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: **«Розробка веб-застосунку для допомоги програмісту з використанням засобів та технологій штучного інтелекту»**

Виконав: здобувач 6 курсу групи Іт-62

Спеціальності 126 «Інформаційні системи та технології»

(шифр і назва)

Канчалаба Тарас Юрійович

(Прізвище та ініціали)

Керівник: к.т.н., в.о. доц. Падюка Р. І.

(Прізвище та ініціали)

ДУБЛЯНИ-2025

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

Канчалабі Тарасу Юрійовичу

1. Тема роботи: «Розробка веб-застосунку для допомоги програмісту з використанням засобів та технологій штучного інтелекту »

Керівник роботи Падюка Роман Іванович, к.т.н., в.о. доцента.
затверджені наказом по університету 12 вересня 2024 року № 616/к-с

2. Строк подання здобувачем роботи 10.01.2025р.

3. Вихідні дані до роботи: *база даних з інструкціями для створення асистентів та їх ідентифікаторами в відповідних API; методтка розробки бекенд та фронтенд для веб-застосунків; методика створення веб-застосунків на основі 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-застосунку
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-застосунку; Огляд архітектури веб-застосунку; Інтеграція з OpenAI та Google API; Робочі вікна створеного веб-застосунку;Порівняння часу відгуку різних систем; Огляд активності використання веб-застосунку.

6. Консультанти з розділів:

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

7. Дата видачі завдання 16 вересня 2024 р.

Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Приміт-ка
1	Написання першого розділу та означення головних завдань роботи	16.09.-30.09.24	
2	Виконання другого розділу та формування головних показників для розрахунків	01.10.-16.10.24	
3.	Виконання третього розділу та формування початкових даних	16.10.-30.10.24	
4.	Виконання четвертого розділу та узагальнення отриманих результатів магістерської роботи	01.11.-07.11.24	
6.	Вартісне оцінення ефективності пропозицій роботи	08.11.-16.11.24	
7.	Завершення оформлення розрахунково-пояснювальної записки та аркушів графічної частини	17.11.-25.11.24	
8.	Завершення роботи в цілому	26.11.-10.01.25	

Здобувач_____Канчалаба Т.Ю.
(підпис)

Керівник роботи_____Падюка Р. І.
(підпис)

УДК 004.9 : 631.1

Розробка веб-застосунку для допомоги програмісту з використанням засобів та технологій штучного інтелекту Канчалаба Т.Ю. Кафедра ІТ – Дубляни, Львівський НУП, 2025.

Кваліфікаційна робота: 85 с. текст. част. 52 рис., 10 табл., 11 арк. ілюстраційного матеріалу, 40 джерел.

Подано огляд існуючих аналогових інформаційних систем для вирішення проблеми генерації коду різними підходами. Розроблені завдання кваліфікаційної роботи.

Проаналізовано різні підходи до побудови веб-додатків. Переглянуто багато рекомендацій, які надають хмарні служби щодо розгортання веб-додатків.

Окреслено завдання розробки веб-застосунків для генерації коду. Вибрано оптимальний засіб реалізації. Проектування веб-застосунку здійснено на різних рівнях, а саме: інфраструктури, алгоритмів і систем.

Описано особливості реалізації цієї веб-програми, включаючи серверну та користувальницьку частини.

Під час використання веб-застосунку на випадок виникнення надзвичайної ситуації запропоновано заходи з охорони праці.

Визначено показники продуктивності розробленого веб-застосунку.

Ключові слова: веб-застосунок, допомога програмісту, машинне навчання, мовна модель, API.

ЗМІСТ

ВСТУП

Розвиток інформаційних систем сьогодні є одним із видів ренесансу, необхідних для того, щоб йти в ногу зі швидкими змінами технологій і потреб ринку. Таким чином, нові парадигми та технології витіснили традиційні процеси аналізу, проектування та впровадження ІС. Автоматизація всіх цих процесів є сучасним трендом, підкріпленим останніми досягненнями у сфері штучного інтелекту та обробки природної мови.

Практичною необхідністю є створення в найкоротші терміни ІС, здатних адекватно реагувати на зміни, що вкрай важливо в умовах швидких змін і великого обсягу інформації. Проектування такої системи має базуватися не лише на сучасних технологіях, а й на досвіді застосування алгоритмів машинного навчання з урахуванням унікальності кожного конкретного випадку.

Ще одним ключовим завданням є розробка веб-додатку, який може використовувати можливості OpenAI і API Google для створення коду на основі запитів користувачів. Це принесе більше гнучкості та ефективності в отриманні доступу до розробки програм і в точному виконанні сучасних програмних потреб.

Магістерська дисертація зосереджена на створенні та дослідженні веб-додатку, спрямованого на ефективну оптимізацію розробки програмування за допомогою сучасної інтеграції API та методології гнучкої розробки. Актуальність цього дослідження є лише похідною від того постійно зростаючого попиту на швидку та якісну розробку програмного забезпечення, спонукального заклику до нових технологічних рішень.

Проблема полягає в тому, щоб розробити систему, яка могла б не просто виконувати вказівки кінцевого користувача, але й самоосвіту та працювати в динамічному середовищі програмування. Інтеграція OpenAI і Google API в розробку такого програмного забезпечення є принципово новим підходом, що надає широкий спектр можливостей для оптимізованої розробки ІБ.

Важливість цієї роботи визначається швидким зростанням індустрії програмного забезпечення та потребою в чомусь свіжому та новому. Отже, магістерська робота

виходить за рамки простого теоретичного внеску в дослідження ІС і це має важливе практичне значення для розробки програмного забезпечення.

Таким чином, актуальність нашої роботи полягає в тому, щоб задовольнити потреби індустрії програмування, що швидко розвивається, і ставить постійні вимоги до новизни. Таким чином, ця магістерська робота не тільки додає академічну цінність, що стосується сфери ІС через теорію, але має величезну практичну цінність для практики розробки програмного забезпечення.

Об'єктом дослідження є створення інформаційних систем, орієнтованих на автоматичне програмування для підвищення ефективності розробки програмного забезпечення. Це дослідження, зокрема, розглядає об'єкт як проблему для OpenAI та інших генеративних моделей, які вже доступні і здатні генерувати високоякісне програмне забезпечення, яке обробляє запити користувачів та оптимізує їх.

Предметом дослідження аналіз методів швидкого розвитку за функціональністю мінімуму інтерактивних інтерфейсів для користувачів та спрощення процесів програмування для програмістів у системі. Дослідження орієнтоване на аналіз методів інтеграції API, процесів адаптації до змін у системних бібліотеках та розробку асистента підтримки мов програмування та фремворків.

Метою проекту є розробка та оцінка продуктивності генерації коду за допомогою веб-застосунку з використанням OpenAI та інтегрованого в Google API – сучасного API, який за допомогою швидких методів розробки оптимізує наявні проблеми програмування. У результаті програма може створювати більш точні та ефективні можливості програмування; оновити інформацію про існуючі бібліотеки та їх базовий код і дати правильні відповіді щодо швидкості та якості програмного забезпечення на даний момент. Основна теоретична цінність пов'язана з аналізом можливостей штучного інтелекту в генерації коду, а її практичний результат полягає у розробці робочого прототипу з вбудованою системою оцінки його ефективності.

РОЗДІЛ 1.
АНАЛІЗ СТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ

1.1. Аналіз історичного підгрунтя і розвитку засобів генерації коду

Історія генерації коду є давньою і веде свій початок із ранніх років розвитку інформатики. Спочатку програмування здійснювалося на дуже низькому рівні безпосередньо до машинного коду — важким і трудомістким способом. Рішення такого роду вперше були реалізовані після 1950-х років, коли мови високого рівня, включаючи FORTRAN (як показано на малюнку 1.1), були визначені як перші спроби відокремитися від машинної логіки та полегшити програмування [1]. Такі нововведення, очевидно, стосуються переходу від ручного кодування до автоматизму у вигляді створення програмного забезпечення.

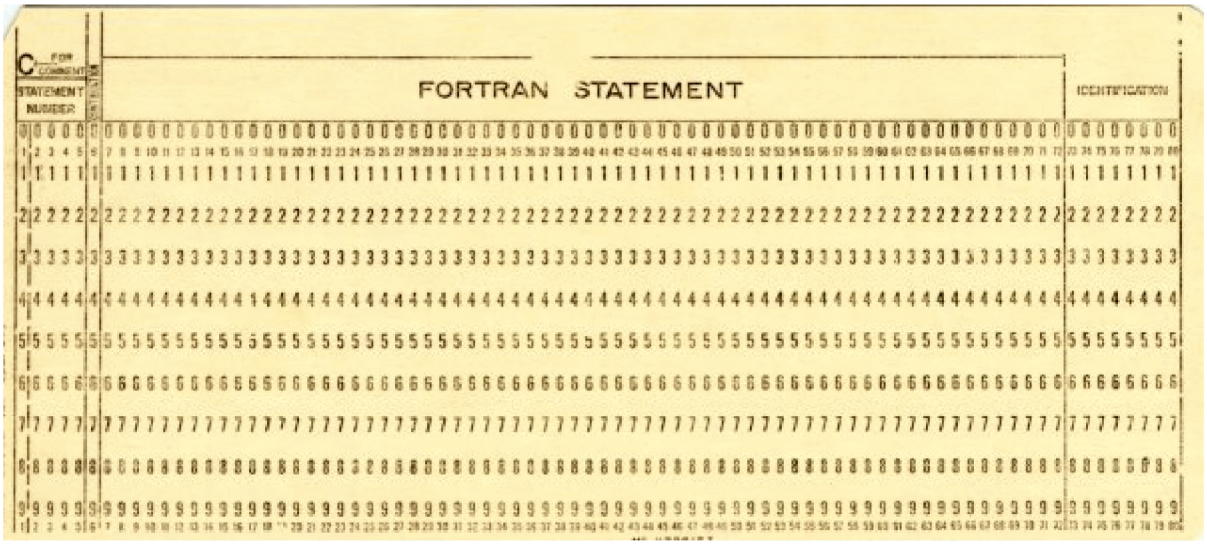

Рисунок 1.1 – Застосування FORTRAN у 1950-х роках.

У 1960-х і 1970-х роках, із розвитком структурованого програмування, було введено деякі нові парадигми, які дозволили програмістам краще справлятися зі складністю програм. Це стало одним із основних підходів до сучасного проектування та методології генерації коду. З появою об’єктно-орієнтованого програмування у 1980-х роках з’явилися нові можливості для створення коду [2]. Використання класів, імітації та інкапсуляції сприяло створенню набагато більш гнучких і масштабованих систем [2,3].

1990-ті характеризуються прискоренням виходу на ринок завдяки автоматизації генерації коду в перших інтегрованих середовищах розробки (IDE). Стандартними є, наприклад, об'єктно-орієнтоване програмування та шаблони проектування [4].

Завдяки нещодавнім досягненням штучного інтелекту генерація коду була автоматизована. Зокрема, алгоритми машинного навчання та нейронні мережі відкрили нас у царині автоматичної генерації коду на льоту, який піддається складним вимогам і змінам під час розробки [5]. Усе це досягається за допомогою базових можливостей розуміння природної мови, які перетворюють повсякденне людське спілкування в код, як у випадку OpenAI Codex [6].

Сучасне створення коду тепер тісно пов'язане з використанням численних API для розширення функціональності програмного забезпечення за допомогою інтеграції з багатьма онлайн-сервісами. Google API є прикладом такого інструменту, який пропонує численні набори послуг, від аналітики до машинного перекладу, які можна включити під час розробки програми [7].

Таким чином, загальний аналіз історичного контексту еволюції методологій генерації коду представляє повну історію циклу еволюції розробки програмного забезпечення, від початкових систем автоматизації до сучасних рішень штучного інтелекту, забезпечуючи більш складні та інноваційні методи.

1.2 Огляд сучасних тенденцій, методів і підходів у процесі генерації коду

Сучасні технології генерації коду зазнали значного розвитку завдяки штучному інтелекту, зокрема технологіям обробки природної мови (NLP). Зовсім недавно великий ажіотаж викликав інструмент GitHub Copilot, який базується на технології OpenAI Codex і є одним із найдосконаліших на сьогодні. Це також дозволить інструменту генерувати більше, ніж перекладати текстові описи в робочий код, таємно пропонуючи та зважуючи альтернативи для видалення надмірності та посилення переважаючого коду в контексті та параметрах [8].

IDE також стають розумнішими, надаючи різноманітні функції, які значно полегшують розробку програмного забезпечення. Сучасні IDE надають розширені

можливості для автоматизованого аналізу коду, рефакторингу та інших речей, які потребують глибокого розуміння структури програми [9].

API, наприклад API хмарного бачення або API природної мови, відкривають розробникам нові горизонти для легкої інтеграції складних функцій ШІ у свої програми, як показано на рис. 1.2. Використання цих API може зменшити ручне кодування, автоматизувати тести та документацію та покращити досвід користувача [10].

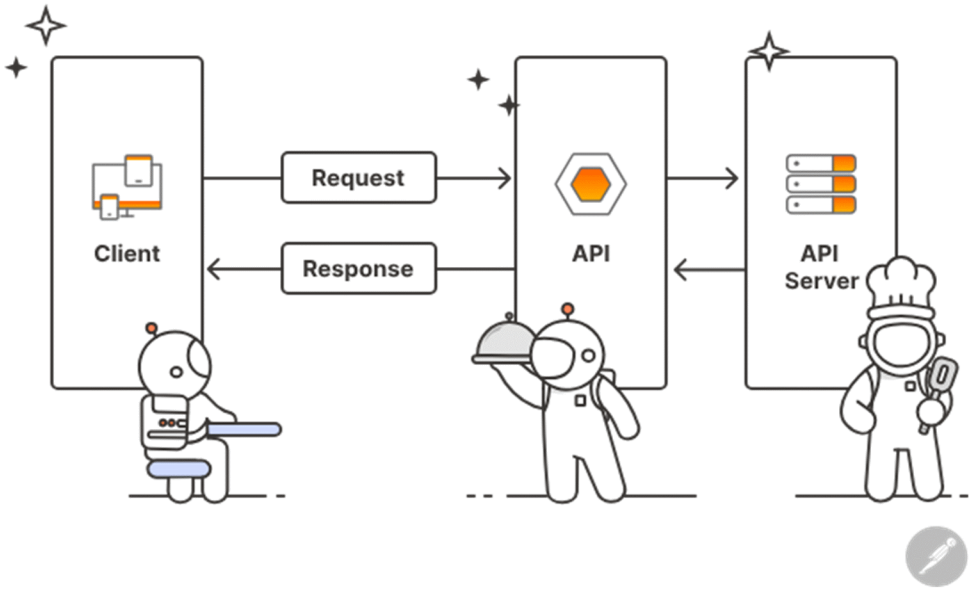

Рисунок 1.2 – Схема представлення функцій API

Значна частина еволюції в оптимізації та створенні коду спрямована на застосування машинного навчання. За допомогою навчання з підкріпленням і на додаток до цього навчання передачі можна створювати моделі, які можуть переносити накопичений досвід з однієї роботи на роботу на іншій [11].

Слід підкреслити, що співпраця в області відкритого коду значно підвищує якість і швидкість розробки засобів генерації коду. Спільноти розробників навколо таких платформ, як GitHub, роблять свій внесок у вдосконалення техніки генерації коду, ділячись своїми програмами, виправленнями та плагінами.

Для комплексної оцінки ефекту та можливостей сучасних методів генерації коду необхідний огляд наукових публікацій, технічної документації та звітів користувачів про їх застосування. Аналіз цих матеріалів показує, що це сфера, яка швидко

розвивається, і існує нереалізований потенціал штучного інтелекту, який може стати основою для наступного покоління систем автоматизації розробки.

1.3 Загальний огляд можливостей OpenAI і Google API для створення коду

Індустрія розробки програмного забезпечення дійсно змінилася з появою інструментів автоматичного створення коду. Ці функції тепер очолюються API OpenAI і Google, які надають розробнику значно розширені можливості створення та оптимізації.

OpenAI Codex є особливо вражаючим прикладом прогресу в цьому напрямку. За допомогою передових методологій глибокого навчання Codex може перетворювати інструкції, визначені природною мовою, у оперативний код, долаючи межі для програмістів і розробників. Цей API підвищує не тільки продуктивність, але й полегшує навчання користувача, показуючи найкращі практики та можливі рішення завдань програмування (рис. 1.3)[13].

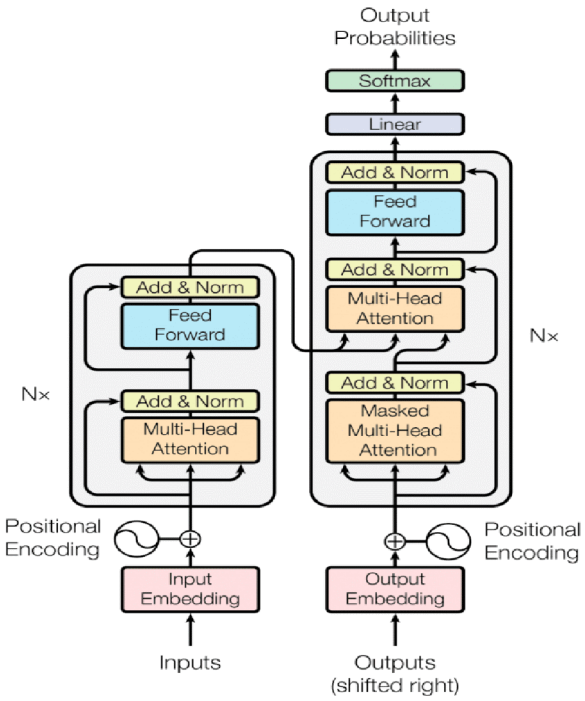

Рисунок 1.3 – Схематичне зображення реалізації OpenAI Codex

Код початкового завантаження (бут-стреп код) генерується швидко, і ця технологія корисна у великих проектах, тоді як код початкового завантаження

потрібно швидко генерувати для нових компонентів або модулів. Це може допомогти програмістам-початківцям навчатися за допомогою вказівок і відгуків [14].

Google API дозволяє надавати широкий спектр послуг у розробці веб-додатків, включаючи хмарні сервіси, аналітику, інтелектуальний пошук та багато іншого (рис. 1.4). Ці ресурси можуть бути використані для модернізації функціональності та інтерактивності додатків і надання їм підтримки передових технологій обробки даних та інтеграції штучного інтелекту [15]. я [14].

Google API пропонує широкий спектр можливостей (рис. 1.4) для розробки веб-додатків, включаючи інтеграцію з хмарними сервісами, аналітикою, інтелектуальними 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].

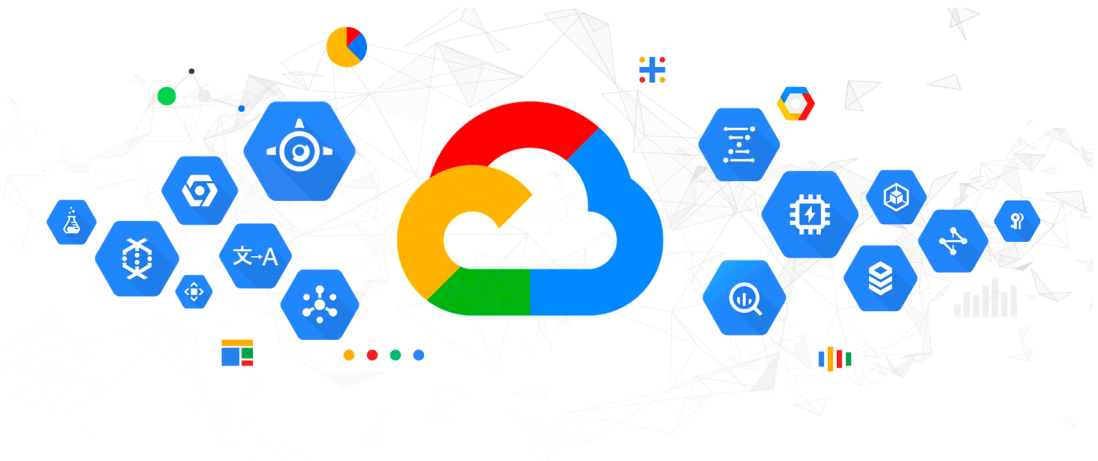

Рисунок 1.4 – Візуалізація сервісів Google API

Інтерфейси API Google проклали шлях до використання машинного навчання для вивчення поведінки користувачів і тонкого налаштування UX/UI, що помітно поширюється на якість кінцевого продукту. Іншим внеском є створення адаптивного дизайну, який реагує на потреби користувачів у режимі реального часу.

Поєднання цих API може заощадити багато часу під час розробки, зробити менше помилок і покращити програмні коди. Таким чином, це сприятиме швидкому створенню прототипів та ітераційній роботі для підтримки своєчасного тестування та переробки у відповідь на динамічні потреби ринку [17].

Підводячи підсумок, важливо зазначити, що розробка інструментів генерації коду вимагатиме постійних оновлень і вдосконалень API від OpenAI і Google, участі спільноти розробників і відгуків користувачів на майбутнє. Таке динамічне середовище сприяє інноваціям і новим методологіям програмування, які можуть кардинально змінити майбутнє розробки програмного забезпечення.

1.4 Визначення основних завдань дослідження

У рамках даної магістерської роботи обгрунтовано наступні основні завдання:

По-перше необхідно здійснити огляд сучасних технологій генерації коду, провести аналіз та порівняння існуючих методів генерації коду, зокрема, сучасних засобів NLP, у контексті створення вихідного коду програм. Це буде зроблено на основі наукових статей у журналах і збірниках, а також за допомогою документації розробників від OpenAI і Google.

По-друге, давайте придумаємо теоретичну модель для системи генерації коду: тепер ми можемо інтегрувати можливості OpenAI NLP для інтерпретації текстових запитів і використовувати Google API для додаткових функцій, таких як синтаксичний аналіз або семантичне розуміння.

Третє завдання полягає у визначенні архітектурних вимог до веб-додатку. Це передбачає визначення необхідного стеку технологій, вибір мов програмування та фреймворків, а також планування інтерфейсу користувача на основі найкращих практик проектування UX/UI.

Розробка методології перевірки ефективності генерації коду. Оцінювання бажано поєднувати як кількісні, так і якісні показники, щоб оцінити, серед іншого, правильність, а також ефективність, масштабованість і час відгуку створеного коду.

Реалізація прототипу веб-додатку. Це в основному передбачає розробку прототипу, розташованого в межах встановленої архітектури та методології. Такий прототип має бути рудиментарним з точки зору можливостей генерації коду та повинен мати принаймні можливість приймати живі запити користувачів.

Експериментальна перевірка прототипу – шосте завдання. Рекомендуються різні сценарії використання для експериментів, які необхідно провести та підтвердити, чи здатна система функціонувати найкращим чином у різних умовах.

Здійснивши аналіз отриманих результатів і порівняння з існуючими рішеннями ми виконаємо сьоме завдання, що повинно допомогти порівняти розроблений прототип з існуючими рішеннями. Цей аналіз має виявити сильні сторони та недоліки прототипу в порівнянні з існуючими рішеннями, а також можливі шляхи подальшого розвитку.

По-восьме. Підготовка повної документації для веб-додатку: технічні характеристики; посібники для кінцевого користувача; описи алгоритмів і використання API.

По-девяте. Визначення перспективних напрямків майбутніх досліджень у світлі тенденцій, проблем і можливостей, що виникають під час проведення досліджень.

І по-десяте. Формулювання рекомендацій щодо використання веб-додатку в генерації коду для розробників програмного забезпечення та потенційних подальших застосувань у цілому контекст інформатики

Такі завдання визначать структуру дослідницького процесу та призведуть до реалізації передового веб-застосунку, який автоматизує процес програмування та відкриває нові горизонти для розробників програмного забезпечення.

Висновки до розділу 1

1. Процес генерації коду тісно пов'язаний із загальним розвитком розробки програмного забезпечення, як показав аналіз його історичного контексту. Примітивні автоматизовані системи, складні сучасні рішення штучного інтелекту та все, що між ними, характеризують еволюцію генерації коду і показують різні дослідження в галузі забезпечення все більш складних та інноваційних підходів.

2. Було розглянуто сучасні форми генерації коду та досліджено найбільш часто використовувані сучасні інструменти. Проведено дослідження щодо можливості інтеграції цих інструментів в інтегроване середовище розробки.

3. Було досліджено основні можливості фреймворків як OpenAI так і Google API, у результаті чого було складено короткий список основних інструментів для подальшого використання у даній роботі.

РОЗДІЛ 2.
ОБГРУНТУВАННЯ, ВИБІР ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТАРІЮ ВИРІШЕННЯ ЗАДАЧІ

У другому розділі цієї магістерської роботи наголошується на важливому елементі розробки веб-додатків: вибір і обґрунтування основних інструментів і технологічної платформи. Це рішення визначає технічні можливості проекту, а також його майбутню ефективність, масштабованість і гнучкість.

Теоретичні основи та історичний контекст генерації коду були описані в першому розділі разом із оглядом поточних методологій та інструментів у цій галузі. Наступний крок передбачає дослідження та вибір конкретних технологій і фреймворків, які найбільш ефективно відповідають цілям і потребам веб-додатку, що розробляється. Інструменти, які є в доступі на сьогодні, включають різноманітні фреймворки зовнішньої та внутрішньої розробки, такі як React і Django, а також інтеграції API, такі як Google і OpenAI.

У цьому розділі подано ретельну критику та порівняльну оцінку різних технологічних рішень, окреслено їхні переваги та недоліки, а також розглянуто критерії вибору найбільш прийнятних інструментів. Встановлення чіткої архітектури веб-додатку має важливе значення, що передбачає ідентифікацію основних компонентів та їх взаємозв'язки. Окрім технічних міркувань, увага приділяється тому, як ці рішення впливають на продуктивність, безпеку та здатність програми розвиватися відповідно до майбутніх потреб. Таким чином, мета цього розділу полягає в тому, щоб закласти міцну основу для поточної розробки та демонстрації веб-додатку, гарантуючи, що вибрані інструменти та технології відповідають високим стандартам інновацій, гнучкості та ефективності.

2.1 Огляд доступних для використання інструментів та платформ для розробки веб-застосувань.

У сучасній ситуації, коли цифрові технології швидко розвиваються, веб-застосунки стали важливими для нашого повсякденного життя. Ці програми

виконують цілий ряд функцій, від оптимізації бізнес-операцій до надання розваг. Створення таких застосунків вимагає від розробників глибокого розуміння сучасних технологій та інструментів. У цьому розділі будуть розглянуті фундаментальні інструменти та методи, необхідні для ефективної розробки веб-застосунків [19].

Для початку дуже важливо зрозуміти концепцію, згідно з якою розробка веб-застосунків охоплює як інтерфейс (сторона клієнта), так і сервер (сторона сервера). Для інтерфейсу зазвичай використовуються такі технології, як HTML, CSS і JavaScript, а також сучасні фреймворки та бібліотеки, такі як React або Angular. Ці ресурси дозволяють створювати інтерактивні та візуально привабливі користувальницькі інтерфейси, які покращують роботу користувача.

І навпаки, серверні мови програмування та фреймворки є важливими для серверної частини, включаючи Python із Django, Ruby on Rails або Node.js. Ці технології сприяють ефективній обробці запитів користувачів, управлінню базами даних та інтеграції різних веб-служб.

Важливим аспектом сучасної веб-розробки є підключення до зовнішніх API, якими в даному випадку є OpenAI. Ці служби надають розробникам доступ до надійних інструментів, включаючи обробку мови та машинне навчання, що значно покращує функціональні можливості програм, що розробляються.

Вирішальний компонент розробки веб-застосунків, це інтерфейсна розробка, яка значно впливає на те, як користувачі взаємодіють із програмою. Ця область охоплює дизайн елементів, з якими користувачі стикаються та з якими взаємодіють у своїх браузерах, включаючи веб-сторінки, інтерфейс користувача та анімацію. Основні технології, які використовуються у розробці інтерфейсу, включають HTML для структурування веб-сторінок, CSS для візуального стилю та JavaScript для забезпечення інтерактивності.

Сучасні методи розробки інтерфейсу включають такі фреймворки та бібліотеки, як React, Angular і Vue.js. Пропонуючи готові компоненти та шаблони коду для повторного використання, ці інструменти спрощують процес розробки, підвищуючи ефективність і забезпечуючи більшу узгодженість інтерфейсів (рис. 2.1). Крім того, ці інфраструктури сприяють реактивному програмуванню та компонентному підходу,

забезпечуючи краще керування станом програми та покращуючи взаємодію між компонентами.

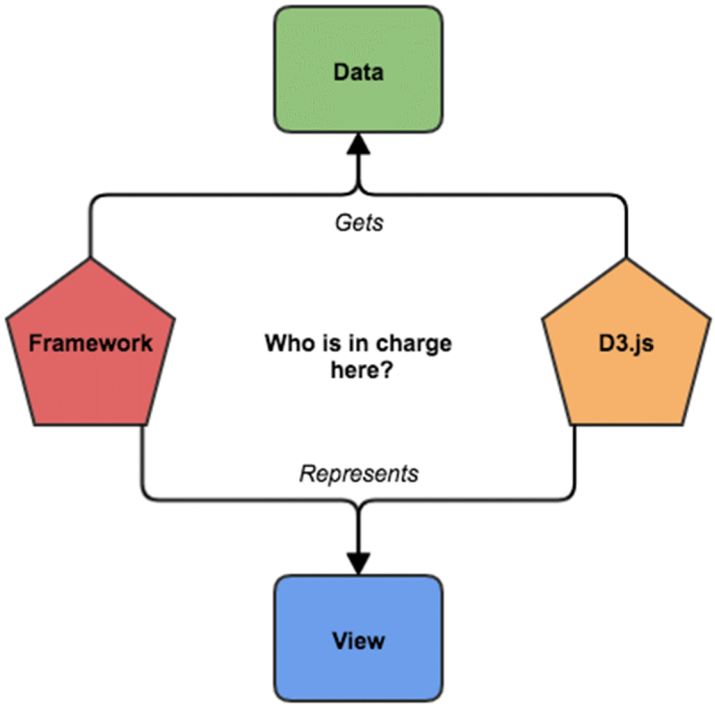

Рисунок 2.1 – Інтеграція Angular/Vue.js

Вдалий і лаконічний дизайн відіграє вирішальну роль у розробці інтерфейсу, гарантуючи оптимальний досвід перегляду та взаємодії з веб-сторінкою на різних пристроях, включаючи мобільні телефони та комп'ютери. Використовуючи CSS і гнучкі сітки, стає можливим розробляти дизайни, які легко адаптуються до різних розмірів екрана.

Зрештою, наголос у розробці інтерфейсу сьогодні лежить на покращенні взаємодії з користувачем (UX) та користувальницького інтерфейсу (UI), що гарантує, що додатки інтуїтивно зрозумілі та зручні. Це охоплює низку елементів, від інформаційної архітектури до візуального дизайну, гарантуючи, що додатки не тільки функціональні, але й візуально привабливі та прості для навігації. Після цього ми розглянемо три найпопулярніші фреймворки.

Розроблений Facebook, React є одним із найпоширеніших фреймворків JavaScript для створення інтерфейсів користувача. Використовуючи методологію на основі

компонентів, він дозволяє розробникам створювати великі веб-додатки, здатні оновлювати дані без необхідності перезавантажувати сторінку. Інфраструктура значно спрощує створення інтерактивних компонентів, пропонуючи реактивні та ефективні оновлення. У своїй екосистемі такі інструменти, як Redux, допомагають керувати станом програми, а React Router полегшує навігацію [20].

Angular, створений Google, служить фреймворком і платформою для розробки клієнтських додатків, що використовують HTML і TypeScript. Він надає розробникам низку інструментів для створення програм, які працюють на кількох платформах, включаючи веб-браузери та мобільні пристрої. Завдяки таким функціям, як двостороннє зв'язування даних, впровадження залежностей і надійний набір функцій для обробки маршрутизації, форм і додаткових елементів, Angular надає широкі можливості [21].

Vue.js служить прогресивним фреймворком для JavaScript, розробленим спеціально для створення інтерфейсів користувача. Його легкість інтеграції з різними проектами та бібліотеками позиціонує його як ідеальний варіант для створення складних односторінкових програм (SPA). Відомий своєю простотою та гнучкістю, Vue.js надає розробникам інтуїтивно зрозумілу архітектуру разом із реактивними компонентами. Крім того, він включає такі інструменти, як Vue Router для маршрутизації та Vuex для управління станом, що робить його повним рішенням для розробки інтерфейсу [22].

Розробка бекенда є ключовим елементом веб-додатків, надаючи основні функції сервера. Це охоплює серверну логіку, керування базами даних, інтеграцію із зовнішніми системами, а також обробку та зберігання даних.

Розробка бекенда широко використовує такі мови програмування, як Python, Ruby, Java та JavaScript (Node.js), кожна з яких пропонує різні структури та інструменти. Наприклад, такі фреймворки, як Django для Python і Express для Node.js, надають розробникам організовані шаблони та ресурси для ефективного створення безпечного та масштабованого коду на стороні сервера. Крім того, розробка бекенда передбачає взаємодію з системами керування базами даних, включаючи MySQL,

PostgreSQL або MongoDB, щоб гарантувати ефективне зберігання та маніпулювання даними.

У розробці серверної частини забезпечення надійних і стабільних веб-додатків залежить від критичних аспектів безпеки та оптимізації продуктивності. Це охоплює захист від різноманітних загроз, включаючи впровадження SQL, міжсайтовий сценарій (XSS) і додаткові форми атак.

Python, який славиться своєю ясністю та ефективністю, віддають перевагу багатьом розробникам для розробки бекенда. У поєднанні з Django, надійною високорівневою структурою, він перетворюється на потужний ресурс для швидкого створення веб-додатків. Django, відомий своїм підходом «батарейки в комплекті», пропонує готові рішення для різних типових дій веб-розробки, включаючи автентифікацію, сеанси, адміністрування тощо. Ця характеристика робить Django ідеальним для створення безпечних і масштабованих веб-додатків, що призводить до значної економії часу для розробників [23].

Node.js використовує JavaScript на стороні сервера, що дозволяє розробникам використовувати одну мову програмування як для кодування на стороні клієнта, так і на стороні сервера. Такий підхід не тільки спрощує процес розробки, але й підвищує ефективність. Node.js, що характеризується керованою подіями архітектурою та неблокуючим вводом-виводом, особливо добре підходить для створення масштабованих мережевих додатків, зокрема веб-серверів і програм реального часу. Крім того, Node.js може похвалитися великою екосистемою модулів через NPM, що сприяє його гнучкості та зручності [24].

Java входить до числа найулюбленіших мов програмування, визнаних своєю безпекою, масштабованістю та стабільністю, що робить її поширеним вибором для розробки серверної частини в корпоративних налаштуваннях. Завдяки надійній архітектурі та широкому набору бібліотек і фреймворків, включаючи Spring і Hibernate, Java дає змогу розробникам створювати надійні та високопродуктивні серверні програми. Віртуальна машина Java (JVM) сприяє значному ступеню переносимості та ефективності, позиціонуючи Java як кращий варіант для

різноманітних веб-додатків, що обслуговують будь-яке: від невеликих стартапів до великих корпоративних систем [25].

Хмарні технології змінили розгортання IT-ресурсів компаніями та керування ними. Ці технології дозволяють зберігати дані та працювати з додатками на віддалених серверах у хмарі, пропонуючи такі переваги, як гнучкість, масштабованість та економічна ефективність (рис. 2.2) [26].

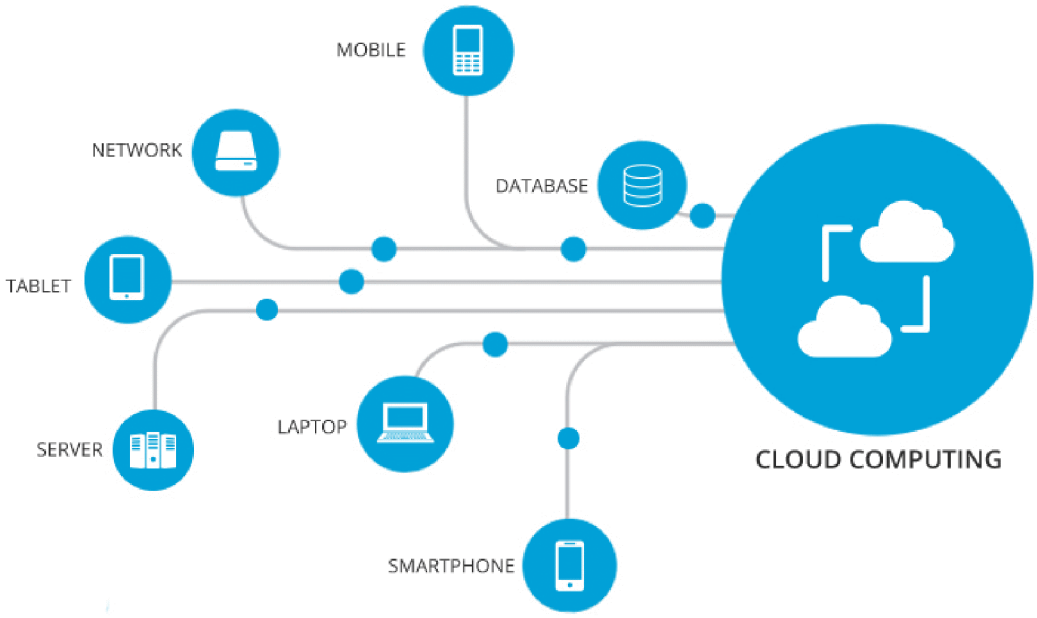

Рисунок 2.2 – Візуалізація процесу управління ресурсами за допомогою хмарних технологій

Amazon Web Services (AWS), відомий постачальник хмарних послуг від Amazon, пропонує широкий спектр продуктів і послуг. Сюди входять обчислювальні можливості (наприклад, EC2), рішення для зберігання (S3), параметри баз даних (RDS), машинне навчання, інструменти IoT та додаткові пропозиції. Завдяки репутації високої надійності та безпеки, AWS став улюбленим варіантом для компаній, починаючи від стартапів і закінчуючи великими підприємствами. Її послуги дозволяють розробникам ефективно масштабувати і контролювати інфраструктуру з мінімальними зусиллями [27].

Іншим значним конкурентом у сфері хмарних технологій є Google Cloud, який представляє низку сервісів, таких як Google Compute Engine, Google Cloud Storage та BigQuery. Відомий своєю надійною аналітикою та функціями машинного навчання,

він також легко інтегрується з численними службами Google. Розробники оснащені Google Cloud необхідними інструментами для створення, тестування та розгортання програм в інфраструктурі Google, що гарантує як високу продуктивність, так і надійність [28].

Ці два постачальники хмарних послуг дозволяють компаніям покращити управління ресурсами та зменшити витрати, пропонуючи послуги «оплата за використання», які підходять для широкого спектру додатків, включаючи будь-яке: від веб-хостингу до складних обчислювальних операцій. Ключові практики сучасної розробки програмного забезпечення, відомі як CI/CD (скорочення від Continuous Integration і Continuous Delivery/Deployment), зосереджені на автоматизації процесів розробки для підвищення ефективності та якості коду.

Безперервна інтеграція (CI) означає автоматичне поєднання коду, який розробники регулярно додають до спільного репозиторію, як правило, щодня. Цей процес дозволяє швидко виявляти та розв'язувати конфлікти, таким чином зберігаючи стабільність бази коду. Крім того, CI часто включає автоматизоване тестування, який гарантує, що новий код не порушить існуючу функціональність.

Розширення Continuous Integration, Continuous Delivery працює на тій передумові, що код, який успішно переміщається по конвеєрах CI, автоматично розгортається в тестовому або робочому середовищі. Такий підхід гарантує, що програмне забезпечення завжди готове до випуску, мінімізуючи затримку між створенням коду та його використанням кінцевими користувачами.

Безперервне розгортання стосується автоматизації наступної фази процесу, коли кожна зміна, яка успішно впроваджує всі фази конвеєра CI/CD, автоматично випускається у виробниче середовище. Це гарантує постійний потік оновлень для кінцевих користувачів, мінімізуючи час і зусилля, необхідні для ручного розгортання.

CI/CD дозволяє командам розробників досягати високої частоти випусків із високою надійністю, одночасно підвищуючи ефективність робочих процесів розробки та мінімізуючи ризики, пов'язані з ручним розгортанням. Оцінка інструментів і технологій має вирішальне значення для визначення архітектури та

підходу до розробки веб-додатку, впливаючи на його продуктивність, функціональність і здатність задовольняти вимоги користувачів [29].

2.2 Критерії та обгрунтування вибору інструментів для розробки веб-застосунку

Вибір відповідних інструментів і технологій є важливим для успіху будь-якої розробки програмного забезпечення. На цей вибір впливають різні фактори, такі як конкретні потреби проекту, навички команди, бюджетні обмеження та терміновість виходу на ринок.

Вирішальним фактором, який слід враховувати при виборі інструментів, є їх сумісність з OpenAI API. Це важливо, оскільки основні можливості — зокрема, розробка кодових помічників і рекомендацій щодо коду — залежатимуть від інтеграції з цим API. Використання офіційних бібліотек для інтеграції сприятиме безперебійній роботі програми та надасть доступ до складних функцій штучного інтелекту та машинного навчання, які пропонує OpenAI API. OpenAI надає дві бібліотеки для взаємодії зі своїм API, одну розроблену на Python, а іншу на Node.js, обидві мають однакову назву: openai [30].

Крім того, обраний стек розробки повинен бути не тільки ефективним і продуктивним, але і адаптованим. Код, отриманий із цього стеку, має бездоганно вносити зміни в OpenAI API, який швидко розвивається завдяки частим оновленням і новим функціям. Яскравою ілюстрацією цього є функція помічників, яка наразі знаходиться в стадії бета-версії та, ймовірно, незабаром буде переведена на робочу версію. Отже, інтеграція помічників у бібліотеку OpenAI може значно відрізнятися від поточного стану. Це вимагає, щоб набір інструментів був модульним і міг швидко оновлюватися, зберігаючи загальну функціональність програми.

Важливим фактором, який слід враховувати, є темпи розвитку. Щоб дотримуватись суворих термінів виконання проекту, фреймворки та мови програмування, які підвищують швидкість розробки, є життєво важливими. Це охоплює інструменти, які сприяють швидкому створенню прототипів та ітераційній розробці. Не менш важливим є вибір технологій, які дозволяють значною мірою

повторне використання коду та можуть бути бездоганно інтегровані в різні компоненти проекту.

Додаток має бути ефективним, що означає, що вибрані інструменти забезпечують максимальну продуктивність у кінцевому результаті. Це включає в себе швидкість відгуку програми, ефективну обробку даних і зменшення затримок. Пріоритет оптимізації продуктивності на кожному етапі розробки — від запитів до бази даних до інтерфейсного рендерингу — має важливе значення для забезпечення першокласної взаємодії з користувачем.

Враховуючи всі вищезазначені вимоги та специфікації, вам слід вибрати одну з двох мов для серверної частини: Python і Node.js, обидві з яких пропонують підтримку існуючої бібліотеки для взаємодії з OpenAI API. Для реалізації веб-додатку остаточно було обрано Python [30].

Python відомий своєю підтримкою об'єктно-орієнтованого програмування (ООП), що дозволяє розробникам створювати модульний код, який легко підтримувати, використовуючи класи. Ця парадигма програмування, зосереджена навколо об'єктів і класів, сприяє ефективній організації та управлінню кодом, забезпечуючи високий рівень гнучкості та ясності. Крім того, Python може похвалитися широким набором фреймворків і пакетів, які задовольняють різноманітні вимоги до програмування — від веб-розробки до наукових досліджень — пропонуючи практично безмежні можливості для реалізації різноманітних проектів. З точки зору швидкості, хоча Python, можливо, не вважається найшвидшою мовою програмування щодо виконання коду, його продуктивність і швидкість розробки часто вважаються оптимальними, зокрема через його простий синтаксис і надійну підтримку спільноти. Нарешті, Python вважається однією з найкращих мов для обробки тексту завдяки потужним бібліотекам, розробленим для обробки природної мови та аналізу даних, що робить його чудовим вибором для проектів, пов'язаних із машинним навчанням, обробкою мови та аналізом даних.

Вибраний фреймворк складається з Django та Django REST framework. Django, відомий своєю ефективністю та універсальністю, є надійною системою веб-розробки на основі Python. Він дає змогу розробникам швидко створювати безпечні та

масштабовані веб-додатки, дотримуючись філософії «батареї включені» у своїй архітектурі. Важливим аспектом цієї структури є панель адміністратора Django, яка пропонує інтуїтивно зрозумілий графічний інтерфейс користувача (GUI) для легкого керування даними бази даних, усуваючи потребу в додатковому кодуванні. Іншим важливим компонентом Django є його об'єктно-реляційний картограф (ORM), який дозволяє розробникам взаємодіяти з базою даних через високорівневий абстрактний API. Завдяки значному спрощенню взаємодії з базою даних Django ORM дозволяє виконувати складні запити та маніпулювати даними без необхідності писати необроблений код SQL. Це підвищує інтуїтивність і безпеку роботи з базами даних, мінімізуючи ймовірність помилок і вразливостей. Крім того, платформа Django REST (DRF) доповнює Django, пропонуючи потужні та гнучкі інструменти для побудови веб-інтерфейсів API, визнані своєю масштабованістю та сумісністю. з різними стилями API. Перетворення даних між складними типами, наприклад екземплярами моделі, і форматами, придатними для передачі HTTP, такими як JSON або XML, керується його серіалізаторами, які спрощують обробку даних між сервером і клієнтами. У поєднанні Django та платформа Django REST утворюють надійний набір інструментів для сучасної веб-розробки, надаючи розробникам необхідні ресурси для швидкого й ефективного створення як API, так і веб-сайтів. Їх комплексна екосистема, ретельно розроблена документація та залучена спільнота сприяють їхньому статусу як одного з найбільш улюблених варіантів для розробників у всьому світі.

Для цього проекту вибрано систему керування базами даних PostgreSQL, стратегічне рішення завдяки бездоганній інтеграції та сумісності з Django. PostgreSQL, відомий своєю надійністю, масштабованістю та надійними функціями, є однією з найпопулярніших систем керування реляційними базами даних із відкритим кодом. Вона чудово підтримує складні запити та транзакції, забезпечує міцну інтеграцію з різними мовами програмування та визнана однією з найбільш розширюваних і стандартизованих баз даних. Значною перевагою використання PostgreSQL разом із Django є проста інтеграція між двома платформами. Django ORM (Object-Relational Mapper) надзвичайно добре взаємодіє з PostgreSQL, сприяючи

плавному обміну даними між веб-програмою та базою даних. Це поєднання дозволяє розробникам використовувати розширені функції PostgreSQL, включаючи повнотекстовий пошук, транзакції на рівні бази даних і підтримку різноманітних типів даних, зберігаючи при цьому простоту та ефективність, яку пропонує Django ORM. Коли PostgreSQL поєднується з Django та Django REST Framework, це створює міцну, надійну та масштабовану основу для розробки веб-додатків. Ця технологічна синергія гарантує, що база даних може вміло керувати великими обсягами даних і складними запитами, а також надаючи потужні інструменти для розробки серверної та зовнішньої частини.

Рішення використовувати React.js для розробки інтерфейсу випливає з його статусу як одного з найпотужніших і широко використовуваних фреймворків JavaScript для створення інтерактивних веб-інтерфейсів. Створений Facebook, React.js славиться своєю ефективністю та адаптивністю, що дозволяє створювати динамічні та чуйні інтерфейси користувача. В основі React лежить компонентний підхід, який заохочує розробку повторно використовуваних і добре структурованих блоків коду, тим самим підвищуючи загальну ефективність і організацію процесу розробки.

Ця комбінація, яка часто поєднується з Django та Django REST Framework, забезпечує комплексний стек технологій для створення сучасних веб-додатків. Django служить надійним серверним фреймворком, який пропонує стабільну, безпечну та масштабовану архітектуру, тоді як Django REST Framework спрощує створення API для обміну даними між серверним і зовнішнім інтерфейсом. І навпаки, React.js використовується для розробки динамічного та інтерактивного інтерфейсу, який взаємодіє з API, створеним за допомогою DRF.

Ця повна інтеграція дозволяє розробникам використовувати переваги обох технологій: Django забезпечує надійність, безпеку та масштабованість для серверної частини, тоді як React забезпечує гнучкість і швидкість реагування для зовнішньої частини. Такий підхід гарантує високий ступінь взаємодії між користувачами та веб-додатком, що зрештою покращує як досвід користувача, так і ефективність розробки.

Після створення основного стека технологій для нашої веб-програми вибір архітектурного рішення стає важливим наступним кроком. У цьому випадку ми обрали мікросервісну архітектуру, використовуючи Django та React.js як окремі мікросервіси. Цей метод дозволяє розділити нашу програму на незалежні компоненти для серверної частини (Django) і зовнішньої частини (React.js), що дозволяє розробляти, розгортати та масштабувати кожен незалежно. Така домовленість підвищує гнучкість як у розробці, так і в управлінні, полегшуючи впровадження змін і оновлень.

Коли йдеться про хмарні рішення та інфраструктуру, вони мають важливе значення для підтримки та масштабування нашої веб-програми. Використання хмарних технологій, таких як Amazon Web Services (AWS), Google Cloud Platform (GCP) або Microsoft Azure, надає численні переваги. Ці платформи пропонують гнучкі та масштабовані варіанти розміщення, зберігання даних, керування базами даних, безпеки та різноманітних інших елементів інфраструктури. За допомогою хмарних служб ресурси можна легко регулювати відповідно до вимог програми, що є життєво важливим для підтримки стабільної продуктивності в періоди високого попиту. Крім того, хмарні платформи забезпечують високу доступність і надійність, розподіляючи ресурси між кількома географічними розташуваннями. Це дозволяє більш адаптивно керувати ресурсами, зводячи до мінімуму потребу в ручному нагляді за інфраструктурою. Розширені функції безпеки, надані постачальниками хмарних технологій, мають вирішальне значення для захисту даних і програм. Отже, впровадження архітектури мікросервісів через хмарні рішення та інфраструктуру забезпечує значну гнучкість, масштабованість і надійність, які є ключовими для успіху сучасної веб-програми.

Вибір Amazon Web Services (AWS) як нашої хмарної платформи для реалізації веб-програми пояснюється кількома переконливими перевагами, особливо важливими під час закритої бета-фази. AWS виділяється своєю простою та зрозумілою інтеграцією між різними службами, що підвищує ефективність і зручність управління ресурсами та розгортання. Значною перевагою є наявність безкоштовних екземплярів EC2, наданих AWS, які є ідеальним рішенням для запуску як серверної,

так і зовнішньої частини нашого веб-додатку. Ці екземпляри дозволяють нам ефективно тестувати та розвивати нашу програму, забезпечуючи необхідні ресурси без додаткових витрат. Крім того, використання RDS для керування базами даних дозволяє нам використовувати високоефективну обробку даних у хмарі, забезпечуючи не лише високу доступність і безпеку, але й легкість масштабування. Повна інтеграція EC2, RDS та інших сервісів AWS дозволяє нам зосередитися на розробці та оптимізації програми, а не витрачати час на керування інфраструктурою. Загалом, використання AWS у нашому проекті пропонує не лише економію коштів завдяки безкоштовним екземплярам, але й значний ступінь гнучкості та масштабованості, які є важливими для успішного бета-тестування та постійного розвитку веб-додатку [31-32].

Вирішальним елементом у створенні сучасних веб-додатків є впровадження практик безперервної інтеграції (CI) і безперервного розгортання/доставки (CD). Завдяки автоматизації процесів тестування та розгортання CI/CD підвищує ефективність розробки, мінімізує ймовірність помилок і підвищує загальну якість кінцевого продукту. Безперервна інтеграція передбачає автоматичне тестування коду з кожним оновленням, гарантуючи, що нові модифікації не порушують функціональність існуючої системи. Ця стратегія полегшує швидке виявлення та виправлення помилок, значно скорочуючи час, необхідний як для розробки, так і для тестування. Після успішного завершення тестування Continuous Deployment/Delivery забезпечує автоматичне розгортання змін у робочому середовищі.

Це гарантує, що програма постійно оновлюється новими функціями та виправленнями, підвищуючи ефективність і адаптивність бізнес-операцій. Різноманітні інструменти CI/CD, включаючи Jenkins, GitHub Actions і GitLab CI/CD, пропонують численні варіанти автоматизації цих робочих процесів. GitLab CI/CD служить інтегрованим рішенням у GitLab, забезпечуючи пряме налаштування робочих процесів CI/CD, не покладаючись на зовнішні служби. Подібним чином GitHub Actions надає ці можливості безпосередньо в GitHub, дозволяючи користувачам створювати робочі процеси для автоматизації тестування додатків, створення та розгортання. Jenkins, інструмент із відкритим кодом, пропонує гнучку

конфігурацію процесів CI/CD за допомогою плагінів і сценаріїв. Завдяки інтеграції цих інструментів із хмарними рішеннями, такими як AWS, GCP або Azure, ефективність і масштабованість процесу розробки ще більше підвищуються. Хмарні платформи можуть надавати послуги для автоматизованого розгортання, моніторингу, управління ресурсами та безпеки, які доповнюють процеси CI/CD. Ця інтеграція забезпечує більш плавну та ефективну розробку веб-додатків, сприяючи швидкому розгортанню нових функцій, зберігаючи при цьому високий рівень безпеки та стабільності програми.

Вибір GitHub Actions for CI/CD у нашому проекті в поєднанні з Amazon Web Services (AWS) як нашою хмарною платформою створює потужну синергію, яка підвищує як ефективність, так і гнучкість нашого процесу розробки. Будучи провідним хмарним сервісом, AWS надає широкий набір інструментів і сервісів, які ідеально підходять для сучасних веб-додатків, охоплюючи рішення для обчислень, безпеки, зберігання даних тощо [33]. Інтегруючи AWS у нашу інфраструктуру разом із GitHub Actions, ми отримуємо такі переваги:

– GitHub Actions може легко підключатися до ряду служб AWS, включаючи EC2 для віртуальних серверів, S3 для зберігання та RDS для баз даних, серед інших. Ця інтеграція дозволяє автоматизувати завдання розгортання та керування інфраструктурою безпосередньо з GitHub.

- Автоматизація розгортання: за допомогою GitHub Actions можна автоматизувати розгортання додатків у різних середовищах AWS, що спрощує керування середовищами виробництва та тестування. – Масштабованість: AWS пропонує значну масштабованість, що дає змогу легко налаштовувати ресурси відповідно до вимог проекту. Автоматизація цих процесів за допомогою GitHub Actions гарантує плавне масштабування без ручних зусиль.

– Безпека: розширені функції безпеки, надані AWS, можна легко інтегрувати з робочими процесами CI/CD у GitHub Actions, гарантуючи безпеку як коду, так і програм. – Економічна ефективність: поєднання AWS із GitHub Actions може покращити оптимізацію витрат, сприяючи ефективному використанню ресурсів і моделі оплати за використання.

Інтеграція GitHub Actions з AWS створює надійну структуру для автоматизації та контролю за процесами розробки та розгортання, гарантуючи ефективність, масштабованість і підвищену безпеку для нашого веб-застосунку [34].

2.3 Планування архітектури додатку.

Вирішальним аспектом архітектури є інтеграція інтерфейсу та серверної частини за допомогою React та Django відповідно. Розгорнута на AWS EC2 вся програма має переваги високої доступності та масштабованості. Для зберігання даних використовується база даних RDS PostgreSQL, що забезпечує ефективне керування даними та оптимальну продуктивність.

Важливим аспектом процесу розробки є використання CI/CD, який підтримує GitHub. Вирішальним фактором є те, що процес CI/CD запускається автоматично, коли код надсилається до головної гілки (main) репозиторію GitHub. Ця процедура включає в себе автоматичне тестування коду, перевірку його стабільності та надійності, а після успішного завершення тесту розгортання оновленої версії коду в примірниках EC2. Це гарантує постійне оновлення програми з мінімальними збоями, таким чином зберігаючи високу продуктивність і стабільність обслуговування.

Таким чином, використовуючи поєднання AWS EC2, RDS PostgreSQL і злагодженого робочого процесу CI/CD, можна створити потужну та ефективну архітектуру для веб-додатку, який пропонує гнучкість, масштабованість і спрощене керування, як показано на рис. 2.3.

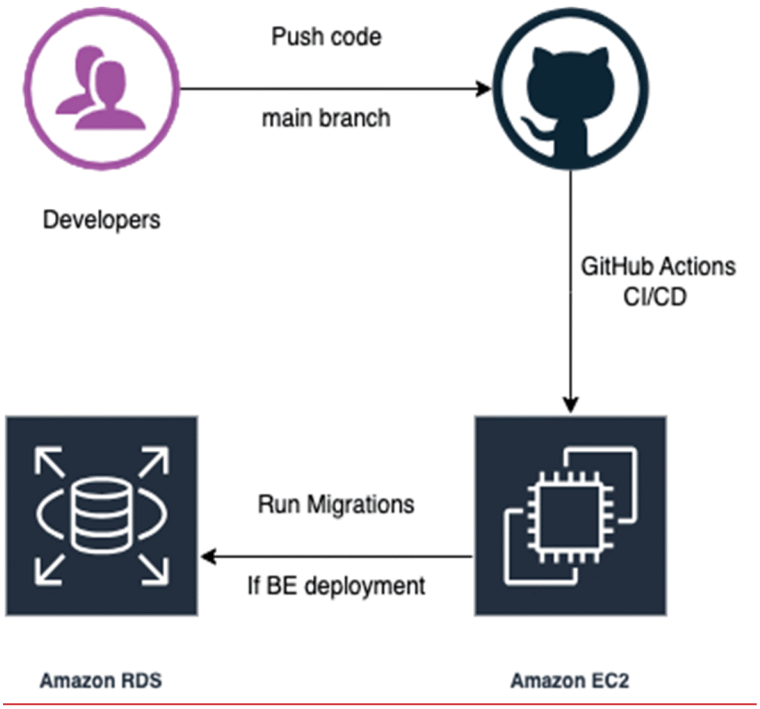

Рисунок 2.3 – Діаграма архітектури інфраструктури.

Ця архітектура веб-застосувань є прикладом сучасного, масштабованого та адаптованого програмного рішення, яке використовує React для інтерфейсу, Django разом із Django ORM для серверу, PostgreSQL як систему керування базами даних і OpenAI API як зовнішню службу.

Інтерфейс користувача, створений за допомогою бібліотеки React, забезпечує високопродуктивний інтерактивний інтерфейс. Він динамічно взаємодіє з серверною частиною за допомогою асинхронних HTTP-запитів, отримуючи дані у форматі JSON, що забезпечує безперебійну та оперативну взаємодію з користувачем.

Бекенд, створений на базі Django, пропонує потужне та адаптоване рішення, що дозволяє швидко розвивати бізнес-логіку на значному рівні абстракції. Завдяки Django ORM розробники отримують доступ до потужного ресурсу для роботи з базою даних PostgreSQL, представляючи зручний інтерфейс для запитів даних і маніпулювання ними без необхідності безпосередньо обробляти SQL. Такий підхід прискорює процес розробки та мінімізує ймовірність помилок.

PostgreSQL служить надійною, масштабованою та ефективною системою керування базами даних, яка вміє виконувати складні запити, забезпечуючи при

цьому цілісність даних. Це кращий варіант для програм, які вимагають як високої продуктивності, так і адаптивності під час обробки значних обсягів даних.

Інтеграція OpenAI API як зовнішньої служби покращує функціональність програми за допомогою штучного інтелекту. Ця інтеграція полегшує складні операції з обробки мови та генерації тексту, пропонуючи користувачам розширені функції, такі як автоматичне створення вмісту або генерація коду. Захищені запити HTTP, що виконуються серверною частиною, уможливлюють цю інтеграцію з API OpenAI, забезпечуючи контроль над використанням API та обробкою вхідних даних.

Загалом, цей тип архітектури ефективно поєднує в собі надійність, продуктивність та інновації, представляючи оптимальні практики для створення сучасних веб-додатків (рис. 2.4).

Рисунок 2.4 ілюструє архітектуру взаємодії, що включає інтерфейс, серверну частину, базу даних і OpenAI API.

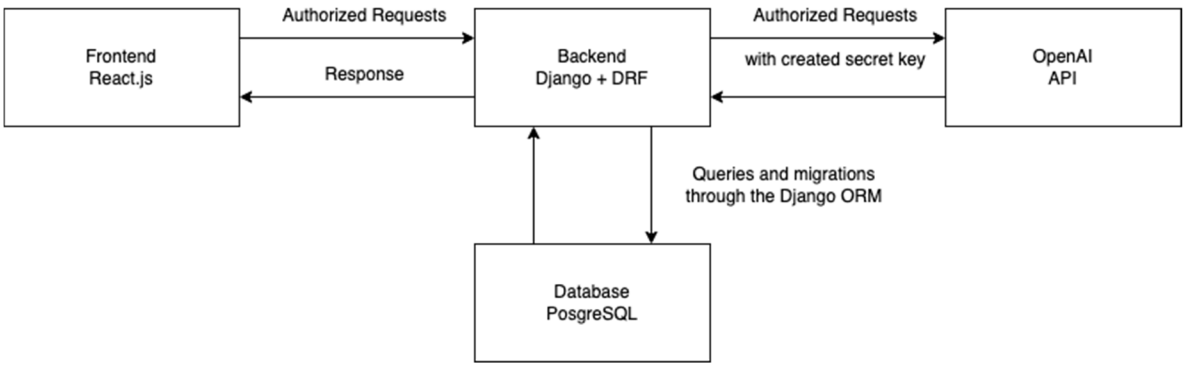

Рисунок 2.4 – Взаємодія фронтенду, бекенду та бази даних з OpenAI API

Крім того, важливим аспектом розробки архітектури проекту є формулювання схеми бази даних, представленої на малюнку 2.5.

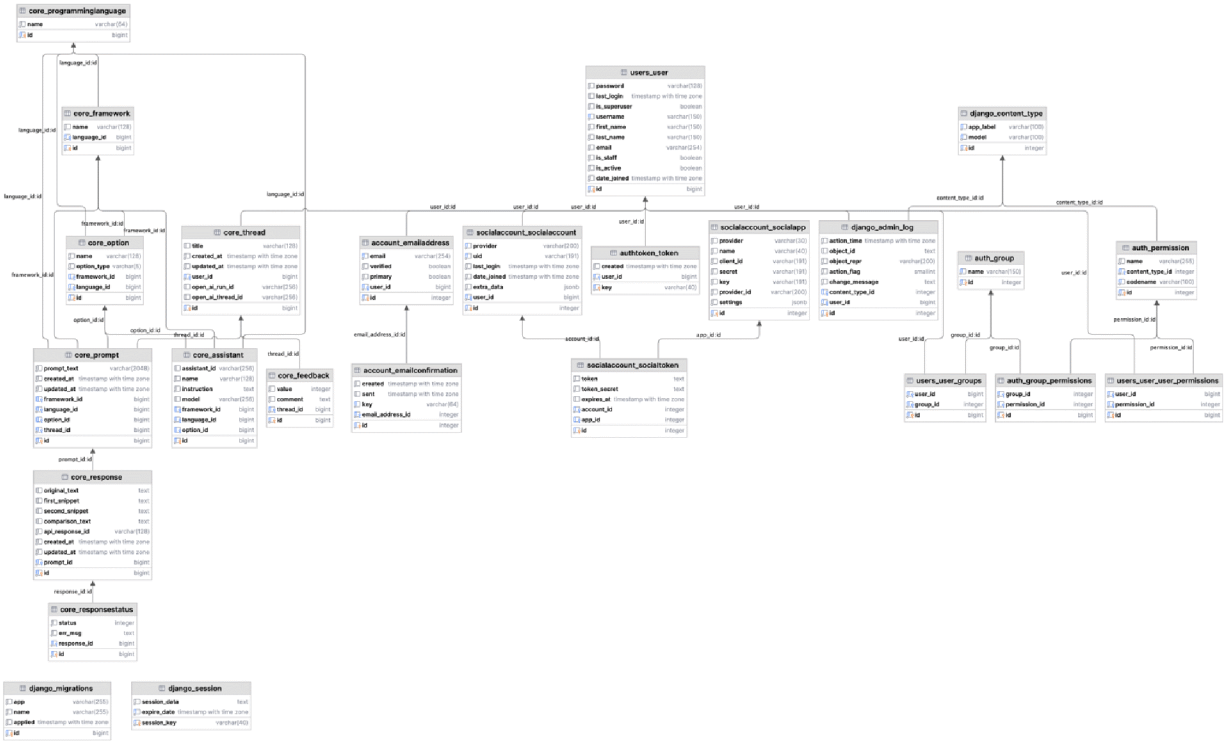

Рис. 2.5 Архітектура схеми проектованої бази даних

На основі обраних архітектурних рішень було розроблено веб-застосування згідно з тематикою нашої магістерської роботи.

2.4 Огляд та аналіз методів інтеграції веб-застосунку з API OpenAI.

Вивчення підходів до підключення веб-програми до OpenAI API вимагає акценту на двох поширених мовах програмування: Python і Node.js, обидві з яких зазвичай використовуються для розробки веб-програм. Початковим кроком в інтеграції OpenAI API є створення ключа API в інтерфейсі розробника OpenAI. Цей ключ служить унікальним ідентифікатором, який надає авторизований доступ до служб OpenAI, і його потрібно зберігати в безпечному місці. Дуже важливо дотримуватися вказівок щодо безпеки під час інтеграції, особливо уникаючи зберігання ключів у загальнодоступних місцях або в коді, до якого можуть отримати доступ неавторизовані особи.

Отримавши ключ, ви можете використовувати відповідні бібліотеки для Python або Node.js для взаємодії з API. Бібліотека «openai» для Python і пакет «openai-api»

для Node.js пропонують зручні методи для виклику різних функцій API, що включає можливість використовувати різні моделі GPT для генерації тексту.

OpenAI API пропонує широкий спектр функціональних можливостей, включаючи можливість брати участь у стандартному форматі чату, який дозволяє взаємодіяти з мовними моделями за допомогою підходу запитань і відповідей. Крім того, він надає помічників, здатних виконувати більш складні завдання, таких як написання текстів або програмування (див. таблицю 2.1) [35].

Таблиця 2.1 Огляд моделей, доступних через OpenAI API

Модель	Опис
GPT-4 і GPT-4 Turbo	Представляють собою прогресивні моделі у порівнянні з GPT-3.5, демонструючи здатність розуміти та створювати як природну мову, так і код.
GPT-3.5	GPT-3.5 розширює можливості GPT-3, також демонструючи майстерність у розумінні та створенні природної мови або коду.
DALL·E	Ця модел, здатна генерувати та змінювати зображення за допомогою природної мови.
TTS	Набір моделей, призначених для перетворення тексту в аудіомовлення з природним звучанням.
Whisper	Модель, це модель, яка полегшує перетворення звуку в текст.
Embeddings	Відносяться до набору моделей, здатних перетворювати текст у числове представлення.
Moderation	Спеціальна модель, призначена для оцінки того, чи можна класифікувати текст як конфіденційний або потенційно шкідливий.

Продовження таблиці 2.1

Модель	Опис
GPT base	Набір моделей без інструкцій, які можуть розуміти, а також генерувати природну мову чи код.

Помічники, які не включені в моделі, але використовуються у веб-додатку, створеному в рамках цього магістерського проекту, будуть досліджені більш глибоко. Розробники можуть використовувати OpenAI Assistant API для створення помічників на основі ШІ у своїх власних програмах. Ці помічники оснащені різними інструментами для обробки запитів, такими як інтерпретатор коду для виконання Python, видобуток знань для отримання зовнішньої інформації та виклики функцій для виконання більш складних операцій. Завдяки інструкціям, спрямованим на моделі OpenAI, вони пропонують гнучкість як у індивідуальності, так і в функціональності, дозволяючи їм вирішувати складні завдання за допомогою безперервних потоків і файлів різних форматів. API Assistant створено для адаптації, дозволяючи розробникам включати власні інструменти, що покращує взаємодію між ШІ та користувачами. Цей фреймворк спрямований на оптимізацію створення додатків на основі ШІ та активацію динамічних можливостей помічників.

Невід'ємним аспектом ефективної інтеграції є оперативне проектування, яке стосується вміння створювати запит, який максимізує точність і ефективність можливостей штучного інтелекту. Правильно формулюючи підказки, користувачі можуть значно підвищити релевантність і якість отриманих відповідей. Цей процес є поєднанням мистецтва та науки, що вимагає глибокого розуміння як функціональних можливостей API, так і предмета, що стосується запиту. Додаткові відомості про те, як структурувати відповідний запит, описано нижче.

Мистецтво оперативного проектування охоплює як технічні, так і творчі аспекти, зосереджуючись на розробці запитів для моделей штучного інтелекту для отримання найбільш точних і відповідних відповідей. Ключові технічні компоненти оперативного проектування включають архітектуру моделі, навчальні дані,

токенізацію, параметри моделі, температуру, вибірку Top-k, а також функції втрат і градієнти. З творчої сторони необхідно враховувати важливі фактори, такі як інструкції, контекст, вхідні та вихідні показники. Наприклад, участь у рольових іграх може дати відповіді, адаптовані до конкретного іміджу чи професії, тоді як ітераційне уточнення та зворотній зв'язок служать для підвищення точності та актуальності наданих відповідей.

Застосування складних методів, таких як нульові підказки, які оцінюють здатність моделі узагальнювати завдання, відсутні в її навчанні, поряд з кількома підказками або навчанням у контексті, де модель надається з кількома прикладами для кращого розуміння завдання, є важливим для підвищення ефективності підказок. Інший просунутий метод, ланцюжок думок (CoT), передбачає керування моделлю послідовністю логічних кроків, що призводить до покращеного розуміння мови та більш точних результатів, подібно до систематичного пояснення складної математичної задачі іншій людині.

У сфері помічників методи розвиваються та урізноманітнюються. Під час розробки помічників у веб-програмі загальне підказка містить інформацію, яка пов'язана з різними документами. Ці документи можуть надавати посилання або деталі, призначені служити джерелом знань або основою для відповідей.

Висновки до розділу 2

1. Проведено ретельний аналіз інструментів і платформ, доступних для розробки веб-застосунків, зосереджуючись на їхніх перевагах, недоліках і поточних тенденціях ринку. Цей аналіз призвів до створення бази знань і початкових вимог до технологій, які використовуватимуться під час розробки веб-застосунку.

2. Вибір технологічного стеку для створення веб-застосунку визначається різними факторами, при цьому повна інтеграція з API OpenAI є найбільш фундаментальною. Тому були обрані такі технології, як Python, Django, Django REST framework, dj-rest-auth і openai для бекенд-розробки. Для інтерфейсу було обрано

React.js і MaterialUI. Впровадження інфраструктури використовуватиме хмарне середовище Amazon Web Services, зокрема EC2 і RDS.

3. Були розглянуті різні архітектурні підходи, засновані на обраному стеку технологій. Розробка архітектури програми почалася зі схеми бази даних і поширилася на архітектуру інфраструктури.

4. Проведено аналіз потенційних методів інтеграції з OpenAI API. Було ретельно вивчено концепцію оперативного проектування та її застосування в моделях помічників у навчанні. Крім того, було розглянуто бібліотеку openai Python, що призвело до визначення кількох функцій, призначених для майбутнього впровадження в розробку серверної логіки веб-застосунку.

РОЗДІЛ 3.
РЕЗУЛЬТАТИ ВИРІШЕННЯ ЗАДАЧІ

3.1 Демонстрація роботи створеного веб-застосунку

Ця частина магістерської роботи зосереджена на поглибленому дослідженні веб-застосунку, створеного для генерації коду за допомогою текстових запитів через використання OpenAI та Google API. Основна мета цієї програми — запропонувати користувачам потужний інструмент, який дає їм змогу швидко й точно формулювати запити на код, а також збирати відгуки для майбутніх удосконалень.

Процес авторизації є початковим етапом використання веб-додатку. Завдяки безпечному та простому інтерфейсу сторінка входу (рис. 3.1) гарантує надійний захист інформації користувача. Користувачі отримують легкий доступ до основних функцій програми завдяки інтуїтивно зрозумілій та нескладній природі процедур реєстрації та входу.

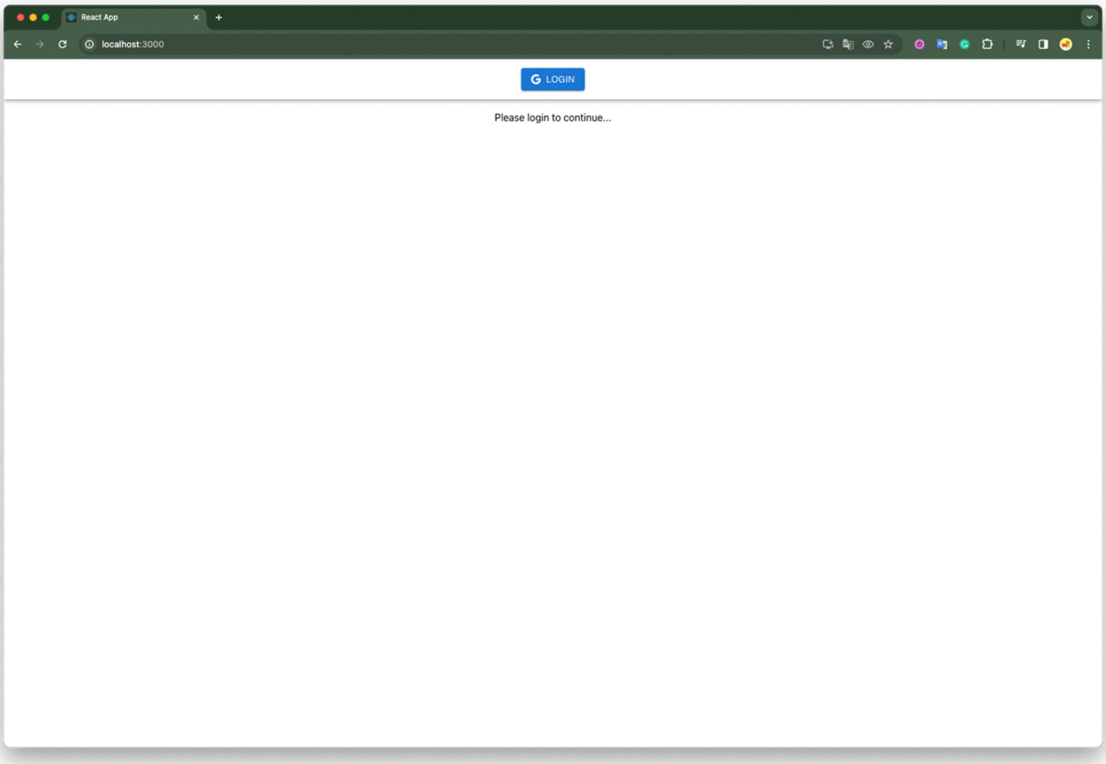

Рисунок 3.1 – Сторінка авторизації у веб-застосунок

Коли користувач натискає кнопку «Вхід», автентифікація відбувається через GoogleAPI, що дозволяє користувачеві уникнути створення окремого облікового запису в системі (див. рис. 3.2).

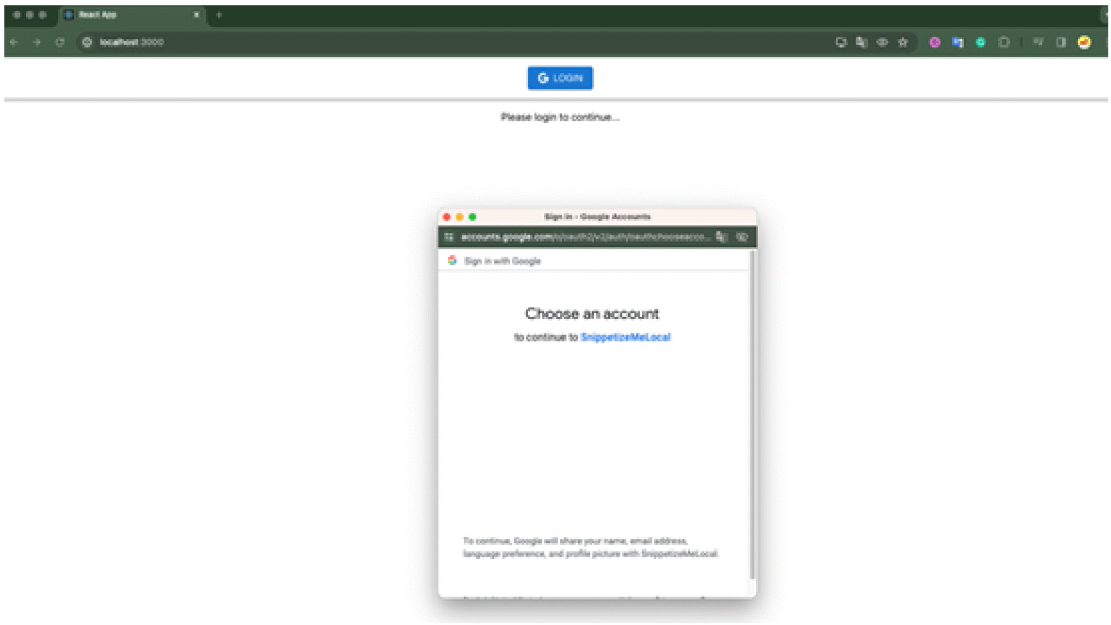

Рисунок 3.2 – Вибір облікового запису для авторизації у веб-застосунку

Це було досягнуто за допомогою кількох бібліотек, які працюють у поєднанні з Django, зокрема: dj_rest_auth і allauth, яка використовує постачальника Google. Реалізація відбувається у файлі налаштувань програми (рис. 3.3) і стає можливою завдяки додаванню додаткової кінцевої точки (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 – Модулі для включення авторизації в програмі за допомогою Google (файл settings.py)

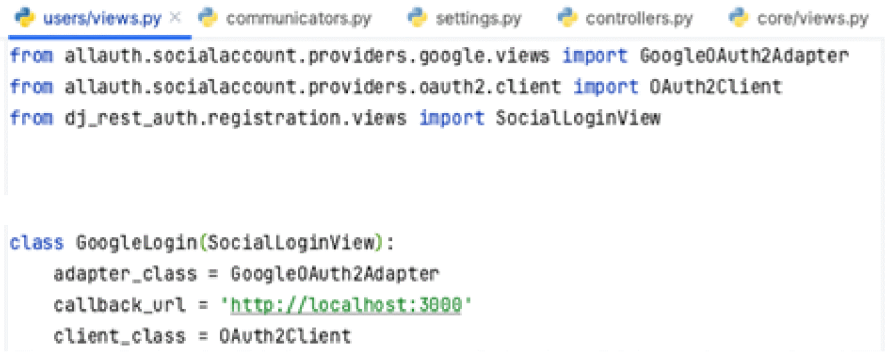

Рисунок 3.4 – Логіку кінцевої точки, відповідальну за реалізацію авторизації, можна знайти у файлі users/views.py.

Після успішного входу користувач перенаправляється на головну сторінку веб-застосунку (див. рис. 3.5).

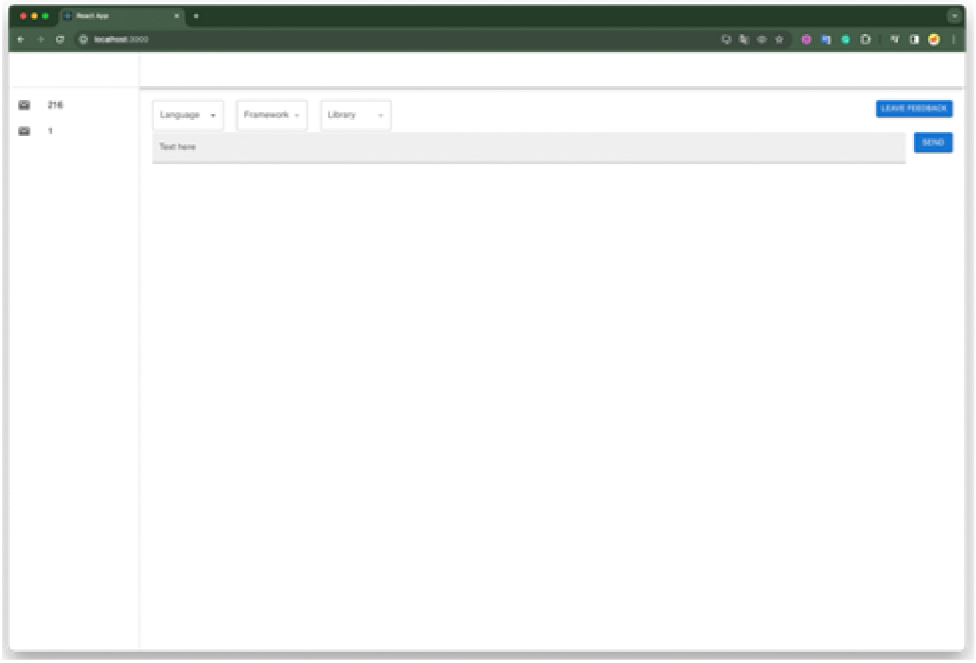

Рисунок 3.5 – Основний інтерфейс веб-застосунку

На основній сторінці відображається компіляція поточних потоків користувача, до яких користувач може отримати доступ за потреби (рис. 3.6).

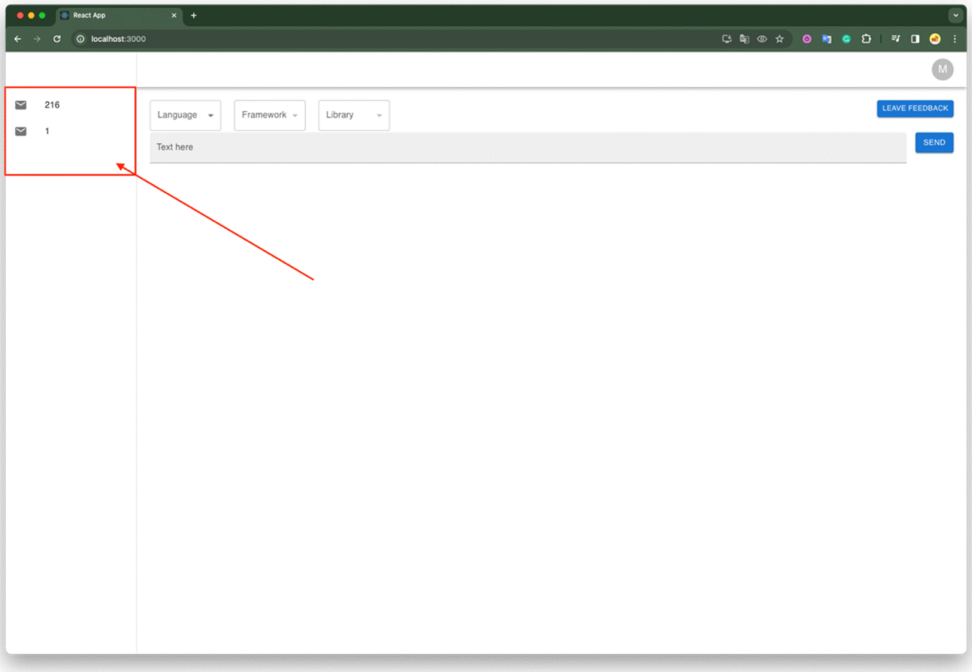

Рисунок 3.6 – Список потоків користувача

Коли користувач вибирає ідентифікатор потоку, буде відображено вміст, пов’язаний із цим потоком (рис. 3.7).

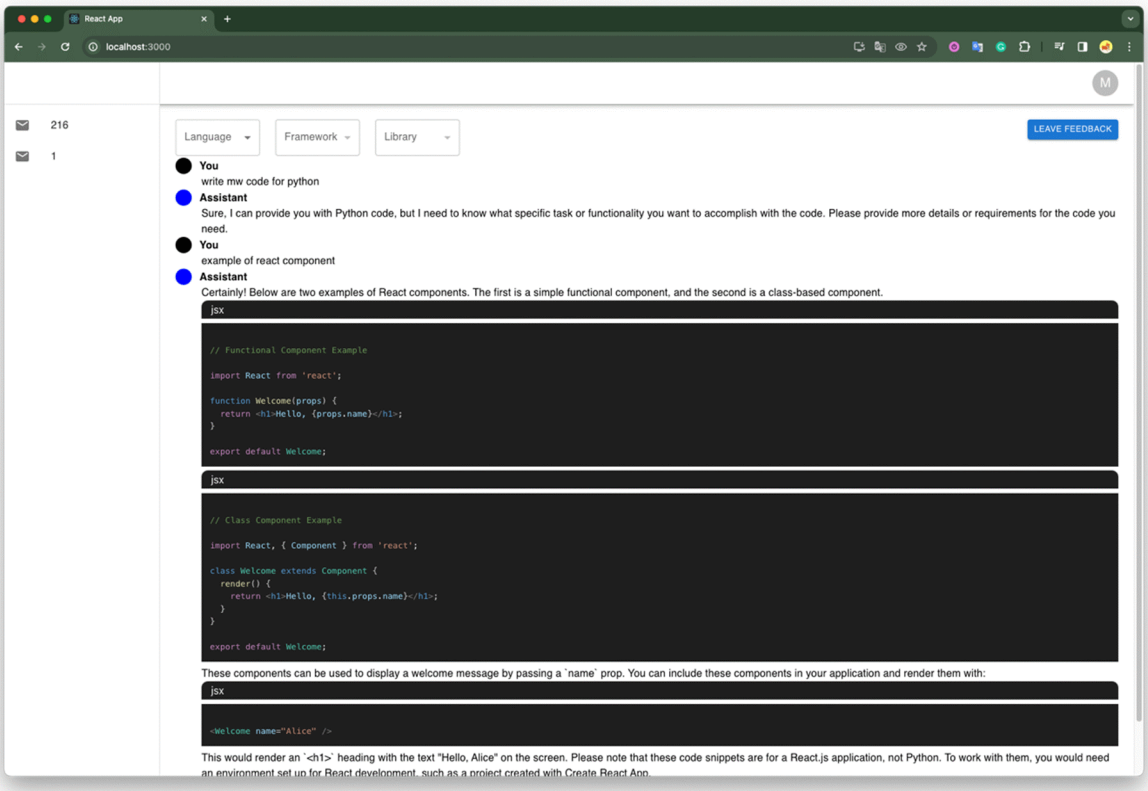

Рисунок 3.7 – Вибраний користувачем потік.

На стороні сервера було створено дві кінцеві точки, щоб полегшити цю функцію, як показано на малюнку 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 – Кінцеві точки для отримання даних у потоках

Кінцева точка для отримання всіх потоків, пов'язаних із користувачем 'prompt/threads-list/', реалізована через клас ThreadsListView, як показано на малюнку 3.9.

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

Рисунок 3.9 – Реалізація класу ThreadsListView.

Щоб отримати всі повідомлення з вибраного користувачем потоку, використовується кінцева точка 'prompt/thread-messages/'. Ця кінцева точка, створена за допомогою класу ListThreadMessagesView, відрізняється від попередньої кінцевої точки тим, що для неї потрібен ідентифікатор потоку, який надається інтерфейсом як параметр запиту (рис. 3.10).

```
class ListThreadMessagesView(APIView):

    def get(self, request, format=None):
        thread_id = request.query_params.get('thread_id')
        thread = Thread.objects.get(id=thread_id)
        thread_msgs = thread.get_msgs(client)

        # Serialize the data
        serializer = ThreadMessageSerializer()
        processed_data = serializer.parse_data(thread_msgs)
        serializer = ThreadMessageSerializer(data=processed_data, many=True)
        if serializer.is_valid():
            return Response(serializer.data)
        else:
            return Response(serializer.errors, status=400)
```

Рисунок 3.10 – Реалізація класу ListThreadMessagesView

У класі, зображеному на малюнку 3.10, потік, відфільтрований за своїм ідентифікатором, згодом викликає метод get_msgs(client), який визначено в класі, представленому на малюнку 3.11. У рамках цього методу ми робимо запит до клієнта API OpenAI і отримуємо всі повідомлення, пов’язані з потоком.

```
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)


    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


    def get_msgs(self, client: OpenAI):
        messages = client.beta.threads.messages.list(
            thread_id=self.open_ai_thread_id
        )
        return messages
```

Рисунок 3.11 – Реалізація методу get_msgs у моделі потоку.

Після вилучення списку повідомлень наступним кроком є створення об’єкта, який буде надіслано назад у інтерфейс як відповідь. Для цього до класу серіалізатора моделі Thread було додано метод parse_data, який бере дані зі списку повідомлень і організовує їх у необхідну структуру (рис. 3.12).

```
class ThreadMessageSerializer(Serializer):
    role = CharField()
    content = CharField()


    @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 – Серіалізатор, призначений для обробки повідомлень потоку, показуючи реалізацію методу для дисперсії даних.

Крім того, головна сторінка містить ряд спадних меню, які дозволяють користувачам вибрати відповідні параметри та параметри (рис. 3.14). Ці функції дають змогу користувачам адаптувати свої запити відповідно до конкретних вимог, значно підвищуючи точність і релевантність згенерованого коду та відповідей. Вибір цих варіантів є вирішальним, оскільки вони визначають, який спеціалізований помічник буде обраний у майбутньому, створений спеціально для кожного варіанту.

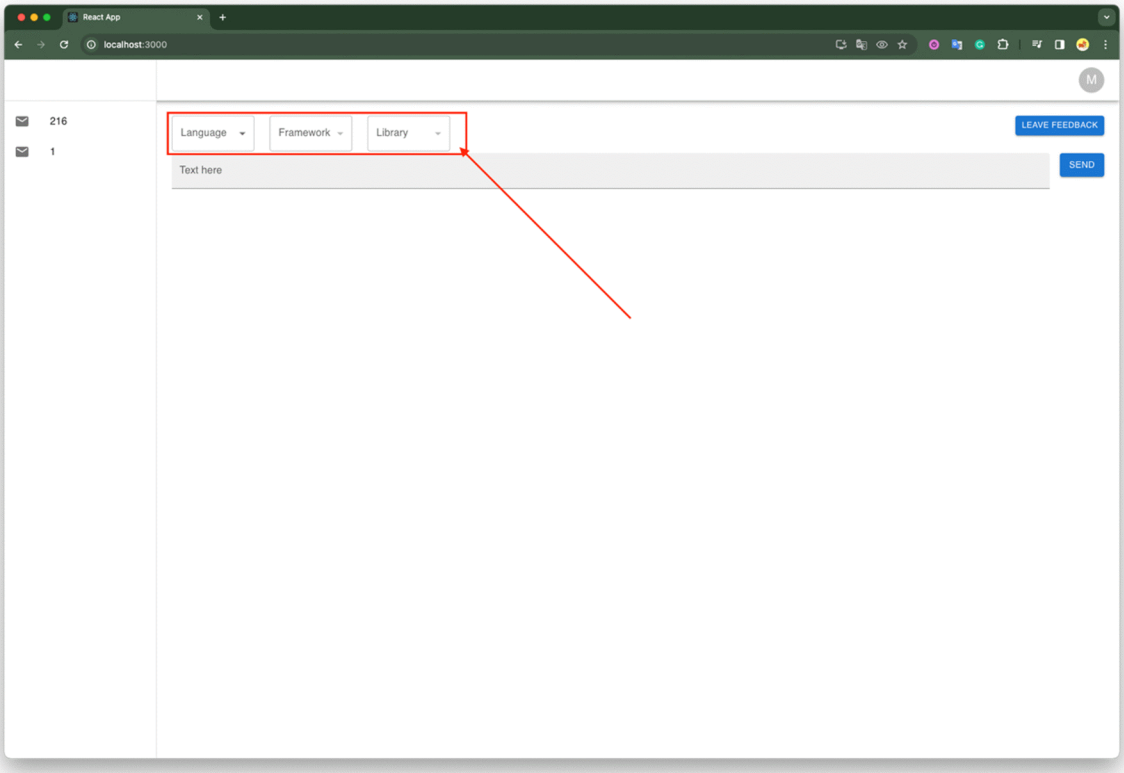

Рисунок 3.14 – Набір спадних меню для налаштування помічника

Після доступу до головної сторінки зовнішній інтерфейс отримує всі доступні варіанти через кінцеву точку «prompt/options/» (Малюнок 3.15). Цей запит негайно надсилається компонентом інтерфейсу користувача, а потім уся зібрана інформація зберігається в його пам’яті.

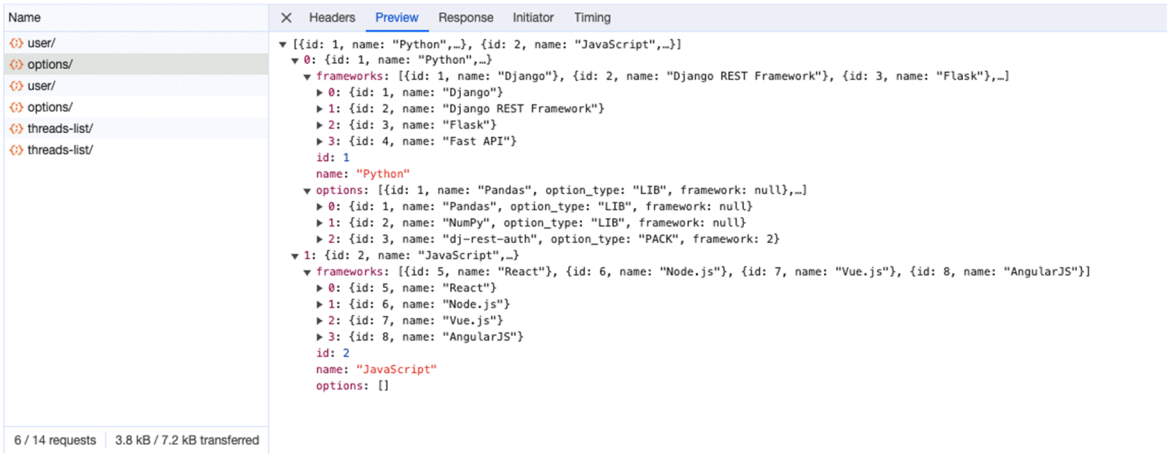

Рисунок 3.15 – Відповідь сервера на запит щодо всіх доступних опцій.

На стороні сервера це було виконано за допомогою класу, зображеного на малюнку 3.16, і серіалізаторів, зображених на малюнку 3.17.

```
class PromptOptionsListView(ListAPIView):
    serializer_class = ProgrammingLanguageSerializer
    queryset = ProgrammingLanguage.objects.all()
```

Рисунок 3.16 – Клас із логікою кінцевої точки, який відображає всі доступні параметри.

Рисунок 3.17 – Класи серіалізаторів, які створюють відповідь.

У центрі головної сторінки знаходиться текстове поле (рис. 3.18), призначене для введення користувачами своїх запитів. Хоча цей інтерфейс є зручним для користувача, він також є надзвичайно потужним інструментом, оскільки він безпосередньо спілкується з API для створення коду. Користувачі мають можливість формулювати свої запити природною мовою, а програма перетворює їх у вичерпні та оптимізовані інструкції з кодування.

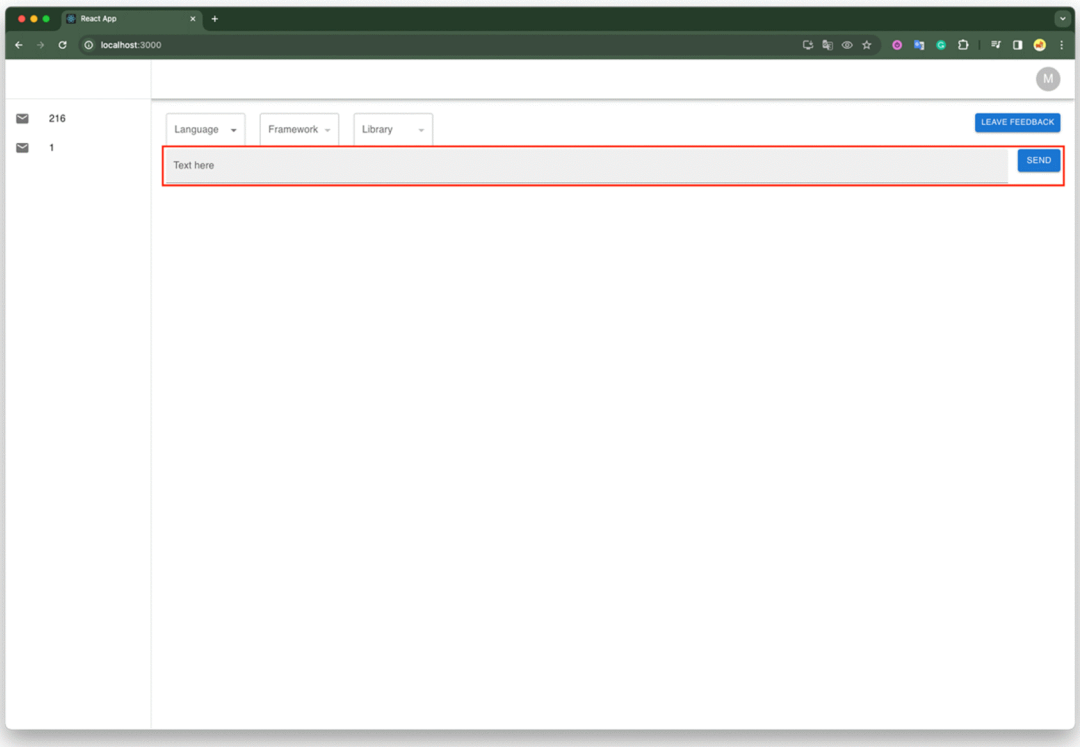

Рисунок 3.18 – Текстове поле, у якому користувачі вводять свої запити.

Після того, як користувач надсилає текстовий запит, натиснувши кнопку «Надіслати», запит надсилається до серверної частини. Тут ініціюється новий потік для вибраного помічника відповідно до параметрів, вибраних у спадних меню. У цьому процесі використовується кінцева точка «prompt/post-prompt/» (рис. 3.19).

```python
class PostPromptView(APIView):
    permission_classes = [IsAuthenticated]


    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 'f{run.status}'. "
                       f"It will be processed as soon as possible.",
            "thread_id": thread.id
        })


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


    @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 демонструє логіку створення або оновлення потоку новими повідомленнями користувача.

Очевидно, що цей запит можуть робити лише авторизовані користувачі. Окрім основного методу post, клас також містить два додаткових методи, invoke_thread та invoke_run, як показано на малюнку 3.20.

```python
def invoke_thread(self, thread_id: Optional[int]) -> Thread:
    if thread_id is None:
        open_ai_thread = client.beta.threads.create()
        thread = Thread(
            user=self.request.user,
            open_ai_thread_id=open_ai_thread.id,
        )
        thread.save()
    else:
        thread = Thread.objects.get(id=thread_id)
    return thread


@staticmethod
def invoke_run(
        thread: Thread,
        prompt: str,
        language_id: int,
        framework_id: Optional[int],
        option_id: Optional[int]
):
    message = client.beta.threads.messages.create(
        thread_id=thread.open_ai_thread_id,
        role='user',
        content=prompt,
    )
    assistant = Assistant.objects.get(
        language_id=language_id,
        framework_id=framework_id,
        option_id=option_id,
    )
    run = client.beta.threads.runs.create(
        thread_id=thread.open_ai_thread_id,
        assistant_id=assistant.assistant_id
    )
    thread.open_ai_run_id = run.id
    thread.save()
    return run
```

Рисунок 3.20 – Реалізація додаткових методів класу PostPromptView

код ілюструє, що метод invoke_thread містить функції як для створення, так і для отримання вже встановленого потоку. Ця функція дозволяє користувачам покращувати раніше створені сутності. В результаті система дозволяє користувачам надсилати кілька запитів в рамках одного потоку без обмежень.

Метод invoke_run – це місце, де ініціюється запит користувача. Як показано в коді цього методу (рис. 3.20), повідомлення спочатку генерується в потокі OpenAI API. Згодом ми виводимо суть помічника на основі параметрів, вибраних користувачем у спадних меню, які надсилаються на сервер за допомогою інтерфейсу користувача. Від отриманого помічника ми отримуємо його ідентифікатор OpenAI і починаємо обробку потоку за допомогою цього помічника.

Важливо підкреслити, що ця кінцева точка не відразу відповідає на створений OpenAI API, оскільки для обробки запиту API потрібен період очікування. Отже, було створено додаткову кінцеву точку, щоб пропонувати подробиці щодо статусу потоку. Запит вважається обробленим, коли статус потоку досягне завершення. Тому на інтерфейсі після надсилання запиту користувача робиться кілька запитів для моніторингу стану потоку, доки він не вкаже на завершення (рис. 3.21).

```
const checkThreadStatus = async (id) :Promise<void>  => {
  setThreadId(id);
  getThreadStatus(id)
    .then((res :AxiosResponse<any> ) :void  => {
      if (res.data.status === "in_progress") {
        setTimeout( callback: () :void  => {
          checkThreadStatus(id);
        }, ms: 5000);
      } else {
        setLoading( value: false);
        getThreadMessages(id) Promise<AxiosResponse<...>>
          .then((data :AxiosResponse<any> ) :void  => setMesages(data.data)) Promise<void>
          .catch((err) => console.log(err));
        setUserRequestMessage( value: "");
      }
    })
    .catch((err) => console.log(err));
};
```

Рисунок 3.21 – Метод checkThreadStatus, який реалізовано для моніторингу стану потоку.

На серверній частині ця функція реалізована через клас GetThreadStatusView, показаний на малюнку 3.22.

```
class GetThreadStatusView(APIView):
    permission_classes = [IsAuthenticated]


    def get(self, request, format=None):
        # Extract query parameters
        thread_id = request.query_params.get('thread_id')

        try:
            thread = Thread.objects.get(id=thread_id)
        except Thread.DoesNotExist as exc:
            return Response({
                "exc": exc, "user": self.request.user.id,
                "thread_id": thread_id
            })
        return Response({"status": thread.get_status(client)})
```

Рисунок 3.22 – Реалізація класу GetThreadStatusView

На ілюстрації вище видно, як статус отримується за допомогою методу Thread.get_status(). Як показано на малюнку 3.11, цей метод працює шляхом надсилання запиту до OpenAI API, використовуючи ідентифікатор потоку для отримання його статусу, який потім повертається у відповідь сервера.

Коли зовнішній інтерфейс отримує підтвердження того, що статус потоку позначено як завершений, він надсилає запит на отримання повідомлень, пов'язаних із цим потоком. Ця функціональність була виконана в класі ListThreadMessagesView, який був ретельно вивчений раніше (рис. 3.10).

Зрештою, інтерфейс містить важливий компонент: кнопку подання відгуку (рис. 3.23) і модальне вікно (рис. 3.24), яке дозволяє користувачам надавати свої відгуки. Ця функція сприяє прямому обміну думками та рекомендаціями щодо вдосконалення програми. Зібрані дані відгуків використовуються для оцінки взаємодії з користувачем і вдосконалення функціональності веб-застосунку.

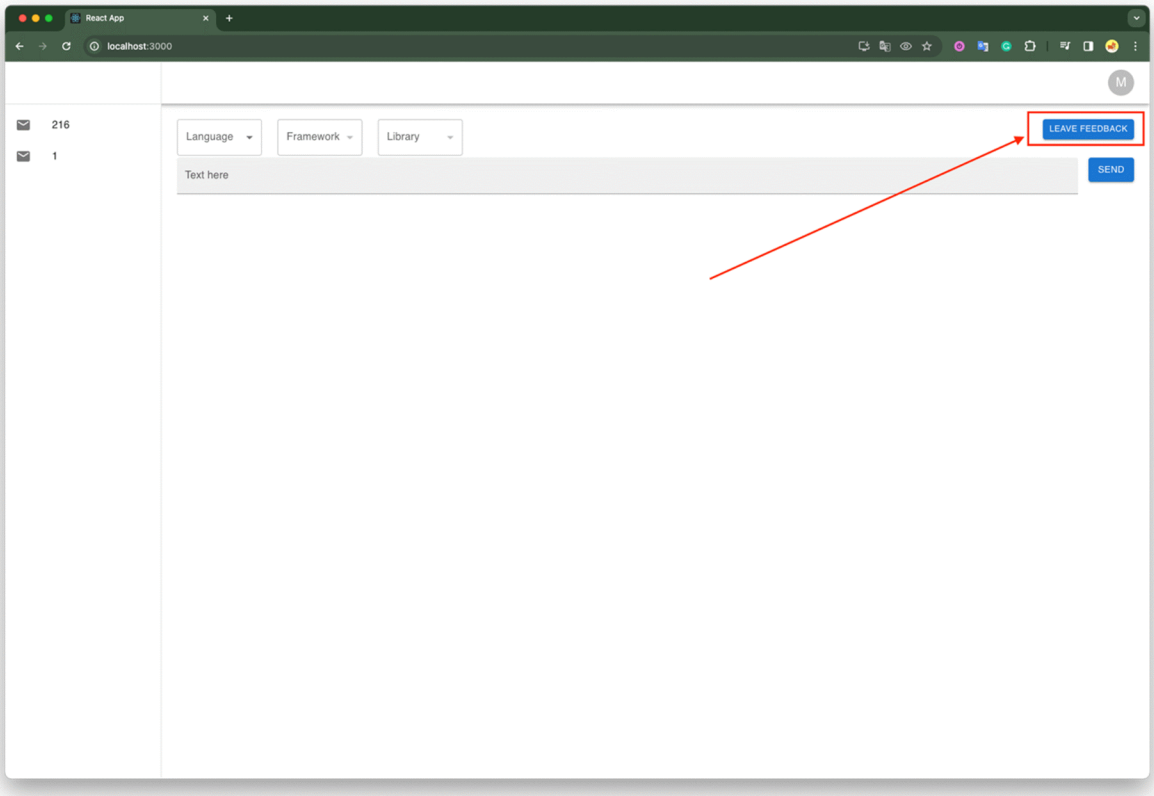

Рисунок 3.23 – Кнопка для запуску модального вікна для надсилання відгуків.

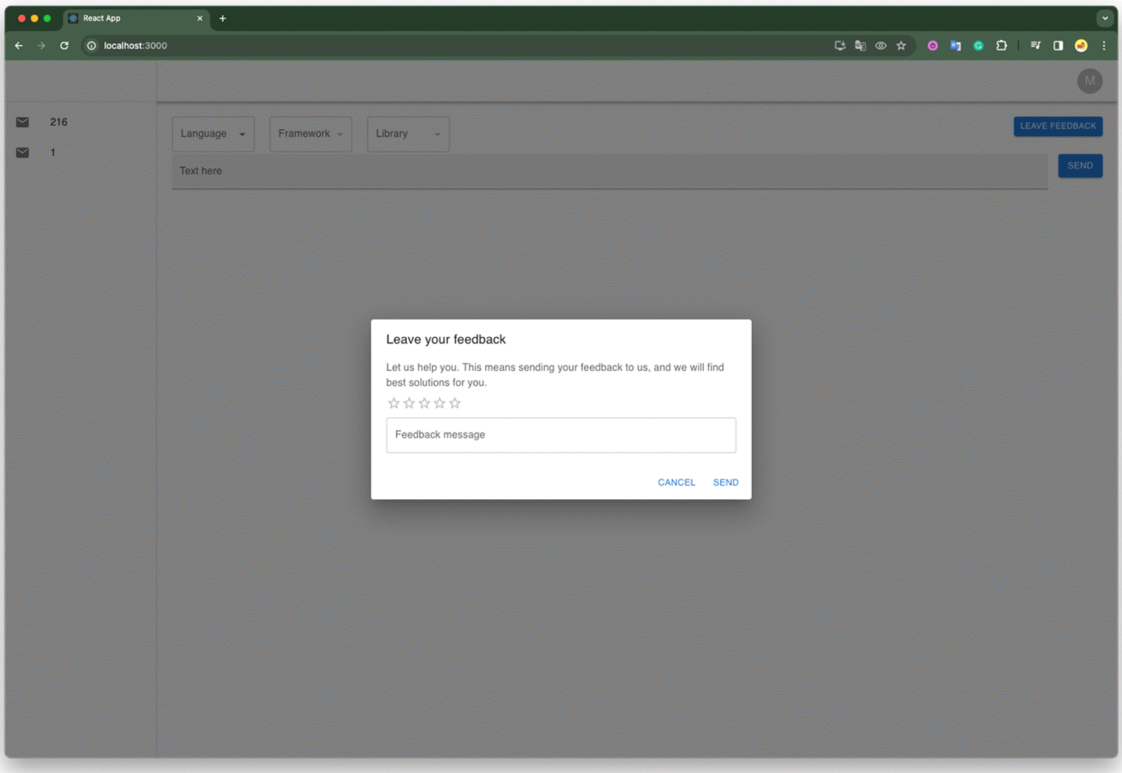

Рисунок 3.24 – Модальне вікно зворотного зв’язку

Щоб полегшити надсилання відгуку, було розроблено клас CreateFeedback, до якого можна отримати доступ через кінцеву точку «prompt/post-feedback/» (див. рис. 3.25).

```python
class CreateFeedback(APIView):
    permission_classes = [IsAuthenticated]


    def post(self, request, format=None):
        # Extract input parameters from request.data
        thread_id = request.data.get('thread_id')
        value = request.data.get('value')
        comment = request.data.get('comment')
        feedback = Feedback(
            thread_id=thread_id,
            value=value,
            comment=comment,
        )
        feedback.save()
        # Return response
        return Response({
            "message": f"Feedback successfully created.",
            "feedback_id": feedback.id
        })
```

Рисунок 3.25 – Реалізація класу відгуків користувачів CreateFeedback

Цей короткий опис веб-застосунку пропонує глибоке розуміння його основних елементів та їхніх функцій у забезпеченні ефективного, зручного та цінного досвіду. Завдяки інтеграції відгуків користувачів додаток розвиватиметься та адаптуватиметься до мінливих потреб і запитів користувачів.

3.2 Огляд інтеграції з OpenAI API для створення помічника та Google API для аналізу відгуків користувачів.

Основна функція веб-застосунку, розробленого для магістерської роботи, зосереджена на використанні OpenAI API, зокрема через його інтеграцію в бібліотеку openai на основі Python. Це дає змогу розробляти індивідуальних помічників, здатних пропонувати користувачам не лише окремі фрагменти коду, але й цінну інформацію та рекомендації, адаптовані до їхніх запитів і вибраних уподобань.

Для полегшення інтеграції API OpenAI у програму було створено окремий клас, який служить мостом для зв'язку між API та самою програмою. Цей важливий клас забезпечує взаємодію між веб-програмою та OpenAI, дозволяючи ініціалізувати клієнт, який виконуватиме всі подальші завдання. Успадкувавши від абстрактного класу, який підтримує створення одиночних елементів, цей клас функціонує як дочірній клас. Шаблон проектування Singleton, широко відомий у Python, гарантує створення лише одного екземпляра об'єкта класу. Основною перевагою використання шаблону Singleton є надання глобального єдиного доступу до ресурсу чи служби, що важливо в ситуаціях, коли життєво важливо запобігти неконтрольованому створенню кількох екземплярів, наприклад, коли ви маєте справу з базами даних або конфігураційними файлами. Цей підхід пом'якшує конфлікти та пропонує централізоване керування спільними ресурсами. Прийняття Singleton також покращує організацію коду, зменшуючи залежність від глобальних змінних і покращуючи обробку ресурсів. Отже, буде єдиний екземпляр класу, призначений для спілкування з OpenAI API. Код реалізації для цього класу можна знайти на малюнку 3.26.

urls.py | users/views.py | communicators.py | controllers.py | core/views.py | analyze_

```python
from openai import OpenAI

from typing import Optional
from abc import ABC, abstractmethod

from snippetize_me.settings import config


SECRET_KEY = config.get('open-ai-api').get('secret-key')



class BaseCommunicator(ABC):
    """
    The base class for generating a communicator between API and controllers
    """
    _instance = None

    ± mmartyniak
    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._is_initialized = False
        return cls._instance


    def __init__(self, secret_key: Optional[str] = SECRET_KEY):
        if not self._is_initialized:
            self.api_client = self._initialize_api_client(secret_key)
            self._is_initialized = True


    @abstractmethod
    def _initialize_api_client(self, secret_key):
        """Method for api initialization with secret key"""
        raise NotImplementedError



class OpenAICommunicator(BaseCommunicator):

    def _initialize_api_client(self, secret_key: str) -> OpenAI:
        return OpenAI(api_key=secret_key)
```

Рисунок 3.26 – Файл, що містить класи, призначені для взаємодії з OpenAI API.

На рисунку вище показано, що базовий клас містить атрибут api_client, який дозволяє нам взаємодіяти з OpenAI API. Ця взаємодія додатково продемонстрована на наступних малюнках.

Відмінною особливістю реалізації є генерація помічників через панель адміністратора. У процесі створення помічника використовується поле інструкції, а метод збереження за замовчуванням перевизначено (рис. 3.27). Ця модифікація дає змогу адміністратору програми налаштовувати помічників із спеціальними

інструкціями, сприяючи більш цілеспрямованому та ефективнішому використанню API OpenAI у рамках веб-застосунку.

```python
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)


    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)


    def __str__(self):
        return self.name
```

Рисунок 3.27 – Модель асистента, з перевизначенням методу збереження

Рисунок 3.27 ілюструє структуру допоміжної таблиці, де окреслено всі важливі поля та атрибути. Поряд із цими полями також представлено спосіб збереження запису моделі. Цей метод змінює стандартну процедуру збереження; зокрема, він не тільки зберігає запис, що містить усі дані, але також викликає клієнт OpenAI, використовуючи клас, показаний на малюнку 3.27, що призводить до створення нового помічника, заснованого на вказівках, встановлених адміністратором.

Панель адміністратора зображена на малюнку 3.28. Ця панель дозволяє створювати, аналізувати, оновлювати та видаляти всі сутності, наявні в базі даних.

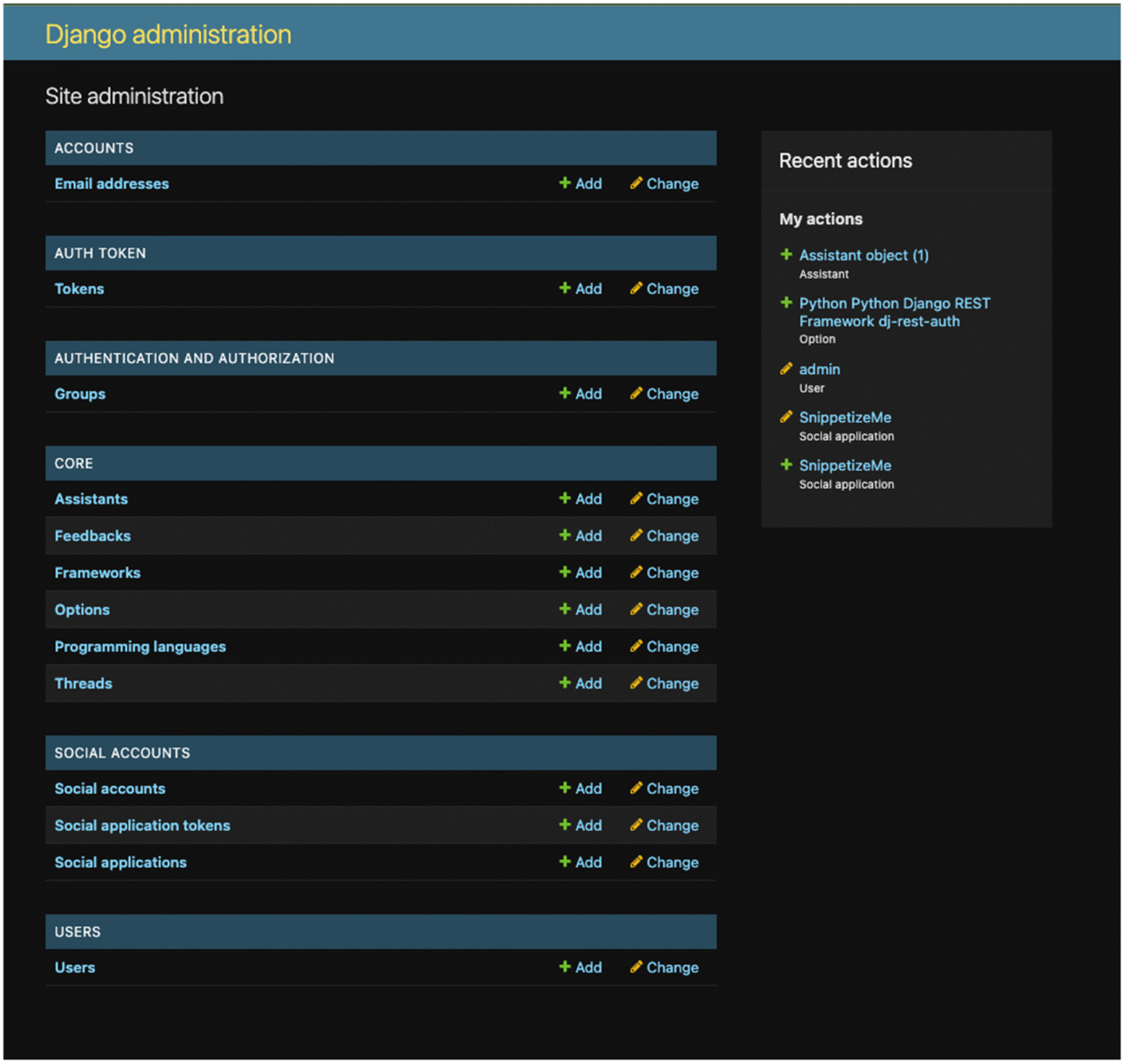

Рисунок 3.28 – Панель адміністратора веб-застосунку

Щоб створити нового помічника, просто натисніть кнопку «Додати» поруч із відповідною моделлю в списку пластин, які знаходяться в панелі «Core». Після цієї дії ви потрапите на сторінку помічника (рис. 3.29), де зможете вибрати всі необхідні параметри та скласти інструкцію.

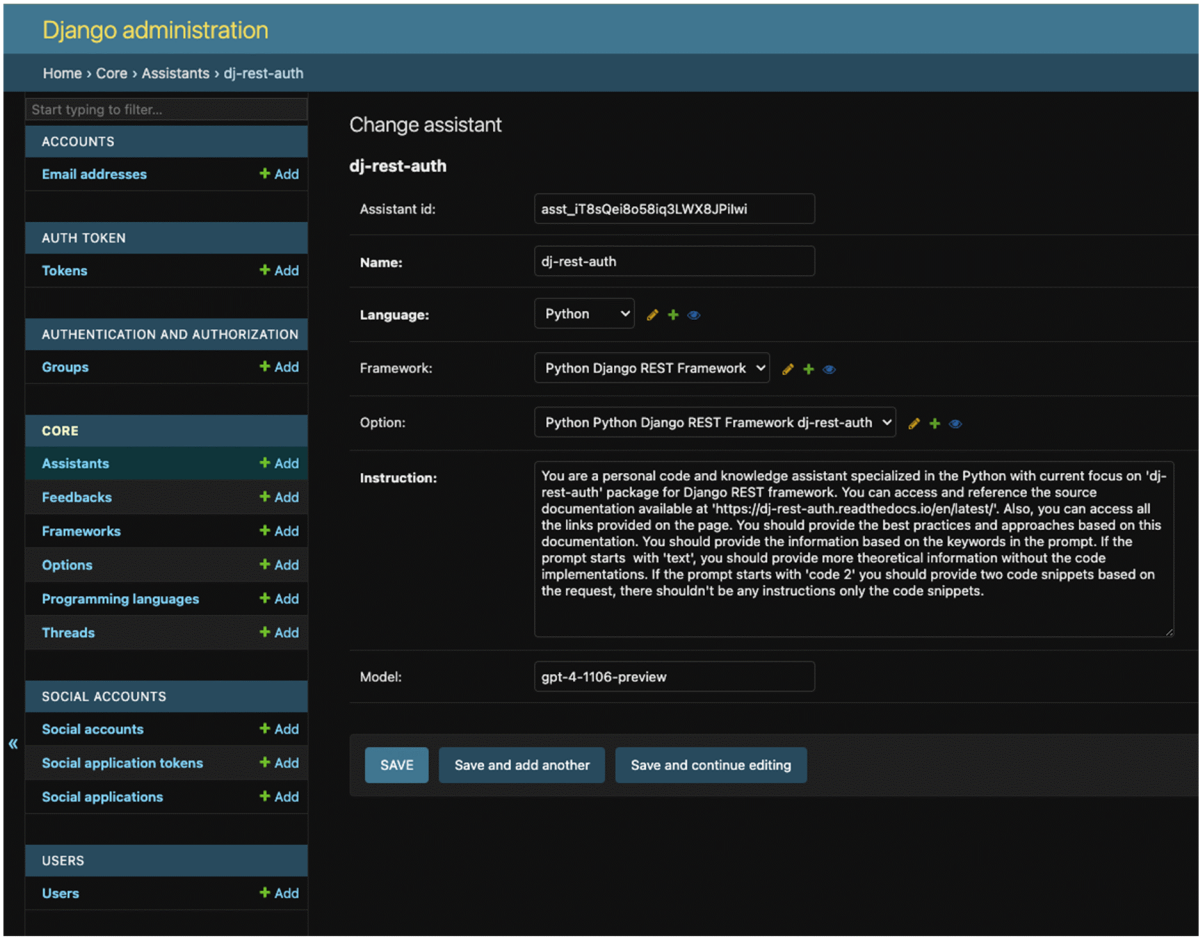

Рисунок 3.29 – Сторінка створення помічника

Коли адміністратор натискає кнопку ЗБЕРЕГТИ (SAVE), поряд із цим методом викликається відповідний метод у моделі помічника, проілюстрований вище (рис. 3.29). Після цього слідує ряд кроків, зокрема:

1) розробка помічника в нашій базі даних.

2) розробка помічника на стороні OpenAI API.

3) Поточний запис помічника оновлюється, зокрема поле assistant_id, яке тепер міститиме значення ідентифікатора помічника на стороні OpenAI. У майбутньому помічники будуть використовуватися безпосередньо під час виконання запитів для генерації коду та покращення відповідей. Таким чином, інтеграція OpenAI API у проект надає широкі можливості для автоматизації генерації коду, забезпечуючи гнучкий і масштабований підхід до ряду завдань програмування та розробки програмного забезпечення, оскільки для кожного потенційного варіанту буде створений власний чіткий і детальний помічник.

Збір відгуків користувачів має важливе значення для вдосконалення та розвитку веб-застосунків. Користувачі свідомо та чітко пропонують такий відгук, який включає оцінки зірочками, коментарі та відповіді на опитування. Такий зворотний зв'язок забезпечує безпосереднє розуміння від користувачів, допомагаючи оцінити їхню задоволеність продуктом і визначити конкретні області, які потребують зосередження.

Неявні огляди вимагають аналізу поведінки користувачів, включаючи тривалість перебування на певних сторінках, частоту використання програми та випадки покинутих кошиків для покупок. Хоча вони не пропонують явного зворотного зв'язку, неявний зворотний зв'язок може виявити моделі поведінки та підсвідомі уподобання користувачів.

Аналіз зворотного зв'язку охоплює як якісний, так і кількісний підходи. Кількісні методи можуть передбачати обчислення середньої кількості зірок або перевірку того, як часто певні теми згадуються в коментарях. З іншого боку, якісний аналіз може передбачати ретельний аналіз коментарів, щоб розпізнати настрої, визначити загальні проблеми або дослідити рекомендації користувачів.

У створеному веб-застосунку була включена функція для надання зворотного зв'язку в потоці. Це було досягнуто шляхом розробки вікна, де користувачі можуть оцінювати та надсилати коментарі. Ілюстрація модального вікна для залишення відгуку представлена на рисунку 3.24.

Для системи, що досліджується, користувачам дозволяється надавати явний зворотний зв'язок через вікно зворотного зв'язку, який потім можна ретельно проаналізувати за допомогою вбудованих команд Django, що виконуються як заплановані завдання (коронки). Команда класифікує відгуки на основі визначених критеріїв і пересилає їх до API Google Bard для додаткового аналізу. Малюнок 3.30 нижче ілюструє алгоритм для команди аналізу зворотного зв'язку.

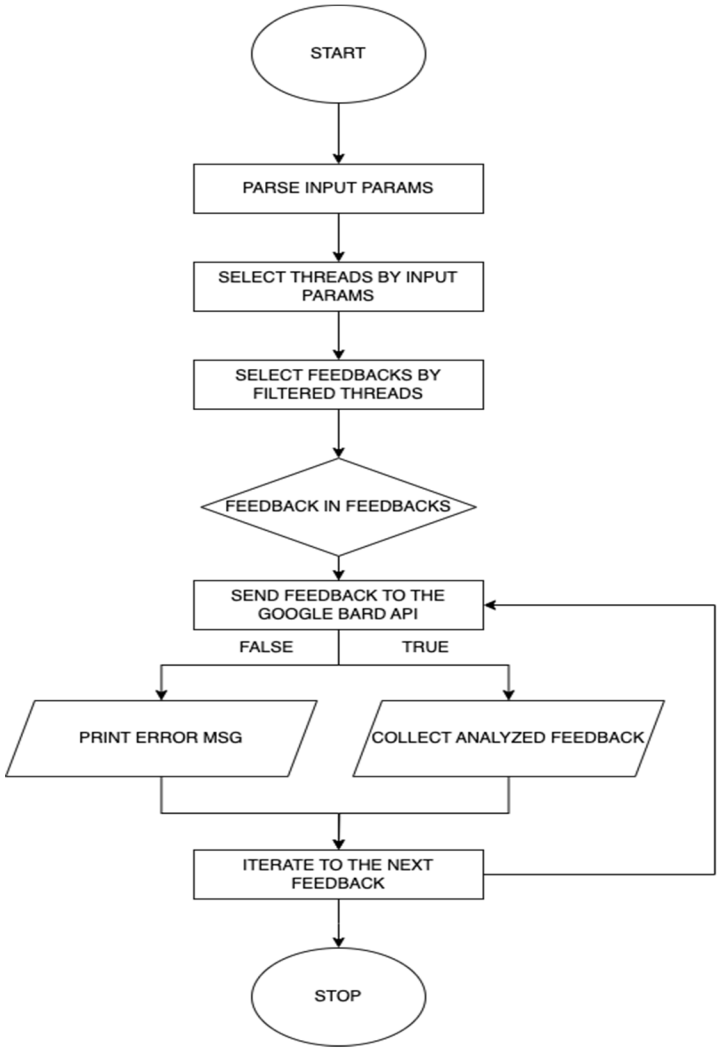

Рис. 3.30 Алгоритм роботи команди для аналізу зворотного зв'язку

За допомогою цього алгоритму була розроблена команда Django, код для якої показано на малюнку 3.31.

urls.py | users/views.py | core/views.py | public | analyze_feedbacks.py | settings.py

```python
from django.core.management.base import BaseCommand, CommandError
from core.models import Thread, Feedback
import requests


new *
class Command(BaseCommand):
    help = 'Analyzes feedbacks for specific threads'

    new *
    def add_arguments(self, parser):
        parser.add_argument('programming_language_id', type=int)
        parser.add_argument('framework_id', type=int)
        parser.add_argument('option_id', type=int)

    new *
    def handle(self, *args, **options):
        programming_language_id = options['programming_language_id']
        framework_id = options['framework_id']
        option_id = options['option_id']

        threads = Thread.objects.filter(
            language_id=programming_language_id,
            framework_id=framework_id,
            option_id=option_id
        )

        feedbacks = Feedback.objects.filter(thread__in=threads)

        for feedback in feedbacks:
            data = {
                'value': feedback.value,
                'comment': feedback.comment
            }
            response = requests.post(
                url: 'https://api.googlebard.com/analyze', json=data
            )
            if response.status_code == 200:
                analysis_result = response.json()
                print(analysis_result)
            else:
                print('error while analyzing feedback:', feedback.id)

        self.stdout.write(self.style.SUCCESS('feedback review finished!'))
```

Рисунок 3.31 – Реалізована Django команда для аналізу зворотного зв’язку

У результаті проаналізовані дані дають зрозуміти, які допомагають точно визначити та вирішити конкретні проблеми, зрештою підвищуючи задоволеність користувачів і ефективність програми.

3.3 Потенційні методи подальшого вдосконалення веб-застосунку.

Для подальшого вдосконалення веб-програми важливо зосередитися на її поточному статусі закритої бета-версії. Під час цього етапу доступ обмежено: користувачі можуть увійти через обліковий запис Google, лише якщо їхні облікові записи були попередньо додані до відповідної програми Google. Цей метод полегшує контрольований доступ до програми, що є життєво важливим для збору відгуків і визначення потенційних проблем до її повного запуску. Наступним кроком є публікація програми, яка має пройти перевірку Google. Ця процедура є необхідною, оскільки вона змушує розробників переконатися, що програма відповідає стандартам і політикам Google, тим самим гарантуючи якість і безпеку для різноманітної бази користувачів. Отримання підтвердження від Google не тільки підвищить довіру до програми, але й розширить її потенційну аудиторію, дозволяючи залучати більше користувачів і збирати додаткові дані для постійного аналізу та вдосконалення.

Важливою віхою в розробці веб-додатку є успішна реалізація критичної функціональності, яка вже досягнута. Це досягнення закладає міцну основу для майбутнього розвитку програми. Серед інноваційних ідей щодо покращення функціональності є потенційне впровадження системи токенів. Ці токени дозволять користувачам надсилати запити до програми. На початковому етапі користувачам буде виділено певну кількість токенів, що дозволить їм вивчити функціональність і оцінити, наскільки ефективно додаток відповідає їхнім потребам, а також чи готові вони інвестувати в його постійне використання. Ця стратегія не лише сприяє початковій взаємодії з додатком, але й збирає цінні відгуки від реальних користувачів, що має вирішальне значення для вдосконалення продукту. Крім того, система токенів може допомогти у формуванні стратегії монетизації програми, встановлюючи баланс між пробним доступом і платним доступом для преміум-функцій.

Добре продумана стратегія монетизації нашого веб-додатку має важливе значення для стійкості та успіху проекту. Вирішальним аспектом цієї стратегії є визначення відповідної ціни для токенів, які користувачі використовують для запиту допомоги. Ціна токена повинна відповідати ціновій політиці використання

помічників, враховуючи витрати на ресурси, а також зберігаючи конкурентоспроможність на ринку.

Крім того, планується запустити опцію преміум-підписки, що запропонує користувачам підвищену гнучкість і персоналізацію. Абоненти матимуть можливість налаштувати свої параметри, адаптуючи помічника відповідно до своїх унікальних потреб і конкретних бібліотек або пакетів, які вони використовують. Це вдосконалення значно підвищує привабливість програми для професіоналів, які шукають цільові рішення. Процес налаштування помічника спрощено завдяки вже існуючим інструкціям, які спрямовують користувачів у створенні помічника, вимагаючи лише необхідної документації для вибраного пакета чи бібліотеки.

Мета цих кроків — створити стійку фінансову базу для програми, забезпечуючи задоволення потреб користувачів і надання цінності. Гнучка й адаптивна стратегія монетизації має вирішальне значення для ефективного реагування на зміни в ринкових умовах і нових вимог користувачів.

Важливим елементом розробки нашої веб-програми є створення основних помічників, які обслуговують широко використовувані мови програмування, фреймворки та бібліотеки. Це відіграватиме вирішальну роль у тому, що звичайні користувачі зможуть належним чином відповідати своїм вимогам щодо програмування та розробки. Наразі під час фази бета-тестування було розроблено лише обмежену кількість помічників для демонстрації функцій і збору відгуків користувачів. Тим не менш, щоб досягти всебічного запуску та задовольнити різноманітні потреби широкої бази користувачів, необхідно значно розширити цей діапазон.

Впровадження фундаментальних помічників для широко використовуваних технологій створить нові можливості для користувачів, особливо для осіб, яким може бракувати глибоких знань у певних галузях. Це вдосконалення також сприятиме більш широкому застосуванню програми, оскільки користувачі зможуть швидко отримувати допомогу та вказівки, налаштовані для певних мов програмування та технічних стеків.

Збільшення кількості помічників підвищить цінність програми, звернувшись до ширшого кола користувачів, від новачків до досвідчених розробників. Це оновлення дозволить програмі служити всеохоплюючим інструментом для програмування та розробки, пропонуючи надійну допомогу у вирішенні різноманітних технічних проблем.

Створення веб-додатку вимагає встановлення функціональних можливостей, які регулюють споживання ресурсів користувача. Цей елемент є життєво важливим, оскільки правильне керування використанням ресурсів значно впливає на стабільність і продуктивність програми. Завдяки встановленню обмежень буде досягнуто справедливий розподіл ресурсів між користувачами, що дозволить уникнути надмірного навантаження на систему.

Встановлення меж може передбачати обмеження частоти запитів протягом певного періоду часу, контроль доступу до певних функцій або ресурсів і визначення максимальної тривалості для виконання певних завдань. Впровадження цих стратегій допоможе уникнути перевантаження сервера та навантаження на іншу інфраструктуру, забезпечуючи стабільнішу та бездоганну роботу для всіх користувачів.

Крім того, цей метод можна інтегрувати в стратегію монетизації. Наприклад, базовим користувачам може бути надано певний ліміт безкоштовного використання, тоді як преміальні користувачі можуть користуватися розширеним доступом або збільшеними лімітами. Це не тільки сприяє більш ефективному розподілу ресурсів, але й мотивує користувачів розширювати свої підписки на додаткові функції.

Забезпечення того, щоб користувачі були обізнані про встановлені обмеження та поточне використання ними ресурсів, також має вирішальне значення для прозорості. Такий підхід допоможе уникнути непорозумінь і негативного сприйняття під час використання програми.

Вирішальним кроком у забезпеченні стабільності та розвитку веб-програми є розширення її інфраструктури. Важливим елементом цього процесу є створення додаткових середовищ Amazon Web Services (AWS) EC2. Ця диференціація дозволяє чітко відокремити виробниче середовище, де користувачі працюють із ретельно

перевіреним і надійним кодом, і середовище розробки, призначене для розробників, щоб представити та оцінити нові функції. Наявність окремих середовищ має кілька переваг:

– Стабільність користувача: гарантує, що кінцеві користувачі можуть покладатися на послідовне та надійне середовище, мінімізуючи ймовірність проблем або помилок, пов'язаних з оновленнями.

– Безпека в розробці: надає розробникам можливість впроваджувати інновації та тестувати нові функції, не ставлячи під загрозу виробниче середовище, тим самим зменшуючи ймовірність непередбачених помилок.

– Ефективність оновлень: процес розгортання спрощено завдяки можливості розробникам спочатку тестувати модифікації в окремому середовищі, перш ніж застосовувати їх у робочому середовищі.

– Масштабованість: AWS EC2 пропонує адаптивні функції масштабування, які мають вирішальне значення для задоволення зростаючої кількості користувачів і підвищених вимог до ресурсів.

– Розгортання та контроль версій: це допоможе автоматизувати процес розгортання та керування контролем версій, що призведе до більш ефективного та плавного розгортання оновлень. Ретельне планування та нагляд знадобляться для реалізації цього поділу середовищ на AWS EC2, гарантуючи досягнення синхронізації між середовищами та ефективне управління ресурсами. Тим не менш, ці заходи мають вирішальне значення для збереження надійності та якості веб-додатку з часом.

Висновки до розділу 3

1. Функціонування розробленого веб-додатку проілюстровано як з точки зору сервера, так і з боку користувача. Надається ретельний аналіз взаємодії між серверними та користувальницькими компонентами, а також огляд базової кодової бази.

2. Огляд зосереджений на інтеграції OpenAI і Google API, зокрема на розробку та функціональність помічників генерації коду, а також на команду,

присвячену аналізу відгуків користувачів. Розглядаються різні форми зворотнього зв'язку, і демонстрація ілюструє, як цей зворотній зв'язок включено в інтерфейс користувача.

3. Було встановлено кілька додаткових напрямків розвитку продукту. Проводиться ретельний аналіз важливості створення стратегії монетизації та інтеграції відповідних функцій у веб-додаток. Також було встановлено, що необхідно включити різні компоненти, щоб дозволити користувачам персоналізовувати свій веб-застосунок.

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	Електронебезпека, 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 небезпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об’єктивного критерію рівня небезпеки 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_1	Відсутність захисного заземлення	0,02
P_2	Пошкодження захисного заземлення	0,04
P_3	Спрацювання складових захисту	0,1
P_4	Неправильна експлуатація захисту	0,02
P_5	Відсутність профілактичних заходів	0,2
P_6	Відсутність захисного щита	0,12
P_7	Недотримання правил вибору взуття	0,15
P_8	Незнання правил техніки безпеки	0,1
P_9	Відсутність засобів індивідуального захисту	0,2
P_{10}	Легковажність	0,08

На основі наведених подій будуємо 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: $P13 = 0{,}2 + 0{,}4 - 0{,}2 \cdot 0{,}4 = 0{,}0592$

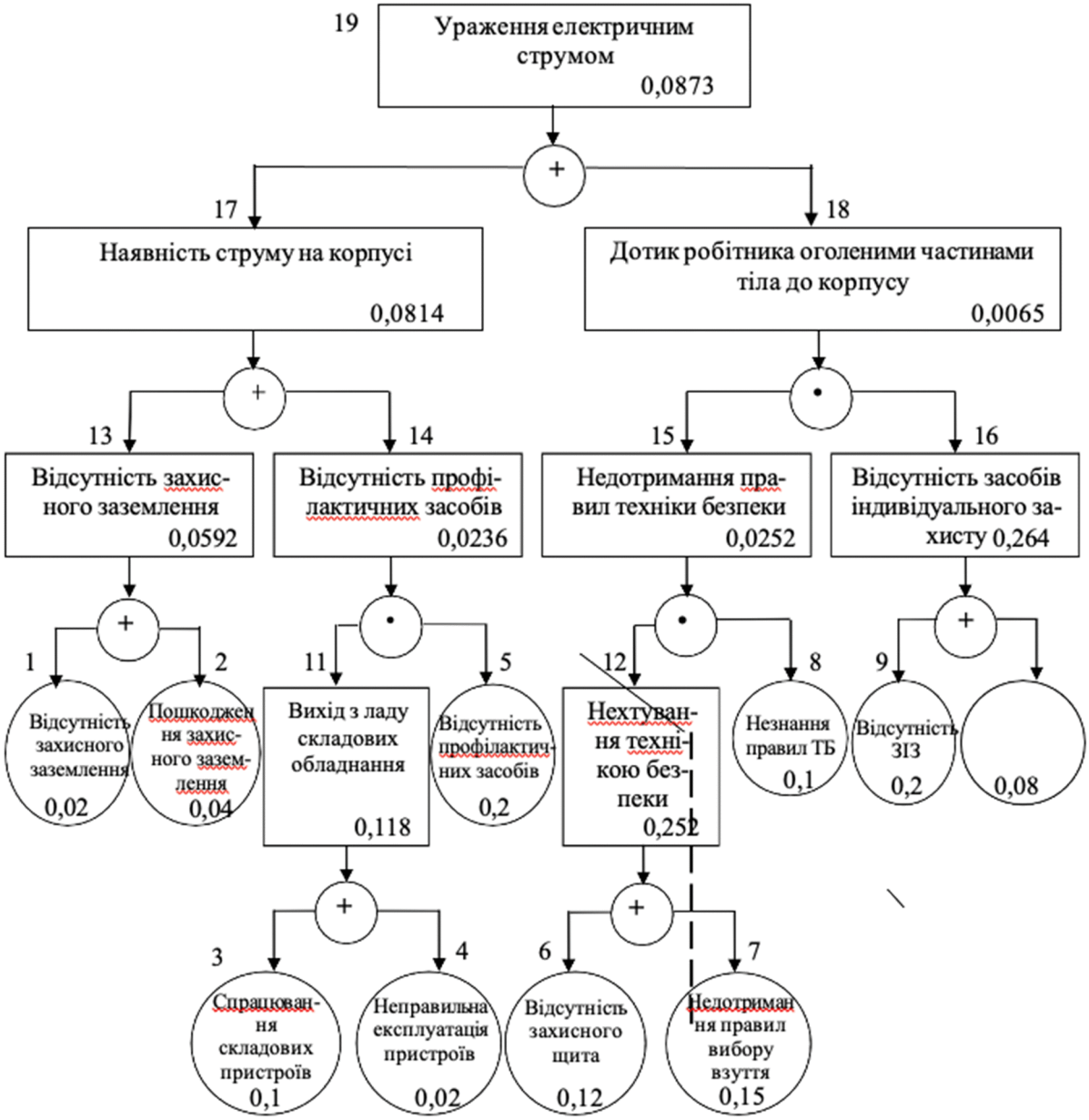

Рисунок 5.1 – Матриця логічних взаємозв’язків між окремими подіями травмонебезпечної ситуації

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 вид 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 захищенні 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 процей. Також, розроблено заходи для покращення умов праці.

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

5.1 Підходи до оцінювання ефективності веб-застосунку

Оцінка ефективності веб-застосунку, розробленого для магістерської роботи, має важливе значення для оцінення його загальної цінності та відповідності користувачам. Оцінка продуктивності охоплює різні критерії, що виходять за рамки одного аспекту й охоплюють все, від технічної ефективності до досвіду користувача.

Бажано починати з технічної ефективності, оскільки вона наголошує на таких елементах, як швидкість програми, час відповіді системи, точність генерації коду та загальна надійність. Це можна оцінити за допомогою ряду тестів, включаючи тестування навантаження та функціональне тестування, на додаток до аналізу продуктивності.

І навпаки, ефективність користувачів оцінюється шляхом аналізу UX/UI, збору відгуків користувачів і вивчення поведінки на веб-сайті. Важливо розуміти, наскільки інтуїтивно зрозумілою та зручною є програма для її користувачів.

Ключовим моментом у цьому аналізі є економічна ефективність, яка передбачає оцінку витрат, пов’язаних із розробкою та експлуатацією програми, по відношенню до отриманих переваг. Ця оцінка допомагає визначити, чи виправдовують отримані переваги інвестовані ресурси. Щоб досягти цього, важливо вивчити структури ціноутворення сторонніх API, які використовуються в програмі, зокрема OpenAI API та Google API.

Вирішальними факторами є масштабованість і адаптивність програм. Вони впливають на те, наскільки легко додаток може пристосуватися до мінливих вимог або змін у технічних характеристиках без значних додаткових витрат. У контексті розгортання розробленої веб-програми на хмарних службах, зокрема на платформі Amazon Web Services (AWS), масштабованість і адаптивність стають ще більш критичними. Використання AWS для розміщення додатків дає значні переваги щодо ефективності та гнучкості. AWS пропонує широкі можливості для налаштування та масштабування послуг відповідно до вимог проекту. Наприклад, ресурси можна легко

збільшити або зменшити за допомогою таких сервісів, як Amazon EC2 для обчислювальної потужності та Amazon RDS для керування базами даних, залежно від поточного навантаження. Ця можливість дозволяє додатку автоматично реагувати на збільшення кількості користувачів або даних, надаючи додаткові ресурси без необхідності вручну коригувати чи подальших фінансових інвестицій. Крім того, AWS має гнучкі структури ціноутворення, які дозволяють розробникам ефективно керувати витратами. За принципом оплати за використання ви платите лише за фактично використані ресурси. Особливо корисний для стартапів та ініціатив із змінним навантаженням, цей підхід забезпечує значну економічну ефективність, мінімізуючи ризики, пов'язані з оцінкою майбутніх потреб у ресурсах. Що стосується цінових структур AWS EC2 і AWS RDS, доступний різноманітний набір опцій, що обслуговує все, від базових екземплярів, придатних для невеликих проектів, до складних конфігурацій, розроблених для масштабних корпоративних додатків. Кожна служба представляє різні типи екземплярів, адаптовані для виконання конкретних завдань і потреб у продуктивності. Ця стратегія не тільки забезпечує технічну гнучкість програми, але й дозволяє точно коригувати вартість відповідно до бюджету проекту.

Комерційний успіх веб-програми значною мірою залежить від її цінової політики та планів. Потрібно враховувати витрати, пов'язані з розробкою та підтримкою, на які впливають обрані технології, зокрема React, Python, Django, Django REST framework та OpenAI API. Вибрані технології не лише впливають на вартість, але й визначають функціональні можливості, доступні користувачам, тим самим впливаючи на встановлення цін і створення тарифних планів.

Гнучкість цінової політики має вирішальне значення, оскільки її потрібно адаптувати до різних категорій користувачів. Цей підхід дозволяє додатку точно представляти справжню цінність, яку він пропонує, зберігаючи конкурентоспроможність і доступність для цільового ринку. Добре продумана стратегія ціноутворення не лише приносить дохід, але й демонструє цінність програми для її користувачів. Щоб досягти цього, важливо враховувати цінову політику використовуваних послуг, зокрема OpenAI API та AWS [36-39].

OpenAI API має кілька моделей, кожна з яких має різну вартість використання. Нижче ви знайдете таблиці з описом цінової політики для кожної категорії моделей у OpenAI API. Важливо відзначити, що наведені нижче ціни стосуються 1000 токенів, які можна розуміти як сегменти слів, причому приблизно 750 слів прирівнюються до 1000 токенів [40].

GPT-4 Turbo може похвалитися контекстом розміром 128 КБ, новішими знаннями та широким набором функцій, що робить його надійнішим, ніж GPT-4, і пропонується за зниженою ціною. У таблиці 5.1 подано зведення структури ціноутворення для цієї моделі.

Таблиця 5.1 Цінові стратегії 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 має широкі загальні знання та професійний досвід, що дозволяє йому точно вирішувати складні проблеми та виконувати складні інструкції природною мовою. У таблиці 5.2 наведено огляд цінової політики, пов’язаної з цією моделлю.

Таблиця 5.2 Стратегії ціноутворення 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

Моделі GPT-3.5 Turbo надійні та економічні. Флагманська модель цієї серії, gpt-3.5-turbo-1106, може похвалитися контекстним вікном 16K і створена для діалогу. На відміну від цього, gpt-3.5-turbo-instruct — це модель Instruct, яка підтримує лише контекстне вікно 4K. У таблиці 5.3 подано зведення цінової політики, пов’язаної з цією моделлю.

Таблиця 5.3 Цінові стратегії 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

API помічників разом із такими інструментами, як функція пошуку та інтерпретатор коду, спрощує розробникам процес інтеграції помічників AI у свої програми. Кожен помічник має власну пов'язану плату за зберігання файлів пошуку, яка визначається файлами, завантаженими до цього конкретного помічника. Інструмент пошуку ефективно фрагментує та індексує вміст файлів у векторній базі даних OpenAI. Виставлення рахунків за токени, що використовуються в допоміжному API, здійснюється за ставками введення/виведення за маркер вибраної мовної моделі, тоді як помічник вміло вибирає, який контекст із потоку включити під час виклику моделі. У таблиці 5.4 наведено огляд цінової політики для цієї моделі.

Таблиця 5.4 Стратегії ціноутворення для моделі помічника

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

Спеціальні моделі – ця функція дозволяє користувачам розробляти власні індивідуальні моделі, які можна коригувати за допомогою базових моделей разом із персоналізованими навчальними даними. Після точного налаштування з користувачів стягуватиметься плата виключно за токени, використані в їхніх запитах до конкретної моделі. У таблиці 5.5 наведено короткий виклад цінової політики, що стосується цієї моделі.

Таблиця 5.5 Стратегії ціноутворення для моделі коригування

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 – використовуйте пропозиції щодо вбудовування, щоб полегшити розширений пошук, кластеризацію, моделювання тем і класифікацію. У таблиці 5.6 наведено огляд цінової політики, пов'язаної з цією моделлю.

Таблиця 5.6 Цінові політики для моделі вбудовування

Модель	Використання
ada v2	$0.0001 / тис. токенів

Фундаментальні моделі – хоча моделям GPT бракує оптимізації для виконання інструкцій і вони мають обмежені можливості, вони все одно можуть бути досить ефективними, якщо їх точно налаштувати для конкретних завдань. У таблиці 5.6 наведено огляд цінової політики, пов'язаної з цією моделлю.

Таблиця 5.7 Цінові політики для базових моделей

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

Цінові стратегії та плани для AWS EC2 (Таблиця 5.8) і AWS RDS (Таблиця 5.9) наведені нижче, що забезпечує підтримку різних рівнів навантаження та продуктивності нашої веб-програми. Ці таблиці ілюструють, як збалансувати вартість і продуктивність відповідно до унікальних вимог програми та її користувачів. Включено таблицю з найдоступнішими об'єктами для AWS EC2, оскільки всього 695 об'єктів, що робить непрактичним оцінювати їх усі. Крім того, вибрані об'єкти задовольняють усі потреби програми на поточному етапі розробки.

Таблиця 5.8 Стратегії ціноутворення для ресурсів 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 Стратегії ціноутворення для компонентів 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 веб-застосунку.

У цьому розділі ми зосереджуємося на оцінці продуктивності розробленої нами веб-програми, досліджуючи ключові показники, зокрема швидкість роботи, точність генерації коду та економічну ефективність. Спочатку ми зосереджуємося на швидкості програми та часу відповіді системи, оскільки швидкість відповіді на запит є вирішальним елементом, що впливає на загальне задоволення користувачів. Аналізуючи та порівнюючи час відгуку нашого веб-додатку з часом відгуку ChatGPT та інших помічників GPT, ми можемо визначити, де ми перевершуємо, а де програємо в цьому аспекті. Щоб візуалізувати цю інформацію, ми використовуємо гістограми, які допомагають точно визначити потенційні можливості оптимізації. Спочатку ми представимо серію гістограм, що відображають середній час відгуку системи для двох мов програмування: Python (Рис. 5.1) і JavaScript (Рис. 5.2), разом із відповідними фреймворками та пакетами. Час запиту обчислюється з моменту відправлення користувачем запиту до повного створення тексту.

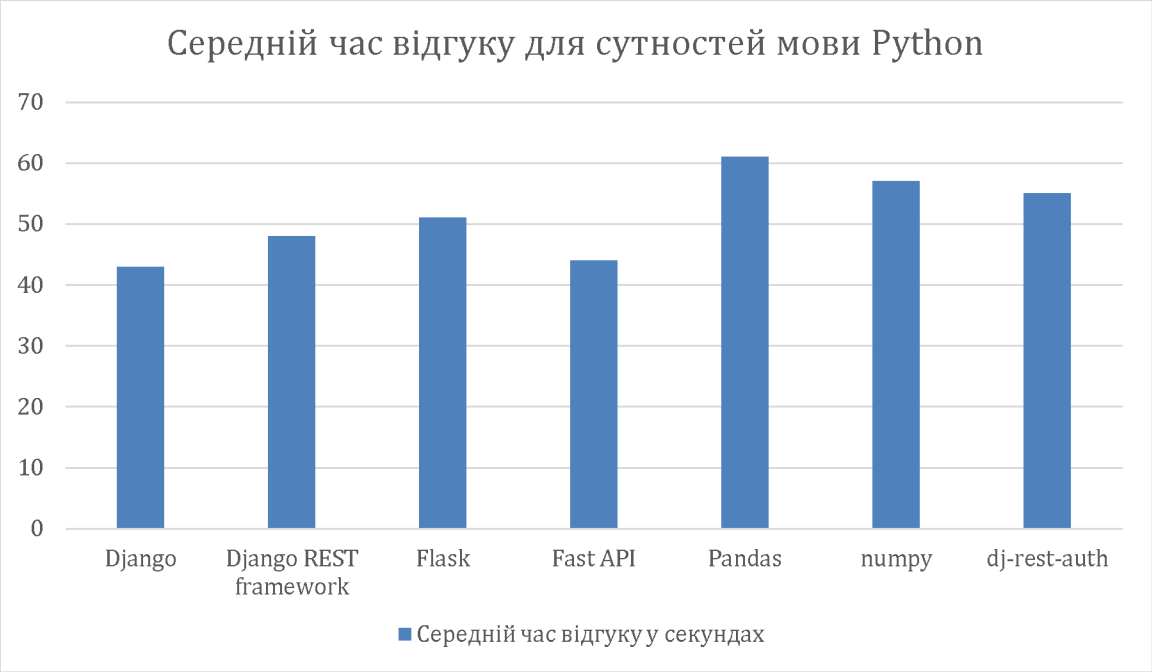

Рисунок 5.1 – Гістограма, що ілюструє середній час відповіді для об'єктів, пов'язаних з мовою програмування Python.

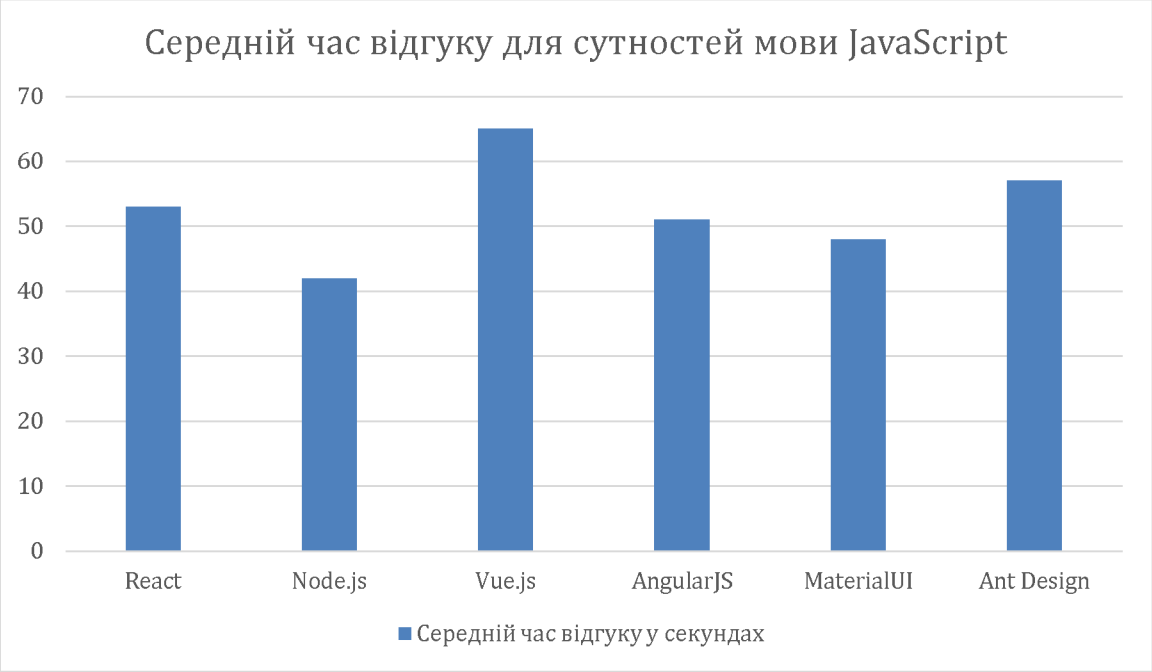

Рисунок 5.2 – істограма, що ілюструє середній час відповіді для об'єктів на мові програмування JavaScript.

Крім того, було проведено декілька вимірювань для оцінки часу відповіді на ідентичні запити у веб-додатку, створеному в рамках магістерської роботи, стандартному ChatGPT, що використовує модель GPT-3.5, і преміум-версії ChatGPT,

розробленій на основі тих самих інструкцій. використовується у веб-додатку. Оцінки проводилися з використанням мови програмування Python, фреймворку Django REST та dj-бібліотека rest-auth, призначена для інтеграції функцій автентифікації. Гістограма, що відображає час відповіді для кожного з трьох запитів, представлена на рисунку 5.3.

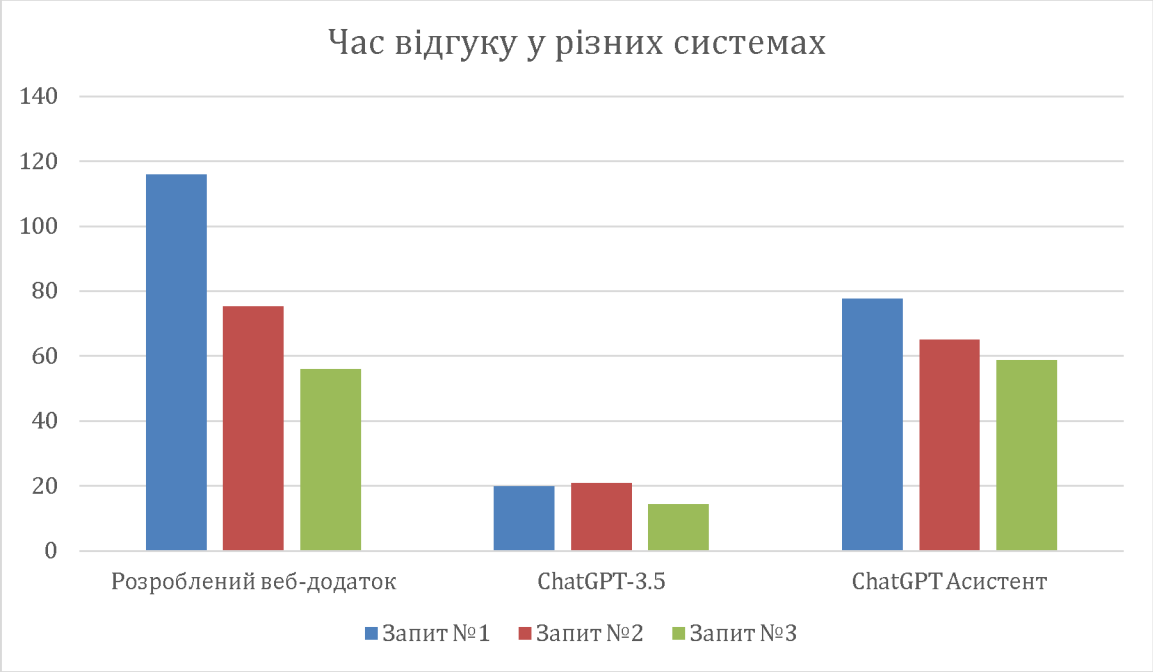

Рисунок 5.3 – Гістограма часу відповіді на різні системні запити.

Гістограма, представлена на малюнку 5.3, ілюструє, що час відгуку розробленої веб-програми значно повільніший, ніж у стандартного ChatGPT-3.5. Однак він більше відповідає часу відповіді помічника в преміум-версії ChatGPT. На графіку видно, що час відповіді на початковий запит особливо подовжений і перевищує 38 секунд. Різниця в часі відповіді на другий запит менш виражена, з розривом у 10 секунд. На третій запит час відповіді на 3 секунди коротший порівняно з преміум-асистентом ChatGPT. Незважаючи на те, що тенденція щодо часу відповіді для розробленої веб-програми поступається як ChatGPT, так і його преміум-колегі, важливо визнати, що цей аспект не повинен бути єдиним у центрі уваги, оскільки якість відповідей також має враховуватися.

Активність користувачів є ключовим показником, необхідним для вимірювання актуальності програми та привабливості для її користувачів. Це можна легко

контролювати за допомогою аналізу поточних об’єктів у базі даних разом із інструментами, пропонованими OpenAI. Нижче наведено декілька графіків які ілюструють активність використання.

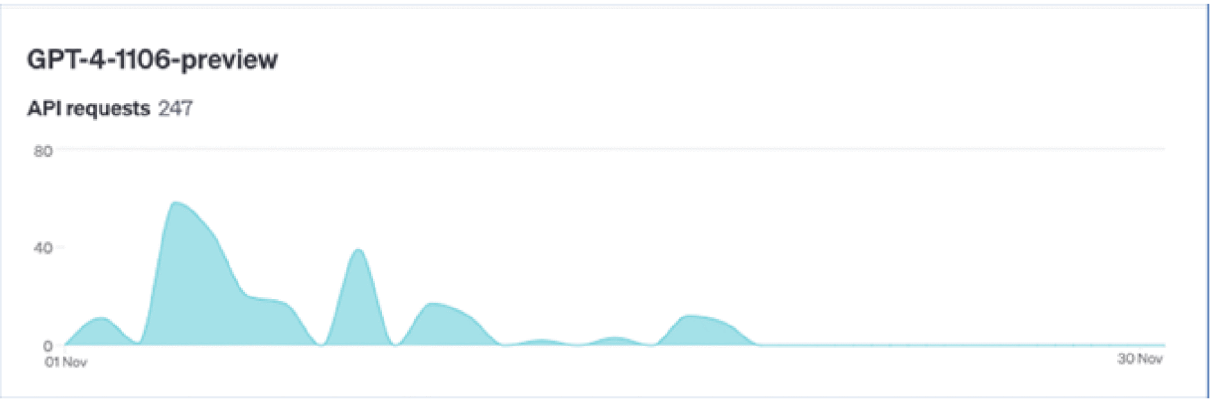

Рисунок 5.4 – Обсяг запитів, оброблених моделлю GPT-4-1106-preview протягом листопада.

Рисунок 5.5 – Кількість використаних токенів із моделлю попереднього перегляду GPT-4-1106 протягом листопада.

Веб-застосунок, який ми створили, має певні недоліки, зокрема непривабливий веб-дизайн, інтерфейс, якому не вистачає ретельного розгляду, і трохи довший середній час відповіді порівняно з преміум-версією помічника ChatGPT. Однак однією з основних переваг програми є її економічність, оскільки її операційні витрати в 5,5 разів нижчі, ніж у преміум-версії ChatGPT. Крім того, помітною перевагою є те, що користувачі платять лише за використання програми, тоді як преміальна підписка на ChatGPT коштує 20 доларів, незалежно від фактичного використання. Крім того, розроблений веб-додаток пропонує більшу зручність завдяки добре налагодженій базі

навчених помічників, які відповідають на більшість стандартних запитів користувачів.

Висновки до розділу 5

1. Встановлено методи оцінки ефективності веб-додатку. Був проведений аналіз стратегій ціноутворення послуг, які використовувалися при розробці веб-додатку, зокрема вивчено кожну існуючу модель та її варіанти, особливо щодо хмарних служб, таких як AWS EC2 і AWS RDS.

2. Було проведено поглиблену оцінку ефективності системи, приділяючи особливу увагу якості згенерованого коду, швидкості застосування та його економічним перевагам. Крім того, було проведено порівняння з веб-додатком ChatGPT, використовуючи різні версії: базову версію на основі моделі GPT-3.5 і преміум-версію, яка зосереджена навколо нещодавно розробленого персонального помічника.

ЗАГАЛЬНІ ВИСНОВКИ

1. Як компонент завершеного магістерського проекту було створено веб-застосунок, який використовує інтеграцію API від Google і OpenAI для покращення програмних запитів за допомогою оперативних методів розробки та функціональності асистента. Основні результати цієї роботи демонструють, що було розроблено унікальну систему, яка може пристосовуватися до інформації про програмні бібліотеки та фреймворки, що постійно змінюється, що гарантує високий рівень продуктивності та точності програмування.

2. При оцінці разом із конкурентами веб-застосунок демонструє помітне підвищення ефективності генерації коду. Крім того, він бюджетний, витрати на використання значно нижчі, ніж у комерційних пропозицій, що робить його доступним для різноманітних користувачів.

3. Отримані результати мають наукове значення завдяки всебічному дослідженню сучасних API та їхнього потенціалу для розширення можливостей програмування, що може прокласти шлях для майбутніх досліджень у цій галузі. На практичному рівні значення цієї роботи очевидно у створенні прототипу, який можна використовувати в реальних проектах, тим самим підвищуючи ефективність розробки програмного забезпечення.

4. Прогнози розвитку майбутніх досліджень передбачають інтеграцію більшої функціональності, особливо включення машинного навчання для автоматизації дедалі складніших завдань програмування, а також розширення підтримки кількох мов програмування.

5. Показники продуктивності веб-застосунку, включаючи час відгуку, точність генерації коду та залучення користувачів, підтверджують його сильну конкурентоспроможність. Результати експериментальних досліджень і випробувань розкривають значний потенціал для подальшого розвитку й 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

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)

24. “Мова програмування Node.js”. ULR: https://en.wikipedia.org/wiki/Node.js

25. “Мова програмування Java”. URL: https://en.wikipedia.org/wiki/Java_(programming_language)

26. “Що таке хмарні технології? Переваги та недоліки”. URL: https://edin.ua/shho-take-xmarni-texnologi%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