Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка автоматизованої системи для обслуговування клієнтів фінансових установ

Виконав: студент 2 курсу, групи ЗАКІТР-23м спеціальності 174 – Автоматизація та комп'ютерно-інтегровані технології, та робототехніка
Максим ЗАВАЛЬНЮК
Ім'я ПРІЗВИЩЕ

Керівник: к.т.н, доцент, доцент кафедри КСУ
степінь, звання, посада
Олег КОВАЛЮК
Ім'я ПРІЗВИЩЕ

«4» грудня 2024 р.

Рецензент: к.т.н., доцент, доцент кафедри АІІТ
степінь, звання, посада
Костянтин ОВЧИННИКОВ
Ім'я ПРІЗВИЩЕ

«4» грудня 2024 р.

Допущено до захисту
Зав. кафедри КСУ
В'ячеслав КОВТУН
«4» грудня 2024р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність – 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо-професійна програма – Інформаційні системи і Інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ

В'ячеслав КОВТУН

"14" жовтня 2024 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
студенту Завальнюку Максиму Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка автоматизованої системи для обслуговування клієнтів фінансових установ
керівник роботи Ковалюк Олег Олександрович, к.т.н., доцент
затверджені наказом ВНТУ від "17" вересня 2024 року №310
2. Термін подання студентом роботи "1" грудня 2024 року
3. Вихідні дані до роботи:
    1. Інформація про клієнтські запити у банківській сфері
    2. Статистичні дані зайнятості фінансових помічників
    3. Інформація про використання штучного інтелекту у сфері бізнесу.
4. Зміст текстової частини: Вступ; Аналіз та обгрунтування проблеми; Аналіз існуючих систем; Розробка автоматизованої системи; Висновки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) 1. Схема архітектури системи. 2. Структури класів та зв'язків. 3. UML діаграма послідовності роботи системи. 4. Результати тестування. 5. Екранні форми програми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв

Дата видачі завдання "14" жовтня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз методів, принципів, підходів. Постановка задачі дослідження	20.10.2024	24.10.2024	
2	Аналіз сучасних технологій вирішення проблеми	25.10.2024	27.10.2024	
2	Проєктування архітектури автоматизованої системи	28.10.2024	01.11.2024	
3	Розробка UML-діаграм системи	02.11.2024	03.11.2024	
4	Програмна реалізація системи	04.11.2024	11.11.2024	
5	Розробка структур окремих підсистеми	12.11.2024	14.11.2024	
6	Тестування програмного забезпечення	15.11.2024	19.11.2024	
7	Оформлення графічних матеріалів	20.11.2024	01.12.2024	
8	Захист МКР	11.12.2024	12.12.2024	

Студент _____ Максим ЗАВАЛЬНЮК.
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ Олег КОВАЛЮК
(підпис) (Ім'я ПРІЗВИЩЕ)

АНОТАЦІЯ

УДК 004.8

Завальнюк М.Є. Розробка автоматизованої системи для обслуговування клієнтів фінансових установ. Магістерська кваліфікаційна робота зі спеціальності 174 – Автоматизація та комп'ютерно-інтегровані технології та робототехніка, освітня програма – Інформаційні системи і Інтернет речей. Вінниця: ВНТУ, 2024 р., 122 с.

На укр. мові. Бібліогр.: 43 назв; рис.: 68.

Метою роботи є підвищення ефективності обслуговування клієнтів фінансових установ шляхом розробки прототипу автоматизованої системи, що дозволить зменшити витрати часу на обробку запитів, покращити точність відповідей та оптимізувати роботу завдяки використанню мовної моделі.

У магістерській кваліфікаційній роботі розроблено автоматизовану систему для обслуговування клієнтів фінансових установ на базі чат-бота. У оглядово-аналітичній частині роботи проаналізовано сучасні тенденції та проблеми в сфері обслуговування клієнтів фінансових установ, обґрунтовано доцільність використання чат-ботів та розглянуто переваги обраних технологій. У теоретично-методичній частині описано архітектуру розробленої системи, її функціональні можливості та алгоритми роботи. У практичній частині детально розглянуто процес розробки та інтеграції компонентів системи, включаючи backend на FastAPI та frontend на Next.js. Наведено результати тестування системи, що підтверджують її ефективність. Виконано розробку програмного забезпечення та наведено результати його тестування.

Ілюстративна частина складається з 9 плакатів із схемами та результатами роботи.

Ключові слова: чат-бот, автоматизована система, обслуговування клієнтів, фінансові установи, Vertex AI, Document AI, FastAPI, Next.js.

ANNOTATION

UDC 004.8

Development of an automated system for customer service of financial institutions. Master's qualification work in the specialty 174 - Automation and computer-integrated technologies and robotics, educational program - Information systems and the Internet of things. Vinnytsia: VNTU, 2024, 115 p.

In Ukrainian. Bibliogr.: 43 titles; figs: 68.

The aim of the work is to improve the efficiency of customer service for financial institutions by developing a prototype automated system that will reduce the time spent on processing requests, improve the accuracy of responses, and optimize work through the use of a language model.

In the master's thesis, an automated system for customer service of financial institutions based on a chatbot was developed. The review and analytical part of the work analyzes current trends and problems in the field of customer service of financial institutions, substantiates the feasibility of using chatbots, and considers the advantages of the selected technologies. The theoretical and methodological part describes the architecture of the developed system, its functionality, and operation algorithms. The practical part describes in detail the process of developing and integrating the system components, including the backend on FastAPI and the frontend on Next.js. The results of the system testing are presented, confirming its effectiveness. The software development is carried out and the results of its testing are presented.

The illustrative part consists of 9 posters with diagrams and work results.

Keywords: chatbot, automated system, customer service, financial institutions, Vertex AI, Document AI, FastAPI, Next.js.

ЗМІСТ

ВСТУП

Сучасний фінансовий сектор характеризується постійним зростанням конкуренції та вимог клієнтів до якості та швидкості обслуговування. В умовах стрімкого розвитку цифрових технологій, фінансові установи стикаються з необхідністю оптимізації бізнес-процесів та пошуку інноваційних рішень для підвищення ефективності взаємодії з клієнтами [2]. Одним з таких рішень є впровадження автоматизованих систем обслуговування на базі чат-ботів, що дозволяє забезпечити цілодобову доступність послуг, персоналізований підхід та суттєве зниження витрат на обслуговування.

Актуальність теми даної магістерської роботи обумовлена саме цими факторами. Розробка автоматизованої системи для обслуговування клієнтів фінансових установ на базі чат-бота дозволить вирішити ряд важливих завдань: підвищити доступність фінансових послуг, скоротити час очікування клієнтів, зменшити навантаження на персонал та покращити якість обслуговування в цілому. Використання передових технологій, таких як хмарне розгортання мовних моделей для розробки та навчання чат-бота, текстові процесори для обробки документів, мікросервіси для побудови серверної частини та односторінкові застосунки для створення сучасного та зручного інтерфейсу користувача, забезпечує високу ефективність та гнучкість розробленої системи [6].

Метою роботи є підвищення ефективності обслуговування клієнтів фінансових установ шляхом розробки прототипу автоматизованої системи. Це дозволить зменшити витрати часу на обробку запитів, покращити точність відповідей та оптимізувати роботу завдяки використанню мовної моделі, навченої на нормативних документах.

Об’єктом дослідження є процес обслуговування клієнтів у фінансових установах.

Предметом дослідження є методи, засоби та апаратна частина для створення хмарної системи для взаємодії з даними клієнтів, їхніми рахунками, тарифами банківських установ.

Завданнями роботи є:

- Створення безпечного з безперебійним доступом сховища даних
- Створення текстового процесора для обробки документів й поєднання його зі сховищем.
- Навчання мовної моделі на базі нормативних документів.
- Створення автоматизованого конвеєру для поєднання усіх сервісів.
- Розробка інтерактивного чат-бота.
- Створення веб-порталу для взаємодії з чат-ботом.

Інноваційність одержаних результатів – підхід розробки інтерактивних форм для електронних повідомлень, використовуючи хмарні середовища для автоматизації процесів надсилання повідомлень. Дане рішення є цілком безкоштовним, простим у застосуванні. Електронні форми можуть бути налаштовані та побудовані у будь якому форматі, який підлягатиме правилам побудови структури інтерактивної форми.

Практична цінність даної роботи полягає у створенні інструменту, що має потенціал для суттєвого покращення ефективності та якості обслуговування клієнтів фінансових установ.

Апробація результатів магістерської кваліфікаційної роботи. Матеріали заслуховувалися на конференції ВНТУ: «LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)» [27].

Публікації результатів магістерської кваліфікаційної роботи. Результати досліджень опубліковані у тезах конференції ВНТУ: «LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024)» [27].

1 АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Причини автоматизації фінансових послуг

Важливим аспектом в сфері бізнес-послуг є підвищення ефективності та швидкості надання послуг різного виду. Для досягнення цього залучуються більше працівників, і при цьому кількість інформації, яку потрібно обробляти, збільшується. Розробка автоматизованої системи обслуговування клієнтів фінансових установ на базі мовної моделі та оптичного розпізнавання даних є необхідністю, продиктованою сучасними реаліями ринку та прагненням до оптимізації бізнес-процесів. Традиційні методи обслуговування, що базуються на роботі операторів, стають менш ефективними та рентабельними. Автоматизація ж дозволяє вирішити одразу декілька ключових проблем [4]. По-перше, вона дає змогу значно скоротити витрати на обслуговування, оптимізуючи роботу персоналу та зменшуючи ризик людського фактору [7]. По-друге, автоматизована система здатна обробляти запити клієнтів цілодобово, що значно підвищує доступність та якість обслуговування. По-третє, використання мовної моделі, що навчається на нормативних документах, гарантує високу точність та актуальність наданої інформації. Впровадження такої системи позитивно впливає на імідж компанії, презентуючи її як інноваційну та клієнтоорієнтовану [5].

Аналіз сучасної тенденції у розвитку мовних моделей та впровадження їх у сфери бізнесу свідчить про активний розвиток високоефективних рішень і використання штучного інтелекту у взаємодії з клієнтами [3].

Використання навченої мовної моделі для автоматизації обслуговування клієнтів фінансових установ відкриває низку переваг, що виводять взаємодію з клієнтами на якісно новий рівень. Чат-бот, будучи мультимодальною моделлю, здатна обробляти не тільки текст, але й інші типи даних, такі як зображення, аудіо та відео [3].

Це означає, що чат-бот зможе:

- Розуміти контекст звернення клієнта ще глибше: Наприклад, клієнт зможе надіслати скріншот з помилкою в мобільному додатку, а чат-бот, проаналізувавши зображення, зможе одразу запропонувати рішення або перенаправити запит відповідному спеціалісту.
- Надавати більш персоналізований та емпатичний досвід: Gemini, завдяки своїм розширеним можливостям обробки природної мови, здатна краще розуміти емоційний стан клієнта та адаптувати стиль спілкування відповідно до ситуації.
- Забезпечити більш інтуїтивну та зручну взаємодію: Клієнти зможуть спілкуватися з системою не тільки за допомогою тексту, але й голосом, що зробить процес більш природнім та комфортним.

Використання OCR(оптичне розпізнавання символів) у сфері обслуговування клієнтів фінансових установ несе в собі значний потенціал для оптимізації та підвищення якості обслуговування [4]. Завдяки можливості розпізнавати та аналізувати текст у відсканованих документах та зображеннях, ця технологія відкриває нові горизонти для автоматизації та покращення роботи з клієнтами.

OCR дає змогу:

- Швидко та точно оцифровувати документи: Клієнти зможуть надсилати документи (паспорти, виписки, договори) в будь-якому форматі (фотографії, скани), а система автоматично розпізнає та витягне всю необхідну інформацію [21].
- Автоматизувати процес обробки заявок та запитів: Система зможе самостійно заповнювати форми, перевіряти дані та класифікувати запити на основі інформації з документів. Це значно

пришвидшить процес обслуговування та звільнить співробітників від рутинної роботи.

- Підвищити точність та надійність обробки даних: Виключення людського фактору при введенні та обробці даних зменшує ризик помилок та гарантує високу точність обслуговування [21].
- Надавати клієнтам більш персоналізований сервіс: Аналізуючи дані з документів, система зможе ідентифікувати потреби клієнта, пропонувати релевантні продукти та послуги, а також оперативно реагувати на індивідуальні запити.

1.2 Аналіз існуючих систем

Оскільки очікування клієнтів у фінансовій галузі постійно зростають, чат-боти на основі штучного інтелекту покращують фінансові послуги та підтримку клієнтів. Чат-боти є корисними для фінансових компаній для вирішення звичайних запитів, таких як запити на платіжні дані та баланси [8].

Це зменшує навантаження на співробітників кол-центру, дозволяючи їм зосередитися на інших складних проектах і стратегіях, які вимагають вищого рівня знань. Для аналізу пропонується аналоги фінансових чатботів.

1.2.1 Tata Mutual Fund

Tata Mutual Fund [1] є найбільш надійним пайовим інвестиційним фондом в Індії і пропонує широкий спектр продуктів для планування фінансів і примноження багатства. Він шукав рішення для міленіалів, які потребують підтримки на XODA. Tata Mutual Fund співпрацював з Haptik, щоб надати їм просту у використанні платформу, яка використовує чат-бота на основі штучного інтелекту, який забезпечує майже ідеальне рішення запитів.

Чат-бот був розроблений для покращення досвіду обслуговування клієнтів (CX) і підвищення залученості клієнтів у популярні канали обміну

повідомленнями.

Haptik запропонував цифрову стратегію за допомогою чат-бота, який обробляв запити від початку до кінця та звільняв агентів, щоб вони могли зосередитися на більш важливих проектах.

Інтелектуальні підказки та потоки контенту, а також безперешкодна інтеграція чат-бота з внутрішньою системою для ефективного поширення інформації призвели до значного покращення взаємодії з клієнтами.

Чат-бот допоміг Tata Mutual Fund досягти 90% наскрізного вирішення запитів і 67% зниження кількості запитів до колл-центру (рис. 1.1).

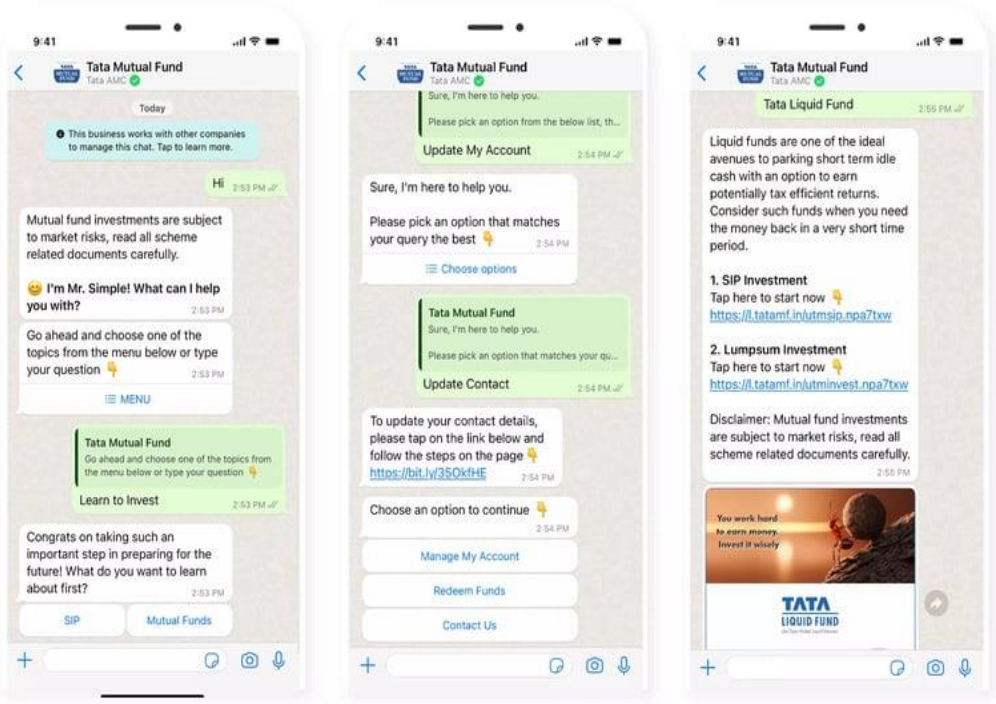


Рисунок 1.1 – Обслуговування клієнтів через Tata Mutual Fund

1.2.2 Kissht (Ring)

Kissht, тепер Ring [1], є провідною фінансовою та технологічною платформою, яка змінила, як клієнти отримують кредит. Метою Ring є надання своїм клієнтам безпроблемного кредитування як в Інтернеті, так і в офлайні. Компанія прагнула надати своїй різноманітній клієнтській базі чудовий користувацький досвід.

У нього було два завдання: створити чат-бота, який міг би покращити обслуговування клієнтів, зокрема, скоротити час першої відповіді, і гарантувати безперебійну інтеграцію. Хаптик допоміг у вирішенні цих завдань з «максимальною ефективністю». (рис. 1.2).

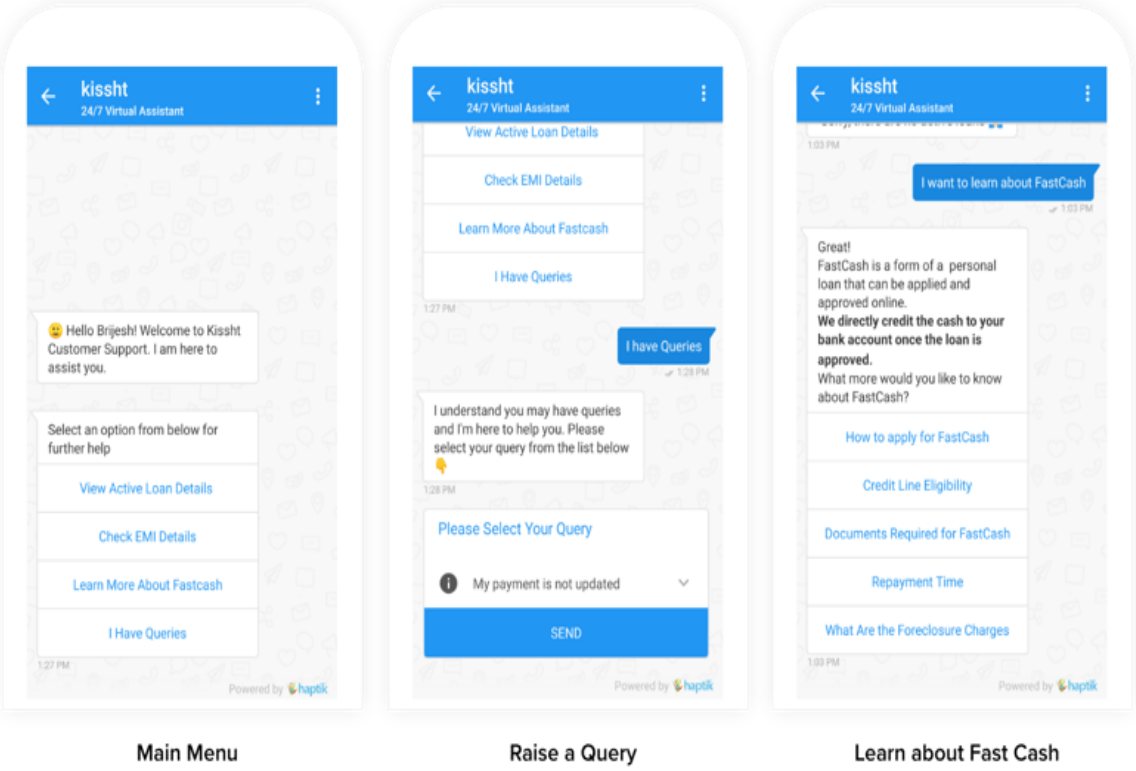


Рисунок 1.2 – Обслуговування клієнтів через Ring

Ring співпрацював з Haptik над створенням розмовної платформи у WhatsApp, яка б підвищила зручність для клієнтів та автоматизувала підтримку.

Розмовна платформа використовує ефективність бота WhatsApp, щоб надати клієнтам миттєвий доступ до інформації, пов'язаної з EMI, та відповісти на їхні нагальні запитання.

Haptik використовував інтерфейс WhatsApp, щоб допомогти поточним користувачам перевіряти активні кредити та статус заявок через чат, тоді як бот допоміг досягти швидшого вирішення запитів та часу відповіді.

Ring досягнув наскрізного показника вирішення запитів на рівні 72%, обробивши 800 000 розмов і досягнувши 84% автоматизації.

1.2.3 CASHe

Мета CASHe [1], ефективної платформи фінансового благополуччя на основі кредитування, полягає в тому, щоб допомогти молодим споживачам середнього класу отримати більше позик. Допомагаючи людям мати більший контроль над своїми грошима, вона має на меті використовувати технології, які забезпечать безпроблемний доступ до кредитів. Крім того, розробники CASHe визнали, що для надання пакетизованих кредитів мільйонам користувачів по всій Індії необхідний автоматизований цифровий канал, такий як WhatsApp.

У партнерстві з Haptik компанія надала своїм клієнтам доступ до миттєвої кредитної лінії без очікування, гарантуючи своєчасне та ефективне реагування на кожну заявку, а також скоротила кількість процедур, пов’язаних із верифікацією, завантаженням документів і автоматизованим андеррайтингом, що прискорило видачу кредитів.

1.2.4 IIFL

IIFL Holdings Limited є диверсифікованою компанією з фінансових послуг в Індії [1], яка обслуговує чотиримільйонних клієнтів у різних сферах бізнесу. Компанія шукала вигідне та ефективне рішення для обробки великої кількості запитів до служби підтримки та покращення загального досвіду клієнтів завдяки швидшому вирішенню запитів.

Компанія створила чат-бота ASK-IIFL (рис. 1.3) у співпраці з Haptik для обробки великої кількості запитів у режимі реального часу, що зменшує потребу втручання людини та зменшує ймовірність помилок.

У результаті Chatbot Haptik підвищив рівень задоволеності клієнтів (CSAT). Крім того, чат-бот також значно скоротив операційні витрати IIFL, надаючи швидкі відповіді на поширені запитання, такі як брокерські калькулятори, баланси та перекази та звіти про торгівлю.

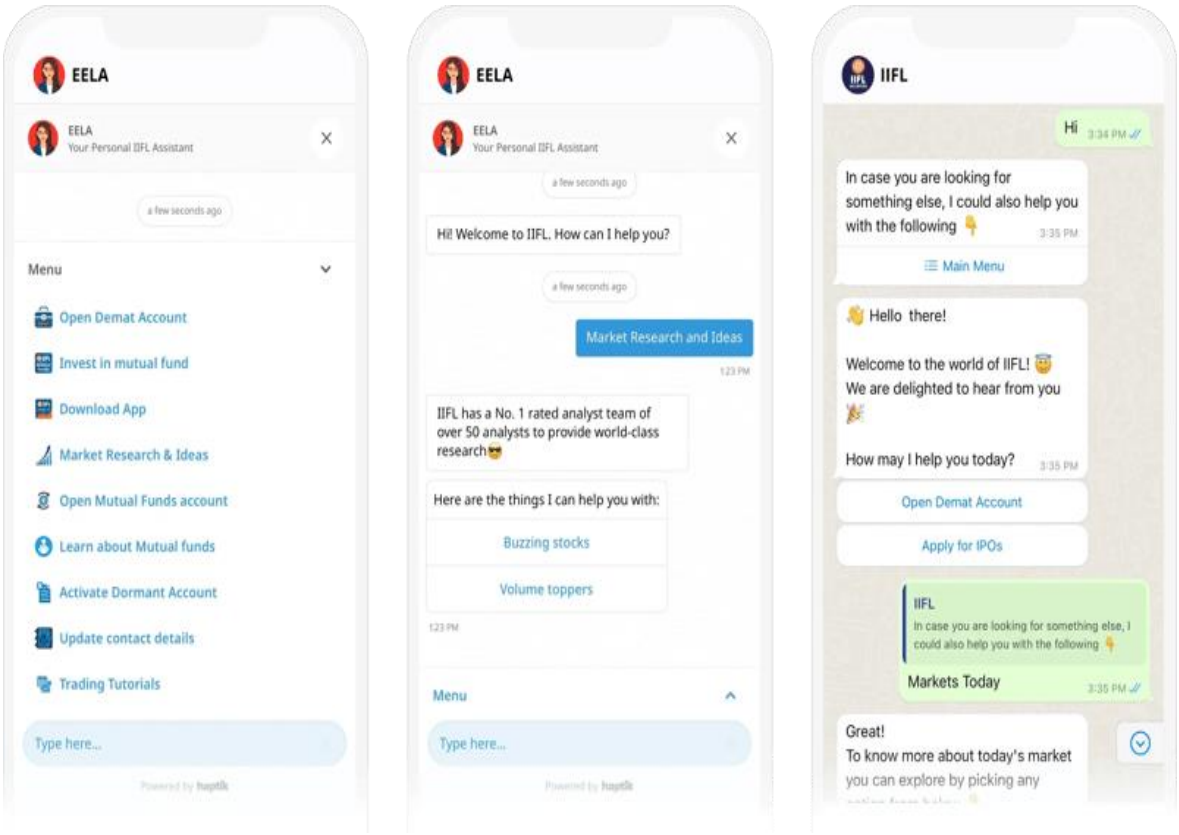


Рисунок 1.3 – Обслуговування клієнтів через IIFL

IIFL обробляв понад 30 000 розмов щомісяця за допомогою чат-бота Haptik, який мав час вирішення 45 секунд, середній бал відгуків користувачів 4 (з 5), і час вирішення 4 секунди.

Існуючі аналоги чат-ботів для фінансових установ часто вирішують лише обмежений набір завдань і не повністю задовольняють потреби сучасних клієнтів. Деякі з них пропонують лише базову інформаційну підтримку, інші ж

обмежені у функціональності та інтеграції з банківськими системами. Розглянуті аналоги не враховують масштабованість бази знань, гнучкість при спілкуванні з клієнтом.

Запропонована в даній роботі автоматизована система на базі чат-бота пропонує комплексне рішення для обслуговування клієнтів фінансових установ. Вона забезпечує персоналізовані рекомендації, можливість здійснювати певні фінансові операції безпосередньо через чат-бот, розширення бази знань через постійне навчання мовної моделі. Завдяки використанню моделі на хмарному середовищі, система здатна розуміти природну мову та навчатися на основі даних, що дозволяє їй постійно покращувати якість обслуговування [10]. Текстовий процесор забезпечує ефективну обробку документів, що спрощує, наприклад, ідентифікацію клієнта, обробку заявок на кредити [11]. Використання мікросервісної архітектури для веб-застосунку гарантує високу продуктивність, надійність та зручність використання системи як для клієнтів, так і для співробітників фінансової установи.

1.3 Мовна модель

Мовні моделі — це основа сучасних систем обробки природної мови

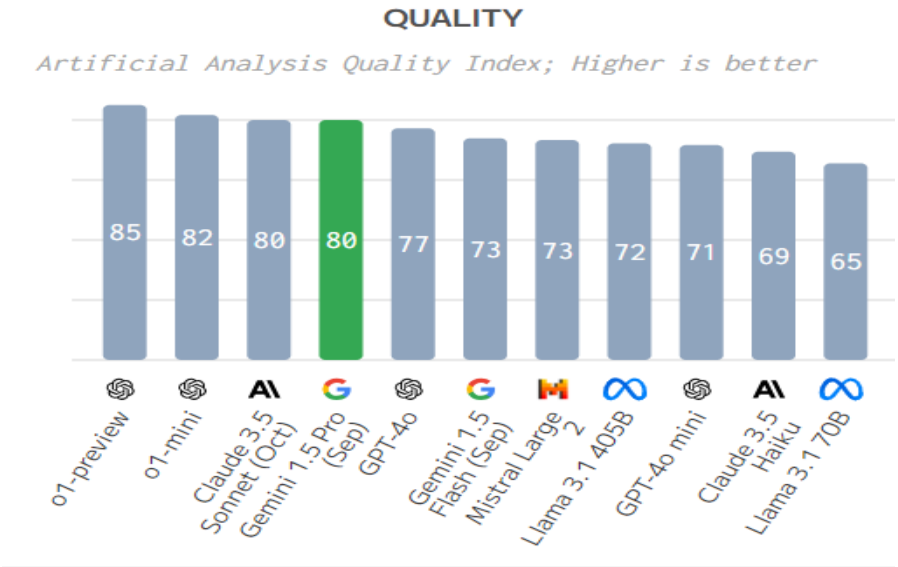


Рисунок 1.4 – Порівняння Gemini 1.5 з іншими моделями

(NLP). Вони являють собою складні алгоритми, навчені на величезних обсягах текстових даних, що дозволяє їм розуміти, генерувати та маніпулювати людською мовою. Розвиток мовних моделей пройшов довгий шлях від простих статистичних моделей до складних нейронних мереж [12].

Gemini 1.5 - це велика мовна модель (LLM), розроблена Google. Дана система застосовується для широкого кола завдань у галузі обробки природної мови, зокрема для розуміння, генерування та перекладу тексту [11]. Gemini 1.5 побудована на основі трансформаторної архітектури, що дозволяє їй ефективно обробляти довгі послідовності тексту та враховувати контекст. І саме це ставить її на високі місця у порівнянні з іншими моделями (рис. 1.4).

Ключові особливості Gemini 1.5:

- Мультимодальність: На відміну від деяких інших LLM, Gemini 1.5 може обробляти не тільки текст, але й інші типи даних, такі як зображення та код. Це відкриває нові можливості для створення більш інтерактивних та інформативних додатків.
- Масштабованість: Gemini 1.5 доступна в різних розмірах, відносно невеликих для мобільних пристроїв до надзвичайно великих для складних завдань. Така гнучкість дозволяє адаптувати рішення до конкретних вимог та обмежень ресурсів.
- Ефективність: Gemini 1.5 оптимізована для високої продуктивності та низького енергоспоживання, що робить її придатною для використання в широкому спектрі додатків.
- Адаптивність: Gemini 1.5 може бути налаштована на конкретні завдання та домени за допомогою методів навчання з підкріпленням та навчання з вчителем. Це дозволяє створювати спеціалізовані моделі, які найкраще відповідають потребам користувачів.
- Розуміння контексту: Gemini 1.5 здатна розуміти контекст та

нюанси мови, що дозволяє їй генерувати більш точні та релевантні відповіді [25].

LLM, такі як Gemini, є найсучаснішими мовними моделями, навченими на масштабних наборах даних [21]. Вони демонструють вражаючі результати в різних завданнях NLP, включаючи генерування тексту, переклад, відповіді на запитання та створення різних видів творчого контенту. Завдяки LLM з'являються нові перспективи розвитку інтелектуальних систем, які можуть спілкуватися з людьми, використовуючи природну мову. Gemini 1.5, розроблена Google, є яскравим прикладом такої моделі, що поєднує в собі переваги трансформерної архітектури та масштабного навчання. Її мультимодальні здібності та гнучкість роблять її ідеальним вибором для створення чат-ботів та інших інтелектуальних додатків.

1.4 Постановка задачі дослідження

Розробка автоматизованої системи для обслуговування клієнтів фінансових установ на основі мовної моделі, навченої на нормативних документах, є комплексною задачею, що вимагає вирішення наступних підзадач:

- Аналіз існуючих рішень та підходів до автоматизації обслуговування клієнтів у фінансовій сфері: важливо дослідити поточний стан ринку, визначити сильні та слабкі сторони існуючих рішень, а також виявити незадоволені потреби клієнтів та фінансових установ. Це дозволить сформувати чітке бачення розроблюваної системи та забезпечити її конкурентні переваги.
- Визначення ключових вимог до розроблюваної системи, з урахуванням специфіки роботи з нормативними документами фінансових установ: система повинна бути адаптована до специфічної термінології, правових норм та бізнес-процесів

фінансової сфери. Важливо забезпечити дотримання всіх вимог безпеки та конфіденційності даних.

- Проектування та розробка архітектури системи, включаючи модулі обробки природної мови, машинного навчання та веб-інтерфейсу: необхідно створити гнучку та масштабовану архітектуру, здатну обробляти великі обсяги даних та запитів від користувачів. Важливо забезпечити інтеграцію різних модулів системи та їх ефективну взаємодію.
- Реалізація функціональності системи з використанням технологій Vertex AI, Document AI, FastAPI та Next.js: використання сучасних технологій гарантує високу продуктивність, надійність та масштабованість системи [9]. Vertex AI дозволить створити ефективну мовну модель, Document AI забезпечить обробку нормативних документів, а FastAPI та Next.js допоможуть створити зручний та інтуїтивно зрозумілий веб-інтерфейс.
- Навчання та налаштування мовної моделі на базі набору нормативних документів фінансових установ: якість роботи системи безпосередньо залежить від якості навченої моделі. Необхідно підібрати релевантний набір даних, налаштувати параметри навчання та оцінити ефективність отриманої моделі.
- Тестування та оцінка ефективності розробленої системи з точки зору точності відповідей, швидкості обробки запитів та зручності використання: перед впровадженням системи необхідно провести комплексне тестування, визначити показники ефективності та усунути можливі недоліки.

1.5 Висновки

У даному розділі були обговорені важливі причини автоматизації надання фінансових послуг. Підвищення продуктивності та ефективності використання штучного інтелекту у вигляді великих мовних моделей є головною запорукою успіху сучасних фінансових компаній. Виходячи з цього, автоматизована система надання послуг на базі хмарних технологій надає безперебійну роботу з мінімальними зусиллями.

Аналіз існуючих систем застосування чат-ботів виявив активний розвиток штучних помічників, зокрема, використання їх вже у повному обсязі й роботі з банківськими рахунками клієнтів. Такі рішення мають велику перевагу у забезпеченні швидкої відповіді на запити клієнтів.

У підсумку, використання сучасних технологій та хмарних технолллогій для контролю створення штучних агентів дозволяє значно покращити якість фінансових послуг, зменшити витрати на персонал та на обробку нормативних документів.

2 ПРОЕКТУВАННЯ ТА ОБГРУНТВУВАННЯ ТЕХНОЛОГІЙ СИСТЕМИ

Зважаючи на обмеження традиційних методів обслуговування клієнтів фінансових установ, таких як телефонні дзвінки, особисті візити до відділень та обробка паперових документів, виникає потреба в розробці сучасних автоматизованих рішень. Ці рішення повинні забезпечити ефективніше управління процесом взаємодії з клієнтами, а також вирішити проблеми, пов'язані з обмеженим часом роботи, необхідністю багаторазового введення даних тощо.

Оскільки фінансові установи витрачають значні ресурси на обслуговування клієнтів, такі як утримання великого штату операторів, оренда приміщень для відділень та обробка паперової документації, важливо визначити ключові функції автоматизованої системи, яка дозволить оптимізувати ці процеси. Впровадження чат-бота дозволить зменшити витрати на зарплату операторів, оренду приміщень, витратні матеріали, а також підвищити ефективність обробки клієнтських запитів, фінансових транзакцій, скарг. Це досягається завдяки автоматизації рутинних операцій, цілодобовій доступності сервісу, швидкій обробці інформації.

Автоматизована система обслуговування клієнтів фінансової установи на базі чат-бота може включати такі ключові компоненти:

1. Обробка запитів та надання інформації: Чат-бот повинен обробляти різні типи запитів клієнтів, надавати інформацію про продукти та послуги, допомагати з вирішенням поширених проблем.
2. Збір та аналіз даних про взаємодію з клієнтами: Система повинна збирати та аналізувати дані про взаємодію з клієнтами.

У сучасному світі, що характеризується швидким розвитком технологій та зростаючими потребами клієнтів, фінансові установи повинні забезпечити зручний та доступний сервіс. Мобільність та гнучкість стають ключовими факторами успіху в фінансовому секторі. Клієнти очікують можливості отримувати послуги з будь-якого пристрою, в будь-який час та з будь-якого місця.

У сфері обслуговування клієнтів фінансових установ також існують певні проблеми, які потребують вирішення. Наприклад, важливо забезпечити безпеку та конфіденційність даних клієнтів під час взаємодії з чат-ботом. Система повинна мати надійні механізми аутентифікації та авторизації, а також захист від шахрайства. Крім того, чат-бот повинен бути здатний обробляти складні та нестандартні запити, які виходять за межі заздалегідь програмованих сценаріїв. Важливо також забезпечити можливість ескалації запиту до оператора в тих випадках, коли чат-бот не може самостійно надати необхідну допомогу.

Для вирішення вищезазначених проблем в рамках даної магістерської роботи пропонується розробка автоматизованої системи обслуговування клієнтів фінансових установ на базі чат-бота з використанням сучасних технологій. Vertex AI використовується для створення та навчання чат-бота, що дозволяє йому розуміти природну мову та адекватно реагувати на запити клієнтів. Document AI забезпечує автоматизовану обробку документів, що спрощує та прискорює різні процеси, такі як, наприклад, відкриття рахунку. FastAPI використовується для розробки серверної частини веб-системи, що забезпечує високу продуктивність та гнучкість. Next.js використовується для створення сучасного та інтерактивного клієнтського інтерфейсу, що забезпечує зручну взаємодію клієнтів з системою. Веб-портал адміністратора дозволяє моніторити роботу системи, аналізувати статистику та керувати налаштуваннями. Завдяки цій архітектурі система забезпечує, наприклад, швидке та ефективне обслуговування, цілодобову доступність,

персоналізований підхід, високий рівень безпеки.

Розробка системи може бути економічно ефективною та масштабованою за допомогою використання хмарних технологій, таких як Vertex AI та Document AI. Фінансові установи можуть оптимізувати витрати на обслуговування IT-інфраструктури, сплачуючи лише за використані ресурси завдяки моделі оплати за використання (pay-as-you-go). Використання хмарних платформ у поєднанні з сучасними технологіями розробки, такими як Next.js та FastAPI, забезпечує економічну ефективність завдяки зниженню витрат на розробку та підтримку, а також пришвидшує вихід на ринок, гарантуючи при цьому високу доступність та надійність системи.

2.1 Проєктування архітектури системи

Щоб краще описати функціонування системи, зобразимо це на UML діаграмі (рис.2.1).

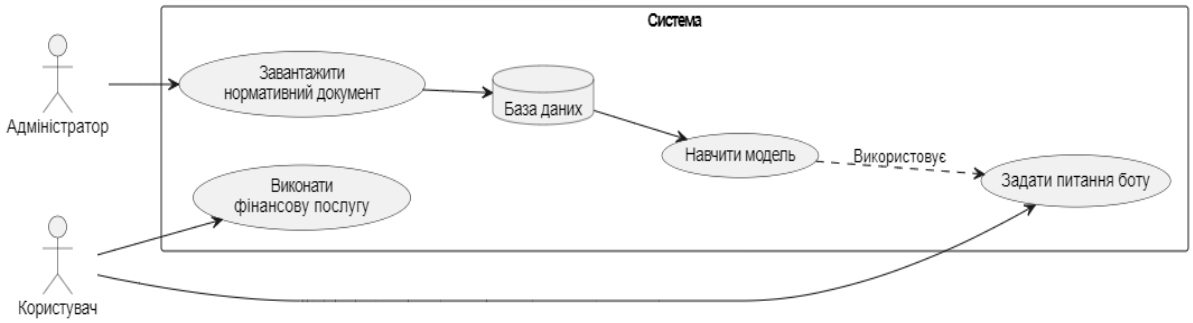


Рисунок 2.1 – Діаграма системи

Отже, існує два основних актори: адміністратор та користувач. Саме вони в основному взаємодіють з системою.

Все, що необхідно робити адміністратору, це завантажувати актуальні нормативні документи свого банку. Це можуть як і файли у текстовому форматі, так і у графічному. Усі файли до обробки зберігаються на хмарному сховищі. Далі обробляються за допомогою OCR. Тренувальні дані для моделі також завантажуються додатково у хмарне сховище. Відбувається тренування моделі,

при цьому попередні дані, які вже були, не видаляються.

Користувач через веб-застосунок має доступ до чат-бота, у якому він задає питання. Отримуючи відповідь, він також отримує посилання на джерело, що підтверджує правильність відповіді. Якщо він хоче виконати фінансові послуги, то повинен бути надати свої дані, а також ту інформацію, яку запитає сам бот.

Для забезпечення описаних функцій, автоматизована система обслуговування клієнтів фінансових установ може бути побудована на базі мікросервісної архітектури з використанням API для взаємодії між компонентами.

Виділимо два основних компоненти:

- Веб-застосунок з доступом до чат-боту.
- Платформа машинного навчання.

Веб-застосунок складається з бекенд(серверної) та фронтенд(клієнтської) частини, які взаємодіють між собою через API(REST). Ендпоїнт чат-боту у вигляді спеціального вікна знаходиться на клієнтській частині. Доступ до бази даних, у якій знаходяться дані користувача, реалізований на серверній частині. Також там є і доступ до платформи машинного навчання через спеціальний API-токен.

Платформа машинного навчання матиме вигляд «конвеєру» (рис. 2.2), де на вхід буде потрапляти документ(або скановене зображення), а на виході буде мовна модель з новими натренованими даними, зв’язок з якою буде через ендпоїнт у вигляді чат-боту. Увесь конвеєр автоматизований, і такі процеси, як обробка тексту, збір даних, збереження їх на хмарному сховищі, передача на тренування, розгортання моделі відбуваються автоматично.

Рисунок 2.2 – Конвеєр платформи машинного навчання

2.2 Оптичне розпізнавання символів (OCR) та текстові процесори

OCR (Optical Character Recognition), або оптичне розпізнавання символів, – це технологія, яка дозволяє перетворювати зображення тексту, такі як скановані документи, фотографії чи PDF-файли, на редагуємий цифровий текст (рис. 2.3). OCR аналізує форму літер на зображенні та зіставляє їх зі зразками в базі даних, щоб визначити відповідні символи. Сучасні OCR-системи використовують методи машинного навчання, зокрема глибокі нейронні мережі, для досягнення високої точності розпізнавання навіть для складних шрифтів та рукописного тексту [12].

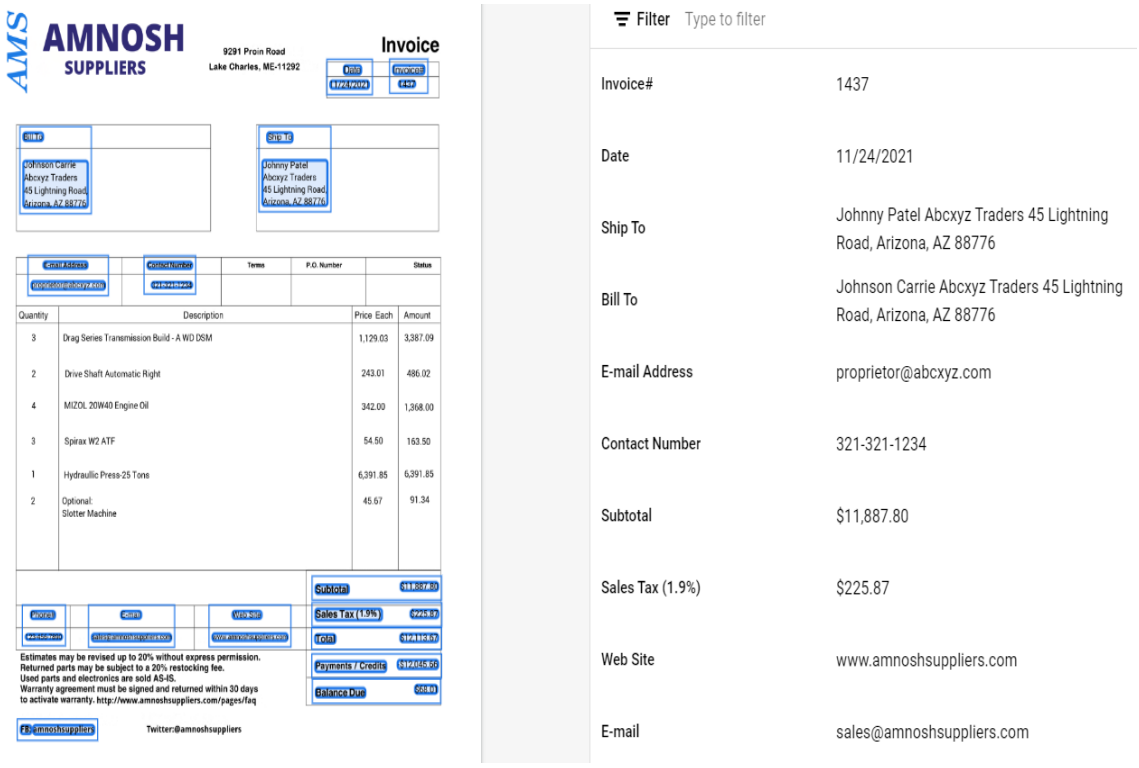


Рисунок 2.3 – Приклад роботи OCR

Текстові процесори — це програмне забезпечення, яке дозволяє створювати, змінювати та форматувати текстові документи. Вони можуть виконувати багато функцій, включаючи введення та редагування тексту, форматування шрифтів і абзаців, додавання зображень і таблиць, перевірку

граматики та правопису та багато іншого. Microsoft Word, Google Docs, LibreOffice Writer та інші є популярними текстовими процесорами.

OCR доповнює функціональність текстових процесорів, дозволяючи працювати з текстом, який спочатку був доступний лише у вигляді зображення. Після розпізнавання тексту за допомогою OCR, його можна скопіювати та вставити в текстовий процесор для подальшого редагування, форматування та збереження в редагованому форматі [20]. Це значно спрощує роботу з документами, усуваючи необхідність ручного переписування тексту.

Сучасні технології в OCR та текстових процесорах:

- Для покращення точності розпізнавання символів, автоматичного форматування та інших функцій використовуються технології машинного навчання та штучного інтелекту.
- Обробка природної мови (NLP): Дозволяє текстовим процесорам розуміти зміст тексту та надавати більш інтелектуальні функції, такі як автоматичне резюмування та переклад.
- Хмарні технології: Забезпечують доступ до OCR та текстових процесорів з будь-якого пристрою та місця, а також співпрацю над документами в реальному часі.

Переваги OCR та сучасних методів обробки тексту:

- Автоматизація процесів: OCR автоматизує вилучення даних з документів, усуваючи потребу в ручному введенні, що економить час та ресурси, а також мінімізує людський фактор та кількість помилок. Це особливо корисно для фінансових установ, які обробляють великі обсяги документів.

- Підвищення ефективності: Швидке та точне розпізнавання тексту прискорює обробку документів та прийняття рішень, що покращує загальну ефективність роботи фінансової установи.

- Зниження витрат: Автоматизація завдяки OCR зменшує потребу в ручній праці, що знижує витрати на персонал та обробку документів.
- Покращення якості обслуговування клієнтів: Швидша обробка документів дозволяє швидше реагувати на запити клієнтів та надавати послуги більш ефективно.
- Зручність пошуку: Перетворення зображень тексту на цифровий формат дозволяє легко шукати інформацію в документах за ключовими словами, що значно спрощує роботу з архівами та базами даних.
- Безпека даних: Зберігання цифрових документів забезпечує кращий контроль доступу та захист від втрати чи пошкодження інформації.
- Масштабованість: OCR-системи легко масштабуються для обробки зростаючих обсягів документів.
- Інтеграція з іншими системами: OCR може бути інтегрований з іншими бізнес-системами, такими як CRM та ERP, для автоматизації потоків робіт.
- Підтримка різних мов та шрифтів: Сучасні OCR-системи можуть розпізнавати текст на різних мовах та шрифтах, що робить їх універсальним інструментом для міжнародних компаній.
- Покращена доступність: Цифрові документи можуть бути легко доступні для людей з обмеженими можливостями, наприклад, за допомогою програм для читання з екрану.

Переваги текстових процесорів з інтегрованим AI [10]:

- Розумне редагування: AI може пропонувати покращення стилю,

граматики та виправлення помилок в режимі реального часу.

- Автоматичне форматування: AI може допомогти автоматично форматувати документи відповідно до заданих стилів.
- Автоматичне створення контенту: Деякі текстові процесори з AI можуть генерувати текстовий контент на основі заданих параметрів.
- Переклад в реальному часі: AI може перекладати текст на різні мови безпосередньо в текстовому процесорі.

2.3 Хмарне середовище

Розробка сучасних фінансових помічників вимагає потужної та гнучкої хмарної інфраструктури. Серед лідерів ринку хмарних обчислень виділяються Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Кожна платформа пропонує широкий спектр сервісів.

AWS є піонером у хмарних обчисленнях та пропонує величезний вибір сервісів, від віртуальних машин (EC2) та безсерверних функцій (Lambda) до баз

Рисунок 2.4 – Google Cloud Platform

даних (RDS, DynamoDB) та інструментів для машинного навчання (SageMaker). AWS відомий своєю зрілістю, широким колом користувачів та великою кількістю доступної документації та ресурсів [28].

Azure тісно інтегрований з екосистемою Microsoft, що робить його привабливим вибором для компаній, які вже використовують продукти Microsoft. Azure пропонує віртуальні машини, сервіси для безсерверних обчислень (Azure Functions), бази даних (Azure SQL Database, Cosmos DB) та платформу для машинного навчання (Azure Machine Learning). Сильна сторона Azure - це його гібридні хмарні рішення, що дозволяють поєднувати локальну інфраструктуру з хмарними ресурсами [29].

Google Cloud Platform (GCP) (рис. 2.4) виділяється серед конкурентів завдяки своїм інноваційним розробкам у сфері штучного інтелекту та машинного навчання, зокрема, пропонуючи платформу Vertex AI. GCP також має сильні сторони в аналізі даних завдяки BigQuery, сервісу для обробки великих даних. Compute Engine надає віртуальні машини, Cloud Functions - безсерверні обчислення, а Cloud SQL та Cloud Spanner - різноманітні варіанти баз даних. GCP також активно розвиває Kubernetes, платформу для оркестрації контейнерів [35].

В процесі аналізу проблем обслуговування клієнтів фінансових установ та пошуку оптимальних рішень було обрано технології Google Cloud, зокрема Vertex AI, Document AI. Вибір Google Cloud обумовлений такими факторами, як висока продуктивність, масштабованість, надійність, безпека, розвинена екосистема, конкурентна ціна, наявність спеціалізованих сервісів для розробки чат-ботів та обробки документів. Використання цих технологій дозволяє створити ефективну, гнучку та економічно вигідну систему обслуговування клієнтів.

GCP — це універсальна хмарна платформа, яка надає повний набір інструментів для розробки, запуску та масштабування програм. GCP надає

різноманітні сервіси, від віртуальних машин та контейнерів до баз даних, сховищ даних та інструментів для аналітики (рис. 2.4). Створення чат-бота та обробка документів у цій роботі базуються на використанні технологій штучного інтелекту, зокрема Vertex AI та Document AI, наданих GCP. Крім того, GCP підтримує технології Інтернету речей (IoT) та надає інструменти для аналізу великих даних. Вибір GCP для даного проекту обумовлений його гнучкістю, масштабованістю, надійністю та широким набором доступних сервісів. GCP дозволяє ефективно розгорнути та підтримувати розроблену систему обслуговування клієнтів, забезпечуючи високу доступність та продуктивність [35].

Хмарні обчислення, або "хмара", надають обчислювальні ресурси через Інтернет за моделлю оплати за використання [36]. Ці ресурси включають сервери, сховища даних, бази даних, мережі, програмне забезпечення, аналітичні інструменти та сервіси штучного інтелекту. В рамках даної магістерської роботи, використання хмарної платформи Google Cloud дозволило скористатися такими перевагами хмарних обчислень:

- Зниження експлуатаційних витрат: Оплата лише за використані ресурси дозволяє мінімізувати витрати на утримання власної IT-інфраструктури.
- Ефективне управління інфраструктурою: Google Cloud бере на себе відповідальність за управління інфраструктурою, що дозволяє зосередитися на розробці та вдосконаленні системи обслуговування клієнтів.
- Гнучке масштабування: Швидке адаптування обсягу ресурсів до поточних потреб забезпечує економію коштів та ефективне використання ресурсів.
- Надійне сховище даних: Google Cloud надає надійне та безпечне сховище даних для зберігання інформації про клієнтів, транзакцій

та інших важливих даних.

- Висока продуктивність та доступність: Хмарна інфраструктура Google Cloud забезпечує високу продуктивність та доступність розробленої системи, що є критично важливим для безперебійного обслуговування клієнтів фінансової установи.
- Завдяки доступу до таких передових технологій, як Vertex AI та Document AI, можливе швидке та ефективне вирішення складних завдань в сфері обслуговування клієнтів.

Фінансові установи можуть швидше розробляти та впроваджувати інноваційні рішення в обслуговуванні клієнтів завдяки використанню Google Cloud Platform [35]. В сучасному динамічному середовищі спостерігаються аналогічні тенденції:

- Швидка розробка та впровадження нового функціоналу: Фінансові установи постійно працюють над вдосконаленням своїх сервісів та пропонують клієнтам нові можливості з високою швидкістю.
- Зростаючі очікування клієнтів: Клієнти фінансових установ очікують зручного, інтерактивного та персоналізованого досвіду обслуговування.

Google Cloud дозволяє фінансовим установам швидко реагувати на зміни

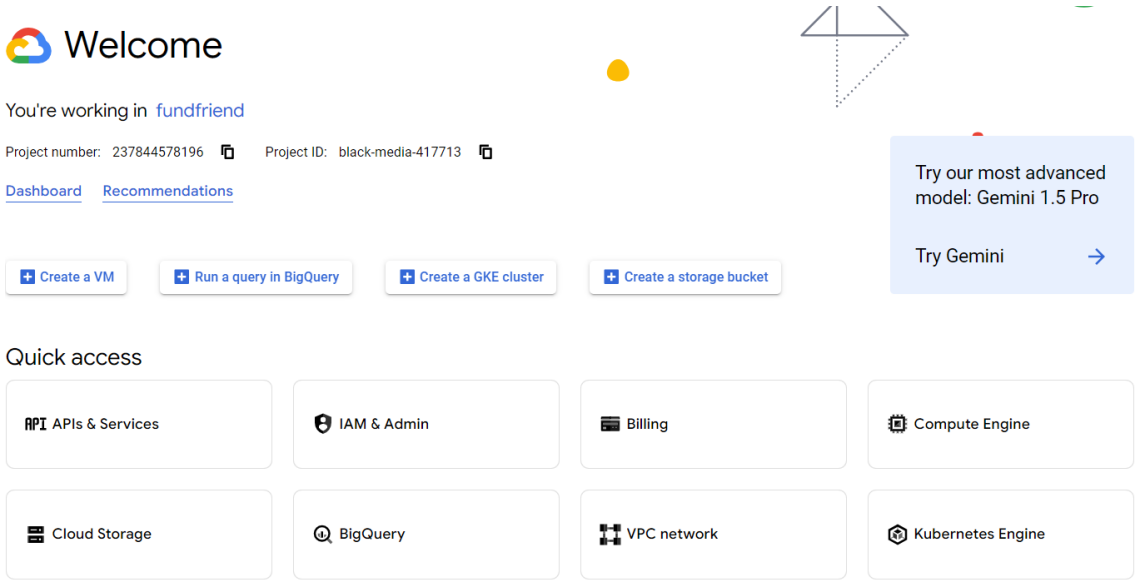


Рисунок 2.5 – Вигляд Google Cloud Console

ринкових вимог та задовольняти зростаючі очікування клієнтів. Завдяки хмарним технологіям, процес розробки та впровадження нового функціоналу значно прискорюється, що дозволяє частіше оновлювати системи та пропонувати клієнтам актуальні послуги. В рамках даної магістерської роботи, використання Google Cloud дозволило ефективно розробити та розгорнути чат-бота для обслуговування клієнтів, забезпечуючи швидкість, гнучкість та масштабованість рішення [35].

Google Cloud Platform пропонує широкий спектр сервісів, що охоплюють різноманітні потреби розробки та розгортання додатків, включаючи віртуальні машини, бази даних, аналітичні інструменти, штучний інтелект та машинне навчання.

Google Cloud Console — це веб-інтерфейс для управління ресурсами та сервісами Google Cloud. Він надає централізовану точку доступу до всіх функцій платформи, дозволяючи моніторити використання ресурсів, налаштовувати сервіси, керувати доступами та виконувати інші операції (рис. 2.5). В рамках даного проекту, Google Cloud Console використовувався для розгортання чат-бота на Vertex AI, налаштування Document AI, моніторингу продуктивності системи. Google Cloud також надає інструменти командного рядка (gcloud) для автоматизації різних завдань [35].

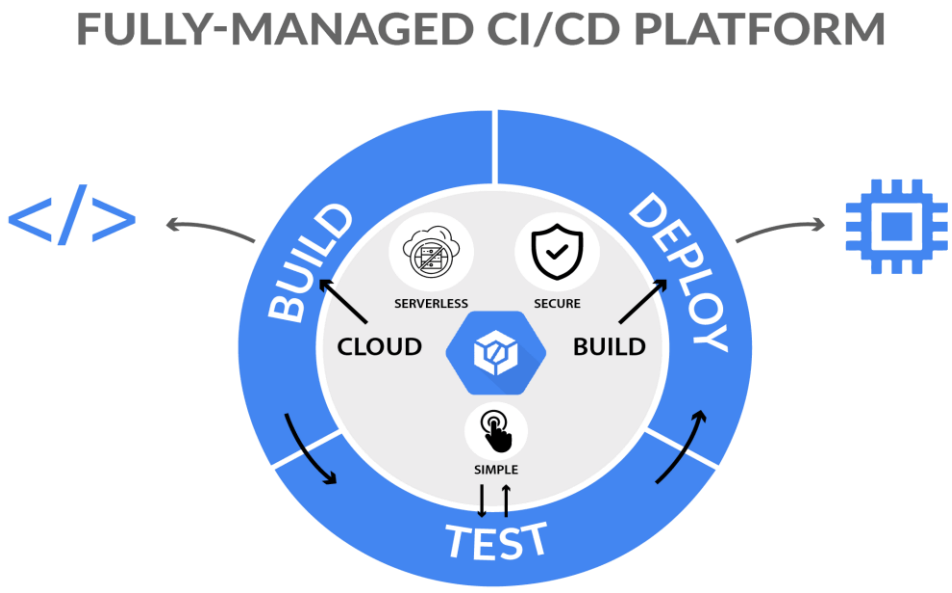


Рисунок 2.6 – Реалізація CI/CD в Google Cloud Build

Google Cloud пропонує широкий спектр сервісів для зберігання даних, мережевої інфраструктури, розпізнавання мовлення та інших когнітивних функцій, які дозволяють створювати потужні та інноваційні рішення. Сервіси аналітики Google Cloud, такі як Cloud Monitoring та Cloud Logging, надають детальну інформацію про роботу системи, що дозволяє швидко виявляти та вирішувати проблеми, а також оптимізувати продуктивність.

2.4 GCP сервіси – Document AI, Vertex AI, Cloud Build

Google Cloud пропонує комплексне рішення для розробки та розгортання додатків, що дозволяє вирішувати складні завдання в сфері обслуговування клієнтів фінансових установ. В рамках даного проекту, для створення чат-бота були використані такі ключові сервіси, як Vertex AI, Document AI та Cloud Build. Для розробки веб-застосунку (frontend) було обрано фреймворк Next.js, що дозволяє створити сучасний та інтерактивний інтерфейс користувача. Для розгортання backend частини застосунку та обробки запитів до чат-бота використовується FastAPI, що забезпечує високу продуктивність та гнучкість. Вибір цих технологій забезпечує ефективне вирішення поставлених завдань та створення оптимального рішення для обслуговування клієнтів фінансової установи.

У сучасному фінансовому секторі забезпечення високоякісного онлайн-досвіду для клієнтів є надзвичайно важливим. Google Cloud надає всебічну підтримку для створення та розгортання веб-додатків і сервісів, забезпечуючи високу продуктивність, надійність та безпеку.

Cloud Build — це сервіс Google Cloud Platform, який надає повністю керовану платформу для створення та автоматизації процесу збирання, тестування та розгортання ваших додатків [37]. Розробка та розгортання програмного забезпечення можливі в різних середовищах, включаючи Google App Engine , Google Kubernetes Engine, Cloud Functions та Cloud Run, з

підтримкою різних мов програмування (рис. 2.6).

Cloud Build — ідеальний інструмент для розробників, які працюють з різними типами проектів, завдяки широкій підтримці мов програмування та фреймворків, а також можливості інтеграції з багатьма сервісами GCP для безпечного розгортання.

Основні переваги Cloud Build:

- Керований сервіс: Вам не потрібно турбуватися про управління інфраструктурою для збирання та розгортання, Cloud Build бере це на себе.
- Швидкість та масштабованість: Навіть для великих проектів Cloud Build забезпечує швидке збирання та розгортання завдяки потужній інфраструктурі Google Cloud.
- Інтеграція Cloud Build з іншими сервісами GCP, такими як Cloud Source Repositories, Container Registry та Kubernetes Engine, спрощує та автоматизує процеси розробки та розгортання.
- Підтримка різних мов програмування та інструментів: Cloud Build підтримує широкий спектр мов програмування, від Java та Golang до Node.js та Python, а також різні інструменти збирання, такі як Docker, Maven, Gradle, npm та yarn.
- Налаштування за допомогою конфігураційних файлів: Процес збирання та розгортання описується в конфігураційних файлах YAML або JSON, що забезпечує гнучкість та контроль над процесом.
- Автоматизація процесу CI/CD: Cloud Build дозволяє легко налаштувати безперервну інтеграцію та безперервне розгортання (CI/CD), що прискорює процес розробки та випуску програмного забезпечення.
- Безпека: Cloud Build інтегрується з іншими сервісами безпеки

Google Cloud, забезпечуючи захист вашого коду та даних.

Document AI - це платформа на Google Cloud, що використовує машинне навчання для автоматизації обробки документів [21]. Вона дозволяє вилучати дані з різних типів документів, таких як рахунки-фактури, контракти, форми та інші, перетворюючи їх на структуровані дані, які можна використовувати для подальшої обробки та аналізу (рис. 2.7). Замість ручного введення даних, Document AI використовує OCR та NLP для розпізнавання та інтерпретації інформації в документах.

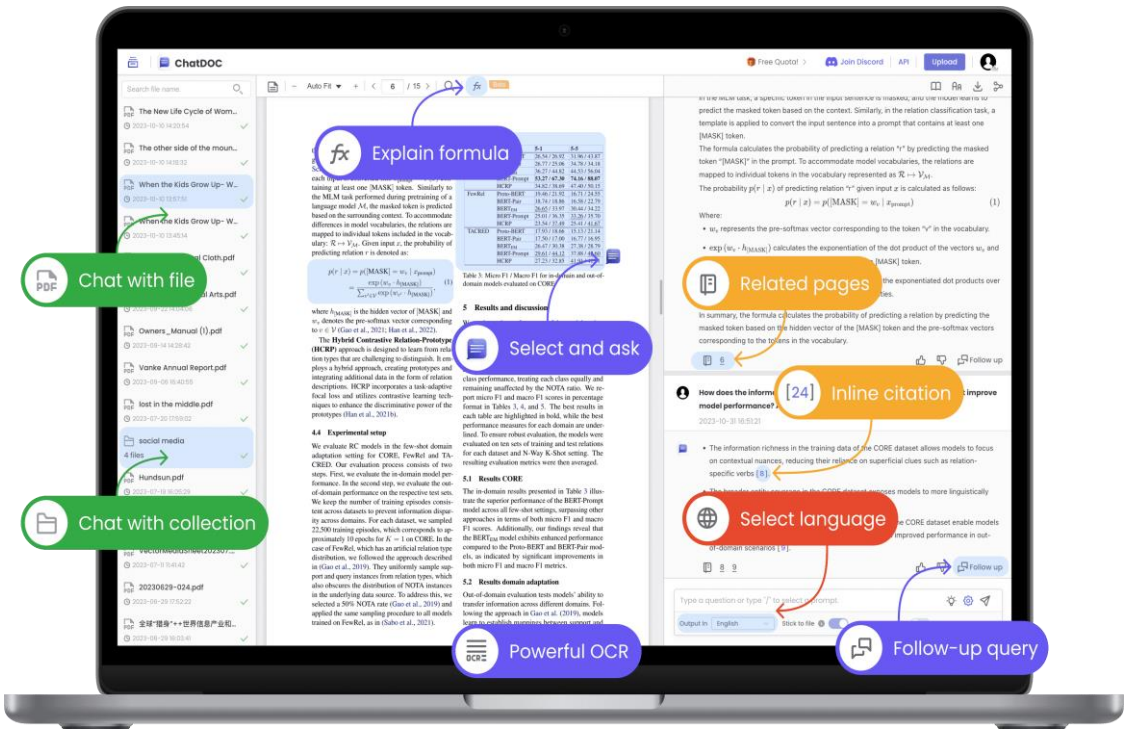


Рисунок 2.7 – Doc AI

Основні можливості Document AI:

- Обробка різних типів документів: Document AI може обробляти широкий спектр документів, включаючи структуровані, напівструктуровані та неструктуровані документи.
- Вилучення ключової інформації: Document AI ідентифікує та вилучає ключову інформацію з документів, таку як дати, суми, імена, адреси та інші важливі дані.

- Класифікація документів: Document AI може класифікувати документи за типом, що спрощує їх обробку та маршрутизацію.
- Перевірка даних: Document AI може перевіряти дані, вилучені з документів, на точність та повноту.
- Інтеграція з іншими сервісами GCP: Document AI легко інтегрується з іншими сервісами Google Cloud, такими як Cloud Storage, Cloud Functions та BigQuery.

Переваги використання Document AI:

- Зниження витрат: Автоматизація обробки документів зменшує потребу в ручній праці, що знижує витрати на персонал.
- Підвищення ефективності: Швидка та точна обробка документів прискорює бізнес-процеси.
- Покращення точності: Автоматизація зменшує ризик помилок, пов'язаних з ручним введенням даних.
- Масштабованість: Document AI легко масштабується для обробки великих обсягів документів.
- Покращення досвіду клієнтів: Швидша обробка документів дозволяє швидше реагувати на запити клієнтів.

Vertex AI – це уніфікована платформа машинного навчання (ML) від

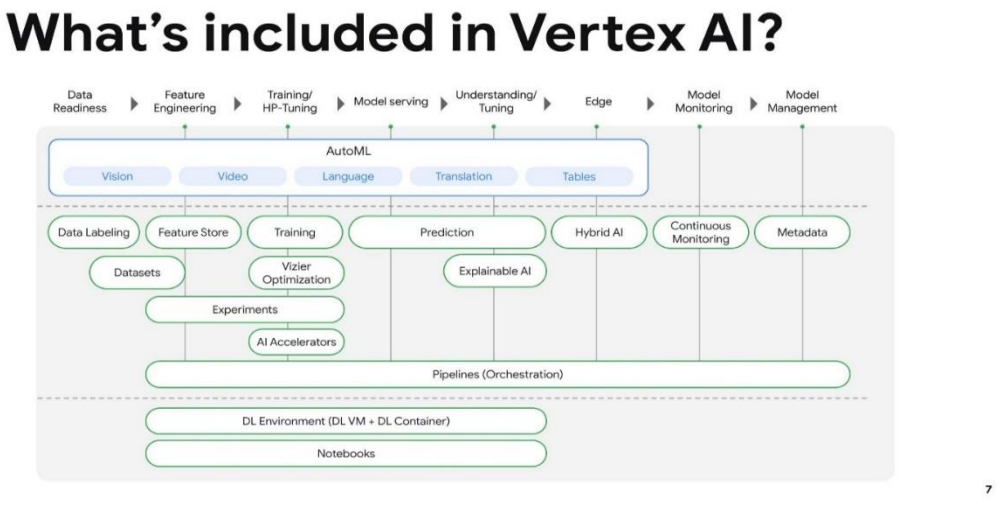


Рисунок 2.8 – Компоненти Vertex AI

Google Cloud, яка спрощує процес розробки, навчання та розгортання моделей ML (рис. 2.8). Широкий спектр інструментів та сервісів платформи дозволяє розробникам різного рівня, від початківців до експертів, створювати та впроваджувати ML-рішення в різних сферах [20].

Основні компоненти та можливості Vertex AI:

- Data Labeling: Сервіс призначений для маркування даних, що є критично важливим кроком у процесі підготовки даних для машинного навчання. Вертекс AI Data Labeling дозволяє експертам маркувати дані за допомогою методів, таких як класифікація зображень, розпізнавання об’єктів і сегментація зображень.
- Feature Store: Централізоване сховище для зберігання та управління ознаками, що використовуються для навчання моделей ML. Це дозволяє уникнути дублювання ознак та забезпечує їх консистентність у різних проектах.
- Training: Платформа для навчання моделей ML, що підтримує різні фреймворки, такі як TensorFlow, PyTorch та scikit-learn. Vertex AI Training надає можливість використовувати потужні обчислювальні ресурси Google Cloud для прискорення процесу навчання.
- Workbench: Кероване середовище розробки Jupyter notebook, що дозволяє експериментувати з моделями ML, аналізувати дані та співпрацювати з колегами.
- Prediction: Сервіс для розгортання навчених моделей ML та отримання прогнозів. Vertex AI Prediction забезпечує масштабованість та надійність розгорнутих моделей.
- Pipelines: Інструмент для створення та управління складними конвеєрами ML, що автоматизують весь процес від підготовки

даних до розгортання моделі.

- Matching Engine: Сервіс для пошуку найближчих сусідів, що може бути використано для рекомендаційних систем та інших завдань.
- Model Monitoring: Інструмент для моніторингу продуктивності розгорнутих моделей ML та виявлення потенційних проблем.

Vertex AI спрощує та прискорює розробку, навчання та розгортання моделей ML. Вона надає єдину платформу для всіх етапів життєвого циклу ML, від підготовки даних до моніторингу моделей. Використання Vertex AI дозволяє скоротити час виходу на ринок, знизити витрати та підвищити ефективність ML-рішень [20].

Аналіз платформи Google Cloud, зокрема Vertex AI, Document AI, FastAPI та інших сервісів, показав, що вони забезпечують оптимальне середовище для розробки та розгортання чат-бота для фінансових установ. Тісна інтеграція між сервісами Google Cloud спрощує процес розробки, дозволяючи ефективно поєднувати різні компоненти системи, такі як розпізнавання мовлення, обробка природної мови, обробка документів та машинне навчання. Використання FastAPI для backend та Next.js для frontend, в поєднанні з іншими сервісами Google Cloud, забезпечує високу продуктивність, масштабованість та надійність розробленої системи. Обрані технології дозволяють створити гнучке та ефективне рішення, що відповідає вимогам сучасного фінансового сектору [21].

2.5 Обґрунтування використання мов програмування Python та JavaScript

Python є однією з найпопулярніших і найбільш швидкозростаючих мов програмування в галузі розробки програмного забезпечення, включаючи сферу штучного інтелекту та машинного навчання. Її використовують для створення широкого спектру додатків, від невеликих скриптів до складних веб-платформ та систем обробки даних, які обслуговують мільйони користувачів (рис. 2.9).

Гнучкість Python дозволяє використовувати її в різних сферах, включаючи розробку веб-застосунків, аналіз даних, машинне навчання та автоматизацію. В контексті даного проекту, Python був обраний для розробки backend частини чат-бота, використовуючи фреймворк FastAPI, що забезпечує високу продуктивність та ефективність [13]. Бібліотеки Python, такі як TensorFlow та PyTorch, широко використовуються в сфері машинного навчання та глибокого навчання, що робить Python ідеальним вибором для розробки чат-ботів на базі штучного інтелекту. Він також добре інтегрується з сервісами Google Cloud, що використовуються в цьому проекті, такими як Vertex AI та Document AI.

Python був вперше випущений Гвідо ван Россумом в 1991 році. З того часу мова пройшла значний шлях розвитку, і сьогодні актуальною є версія Python 3. Python має чіткий та легко читаємий синтаксис, що сприяє швидкій розробці та підтримці коду. Хоча Python має динамічну типізацію, що відрізняє його від C#

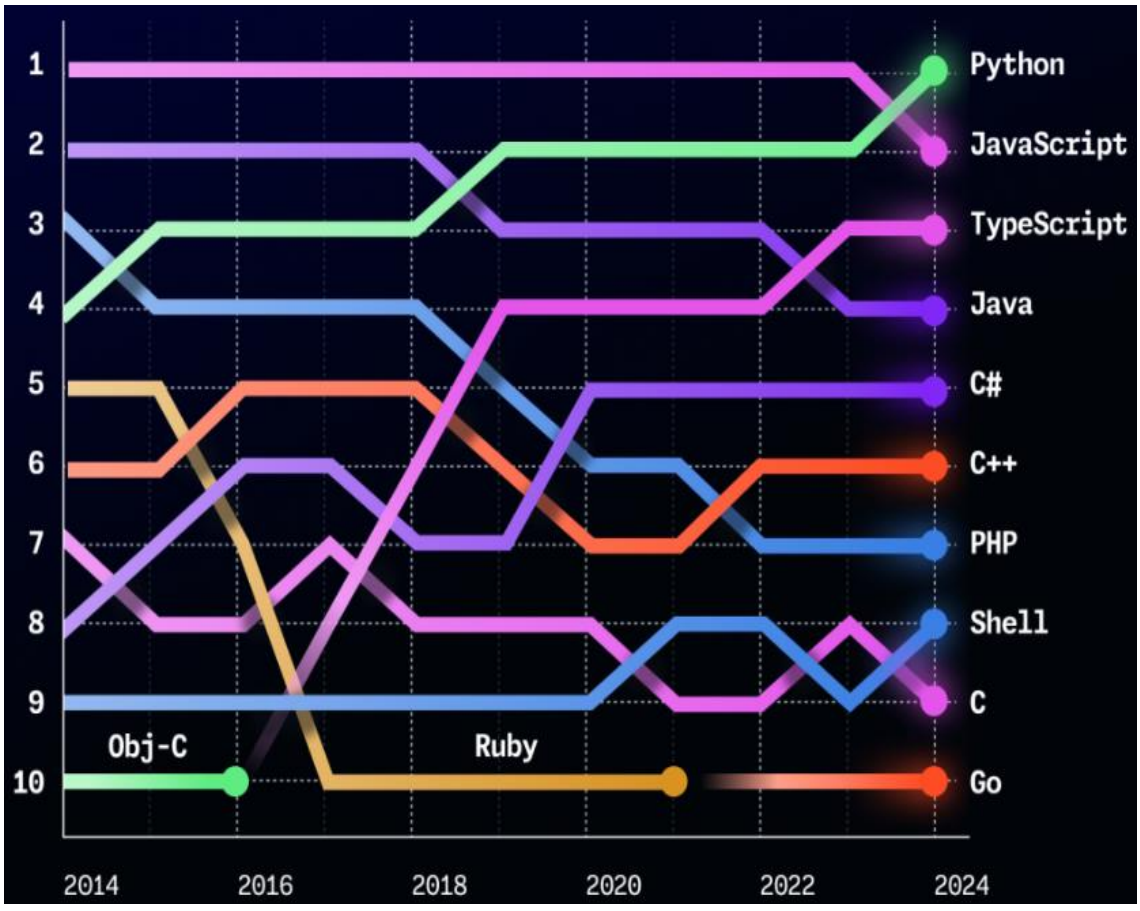


Рисунок 2.9 – К-сть користувачів мов програмувань

та Java, сучасні версії мови також підтримують опціональні типи даних, що дозволяє покращити читабельність та надійність коду [37].

Python — це об'єктно-орієнтована мова програмування, що підтримує ключові концепції ООП, такі як успадкування, поліморфізм та інкапсуляція. Об'єктно-орієнтований підхід дозволяє створювати модульний, легко підтримуваний та масштабований код, що є важливим для розробки складних систем, таких як чат-бот для фінансової установи. Python також підтримує динамічну типізацію, що забезпечує гнучкість та швидкість розробки, а також опціональні типи даних для підвищення надійності коду. Мова постійно розвивається, з'являються нові бібліотеки та фреймворки, що розширюють її можливості та роблять її ще більш придатною для вирішення різноманітних завдань, включаючи розробку чат-ботів та інших систем на базі штучного інтелекту [37].

Переваги Python:

- Простота та читабельність: Python відрізняється чітким та легко читаємим синтаксисом, що сприяє швидкій розробці та підтримці коду. Це особливо важливо для проектів з штучним інтелектом, де код може бути досить складним.
- Інтерпретована мова: Python є інтерпретованою мовою, що спрощує процес розробки та налагодження.
- Широке застосування: Python використовується в різних галузях, від веб-розробки та аналізу даних до машинного навчання та наукових обчислень. Це свідчить про її гнучкість та універсальність.
- Багата кількість бібліотек і фреймворків: велика кількість бібліотек і фреймворків Python полегшує розробку багатьох додатків. Бібліотеки машинного навчання (TensorFlow, PyTorch), веб-розробки (FastAPI, Flask, Django) та обробки даних (Pandas,

NumPy) є важливими для конкретного проекту.

- Контроль над виконанням програми: Python надає розробникам достатній рівень контролю над виконанням програми.
- Активний розвиток: Python постійно розвивається, з'являються нові функції та покращення продуктивності.
- Надійність та елегантність: Python відомий своєю надійністю та елегантністю коду, що сприяє створенню якісного та підтримуваного програмного забезпечення. Він добре підходить для розробки складних систем, таких як чат-боти на базі штучного інтелекту.

Python тісно інтегрується з різноманітними бібліотеками та фреймворками, що розширюють його можливості та спрощують розробку різних додатків [37]. В контексті даного проекту ключову роль відіграє фреймворк FastAPI, який використовується для створення backend-у чат-бота. FastAPI — це сучасний, високопродуктивний веб-фреймворк для створення API на Python. Він базується на стандартах ASGI та використовує типи даних Python для валідації та документування API. FastAPI також підтримує асинхронне програмування, що дозволяє створювати швидкі та ефективні веб-додатки. Використання FastAPI в поєднанні з іншими бібліотеками Python, такими як Pydantic для валідації даних та Uvicorn як ASGI-сервер, створює потужний та ефективний інструментарій для розробки backend-частини чат-бота. Цей набір технологій можна розглядати як взаємопов'язаний стек, оптимізований для створення високопродуктивних веб-додатків на Python [37].

JavaScript - надзвичайно популярна мова програмування, що широко використовується, особливо для веб-розробки. Від інтерактивних веб-сторінок до backend-сервісів, що обслуговують мільйони користувачів, JavaScript використовується для створення різноманітних застосунків, включаючи односторінкові додатки (SPA) та мобільні додатки. JavaScript відрізняється

великою гнучкістю, що дозволяє розробникам створювати різноманітні продукти для веб, мобільних платформ та серверів. В контексті даного проекту, JavaScript (разом з фреймворком Next.js) був обраний для розробки frontend частини чат-бота, забезпечуючи інтерактивність та зручність користувацького інтерфейсу. Next.js — це фреймворк на базі React, що дозволяє створювати оптимізовані та SEO-дружні веб-додатки. Він надає такі можливості, як серверний рендеринг, автоматичне розділення коду та просте розгортання. Використання JavaScript та Next.js дозволяє створити сучасний та ефективний frontend для чат-бота, що забезпечує позитивний досвід взаємодії для клієнтів фінансової установи.

JavaScript був створений Бренданом Айхом в 1995 році і спочатку був призначений для додання інтерактивності веб-сторінкам. З того часу мова знайшла застосування в багатьох інших сферах. JavaScript — мова, що перебуває в стані постійного розвитку, і нові версії стандарту ECMAScript систематично випускаються, доповнюючи її функціональність та підвищуючи ефективність. JavaScript має C-подібний синтаксис, що робить його доступним для розробників, знайомих з мовами C++, Java та C#. Однак, на відміну від C#, JavaScript є динамічно типізованою мовою, що забезпечує більшу гнучкість під час розробки, але потребує більшої уваги до типів даних під час написання коду [38].

Хоча JavaScript спочатку не був розроблений як чисто об'єктно-орієнтована мова, він підтримує об'єктно-орієнтований стиль програмування, включаючи прототипне успадкування. Це дозволяє створювати складні та масштабовані додатки, використовуючи концепції об'єктів та методів. Сучасні версії JavaScript також підтримують класи та модулі, що полегшує організацію та підтримку коду. В контексті даного проекту, використання React (бібліотека, на якій базується Next.js), що сповідує компонентний підхід, дозволяє створювати гнучкий та зручний у підтримці користувацький інтерфейс для чат-

бота. JavaScript — це мова, що динамічно розвивається, і нові функції та підходи постійно з'являються, роблячи її ще більш потужним інструментом для веб-розробки [38].

Переваги JavaScript:

- Універсальність і широке застосування: JavaScript домінує у веб-розробці, забезпечуючи створення інтерактивних сторінок та front-end додатків, а також використовується для back-end розробки, мобільних додатків та ігор.
- Гнучкість та динамічність: Завдяки динамічній типізації та гнучкому синтаксису, JavaScript ідеально підходить для швидкого прототипування та експериментальної розробки.
- Велика спільнота та екосистема: JavaScript має величезну спільноту розробників, що забезпечує широкий вибір бібліотек, фреймворків та інструментів, що спрощують розробку та підтримку проектів. В контексті даного проекту, використання Next.js, популярного фреймворку на базі React, значно спростило розробку frontend частини чат-бота.
- Інтерпретована мова: JavaScript є інтерпретованою мовою, що спрощує процес розробки та налагодження.
- Постійний розвиток: JavaScript постійно розвивається, нові стандарти ECMAScript регулярно випускаються, додаючи нові функції та покращення мови.
- Інтеграція з веб-технологіями: JavaScript бездоганно інтегрується з HTML та CSS, що робить його ідеальним вибором для веб-розробки.

Таким чином, обраний для розробки чат-бота стек технологій на базі Python та FastAPI, в поєднанні з сервісами Google Cloud, є ефективним та сучасним рішенням для створення backend систем. Висока продуктивність

FastAPI, розширені можливості Python та інтеграція з Google Cloud дозволяють створювати масштабовані та надійні рішення для обслуговування клієнтів фінансових установ. Цей підхід є оптимальним як для невеликих проектів, так і для складних систем підприємницького рівня.

2.6 Мікросервіси

Потрібно бути в курсі подій у період, коли технології розвиваються і тенденції для додатків змінюються майже щодня. Розробники все частіше переходять на мікросервісну архітектуру, оскільки монолітні програми важко розширювати та їм потрібно платити за постійну підтримку функціоналу.

Моноліти були створені для об’єднання. Навіть незначні зміни вимагають перебудови та запуску програми в цілому. Згодом зберігати ефективну модульну структуру стає все складніше. Внесення змін до монолітних програм є складним завданням, оскільки зміна одного модуля може вплинути на інші. Масштабування також є проблемою через неоднорідні потреби модулів в ресурсах.

У монолітних застосунках різні модулі можуть мати суперечливі вимоги до ресурсів. Наприклад, модуль обробки зображень може бути ресурсомістким для процесора, тоді як іншому модулю потрібна велика кількість оперативної пам'яті. Це призводить до компромісів при виборі обладнання та неефективного масштабування, оскільки масштабувати доводиться всю програму, навіть якщо це потрібно лише одному модулю. Така неефективність негативно впливає на витрати на ресурси, команду та підтримку проекту [40].

Поняття "мікросервісна архітектура" або "мікросервіси" виникло в середині 2010-х років для опису нового підходу до розробки програмного забезпечення, що набув популярності завдяки поширенню гнучкої розробки та DevOps [39].

При використанні мікросервісної архітектури застосунок створюється як комбінація невеликих, автономних і незалежних сервісів, які взаємодіють один з одним за допомогою легких протоколів, таких як HTTP, gRPC та AMQP (рис. 2.10). Дані сервіси створені відповідно до бізнес-потреб, причому кожен з них відповідає за конкретний процес та функціонує самостійно в повністю автоматизованому середовищі з мінімальною централізацією. В рамках

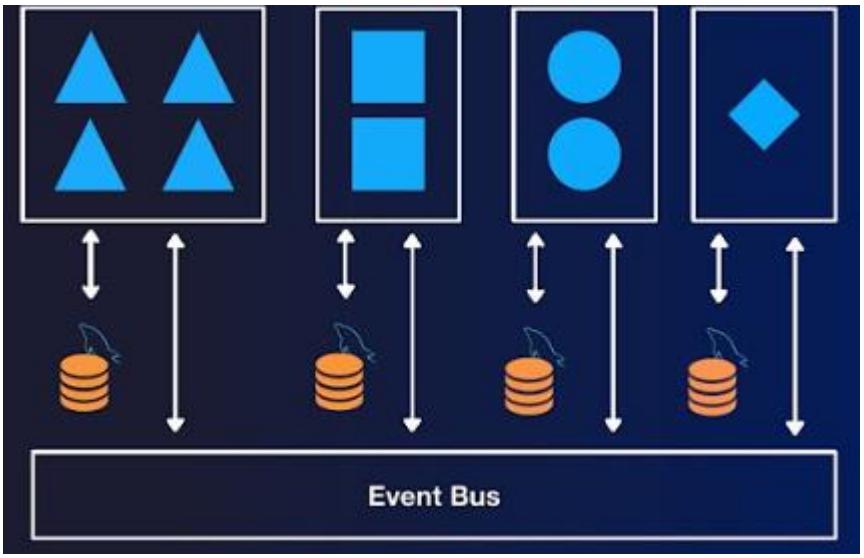


Рисунок 2.10 – Схема мікросервісної архітектури

мікросервісної архітектури, кожен сервіс може використовувати свою мову програмування та технологію зберігання даних [40].

Однією з ключових переваг мікросервісів є можливість швидкого реагування на зміни ринку та бізнес-вимог. Незалежність сервісів дозволяє розробникам вносити зміни швидше, безпечніше та з меншим ризиком для всієї системи, підтримуючи високу швидкість розробки навіть для великих проектів. Завдяки меншій взаємозалежності модулів, ітерації стають частішими, а вплив змін мінімізується [39].

Важливо відзначити, що Python та FastAPI добре підходять для розробки мікросервісної архітектури. Зручність розробки на Python, висока продуктивність FastAPI та можливість легкої інтеграції з іншими сервісами

Google Cloud, такими як Kubernetes Engine, роблять цей стек технологій привабливим вибором для створення мікросервісів[16]. Це дозволяє створювати гнучкі, масштабовані та легко підтримувані системи обслуговування клієнтів.

FastAPI - це сучасний високопродуктивний фреймворк для створення веб-додатків та API на Python. Він відмінно підходить для розробки кросплатформних мікросервісів, які можна легко розгорнути на Google Cloud та інших хмарних платформах, а також на різних операційних системах. FastAPI відрізняється простотою використання, високою продуктивністю та відмінною інтеграцією з сучасними інструментами та технологіями, що робить його ефективним вибором для розробки мікросервісної архітектури в рамках даного проекту. Його підтримка асинхронного програмування та вбудована валідація даних сприяють створенню надійних та ефективних мікросервісів [15].

Застосування гнучкої розробки та доставки складних корпоративних додатків має значні переваги:

- Сервіси запускаються швидше, що збільшує продуктивність розробників і прискорює розгортання.
- Кожен сервіс може працювати окремо від інших. Таким чином, ви можете запровадити ці зміни, якщо щось зміниться в одному з них, не зашкоджуючи іншим мікросервісам, які все ще можуть працювати.
- Ви можете масштабувати кожен окремий сервіс.
- Неполадки в одному мікросервісі не зменшать продуктивність системи. Несправності мікросервісу не повинні зупинити весь програмний продукт. Вони, швидше за все, не вплинуть на роботу програми, особливо великої.
- Мікросервіси усувають прив'язку до конкретних технологій. При розробці нового сервісу можна обрати будь-який стек технологій,

а існуючі сервіси можна замінити або переписати без необхідності змінювати всю систему. Це значно зменшує витрати та час на модернізацію.

- Завдяки чіткому розподілу відповідальності, мікросервіси, як правило, краще структуровані, що спрощує їх розуміння, підтримку та тестування. Кожен мікросервіс зосереджений на виконанні однієї конкретної функції.
- Окремі мікросервіси легко комбінуються та налаштовуються для виконання різних завдань в різних додатках, наприклад, для обслуговування веб-користувачів та публічного API [17].

Недоліки мікросервісів:

- Розробка розподілених систем на базі мікросервісів представляє собою додаткову складність для розробників.
- Труднощі з розгортанням У виробництві також існує оперативна складність розгортання та управління системою, яка складається з багатьох різних типів послуг.
- При розробці нової архітектури мікросервісу, ймовірно, виникне багато проблем, пов’язаних із різними планами, які не передбачалися під час розробки.

В умовах постійних змін на ринку підтримувати стійкість та адаптивність досить складно. Саме тому мікросервісний підхід до розробки програмного забезпечення набуває популярності як альтернатива монолітному.

2.7 Зберігання даних і їх аналіз

В рамках розробки чат-бота для фінансових установ, вибір відповідних технологій для зберігання та управління даними є критично важливим для

забезпечення продуктивності, масштабованості та надійності системи. В даній роботі було обрано поєднання AlloyDB та Cloud Storage для вирішення різних задач, пов'язаних зі зберіганням та обробкою даних.

Cloud Storage - це об'єктне сховище від Google Cloud Platform, що пропонує надійне, масштабоване та економічно вигідне рішення для зберігання даних будь-якого типу та розміру [41]. Від невеликих файлів до петабайтів даних, Cloud Storage забезпечує їх безпечне зберігання та доступ до них з будь-якої точки світу (рис. 2.11).

Рисунок 2.11 – Cloud Storage

Основні переваги Cloud Storage:

- Масштабованість та доступність: Cloud Storage автоматично масштабується для задоволення ваших потреб у зберіганні даних, забезпечуючи високу доступність та низьку латентність.
- Безпека даних: Дані, що зберігаються в Cloud Storage, захищені за допомогою передових технологій шифрування та керування доступом.

- Економічна ефективність: Ви сплачуєте лише за використаний обсяг сховища та операції з даними, що дозволяє оптимізувати витрати.
- Інтеграція з іншими сервісами GCP: Cloud Storage легко інтегрується з іншими сервісами Google Cloud, такими як Compute Engine, Kubernetes Engine, BigQuery та іншими.
- Різні класи сховища: Cloud Storage пропонує різні класи сховища для різних випадків використання, що дозволяє вибрати оптимальний варіант за ціною та продуктивністю. Наприклад, для даних, до яких потрібен швидкий доступ, підійде клас Standard, а для архівних даних — клас Archive.
- Керування життєвим циклом даних: Cloud Storage дозволяє автоматизувати процес переміщення даних між різними класами сховища залежно від їх віку та частоти доступу.
- Простота використання: Cloud Storage надає зручний веб-інтерфейс та інструменти командного рядка для управління даними.

Cloud Storage і Document AI тісно інтегровані, що дозволяє автоматизувати обробку документів, що зберігаються в хмарному сховищі. Document AI може безпосередньо отримувати доступ до документів, що зберігаються в Cloud Storage, без необхідності їх завантаження чи передачі. Це спрощує процес обробки та підвищує його ефективність [41].

Cloud Storage, об'єктне сховище, використовується для зберігання неструктурованих даних, таких як документи, зображення та інші файли, що можуть бути надані клієнтами або використані для навчання моделей машинного навчання [41]. Cloud Storage забезпечує безпечне та масштабоване сховище для таких даних, а також легку інтеграцію з іншими сервісами Google Cloud, такими як Document AI, що використовується для автоматичної

обробки документів.

AlloyDB – це повністю керований сервіс баз даних, що пропонується Google Cloud Platform, і є сумісним з PostgreSQL. Він розроблений для високої продуктивності та доступності, що робить його ідеальним вибором для вимогливих транзакційних додатків, включаючи ті, що використовуються в фінансовому секторі [42]. AlloyDB поєднує в собі переваги відкритого коду PostgreSQL з інноваціями Google в галузі розподілених систем та машинного навчання (рис. 2.12).

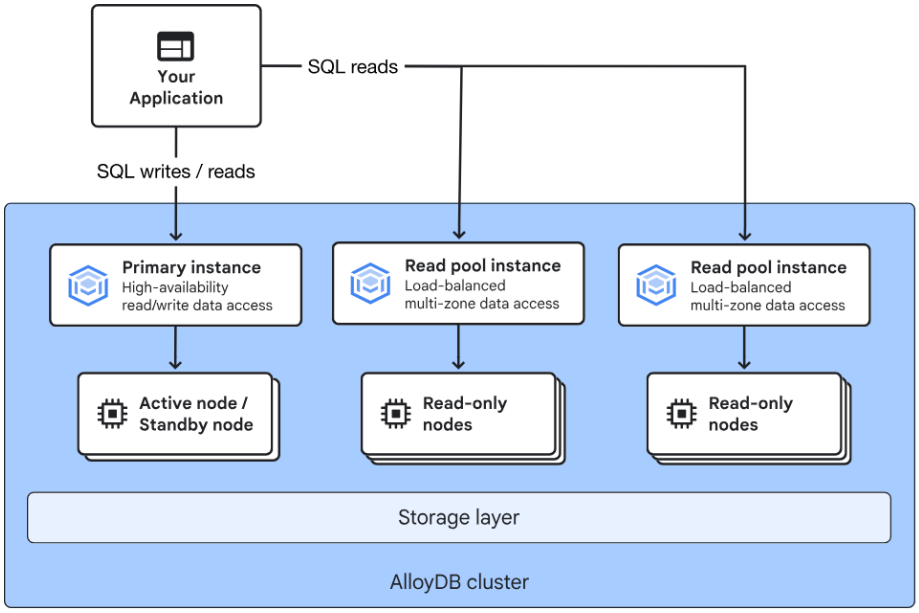


Рисунок 2.12 – AlloyDB

Ключові переваги AlloyDB:

- Висока продуктивність: AlloyDB забезпечує значно вищу продуктивність порівняно зі стандартним PostgreSQL, завдяки оптимізованій архітектурі та використанню технологій Google. Тести показують, що AlloyDB може бути до 4x швидшим для транзакційних навантажень та до 10x швидшим для аналітичних запитів.

- Сумісність з PostgreSQL: AlloyDB повністю сумісний з PostgreSQL, що дозволяє легко мігрувати існуючі додатки та використовувати знайомі інструменти та бібліотеки.
- Простота управління: AlloyDB — це повністю керований сервіс, тому вам не потрібно турбуватися про налаштування, оновлення, резервне копіювання та інші адміністративні завдання. Google Cloud бере на себе відповідальність за управління інфраструктурою бази даних.
- Безпека: AlloyDB інтегрується з іншими сервісами безпеки Google Cloud, забезпечуючи захист ваших даних.
- Масштабованість: AlloyDB дозволяє легко масштабувати ресурси бази даних залежно від ваших потреб.
- Висока доступність: Користувачі AlloyDB можуть бути впевнені у високій доступності своїх даних завдяки вбудованим механізмам резервного копіювання та відновлення.

AlloyDB, керована служба баз даних, сумісна з PostgreSQL, використовується для зберігання структурованих даних, таких як інформація про клієнтів, історія транзакцій, налаштування чат-бота та інші дані, що потребують швидкого та надійного доступу [42]. Висока продуктивність та масштабованість AlloyDB дозволяють ефективно обробляти велику кількість запитів від клієнтів та забезпечувати швидкий час відгуку чат-бота.

В сучасних системах обробки даних, особливо в таких галузях, як фінанси, де обсяги даних постійно зростають, використання ефективних інструментів для аналізу даних є критично важливим. В рамках розробки чат-бота для фінансових установ виникає потреба в аналізі великих обсягів даних, таких як історія діалогів з клієнтами, статистика використання сервісу, дані про фінансові операції та інше. BigQuery, хмарне сховище даних від Google Cloud Platform, було обрано для вирішення цих завдань завдяки його високій

продуктивності, масштабованості та гнучкості при аналізі великих обсягів даних. BigQuery дозволяє швидко отримувати аналітичні висновки з даних, що допомагає покращити якість обслуговування клієнтів, оптимізувати роботу чат-бота та виявляти нові можливості для розвитку бізнесу.

BigQuery — це повністю керований, безсерверний хмарний сервіс сховища даних, що пропонується Google Cloud Platform (рис. 2.13). Він призначений для аналізу величезних наборів даних за допомогою SQL-

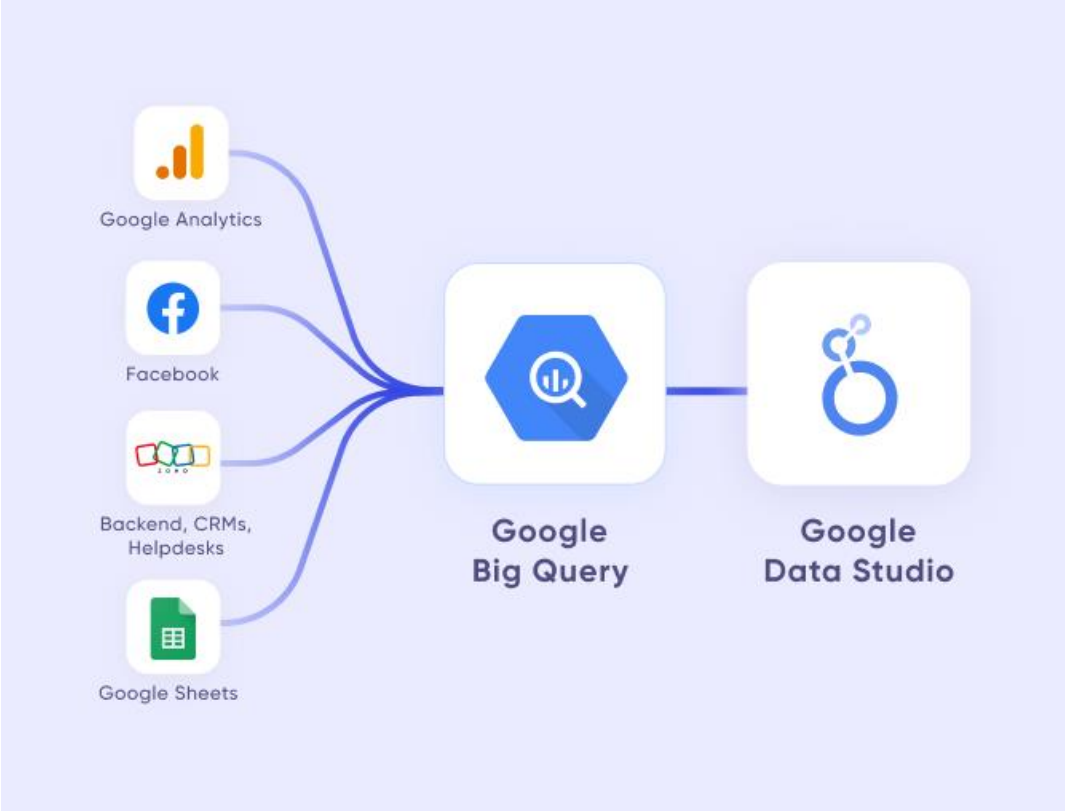


Рисунок 2.13 – BigQuery

подібної мови запитів. BigQuery пропонує високу швидкість обробки, можливість масштабування та економічну вигоду, що робить його оптимальним вибором для аналізу даних будь-якого розміру [19].

Основні можливості BigQuery:

- Масштабованість та продуктивність: BigQuery може обробляти петабайти даних за лічені секунди, використовуючи розподілену

архітектуру Google.

- SQL-подібна мова запитів: Ви можете використовувати стандартну SQL-подібну мову для запитів до даних, що спрощує аналіз та звітування [31].
- Безсерверна архітектура: Вам не потрібно турбуватися про управління інфраструктурою, BigQuery автоматично масштабується та керує ресурсами.
- Інтеграція з іншими сервісами GCP: BigQuery легко інтегрується з іншими сервісами Google Cloud, такими як Cloud Storage, Dataflow та Dataproc.
- Машинне навчання: BigQuery підтримує вбудовані функції машинного навчання, що дозволяють створювати та використовувати моделі ML безпосередньо в BigQuery.
- Безпека даних: BigQuery забезпечує безпеку ваших даних за допомогою шифрування та керування доступом.
- Економічна ефективність: Ви сплачуєте лише за обсяг оброблених даних та обсяг сховища.

2.8 Клієнтська частина на мові JavaScript

Nuxt.js, побудований на базі Vue.js, пропонує потужний та зручний інструментарій для створення універсальних веб-застосунків. Його ключова перевага полягає в спрощенні розробки завдяки функціям автоматичного розбиття коду, серверного рендерингу (SSR) та статичної генерації сайту (SSG). SSR покращує SEO та продуктивність, попередньо рендерячи сторінки на сервері, тоді як SSG дозволяє генерувати статичні HTML-файли для ще швидшого завантаження. Nuxt.js також надає доступ до багатої екосистеми Vue.js, що включає велику кількість плагінів та бібліотек. Простота

використання робить його привабливим вибором, особливо для розробників, знайомих з Vue.js. Проте, варто враховувати, що спільнота Nuxt.js менша за спільноту React/Next.js, а деякі особливості фреймворку можуть дещо обмежувати гнучкість [32].

Фреймворк Next.js, побудований на основі React, використовувався для створення інтерактивного та SEO-дружнього frontend інтерфейсу чат-бота [14]. Він пропонує можливість рендерингу як на сервері, так і на клієнті.

SvelteKit, побудований на базі Svelte, вирізняється своїм підходом до компіляції коду, що призводить до меншого розміру бандлу та високої продуктивності. SvelteKit також підтримує SSR та SSG, забезпечуючи швидке завантаження сторінок та гарну SEO-оптимізацію. SvelteKit пропонує зручний та інтуїтивно зрозумілий інтерфейс розробки. Хоча спільнота SvelteKit менша за спільноти React та Vue.js, вона швидко зростає. SvelteKit є перспективним фреймворком, який варто розглянути для створення швидких та ефективних веб-застосунків. Він добре підходить для проектів, де продуктивність є критично важливим фактором [33].

Next.js, як фреймворк на базі React, також зазнав впливу сучасних підходів до розробки frontend, зокрема компонентного підходу. Next.js, завдяки використанню компонентів React, забезпечує повторне використання коду, що спрощує процес розробки та підтримки програмного забезпечення. Для розробки frontend в Next.js використовується JavaScript (або TypeScript), а візуальний інтерфейс описується за допомогою JSX, що є розширенням синтаксису JavaScript, яке дозволяє вбудовувати HTML-подібні елементи в код JavaScript. Також використовуються стандартні CSS для стилізації компонентів.

Фреймворк Next.js активно розвивається як open-source проєкт, і його вихідний код доступний у репозиторії GitHub за посиланням: https://github.com/vercel/next.js.

Next.js надає розробникам такі переваги:

- Написання коду веб-застосунків за допомогою JavaScript або TypeScript, використовуючи сучасний стек технологій.
- Використання можливостей екосистеми React, зокрема великої кількості бібліотек, компонентів і розширень для створення динамічних додатків із високою продуктивністю.
- Легка інтеграція клієнтської та серверної частини програми завдяки підтримці серверного рендерингу (SSR) та статичної генерації (SSG).
- Використання Visual Studio Code або інших популярних IDE, які підтримують Next.js і мають розширення для прискорення розробки.

Ці переваги роблять Next.js ідеальним вибором для створення фронтенд-частини автоматизованої системи обслуговування клієнтів, оскільки він забезпечує гнучкість і швидкість розробки, а також підтримує масштабованість і інтеграцію з бекендом (рис. 2.14).

Функціонально на даний момент Next.js також поділяється на кілька ключових режимів роботи:

- Server-Side Rendering (SSR): дозволяє створювати серверні веб-додатки, де HTML-сторінки генеруються на сервері під час кожного запиту, забезпечуючи високу продуктивність і оптимізацію для пошукових систем.
- Static Site Generation (SSG): дозволяє створювати статичні сторінки, які генеруються під час збірки проєкту, що підходить для швидкого рендерингу контенту, який рідко змінюється.
- Client-Side Rendering (CSR): використовується для створення динамічних, інтерактивних односторінкових додатків, де більшість обчислень відбувається на стороні клієнта.

Ці функціональні можливості дозволяють гнучко комбінувати підходи залежно від потреб автоматизованої системи обслуговування клієнтів, забезпечуючи високу продуктивність, інтерактивність та оптимальний користувацький досвід.

У ході розробки фронтенд-частини автоматизованої системи обслуговування клієнтів фінансових установ використання **Next.js** є необхідним і достатнім для створення односторінкової інтерактивної програми. Next.js забезпечує оптимальну продуктивність, підтримку серверного рендерингу, а також має вбудовані засоби для роботи з маршрутизацією та API-запитами, що значно спрощує створення та підтримку

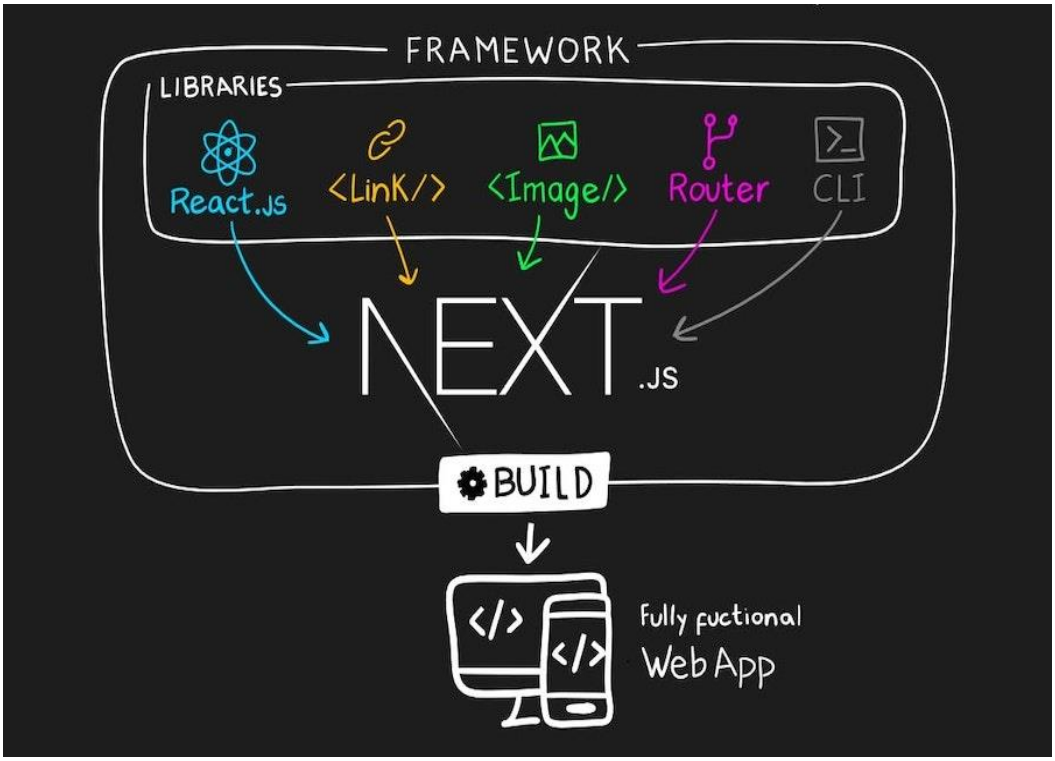


Рисунок 2.14 – Next.JS

складних веб-додатків.

Однією з ключових переваг Next.js є оптимізація завантаження додатків. Зокрема, під час збірки застосунку Next.js автоматично виконує tree-shaking — видаляє невикористаний код із JavaScript-модулів, що значно зменшує розмір

фінального пакету.

Крім того, Next.js підтримує автоматичне кешування статичних файлів і сторінок за допомогою заголовків HTTP, що дозволяє браузеру користувача зберігати необхідні ресурси для повторного використання, скорочуючи час завантаження та підвищуючи продуктивність застосунку.

Next.js підтримує режим **Static Site Generation (SSG)**, який дозволяє створювати повністю статичні веб-застосунки, що не залежать від сервера під час роботи. У цьому випадку всі необхідні файли — HTML, CSS і JavaScript — генеруються під час збірки й можуть бути розміщені на будь-якому статичному сервері або CDN.

Після завантаження статичних файлів браузером додаток працює автономно, виконуючи всю логіку на стороні клієнта. Це забезпечує високу продуктивність і мінімізує затримки, оскільки після початкового завантаження серверний компонент більше не використовується.

Next.js також має певні обмеження, які слід враховувати під час розробки. Наприклад, при використанні **Client-Side Rendering (CSR)** браузер виконує основну частину логіки застосунку, що може збільшити початковий час завантаження через необхідність завантаження великих JavaScript-файлів.

Крім того, робота застосунку залежить від можливостей браузера користувача, зокрема від підтримки сучасних функцій JavaScript. Незважаючи на те, що більшість популярних браузерів (Google Chrome, Mozilla Firefox, Opera, Microsoft Edge, Safari) підтримують необхідні стандарти, старіші версії можуть створювати обмеження.

Щоб мінімізувати ці проблеми, Next.js дозволяє комбінувати серверне рендеринг (SSR) і статичну генерацію (SSG), забезпечуючи швидке завантаження та оптимальну продуктивність навіть на повільних мережах і старіших пристроях.

Рішення з використання фреймворку **Next.js** також покриває всі потреби у створенні веб-додатків як на стороні сервера, так і на клієнта. Завдяки підтримці різних режимів рендерингу — серверного (SSR), статичної генерації (SSG) і клієнтського рендерингу (CSR) — Next.js забезпечує високий рівень гнучкості для розробки масштабованих і швидких веб-додатків.

Vercel, розробник Next.js, активно працює над вдосконаленням фреймворку, додаючи нові можливості та оптимізуючи його продуктивність. Оскільки Next.js інтегрується з React, його можна використовувати не тільки для веб-додатків, але й для створення мобільних додатків за допомогою React Native. Це робить Next.js перспективним інструментом для розробки повноцінних кросплатформених рішень, включаючи мобільні та десктопні додатки.

2.9 Система моніторінгу

У процесі створення автоматизованої системи для обслуговування клієнтів фінансових установ ключовим аспектом є забезпечення стабільної

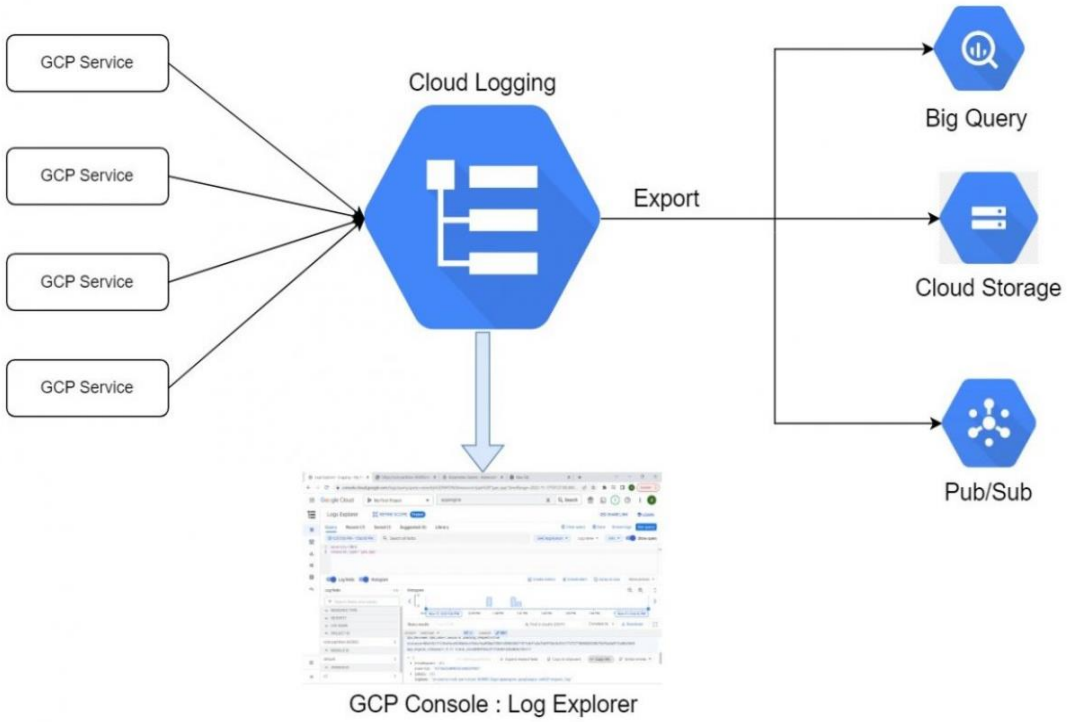


Рисунок 2.15 – Схема роботи Cloud Logging

роботи та швидкого виявлення потенційних проблем. Логування відіграє центральну роль у моніторингу, діагностиці та вдосконаленні таких систем, забезпечуючи розробників і адміністраторів необхідними даними про внутрішні процеси та взаємодію користувачів із системою.

Cloud Logging - це повністю керований сервіс на Google Cloud Platform, який дозволяє збирати, зберігати, аналізувати та відстежувати логи з різних джерел, включаючи додатки, віртуальні машини та інші сервіси Google Cloud [30]. Він надає централізоване місце для зберігання та аналізу логів, що спрощує моніторинг, усунення несправностей та аудит вашої інфраструктури та додатків (рис. 2.15).

Cloud Logging, як частина хмарної інфраструктури Google Cloud, надає потужні інструменти для збору, аналізу та управління логами. Цей сервіс дозволяє збирати логи з усіх компонентів системи — чат-бота, побудованого на основі Vertex AI, інтеграції з Document AI, бекенда FastAPI та фронтенда на базі Next.js. Завдяки Cloud Logging можна централізовано відстежувати запити клієнтів, час їх обробки, виникнення помилок і критичних подій, забезпечуючи ефективну підтримку та моніторинг.

Інтеграція логування в архітектуру системи дозволяє не лише забезпечувати високу якість обслуговування клієнтів, але й створює можливості для подальшого аналізу й удосконалення роботи чат-бота, виявлення «вузьких місць» і прогнозування можливих проблем [30]. Це робить логування невід'ємною складовою сучасних автоматизованих рішень для фінансових установ.

Основні можливості Cloud Logging:

- Збір логів: Підтримує автоматичне збирання логів з хмарних сервісів, таких як Compute Engine, Kubernetes Engine, App Engine тощо.
- Фільтрування: Дозволяє створювати фільтри для вибірки

потрібних логів за певними критеріями, наприклад, за рівнем серйозності (INFO, ERROR, WARNING).

- Моніторинг у реальному часі: Підтримує стрімінг логів для миттєвого аналізу роботи системи.
- Аналіз: Інтеграція з Google Cloud Operations Suite для створення дашбордів, виявлення аномалій та аналізу помилок.
- Збереження: Логи можуть зберігатися у хмарному сховищі для подальшого аналізу або перенаправлятися в системи BigQuery для більш складного аналізу.
- Сповіщення: Інтегрується з Cloud Monitoring для налаштування сповіщень про критичні події.

У межах автоматизованої системи для обслуговування клієнтів фінансових установ Cloud Logging може використовуватися для [30]:

1. Моніторинг роботи чат-бота:
   - Логування запитів клієнтів і відповідей бота для аналізу роботи та виявлення помилок у логіці.
   - Відстеження статистики використання, наприклад, найпоширеніших запитів чи пікових періодів навантаження.
2. Робота з бекендом FastAPI:
   - Логування запитів до API, часу їх виконання, помилок під час обробки даних.
   - Налаштування сповіщень при аномально високій кількості помилкових запитів.
3. Моніторинг інтеграції з Document AI та Vertex AI:
   - Логування кожного виклику до цих сервісів, часу відповіді, успішності виконання завдань.
   - Виявлення збоїв у роботі з зовнішніми сервісами, наприклад, через неправильні вхідні дані чи проблеми з мережею.

4. Фронтенд на Next.js:
    - Логування дій користувачів, наприклад, навігації між сторінками чи використання певних функцій.
    - Відстеження помилок JavaScript, які можуть впливати на роботу інтерфейсу.
5. Безпека та аудит:
    - Логування критичних подій, таких як спроби несанкціонованого доступу або зміни даних.
    - Аналіз логів для виявлення потенційних загроз чи атак на систему.

2.10 Vertex Pipelines

У сучасних автоматизованих системах для обслуговування клієнтів фінансових установ критично важливою є здатність інтегрувати та оркеструвати складні процеси машинного навчання (ML). Для створення високоякісного рішення необхідно забезпечити ефективну обробку даних, навчання моделей, їх тестування, розгортання та моніторинг. Vertex AI Pipelines (рис. 2.16), як частина Google Cloud, пропонує потужний

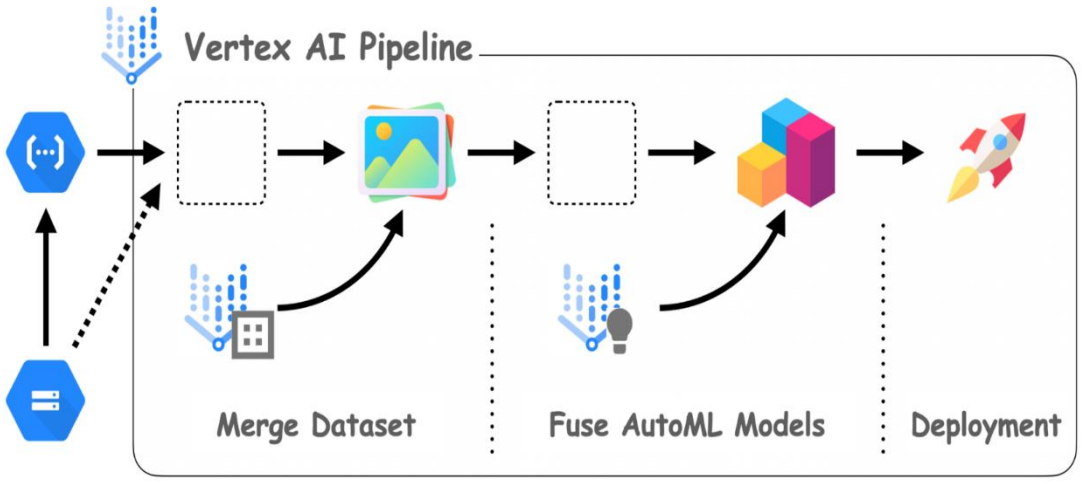


Рисунок 2.16 – Реалізація Vertex Pipeline

інструментарій для автоматизації цих процесів, забезпечуючи масштабованість і надійність [18].

Vertex AI Pipelines дозволяє розробляти й виконувати робочі процеси ML, забезпечуючи простоту інтеграції з іншими сервісами Google Cloud, такими як Vertex AI Model Training та Document AI. У рамках роботи системи, створеної для обслуговування клієнтів, цей інструмент використовується для автоматизації ключових етапів: підготовки даних, навчання моделей штучного інтелекту, їх оцінки та розгортання.

В контексті розробки чат-бота, Vertex Pipelines може бути використаний для автоматизації процесу навчання та розгортання моделі. Конвеєр буде включати такі кроки:

- Підготовка даних: Очищення, трансформація та підготовка даних для навчання моделі.
- Навчання моделі: Навчання моделі чат-бота на підготовлених даних за допомогою Vertex AI Training.
- Оцінка моделі: Оцінка продуктивності навченої моделі на тестових даних.
- Розгортання моделі: Розгортання навченої моделі на Vertex AI Prediction для використання в чат-боті.

2.11 Висновки

Цей розділ дослідження був присвячений вибору оптимального середовища розробки для створення автоматизованої платформи з надання фінансових послуг. В рамках проведеного аналізу було не тільки ідентифіковано та обґрунтовано вибір конкретного середовища, але й детально розглянуто проблеми та потреби кінцевих користувачів(як клієнтів, так і працівників), що безпосередньо впливають на вимоги до функціоналу та

доступності тих чи інших інструментів. Крім того, проведено комплексну оцінку сучасних технологій, що можуть бути застосовані в розробці, враховуючи як їх переваги, так і потенційні недоліки. Це дозволило сформувати обґрунтовану основу для подальших етапів проектування та реалізації, з урахуванням як технічних аспектів, так і потреб цільової аудиторії. Вибір середовища розробки, з огляду на проведений аналіз, забезпечить ефективність створення та подальшої підтримки мовної моделі.

3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Проєктування архітектури системи

Для забезпечення описаних функцій, автоматизована система обслуговування клієнтів фінансових установ може бути побудована на базі мікросервісної архітектури з використанням API для взаємодії між компонентами.

Виділимо два основних компоненти:

- Веб-застосунок з доступом до чат-боту.
- Платформа машинного навчання.

Веб-застосунок складається з бекенд(серверної) та фронтенд(клієнтської) частини, які взаємодіють між собою через API(REST). Ендпоїнт чат-боту у вигляді спеціального вікна знаходиться на клієнтській частині. Доступ до бази даних, у якій знаходяться дані користувача, реалізований на серверній частині. Також там є і доступ до платформи машинного навчання через спеціальний API-токен.

Платформа машинного навчання матиме вигляд «конвеєру» (рис. 3.2), де на вхід буде потрапляти документ(або скановене зображення), а на виході буде мовна модель з новими натренованими даними, зв’язок з якою буде через ендпоїнт у вигляді чат-боту. Увесь конвеєр автоматизований, і такі процеси, як обробка тексту, збір даних, збереження їх на хмарному сховищі, передача на тренування, розгортання моделі відбуваються автоматично.

Рисунок 3.2 – Конвеєр платформи машинного навчання

3.2 Побудова серверної частини

Серцем backend-частини нашої системи стане FastAPI — сучасний, високопродуктивний веб-фреймворк для створення API на Python. Ми обрали FastAPI за його вражаючу швидкість, яка є критично важливою для забезпечення швидкої реакції системи та обробки запитів від користувачів у режимі реального часу. FastAPI спрощує розробку, пропонуючи інтуїтивно зрозумілий синтаксис та вбудовану підтримку асинхронного програмування. Важливою перевагою є автоматична генерація документації API за допомогою Swagger/OpenAPI, що значно спрощує інтеграцію з іншими сервісами та полегшує роботу розробників.

Хоча FastAPI є відносно молодим фреймворком, його стрімка популярність, висока продуктивність та активна спільнота роблять його ідеальним вибором для нашого проекту. Звісно, існують альтернативи, такі як Flask та Django, які також написані на Python. Проте, Flask, будучи більш простим фреймворком, може виявитися недостатньо функціональним для наших потреб. Django, навпаки, є надто "важким" фреймворком, що орієнтований на розробку складних монолітних додатків, тоді як ми прагнемо створити легку та гнучку мікросервісну архітектуру.

Спочатку потрібно зробити хендлери для серверної частини. Саме

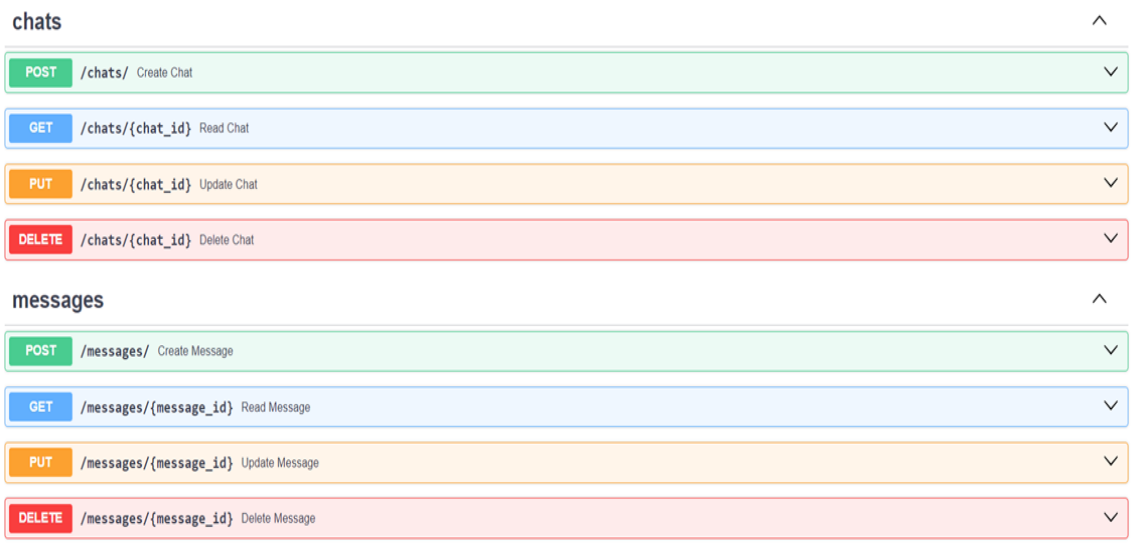


Рисунок 3.3 – Автоматична документація запитів

завдяки FastAPI можна отримати автоматичну документації по усім доступним запитам (рис. 3.3), у якому вигляді передавати дані (рис. 3.4), та у якому вигляді їх отримувати (рис. 3.5). Було розроблено схеми запитів та відповідей для таких сутностей як: користувач, документ, чат, повідомлення. Також для них були реалізовані CRUD-операції: створення, отримання, редагування ти видалення об’єктів.

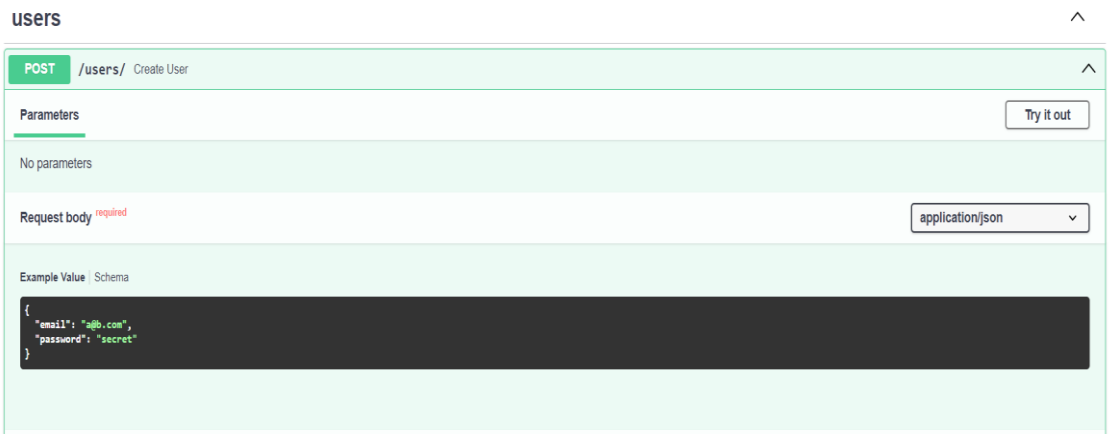


Рис. 3.4 – Схема запиту

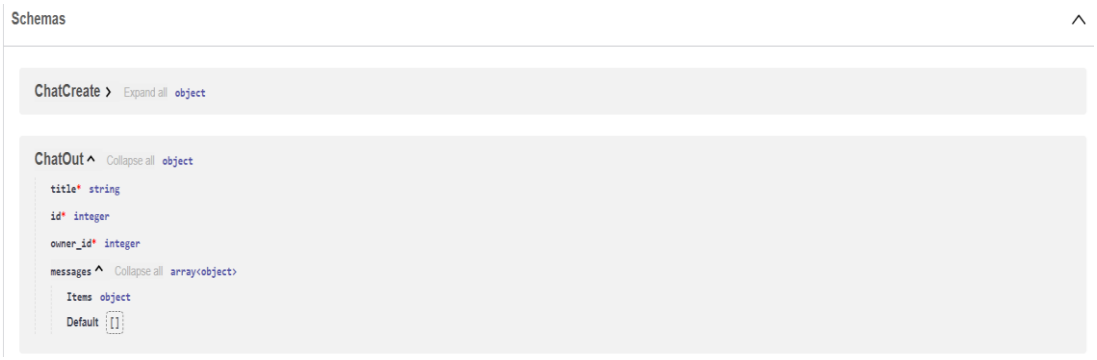


Рисунок 3.5 – Схема відповіді

```
You, last month | 1 author (You) | Codeium: Refactor | Explain
class UserBase(SQLModel):
    email: EmailStr = Field(index=True, unique=True)
    name: str = Field()
    is_active: bool = Field(default=True)


You, last month | 1 author (You) | Codeium: Refactor | Explain
class User(UserBase, table=True):
    __table_args__ = (UniqueConstraint("email", name="uq_user_email"),)
    id: int | None = Field(default=None, primary_key=True)
    hashed_password: str
    chats: list["Chat"] = Relationship(back_populates="owner")
    messages: list["Message"] = Relationship(back_populates="owner")
    documents: list["Document"] = Relationship(back_populates="owner")
```

Рисунок 3.6 – Схема таблиці User

На серверній частині ми описуємо схеми сутностей, те як вони будуть в базі даних, і те як вони виглядати в API. Наприклад, таблиця User матиме наступний вигляд (рис. 3.6).

Нас цікавить як виглядає безпосередньо структура база даних.

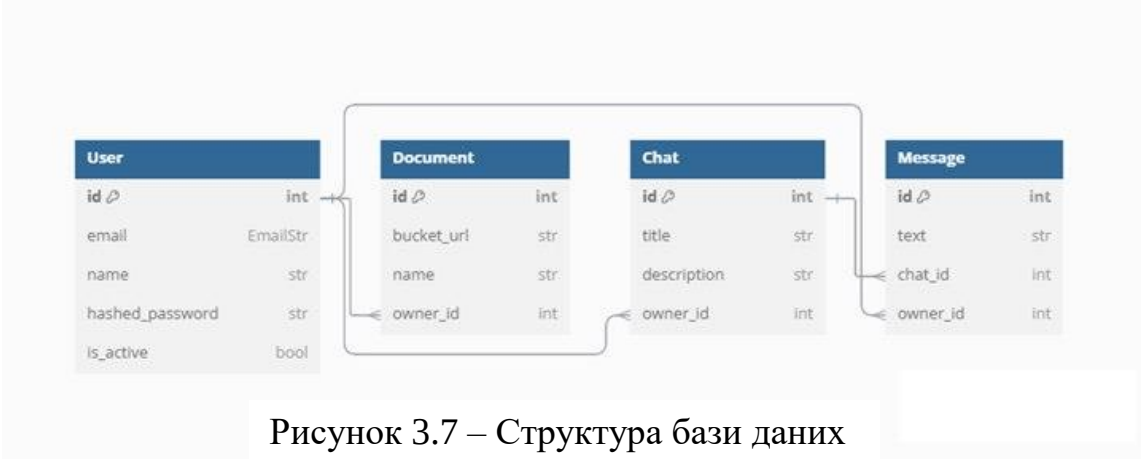


Рисунок 3.7 – Структура бази даних

Зобразимо її на рисунку 3.7:

3.3 Створення хмарних сховищ

Для початку створимо кластер AlloyDB (рис. 3.8). У нас є змога обрати регіон, де буде знаходитись сервер кластеру, тип машини, тобто к-сть

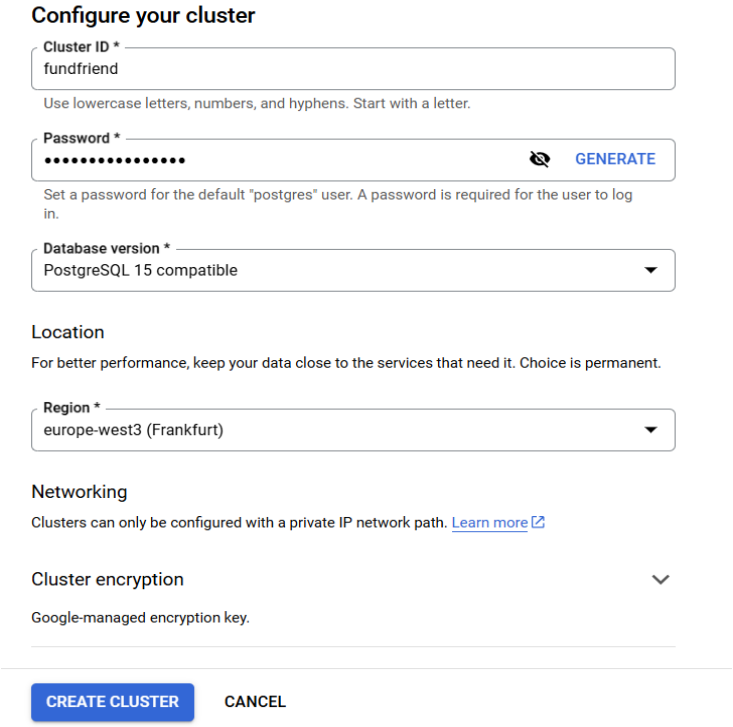


Рисунок 3.8 – Конфігурація AlloyDB

процесорів та обсяг оперативної пам’яті, а також версію СУБД PostgreSQL.

Після того, як було створено кластер, з’являється змога легко керувати ним. Можна переглядати графіки по різним параметрам (рис. 3.9). І так само можна отримати статистику по запитам до інстансів.

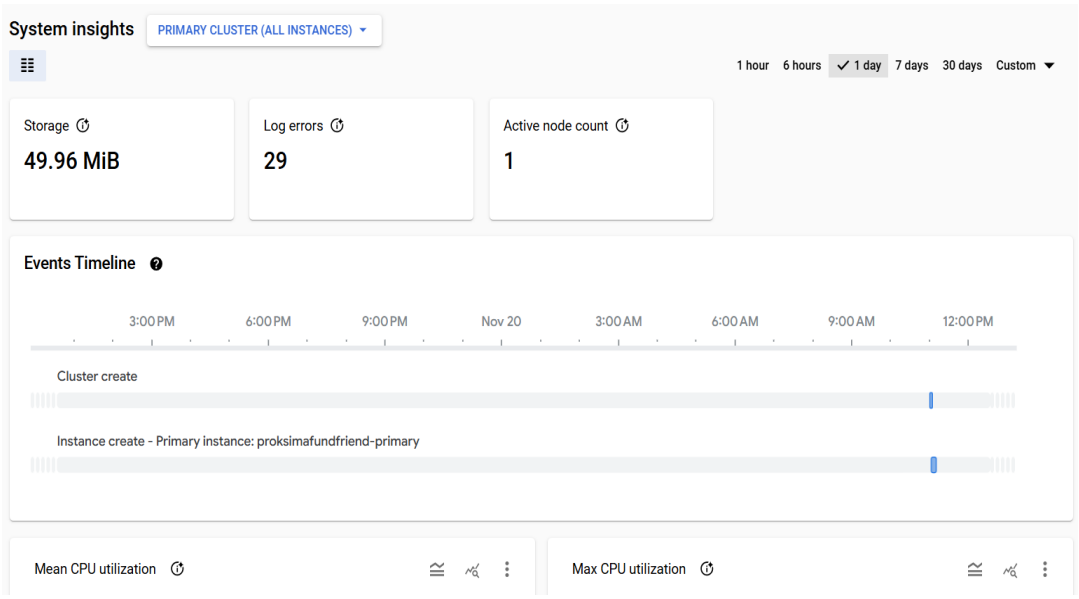


Рисунок 3.9 – Графіки використання ресурсів

У чому ще перевага кластеру AlloyDB для моєї роботи? Оскільки я застосовую мікросервісну архітектуру, то якраз в одному кластері AlloyDB я можу створити декілька інстансів PostgreSQL.

Також AlloyDB має такий сервіс, як AlloyDB Studio, що допомагає переглядати інстанси одразу у GCP (рис. 3.10).

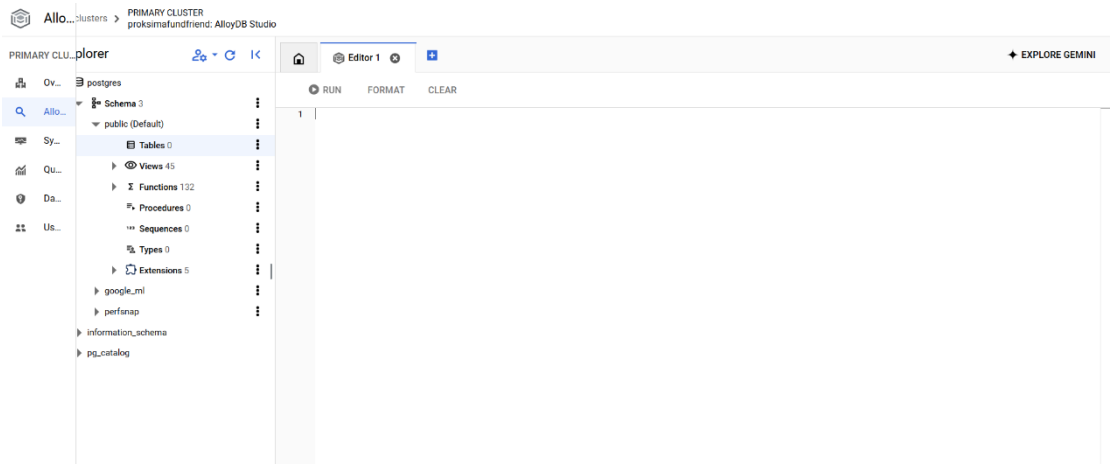


Рисунок 3.10 – Вигляд AlloyDB Studio

І, наприклад, можна створити таблиці як через ORM, так і через «студію», використовуючи мову запитів SQL (рис. 3.11). Одразу ж тут відбувається валідація запитів, та змога створювати запити за допомогою ШІ.

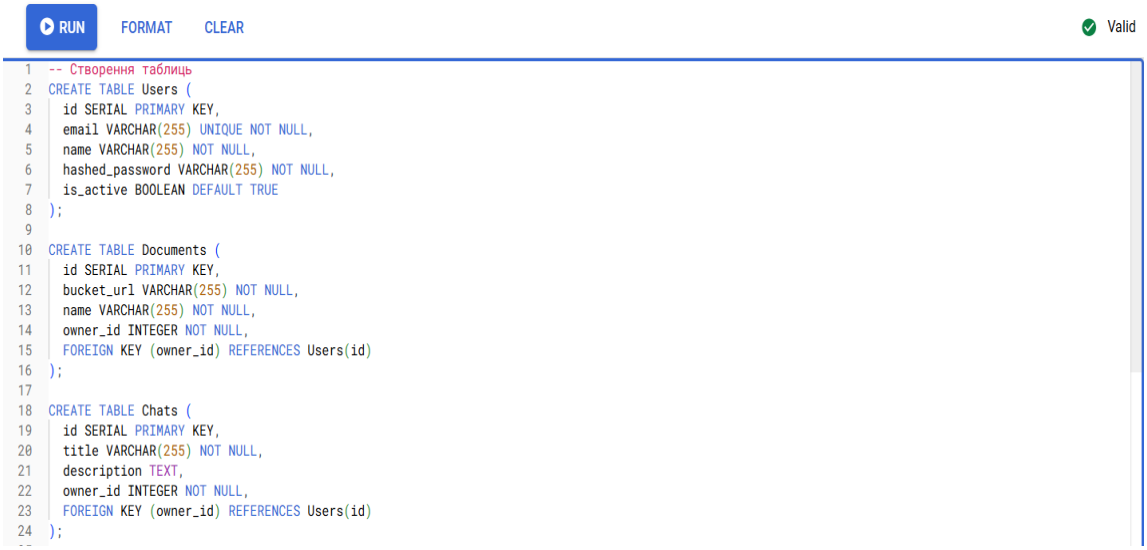


Рисунок 3.11 – SQL Editor

Тому в кінцевому результаті будемо мати дані таблиці (рис. 3.12).

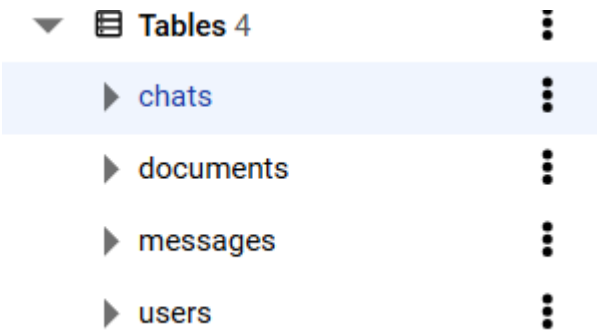


Рисунок 3.12 – Список таблиць в БД

І ось такі графіки, після тестових запитів до інстансу (рис. 3.13). Це в майбутньому допоможе оптимізувати запити та переглядати, чи впливатимуть зміни на продуктивність.

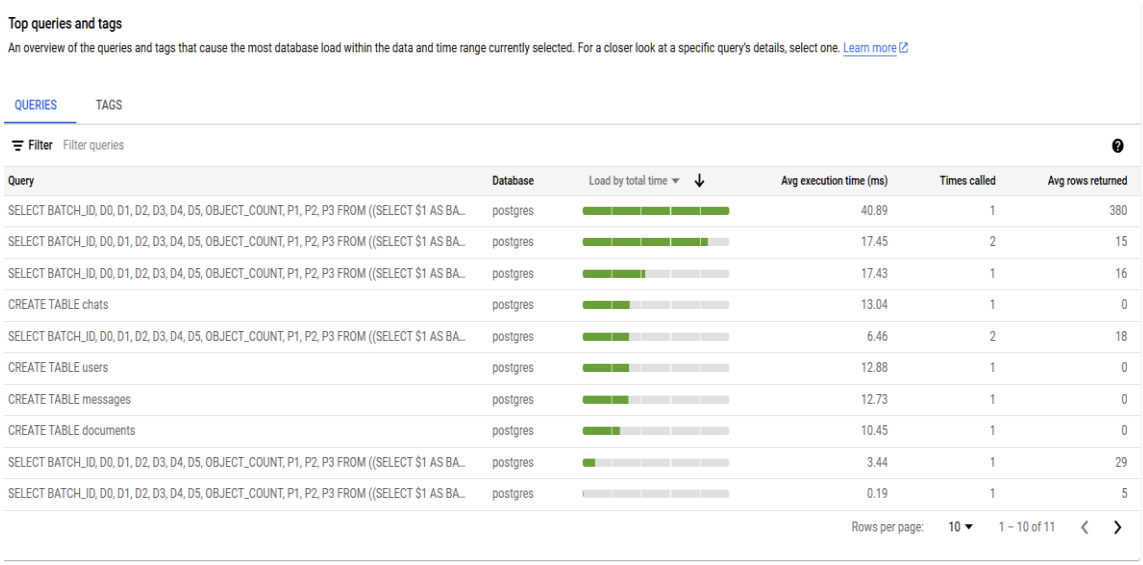


Рисунок 3.13 – Статистика по запитам до інстансу

Тепер для того, щоб були дані для навчання моделі після оброки текстових документів, їх потрібно десь зберігати. І тут найкраще якраз підійде не база даних, а сховище даних, оскільки воно дасть змогу зберігати файли та швидкий і надійний доступ до них. Також «бакети» дуже добре поєднуються з іншими сервісами Google Cloud Platform.

Під час створення «бакету» є опція обрати регіон знаходження сховища і також налаштування доступу до інформації (рис. 3.14).

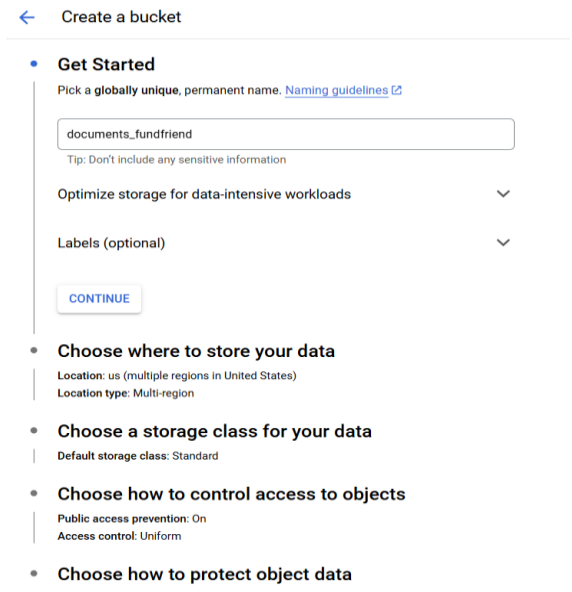


Рисунок 3.14 – Створення бакету

Додавши перші файли, сховище буде мати наступний вигляд (рис. 3.15).

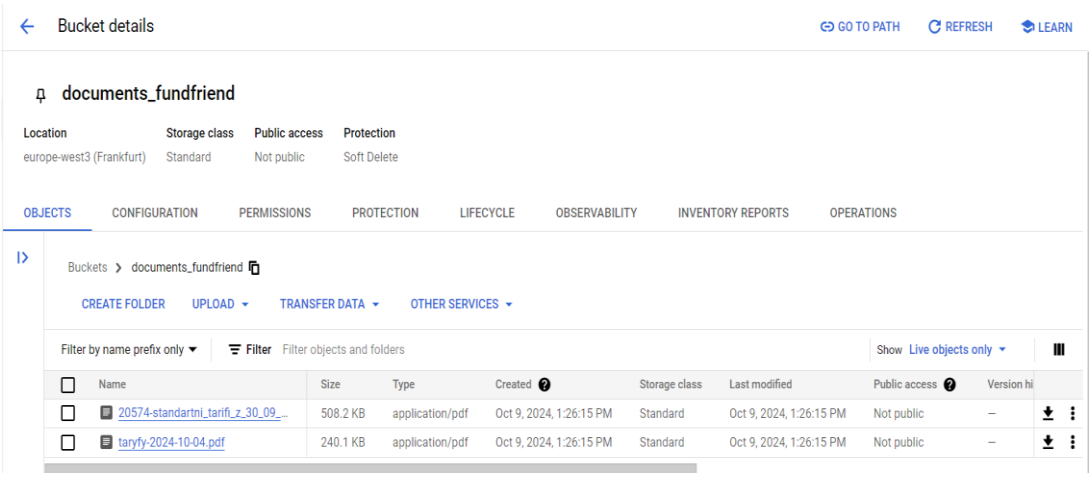


Рисунок 3.15 – Файли у сховищі

3.4 Побудова клієнтської частини

Фронтенд частина – це більше про те, як користувач бачить свою взаємодію з системою. За допомогою Next.js розробляється структура сайту у вигляді дерева (рис. 3.16), де кожна сторінка є підкоренем початкової точки.

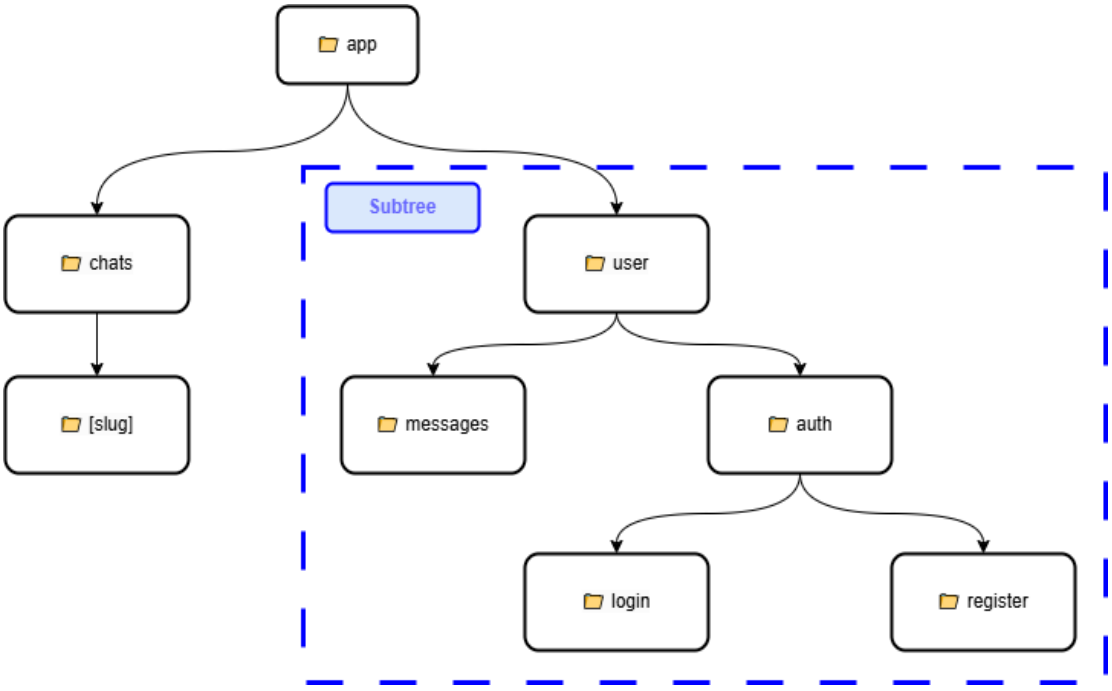


Рисунок 3.16 – Структура фронтенд частини

Давайте детальніше розберемо дану структуру:

- Дерево — це угода для візуального представлення ієрархічних структур, таких як дерево компонентів з батьківськими та дочірніми елементами або структура папок.
- Піддерево починається з вузла, який вважається його коренем, і включає всі його нащадки аж до листів.
- Вершиною ієрархії в дереві або піддереві є корінь, прикладом якого може бути кореневий макет.
- Вузли без нащадків у піддереві називаються листками. Прикладом може слугувати останній сегмент URL-адреси.

І тепер наглядніше це можна побачити на структурі URL-адреси (рис.3.17).

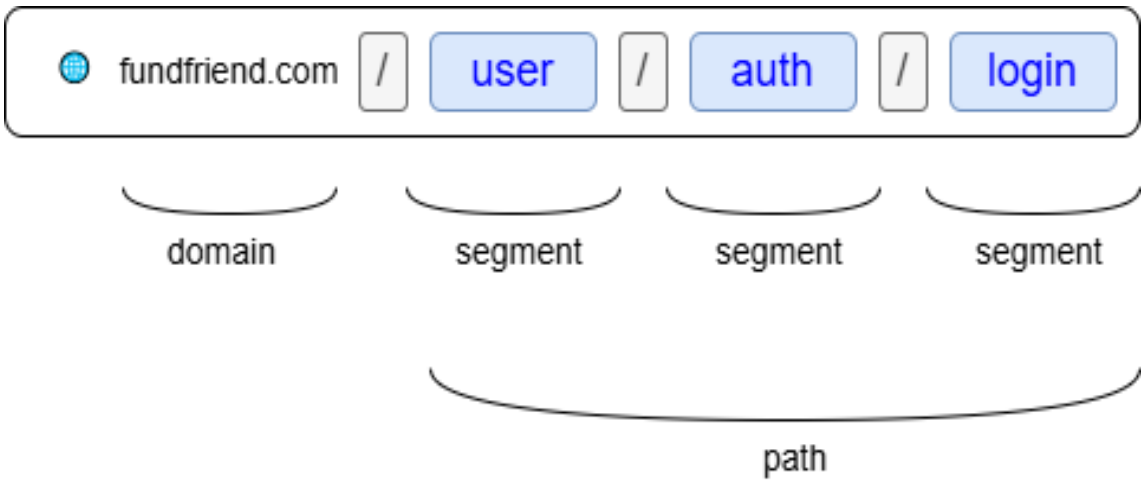


Рисунок 3.17 – Структура url-адреси

Усі сторінки в додатку пишуться у вигляді компонент. Тут використовується синтаксис JSX, що дозволяє писати HTML у JavaScript (рис. 3.18). Це полегшує розробку. Також можна користуватись хуками – це такі допоможні методи, що дозволяють керувати станом сторінки на стороні клієнту. Але за допомогою Next.js можна вказувати точно, де повинно, наприклад, відбуватись підвантаження даних. Тобто можна опитимізувати таким чином сторінки і зробити їх завантаження швидшим.

```
import { useState, useEffect } from 'react';
import Link from 'next/link';

Codeium: Refactor | Explain | Generate JSDoc | ×
const DocumentPage = () => {
  const [documents, setDocuments] = useState([]);

  useEffect(() => {
    Codeium: Refactor | Explain | Generate JSDoc | ×
    const fetchDocuments = async () => {
      const response = await fetch('/api/documents');
      const data = await response.json();
      setDocuments(data);
    };
    fetchDocuments();
  }, []);

  return (
    <div className="container mx-auto p-4 pt-6 mt-10">
      <h1 className="text-3xl font-bold">Документи</h1>
      <ul>
        {documents.map((document) => (
          <li key={document.id} className="py-2">
            <Link href={`/documents/${document.id}`}>
              <a className="text-blue-500 hover:text-blue-700">{document.title}</a>
            </Link>
          </li>
        ))}
      </ul>
    </div>
  );
};


export default DocumentPage;
```

Рисунок 3.18 – Вигляд компоненти для сторінки документів

Також створимо клас для взаємодії із АПІ. Для цього можна застосувати патерн Repository. Патерн "Репозиторій" (Repository) — це дизайн-патерн, який використовується для ізоляції логіки доступу до бази даних або іншого джерела даних від бізнес-логіки програми. Основна ідея полягає в тому, щоб

надати рівень абстракції між додатком і джерелом даних, що полегшує

```
You, 2 seconds ago | 1 author (You) | Codeium: Refactor | Explain
class DocumentRepository {
  Codeium: Refactor | Explain | Generate JSDoc | ×
  async getAll() {
    const { data, error } = useSWR("/api/documents", fetcher);
    if (error) {
      throw error;
    }
    return data.map(
      (document) =>
        new Document(document.id, document.title, document.bucket_url)
    );
  }

  Codeium: Refactor | Explain | Generate JSDoc | ×
  async getById(id) {
    const { data, error } = useSWR(`/api/documents/${id}`, fetcher);
    if (error) {
      throw error;
    }
    return new Document(data.id, data.title, data.content);
  }
```

Рисунок 3.19 – Репозиторій для документів

підтримку, тестування та модифікацію коду. На прикладі «репозиторію» для документів (рис. 3.19) можна побачити як реалізуються методи. Також тут використовується бібліотека SWR. SWR (Stale-While-Revalidate) — це стратегія кешування і бібліотека для React, яка допомагає ефективно працювати з даними, забезпечуючи їхню актуальність і продуктивність. Її назва походить від HTTP-директиви stale-while-revalidate, яка дозволяє використовувати кешовані дані, поки нові дані отримуються у фоновому режимі. За допомогою з’являються такі можливості, що пришвидшують процес завантаження сторінки на стороні клієнту:

- Автоматичне оновлення кешу.
- Підтримка фокусування (оновлення даних при поверненні у вікно).

- Інтеграція з серверними API (REST, GraphQL).
- Вбудована підтримка TypeScript.
- Можливість використання з React Suspense.

Тому основним результатом є вікно чату, де користувач може задавати питання до чат-боту:

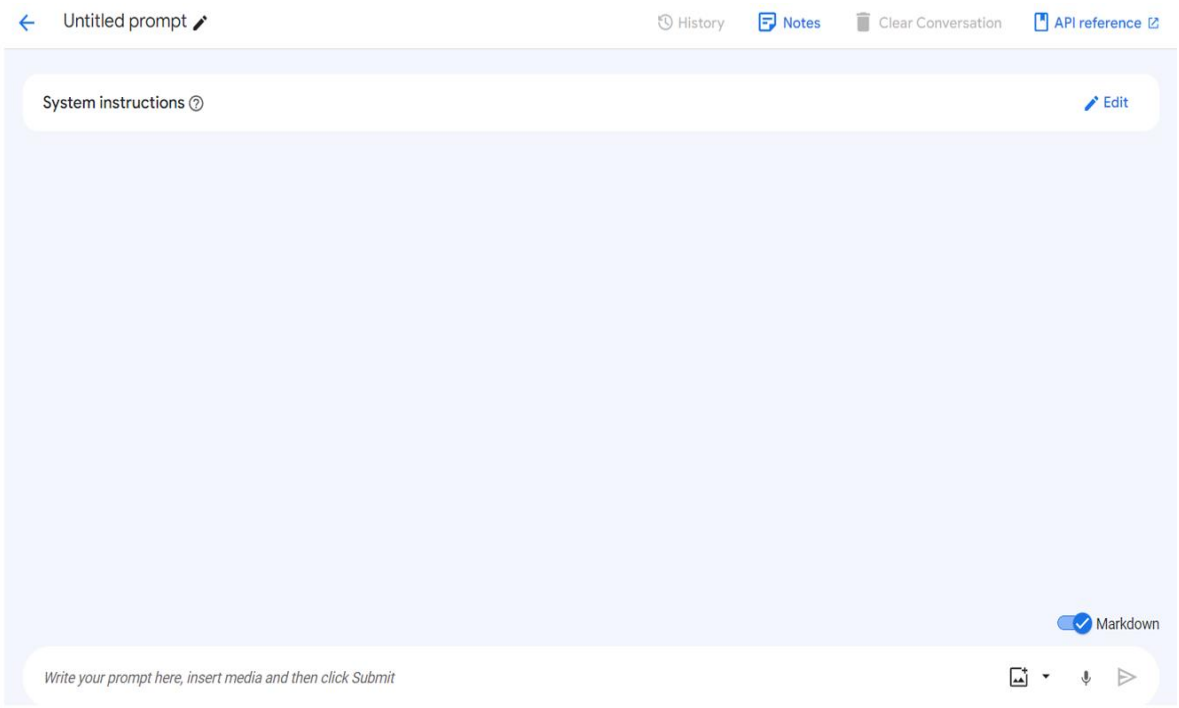


Рисунок 3.20 – Сторінка розмови з чатом

3.5 Створення текстового процесору

Тепер перейдімо до реалізації нашого конвеєру. Оскільки сховища ми

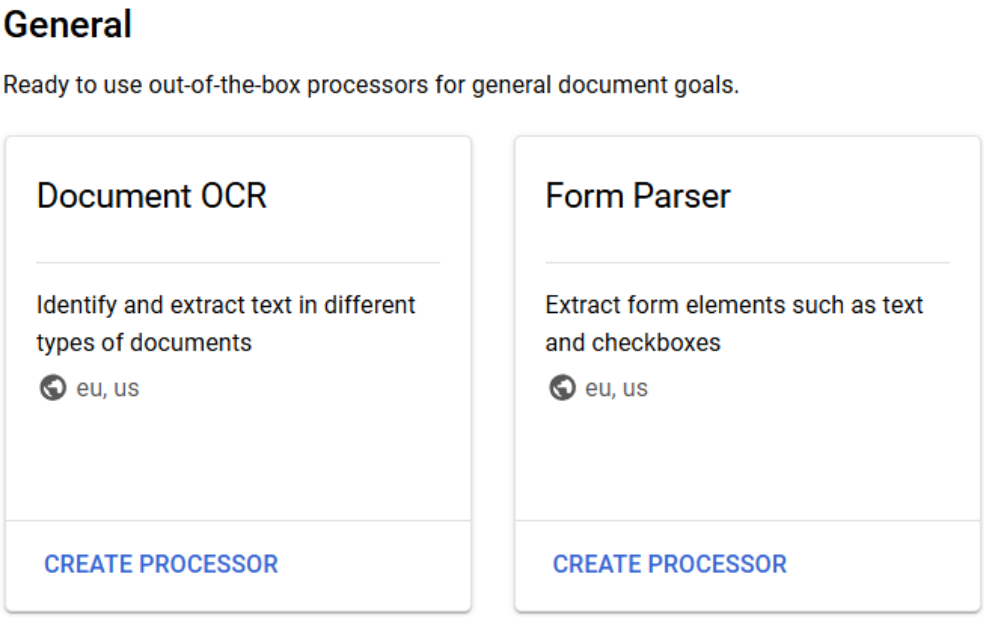


Рисунок 3.21 – Набір шаблонів для текстових процесорів

вже маємо, тепер можна створити текстові процесори, які будуть брати файли, та завантажувати результати. На вибір існує два типи передготових процесорів (рис. 3.21). Document OCR – це технологія оптичного розпізнавання тексту,

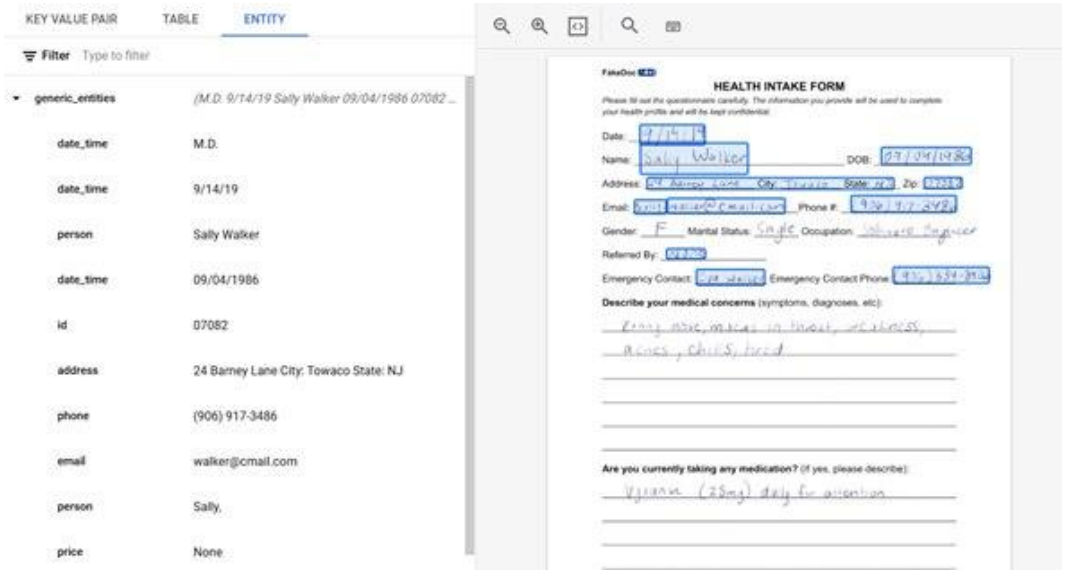


Рисунок 3.22 – Тестовий запуск процесора

яка перетворює зображення (скани документів, фотографії) у текст [22]. Використовується для базового витягування тексту з будь-яких зображень, включаючи документи, листи, чи навіть рукописні тексти. Form Parser – це покращений інструмент OCR, який не лише витягує текст, але й розуміє структуру документа. Використовується для роботи з формами, таблицями чи структурованими документами, де важливо зберегти та інтерпретувати контекст текстових полів. Найкращою опцією буде обрати Form Parser, оскільки він зможе згрупувати дані, наприклад, інформацію про клієнта.

Після створення процесора можна завантажити текстовий документ та подивитись на результат. Як видно на рисунку 3.22, процесор дуже добре знайшов пари ключ-значення та згрупував їх по сутності(entity).

Також цей тест показує, що процесор прекрасно впорюється і з рукописним текстом. Також процесор може повертати значення просто як таблицю усього тексту, що він знайшов (рис. 3.23).

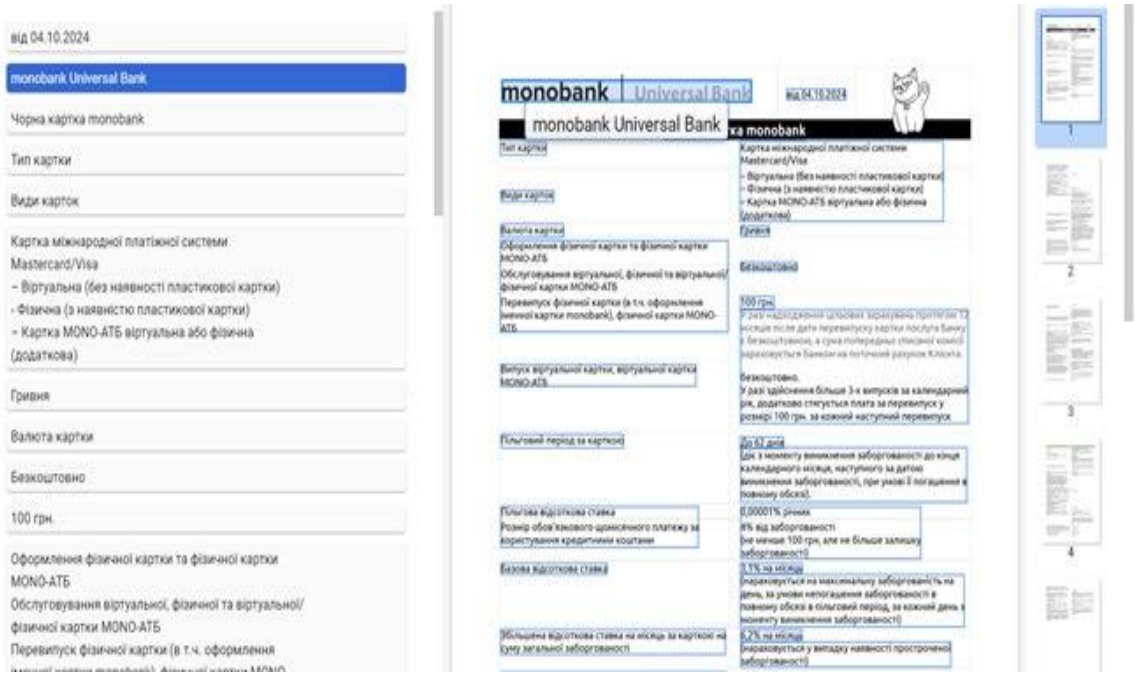


Рисунок 3.23 – Збір усіх даних у табличному вигляді

3.6 Навчання мовної моделі

Після цього найголовніший етап – створення та навчання моделі. GCP вже надає готові мовні моделі Gemini для того, щоб мати максимальні результати точності при розробці власної. На вибір надаються їхні найновіші на даний момент моделі: 1.5-flash, 1.5-pro, 1.0-pro. У чому різниця між Pro та Flash? Перша далі розлогі відповіді й при цьому витрачає більше часу на відповідь. Flash при цьому ж дає короткі, лаконічні відповіді за короткий проміжок часу. Оскільки модель буде призначатись для спілкування з клієнтами у фінансових установах, час грає визначну роль, тому обираємо Gemini 1.5-Flash [26]. Після цього звичайно ж потрібно обрати дані для навчання. Тобто це та інформація, що й так вже буде поверх всіх тих даних, що містить у собі модель Flash. У нашому випадку - це створений у

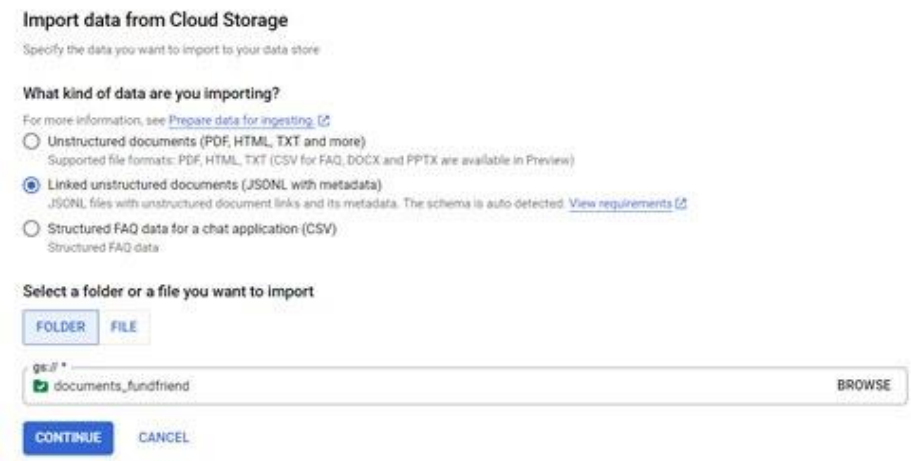


Рисунок 3.24 – Вибір джерела навчальних даних

попередньому етапі бакет з документами (рис. 3.24).

І тут знову ми отримуємо одну з переваг використання Document AI, а саме те, що після обробки даних він повертає дані в типі JSONL. Це формат файлів, який містить дані у вигляді окремих об’єктів JSON, розділених символами нового рядка. Це зручний формат для зберігання великих обсягів структурованих даних, особливо для обробки потоків даних. Також цей тип дозволяє обробляти і найголовніше змінювати файл частинами, не передававши всі дані у потік інформації. Від нас вимагається тільки вписати

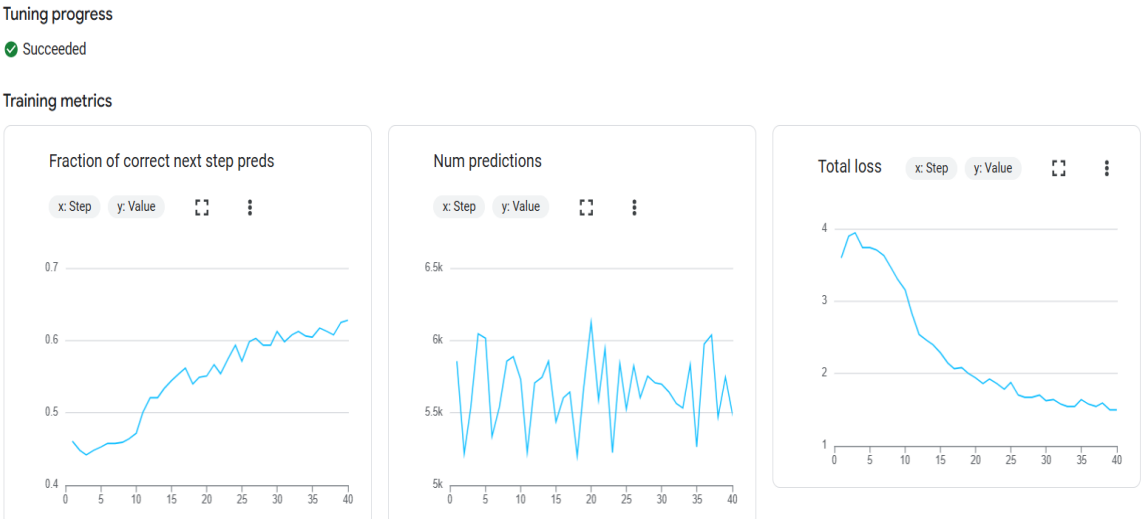


Рисунок 3.25 – Графіки результатів тренування моделі

шлях до Cloud Storage, і система сама валідує, чи є вказаний шлях

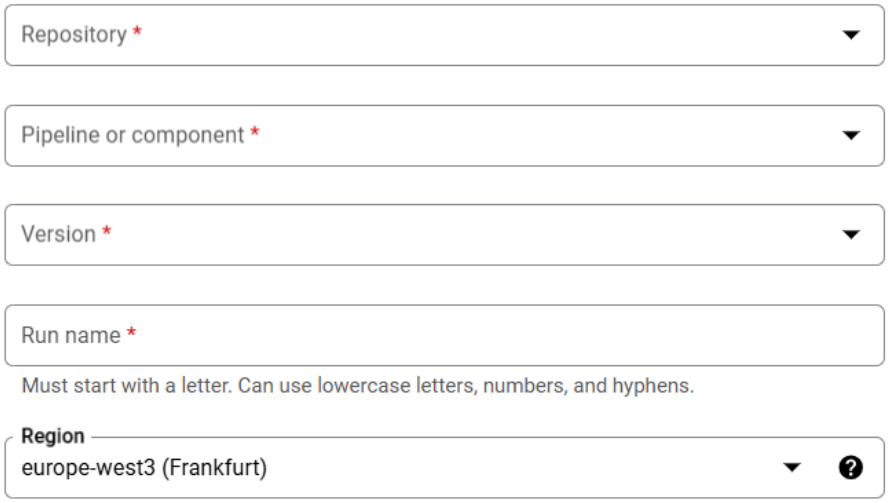


Рисунок 3.26 – створення Vertex Pipeline

правильним. Після деякого часу (це тільки перший раз, коли модель буде навчатись) ми отримуємо кінцеву точку доступу для «спілкування» з моделлю, яке потім можна вставити до нашого веб-додатку у клієнтській частині.

Також ми тут можемо отримати графіки, наскільки «добре» навчилась модель, наскільки успішно пройшла валідація (рис. 3.25).

3.7 Створення автоматизованого конвеєру

У підсумку все це потрібно поєднати у Vertex AI Pipeline. Це і буде нашим конвеєром, що буде автоматизовувати усі процеси [23]. При створенні

Input Parameters

Name	Type	Description
location	string	Location for running the Cross-validation trainer.
project	string	Project to run Cross-validation trainer.
root_dir	string	The Cloud Storage location to store the output.
encryption_spec_key_name	string	Customer-managed encryption key.

Output Parameters

Name	Type	Description
gcp_resources	string	GCP resources created by this component. For more d https://github.com/kubeflow/pipelines/blob/master/c cloud/google_cloud_pipeline_components/proto/REAI

Рисунок 3.27 – Параметри компоненти

пайплайну, потрібно вказувати, які компоненти будуть використовуватись (рис. 3.26).

І також при цьому пізніше потрібно налаштувати як часто буде працювати пайплайн (рис. 3.29). У нашому випадку не буде якогось визначеного розкладу, оскільки як тільки з’являється новий нормативний документ, то модель повинна одразу навчитись на новій інформації, що їй

надасть текстовий процесор. Також консоль надає змогу побачити компоненти

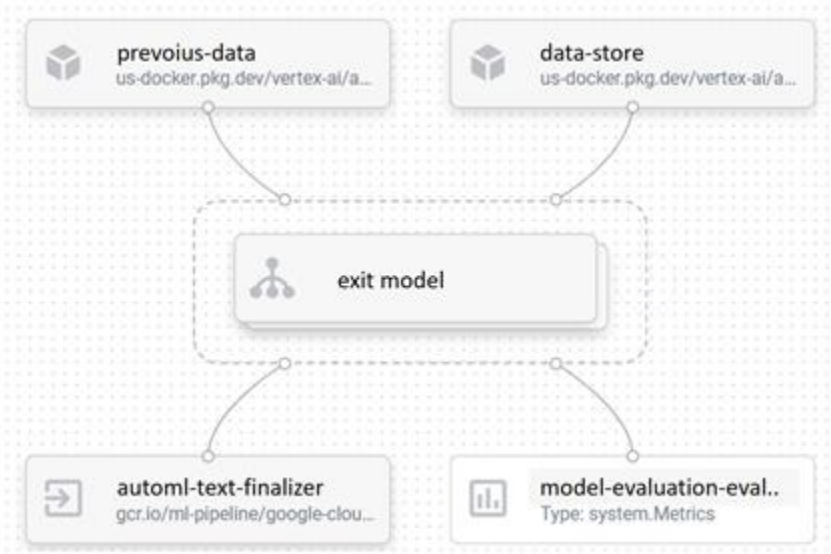


Рисунок 3.28 – Схема пайплайну

у графічному представленні (рис. 3.28). Тут можна переглянути як інформацію по всьому пайплайну, так і по окремій ноді(вузлі) (рис. 3.27). Як видно на рисунку 3.28, усі попердньо описані етапи сходяться до вихідної мовної

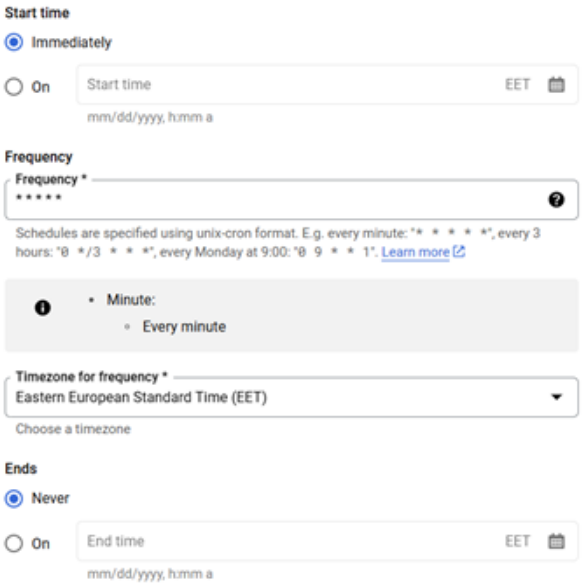


Рисунок 3.29 – Розклад запуску пайплайну

моделі: сховище даних, автоматично класифіковані дані зі сховища, попередні

дані, і відповідно саме навчання.

На кожну компоненту вказується процес, що потрібно виконати. Вибір сервісів представляє сам GCP. Також він надає список елементів, що отримуються на вхід до компоненти, та на вихід з неї.

```python
# Визначення пайплайну
Codeium: Refactor | Explain | Generate Docstring | ×
@pipeline(name="train-model-pipeline", pipeline_root=f"gs://{BUCKET_NAME}/pipeline-artifacts")
def train_model_pipeline(project: str = PROJECT_ID, location: str = REGION):
    # Крок 1: Запуск кастомного тренування
    train_task = gcc_aip.CustomTrainingJobRunOp(
        project=project,
        location=location,
        display_name="train-model",
        container_uri="us-docker.pkg.dev/vertex-ai/training/tf-cpu.2-8:latest",
        command=["python3", "train.py"],
        args=[
            "--data-uri", TRAINING_DATA_URI,
            "--model-dir", "/model"
        ],
        model_display_name=MODEL_DISPLAY_NAME,
        staging_bucket=f"gs://{BUCKET_NAME}",
    )

    # Крок 2: Експорт моделі
    upload_model_task = gcc_aip.ModelUploadOp(
        project=project,
        display_name=MODEL_DISPLAY_NAME,
        artifact_uri=train_task.outputs["model_output_path"],
        serving_container_image_uri="us-docker.pkg.dev/vertex-ai/prediction/tf2-cpu.2-8:latest",
    )
    return upload_model_task

# Компіляція пайплайну
compiler.Compiler().compile(
    pipeline_func=train_model_pipeline,
    package_path="train_model_pipeline.json"
)
```

Рисунок 3.30 – Програмний спосіб реалізації пайплайну

Також дані етапи можна налаштувати і за допомогою мови програмування, наприклад, Python. Для цього потрібно встановити SDK, яке дасть змогу викликати сервіси GCP програмним способом. Після цього власне пройти аутентифікацію в GCP. Для цього скористаємось інтерфейсом командного рядку(CLI) gcloud.

І після цього вже можна будувати власний пайплайн у вигляді функції. Наприклад, етап навчання моделі буде виглядати як на рисунку 3.30.

3.8 Програмування чат-бота

Після того, як ми створили конвеєр із навчанням моделі, потрібно

створити чат-бота. Чому це необхідно? На даному етапі модель вже може

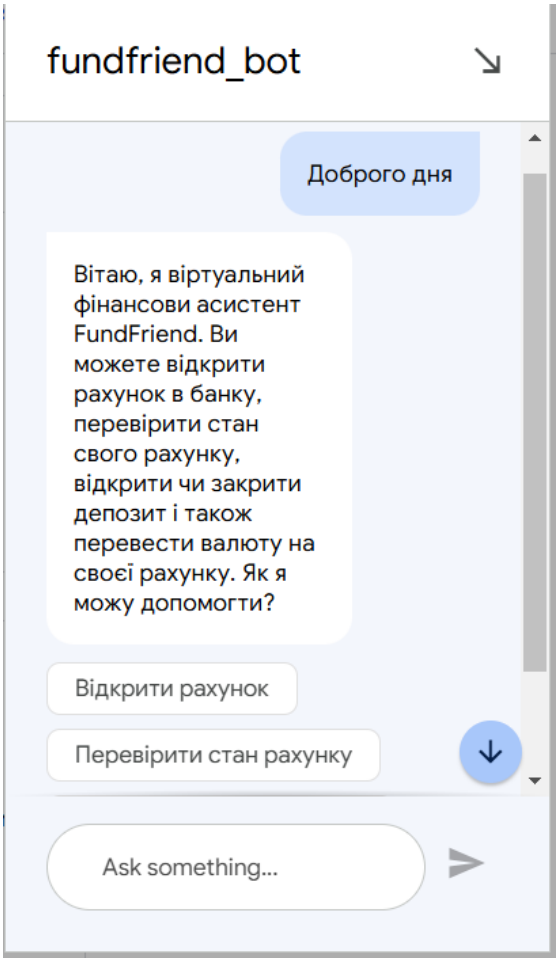


Рисунок 3.31 – Вигляд стартової сторінки

спілкуватись із користувачем та відповідати на питання відповідно до

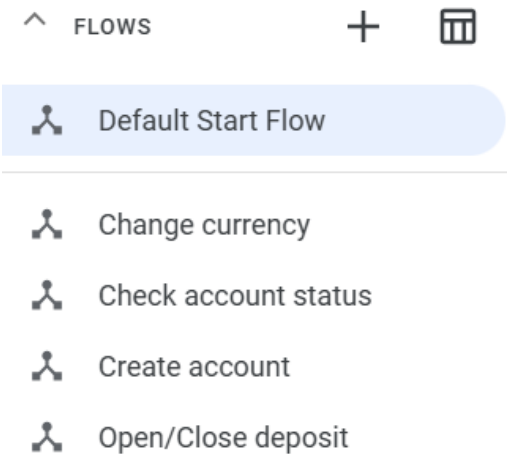


Рисунок 3.32 – Потоки чат-бота

нормативних документів. Але, наприклад, створити рахунок, відкрити/закрити депозит, обміняти валюту мовна модель не може. Тому для цього й потрібно розробити чат-бота. З цим допоможе платформа Dialogflow. Dialogflow — це платформа від Google Cloud для створення чат-ботів, голосових помічників та інших систем обробки природної мови. Вона дозволяє розробникам створювати інтерфейси (рис. 3.31) на основі природної мови для взаємодії з користувачами через текст або голос [24].

Складні діалоги часто включають кілька тем для розмови. У випадку з чат-ботом, якого ми створюємо, це різні фінансові питання. Ми можемо розділити ці теми на потоки.

Потоки - це нова концепція для Dialogflow CX [34]. Потоки дозволяють чат-боту працювати над індивідуальними шляхами розмови. Давайте створимо декілька потоків (рис. 3.32). Кожен потік має свої вузли, які мають налаштування на те як сприймати інформацію, і як відповідати (рис. 3.33).

До речі, сам месенджер можна інтегрувати у будь-яку систему: Slack, Microsoft Teams, Facebook Messenger.

Розмову в Dialogflow CX (сеанс) можна описати та візуалізувати як кінцевий автомат. Візьмемо для прикладу торговий автомат, який можна змоделювати як автомат зі скінченними станами. Він має наступні стани: Чекає на монети, Вибирає цукерки, Видає цукерки, і, маючи набір вхідних даних, він рухається між цими станами. Наприклад, вкидання монети

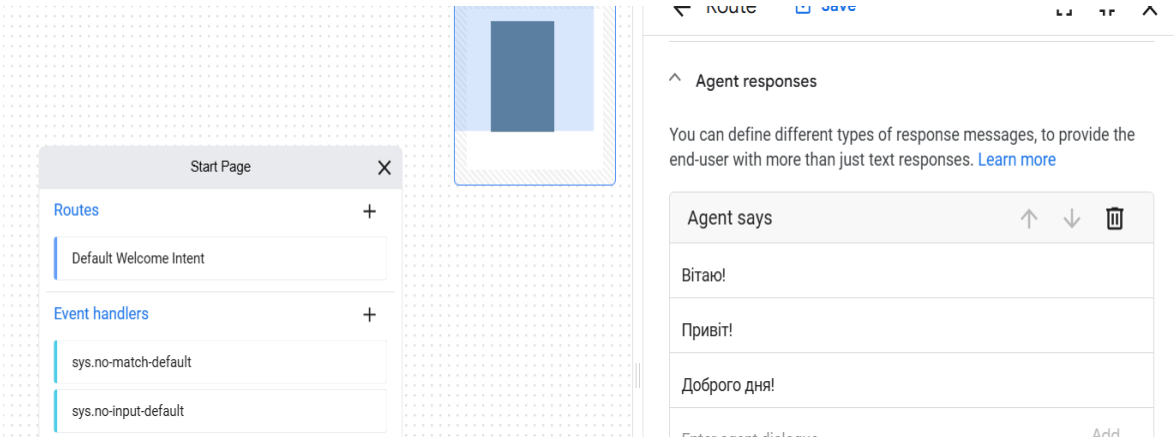


Рисунок 3.33 – Налаштування вузла

переводить автомат зі стану «Чекає на монети» у стан «Вибирає цукерки». Сторінки - це спосіб моделювання цих станів для віртуального агента Dialogflow CX.

Коли кінцевий користувач взаємодіє з Dialogflow CX під час розмови, розмова переходить зі сторінки на сторінку, тому в будь-який момент часу саме одна сторінка є поточною, поточна сторінка вважається активною, а також потік, пов'язаний з цією сторінкою, вважається активним.

Намір(Intent) класифікує наміри кінцевого користувача на один хід діалогу. У Dialogflow CX їх було значно спрощено, вони більше не є будівельним блоком для керування діалогом. Dialogflow CX використовує наміри лише для того, щоб відповідати тому, що говорять користувачі. У Dialogflow ES вам потрібно було прив'язувати все до намірів (параметри, події, виконання і т.д.). Наміри в Dialogflow CX містять лише навчальні фрази і тому можуть використовуватися багаторазово. Вони більше не керують розмовою. Тож процес створення намірів буде простим. Наприклад, процес створення наміру «credit_card.check_balance» виглядатиме наступним чином

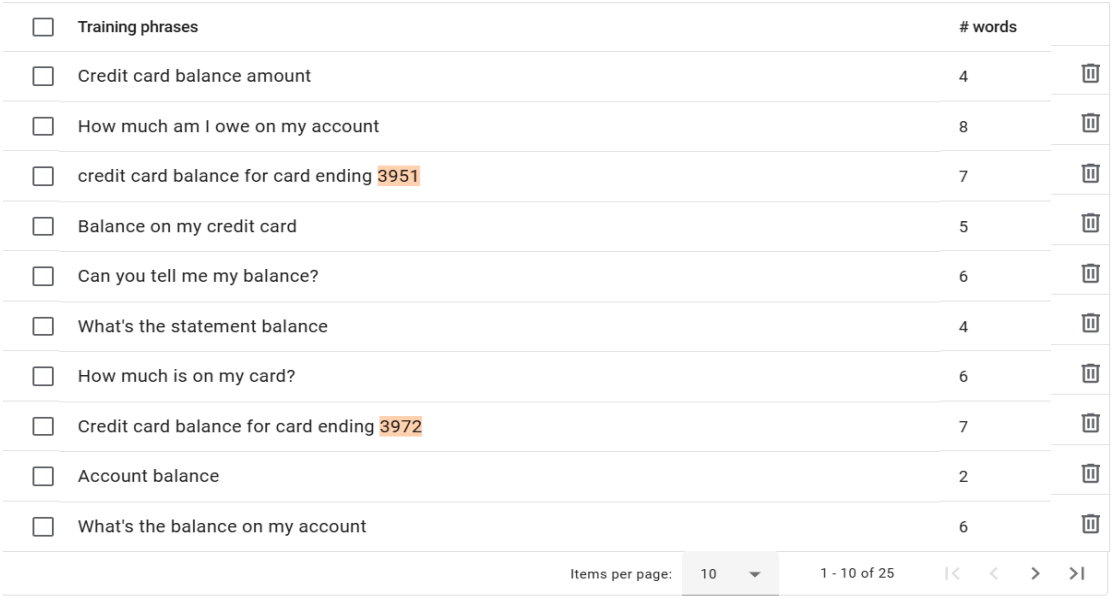


Рисунок 3.34 – Створення наміру

(рис. 3.34). Ми вказуємо тренувальні фрази(будь-якої кількості), по яким би

хотіли, щоб чат-бот розпізнав намір. Як видно з рисунку, деякі частини фраз підсвідчуються. Це платформа автоматично визначає сутності. Типи сутностей використовуються для керування тим, як витягуються дані з даних, введених кінцевим користувачем. Типи сутностей Dialogflow CX дуже схожі на типи сутностей Dialogflow ES. Dialogflow надає заздалегідь визначені системні

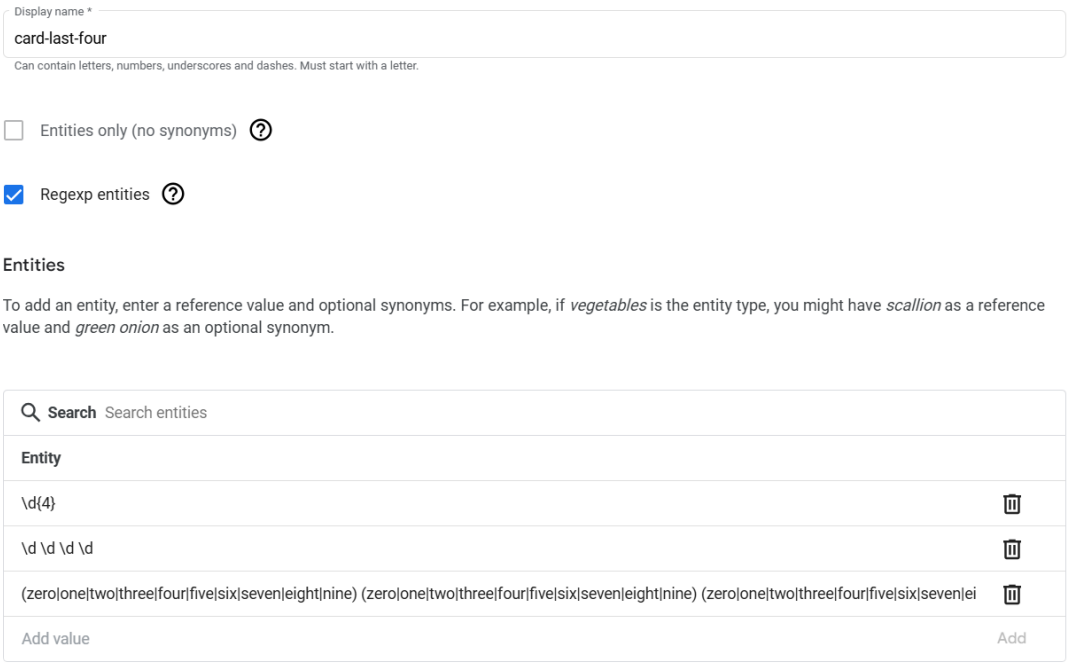


Рисунок 3.35 – Вигляд сутності

сутності, які можуть відповідати багатьом поширеним типам даних. Наприклад, існують системні сутності для зіставлення дат, часу, кольорів, адрес електронної пошти тощо. Ви також можете створювати власні користувацькі сутності для зіставлення користувацьких даних. У даному випадку сутність «card-last-four» має наступний вигляд, як на рисунку 3.35.

Для кожного потоку ви визначаєте багато сторінок, на яких разом можна вести повноцінну розмову на тему(и), для якої(их) призначений потік. Кожен потік має спеціальну стартову сторінку. Коли потік стає активним, стартова сторінка стає поточною сторінкою. Для кожного повороту розмови поточна сторінка або залишається незмінною, або переходить на іншу сторінку. Ця концепція дозволить вам створювати більші агенти з багатьма сторінками і

декількома ходами розмови.

Сторінки містять виконання (статичні діалоги введення та/або веб-хуки), параметри та обробники станів. Керування діалогом відбувається через обробники станів, що дозволяє створювати різні маршрути переходу на іншу сторінку Dialogflow CX, в тому числі робити його умовним (для розгалуження діалогів).

У кінцевому результаті ми будемо мати такі потоки (рис. 3.36):

- Валідація облікового запису.
- Перевірити баланс.
- Провести платіжну перевірку.
- Статус заявки на кредит.
- Здійснити платіж.
- Звернутися до агента.

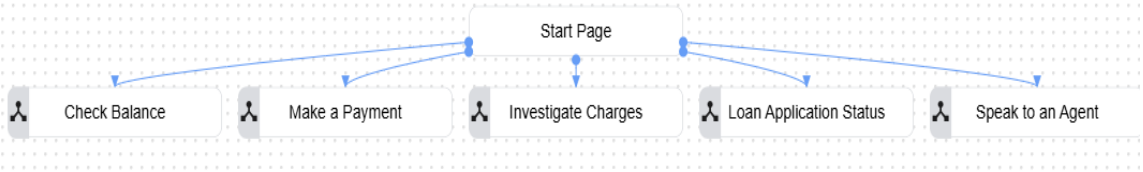


Рисунок 3.36 – Схема потоків у чат-боті

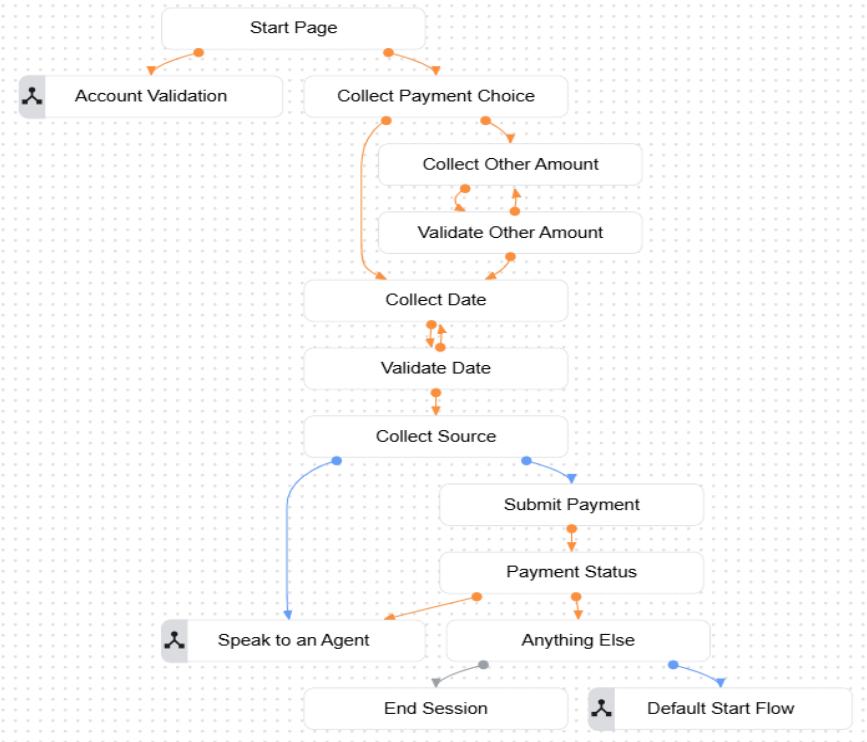


Рисунок 3.37 – Потік по проведенню платежів

Кожен потік можна розглянути окремо, а також використовувати інші потоки по декілька разів. Наприклад, у потоці по проведенню платежів (рис. 3.37) використовується потік «Account Validation». Він ж використовується і у перевірці платежів, балансу.

3.9 Налаштування системи логування

Як не дивно, це одна з найлегших частин розробки. Оскільки Google Cloud Project сам передає значнення для логування з різних сервісів у центр лог-повідомлень. Журнали платформи отримуються з хмарних сервісів Google і надійно зберігаються без необхідності налаштування. Наприклад, Журнали робочих навантажень GKE збираються автоматично, а Ops Agent збирає журнали робочих навантажень з віртуальних машин. Журналювання в хмарі інтегровано з хмарним моніторингом, повідомленнями про помилки та хмарним трасуванням, щоб ви могли усунути несправності в усіх ваших службах.

Logs Explorer дозволяє шукати, сортувати та аналізувати журнали за допомогою гнучких запитів, а також візуалізації гістограм, простого провідника полів і можливості зберігати запити. Налаштуйте оповіщення, щоб сповіщати вас про появу певного повідомлення у включених журналах, або скористайтеся хмарним моніторингом, щоб отримувати сповіщення за визначеними вами метриками на основі журналів.

Наприклад, на backend частині було реалізовано middleware для логування запитів і відповідей від серверу. «Проміжне програмне забезпечення»(middleware) - це функція, яка працює з кожним запитом до того, як він буде оброблений будь-якою конкретною операцією шляху. А також з кожною відповіддю, перш ніж повернути її. Ініціалізувавши клієнт від Cloud Logging, код буде наступним (рис. 3.38). Сама платформа надає зручні графіки для перегляду різних повідомлень з можливістю фільтрувати по

```python
async def log_middleware(request: Request, call_next):
    start_time = time.time()

    # Extract details from the request
    request_body = await request.body()
    request_data = {
        "method": request.method,
        "url": str(request.url),
        "query_params": dict(request.query_params),
        "body": request_body.decode("utf-8") if request_body else None,
    }

    response = await call_next(request)

    duration = time.time() - start_time

    log_dict = {
        "method": request.method,
        "url": str(request.url),
        "status_code": response.status_code,
        "duration": duration,
        "request": request_data,
        "response_status_code": response.status_code,
    }

    if response.status_code >= 400:        You, 2 weeks ago • 59-implemen
        logger.error(json.dumps(log_dict, indent=2))
    else:
        logger.info(json.dumps(log_dict, indent=2))

    return response
```

Рисунок 3.38 – Налаштування логування запитів і відповідей

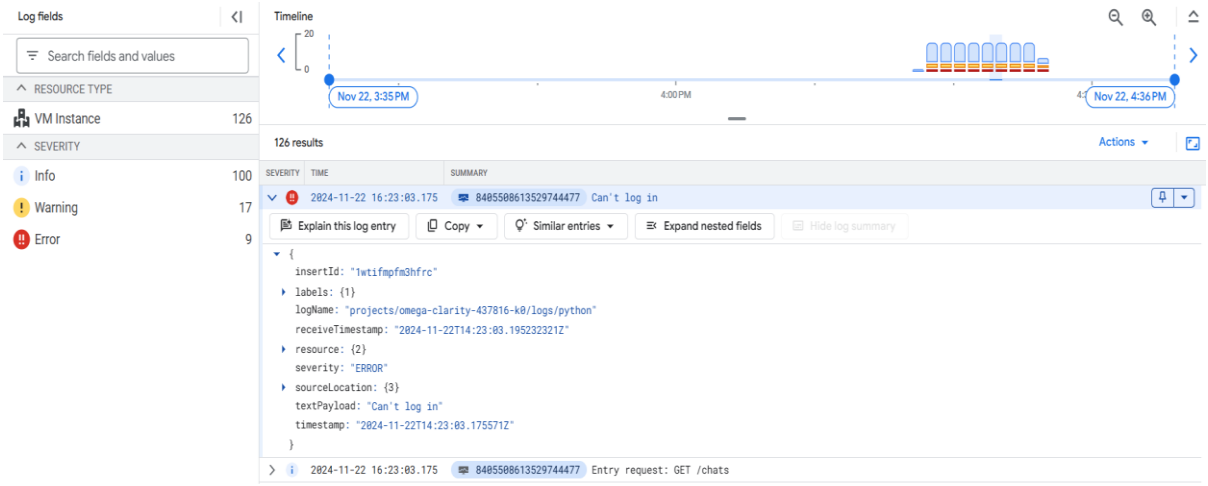


Рисунок 3.39 – Log explorer

джерелу. Можна як обирати окремі сервіси від GCP, так і загальні повідомлення. Нас цікавить вкладка «Logs explorer» (рис. 3.39). Перегляд і аналіз окремих записів журналу і послідовності записів журналу може допомогти усунути неполадки. Щоб виконати агреговані операції над записами журналу, наприклад, підрахувати кількість записів журналу, які

містять певний шаблон – все це можна зробити за допомогою користувацького інтерфейсу в Logs Explorer. На рисунку 3.39 видно повідомлення, які показують статус запитів, їхній вміст та частину URL.

3.10 Тестування програмного продукту

У процесі розробки продукту замовники та розробники часто стикаються з різними проблемами та причинами помилок, які можуть бути неочевидними на ранніх стадіях процесу. Це лише підкреслює, наскільки важливий процес «тестування програмного забезпечення». Зрозуміло, що чим пізніше почнеться тестування програмної системи, тим більше ризиків воно несе, і тим менш надійною вона може вийти.

Тестування програмного забезпечення — це технологічний процес, який перевіряє систему або продукт на наявність помилок або невиконання вимог і стандартів.

Існує безліч різних типів тестування, які суттєво відрізняються за своєю метою та очікуваним результатом після завершення. Їх техніка та завдання, які вони вирішують, відрізняються. Стандарти якості, які перевіряються за їх допомогою, визначають, як їх можна класифікувати.

Для перевірки функціональності програмного забезпечення використовуються власні функціональні тести, а також тести обсягу, безпеки та інших.

У даній роботі проект повинен мати зручний графічний інтерфейс сайту для клієнта, месенджер, який дозволяє запитувати про функціональні

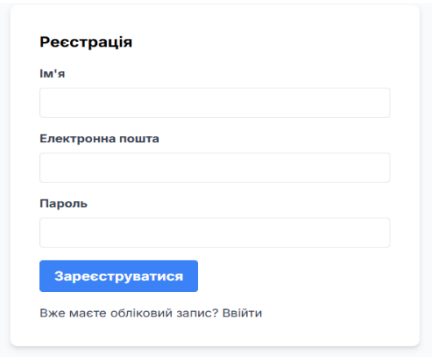


Рисунок 3.40 – Форма реєстрації

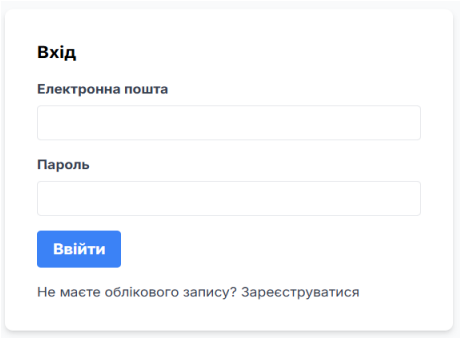


Рисунок 3.41 – Форма входу

можливості банківської установи, месенджер для взаємодії з банківським рахуноком.

Для початку користувач повинен увійти (рис. 3.40-3.41) у свій акаунт:

Після цього користувач потрапляє на сторінку, де видно документи, що він завантажив (рис. 3.42). Є форма для завантаження нового документу, бічне

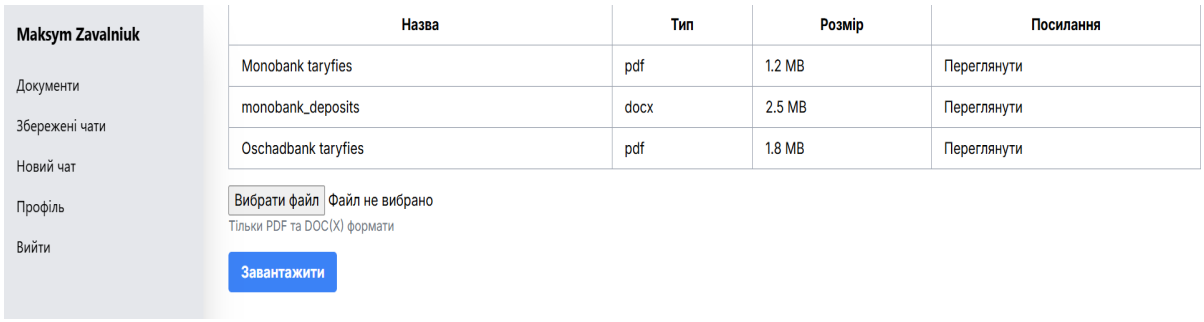


Рисунок 3.42 – Сторінка документів

меню, яке дозволяє переміщатись між різними вкладками та можливість переглянути інформацію про кожен документ окремо. Нас найбільше цікавить можливість почати новий чат. Увівши свій запит у вікно чату (рис. 3.19), наприклад, про види карток у Монобанку, ми отримаємо наступну відповідь

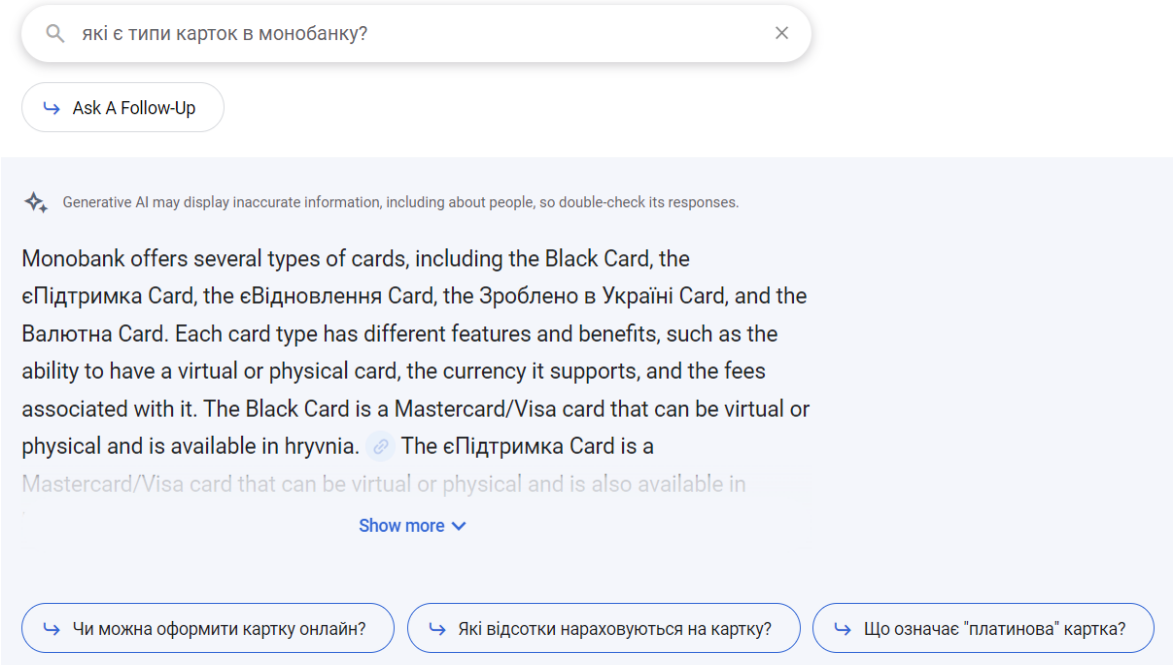


Рисунок 3.43 – Відповідь мовної моделі

(рис. 3.43). Як бачимо, відповідь є точною. Також модель пропонує наступні

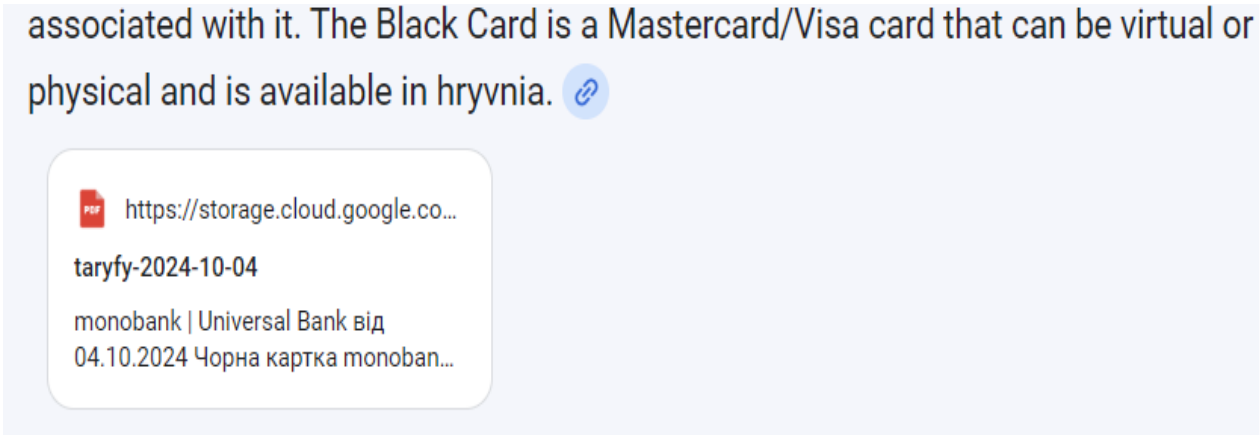


Рисунок 3.44 – Посилання на джерела

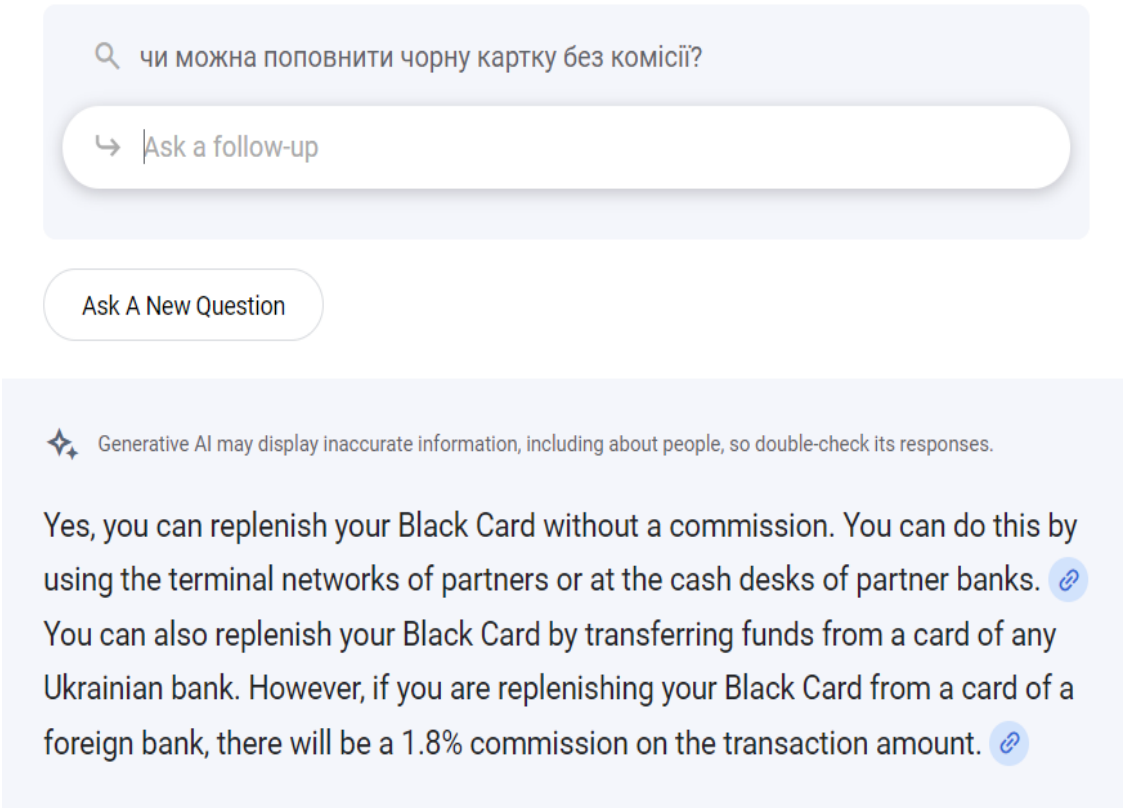


Рисунок 3.45 – Використання запропонованого запитання

питання виходячи з контексту. Однією з корисних функцій моделі є посилання на джерела (рис. 3.44), що дозволяє клієнту ширші маневри у вирішенні неточностей при використанні цієї платформи з банком. Ще це дозволяє перевірити, наскільки актуальними є дані, що їх використовує модель. Давайте

ще спробуємо використати одне із запропонованих наступних запитань від самої моделі. Як видно на рисунку 3.45, модель цілком справилась і з цим, при цьому маючи в пам’яті весь контекст розмови(окремого чату).

Іншою важливою функцією платформи є здійснення фінансових послуг.

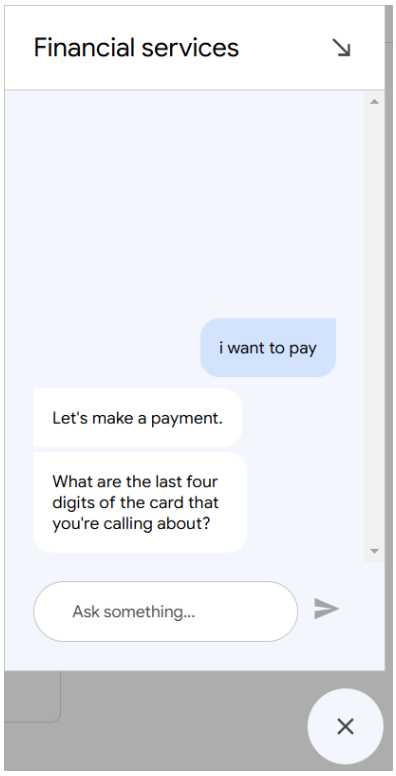


Рисунок 3.46 – Вигляд бота-помічника

Це можна зробити, викликавши чат-бот за допомогою спеціальної кнопки в правому нижньому кутку (рис. 3.46). Наприклад, ми хочемо провести платіж. Перш за все, система запитує останні чотири цифри картки клієнту. Пошта, ім’я та інші дані для авторизації передаються із сесії, оскільки користувач вже

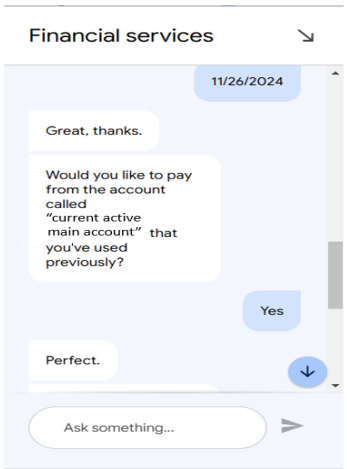


Рисунок 3.47 – Проведення платежу

увійшов у свій профіль до цього. Після декількох питань про рахуки, дату, картку отримувача, бот проводить платіж (рис.3.47).

Усі рахунки є тестові на даний момент. Тому реальних платежів провести неможливо. Але оскільки ми ще використовуємо BigQuery для аналізу даних, то можна проводити ось такі запити, як наприклад, ідентифікувати клієнтів, у яких незвично малий баланс відносно інший людей у своїй віковій категорії. Як видно на рисунку 3.48, запити проводяться за допомогою мови запитів SQL, а результати представлені у зручні таблиці. І запит, і результати можливо зберегти для майбутніх використань.

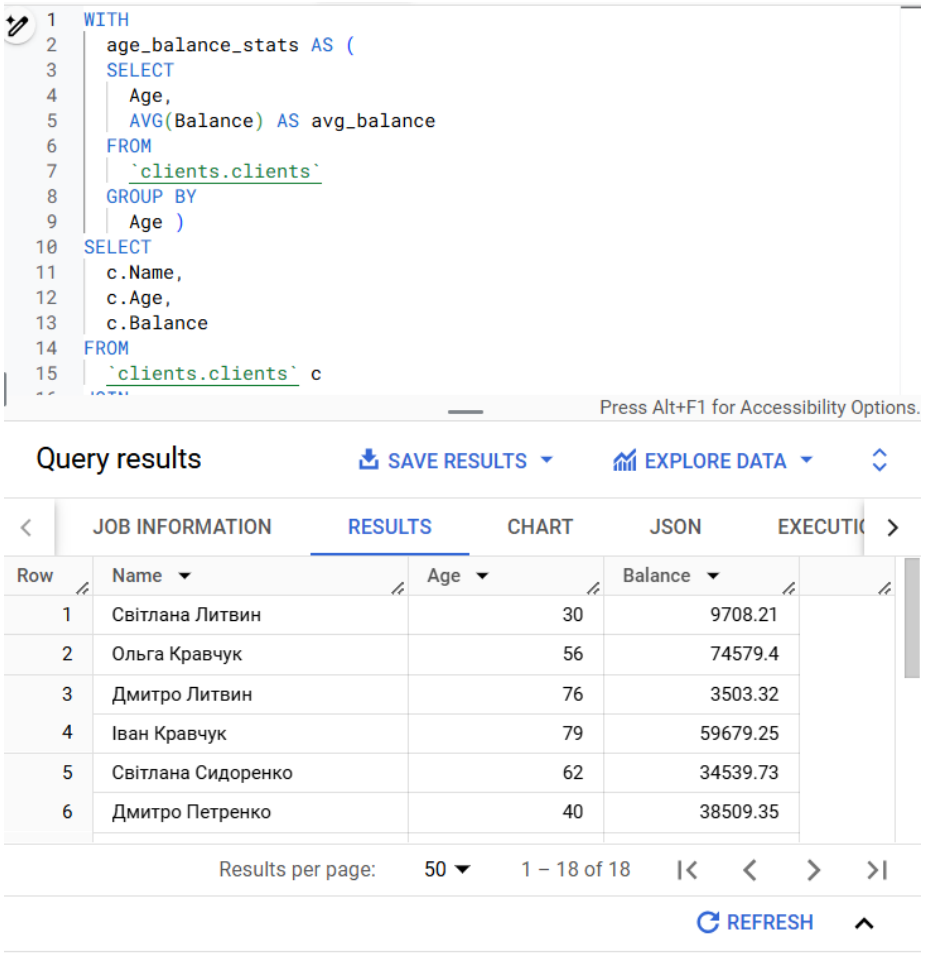


Рисунок 3.48 – Результат запиту BigQuery

3.11 Висновки

Відповідно до поставленої мети дослідження, розроблено архітектуру застосунку для автоматизації надання фінансових послуг, центральним елементом якої є створення мовної моделі з постійним навчнням за допомогою платформи Vertex AI. Детально описано логіку роботи автоматизованого конвеєру для навчання та роботи мовної моделі. Розробка здійснювалась в середовищі Microsoft Visual Studio Code з використанням мов програмування Python та JavaScript, що дозволяє інтегрувати застосунок з хмарними сервісами Google Cloud Project. Такий підхід забезпечує масштабованість та надійність рішення. Обрані технології та платформа формують оптимальне середовище для розробки та розгортання застосунку, спрямованого на автоматизацію навчального процесу. Використання хмарних сервісів GCP забезпечує гнучкість та масштабованість системи в цілому.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було здійснено огляд та аналіз наявних рішень для автоматизації надання фінансових послуг, зокрема їх функціональних можливостей та вимог користувачів. Було визначено завдання дослідження. За результатами дослідження розв’язана поставлена задача щодо автоматизації фінансових послуг.

Відповідно до сформованих завдань було отримано наступні результати:

- За допомогою кластеру Alloy DB та об’єктному сховищу Cloud Storage забезпечено безперебійний доступ до даних, а їхню надійність забезпечує Google Cloud Platform.
- Власний текстовий процесор на базі Document AI, який поєднання з Cloud Storage для отримання матеріалів та надсилання даних.
- Навчену модель Gemini Flash 1.5 на базі даних, отриманих після обробки нормативних документів за допомогою «Оптичного розпізнавання символів».
- Автоматизований конвеєр, який поєднує вихідні дані з текстового процесору і вхідні дані для навчання мовної моделі.
- Інтерактивний чат-бот, який дозволяє проводити платежі, переглядати баланс, перевіряти заборгованість.
- Веб-портал на базі FastAPI та Next.js, що дозволяє користувачам задавати питання мовній моделі, завантажувати докумени та взаємодіяти із чат-ботом.

Згідно визначенням ключових вимог по обробці нормативних документів, мовна модель повністю вчиться на контексті фінансової установи і ознайомлена з усієї термінологією, що використовується в банківських документах. Хмарні сховища беруть на себе відповідальність за доступність, безпеку та конфідеційність інформації, що полегшило та зробило більш

гнучку розробку.

Уся система була спроектована у вигляді автоматизованого конвеєру. І тому вибір мікросервісної архітектури виправдав себе. Він дозволяє масштабувати різні компоненти процесу у вертикальному та горизонтальному напрямках без перешкоджанню іншим компонентам. Використання хмарних сервісів дозволило масштабувати ресурси у відповідності до завантаження в реальному часі та ще й з економією грошових витрат.

У процесі тестування було підтверджено ефективність використання мовної моделі Gemini 1.5 з попереднім власним навчанням на платформі Vertex AI. Також використання текстового процесору від Document AI дозволило з великою точністю отримувати дані з документів різних типів та одразу передавати їх для навчання мовної моделі за допомогою Vertex Pipeline.

Автоматичне розгортання середовища було створено за допомогою Vertex AI та Cloud Build, що дозволяє створювати автоматичний процес CI/CD.

У процесі написання коду використовується Microsoft Visual Studio Code з мовами програмування Python та JavaScript.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Team Haptik. 5 Best Chatbots in the Financial Services Industry in 2024. Drive Business Efficiency at Scale with Generative AI. Jio Haptik. URL: https://www.haptik.ai/blog/best-chatbots-in-financial-industry (дата звернення: 09.10.2024).
2. What Is a Chatbot?. Oracle. Cloud Applications and Cloud Platform. URL: https://www.oracle.com/chatbots/what-is-a-chatbot/ (дата звернення: 09.10.2024).
3. Що таке чат-бот: визначення, приклади, відео. SendPulse UA. SendPulse. URL: https://sendpulse.ua/support/glossary/chatbot?_gl=1*17eokox*_ga*MTQ5NzAzNzQyMi4xNzI4NTAyOTk0*_ga_46NQ594GKJ*MTcyODUwMjk5My4xLjAuMTcyODUwMjk5My4wLjAuMTAzNDk2OTc.*_fplc*TG85RyUyRnd5dzNvYTFXM1hra1MzRWdvWTl5NTFWMHF2STdrT2cweTBiJTJCWXNTUmpCdUY4dXdHQ2RDYnl2dnZ3ajNuVlVFSElab1lSem9YUFYzUGpKNmZOTiUyRjIwQ3g2NE1DeDNOS2tLSFUlMkYlMkZseFhPVmthSEJuT2hEdm5taW13ZyUzRCUzRA.. (дата звернення: 09.10.2024).
4. Salesloft: The Leading Revenue Orchestration Platform. Welcome Drift. URL: https://www.drift.com/wp-content/uploads/2018/01/2018-state-of-chatbots-report.pdf (дата звернення: 09.10.2024).
5. Best Finance Chatbot for Banking & Personal Financial Advice. Tidio. URL: https://www.tidio.com/blog/finance-chatbots/ (дата звернення: 09.10.2024).
6. Chatbots in Finance [Benefits, Examples, Future]. Yellow. Software Product Agency. Yellow. URL: https://yellow.systems/blog/chatbots-in-finance (дата звернення: 09.10.2024).
7. What are the limitations of AI-based chatbots for financial services?. URL: https://www.linkedin.com/advice/1/what-limitations-ai-based-chatbots-

financial-vqbef (дата звернення: 09.10.2024).

8. Singh A., Ramasubramanian K., Shivam S. Building an Enterprise Chatbot. Berkeley, CA : Apress, 2019. URL: https://doi.org/10.1007/978-1-4842-5034-1 (дата звернення: 09.10.2024).
9. Sabharwal N., Agrawal A. Cognitive Virtual Assistants Using Google Dialogflow. Berkeley, CA : Apress, 2020. URL: https://doi.org/10.1007/978-1-4842-5741-8 (дата звернення: 09.10.2024).
10. Sary D. Google Cloud Platform : a Practical Guide to Google Cloud from Scratch: Google Cloud Platform for Developers. Independently Published, 2021.
11. Bhatia J., Chaudhary K. Definitive Guide to Google Vertex AI: Implement Machine Learning Pipelines with Google Cloud Vertex AI. Packt Publishing, Limited, 2023.
12. Amaratunga T. Understanding Large Language Models. Berkeley, CA : Apress, 2023. URL: https://doi.org/10.1007/979-8-8688-0017-7 (дата звернення: 09.10.2024).
13. FastAPI: Modern Python Web Development. O'Reilly Media, Incorporated, 2023.
14. Frost A. JS. Next: ECMAScript 6. O'Reilly Media, Incorporated, 2017.
15. Lathkar M. High-Performance Web Apps with FastAPI. Berkeley, CA : Apress, 2023. URL: https://doi.org/10.1007/978-1-4842-9178-8 (дата звернення: 09.10.2024).
16. Python API Development Fundamentals: Develop a Full-Stack Web Application with Python and Flask. Packt Publishing, Limited, 2019. 372 с.
17. Modern Full-Stack React Projects: Build, Maintain, and Deploy Modern Web Apps Using MongoDB, Express, React, and Node.js. Packt Publishing, Limited, 2024.
18. Cabianca D. Google Cloud Platform (GCP) Professional Cloud Network

Engineer Certification Companion. Berkeley, CA : Apress, 2023. URL: https://doi.org/10.1007/978-1-4842-9354-6 (дата звернення: 09.10.2024).

19. Віртуальні помічники для бізнесу: як чат-боти з AI автоматизують бізнес-процеси. Досвід Netpeak. dev.ua. URL: https://dev.ua/blogs/posts/virtualni-pomichnyky-dlia-biznesu-blog (дата звернення: 09.10.2024).
20. Vertex AI with Gemini 1.5 Pro and Gemini 1.5 Flash. Google Cloud. URL: https://cloud.google.com/vertex-ai?hl=uk (дата звернення: 09.10.2024).
21. Document AI Google Cloud. Google Cloud. URL: https://cloud.google.com/document-ai?hl=uk (дата звернення: 09.10.2024).
22. Dr Sophia Lee. Vertex AI: Unleashing the Power of Machine Learning. London, 2023. 512 c.
23. Williams M., Chen E. Building the Future with Vertex AI: A Practical Guide for Developers. 2024. 384 c.
24. Dr Anya Kumar. Gemini: Dawn of Multimodal AI. 2024. 420 c.
25. Chen D., Wilson E. The Gemini Effect: Transforming Industries with Multimodal AI. 2023. 384 c.
26. Brown M. Demystifying Gemini: A Practical Guide for Developers and Businesses. 2024. 512 c.
27. Завальнюк М., Ковалюк О. ВИКОРИСТАННЯ ЧАТ-БОТІВ У БІЗНЕСІ. КОНФЕРЕНЦІЇ ВНТУ електронні наукові видання. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20395/17044 (дата звернення: 10.10.2024).
28. Cloud Services - Build and Scale Securly - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/products/?nc1=h_ls&aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re:Invent=*all&awsf.Free%20Tier%20Type=*all

&awsf.tech-category=*all (дата звернення: 25.11.2024).

29. Microsoft Azure Review 2024: Not Bad, but We've Seen Better. Website Planet. URL: https://www.websiteplanet.com/web-hosting/microsoft-azure/ (дата звернення: 25.11.2024).
30. Cloud Logging Google Cloud. Google Cloud. Google Cloud. URL: https://cloud.google.com/logging?hl=uk (дата звернення: 25.11.2024).
31. BigQuery enterprise data warehouse. Google Cloud. URL: https://cloud.google.com/bigquery (дата звернення: 25.11.2024).
32. Quinten J. Building Real-World Web Applications with Vue. js 3: Build a Portfolio of Vue. js and TypeScript Web Applications to Advance Your Career in Web Development. de Gruyter GmbH, Walter, 2024.
33. Practical Svelte: Create Performant Applications with the Svelte Component Framework. Apress L. P., 2021. 340 с.
34. Conversational Agents (Dialogflow CX) documentation. Google Cloud. URL: https://cloud.google.com/dialogflow/cx/docs (дата звернення: 25.11.2024).
35. Cloud Computing Services Google Cloud. Google Cloud. URL: https://cloud.google.com/?hl=uk (дата звернення: 27.11.2024).
36. Cloud Services. URL: https://www.hpe.com/emea_africa/en/what-is/cloud-services.html (дата звернення: 27.11.2024).
37. GeeksforGeeks. Python Language advantages and applications - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/python-language-advantages-applications/ (дата звернення: 27.11.2024).
38. GeeksforGeeks. Advantages and Disadvantages of JavaScript - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-javascript/ (дата звернення: 27.11.2024).

39. What are microservices?. microservices.io. URL: https://microservices.io/ (date of access: 27.11.2024).
40. What are Microservices in AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/microservices/ (дата звернення: 27.11.2024).
41. Cloud Storage. Google Cloud. URL: https://cloud.google.com/storage?hl=uk (дата звернення: 27.11.2024).
42. AlloyDB for PostgreSQL. Google Cloud. URL: https://cloud.google.com/products/alloydb?hl=uk (дата звернення: 27.11.2024).
43. В. М. Дубовой В. М. МЕТОДИЧНІ ВКАЗІВКИ до виконання магістерських кваліфікаційних робіт для студентів спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» (кафедра комп'ютерних систем управління). Вінниця, 2023. 45 с.

ДОДАТКИ

ДОДАТОК А

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_ « Розробка автоматизованої системи для обслуговуванняклієнтів фінансових установ »

Тип роботи:_____магістерська кваліфікаційна робота_____

(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ __кафедра комп'ютерних систем управління_____

(кафедра, факультет (інститут), навчальна група)

Керівник___Ковалюк О.О., доцент кафедри КСУ____

(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	99%
Загальна схожість	1%

Аналіз звіту подібності (відмітити потрібне)

√ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор_____ Максим ЗАВАЛЬНЮК

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____

_____

Особа, відповідальна за перевірку_ _____ _ Володимир ДУБОВОЙ _

(підпис) (прізвище, ініціал

Експерт _____ _____

(за потреби) (підпис)

ДОДАТОК Б
(обов'язковий)
Технічне завдання
Вінницький національний технічний університет

ЗАТВЕРДЖЕНО
Зав. кафедри КСУ,
д.т.н., професор
_____В'ячеслав КОВТУН
"14" жовтня 2024 р

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи
Розробка автоматизованої системи для обслуговування клієнтів фінансових установ

08-33.МКР.04.00.000 ТЗ

Студент групи ЗАКІТР-23м

Підпис Максим ЗАВАЛЬНЮК
Ім'я ПРІЗВИЩЕ

Керівник к.т.н., доцент кафедри

Підпис Олег КОВАЛЮК
Ім'я ПРІЗВИЩЕ

1. Назва та галузь застосування

1.1. Назва – Розробка автоматизованої системи для обслуговування клієнтів фінансових установ.

1.2. Галузь застосування – фінансові послуги.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від №310 від «17» вересня 2024 року.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є забезпечення автоматизації введення фінансових послуг у банківських установ.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Що таке велика мовна модель. URL: https://thetransmitted.com/adlucem/shho-take-velyka-movna-model-large-language-model-llm/

2. What is artificial intelligence (AI) in finance?. URL: https://cloud.google.com/discover/finance-ai?hl=uk

3. The Gemini ecosystem. URL: https://ai.google/gemini-ecosystem

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- аналіз і розпізнавання тексту з документів;
- автоматизація процесу навчання мовної моделі;
- автоматичне проведенння фінансових послуг;

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- PostgreSQL,
- Next.js,
- Python,
- GCP,
- Windows 10,

5.2.2. Умови експлуатації системи:

- безперебійне цілодобове функціонування системи;
- регулярне оновлення та підтримка актуальності даних
- надійний захист даних

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в фінансових управліннях відповідно до теми кваліфікаційної роботи. Постановка задач дослідження «03» вересня 2024 р.
2. Удосконалення технології прийняття рішень при автоматизації фінансових послуг «1» листопада 2024 р.
3. Визначення технічних характеристик системи «3» листопада 2024 р.
4. Розробка програмного забезпечення системи «5» листопада 2024 р.

6.2 Графічні матеріали:

1. Розробка UML-діаграм системи «3» листопада 2024 р.
2. Розробка моделі бази даних системи «11» листопада 2024 р.
3. Тестування програмного забезпечення «19» листопада 2024 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «29» листопада 2024 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «1» грудня 2024 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «11» грудня 2024 р.

ДОДАТОК В
(довідниковий)
Лістинг програми

Базовий клас CRUD для взаємодії з базою даних

```
from typing import Annotated

from fastapi import Query
from pydantic import BaseModel
from sqlmodel import Session, select

from .enums import Model


class CRUD:
    def __init__(self, model: Model):
        self.model = model

    def get(self, session: Session, id: int):
        return session.get(self.model, id)

    def get_all(
        self,
        session: Session,
        offset: int = 0,
        limit: Annotated[int, Query(le=100)] = 100,
    ):
        return session.exec(
            select(self.model).offset(offset).limit(limit)
        ).all()

    def create(
        self, session: Session, obj_in: BaseModel, additional_data: dict = None
    ):
        obj_in_data = obj_in.model_dump()
        if additional_data:
            obj_in_data.update(additional_data)
        db_obj = self.model(**obj_in_data)
        session.add(db_obj)
        session.commit()
        session.refresh(db_obj)
        return db_obj

    def update(self, session: Session, id: int, obj_in: BaseModel):
```

```
        obj = session.get(self.model, id)
        if not obj:
            return None
        obj_in_data = obj_in.model_dump(exclude_unset=True)
        obj.sqlmodel_update(obj_in_data)
        session.add(obj)
        session.commit()
        session.refresh(obj)
        return obj

    def delete(self, session: Session, id: int):
        obj = session.get(self.model, id)
        if not obj:
            return None
        session.delete(obj)
        session.commit()
        return {"ok": True}
```

Код middleware для логування запитів на серверній частині

```
import json
import os
import time

from fastapi import Request
from google.cloud import logging as cloud_logging
from loguru import logger

# Create a Cloud Logging client
client = cloud_logging.Client()
logger_name = "my_logger"
cloud_logger = client.logger(logger_name)

# Configure Loguru to use Cloud Logging
logger.add(
    cloud_logger,
    format="{time} | {level} | {message}",
    level="INFO",
    serialize=True,
)


async def log_middleware(request: Request, call_next):
    start_time = time.time()

    # Extract details from the request
    request_body = await request.body()
```

```
    request_data = {
        "method": request.method,
        "url": str(request.url),
        "query_params": dict(request.query_params),
        "body": request_body.decode("utf-8") if request_body else None,
    }

    # Log the request data using Cloud Logging
    cloud_logger.log_struct(request_data, severity="INFO")

    response = await call_next(request)

    duration = time.time() - start_time

    log_dict = {
        "method": request.method,
        "url": str(request.url),
        "status_code": response.status_code,
        "duration": duration,
        "request": request_data,
        "response_status_code": response.status_code,
    }

    # Log the response data using Cloud Logging
    cloud_logger.log_struct(log_dict, severity="INFO")

    return response
```

Сторінка з документами

```
import React, { useState, useEffect } from "react";
import Link from "next/link";
import axios from "axios";

function DocumentsPage() {
  const [documents, setDocuments] = useState([]);
  const [newDocument, setNewDocument] = useState(null);
  const [uploading, setUploading] = useState(false);
  const [token, setToken] = useState(null);

  useEffect(() => {
    const storedToken = localStorage.getItem("token");
    if (storedToken) {
      setToken(storedToken);
    }
  }, []);
```

```
useEffect(() => {
  if (token) {
    fetchDocuments();
  }
}, [token]);

const fetchDocuments = async () => {
  try {
    const response = await axios.get("/api/documents", {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    setDocuments(response.data);
  } catch (error) {
    console.error(error);
  }
};

const handleUpload = async (event) => {
  event.preventDefault();
  setUploading(true);

  const file = event.target.files[0];
  const formData = new FormData();
  formData.append("file", file);

  try {
    const response = await axios.post("/api/documents", formData, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    setNewDocument(response.data);
    setUploading(false);
  } catch (error) {
    console.error(error);
    setUploading(false);
  }
};

return (
  <div className="flex flex-col h-screen">
    <div className="flex-1 overflow-y-auto">
      <table className="w-full table-auto border-collapse border border-gray-400">
        <thead>
          <tr>
```

```
    <th className="px-4 py-2 border border-gray-400">Назва</th>
    <th className="px-4 py-2 border border-gray-400">Тип</th>
    <th className="px-4 py-2 border border-gray-400">Розмір</th>
    <th className="px-4 py-2 border border-gray-400">Посилання</th>
  </tr>
</thead>
<tbody>
  {documents.map((document) => (
    <tr key={document.id} className="hover:bg-gray-100">
      <td className="px-4 py-2 border border-gray-400">
        {document.name}
      </td>
      <td className="px-4 py-2 border border-gray-400">
        {document.type}
      </td>
      <td className="px-4 py-2 border border-gray-400">
        {document.size}
      </td>
      <td className="px-4 py-2 border border-gray-400">
        <Link
          href={{
            pathname: `/user/documents/${document.id}`,
          }}
        >
          Переглянути
        </Link>
      </td>
    </tr>
  ))}
  {newDocument && (
    <tr key={newDocument.id} className="hover:bg-gray-100">
      <td className="px-4 py-2 border border-gray-400">
        {newDocument.name}
      </td>
      <td className="px-4 py-2 border border-gray-400">
        {newDocument.type}
      </td>
      <td className="px-4 py-2 border border-gray-400">
        {newDocument.size}
      </td>
      <td className="px-4 py-2 border border-gray-400">
        <Link
          href={{
            pathname: `/user/documents/${newDocument.id}`,
          }}
        >
          Переглянути
```

```
                  </Link>
                </td>
              </tr>
            )}
          </tbody>
        </table>
        <div className="flex flex-col mt-4">
          <input
            type="file"
            accept=".pdf, .docx"
            onChange={handleUpload}
            disabled={uploading}
          />
          {uploading && (
            <span className="text-sm text-gray-500">Завантаження...</span>
          )}
        </div>
      </div>
    </div>
  );
}

export default DocumentsPage;
```

ДОДАТОК Г
(Обов'язковий)
Ілюстративна частина

РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ДЛЯ ОБСЛУГОВУВАННЯ КЛІЄНТІВ ФІНАНСОВИХ УСТАНОВ

1. Схема архітектури системи
2. Схема розгортання ресурсів та доставка коду
3. Структура класів та зв'язків, що відповідають за створення та управління даними про клієнтів, чати, повідомлення та документи
4. UML діаграми послідовності роботи системи
5. Результати тестування. Зображення інтерфейсу користувача: сторінка завантажених документів
6. Результати тестування. Зображення нового чату з мовною моделлю
7. Результати тестування. Зображення відповіді чат-боту
8. Результати тестування. Скріншот проведення фінансового платежу через бота
9. Результати тестування. Зображення результату запиту до даних з подальшими можливостями до аналізу

Студент групи ЗАКІТР-23м

Максим ЗАВАЛЬНЮК
Підпис *Ім'я ПРІЗВИЩЕ*

Керівник к.т.н., доцент

Олег КОВАЛЮК
Підпис *Ім'я ПРІЗВИЩЕ*

СХЕМА АРХІТЕКТУРИ СИСТЕМИ

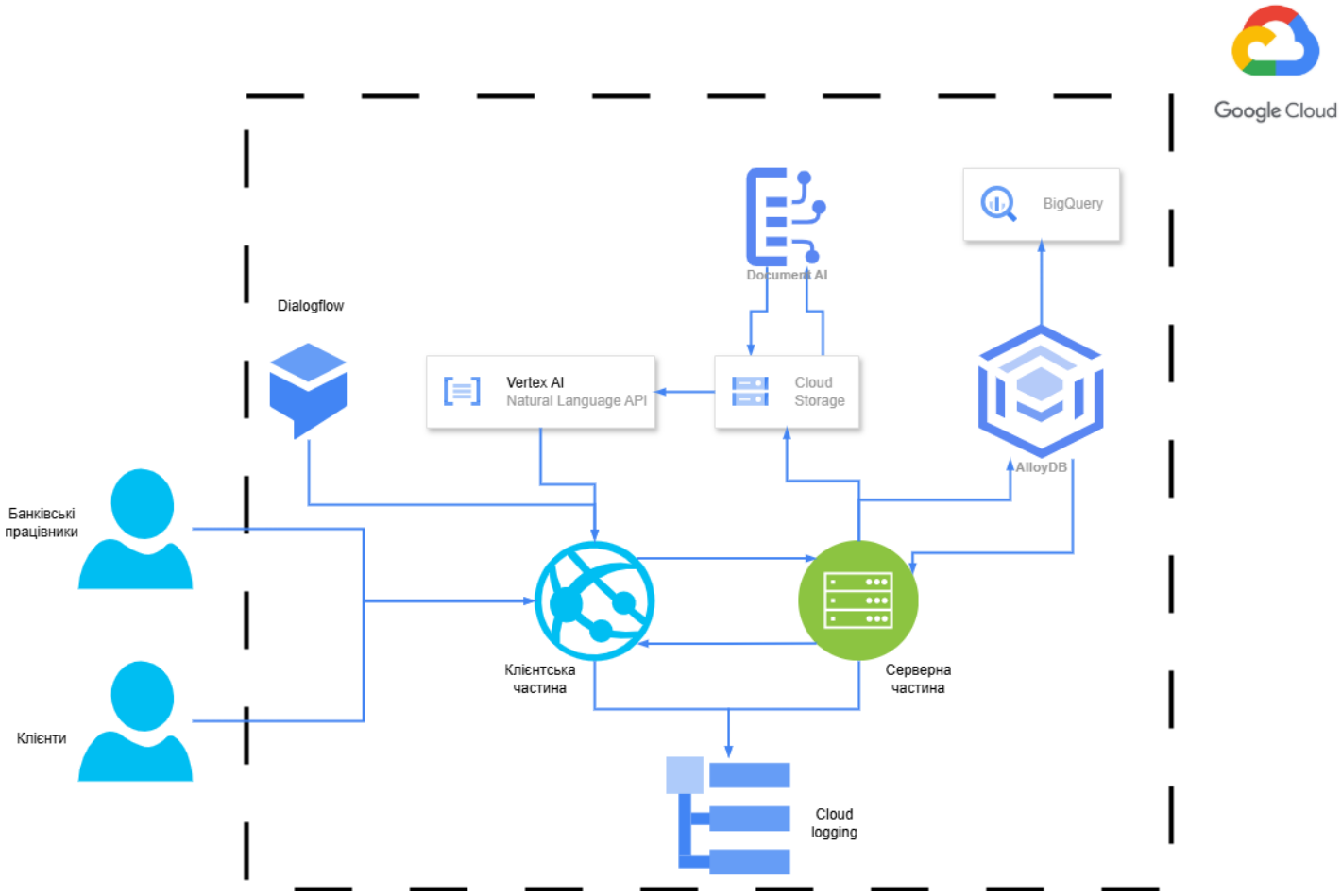
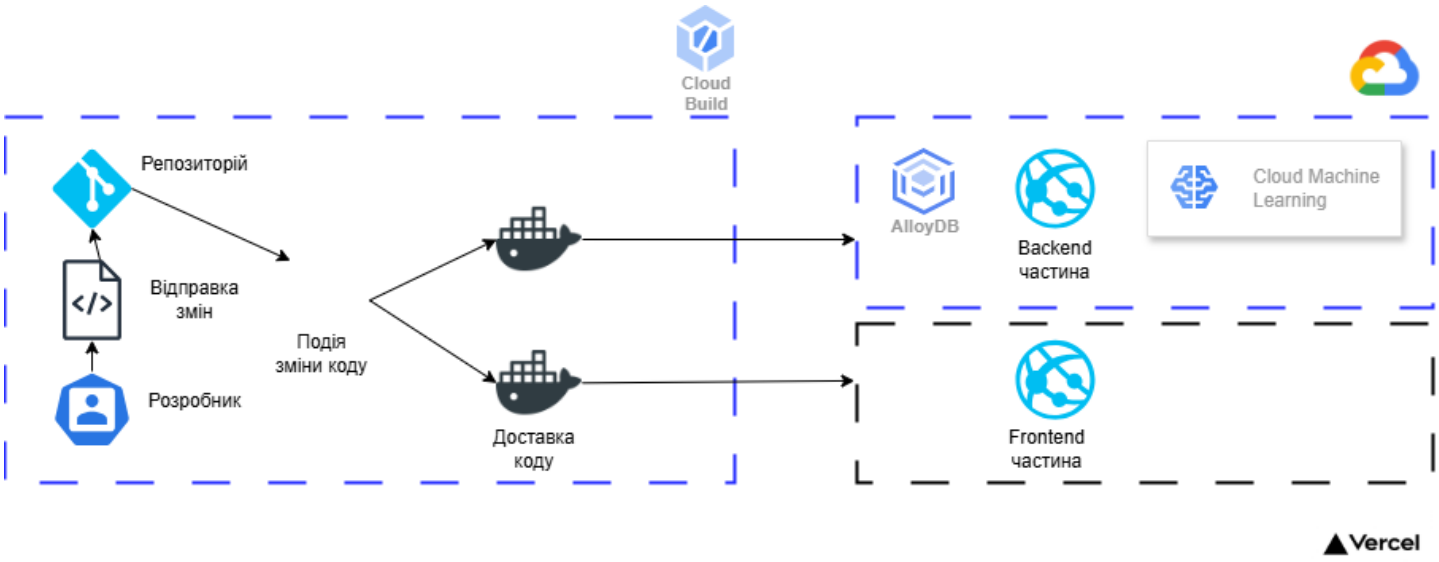
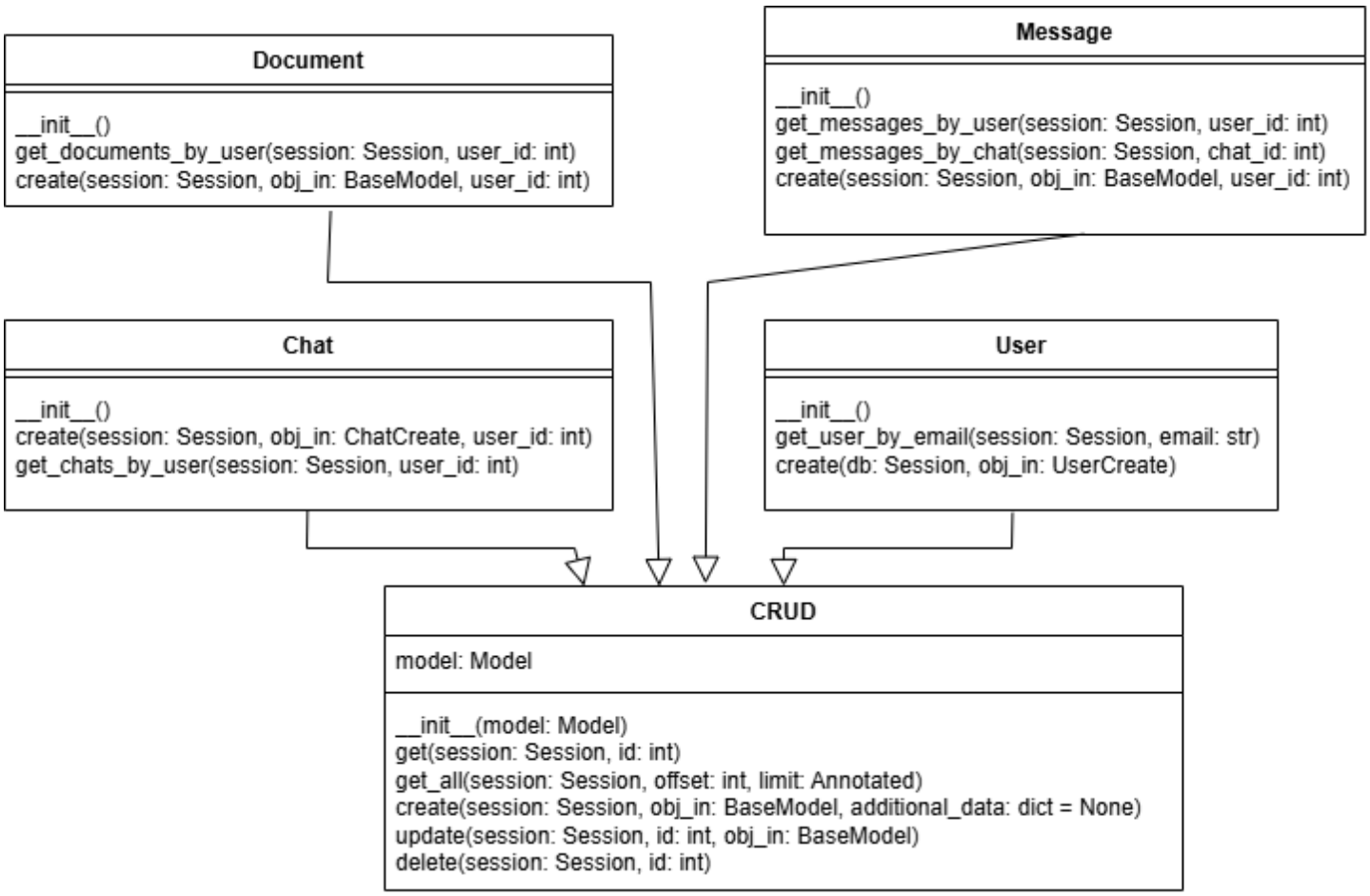


СХЕМА РОЗГОРТАННЯ РЕСУРСІВ ТА ДОСТАВКА КОДУ

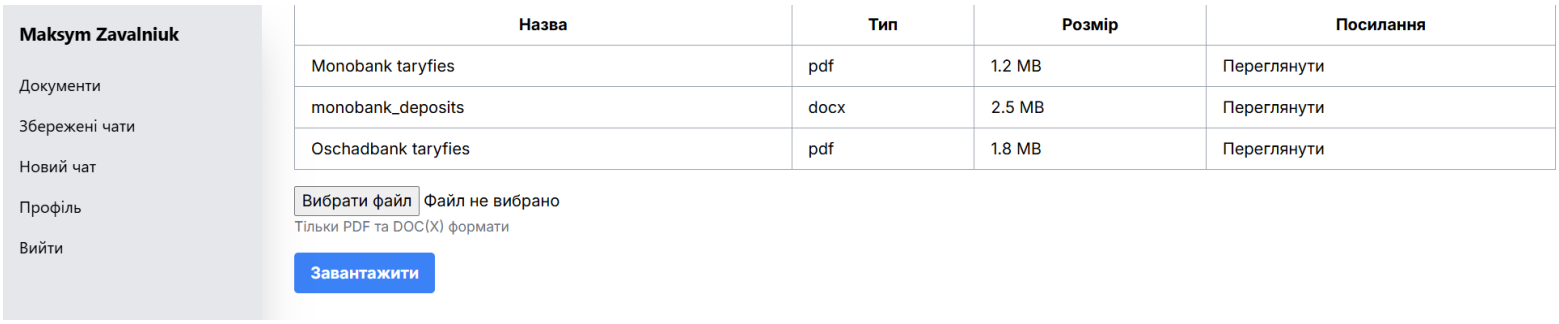


СТРУКТУРА КЛАСІВ ТА ЗВ'ЯЗКІВ, ЩО ВІДПОВІДАЮТЬ ЗА СТВОРЕННЯ ТА УПРАВЛІННЯ ДАНИМИ ПРО КЛІЄНТІВ, ЧАТИ, ПОВІДОМЛЕННЯ ТА ДОКУМЕНТИ

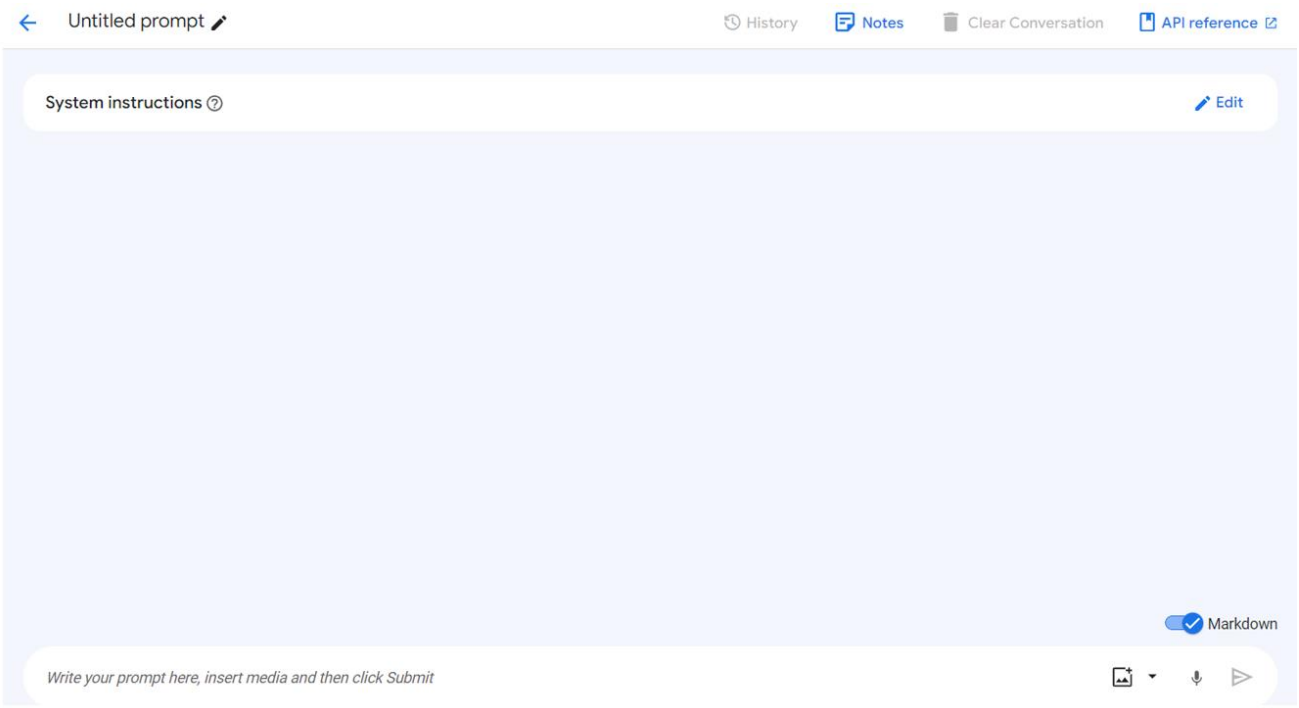


UML ДІАГРАМИ ПОСЛІДОВНОСТІ РОБОТИ СИСТЕМИ

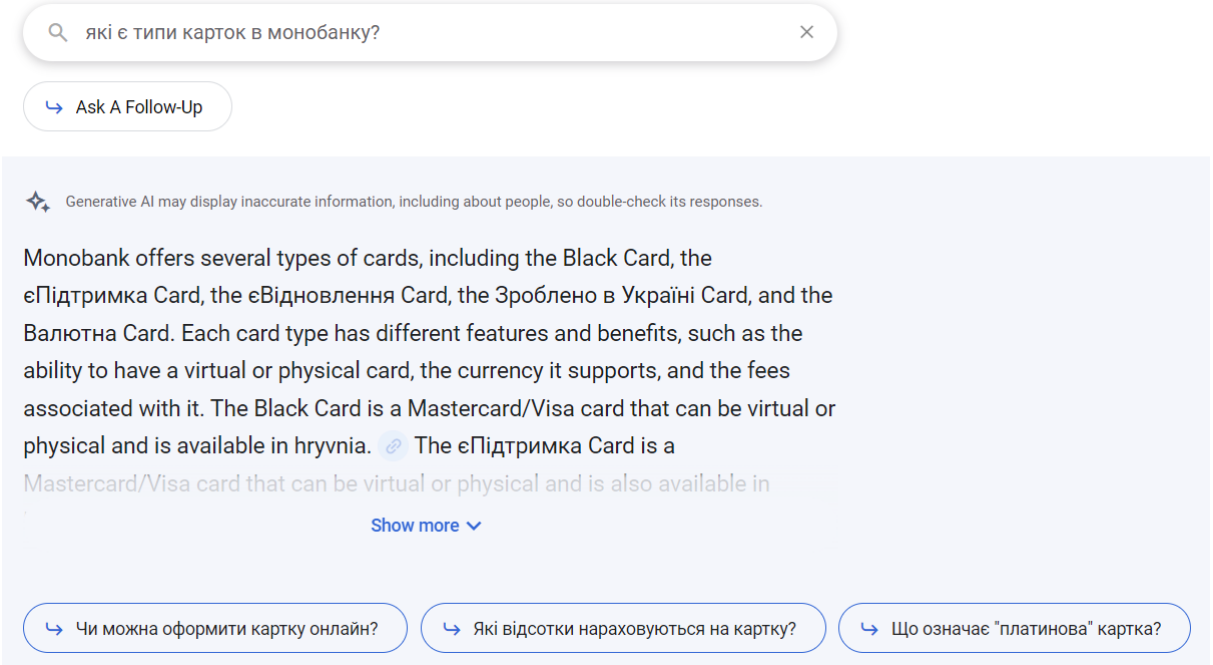
РЕЗУЛЬТАТИ ТЕСТУВАННЯ



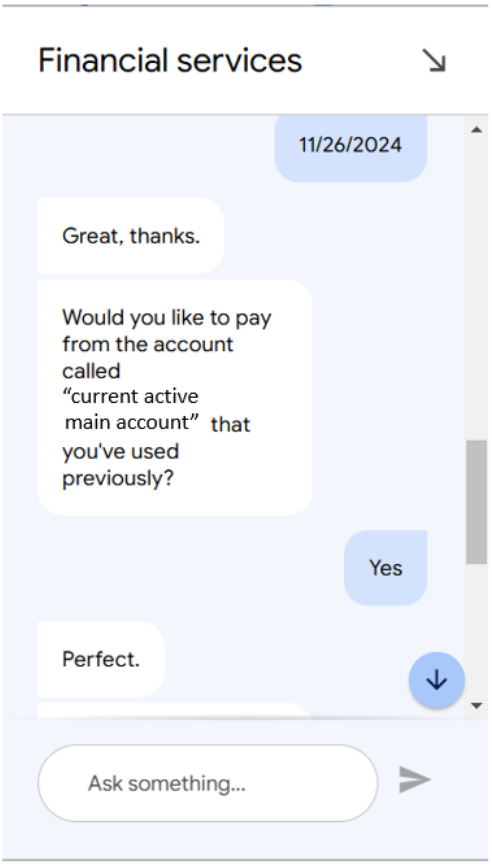
Зображення інтерфейсу користувача: сторінка завантажених документів



Зображення нового чату з мовною моделлю



Зображення відповіді чат-боту



Скріншот проведення фінансового платежу через бота

```
WITH
  age_balance_stats AS (
  SELECT
    Age,
    AVG(Balance) AS avg_balance
  FROM
    `clients.clients`
  GROUP BY
    Age )
SELECT
  c.Name,
  c.Age,
  c.Balance
FROM
  `clients.clients` c
```

Press Alt+F1 for Accessibility Options.

Query results

SAVE RESULTS | EXPLORE DATA

JOB INFORMATION | RESULTS | CHART | JSON | EXECUTI

Row	Name	Age	Balance
1	Світлана Литвин	30	9708.21
2	Ольга Кравчук	56	74579.4
3	Дмитро Литвин	76	3503.32
4	Іван Кравчук	79	59679.25
5	Світлана Сидоренко	62	34539.73
6	Дмитро Петренко	40	38509.35

Results per page: 50 1 – 18 of 18

REFRESH

Зображення результату запиту до даних з подальшими можливостями до аналізу