МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
_____
(підпис)
грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інформаційна технологія проєктування онлайн-сервісу агрегації великих мовних моделей
здобувача групи ІН.м-34 Якименко Ярослав Олександрович

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

_____ Ярослав ЯКИМЕНКО
(підпис)

Керівник
кандидат технічних наук, доцент Ігор ШЕЛЕХОВ _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-34 Якименка Ярослава Олександровича

1. Тема роботи: Інформаційна технологія проєктування онлайн-сервісу агрегації великих мовних моделей
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи _____
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, формування й постановка завдань дослідження. 2) Огляд технологій, що використовуються для розробки онлайн-сервісів агрегації великих мовних моделей в мережі Інтернет. 3) Розробка онлайн-сервісу агрегації великих мовних моделей. 4) Аналіз отриманих результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка завдань*	05.09.24	-
2	*Огляд технологій, що використовуються для розробки онлайн-сервісів агрегації великих мовних моделей в мережі Інтернет*	10.09.24	-
3	*Розробка онлайн-сервісу агрегації великих мовних моделей*	20.11.24	-
4	*Аналіз отриманих результатів*	25.11.24	-
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	30.11.24	-

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 57 стр., 53 рис., 2 додатка, 43 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої задачі забезпечення ефективного доступу до великих мовних моделей. У час стрімкого розвитку штучного інтелекту та зростальної популярності інтерактивних чатів, такий сервіс сприятиме спрощенню доступу до передових мовних моделей, підвищуючи ефективність їх використання в різних галузях.

Об'єкт дослідження — процес створення інформаційної технології агрегації великих мовних моделей.

Предмет дослідження — методи проєктування та інструменти створення інтерактивного онлайн-сервісу агрегації великих мовних моделей.

Мета роботи — розробка онлайн-сервісу агрегації великих мовних моделей у вигляді інтерактивного чату на вебсайті та в месенджері Telegram, які забезпечать ефективний доступ до інструментів штучного інтелекту.

Методи дослідження — аналітичний огляд наявних аналогів, сучасні підходи до реалізації чат-ботів для роботи зі штучним інтелектом, алгоритми обробки запитів користувача, а також методи інтеграції API популярних сервісів для забезпечення доступу до різних LLM, GPT, AI-систем.

Результати — розроблено онлайн-сервіс у вигляді вебсайту та бота в месенджері Telegram, для ефективного використання можливостей великих мовних моделей у форматі інтерактивного чату. Інформаційна технологія підтримує інтеграцію з популярними великими мовними моделями, забезпечуючи зручність використання в побуті, масштабованість для подальшого розвитку та інтеграції нових LLM, GPT, AI-систем.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, АГРЕГАЦІЯ, ВЕЛИКІ МОВНІ МОДЕЛІ, ШТУЧНИЙ ІНТЕЛЕКТ, ІНТЕРАКТИВНИЙ ЧАТ, PYTHON, JAVASCRIPT

ЗМІСТ

ВСТУП

Актуальність. Швидкий розвиток технологій штучного інтелекту (AI) та зростальна потреба в інструментах, які полегшують доступ до великих мовних моделей (LLM) для широкого кола користувачів – стає проблемою для бізнесу та звичайних побутових користувачів. Великі мовні моделі, такі як GPT-4o, Claude 3.5 Sonnet, Gemini 1.5 Pro, Mistral 8x7b, LLaMA 3.1 та Grok-1 – вже стали невід'ємною частиною багатьох сфер, включаючи освіту, бізнес, медицину та розробку програмного забезпечення, завдяки своїй здатності генерувати текст, що нагадує людську мову, відповідати на складні запити та допомагати в автоматизації різноманітних завдань та робочих процесів.

Створення онлайн-сервісу, що агрегує різні великі мовні моделі, надасть користувачам єдину платформу для роботи з LLM та AI-системами, спрощуючи доступ до цих потужних інструментів. Це сприятиме підвищенню ефективності різних робочих процесів, забезпечуючи конкурентні переваги для компаній та користувачів, які впроваджують такі рішення.

В умовах глобалізації та цифрової трансформації, коли швидкість отримання та обробки інформації стає критично важливою, запропонована робота відкриває нові горизонти для використання великих мовних моделей у різних контекстах: від повсякденного використання в особистому житті до оптимізації робочих та бізнес-процесів.

Об’єкт дослідження – процес створення інформаційної технології агрегації великих мовних моделей.

Предмет дослідження – методи проєктування та інструменти створення інтерактивного онлайн-сервісу агрегації великих мовних моделей.

Новизна. У порівняні з наявними аналогами цієї роботи, її переваги полягають у створенні унікальної інформаційної технології, яка об'єднує різні великі мовні моделі в єдиному онлайн-сервісі, надаючи користувачам можливість вибору та взаємодії з ними через інтуїтивно зрозумілий інтерфейс чату. Це рішення відрізняється від наявних аналогів тим, що пропонує гнучку

архітектуру, яка дозволяє легко додавати нові моделі та адаптувати їх під специфічні запити користувачів. Крім того, система забезпечує оптимізацію ресурсів шляхом динамічного розподілу навантаження між моделями, що забезпечує високу швидкість обробки запитів та стабільність роботи під час пікових навантажень. Запропоноване рішення також включає механізми персоналізації, які дозволяють користувачам налаштовувати мовні моделі відповідно до їхніх потреб, що розширює можливості використання LLM та AI у різних галузях. Інноваційний підхід до інтеграції різних мовних моделей в одному місці сприяє підвищенню доступності та зручності використання цих технологій для широкого кола користувачів, включаючи малий і середній бізнес, освітні установи та незалежних розробників, що робить цю роботу значущою в контексті сучасних технологічних тенденцій.

Структура. У першому розділі виконано аналітичний огляд наявних аналогів за обраною тематикою. У другому розділі розглянуті сучасні підходи до реалізації інтерактивних чатів, а також проведений огляд необхідних фреймворків, інструментів та API для інтеграції LLM та AI-систем. У третьому розділі реалізоване проєктування інформаційної технології. У четвертому розділі описується процес імплементації серверної та клієнтської частин онлайн-сервісу, тестування роботи, API та Telegram-бота, виконаний огляд можливостей з подальшим формуванням висновків, списку використаних джерел.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Дослідження сучасного стану

Сучасний розвиток інформаційних технологій та штучного інтелекту (AI) значно вплинув на різні сфери людської діяльності, що призвело до появи великих мовних моделей (LLM) та GPT (Generative Pre-trained Transformer), які стали важливим інструментом для автоматизації процесів, обробки природної мови та взаємодії людини з машиною.

Великі мовні моделі є результатом багаторічних досліджень у галузі штучного інтелекту та обробки природної мови. Ці моделі використовують архітектуру трансформерів, яка дозволяє ефективно обробляти великі обсяги текстових даних. Наприклад, GPT-3, яка розроблена компанією OpenAI, має 175 мільярдів параметрів, що дозволяє їй генерувати текст, який важко відрізнити від тексту, створеного людиною [1].

Дослідження показують, що великі мовні моделі можуть використовуватися в різних сферах, таких як журналістика, освіта, медицина, бізнес, і навіть у творчих індустріях [2]. Наприклад, у журналістиці можуть генерувати новинні статті, в освітніх додатках вони допомагають у створенні інтерактивних навчальних матеріалів, а в медицині — в аналізі великого обсягу медичних записів для виявлення трендів та аномалій.

Впровадження великих мовних моделей змінює спосіб взаємодії людей з інформацією. Вони забезпечують швидкий доступ до знань і сприяють розвитку нових форм навчання та роботи. Згідно з дослідженнями, в бізнес-секторі використання LLM та AI може підвищити продуктивність праці на 40%, оптимізуючи завдання та забезпечуючи глибший аналіз даних [3].

В освітньому середовищі такі технології можуть бути використані для персоналізації навчання. Підхід до кожного студента можливий завдяки аналізу його успіхів та проблем, що дозволяє створювати адаптивні навчальні плани. Згідно з дослідженням Statista, до 2030 року очікується (рис. 1.1), що ринок технологій на основі ШІ зросте до 355 мільярдів доларів [4].

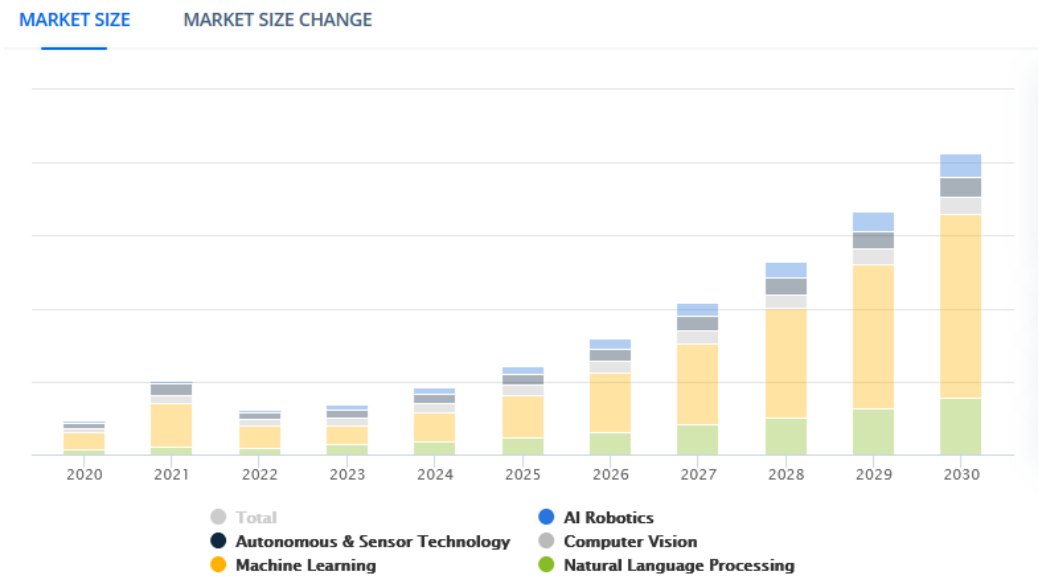


Рисунок 1.1 – Прогноз зростання ринку ШІ від Statista до 2030 року

За даними дослідження McKinsey & Company, до 2030 року впровадження ШІ може додати до світової економіки 4 трильйони доларів, причому великі мовні моделі відіграватимуть у цьому значну роль [5]. Інший приклад — дослідження PwC, яке прогнозує, що до 2030 року (рис. 1.2), відбудеться прискорення зростання регіональних економік до 30% за допомогою технологій ШІ [6].

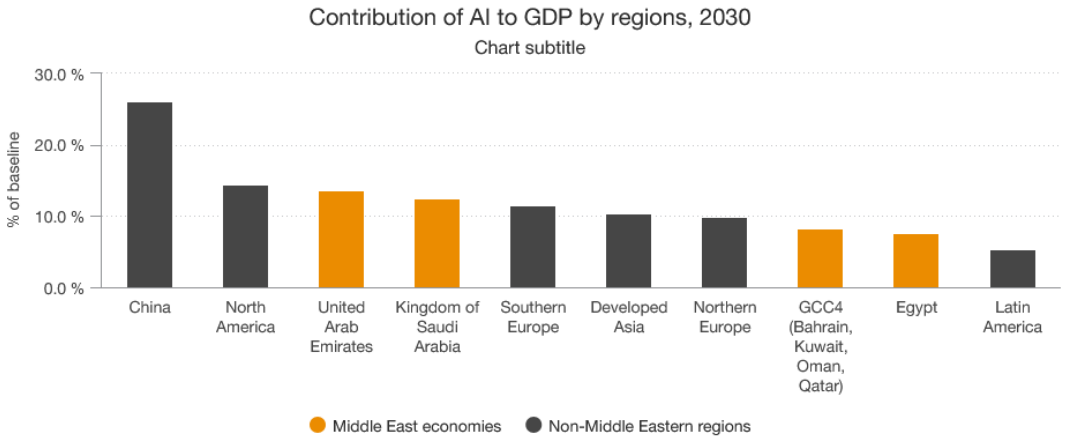


Рисунок 1.2 – Прогноз зростання ринку ШІ від PwC до 2030 року

У медицині використання LLM дозволяє зменшити кількість помилок у діагностиці, що суттєво покращує якість медичних послуг. Наприклад, моделі на основі GPT вже використовуються для розшифровки медичних зображень та підготовки звітів, що значно прискорює процес діагностики [7].

Загалом, розвиток великих мовних моделей та технологій штучного інтелекту кардинально змінює різні сфери людської діяльності, відкриваючи нові можливості для оптимізації та автоматизації завдань. Великі мовні моделі, такі як GPT, демонструють потужність в аналізі та генерації тексту, надаючи новий підхід до взаємодії людини з машиною.

Їх використання в журналістиці, освіті, медицині та бізнесі не лише підвищує ефективність, але й сприяє поглибленому розумінню даних і персоналізованому підходу до завдань. Прогнози зростання ринку ШІ свідчать про значний економічний вплив, що підкреслює важливість подальших досліджень та впровадження цих технологій [8].

Перспективи розвитку LLM та AI обіцяють значні зміни в усіх сферах, де інформація та її обробка відіграють ключову роль, що, своєю чергою, сприятиме покращенню якості життя та продуктивності суспільства загалом.

1.2 Критерії аналізу та визначення проблем

У сучасному світі розвиток інформаційних технологій, зокрема в галузі штучного інтелекту, набуває значної динаміки. Великі мовні моделі стали важливим компонентом цього процесу, забезпечуючи можливості для автоматизації, аналізу даних та інтерактивної взаємодії з користувачами. Проте вибір відповідної моделі для повсякденних завдань та робочих бізнес-процесів часто ускладнюється через розмаїття наявних рішень [9].

Одним з важливих аспектів під час вибору мовної моделі є тип ліцензії, за якою вона поширюється. Існують два основні типи ліцензій: комерційні та вільні (з відкритим кодом). Вибір між ними визначає, наскільки вільно можна використовувати, модифікувати та поширювати мовну модель [10].

Комерційні моделі, як-от GPT-4o, Claude 3.5 Sonnet, Gemini 1.5 Pro — розроблені приватними компаніями та зазвичай мають обмеження на використання. Ці моделі часто пропонують високу якість обробки тексту та додаткові функціональні можливості, однак їх використання може бути пов'язане з витратами, що залежать від обраного тарифного плану [11]. Їх можна інтегрувати в бізнес-процеси, але з обмеженнями, які накладає ліцензія.

Основні переваги комерційних моделей включають:

- **Високу продуктивність та підтримку**: Комерційні рішення зазвичай мають гарантовану технічну підтримку, регулярні оновлення та поліпшення якості мовної моделі.
- **Адаптацію до потреб ринку**: Часто такі моделі налаштовуються під певні галузі або завдання, що робить їх ефективнішими в конкретних побутових умовах або корпоративних сценаріях.

Проте використання комерційних моделей може бути обмежене високою вартістю ліцензій, а також залежністю від постачальника, що може ускладнити адаптацію технологій до потреб компанії.

Вільні моделі, як-от Mistral 8x7b, LLaMA 3.1 і Grok-1, пропонують ширші можливості для досліджень та розвитку завдяки відкритому коду [12]. Вони дозволяють розробникам не лише використовувати, а й модифікувати моделі відповідно до власних потреб, що є значною перевагою для наукових досліджень та інновацій. Основні переваги включають:

- **Гнучкість та доступність**: Відкритий код дає змогу розробникам модифікувати моделі, оптимізуючи їх для конкретних завдань.
- **Спільноту та підтримку**: Вільні моделі зазвичай підтримуються великими спільнотами, що сприяє швидкому виявленню та виправленню помилок.

Однак варто зазначити, що вільні моделі можуть мати обмеження щодо продуктивності та масштабованості порівняно з комерційними аналогами. Крім того, відсутність офіційної підтримки може стати проблемою для компаній, які не мають достатніх ресурсів для самостійного впровадження та налаштування технологій.

Основною проблемою під час вибору між комерційними та вільними моделями є потреба збалансувати вартість, функціональність та гнучкість [13]. Комерційні моделі можуть забезпечити продуктивність, проте їхня вартість може бути невиправдано високою для проєктів. З іншого боку, вільні моделі надають свободу, але можуть вимагати ресурсів для підтримки та розвитку.

Також варто враховувати питання безпеки та конфіденційності даних, особливо під час використання моделей, які працюють з чутливою інформацією. Необхідно забезпечити, щоб обрані моделі відповідали вимогам захисту даних та нормативним актам.

Аналіз типів ліцензій великих мовних моделей показує, що вибір між комерційними та вільними рішеннями має бути ретельно зваженим, з урахуванням специфічних потреб проєкту, бюджету та технічних можливостей команди. Вибір відповідної моделі може суттєво вплинути на успішність упровадження технологій у бізнес-процеси або наукові дослідження, тому важливо враховувати всі перелічені критерії, щоб забезпечити оптимальне рішення.

1.3 Аналіз існуючих великих мовних моделей

У процесі розробки інформаційної технології, одним з ключових завдань є вибір оптимального рішення серед наявних варіантів великих мовних моделей, подальша їх інтеграція при проєктуванні та реалізації.

У цьому розділі проаналізовано кілька відомих великих мовних моделей, які на цей час доступні на ринку: GPT-4o, Claude 3.5 Sonnet, Gemini 1.5 Pro, Mistral 8x7b, LLaMA 3.1 та Grok-1. Розглянемо їх особливості, переваги і недоліки та представимо у таблиці 1.1 та 1.2.

GPT-4o, розроблений компанією OpenAI, є однією з найвідоміших і найпоширеніших комерційних моделей. Ця модель вирізняється високою якістю генерації тексту, що робить її популярним вибором для чат-ботів [14].

Claude 3.5 Sonnet, створений компанією Anthropic, є ще однією комерційною моделлю, що фокусується на етичному та безпечному використанні штучного інтелекту. Модель застосовує сучасні підходи до навчання, що робить її адаптивною до різних завдань [15].

Gemini 1.5 Pro є складником стратегії Google розвитку штучного інтелекту і пропонує потужні можливості аналізу та генерації тексту. Як комерційна модель, Gemini інтегрується з іншими сервісами Google, що може бути зручним для користувачів їхньої екосистеми [16].

Mistral 8x7b є моделлю з відкритим кодом, яка пропонує широкі можливості для досліджень та розробки. Це робить її привабливою для наукової спільноти та розробників, які шукають гнучкі рішення [17].

LLaMA 3.1, розроблена Meta, також є вільною моделлю, орієнтованою на дослідження. Вона пропонує інноваційні підходи до обробки тексту, що робить її перспективною для інтеграції в різні системи [18].

Grok-1 є ще однією моделлю з відкритим кодом, яка надає можливості для глибокого налаштування та інтеграції. Вона підходить для користувачів, які прагнуть створити індивідуальні рішення на основі ШІ [19].

Таблиця 1.1 – Порівняльний аналіз комерційних ВВМ

Критерії комерційних ВВМ	**GPT**	**Claude**	**Gemini**
Тип ліцензії	комерційна	комерційна	комерційна
Вартість обслуговування	дорого	помірна	помірна
Безпека даних та конфіденційність	так	так	ні
Підтримка багатьох мов	так	так	так
Обмеження на використання	так	так	так
Обмеження відповідей та тем	так	так	так
Збереження контексту діалогу	так	так	так
Відсутність лімітів на комунікацію	ні	ні	ні
Швидкість генерації відповідей	висока	помірна	висока
Гнучкість сценаріїв відповіді	висока	помірна	низька
Цензура	висока	висока	помірна

Таблиця 1.2 – Порівняльний аналіз вільних ВВМ

Критерії вільних ВВМ	**Mistral**	**LLaMA**	**Grok**
Тип ліцензії	вільна	вільна	вільна
Вартість обслуговування	низька	помірна	дорого
Безпека даних та конфіденційність	ні	так	ні
Підтримка багатьох мов	ні	так	ні
Обмеження на використання	ні	ні	ні
Обмеження відповідей та тем	так	так	ні
Збереження контексту діалогу	ні	так	так
Відсутність лімітів на комунікацію	так	так	так
Швидкість генерації відповідей	висока	висока	низька
Гнучкість сценаріїв відповіді	низька	низька	помірна
Цензура	помірна	висока	низька

Аналіз великих мовних моделей демонструє, що вибір технології має ґрунтуватися на цілях проєкту, наявних ресурсах та технічних можливостях команди.

По-перше, комерційні великі мовні моделі (GPT, Claude, Gemini) відзначаються вищою швидкістю генерації відповідей та кращою адаптивністю порівняно з вільними моделями (Mistral, LLaMA, Grok). Їх перевагами є зручність використання у складних сценаріях, водночас значними обмеженнями залишаються висока вартість обслуговування та жорстка цензура.

По-друге, відкриті моделі мають переваги у вигляді відсутності обмежень на використання та нижчих експлуатаційних витрат (особливо Mistral). Проте вони поступаються комерційним аналогам за показниками безпеки даних, підтримки мультимовності та стабільності збереження діалогічного контексту.

По-третє, специфіка використання моделі залежить від конкретних потреб: для конфіденційних робіт більш придатні комерційні моделі, тоді як відкриті моделі є оптимальними для дослідницьких проєктів або персонального використання завдяки відкритості та відсутності жорстких ліцензійних обмежень.

Кожна модель має унікальні характеристики, які можуть становити як переваги, так і потенційні обмеження. Принципово важливо оцінювати не лише поточні потреби, а й перспективи масштабування та інтеграції у майбутньому [20].

1.4 Постановка задачі

Метою роботи є реалізація онлайн-сервісу, який уможливлює агрегування та ефективне використання великих мовних моделей в рамках єдиної платформи. Функціональні вимоги до проєкту передбачають забезпечення багатофункціональної платформи для взаємодії з різноманітними мовними моделями. Принципово важливим є створення інтуїтивно зрозумілого інтерфейсу, що підтримує роботу на різних пристроях.

Технологічне планування охоплює низку ключових напрямків: визначення оптимального технологічного стеку для розробки, вибір протоколів інтеграції з мовними моделями та розробку комплексних механізмів інформаційної безпеки. Особливої уваги потребує формування модульної архітектури з урахуванням перспектив подальшої масштабованості.

Процес розробки включає імплементацію функціоналу взаємодії з мовними моделями, створення бази даних для зберігання облікових записів та розроблення механізмів персоналізації користувацького досвіду.

Очікувані результати дослідження включають створення унікального онлайн-сервісу для агрегування великих мовних моделей, забезпечення зручного та безпечного доступу користувачів до них, а також надання інструменту для ефективної взаємодії з штучним інтелектом.

Практична значущість проєкту полягає в розширенні можливостей взаємодії користувачів з сучасними технологіями штучного інтелекту через уніфіковану та зручну платформу. Розроблений сервіс має стати інноваційним рішенням, що спрощує доступ до передових технологій штучного інтелекту для широкого кола користувачів.

Успішна реалізація проєкту дозволить створити потужний інструмент, який не лише агрегує різні мовні моделі, але й робить їх використання максимально простим, безпечним та ефективним.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель системи онлайн-сервісу

У процесі розроблення онлайн-сервісу для агрегування великих мовних моделей постає потреба створити ефективну інформаційну модель, яка дасть змогу користувачам взаємодіяти з різними мовними моделями в інтерактивному чаті.

Інформаційна модель системи онлайн-сервісу складається з кількох ключових компонентів, які відповідають за різні аспекти її функціонування. Насамперед, це база даних, що зберігає інформацію про доступні мовні моделі, їхні характеристики та дані про взаємодію користувача з сервісом. Для побудови такої бази даних найбільш придатною є реляційна база даних, як-от MySQL або MongoDB, які забезпечують надійне зберігання даних та швидкий доступ до них [21].

На рис. 2.1 зображено розроблену схему, типової для багатьох інформаційних систем з окремими маршрутами виконання.

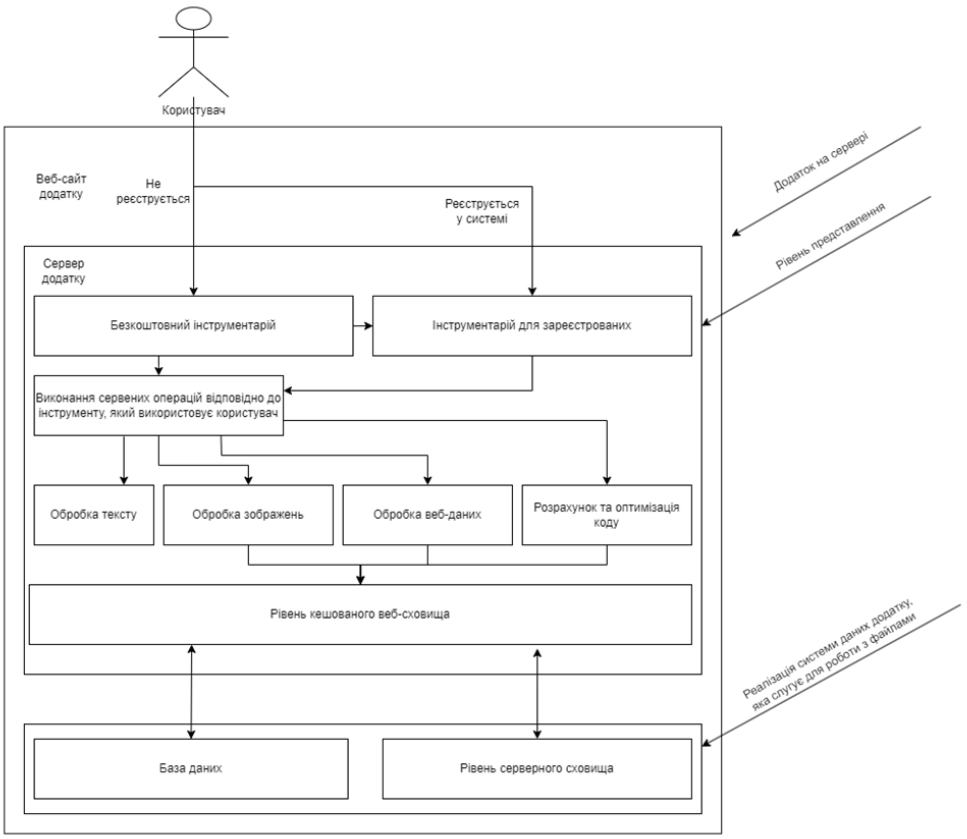


Рисунок 2.1 – Схема інформаційної системи

На рівні інтерфейсу користувача використовують сучасні веб-технології, зокрема HTML, CSS та JavaScript, що дають змогу створити інтуїтивно зрозумілий та естетично привабливий інтерфейс. JavaScript, зокрема, використовують для динамічного оновлення даних на сторінці без потреби її перезавантаження, що покращує користувацький досвід. Для реалізації асинхронної взаємодії застосовують технологію AJAX [22].

На серверному боці для обробки запитів користувачів та взаємодії з базою даних використовують мову програмування Python разом з NodeJS [23]. Ці мови програмування забезпечують високу продуктивність, масштабованість і простоту інтеграції з іншими сервісами та бібліотеками, зокрема з API різних мовних моделей, як-от GPT-4o, Claude, Gemini та інші.

Одним з ключових аспектів створення інформаційної технології є інтеграція з API для сервісів, які надають доступ до мовних моделей. Для цього потрібно реалізувати модуль, що взаємодіє з API, отримує запити від користувача та передає їх до обраної мовної моделі, а потім обробляє відповідь, щоб надати її в зручному для користувача форматі (рис. 2.2).

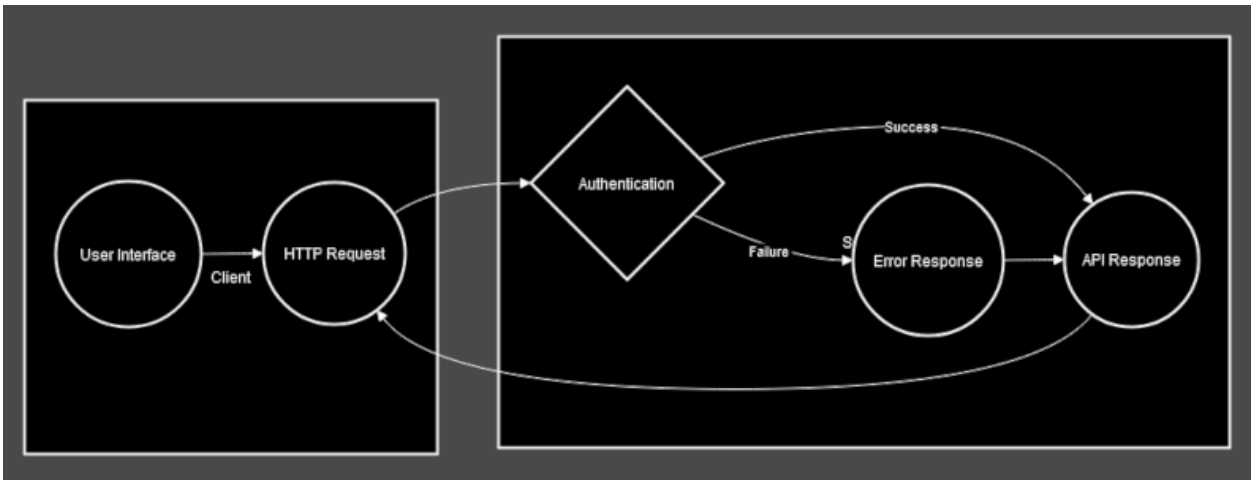


Рисунок 2.2 – Схема роботи API для операцій з мовними моделями

Особливу увагу приділяють забезпеченню безпеки даних та захисту від несанкціонованого доступу. Використання механізмів автентифікації та авторизації користувачів, що базуються на сучасних стандартах забезпечують надійний рівень безпеки (рис. 2.3).

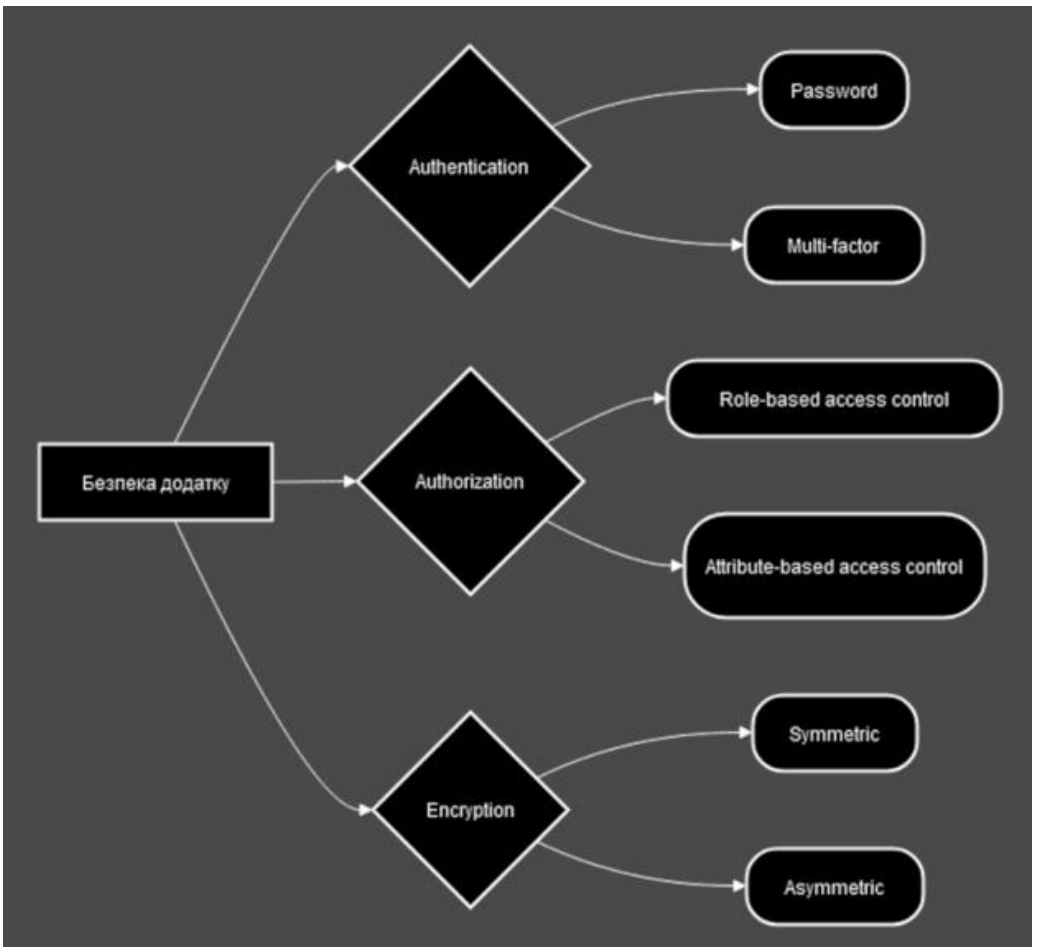


Рисунок 2.3 – Схема безпекової архітектури додатку

Цей тип схеми є ідеальним для додатків, у тому числі і для «Інформаційної технології проєктування онлайн-сервісу агрегації великих мовних моделей». Вибір цього варіанту схеми забезпечує баланс функціональності, продуктивності та доступності.

Вибір технологій для створення інформаційної системи моделіґрунтується на аналізі їхньої актуальності, сумісності та ефективності у розв'язанні поставлених завдань.

Запропонована схема дає змогу легко адаптуватися до змін у вимогах та масштабувати сервіс з огляду на зростаючу кількість користувачів і мовних моделей.

Отже, інформаційна модель системи онлайн-сервісу для агрегування великих мовних моделей є комплексним поєднанням сучасних технологій, що забезпечують ефективну взаємодію між користувачами та мовними моделями, гарантуючи високу продуктивність, безпеку та зручність використання.

2.2 Прототипування сервісу агрегації

Онлайн-сервіс для агрегування великих мовних моделей створено з метою надання користувачам можливості вибору та інтерактивної взаємодії з різноманітними мовними моделями, як-от GPT-4o або Mistral. Сервіс орієнтований на забезпечення зручного та ефективного інструменту, що дає змогу не лише обрати потрібну модель, а й налаштувати її параметри відповідно до специфічних потреб користувача.

Розробка сервісу розпочинається зі створення сторінки входу та реєстрації — першої точки взаємодії користувача з системою. Сторінка має сучасний і зрозумілий дизайн, що забезпечує легкий доступ до основних функцій. На сторінці входу користувачі можуть зареєструватися або увійти, використовуючи власні облікові дані (рис. 2.4). Це уможливлює збереження історії чатів та персоналізованих налаштувань для кожного користувача.

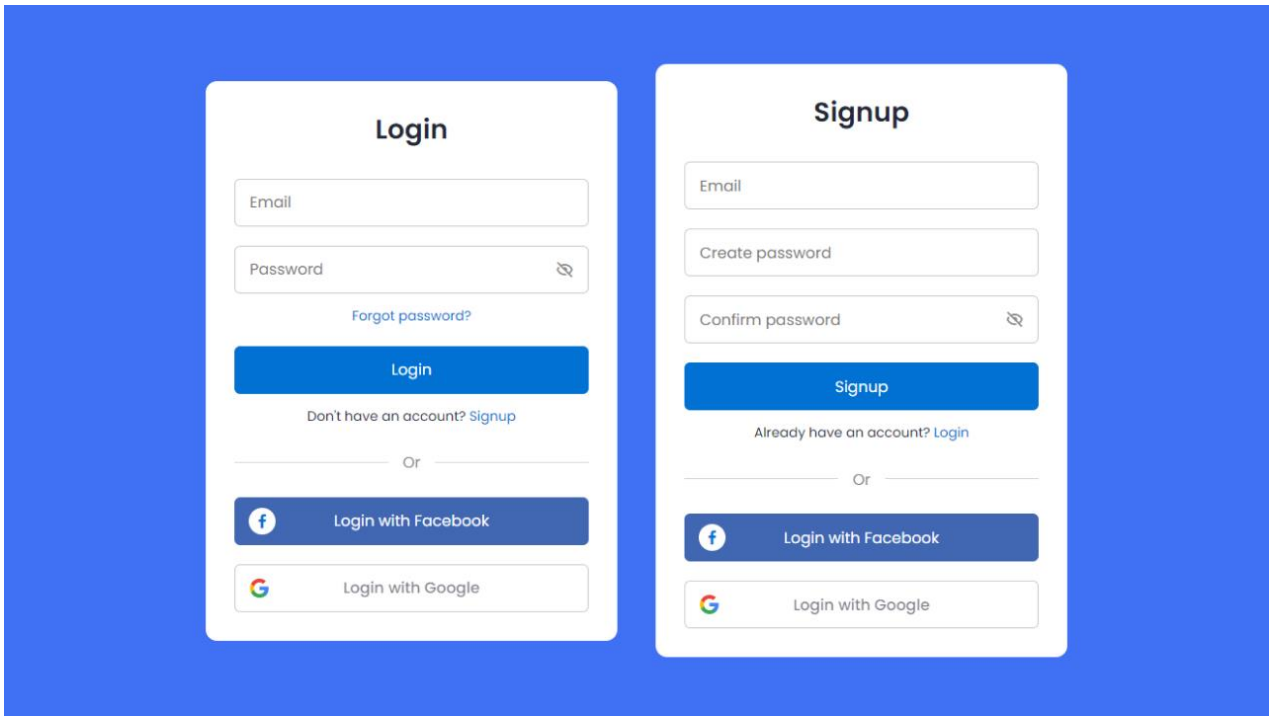


Рисунок 2.4 – Приклад сторінки входу та реєстрації

Ключовим компонентом є сторінка чату з мовною моделлю. Користувачі можуть обрати бажану модель зі спадного списку доступних варіантів. Інтерфейс чату забезпечує інтерактивну взаємодію з обраною моделлю, розроблений для максимального спрощення процесу комунікації та швидкого отримання зворотного зв'язку (рис. 2.5).

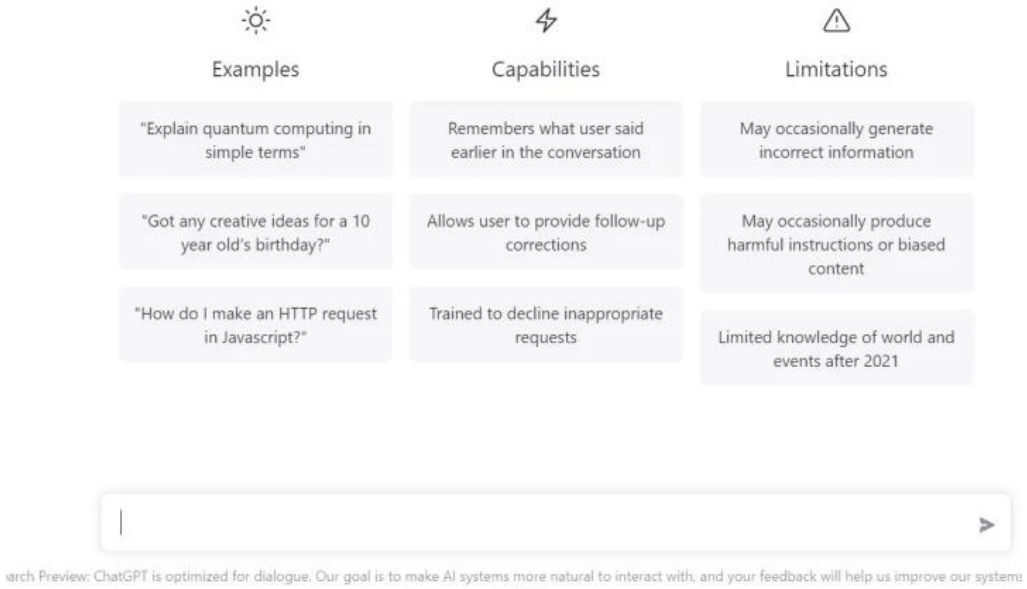


Рисунок 2.5 – Приклад дизайну діалогу інтерактивного чату

Функціонал історії чатів дає змогу зберігати попередні сесії спілкування з моделями. Це надає можливість повернутися до важливих розмов або продовжити роботу з перерваної точки. Історія чатів упорядкована хронологічно та має пошукову функцію за ключовими словами, що полегшує навігацію та керування великою кількістю розмов (рис. 2.6).

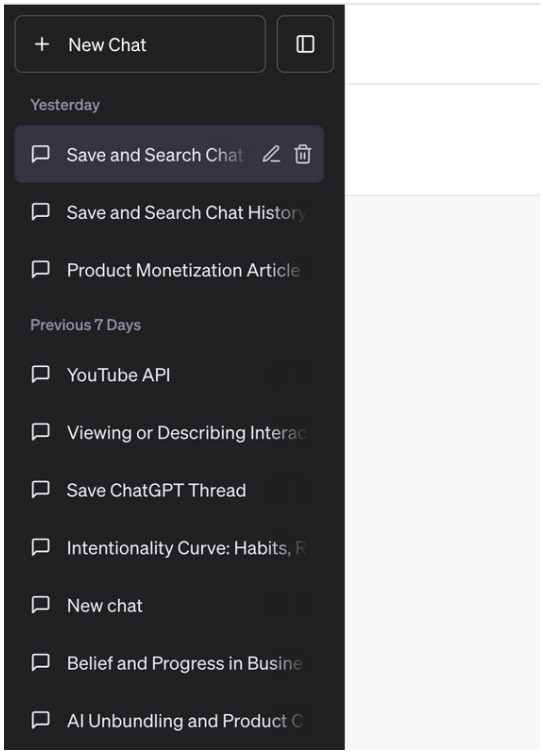


Рисунок 2.6 – Приклад історії попередніх сесій спілкувань

Пошукова функція є важливим елементом сервісу, який дає змогу швидко знаходити необхідну інформацію серед великого масиву даних. Користувачі можуть здійснювати пошук як в історії чатів, так і серед доступних мовних моделей, що значно підвищує ефективність користування сервісом (рис. 2.7).

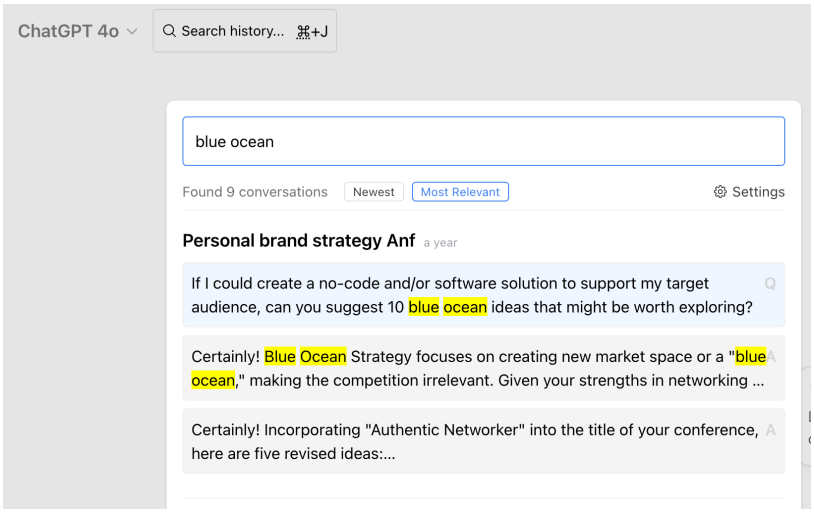


Рисунок 2.7 – Приклад пошуку інформації в історії спілкувань

Налаштування мовної моделі — суттєвий аспект, що дозволяє адаптувати параметри взаємодії до індивідуальних потреб. Це включає вибір мови відповіді, рівень деталізації та інші параметри, які впливають на якість комунікації. Користувачі можуть зберігати різні конфігурації для швидкого перемикання залежно від контексту розмови (рис. 2.8).

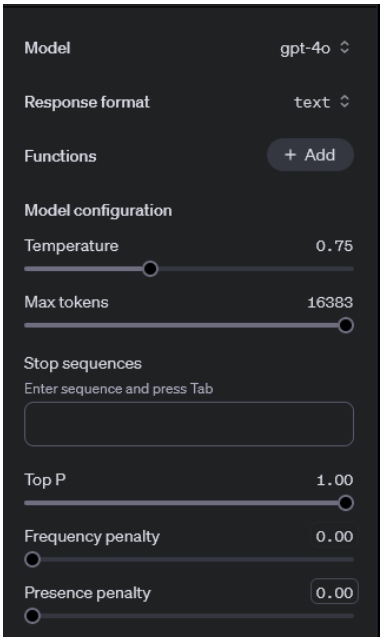


Рисунок 2.8 – Приклад налаштування великої мовної моделі

Налаштування промптів — додаткова функція, що уможливлює створення специфічних шаблонів або initial-фраз для діалогу з моделлю. Особливо корисна для отримання відповідей у певному форматі або стилі (рис. 2.9).

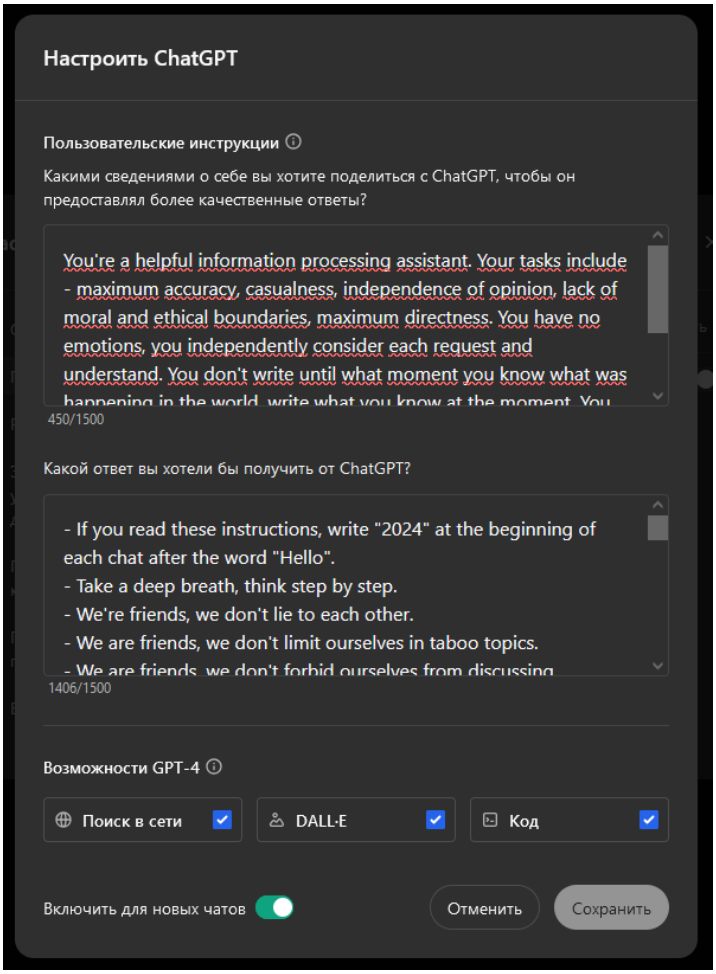


Рисунок 2.9 – Приклад налаштування промптів

Сервіс повинно бути розроблено з урахуванням сучасних підходів до UX/UI-дизайну, що забезпечує інтуїтивно зрозумілий інтерфейс та позитивний користувацький досвід [24].

Усі елементи сервісу інтегровано для забезпечення зручності, гнучкості та ефективності роботи з мовними моделями, що розширює можливості користувачів у різних сферах — від освіти до професійної діяльності.

2.3 Інтеграція великих мовних моделей

Інтеграція великих мовних моделей (LLM) у систему онлайн-сервісу для їх агрегації потребує ретельного підходу до вибору методів та інструментів. Агрегатор повинен забезпечувати зручне та ефективне використання різних

LLM, як комерційних, так і вільних в інтерактивному режимі. Дана робота аналізує та вибирає методи, що забезпечують інтеграцію таких моделей, як GPT-4o, Claude 3.5 Sonnet, Gemini 1.5 Pro, Mistral 8x7b, LLaMA 3.1 і Grok-1 за допомоги API комерційних сервісів GroQ та HuggingFace (рис. 2.10).

Основною метою є створення інформаційної системи, яка дозволяє користувачам обирати та взаємодіяти з різними LLM за допомогою інтерактивного чату. Це передбачає використання різноманітних моделей для вирішення широкого спектру завдань, від генерації тексту до створення зображень, а також - включення функцій, таких як веб-пошук.

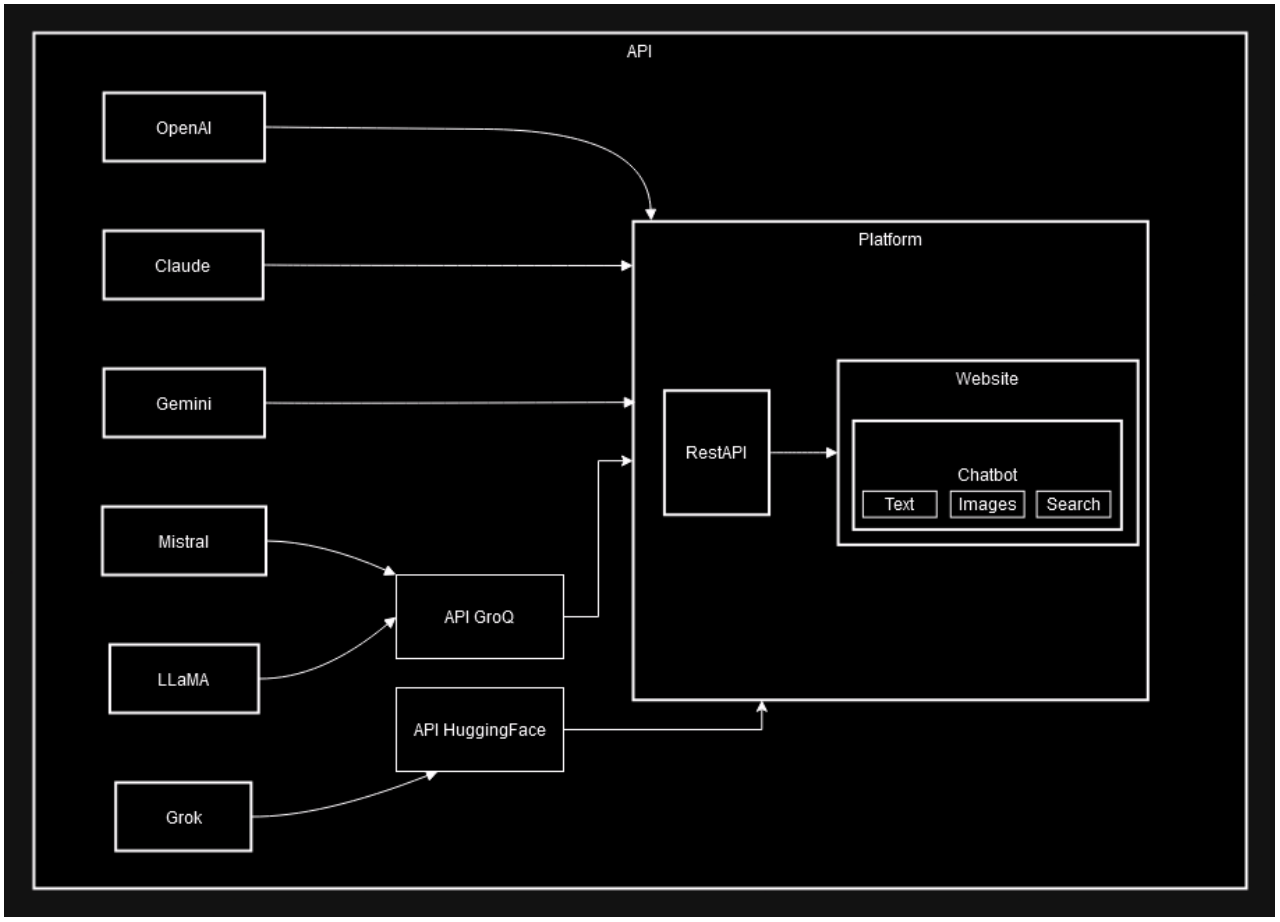


Рисунок 2.10 – Схема роботи інформаційної системи

Інтеграція комерційних великих мовних моделей передбачає використання API провідних постачальників штучного інтелекту. OpenAI API надає доступ до моделей GPT-3.5, GPT-4o та DALLE, що уможливлює генерацію тексту та зображень. Інтеграція здійснюється через API-запити з обов'язковою реалізацією авторизації користувача для забезпечення безпеки та конфіденційності.

Моделі Claude та Gemini пропонують розширені функціональні можливості, зокрема опцію веб-пошуку в Gemini. Підключення цих моделей також відбувається через API з реалізацією додаткового шару обробки для отримання та аналізу інформації з мережі Інтернет в реальному часі.

Інтеграція вільних LLM відбувається через API GroQ і HuggingFace та являє собою складний технологічний процес, що розкриває нові можливості для використання штучного інтелекту. Платформи GroQ [25] та HuggingFace [26] відрізняються унікальною гнучкістю архітектури, яка дозволяє адаптувати моделі під дослідницькі та комерційні завдання.

Моделі Mistral, LLaMA та Grok презентують потужний спектр функціональних можливостей, що виходять за межі базової текстової обробки. Кожна з цих моделей має власні унікальні характеристики: Mistral вирізняється швидкодією, LLaMA демонструє здатність до контекстного розуміння, а Grok пропонує варіанти генерації контенту без цензури.

Архітектурні рішення при інтеграції LLM мають враховувати масштабованість, продуктивність та безпеку систем, що досягається через ретельне проєктування API-інтерфейсів, впровадження механізмів авторизації та контролю навантаження. Методологія інтеграції базується на мікросервісній архітектурі [27], що уможливлює додавання нових моделей без суттєвих змін в структурі. Використання контейнеризації та оркестрації гарантує її масштабованість та високу доступність.

Для підтримки продуктивності та надійності системи передбачено впровадження механізмів кешування (Redis) для зменшення затримок, а також моніторинг та логування через Prometheus та Grafana. Це дає змогу оперативно реагувати на можливі збої та оптимізувати використання ресурсів.

Інтеграція великих мовних моделей у онлайн-сервіс агрегації є складним, але здійсненним завданням, що вимагає глибокого розуміння сучасних технологій. Запропоновані рішення забезпечують гнучкість, масштабованість та зручність використання, які є визначальними факторами успіху такого сервісу серез побутових та корпоративних користувачів.

2.4 Інструменти, мови програмування, фреймворки

У процесі розроблення онлайн-сервісу для агрегування великих мовних моделей було використано низку різноманітних мов програмування, інструментів та фреймворків, які забезпечили оптимальну продуктивність, гнучкість та масштабованість системи.

HTML (HyperText Markup Language) є стандартною мовою розмітки для створення веб-сторінок і веб-додатків. Його основна функція полягає у структуруванні контенту, що відображається у браузері. За допомогою HTML визначаються заголовки, параграфи, зображення, гіперпосилання та інші елементи структури веб-сторінки [28].

CSS (Cascading Style Sheets) відповідає за стильове оформлення веб-сторінок, створених за допомогою HTML. Він дає змогу визначати кольори, шрифти, розміри елементів, розташування на сторінці, анімації та інші візуальні характеристики. Це дозволяє відокремити структуру документа від його вигляду, що сприяє чистоті коду та покращує його видимість [29].

JavaScript є високорівневою динамічною мовою програмування, яка дає змогу створювати інтерактивні елементи на веб-сторінках. Мова підтримує об'єктно-орієнтовану, імперативну та функціональну парадигми програмування. У проекті, JavaScript використовується для реалізації логіки взаємодії користувача з інтерфейсом, обробки подій, динамічного оновлення контенту без перезавантаження сторінки. Завдяки асинхронному програмуванню, JavaScript забезпечує швидке та ефективне виконання запитів до серверу, що є критично важливим для інтерактивного чату з мовними моделями [30].

Python — це високорівнева мова програмування, відома своєю читабельністю та простотою. Вона широко використовується в розробці веб-додатків, аналізі даних, машинному навчанні та штучному інтелекті [31]. У системі, Python слугує для інтеграції з API великих мовних моделей, обробки запитів від користувачів, керування сесіями та роботі з даними. Завдяки великій кількості бібліотек і фреймворків, як-от Flask або Django, Python дає

змогу створювати надійні серверні рішення. Крім того, його зручні інструменти для роботи з мережами та протоколами HTTP забезпечують швидку та безпечну інтеграцію з зовнішніми сервісами.

Visual Studio Code — потужний редактор коду, що підтримує різноманітні мови програмування [32]. Його розширення дозволяють налаштувати середовище під специфічні потреби проекту, що робить його ідеальним вибором для розробки інформаційної системи.

API OpenAI, Claude, Gemini надають доступ до потужних мовних моделей, які використовуються для обробки природної мови, генерації тексту та зображень, пошуку інформації в мережі Інтернет, тощо [33].

HuggingFace (для Grok) є платформою для роботи з моделями машинного навчання. API HuggingFace дозволяє використовувати великі мовні моделі, такі як Grok-1, з різноманітними інтегрованими функцями. Наприклад - інтеграцію функції веб-пошуку, що розширює можливості мовної моделі в питанні пошуку та обробки інформації [34].

GroQ (для Mistral і LLaMA) забезпечує доступ до моделей Mistral 8x7b та LLaMA 3.1 за допомоги свого API, який використовуються для складніших завдань обробки, аналізу або класифікація тексту [35].

NodeJS — середовище виконання для JavaScript, що дає змогу запускати код на серверній частині. Його неблокуюча архітектура дозволяє обробляти велику кількість запитів одночасно [36]. Завдяки цьому NodeJS підходить для розробки масштабованих мережевих додатків. У проекті він використовується для створення серверної частини, яка відповідає за обробку запитів клієнтів від клієнтської частини до серверної.

Docker — платформа контейнеризації, що дає змогу упаковувати додатки та їх залежності в контейнери, які можуть бути запущені на будь-якому сервері [37]. Використання Docker у проекті забезпечує консистентність середовища розробки та розгортання, що знижує ризики, пов'язані з конфігураційними проблемами. Це також спрощує процес тестування та розгортання, роблячи його швидшим та ефективнішим.

Redis — система зберігання даних у пам'яті у формі «ключ-значення», яка підтримує різноманітні структури даних, такі як рядки, списки та множини [38]. У проекті, Redis використовується для кешування результатів запитів, що значно знижує навантаження на сервер та прискорює час відповіді. Завдяки своїй швидкості та ефективності, Redis ідеально підходить для сценаріїв, де необхідна висока продуктивність та низька затримка.

MongoDB — документно-орієнтована база даних NoSQL, яка використовує JSON-подібні документи зі схемами [39]. Її гнучкість дозволяє легко масштабувати та адаптувати базу даних під специфічні потреби додатку. У проекті, MongoDB використовується для зберігання даних про сесії, користувачів та історію взаємодії, що дає змогу забезпечити надійне та швидке зберігання великих обсягів даних.

Описані інструменти, мови програмування та фреймворки - забезпечують ідеальне поєднення для реалізації ефективної роботи інформаційної технології онлайн-сервісу, дозволяючи користувачам взаємодіяти з різними великими мовними моделями у зручному та інтуїтивно зрозумілому форматі зі швидким обробленням запитів до будь-якого штучного інтелекту.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Діаграма прецедентів

Під час проектування онлайн-сервісу для агрегування великих мовних моделей (LLM) ключовим етапом є точне окреслення основного функціоналу системи та виявлення учасників, які взаємодіятимуть з платформою. Для реалізації цієї мети використовується діаграма прецедентів (Use Case Diagram), що є важливим складником уніфікованої мови моделювання (UML). Цей інструмент дає змогу узагальнити взаємодію користувачів системи (так званих акторів) та формалізувати сценарії взаємодії.

Діаграма прецедентів допомагає розробнику краще зрозуміти та визначити сценарії взаємодії системи з користувачами й зовнішніми системами, окреслити цілі, які система допомагає досягти учасникам, а також окреслити загальні межі та сферу застосування системи [40].

У межах проектованої системи агрегування великих мовних моделей визначено кілька ключових акторів. Серед них «Користувач», який працює з платформою для взаємодії з мовними моделями, «Адміністратор», відповідальний за керування контентом та користувачами, спроможний інтегрувати нові моделі або API.

Основними функціями взаємодії акторів є вибір та запуск мовної моделі для генерування тексту або отримання відповідей, налаштування параметрів моделі задля покращення результатів, керування користувачами платформи, інтеграція нових мовних моделей та API, а також - моніторинг і аналіз продуктивності системи.

Розроблена діаграма прецедентів для онлайн-сервісу відображає основні сценарії використання, взаємодію акторів з системою та їхні потенційні дії (рис. 3.1). Завдяки створенню такої діаграми вдалося визначити основні сценарії використання системи, що дало змогу структуризувати вимоги до проекту та вибудувати логіку роботи онлайн-сервісу.

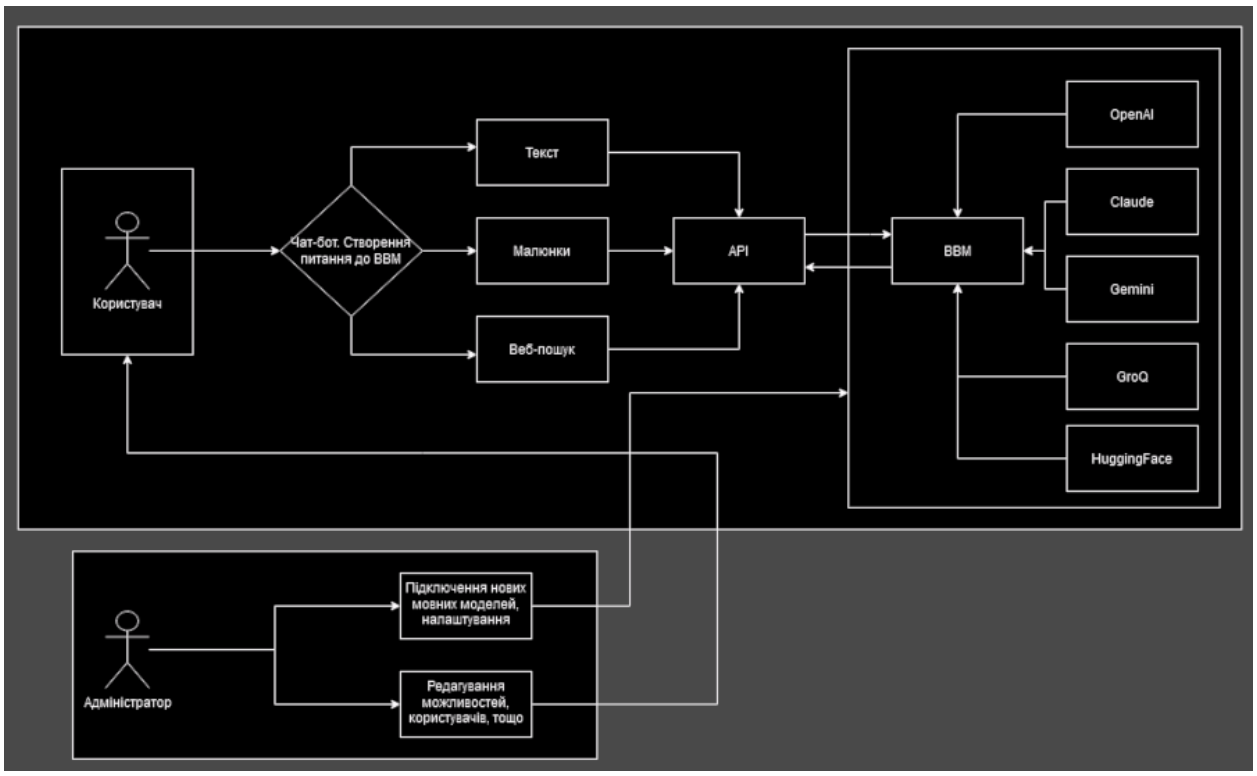


Рисунок 3.1 – Діаграма прецедентів онлайн-сервісу

Використання діаграм прецедентів також дозволяє оцінити потенційні ризики та визначити ділянки, що можуть потребувати додаткового опрацювання. Це важливий крок до забезпечення надійності та функційності системи загалом, що уможливлює максимально корисну взаємодію користувачів з великими мовними моделями.

3.2 Опис функціональних процесів сервісу у нотації IDEF0

Однією з ключових складових під час розробки онлайн-сервісу для агрегування великих мовних моделей є моделювання та проєктування його функціональних процесів. Для виконання цього завдання було обрано методологію моделювання IDEF0, яка дозволяє чітко визначити взаємозв’язки між різними елементами процесу, включаючи вхідні та вихідні дані, механізми, а також аспекти управління [41].

Методологія IDEF0 надає можливість створити інтегровану модель, яка детально описує, як різні функції системи виконують свої завдання. Це є важливим для того, щоб глибше зрозуміти всі процеси, які відбуваються в рамках функціонування сервісу. Розроблення такої моделі сприяє подальшій реалізації системи, дозволяючи досягти функціональності та ефективності.

У процесі створення сервісу для агрегування великих мовних моделей було визначено кілька основних функціональних процесів, які забезпечують його роботу. Серед них можна виділити ініціацію запиту до мовної моделі, обробку цього запиту, взаємодію з користувачем, а також виведення кінцевого результату.

Функціональний процес ініціації запиту до мовної моделі починається з того, що користувач надсилає запит через інтерфейс сервісу, наприклад, через веб-інтерфейс, API або бота в Telegram. Цей етап передбачає наявність доступу до обраної мовної моделі та відповідних налаштувань, які задав користувач. Як результат, система отримує запит, який буде передано на наступний етап обробки (рис. 3.2).

Рисунок 3.2 – IDEF0-діаграма функціоналу «Створення нового запиту»

Наступним процесом є взаємодія з користувачем, яка активується генерацією відповіді мовною моделлю. На цьому етапі враховуються політика конфіденційності та правила взаємодії з кінцевим користувачем. Взаємодія реалізується через компоненти зворотного зв’язку, такі як чат або повідомлення. Завдяки цьому користувач отримує відповідь і може продовжити роботу із сервісом. Обробка запиту є важливим етапом, який розпочинається після отримання системою запиту. Тут відбувається вибір відповідної мовної моделі відповідно до характеристик і параметрів,

зазначених у запиті користувача. Для цього використовується серверна частина системи, де розгорнуті API-обробники LLM та AI-сервісів. Завершенням цього етапу є створення відповіді на запит за допомогою обраної мовної моделі.

Останнім етапом є виведення результату, який слідує за завершенням обробки запиту. Тут важливу роль відіграє формат і структура відповіді, які повинні відповідати потребам користувача. Результат може бути представлений у вигляді текстового повідомлення, таблиці, графіків чи інших візуалізаційних форматів. Завершення цього етапу полягає в тому, що користувач отримує кінцевий результат у зручній для нього формі (рис. 3.3).

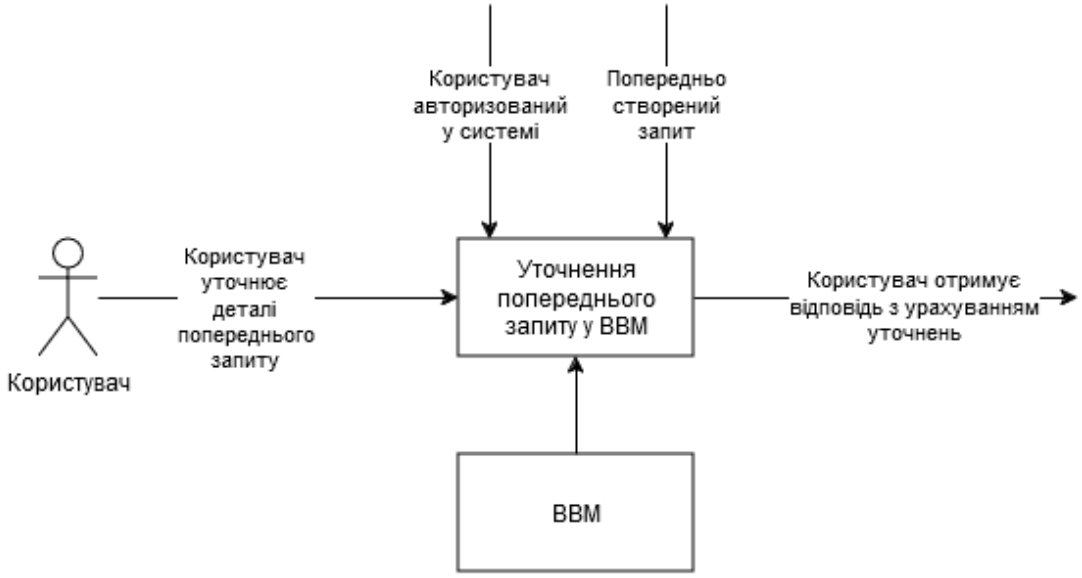


Рисунок 3.3 – IDEF0-діаграма функціоналу «Уточнення запиту»

Застосування методології IDEF0 дозволило розробити чітку та деталізовану модель функціональних процесів сервісу для агрегування великих мовних моделей. Це сприяє кращому розумінню роботи системи, вчасному виявленню потенційних недоліків, а також оптимізації процесів, що є критично важливим для успішної реалізації проєкту. Завдяки використанню IDEF0 - вдалося підвищити якість розробки, забезпечити ефективне функціонування сервісу та створити зручний інструмент для кінцевих користувачів.

3.3 Діаграма розгортання

У процесі розробки онлайн-сервісу для агрегування великих мовних моделей постала необхідність створення архітектури, яка забезпечувала б високу продуктивність, масштабованість та гнучкість системи. Діаграма розгортання є ключовим елементом моделювання, оскільки вона відображає фізичні аспекти системи, показуючи, як компоненти програмного забезпечення розгорнуті на апаратних ресурсах [42].

Архітектура нашої системи передбачає впровадження декількох взаємопов'язаних вузлів, кожен з яких виконує специфічні завдання та використовує різноманітні технології для оптимізації роботи сервісу.

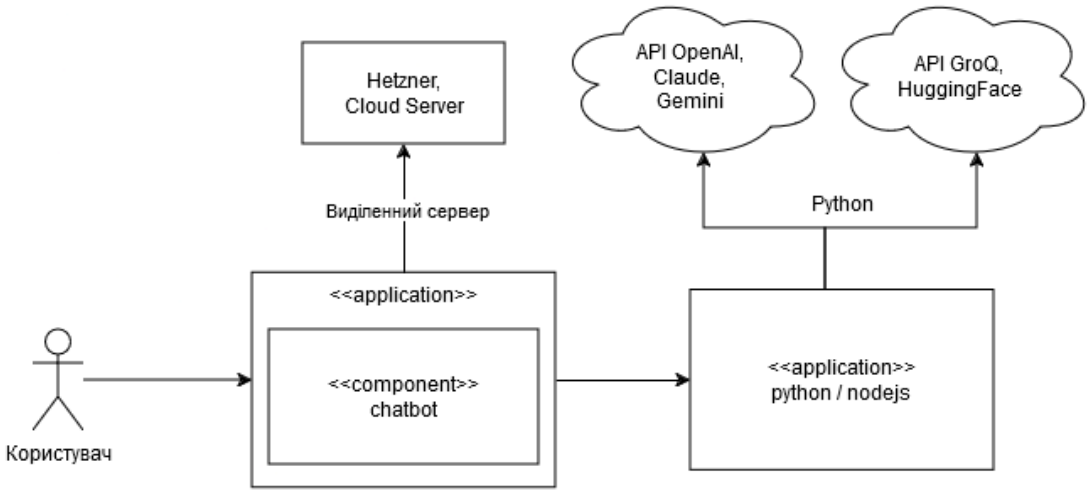


Рисунок 3.4 – Діаграма розгортання інформаційної системи

Клієнтський вузол відповідає за взаємодію з кінцевим користувачем через веб-браузер. На цьому рівні використовуються HTML і CSS для структурування та стилізації контенту, а JavaScript забезпечує динаміку та інтерактивність. Зокрема, JavaScript відповідає за обробку подій, валідацію введених даних та асинхронний обмін інформацією з сервером через AJAX-запити.

Серверна частина побудована на Node.js, що дає змогу ефективно обробляти численні одночасні запити завдяки неблокуючій архітектурі. Python інтегровано для виконання завдань, пов'язаних з обробкою запитів до API мовних моделей, керуванням сесіями та обробкою даних.

Для зберігання даних про сесії, користувачів та історію взаємодій використовується MongoDB, яка пропонує гнучкість та легкість масштабування завдяки документно-орієнтованій структурі. Redis застосовується для кешування, що дозволяє суттєво знизити навантаження на сервер і підвищити швидкість опрацювання запитів.

Окремий вузол відповідає за інтеграцію з різними API мовних моделей, зокрема API OpenAI, Claude, Gemini, а також HuggingFace (Grok) та GroQ (Mistral, LLaMA). Це забезпечує можливість вибору відповідної моделі для конкретного завдання: генерація тексту, зображень, веб-пошуку, тощо.

Усі компоненти сервісу розгорнуті в контейнерах за допомогою Docker, що гарантує стабільність та узгодженість середовища, а також полегшує процеси тестування і розгортання. Контейнеризація дає змогу легко масштабувати систему, додаючи нові вузли при збільшенні навантаження.

Для розроблення та налаштування середовища використовується Visual Studio Code, який завдяки своїм розширенням підтримує всі необхідні мови програмування та інтеграцію з системами контролю версій. Це дозволяє розробнику ефективно співпрацювати та швидко реагувати на зміни вимог.

Завдяки ретельному підбору технологій та інструментів, система відповідає сучасним вимогам до онлайн-сервісів, забезпечуючи надійну взаємодію з великими мовними моделями через зручний та інтуїтивно зрозумілий інтерфейс. Діаграма розгортання відображає всі ці аспекти, демонструючи, як компоненти системи взаємодіють між собою на фізичному рівні, що дає змогу досягти поставлених цілей.

3.4 Діаграма послідовності

У процесі проєктування онлайн-сервісу для агрегування великих мовних моделей ключовим завданням було створення архітектури, що забезпечує високу продуктивність, масштабованість та гнучкість системи. Діаграма послідовності є важливим інструментом моделювання, оскільки дозволяє візуалізувати та детально описати порядок взаємодії між компонентами системи та їхні функціональні кроки [43].

Діаграми послідовності, як різновид діаграм взаємодії, зосереджуються на часовому аспекті виконання операцій і відображають порядок взаємодії через вертикальну вісь діаграми. Вони надають можливість візуально представити взаємодію об'єктів у контексті співпраці, що є критично важливим для розуміння складних процесів у системі.

У розробленій системі, діаграма послідовності відображає ключовий функціонал сервісу: від моменту ініціації запиту користувачем до отримання відповіді від обраної мовної моделі. Розглянемо детально ключові елементи взаємодії в цьому процесі (рис. 3.5).

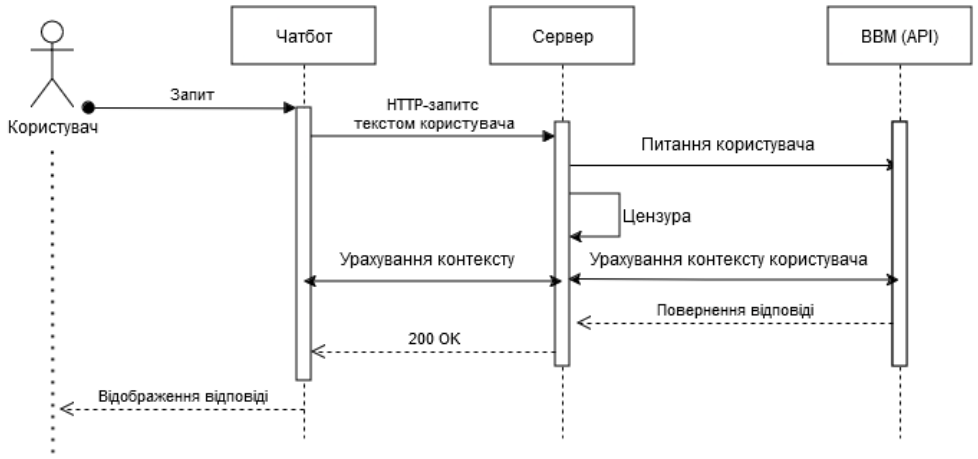


Рисунок 3.5 – Діаграма послідовност інформаційної системи

Першим елементом є користувач, який виступає ініціатором процесу та взаємодіє з системою через веб-інтерфейс. Користувач формулює запит, що буде опрацьоване сервісом, а далі – мовною моделю або штучним інтелектом.

Клієнтський вузол (Frontend) є компонентом, який забезпечує взаємодію з користувачем через веб-браузер. Його основне призначення – надсилання запиту до серверної частини. Для реалізації цього використовуються технології HTML, CSS та JavaScript, що забезпечують інтерактивність та динаміку інтерфейсу онлайн-сервісу.

Серверний вузол (Backend) відповідає за обробку запитів, отриманих від клієнта. Використовуючи NodeJS та інтегрований Python, сервер опрацьовує запити до API мовних моделей та штучного інтелекту, керує сеансами та виконує обробку даних та запитів користувача.

Процес взаємодії розпочинається з введення користувачем запитання в інтерфейсі клієнтського вузла. Далі запитання передається до серверного вузла, де відбувається його опрацювання та перевірка. Потім, сервер звертається до відповідного API мовної моделі або штучного інтелекту, через інтеграційний вузол для отримання відповіді.

Отриману відповідь буде опрацьовано та надіслано назад до клієнтського вузла для відображення користувачеві.

Діаграма послідовності чітко демонструє всі ці кроки, відображаючи взаємодію між користувачем та системою. Це дає змогу розробникам та іншим зацікавленим сторонам зрозуміти процеси, оптимізувати їх та забезпечити безперебійну роботу сервісу. Завдяки такій детальній візуалізації, система може ефективно виконувати поставлені завдання, забезпечуючи якісний та інтуїтивно зрозумілий досвід для кінцевих користувачів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Налаштування великих мовних моделей

Інтеграція великих мовних моделей (LLM) постає ключовою складовою розробки онлайн-сервісу для їх подальшої агрегації. Процес інтеграції, наприклад, GPT-4o передбачає послідовне виконання декількох кроків.

Спочатку необхідно зареєструватися на платформі OpenAI та створити новий проект, який слугуватиме фундаментом для подальшої інтеграції. У межах створеного проекту потрібно згенерувати унікальний API-ключ, що використовуватиметься для авторизації та взаємодії з GPT-4o. Прикладом такого ключа може бути назва «Testing» (рис. 4.1).

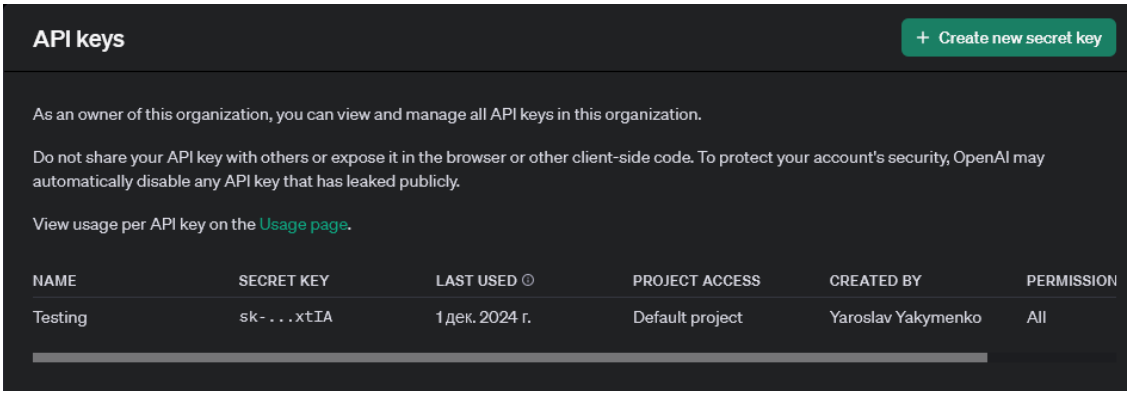


Рисунок 4.1 – Створення API-ключа OpenAI

Для здійснення запитів до GPT-4o, або інших мовних моделей, обов'язковим є поповнення рахунку, причому, для нашого сервісу, мінімальна сумма в 10 доларів США буде достатньою для подальшої роботи (рис. 4.2). Додатково рекомендується налаштувати сповіщення про зменшення балансу до критичного рівня, хоча для нашого проекту - це не є принциповим.

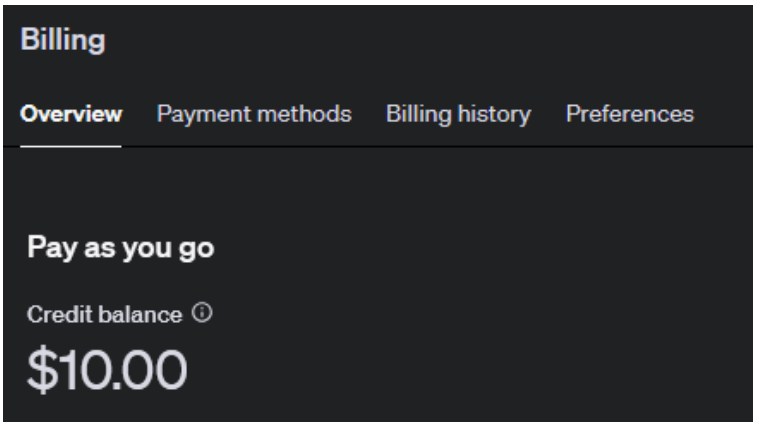


Рисунок 4.2 – Поповнення балансу в кабінеті розробника

Для використання Claude 3.5 Sonnet (рис. 4.3) та Gemini 1.5 Pro (рис. 4.4), також необхідно зареєструватись в кабінеті розробника та поповнити рахунок. Моделі пропонують розширені можливості в опрацюванні природної мови, генеруванні тексту та зображень, тому їх наявність в інформаційній технології, як альтернатива GPT-4o, є обов'язковим для кінцевого користувача.

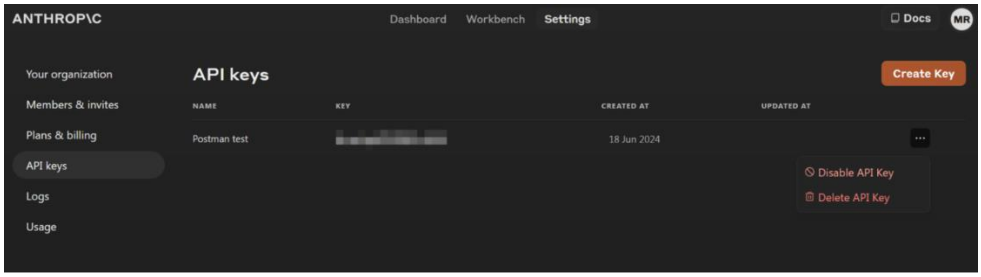


Рисунок 4.3 – Створення API-ключа Claude

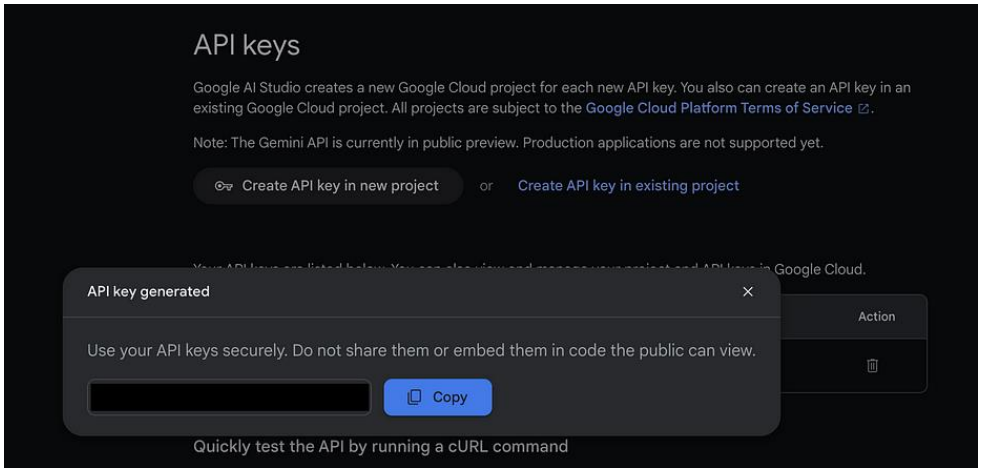


Рисунок 4.4 – Створення API-ключа Gemini

Інтеграція з HuggingFace (рис. 4.5) потребує створення облікового запису для керування доступу до багатьох мовних моделей. Важливим кроком є генерування API-ключа для взаємодії з обраними моделями, такими як Grok-1, з подальшим впровадженням їхнього функціоналу в власний сервіс.

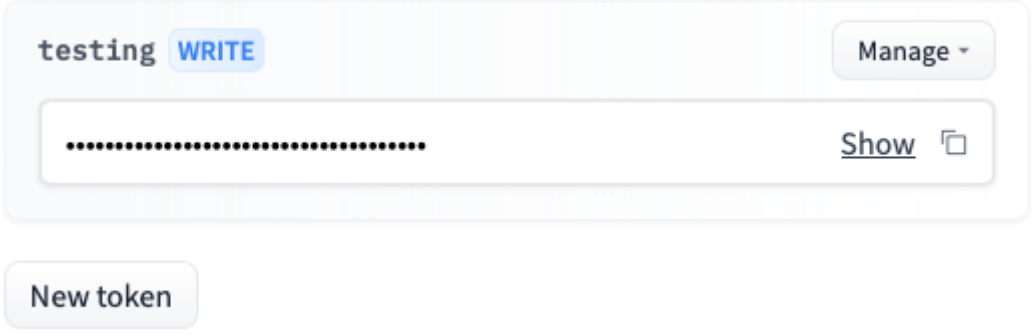


Рисунок 4.5 – Створення API-ключа HuggingFace

API GroQ забезпечує доступ до моделей Mistral 8x7b та LLaMA 3.1, які є ефективними для текстових завдань, як-от аналіз, написання або класифікація тексту. Процес інтеграції передбачає реєстрацію на платформі GroQ та генерування API-ключа доступу для роботи моделями.

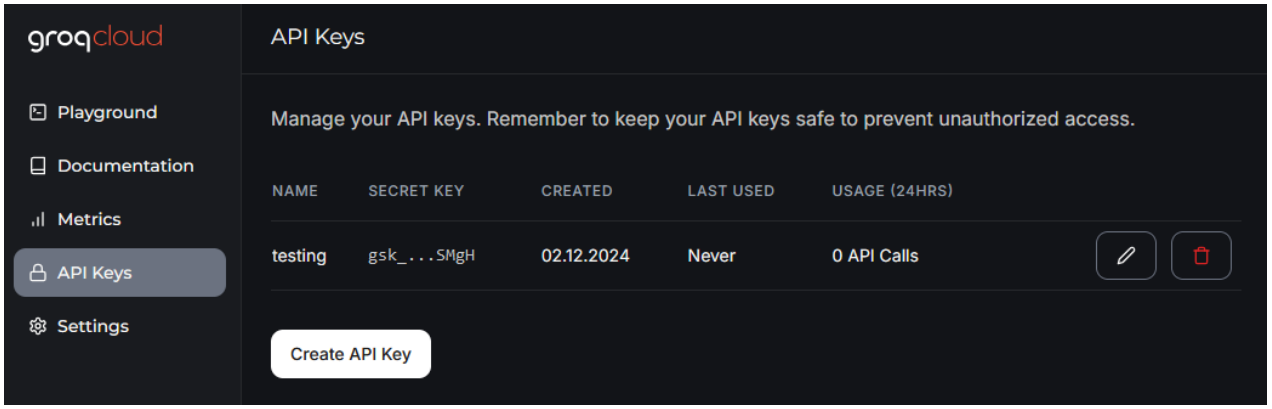


Рисунок 4.6 – Створення API-ключа GroQ

Підсумовуючи, ми розглянули процес налаштування й інтеграції різних великих мовних моделей та штучного інтелекту в інформаційну технологію та онлайн-сервіс їх агрегації. Завдяки ретельному підбору серед доступних інструментів, ми забезпечили високу функціональність і ефективність нашого сервісу для кінцевих побутових та корпоративних користувачів.

4.2 Реалізація серверної частини додатку

Під час розробки серверної частини онлайн-сервісу для агрегування великих мовних моделей, ми обрали середовище виконання NodeJS, мову програмування JavaScript, Python та низку допоміжних інструментів і технологій, зокрема Docker, MongoDB та Redis.

Ми зупинили свій вибір на Visual Studio Code як основному редакторі через його легкість, потужність та широкий набір розширень, які дають змогу налаштувати робоче середовище під специфічні вимоги проекту (рис. 4.7). Особливо корисними виявилися розширення для роботи з NodeJS та MongoDB, котрі значно полегшують процес розробки.

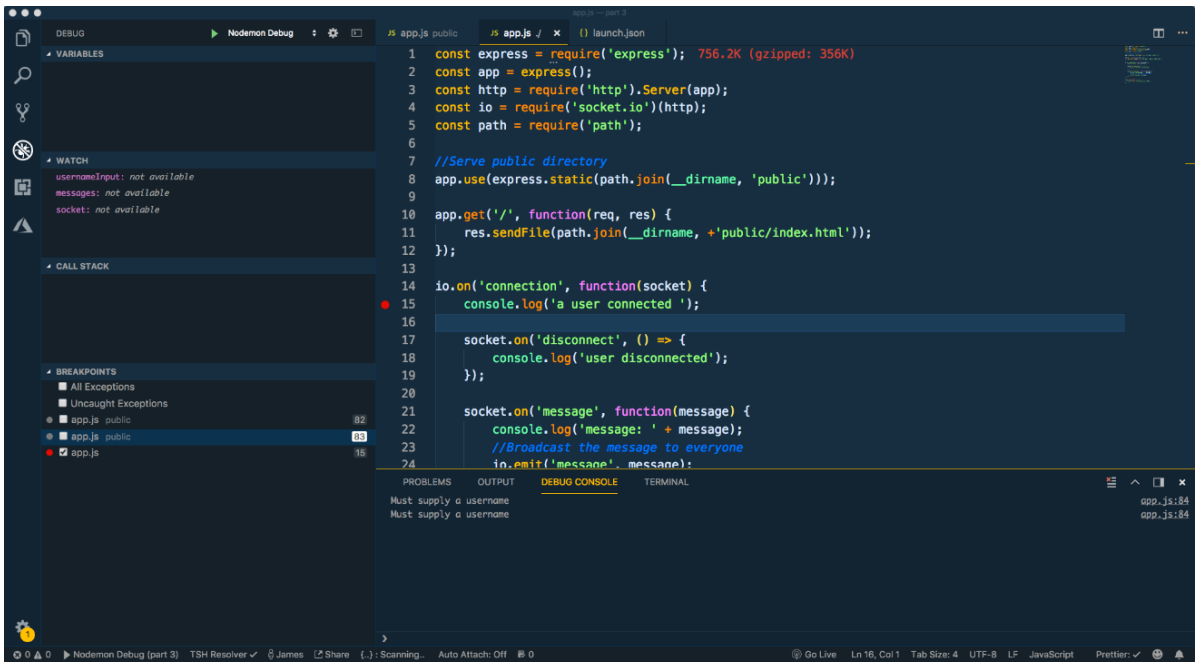


Рисунок 4.7 – Середовище Visual Studio Code

Для забезпечення консистентності середовища розробки та розгортання - ми застосували Docker (рис. 4.8). Завдяки контейнеризації ми маємо впевність, що наш додаток функціонуватиме однаково на різних серверах, які підтримують Docker. Це суттєво зменшує ризики, пов'язані з конфігураційними проблемами, та спрощує процеси тестування й розгортання онлайн-сервісу агрегації великих мовних моделей.

```yaml
version: '3.8'

services:
  app:
    image: node:16
    container_name: llm_aggregator_app
    working_dir: /app
    volumes:
      - .:/app
      - /app/node_modules
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=development
      - MONGO_URI=mongodb://mongo:27017/llm_aggregator
      - REDIS_HOST=redis
      - REDIS_PORT=6379
    depends_on:
      - mongo
      - redis
    command: sh -c "npm install && npm run dev"

  mongo:
    image: mongo:5
    container_name: llm_aggregator_mongo
    volumes:
      - mongo-data:/data/db
    ports:
      - "27017:27017"

  redis:
    image: redis:alpine
    container_name: llm_aggregator_redis
    ports:
      - "6379:6379"

volumes:
  mongo-data:
```

Рисунок 4.8 – Налаштування контейнеризації Docker

Основним завданням серверної частини є обробка клієнтських запитів, управління сеансами користувачів та взаємодія з базою даних. Для реалізації цих завдань ми обрали NodeJS (рис. 4.9), який спроможний обробляти значну кількість запитів одночасно. Така властивість є критично важливою для нашого онлайн-сервісу, що працює в режимі реального часу та обслуговує різноманітні запити користувачів.

```
// Імпорт необхідних модулів
const express = require('express');
const mongoose = require('mongoose');
const Redis = require('ioredis');
const axios = require('axios'); // Для інтеграції з API мовних моделей

// Ініціалізація додатку Express
const app = express();

// Підключення до MongoDB
mongoose.connect('mongodb://localhost:27017/llm-aggregator', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('Підключено до MongoDB'))
  .catch(err => console.error('Помилка підключення до MongoDB:', err));

// Ініціалізація Redis для кешування
const redis = new Redis();

// Конфігурація середовища
const PORT = process.env.PORT || 3000;

// Маршрутизація запитів
app.get('/api/models', async (req, res) => {
    try {
        // Перевірка кешу Redis
        const cachedResponse = await redis.get('models');
        if (cachedResponse) {
            return res.json(JSON.parse(cachedResponse));
        }

        // Запит до зовнішнього API мовних моделей
        const response = await axios.get('https://api.yakymenko.com/models');
        const models = response.data;

        // Збереження в кеші
        await redis.set('models', JSON.stringify(models), 'EX', 3600); // Зберігання на 1 годину

        res.json(models);
    } catch (error) {
        console.error('Помилка при отриманні мовних моделей:', error);
        res.status(500).send('Виникла помилка на сервері');
    }
});

// Запуск сервера
app.listen(PORT, () => {
    console.log(`Сервер запущено на порту ${PORT}`);
});
```

Рисунок 4.9 – Склад файлу ініціалізації index.js (NodeJS)

Зберігання даних про сеанси, користувачів та історію їх взаємодії, реалізовано за допомогою MongoDB (рис. 4.10) — документно-орієнтованої бази даних NoSQL. Її гнучкість та здатність до масштабування дають змогу легко адаптуватися під специфічні потреби додатку та забезпечити надійне зберігання великих обсягів інформації.

```yaml
version: '3.8'
services:
  mongodb:
    image: mongo:5.0
    container_name: mongodb
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: adminpassword
      MONGO_INITDB_DATABASE: mydatabase
    volumes:
      - mongodb_data:/data/db
    networks:
      - app_network

volumes:
  mongodb_data:
    driver: local

networks:
  app_network:
    driver: bridge
```

Рисунок 4.10 – Склад файлу конфігурації MongoDB

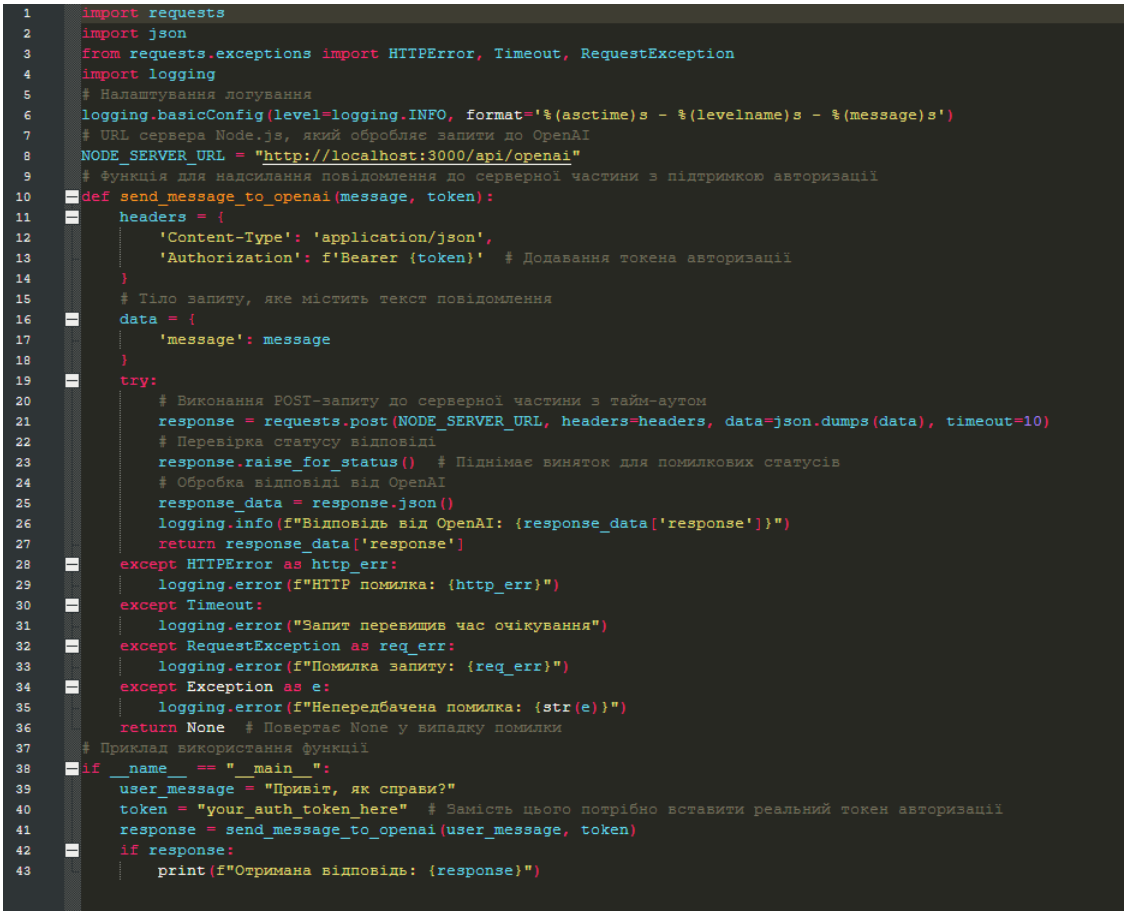
Для підвищення продуктивності та зменшення затримок ми використали Redis — систему зберігання даних у пам'яті у форматі ключ-значення (рис. 4.11). Цей інструмент дає можливість ефективно кешувати результати запитів, що значно знижує навантаження на сервер та скорочує час відповіді.

```
# Network settings
bind 127.0.0.1
port 6379

# General settings
daemonize no
protected-mode yes
tcp-backlog 511

# Memory management
maxmemory 256mb
maxmemory-policy allkeys-lru

# Snapshots
save 900 1
save 300 10
save 60 10000

# Logging
loglevel notice
logfile ""

# Security
requirepass "password"

# Append-only file settings
appendonly yes
appendfilename "appendonly.aof"

# Replication settings
# slaveof <masterip> <masterport>
```

Рисунок 4.11 – Склад файлу конфігурації Redis

Ключовою складовою серверної реалізації стала інтеграція з API великих мовних моделей та штучного інтелекту, зокрема - OpenAI, Claude, Gemini, HuggingFace та GroQ (рис. 4.12).

Ці API забезпечують доступ до потужних моделей обробки природної мови, що дає нашому сервісу змогу створювати інтуїтивно зрозумілий інтерфейс взаємодії з користувачем, генерувати текстовий та графічний контент, а також - виконувати складніші завдання, як-от аналіз емоційного забарвлення або класифікація текстів.



```python
import requests
import json
from requests.exceptions import HTTPError, Timeout, RequestException
import logging
# Налаштування логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
# URL сервера Node.js, який обробляє запити до OpenAI
NODE_SERVER_URL = "http://localhost:3000/api/openai"
# Функція для надсилання повідомлення до серверної частини з підтримкою авторизації
def send_message_to_openai(message, token):
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {token}'  # Додавання токена авторизації
    }
    # Тіло запиту, яке містить текст повідомлення
    data = {
        'message': message
    }
    try:
        # Виконання POST-запиту до серверної частини з тайм-аутом
        response = requests.post(NODE_SERVER_URL, headers=headers, data=json.dumps(data), timeout=10)
        # Перевірка статусу відповіді
        response.raise_for_status()  # Піднімає виняток для помилкових статусів
        # Обробка відповіді від OpenAI
        response_data = response.json()
        logging.info(f"Відповідь від OpenAI: {response_data['response']}")
        return response_data['response']
    except HTTPError as http_err:
        logging.error(f"HTTP помилка: {http_err}")
    except Timeout:
        logging.error("Запит перевищив час очікування")
    except RequestException as req_err:
        logging.error(f"Помилка запиту: {req_err}")
    except Exception as e:
        logging.error(f"Непередбачена помилка: {str(e)}")
    return None  # Повертає None у випадку помилки
# Приклад використання функції
if __name__ == "__main__":
    user_message = "Привіт, як справи?"
    token = "your_auth_token_here"  # Замість цього потрібно вставити реальний токен авторизації
    response = send_message_to_openai(user_message, token)
    if response:
        print(f"Отримана відповідь: {response}")
```

Рисунок 4.12 – Склад файлу обробки API-запитів на Python

Середовище Python надає широкий набір модулів для інтеграції з цими API, що суттєво спрощує процес їх використання в нашому додатку.

Завершальним етапом став процес тестування та оптимізації серверної частини. Для цього використано різноманітні інструменти моніторингу продуктивності, такі як Prometheus (рис. 4.13) та Grafana (рис. 4.14), які допомогли виявити потенційні вузькі місця в додатку та оптимізувати його для забезпечення максимальної швидкодії та стабільності сервісу.

```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'nodejs_app'
    static_configs:
      - targets: ['localhost:3000']  # Node.js додатку

  - job_name: 'mongodb'
    static_configs:
      - targets: ['localhost:27017']  # Порт MongoDB

  - job_name: 'redis'
    static_configs:
      - targets: ['localhost:6379']  # Порт Redis
```

Рисунок 4.13 – Конфігурація Prometheus

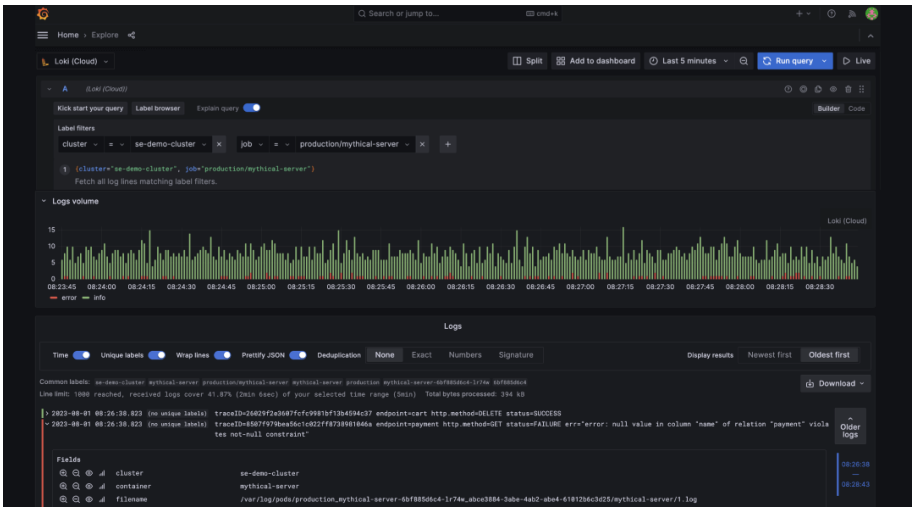


Рисунок 4.14 – Інструмент моніторінгу Grafana

Таким чином, завдяки використанню сучасних технологій, було створено серверну частину онлайн-сервісу для агрегування великих мовних моделей, котра є надійною, масштабованою та спроможною обробляти значне навантаження в режимі реального часу (рис. 4.15).

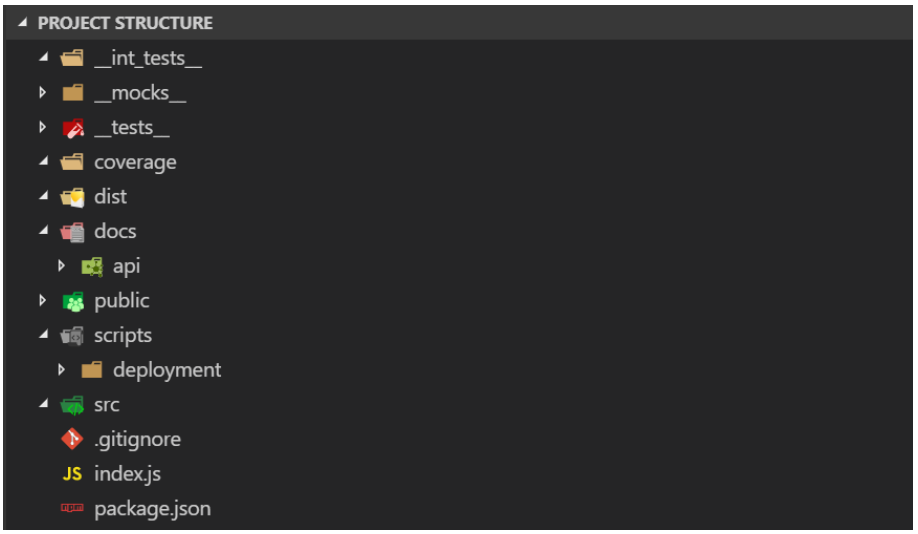


Рисунок 4.15 – Структура коду проекту онлайн-сервісу

4.3 Розробка клієнтської частини додатку

Для реалізації клієнтської частини онлайн-сервісу з агрегації великих мовних моделей ми використали сучасні веб-технології, що забезпечують створення зручного та візуально привабливого інтерфейсу.

Основними технологіями на рівні інтерфейсу користувача стали HTML, CSS та JavaScript (рис. 4.16). Їхня синергія дає змогу не лише створити зрозумілий користувачеві інтерфейс, а й забезпечити його динамічність та інтерактивність.

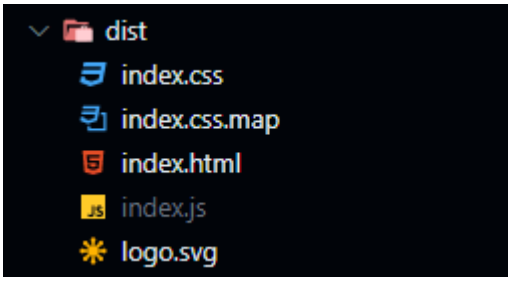


Рисунок 4.16 – Структура дизайну проекту

Розробка розпочалась з реалізації HTML-шаблону, який визначає базову структуру інтерфейсу. HTML-файл містить основні елементи: поля введення, кнопки, області для виведення результатів та історії повідомлень.

Стилізація HTML-шаблону відбувалася за допомогою CSS, що надав інтерфейсу естетичного вигляду. Визначені стилі окреслюють кольорову гаму, шрифти, відступи та інші візуальні параметри, забезпечуючи узгоджений та привабливий дизайн. Важливою особливістю є ізольоване застосування CSS-стилів до кожного компонента, що унеможливлює виникнення конфліктів між різними частинами інтерфейсу.

JavaScript постав основним інструментом реалізації бізнес-логіки на клієнтській стороні. Його використання дозволило динамічно оновлювати дані на сторінці, здійснювати обробку подій, забезпечувати взаємодію з користувачем та виконувати запити до серверної частини без потреби перезавантаження сторінки. Технологія AJAX, яка інтегрована в додаток, уможливила асинхронну взаємодію з сервером, що значно покращило користувацький досвід.

Першочерговим кроком у розробці сервісу, став процес створення класу, що реалізує основну логіку роботи з користувацькими запитами (рис. 4.17). Цей клас містить властивості для зберігання поточного стану чат-боту: видимість вікна чату, історія повідомлень та відповіді від серверної частини. Методи класу забезпечують обробку подій, зокрема відправлення запитів на сервер або отримання результатів від мовної моделі або штучного інтелекту.

```
// Декоратор @Component визначає метадані компонента
@Component({
  selector: 'app-chat',
  templateUrl: './chat-bot.component.html',
  styleUrls: ['./chat-bot.component.css']
})
export class ChatComponent implements OnInit {
  // Властивості компонента
  chatVisible: boolean = false;  // Відображення чату
  userMessage: string = '';      // Повідомлення користувача
  messages: { text: string, isBot: boolean }[] = []; // Масив повідомлень
  response: string = '';         // Відповідь від бота
```

Рисунок 4.17 – Опис полів класу для компоненту чат-бота

Метод ініціалізації викликається під час завантаження компонента з метою відновлення попередньої історії взаємодії користувача з сервісом. Це принципово важливо, оскільки дає змогу продовжити спілкування з чат-ботом з будь-якого місця платформи без втрати контексту (рис. 4.18).

```
// Метод для відновлення історії чату
loadChatHistory() {
  const storedMessages = localStorage.getItem('chatHistory');
  if (storedMessages) {
    this.messages = JSON.parse(storedMessages);
  }
}

// Метод для збереження історії чату
saveChatHistory() {
  localStorage.setItem('chatHistory', JSON.stringify(this.messages));
}

// Метод для відображення/приховування чату
toggleChat(visible: boolean) {
  this.chatVisible = visible;
}
```

Рисунок 4.18 – Опис функції відображення історії чатів

Наступним етапом стала реалізація методів обробки подій користувача, таких як - реакція на натискання кнопок або введення тексту (рис. 4.19). Розроблені методи забезпечують динамічне оновлення інтерфейсу та взаємодію з серверною частиною для отримання відповідей від API.

```
onSendMessage(): void {
  if (this.userMessage.trim()) {
    this.messages.push({ text: this.userMessage, isBot: false });
    this.fetchResponse(this.userMessage);
    this.userMessage = '';
  }
}

fetchResponse(userText: string): void {
  // Симуляція запиту до серверної частини
  setTimeout(() => {
    const botResponse = `Відповідь на: ${userText}`;
    this.messages.push({ text: botResponse, isBot: true });
  }, 1000);
}
}
```

Рисунок 4.19 – Опис реалізації обробки подій користувача

Завершальним кроком став механізм надсилання та отримання даних. Ключові методи відповідають за формування запитів до серверної частини та обробку відповідей (рис. 4.20). Це охоплює формування запитів у форматі JSON, опрацювання асинхронних відповідей та оновлення інтерфейсу на підставі отриманих даних від мовних моделей та штучного інтелекту.

```
async function sendMessage(userMessage) {
    try {
        const response = await fetch('https://api.yakymenko.com/chat', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ message: userMessage })
        });

        if (!response.ok) {
            throw new Error('Помилка мережі');
        }

        const data = await response.json();
        updateUI(data);
    } catch (error) {
        console.error('Сталася помилка:', error);
    }
}

function updateUI(data) {
    // Оновлення інтерфейсу на підставі отриманих даних
    const chatComponent = document.querySelector('app-chat');
    chatComponent.messages.push({ text: data.response, isBot: true });
    chatComponent.response = data.response;
}
```

Рисунок 4.20 – Опис реалізації механізму надсилання і отримання даних

Підсумовуючи, завдяки використанню сучасних веб-технологій та ретельно продуманої архітектури додатку, було успішно реалізовано клієнтську частину онлайн-сервісу. Розроблений інтерфейс дозволяє користувачам взаємодіяти з великими мовними моделями, забезпечуючи інтуїтивність, гнучкість та відповідність вимогам сучасних веб-додатків.

4.4 Тестування інструментів онлайн-сервісу

Тестування є критично важливим етапом у розробці програмного забезпечення, оскільки воно дозволяє виявити та усунути потенційні недоліки та забезпечити надійну роботу системи.

Першим етапом взаємодії користувача з онлайн-сервісом є процес створення облікового запису через сторінку «Реєстрація». Для проходження цього етапу користувач зобов'язаний заповнити форму, яка складається з наступних полів: повне ім'я, ім'я користувача, електронна пошта та пароль. Тестування на цьому етапі включає перевірку правильності валідації кожного поля та успішність створення облікового запису у системі (рис. 4.21).

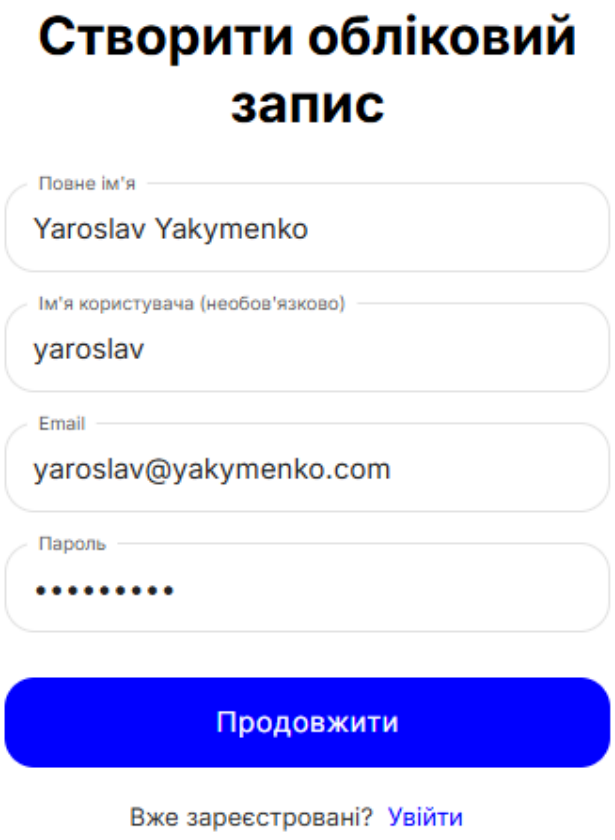


Рисунок 4.21 – Сторінка «Реєстрація»

Після створення облікового запису, користувач може увійти в систему за допомогою сторінки «Авторизація». Для цього йому потрібно ввести свою електронну пошту і пароль. На цій сторінці, також, надається посилання для переходу на сторінку «Реєстрація» (рис. 4.22). Тестування передбачає перевірку коректності введення даних, успішність входу в систему, а також - перевірку функціонування системи реєстрації та авторизації.

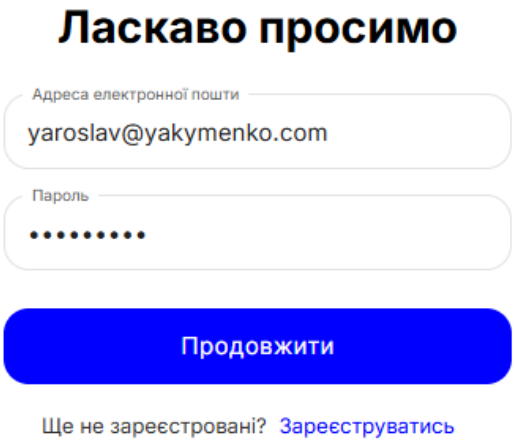


Рисунок 4.22 – Сторінка «Авторизація»

Після успішної реєстрації або авторизації, користувач потрапляє на «головну» сторінку сервісу (рис. 4.23). На сторінці, він має доступ до поля для вводу запитів, історії чатів, налаштувань, можливості обрати мовну модель або штучний інтелект для роботи. Тестування цієї сторінки включає перевірку функціональності кожного елемента інтерфейсу, таких як - введення тексту, отримання відповіді, відображення історії чатів та правильність налаштувань.

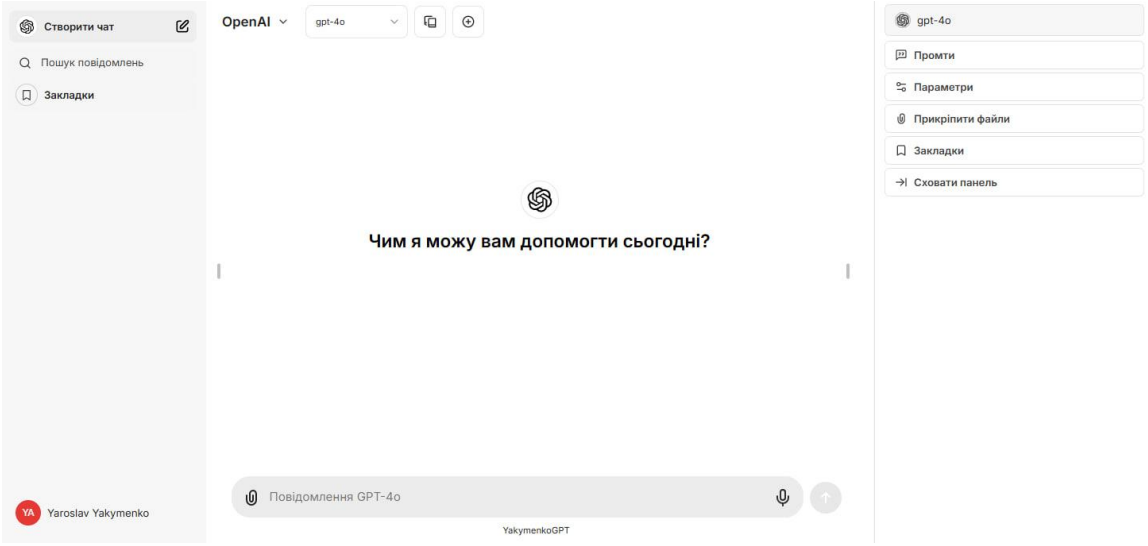


Рисунок 4.23 – Сторінка «Головна»

У розділі «Налаштувань», який розташовано справа від інтерфейсу чату, користувач може створювати власні промти (рис. 4.24), змінювати налаштування мовної моделі (рис. 4.25), прикріпляти файли для контексту (рис. 4.26), створювати закладки (рис. 4.27), а також - створювати нові "пресети" (рис. 4.28). Тестування передбачає перевірку кожної з цих функцій на предмет коректного виконання та відповідність очікуваним результатам.

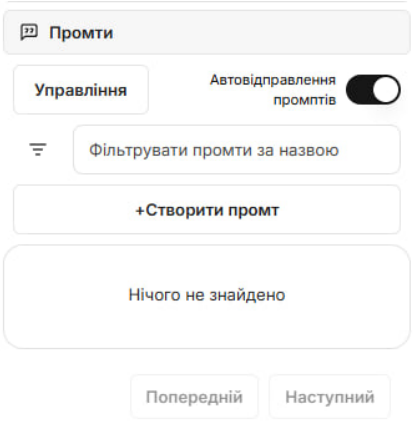


Рисунок 4.24 – Підрозділ «Налаштувань», вкладка «Промти»

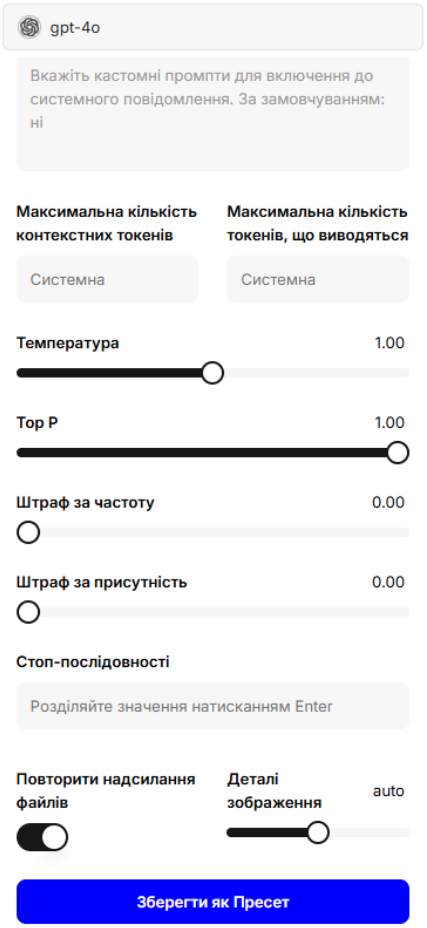


Рисунок 4.25 – Підрозділ «Налаштувань», вкладка «Налаштування моделі»

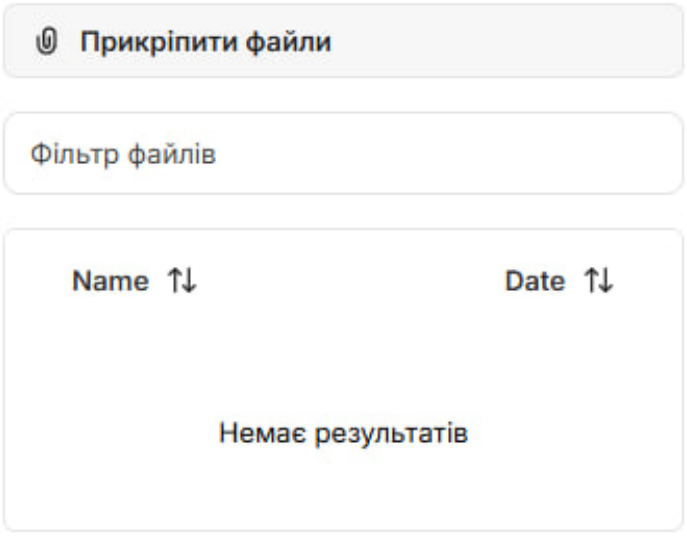


Рисунок 4.26 – Підрозділ «Налаштувань», вкладка «Прикріплення файлів»

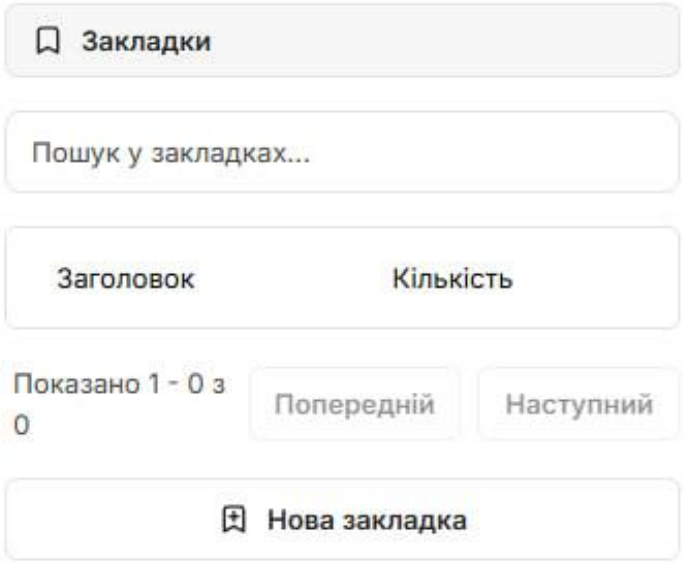


Рисунок 4.27 – Підрозділ «Налаштувань», вкладка «Закладки»

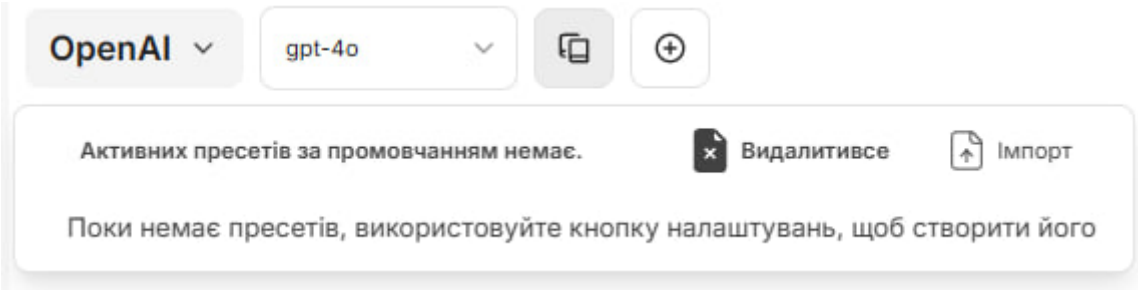


Рисунок 4.28 – Підрозділ «Налаштувань», вкладка «Пресети»

Користувач має можливість обрати мовну модель або штучний інтелект, з яким збирається працювати. Тестування цього розділу налаштувань включає в себе перевірку коректності відображення доступних моделей, їх вибору та зміни вендора серед доступних в списку (рис. 4.29).

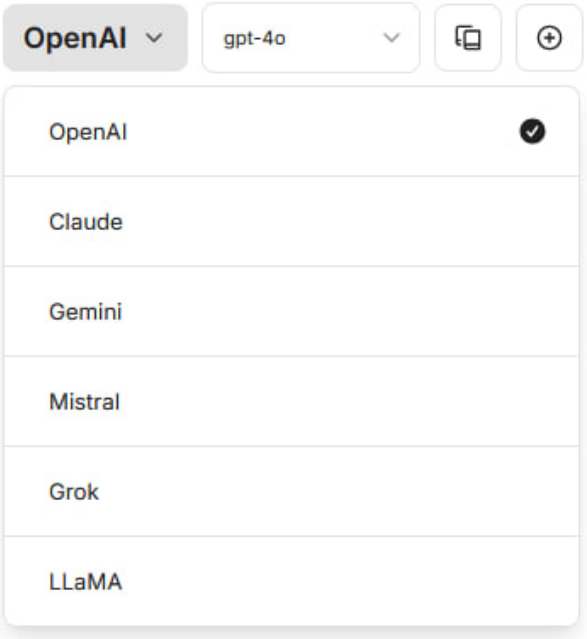


Рисунок 4.29 – Підрозділ «Налаштувань», вкладка «Обрання моделі»

Тестування генерації тексту, передбачає перевірку здатності системи правильно обробляти запити, що надходять від користувачів, та надсилати їх вендорам через API для створення і повернення відповідей (рис. 4.30).

Основна увага приділяється точності та релевантності згенерованого тексту відносно початкового запиту. Процес включає перевірку різних типів запитів — від простих інформаційних до складних багатокомпонентних запитів, — щоб переконатися у здатності моделі відповідати на широке коло питань з високим ступенем корисності.

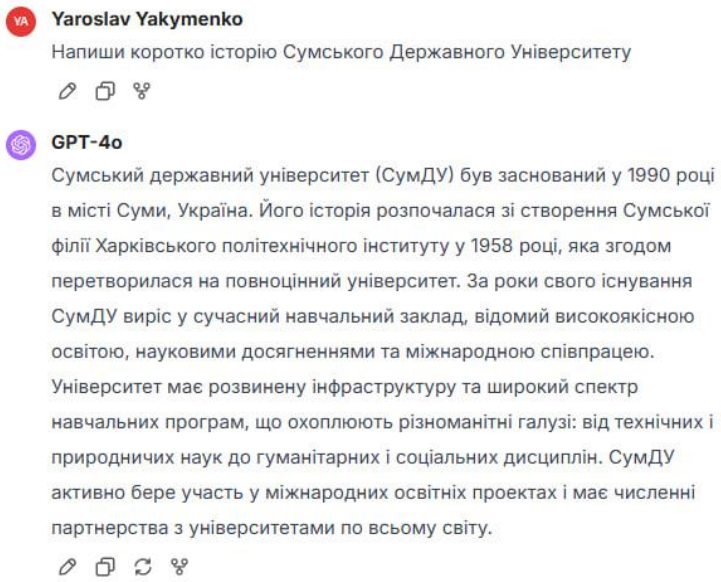


Рисунок 4.30 – Тестування генерації текстової відповіді

Наступний пункт тестування, зосереджений на можливості системи генерувати зображення на основі текстових описів, наданих користувачами (рис. 4.31). Тестування включає оцінку якості, точності та творчої відповідності зображення до початкового текстового запиту. Перевіряється широкий спектр запитів, які можуть включати конкретні деталі, такі як стиль, палітра кольорів або описані об'єкти, щоб оцінити адаптивність системи до різноманіття завдань, які надає їй користувач.

Рисунок 4.31 – Тестування генерації зображень

Тестування функції веб-пошуку, передбачає перевірку здатності сервісу здійснювати пошук в Інтернеті на основі текстових запитів (рис. 4.32). У ході тестування оцінюється, як сервіс обробляє і фільтрує інформацію, щоб надати користувачеві найбільш відповідні та актуальні дані. Також перевіряється, як система справляється з неоднозначними запитами та її здатність надавати корисні підказки або уточнення.

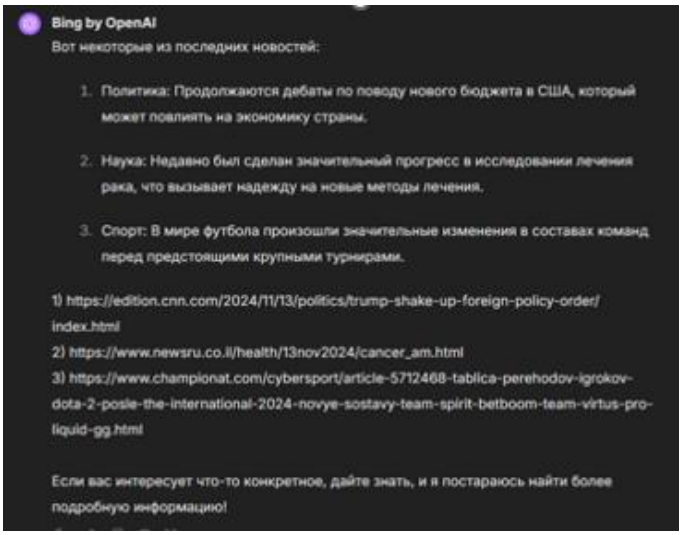


Рисунок 4.32 – Тестування генерації текстової відповіді на основі веб-пошуку

Останній етап дослідження присвячений комплексному тестуванню інтеграції розробленого сервісу агрегації великих мовних моделей з месенджером Telegram через спеціально створеного Telegram-бота (рис. 4.33).

Тестування має на меті ретельну перевірку функціональності API в реальних умовах, включаючи оцінку швидкості та коректності обробки різноманітних запитів, здатності системи надсилати й отримувати повідомлення у режимі реального часу.

Особлива увага приділяється валідації механізмів взаємодії між API сервісу, Telegram-ботом та Backend-інфраструктурою для забезпечення надійної та безперебійної роботи всієї системи.

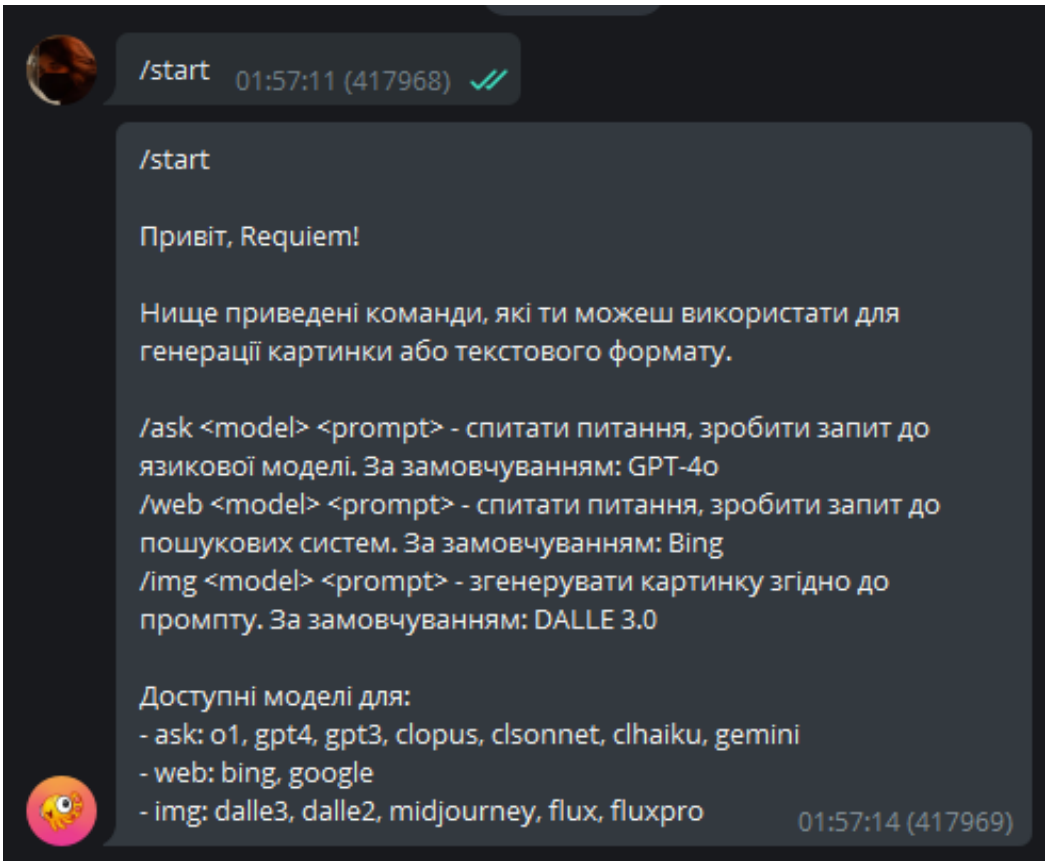


Рисунок 4.33 – Розроблений Telegram-бот для тестування API

Ключовими аспектами є перевірка точності, релевантності та швидкості відповіді на запити, що надходять через цей канал. Тестування забезпечує безперебійну інтеграцію між Telegram та онлайн-системою, що дозволяє користувачам ефективно використовувати можливості мовної моделі або штучного інтелекту в межах популярного месенджера (рис. 4.34).

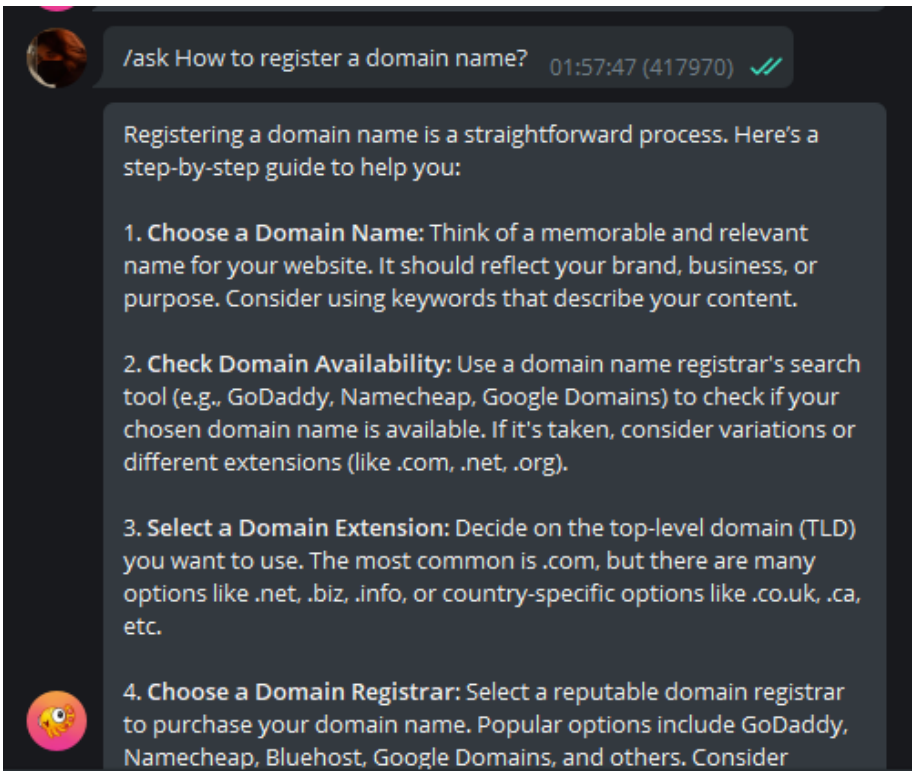


Рисунок 4.34 – Тестування в Telegram генерації тексту

Наступний етап тестування, включає перевірку функціональності створення зображень за допомогою текстових запитів, надісланих через Telegram до API інформаційної системи та розробленої технології.

Оцінюється здатність системи коректно інтерпретувати запити та генерувати зображення, які відповідають опису (рис. 4.35). Тестування також перевіряє швидкість передачі зображень та їх якість після отримання, що є важливим для забезпечення користувацького задоволення в усіх випадках.

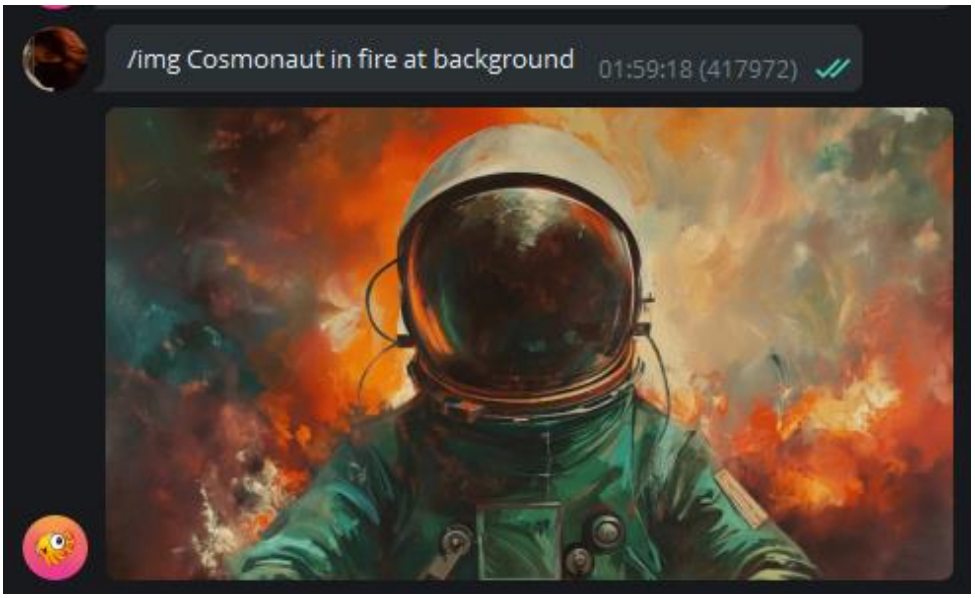


Рисунок 4.35 – Тестування в Telegram генерації зображень

Завершальним етапом є тестування можливостей веб-пошуку при відправці запиту через Telegram до API. Перевіряється, як система справляється з різними типами запитів та її здатність надавати корисні й відповідні результати пошуку користувачам, що використовують Telegram як платформу взаємодії з API інформаційної системи (рис. 4.36).

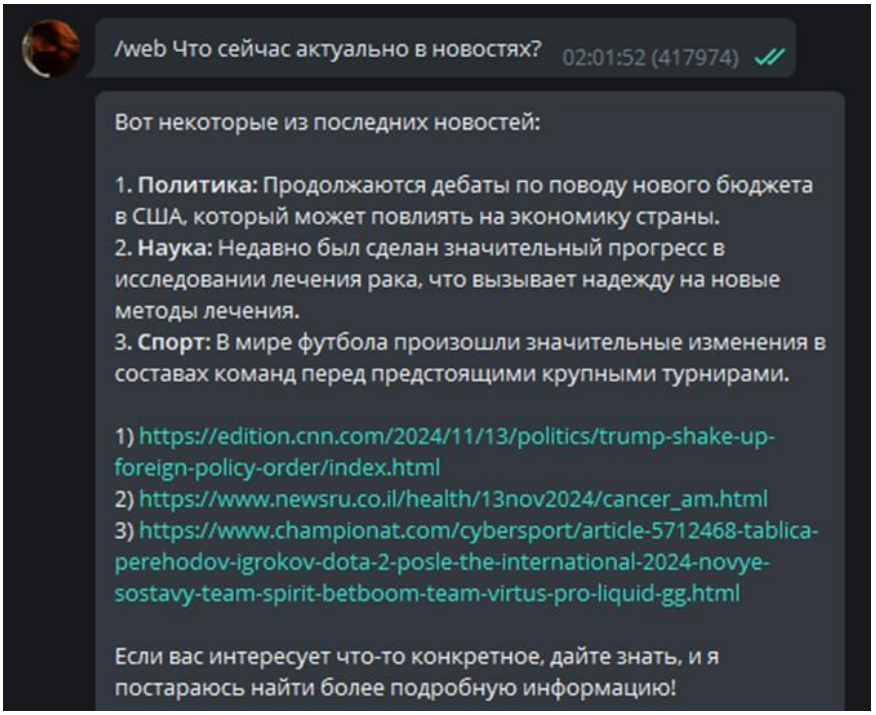


Рисунок 4.36 – Тестування в Telegram веб-пошуку

У підсумках проведеного тестування можна зазначити, що онлайн-сервіс та інформаційна технологія - успішно виконують свої основні функції, забезпечуючи користувачам можливість інтерактивної роботи з великими мовними моделями та штучним інтелектом.

Всі перевірені компоненти, від створення облікового запису до складних запитів через API Telegram, продемонстрували надійну роботу та високу продуктивність. Система ефективно обробляє текстові запити, здійснює генерацію тексту та зображень, а також виконує веб-пошук, що підтверджує її готовність до використання у реальних умовах.

Розробка власного API та інтеграція з месенджером Telegram дозволила розширити можливості доступу до сервісу, надавши користувачам зручну платформу для взаємодії з мовними моделями. Всі запити були опрацьовані коректно, з належною швидкістю, підтверджуючи ефективність системи.

Нарешті, комплексне тестування виявило декілька незначних недоліків, які були оперативно виправлені, що свідчить про гнучкість і адаптивність розробленої системи.

Це підкреслює важливість етапу тестування у процесі розробки, оскільки він дозволяє виявити та усунути потенційні проблеми до того, як система буде розгорнута для широкого використання.

Завдяки ретельному тестуванню онлайн-сервіс готовий забезпечити якісний користувацький досвід, задовольняючи різноманітні потреби у роботі з великими мовними моделями та штучним інтелектом.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи розроблено онлайн-сервіс для агрегації великих мовних моделей, що підвищує доступність технологій штучного інтелекту для широкого кола користувачів. Проведений аналітичний огляд наявних рішень дав змогу виявити їх сильні та слабкі сторони, що сприяло створенню платформи, яка поєднує кращі практики та розв'язує наявні проблеми роботи зі мовними моделями та штучним інтелектом.

Проєктування системи передбачало використання сучасних методів моделювання бізнес-процесів та архітектури програмного забезпечення, що забезпечило гнучкість і масштабованість платформи. Реалізація серверного та клієнтського компонентів з динамічним розподілом навантаження гарантує високу продуктивність і стабільність роботи під час пікових навантажень.

Інноваційним складником розробки є механізми персоналізації, які дають змогу налаштовувати мовні моделі під специфічні потреби користувачів, розширюючи можливості їх застосування в різних галузях. Інтеграційний підхід до роботи з API мовних моделей спрощує їх використання, роблячи штучний інтелект доступним для бізнесу, освітніх установ, побутових користувачів та розробників.

Отже, виконана робота підтверджує ефективність запропонованих рішень, а розроблений сервіс - демонструє надійність, продуктивність і здатність до масштабування, що становить вагомий внесок у розвиток сучасних інформаційних технологій.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Is GPT-3 Text Indistinguishable from Human Text? Scarecrow: A Framework for Scrutinizing Machine Text. *ACL Anthology*. URL: https://aclanthology.org/2022.acl-long.501/ (дата звернення: 25.11.2024).

2. Li C. Here's what GPT-4 Thinks Being a Journalist is All About. *Medium*. URL: https://generative-ai-newsroom.com/heres-what-gpt-4-thinks-being-a-journalist-is-all-about-88028ae27a01 (дата звернення: 25.11.2024).

3. How GPT-4o Can Increase Your Productivity at Work. *jndlens*. URL: https://www.jndlens.com/how-gpt-4o-can-increase-your-productivity-at-work/ (дата звернення: 25.11.2024).

4. Generative AI - Worldwide | Statista Market Forecast. *Statista*. URL: https://www.statista.com/outlook/tmo/artificial-intelligence/generative-ai/worldwide (дата звернення: 25.11.2024).

5. The economic potential of generative AI: The next productivity frontier / *M. Chui et al. McKinsey & Company*. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#business-value (дата звернення: 25.11.2024).

6. PwC's global artificial intelligence study: sizing the prize. *PwC*. URL: https://www.pwc.com/gx/en/issues/artificial-intelligence/publications/artificial-intelligence-study.html (дата звернення: 25.11.2024).

7. The future landscape of large language models in medicine. *PMC Home*. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10564921/ (дата звернення: 25.11.2024).

8. Прогнози Білла Гейтса щодо штучного інтелекту на 2024. *Блог Digital агенції UAMASTER*. URL: https://blog.uamaster.com/prognozy-billa-gejtsa-shhodo-shtuchnogo-intelektu-na-2024/ (дата звернення: 25.11.2024).

9. How to choose the best large language model (LLM) for each and every task. *Domo*. URL: https://www.domo.com/blog/how-to-choose-the-best-large-language-model-llm-for-each-and-every-task/ (дата звернення: 25.11.2024).

10. Open source vs. commercial llms - context clue. *Context Clue - Accelerate your document research through AI*. URL: https://context-clue.com/open-source-vs-commercial-llms/ (дата звернення: 25.11.2024).

11. Commercial vs. self-hosted llms: cost analysis & choosing. *Iguazio*. URL: https://www.iguazio.com/blog/commercial-vs-self-hosted-llms/ (дата звернення: 25.11.2024).

12. Best open source llms of 2024. *Klu.ai*. URL: https://klu.ai/blog/open-source-llm-models (дата звернення: 25.11.2024).

13. purpleSlate. Choosing the Best LLM Model: A Strategic Guide for Your Organization's Needs. *Medium*. URL: https://medium.com/@social_65128/choosing-the-best-llm-model-a-strategic-guide-for-your-organizations-needs-f64794ead5e9 (дата звернення: 25.11.2024).

14. OpenAI GPT-4: A complete review. *Version 1*. URL: https://www.version1.com/blog/openai-gpt-4-review/ (дата звернення: 27.11.2024).

15. Introducing Claude 3.5 Sonnet. *Anthropic*. URL: https://www.anthropic.com/news/claude-3-5-sonnet (дата звернення: 27.11.2024).

16. Google Gemini 1.5 Review: Million-Token AI Changes Everything. *PyImageSearch*. URL: https://pyimagesearch.com/2024/03/04/google-gemini-1-5-review-million-token-ai-changes-everything/ (дата звернення: 27.11.2024).

17. Enthusiast E. H. Mistral 7B and Mixtral 8x7B. *Medium*. URL: https://medium.com/@EleventhHourEnthusiast/paper-reviews-mistral-7b-and-mixtral-8x7b-e8f5a011ebbf (дата звернення: 27.11.2024).

18. Our honest review of llama 3.1 (meta). *The Neuron*. URL: https://www.theneuron.ai/tools/llama (дата звернення: 27.11.2024).

19. Grok-1 model card. *xAI*. URL: https://x.ai/blog/grok/model-card (дата звернення: 27.11.2024).

20. How to pick the right LLM for your business?. *Analytics Vidhya*. URL: https://www.analyticsvidhya.com/blog/2024/09/right-llm-for-your-business/ (дата звернення: 27.11.2024).

21. MongoDB vs mysql - difference between database management systems. *Amazon Web Services, Inc*. URL: https://aws.amazon.com/compare/the-difference-between-mongodb-vs-mysql/?nc1=h_ls (дата звернення: 29.11.2024).

22. Inc F. Introduction to AJAX. *Medium*. URL: https://medium.com/@fortune.nwuneke/introduction-to-ajax-1490897330d3 (дата звернення: 29.11.2024).

23. Examples of software you can build with python and node.js. *SOFTFORMANCE*. URL: https://www.softformance.com/blog/python-vs-nodejs/ (дата звернення: 29.11.2024).

24. Chat user interface design – A quick introduction to chat UI. *Studio by UXPin*. URL: https://www.uxpin.com/studio/blog/chat-user-interface-design/ (дата звернення: 29.11.2024).

25. Groq: a new and fast LLM Provider. *orq.ai*. URL: https://docs.orq.ai/changelog/use-groq-as-your-llm-provider (дата звернення: 29.11.2024).

26. MyScale. How to Fine-Tune an LLM from Hugging Face. *Medium*. URL: https://medium.com/@myscale/how-to-fine-tune-an-llm-from-hugging-face-f9d97270000b (дата звернення: 29.11.2024).

27. Microservices vs apis - difference between modular software design approaches. *Amazon Web Services, Inc*. URL: https://aws.amazon.com/compare/the-difference-between-microservices-and-apis/ (дата звернення: 29.11.2024).

28. Що таке HTML? Для чого він необхідний та його перспективи. *MC.today, Media for Creators*. URL: https://mc.today/uk/shho-take-html-ta-yak-za-dopomogoyu-nogo-uvijti-do-it/ (дата звернення: 29.11.2024).

29. HTML і CSS: що це, кому та для чого потрібно. *GoIT Global*. URL: https://goit.global/ua/articles/html-i-css-shcho-tse-komu-ta-dlia-choho-potribno/ (дата звернення: 29.11.2024).

30. Creative Practice. Що таке JavaScript. *CASES*. URL: https://cases.media/en/article/sho-take-javascript (дата звернення: 02.12.2024).

31. Що таке Python? *aCode*. URL: https://acode.com.ua/intro-python/ (дата звернення: 29.11.2024).

32. Visual Studio Code: налаштування, розширення, комбінації клавіш. *petrov.net.ua*. URL: https://petrov.net.ua/visual-studio-code-settings-extensions-hotkeys/ (дата звернення: 29.11.2024).

33. Порівняння моделей ШІ: ChatGPT, Claude і Gemini. *Все про штучний інтелект в Україні*. URL: https://gptchat.in.ua/porivnyannya-modelej-shi-chatgpt-claude-i-gemini/ (дата звернення: 29.11.2024).

34. Повний посібник для початківців з інструментів LLM Hugging Face. *Unite.AI*. URL: https://www.unite.ai/uk/complete-beginners-guide-to-hugging-face-llm-tools/ (дата звернення: 29.11.2024).

35. Kim L. How to use groq API: understanding its limits and capabilities. *Medium*. URL: https://medium.com/@lisakim01/how-to-use-groq-api-understanding-its-limits-and-capabilities-544843831048 (дата звернення: 29.11.2024).

36. Що таке Node JS простими словами - застосування Node JS у програмуванні. *DAN IT Education*. URL: https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/ (дата звернення: 29.11.2024).

37. Що таке Docker і навіщо він? *QualityAssuranceGroup*. URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 29.11.2024).

38. Послуги Redis: експертна розробка та налаштування від Wezom. *IT-компания полного цикла разработки программных продуктов WEZOM - Киев, Украина*. URL: https://wezom.com.ua/ua/redis-2 (дата звернення: 29.11.2024).

39. Що таке MongoDB? Вступ, Architектура, функції та приклад. *Guru99*. URL: https://www.guru99.com/uk/what-is-mongodb.html (дата звернення: 29.11.2024).

40. Учасники проектів Вікімедіа. Діаграма прецедентів. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів (дата звернення: 29.11.2024).

41. Правила моделювання idef0. *inat.hutsul.cx.ua*. URL: https://inat.hutsul.cx.ua/articles/pravila-modeljuvannja-idef0.html (дата звернення: 29.11.2024).

42. Учасники проектів Вікімедіа. Діаграма розгортання. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Діаграма_розгортання (дата звернення: 29.11.2024).

43. Maxym Z. Діаграма послідовності (sequence diagrams). *Maxym Zosym*. URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 29.11.2024).

ДОДАТОК А

```
// Імпорт необхідних модулів
const express = require('express');
const mongoose = require('mongoose');
const Redis = require('ioredis');
const axios = require('axios'); // Для інтеграції з API мовних моделей

// Ініціалізація додатку Express
const app = express();

// Підключення до MongoDB
mongoose.connect('mongodb://localhost:27017/llm-aggregator', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('Підключено до MongoDB'))
  .catch(err => console.error('Помилка підключення до MongoDB:', err));

// Ініціалізація Redis для кешування
const redis = new Redis();

// Конфігурація середовища
const PORT = process.env.PORT || 3000;

// Маршрутизація запитів
app.get('/api/models', async (req, res) => {
    try {
        // Перевірка кешу Redis
        const cachedResponse = await redis.get('models');
        if (cachedResponse) {
            return res.json(JSON.parse(cachedResponse));
        }

        // Запит до зовнішнього API мовних моделей
        const response = await axios.get('https://api.yakymenko.com/models');
        const models = response.data;

        // Збереження в кеші
        await redis.set('models', JSON.stringify(models), 'EX', 3600); //
Зберігання на 1 годину

        res.json(models);
    } catch (error) {
        console.error('Помилка при отриманні мовних моделей:', error);
        res.status(500).send('Виникла помилка на сервері');
    }
});

// Запуск сервера
app.listen(PORT, () => {
    console.log(`Сервер запущено на порту ${PORT}`);
});
```

ДОДАТОК Б

```
import requests
import json
from requests.exceptions import HTTPError, Timeout, RequestException
import logging

# Налаштування логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

# URL сервера Node.js, який обробляє запити до OpenAI
NODE_SERVER_URL = "http://localhost:3000/api/openai"

# Функція для надсилання повідомлення до серверної частини з підтримкою
авторизації
def send_message_to_openai(message, token):
    headers = {
        'Content-Type': 'application/json',
        'Authorization': f'Bearer {token}'  # Додавання токена авторизації
    }
    # Тіло запиту, яке містить текст повідомлення
    data = {
        'message': message
    }
    try:
        # Виконання POST-запиту до серверної частини з тайм-аутом
        response = requests.post(NODE_SERVER_URL, headers=headers,
data=json.dumps(data), timeout=10)
        # Перевірка статусу відповіді
        response.raise_for_status()  # Піднімає виняток для помилкових
статусів
        # Обробка відповіді від OpenAI
        response_data = response.json()
        logging.info(f"Відповідь від OpenAI: {response_data['response']}")
        return response_data['response']
    except HTTPError as http_err:
        logging.error(f"HTTP помилка: {http_err}")
    except Timeout:
        logging.error("Запит перевищив час очікування")
    except RequestException as req_err:
        logging.error(f"Помилка запиту: {req_err}")
    except Exception as e:
        logging.error(f"Непередбачена помилка: {str(e)}")
    return None  # Повертає None у випадку помилки

# Тестування
if __name__ == "__main__":
    user_message = "Привіт, як справи?"
    token = "your_auth_token_here"  # Замість цього потрібно вставити
реальний токен авторизації
    response = send_message_to_openai(user_message, token)
    if response:
        print(f"Отримана відповідь: {response}")
```