіх. Більше того розглянуті сутності, покривають всі потреби додатку, на тій стадії розробки, на якій він перебуває.

Таблиця 5.8 Цінові політики для сутностей AWS EC2

Назва екземпляру	Погодинна плата за вимогою	vCPU	Пам'ять	Зберігання	Продуктивність мережі
t4g.nano	\$0.0042	2	0.5 GiB	Лише EBS	До 5 Gigabit
t4g.micro	\$0.0084	2	1 GiB	Лише EBS	До 5 Gigabit
t4g.small	\$0.0168	2	2 GiB	Лише EBS	До 5 Gigabit
t4g.medium	\$0.0336	2	4 GiB	Лише EBS	До 5 Gigabit
t4g.large	\$0.0672	2	8 GiB	Лише EBS	До 5 Gigabit
t4g.xlarge	\$0.1344	4	16 GiB	Лише EBS	До 5 Gigabit
t4g.2xlarge	\$0.2688	8	32 GiB	Лише EBS	До 5 Gigabit
t3.nano	\$0.0052	2	0.5 GiB	Лише EBS	До 5 Gigabit
t3.micro	\$0.0104	2	1 GiB	Лише EBS	До 5 Gigabit

Таблиця 5.9 Цінові політики для сутностей AWS RDS

Стандартні екземпляри - поточне покоління	Ціна за годину
db.t4g.micro	\$0.037
db.t4g.small	\$0.074
db.t4g.medium	\$0.149
db.t4g.large	\$0.298
db.t4g.xlarge	\$0.595
db.t4g.2xlarge	\$1.19
db.t3.micro	\$0.042
db.t3.small	\$0.084
db.t3.medium	\$0.168

## 5.2. Результати оцінки ефективності роботи веб-додатку.

У цьому розділі ми зосереджуємо нашу увагу на оцінці ефективності розробленого веб-додатку, розглядаючи ключові показники, такі як швидкість роботи, точність генерації коду та економічна ефективність.

Перше, на що ми звертаємо увагу – це швидкість роботи додатку та час відгуку системи. Швидкість реагування на запити є фундаментальним фактором, який впливає на загальну задоволеність користувачів. Вивчення та порівняння часу відгуку нашого веб-додатку з ChatGPT та іншими GPT-асистентами демонструє нашу перевагу або відставання у цій області. Ми використовуємо гістограми для наочного представлення цих даних, що допомагає виявити потенційні можливості для оптимізації. Спершу буде представлено ряд гістограм, на яких зображено середній час відгуку системи для двох мов програмування, а саме Python (рис. 5.1) та JavaScript (рис. 5.2) та їх фреймворків і пакетів. Час запиту вимірюється від моменту, коли користувач надіслав запит, і до моменту коли було завершено написання тексту.



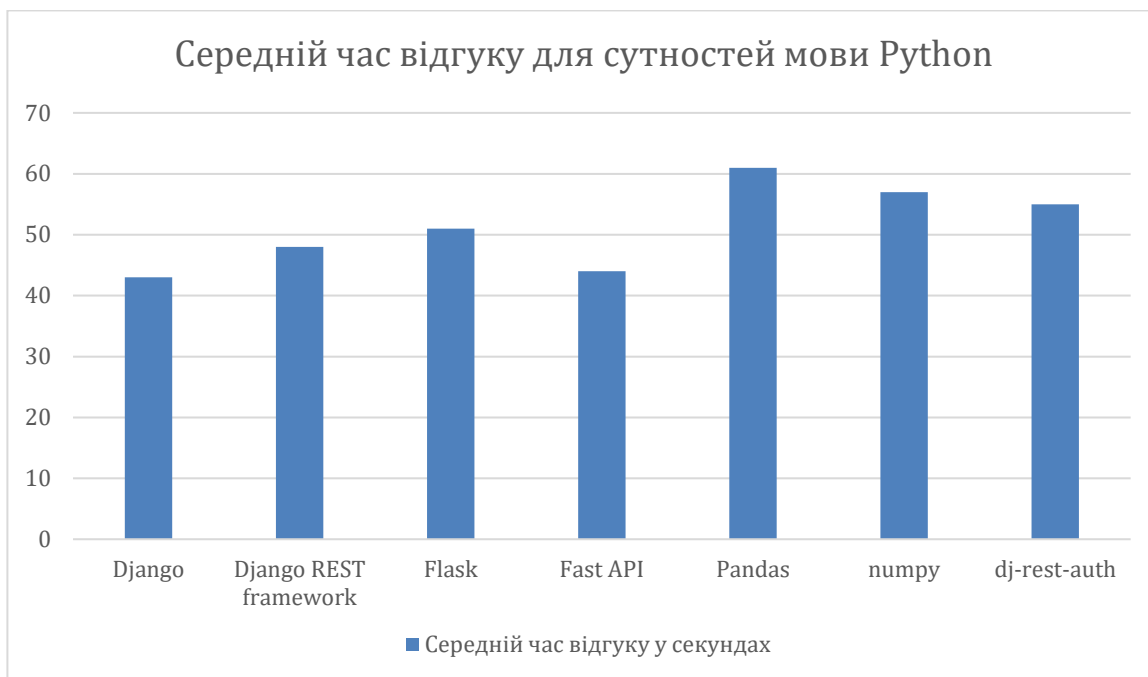


Рисунок 5.1 – Гістограма середнього часу відгуку для сутностей мови програмування Python

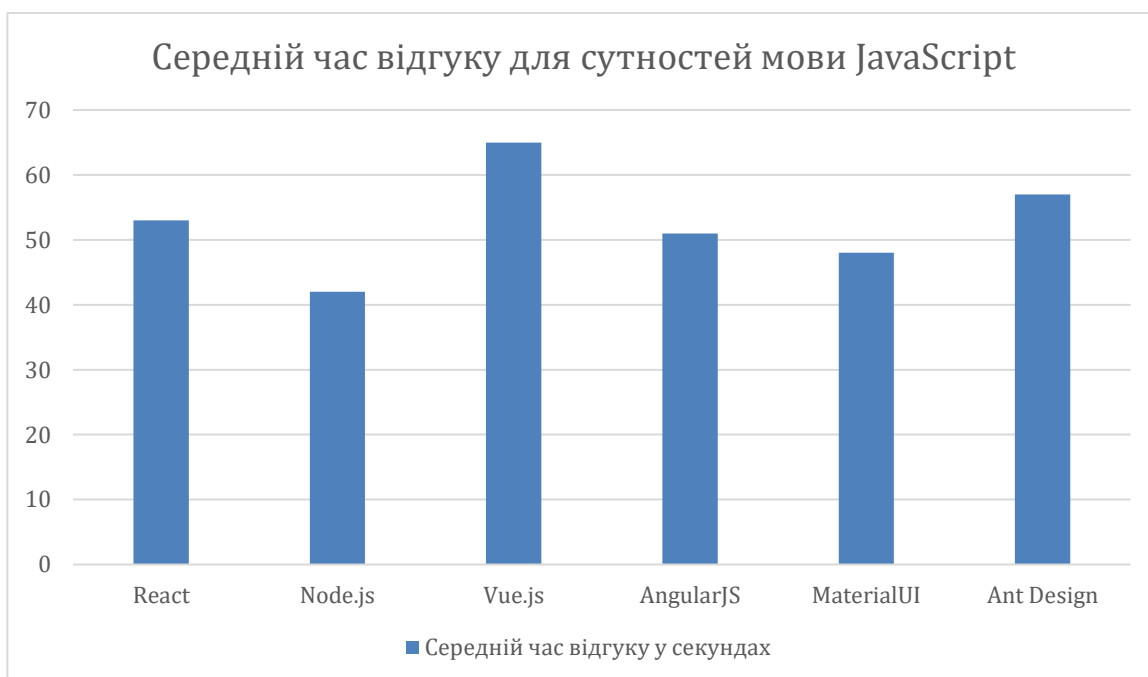


Рисунок 5.2 – Гістограма середнього часу відгуку для сутностей мови програмування JavaScript

Також, було проведено ряд замірів для порівняння часу відгуку на однакові запити у веб-додатку розробленому в рамках магістерської роботи, звичайного ChatGPT з використанням моделі GPT-3.5 та асистента, якого можна зробити у

преміум версії ChatGPT розробленого на основі такої самої інструкції, яка використовується у розробленому веб-додатку. Тести були проведені для мови програмування Python, фреймворку Django REST framework та бібліотеки dj-rest-auth, створеної для інтеграції можливостей аутентифікації. На рисунку 5.3 наведено гістограму, на якій детально зображено час відгуку для кожного з трьох запитів.

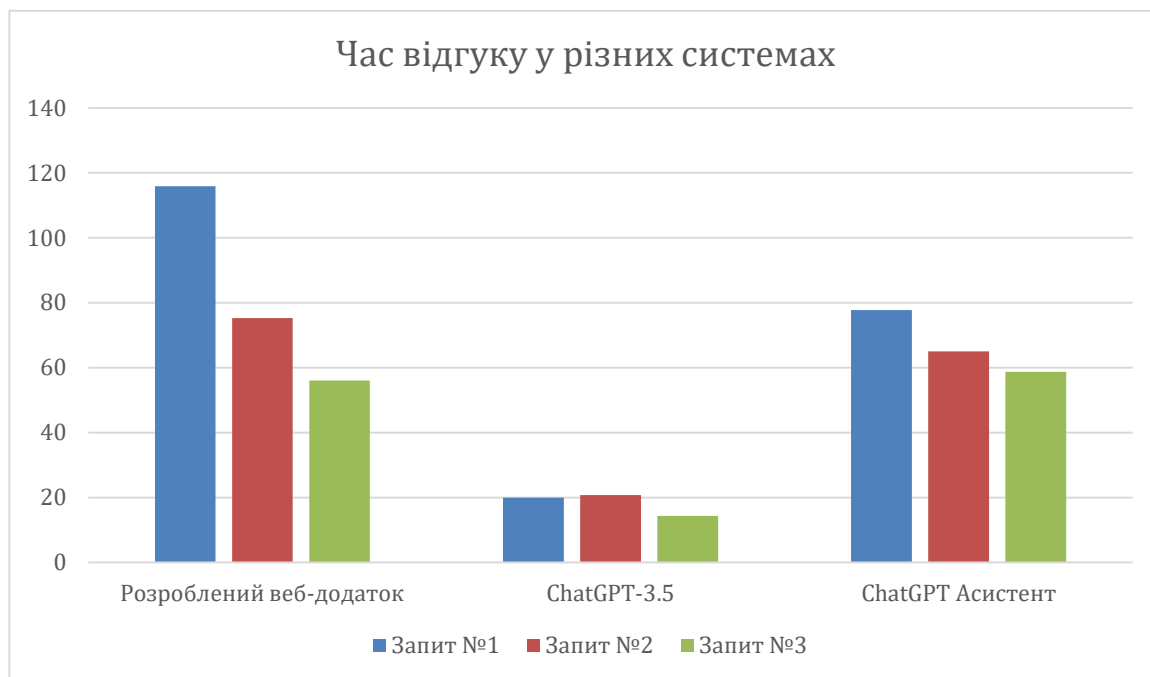
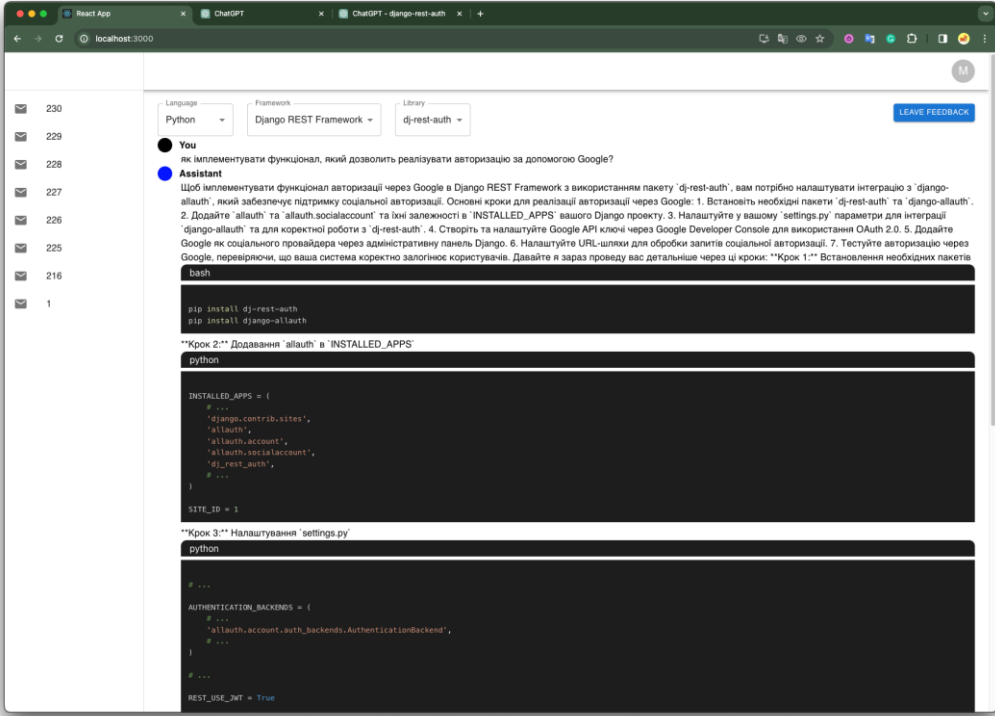


Рисунок 5.3 – Гістограма часу відгуку для кожного з запитів різними системами

З гістограми зображеної на рисунку 5.3 можна побачити, що час відгуку розробленого веб-додатку є в рази повільнішим ніж використання звичайного ChatGPT-3.5. Проте, він є подібним до часу відгуку асистента зробленого у преміум версії ChatGPT. Як видно з графіку, час відповіді на перший запит є значно довшим, а саме на 38 секунд. Різниця у другому не є такою суттєвою, а саме 10 секунд. Час третього запиту становить на 3 секунди менше ніж час асистента преміум версії ChatGPT. Хоча, тенденція часу відгуку у розробленому веб-додатку є гіршою ніж у ChatGPT та його преміум версії, слід розуміти, що це не основне, адже слід враховувати якість відповідей.

Наступний аспект, який ми аналізуємо, це точність генерації коду. Важливість цього параметра є ключовим показником, оскільки він безпосередньо впливає на якість кінцевого продукту, який отримує користувач. Порівняння відповідей нашого

веб-додатку з тими, які надає ChatGPT, дозволяє нам оцінити точність та надійність нашої системи. Цей аналіз допомагає виявити області, де наш додаток може бути удосконалений. Далі наведено ряд знімків екрана з результатами, які були отримані в ході виконання тестування часу відгуку (рис. 5.4–5.16).



The screenshot shows a web browser window with a ChatGPT interface. The browser tabs include 'React App', 'ChatGPT', and 'ChatGPT - django-rest-auth'. The address bar shows 'localhost:3000'. The ChatGPT interface has a language dropdown set to 'Python', a framework dropdown set to 'Django REST Framework', and a library dropdown set to 'dj-rest-auth'. A 'LEAVE FEEDBACK' button is visible in the top right. The chat history on the left shows messages 230, 229, 228, 227, 226, 225, 216, and 1. The main chat area shows a user question: 'Як імплементувати функціонал, який дозволить реалізувати авторизацію за допомогою Google?'. The assistant's response includes an introduction, a list of steps for implementation, and three code snippets: 1. Terminal commands to install 'dj-rest-auth' and 'django-allauth'. 2. Python code to add 'allauth' to the 'INSTALLED\_APPS' list in 'settings.py'. 3. Python code to configure 'AUTHENTICATION\_BACKENDS' in 'settings.py' to include 'allauth.account\_auth\_backends.AuthenticationBackend' and set 'REST\_USE\_JWT = True'.

```
bash

pip install dj-rest-auth
pip install django-allauth

**Крок 2** Додавання 'allauth' в 'INSTALLED_APPS'

python

INSTALLED_APPS = (
    # ...
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'dj_rest_auth',
    # ...
)

SITE_ID = 1

**Крок 3** Налаштування 'settings.py'

python

# ...

AUTHENTICATION_BACKENDS = (
    # ...
    'allauth.account_auth_backends.AuthenticationBackend',
    # ...
)

# ...

REST_USE_JWT = True
```

Рисунок 5.4 – Перша частина відповіді на перший запит за допомогою розробленого веб-додатку

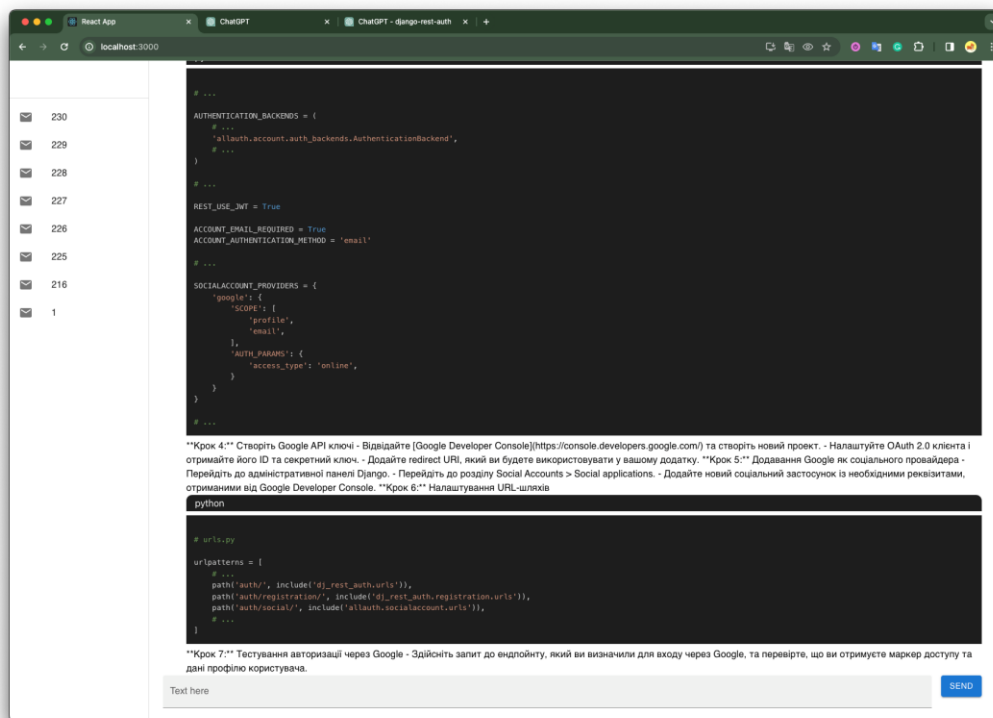


Рисунок 5.5 – Друга частина відповіді на перший запит за допомогою розробленого веб-додатку

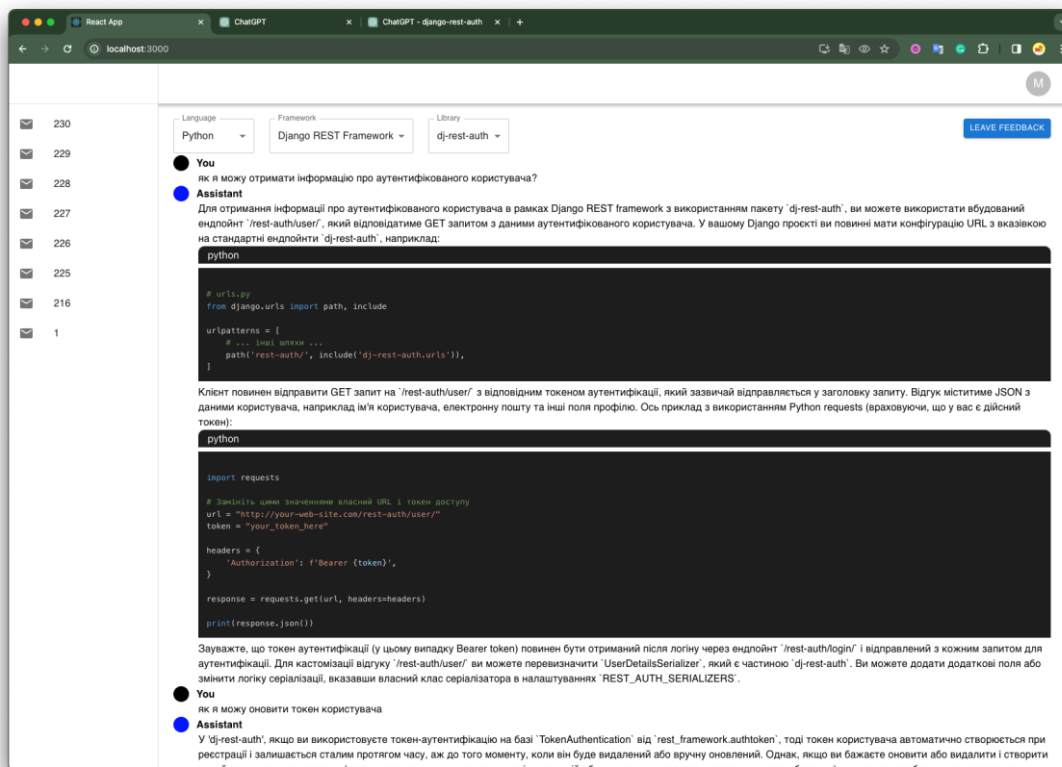


Рисунок 5.6 – Перша частина відповіді на другий та третій запити за допомогою розробленого веб-додатку

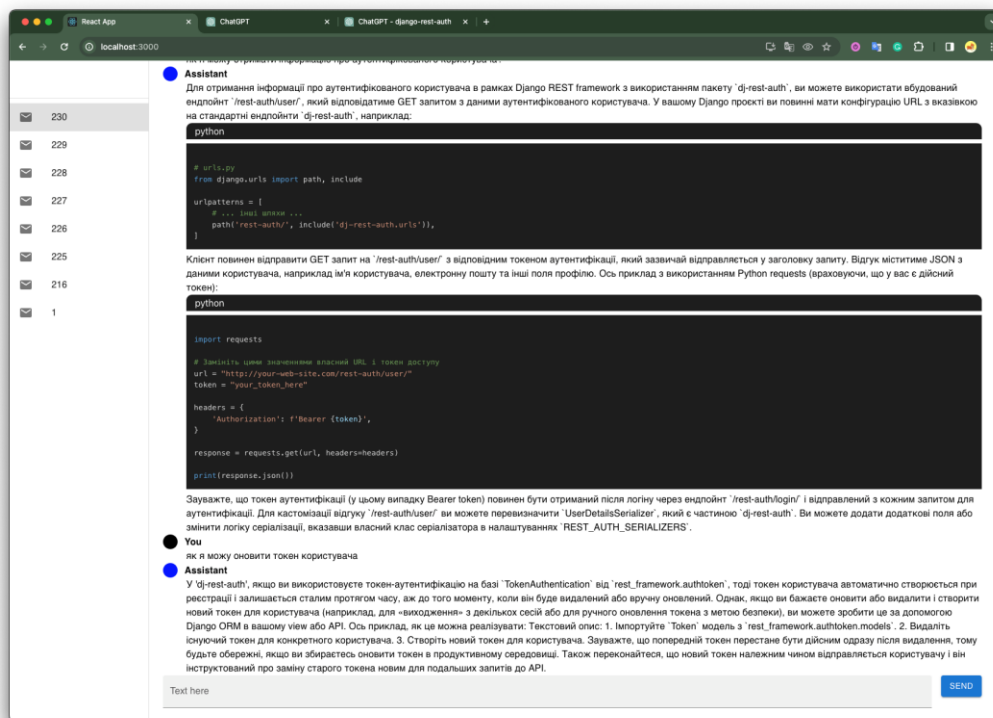


Рисунок 5.7 – Друга частина відповіді на другий татретій запитити за допомогою розробленого веб-додатку

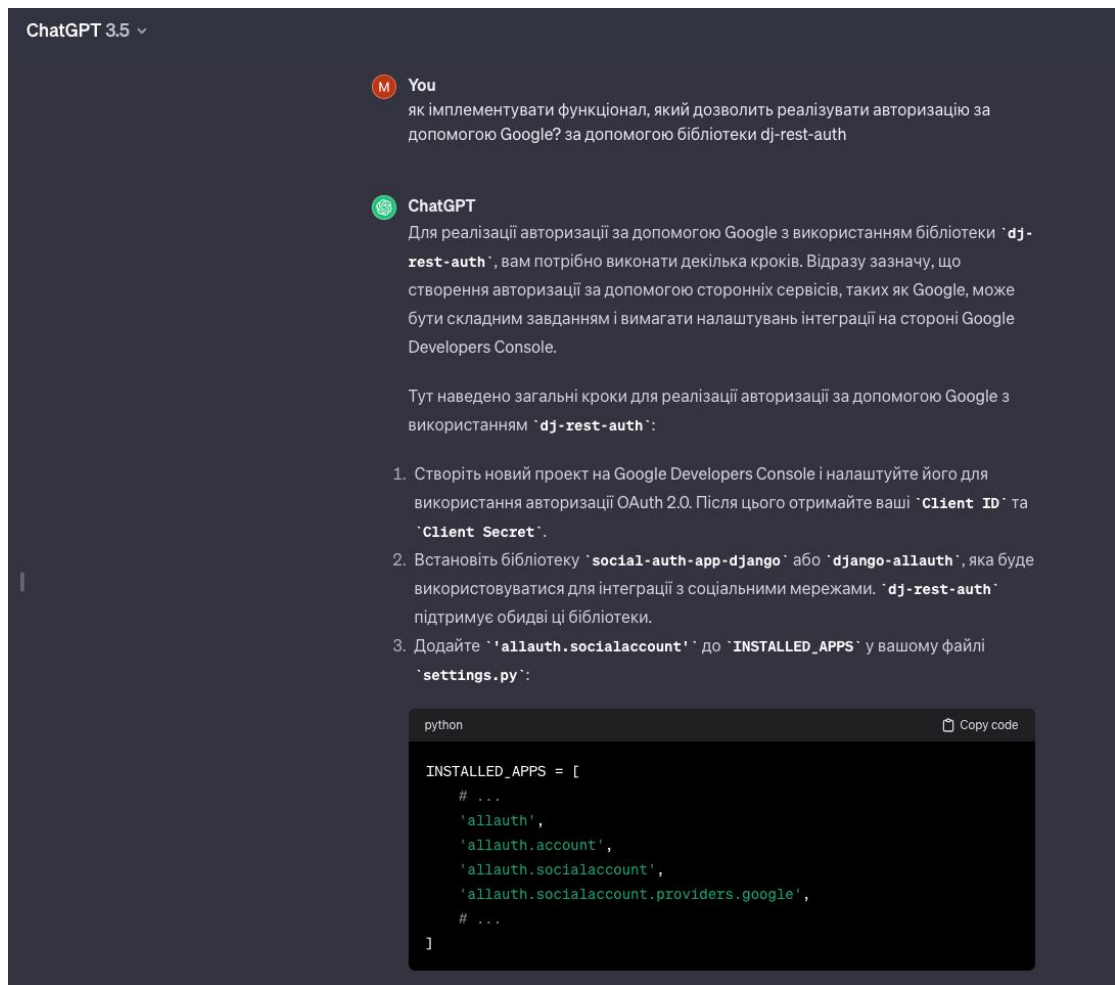


Рисунок 5.8 – Перша частина відповіді на перший запит за допомогою ChatGPT-3.5

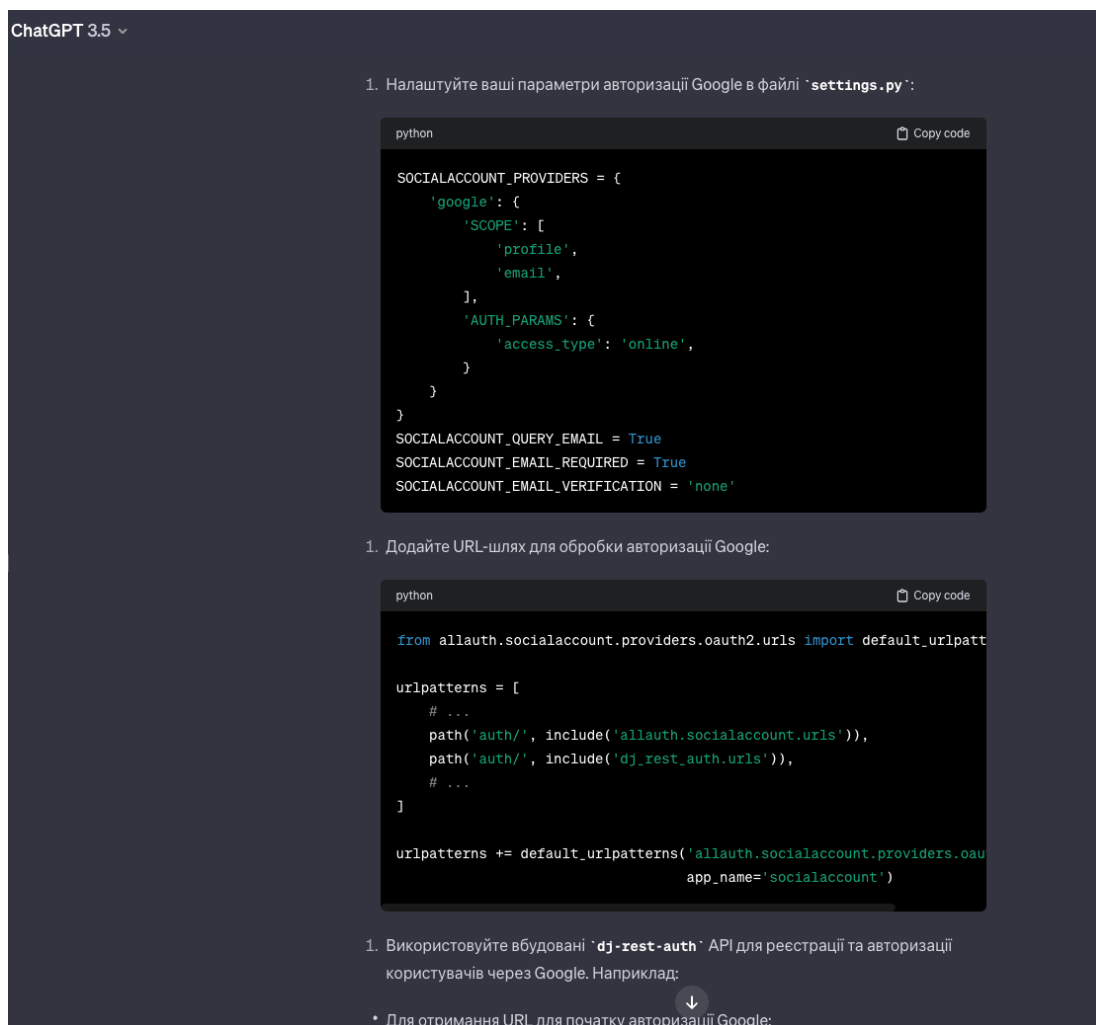


Рисунок 5.9 – Друга частина відповіді на перший запит за допомогою ChatGPT-3.5

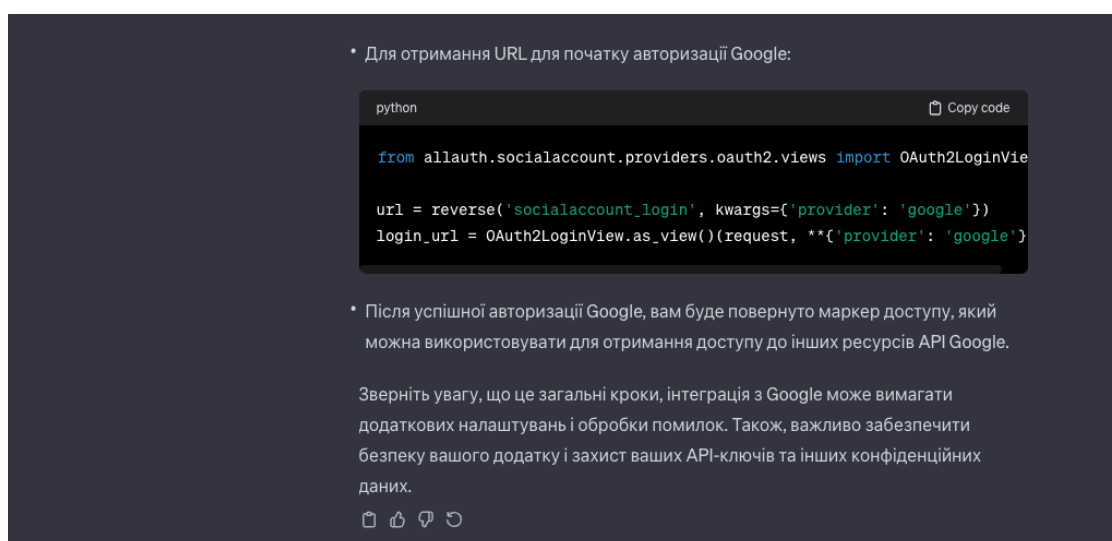


Рисунок 5.10 – Третя частина відповіді на перший запит за допомогою ChatGPT-3.5

**You**

як я можу отримати інформацію про аутентифікованого користувача? за допомогою бібліотеки `django-rest-auth`

**ChatGPT**

Для отримання інформації про аутентифікованого користувача за допомогою бібліотеки `django-rest-auth`, вам потрібно буде використовувати токен або сесію, якщо вони вже були створені під час аутентифікації користувача. Тут є кілька кроків, які ви можете виконати:

1. Переконайтеся, що `django-rest-auth` встановлений і налаштований у вашому проєкті.
2. Аутентифікуйте користувача. Це може бути зроблено через вхід по електронній пошті та паролю або іншим способом, який ви налаштували.
3. Після аутентифікації користувача, ви можете отримати інформацію про нього з допомогою звернення до API бібліотеки `django-rest-auth`. Для цього зазвичай використовуються HTTP-запити (GET, POST, тощо).
4. Зазвичай інформація про аутентифікованого користувача може бути отримана за допомогою запиту до URL, який надає таку інформацію. Наприклад:

```
python Copy code

from rest_framework.response import Response
from rest_framework.decorators import api_view, authentication_classes
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated

@api_view(['GET'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def get_authenticated_user_info(request):
    user = request.user # Отримати об'єкт користувача
    user_info = {
        'id': user.id,
        'username': user.username,
        'email': user.email,
        # Додаткова інформація про користувача, яку ви хочете надати
    }
    return Response(user_info)
```

У цьому прикладі ми створюємо API-вид на серверному боці, який вимагає аутентифікації за допомогою токена (`TokenAuthentication`) і має дозвіл `IsAuthenticated`. Потім ми отримуємо інформацію про аутентифікованого користувача і повертаємо її відповідями JSON.

5. Ваш фронтенд (або інша частина системи) може виконати HTTP-запит до цього API-виду і отримати інформацію про аутентифікованого користувача.

Не забудьте налаштувати маршрутизацію і URL для вашого API-виду та включити бібліотеку `django-rest-auth` у налаштуваннях вашого Django-проєкту. Також, залежно від вашого застосунку, вам може бути потрібно виконати додаткові налаштування для автентифікації користувачів, наприклад, встановити належні дозволи та вказати методи аутентифікації.

Рисунок 5.11 – Відповіді на другий запит за допомогою ChatGPT-3.5



**You**  
як я можу оновити токен користувача, за допомогою цієї ж бібліотеки?

**ChatGPT**  
Для оновлення токена користувача за допомогою бібліотеки `django-rest-auth`, ви можете використовувати вбудований API-вид для оновлення токена, або створити власний відображення (view) для цієї операції.

Оновлення токена користувача може бути корисним, якщо ви хочете перегенерувати токен після зміни паролю або за інших обставин.

Спосіб 1: Використання вбудованого API-виду:

Бібліотека `django-rest-auth` надає API-вид з ім'ям `rest_auth.views.LoginView`. Ви можете використовувати цей вид для оновлення токена користувача. Вам просто потрібно зробити POST-запит на відповідний URL з необхідними даними.

Спосіб 2: Створення власного API-виду:

Якщо ви хочете мати більше контролю над процесом оновлення токена, ви можете створити власний API-вид. Ось приклад:

```
python Copy code

from rest_framework import status
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
from rest_auth.views import LoginView

class UpdateTokenView(APIView):
    authentication_classes = (TokenAuthentication,)
    permission_classes = (IsAuthenticated,)

    def post(self, request):
        user = request.user
        token, created = Token.objects.get_or_create(user=user)

        if not created:
            token.delete()
            token = Token.objects.create(user=user)

        return Response({'token': token.key}, status=status.HTTP_200_OK

# Ваші маршрути та URL для власного API-виду
# ...
```

Цей приклад створює власний API-вид `UpdateTokenView`, який оновлює токен користувача, перегенеруючи його. Ви можете налаштувати відповідні URL та маршрути для цього виду в залежності від вашого проекту.

Після створення API-виду для оновлення токена, ви можете зробити POST-запит до відповідного URL з коректними даними для оновлення токена користувача.





   

Рисунок 5.12 – Відповідь на третій запит за допомогою ChatGPT-3.5

**You**  
як імплементувати функціонал, який дозволить реалізувати авторизацію за допомогою Google?

**django-rest-auth**  
Щоб імплементувати функціонал авторизації за допомогою Google у Django проекті з використанням `django-rest-auth`, вам потрібно виконати кілька кроків. Цей процес включає налаштування `django-allauth`, який є інтегрованим з `django-rest-auth` для соціальної авторизації. Ось основні кроки:

- Встановіть `django-allauth` та `django-rest-auth`:**  
Якщо ви ще не встановили `django-rest-auth` та `django-allauth`, вам потрібно їх встановити через pip:
 

```
bash
pip install django-rest-auth
pip install django-allauth
```
- Додайте відповідні додатки до `INSTALLED_APPS` у вашому `settings.py` файлі:**

```
python
INSTALLED_APPS = [
    # ...
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.google',
    'dj_rest_auth',
    # ...
]
```
- Налаштуйте `django.contrib.sites`:**  
У `settings.py`, встановіть `SITE_ID`. Це зазвичай `1` для першого сайту, який ви створили.
 

```
python
SITE_ID = 1
```
- Налаштуйте URL-адреси:**  
В файлі `urls.py` вашого проекту додайте наступний URL-шлях:
 

```
python
urlpatterns = [
    # ...
    path('auth/', include('dj_rest_auth.urls')),
    path('auth/registration/', include('dj_rest_auth.registration.urls')),
    path('auth/social/', include('allauth.socialaccount.urls')),
    # ...
]
```
- Створіть OAuth 2.0 Client ID у Google Cloud Console:**
  - Перейдіть до [Google Cloud Console](#).
  - Створіть новий проект або використовуйте існуючий.
  - В меню "APIs & Services" > "Credentials", створіть новий OAuth 2.0 Client ID.
  - Вкажіть URI перенаправлення, які використовуються `django-allauth`.  
Наприклад: `http://yourdomain.com/accounts/google/login/callback/`.

Рисунок 5.13 – Перша частина відповіді на перший запит за допомогою преміум асистента ChatGPT

5. Створіть OAuth 2.0 Client ID у Google Cloud Console:

- Перейдіть до Google Cloud Console.
- Створіть новий проект або використовуйте існуючий.
- В меню "APIs & Services" > "Credentials", створіть новий OAuth 2.0 Client ID.
- Вкажіть URI перенаправлення, які використовуються `django-allauth`.  
Наприклад: `http://yourdomain.com/accounts/google/login/callback/`.

6. Конфігуруйте провайдера соціальної авторизації в Django admin:

- Перейдіть до Django admin.
- Перейдіть у розділ "Social accounts" > "Social applications".
- Додайте новий соціальний додаток для Google.
- Введіть ім'я, ID клієнта, таємницю клієнта (отримані у Google Cloud Console) та оберіть сайт зі списку.

7. Налаштуйте додаткові параметри (опціонально):

В `settings.py`, ви можете налаштувати додаткові параметри для `django-allauth` та `dj-rest-auth`, наприклад:

```
python Copy code  
  
# Параметри django-allauth  
SOCIALACCOUNT_AUTO_SIGNUP = True  
SOCIALACCOUNT_EMAIL_VERIFICATION = 'none'  
  
# Параметри dj-rest-auth  
REST_USE_JWT = True
```

Після завершення цих кроків, ваш Django проект буде готовий приймати авторизацію через Google. Ваші користувачі зможуть використовувати свої Google аккаунти для входу в систему.

📄 📌 🔗 ↻

Рисунок 5.14 – Друга частина відповіді на перший запит за допомогою преміум асистента ChatGPT

**You**  
як я можу отримати інформацію про аутентифікованого користувача?

**django-rest-auth**  
Щоб отримати інформацію про аутентифікованого користувача у Django REST Framework, використовуючи пакет `dj-rest-auth`, вам потрібно налаштувати ендпоінт, який буде повертати дані про поточного користувача. Ось кроки, які ви можете виконати:

- Встановлення `dj-rest-auth`:** Якщо ви ще не встановили `dj-rest-auth`, ви можете зробити це за допомогою `pip`:
 

```
bash
pip install dj-rest-auth
```
- Додавання `dj-rest-auth` у ваші `INSTALLED_APPS`:** В файлі `settings.py` вашого Django проекту додайте `dj-rest-auth`:
 

```
python
INSTALLED_APPS = [
    # ...
    'dj_rest_auth',
    # ...
]
```
- URL конфігурація:** В файлі `urls.py` вашого Django проекту імпортуйте `include` і додайте шлях до `dj-rest-auth`:
 

```
python
from django.urls import path, include

urlpatterns = [
    # ...
    path('dj-rest-auth/', include('dj_rest_auth.urls')),
    # ...
]
```
- Створення ендпоінту для отримання даних користувача:** `dj-rest-auth` надає вбудований ендпоінт `/login/` для аутентифікації і `/logout/` для виходу. Для отримання даних про поточного користувача, ви можете використовувати ендпоінт `/user/`. Це виглядає так:
 

```
python
path('dj-rest-auth/user/', include('dj_rest_auth.urls'))
```
- Перевірка:** Тепер, коли ви виконаєте запит GET до `/dj-rest-auth/user/`, ви отримаєте інформацію про поточного аутентифікованого користувача. Важливо, що запит повинен містити валідний токен аутентифікації, якщо ви використовуєте токен-базовану аутентифікацію.

Це загальний підхід, і він може варіюватися в залежності від вашої конкретної реалізації та налаштувань. Переконайтеся, що ваша система аутентифікації налаштована правильно, і ви використовуєте відповідні налаштування безпеки для захисту даних користувача.

Рисунок 5.15 – Відповідь на другий запит за допомогою преміум асистента ChatGPT

**You**  
як я можу оновити токен користувача

**django-rest-auth**  
Оновлення токена користувача в `django-rest-auth` зазвичай відноситься до процесу оновлення доступного токена (access token) за допомогою токена оновлення (refresh token), якщо ви використовуєте JSON Web Token (JWT) для аутентифікації. Ось як це можна зробити:

- Встановлення та налаштування `django-rest-auth-simplejwt`:** Щоб використовувати JWT з `django-rest-auth`, вам потрібно встановити `django-rest-auth-simplejwt`. Це можна зробити за допомогою pip:
 

```
bash
pip install django-rest-auth-simplejwt
```
- Додавання налаштувань Simple JWT:** У файлі `settings.py` вашого Django проекту додайте наступні налаштування:
 

```
python
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
}

from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,

    # інші налаштування за потреби...
}
```
- URL конфігурація для JWT:** У файлі `urls.py` вашого Django проекту додайте шляхи для обробки токенів JWT:
 

```
python
from django.urls import path, include
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
    # ...
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    # ...
]
```
- Оновлення Access Token за допомогою Refresh Token:** Коли Access Token закінчується, ви можете відправити запит POST на `/api/token/refresh/` з вашим Refresh Token, щоб отримати новий Access Token. Тіло запиту повинно містити Refresh Token:
 

```
json
{
    "refresh": "<your_refresh_token>"
}
```
- Отримання нового Access Token:** У відповідь на ваш запит буде новий Access Token, який ви можете використовувати для подальшої аутентифікації.

Переконайтеся, що ваша система безпеки налаштована правильно і ви дотримуетесь найкращих практик управління токенами, включаючи безпечне зберігання Refresh Token.

Рисунок 5.16 – Відповідь на третій запит за допомогою преміум асистента ChatGPT

Як можна побачити на знімках екрану (рис. 5.4–5.16), відповіді, які надав ChatGPT-3.5 не можна вважати коректними, зважаючи на те, що представлений опис та код не містять інформації про функціонал бібліотеки на основі якої був сформований запит. Не зважаючи на це, сформований код та пояснення до нього не є корисними, а лише можуть заплутати користувача.

Раціонально порівнювати відповіді реалізованого веб-додатку і асистента створеного у преміум версії ChatGPT. Зі знімків можна побачити, що відповіді на перший запит є дуже схожі. Вони містять всю необхідну інформацію, покрокову інструкцію та посилання на важливі джерела. Тут перевагу має асистент від ChatGPT, через те що час його відгуку на 40 секунд швидший. При порівнянні відповідей на другий запит, можна виокремити те, що відповідь розробленого веб-додатку є більш лаконічною та точною, хоча обидва асистенти у відповідях містять вирішення поставленої задачі. У результатах на третій запит, фаворитом є розроблений веб-додаток, так як він містить покрокову лаконічну інструкцію, на основі якої можна легко написати необхідний код, в той час як у відповіді преміум асистент згадує сторонню бібліотеку, описує як її інсталювати, та демонструє імплементацію, яка в результаті дає помилки, через несумісність бібліотек. Також, слід згадати про те, що у розроблений веб-додаток було задано уточнююче запитання, щодо написання коду по наданій інструкції і з цим завданням розроблений асистент впорався на «відмінно». Функцію, яку надав веб-додаток можна легко імплементувати у нову кінцеву точку, яку згодом можна використовувати для оновлення токена. Нижче представлено рисунок 5.17 з відповіддю для уточнення.

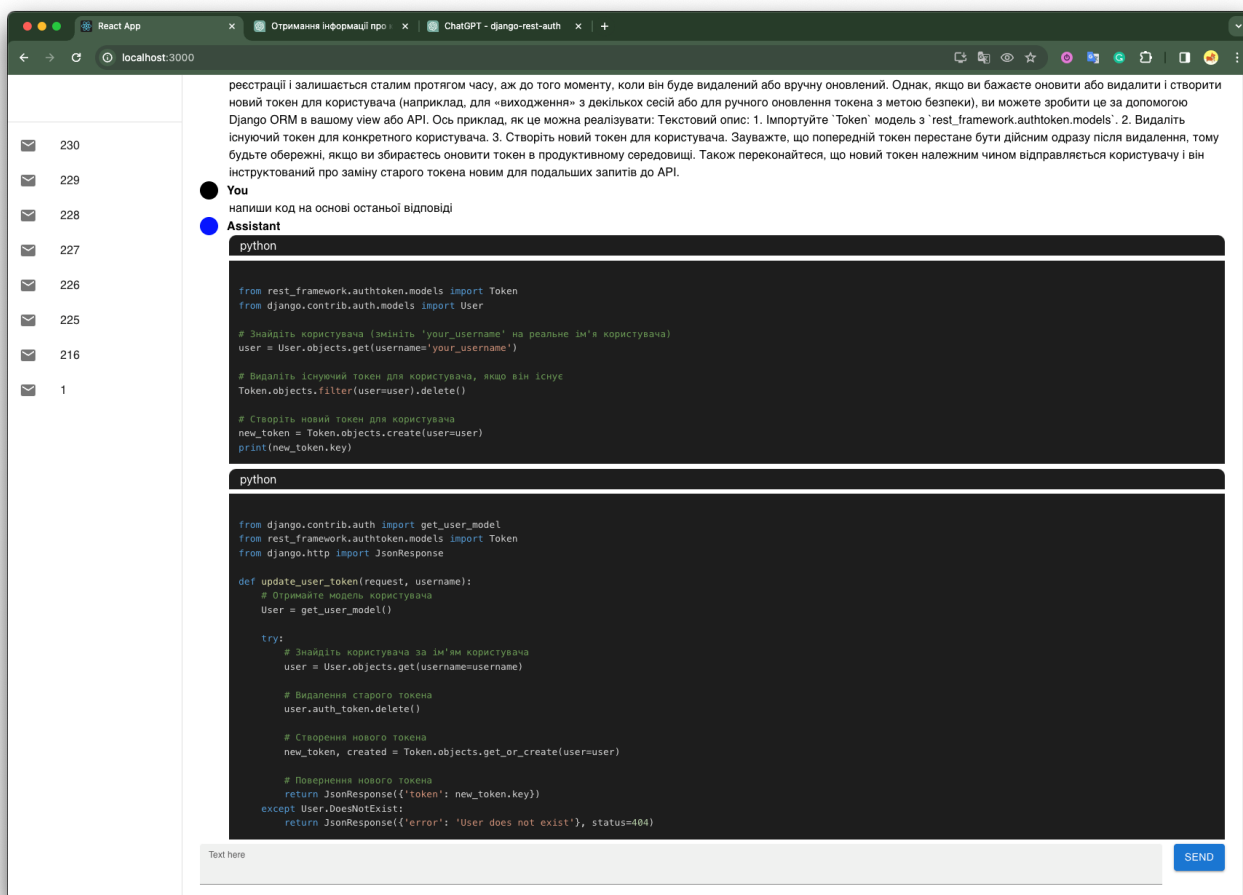


Рисунок 5.17 – Результат уточнюючої відповіді

Також, важливим показником є користувацька активність. Це важливо для розуміння на скільки додаток є актуальним і привабливим для користувачів. Це легко можна відслідкувати за допомогою аналізу існуючих сутностей в рамках бази даних та інструментів, які надає OpenAI. Нижче наведено ряд графіків (рис. 5.18–5.21), на яких зображено активність використання.

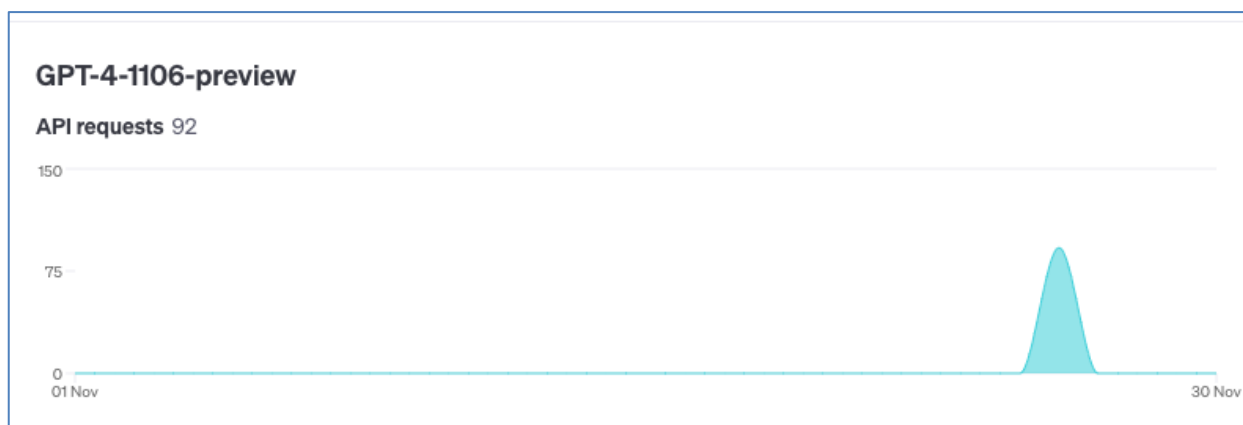


Рисунок 5.18 – Кількість запитів здійснених за використання моделі GPT-4-1106-preview у листопаді

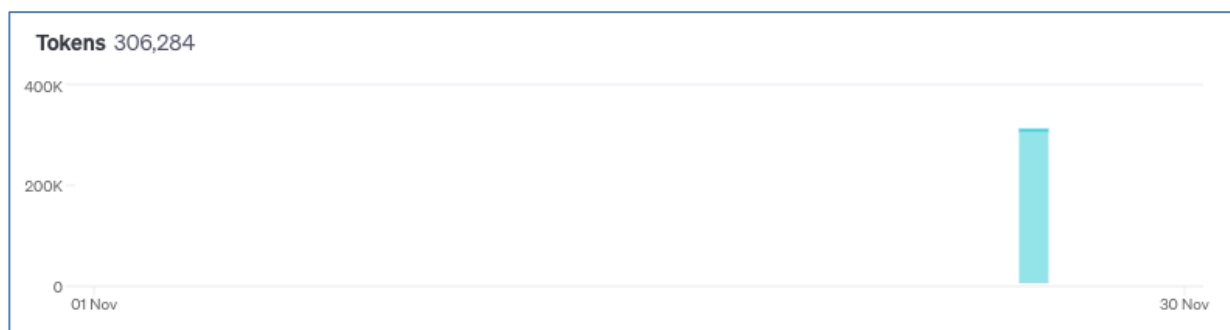


Рисунок 5.19 – Кількість використаних токенів за використання моделі GPT-4-1106-preview у листопаді

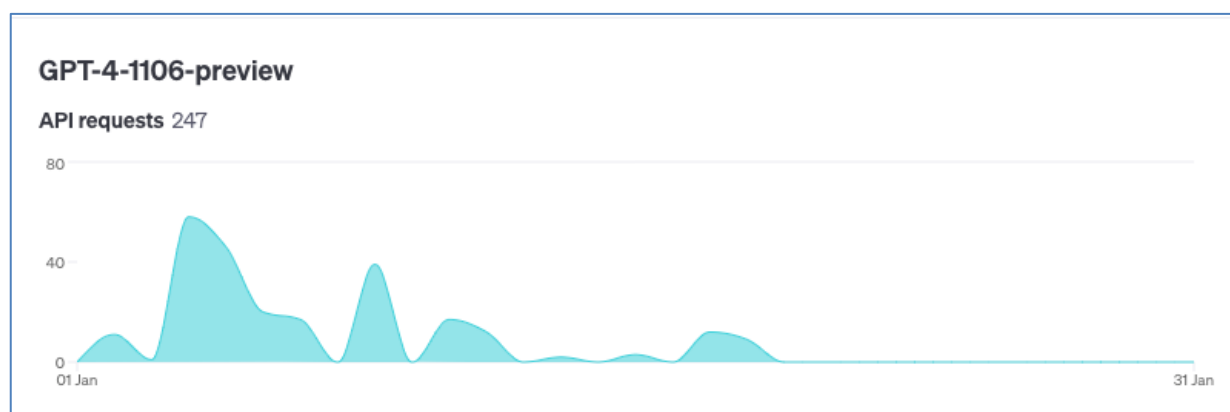


Рисунок 5.20 – Кількість запитів здійснених за використання моделі GPT-4-1106-preview у січні



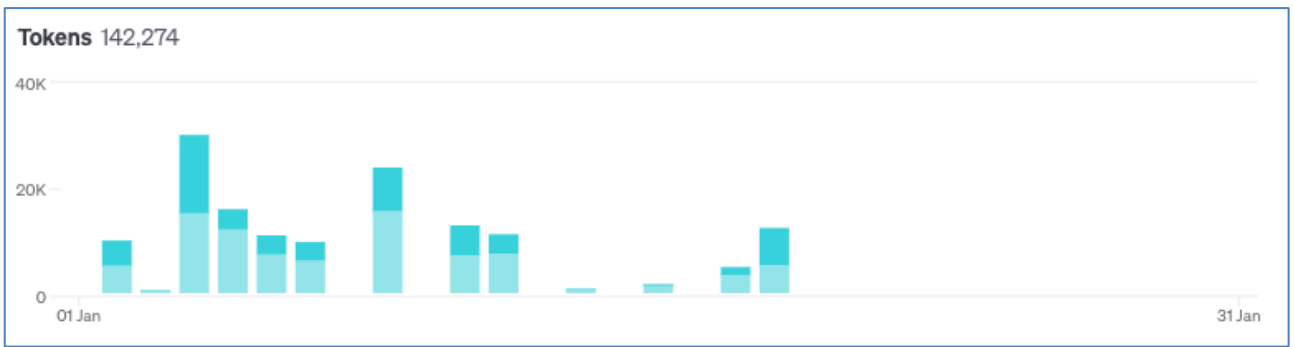


Рисунок 5.21 – Кількість використаних токенів за використання моделі GPT-4-1106-preview у січні

Останнім, але не менш важливим, є вимір економічної ефективності. Витрати на використання ChatGPT, порівняно з витратами на використання нашого додатку, можуть бути значним чинником для користувачів. Також цей аналіз є важливим для розробки ефективної монетизаційної стратегії. Слід почати з того що звичайна версія ChatGPT-3.5 є безкоштовною, проте з результатів продемонстрованих вище, було визначено що цей варіант не дає точних відповідей та може суттєво ускладнити процес розробки, заплутавши користувачів. Преміум версія ChatGPT, яка відкриває можливості використання новіших моделей та створення персональних асистентів коштує 20\$ в місяць. Проте, користувачеві буде необхідно самостійно освоїти методики написання інструкцій для створення асистентів, які надаватимуть точні відповіді та підходи формування просунутих запитів (prompt engineering). В цей час, розроблений додаток, надає користувачеві можливість користуватись вже створеними асистентами, з використанням найновіших підходів формування інструкцій під час їх створення. Це вже значно спростить користувацький досвід та дозволить користувачеві економити свій час. Повертаючись до економічної складової, нижче буде приведено графік вартості використання OpenAI API в рамках розробленого веб-додатку, за січень місяць.



Рисунок 5.22 – Кількість грошей витрачених у січні за використання OpenAI API

З рисунку 5.22, можна побачити те що за січень було використано 2 долара 63 центи. За цей час було здійснено 247 запитів, це можна побачити на рисунку 5.20. Це означає що за 10 днів активної експлуатації розробленого веб-додатку користувач робить 100-150 запитів і ціна одного запиту становить приблизно 1 цент. З цього можна зробити висновок, що за місяць активного користування буде здійснено від 300 до 450 запитів і зважаючи на середню ціну одного запиту в 1 цент за місяць буде стягуватись від 3 до 4.5 доларів. Це означає, якщо брати середні показники, що розроблений веб-додаток дешевший у 5.5 разів. Більше того, якщо користувач не користується веб-додатком з нього не будуть стягуватись кошти.

Таким чином, аналізуючи ці три ключові аспекти, ми отримуємо цілісне розуміння ефективності нашого веб-додатку, що є важливим для його подальшого розвитку та вдосконалення. Розроблений веб-додаток має декілька мінусів, а саме: не дуже привабливий веб-дизайн, не до кінця продуманий інтерфейс, а також трішки довший час середньостатистично відгуку у порівнянні з асистентом преміум версії ChatGPT. Серед ключових переваг додатку є те, що він економічно вигідніший. Вартість його активного використання є у 5.5 разів нижчою, ніж вартість використання преміум версії ChatGPT. Також, суттєвим плюсом є те, що користувач платить за використання додатку, в той час як підписка преміум версії ChatGPT становить 20 доларів, незважаючи на те чи користується користувач нею, чи ні. Також, розроблений веб-додаток є зручніший у використанні за рахунок того, що має сформовану базу натренованих асистентів, які покривають більшість стандартних запитів користувачів.

## Висновки до розділу 5

1. Визначено методи оцінки ефективності веб-додатку. Проаналізовано цінові політики сервісів використаних при розробці веб-додатку, а саме: було детально розібрано кожну існуючу модель та її варіації, зі сторони хмарних сервісів – AWS EC2 та AWS RDS.

2. Проведено тестування веб-додатку. Проведений детальний аналіз ефективності розробленої системи, зокрема: якість згенерованого коду, швидкодію застосунку та його економічну вигоду. Зроблено порівняння з веб-додатком ChatGPT за використання його різних версій: базовою, на основі моделі GPT-3.5, преміум, на основі побудованого особистого асистента.

## ЗАГАЛЬНІ ВИСНОВКИ

1. У рамках виконаної магістерської роботи було розроблено веб-додаток, який використовує інтеграцію API, наданих OpenAI та Google, для оптимізації програмувальних запитів через техніки prompt engineering та використання функціоналу асистентів. Основні результати роботи свідчать про те, що досягнутий ступінь новизни полягає у створенні унікальної системи, здатної адаптуватися до постійно змінюваної інформації про програмні бібліотеки та фреймворки, що забезпечує високу продуктивність і точність у програмуванні.

2. Порівняно з аналогами, веб-додаток демонструє значне поліпшення ефективності генерації коду. Він також відзначається економічною ефективністю, оскільки витрати на його використання значно нижчі порівняно з комерційними продуктами, що забезпечує доступність для широкого кола користувачів.

3. Наукове значення отриманих результатів полягає у глибокому аналізі можливостей сучасних API для розширення можливостей програмування, що може бути використане для подальших досліджень у цій галузі. Практична значущість роботи проявляється у розробці прототипу, який може бути застосований у реальних проектах, сприяючи зростанню ефективності розробки програмного забезпечення.

4. Прогнозні припущення на подальший розвиток дослідження включають інтеграцію додаткових функціональних можливостей, зокрема машинного навчання для автоматизації більш складних завдань програмування, а також розширення підтримки різноманітних мов програмування.

5. Показники ефективності веб-додатку, такі як час реагування, точність генерації коду та користувацька активність, підтверджують його високу конкурентоспроможність. Результати експериментальних досліджень та тестувань вказують на значний потенціал для подальшого розвитку та оптимізації системи.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Історія програмування", Вікіпедія. URL: <https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F>
2. "Структурне програмування: концепції, техніки та методи", Томас Р. Гейл, 1975.
3. "Об'єктно-орієнтоване програмування: принципи та практика", Граді Буч, 1991.
4. "Інтегровані середовища розробки", Вікіпедія. URL: [https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5\\_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5\\_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8)
5. "Машинне навчання", Кевін П. Мерфі, 2012.
6. "OpenAI Codex: автоматизація програмування за допомогою штучного інтелекту", OpenAI Blog. URL: <https://openai.com/blog/openai-codex>
7. "Google Developers", офіційний веб-сайт. URL: <https://developers.google.com/>
8. "The Next Generation of Programming with GitHub Copilot", GitHub Blog. URL: <https://github.com/features/copilot>
9. "Enhanced Coding: The Future of IDEs", JetBrains Whitepapers. URL: <https://www.jetbrains.com/fleet/>
10. "Automating Development with Google APIs", Google Cloud Documentation. URL: <https://cloud.google.com/blog/products/api-management/automating-api-delivery-with-cicd-pipelines>
11. "Advanced Machine Learning Techniques for Code Generation", Journal of Artificial Intelligence Research.
12. "Open Source Software: A Source for Innovation in Code Generation", Open Source Journal.

13. "OpenAI Codex: Powering the Next Generation of Programming", OpenAI Documentation.
14. "Exploring the Use of OpenAI Codex for Educational Purposes", International Journal of Computer Science Education.
15. "Leveraging Google's Machine Learning APIs for Code Generation", Google Cloud Platform Documentation.
16. "Utilizing Google API's for User Behavior Analysis and UX Optimization", UX Design Institute.
17. "Rapid Prototyping and Iterative Development with OpenAI and Google APIs", Software Development Magazine.
18. "Community Engagement and User Feedback in the Evolution of AI-Powered Programming Tools", Developer Community Journals.
19. "What is Web Application Development and How do I get started". URL: <https://www.upwork.com/resources/what-is-web-application-development>
20. "React Official Web Page". URL: <https://uk.legacy.reactjs.org/>
21. "Angular Official Web Page". URL: <https://angular.io/>
22. "Vue.js Official Web Page". URL: <https://vuejs.org/>
23. "Мова програмування Python". URL: [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
24. "Мова програмування Node.js". URL: <https://en.wikipedia.org/wiki/Node.js>
25. "Мова програмування Java". URL: [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
26. "Що таке хмарні технології? Переваги та недоліки". URL: <https://edin.ua/shho-take-xmarni-texnologii%D1%97-i-navishho-voni-potribni/>
27. "Overview of Amazon Web Services". URL: <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html>
28. "Google Cloud overview". URL: <https://cloud.google.com/docs/overview>
29. "Continuous Integration and Delivery (CI/CD) Explained". URL: <https://www.abtasty.com/ci-cd/>
30. "OpenAI Python API library". URL: <https://github.com/openai/openai-python>

31. “What is Amazon EC2?”. URL: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>
32. “What is Amazon RDS?”. URL: <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html>
33. “GitHub Actions documentation”. URL: <https://docs.github.com/en/actions>
34. “GitHub Actions features”. URL: <https://github.com/features/actions>
35. “OpenAI Models Overview”. URL: <https://platform.openai.com/docs/models>
36. “10 Key Application Performance Metrics & How to Measure Them”. URL: <https://stackify.com/application-performance-metrics/>
37. “Top 8 Web Application Performance Metrics”. URL: <https://www.metricfire.com/blog/top-8-web-application-performance-metrics/>
38. “Top 10 Most Important Website Performance Metrics & KPIs Developers Should Measure”. URL: <https://sematext.com/blog/website-performance-metrics/>
39. “What metrics should be tracked when measuring the performance of a web application?”. URL: <https://www.quora.com/What-metrics-should-be-tracked-when-measuring-the-performance-of-a-web-application>
40. “OpenAI Models Pricing”. URL: <https://openai.com/pricing>