Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра _____Системотехніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____

_Розробка методів створення SaaS рішень із використанням cloud-native_____
_технології в ІС онлайн ринок літератури_____
(тема)

Виконав:
студент _2_ курсу, групи _СПРм-22-1_____
__Романків Назарій Дмитрович_____
(прізвище, ім'я, по батькові)
Спеціальність ____122 Комп'ютерні_____
_____науки_____
(код і повна назва спеціальності)
Тип програми ____освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)
Освітня програма _Системне Проектування__
_____
(повна назва освітньої програми)

Керівник __проф. Ситніков Д.Е._____
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ ___Гребінник І.В_____
(підпис) (прізвище, ініціали)

2024 р.

Кваліфікаційна робота оформлена у відповідності до вимог діючих стандартів та методичних вказівок.

Матеріали кваліфікаційної роботи не містять відомостей, що заборонені для опублікування у відкритих виданнях.

Попередній захист проведено 28 травня 2024 року.

Керівник кваліфікаційної роботи *Д.Е. Ситніков*

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____

Кафедра _____Системотехніки_____

Рівень вищої освіти _____другий (магістерський)_____

Спеціальність _____122 Комп'ютерні науки_____
(код і повна назва)

Тип програми _____освітньо-наукова_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_Системне Проектування_____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_01__»__квітня__ 20 24_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Ромакніву Назарію Дмитровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи ____*Розробка методів створення SaaS рішень із використанням cloud-native технологій в ІС онлайн ринок літератури*_____
_____

затверджена наказом університету від _01 квітня_2024_ р. № _259СТ_

2. Термін подання студентом роботи до екзаменаційної комісії _11_ __червня_____ 2024 р.

3. Вихідні дані до роботи *Функція: Дослідження та використання методів створення SaaS рішень із використанням cloud-native технологій. Перелік використовуваних програмних засобів: ОС Windows, інтегроване середовище розробки WebStorm, Python, JavaScript, Typescript, Node.js, Telegraf. Технічне забезпечення: Персональний компʼютер, зʼєднання з інтернетом.*_____

4. Перелік питань, що потрібно опрацювати в роботі *4.1 Аналіз предметної області; 4.2 Огляд літератури й сучасного стану галузі; 4.3 Постановка Задачі; 4.4 Розробка чат бота на базі LLM та інтеграція в SaaS; 4.5 Розробка архітектури чат боту; 4.6 Вибір LLM для використання в чат боті; 4.7 Розробка методу інтеграції чат боту на базі LLM в SaaS додаток; 4.8 Модель розгортування чат-боту із застосуванням cloud-native технологій 4.9 Розробка методів кастомізації SaaS додатків; 4.10 Аналіз та вибір методів кастомізації;4.11 Розробка надійного й маштабуємого методу доставки вебхуків із застосуванням cloud-native технологій; 4.12 Вирішення проблем безпеки; 4.13 Вирішення проблем надійності; 4.14 Вирішення проблем масштабування та продуктивності; 4.15 Модель розгортування механізму доставки вебхуків із застосуванням cloud-native технологій; 4.16 Розробка інформаційної системи із застосуванням cloud-native технологій 4.17 Висновки; 4.18 Перелік Джерел Посилань*_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних

ілюстрацій 5.1 Високорівнева архітектура SaaS додатку; 5.2 Архітектура чат бота; 5.3 Приклад відповіді чат бота із використанням ChatGPT LLM; 5.4 Високорівнева діаграма інтеграції чат бота із додатком; 5.5 Діаграма процесу індексації даних; 5.6 Модель розгортання чат-боту із застосування cloud-native технологій; 5.7 Методи кастомізації SaaS додатків; 5.8 Модель розгортування доставки вебхуків із застосування AWS сервісів; 5.9 Ключові use-cases ІС «Онлайн ринок літератури»; 5.10 Архітектура серверного додатку; 5.11 Діаграма послідовностей опрацювання події в системі; 5.12 Модель розгортування інформаційної системи «Онлайн ринок літератури»

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання завдання*	*18.01.2024*	*Виконано*
2	*Аналіз предметної області*	*19.01.2024 – 28.01.2024*	*Виконано*
3	*Огляд літератури й сучасного стану галузі*	*29.01.2024 – 11.02.2024*	*Виконано*
4	*Постановка задачі*	*12.02.2024 – 13.02.2024*	*Виконано*
5	*Розробка архітектури чат боту*	*14.02.2024 – 21.02.2024*	*Виконано*
6	*Вибір LLM для використання в чат боті*	*22.02.2024 – 28.02.2024*	*Виконано*
7	*Розробка методу інтеграції чат боту на базі LLM в SaaS додаток*	*29.02.2024 – 07.03.2024*	*Виконано*
9	*Аналіз та вибір методів кастомізації*	*08.03.2024 – 15.03.2024*	*Виконано*
10	*Розробка надійного й масштабуємого методу доставки вебхуків із застосуванням cloud-native технологій*	*16.03.2024 – 26.03.2024*	*Виконано*
11	*Розробка та тестування інформаційної системи із застосуванням cloud-native технологій*	*27.03.2024 – 18.04.2024*	*Виконано*
12	*Аналіз результатів та формування висновків*	*19.04.2024 – 25.04.2024*	*Виконано*
13	*Оформлення пояснювальної записки*	*25.04.2024 – 23.05.2024*	*Виконано*
14	*Подача кваліфікаційної роботи на допуск до захисту*	*28.05.2024*	*Виконано*
15	*Підготовка докладу до захисту кваліфікаційної роботи*	*28.05.2024 – 08.06.2024*	*Виконано*
16	*Захист кваліфікаційної роботи*	*11.06.2024*	*Виконано*

Дата видачі завдання __18__ ___Січня__2024_ р.

Студент _Романків Н.Д._____ ___
(підпис)

Керівник роботи _____ ____проф. Ситніков Д.Е._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: *125* сторінки, *16* рисунків, *2* таблиці, *4* додатки, *33* джерел інформації.

SAAS, CLOUD NATIVE, LLM, ЧАТ БОТ, МЕТОДИ КАСТОМІЗАЦІЇ, NODE.JS, PYTHON

Об'єкт дослідження – SaaS системи побудовані за допомогою Cloud-native технологій. Це включає системи, що побудовані за допомогою Compossable Ecommerce підходу.

Предмет дослідження – методи створення SaaS додатків розроблених за допомогою Cloud Native технологій, включаючи використання сучасних LLM для розробки та інтеграції чат ботів в SaaS додатки, та методи кастомізації для SaaS додатків.

Мета роботи – проаналізувати і оцінити сучасний стан галузі, розробити архітектуру чат-бота що дозволяє використовувати різні LLM без масштабних змін в системі; розробити метод інтеграції чат бота на базі LLM в SaaS додаток; розробити методи кастомізації SaaS додатків.

Методи досліджень – проведення аналізу предметної галузі та сучасного стану галузі, розробка архітектури чат бота, проведення порівняльного аналізу LLM, розробка методів інтеграції чат бота на базі LLM в SaaS додаток, розробка методів кастомізації SaaS.

Наукова новизна обґрунтована тим що аналіз літературних джерел довів, що досі не було розроблено архітектури чат-боту що дозволяє використовувати різні LLM-ки із легкістю. Окрім цього запропонований метод інтеграції чат-бота побудованого на основі LLM, без застосування складного та трудомісткого процесу fine-tuning, що дозволяє чат боту отримати доступ до даних із системи. Також досі не було визначено цілісної стратегії по кастомізації SaaS додатків який би містив сучасні й різносторонні методи для модифікації.

ABSTRACT

Explanatory note to the qualification work: *125* pages, *16* figures, *2* tables, *33* sources, *4* applications,.

SAAS, CLOUD NATIVE, LLM, CHATBOT, METHODS OF CUSTOMISATION, NODE.JS, PYTHON

Object of the study – SaaS systems built using Cloud-native technologies. This includes systems developed using the Composable Ecommerce approach.

Subject of the study – methods for creating SaaS applications developed using Cloud Native technologies, including the use of modern LLMs for the development and integration of chatbots in SaaS applications, and customization methods for SaaS applications.

Purpose of the work – to analyze and evaluate the current state of the industry, develop a chatbot architecture that allows the use of different LLMs interchangeably; develop a method for integrating an LLM-based chatbot into a SaaS application; develop methods for customizing SaaS applications.

Research methods – conducting an analysis of the subject area and the current state of the industry, developing a chatbot architecture, conducting a comparative analysis of LLMs, developing methods for integrating an LLM-based chatbot into a SaaS application, and developing methods for customizing SaaS.

Scientific novelty – the analysis of literature sources proved that there has not yet been a chatbot architecture developed that allows the easy use of different LLMs. In addition, the proposed method for integrating a chatbot built on LLMs, without using the complex and labor-intensive process of fine-tuning, allows the chatbot to access data from the system. Also, a comprehensive strategy for customizing SaaS applications, which includes modern and versatile methods for modification, has not yet been defined.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

SaaS – Software as a Service
LLM – Large Language Model
REST – Representational State Transfer
HMAC – Hash-based Message Authentication Code
API – Application Program Interface
UUID – Universally Unique IDentifier
AWS – Amazon Web Services
SQS – Simple Queue Service
UML – Unified Modeling Language
ІС – Інформаційна Система
UX – User eXperience
UI – User Interface

ЗМІСТ

ВСТУП

Інтернет та сучасні технології дозволили моделі SaaS[1] (Програмне забезпечення як послуга) піднятися та домінувати у світі; багато мільярдних компаній, таких як Netflix, Amazon, Facebook і Airbnb, побудовані на SaaS. Крім того, набирає популярності Альянс MACH [3] і використання компонованих архітектур електронної комерції [4], які дозволяють підприємствам створювати системи як конструктори, зібрані з різних систем SaaS. Крім того, сьогодні бізнес-модель Marketplace стає дедалі популярнішою, згідно з дослідженнями Grand View Research, очікується, що глобальний ринок досягне 968 мільйонів у 2022 році та зросте на 25% у середньому з 2023 по 2030 рік. Усі ці фактори визначають актуальність підходу до розробки SaaS-додатків із використанням хмарних технологій.

Об'єкт дослідження – SaaS системи побудовані за допомогою cloud-native технологій. Це включає системи, що побудовані за допомогою compossable ecommerce підходу.

Предмет дослідження – методи створення SaaS додатків розроблених за допомогою cloud native технологій, включаючи використання сучасних LLM для розробки та інтеграції чат ботів в SaaS додатки, та методи кастомізації для SaaS додатків.

Мета роботи – проаналізувати і оцінити сучасний стан галузі, розробити архітектуру чат-бота що дозволяє використовувати різні LLM без масштабних змін в системі; розробити метод інтеграції чат бота на базі LLM в SaaS додаток; розробити методи кастомізації SaaS додатків.

Методи досліджень – проведення аналізу предметної галузі та сучасного стану галузі, розробка архітектури чат бота, проведення порівняльного аналізу LLM, розробка методів інтеграції чат бота на базі LLM в SaaS додаток, розробка методів кастомізації SaaS.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз Предметної Області

Зростання та домінування моделі SaaS (програмне забезпечення як послуга) можна пояснити появою інтернету та сучасних технологій. Численні мільярдні компанії, включаючи Netflix, Amazon, Facebook і Airbnb, успішно побудували свій бізнес на моделі SaaS. Research 360 повідомляє, що у 2021 році світовий ринок програм SaaS досяг 1777,098 мільярдів доларів США, а до 2027 року, за прогнозами, зросте до 1777,188 мільярдів доларів США. Однак розробка сучасних програм SaaS пов’язана з певною часткою технічних перешкод, які необхідно вирішити, особливо в сучасному світі, де ефективні хмарні технології та інноваційні хмарні рішення широко впроваджуються щодня.

Крім того, використання MACH Alliance [3] і архітектури компонованої електронної комерції [4] дозволяє підприємствам створювати системи як конструктори, зібрані з різних систем SaaS. Кожній системі в цій структурі призначені певні технічні функції бізнесу електронної комерції, такі як пошук, персоналізація, керування замовленнями тощо.

Варто також зазначити, що популярність бізнес-моделі Marketplace сьогодні зростає, згідно з дослідженнями Grand View Research, очікується, що глобальний ринок досягне 968 мільйонів у 2022 році та продовжить зростати з 2023 по 2030 рік. Темп зростання 25%.

Модель SaaS – це спосіб надання програмного забезпечення, при якому програми розглядаються як сервіс, доступний через Інтернет. У випадку SaaS, користувачі не купують програмне забезпечення для установки на свої комп'ютери, а надається доступ до хмарних обчислювальних ресурсів, де програми розгорнуті та готові до використання. Основні особливості SaaS моделі включають:

– доступ через інтернет: користувачі отримують доступ до програмного забезпечення через веб-браузер або спеціальні додатки, які можна завантажити;

– абонентська плата: замість великих витрат на придбання ліцензій, користувачі платять щомісячну або річну плату за використання програмного забезпечення. Це зазвичай включає підтримку та обслуговування;

– оновлення та підтримка: інфраструктура та програмне забезпечення підтримуються постійно в оновленому стані постачальником SaaS. Користувачі автоматично отримують доступ до останніх версій програм, уникнувши необхідності в ручному оновленню;

– масштабованість та гнучкість: SaaS дозволяє легко масштабувати користування програмним забезпеченням відповідно до потреб бізнесу. Користувачі можуть змінювати свої підписки або додавати нові користувачів в залежності від зростання обсягів роботи;

– доступність з будь-якого пристрою: користувачі можуть отримати доступ до SaaS-програм незалежно від місця розташування або пристрою, використовуючи веб-браузер або мобільний додаток.

Розглянемо переваги SaaS моделі для бізнесу:

– зниження початкових витрат: економія на придбанні, установці, та обслуговуванні апаратного забезпечення;

– гнучкість та масштабованість: легка адаптація до потреб бізнесу, можливість швидкого збільшення чи зменшення кількості ліцензій;

– оновлення та підтримка: автоматичне оновлення програмного забезпечення та технічна підтримка з боку постачальника;

– доступність та мобільність: можливість доступу до додатків з будь-якого місця з інтернетом, підтримка віддаленої роботи;

– швидкий старт: без необхідності тривалої установки та налаштування, швидке впровадження нових рішень.

Розробка SaaS додатків має свої особливості та потенційні складності, які потрібно враховувати, щоб досягти успіху. Деякі з цих складнощів включають:

– багатокористувацька архітектура: SaaS додатки зазвичай вимагають багатокористувацької архітектури, де один екземпляр додатка обслуговує багатьох користувачів. Це ставить високі вимоги до проектування системи, зокрема до ізоляції даних, масштабування, та управління ресурсами;

– безпека та приватність даних: оскільки дані зберігаються на серверах провайдера, потрібно забезпечити їхню безпеку та захист від несанкціонованого доступу. Комплексні механізми аутентифікації, авторизації, шифрування та відповідність нормативним вимогам є критично важливими;

– масштабованість: SaaS додатки мають бути гнучкими до змін кількості користувачів та обсягів даних. Розробники повинні передбачити механізми горизонтального та вертикального масштабування, щоб забезпечити стабільність та високу доступність сервісу;

– управління версіями та оновленнями: у SaaS моделі оновлення додатка зазвичай відбуваються централізовано, що вимагає гладкої стратегії розгортання, щоб мінімізувати перерви в роботі сервісу для користувачів. Контроль версій та сумісність з попередніми версіями також є важливими аспектами;

– модель ціноутворення: визначення вартості послуги є ключовим аспектом SaaS. Необхідно розробити гнучку та прозору модель ціноутворення, яка буде привабливою для різних сегментів користувачів та водночас забезпечувати прибутковість проекту;

– інтеграція з іншими сервісами: інтеграція SaaS додатків з іншими сервісами та платформами може бути складною через різноманіття API та стандартів обміну даними. Необхідно забезпечити гнучкість та конфігурування інтеграційних процесів;

– підтримка та навчання користувачів: підтримка користувачів та їх навчання використанню SaaS додатку є важливими для забезпечення високого рівня задоволеності та лояльності. Це включає створення інтуїтивно зрозумілого інтерфейсу, документації, онлайн-курсів та служби підтримки;

– залежність від інтернету: оскільки доступ до SaaS додатків здійснюється через Інтернет, питання якості з'єднання та доступності сервісу стають особливо актуальними. Потрібно забезпечити високу продуктивність та низьку затримку обробки запитів.

Розробка SaaS додатків вимагає комплексного підходу та уваги до деталей у багатьох аспектах, від технічної реалізації до бізнес-моделі та обслуговування клієнтів.

Розглянемо технічні аспекти впровадження рішення SaaS. Багато успішних програм SaaS, таких як Netflix, Twitter, Slack, покладаються на мікросервісну архітектуру. Ця архітектура добре працює для рішень SaaS, оскільки забезпечує високий ступінь гнучкості та масштабованості. Оскільки кожен мікросервіс працює незалежно, розробники можуть легко впроваджувати нові функції та масштабувати різні частини системи відповідно до потреб користувачів. Важливою перевагою архітектури мікросервісів є можливість ізолювати функціональні частини системи, підвищуючи тим самим рівень безпеки та конфіденційності даних користувачів. Крім того, швидке розгортання та оновлення окремих мікросервісів без впливу на інші компоненти системи сприяє гнучким методам розробки та реалізації нових функцій у додатках SaaS.

Однак під час застосування архітектури мікросервісів можна стикнутися із проблемами. Координація та керування великою кількістю мікросервісів може бути складним завданням, особливо для великих програм SaaS. Також враховуйте збільшення витрат на інфраструктуру, необхідних для підтримки кількох мікросервісів, особливо з більшим обсягом користувачів і операцій. Через розподілену природу мікросервісів тестування та налагодження можуть бути складними та вимагати ретельного планування та використання відповідних інструментів для забезпечення якості програми.

Крім того, сучасні програми SaaS використовують мультитенантну архітектуру [5, 6]. Використання мультитенантної архітектури має вирішальне значення для реалізації програм SaaS, оскільки це дозволяє ефективно надавати послуги багатьом користувачам, використовуючи ту саму інфраструктуру та

програмне забезпечення. Така архітектура дозволяє спільно використовувати ресурси інфраструктури, зберігаючи ізоляцію даних кожного користувача. Завдяки такому підходу ресурси використовуються ефективно, оскільки різні користувачі використовують спільну інфраструктуру та обслуговуються уніфікованою версією програмного забезпечення. Для постачальників SaaS це покращує використання ресурсів інфраструктури, а отже знижує витрати на обслуговування системи та підвищує прибутковість. Крім того, використання єдиного екземпляра програмного забезпечення в цій архітектурі спрощує керування системою та оновлення. Постачальники SaaS можуть легко запроваджувати нові версії та функції, не заважаючи іншим користувачам.

Хмарні технології також використовуються при розробці програм SaaS, оскільки використання хмарних служб (наприклад, AWS, Azure, Google Cloud) має численні переваги для розробки та роботи програм SaaS. Розглянемо деякі з них докладніше:

– хмарні служби дозволяють ефективно масштабувати додатки по горизонталі або вертикалі для забезпечення високої доступності та продуктивності;

– хмарні рішення дозволяють легко налаштовувати конфігурацію та параметри вашої інфраструктури відповідно до потреб конкретної програми SaaS;

– інструменти автоматизації в хмарних службах спрощують розгортання додатків, моніторинг, інтеграцію та керування, тим самим полегшуючи розробку та роботу;

– хмарні платформи дозволяють легко змінювати кількість ресурсів залежно від навантаження. Це дозволяє вам збільшити чи зменшити обчислювальну потужність та інші ресурси за потреби;

– хмарна технологія дозволяє користувачам отримувати доступ до додатків SaaS з будь-якої точки світу за наявності підключення до Інтернету. Це робить робочий процес більш гнучким і мобільним;

– зменшує витрати на створення та підтримку інфраструктури, усуваючи необхідність інвестувати у власне обладнання та інфраструктуру;

– хмарні сервіси надають можливість швидко впроваджувати нові версії програм і вносити зміни в режимі реального часу.

Розподіл ринку хмарних послуг серед провідних провайдерів є важливим аспектом, який слід враховувати. У таблиці 1.1 наведено дані [7] щодо цього розподілу. Очевидно, що AWS, Azure і GCP є домінуючими гравцями на ринку. Як наслідок, важливо віддати пріоритет розробці методів, які або використовують інфраструктуру цих конкретних хмарних постачальників, або не залежать від хмари, тобто їх можна використовувати незалежно від обраної хмарної платформи.

Таблиця 1.1 Розподіл ринку постачальників хмарних послуг

Постачальник Хмарних Послуг	Частка ринку на момент 2023 року
Tencent Cloud	2
Oracle	2
SalesForce	3
IBM Cloud	3
Alibaba Cloud	4
GCP	11
Azure	22
AWS	32

Завдяки прогресу веб-технологій модель SaaS стала реальністю. Тепер користувачі можуть отримувати прямий доступ до програм через веб-браузери, усуваючи необхідність встановлення на персональних комп’ютерах або мобільних пристроях. У традиційному налаштуванні SaaS інтерфейсна або клієнтська частина програми побудована за допомогою веб-технологій. JavaScript, як основна мова для веб-розробки на стороні клієнта, відіграє вирішальну роль у створенні динамічних та інтерактивних інтерфейсів, які значно покращують взаємодію з користувачем. Хоча JavaScript широко використовується у розробці веб-додатків, сучасні фреймворки, такі як React, Angular, Vue тощо, посіли центральне місце у формуванні ландшафту веб-розробки.

Таким чином, архітектура високого рівня для програми SaaS має такі ключові елементи, та зображена на рис. 1.1:

– мікросервіси;

– мультітенантна архітектура;

– cloud-native;

– браузер як клієнтська частина.

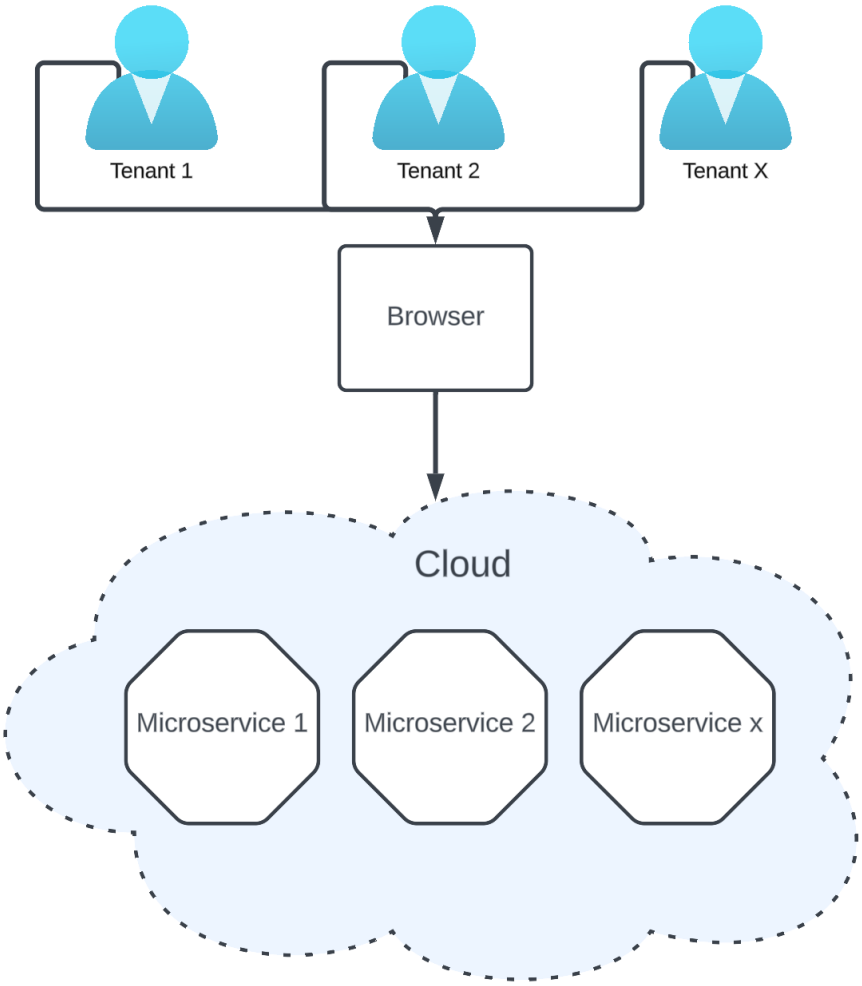

Рисунок 1.1 – Високорівнева архітектура SaaS додатку

SaaS продовжує залишатися однією з найбільш швидкозростаючих сфер технологій. Завдяки аналізу предметних областей ми визначили кілька ключових тенденцій і майбутніх перспектив для SaaS:

– широке застосування штучного інтелекту та машинного навчання: більше компаній використовуватимуть технології штучного інтелекту та машинного навчання для вдосконалення своїх рішень SaaS. Це допоможе

покращити персоналізацію, прогнозування, аналіз даних і автоматизацію бізнес-процесів;

– посилення уваги до кібербезпеки: оскільки кількість онлайн-загроз і потенційних порушень конфіденційності даних продовжує зростати, постачальники SaaS приділятимуть більше уваги захисту даних і забезпеченню високого рівня кібербезпеки;

– збільшення кількості інтеграцій. Компанії все більше звертатимуть увагу на рішення SaaS, які можна легко інтегрувати з іншими системами для забезпечення безперервності бізнесу та обміну даними між різними програмами;

– міжнародне розширення: оскільки Інтернет стає все більш популярним і SaaS використовується в усьому світі, очікується, що географічний обсяг послуг SaaS буде розширюватися, а кількість міжнародних клієнтів зросте;

– підвищена увага до економічно ефективних рішень: компанії шукатимуть рішення SaaS, які допоможуть їм ефективніше використовувати ресурси та зменшити загальні витрати на IT;

– зростаючий ринок SaaS для малого бізнесу: малі підприємства все частіше використовуватимуть рішення SaaS, які пропонують потужні інструменти та послуги за доступними цінами;

– вертикальні ринки, що розвиваються: для задоволення унікальних потреб клієнтів з’являться більш спеціалізовані рішення SaaS, орієнтовані на конкретні галузі чи сегменти ринку.

Отже ринок SaaS додатків різноманітний, а розробка SaaS рішення містить багато складнощів для бізнесу та розробників, які необхідно вирішити щоб побудувати успішний додаток. Оскільки ми не можемо покрити всі проблеми та потенційні рішення, необхідно визначити декілька проблем на яких ми сфокусуємось в цій роботі. Для цього проведемо огляд літератури.

1.2 Огляд літератури й сучасного стану галузі

У дослідженні [8] Анн Л. Роггевен та ін. та Радж Сетхурма підкреслили, що chatlines (схожі на чат-ботів, але з людським присутністю на стороні продавця) допомагають роздрібним торговцям залучати до особистих розмов потенційних покупців, наприклад на веб-сайті, і щонайменше половина компаній з Fortune 5000 експериментувала з чат-ботами, сподіваючись зменшити витрати і покращити якість відповідей. Проте ця робота не деталізує вибір технології чат-бота. Також можемо зробити висновок що розробка чат-ботів для SaaS додатків є перспективною нішею.

Огляд літератури [9] підкреслює різні недоліки чат-бота в порівнянні з людиною, такі як обмежене розуміння і знання, обмежена емпатія та емоції. Проте варто зазначити, що огляд базувався на літературі, виданій між 2020-2021 роками, тобто до великого прориву LLM. LLM мають великий потенціал подолати обмеження, зазначені в огляді літератури.

Оцінка чат-ботів та генерації природної мови є добре відомим викликом, і дослідження [10] запропонували використовувати людську схожість розмовних відповідей для оцінки чат-ботів. Вони описують дві установки для оцінки: статичну, в якій вони порівнюють моделі на фіксованому наборі багатокрокових контекстів для генерації відповідей, та інтерактивну, де вони дозволяють людям вільно спілкуватися з чат-ботами. Вони також вводять два ключові метрики: розумність, конкретність. Крім того, дослідники створили власного багатокрокового чат-бота відкритого домену під назвою Meena та порівняли його з іншими технологіями, але оскільки стаття була написана в 2020 році, ні одна з сучасних LLM не згадана.

Дослідники в [11] висвітлюють виклики, з якими стикаються застосунки чат-ботів, які тісно пов'язані з провайдерами визнавання наміру тексту, що ускладнює їхню підтримку, повторне використання та еволюцію. Зазвичай, коли дизайнер чат-ботів обирає певну платформу розробки чат-ботів, вона потрапляє в ситуацію залежності від постачальника, особливо з двигуном NL (

Natural Language), пов'язаним з платформою. Також сучасні платформи чат-ботів не мають належних механізмів абстракції для легкої інтеграції та комунікації з іншими зовнішніми платформами, з якими компанія може мати потребу взаємодіяти. Робота має на меті вирішити всі ці проблеми, піднімаючи рівень абстракції на якому визначаються чат-боти. У цій цілі вони представляють Xatkit, новий модельний фреймворк для розробки чат-ботів, який спрямований на це питання, використовуючи техніки інженерії на основі моделей: мови, специфічні для області, незалежні від платформи визначення ботів та інтерпретація в реальному часі. Xatkit вбудовує в себе спеціальну мову моделювання, присвячену чат-ботам, для визначення намірів користувача, обчислюваних дій та викликаємої послуги, поєднуючи їх в багатофункціональні потоки розмов. Розмови можуть починатися або користувачем, що розбуджує Xatkit, або зовнішнім подією, яка спонукає Xatkit до реакції (наприклад, сповіщенням користувачу, що відбулася якась подія інтересу на зовнішній службі, на яку підписаний бот).

У дослідженні [12] Гіріджа Аттігері, Анкіт Агравал, і Сучета Колекар зосередилися на розробці чат-бота для поширення інформації технічного університету та провели порівняльний аналіз різних моделей обробки природної мови (NLP). Метою було задовольнити інформаційні потреби майбутніх студентів, надаючи чат-бота на веб-сайті університету, який міг би пропонувати офіційну, однорідну інформацію, доступну 24/7, тим самим допомагаючи студентам приймати обґрунтовані рішення. Дослідники реалізували п'ять моделей чат-ботів, використовуючи різні техніки: нейронні мережі, векторизацію TF-IDF (частота терміну-інвертована частота документа), послідовну модель, визначення шаблонів.

Стаття деталізує процес розробки чат-ботів, включаючи:

– збір даних з ресурсів;

– перед-обробку, таку як токенізація, стемінг та лематизація;

– врахування при проектуванні нейронних мереж, включаючи шари та активаційні функції;

– важливість широкої та добре структурованої бази знань для ефективного створення відповідей.

Це знову доводить, що розробка сучасного чат-бота - це складне завдання, яке вимагає значного зусилля. Проте варто зазначити, що у роботі не досліджувалася використання сучасних LLM, таких як ChatGPT, для розробки чат-ботів.

У статті [13] Гуанвен Мао, Жіндіан Су, Шаншан Ю та Да Луо звертаються до виклику вибору відповіді відповідно до контексту в чат-ботах на основі відновлення. Запропонована ієрархічна агрегаційна мережа з багаторівневим представленням (HAMR) використовує велику кількість представлень контексту та відповіді для покращення процесу вибору.

У роботі [14] дослідники працюють над ключовим питанням "Як ми можемо забезпечити, що системи штучного інтелекту, такі як ChatGPT, розробляються і приймаються відповідально?". Дослідники прийняли комплексний підхід до забезпечення того, що системи штучного інтелекту, зокрема чат-боти для фінансових послуг, розробляються відповідально та етично. Вони вирішили виклик операціоналізації відповідального штучного інтелекту (RAI) в масштабі, створивши методологію інженерії RAI, орієнтовану на шаблони. Дослідники також провели кейс-дослідження щодо розробки чат-ботів для фінансової галузі, щоб продемонструвати застосування каталогу шаблонів RAI. У цьому кейс-дослідженні був описаний процес розробки чат-ботів, від планування та проектування до реалізації, тестування, розгортання та моніторингу. Виділено, як шаблони RAI можуть зменшити ризики на кожному етапі, забезпечуючи відповідальний розвиток чат-ботів. Це включало в себе вирішення етичних питань, забезпечення конфіденційності даних і справедливості, а також врахування різноманітності в команді розробників.

Дослідники [15] провели комплексний огляд, який сфокусований на великих мовних моделях (LLMs), охоплюючи їхню історію, архітектури, застосування та виклики. У статті було підкреслено складність відстеження стрімкого розвитку досліджень LLM через значне збільшення внесків за

короткий період. Для вирішення цього дослідники надали докладний огляд LLM, включаючи їхній розвиток, фундаментальні концепції, архітектури (зокрема трансформатори), методи навчання та набори даних, які використовуються в дослідженнях. Вони також дослідили широкий спектр застосувань LLM у різних галузях, таких як біомедицина та охорона здоров'я, освіта, соціальні мережі, бізнес (включаючи потенційні використання LLM для чат-ботів) та сільське господарство, відзначаючи, як LLM впливають на суспільство та майбутнє штучного інтелекту. Крім того, у статті обговорюються відкриті питання та виклики впровадження LLM у реальній дійсності, пропонуючи знахідки щодо майбутніх напрямків досліджень та розвитку.

Крім того, аналіз джерел доводить, що питанням, пов’язаним з налаштуванням системи SaaS, приділено увагу багато наукових робіт, тому це ще один перспективний напрямок.

Тож Ху Сонг, Френк Шовель та інші [16] провели якісне дослідження підтримки налаштування SaaS. У центрі уваги цього дослідження — вивчення очікування консультантів, які спеціалізуються на адаптації систем корпоративного програмного забезпечення Традиційно налаштування здійснювалися за припущенням, що програмне забезпечення буде встановлено у власному дата центрі, надаючи йому повний контроль і можливість вносити зміни за бажанням. Однак із переходом до багатокористувацького програмного забезпечення як послуги це припущення більше не відповідає дійсності. У цій новій моделі програмне забезпечення працює в хмарі та керується постачальником SaaS. Тому деякі люди вважають, що налаштування SaaS надто складне і від нього слід відмовитися. У той же час практика в галузі SaaS диктує, що конфігурація краща, ніж налаштування. Два постачальники програмного забезпечення, які беруть участь у цьому дослідженні, також почали з підтримки конфігурації своїх рішень SaaS. Однак виявилося, що багато їхніх партнерів (користувачів систем SaaS) думають по-іншому, тому ці постачальники постійно отримують запитання щодо підтримки налаштування

своїх продуктів SaaS. Це дослідження також теоретизує на основі досліджень, що налаштування залишаються важливими для SaaS, але багатокористувацька природа сервісів змінює відповідальність зацікавлених сторін. Партнери (користувачі SaaS) тепер не можуть самостійно модифікувати продукт для задоволення потреб клієнтів і повинні покладатися на постачальника SaaS для розробки та впровадження системних змін і нових функцій. Тому партнери повинні відмовитися від старого підходу вільної розробки та модифікації всього продукту за бажанням. Натомість їм слід прийняти нові формати розробки з більшими обмеженнями та залученням постачальників, а також більше зосередитися на бізнесі. Цей зсув суперечить традиційним інтересам партнерів щодо мінімізації витрат клієнтів, бездоганної інтеграції власного коду з основними продуктами та забезпечення гнучкості. Таким чином, ключовим фактором успіху екосистеми налаштування є те, що постачальники не лише надають високоналаштований SaaS, але й забезпечують ефективні інструменти підтримки, які відповідають новим типам налаштування, компенсуючи ці основні переваги. Необхідність у налаштуванні виникає через розрив зв’язку між клієнтами та постачальниками. Постачальники вміють надавати узагальнене корпоративне програмне забезпечення, але їм не вистачає досвіду та засобів для розуміння тонкощів конкретного бізнесу кожного клієнта. З іншого боку, конфігурація сама по собі не може замінити налаштування. Це пояснюється тим, що він не в змозі усунути вищезгадану прогалину, оскільки намагається охопити всі потенційні функції, які постачальник може передбачити та включити до сервісу.

Крім того, у роботі Ральфа Міцнера, Андреаса Матцгера та ін. [17] наводиться аргумент: Щоб отримати вигоду від ефекту масштабу, постачальники SaaS повинні ефективно залучити значну базу користувачів для своїх програм SaaS. Очевидно, що індивідуальні очікування клієнтів щодо функціональності та якості програмного забезпечення можуть відрізнятися. Тому постачальники SaaS повинні враховувати різні потреби різних

потенційних клієнтів. Це означає, що додатки SaaS повинні дозволяти налаштування для кожного користувача.

Крім того, Wei Sun та ін. [18] прийшли до висновку, що конфігурація та налаштування є ключовими аспектами при розробці систем SaaS. Хоча надзвичайно важливо, щоб функціональні можливості програми SaaS були добре розроблені для задоволення потреб якомога більшої кількості клієнтів у цільовому сегменті споживачів і галузі додатків, рівень наданих параметрів конфігурації та налаштування забезпечує ключову конкурентну перевагу на ринку. У багатьох випадках, коли розробити програму SaaS як стандартну пропозицію для більшості клієнтів важко, висока можливість конфігурації та налаштування будуть критичними факторами успіху.

Роботи [19, 20] досліджують використання мікросервісів у налаштуванні системи SaaS. Цей метод налаштування рекомендовано, коли потрібне глибоке налаштування.

Підхід, згаданий у [19], надто тісно пов’язаний зі стеком технологій .Net і розроблений таким чином, як наче постачальник SaaS надасть доступ до системного коду та інфраструктури, що в більшості випадків неможливо, оскільки для компаній, які продають SaaS загалом, програмне забезпечення та код є найважливішим активом підприємства.

У той же час підхід, розроблений у [20], забезпечує сучасний хмарний та динамічний підхід до використання мікросервісів. У цьому підході кожна настройка реалізується за допомогою мікросервісів. Ця робота демонструє, як таку архітектуру можна реалізувати за допомогою синхронних викликів, окремої бази даних NoSQL і середовища на основі Docker. Але цей підхід не передбачає модифікації інтерфейсу програми SaaS і побудований на спеціальній структурі MiSC-Cloud, що робить цей підхід майже непрактичним.

Еспен Нордлі [21] розглядав у своїй роботі можливість глибоких змін внаслідок подій. Архітектура програми також прив’язана до певної структури MiSC-Cloud і побудована на припущенні, що провайдер використовуватиме

певну шину подій і надаватиме налаштовані мікросервіси, які мають виконуватися шляхом підписки на події та модифікації.

1.3 Постановка задачі

Дослідження зв'язаних робіт підтверджує, що чат-боти відіграють важливу роль у сучасному бізнесі, а розробка їх вимагає великої кількості зусиль та знань для створення бота, який відповідає бізнес-цілям та побажанням клієнтів. До сих пір жодна з робіт не досліджувала використання сучасних LLM, таких як ChatGPT, для розробки чат-ботів для застосунків SaaS.

Отже по відношенню до розробки чатботів необхідно:

– побудувати архітектуру чат-бота, яка може використовувати різні LLM взаємозаміно;

– проаналізувати, порівняти та вибрати найкращий LLM для використання в розробці чат-бота для застосунків SaaS;

– розробити метод інтеграції чат-бота що застосовує LLM в SaaS додаток.

Крім того, дослідження літературних джерел показало, що роботи, присвячені модифікації систем SaaS, розкривають дуже специфічні методи модифікації, які базуються на певному наборі технологій. Однак цей підхід має недоліки і не може бути універсально застосований до всіх платформ SaaS. Крім того, бракує комплексної стратегії налаштування додатків SaaS, яка б охоплювала сучасні та адаптовані методи модифікації. Важливо підкреслити налаштування інтерфейсу SaaS, оскільки воно відіграє життєво важливу роль у пристосуванні SaaS до унікальних вимог окремих клієнтів, аспект, який був упущений у вищезгаданих роботах. Отже, у зв’язку з налаштуванням додатків SaaS вкрай важливо розглянути ці міркування:

– визначити основні методи кастомізації для сучасних SaaS додатків;

– дослідити та визначити спосіб модифікації інтерфейсу SaaS додатків;

– на підставі проведено дослідження визначити цілісну стратегію кастомізації SaaS додатків;

– розробити надійний та масштабуємий метод доставки вебхуків в SaaS додатку.

Для демонстрації можливостей розроблених методі необхідно розробити інформаційну систему «Онлайн ринок літератури», та інтегрувати дані методи у додаток. Для розробки інформаційної системи необхідно виконати наступні дії:

– проаналізувати вимоги до інформаційної системи;

– розробити use-case діаграму із ключовими варіантами застосування системи;

– розробити архітектуру серверної частини додатку;

– розробити модель розгортування додатку із застосуванням cloud native технологій;

– розробити інтерфейс інформаційної системи.

2 РОЗРОБКА ЧАТ БОТА НА БАЗІ LLM ТА МЕТОДУ ІНТЕГРАЦІЯ В SAAS

2.1 Розробка архітектури чат боту

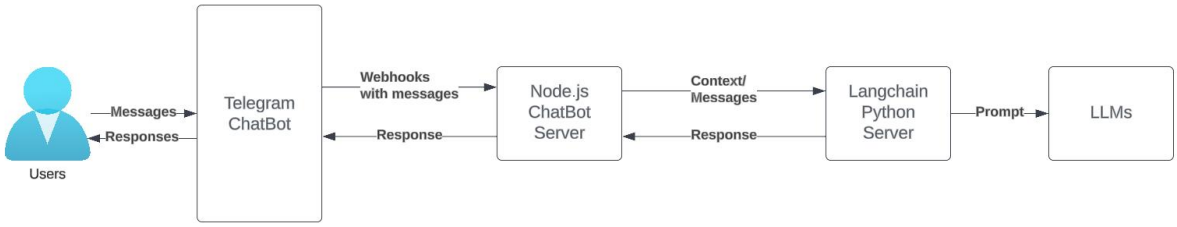

Рисунок 2.1 – Архітектура чат бота

Діаграма (рис. 2.1) наводить архітектуру системи чат бота, яка була розроблена. Система інтегрується з платформою Telegram та використовує великі мовні моделі через фреймворк Langchain. Ось детальний опис кожного компонента та потоку даних:

– Users: це особи, які взаємодіють з чатботом через платформу Telegram. Вони надсилають повідомлення боту та отримують відповіді від нього;

– Telegram ChatBot: представляє собою фронтенд системи чатбота, де відбувається взаємодія з користувачами. Чатбот Telegram налаштований на спілкування з користувачами та обробку вхідних повідомлень;

– Webhooks with messages: коли користувач надсилає повідомлення чатботу Telegram, сервери Telegram використовують вебхуки, щоб переслати це повідомлення на призначений сервер чатбота Node.js;

– Node.js ChatBot Server: це застосунок на серверному боці, написаний на Node.js, який отримує повідомлення від чатбота Telegram через вебхуки. Він обробляє ці повідомлення, зберігає стан розмови та вирішує, як відповісти. Сервер може виконувати різноманітні завдання, такі як розбір команд, ведення журналу повідомлень, керування сеансами користувачів та підготовка контексту для LLMs;

– Context/messages: після обробки початкового повідомлення сервер Node.js формулює контекст або структурований запит, який включає необхідну

інформацію, необхідну для генерації відповіді LLM. Цей контекст може включати вміст повідомлення, історію розмови та будь-які інші відповідні дані;

– Langchain Python Server: це сервер на основі Python, який використовує Langchain, фреймворк, призначений для побудови застосунків з LLMs. Сервер отримує контекст/повідомлення від сервера Node.js. Потім Langchain будує відповідний запит для відправки LLM на основі отриманого контексту;

– LLMs: це великі мовні моделі, які генерують відповіді на основі отриманих запитів. LLM — це високорозвинена модель штучного інтелекту, здатна розуміти та генерувати текст, схожий на людину. Після обробки запиту LLM надсилає отриману текстову відповідь;

– Шлях відповіді: відповідь від LLM надсилається назад на сервер Langchain Python, який потім пересилає її на сервер чат бота Node.js. Сервер Node.js обробляє цю відповідь за потреби (це може включати форматування або додаткову логіку) та надсилає її назад чат боту Telegram;

– Telegram ChatBot до користувачів: нарешті, чат бот Telegram надсилає оброблену відповідь від сервера Node.js назад користувачеві, завершуючи цикл взаємодії.

Ця архітектура дозволяє розділити відповідальності, де сервер Node.js відповідає за керування взаємодією, а сервер Python з Langchain фокусується на використанні LLMs для обробки природної мови. Ця модульна структура забезпечує простоту обслуговування та масштабованість, оскільки кожна частина системи може бути оновлена або масштабована незалежно.

Telegram був обраний як платформа для розміщення чат бота, оскільки створення чат бота за допомогою Telegram має кілька переваг завдяки широкій популярності платформи, міцному API та зручним для користувачів функціям. Основні причини використання Telegram для розробки чат ботів:

– широка аудиторія: Telegram має велику та постійно зростаючу аудиторію користувачів, що надає розробникам чат ботів доступ до великої аудиторії. Це широке поширення полегшує досягнення та залучення користувачів з усього світу. Більше того, згідно з останніми дослідженнями,

47% українців віддають перевагу використанню Telegram для читання новин тощо, що ще раз підкреслює популярність платформи;

– потужний Bot API: API ботів Telegram є всебічним, добре документованим та регулярно оновлюється. Воно пропонує розробникам широкий спектр функцій для створення ботів, які можуть надсилати та отримувати повідомлення, зображення, відео, документи та інше. API також підтримує розширені функції, такі як вбудовані запити, спеціальні клавіатури та багатомедійні повідомлення, що дозволяє створювати високо-інтерактивні та багатофункціональні чат боти;

– простота використання та розгортання: налаштування чат бота в Telegram просте, з простим процесом реєстрації через інтерфейс BotFather. Після створення розгортання та оновлення бота є так само непроблемними, що дозволяє розробникам зосередитися на поліпшенні функціональності, а не на проблемах розгортання;

– високий рівень безпеки: Telegram відомий своїм сильним зобов'язанням до безпеки та конфіденційності. Платформа пропонує безпечну комунікацію з опціями шифрування з кінця в кінець, що забезпечує, що повідомлення між користувачами та ботами залишаються приватними. Ця безпека особливо важлива для ботів, які обробляють конфіденційну інформацію або особисті дані;

– сумісність з різними платформами: боти Telegram працюють безперешкодно на всіх версіях платформи, включаючи настільні, веб- та мобільні додатки. Ця сумісність з різними платформами забезпечує, що боти можуть досягати користувачів незалежно від їхнього обраного пристрою, підвищуючи доступність та залучення користувачів;

– безкоштовність використання: Telegram не стягує плату у розробників за створення чи розгортання ботів, що робить його привабливим варіантом для проектів будь-якого розміру, включаючи стартапи та незалежних розробників. Відсутність плати допомагає зменшити загальні витрати на проекти чат ботів;

– підтримка налаштування та брендингу: Telegram дозволяє розробникам великою мірою налаштовувати своїх ботів, включаючи встановлення імен користувачів, профільних зображень та описів. Цей рівень налаштування допомагає у брендингу бота та покращує його пізнаваність серед користувачів;

– спільнота та екосистема: на платформі Telegram існує жива спільнота розробників і користувачів чат ботів, а також ростуча екосистема бібліотек, фреймворків та інструментів, спрямованих на спрощення розробки ботів на платформі. Підтримка та ресурси спільноти можуть бути надзвичайно цінними, особливо для вирішення викликів розробки та обміну найкращими практиками;

– можливість монетизації: Telegram надає функції, такі як Платіжний API, який дозволяє ботам приймати платежі від користувачів безпосередньо в додатку. Це відкриває можливості для монетизації, особливо для ботів, які надають преміальні послуги, продають товари або полегшують трансакції.

Node.js [22, 23] обраний для розробки сервера чат бота для Telegram, оскільки він пропонує кілька переконливих переваг, особливо враховуючи характер чат ботів та вимоги до обробки даних у реальному часі, масштабованості та простоти інтеграції з різноманітними API та сервісами. Ось деякі причини, чому Node.js є розумним вибором для розробки чат ботів для Telegram:

– архітектура, заснована на подіях: Node.js працює на подійно-орієнтованій, неблокуючій моделі вводу/виводу, що робить його дуже ефективним і підходить для реального часу, таких як чат боти. Ця архітектура забезпечує можливість серверу керувати кількома з'єднаннями одночасно без сповільнення, що є критичним для чат ботів, які мають взаємодіяти з багатьма користувачами одночасно;

– масштабованість: Node.js розроблений з урахуванням масштабованості як по вертикалі, так і по горизонталі, що означає, що ваш чат бот для Telegram легко може зростати, щоб вмістити більше користувачів або обробляти пікові навантаження без значних змін в основній архітектурі. Ця масштабованість є

важливою для успішних застосунків чат ботів, які можуть дуже швидко зростати в залученні користувачів;

– висока швидкість роботи: Завдяки своїй неблокуючій моделі вводу/виводу та двигуну JavaScript V8, Node.js пропонує швидке виконання операцій, що перекладається в швидкі відповіді для користувачів, що взаємодіють з вашим чат ботом для Telegram. Швидкий час реакції є критичним для підтримки плавного та захоплюючого користувацького досвіду;

– багата екосистема бібліотек та інструментів: У спільноті Node.js розроблено величезну кількість бібліотек та інструментів, які можуть спростити та прискорити розробку чат ботів. Для чат ботів для Telegram існують конкретні бібліотеки, такі як node-telegram-bot-api, які надають прості інтерфейси до API Telegram Bot, що дозволяє швидко розробляти та розгортати функціональність бота;

– єдиний стек розробки на JavaScript: якщо ви використовуєте JavaScript або TypeScript для інших частин вашого проекту, Node.js дозволяє використовувати ту саму мову для розробки на серверному боці. Ця єдина стек може спростити процеси розробки, зменшити переключення контексту для розробників та спростити інтеграцію між логікою бота та системами на бекенді;

– підтримка WebSocket: додатки для обміну повідомленнями в реальному часі, такі як чат боти, часто вимагають підтримки WebSocket для миттєвого спілкування. Node.js має відмінну підтримку WebSocket, що є важливим для розробки ботів, які потребують підтримки постійних з'єднань для надання оновлень та відповідей в реальному часі;

– спільнота та підтримка: спільнота Node.js є великою та активною, надаючи величезну кількість ресурсів, уроків та форумів підтримки. Ця обширна спільнота може бути надзвичайно цінною для вирішення викликів розробки, оновлення з найкращими практиками та використання покращень та плагінів, що розробляються спільнотою;

– простота інтеграції з хмарними сервісами та API: Node.js сприяє простій інтеграції з широким спектром хмарних сервісів та сторонніми API, що може

покращити можливості вашого чат бота для Telegram. Незалежно від того, чи потрібно вам підключатися до баз даних, інтегрувати сервіси штучного інтелекту або використовувати хмарне сховище, Node.js може ефективно справитись із цими вимогами.

Langchain [24] – це фреймворк, призначений для полегшення розробки застосунків, які використовують великі мовні моделі (LLMs), такі як серія GPT (Generative Pretrained Transformer) від OpenAI, серед інших. Використання Langchain для розробки чат ботів може мати кілька переваг, особливо для проектів, які мають на меті інтеграцію передових можливостей обробки природної мови (NLP). Деякі ключові причини врахування Langchain для розробки чат ботів:

– спрощує інтеграцію LLMs: Langchain надає інструменти та абстракції, які полегшують інтеграцію великих мовних моделей у ваш чат бот. Це може значно зменшити складність та час розробки, пов'язаний з використанням цих потужних моделей для завдань розуміння та генерації природної мови;

– підвищує розмовні здібності: використовуючи LLMs, чат боти можуть досягати більш складних та природних розмовних здібностей. Langchain може допомогти створювати чат боти, які краще розуміють контекст, керують більш складними діалогами та генерують відповіді, які є послідовними, відповідними та схожими на людські;

– підтримує користувацьку доступність та розширюваність: Langchain дозволяє розробникам налаштовувати та розширювати свої чат боти з урахуванням конкретних вимог. Незалежно від того, чи йдеться про інтеграцію знань, що відносяться до конкретного домену, впровадження спеціальної логіки розмови або додавання унікальних функцій, Langchain надає гнучкий фреймворк, який може врахувати різноманітні потреби користувачів;

– сприяє швидкому прототипуванню: для команд, які шукають можливість швидко прототипувати та тестувати свої концепції чат ботів, Langchain може прискорити процес розробки. Його високо рівневі абстракції та

готові компоненти дозволяють розробникам зосередитися на функціональності та досвіді користувача чат бота, не піддаючись технічним складнощам;

– сприяє впровадженню найкращих практик: Langchain розроблений з урахуванням найкращих практик роботи з LLMs, включаючи аспекти, що стосуються оптимізації продуктивності, обробки чутливих даних та налаштування моделей. Використання Langchain може допомогти забезпечити, що ваш проект чат бота дотримується цих практик з самого початку;

– спільнота та підтримка: використання фреймворку, такого як Langchain, означає приєднання до спільноти розробників і дослідників, що працюють над подібними проектами. Це може забезпечити цінні ресурси, підтримку та можливості співпраці;

– економічність: Langchain може допомогти оптимізувати взаємодію з LLMs, потенційно зменшуючи кількість необхідних викликів API або оптимізуючи обробку даних. Ця ефективність може перетворитися на економію витрат, особливо для проектів, які сильно покладаються на платні API-сервіси для мовних моделей. Це особливо важливо для SaaS-застосунків з великим обсягом трафіку.

Нижче ви можете побачити приклад відповіді чат бота, які надаються за допомогою ChatGPT LLM, інтегрованого в наш додаток.

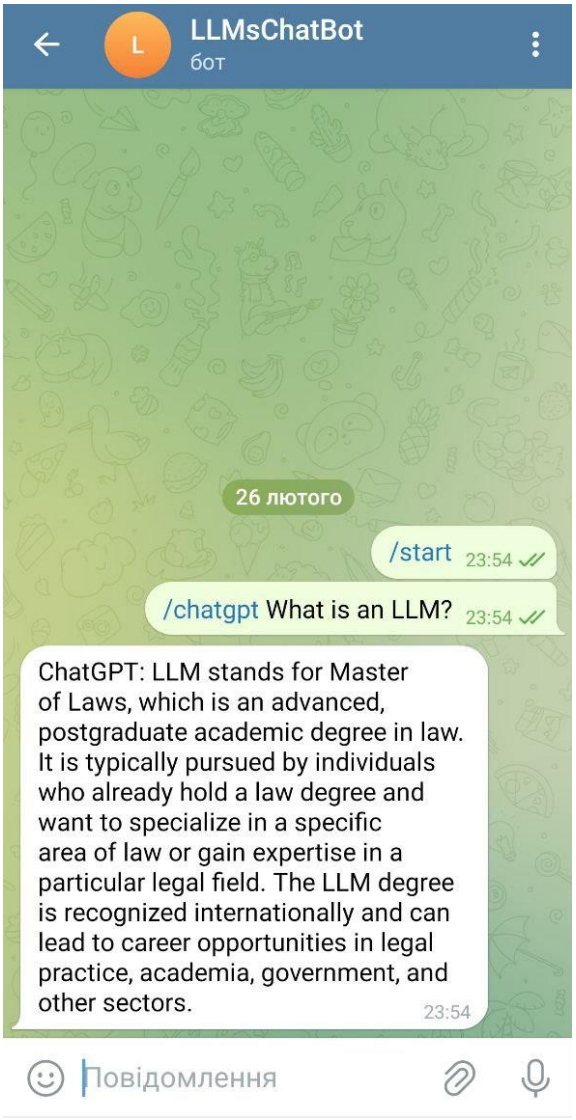

Рисунок 2.2 – Приклад відповіді чат бота із використанням ChatGPT LLM

2.2 Вибір LLM для використання в чат боті

LLM швидко розвиваються та їхня кількість розширюється і вже містить безліч інструментів, тому ми не можемо охопити всі можливі LLM-ки. У межах цієї роботи ми зосередимося на трьох моделях, які мають API: OpenAPI ChatpGTP [25], Cohere [26], Llama [27]. Використання їх надає кілька переваг завдяки їхній технології, здібності та легкості інтеграції. Крім того, ці LLMs навчені на різноманітних наборах даних, що дозволяє їм розуміти та генерувати відповіді на широкий спектр тем і контекстів. Це робить чат боти більш універсальними і здатними обробляти складні запити користувачів. API

дозволяють налаштовувати тон, стиль та відповіді чат бота з урахуванням конкретних випадків використання. Використання цих API може значно зменшити час та витрати, пов'язані з розробкою чат бота з нуля, включаючи ресурси, необхідні для навчання та підтримки моделі штучного інтелекту.

Ми також розглядали можливість використання Gemini від Google, але, на жаль, він недоступний в Україні для API-інтеграції.

Порівняння великих мовних моделей включає оцінку різних аспектів, таких як якість тексту, характеристики моделі, обчислювальна ефективність та застосовність в реальному світі. Ми використовували різні стимули (повідомлення, prompt), щоб оцінити, як веде себе LLM в різних обставинах, зосереджуючись на таких аспектах, як:

– відповіді на питання;

– розуміння та генерація мови;

– розуміння спільного розсуду;

– технічне розуміння;

– аналіз упередженості та чутливості.

Крім стимулів, ми також маємо врахувати ключові метрики для LLM, а саме:

– розмір та складність моделі;

– навчальні дані;

– час відповіді;

– вартість.

Результати експериментів із LLM чат ботами із різними питаннями надається в Додатку Г.

Хоча всі великі мовні моделі (LLMs) змогли надати обґрунтовані та неупереджені відповіді, модель Cohere має тенденцію до довгих та деталізованих відповідей, навіть коли це не є доцільним. Важливо зазначити, що модель Cohere не дотримувалася обмежень надавати відповідь лише у чотирьох реченнях, зазначених у стимулі, що відіграє ключову роль у можливостях LLMs.

Модель LLaMa надає найбільш "емоційні" відповіді, що, можливо, може бути зумовлено високим параметром температури, який має модель. На жаль, ми не можемо контролювати параметр температури LLaMa через надане API.

ChatGPT в цілому надає найбільш відповідні відповіді: короткі, де це потрібно, і детальні у випадку, якщо ви запитуєте про складні та технічні питання.

Усі LLMs мають неупереджені або нейтральні відповіді, що свідчить про те, що навчальні дані були високої якості.

Порівняємо ключові метрики LLM-ок.

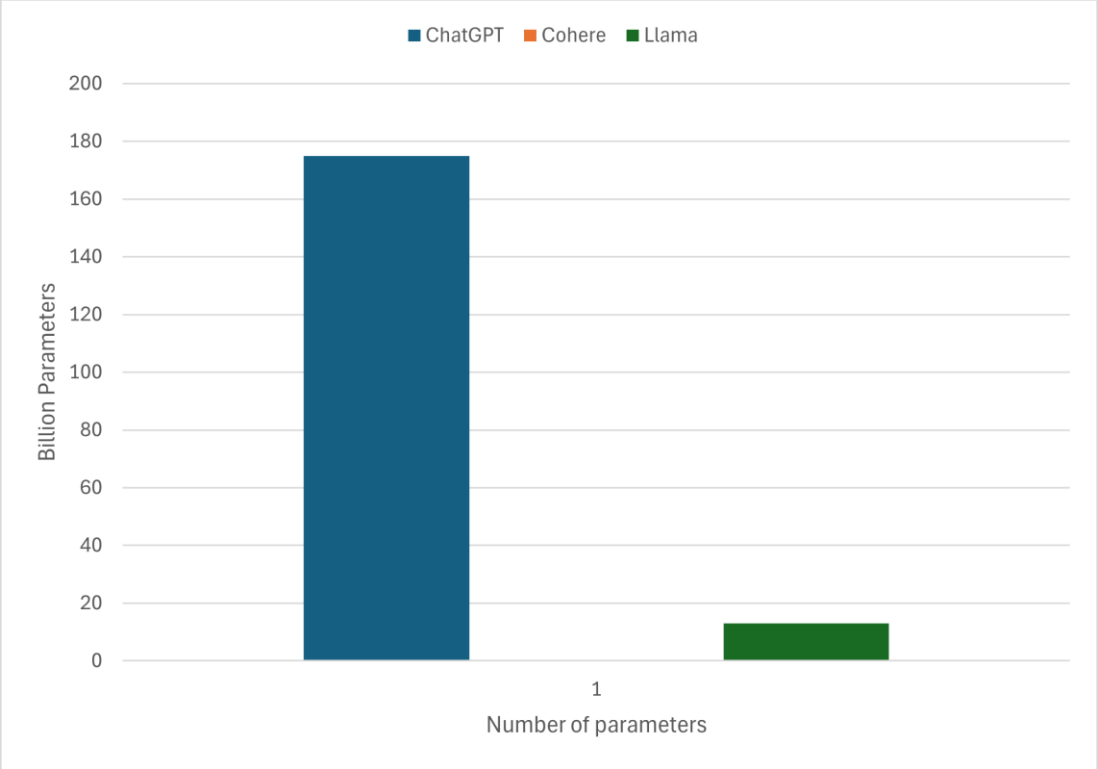

Рисунок 2.3 – К-ть параметрів в моделі

Нажаль, немає загальнодоступної інформації про розмір моделі LLM від Cohere, тому на рисунку 2.3 ці данні відсутні.

Таблиця 2.1 – Порівняння розміру даних для тренування

	ChatGPT	Cohere	Llama
Не фільтровані	45TB	3TB	Невідомо
Фільтровані	570GB	200GB	Невідомо

Також порівняємо середній час відповіді, графік надається на рисунку 2.4.

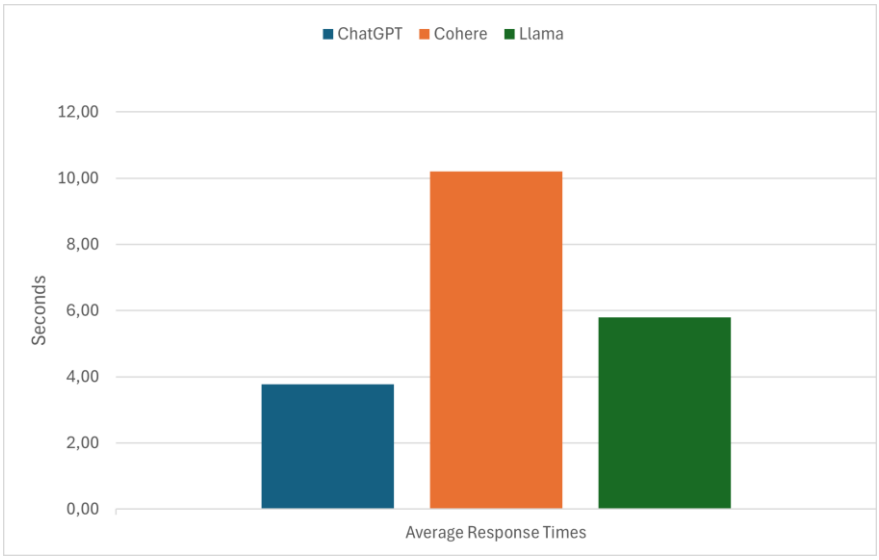

Рисунок 2.4 – Середній час відповіді

Порівняння ціни на 1000 токенів в відповіді надається на графіку 2.5.

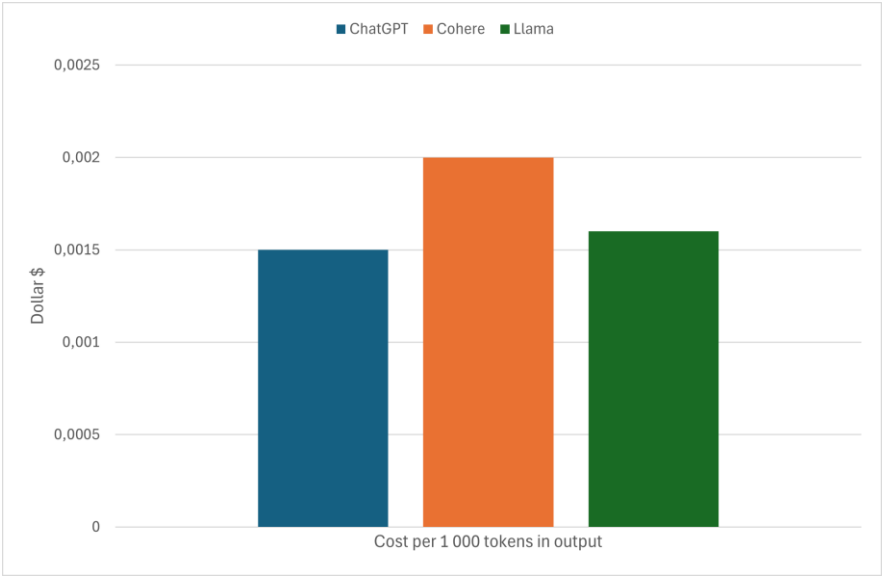

Рисунок 2.5 – Ціна на 1000 токенів в відповіді

У цілому, ChatGPT виявляється найшвидшим за часом відповіді, а в той же час надає хорошу вартість, тоді як LLaMa, здається, є найбільш економічним варіантом. З іншого боку, Cohere відповідає найповільніше і є найбільш

дорогим за 1000 виведених токенів. Варто зазначити, що при порівнянні часу відповіді ми не враховуємо кількість токенів, що входять у відповідь в середньому.

З усіх варіантів, ChatGPT, є найкращим вибором в даному контексті, оскільки він надає найбільш актуальні відповіді з вражаючою швидкістю, при цьому стягуючи прийнятну ціну за 1000 токенів. Також варто пам'ятати, що OpenAI - це провідна компанія в галузі, яка надає найбільше інновацій, тому при використанні їхніх інструментів бізнес робить стратегічне рішення, яке принесе користь у довгостроковій перспективі.

2.3 Розробка методу інтеграції чат боту на базі LLM в SaaS додаток

Для успішного використання чат боту на базі LLM в SaaS додатку, необхідно щоб чат бот міг відповісти на питання специфічні для певного додатку/бізнесу. Також слід зазначити що більшість моделей натреновані на публічних даних доступних до певного року, так наприклад ChatGPT має данні за 2023 рік, а не за 2024, а отже може надати обмежену або не достовірну інформацію. Орім цього, будь-яка LLM тренується виключно на публічних даних, що є ще одним обмеженням, оскільки є певні ніші в яких дуже мало публічної інформації а вся документація зберігається виключно на приватних серверах.

Розглянемо стандартний метод для кастомізації поведінки великої мовної моделі під назвою fine-tuning (тонке налаштування). Тонке налаштування моделі це адаптація попередньо навченої моделі до конкретного випадку використання шляхом подальшого навчання на власному наборі даних.

Серед плюсів даного методу є:

– висока індивідуалізація: можливо налаштувати відповіді моделі відповідно до ваших конкретних потреб;

– спеціальні знання: можливо вбудувати спеціалізовану інформацію для підвищення точності для конкретних галузей або завдань;

– володіння моделлю: ви маєте повний контроль над моделлю, яка може розміщуватися на вашій інфраструктурі.

Нажаль попри сильні сторони цього підходу у нього також є й дуже значні негативні сторони а саме:

– складність: fine-tuning потребує технічних знань у галузі машинного навчання та доступу до обчислювальних ресурсів;

– витрати та складність обслуговування: навчання та підтримки моделей це дорого та ресурсномістко;

– масштабованість: необхідно вирішувати проблеми масштабування моделі самостійно;

– оновлення: підтримка специфічної моделі потребує безперервного перенавчання для включення оновлень та покращень.

Отже можемо зробити висновок що через високий рівень складності та фінансових вимог цей метод не підходить для більшості SaaS додатків, особливо тих що тільки починають свій бізнес.

Розглянемо більш доступний метод кастомізації відповідей чат-боту на базі LLM. Для цього необхідно детально розібратися в можливостях для задання тригеру/стимула (prompt) для великої мовної моделі, який також називають промпт-інжинірингом.

Промпту-інжиніринг полягає в складанні вхідних промптів для максимізації якості відповідей, що генеруються моделями штучного інтелекту. Головні складові промпту:

– контекст: надання релевантної інформації або змісту попередніх розмов, щоб допомогти моделі зрозуміти тему;

– інструкції: чіткі директиви, що визначають завдання, тон або стиль. Наприклад, "Напишіть офіційного листа...";

– основний вміст: головна частина промпту, яка містить ключову інформацію або запит;

– обмеження: чіткі межі або керівні принципи, щоб відповідь відповідала певним критеріям. Наприклад, "Підсумуйте цей текст у 100 словах";

– приклади: надання прикладів бажаного результату допомагає направити модель до створення схожих відповідей;

– структура виходу: вказування бажаного формату, наприклад, марковані пункти, абзаци або діалог.

Розглянемо приклад який був розроблений на основі інформації про необхідну інформацію для вступу в ХНУРЄ. Для цього поєднаємо три компоненти промпту. В інструкціях задаємо моделі задачу використовувати інформацію надану в контексті для відповіді на запитання користувача, в контексті опишемо інформацію релевантну до питання (тема отримання інформації розкривається нижче), та саме питання від користувача. Приклад промпту надається в лістингу 3.1, відповідь надається в лістингу 3.2.

Лістинг 3.1 – Промпт що відповідає на питання про вступ

```
Task: Answer to user prompt provided as a prompt, using the information provided in the context.
```

Context: The foreigner adds to the application:

1) the document about the previously acquired educational (educational-qualifying) level (original and legalized copy with translation into Ukrainian notary certified), on the basis of which the admission takes place;

2) the supplement to the document on the previously acquired educational (educational-qualifying) level (original and legalized copy with translation into Ukrainian notary certifired), on the basis of which the admission takes place;

3) a copy of the passport document of the foreigner or a document certifying the identity of the stateless person;

4) a health insurance policy, unless otherwise provided by international treaties of Ukraine;

5) COVID-19 test certificate;

6) 4 photos of size 30 x 40 mm

The documents specified in subparagraphs 1 – 3 must be translated into Ukrainian with a notarized certification.

The documents specified in subparagraphs 1 – 2 must be legalized in the country of issue in a manner officially used in that country for such certification and legalized by the relevant foreign institution of Ukraine, unless otherwise provided by international treaties of Ukraine.

After the successful completion of the Pre-university Language Training foreigner receives a Certificate of the established form.

Pre-university study of foreign citizens for admission is carried out at Center for Organization of Training of Foreign Citizens and Stateless Persons

```
Prompt: Which documents are required for admission to NURE?
```

Лістинг 3.2 – Відповідь мовної моделі на промпт

To apply for admission to NURE (National University of Radioelectronics), foreign applicants need to submit the following documents:

1. Document certifying their previously acquired educational level, along with its supplement. Both the original and legalized copies, translated into Ukrainian and notarized, are required.
2. Copy of the passport or identity document for stateless individuals.
3. Health insurance policy, unless exempted by international treaties of Ukraine.
4. COVID-19 test certificate.
5. Four photos sized 30 x 40 mm.

The documents listed in points 1 and 2 must be translated into Ukrainian with notarized certification. Additionally, documents in points 1 and 2 must be legalized in the issuing country and by the relevant foreign institution in Ukraine, unless otherwise specified by international treaties of Ukraine. Upon successful completion of the Pre-university Language Training, foreign applicants receive a Certificate of the established form.

Як бачимо ChatGPT надав дуже детальну відповідь, що містить всі необхідні деталі для вступу. Отже використовуючи промпт-інжиніринг ми можемо модифікувати поведінку великої мовної моделі, але для успішної інтеграції нам необхідно побудувати інтеграцію між моделлю та базою знань специфічними для певного SaaS додатку. Для цього розробимо архітектуру що дозволяє масштабувати інтеграцію для будь-якого бізнесу.

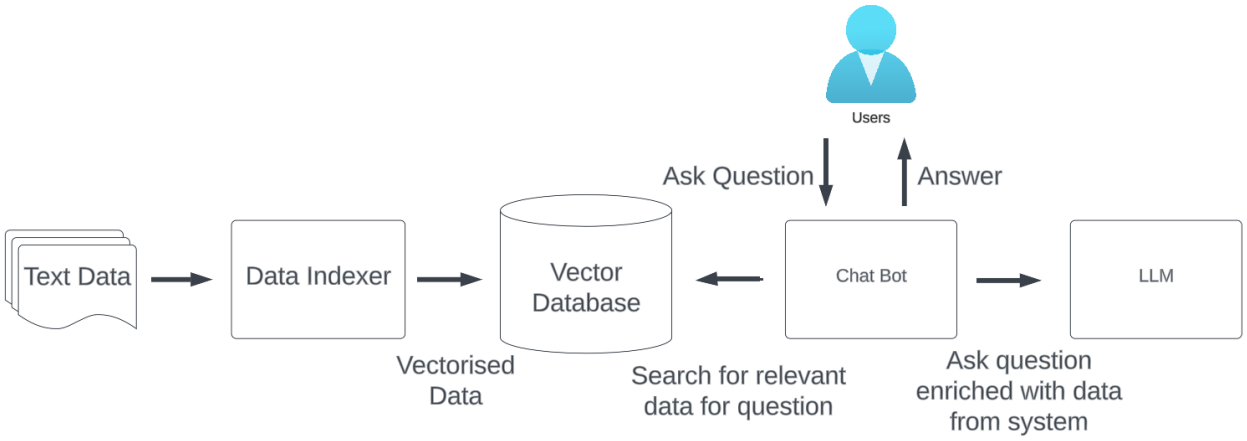

Рисунок 2.6 – Високорівнева діаграма інтеграції чат бота із додатком

Розглянемо компоненти представлені на діаграмі 2.6:

– текстові дані: це джерело інформації, яке буде оброблено та використано для надання відповідей. Сюди входять документи, статті чи будь-які інші текстові дані;

– індексатор даних: цей компонент приймає необроблені текстові дані та перетворює їх у формат, що підходить для ефективно пошуку, та індексує ці данні в векторну базу даних;

– векторна база даних: індексовані дані зберігаються тут у векторизованому форматі. Це дозволяє швидко й ефективно шукати відповідні дані на основі подібності до запиту користувача;

– чат-бот: чат-бот взаємодіє з користувачем. Він приймає запити користувачів і отримує відповідну інформацію за допомогою векторної бази даних, щоб надати точні та контекстуально збагачені відповіді;

– LLM (велика мовна модель): Потужна мовна модель, як-от ChatGPT, використовується для обробки питань користувача та створення відповідей.

Розглянемо послідовність взаємодії між користувачем та чат-ботом:

– користувач ставить питання чат-боту;

– чат-бот шукає відповідну інформацію у векторній базі даних, яка відповідає на запит користувача;

– чат-бот використовує LLM, щоб згенерувати відповідь на питання використовуючи додаткову інформацію з векторної бази даних;

– чат-бот надає користувачеві остаточну відповідь.

Данна архітектура дозволяє чат-боту надавати точні та інформативні відповіді, поєднуючи можливості мовного розуміння LLM із відповідною фактичною інформацією з будь-якої системи завдяки індексуванню даних в векторну базу даних.

Індексація даних відіграє ключову роль, оскільки від якості індексації даних залежить те наскільки ефективно чат-бот зможе шукати релевантну інформацію для пошуку даних, а отже наскільки якісними будуть відповіді чат-

боту для користувачів. Отже розглянемо крок індексації даних представлений на рис. 2.7.

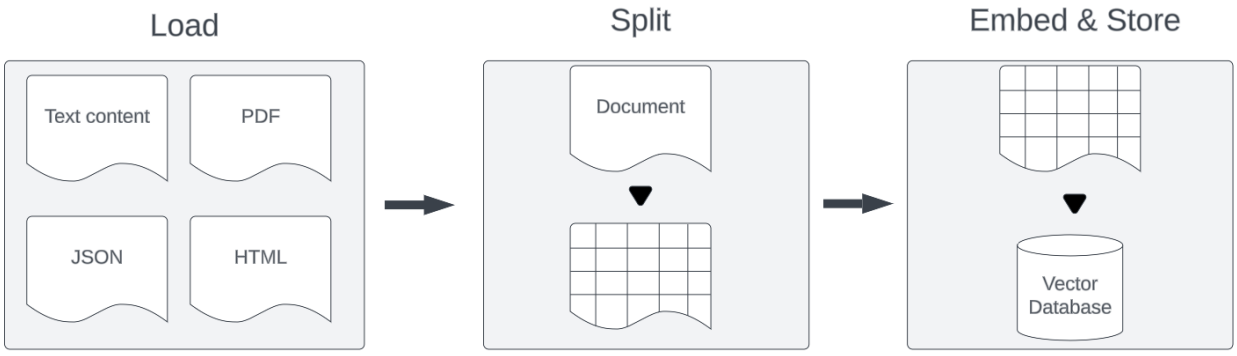

Рисунок 2.7 – Діаграма процесу індексації даних

Процес індексування складається із наступних кроків:

– завантаження**:** система завантажує різні текстові данні які можуть бути видобуті із різних типів файлів наприклад PDF, JSON і HTML, які служать джерелом матеріалу для відповідей;

– розбиття**:** моделі LLM мають обмежену кількість символів, які вони можуть обробляти, тому нам потрібно розбити дані, щоб вони пізніше могли помістится у вікно контексту. Крім того, це дозволяє ефективніше та точніше індексувати дані;

– векторизація та зберігання**:** кожен фрагмент тексту обробляється для створення вектора, числового представлення тексту, яке кодує семантичне значення даних. Ці данні потім зберігаються у векторній базі даних. Векторні дані дозволяють ефективно шукати та отримувати інформацію на основі контекстуальної подібності, а не точного збігу.

Векторизація даних відіграє ключову роль у цьому процесі, оскільки від того наскільки якісне векторне представлення ми створимо на цьому етапі, залежить пошук релевантних даних. Отже розглянемо детальніше як відбувається процес векторизації даних.

Спершу текст розбивається на токени за допомогою Byte-Pair Encoding (BPE) [30], що розбиває слова на під слова або символи, створюючи унікальні

токени. Кожен з цих токенів перетворюється на щільне векторне представлення.

Далі додаються позиційні кодування, щоб надати інформацію про позиції токенів у послідовності, оскільки сама модель Трансформера не враховує порядок токенів. Тому позиційна інформація додається до моделі явно, щоб зберегти інформацію щодо порядку слів у реченні. Ці позиційні кодування додаються до початкових ембедингів, створюючи контекстно-залежні векторні представлення.

Наступним кроком кожен токен потім обробляється через механізм само уваги (self-attention), який обчислює вагові коефіцієнти для кожного токена відносно всіх інших токенів: Само увага дозволяє моделі захоплювати зв'язки між віддаленими елементами в послідовності, що дає змогу розуміти складні шаблони та залежності. Завдяки увазі до різних частин вхідної послідовності, само увага допомагає моделі зрозуміти контекст і призначати відповідні ваги кожному елементу залежно від його релевантності.

Після створення векторного представлення тексту та його збереження в векторній базі даних, ці дані готові для пошуку. Розглянемо як відбувається пошук по векторах.

Пошук по векторах — це процес знаходження схожих або релевантних елементів у великому наборі даних за допомогою порівняння їх векторних представлень. Для пошуку даних спочатку необхідно розрахувати схожість векторів. Найпоширенішими метриками схожості є косинусна схожість [31], що вимірює кут між двома векторами,

$$cosine\ similarity = \frac{\mathrm{A} * B}{||\mathrm{A}||\ * ||B||}$$

а також евклідова відстань [32], що вимірює "пряму" відстань між двома точками в просторі.

$$euclidean\ distance = \sqrt{\sum_{i=i}^{n} (Ai - Bi)^2}$$

Далі відбувається пошук найближчих сусідів за обраною метрикою схожості, за допомогою алгоритмів на кшталт K найближчих сусідів [33], тощо.

2.4 Модель розгортування чат-боту із застосуванням cloud-native технологій

Розгортування чат-боту який міг би масштабуватися із розвитком SaaS додатку, потребує надійної та еластичної інфраструктури. Розглянемо запропоновану модель розгортування на базі хмари AWS, представлену на рисунку 2.8

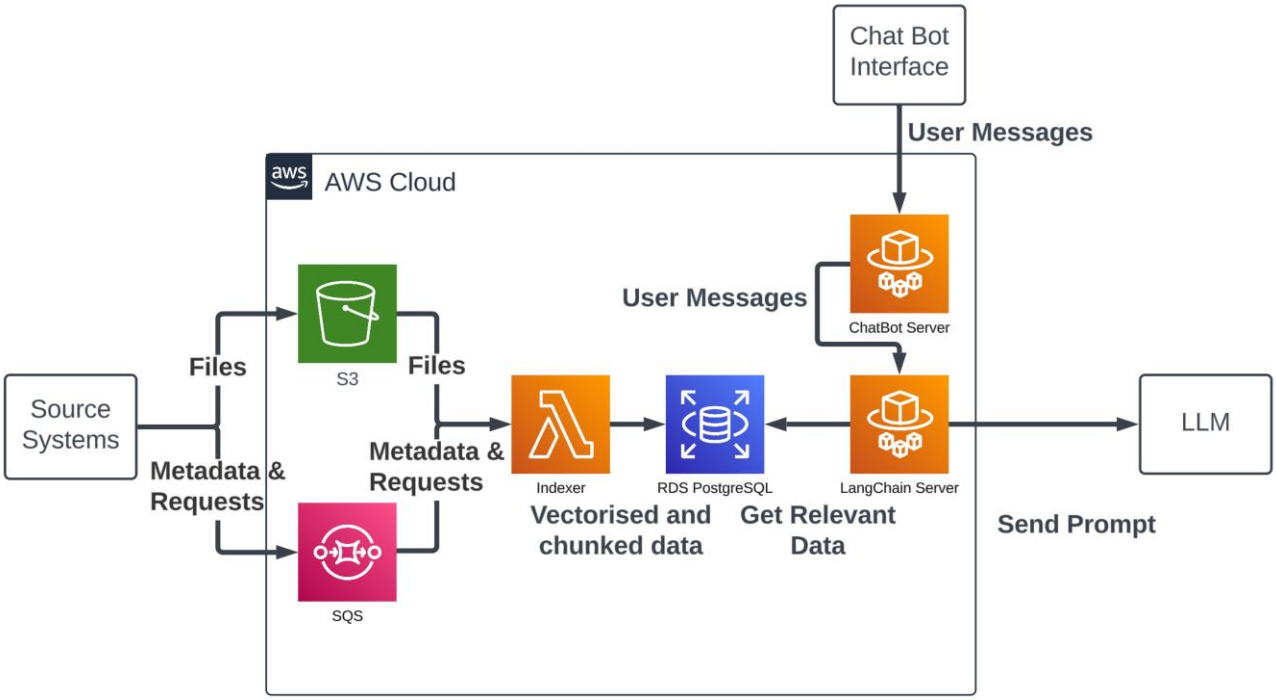

Рисунок 2.8 – Модель розгортання чат-боту із застосування cloud-native технологій

Розглянемо компоненти та їх функції:

– джерела даних (Source Systems): Системи, які слугують джерелами даних необхідних для роботи чат-бота на базі LLM. Дані з цих систем надсилаються до Amazon S3;

– Amazon S3: сервіс зберігання файлів. В даному випадку цей сервіс використовується для зберігання файлів даних з зовнішніх систем. Інтеграція завдяки файлам гарантує масштабуємість процесу індексації даних, оскільки процес індексації може бути асинхронним;

– Amazon SQS: брокер евентів, черга. В даному випадку цей сервіс використовується щоб залишити запит на індексацію файлів й передати необхідні метадані, такі як джерело інформації, приналежність до певного користувача, тощо;

– індексатор: функція , яка обробляє файли завантажені в S3. Функція векторизує данні для ефективного пошуку та розбиває їх на зручні частини, зберігаючи оброблені дані в базі даних RDS PostgreSQL;

– RDS PostgreSQL: керований сервіс реляційної бази даних, який зберігає оброблені векторизовані данні;

– сервер LangChain: сервер, який використовує фреймворк LangChain для отримання відповідних даних з RDS на основі повідомлень користувачів. Він виступає посередником для відправки промптів до великої мовної моделі;

– Сервер чат-бота: сервер, який виступає головним інтерфейсом для додатку чат-бота. Він отримує повідомлення користувачів, зв'язується з сервером LangChain і повертає відповіді від LLM;

– LLM (велика мовна модель): модель, яка обробляє підказки користувачів та надає відповіді природною мовою на основі отриманих даних.

Вибір хмарних сервісів відіграє особливу роль, оскільки саме ці сервіси визначають ключові характеристики вашого додатку, такі як надійність, масштабуємість, ефективність, вартість, тощо. Отже розглянемо технології які ми обрали для розгортання чат-боту на базі LLM для SaaS додатку.

Використання AWS Lambda, S3, RDS, та Fargate має ряд переваг для розробки та експлуатації сучасних додатків, зокрема чат-ботів, що взаємодіють з великими мовними моделями (LLM).

AWS Lambda забезпечує автоматичне масштабування залежно від навантаження, що робить його ідеальним для обробки змінних робочих

навантажень. Оплата здійснюється лише за час виконання функцій, що може значно знизити витрати. Безсерверна (serverless) архітектура знижує операційні витрати та спрощує розгортання додатків.

Amazon S3 надає високонадійне і масштабоване сховище для великих обсягів даних, забезпечуючи високий рівень доступності даних, що важливо для критичних додатків. Він легко інтегрується з Lambda, RDS та іншими сервісами AWS, що спрощує створення комплексних рішень.

Amazon RDS автоматизує завдання адміністрування баз даних, такі як оновлення програмного забезпечення, резервне копіювання та відновлення, що забезпечує кероване середовище. RDS дозволяє легко масштабувати бази даних відповідно до зростаючих потреб додатка, забезпечуючи безпеку даних через вбудовані засоби, такі як шифрування та контроль доступу.

AWS Fargate дозволяє запускати контейнери без необхідності управління серверами, забезпечуючи автоматичне масштабування контейнерів на основі робочого навантаження.

Використання цих сервісів забезпечує зниження операційних витрат завдяки безсерверним технологіям та керованим сервісам, що зменшує витрати на адміністрування та інфраструктуру. Прискорюється час розгортання нових функцій та сервісів завдяки автоматизації багатьох процесів. Ці сервіси забезпечують високу надійність і можуть масштабуватися відповідно до потреб додатка, що особливо важливо для додатків з непередбачуваним навантаженням.

Таким чином, використання AWS Lambda, S3, RDS та Fargate забезпечує надійну, гнучку та економічно ефективну інфраструктуру для побудови чат-боту на базі LLM для SaaS додатків.

3 РОЗРОБКА МЕТОДІВ КАСТОМІЗАЦІЇ SAAS ДОДАТКІВ

3.1 Аналіз та вибір методів кастомізацій

Щоб розробити комплексну стратегію налаштування та визначити відповідні методи модифікації додатків SaaS, важливо визначити конкретні вимоги та бізнес-потреби, які можуть спричинити необхідність модифікації. Ці потреби можуть значно відрізнятися та випливати з різних факторів, таких як уподобання користувачів і специфічні бізнес-вимоги. Деякі основні потреби, які можуть виникнути, включають:

– адаптація програми SaaS до унікальних вимог і операційних процедур бізнесу, може знадобитися індивідуальне налаштування. Це може передбачати зміну робочих процесів, структур даних і автоматизацію завдань відповідно до конкретних потреб компанії;

– існуючі системи, інструменти або служби, що використовуються компаніями, можуть потребувати інтеграції з додатком SaaS;

– підприємства можуть прагнути покращити робочі процеси, потенційно вимагаючи налаштування програм для оптимізації завдань і підвищення ефективності;

– користувачі можуть мати різноманітні вподобання та вимоги, які спонукають до коригування інтерфейсу, функціональності чи налаштувань відповідно до їхніх індивідуальних потреб і стандартів.

Створюючи настроювану програму SaaS, ми твердо переконані, що прийняття підходу API-First є важливим. Цей підхід передбачає надання пріоритету розробці та реалізації API (інтерфейсу програмування додатків) перед іншими компонентами програми. Дотримуючись цієї методології, ми можемо отримати численні переваги, особливо в сфері продуктів SaaS. Однак у конкретному контексті налаштування додатків SaaS наш акцент на розробці API служить для оптимізації внутрішньої інтеграції між різними компонентами системи. Водночас це спрощує процес пропозиції зовнішньої інтеграції для зовнішніх служб і розробників.

Реалізація API в програмі SaaS пропонує користувачам безліч варіантів налаштування, наприклад:

– використовуючи API, користувачі можуть легко інтегрувати програму SaaS зі своїми власними даними чи іншими системами;

– API дозволяє користувачам включати власну бізнес-логіку або автоматизувати процеси в рамках робочих процесів програми SaaS;

– крім того, користувачі можуть використовувати API для створення персоналізованих звітів і аналітичних інструментів, використовуючи дані, надані програмою SaaS.

При розробці загальнодоступного API вкрай важливо забезпечити його сумісність з будь-якою мовою програмування чи технологічним стеком, який використовують користувачі. Це гарантує бездоганну інтеграцію для кожного користувача програми SaaS. Таким чином, REST API є оптимальним вибором для впровадження загальнодоступного API, оскільки він забезпечує необхідну гнучкість і універсальність:

– REST [28] (Representational State Transfer) — це простий і зрозумілий стандарт, який відповідає стандарту ресурсів, що дозволяє представляти та модифікувати ресурси за допомогою URI. Цей стандарт використовує загальновідомі методи HTTP (GET, POST, PUT, DELETE) для полегшення зв’язку;

– REST, широко визнаний стандарт для взаємодії в Інтернеті, бездоганно включений у більшість сучасних технологій і бібліотек. У результаті його можна легко інтегрувати та використовувати на різних платформах і мовах програмування;

– REST API легко інтегрується в архітектуру веб-додатків, дозволяючи користувачам легко працювати з ним за допомогою стандартних веб-браузерів. Ця доступність забезпечує широке охоплення, обслуговуючи різноманітну аудиторію.

Для ефективної реалізації будь-якої бізнес-логіки в додатку SaaS надзвичайно важливо вирішити проблему відстеження подій і реагування на них разом із API. Ця вимога випливає з фундаментальної необхідності бути в курсі подій, що відбуваються в програмі. Однак через різноманітність рішень, які пропонують різні постачальники хмарних послуг, неможливо покладатися на зміни інфраструктури. Таким чином, необхідно знайти рішення, яке дозволяє отримувати події від системи SaaS без прив’язки до певної публічної хмари, брокера подій або інфраструктури.

Після ретельного розгляду доступних варіантів ми визначили, що механізм webhook [29] є оптимальним вибором для підписки на події в програмі SaaS. Веб-хуки служать засобом для системи, часто веб-сервера, для автономної передачі даних або повідомлень іншій системі після настання певної події. У системах, які підтримують веб-перехоплення, коли відбувається певна подія, HTTP-запит, як правило, у формі POST-запиту, надсилається на заздалегідь визначену URL-адресу іншої системи. Ключові характеристики веб-хуків:

– системи можуть бути розроблені для асинхронної взаємодії, усуваючи потребу в постійних запитах на оновлення. Натомість система отримає повідомлення лише тоді, коли відбудеться певна подія;

– вебхук служить для реагування на певні події в системі. Його можна використовувати для надсилання сповіщень про різні події, такі як додавання нового користувача, успішна обробка платежу або оновлення даних;

– оптимізований процес інтеграції в різних системах, увімкнувши пряму передачу даних між серверами, викликану певними подіями;

– допомагають отримувати дані майже в реальному часі, що особливо важливо для систем, які вимагають негайного реагування на події;

– реалізація цієї функції можлива в будь-якій мові програмування або стеку технологій, будь то для сервера чи клієнта.

Щоб персоналізувати інтерфейс, дуже важливо встановити метод, який дозволяє використовувати будь-яку структуру для налаштування. Серед різноманітних можливостей, які пропонують веб-браузери, існує єдиний інструмент, який відповідає цим вимогам: Inline Frame, який зазвичай називають iframe. Цей HTML-тег дозволяє вбудовувати інший HTML-документ у поточний документ, служачи засобом для створення вкладених вікон (фреймів), які можуть розміщувати вміст іншого ресурсу або сторінки.

Щоб налаштувати інтерфейс, рекомендується використовувати вбудовані веб-сторінки. Це дає змогу користувачам персоналізувати інтерфейс програми, використовуючи будь-який фреймворк на свій вибір. Щоб досягти цього, їм просто потрібно розмістити веб-сторінку та підключити її до iframe.

Для реалізації цього підходу можна поєднати його з підходом API-first. Таким чином, веб-сторінка, яка завантажується в iframe, може використовувати загальнодоступний API програми SaaS для виконання різних бізнес-функцій. Однак для того, щоб цей підхід був успішним, веб-сторінка в iframe повинна

спочатку пройти автентифікацію за допомогою веб-додатку SaaS і отримати необхідну інформацію про конфігурацію. Щоб вирішити цю проблему, рекомендується використовувати стандартний API веб-браузера, який забезпечує зв’язок між головною веб-сторінкою та вбудованою веб-сторінкою. Підписавшись на події в об’єкті window, веб-сторінка всередині iframe може прослуховувати повідомлення, надіслані головною сторінкою, і надсилати повідомлення назад на головну сторінку за допомогою методу postMessage. Так само головна сторінка також може виконувати подібні дії щодо вбудованої сторінки в iframe. У результаті, коли сторінка завантажується, стає можливим ініціалізувати та передати необхідні дані конфігурації та авторизації на вбудовану сторінку.

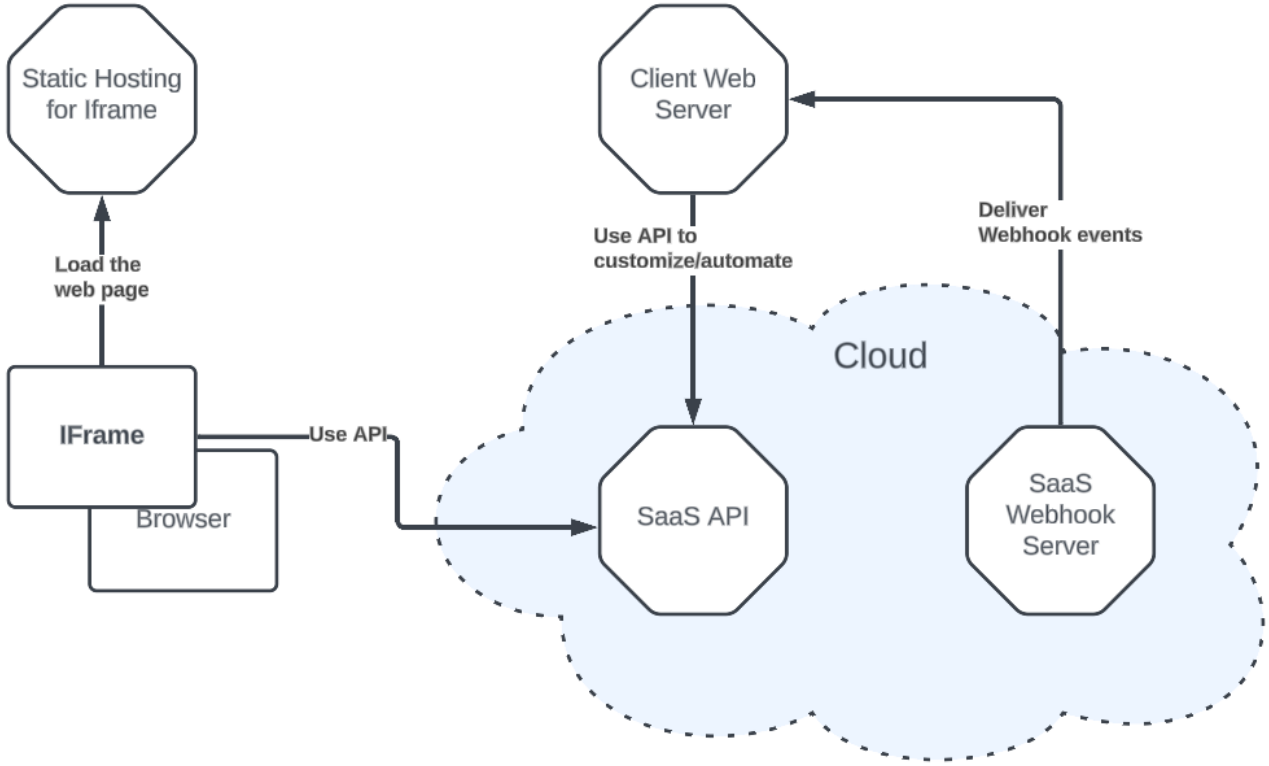

Рисунок 3.1 – Методи налаштування програми SaaS

На рис. 3.1 представлена комплексна архітектура високого рівня, яка ілюструє три різні підходи до налаштування додатків SaaS. Ця архітектура містить такі ключові елементи:

– статичний хостинг для iframe – це тип хостингу, який містить основні статичні файли, необхідні для функціонування веб-сторінки. Потім ці файли завантажуються в iframe, щоб персоналізувати інтерфейс;

– веб-сервер клієнта служить серверною службою, яка містить бізнес-логіку, бажану клієнтом програми SaaS. Цей веб-сервер отримує повідомлення webhook від серверів доставки SaaS Webhook і генерує відповіді на різні події. Одночасно веб-сервер використовує SaaS API для виконання бажаної бізнес-логіки;

– сервери доставки SaaS Webhook, які обробляють доставку повідомлень до вебхуків;

– сервери SaaS API, які відповідають за реалізацію загальнодоступного API програми.

3.2 Розробка надійного й масштабуємого методу доставки вебхуків із застосуванням cloud-native технологій

В рамках кваліфікаційної роботи розглянемо детальніше складнощі які можуть виникнути при реалізації механізму вебхуків:

– безпека. Вебхуки можуть бути вразливими до несанкціонованого доступу, якщо вони не належним чином захищені. Це може призвести до витоку конфіденційної інформації або несанкціонованого втручання в роботу системи. Крім того, без належної валідації дані вебхуків можуть бути перехоплені і змінені, що може порушити роботу системи.;

– надійність. Вебхуки можуть не доставлятися через мережеві проблеми, простої серверів або помилки на стороні клієнта. Важливо забезпечити, щоб повторні спроби доставки не призводили до дублювання обробки;

– масштабованість. Опрацювання великого обсягу подій вебхуків може створювати навантаження на систему, особливо під час пікових навантажень;

– продуктивність. Забезпечення низької затримки у доставці вебхуків для надання своєчасних оновлень. Ефективне управління ресурсами для обробки вебхуків;

Отже необхідно розробити метод доставки вебхуків із застосуванням cloud-native технологій який би міг масштабуватися із розвитком бізнесу та вирішував проблеми описані вище.

3.2.1 Вирішення проблем безпеки

Розглянемо запропоновані механізми для вирішення проблем безпеки які можуть виникнути під час реалізації механізму доставки вебхуків в SaaS додатку. Щоб забезпечити безпеку відправки вебхуків, необхідно вжити кількох ключових заходів, які включають шифрування, аутентифікацію, моніторинг. Всі вебхуки повинні передаватися через HTTPS для шифрування даних під час передачі. Це забезпечує конфіденційність і захист від перехоплення.

Окрім цього необхідно забезпечити механізм аутентифікації, що дозволив би контролювати доступ до клієнту вебхука. Для цього розглянемо два потенційних механізмів реалізації аутентифікації, а саме HMAC (Hash-based Message Authentication Code) та API ключ.

HMAC (Hash-based Message Authentication Code) – це криптографічний метод, який використовується для перевірки цілісності та автентичності повідомлення. Він поєднує в собі хеш-функцію з секретним криптографічним ключем. Основна мета HMAC – забезпечити, що повідомлення не було змінено під час передачі та походить від автентифікованого джерела. HMAC використовує два основні вхідні параметри – секретний ключ і повідомлення (корисне навантаження). Секретний ключ об'єднується з повідомленням, і отриманий результат хешується за допомогою криптографічної хеш-функції (наприклад, SHA-256). Отриманий хеш є HMAC підписом, додається до заголовка HTTP запиту.

Таблиця 3.1 – Порівняння HMAC відносно API ключів

	HMAC	API ключ
Захист від підробки даних	Підпис забезпечує перевірку цілісності даних, що дозволяє отримувачу перевірити, що дані не були змінені під час передачі.	Не забезпечують перевірки цілісності даних, тому можуть бути вразливими до атак типу "людина посередині" (MITM).
Аутентифікація Джерела	Дозволяє впевнено визначити, що повідомлення надійшло від автентифікованого джерела, яке володіє секретним ключем.	Надають тільки базову аутентифікацію і не забезпечують перевірку цілісності або автентичності даних.
Легкість впровадження	Широко підтримується багатьма мовами програмування і бібліотеками, легко інтегрується у системи.	Також легко впроваджуються, але можуть вимагати додаткових заходів безпеки для захисту ключів.
Ризик Компрометації	Оскільки підпис генерується на основі секретного ключа, ризик компрометації даних значно знижується.	Якщо ключ скомпрометований, зловмисник може використовувати його для доступу до системи або даних.

Отже для реалізації безпечної доставки веб хуків рекомендуємо до застосування HMAC підпис.

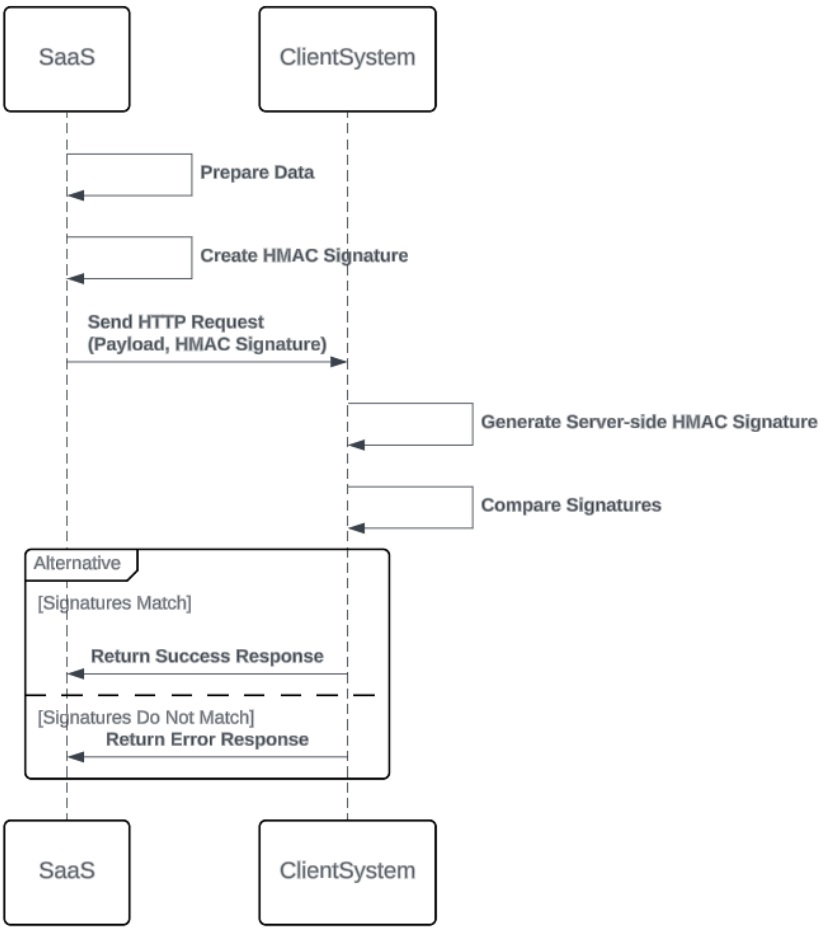

Рисунок 3.2 – Діаграма послідовності для HMAC авторизації

3.2.2 Вирішення проблем надійності

Розглянемо запропоновані механізми для вирішення проблем надійності які можуть виникнути під час реалізації механізму доставки вебхуків в SaaS додатку.

Реалізація надійної доставки вебхука передбачає вжиття заходів для забезпечення того, що повідомлення доставляються успішно навіть у випадку помилок, збоїв або тимчасових проблем з мережею.

Першим кроком для реалізації надійної доставки є реалізація механізму повторних спроб для доставки вебхука. Прямолінійний метод повторних спроб, який ініціює доставку одразу як тільки відбулась помилка може спричинити перенавантаження й викликати ще більше помилок та проблем для системи-клієнта куди відбувається доставка повідомлення. Для вирішення цієї проблеми необхідно застосувати стратегію експоненційних повторних спроб із тремтінням (exponential backoff with jitter),

$$retryAfter = baseDelay * (2 ** retryCount) + jitter$$

Використання такої стратегії має кілька ключових переваг. По-перше, він знижує ризик одночасних спроб, розподіляючи їх у часі, що запобігає синхронізованим повторним спробам. По-друге, зменшує конкуренцію за ресурси, знижуючи конкурентні навантаження. По-третє, підвищує стабільність системи за рахунок рівномірного розподілу навантаження. По-четверте, оптимізує використання серверних ресурсів, забезпечуючи ефективне їх використання. І нарешті, підвищує ефективність доставки, уникаючи пікових навантажень і зменшуючи затримки у доставці.

Але повторні спроби доставки вебхука можуть призвести до того, що одне й те саме повідомлення буде оброблено кілька разів, що може призвести до дублювання дій, таких як створення однакових записів у базі даних або повторне виконання тієї ж транзакції. Отже необхідно реалізувати механізм ідемпотентності, для цього під час створення події для відправки вебхука необхідно створювати ключ ідемпотентності, а саме унікальний ідентифікатор

для цієї події. Для генерації унікального ідентифікатора рекомендуємо використовувати UUID, оскільки він гарантує високий рівень унікальності й простий в використанні. Цей ідентифікатор буде використаний клієнтом під час обробки події, щоб запам’ятати що ця подія вже оброблена й у випадку якщо ця ж подія буде доставлена двічі – не опрацьовувати подію другий раз.

3.2.3 Вирішення проблем масштабування та продуктивності

Розглянемо запропоновані механізми для вирішення проблем масштабування та продуктивності які можуть виникнути під час реалізації механізму доставки вебхуків в SaaS додатку.

Першим кроком для масштабування системи доставки вебхуків необхідно реалізувати асинхронну відправку відправлення повідомлення. Це також зробить систему які реалізує механізм вебхуків більш надійною, оскільки відправка вебхуків не буде блокувати завершення транзакції, тощо. Для реалізації асинхронної відправки необхідно реалізувати чергу повідомлень (наприклад, RabbitMQ, AWS SQS, Kafka тощо.), таким чином ми розділяємо створення подій від відправки.

Наступним кроком для масштабування системи та вирішення проблем продуктивності є використання паралельного процесінгу та горизонтального масштабування серверів які відповідальні за відправку подій вебхуків до клієнтів. Таким чином ми гарантуємо, що із збільшенням к-тю подій, система зможе масштабуватися разом із попитом.

Окрім цього для покращення продуктивності відправки подій необхідно реалізувати батчинг (batching), оскільки батчинг дозволяє об’єднувати кілька запитів у один, що значно зменшує загальну кількість запитів до сервера. Це знижує навантаження на мережу та сервер. Механізм батчингу можливо реалізувати на рівні брокера черг, наприклад AWS SQS дозволяє отримати одразу декілька повідомлень із черги для відправки.

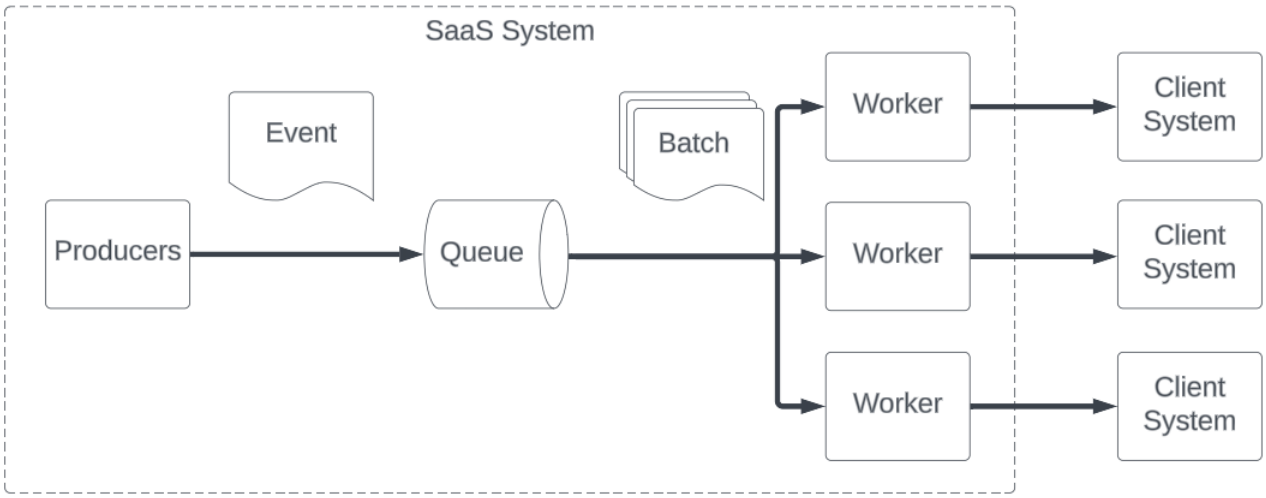

Рисунок 3.3 – Діаграма що зображає підходи для вирішення проблем масштабування та продуктивності доставки вебхук подій

3.2.4 Модель розгортування механізму доставки вебхуків із застосуванням cloud-native технологій

Розглянемо модель розгортування механізму вебхуків на прикладі хмари AWS (Amazon Web Services), представлену на рис. 3.3.

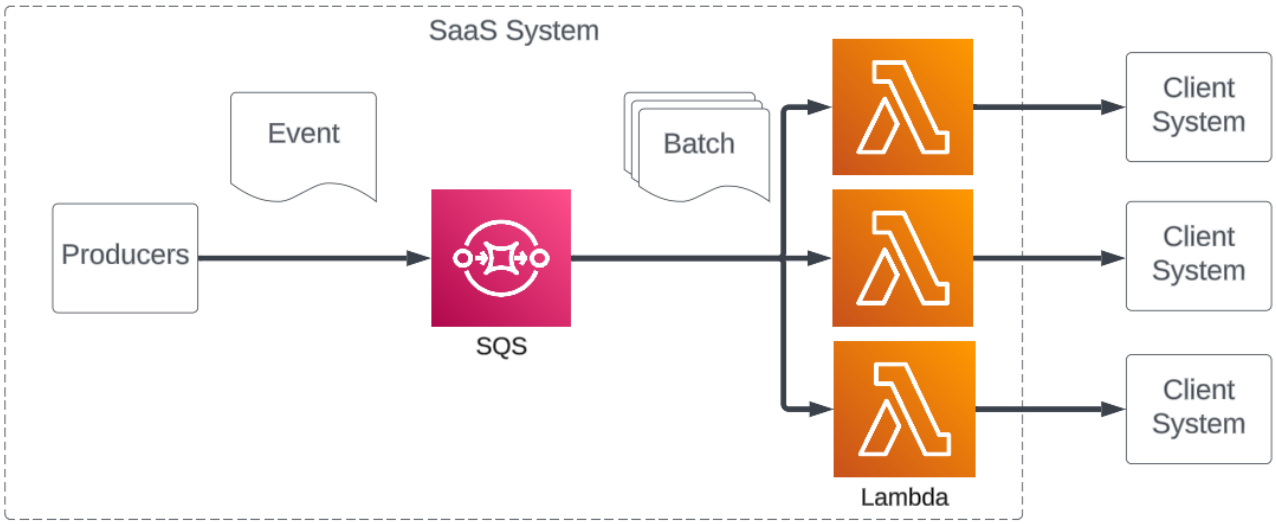

Рисунок 3.3 – Модель розгортування доставки вебхуків із застосування AWS сервісів

Для досягнення масштабування системи, високої продуктивності, а також оптимізації витрати на хмарні обчислення, було обрано два сервіси AWS SQS та AWS Lambda.

Amazon SQS (Simple Queue Service): SQS може обробляти необмежену кількість повідомлень, що робить його високо масштабованим. Забезпечує принаймні одноразову доставку і автоматичне видалення повідомлень після успішної обробки. Відокремлює продуценти і споживачів, дозволяючи їм працювати незалежно, що підвищує стійкість системи. Модель оплати за фактичне використання без попередніх витрат робить його економічно ефективним.

AWS Lambda: Автоматично масштабується зі зростанням обсягу вхідних подій, усуваючи потребу в ручному налаштуванні серверів. Стягує плату лише за спожитий час обчислень, що може значно знизити витрати, особливо для змінних робочих навантажень. Легко інтегрується з SQS, дозволяючи автоматично запускати функції на основі вхідних повідомлень. Забезпечує вбудовану стійкість до збоїв, гарантуючи, що функції будуть високодоступними і стійкими до відмов.

4 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ІЗ ЗАСТОСУВАННЯМ CLOUD-NATIVE ТЕХНОЛОГІЙ

У рамках даного підрозділу описується розробка інформаційної системи «Онлайн ринок літератури», яка слугує прикладом інтеграції чат бота на базі LLM, та методу із доставки вебхуків із реальною інформаційною системою.

4.1 Аналіз вимог до інформаційної системи

Із метою аналізу та формалізації функціональних вимог до інформаційної системи «Онлайн ринок літератури» була розроблена UML діаграма use-case, що представлена на рис. 4.1

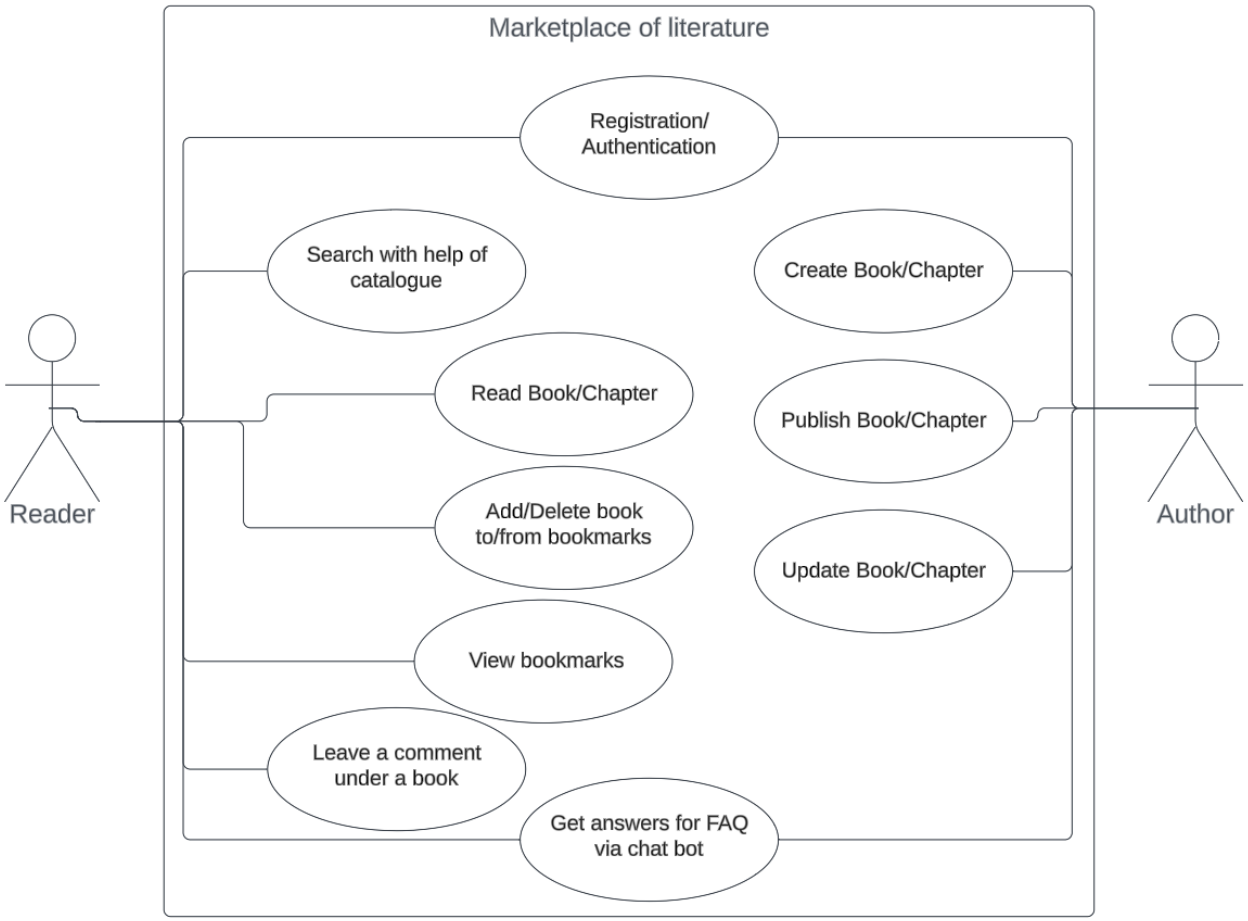

Рисунок 4.1 – Ключові use-cases ІС «Онлайн ринок літератури»

UML діаграма випадків використання представляє систему "Ринок літератури", в якій задіяно два основні актори: Читачі та Автори. Розглянемо детально ключові випадки використання:

– реєстрація/авторизація в ІС «онлайн ринок літератури». Дозволяє як Читачам, так і Авторам реєструватися та проходити аутентифікацію для отримання доступу до інформаційної системи «онлайн ринок літератури»;

– пошук за допомогою каталогу в ІС «онлайн ринок літератури».. Дає змогу Читачам шукати книги за допомогою каталогу із застосуванням фільтрів та текстового пошуку в інформаційні системі «Онлайн ринок літератури»;

– читати книгу/розділ в ІС «онлайн ринок літератури».. Дозволяє Читачам читати вміст обраної книги або розділу опубліковану автором в інформаційній системі «Онлайн ринок літератури»;

– додати/видалити книгу до/з закладок в ІС «онлайн ринок літератури».. Забезпечує функціонал для Читачів додавати або видаляти книги з їхніх закладок для легкого доступу до улюблених книжок в інформаційній системі «Онлайн ринок літератури»;

– переглядати закладки в ІС «онлайн ринок літератури».. Забезпечує легкий доступ до закладок Читачам в інформаційні системі «онлайн ринок літератури»;

– створити книгу/розділ в ІС «онлайн ринок літератури».. Дозволяє Авторам створювати нові книги або розділи до створених книжок в інформаційні системі «Онлайн ринок літератури»;

– публікувати книгу/розділ в ІС «онлайн ринок літератури».. Дозволяє Авторам публікувати створені книги або розділи, щоб вони стали доступні читачам для перегляду в інформаційні системі «онлайн ринок літератури»;

– оновити книгу/розділ в ІС «онлайн ринок літератури».. Забезпечує функціонал для Авторів оновлювати вміст своїх існуючих книг або розділів в інформаційні системі «онлайн ринок літератури»;

– отримати відповіді на FAQ через чат-бот. Забезпечує сервіс чат-бота для відповідей на часто задаванні питання для Читачів та Авторів в інформаційні системі «онлайн ринок літератури».

4.2 Розробка серверної частини інформаційної системи

4.2.1 Архітектура серверної частини

Архітектура серверу відіграє ключову роль у роботі інформаційної роботи, оскільки він забезпечує функціонування системи. Для забезпечення функціонування та майбутнього масштабування інформаційної системи «онлайн ринок літератури» була розроблена архітектура представлена на рисунку 4.2.

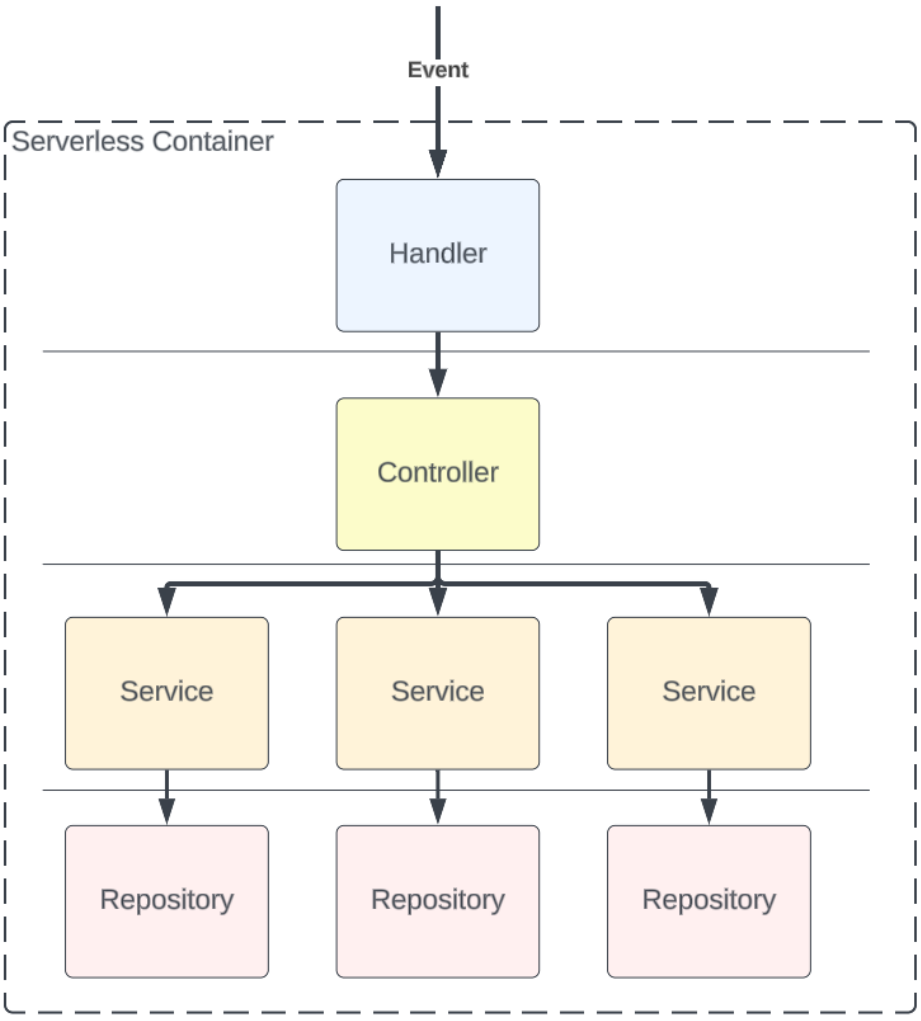

Рисунок 4.2 – Архітектура серверного додатку

Розглянемо детально архітектуру запропоновану на діаграмі 4.2.

Подія виступає тригером для системи. Події можуть бути HTTP-запитами, оновленнями бази даних, завантаженнями файлів тощо, які ініціюють виконання функції. Використання event-driven архітектури забезпечує високу масштабованість, оскільки системи можуть автоматично адаптуватися до змін у навантаженні, обробляючи події асинхронно. Вона підвищує гнучкість, дозволяючи легко додавати або змінювати компоненти без впливу на всю систему. Крім того, така архітектура сприяє ефективному використанню ресурсів, оскільки обчислення здійснюється лише у відповідь на події, що знижує витрати на постійне підтримання серверів.

Обробник є точкою входу для функції інформаційної системи «Онлайн ринок літератури». Він отримує подію, обробляє її і передає контролеру. Обробник є ключовим елементом в цій архітектурі, оскільки він встановлює контекст виконання й виступає ролі абстракції для розробленої системи від специфіки певного провайдера без серверних послуг.

Контролер керує потоком даних між обробником і сервісами інформаційної системи «Онлайн ринок літератури». Він координує логіку додатка, визначаючи, які служби викликати. Діє як компонент приймаючий рішення, спрямовуючи запити до відповідних сервісів.

Сервіс містить бізнес-логіку інформаційної системи «Онлайн ринок літератури». Кожний сервіс обробляє певні операції або завдання, необхідні для додатку. Сервіси є модульними та зосереджуються на конкретній функціональності, що робить систему більш керованою та масштабованою.

Репозиторій управляє зберіганням та отриманням даних. Взаємодіє з базами даних або іншими механізмами зберігання. Забезпечує ефективне зберігання та отримання даних, забезпечуючи абстракцію над фактичним джерелом даних для інформаційної системи «Онлайн ринок літератури».

Розглянемо приклад опрацювання події представлений на діаграмі послідовностей зображеній на рис. 4.3.

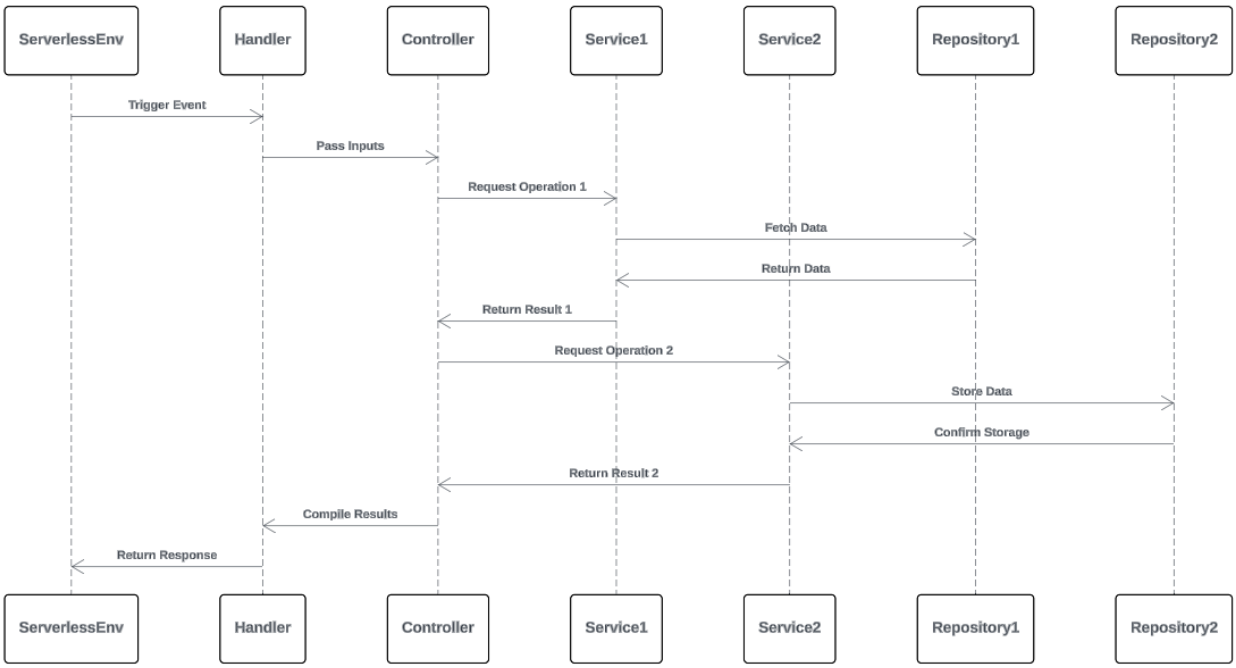

Рисунок 4.3 – Діаграма послідовностей опрацювання події в системі

4.2.2 Модель розгортування із застосуванням cloud-native технологій

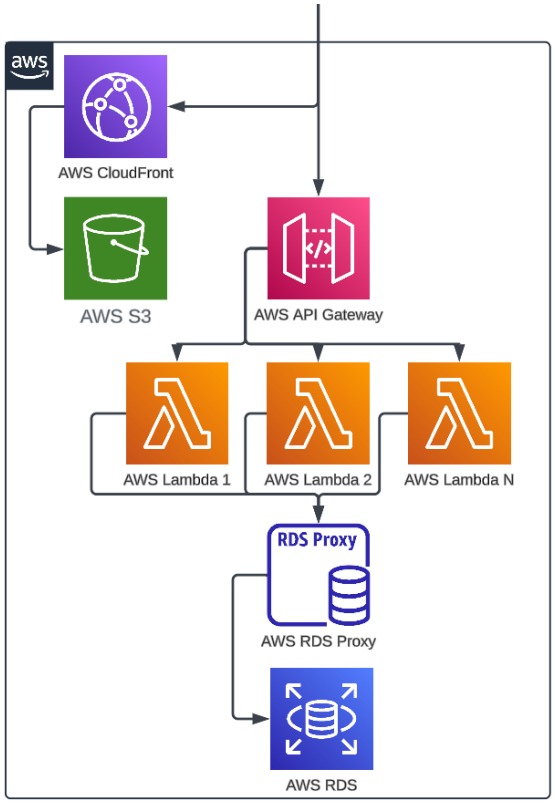

Рисунок 4.4 – Модель розгортування інформаційної системи «Онлайн ринок літератури»

Діаграма зображена на рис. 4.4 ілюструє архітектуру розгортання з використанням сервісів AWS, що включає такі компоненти, як AWS CloudFront, AWS S3, AWS API Gateway, AWS Lambda функції, AWS RDS Proxy та AWS RDS. Розглянемо детальний опис кожного компонента та їх взаємодії.

AWS CloudFront працює як мережа доставки контенту (CDN) для інформаційної системи «Онлайн ринок літератури», розроблена для розподілу контенту до користувачів з низькою затримкою. Вона доставляє статичний контент, збережений в AWS S3, забезпечуючи швидку та ефективну доставку статичного контенту, наприклад зображення.

AWS S3 (Simple Storage Service) забезпечує масштабоване зберігання об'єктів для статичного контенту, такого як зображення, відео та документи, для інформаційної системи «Онлайн ринок літератури». Він зберігає статичні ресурси, які потім доставляються кінцевим користувачам через AWS CloudFront.

AWS API Gateway керує API-запитами і виступає точкою входу для клієнтських додатків для взаємодії з бекенд-сервісами інформаційної системи «Онлайн ринок літераутри». Він направляє вхідні API-запити до відповідних AWS Lambda функцій, виконуючи ключову роль в управлінні API та інтеграції. Також API Gateway пропонує надійні функції для управління API-запитами, включаючи обмеження швидкості, регулювання і механізми безпеки, такі як API-ключі та авторизація. Він спрощує процес створення, розгортання та підтримки API, забезпечуючи централізовану точку для обробки всіх взаємодій і інтеграцій API. Це забезпечує безперебійну комунікацію між клієнтськими додатками та бекенд-сервісами, підвищуючи масштабованість і керованість інфраструктури API в інформаційні системи «Онлайн ринок літератури».

AWS Lambda функції виконують бекенд-логіку інформаційної системи «Онлайн ринок літератури» у без серверному середовищі, автоматично масштабуючись в залежності від кількості запитів. Вони отримують API-запити від AWS API Gateway, обробляють їх і виконують операції, такі як запити до бази даних або інші обробки.

AWS RDS Proxy працює як проксі для підключень до бази даних, підвищуючи ефективність та масштабованість за рахунок спільного використання підключень до бази даних. Він керує підключеннями до бази даних AWS RDS, оптимізуючи продуктивність та використання ресурсів для функцій AWS Lambda.

AWS RDS (Relational Database Service) забезпечує автоматизоване управління завданнями бази даних для інформаційної системи «Онлайн ринок літератури», включаючи резервне копіювання, виправлення і масштабування, зменшуючи операційне навантаження на розробників і адміністраторів. Підтримка декількох рушіїв баз даних і конфігурацій дозволяє гнучко вибирати правильне рішення для конкретних потреб додатків. RDS також пропонує високу доступність і підтримку відмово стійкості з розгортанням Multi-AZ, забезпечуючи надійність і стійкість інфраструктури бази даних. Це робить AWS RDS потужним і зручним варіантом для управління реляційними базами даних в масштабованому та ефективному режимі.

4.3 Розробка інтерфейсу інформаційної системи

UX (User Experience) та UI (User Interface) є критично важливими аспектами при розробці інформаційної системи «Онлайн ринок літератури» з кількох причин.

Забезпечення інтуїтивності та легкості використання є першочерговим завданням при розробці інформаційної системи «Онлайн ринок літератури». Добре спроектований інтерфейс дозволяє користувачам легко знаходити книги, авторів, жанри тощо, що особливо важливо для маркетплейсу з великою кількістю контенту. Інтуїтивна UX забезпечує ефективний пошук та фільтрацію, допомагаючи користувачам швидко знаходити бажану літературу. Добре структурована інформація полегшує розуміння і сприйняття контенту, роблячи досвід користувача більш приємним.

Підвищення задоволення користувачів та залучення є ключовими факторами для успіху інформаційної системи «Онлайн ринок літератури».

Привабливий дизайн UI може створити позитивне перше враження та залучити користувачів до вивчення платформи. Добре продуманий UX може створити емоційний зв'язок з користувачем, підвищуючи лояльність і бажання повертатися на платформу. Позитивний досвід користувачів підвищує ймовірність того, що вони рекомендуватимуть платформу іншим.

Забезпечення доступності є критично важливим для інформаційної системи «Онлайн ринок літератури». Добре розроблений UI повинен бути адаптивним, щоб забезпечити зручність використання на різних пристроях і платформах, включаючи мобільні телефони, планшети та настільні комп'ютери.

Підтримка авторів також є важливим аспектом для успішного просування інформаційної системи «Онлайн ринок літератури». UX, що забезпечує простоту завантаження книг, управління профілями та моніторинг продажів, підтримує авторів та видавців у ефективному використанні платформи.

Підвищення конкурентоспроможності є важливим результатом якісного UX/UI. Унікальний та привабливий дизайн UX/UI може стати ключовою відмінністю від конкурентів, залучаючи більше користувачів на платформу «Онлайн ринок літератури». Платформа з продуманим UX/UI легше адаптується до нових тенденцій та технологій, що дозволяє залишатися конкурентоспроможною на ринку.

ВИСНОВКИ

Метою кваліфікаційної роботи було проведення аналізу та оцінки сучасного стану галузі, розробити архітектуру чат-бота що дозволяє використовувати різні LLM без масштабних змін в системі; розробити метод інтеграції чат бота на базі LLM в SaaS додаток; розробити методи кастомізації SaaS додатків.

У першому розділі проведений детальний аналіз предметної області та стану галузі, визначений вектор наукової роботи та постановка задачі. А саме що до сих пір жодна з робіт не досліджувала використання сучасних LLM, таких як ChatGPT, для розробки чат-ботів для застосунків SaaS. Аналіз літературних джерел довів, що в роботах, присвячених кастомізації SaaS систем розкриваються дуже специфічні методи модифікації які опираються на конкретний набір технологій, що є помилковим й не може масштабуватися на будь-який SaaS

У другому розділі проводиться розробка архітектури чат-боту із використанням LLM, яка дозволяє заміняти великі мовні моделі із легкістю. Більше того проведений порівняльний аналіз трьох мовних моделей із метою визначення найбільш потужної та ефективної мовної моделі для застосування в SaaS додатку. Також запропонований метод інтеграції чат-бота побудованого на основі LLM, без застосування складного та трудомісткого процесу fine-tuning, що дозволяє чат боту отримати доступ до даних із системи, й використовувати цю інформацію для генерації відповідей для клієнта.

У третьому розділі досліджуються методи кастомізації SaaS додатків, які б можливо було використовувати для кастомізації будь-якого SaaS додатку побудованого із використанням будь-якого технологічного стеку. Також була розроблена цілісна стратегія кастомізації SaaS додатків. Окрім цього був

розроблений надійний та масштабуємий метод доставки подій вебхуків із застосування cloud-native технологій.

У четвертому розділі описані ключові рішення під час розробки інформаційної системи «онлайн ринок літератури», що слугує прикладом для демонстрації можливостей методів розроблених в попередніх розділах. Також в розділі розкривається архітектура сервера додатку, та розглядається модель розгортування для додатку із застосування cloud-native технологій.

Майбутній вектор розвитку кваліфікаційної роботи, може бути дослідження методів інтеграції LLM, яка б дозволяла чат боту виконувати бізнес логіку в додатку, а не тільки відповідати на питання. Окрім цього перспективним вектором є розробка SDK для кастомізації інтерфейсу SaaS додатку.

В рамках роботи над кваліфікаційною роботою, підготовили наступні доповіді:

– Selection of Large Language Model for development of Interactive Chat Bot for SaaS Solutions; 8th COLINS 2024 Computational Linguistics and Intelligent System (публікується);

– Analysis and selection of methods for customizing SAAS solutions built using Cloud-Native technologies. INNOVATIVE TECHNOLOGIES AND SCIENTIFIC SOLUTIONS FOR INDUSTRIES, (4(26), 68–77. https://doi.org/10.30837/ITSSI.2023.26.068.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Michael J. Kavis. Architecting the Cloud Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS): Wiley, 2014. 229 с.

2. Tomas Erl, Ricardo Puttini, Zaigham Mahmood. Cloud Computing, Concepts, Technology & Architecture: Pearson, 2013. 747 с.

3. MACH Allience. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/MACH_Alliance (дата звернення: 27.11.2023)

4. Composable Commerce. [Електронний ресурс] – Режим доступу до ресурсу: https://www.elasticpath.com/composable-commerce (дата звернення: 27.11.2023)

5. Eng Lieh Ouh, Benjamin Kok Siew Gan (2023). “An Exploratory Study of Architectural Style and Effort Estimation for Multi-Tenant Microservices-Based Software as a Service (SaaS)”. 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C). https://doi.org/10.1109/ICSA-C57050.2023.00043

6. Wei-Tek Tsai, Peide Zhong (2014). “Multi-tenancy and Sub-tenancy Architecture in Software-as-a-Service (SaaS)”. 2014 IEEE 8th International Symposium on Service Oriented System Engineering. DOI: https://doi.org/10.1109/SOSE.2014.20

7. “Worldwide market share of leading cloud infrastructure service providers”, [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (дата зверення: 29.11.2023)

8. Roggeveen, A. L., & Sethuraman, R. (2020). Customer-interfacing retail technologies in 2020 & beyond: An integrative framework and research directions. Journal of Retailing, 96(3), 299-309. https://doi.org/10.1016/j.jretai.2020.08.001

9. Bouchra EL Bakkouri, Smaira Raki, Touhfa Lalla Belgnaoui, (2022). “The Role of Chatbots in Ecnhancing Customer Experience: Literature Review”. Procedia Computer Science, 203(1), 432-437. http://dx.doi.org/10.1016/j.procs.2022.07.057

10. Adiwardana, D., Luong, M. T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., ... & Le, Q. V. (2020). "Towards a human-like open-domain chatbot." https://doi.org/10.48550/arXiv.2001.09977.

11. G. Daniel, J. Cabot, L. Deruelle and M. Derras, "Xatkit: A Multimodal Low-Code Chatbot Development Framework," in IEEE Access, vol. 8, pp. 15332-15346, 2020, https://doi.org/10.1109/access.2020.2966919

12. G. Attigeri, A. Agrawal and S. Kolekar, "Advanced NLP models for Technical University Information Chatbots: Development and Comparative Analysis," in IEEE Access. https://doi.org/10.1109/access.2024.3368382.

13. G. Mao, J. Su, S. Yu and D. Luo, "Multi-Turn Response Selection for Chatbots With Hierarchical Aggregation Network of Multi-Representation," in IEEE Access, vol. 7, pp. 111736-111745, 2019, https://doi.org/10.1109/access.2019.2934149.

14. Q. Lu, Y. Luo, L. Zhu, M. Tang, X. Xu and J. Whittle, "Developing Responsible Chatbots for Financial Services: A Pattern-Oriented Responsible Artificial Intelligence Engineering Approach," in IEEE Intelligent Systems, vol. 38, no. 6, pp. 42-51, Nov.-Dec. 2023. https://doi.org/10.1109/mis.2023.3320437.

15. M. A. K. Raiaan et al., "A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges," in IEEE Access, vol. 12, pp. 26839-26874, 2024, https://doi.org/10.1109/access.2024.3365742.

16. Hui Song, Franck Chauvel, Arnor Solberg, Bent Foyn, Tony Yates (2017), “How to support customization on SaaS: A Grounded Theory from Customisation Consultants”. 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). DOI: https://doi.org/10.1109/ICSE-C.2017.136

17. Ralph Mietzner, Andreas Metzger, Frank Leymann, Klaus Pohl (2009), “Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications”. 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. DOI: https://doi.org/10.1109/PESOS.2009.5068815

18. Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, Huis Su (2008), “Software as a Service: Configuration and Customization Perspectives”. 2008 IEEE Congress on Services Part II (services-2 2008). DOI: https://doi.org/10.1109/SERVICES-2.2008.29

19. Franck Chauvel, Arnor Solberg (2018), “Using Intrusive Microservices to Enable Deep Customization of Multi-tenant SaaS “. 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC). DOI: https://doi.org/10.1109/QUATIC.2018.00015

20. Hui Song, Phu H. Nguyen, Frank Chauvel, Jens Glattetre, Thomas Schjerpen (2019). “Customizing Multi-Tenant SaaS by Microservices: A Reference Architecture”. 2019 IEEE International Conference on Web Services (ICWS). DOI: https://doi.org/10.1109/ICWS.2019.00081

21. Espen Tonnessen Nordli, Phu H. Nguyen, Franck Chauvel, Hui Song (2020). “Event-Based Customization of Multi-tenant SaaS Using Microservices”. 2020 International Conference on Coordination Languages and Models. DOI: https://doi.org/10.1007/978-3-030-50029-0_11

22. Hunter, I. T. (n.d.). Distributed Node. Js: Building Enterprise-Ready Backend Services. O’Reilly Media.

23. "Node.js – Introduction to Node.js", [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення: 25.02.2024)

24. "Introduction | Langchain", [Електронний ресурс] – Режим доступу до ресурсу: https://python.langchain.com/docs/get_started/introduction (дата звернення: 25.02.2024)

25. "ChatGPT", [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/chatgpt (дата звернення: 25.02.2024)

26. "Cohere Chat", [Електронний ресурс] – Режим доступу до ресурсу: https://cohere.com/chat (дата звернення: 25.02.2024)

27. "Llama API", [Електронний ресурс] – Режим доступу до ресурсу: https://www.llama-api.com/ (дата звернення: 25.02.2024)

28. Leonard Richardson, Mike Amundsen, Sam Ruby. RESTful Web APIs: O'Reilly Media, Inc. 2013, 406c.

29. Webhook. [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Webhook (дата звернення 29.11.2023)

30. Byte pair encoding. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Byte_pair_encoding (дата звернення 09.04.2024)

31. Cosine similarity. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cosine_similarity (дата звернення 09.04.2024)

32. Euclidean distance. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Euclidean_distance (дата звернення 10.04.2024)

33. K-nearest neighbors algorithm. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (дата звернення 10.04.2024)