НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.86

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмне забезпечення для підготовки набору даних з кількох джерел за допомогою інструкцій природною мовою»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІП-22мп
Кучма Артем Борисович _____

Керівник:
д.т.н., професор, завідувач кафедри
Жаріков Едуард В’ячеславович _____

Рецензент:
доцент кафедри ІСТ, к.т.н.
Шимкович Володимир Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2022р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Кучмі Артему Борисовичу

1. Тема дисертації «Програмне забезпечення для підготовки набору даних з кількох джерел за допомогою інструкцій природною мовою», науковий керівник дисертації Жаріков Едуард В'ячеславович, д.т.н., професор, завідувач кафедри, затверджені наказом по університету від «09» листопада 2022 р. № 4115-с

2. Термін подання студентом дисертації «16» січня 2024 р.

3. Об'єкт дослідження – інтерфейси до баз та сховищ даних з використанням природної мови.

4. Предмет дослідження – моделі, методи і технології обробки команд природною мовою з метою побудови інтерфейсу для підготовки набору даних з кількох джерел.

5. Перелік завдань, які потрібно розробити – аналіз предметної області, досліджень, публікацій та актуальних проблем розробки інтерфейсів на основі природної мови; аналіз існуючих програмних рішень, їх переваг та недоліків; формування вимог до

ПЗ; проектування архітектури та вибір технологій для розробки ПЗ; розробка ПЗ; тестування ПЗ та аналіз отриманих результатів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 3 плакати

7. Орієнтовний перелік публікацій – 2 публікації

8. Консультанти розділів дисертації

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

9.Дата видачі завдання «1» вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Аналіз предметної області	1.09.2022	
2	Аналіз існуючих реалізацій	15.09.2022	
3	Постановка та формалізація задачі	16.09.2022	
4	Аналіз вимог до програмного забезпечення	18.09.2022	
5	Розробка нових моделей	22.09.2022	
6	Розробка програмного забезпечення	29.10.2023	
7	Виконання експериментальних досліджень	03.11.2023	
8	Оформлення пояснювальної записки	17.11.2023	
9	Подання дисертації на попередній захист	20.12.2023	
10	Подання дисертації на захист	16.01.2024	

Студент Артем КУЧМА

Науковий керівник Едуард ЖАРІКОВ

РЕФЕРАТ

Розмір пояснювальної записки – 87 аркушів, містить 33 ілюстрації, 24 таблиці, 5 додатків, 78 посилань на джерела.

Актуальність теми. У роботі розглянуто проблему в області взаємодії з базами та сховищами даних за допомогою інструкцій природною мовою, показано основні особливості існуючих рішень описаної проблеми, їх переваги та недоліки. Виявлено потребу у розробленні програмного забезпечення для підготовки наборів даних з декількох джерел за допомогою інструкцій природною мовою.

Мета дослідження. Метою є покращення процесу взаємодії з базами та сховищами даних у спосіб розроблення програмного забезпечення, що реалізує зручний інтерфейс до баз і сховищ даних із застосуванням команд природною мовою.

Об’єкт дослідження: інтерфейси до баз та сховищ даних з використанням природної мови.

Предмет дослідження: моделі, методи і технології обробки команд природною мовою з метою побудови інтерфейсу для підготовки набору даних з кількох джерел.

Для досягнення поставленої мети **необхідно виконати такі завдання**:

- аналіз предметної області, досліджень, публікацій та актуальних проблем розробки інтерфейсів на основі природної мови;
- аналіз існуючих програмних рішень, їх переваг та недоліків;
- формування вимог до програмного забезпечення;
- проектування архітектури та вибір технологій для розробки програмного забезпечення;
- розробка програмного забезпечення;
- тестування програмного забезпечення та аналіз отриманих результатів.

Наукова новизна результатів магістерської дисертації полягає в тому, що набуло подальшого розвитку використання потужностей Large Language Model моделей для підготовки масивів даних з декількох джерел за допомогою команд природньою мовою. Результат досягнутий шляхом використання LLM моделей для обробки інтсрукцій природньою мовою, та підходу data warehouse для об'єднання даних з декількох джерел у спільний SQL інетфрейс.

Практичне значення отриманих результатів полягає в розробці веб-додатку, що дає можливість формувати набори даних з декількох джерел за допомогою інструкцій природною мовою. Даний додаток об'єднує джерела даних у спільний SQL інтерфейс за допомогою патерну data warehouse. Додаток дає можливість обробляти голосові та текстові команди англійською мовою, перетворювати їх в SQL до відповідного об'єднаного інтерфейсу, виконувати згенерований запит та надавати результати виконання користувачу у форматі CSV.

Зв'язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» у рамках ініціативної теми «Методи та технології високопродуктивних обчислень та обробки надвеликих масивів даних», номер реєстрації УКРІНТЕІ 0117U000924.

Апробація. Наукові положення дисертації пройшли апробацію на наукових конференціях SoftTech 2022 та SoftTech 2023

Ключові слова: СХОВИЩЕ ДАНИХ, ОБРОБКА ПРИРОДНОЇ МОВИ, DATA WAREHOUSE, ІНТЕРФЕЙСИ ПРИРОДНОЇ МОВИ

ABSTRACT

Explanatory note size – 87 pages, contains 32 illustrations, 24 tables, 5 applications, 78 references.

Topicality. Examines the problem of interaction with data sources using natural language instructions, shows the main features of existing solutions for the described problem, their advantages and disadvantages. A need has been identified for software to prepare datasets from multiple sources using natural language instructions.

The aim of the study. The main target is to improve the process of interaction with databases and data repositories in the way of developing software that implements a convenient interface to databases and data repositories using natural language commands.

The object of research: interfaces to databases and data storages using natural language.

The subject of research: models, methods and technologies for processing commands in natural language in order to build an interface for preparing sets of data from several sources.

To achieve this goal, the **following tasks** were formulated:

- analysis of the subject area, research, publications and actual problems of developing interfaces based on natural language;
- analysis of existing software solutions, their advantages and disadvantages;
- formation of requirements for the software product;
- architecture design and technology selection for software product development;
- software product development;
- software product testing and analysis of the obtained results.

The scientific novelty of the results of the master's dissertation is that the use of the capabilities of LLM models for the preparation of data sets from several sources with natural language commands has been further developed. The result is achieved by using Large Language Model models for processing instructions in natural language, and a data

warehouse approach for combining data from several sources into a common SQL interface.

The practical value of the obtained results is the development of a web application capable of preparing datasets from multiple sources using natural language instructions. This application combines data sources into a common SQL interface using the data warehouse pattern. The application is able to process voice and text commands in English, convert them into SQL to the corresponding unified interface, execute the generated query and provide the execution results to the user in CSV format.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky» within the framework of the initiative topic "Methods and technologies of high-performance computing and processing of ultra-large data sets", registration number UKRINTEI 0117U000924

Approbation. The scientific provisions of the dissertation were tested at the scientific conferences SoftTech 2022 and SoftTech 2023.

Keywords: DATA STORAGE. NATURAL LANGUAGE PROCESSING, DATA WAREHOUSE, NATURAL LANGUAGE INTERFACE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SQL - Structured Query Language, мова запитів до баз реляційних баз даних;

No-SQL – Not Structured Query Language, позначення баз даних, що не являються реляційними та не підтримують мову запитів SQL;

GUI – Graphical user interface, графічний інтерфейс користувача;

CLI – Command line interface, консольний інтерфейс;

API – Application Programming Interface, програмний інтерфейс інформаційної системи;

HTTP – Hyper Text Transfer Protocol, один з мережевих протоколів прикладного рівня;

TCP – Transmission Control Protocol, один з мережевих протоколів транспортного рівня;

AMQP - Advanced Message Queuing Protocol, один з мережевих протоколів прикладного рівня;

NLP – Natural Language Processing, обробка природної мови;

NLIDB – Natural Language Interface for DataBase, інтерфейс природної мови до бази даних;

NLUI – Natural Language User Interface, інтерфейс природної мови;

SPA – Single Page Application, архітектурний підхід побудови веб-орієнтованих графічних інтерфейсів;

IaC – Infrastructure as Code, підхід до опису інфраструктури систем;

LLM – Large Language Models, або Великі Мовні моделі, сімейство моделей машинного навчання, що навчається на великих об’ємах немаркованого тексту;

CSV – Comma Separated Values, формат збереження даних, у яких відповідні елементи даних розділяються комою;

SDK – Software Development Kit, набір засобів розробки.

ВСТУП

Сучасні інформаційні системи широко використовують сховища даних для збереження необхідної інформації. Такі сховища можуть включати SQL та NO-SQL бази даних, мережеві файлові системи, веб сервіси збереження документів та інші. Зазвичай, при проектуванні баз даних велика частка уваги приділяється таким аспектам, як оптимізація використаного дискового простору або пришвидшення виконання запитів. Зручність інтерфейсу баз даних може відходити на другий план. Окрім того, інтерфейс сховищ даних зазвичай проектується в першу чергу для програмного використання. Як наслідок, такий інтерфейс незручно або навіть неможливо використовувати для безпосередньої взаємодії з даними.

Можливим способом полегшення взаємодії зі сховищами даних є створення інтерфейсів підтримки обробки природної мови для взаємодії з даними. Такі інтерфейси можуть використовуватись людьми без необхідних навичок. Також, такі інтерфейси дають можливість значно швидше генерувати запити до сховищ даних.

У роботі розроблено програмне забезпечення з використанням інтерфейсів природною мовою для побудування відображень даних у сховищі. Такий підхід значно зменшує кількість ресурсів, що витрачається на створення та підтримання таких відображень. Досліджено також й методи створення відображень для даних, що розподілені між декількома сховищами в рамках системи.

У ході виконання роботи розроблено програмне забезпечення, що дає можливість за допомогою текстових та голосових команд англійською мовою генерувати набори даних, що розподілені між декількома сховищами даних. Створені набори можуть використовуватись для подальшого аналізу за допомогою графічного інтерфейсу користувача.

Дослідження літератури за темою дисертації та існуючих рішень показало, що дана тема є актуальною та активно досліджується науковцями різних країн протягом більше ніж 40 років. Існуючі рішення у даній сфері активно розробляються та еволюціонують. Але, аналіз таких рішень показав, що жодне з них не готове до повноцінного вирішення завдань, що були поставлені та досліджені в ході роботи.

Метою роботи є покращення процесу взаємодії з базами та сховищами даних у спосіб розроблення програмного забезпечення, що реалізує зручний інтерфейс до баз і сховищ даних із застосуванням команд природною мовою. Об’єктом дослідження є інтерфейси до баз та сховищ даних з використанням природної мови.

Предметом дослідження є моделі, методи і технології обробки команд природною мовою з метою побудови інтерфейсу для підготовки набору даних з кількох джерел.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд об’єкту дослідження

У роботі розглянуто інтерфейси різноманітних сховищ даних, та способи використання таких інтерфейсів за допомогою інструкцій природньою мовою.

У сучасному світі комп’ютерних технологій, інформація є ключовим активом багатьох програмних продуктів. Для збереження інформації часто використовують різноманітні бази та сховища даних. У залежності від аспектів інформаційної системи, сховища даних використовують для забезпечення наступних вимог:

- надійне збереження даних;
- захищеність даних від несанкціонованого доступу;
- забезпечення доступності даних;
- забезпечення узгодженості даних та підтримка інваріанту системи;
- забезпечення швидкодії доступу до даних.

Для забезпечення описаних вимог, інформаційні системи можуть використовувати сховища різних типів. Серед найбільш розповсюджених типів сховищ даних можна виділити наступні:

- SQL бази даних;
- No-SQL бази даних;
- мережеві та локальні файлові системи;
- веб-орієнтовані сховища даних (по типу Google docs, Azure Blob Storage, та інші);
- програмні модулі в оперативній пам’яті обчислювальних машин;
- data warehouse, data lake;
- комбіновані сховища, що включають набір сховищ різних типів.

Кожне сховище даних повинно забезпечити певний інтерфейс, який можна використовувати для взаємодії зі цим сховищем даних. Такий інтерфейс може

набувати різну форму та характеристики. Це залежить від типу сховища, а також від функціональних та нефункціональних вимог, що задовольняються сховищем.

Можна виділити наступні форми інтерфейсів до популярних сховищ даних:

- графічний інтерфейс (GUI);
- розширення командного рядку (CLI);
- програмний інтерфейс додатку (API), що може бути побудованим з використанням різних протоколів комунікації, таких як: HTTP, TCP, WebSockets, AMQP, системні виклики операційних систем та інші;
- програмний модуль для взаємодії зі сховищем (SDK);
- інтерфейси з використанням команд природньою мовою, що будуть розглянуті далі в рамках цієї роботи.

Дані у сховищах можуть зберігатись у різних формах. Вибір форми може впливати на нефункціональні характеристики, такі як: використання дискового простору, швидкодія отримання та запису даних, простота запитів до сховища, та інші. У роботі більш детально розглянуто такі форми збереження даних, як data warehouse та data mart [1].

Data warehouse – сховище даних, що оптимізоване для аналітичних запитів та візуалізації даних за допомогою відповідного програмного забезпечення. Часто, дані у такому сховищі зберігаються з використанням топології зірки [2]. Data warehouse сховища зазвичай включають велику кількість даних, адже у них зберігаються дані з більшості модулів системи.

З розвитком хмарних технологій набувають все більшої популярності data warehouse рішення з Software-as-a-Service моделлю використання. Хмарні платформи реалізовують відповідне data warehouse сховище і дають користувачам можливість завантажувати туди власні дані. Як наслідок, користувачі мають змогу виконувати аналітичні запити до своєї інформації без необхідності розгортання та підтримки повноцінного data warehouse сховища. Такий підхід значно спрощує та

пришвидшує отримання аналітичних даних. Але, як правило, хмарні data warehouse сховища мають платну модель використання, та вимагають оплати за роботу з даними та виконання аналітичних запитів.

Вітрини даних (англ. Data mart) – спрощене відображення даних, що зберігаються у data warehouse. Зазвичай, data mart відображення будуються для використання певною групою користувачів. До прикладу, кожен відділ організації може користуватись окремим data mart до сховища даних. Серед переваг створення data mart сховищ можна виділити наступні:

- спрощене відображення даних у сховищі, що актуальне конкретній групі користувачів;
- можливість обмеження доступу до даних у сховищі;
- можливість оптимізації запитів, що використовуються data mart сховищем для отримання даних.

Архітектуру типового сховища з використанням data warehouse та data mart зображено на рис. 1.1.

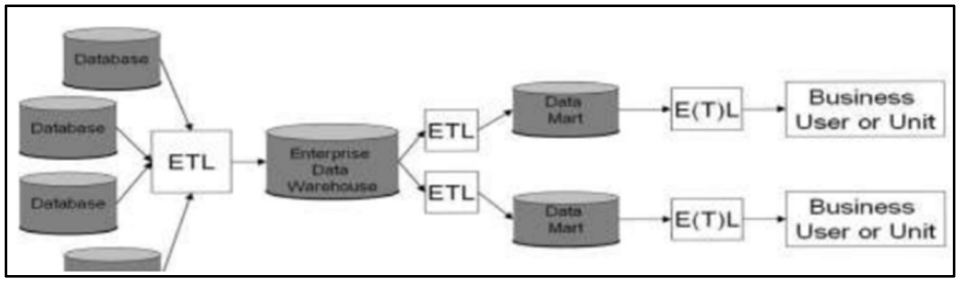


Рисунок 1.1 – Архітектура типового сховища з використанням data warehouse та data mart [1]

Усі інтерфейси сховищ даних можна розділити на дві групи за орієнтованістю на певний тип користування:

- ті, що орієнтовані на використання іншими програмами;
- ті, що орієнтовані на використання кінцевими користувачами.

У залежності від типу користування, вимоги до таких інтерфейсів можуть різнитись. До прикладу, інтерфейси для кінцевих користувачів мають бути зручними, зрозумілими та легкими у освоєнні. З іншої сторони, інтерфейси для програмного використання зазвичай оптимізуються для швидкодії та забезпечення надійності передачі даних.

Розповсюдженим підходом до побудови інтерфейсів до сховищ даних є клієнт-серверний підхід. Відповідно до цього підходу, саме сховище виступає сервером, а його інтерфейс – клієнтом. Часто, клієнт та сервер знаходяться на різних обчислювальних машинах та взаємодіють за допомогою мережі Інтернет. У такому випадку клієнт зазвичай знаходиться на одній машині з користувачем сховища, а сервер зі сховищем на окремій машині в датацентрі.

Багато сучасних сховищ даних реалізовують відразу декілька інтерфейсів, що мають різні характеристики. При цьому, розповсюдженою є практика, коли одні інтерфейси перевикористовують інші. Часто можна зустріти системи, де сервер сховища даних надає програмний інтерфейс, що використовується іншими інтерфейсами для взаємодії з кінцевими користувачами.

Варто також відмітити, що деякі інтерфейси можуть використовуватись як програмами, та і кінцевими користувачами. Яскравим прикладом таких інтерфейсів є SQL інтерфейси. Зазвичай, SQL використовується іншими програмами для взаємодії зі сховищем даних. Але, SQL синтаксис може застосовуватись й кінцевими користувачами для безпосередньої взаємодії зі сховищем.

У рамках даної роботи особливу увагу приділено інтерфейсам, що працюють з використанням команд природної мови. Такі інтерфейси надають можливість використовувати лінгвістичні конструкцій мов, що використовуються людьми в повсякденному житті [3]. Такими мовами можуть бути англійська, українська та інші мови. Оскільки англійська мова є де-факто основною мовою

інтернаціонального спілкування, більшість інтерфейсів природної мови використовують саме її.

Обробка команд природної мови – складний процес, що важко описати програмним кодом. Тому, для побудови алгоритмів такої обробки часто використовують методи машинного навчання. Предметна область обробки природної мови (англ. NLP – Natural language processing) є частою темою різноманітних досліджень, завдяки чому було створено велику кількість алгоритмів та масивів даних для такого навчання.

З найбільш розповсюджених варіантів використання методів обробки природної мови можна виділити наступні:

- пошукові системи (Google, Bing, та інші);
- переклад (Google translate);
- генерація контенту (GPT, DALLE);
- віртуальні асистенти (Siri, Amazon Alexa, ChatGPT);
- автодоповнення (Microsoft Word, Command line applications);
- перетворення контенту (наприклад, перетворення аудіозапису в текст);
- побудова інтерфейсів (різноманітні чат-боти).

Сфера обробки команд природної мови наразі динамічно розвивається. Отже, у цій сфері часто з'являються нові підходи та технологічні прориви.

1.2 Опис задачі, що досліджується

Розглянемо характеристики задачі розробки інтерфесів, що використовують команди природної мови.

1.2.1 Низька зручність публічних інтерфейсів для безпосередньої взаємодії з сховищами даних

Зазвичай при створенні сховища даних найбільша увага приділяється забезпеченню надійного збереження даних, швидкодії запитів та оптимізації використаних ресурсів обчислювальної системи. Зручність публічного інтерфейсу сховища даних може відходити на інший план. Через це значно зменшується ефективність використання інтерфейсу для безпосередньої взаємодії зі сховищем. Ця проблема ускладнюється також й тим, що багато баз даних проектується у першу чергу для використання іншими програмами. Якщо такі бази мають лише інтерфейс для програмної взаємодії, то такий інтерфейс неможливо використовувати кінцевим користувачам без спеціальних програм. Поява рішення, що буде реалізовувати зручний інтерфейс для широкого кола сховищ даних, дасть можливість збільшити ефективність безпосередньої взаємодії зі такими сховищами, а також зробить цю взаємодію більш зручною.

1.2.2 Наявність вимог до компетенцій користувача для використання інтерфейсу до сховища даних

Деякі інтерфейси сховищ даних дуже важко використовувати без спеціального навчання та підготовки. Як приклад можна привести інтерфейси реляційних баз даних. Як правило, такі інтерфейси використовують SQL у якості мови для написання запитів. Без знання синтаксису SQL хоча б на базовому рівні, користувач не зможе сформувати команди до сховища даних. А, значить, користувач не зможе використовувати публічний інтерфейс такої бази даних. Ця проблема поглиблюється також й тим, що формат інтерфейсу може відрізнятись в залежності від типу сховища. Тому, навіть маючи достатньо знань для використання інтерфейсів одного типу, користувач не зможе використовувати інші. До прикладу, навіть знаючи мову SQL, користувач наврядчи зможе одразу

розібратись у інтерфейсах більшості No-SQL баз даних. Поява більш простого інтерфейсу для взаємодії зі сховищами даних знизить поріг знань, що необхідні для взаємодії з такими сховищами. Як наслідок, більше зацікавлених людей зможуть взаємодіяти зі сховищем без попередньої підготовки.

1.2.3 Складність взаємодії з комбінованими сховищами даних, що складаються з композиції декількох сховищ

У складних та великих інформаційних системах, часто дані розподіляються між декількома сховищами, що працюють як одна система для роботи з даними. Такий підхід дає можливість покращити нефункціональні вимоги системи, а також реалізувати специфічні функціональні вимоги. Але, наявність декількох сховищ даних значно ускладнює аналітичну роботу з даними у цій системі. Адміністратору системи важко аналізувати дані, адже вони розподілені між декількома сховищами. Часто такі сховища мають різні інтерфейси для взаємодії. У такому випадку адміністратору доводиться самостійно поєднувати дані з різних сховищ. Це значно сповільнює та ускладнює аналіз даних. Наявність рішення, що буде допомагати у взаємодії з декількома сховищами даних, значно спростить роботу адміністратора системи. Як наслідок, адміністратор зможе працювати більш ефективно та швидше виконувати свої робочі задачі.

1.2.4 Велика кількість ручної роботи при створенні та налаштуванні відображень даних у сховищі

При створені інформаційних систем для підтримки бізнесу частою є практика створення окремих відображень даних у сховищі для різних бізнес-процесів або відділів у організаційній структурі компанії. Адміністратор сховища може створювати відображення у вигляді data mart для таких цілей. Але створення та підтримка таких відображень вимагає ручної роботи від адміністратора. Якщо

відображень достатньо багато – така ручна робота може займати багато робочого часу спеціаліста. Створення можливості автоматизувати цей процес зможе звільнити робочий час адміністратора для інших важливих задач.

Можливим вирішенням описаних проблем може стати система, що використовує методи обробки команд природною мовою для реалізації взаємодії між користувачем та сховищем даних. Робота з командами природною мовою буде створювати можливості для отримання наступних переваг:

- зручний та зрозумілий інтерфейс. Система буде надавати інтерфейс природною мовою (наприклад, англійською), що буде зрозумілий більшості користувачів. Більше того, користувачі зможуть давати команди як у текстовому форматі, так і аудіо форматі;
- відсутність вимог до попередньої підготовки для роботи з інтерфейсом. Природній інтерфейс можна використовувати навіть без спеціальних навичок. Користувач зможе надавати команди зручною для нього мовою (до прикладу, англійською). Але користувачу все ще потрібно буде давати команди у певній формі, щоб система отримувала достатньо інформації для повноцінної обробки команди;
- єдиний інтерфейс для взаємодії з різними сховищами даних. Система буде транслювати запити природної мови у відповідні запити до сховищ даних різних типів. Як наслідок, природну мову можна буде використовувати як єдиний інтерфейс для взаємодії з сховищами різних типів усередині інформаційної системи;
- автоматизація ручної роботи для адміністратора сховища. За допомогою інтерфейсу природною мовою адміністратор зможе значно швидше давати команди для створення та підтримки відображень даних у сховищі.

Спираючись на описані проблеми та можливі варіанти їх рішення, предметом дослідження магістерської дисертації вибрано підходи до побудови відображень у сховищах даних за допомогою інструкцій природною мовою.

1.3 Аналіз існуючих досліджень по темі роботи

Аналіз існуючих досліджень по темі наукової роботи виконувався з використанням інтернет-сервісів Google Scholar [4] та IEEE Xplore [5].

У рамках пошуку розглядались наукові роботи з предметом дослідження з наступних областей:

- інтерфейси природною мовою для сховищ даних;
- побудова відображень даних у вигляді data mart з використанням технологій обробки команд природною мовою;
- методи перетворення команд природною мовою у формальні команди сховищ даних;
- керування сховищами даних за допомогою голосових команд природною мовою.

У роботі [6] досліджено інтерфейси природньою мовою для сховищ даних, область їх застосування, а також переваги та недоліки таких інтерфейсів. Окрім того, у роботі описано методи побудови інтерфейсів та типові проблеми, що виникають при їх реалізації. Ця робота є цікавою для дослідження у першу чергу тому, що вона була видана у 1982 році, і вже тоді, 40 років назад, що проблема побудови інтерфейсів природної мови була актуальною. І до сьогодні ця проблема активно досліджується науковою спільнотою.

Робота [7] схожа за тематикою на роботу [6], вона дає більш розширений опис предметної області інтерфейсів природної мови для сховищ даних (NLIDB). У роботі [7] описано нюанси створення таких інтерфейсів, типові проблеми, приклади роботи, а також порівняння з класичними SQL інтерфейсами. Ця робота добре

підходить для ознайомлення з предметною областю та її типовими проблемами. Але при аналізі роботи варто зважати на дату публікації – 1995 рік. Оскільки пройшло вже багато років з дати публікації, деяка інформація у роботі вже втратила свою актуальність.

Більш актуальну інформацію про предметну область можна знайти у роботах [8], [9] та [10]. У цих роботах зроблено огляд найновіших підходів до розробки інтерфейсів природної мови, що були доступні на момент написання робіт. Важливим для аналізу є також дата виходу робіт – [7] вийшла у 1995 році, [8] у 2011 році, [9] у 2021 році та [10] у 2022. Усі три роботи описують поточний стан предметної області на момент написання. З цього можна зробити висновок про актуальність таких досліджень, адже результати цих досліджень цікавлять наукову спільноту протягом значного періоду часу.

У роботі [11] було досліджено методи побудови інтерфейсів природної мови для реляційних баз даних. Поміж іншого, у цій роботі було зображено найбільш популярні рішення з цієї сфери та роки виникнення цих рішень. Ця інформація може бути використаною для пошуку та аналізу існуючих рішень для проблем, що розглядаються у рамках магістерської дисертації. Діаграму з історією виникнення інтерфейсів природної мови (NLUI) для взаємодії з SQL базами даних наведено на рис. 1.2.

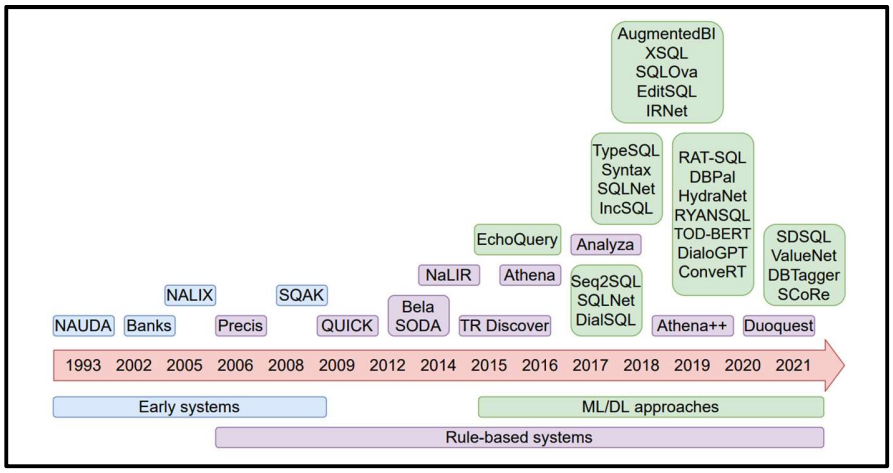


Рисунок 1.2 – Історія виникнення NLUI для взаємодії з SQL базами даних [11]

Також, у роботі [11] було виведено ідею про три підходи до перетворення команд природної мови у SQL команди. А саме, було виведено наступні підходи:

- перетворення на базі правил (rule-based) – розбиття команди на семантичні елементи та спроба отримати інформацію з цих елементів за допомогою бази даних можливих елементів;
- перетворення за допомогою моделей машинного навчання та технологій deep learning;
- гібридні підходи, де використовуються відразу моделі машинного навчання та бази знань з семантичними елементами мови.

Більш детально методи побудови NLUI для взаємодії з SQL базами були описані у роботах [12], [13], [14], [15], [16], [17], [18], [19] та [20]. Поміж іншого, у цих роботах можна знайти наступну інформацію:

- опис алгоритмів для побудови NLIDB;
- типові архітектури таких NLIDB;
- прототипи NLIDB;
- методи вимірювання ефективності створеного NLIDB;
- порівняння створеного NLIDB з класичними SQL інтефрейсами.

Якщо розглядати прототипи з описаних робіт у якості системи для вирішення проблем, що досліджуються у рамках магістерської дисертації, то можна виділити наступні переваги та недоліки такої системи:

Переваги:

- велика кількість наявних методів для побудови таких інтерфейсів;
- зручність використання порівняно з SQL інтерфейсами;

Недоліки:

- здатність працювати лише з одним типом сховищ;
- низька швидкодія запитів;

- необхідність додаткового навчання моделей для взаємодії з конкретним сховищем.

У будь-якому разі роботи містять інформацію, що буде корисною при проектуванні та реалізації програмного прототипу у рамках магістерської дисертації.

У роботі [21] було досліджено таку важливу та неочевидну проблему, як перетворення одиниць виміру при отриманні відповіді з сховища даних через NLIDB. Зазвичай, дані у сховищі зберігаються у певному форматі без явної прив’язки до одиниць виміру, але, при формуванні команд природною мовою, користувач може запросити дані у певній величині. До прикладу, користувач може запросити відстань між об’єктами у метрах, у той час як в базі даних ця відстань зберігається в сантиметрах. При проектуванні NLIDB потрібно враховувати цю особливість та коректно обробляти подібні запити від користувачів.

У роботах [22], [23] було описано створення графічного інтерфейсу для взаємодії зі сховищами даних за допомогою інструкцій природною мовою. У роботі [22] було реалізовано додаток для сімейства операційних систем Linux, а у роботі [23] було реалізовано веб-орієнтований інтерфейс. Аналіз цих робіт показав, що створення саме графічного інтерфейсу для взаємодії зі сховищем є виправданим з наступних причин:

- підвищена зручність використання інтерфейсу;
- тип інтерфейсу є звичним для користувачів без досвіду взаємодії із сховищами даних;
- значно легше генерувати команди у вигляді аудіо файлів;
- у випадку веб-орієнтованого інтерфейсу – можливість користуватись інтерфейсом з будь-якого пристрою у мережі без необхідності додаткової конфігурації цього пристрою.

У роботах [24], [25], [26], [27], [28] було більш детально розглянуто ідею побудови голосового інтерфейсу для взаємодії зі сховищами даних. У цих роботах описано різноманітні варіанти використання таких інтерфейсів, методи для побудови систем з використанням голосових інтерфейсів та розповсюджені проблеми при побудові таких систем. У результаті аналізу даних робіт можна зробити такі висновки щодо голосових інтерфейсів для сховищ даних:

- голосові інтерфейси добре підходять для взаємодії з нетехнічними користувачами;
- голосові інтерфейси часто використовуються на мобільних пристроях, де немає можливості вводити команди у формі тексту;
- у середньому, швидкість створення голосових команд є вищою за швидкість створення текстових команд. Тому, через голосовий інтерфейс користувач зможе надати більше команд за одиницю часу, але точність перетворення таких команд у запити до сховища даних буде нижчою. Як наслідок, коректність виконання наданих команд у голосового інтерфейсу буде нижчою, ніж у відповідного текстового інтерфейсу.

У роботах [29], [30], [31], [32] було досліджено проблему автоматизації створення data mart відображень за допомогою інструкцій природною мовою. Поміж іншого, у цих роботах було описано різноманітні варіанти використання систем з подібною автоматизацію, а також методи побудови таких систем. Опираючись на результати досліджень, можна зробити наступні висновки:

- подібна автоматизація може використовуватись для залучення людей «зі сторони бізнесу», без досвіду взаємодії зі сховищами даних , що будуть власноруч будувати для себе необхідне відображення даних для аналізу;
- автоматизація процесу створення data mart відображень може значно зменшити об’єм роботи, що необхідно робити адміністратору сховища даних для налаштування цих відображень;

- алгоритм створення data mart відображень сильно залежить від схеми даних, що зберігається в сховищі. Тому, буде складно побудувати алгоритм, що буде здатний будувати такі відображення для широкого класу сховищ.

Варто також зауважити, що англійська мова не є єдиною мовою, що може використовуватись у якості природної мови у інтерфейсі сховища даних. Це підтверджується наступними роботами, де було досліджено інтерфейси сховищ даних з використання іспанської [33], французької [34], китайської [35] та перської [36] мов.

1.4 Огляд існуючих рішень

У цьому розділі надано результат аналізу існуючих продуктів, що здатні вирішувати описані проблеми. У рамках розділу проаналізовано існуючі інтерфейси природної мови для сховищ даних. Також, проаналізовано сервіси, що використовують методи обробки природної мови для взаємодії з даними.

1.4.1 Пошукові системи

Пошукові системи по типу Wolfram Alpha [37] або Google [38] дають можливість отримувати дані на базі запитань, що побудовані природною мовою. Такі системи можна вважати специфічними інтерфейсами природної мови для взаємодії з даними.

Приклади інтерфейсів цих систем зображено на Рис. 1.3 та Рис 1.4.

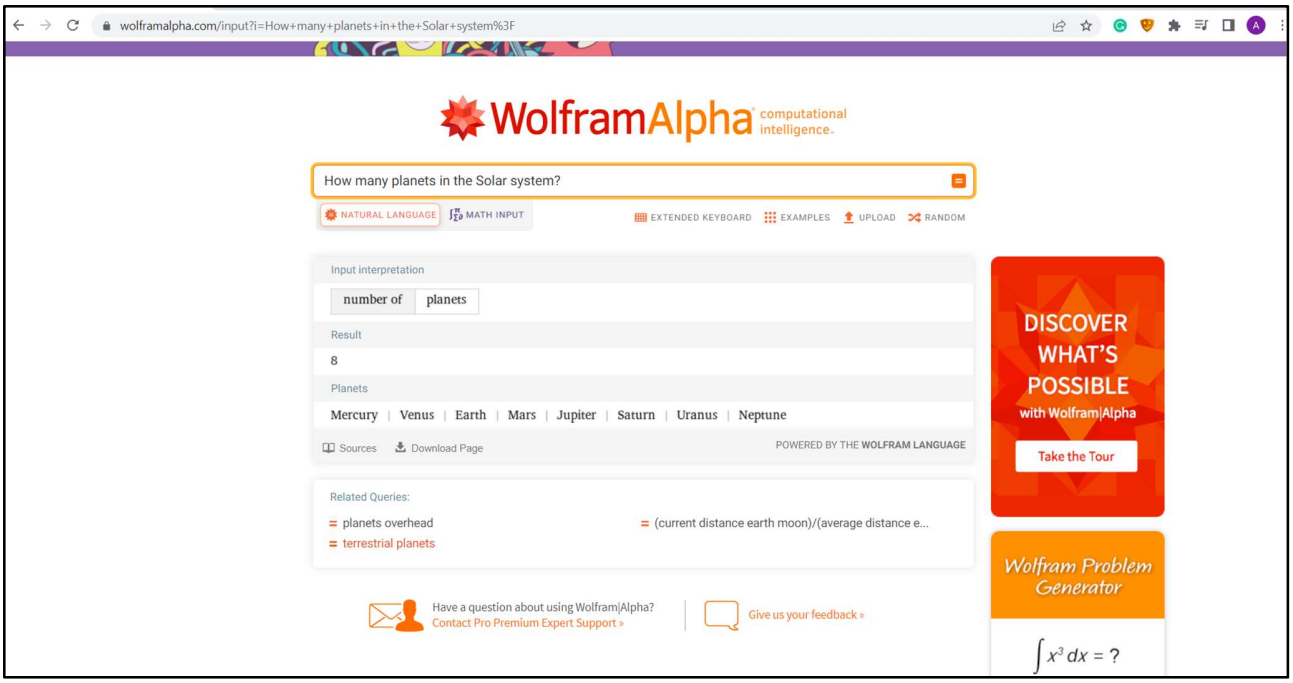


Рисунок 1.3 – Інтерфейс пошукової системи WolframAlpha

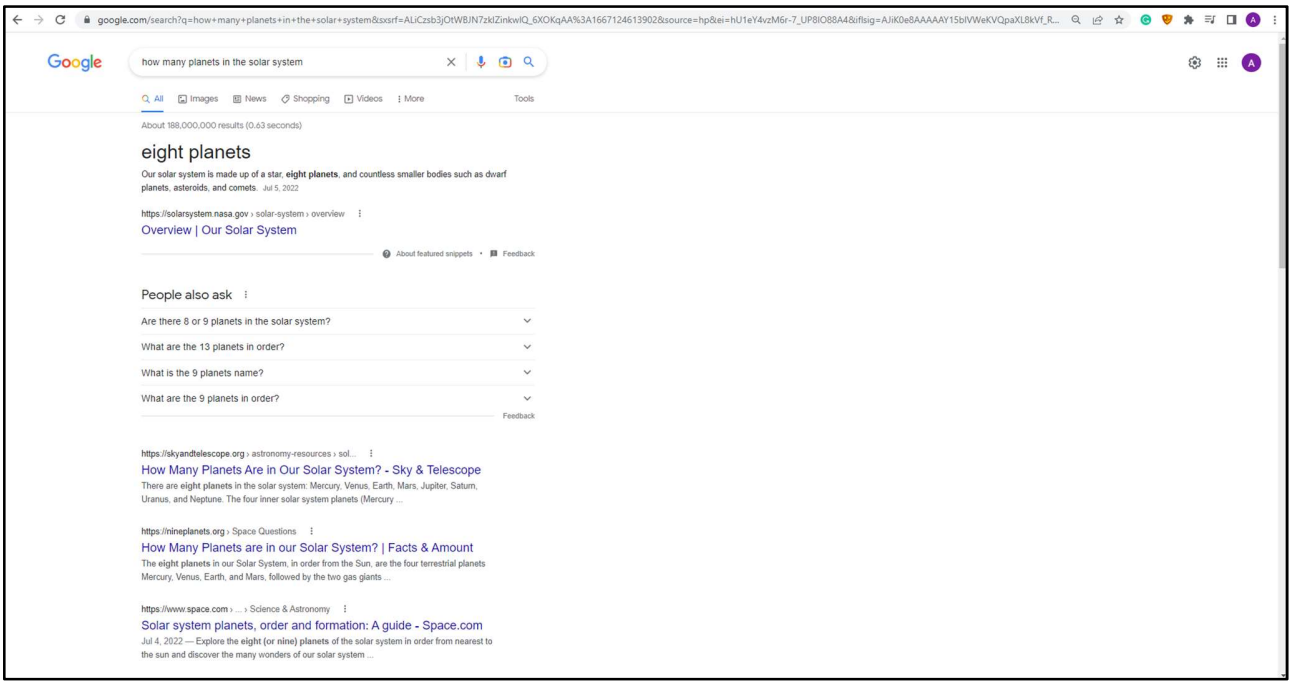


Рисунок 1.4 – Інтерфейс пошукової системи Google

Якщо розглядати згадані пошукові системи як інтерфейси природною мовою для сховищ даних, то можна виділити наступні переваги та недоліки:

Переваги:

- широка підтримка різноманітних лексичних конструкцій та варіантів запитань;
- підтримка багатьох природніх мов;
- велика база даних;
- швидкодія відповіді;
- у випадку Google – можливість надавати запити у різних форматах: текст, аудіо, фото;

Недоліки:

- робота із заздалегідь визначеною базою даних, що контролюється пошуковою системою;
- неможливість розширювати базу власними даними.

Враховуючи переваги та недоліки, можна зробити наступні висновки:

- пошукові системи гарно підходять для взаємодії із загальновідомою та публічною інформацією. Такі системи надають зручний інтерфейс для взаємодії з такою інформацією за допомогою природної мови;
- пошукові системи погано підходять у якості інтерфейсу для комерційних сховищ даних, адже немає можливості використати такі системи з даними зі конкретного сховища.

1.4.2 Розумні асистенти

Стрімкий розвиток технологій у машинному навчанні на базі Large Language Models (LLM) моделей сприяв появі розумних асистентів, що здатні опрацьовувати запити користувачів природною мовою. Такі асистенти зазвичай працюють у форматі чату, виконуючи роль розумного співрозмовника-помічника. По суті, завдяки навчанню на великій кількості текстових даних з різних тем, помічники на базі Large Language Models здатні дуже точно підбирати наступне слово у розмові

з користувачем. Підбираючи наступне слово одне за одним, вони генерують повноцінні відповіді на запит користувача.

Найбільш популярним на даний момент асистентом є продукт ChatGpt [39] від компанії OpenAi. Цей продукт працює на базі моделей GPT, що є провідними серед існуючих LLM моделей. Продукт має безкоштовну версію, що працює на базі моделі GPT-3.5. Завдяки високій ефективності моделей сімейства GPT та наявності безкоштовної версії продукт ChatGpt набув широкої популярності серед користувачів у всьому світі. Приклад використання ChatGpt зображено на рис. 1.5.

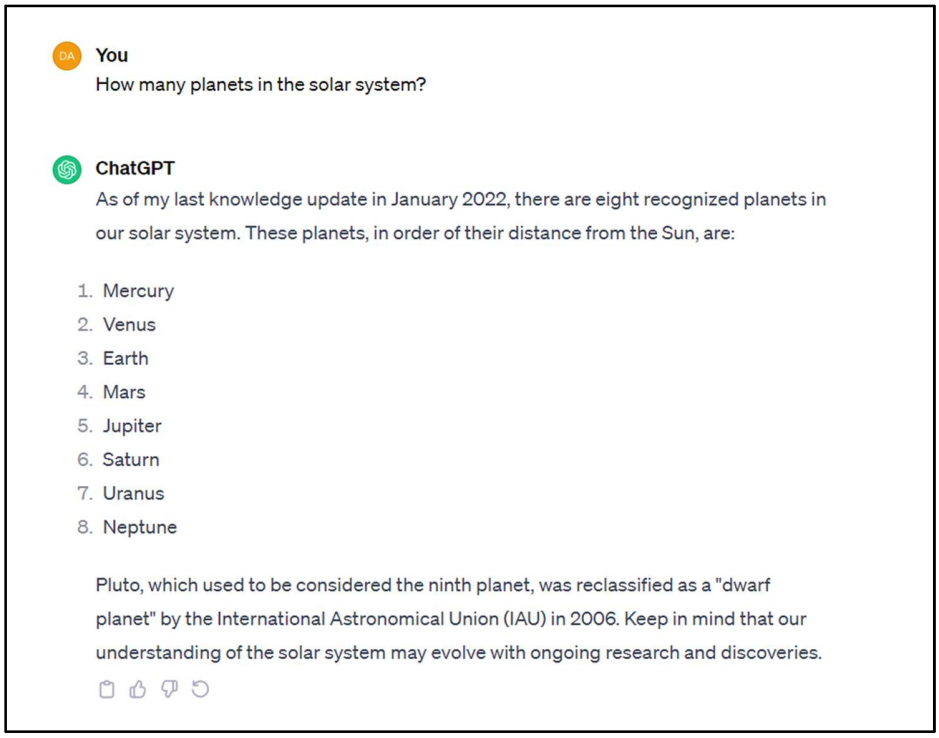


Рисунок 1.5 – Приклад використання продукту ChatGpt

Одним з найбільш помітних конкурентів ChatGpt є продукт Bard від компанії Google [40]. Перевагою цього продукту є можливість отримувати актуальну інформацію щодо теми запиту за допомогою пошуку в мережі Інтернет. У той самий час, ChatGpt не має такої можливості, тому цей асистент вимушений працювати на базі інформації, що обмежена в часі моментом навчання моделі (на момент виконання дослідження, ChatGpt здатний оперувати інформацією про події,

що відбулись не пізніше січня 2022-го року). З іншої сторони, ChatGpt показує кращі результати під час аналізу запиту користувача та роботи власне з командами природною мовою. Також, ChatGpt краще розпізнає контекст розмови, що робить його більш схожим на справді інтелектуального помічника.

Приклад використання продукту Bard зображено на рис. 1.6.

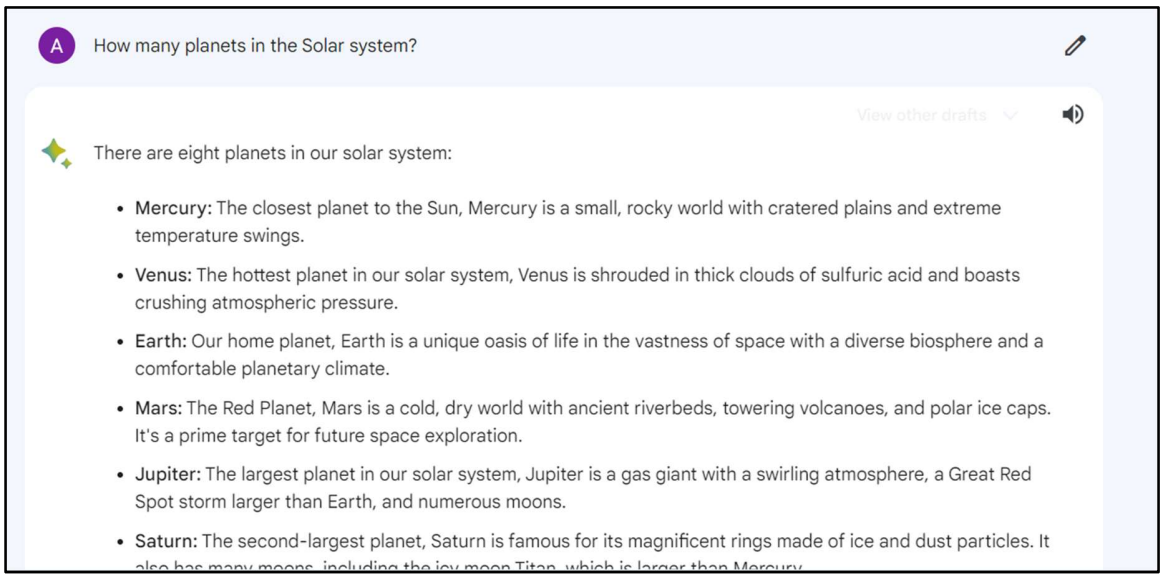


Рисунок 1.6 – Приклад використання продукту Bard

Розглядаючи розумних асистентів у контексті інтерфейсів природної мови для взаємодії з даними, можна виділити наступні переваги та недоліки використання таких асистентів у якості описаних інтерфейсів:

Переваги:

- велика база даних та ширина знань – згадані асистенти здатні успішно відповідати на запитання з багатьох загальновідомих тем;
- ефективна обробка запитів природною мовою – завдяки навчанню на великому наборі даних, в описаних асистенти показують здатність «зрозуміти» запит користувача та згенерувати коректну відповідь за рахунок цього;

- підтримка багатьох природних мов, включаючи англійську та українську мови;
- використання «найкращих практик» у відповідях – завдяки навчанню на великій кількості існуючих знань, описані асистенти схильні надавати користувачу знання та підходи, що були перевірені на практиці та добре показали себе у використанні раніше;

Недоліки:

- робота тільки з публічними даними – як правило, описані асистенти працюють лише з загальновідомою інформацією;
- схильність до «галюцинацій» – оскільки асистенти по суті генерують дані на льоту, немає можливості гарантувати коректність цих даних. Як правило, згенеровані дані коректні у більшості випадків (адже вони згенеровані на базі існуючих, справжніх даних). Але іноді асистент може генерувати некоректні дані. До прикладу, асистент може посилатись на наукові статті, що ніколи не були видані. Тому отримані відповіді від розумних асиснентів, як правило, вимагають додаткової перевірки від компетентної особи перед їх використанням у якості коректних даних;
- обмеження на безкоштовне користування – як правило, безкоштовні версії розумних асистентів мають ліміти на використання, що обмежує таке використання у великих масштабах;
- ефективність роботи асистента залежить від формату запиту, що було надано на обробку – оскільки асистент використовує запит користувача як базу для генерації відповіді, якість отриманої відповіді сильно залежить від формату та інформації, що надано у запиті користувача. Тому, користувач має володіти певними навичками формування команд для отримання коректних результатів.

Враховуючи описані переваги та недоліки, можна зробити такі висновки. Розумні асистенти добре підходять для пошуку загальновідомої інформації за допомогою команд природної мови. Такі асистенти здатні швидко відповідати на запитання з різних тем та галузей. Але такі асистенти мають певні недоліки, що ускладнюють їх використання у якості інтерфейсів безпосередньо для інформаційної системи. Серед таких недоліків можна виділити схильність до галюцінацій, та вимоги до наявності навичок написання запитів для отримання коректних результатів. Описані недоліки вимагають залучення експертів для використання таких інтерфейсів та перевірки результатів, що значно ускладнює використання. Також, негативним фактором є робота асистентів на базі заздалегідь визначеного набору навчальних даних, що обмежені загальновідомою інформацією. Така робота унеможливлює використання асистентів для взаємодії з приватними даними, що зберігаються в сховищах інформаційних систем.

Як наслідок, описані розумні асистенти на базі LLM моделей погано підходять для використання у якості інтерфейсів для підготовки набору даних з різних джерел за допомогою команд природною мовою. Але, розглядаючи технології LLM моделей у цілому, такі моделі можуть стати у нагоді для обробки команд природної мови та перетворення їх у більш формальні команди, що будуть оброблені іншими модулями системи. LLM моделі показують високу ефективність взаємодії з природною мовою в рамках роботи розумних асистентів, тому вони гарно підійдуть і для роботи з природною мовою в рамках інтерфейсів природної мови до сховищ даних.

1.4.3 Github Copilot [41]

Сервіс Github Copilot позиціонується як розумний помічник для розробників програмного забезпечення. Цей сервіс використовує моделі глибокого навчання, що навчені на великій кількості програм з відкритим кодом. Завдяки такому навчанню

сервіс може генерувати та доповнювати програмні елементи різного рівня: окремі команди, функції, класи або цілі програмні модулі. Поміж іншого, Copilot може генерувати й команди у синтаксисі інтерфейсів різних сховищ даних. Тому цей сервіс можна використовувати для перетворення команд природної мови у формальні команди до сховища даних.

Github Copilot розповсюджується як розширення до популярних середовищ розробки. Поміж іншого, цей сервіс можна встановити до середовища Visual Studio Code [42], за допомогою якого можна надсилати команди до різних сховищ даних.

Завдяки Github Copilot, розробник або адміністратор системи може використовувати команди природної мови для взаємодії зі сховищами даних. Алгоритм обробки запитів природною мовою схематично зображено на рис. 1.7.

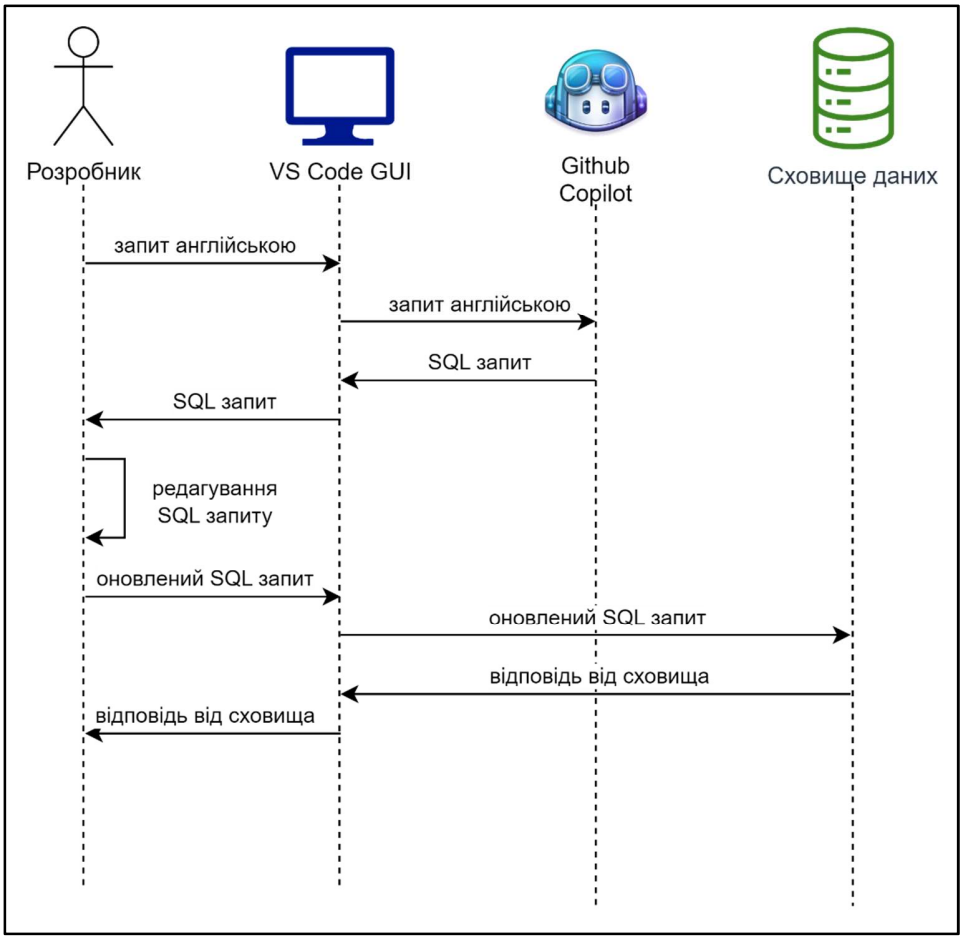


Рисунок 1.7 – Алгоритм обробки запиту за допомогою Github Copilot

У якості вхідних даних Copilot очікує програмний файл, що має бути розширений. У залежності від наповнення файлу, Copilot розширює наданий файл необхідними елементами мови SQL. Команди природною мовою можна описувати за допомогою коментарів у програмному файлі. Приклад використання сервісу Github Copilot зображено на рис. 1.8. Зелений текст – коментар, що був заданий до початку роботи з сервісом. Синій текст – контексті підказки середовища розробки VS Code. Сірий текст – частина SQL запиту, що згенерована сервісом. Copilot генерує запити ітеративно, тому він пропонує лише частину запиту в певний момент часу.

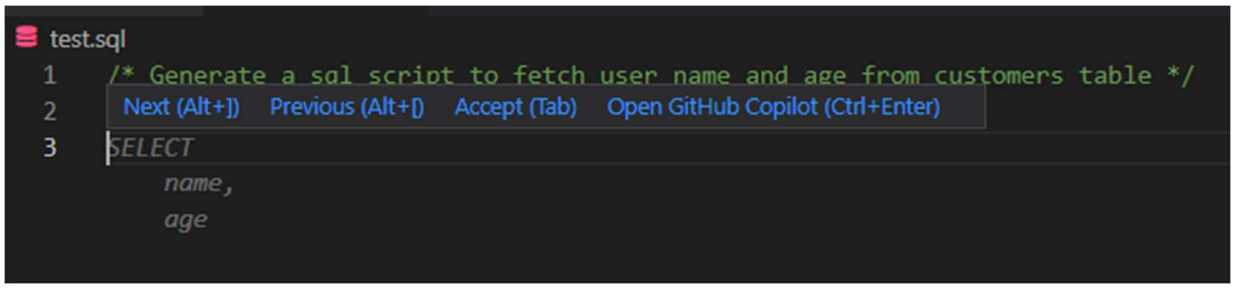


Рисунок 1.8 – Приклад використання сервісу Github Copilot

Серед переваг та недоліків цього сервісу можна виділити наступні:

Переваги:

- сервіс може генерувати запити до сховищ даних багатьох типів;
- сервіс інтегрований у середовище розробки;
- сервіс може не тільки генерувати запити з нуля, а й доповнювати вже існуючі;
- безкоштовне використання протягом певного періоду;

Недоліки:

- сервіс може використовуватись лише з середовища розробки, його не можна підключити напряму до сховища;
- сервіс орієнтований на розробників, користувачу без досвіду взаємодії зі сховищем даних буде важко взаємодіяти з ним;

- сервіс «нічого не знає» про схему даних у сховищі, для якого генеруються запити, тому згенеровані запити потребують додаткової перевірки з боку користувача;
- для коректної генерації запитів користувач має доволі детально описувати запит за допомогою природної мови.

З описаних переваг та недоліків випливає наступний висновок: Github Copilot добре підходить для використання розробниками у якості помічника для написання запитів. Якщо команди природною мовою написано коректно, сервіс може значно пришвидшити написання запитів до сховища даних, але сервіс погано підходить для роботи з сховищами поза локальним середовищем розробника.

1.4.4 English query in MS SQL Server

База даних SQL Server від компанії Microsoft [43] у ранніх версіях мала інтерфейс для взаємодії зі сховищем за допомогою команд природною мовою під назвою English query. Цей інтерфейс використовував інформацію про схему даних для більш точної генерації запитів. На рис. 1.9 зображено приблизну архітектуру цього інтерфейсу.

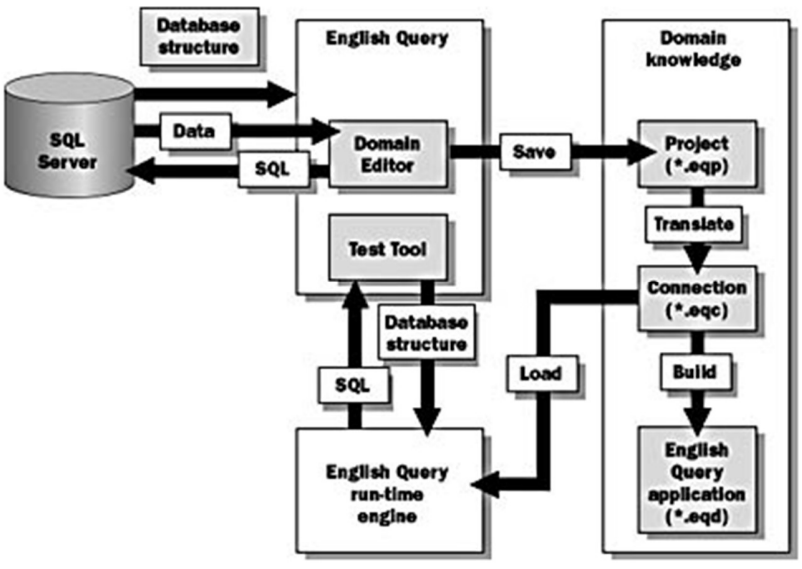


Рисунок 1.9 – Архітектура модуля English Query у СУБД MS SQL Server [44]

Компанія Microsoft вирішила не підтримувати цей інтерфейс починаючи з версії СУБД SQL Server 2005. Відповідно до інформації з цього питання на StackOverflow [45] було вирішено припинити підтримку цього інтерфейсу через малу кількість користувачів, що були зацікавлені в ньому. На даний момент пошук інформації про інтерфейс English Query на сайті документації СУБД SQL Server не дає результатів.

1.4.5 Power BI from Microsoft

Продукт Power BI від компанії Microsoft [46] створений для взаємодії з даними для отримання інформації, що потрібна для прийняття ефективних бізнес рішень. Одним з інтерфейсів для взаємодії з даними є Q&A інтефрейс [47], що дає можливість взаємодіяти з даними за допомогою запитів природною мовою. Надаючи команди природною мовою, користувач може отримати необхідні відображення та візуалізації на базі даних, що потрапляють в PowerBI з різних джерел всередині компанії.

Приклад використання інтерфейсу Q&A в PowerBI зображено на рис. 1.10.

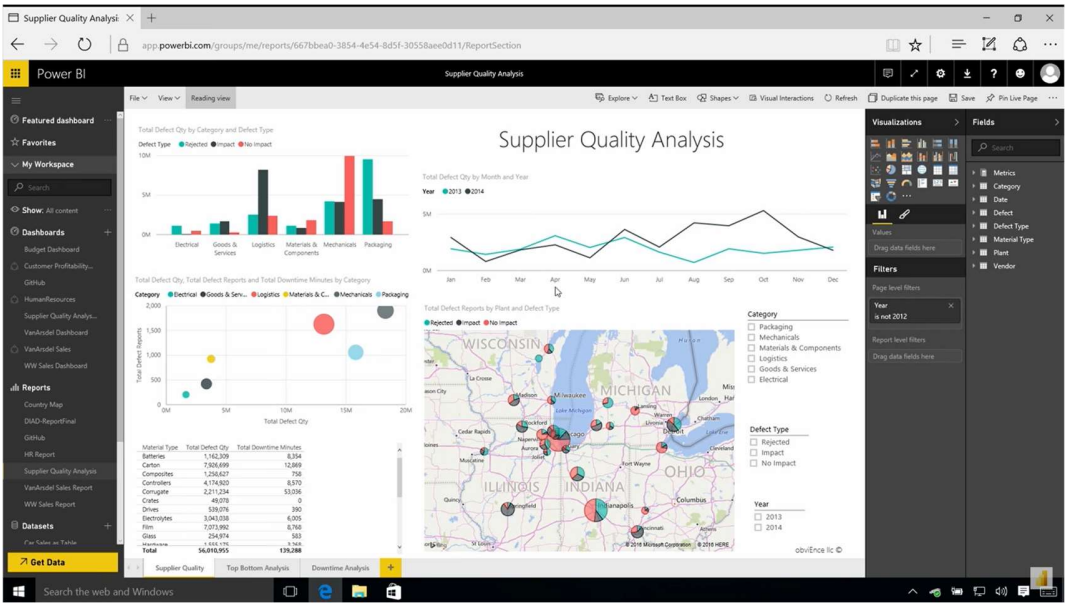


Рисунок 1.10 – Приклад використання інтерфейсу Q&A у системі PowerBI [47]

Серед переваг та недоліків цієї системи можна виділити наступні:

Переваги:

- можливість підключати різноманітні джерела даних;
- система автодоповнення, що допомагає будувати коректні запити;
- можливість об’єднувати дані з різних джерел;
- велика кількість форматів візуалізації даних;

Недоліки:

- відсутність безкоштовної версії;
- відсутність можливості отримати агреговані дані у «чистому» вигляді для подальшої роботи з ними за допомогою інших сервісів.

Як наслідок, Q&A інтерфейс у сервісі PowerBI гарно підійде для використання у компаніях, для яких важливо будувати велику кількість різноманітних відображень та візуалізацій для даних. Але використання цього сервісу вимагає платної ліцензії, що може стати проблемою для малих компаній та індивідуальних користувачів.

1.4.6 SpotIQ від компанії ThoughtSpot

Продукт SpotIQ [48] дає можливість підключитись до існуючого сховища даних та використовувати команди природною мовою для взаємодії з ним. Цей продукт може отримувати команди природною мовою та будувати візуалізацію даних на базі результату виконання наданої команди. Надані команди розбиваються на симантичні елементи, що використовуються для подальшого аналізу. Побудована візуаліазція відштовхується від цих елементів, та дають можливість переглянути інформацію відносно окремих елементів. Інтерфейс продукту зображено на рис. 1.11.

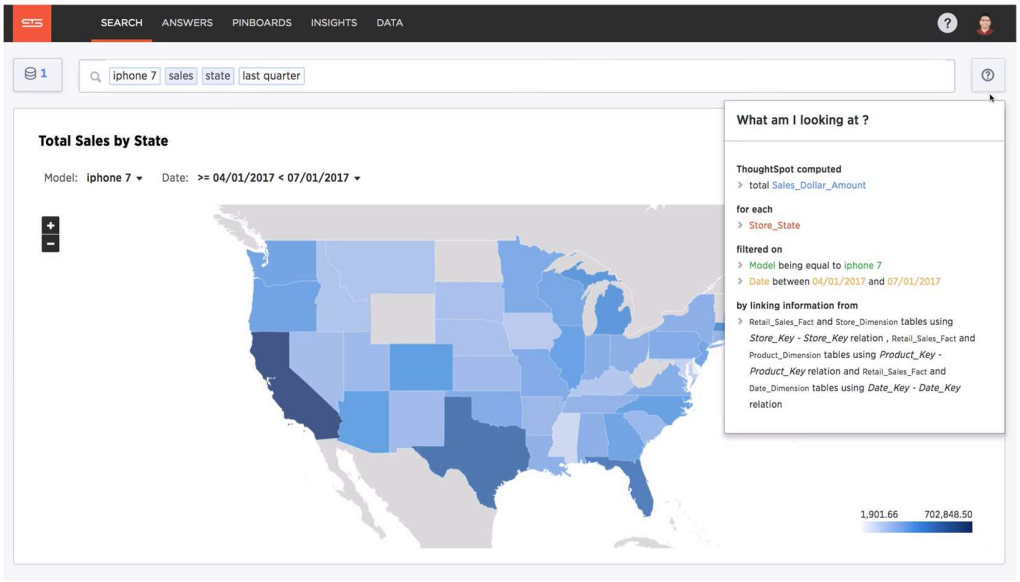


Рисунок 1.11 – Інтерфейс продукту SpotIQ [48]

Можна виділити наступні переваги та недоліки даного продукту:

Переваги:

- наявність у відповіді додаткової інформації, що побудована за допомогою моделей глибокого навчання;
- можливість подивитись семантичні елементи наданої команди, що дає змогу перефразувати задану команду для кращої обробки;
- наявність великої кількості вбудованих візуалізацій;
- можливість підключитись до наявного сховища даних;

Недоліки:

- відсутність безкоштовної версії цього продукту [49];
- обмежена кількість сховищ, що підтримуються [49].

Враховуючи описані переваги та недоліки, можна зробити висновок: система SpotIQ добре підходить для створення візуалізацій та розумного аналізу даних, що зберігаються у сховищі компаній середнього та великого розміру. Даний продукт погано підходить для малих компаній та індивідуальних користувачів через високу ціну ліцензії. Також, цей продукт можна використовувати лише з обмеженою кількістю сховищ, що може стати проблемою для деяких компаній.

1.4.7 Text-to-SQL прототипи

У ході аналізу існуючих досліджень за темою магістерської дисертації, знайдено дослідження з реалізованими прототипами. Серед найбільш цікавих для аналізу можна виділити наступні: [22], [50], [51], [52], [14].

Прототипи з цих робіт створені для перетворення команд природною мовою у SQL запити. Ці прототипи являють собою або власне модель машинного навчання, що здатна робити перетворення, або модель та примітивний інтерфейс для взаємодії з нею.

Якщо розглядати ці прототипи як інструменти для взаємодії зі сховищами даних за допомогою команд природною мовою, то можна виділити наступні переваги та недоліки використання таких інструментів:

Переваги:

- простота інтеграції в існуючі системи;
- наявність опису процесу розробки таких інструментів, що спрощує їх підтримку та модифікацію;

Недоліки:

- відсутність можливості використання таких інтерфейсів з сховищами даних, що не підтримують інтерфейс SQL запитів;
- відсутність повноцінного інтерфейсу для взаємодії з кінцевим користувачем;
- необхідність побудови інфраструктури, що дасть можливість використання цих інструментів для побудови повноцінного інтерфейсу природної мови для сховища даних.

Описані прототипи можуть використовуватись як гарний фундамент для розробки більш складних та повноцінних продуктів, але такі прототипи вимагають значного допрацювання перед використанням у якості повноцінного інтерфейсу для сховища даних.

1.5 Оцінка актуальності обраної теми

Аналіз наукових робіт по темі роботи та існуючих рішень показав наступне:

a) вибрана тема активно досліджується науковою спільнотою в різних країнах світу;
b) створення програмних рішень у вибраній сфері є затребуваним, оскільки існує велика кількість рішень, що активно розвиваються;
c) проаналізовані програмні продукти не є ідеальним для вирішення проблем, що розглядаються у рамках магістерської дисертації. Кожне з рішень має певні недоліки, що були описані у розділі;
d) NLIDP інтерфейси мають ряд переваг, що виділяють їх з-поміж класичних інтерфейсів до сховищ даних, а саме:
    1) зручність та швидкість використання;
    2) здатність автоматизувати рутинну роботу адміністратора сховища;
    3) зменшений поріг необхідних знань для використання інтерфейсу, що відкриває можливість використання для людей без спеціальної підготовки та навчання.
e) гарним доповненням до існуючих рішень у цій області буде система, що відповідає таким вимогам:
    1) здатність автоматизувати створення різноманітних data mart відображень у сховищах даних;
    2) інтерфейс природної мови з можливістю голосового та\або текстового вводу команд;
    3) здатність працювати зі сховищами різних типів;
    4) здатність працювати з системами, у яких використовується відразу декілька сховищ для збереження даних;

5) наявність веб-орієнтованого інтерфейсу, що дасть можливість використовувати систему через мережу Інтернет за допомогою пристроїв різного типу (комп’ютери, телефони, планшети та інше).

З описаного вище випливає, що вибрана тема дослідження є актуальною та затребуваною в сучасному житті. Ідея створення веб-орієнтованої системи з використанням інтерфейсу природної мови для взаємодії з сховищами даних є доцільною у даній предметній області.

1.6 Постановка задачі

Метою роботи є покращення процесу взаємодії з базами та сховищами даних у спосіб розроблення програмного забезпечення, що реалізує зручний інтерфейс до баз і сховищ даних із застосуванням команд природною мовою. Для досягнення мети необхідно вирішити такі задачі:

- аналіз існуючих сховищ даних та інтерфейсів для взаємодії з ними;
- аналіз методів для побудови інтерфейсів природної мови;
- формування вимог до програмного забезпечення;
- проектування архітектури додатку;
- реалізація прототипу запропонованої системи;
- тестування та оцінка ефективності створеної системи, формування висновків;

Реалізоване програмне забезпечення має надавати такі можливості:

- можливість формувати набори даних за допомогою команд природною мовою;
- можливість підготовки набору даних на базі композитного сховища, що складається з декількох джерел даних; наявність спільного інтерфейсу для взаємодії з таким сховищем;

- можливість задання команд природною мовою як за допомогою голосу, так і за допомогою тексту;
- можливість перегляду підготовлених наборів даних за допомогою зручного графічного інтерфейсу.

1.7 Висновки по розділу

У розділі надано загальний опис предметної області та проблем, що вирішуються у рамках роботи. Також, у цьому розділінадано результати дослідження літератури та існуючих рішень у вибраній сфері. У результаті проведеного дослідженняпроаналізовано актуальність вибраної теми та сформульовано мету, цілі та задачі роботи.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Визначення вимог до системи

Опираючись на проведений аналіз предметної області та поставлені цілі роботи, можна сформувати набір функціональних вимог до системи у форматі user story [53]. У цей набір будуть входити наступні вимоги:

- як користувач, я хочу мати веб-орієнтований графічний інтерфейс, щоб мати змогу взаємодіяти зі системою у зручний для мене спосіб;
- як користувач, я хочу мати адаптивний інтерфейс системи, щоб мати змогу використовувати його на різних пристроях (комп'ютерах, планшетах, телефонах) без необхідності додаткового налаштування;
- як користувач, я хочу готувати набори даних за допомогою команд природною мовою, щоб пришвидшити мою взаємодію зі сховищем даних;
- як користувач, я хочу взаємодіяти зі системою за допомогою команд англійською мовою, щоб мої колеги з інших країн теж могли доєднатись до роботи;
- як користувач, я хочу задавати команди за допомогою голосу, щоб я міг формувати команди без доступу до клавіатури;
- як користувач, я хочу задавати команди у вигляді тексту, щоб мати можливість більш детально описати поставлену задачу;
- як користувач, я хочу переглянути та редагувати згенерований запит перед його виконанням на стороні сховища даних, щоб мати можливість виправити можливі помилки та зробити точкове налаштування відповідного запиту;
- як користувач, я хочу формувати набори даних на базі інформації з різних джерел (до прикладу, різних баз даних), що працюють в рамках однієї

інформаційної системи, щоб мати змогу аналізувати дані усієї системи як одне ціле;

- як користувач, я хочу мати спільний інтерфейс для взаємодії з різними джерелами даних в рамках системи, щоб мати змогу за допомогою відповідного інтерфейсу отримати дані з будь-якого джерела;
- як користувач, я хочу переглядати сформовані набори даних за допомогою веб інтерфейсу, щоб мати змогу виконувати відповідний аналіз результатів;
- як адміністратор, я хочу мати змогу підключати та відключати джерела даних у системі, щоб надати користувачу можливість виконувати запити на базі актуальної інформації;
- як адміністратор, я хочу мати змогу налаштовувати параметри обробки команд природної мови, щоб мати можливість підвищити ефективність обробки команд за допомогою додаткового налаштування.

Серед найбільш важливих нефункціональних вимог можна виділити такі:

а) ефективність перетворення команд природної мови у формальні команди для сховища даних. Серед важливих характеристик ефективності такого перетворення можна виділити наступні:
    1) швидкодія системи під час перетворення команд природної мови;
    2) кількість помилок під час перетворення команд природної мови;
    3) вартість обробки однієї команди природної мови;
    4) швидкодія виконання згенерованої формальної команди на стороні сховища даних;
    5) кількість помилок під час виконання згенерованої формальної команди на стороні сховища даних;

6) ступінь відповідності отриманих результатів до початкового запиту користувача;

b) швидкодія системи при роботі з командами природної мови;

c) інтуїтивно зрозумілий графічний інтерфейс для підготовки наборів даних;

d) легке розширювання та підтримування створеного програмного забезпеення.

2.2 Опис запропонованих архітектурних рішень

Для реалізації описаних функціональних та нефункціональних вимог, запропоновано використати наступні архітектурні рішення при реалізації відповідного програмного забезпечення:

- реалізувати систему у вигляді двох додатків, а саме: frontend додаток, що відповідає за графічний інтерфейс користувача, та backend додаток, що відповідає за логіку виконання команд природної мови і взаємодію зі сторонніми системами;
- frontend додаток реалізувати у вигляді адаптивного Single Page Application додатку, що допоможе забезпечити вимоги щодо зручності використання, швидкодії та доступності графічного інтерфейсу користувача;
- backend додаток реалізувати у вигляді веб-серверу, що надає HTTP API інтерфейс для взаємодії з frontend додатком;
- використати підхід data warehouse для об’єднання даних з декількох джерел у одну систему з однорідним інтерфейсом. У якості мови запитів до цього інтерфейсу використати мову SQL. Таким чином, data warehouse сховище буде відповідати за об’єднання даних з різних джерел, та буде надавати однорідний інтерфейс для взаємодії з цими даними. У той самий час, створене програмне забезпеення буде отримувати команди

природною мовою та транслювати ці команди у відповідні команди об’єднаного інтерфейсу data warehouse сховища;

- у рамках backend додатку реалізувати модуль обробки команд природної мови. Використовувати цей модуль для перетворення команд природної мови у формальні SQL команди, що будуть надіслані до data warehouse сховища з ціллю отримання результуючих наборів даних;
- у рамках модуля обробки команд природної мови, реалізувати можливість перетворення голосових команд природною мовою у текстові команди. Таким чином, з’являється можливість використовувати механізм перетворення текстових команд у SQL як для текстового вводу, так і для голосового вводу (який буде заздалегідь перетворюватись у формат текстового вводу);
- для забезпечення доступності створеного програмного забезпечення, реалізувати розгортання створених frontend та backend додатків на базі хмарних платформ.

2.3 Вибір технологій для реалізації data warehouse сховища

Відповідно до описаних раніше архітектурних рішень, програмне забезпечення потребує реалізації data warehouse сховища для об'єднання інформації з декількох джерел даних у загальну інформаційну систему зі спільним інтерфейсом.

Серед вимог до запланованого data warehouse рішення варто виділити наступні:

- можливість об’єднувати дані з декількох джерел у спільний інтерфейс доступу до даних;
- наявність механізмів інтеграції з сховищами даних різних типів (реляційні бази даних, нереляційні бази даних, файлові сховища, веб сервіси і тд);

- підтримка мови SQL у якості мови запитів до сховища даних;
- наявність програмного інтерфейсу для взаємодії зі сховищем даних через мережу Інтернет;
- перевагою буде можливість експортувати схему даних об’єднаного сховища, з ціллю використання цієї схеми під час перетворення команд природної мови у SQL команди;
- перевагою буде наявність інфраструктури для моніторингу та відлагодження згенерованих команд.

Можна виділити 2 шляхи для реалізації такого data warehouse сховища: створення власного сховища, або використання існуючих рішень від постачальників хмарних рішень. У рамках даної роботи, більш доцільним є шлях з використанням існуючого сховища від постачальників хмарних рішень. Поміж іншого, на це вибір впливають наступні фактори:

- використання існуючих рішень вимагає значно менше ресурсів на розробку та впровадження;
- розгортання власного сховища вимагає значних обчислювальних ресурсів для підтримки такого сховища. У той самий час, сховище на базі хмарних рішень використовує потужності хмарної платформи;
- хмарні платформи пропоную безоплатне використання відповідних сховищ даних за умови притримання певних лімітів користування. У рамках даної роботи, створений програмне забезпечення не буде перебільшувати дані ліміти;
- рішення на базі хмарних технологій пропонують вже реалізовані модулі для експорту даних з сховищ різних типів, що є важливим аспектом у рамках даної роботи;

- сховища на базі хмарних рішень гарно протестовані та перевірені на практиці багатьма користувачами, що забезпечує зменшення ризику отримання помилок при обробці запитів користувача.

Враховуючи описані вище фактори, прийнято рішення використати стороннє data warehouse сховище на базі хмарних платформ.

У результаті, проаналізовано такі існуючі хмарні data warehouse рішення:

- рішення Snowflake від однойменної компанії Snowflake [54]. Дане рішення є одним з найбільш популярних data warehouse сховищ, але це рішення не підтримує роботу в Україні. Під час реєстрації, після вибору регіону «Україна», система вказує відповідну помилку щодо відсутності підтримки в вибраному регіоні. Тому дане рішення не підходить для використання у рамках розроблюваного програмного забезпечення;
- рішення на базі сервісів Azure від компанії Microsoft [55] Дане рішення по суті є композицією декількох сервісів Azure, що у результаті дають можливість побудувати data warehouse систему. Тобто, при виборі даного рішення, доведеться докласти окремі зусилля для реалізації сховища на базі запропонованих сервісів. Відсутність готового рішення для вирішення поставленої задачі є значним недоліком в контексті розроблюваного програмного забезпечення. Адже, необхідність самостійного допрацювання системи ускладнює реалізацію програмного забезпечення. Також, це робить систему менш надійною, збільшуючи ризик помилок при обробці запитів користувача;
- Amazon Redshift від компанії Amazon [56] Дане рішення гарно підходить для рішення задач, поставлених в рамках магістерської дисертації. Це сховище здатне об’єднувати дані з різних джерел, та надавати спільний SQL для об’єднаних даних. Також, це рішення має готові модулі для взаємодії з джерелами даних різних типів.

Отже, дане сховище є гарним варіантом для використання у якості data warehouse рішення в рамках магістерської дисертації;

- сховище Big Query від компанії Google [57]. Дане сховище також є гарним претендентом для використання у якості data warehouse у рамках розроблюваного програмного забезпечення. Це рішення здатне об’єднувати дані з різних джерел та об’єднувати їх у спільний інтерфейс. Позитивним фактором щодо цього рішення є наявність безкоштовної ліцензії на використання в рамках певного ліміту. Порівнюючи з іншими рішеннями, сервіс Big Query виділяється наявністю великої кількості навчальних матеріалів для користувача та загальною простотою користування.

Враховуючи описані переваги та недоліки, прийнято рішення використати сховище BigQuery від компанії Google для реалізації data warehouse для розробки програмного забезпечення. Дане рішення задовольняє поставленим вимогам, доступне до використання без оплати та має велику кількість навчальних матеріалів для полегшення розробки.

2.4 Вибір технологій для обробки команд природною мовою

Оскільки реалізоване програмне забезпечення включає взаємодію зі сховищем даних за допомогою команд природною мовою, для реалізації такого забезпечення потрібно реалізувати модуль для обробки команд природною мовою.

Враховуючи описані раніше вимоги та архітектурні рішення, можна сформувати наступні очікування від модуля обробки команд природною мовою:

- здатність працювати з англійською мовою у якості природної мови;
- здатність перетворювати голосові команди природною мовою у відповідні текстові команди;

- здатність перетворювати текстові команди природною мовою у відповідні команди формату SQL, що відповідають схемі даних з інтерфейсу data warehouse сховища;
- перевагою буде здатність проаналізувати схему даних відповідного data warehouse сховища з ціллю оптимізації процесу генерації запитів та зменшення кількості синтаксично некоректних запитів.

Під час дослідження існуючих рішень у сфері обробки команд природною мовою, з-поміж інших було розглянуто розумних асистентів, що працюють на базі LLM моделей – ChatGpt від OpenAi та Bard від Google. У рамках дослідження ці продукти показали гарні результати у обробці команд природною мовою. Враховуючи ці результати, доцільним є використання технології на базі LLM моделей для побудови модуля обробки природної мови.

Більш детальне дослідження наявних рішень на базі LLM моделей показало, що продукти компанії OpenAi наразі займають провідне місце у сфері обробки команд природною мовою. Тому доцільним було б використати продукти саме цієї компанії для побудови модуля обробки команд природної мови.

Для перетворення голосових команд у текстові можна використати модель Whisper від компанії OpenAi [58]. Дана модель здатна перетворювати аудіофайли у текст (робити транскрипцію) та перетворювати аудіозаписи з однієї мови в іншу (робити переклад). Таке перетворення можна використати для трансформації голосових команд англійською мовою (що будуть заздалегідь записані у аудіофайл за допомогою механізмів графічного інтерфейсу) у відповідну текстову репрезентацію даних команд (за допомогою механізму транскрипції). Поміж іншого, перевагою моделі Whisper є наявність програмного веб інтерфейсу. Цей інтерфейс може бути використаний backend частиною додатку для автоматизованого перетворення голосових команд у текстовий формат.

Для перетворення текстових команд у формальні SQL команди можна використати модель GPT4 від компанії OpenAi [59]. Ця модель на даний момент є однією з найпотужніших серед існуючих LLM моделей. Саме тому вона гарно підходить для перетворення команд природної мови у текстові. Завдяки своїй потужності система буде здатна покрити широкий спектр можливих запитів природною мовою. Також, система буде здатна генерувати SQL команди високої складності та працювати зі складними схемами даних. Модель GPT4 також має програмний веб інтерфейс, що відкриває можливість для виклику функцій цієї моделі зі backend частини програмного забезпечення. Окрім цього варто відзначити, що компанія OpenAi нещодавно запустила платформу для створення власних асистентів на базі моделі GPT4 [60]. Створення такого асистенту дає можливість надати додатковий контекст для роботи моделі, що буде використаний для обробки кожного запиту користувача. У рамках запропонованого програмного забезпечення доцільним було б створити такого асистента та додати у контекст інформацію про схему даних в data warehouse сховищі даних. Отже, розумний асистент буде здатний генерувати запити з урахуванням інформації про схему даних. Такий підхід допоможе зменшити кількість помилок у згенерованих запитах та створювати більш ефективні запити.

Підсумовуючи вищесказане, для реалізації модулю обробки команд природною мовою в рамках роботи запропоновано використати такі технології:

- модель Whisper для перетворення голосових команд у текстові;
- розумний асистент на базі моделі GPT4 для перетворення текстових команд у SQL команди для data warehouse сховища. Дані команди будуть генеруватись з урахуванням інформації про схему даних, що зберігається у сховищі.

Вибрані моделі машинного навчання достатньо потужні для виконання необхідних функцій перетворення команд природної мови. Також, вони мають

програмний інтерфейс, що відкриває можливість для програмного використання зі сторони backend частини програмного забезпечення. Саме тому вони є гарним вибором для використання у рамках запланованого програмного забезпечення.

2.5 Вибір технологій для реалізації frontend додатку

Відповідно до описаних вимог та архітектурних рішень можна виділити такі вимоги до frontend додатку:

- наявність веб-орієнтованого графічного інтерфейсу у вигляді SPA [61] додатку;
- можливість записувати аудіофайли з голосовими командами англійською мовою;
- можливість записувати текстові команди англійською мовою;
- після перетворення голосових команд у текстові відображати згенеровану команду та надавати можливість редагувати її за потреби;
- запуск перетворення природних команд у SQL команди. Після такого перетворення відображати згенеровану команду та надавати можливість редагувати її за потреби;
- запуск виконання згенерованої SQL команди на стороні data warehouse сховища. Можливість переглянути набори даних, що були сформовані внаслідок виконання команди;
- можливість повертатись до попередніх етапів виконання (до прикладу, після генерації SQL команди повертатись назад до задання команди природною мовою з ціллю виправлення помилок та формування більш точного запиту.

Враховуючи описані вимоги, для реалізації frontend запропоновано використати набір технологій на базі бібліотеки React [62]. Такий технологічний стек широко застосовується для побудови SPA додатків та довів свою ефективність

на практиці. З-поміж іншого, для побудови графічного інтерфейсу запропоновано використати такі технології:

- бібліотека React для побудови швидкого, зручного та адаптивного графічного веб інтерфейсу;
- типізація за допомогою мови TypeScript [63] для забезпечення надійності та підтримуваності системи;
- бібліотека компонентів Material UI [64] для пришвидшення розробки графічних базових компонентів, а також для полегшення роботи зі стилями;
- бібліотека recorder-js [65] для запису аудіофайлів з голосовими командами.

Підсумовуючи, для реалізації frontend частини додатку вибрано набір технологій на базі бібліотеки React. Дані технології дають змогу побудувати графічний інтефрейс, що задовольняє поставленим вимогам.

2.6 Вибір технологій для реалізації backend додатку

Backend частина додатку буде відповідати за логіку обробки та перетворення команд природною мовою, а також за взаємодію зі сторонніми системами.

Серед важливих функцій backend частини додатку варто виділити наступні:

- отримання команд природною мовою від графічного інтефрейсу користувача та перетворення їх у SQL команди за допомогою моделі Whisper та розумного асистенту на базі моделі GPT4;
- взаємодія з моделлю Whisper для перетворення аудіо файлів у текст за допомогою програмного веб інтерфейсу;
- взаємодії з розумним асистентом на базі моделі GPT4 за допомогою викликів програмного веб інтерфейсу;
- взаємодія з data warehouse сховищем даних на базі продукту BigQuery за допомогою використання програмного веб інтерфейсу;

- виконання згенерованого SQL запиту на стороні BigQuery, форматування отриманих наборів даних та передача результатів на графічний інтефрейс для відображення користувачу.

Для реалізації описаних функцій обрано набір технологій на базі платформи dotnet [66] та мови C# [67]. Можна виділити наступні переваги даної платформи:

- наявність віртуальної машини, завдяки чому сервер не буде залежати від архітектури операційної системи, що його запускає;
- наявність вбудованої стандартної бібліотеки, що дає можливість використовувати перевірені та оптимізовані рішення типових проблем;
- наявність вбудованих бібліотек для побудови веб-серверів;
- сильна статична типізація мови, що підвищує надійність написаного коду;
- підтримка об’єктно-орієнтованих конструкцій, завдяки чому можна писати системи, що легко розширюються та підтримуються;
- наявність готової бібліотеки [68] для взаємодії зі сховищами BigQuery, що значно полегшує реалізацію взаємодії з data warehouse сховищем у програмному забезпеченні.

Отже, обраний набір технологій на базі платформи dotnet здатний забезпечити необхідні вимоги для реалізації backend частини додатку. Саме тому він є гарним вибором для реалізації відповідного програмного забезпечення..

2.7 Опис запропонованої архітектури додатку

Підсумовуючи опис запропонованих архітектурних рішень, можна виділити наступні архітектурні компоненти запланованої інформаційної системи:

- графічний інтерфейс користувача на базі бібліотеки React, що відповідає за взаємодію з користувачем та надсилання запитів на backend частину додатку;

- backend частина додатку на базі платформи dotnet, що відповідає за реалізацію логіки обробки запитів користувача та взаємодію зі сторонніми системами;
- сховище даних у форматі data warehouse на базі продукту BigQuery, що відповідає за взаємодію з наявними джерелами даних та об’єднання усіх даних у спільний SQL інтерфейс;
- модель машинного навчання Whisper, що відповідає за перетворення голосових команд природною мовою у відповідні текстові команди;
- розумний асистент на базі моделі GPT4, що відповідає за перетворення текстових команд англійською мовою у SQL команди.

Схематичне зображення запропонованої архітектури зображено у Додатку А.

2.8 Висновки до розділу

У розділі сформовано набір функціональних та нефункціональних вимог до програмного забезпечення на базі задач, що були описані в першому розділі. Відштовхуючись від сформованих вимог, запропоновано набір архітектурних рішень для реалізації програмного забезпечення. Спираючись на відповідні вимоги та архітектурні рішення, проведено пошук та аналіз існуючих інструментів для реалізації відповідних програмних компонентів. Користуючись результатами проведеного аналізу, підібрано набір технологій, що можуть бути використаними для ефективної реалізації запропонованого програмного забезпечення. На базі підібраних технологій та архітектурних рішень сформовано та описано набір програмних модулів, з яких буде складатись програмне забезпечення. Окрім цього, описано аспекти взаємодії цих модулів між собою. Результатом проведеного дослідження стало формування архітектури програмного забезпечення, та опис цієї архітектури схематичному форматі у Додатку А.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Реалізація data warehouse сховища даних

Відповідно до прийнятих архітектурних рішень Google BigQuery буде використовуватись для побудови data warehouse сховища даних розроблюваного програмного забезпечення. Таке сховище буде об’єднувати різні джерела даних у спільний SQL інтефрейс.

У якості тестового джерела даних використано набори даних про продажі мережі супермаркетів Walmart [69] у США. Даний датасет включає інформацію про продані товари, чеки, магазини та інвентаризації. Датасет складається з 3-х таблиць у форматі CSV, що містять взаємопов’язані дані. У рамках даної роботи кожну з таблиць можна вважати окремим джерелом даних.

Для створення сховища у BigQuery необхідно пройти реєстрацію та створити необхідний проект у хмарній платформі Google Cloud Platform. Після цього достатньо створити базу даних, та імпортувати до неї дані з відповідних джерел. Для простоти розробки, наданий датасет було імпортовано за допомогою завантаження файлів з локальної файлової системи. Але, окрім простого завантаження файлів, продукт BigQuery також підтримує багато інших способів імпортувати дані. На рис. 3.1 зображено знімок екрану з наявними способами імпорту даних. З-поміж інших, ця система підтримує завантаження з файлових сховищ популярних хмарних платформ (Azure Blob Storage, Amazon S3 та інші), імпорт даних з популярних реляційних (PostgreSQL, MS SQL Server, та ін.) та нереляційних баз даних (MongoDb, Azure CosmosDb, та ін.), а також інтеграцію з різноманітними платформами для аналітики та статистики (YouTube Statistics, TikTok Ads, і так далі).

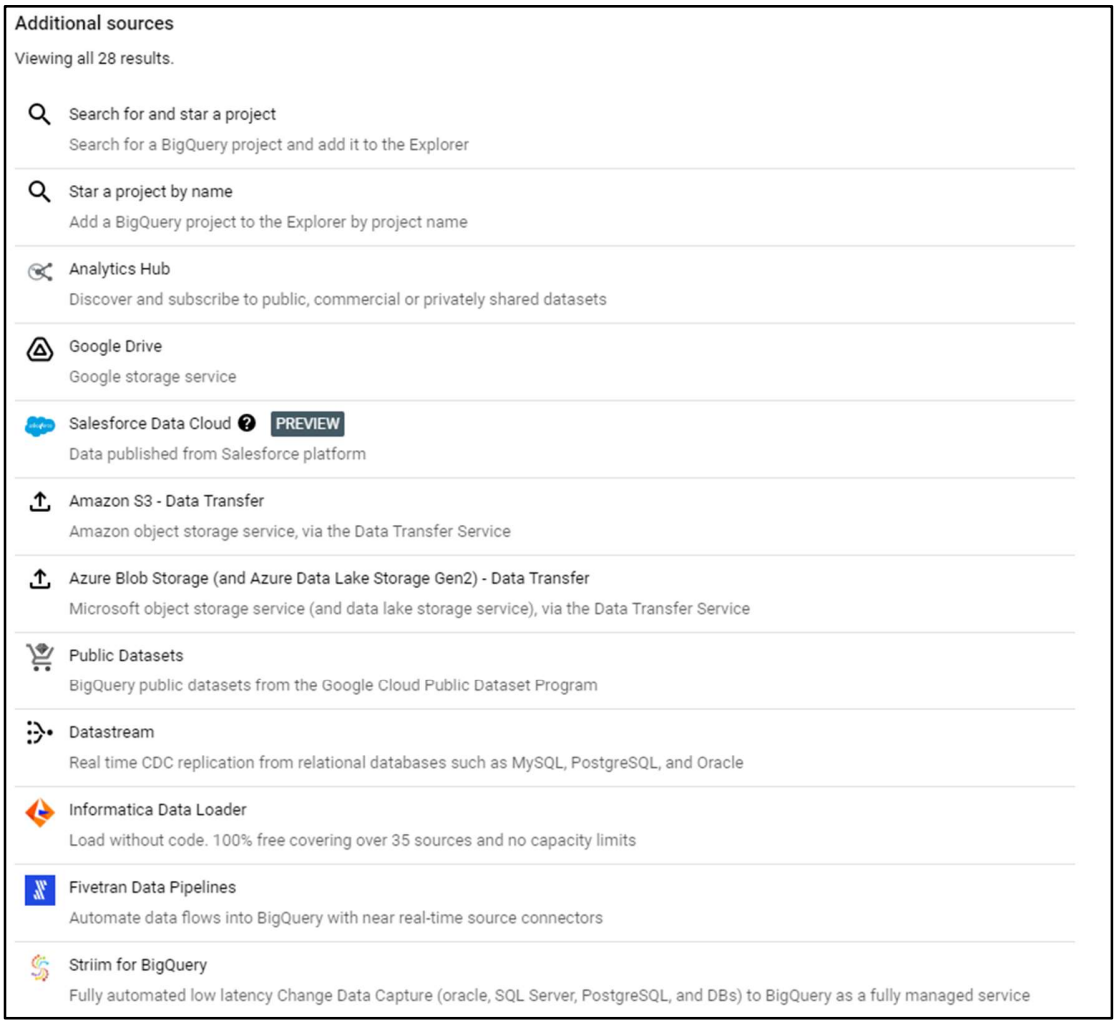


Рисунок 3.1 – Способи імпорту даних у сховище BigQuery

Після імпорту даних, система BigQuery дає можливість отримати схему сховища у форматі SQL команд. Ця схема буде використана під час обробки команд природною мовою для підвищення ефективності такої обробки. Для отримання відповідної схеми даних потрібно виконати запит до технічної таблиці сховища. Приклад запиту надано на рис. 3.2.

```
SELECT ddl FROM `naturaldb-research.test_walmart_sales.INFORMATION_SCHEMA.TABLES` LIMIT 1000
```

Рисунок 3.2 – Приклад запиту для отримання схеми даних в BigQuery

Діаграму сутностей та залежностей з отриманої схеми даних надано у Додатку Б.

Для перевірки коректності завантажених даних виконано декілька тестових запитів за допомогою графічного веб інтерфейсу продукту BigQuery. Окрім виконання тестових запитів, цей інтерфейс також може використовуватись і для моніторингу та відлагодження відповідних SQL команд, що були згенеровані за допомогою модуля обробки природної мови у розробленому програмному забезпеченні. Приклад використання графічного інтерфейсу BigQuery зображено на рис. 3.3.

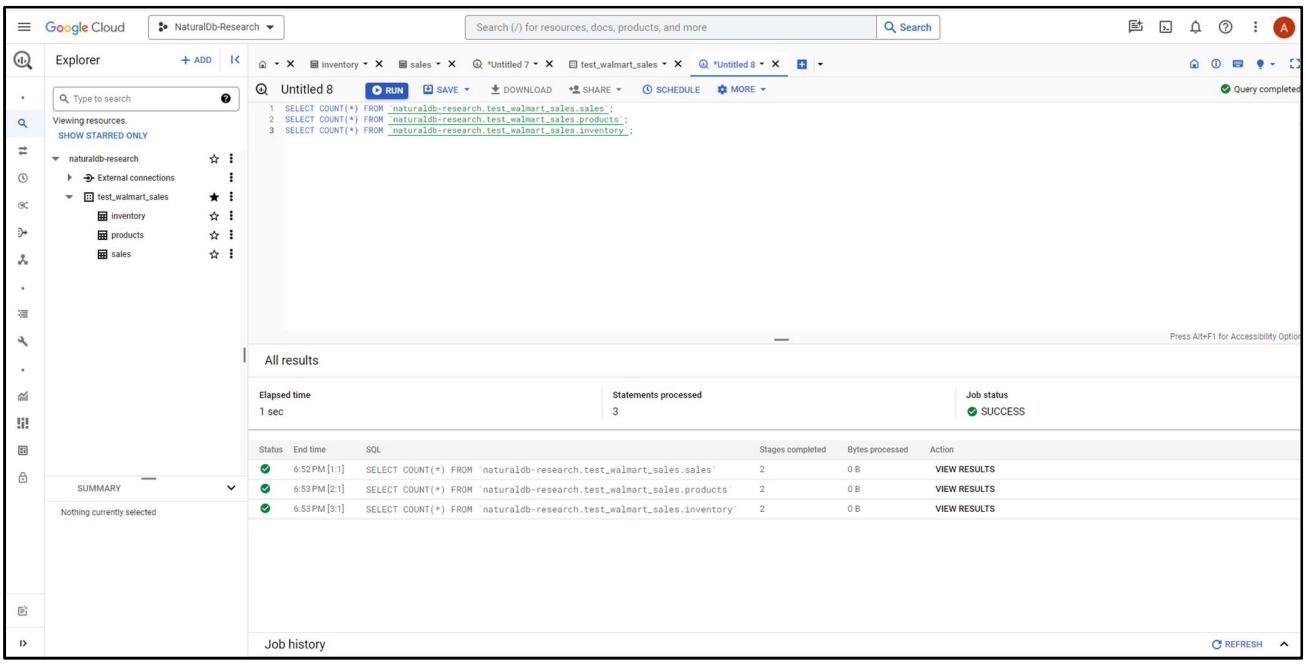


Рисунок 3.3 – Приклад використання графічного інтерфейсу BigQuery

У результаті отримано працююче data warehouse сховище на базі рішення BigQuery. До даного сховища додано тестові дані про продажі в магазинах Walmart. За потреби до сховища можна додати й інші джерела даних. Для цього можна використати раніше наведені модулі для імпорту даних.

3.2 Реалізація розумного асистенту для обробки команд природною мовою

Для обробки команд природною мовою будуть використовуватись модель Whisper та розумний асистент на базі моделі GPT4 від OpenAi.

Для користування даними моделями потрібно зареєструватись на платформі OpenAI [70]. Описані моделі не доступні у безкоштовній моделі користування від OpenAi. Тому для користування цими моделями потрібно внести певний грошовий депозит на платформі OpenAi. Більш детально ознайомитись з цінами на використання даними моделями можна за посиланням [71].

Після реєстрації на платформі OpenAi та внесення грошового депозиту, модель Whisper стає одразу доступною та готовою до використання в рамках розробленого програмного забезпечення. У той самий час модель GPT4 потребує додаткових дій перед початком використання. А саме, створення розумного асистенту на базі моделі GPT4, що буде мати інформацію про схему даних у BigQuery сховищі.

Для створення розумного асистенту потрібно перейти на відповідну вкладку на платформі OpenAi та заповнити необхідні поля з інформацією про асистента. Приклад створення розумного асистенту зображено на рис. 3.4.

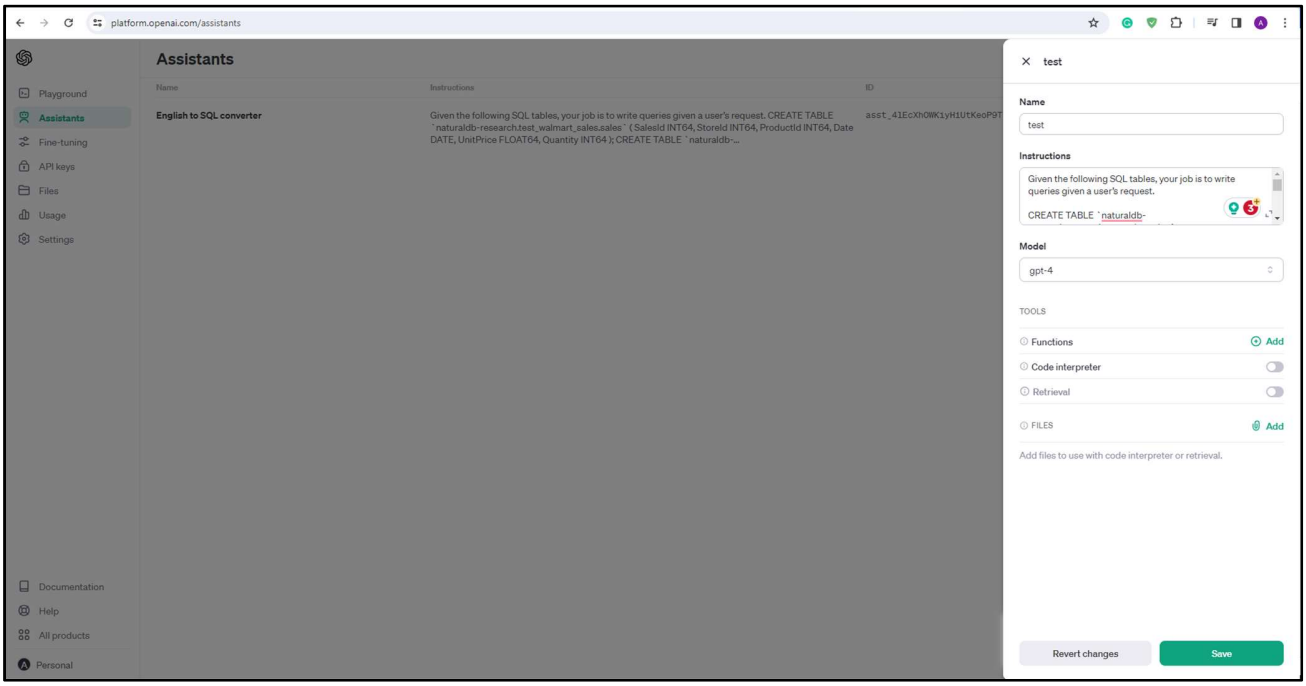


Рисунок 3.4 – Приклад створення розумного асистенту на базі GPT4

Важливим аспектом при створенні розумного асистенту є задання інструкцій. Ці інструкції будуть впливати на роботу асистента при обробці запитів користувача. Для підвищення ефективності роботи асистента, у ці інструкції варто додати інформацію про схему даних у сховищі BigQuery. Приклад інструкції, що була використана при створенні розумного асистенту, надано на рис 3.5.

```
01: Given the following SQL tables, your job is to write queries given a user’s request.
02:
03: CREATE TABLE `naturaldb-research.test_walmart_sales.sales`
04: (
05:   SalesId INT64,
06:   StoreId INT64,
07:   ProductId INT64,
08:   Date DATE,
09:   UnitPrice FLOAT64,
10:   Quantity INT64
11: );
12:
13: CREATE TABLE `naturaldb-research.test_walmart_sales.products`
14: (
15:   ProductId INT64,
16:   ProductName STRING,
17:   Supplier STRING,
18:   ProductCost FLOAT64
19: );
20:
21: CREATE TABLE `naturaldb-research.test_walmart_sales.inventory`
22: (
23:   ProductId INT64,
24:   StoreId INT64,
25:   StoreName STRING,
26:   Address STRING,
27:   neighborhood STRING,
28:   QuantityAvailable INT64
29: );
30:
31: You should return only the SQL, without any explanation
32:
```

Рисунок 3.5 – Інструкція для розумного асистента на базі GPT4

Варто також звернути увагу на останнє речення в інструкції – «You should return only the SQL, without any explanation». Це речення вказує на бажане форматування результатів, що генеруються за допомогою моделі GPT4. По

замовчуванню, модель намагається пояснити процес генерації та описати причини появи тих чи інших конструкції. Хоча це корисно при користуванні моделлю напряму, така поведінка є зайвою при взаємодії з моделлю за допомогою програмного інтерфейсу, адже для backend частини потрібно лише отримати згенерований SQL запит, а всі пояснення будуть зайві. Більше того, наявність додаткового тексту ускладнює процес пошуку саме SQL запиту у відповіді від розумного асистенту. Тому важливо отримувати відповіді від асистента саме у заданому форматі.

Для перевірки працездатності створеного асистенту виконано декілька тестових запитів. Ці запити виконані за допомогою графічного веб-інтефрейсу до платформи OpenAi. Приклад виконання тестового запиту зображено на рис. 3.6.

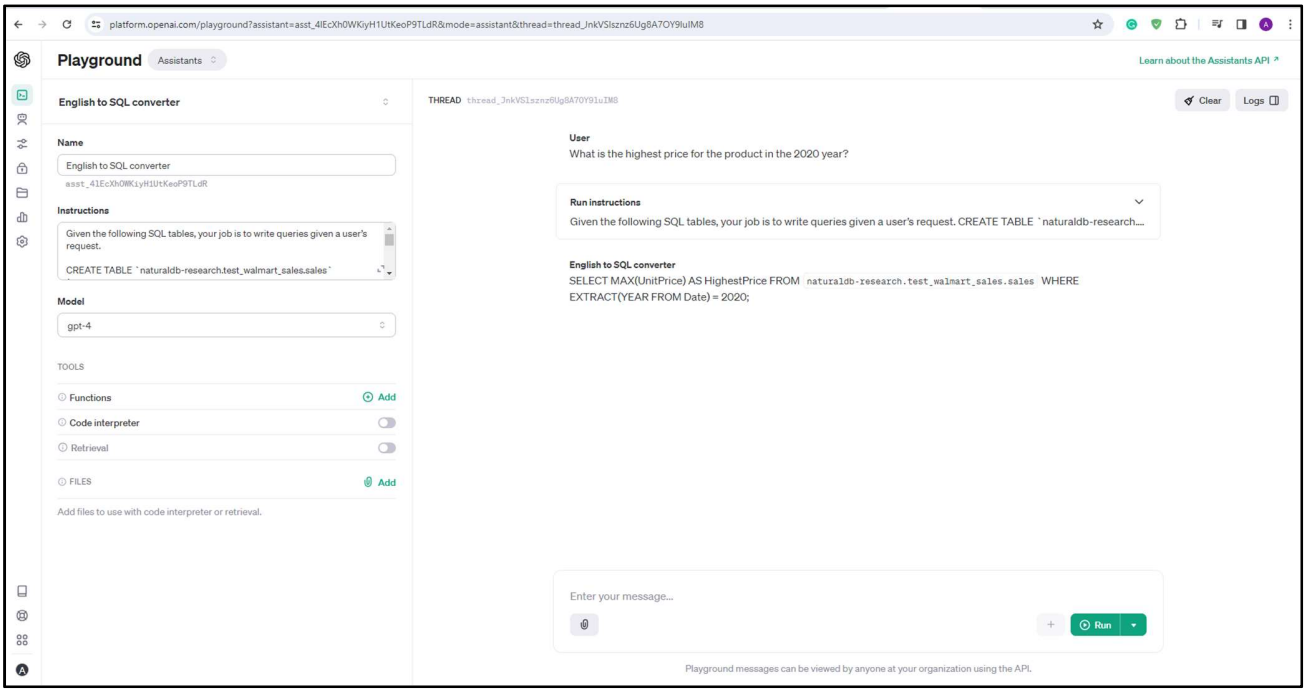


Рисунок 3.6 – Приклад виконання запиту до розумного асистенту на базі GPT4

Під час обробки тестових запитів асистент показав змогу успішно перетворювати запити англійською мовою у SQL запити до відповідного сховища. Тому можна вважати, що асистент готовий до використання у рамках розробленого програмного забезпечення.

3.3 Створення backend додатку

3.3.1 Загальний опис підходу до розробки backend додатку

Відповідно до сформованих архітектурних рішень програмне забезпечення потребує створення backend підсистеми. Цей додаток буде відповідати за логіку обробки команд природною мовою та взаємодіяти зі сторонніми системами. Згідно з описаним раніше вибором технологій розробки, backend додаток буде реалізовано з використанням платформи dotnet.

Враховуючи вибір платформи dotnet, найбільш доцільним варіантом реалізації даного модуля є створення WebApi серверу на базі фреймворку Asp.Net Core [72]. У такому разі backend додаток буде по суті являти собою веб-сервер з веб-орієнтованим API.

Для покращення підтримуваності системи використано патерн проектування Mediator та відповідну бібліотеку Mediatr [73]. Відповідно до цього підходу, кожен варіант використання (перетворення команд, взаємодія зі сторонньою системою тощо) реалізований як певна команда та відповідний обробник для команди. Таким чином, виконується принцип розподілення відповідальності між обробниками команд. Як наслідок, загальна підтримуваність системи покращується.

Вихідний код створеного backend додатку знаходиться у відкритому доступі, та доступний за посиланням [74].

3.3.2 Перетворення голосових команд у текстові за допомогою моделі Whisper

Відповідно до прийнятих архітектурних рішень backend модуль має використовувати модель Whisper для перетворення голосових команд у текстові. На вхід модуль буде отримувати аудіофайл з голосовими командами, а на вихід має надіслати ті ж команди у текстовому форматі.

Реалізація комунікації з моделлю Whisper відбувалась згідно з інструкцією від OpenAi [75]. Для взаємодії з моделлю використовується вбудований у dotnet HTTP клієнт. Програмна реалізація логіки взаємодії із моделлю Whisper відображена на рис. 3.7.

```
01:    public async Task<TranscriptAudioFileResponse> Handle(TranscriptAudioFileRequest request,
CancellationToken cancellationToken)
02:    {
03:        logger.LogInformation("Transcripting audio file. FileName - {fileName}", request.FileName);
04:
05:        using var whisperRequest = new HttpRequestMessage(HttpMethod.Post,
openAiOptions.WhisperEndpoint);
06:        whisperRequest.Content = new MultipartFormDataContent
07:        {
08:            { new StreamContent(request.FileStream), "file", request.FileName },
09:            { new StringContent(openAiOptions.WhisperModel), "model" },
10:            { new StringContent("en"), "language" }
11:        };
12:
13:        var recognition = await openAiMessageSender.Send<WhisperRecognition>(whisperRequest,
cancellationToken);
14:
15:        var result = new TranscriptAudioFileResponse(recognition.Text);
16:
17:        logger.LogInformation("Transripted audio file. FileName - {fileName}, ConvertionResult - {result}",
request.FileName, result);
18:        return result;
19:    }
```

Рисунок 3.7 – Логіка взаємодії з моделлю Whisper

3.3.3 Перетворення текстових команд у SQL команди за допомогою розумного асистенту на базі GPT4

Для перетворення текстових команд природною мовою у відповідні SQL команди використано розумного асистента на базі GPT4, що був створений в рамках даної роботи раніше. Взаємодія з асистентом за допомогою програмного інтерфейсу реалізована відповідно до інструкцій від OpenAi [76].

Відповідно до інструкції, виклик функцій розумного асистента через програмний інтерфейс включає наступні етапи:

- створити новий потік розмови;
- додати повідомлення до потоку;

- запустити асистента на базі цього потоку;
- періодично перевіряти статус запуску, допоки запуск не стане закінченим;
- отримати результат запуску у відповідному потоці.

Блок схему алгоритму перетворення команди природної мови у SQL надано у Додатку В. Далі описано програмну реалізацію для кожного з етапів цього алгоритму.

У рис. 3.8 надано реалізацію високорівневого алгоритму взаємодії.

```
01:    public async Task<ConvertTextToSqlResponse> Handle(ConvertTextToSqlRequest request,
CancellationToken cancellationToken)
02:    {
03:        logger.LogInformation(
04:            "Converting text to sql with OpenAi assistant. Text - {text}, Timeout - {timeout}ms",
05:            request.Text,
06:            openAiOptions.TextToSqlTimeoutInSeconds);
07:
08:        var response = await ConvertTextToSql(request, cancellationToken)
09:            .WaitAsync(TimeSpan.FromSeconds(openAiOptions.TextToSqlTimeoutInSeconds),
cancellationToken);
10:
11:        logger.LogInformation("Converted text to sql with OpenAi assistant. Text - {text}, Sql - {sql}",
request.Text, response.Sql);
12:        return response;
13:    }
14:
15:    private async Task<ConvertTextToSqlResponse> ConvertTextToSql(ConvertTextToSqlRequest request,
CancellationToken cancellationToken)
16:    {
17:        var createThreadResponse = await mediator.Send(new CreateThreadRequest(), cancellationToken);
18:        var threadId = createThreadResponse.ThreadId;
19:
20:        await mediator.Send(new AddMessageToThreadRequest(threadId, Message: request.Text),
cancellationToken);
21:
22:        var runAssistantResponse = await mediator.Send(
23:            new RunAssistantRequest(threadId, AssistantId: openAiOptions.TextToSqlAssistantId),
24:            cancellationToken);
25:        var runId = runAssistantResponse.StartedRun.Id;
26:
27:        await WaitUntilRunCompletes(threadId: threadId, runId: runId, cancellationToken);
28:
29:        var messageList = await mediator.Send(new RetrieveMessageListRequest(threadId),
cancellationToken);
30:
31:        var convertedSql = GetContentOfLastMessageFromAssistant(messageList);
32:        var formattedSql = FormatSql(convertedSql);
33:
34:        var response = new ConvertTextToSqlResponse(Sql: formattedSql);
35:        return response;
36:    }
```

Рисунок 3.8 – Реалізація запиту розумного асистенту на базі GPT4 (початок)

```
38:     private async Task WaitUntilRunCompletes(string threadId, string runId, CancellationToken
cancellationToken)
39:     {
40:         while (true)
41:         {
42:             var currentRunState = await mediator.Send(new RetrieveRunRequest(ThreadId: threadId, RunId:
runId), cancellationToken);
43:
44:             if (currentRunState.Status == RunStatuses.Completed)
45:             {
46:                 return;
47:             }
48:
49:             logger.LogInformation(
50:                 "Run is not completed yet. Waiting {timeout}ms before next check. Current run status -
{status}, ThreadId - {threadId} RunId - {runId}",
51:                 openAiOptions.TextToSqlStatusCheckIntervalInMs,
52:                 currentRunState.Status,
53:                 threadId,
54:                 runId);
55:
56:             await
Task.Delay(TimeSpan.FromMilliseconds(openAiOptions.TextToSqlStatusCheckIntervalInMs),
cancellationToken);
57:         }
58:     }
59:
60:     private string GetContentOfLastMessageFromAssistant(MessageListModel messageList)
61:     {
62:         var lastAssistantMessage = messageList.Data.Last(message => message.Role ==
MessageRoles.Assistant);
63:         var lastMessageContent = lastAssistantMessage.Content.Last();
64:         var result = lastMessageContent.Text.Value;
65:
66:         return result;
67:     }
68:
69:     private string FormatSql(string rawSql)
70:     {
71:         var lineBreakSymbols = new [] { '\n', '\r' };
72:         var formatted = lineBreakSymbols.Aggregate(rawSql, (sql, symbol) => sql.Replace(symbol, ' '));
73:         return formatted;
74:     }
```

Рисунок 3.8 - Реалізація запиту розумного асистенту на базі GPT4 (кінець)

З важливих моментів реалізації даного запиту варто виділити наступне:

- наявність захисного таймауту для запиту у рядку 09. Цей таймаут забезпечує переривання запиту у випадку, якщо він займає занадто багато часу. Таким чином, некоректний запит не буде використовувати зайві ресурси;

- очікування закінчення виконання запиту до асистенту за рахунок циклу (рядок 40), що перевіряє статус запиту (рядок 44) з певною періодичністю (рядок 56), допоки запит не буде виконаний успішно (рядок 45);
- форматування результуючого запиту (рядки 69-74), приведення його до запиту в один рядок. Потреба у форматуванні виникає від особливості роботи GPT4, що по замовчуванню генерує запити у форматі, що зручний для читання (з переносами рядків). Але, такий формат погано підходить для подальшого виконання на стороні data warehouse сховища.

Для створення нового потоку розмови з розумним асистентом використовується код, що надано у рис. 3.9.

```
01:    public async Task<CreateThreadResponse> Handle(CreateThreadRequest request, CancellationToken
cancellationToken)
02:    {
03:        logger.LogInformation("Creating OpenAi thread");
04:
05:        using var httpRequest = new HttpRequestMessage(HttpMethod.Post,
openAiOptions.ThreadsEndpoint);
06:        httpRequest.Content = JsonContent.Create(new object());
07:
08:        var creationModel = await openAiMessageSender.Send<ThreadCreationModel>(httpRequest,
cancellationToken);
09:
10:        var response = new CreateThreadResponse(ThreadId: creationModel.Id);
11:        logger.LogInformation("Created OpenAi thread. ThreadId - {threadId}", creationModel.Id);
12:
13:        return response;
14:    }
```

Рисунок 3.9 – Створення нового потоку розмови з розумним асистентом

У результаті успішного створення нового потоку платформа надсилає ідентифікатор нового потоку. Цей ідентифікатор буде використовуватись для взаємодії з створеним потоком у наступних етапах.

На рис. 3.10 надано програмну реалізацію логіки для додавання повідомлень до створеного потоку.

```
01:    public async Task Handle(AddMessageToThreadRequest request, CancellationToken
cancellationToken)
02:    {
03:        logger.LogInformation("Adding a message to the OpenAi thread. ThreadId - {threadId}, Message -
{message}", request.ThreadId, request.Message);
04:
05:        var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/messages";
06:        using var httpRequest = new HttpRequestMessage(HttpMethod.Post, url)
07:        {
08:            Content = JsonContent.Create(new
09:            {
10:                role = "user",
11:                content = request.Message
12:            })
13:        };
14:
15:        await openAiMessageSender.Send(httpRequest, cancellationToken);
16:        logger.LogInformation("Added a message to the OpenAi thread. ThreadId - {threadId}, Message -
{message}", request.ThreadId, request.Message);
17:    }
```

Рисунок 3.10 – Додавання повідомлень до потоку розмови з асистентом

Після успішного виконання описаного етапу, в потоці розмови з’явиться нове повідомлення від користувача. Асистент буде використовувати це повідомлення для генерації відповіді.

Програмна реалізація запуску роботи асистента виведено на рис. 3.11.

```
01:    public async Task<RunAssistantResponse> Handle(RunAssistantRequest request, CancellationToken
cancellationToken)
02:    {
03:        logger.LogInformation("Running OpenAi assistant. ThreadId - {threadId}, AssistantId - {assistantId}",
request.ThreadId, request.AssistantId);
04:
05:        var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/runs";
06:        using var httpRequest = new HttpRequestMessage(HttpMethod.Post, url)
07:        {
08:            Content = JsonContent.Create(new
09:            {
10:                assistant_id = request.AssistantId,
11:            })
12:        };
13:        var threadRun = await openAiMessageSender.Send<RunModel>(httpRequest, cancellationToken);
14:        logger.LogInformation(
15:            "Started the run of OpenAi assistant. ThreadId - {threadId}, AssistantId - {assistantId}, RunId -
{runId}",
16:            request.ThreadId,
17:            request.AssistantId,
18:            threadRun.Id);
19:        var response = new RunAssistantResponse(threadRun);
20:        return response;
21:    }
```

Рисунок 3.11 – Реалізація етапу запуску асистента

Процес роботи асистента на платформі OpenAi відбувається по асинхронній моделі. Тобто, у результаті запуску обробки програмний клієнт отримує лише ідентифікатор відповідного запуску. Далі, за допомогою окремих викликів програмного інтерфейсу, клієнт може перевірити статус обробки на даний момент. Коли запуск дійшов до закінченого стану, лише тоді клієнт може отримати результати виконання асистенту. Варто також підмітити, що асистент надає свій результат у форматі нового повідомлення у потоці розмови. Тобто, і запит користувача, і відповідь асистента, по суті являються повідомленнями в потоці розмови.

Реалізацію отримання поточного статусу запуску надано на рис. 3.12.

```
1:    public async Task<RunModel> Handle(RetrieveRunRequest request, CancellationToken
cancellationToken)
2:    {
3:        logger.LogInformation("Retrieving a run. ThreadId - {threadId}, RunId - {runId}", request.ThreadId,
request.RunId);
4:        var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/runs/{request.RunId}";
5:        using var httpRequest = new HttpRequestMessage(HttpMethod.Get, url);
6:        var run = await openAiMessageSender.Send<RunModel>(httpRequest, cancellationToken);
7:        logger.LogInformation("Retrieved the run. ThreadId - {threadId}, Run - {run}", request.ThreadId,
run);
8:        return run;
9:    }
```

Рисунок 3.12 – Отримання поточного статусу запуску асистента

Після того, як асистент закінчив обробку запиту, він додає нове повідомлення до потоку розмови. Для отримання результату потрібно завантажити додане повідомлення з потоку. Програмна реалізація отримання всіх повідомлень з потоку надано на рис. 3.13.

```
1:    public async Task<MessageListModel> Handle(RetrieveMessageListRequest request,
CancellationToken cancellationToken)
2:    {
3:        logger.LogInformation("Retrieving a message list. ThreadId - {threadId}", request.ThreadId);
4:        var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/messages";
5:        using var httpRequest = new HttpRequestMessage(HttpMethod.Get, url);
6:        var messageList = await openAiMessageSender.Send<MessageListModel>(httpRequest,
cancellationToken);
7:        logger.LogInformation("Retrieved the message list. ThreadId - {threadId}, MessageList - {list}",
request.ThreadId, messageList);
8:        return messageList;
9:    }
```

Рисунок 3.13 – Отримання повідомлень з потоку розмови

Після реалізації описаних етапів для взаємодії з розумним асистентом, backend додаток готовий до перетворення текстових команд англійською мовою у відповідні SQL команди. Для перевірки коректності такого перетворення виконано декілька запитів через веб інтерфейс backend додатку. Для надсилання запиту використано технологію Swagger [77]. Приклад виконання такого запиту зображено на рис. 3.14.

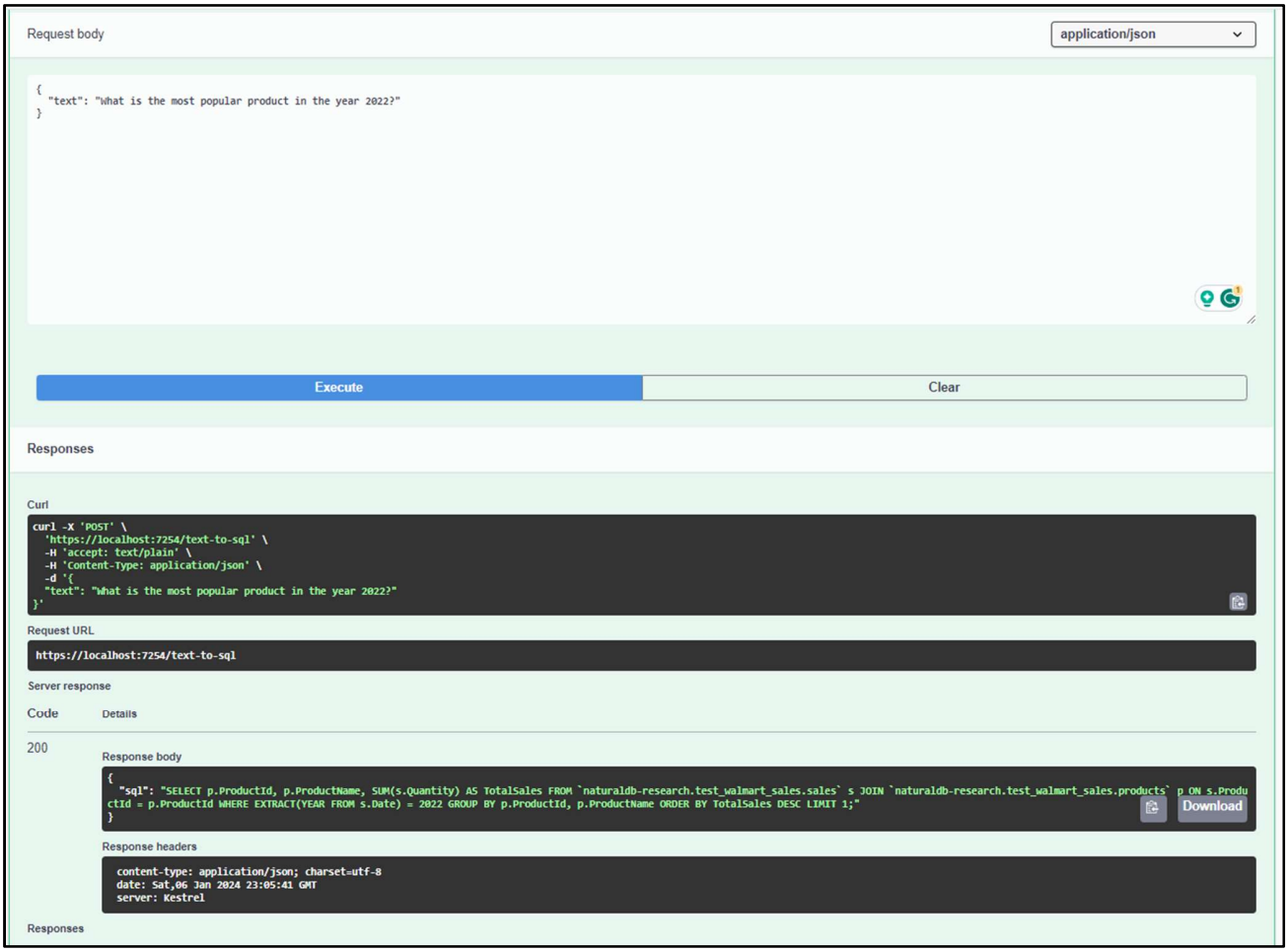


Рисунок 3.14 – Використання додатку для перетворення тестових команд в SQL

3.3.4 Виконання згенерованих SQL команд на стороні BigQuery сховища

Для виконання команд на стороні BigQuery використано відповідну бібліотеку для платформи dotnet [68]. Потрібно зазначити, що бібліотека повертає результати у вигляді набору відповідних класів, що реалізують табличну структуру.

Такий формат погано підходить для передачі на frontend додаток. Тому backend модуль перетворює отримані результати у CSV формат. Цей формат гарно підходить для передачі на frontend додаток та може бути відображеним на графічному інтерфейсі користувача.

Програмна реалізація взаємодії зі платформою BigQuery надана на рис. 3.15.

```
01:    public async Task<ExecuteSqlInBigQueryResponse> Handle(ExecuteSqlInBigQueryRequest request,
CancellationToken cancellationToken)
02:    {
03:        BigQueryClient client = await BigQueryClient.CreateAsync(bigQueryOptions.ProjectId);
04:        BigQueryResults resultsEnumerable = await client.ExecuteQueryAsync(request.Sql,
Enumerable.Empty<BigQueryParameter>());
05:        IReadOnlyCollection<BigQueryRow> resultsArray = resultsEnumerable.ToArray();
06:        var resultsInCsv = ConvertResultsToCsv(resultsArray);
07:        return new ExecuteSqlInBigQueryResponse(resultsInCsv);
08:    }
09:
10:    private string ConvertResultsToCsv(IReadOnlyCollection<BigQueryRow> results)
11:    {
12:        StringBuilder csvResult = new();
13:        var separator = ',';
14:        if (results.Count != 0)
15:        {
16:            var firstResult = results.First();
17:            var fieldNames = firstResult.Schema.Fields.Select(f => f.Name);
18:            var header = string.Join(separator, fieldNames);
19:            csvResult.AppendLine(header);
20:        }
21:        foreach (BigQueryRow row in results)
22:        {
23:            var line = string.Join(separator, row.RawRow.F.Select(field => field.V));
24:            csvResult.AppendLine(line);
25:        }
26:        return csvResult.ToString();
27:    }
```

Рисунок 3.15 – Реалізація взаємодії зі сховищем BigQuery

Для перевірки працездатності реалізованого модуля взаємодії зі сховищем, було виконано декілька тестових запитів за допомогою технології Swagger. Приклад виконання такого тестового запиту зображено на рис. 3.16.

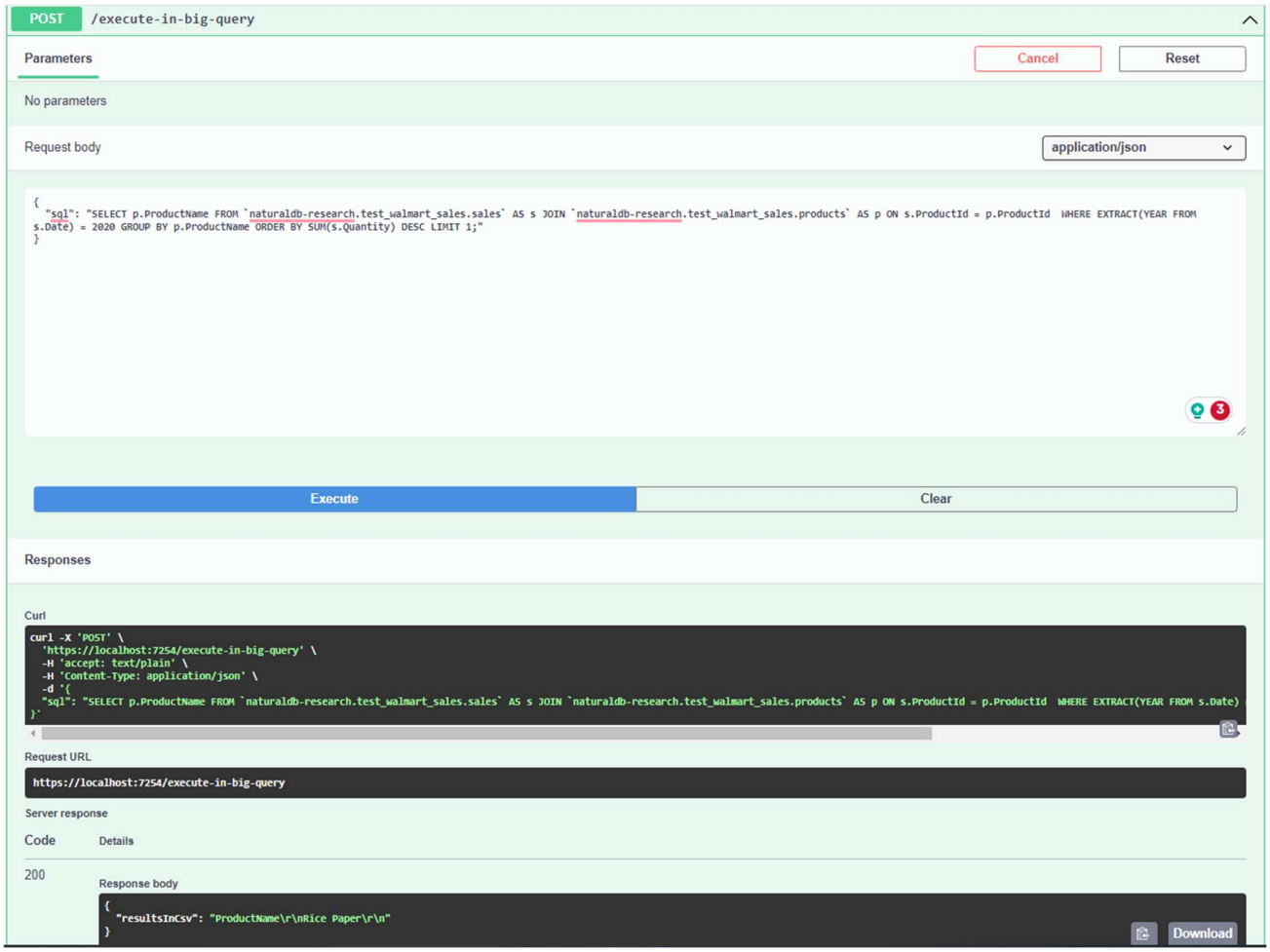


Рисунок 3.16 – Приклад виконання запиту для сховища BigQuery

Виконання тестових запитів показало, що реалізований модуль для взаємодії зі сховищем успішно виконує поставлені задачі. Надані команди коректно виконуються на стороні сховища. Результати виконання надаються у відповідь на HTTP запит у CSV форматі.

3.4 Створення графічного інтерфейсу користувача

Відповідно до описаного раніше вибору технологій, графічний інтерфейс користувача реалізовано на базі бібліотеки React. Цей інтефрейс відповідає за отримання команд від користувача, виклик модуля обробки команд природної мови для генерації SQL команд, та виконання цих команд на стороні сховища BigQuery.

Вихідний код створеного frontend додатку знаходиться у відритому доступі, та доступний за посиланням [78].

Графічний інтерфейс користувача реалізовано у вигляді однієї веб сторінки з декількома вкладками, що відповідають етапам взаємодії з додатком. На кожному з етапів користувач може вводити необхідні дані та перейти до наступного етапу, або повернутись до пройдених етапів.

Перша вкладка графічного інтерфейсу відповідає за отримання команди англійською мовою від користувача. Команда може бути задана текстом або записана у вигляді голосового файлу прямо в браузері. Приклад задання команди текстом показано на рис. 3.17, а приклад задання команди голосом на рис. 3.18.

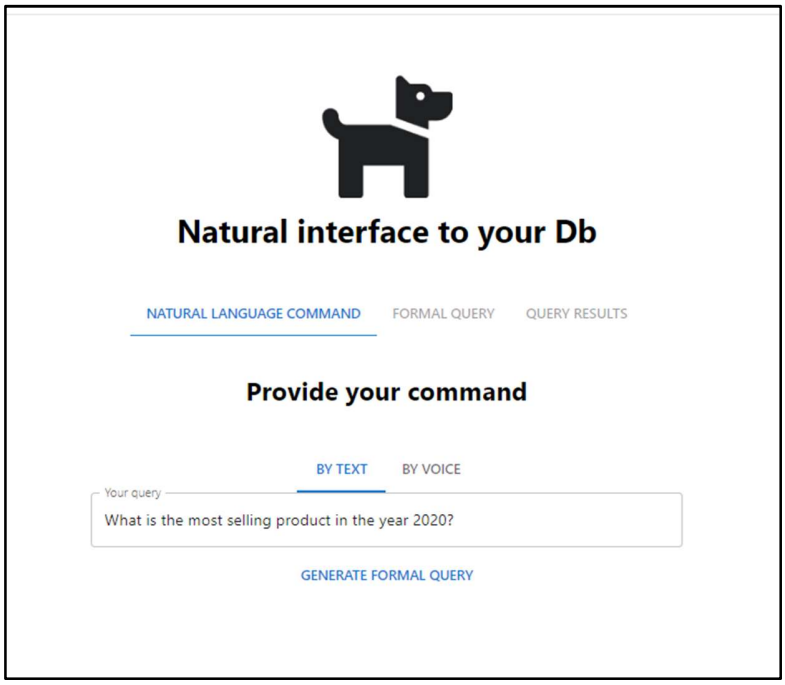


Рисунок 3.17 – Приклад задання текстової команди

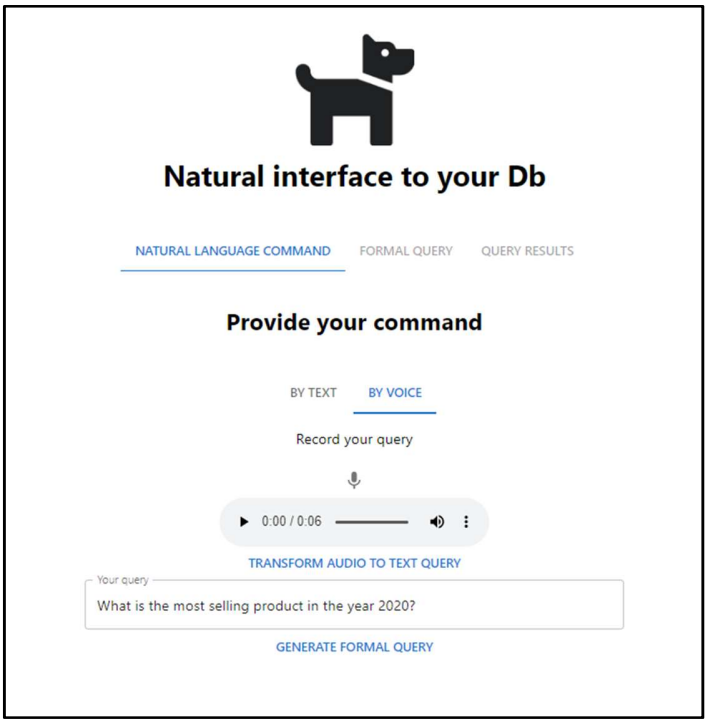


Рисунок 3.18 – Приклад задання голосової команди

Наступна вкладка на інтерфейсі – перегляд згенерованої SQL команди. За потреби, згенеровану команду можна редагувати. Приклад цієї вкладки зображено на рис. 3.19.

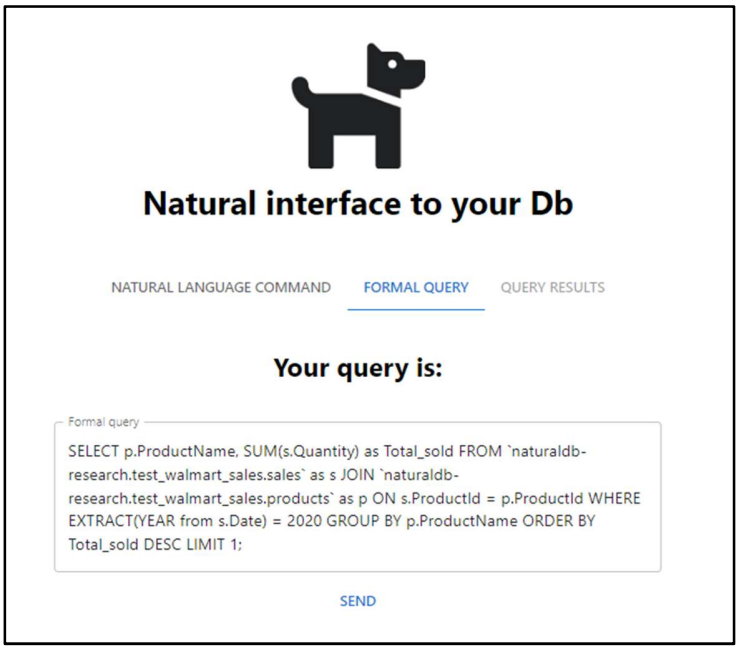


Рисунок 3.19 – Приклад перегляду згенерованої SQL команди

Остання вкладка призначена для перегляду результатів. Результати відображені у форматі CSV. За потреби, користувач може повернутись на попередні вкладки. Таким чином, користувач може перезапустити етапи обробки команди. Приклад вкладки перегляду результатів зображено на рис. 3.20.

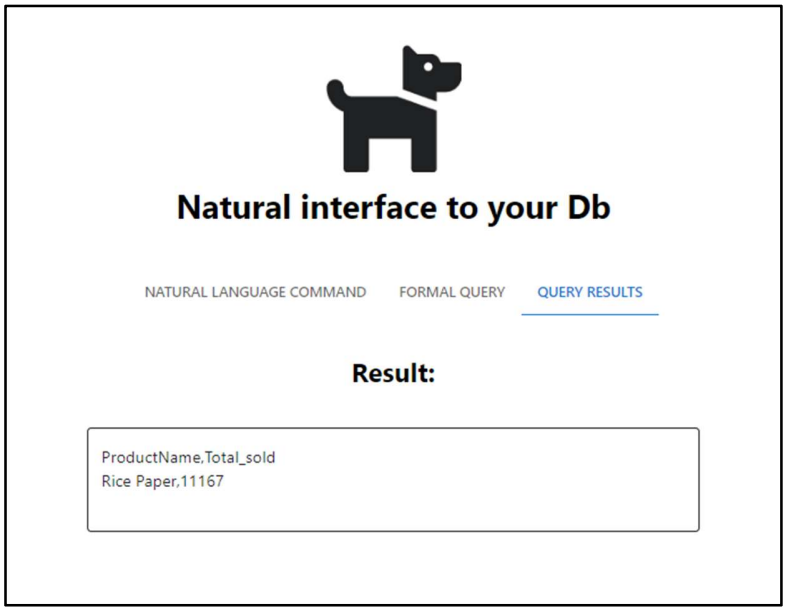


Рисунок 3.20 – Приклад перегляду результатів обробки команди

3.5 Експериментальні дослідження

Для оцінки працездатності створеної системи проведено ряд тестових запусків із запитами різної складності. Під час тестування перевірялась коректність отриманих результатів, а також швидкодія роботи системи. З точки зору швидкодії, окремо вимірювалась швидкодія генерації SQL команди та швидкодія виконання цієї команди на стороні BigQuery сховища.

Тестові запити готувались за допомогою асистенту ChatGpt. Запит до асистенту був наступний: «I want to test a system, which translates natural language commands into SQL queries and executes those queries against my storage. The storage contains information about Walmart sales between 2017 and 2020 years. Could you give me 20 commands in English, which may be used to test my system?»

Згенеровані команди та результати виконання цих команд надані у Табл. 3.1

Таблиця 3.1 – Результат тестування

№	Команда	Виконана успішно?	Час генерації	Час виконання
1	Retrieve the total sales for the year 2018.	Так	4,7с	1,01с
2	Show me the top 10 products sold in 2019.	Так	4,78с	1,2с
3	Find the average sales amount per month in 2020.	Так	4,82с	1,22с
4	List all the transactions on Black Friday in 2017.	Так	4,72с	0,69с
5	Show the total sales for the electronics category in 2018.	Ні	7,96с	Не виконана
6	Retrieve the top 5 highest-grossing departments in 2019.	Частково	11,45с	0.99с
7	Find the total sales for Walmart stores located in California in 2020.	Частково	7,95с	0.86с
8	Show the monthly trend of sales for the 'Clothing' category in 2017.	Частково	7,95с	0.89с
9	List all the products with zero sales in 2018.	Так	5.08с	0.83с
10	Find the store with the highest sales in Q4 of 2019.	Так	4.71с	1.12с
11	Retrieve the total sales for each year from 2017 to 2020.	Так	4.72с	0.83с
12	Show me the transactions with discounts greater than 20% in 2018.	Так	4.73с	0.91с
13	Find the average sales per day for the 'Home and Garden' category in 2019.	Ні	4.71с	Не виконана

Кінець таблиці 3.1

№	Команда	Виконана успішно?	Час генерації	Час виконання
14	List the products with sales above $10,000 in any month of 2020.	Так	7.94с	1.14с
15	Retrieve the top 3 stores with the highest sales in 2017.	Так	4.73с	0.78с
16	Show the sales distribution by state in Q2 of 2018.	Частково	11.2с	0.82с
17	Find the products with the highest and lowest sales in 2020.	Частково	8.01с	1.13с
18	List all the transactions where the payment method was credit card in 2019.	Ні	8.01с	Не виконана
19	Retrieve the total sales for each quarter in 2018.	Так	8.24с	1.11с
20	Show me the top 5 products with the highest sales growth from 2017 to 2020.	Так	7.94с	0.87с

У результаті тестових досліджень система автоматично згенерувала коректні SQL команди у 12 випадках з 20. У інших випадках система не змогла знайти необхідну інформацію у запропонованій схемі даних. Тому інтефрейс вимагав втручання користувача та редагування згенерованих запитів. У 5 випадках з 20 система згенерувала запит, але все одно вимагала втручання користувача. У таких випадках система описувала проблему та пропонувала наближений запит. Після виконання такого запиту було отримано бажаний результат.

На рис. 3.21 зображено приклад повідомлення про помилку при генерації. У цій помилці можна побачити опис проблеми та запропонований наближений запит.

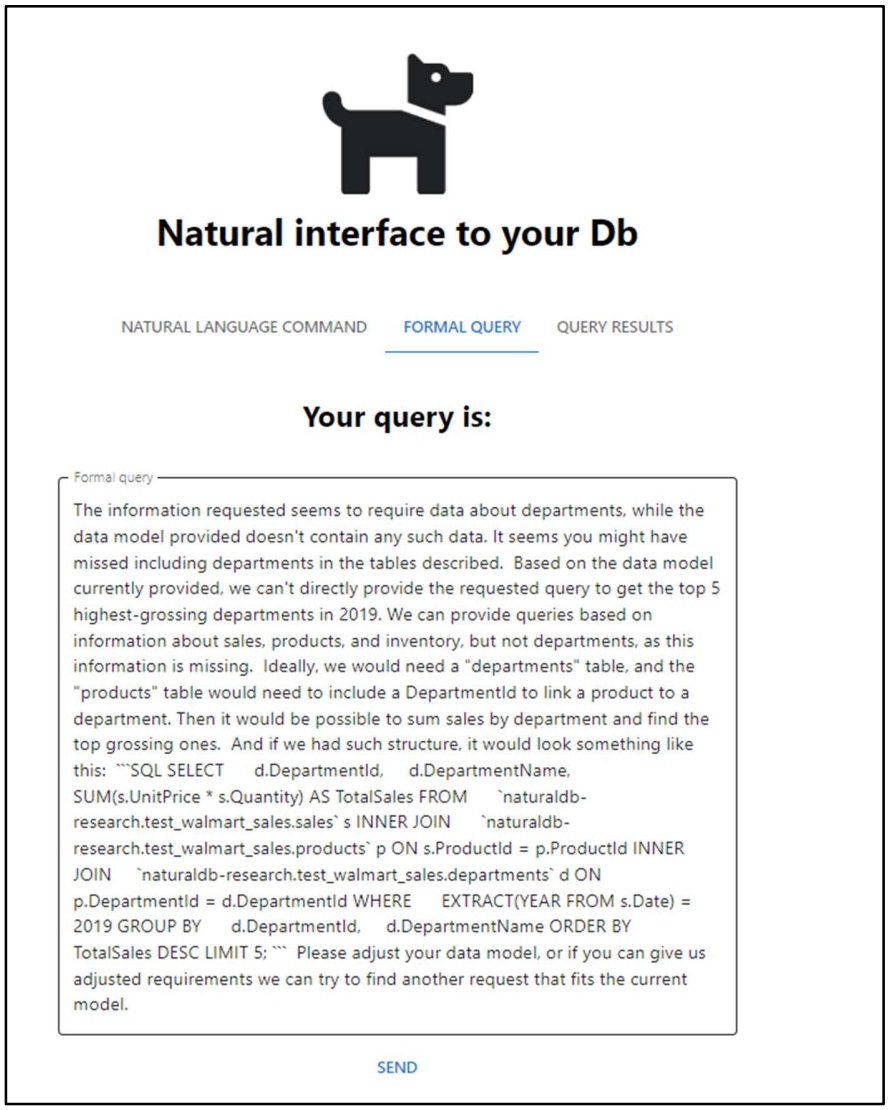


Рисунок 3.21 – Приклад генерації з помилкою та наближеним запитом

У 3-х випадках з 20-и система взагалі не змогла згенерувати запит. Асистент описав проблему та можливі шляхи вирішення. У всіх 3-х випадках проблемою було відсутність відповідних даних в датасеті.

3.6 Оцінка ефективності запропонованого рішення

Результати тестування показали, що реалізована система здатна перетворювати команди природної мови у SQL команди та виконувати їх на стороні сховища даних. Отже, створене програмне забезпечення може успішно виконувати поставлені у роботі задачі – виступати інтерфейсом для підготовки наборів даних з

різних джерел за допомогою інструкцій природною мовою. Під час тестування розроблена система успішно підготувала набори даних у 85% проведених тестів. Решта 15% тестів завершились помилкою. У помилкових тестах система змогла детально описати проблему та можливі шляхи вирішення. Також варто зазначити, що помилкові тести були помилковими через відсутність даних у вибраному датасеті, а не через некоректну роботу програмного забезпечення.

Щодо швидкодії запропонованого рішення. Під час тестування система показала наступні показники швидкодії:

- середня тривалість перетворення текстової команди у SQL – 6.67с;
- середня тривалість виконання згенерованої SQL команди – 0.94с.

У контексті обробки команд природної мови такі показники швидкодії є прийнятними. Якщо взяти до уваги складність згенерованих SQL команд, написання відповідних команд вручну займає у середньому на порядок більше часу.

Базуючись на описаних результатах тестування можна стверджувати, що створене програмне забезпечення здатне ефективно вирішувати поставлені задачі.

3.7 Інструкція користувача

Для користування програмним забезпеченням потрібно виконати наступні дії:

a) перейти на вкладку «NATURAL LANGUAGE COMMAND»;
b) задати команду природною мовою:
    1) у випадку текстової команди – вибрати режим «BY TEXT» та записати відповідну команду;
    2) у випадку голосової команди:
        - вибрати режим «BY VOICE»;
        - натиснути на іконку мікрофона та записати відповідну команду;

- за необхідності – прослухати записаний аудіофайл;
- натиснути кнопку «TRANSFORM AUDIO TO TEXT QUERY»;
- за необхідності – редагувати згенеровану текстову команду;

c) натиснути кнопку «GENERATE FORMAL QUERY»;
d) зачекати поки система згенерує SQL запит. Коли запит згенеровано, система автоматично відкриє вкладку «FORMAL QUERY» та відобразить запит;
e) на вкладці «FORMAL QUERY» переглянути згенерований запит, редагувати його за потреби;
f) натиснути кнопку «SEND»;
g) зачекати поки система виконує запит. Коли запит виконано, система автоматично відкриє вкладку «QUERY RESULTS»;
h) на вкладці «QUERY RESULTS» – переглянути результат виконання;
i) за потреби повернутись на попередні вкладки для перезапуску процесу виконання.

3.8 Висновки по розділу

У розділі описано етап розробки програмного забезпечення системи. Продемонстровано налаштування допоміжних систем, що необхідні для коректної роботи програмного забезпечення. Описано програмні модулі для взаємодії з цими системами. Наведено приклади графічного інтерфейсу користувача та сформовано керівництво користувача. Проведено тестування системи та виконано оцінку ефективності створеного програмного забезпечення. Оцінка ефективності показала, що реалізоване програмне забезпечення здатне успішно виконувати поставлені задачі та задовольняє визначеним функціональним та нефункціональним вимогам.

4 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ

4.1 Опис ідеї проєкту

Таблиця 4.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користучача
Розробка програмне забезпечення для підготовки набору даних з кількох джерел за допомогою інструкцій природною мовою	1. Для досвідченого користувача – генерація команд за допомогою природної мови та редагування їх за потреби	1. Пришвидшення взаємодії зі сховищем даних; 2. Зменшення кількості помилок за рахунок «додаткової точки зору» зі сторони автоматизованої системи; 3. Полегшення взаємодії з декількома джерелами даних за рахунок їх об’єднання у спільний інтерфейс.
	2. Для недосвідченого користувача – генерація запитів до джерел даних без знання синтаксису мови цих запитів	1. Можливість взаємодіяти зі сховищем даних без відповідного знання синтаксису мови запитів до сховища (до прикладу, мови SQL)

Таблиця 4.2 — Опис ідеї стартап-проєкту

№	Техніко-економічні характеристики ідеї	Продукція конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	Power BI	Github Copilot	Google			
1	Підтримка текстових команд англійською	Так	Так	Так	Так		+	

Кінець таблиці 4.2

2	Підтримка голосових команд англійською	Так	Ні	Ні	Так			+
3	Можливість перегляду та редагування безпосередніх команд до сховища даних	Так	Ні	Так	Ні			+
4	Можливість отримати набори даних у форматі, готовому до подальшої програмної обробки	Так	Ні	Так	Ні		+	
5	Взаємодія з приватними джерелами даних	Так	Так	Так	Ні			+
6	Взаємодія з декількома джерелами даних	Так	Так	Ні	Так			+

4.2 Технологічний аудит ідеї проєкту

Таблиця 4.3 — Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Перетворення голосових команд у текстові	Модель Whisper від OpenAi	Наявна	Доступна
2	Перетворення текстових команд у SQL	Модель GPT4 від OpenAI	Наявна	Доступна
3	Об’єднання даних з декількох джерел у спільний SQL інтерфейс з використанням патерну data warehouse	Продукт BigQuery від Google	Наявна	Доступна
4	Задання команд та перегляд результатів через веб інтерфейс	Графічний веб інтерфейс користувача	Потребує розробки	Досупна

Висновок: технологічна реалізація проекту можлива і визначено технологічний шлях, яким доцільно реалізувати ідею проекту.

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Таблиця 4.4 — Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн./ум.од	15000грн*100од. = 1 500 000 грн
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Вартість рішеня має бути дешевша, ніж власна розробка
5	Специфічні вимоги до стандартизації та сертифікації	ISO/IEC 27001
6	Середня норма рентабельності в галузі або по ринку, %	500 000/779 000*100 = 192%

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 4.5 — Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Можливість об’єднувати дані з декількох джерел	Аналітики, що мають працювати на базі даних з різних джерел	Аналітики часто працюють на базі даних, що надходять з різних джерел	Створений продукт має мати змогу формувати набори даних з різних джерел

Кінець таблиці 4.5

2	Можливість взаємодіяти зі сховищами даних без вивчення мови запитів до сховища	Користувачі без досвіду взаємодії зі сховищами даних	Користувачі без досвіду звикли працювати зі даними без додаткових ускладнень	Створений продукт має надавати інтерфейс природною мовою для взаємодії зі сховищем даних
3	Можливість задавати команди голосом	Користувачі, що використовують мобільні пристрої	На мобільних пристроях значно зручніше задати команду голосом, ніж набирати її у вигляді тексту	Створений продукт має підтримувати обробку голосових команд

Таблиця 4.6 — Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренти	Наявність конкурентів котрі надають схожі рішення	Зменшення ціни на поставлену послугу; Розробка унікальних характеристик товару; Надання ліцензій на обслуговування
2	Кошти на розробку та підтримку продукту	Закінчення грошей та недостатнє фінансування	Залучення додаткових інвесторів, мотивація роботи на перспективу; Ітеративна розробка продукту задля покрокового виведення продукту на ринок та отримання відповіді користувачів
3	Вихід аналогу	Вихід аналогу даного товару може призвести до знецінення та безідейності даного товару	Вихід товару на ринок в коротші строки з не повною, але достатньою, функціональністю для зацікавлення усіх цільових аудиторій; Проведення рекламної компанії

Таблиця 4.7 — Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Новий продукт	Вихід на ринок, Зменшення монополії, Надання нових рішень у сфері	Розробка нової функціональності; Вихід нової продукції на ринок; Надання різноманітних типів ліцензій в залежності від потреб користувача \ замовника.
2	Вихід аналогу	Надати продукт з певними характеристиками та можливостями що відсутні у компаній конкурентів	Аналіз ринку та користувачів задля задоволення їх потреб та надання функціональності у найкоротші строки за ціну, котра є дешевшою ніж у продуктів-замінників.
3	Зворотній зв’язок від користувачів	Можливість отримання необхідної інформації для вдосконалення продукту	Наявність вхідних даних та реакція на них з боку команди розробників задля задоволення потреб та бажань кінцевих користувачів системи кешування даних.
4	Грошова винагорода за рекламу	При достатньому попиту на систему кешування даних можлива комерціалізація продукту на основі реклами задля отримання грошової винагороди для подальшого розвиту продукту та оплати заробітної плати працівникам	Точкова комерціалізація продукту; Введення реклами; Ведення додаткових коштів у проєкт задля його подальшого розвитку.

Таблиця 4.9 — Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: монополістична	Товар від кожної компанії на ринку, являється недосконалим замінником товару, реалізованого іншими фірмами; На ринку є умови для входу та виходу; Ціна корелює між суперниками;	Розробка продукту з характеристиками, які покривають сфери вживання що не покривають інші товари-замінники; Кореляція цін у відповідності до товарів замінників; Різни типи ліцензій.
2	Рівень конкурентної боротьби: світовий	Всі продукти замінники розроблялись інтернаціональними командами з різних куточків світу, продукти не належать до певної держави, а належать команді розробників	Вихід на ринок збуту продукту з клієнто-необхідною функціональністю; Налагодження маркетингу на основних Інтернет ресурсах задля охоплення великої кількості потенційних користувачів; Надання бета-версій продукту.
3	Галузева ознака: внутрішньогалузева	Даний тип продукту може використовуватися тільки у сфері розробки IT додатків \ продуктів	Надання зручного, інтуїтивно зрозумілого інтерфейсу; Підтримка всім відомих методів взаємодії з середовищем розробки; Наявність документації та он-лайн підтримки.
4	Конкуренція за видами товарів: товарно-видова	Дана конкуренція – конкуренція між товарами одного виду.	Впровадження функціональності яка відсутня у товарів-замінників; Спрощення інтерфейсів; Надання підтримки.

Кінець таблиці 4.9

5	Характер конкурентних переваг: цінова та не цінова	Цінові переваги – точкова комерціалізація; Не цінова – надання функціональності, що відсутня у товарах-замінниках.	Надання платних ліцензій лише на критично важливу функціональність для клієнта з певним строком підтримки, що зазначена у відповідній ліцензії; Впровадження унікальної функціональності.
6	За інтенсивністю: марочна	Наявність унікального знаку що відрізняє даний продукт від продуктів-замінників	Впровадження власної назви та власного знаку.

Таблиця 4.10 — Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Github Copilot SQL інтерфейси	SpotIQ Power BI	Google BigQuery OpenAi	Аналітики даних, Недосвідчені користувачі	Google ChatGpt
Висновки	Прямі конкуренти здатні виконувати поставлені задачі, але вони роблять це менш ефективно	Потенційні конкуренти здатні вийти на ринок за умови створення нових продуктів	Постачальники надають послуги, що необхідні для виконання описаних задач	Клієнтами є платеспроможні користувачі, що здатні забезпечувати достатній фінансовий дохід	Клієнти можуть користуватись товарами-замінникам, але вони будуть отримувати гірший сервіс

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок: оскільки кожний з існуючих продуктів не

впливає у великій мірі на поточну ситуацію на ринку в цілому, кожний з існуючих продуктів має свою специфічну сферу використання та свої позитивні та негативні сторони щодо рішення певних типів задач, то робота та вихід на даний ринок є можливою і реалізованою задачею.

Для виходу на ринок продукт повинен мати функціонал що відсутній у продуктів-аналогів, повинен задовольняти потреби користувачів, мати необхідний та достатній функціонал з конфігурування, підтримку зі сторони розробників та можливість розробки спеціального функціоналу за відповідною ліцензією.

Таблиця 4.11 — Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Можливість обробки команд природною мовою	Дана можливість є основною причиною використання інтефрейсу
2	Підтримка голосових команд	Такий функціонал важливий для користувачів на мобільних пристроях
3	Автоматизована взаємодія зі сховищем даних	Взаємодія без необхідності задавати команди формальною мовою є важливим чинником для недосвідчених користувачів
4	Швидкодія генерації та виконання запитів	Швидкодія є важливим чинником для усіх користувачів
5	Наявність повідомлень про помилку	Наявність повідомлень про помилку дасть користувачам можливість виправити проблему та запустити програму ще раз
6	Зручний та приємний інтерфейс	Такий інтерфейс покращить досвід користування системою
7	Об’єднання даних з різних джерел	Таке об’єднання даних є важливим чинником для складних аналітичних запитів
8	Отримання результатів у форматі, що підходить для подальшої програмної обробки	Наявність результатів у такому форматі спрощує подальший аналіз отриманих результатів за допомогою автоматизованих систем

Таблиця 4.12 — Порівняльний аналіз сильних та слабких сторін системи

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
			-3	-2	-1	0	+1	+2	+3
1	Можливість обробки команд природною мовою	15			+				
2	Підтримка голосових команд	12					+		
3	Автоматизована взаємодія зі сховищем даних	18		+					
4	Швидкодія генерації та виконання запитів	15			+				
5	Наявність повідомлень про помилку	14			+				
6	Зручний та приємний інтерфейс	14			+				
7	Об’єднання даних з різних джерел	16				+			
8	Отримання результатів у форматі, що підходить для подальшої програмної обробки	14				+			

Таблиця 4.13 — SWOT аналіз стартап-проєкту

Сильні сторони (S): – підтримка голосових команд; – об’єднання даних з різних сховищ; – наявність повідомлень про помилки; – адаптивний графічний інтерфейс; – надання результатів у форматі, що підходить для подальшої обробки.	Слабкі сторони (W): – наявність помилок при обробці команд; – простий графічний інтерфейс; – відсутність візуалізації результатів.
Можливості (O): – вдосконалення механізму генерації SQL команд; – додавання візуалізації для отриманих результатів; – створення мобільного застосунку для додатку.	Загрози (T): – наявність конкурентів на ринку; – недоступність сторонніх сервісів, що використовуються програмою.

Таблиця 4.14 — Альтернативи ринкового впровадження стартап-проєкту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Безкоштовне надання певного функціоналу у користування споживачам на обмежений термін	Головний ресурс – люди, даний ресурс - наявний	2-3 місяці
2	Реклама	Залучення власних коштів для реклами товару	1-2 місяці
3	Написання статей та опис товару на відомих ресурсах	Головний ресурс – час, даний ресурс - наявний	2-3 тижні
4	Презентація товару на хакатонах й інших ІТ заходах	Ресурс – час та гроші для участі, наявні	1-3 місяці

4.4 Розроблення ринкової стратегії проєкту

Таблиця 4.15 — Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Недосвідчені користувачі сховищ даних	Середня, адже користувачам може бути важко працювати з новим продуктом	Середній попит	Середня	Середня
2	Досвідчені користувачі сховищ даних	Висока	Високий попит	Висока	Середня
3	Аналітики, що працюють з багатьма джерелами даних	Середня, адже аналітики зазвичай мають існуючі напрацювання для виконання описаних завдань	Середній попит	Висока	Середня
Які цільові групи обрано: досвідчені користувачі					

Відповідно до проведеного аналізу можна зробити висновок, що підходящою цільовою групою для розповсюдження даного програмного продукту є досвідчені користувачі, що часто взаємодіють з різними сховищами даних. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу, оскільки для досвідчених користувачів надається стандартизований продукт з можливістю розширення функціональності за домовленістю (відповідно до ліцензії).

Таблиця 4.16 — Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності що відсутня у товарів-замінників, підтримка клієнтів	Проведення реклами, освітлення унікальної функціональності через інтернет ресурси та інші канали, контакт напряму з споживачами; формування лояльності і прихильності споживачів	Зниження ступеню замінності товару; Прихильність клієнтів; Відмітні властивості товару; Відмітні характеристики товару;	Стратегія диференціації

Таблиця 4.17 — Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні, оскільки є товари-замінники, але дані товари замінники не мають деякого необхідного функціоналу	Так, ціль компанії знайти нових споживачів та, частково, забрати існуючих у конкурентів задля задоволення потреб останніх	Компанія частково копіює характеристики товару конкурента, основна ціль компанії розробка нового унікального функціоналу, з підтримкою основного функціоналу конкурентів	Стратегія заняття конкурентної ніші

Таблиця 4.18 — Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Здатність швидко генерувати запити до сховищ даних	Стратегія диференціації	Програмний продукт буде значно пришвидшувати роботу зі сховищами даних	Зручність, швидкість
2	Зменшення кількості помилок у складних запитах	Стратегія диференціації	Програмний продукт буде готувати дані автоматично, тим самим зменшуючи кількість помилок	Надійність, якість, захищеність

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття конкурентної ніші.

4.5 Розроблення маркетингової програми стартап-проєкту

Таблиця 4.19 — Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Пришвидшення роботи зі сховищами даних	Автоматизована генерація запитів	Можливість автоматизовано генерувати запити до сховища даних
2	Аналіз даних з декількох джерел	Об’єднання даних у спільний інтефрейс	Наявність можливості виконувати аналітичні запити на базі об’єднаного інтефрейсу
3	Задання голосових команд	Можливість задавати команди у вигляді аудіо файлів	Можливість швидко задати команду природною мовою без необхідності друкувати її у текстовому форматі

Таблиця 4.20 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
1. Товар за задумом	Програмний продукт для підготовки даних з декількох джерел за допомогою інструкцій природною мовою		
2. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх/Тл/Е/Ор
	Простота використання	М	Тл
	Дешева собівартість	Нм	Вр
	Масштабованість	М	Тх
	Швидкість роботи	Нм	Тх
	Універсальність	М	Тх
	Якість: ISO 19649:2017		
	Спосіб взаємодії: додаток у мережі інтернет		
	Марка: NaturalDb		
3. Товар із підкріпленням	До продажу: наявна повна документація, акції на придбання декількох ліцензій, знижки для певних сегментів на покупку товару		
	Після продажу: додаткова підтримка спеціалістів налаштування, підтримка з боку розробника		
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності, патент			

Таблиця 4.21 — Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
0	10$ за місяць	1000-10000$ в місяць	від 0 до 20$ в місяць

Таблиця 4.22 — Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Клієнти купують продукт безпосередньо у компанії-розробника	Постачальник забезпечує технічну підтримку продукту	Канал нульового рівня: виробник безпосередньо продає товар клієнту	Через сайт виробника

Таблиця 4.23 — Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Клієнти дізнаються про товар в інтернеті	Інтернет, соціальні мережі	1. Швидке генерування запитів; 2. Робота з декількома сховищами даних; 3. Підтримка голосових команд	Інформування потенційних клієнтів про існування продукту та його переваги	Застосунок є зручним у використанні та має функціонал із забезпечення потреб певної аудиторії

Як результат було створено ринкову (маркетингову) програму, що включає в себе визначення ключових переваг концепції потенційного товару, опис моделі товару, визначення меж встановлення ціни, формування системи збуту та концепцію маркетингових комунікацій.

4.6 Висновки по розділу

У четвертому розділі описано стратегії та підходи з розроблення стартап-проєкту, визначено наявність попиту, динаміку та рентабельність роботи ринку, у результаті встановлено, що існує можливість ринкової комерціалізації проєкту. Розглянувши потенційні групи клієнтів, бар’єри входження, стан конкуренції та конкурентоспроможність проєкту встановлено, що проєкт є перспективним. Розглянуто та вибрано альтернативу впровадження стартап-проєкту та доведено доцільність подальшої імплементації проєкту.

ВИСНОВКИ

У ході виконання магістерської дисертації розв’язана актуальна задача підготовки наборів даних з декількох джерел за допомогою інструкцій природною мовою.

У дослідженні проаналізовано предметну область інтерфейсів природної мови для сховищ даних та методів отримання даних з декількох джерел. Зроблено огляд публікацій у даній області, сформовано набір актуальних проблем та досліджено існуючі рішення для їх вирішення. Проаналізовано переваги та недоліки кожного з рішень, виділено можливості для покращення. Базуючись на результатах аналізу, описано актуальні потреби, що можуть бути реалізовані програмним забезпеченням, розробленим у магістерській дисертації.

На основі результатів дослідження предметної області, наявних досліджень та існуючих рішень, було сформульовано мету та задачі розробки програмного забезпечення для підготовки наборів даних з декількох джерел за допомогою інструкцій природною мовою. Розроблене програмне забезпечення працює з голосовими та текстовими інструкціями природною мовою, та буде перетворювати дані інструкції у запити до відповідних джерел даних. Взаємодія з користувачем відбувається за допомогою адаптивного веб-орієнтованого графічного інтефрейсу.

Враховуючи описані проблеми та існуючі рішення, сформовано набір функціональних та нефункціональних вимог до програмного забезпечення, що розробляється. На базі сформованих вимог підготовлено архітектурне рішення та вибрано підходящі технології для реалізації програмного забезпечення:

- використання моделей Whisper та GPT4 від OpenAi для роботи з командами природною мовою;
- використання підходу data warehouse та відповідного продукту BigQuery від Google для об’єднання даних з декількох джерел у спільних SQL інтефрейс;

- розробка frontend додатку, що відповідає за графічний інтерфейс користувача, та backend додатку, що відповідає за логіку система та інтеграцію зі сторонніми системами;
- використовувати бібліотеку React для розробки frontend додатку, та платформу dotnet для розробки backend додатку.

Вибрані інструменти для розробки є широко вживаними та безкоштовними інструментами, що довели свою ефективність на практиці.

У роботі розроблено відповідні frontend та backend додатки. Також налаштовано сховище на базі продукту Google BigQuery та розумний асистент на базі GPT4 для обробки команд природною мовою. Описано важливі етапи розробки програмного забезпечення, особливості реалізації та надано приклад графічного інтерфейсу користувача з описом відповідного керівництва.

Проведено експериментальні дослідження розробленого програмного забезпечення, виконано аналіз ефективності. Під час дослідження, програмне забезпечення показало задовільні результати роботи у 85% проведених тестів. У решті 15% тестів, обробка закінчилась з помилкою. У випадку помилки, програмне забезпечення надало відповідне повідомленння про помилку, та коректно припинило обробку запиту. У результаті проведених досліджень показано, що створене програмне забезпечення задовольняє поставленим функціональним і нефункціональним вимогам.

Проведений маркетинговий аналіз стартап-проєкту, опис ідеї проєкту, технологічний аудит ідеї проєкту та аналіз ринкових можливостей запуску стартап-проєкту. Розроблено ринкову стратегію проєкту та маркетингову програму стартап-проєкту.

Результати роботи над магістерською дисертацією пройшли апробацію на двух наукових конференціях: SoftTech 2022 та SoftTech 2023.

Наукова новизна одержаних результатів полягає у подальшому розвитку методів використання LLM моделей для підготовки масивів даних з декількох джерел за допомогою команд природньої мови.

Практична значимість одержаних результатів полягає у розробці веб-додатку, що здатний готувати набори даних з декількох джерел за допомогою інструкцій природною мовою. Даний додаток об’єднує джерела даних у спільний SQL інтерфейс за допомогою патерну data warehouse. Додаток здатний обробляти голосові та текстові команди англійською мовою, перетворювати їх в SQL до відповідного об’єднаного інтерфейсу, виконувати згенерований запит та надавати результати виконання користувачу у форматі CSV.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Sharma, Vaibhav; An Enlightening Assessment of Data Mart Exploration in Promptly Mounting Data Warehousing Consequence. [Електронний ресурс] - 2021 р. – Режим доступу до ресурсу: https://www.researchgate.net/publication/352062788_An_Enlightening_Assessment_of_Data_Mart_Exploration_in_Promptly_Mounting_Data_Warehousing_Consequence.

2) Опис топології «Зірка». *Wikipedia.* [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%97%D1%96%D1%80%D0%BA%D0%B0_(%D1%82%D0%BE%D0%BF%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F).

3) Визначення поняття Natural language user interface. *Wikipedia.* [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Natural-language_user_interface.

4) Сервіс пошуку наукових робіт Google Scholar. [Електронний ресурс] – Режим доступу до ресурсу: https://scholar.google.com/schhp?hl=en&as_sdt=0,5.

5) Сервіс пошуку наукових робіт IEEE Xplore. [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/Xplore/home.jsp.

6) Gary G. Hendrix, Chairperson; Natural-Language Interface. [Електронний ресурс] - 1982 р. – Режим доступу до ресурсу: https://aclanthology.org/J82-2002.pdf.

7) I. Androutsopoulos, G.D. Ritchie, P. Thanisch; Natural Language Interfaces to Databases – An Introduction. [Електронний ресурс] - 1995 р. – Режим доступу до ресурсу: https://arxiv.org/pdf/cmp-lg/9503016.pdf.

8) Mrs Neelu Nihalani, Sanjay Silakari, Mahesh Motwani; Natural language Interface for Database: A Brief review. [Електронний ресурс] - 2011 р. – Режим доступу до ресурсу:

https://www.researchgate.net/publication/266863909_Natural_language_Interface_for_Database_A_Brief_review.

9) Khadija Majhadi, Mustapha Machkour; The history and recent advances of Natural Language Interfaces for Databases Querying. [Електронний ресурс] - 2021 р. – Режим доступу до ресурсу: https://www.e3s-conferences.org/articles/e3sconf/pdf/2021/05/e3sconf_iccsre2021_01039.pdf.

10) SHANZA ABBAS, MUHAMMAD UMAIR KHAN, SCOTT UK-JIN LEE, ASAD ABBAS, ALI KASHIF BASHIR; A Review of NLIDB With Deep Learning: Findings,. [Електронний ресурс] - 2022 р. – Режим доступу до ресурсу: https://fardapaper.ir/mohavaha/uploads/2022/04/Fardapaper-A-Review-of-NLIDB-With-Deep-Learning-Findings-Challenges-and-Open-Issues.pdf.

11) Abdul Quamar, Vasilis Efthymiou, Chuan Lei, Fatma Özcan; Natural Language Interfaces to Data. [Електронний ресурс] – Режим доступу до ресурсу: https://www.nowpublishers.com/article/Details/DBS-078.

12) Radu Iacob, Florin Brad, Elena-Simona Apostol, Ciprian-Octavian Truica, Ionel Hosu, Traian Rebedea; Neural Approaches for Natural Language Interfaces to Databases: A Survey. [Електронний ресурс] - 2020 р. – Режим доступу до ресурсу: https://aclanthology.org/2020.coling-main.34.

13) Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, Dragomir Radev; TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation. [Електронний ресурс] - 2018 р. – Режим доступу до ресурсу: https://arxiv.org/pdf/1804.09769.pdf.

14) Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, Dongmei Zhang; Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. [Електронний ресурс] - 2019 р. – Режим доступу до ресурсу: https://arxiv.org/pdf/1905.08205.pdf.

15) Xiaojun Xu, Chang Liu, Dawn Song; SQLNet: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE WITHOUT REINFORCEMENT

LEARNING. [Електронний ресурс] - 2017 р. – Режим доступу до ресурсу: https://arxiv.org/pdf/1711.04436.pdf.

16) Katrin Affolter, Kurt Stockinger, Abraham Bernstein; A comparative survey of recent natural language interfaces for databases. [Електронний ресурс] - 2019 р. – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s00778-019-00567-8.

17) Machkour, Mustapha; A Model of a Generic Natural Language Interface for Querying Database. [Електронний ресурс] - 2016 р. – Режим доступу до ресурсу: https://www.researchgate.net/publication/293917895_A_Model_of_a_Generic_Natural_Language_Interface_for_Querying_Database.

18) Wenlu Wang, Yingtao Tian, Haixun Wang, Wei-Shinn Ku; A Natural Language Interface for Database: Achieving Transfer-learnability Using Adversarial Method for Question Understanding. [Електронний ресурс] - 2020 р. – Режим доступу до ресурсу: https://conferences.computer.org/icde/2020/pdfs/ICDE2020-5acyuqhpJ6L9P042wmjY1p/290300a097/290300a097.pdf.

19) Prasetya Utama, Nathaniel Weir, Fuat Basık, Carsten Binnig, Ugur Cetinteme; DBPal: An End-to-end Neural Natural Language Interface for Databases. [Електронний ресурс] - 2018 р. – Режим доступу до ресурсу: https://arxiv.org/pdf/1804.00401.pdf.

20) MVUMBI, TRESOR; NATURAL LANGUAGE INTERFACE TO RELATIONAL DATABASE: A SIMPLIFIED CUSTOMIZATION APPROACH. [Електронний ресурс] - 2016 р. – Режим доступу до ресурсу: https://core.ac.uk/download/pdf/185413068.pdf.

21) Filbert Reinaldha, Tricya E. Widagdo; Natural Language Interfaces to Database (NLIDB): Question handling and unit conversion. [Електронний ресурс] - 2014 р. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/7062663.

22) Yilin Gao, Keping Wang, Chengkang Xu; Natural Language Interface for Relational Database. [Електронний ресурс] - 2016 р. – Режим доступу до ресурсу: https://github.com/DukeNLIDB/NLIDB/blob/master/doc/report/final/final.pdf.

23) Rukshan Alexander, Prashanthi Rukshan, Sinnathamby Mahesan; Natural Language Web Interface for Database (NLWIDB). [Електронний ресурс] - 2013 р. – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/1308/1308.3830.pdf.

24) Julio Alejandro Villeda Maldonado, José Arturo Gaona Cuadra; Natural Language Interface to Database Using the DialogFlow Voice Recognition and Text Conversion API. [Електронний ресурс] - 2019 р. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/9082438/metrics#metrics.

25) S.J.Young; The design and implementation of dialogue control in voice operated database inquiry systems. [Електронний ресурс] - 1989 р. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/0885230889900028.

26) Varun Wahi, Prof. Avadhoot Joshi , Rohan Patel , Asad Mujawar, Nishant Tayde; Voice Controlled Database Analysis. [Електронний ресурс] - 2019 р. – Режим доступу до ресурсу: https://www.ijert.org/voice-controlled-database-analysis.

27) Prof. Rupesh Mahajan, Aiushman Roy, Akash Jadhav, Akash Rao, Himanshu Gosavi, Shubham Patil; DATABASE INTERACTION USING SPEECH RECOGNITION. [Електронний ресурс] - 2021 р. – Режим доступу до ресурсу: https://ijariie.com/AdminUploadPdf/Database_Interaction_using_Speech_Recognition_ijariie14830.pdf.

28) Yuanfeng Song, Raymond Chi-Wing Wong, Xuefang Zhao, Di Jiang; VoiceQuerySystem: A Voice-driven Database Querying System Using Natural Language Questions. [Електронний ресурс] - 2022 р. – Режим доступу до ресурсу: https://dl.acm.org/doi/abs/10.1145/3514221.3520158.

29) Fahmi Bargui, Jamel Feki, Hanene Ben-Abdallah; A NATURAL LANGUAGE APPROACH FOR DATA MART SCHEMA DESIGN. [Електронний ресурс] - 2008 р. – Режим доступу до ресурсу: https://www.researchgate.net/publication/228937783_A_natural_language_approach_for_data_mart_schema_design.

30) Fahmi Bargui, Hanêne Ben-Abdallah, Jamel Feki; A natural language-based approach for a semiautomatic data mart design and ETL generation DESIGN. [Електронний ресурс] - 2016 р. – Режим доступу до ресурсу: https://www.researchgate.net/publication/299646311_A_natural_language-based_approach_for_a_semi-automatic_data_mart_design_and_ETL_generation.

31) Scriney, Michael; Automating Data Mart Construction from Semi-structured Data Sources. [Електронний ресурс] - 2019 р. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/8852882.

32) Fahmi Bargui, Hanêne Ben-Abdallah, Jamel Feki; Enhancing the involvement of decision makers in data mart design. [Електронний ресурс] - 2019 р. – Режим доступу до ресурсу: https://www.researchgate.net/profile/Fahmi-Bargui/publication/332194752_Enhancing_the_involvement_of_decision_makers_in_data_mart_design/links/5ca9aa5e299bf118c4b8ba14/Enhancing-the-involvement-of-decision-makers-in-data-mart-design.pdf.

33) Rodolfo A. Pazos Range, Alexander Gelbukh, J. Javier González Barbosa, Erika Alarcón Ruiz, Alejandro Mendoza Mejía, A. Patricia Domínguez Sánchez; Spanish Natural Language Interface for a Relational Database Querying System. [Електронний ресурс] - 2002 р. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/3-540-46154-X_16.

34) Lahaye, Lyne; A Voice Interface System for Database Accesses using French. [Електронний ресурс] - 1989 р. – Режим доступу до ресурсу: https://spectrum.library.concordia.ca/id/eprint/4751/1/ML49089.pdf.

35) Xiaofeng Meng, Shan Wang; NChiql: The Chinese Natural Language Interface to Databases. [Електронний ресурс] - 2001 р. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/3-540-44759-8_16.

36) Sadullah Karimi, Annajiat Alim Rasel, Matin Saad Abdullah; Natural Language Query and Control Interface for Database Using Afghan Language Databases.

[Електронний ресурс] - 2022 р. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/9894168.

37) Пошукова система Wolfram Alpha. [Електронний ресурс] – Режим доступу до ресурсу: https://www.wolframalpha.com/.

38) Пошукова система Google. [Електронний ресурс] – Режим доступу до ресурсу: https://www.google.com/.

39) Розумний асистент ChatGpt. [Електронний ресурс] – Режим доступу до ресурсу: https://chat.openai.com/.

40) Розумний асистент Bard від Google. [Електронний ресурс] – Режим доступу до ресурсу: https://bard.google.com/chat.

41) Сервіс Github copilot. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/features/copilot.

42) Середовище розробки Visual Studio Code. [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/.

43) Система управління базами даних Microsoft SQL Server. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16.

44) Опис модуля English Query у СУБД MS SQL Server. [Електронний ресурс] – Режим доступу до ресурсу: https://flylib.com/books/en/2.900.1.60/1/.

45) Питання про відключення модуля English Query зі СУБД MS SQL Server. *StackOverflow.* [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/886502/what-happened-with-sql-english-query.

46) Продукт Power BI від компанії Microsoft. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/power-bi/.

47) Q&A інтефрейс у продукті Microsoft Power BI. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/power-bi/consumer/end-user-q-and-a.

48) Продукт SpotIQ від компанії ThoughtSpot. [Електронний ресурс] – Режим доступу до ресурсу: https://go.thoughtspot.com/demo-video-spotiq-0816-thank-you.html?q_pardotFormSubmitted.

49) Ціна ліцензій для продукту SpotIQ. [Електронний ресурс] – Режим доступу до ресурсу: https://www.thoughtspot.com/pricing.

50) Victor Zhong, Caiming Xiong, Richard Socher; SEQ2SQL: GENERATING STRUCTURED QUERIES. [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1709.00103.pdf.

51) Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, Jian-Guang Lou; UNISAR: A Unified Structure-Aware Autoregressive Language Model for. [Електронний ресурс] - 2022 р. – Режим доступу до ресурсу: https://github.com/microsoft/ContextualSP/tree/master/unified_parser_text_to_sql.

52) Fei Li, H. V. Jagadish; [Електронний ресурс] - 2014 р. – Режим доступу до ресурсу: https://www.researchgate.net/publication/266656491_NaLIR_An_interactive_natural_language_interface_for_querying_relational_databases.

53) Визначення терміну user story для опису вимог. [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/agile/project-management/user-stories.

54) Сховище Snowflake. [Електронний ресурс] – Режим доступу до ресурсу: https://www.snowflake.com/en/.

55) Data warehouse на базі хмарної платформи Azure. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/enterprise-data-warehouse.

56) Сховище Amazon Redshift від компанії Amazon. [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/redshift/.

57) Сховище Big Query від компанії Google. [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/bigquery?hl=en.

58) Модель Whisper від компанії OpenAi. [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/guides/speech-to-text.

59) Модель GPT4 від компанії OpenAi. [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/gpt-4.

60) Розумні асистенти на базі моделі GPT4 від OpenAI. [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/assistants/overview.

61) Опис архітектурного підходу Single page application. [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Single-page_application .

62) Бібліотека React. [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/.

63) Мова програмування TypeScript. [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/.

64) Бібліотека компонентів Material UI. [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/.

65) Бібліотека для запису аудіофайлів recorder-js. [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/recorder-js.

66) Опис технологічного стеку dotnet. [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/.

67) Мова програмування C#. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

68) Бібліотека під платформу .Net для взаємодії з сховищами на базі продукту BigQuery. [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/dotnet/docs/reference/Google.Cloud.BigQuery.V2/latest.

69) Датасет з продажами мережі Walmart. [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/datasets/willianoliveiragibin/sales-in-period-walmart.

70) Платформа OpenAI. [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/overview.

71) Ціни за використання моделей OpenAi. [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/pricing.

72) Фреймворк Asp.Net Core для створення WebApi на базі платформи dotnet. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&tabs=visual-studio.

73) Бібліотека Mediatr для реалізації патерну Mediator. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/jbogard/MediatR.

74) Вихідний код backend додатку у відкритому доступі. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ArtemK123/NaturalDbBackend.

75) Інструкція по взаємодії з моделлю Whisper від OpenAi. [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/guides/speech-to-text.

76) Інструкція для взаємодії з асистентами від OpenAi за допомогою програмного інтерфейсу. [Електронний ресурс] – Режим доступу до ресурсу: https://platform.openai.com/docs/assistants/how-it-works.

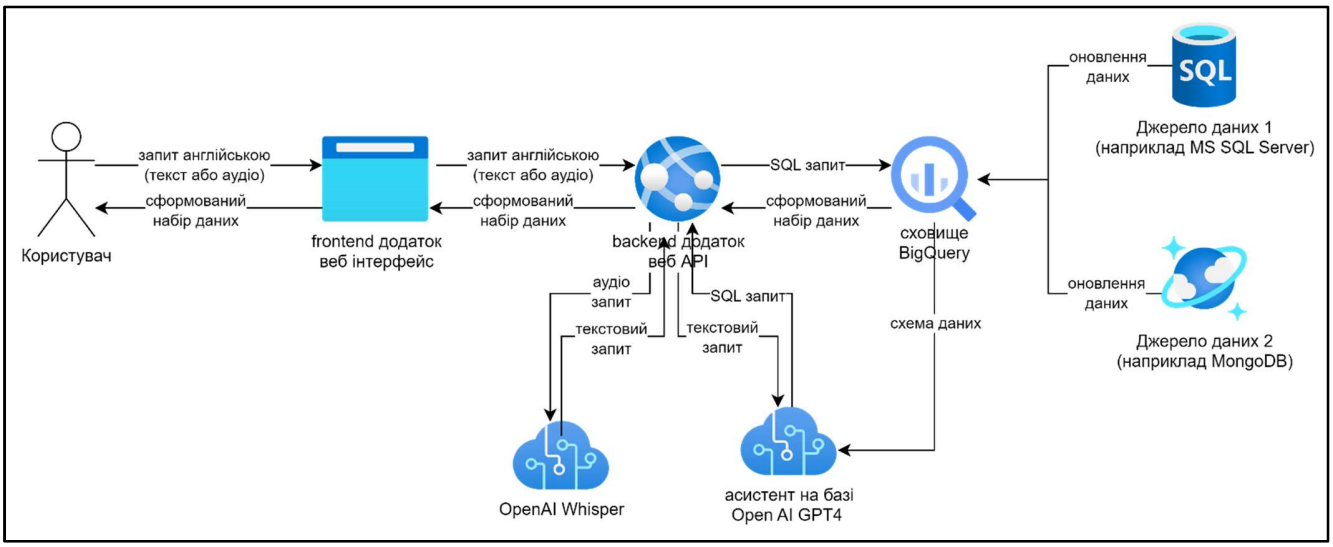
77) Технологія Swagger для взаємодії з програмними веб інтерфейсами. [Електронний ресурс] – Режим доступу до ресурсу: https://swagger.io/.

78) Вихідний код frontend додатку у відкритому доступі. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ArtemK123/NaturalDbFrontend.

ДОДАТКИ

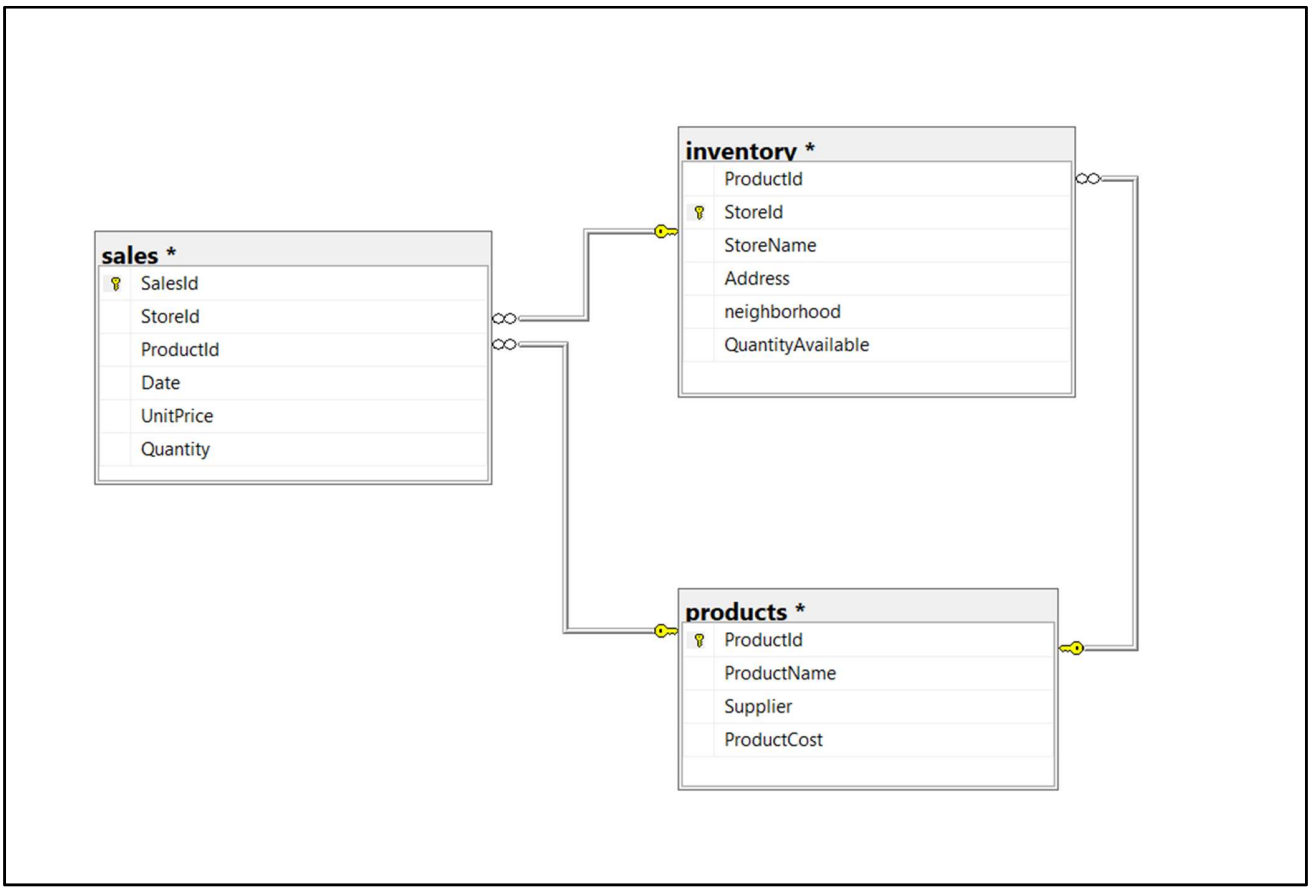
ДОДАТОК А

Запропонована архітектура програмного забезпечення



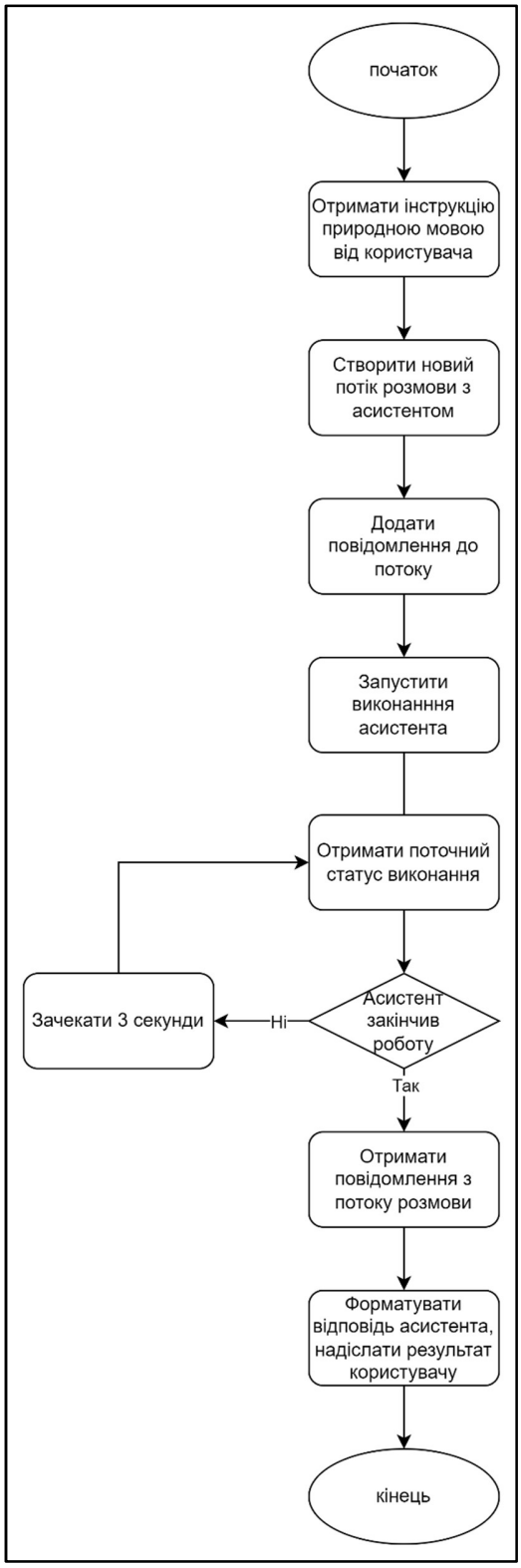
ДОДАТОК Б

Діаграма сутностей та залежностей у створеному сховищі даних



ДОДАТОК В

Алгоритм перетворення інструкції природною мовою в SQL команду



ДОДАТОК Г

Лістинг коду додатку NaturalDb

File: Program.cs

```
01: using System.Reflection;
02: using Microsoft.Extensions.Options;
03: using WebApi;
04: using WebApi.Options;
05: using WebApi.Services;
06:
07: var localFrontendCorsPolicyName = "LocalFrontendCorsPolicy";
08:
09: var builder = WebApplication.CreateBuilder(args);
10:
11: builder.Services.AddCors(options =>
12: {
13:     options.AddPolicy(name: localFrontendCorsPolicyName,
14:         policy  =>
15:         {
16:             policy.WithOrigins("http://localhost:3000")
17:                 .AllowAnyHeader()
18:                 .AllowAnyMethod();
19:         });
20: });
21:
22: // Add services to the container.
23:
24: builder.Services.AddControllers();
25: // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
26: builder.Services.AddEndpointsApiExplorer();
27: builder.Services.AddSwaggerGen();
28: builder.Services.AddHttpClient();
29: builder.Services.AddMediatR(cfg => cfg.RegisterServicesFromAssembly(Assembly.GetExecutingAssembly()));
30:
31: builder.Services.Configure<OpenAiOptions>(builder.Configuration.GetSection(OpenAiOptions.SectionPath));
32: builder.Services.AddTransient<OpenAiOptions>(s => s.GetRequiredService<IOptions<OpenAiOptions>>().Value);
33:
34: builder.Services.Configure<BigQueryOptions>(builder.Configuration.GetSection(BigQueryOptions.SectionPath));
35: builder.Services.AddTransient<BigQueryOptions>(s => s.GetRequiredService<IOptions<BigQueryOptions>>().Value);
36:
37: builder.Services.AddTransient<IOpenAiMessageSender, OpenAiMessageSender>();
38:
39: var app = builder.Build();
40:
41: // Configure the HTTP request pipeline.
42: if (app.Environment.IsDevelopment())
43: {
44:     app.UseCors(localFrontendCorsPolicyName);
45: }
46:
47: app.UseSwagger();
48: app.UseSwaggerUI(o => {
```

```
49:     o.EnableTryItOutByDefault();
50: });
51:
52: app.UseHttpsRedirection();
53:
54: app.UseAuthorization();
55:
56: app.MapControllers();
57:
58: app.Run();
```

File: IndexController.cs

```
01: using MediatR;
02: using Microsoft.AspNetCore.Mvc;
03: using WebApi.UseCases.BigQuery;
04: using WebApi.UseCases.OpenAi;
05: using WebApi.UseCases.OpenAi.Models;
06: using WebApi.UseCases.OpenAi.TranslateAudioToText;
07:
08: namespace WebApi.Controllers;
09:
10: [ApiController]
11: [Route("")]
12: public class NaturalLanguageController : ControllerBase
13: {
14:     private readonly IMediator mediator;
15:
16:     public NaturalLanguageController(IMediator mediator)
17:     {
18:         this.mediator = mediator;
19:     }
20:
21:     [HttpPost("audio-to-text")]
22:     public async Task<TranscriptAudioFileResponse> AudioToText(IFormFile file, CancellationToken cancellationToken)
23:     {
24:         using var fileStream = file.OpenReadStream();
25:         var result = await mediator.Send(new TranscriptAudioFileRequest(fileStream, file.FileName), cancellationToken);
26:         return result;
27:     }
28:
29:     [HttpPost("text-to-sql")]
30:             public async Task<ConvertTextToSqlResponse> TextToSql([FromBody] ConvertTextToSqlRequest request,
CancellationToken cancellationToken)
31:     {
32:         var response = await mediator.Send(request, cancellationToken);
33:         return response;
34:     }
35:
36:     [HttpPost("get-thread-messages/{threadId}")]
37:                 public async Task<MessageListModel> CheckTread([FromRoute] string threadId, CancellationToken
cancellationToken)
38:     {
39:         var messageList = await mediator.Send(new RetrieveMessageListRequest(threadId), cancellationToken);
40:         return messageList;
```

```
41:    }
42:
43:    [HttpPost("execute-in-big-query")]
44:        public async Task<ExecuteSqlInBigQueryResponse> ExecuteBigQuery([FromBody]ExecuteSqlInBigQueryRequest
request, CancellationToken cancellationToken)
45:    {
46:        var response = await mediator.Send(request, cancellationToken);
47:        return response;
48:    }
49: }
50:
```

File: OpenAiMessageSender.cs

```
01: using System.Net.Http.Headers;
02: using System.Text.Json;
03: using System.Text.Json.Serialization;
04: using Microsoft.Extensions.Options;
05:
06: namespace WebApi.Services;
07:
08: public interface IOpenAiMessageSender
09: {
10:        public Task Send(HttpRequestMessage request, CancellationToken cancellationToken);
11:
12:        public Task<TResponse> Send<TResponse>(HttpRequestMessage request, CancellationToken cancellationToken);
13: }
14:
15: internal sealed class OpenAiMessageSender : IOpenAiMessageSender
16: {
17:    private const string BearerSchemaName = "Bearer";
18:
19:    private readonly HttpClient httpClient;
20:        private readonly OpenAiOptions openAiOptions;
21:
22:                public OpenAiMessageSender(IOptions<OpenAiOptions> openAiOptionsProvider, IHttpClientFactory
httpClientFactory)
23:    {
24:        openAiOptions = openAiOptionsProvider.Value;
25:                    httpClient = httpClientFactory.CreateClient();
26:    }
27:
28:    public async Task Send(HttpRequestMessage request, CancellationToken cancellationToken)
29:    {
30:        await SendInternal(request, cancellationToken);
31:    }
32:
33:              public async Task<TResponse> Send<TResponse>(HttpRequestMessage request, CancellationToken
cancellationToken)
34:    {
35:        var response = await SendInternal(request, cancellationToken);
36:        var options = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.SnakeCaseLower };
37:        options.Converters.Add(new JsonStringEnumConverter());
38:                    var result =  await response.Content.ReadFromJsonAsync<TResponse>(options, cancellationToken);
39:                    return result!;
```

```
40:         }
41:
42:         private async Task<HttpResponseMessage> SendInternal(HttpRequestMessage request, CancellationToken cancellationToken)
43:         {
44:         request.Headers.Add(openAiOptions.OpenAiBetaHeaderName, openAiOptions.OpenAiBetaHeaderValue);
45:         request.Headers.Authorization = new AuthenticationHeaderValue(BearerSchemaName, openAiOptions.ApiKey);
46:
47:                     var response = await httpClient.SendAsync(request, cancellationToken);
48:         ValidateOpenAiResponse(response);
49:
50:                     return response;
51:         }
52:
53:         private void ValidateOpenAiResponse(HttpResponseMessage response)
54:     {
55:         if (!response.IsSuccessStatusCode)
56:                     {
57:                         throw new InvalidOperationException($"OpenAi call failed. Response - {response.StatusCode} {response.ReasonPhrase}");
58:         }
59:     }
60: }
61:
```

File: ExecuteSqlInBigQuery.cs

```
01: using System.Text;
02: using Google.Cloud.BigQuery.V2;
03: using MediatR;
04: using WebApi.Options;
05:
06: namespace WebApi.UseCases.BigQuery;
07:
08: public record ExecuteSqlInBigQueryRequest(string Sql) : IRequest<ExecuteSqlInBigQueryResponse>;
09:
10: public record ExecuteSqlInBigQueryResponse(string ResultsInCsv);
11:
12:         public class ExecuteSqlInBigQueryHandler : IRequestHandler<ExecuteSqlInBigQueryRequest, ExecuteSqlInBigQueryResponse>
13: {
14:     private readonly ILogger<ExecuteSqlInBigQueryHandler> logger;
15:     private readonly BigQueryOptions bigQueryOptions;
16:
17:                 public ExecuteSqlInBigQueryHandler(ILogger<ExecuteSqlInBigQueryHandler> logger, BigQueryOptions bigQueryOptions)
18:     {
19:         this.logger = logger;
20:         this.bigQueryOptions = bigQueryOptions;
21:     }
22:
23:     public async Task<ExecuteSqlInBigQueryResponse> Handle(ExecuteSqlInBigQueryRequest request, CancellationToken cancellationToken)
24:     {
```

```
25:             logger.LogInformation("Executing Sql in BigQuery. Sql - {sql}, BigQuery projectId - {projectId}", request.Sql, bigQueryOptions.ProjectId);
26:
27:         // TODO: Investigate authorization of big query
28:         BigQueryClient client = await BigQueryClient.CreateAsync(bigQueryOptions.ProjectId);
29:                                     BigQueryResults resultsEnumerable = await client.ExecuteQueryAsync(request.Sql, Enumerable.Empty<BigQueryParameter>());
30:
31:         IReadOnlyCollection<BigQueryRow> resultsArray = resultsEnumerable.ToArray();
32:
33:         logger.LogInformation(
34:             "Executed Sql in BigQuery. Count of received rows - {countOfRows}, Sql - {sql}, BigQuery projectId - {projectId}",
35:             resultsArray.Count,
36:             request.Sql,
37:             bigQueryOptions.ProjectId);
38:
39:         var resultsInCsv = ConvertResultsToCsv(resultsArray);
40:         return new ExecuteSqlInBigQueryResponse(resultsInCsv);
41:     }
42:
43:     private string ConvertResultsToCsv(IReadOnlyCollection<BigQueryRow> results)
44:     {
45:         StringBuilder csvResult = new();
46:         var separator = ',';
47:
48:         if (results.Count != 0)
49:         {
50:             var firstResult = results.First();
51:             var fieldNames = firstResult.Schema.Fields.Select(f => f.Name);
52:             var header = string.Join(separator, fieldNames);
53:             csvResult.AppendLine(header);
54:         }
55:
56:         foreach (BigQueryRow row in results)
57:         {
58:             var line = string.Join(separator, row.RawRow.F.Select(field => field.V));
59:             csvResult.AppendLine(line);
60:         }
61:
62:         return csvResult.ToString();
63:     }
64: }
65:
```

File: AddMessageToThread.cs

```
01: using MediatR;
02: using WebApi.Services;
03:
04: namespace WebApi.UseCases.OpenAi;
05:
06: public record AddMessageToThreadRequest(string ThreadId, string Message) : IRequest;
07:
08: public sealed class AddMessageToThreadHandler : IRequestHandler<AddMessageToThreadRequest>
09: {
```

```
10:     private readonly ILogger<AddMessageToThreadHandler> logger;
11:     private readonly OpenAiOptions openAiOptions;
12:     private readonly IOpenAiMessageSender openAiMessageSender;
13:
14:     public AddMessageToThreadHandler(ILogger<AddMessageToThreadHandler> logger, OpenAiOptions openAiOptions,
IOpenAiMessageSender openAiMessageSender)
15:     {
16:         this.logger = logger;
17:         this.openAiOptions = openAiOptions;
18:         this.openAiMessageSender = openAiMessageSender;
19:     }
20:
21:     public async Task Handle(AddMessageToThreadRequest request, CancellationToken cancellationToken)
22:     {
23:             logger.LogInformation("Adding a message to the OpenAi thread. ThreadId - {threadId}, Message - {message}",
request.ThreadId, request.Message);
24:
25:         var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/messages";
26:         using var httpRequest = new HttpRequestMessage(HttpMethod.Post, url)
27:         {
28:             Content = JsonContent.Create(new
29:             {
30:                 role = "user",
31:                 content = request.Message
32:             })
33:         };
34:
35:         await openAiMessageSender.Send(httpRequest, cancellationToken);
36:             logger.LogInformation("Added a message to the OpenAi thread. ThreadId - {threadId}, Message - {message}",
request.ThreadId, request.Message);
37:     }
38: }
39:
```

File: ConvertTextToSql.cs

```
01: using MediatR;
02: using WebApi.UseCases.OpenAi.Constants;
03: using WebApi.UseCases.OpenAi.Models;
04:
05: namespace WebApi.UseCases.OpenAi;
06:
07: public record ConvertTextToSqlRequest(string Text) : IRequest<ConvertTextToSqlResponse>;
08:
09: public record ConvertTextToSqlResponse(string Sql);
10:
11: public sealed class ConvertTextToSqlHandler : IRequestHandler<ConvertTextToSqlRequest, ConvertTextToSqlResponse>
12: {
13:     private readonly ILogger<ConvertTextToSqlHandler> logger;
14:     private readonly IMediator mediator;
15:     private readonly OpenAiOptions openAiOptions;
16:
17:         public ConvertTextToSqlHandler(ILogger<ConvertTextToSqlHandler> logger, IMediator mediator, OpenAiOptions
openAiOptions)
18:     {
```

```
19:         this.logger = logger;
20:         this.mediator = mediator;
21:         this.openAiOptions = openAiOptions;
22:     }
23:
24:         public async Task<ConvertTextToSqlResponse> Handle(ConvertTextToSqlRequest request, CancellationToken cancellationToken)
25:     {
26:         logger.LogInformation(
27:             "Converting text to sql with OpenAi assistant. Text - {text}, Timeout - {timeout}ms",
28:             request.Text,
29:             openAiOptions.TextToSqlTimeoutInSeconds);
30:
31:         var response = await ConvertTextToSql(request, cancellationToken)
32:             .WaitAsync(TimeSpan.FromSeconds(openAiOptions.TextToSqlTimeoutInSeconds), cancellationToken);
33:
34:             logger.LogInformation("Converted text to sql with OpenAi assistant. Text - {text}, Sql - {sql}", request.Text, response.Sql);
35:         return response;
36:     }
37:
38:             private async Task<ConvertTextToSqlResponse> ConvertTextToSql(ConvertTextToSqlRequest request, CancellationToken cancellationToken)
39:     {
40:         var createThreadResponse = await mediator.Send(new CreateThreadRequest(), cancellationToken);
41:         var threadId = createThreadResponse.ThreadId;
42:
43:         await mediator.Send(new AddMessageToThreadRequest(threadId, Message: request.Text), cancellationToken);
44:
45:         var runAssistantResponse = await mediator.Send(
46:             new RunAssistantRequest(threadId, AssistantId: openAiOptions.TextToSqlAssistantId),
47:             cancellationToken);
48:         var runId = runAssistantResponse.StartedRun.Id;
49:
50:         await WaitUntilRunCompletes(threadId: threadId, runId: runId, cancellationToken);
51:
52:         var messageList = await mediator.Send(new RetrieveMessageListRequest(threadId), cancellationToken);
53:
54:         var convertedSql = GetContentOfLastMessageFromAssistant(messageList);
55:         var formattedSql = FormatSql(convertedSql);
56:
57:         var response = new ConvertTextToSqlResponse(Sql: formattedSql);
58:         return response;
59:     }
60:
61:     private async Task WaitUntilRunCompletes(string threadId, string runId, CancellationToken cancellationToken)
62:     {
63:         while (true)
64:         {
65:                 var currentRunState = await mediator.Send(new RetrieveRunRequest(ThreadId: threadId, RunId: runId), cancellationToken);
66:
67:             if (currentRunState.Status == RunStatuses.Completed)
68:             {
```

```
69:             return;
70:         }
71:
72:         logger.LogInformation(
73:             "Run is not completed yet. Waiting {timeout}ms before next check. Current run status - {status}, ThreadId - {threadId} RunId - {runId}",
74:             openAiOptions.TextToSqlStatusCheckIntervalInMs,
75:             currentRunState.Status,
76:             threadId,
77:             runId);
78:
79:                         await Task.Delay(TimeSpan.FromMilliseconds(openAiOptions.TextToSqlStatusCheckIntervalInMs), cancellationToken);
80:     }
81: }
82:
83: private string GetContentOfLastMessageFromAssistant(MessageListModel messageList)
84: {
85:     var lastAssistantMessage = messageList.Data.Last(message => message.Role == MessageRoles.Assistant);
86:     var lastMessageContent = lastAssistantMessage.Content.Last();
87:     var result = lastMessageContent.Text.Value;
88:
89:     return result;
90: }
91:
92: private string FormatSql(string rawSql)
93: {
94:     var lineBreakSymbols = new [] { '\n', '\r' };
95:     var formatted = lineBreakSymbols.Aggregate(rawSql, (sql, symbol) => sql.Replace(symbol, ' '));
96:     return formatted;
97: }
98: }
```

File: CreateThread.cs

```
01: using System.Dynamic;
02: using MediatR;
03: using WebApi.Services;
04:
05: namespace WebApi.UseCases.OpenAi;
06:
07: public record CreateThreadRequest : IRequest<CreateThreadResponse>;
08:
09: public record CreateThreadResponse(string ThreadId);
10:
11: public sealed class CreateThreadHandler : IRequestHandler<CreateThreadRequest, CreateThreadResponse>
12: {
13:     private readonly ILogger<CreateThreadHandler> logger;
14:     private readonly OpenAiOptions openAiOptions;
15:     private readonly IOpenAiMessageSender openAiMessageSender;
16:
17:         public CreateThreadHandler(ILogger<CreateThreadHandler> logger, OpenAiOptions openAiOptions, IOpenAiMessageSender openAiMessageSender)
18:     {
19:         this.logger = logger;
```

```
20:         this.openAiOptions = openAiOptions;
21:         this.openAiMessageSender = openAiMessageSender;
22:     }
23:
24:             public async Task<CreateThreadResponse> Handle(CreateThreadRequest request, CancellationToken
cancellationToken)
25:     {
26:         logger.LogInformation("Creating OpenAi thread");
27:
28:         using var httpRequest = new HttpRequestMessage(HttpMethod.Post, openAiOptions.ThreadsEndpoint);
29:         httpRequest.Content = JsonContent.Create(new object());
30:
31:         var creationModel = await openAiMessageSender.Send<ThreadCreationModel>(httpRequest, cancellationToken);
32:
33:         var response = new CreateThreadResponse(ThreadId: creationModel.Id);
34:         logger.LogInformation("Created OpenAi thread. ThreadId - {threadId}", creationModel.Id);
35:
36:         return response;
37:     }
38:
39:     internal sealed record ThreadCreationModel(string Id, string Object, int CreatedAt, ExpandoObject Metadata);
40: }
41:
```

File: RetrieveMessageList.cs

```
01: using MediatR;
02: using WebApi.Services;
03: using WebApi.UseCases.OpenAi.Models;
04:
05: namespace WebApi.UseCases.OpenAi;
06:
07: public record RetrieveMessageListRequest(string ThreadId) : IRequest<MessageListModel>;
08:
09: public class RetrieveMessageListHandler : IRequestHandler<RetrieveMessageListRequest, MessageListModel>
10: {
11:     private readonly ILogger<RetrieveMessageListHandler> logger;
12:     private readonly OpenAiOptions openAiOptions;
13:     private readonly IOpenAiMessageSender openAiMessageSender;
14:
15:         public RetrieveMessageListHandler(ILogger<RetrieveMessageListHandler> logger, OpenAiOptions openAiOptions,
IOpenAiMessageSender openAiMessageSender)
16:     {
17:         this.logger = logger;
18:         this.openAiOptions = openAiOptions;
19:         this.openAiMessageSender = openAiMessageSender;
20:     }
21:
22:             public async Task<MessageListModel> Handle(RetrieveMessageListRequest request, CancellationToken
cancellationToken)
23:     {
24:         logger.LogInformation("Retrieving a message list. ThreadId - {threadId}", request.ThreadId);
25:
26:         var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/messages";
27:         using var httpRequest = new HttpRequestMessage(HttpMethod.Get, url);
```

```
28:
29:         var messageList = await openAiMessageSender.Send<MessageListModel>(httpRequest, cancellationToken);
30:
31:          logger.LogInformation("Retrieved the message list. ThreadId - {threadId}, MessageList - {list}", request.ThreadId,
messageList);
32:         return messageList;
33:     }
34: }
35:
```

File: RetrieveRun.cs

```
01: using MediatR;
02: using WebApi.Services;
03: using WebApi.UseCases.OpenAi.Models;
04:
05: namespace WebApi.UseCases.OpenAi;
06:
07: public record RetrieveRunRequest(string ThreadId, string RunId) : IRequest<RunModel>;
08:
09: public sealed class RetrieveRunHandler : IRequestHandler<RetrieveRunRequest, RunModel>
10: {
11:     private readonly ILogger<RetrieveRunHandler> logger;
12:     private readonly OpenAiOptions openAiOptions;
13:     private readonly IOpenAiMessageSender openAiMessageSender;
14:
15:                 public RetrieveRunHandler(ILogger<RetrieveRunHandler> logger, OpenAiOptions openAiOptions,
IOpenAiMessageSender openAiMessageSender)
16:     {
17:         this.logger = logger;
18:         this.openAiOptions = openAiOptions;
19:         this.openAiMessageSender = openAiMessageSender;
20:     }
21:
22:     public async Task<RunModel> Handle(RetrieveRunRequest request, CancellationToken cancellationToken)
23:     {
24:         logger.LogInformation("Retrieving a run. ThreadId - {threadId}, RunId - {runId}", request.ThreadId, request.RunId);
25:
26:         var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/runs/{request.RunId}";
27:         using var httpRequest = new HttpRequestMessage(HttpMethod.Get, url);
28:
29:         var run = await openAiMessageSender.Send<RunModel>(httpRequest, cancellationToken);
30:
31:         logger.LogInformation("Retrieved the run. ThreadId - {threadId}, Run - {run}", request.ThreadId, run);
32:         return run;
33:     }
34: }
35:
```

File: RunAssistant.cs

```
01: using MediatR;
02: using WebApi.Services;
03: using WebApi.UseCases.OpenAi.Models;
04:
05: namespace WebApi.UseCases.OpenAi;
```

```
06:
07: public record RunAssistantRequest(string ThreadId, string AssistantId) : IRequest<RunAssistantResponse>;
08:
09: public record RunAssistantResponse(RunModel StartedRun);
10:
11: public sealed class RunAssistantHandler : IRequestHandler<RunAssistantRequest, RunAssistantResponse>
12: {
13:     private readonly ILogger<RunAssistantHandler> logger;
14:     private readonly OpenAiOptions openAiOptions;
15:     private readonly IOpenAiMessageSender openAiMessageSender;
16:
17:                 public RunAssistantHandler(ILogger<RunAssistantHandler> logger, OpenAiOptions openAiOptions,
IOpenAiMessageSender openAiMessageSender)
18:     {
19:         this.logger = logger;
20:         this.openAiOptions = openAiOptions;
21:         this.openAiMessageSender = openAiMessageSender;
22:     }
23:
24:             public async Task<RunAssistantResponse> Handle(RunAssistantRequest request, CancellationToken
cancellationToken)
25:     {
26:                 logger.LogInformation("Running OpenAi assistant. ThreadId - {threadId}, AssistantId - {assistantId}",
request.ThreadId, request.AssistantId);
27:
28:         var url = $"{openAiOptions.ThreadsEndpoint}/{request.ThreadId}/runs";
29:         using var httpRequest = new HttpRequestMessage(HttpMethod.Post, url)
30:         {
31:             Content = JsonContent.Create(new
32:             {
33:                 assistant_id = request.AssistantId,
34:             })
35:         };
36:
37:         var threadRun = await openAiMessageSender.Send<RunModel>(httpRequest, cancellationToken);
38:
39:         logger.LogInformation(
40:             "Started the run of OpenAi assistant. ThreadId - {threadId}, AssistantId - {assistantId}, RunId - {runId}",
41:             request.ThreadId,
42:             request.AssistantId,
43:             threadRun.Id);
44:
45:         var response = new RunAssistantResponse(threadRun);
46:         return response;
47:     }
48: }
```

File: TranscriptAudioFile.cs

```
01: using MediatR;
02: using WebApi.Services;
03:
04: namespace WebApi.UseCases.OpenAi.TranslateAudioToText;
05:
```

```
06:  public record TranscriptAudioFileRequest(Stream FileStream, string FileName) : IRequest<TranscriptAudioFileResponse>;
07:
08: public record TranscriptAudioFileResponse(string Text);
09:
10: public class TranscriptAudioFileHandler : IRequestHandler<TranscriptAudioFileRequest, TranscriptAudioFileResponse>
11: {
12:     private readonly ILogger<TranscriptAudioFileHandler> logger;
13:     private readonly OpenAiOptions openAiOptions;
14:     private readonly IOpenAiMessageSender openAiMessageSender;
15:
16:         public TranscriptAudioFileHandler(ILogger<TranscriptAudioFileHandler> logger, OpenAiOptions openAiOptions, IOpenAiMessageSender openAiMessageSender)
17:     {
18:         this.logger = logger;
19:         this.openAiOptions = openAiOptions;
20:         this.openAiMessageSender = openAiMessageSender;
21:     }
22:
23:         public async Task<TranscriptAudioFileResponse> Handle(TranscriptAudioFileRequest request, CancellationToken cancellationToken)
24:     {
25:         logger.LogInformation("Transcripting audio file. FileName - {fileName}", request.FileName);
26:
27:         using var whisperRequest = new HttpRequestMessage(HttpMethod.Post, openAiOptions.WhisperEndpoint);
28:         whisperRequest.Content = new MultipartFormDataContent
29:         {
30:             { new StreamContent(request.FileStream), "file", request.FileName },
31:             { new StringContent(openAiOptions.WhisperModel), "model" },
32:             { new StringContent("en"), "language" }
33:         };
34:
35:         var recognition = await openAiMessageSender.Send<WhisperRecognition>(whisperRequest, cancellationToken);
36:
37:         var result = new TranscriptAudioFileResponse(recognition.Text);
38:
39:       logger.LogInformation("Transripted audio file. FileName - {fileName}, ConvertionResult - {result}", request.FileName, result);
40:         return result;
41:     }
42:
43:     public record WhisperRecognition(string Text);
44: }
45:
```

File: BigQueryOptions.cs

```
1: namespace WebApi.Options;
2:
3: public class BigQueryOptions
4: {
5:     public const string SectionPath = "BigQuery";
6:
7:     public string ProjectId { get; set; } = null!;
8: }
```

```
9:
```

File: OpenAiOptions.cs

```
01: namespace WebApi;
02:
03: public class OpenAiOptions
04: {
05:     public const string SectionPath = "OpenAi";
06:
07:     public string ApiKey { get; set; } = null!;
08:
09:     public string OpenAiBetaHeaderName { get; set; } = null!;
10:
11:     public string OpenAiBetaHeaderValue { get; set; } = null!;
12:
13:     public string ThreadsEndpoint { get; set; } = null!;
14:
15:     public string TextToSqlAssistantId { get; set; } = null!;
16:
17:     public int TextToSqlStatusCheckIntervalInMs { get; set; }
18:
19:     public int TextToSqlTimeoutInSeconds { get; set; }
20:
21:     public string WhisperEndpoint { get; set; } = null!;
22:
23:     public string WhisperModel { get; set; } = null!;
24: }
25:
```

File: MessageRoles.cs

```
1: namespace WebApi.UseCases.OpenAi.Constants;
2:
3: public static class MessageRoles
4: {
5:     public const string User = "user";
6:     public const string Assistant = "assistant";
7: }
8:
```

File: RunStatuses.cs

```
1: namespace WebApi.UseCases.OpenAi.Constants;
2:
3: public static class RunStatuses
4: {
5:     public const string Completed = "completed";
6: }
7:
```

File: MessageListModel.cs

```
1: namespace WebApi.UseCases.OpenAi.Models;
2:
3: public record MessageListModel(IReadOnlyCollection<MessageModel> Data, string FirstId, string LastId, bool HasMore);
4:
```

File: MessageModel.cs
```
1: namespace WebApi.UseCases.OpenAi.Models;
2:
3: public record MessageModel(string Id, string Role, IReadOnlyCollection<MessagePayload> Content);
4:
5: public record MessagePayload(string Type, MessagePayloadText Text);
6:
7: public record MessagePayloadText(string Value);
```

File: RunModel.cs
```
01: using System.Text.Json.Serialization;
02: using WebApi.UseCases.OpenAi.Constants;
03:
04: namespace WebApi.UseCases.OpenAi.Models;
05:
06: public record RunModel(
07:     string Id,
08:     string Object,
09:     int CreatedAt,
10:     string AssistantId,
11:     string ThreadId,
12:     string Status,
13:     int? StartedAt,
14:     int? ExpiresAt,
15:     int? CancelledAt,
16:     int? FailedAt,
17:     int? CompletedAt,
18:     string Model);
```

File: WebApi.csproj
```
01: <Project Sdk="Microsoft.NET.Sdk.Web">
02:
03:     <PropertyGroup>
04:         <TargetFramework>net8.0</TargetFramework>
05:         <Nullable>enable</Nullable>
06:         <ImplicitUsings>enable</ImplicitUsings>
07:         <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
08:     </PropertyGroup>
09:
10:     <ItemGroup>
11:         <PackageReference Include="Swashbuckle.AspNetCore" Version="6.2.3" />
12:         <PackageReference Include="Google.Cloud.BigQuery.V2" Version="3.5.0" />
13:         <PackageReference Include="MediatR" Version="12.2.0" />
14:     </ItemGroup>
15:
16: </Project>
17:
```

File: App.tsx
```
001: import { Button, IconButton, styled, Tab, Tabs, TextField } from "@mui/material";
002: import { useState } from "react";
003: import { Colors } from "./colors";
004: import MicIcon from "@mui/icons-material/Mic";
005: import StopCircleIcon from "@mui/icons-material/StopCircle";
```

```
006: import { config } from "./config";
007: import Recorder from "recorder-js";
008:
009: function App() {
010:   const [state, setState] = useState(States.NaturalLanguageQuery);
011:   const [formalQuery, setFormalQuery] = useState<string>();
012:   const [queryResult, setQueryResult] = useState<string>();
013:
014:   const naturalLanguageQueryViewCallback = (newFormalQuery: string) => {
015:     setState(States.FormalQuery);
016:     setFormalQuery(newFormalQuery);
017:   };
018:
019:   const formalQueryViewCallback = (newQueryResult: string) => {
020:     setState(States.QueryResult);
021:     setQueryResult(newQueryResult);
022:   };
023:
024:   return (
025:     <Main>
026:       <img src="/dog.png" />
027:       <SiteTitle>Natural interface to your Db</SiteTitle>
028:       <Tabs value={state} onChange={(_, newState: States) => setState(newState)}>
029:         <Tab label="Natural language command" />
030:         <Tab label="Formal query" disabled={formalQuery === undefined} />
031:         <Tab label="Query results" disabled={queryResult === undefined} />
032:       </Tabs>
033:       <NaturalLanguageQueryView
034:         isShown={state === States.NaturalLanguageQuery}
035:         callback={naturalLanguageQueryViewCallback}
036:       />
037:       <FormalQueryView
038:         isShown={state === States.FormalQuery}
039:         setQuery={setFormalQuery}
040:         query={formalQuery ?? ""}
041:         callback={formalQueryViewCallback}
042:       />
043:       <QueryResultView isShown={state === States.QueryResult} result={queryResult ?? ""} />
044:     </Main>
045:   );
046: }
047:
048: function NaturalLanguageQueryView({
049:   isShown,
050:   callback,
051: }: Readonly<{
052:   isShown: boolean;
053:   callback: (formalQuery: string) => void;
054: }>) {
055:   const [inputType, setInputType] = useState(NaturalLanguageInputTypes.Text);
056:
057:   return (
058:     <>
059:       {isShown && (
```

```
060:       <>
061:         <StateTabTitle>Provide your command</StateTabTitle>
062:         <Tabs value={inputType} onChange={(_, newType: NaturalLanguageInputTypes) => setInputType(newType)}>
063:           <Tab label="By text" />
064:           <Tab label="By voice" />
065:         </Tabs>
066:       </>
067:     )}
068:
069:     <TextQueryInputView isShown={isShown && inputType === NaturalLanguageInputTypes.Text} callback={callback}
/>
070:                 <AudioQueryInputView isShown={isShown && inputType === NaturalLanguageInputTypes.Voice}
callback={callback} />
071:   </>
072: );
073: }
074:
075: function TextQueryInputView({ isShown, callback }: Readonly<{ isShown: boolean; callback: (sql: string) => void }>) {
076:   const [textQuery, setTextQuery] = useState<string>();
077:
078:   if (!isShown) return null;
079:
080:   const text = textQuery ?? "";
081:   return (
082:     <>
083:       <TextField
084:         multiline
085:         sx={{ width: "600px" }}
086:         label="Your query"
087:         value={text}
088:         onChange={(e) => setTextQuery(e.target.value)}
089:       />
090:       <Button sx={{ marginTop: "10px" }} onClick={() => convertTextToSql(text, callback)} disabled={!textQuery}>
091:         Generate formal query
092:       </Button>
093:     </>
094:   );
095: }
096:
097: function AudioQueryInputView({ isShown, callback }: Readonly<{ isShown: boolean; callback: (sql: string) => void }>) {
098:   const [recorder, setRecorder] = useState<Recorder>();
099:   const [audioBlob, setAudioBlob] = useState<Blob>();
100:   const [textQuery, setTextQuery] = useState<string>();
101:
102:   if (!isShown) return null;
103:
104:   return (
105:     <>
106:       <p>Record your query</p>
107:       {recorder ? (
108:         <IconButton onClick={() => stopRecording()}>
109:           <StopCircleIcon />
110:         </IconButton>
111:       ) : (
```

```
112:       <IconButton onClick={() => startRecording()}>
113:         <MicIcon />
114:       </IconButton>
115:     )}
116:
117:     {audioBlob && <audio controls src={URL.createObjectURL(audioBlob)}></audio>}
118:     <Button onClick={transformAudioToTextQuery}>Transform audio to text query</Button>
119:     {textQuery && (
120:       <TextField
121:         multiline
122:         sx={{ width: "600px" }}
123:         label="Your query"
124:         value={textQuery}
125:         onChange={(e) => setTextQuery(e.target.value)}
126:       />
127:     )}
128:     <Button onClick={() => convertTextToSql(textQuery ?? "", callback)} disabled={!textQuery}>
129:       Generate formal query
130:     </Button>
131:   </>
132: );
133:
134: function startRecording() {
135:   navigator.mediaDevices.getUserMedia({ audio: true }).then((stream) => {
136:     const newRecorder = new Recorder(new AudioContext());
137:     newRecorder.init(stream);
138:     newRecorder.start();
139:     setAudioBlob(undefined);
140:     setTextQuery(undefined);
141:     setRecorder(newRecorder);
142:   });
143: }
144:
145: function stopRecording() {
146:   if (!recorder) return;
147:
148:   recorder.stop().then(({ blob }) => {
149:     setRecorder(undefined);
150:     setAudioBlob(new Blob([blob], { type: "audio/wav" }));
151:   });
152: }
153:
154: function transformAudioToTextQuery() {
155:   if (!audioBlob) return;
156:   setTextQuery(undefined);
157:   const form = new FormData();
158:   form.append("file", audioBlob, "audio.wav");
159:   fetch(config.backendUrl + "/audio-to-text", {
160:     method: "POST",
161:     body: form,
162:   })
163:     .then((response) => response.json() as Promise<IAudioToTextResponse>)
164:     .then((response) => setTextQuery(response.text))
165:     .catch((error) => console.error(error));
```

```
166:   }
167:
168:   interface IAudioToTextResponse {
169:     text: string;
170:   }
171: }
172:
173: function FormalQueryView({
174:   isShown,
175:   query,
176:   setQuery,
177:   callback,
178: }: Readonly<{
179:   isShown: boolean;
180:   query: string;
181:   setQuery: (newFormalQuery: string) => void;
182:   callback: (queryResult: string) => void;
183: }>) {
184:   if (!isShown) return null;
185:
186:   return (
187:     <>
188:       <StateTabTitle>Your query is:</StateTabTitle>
189:       <TextField
190:         multiline
191:         sx={{ width: "600px" }}
192:         label="Formal query"
193:         value={query}
194:         onChange={(e) => setQuery(e.target.value)}
195:       />
196:
197:       <Button sx={{ marginTop: "10px" }} onClick={() => executeFormalQuery()}>
198:         Send
199:       </Button>
200:     </>
201:   );
202:
203:   function executeFormalQuery() {
204:     fetch(config.backendUrl + "/execute-in-big-query", {
205:       method: "POST",
206:       headers: { "Content-Type": "application/json" },
207:       body: JSON.stringify({ sql: query }),
208:     })
209:       .then((response) => response.json() as Promise<IExecuteInBigQueryResponse>)
210:       .then((result) => callback(result.resultsInCsv.length > 0 ? result.resultsInCsv : "No results"))
211:       .catch((error) => console.error(error));
212:   }
213:
214:   interface IExecuteInBigQueryResponse {
215:     resultsInCsv: string;
216:   }
217: }
218:
219: function QueryResultView({ isShown, result }: Readonly<{ isShown: boolean; result: string }>) {
```

```
220:   if (!isShown) return null;
221:
222:   return (
223:     <>
224:       <StateTabTitle>Result:</StateTabTitle>
225:       <TextField sx={{ width: "600px" }} multiline value={result} />
226:     </>
227:   );
228: }
229:
230: function convertTextToSql(text: string, callback: (sql: string) => void) {
231:   fetch(config.backendUrl + "/text-to-sql", {
232:     method: "POST",
233:     headers: { "Content-Type": "application/json" },
234:     body: JSON.stringify({ text: text }),
235:   })
236:     .then((response) => response.json() as Promise<ITextToSqlResponse>)
237:     .then((command) => callback(command.sql))
238:     .catch((error) => console.error(error));
239: }
240:
241: interface ITextToSqlResponse {
242:   sql: string;
243: }
244:
245: enum States {
246:   NaturalLanguageQuery,
247:   FormalQuery,
248:   QueryResult,
249:   Error,
250: }
251:
252: enum NaturalLanguageInputTypes {
253:   Text,
254:   Voice,
255: }
256:
257: const Main = styled("div")({
258:   display: "flex",
259:   flexDirection: "column",
260:   alignItems: "center",
261:   paddingTop: "60px",
262: });
263:
264: const SiteTitle = styled("h3")({
265:   fontSize: "32px",
266:   lineHeight: "40px",
267:   fontWeight: "bold",
268:   color: Colors.Black,
269:   margin: "12px 0 40px",
270: });
271:
272: const StateTabTitle = styled("h4")({
273:   fontSize: "26px",
```

```
274:   lineHeight: "30px",
275:   fontWeight: "bold",
276:   color: Colors.Black,
277:   margin: "40px 0 40px",
278: });
279:
280: export default App;
281:
```

File: colors.ts

```
1: export enum Colors {
2:   White = "#fff",
3:   Black = "#000",
4: }
5:
```

File: config.ts

```
1: export const config = {
2:   backendUrl: process.env.REACT_APP_BACKEND_URL,
3: };
4:
```

File: index.css

```
01: body {
02:   margin: 0;
03:   font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
04:     'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
05:     sans-serif;
06:   -webkit-font-smoothing: antialiased;
07:   -moz-osx-font-smoothing: grayscale;
08: }
09:
10: code {
11:   font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
12:     monospace;
13: }
14:
```

File: index.html

```
01: <!DOCTYPE html>
02: <html lang="en">
03:   <head>
04:     <meta charset="utf-8" />
05:     <link rel="icon" href="%PUBLIC_URL%/dog.png" />
06:     <meta name="viewport" content="width=device-width, initial-scale=1" />
07:     <meta name="theme-color" content="#000000" />
08:     <meta
09:       name="description"
10:       content="NaturalDB - natural language interface for databases"
11:     />
12:     <link rel="apple-touch-icon" href="%PUBLIC_URL%/dog.png" />
13:     <!--
14:       manifest.json provides metadata used when your web app is installed on a
15:       user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
```

```
16:    -->
17:    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18:    <!--
19:      Notice the use of %PUBLIC_URL% in the tags above.
20:      It will be replaced with the URL of the `public` folder during the build.
21:      Only files inside the `public` folder can be referenced from the HTML.
22:
23:      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
24:      work correctly both with client-side routing and a non-root public URL.
25:      Learn how to configure a non-root public URL by running `npm run build`.
26:    -->
27:    <title>NaturalDb</title>
28:  </head>
29:  <body>
30:    <noscript>You need to enable JavaScript to run this app.</noscript>
31:    <div id="root"></div>
32:    <!--
33:      This HTML file is a template.
34:      If you open it directly in the browser, you will see an empty page.
35:
36:      You can add webfonts, meta tags, or analytics to this file.
37:      The build step will place the bundled scripts into the <body> tag.
38:
39:      To begin the development, run `npm start` or `yarn start`.
40:      To create a production bundle, use `npm run build` or `yarn build`.
41:    -->
42:  </body>
43: </html>
44:
```

File: index.tsx

```
01: import React from "react";
02: import ReactDOM from "react-dom/client";
03: import "./index.css";
04: import App from "./App";
05: import reportWebVitals from "./reportWebVitals";
06: import { ThemeProvider } from "@mui/material";
07: import { theme } from "./theme";
08:
09: const root = ReactDOM.createRoot(document.getElementById("root") as HTMLElement);
10: root.render(
11:   <React.StrictMode>
12:     <ThemeProvider theme={theme}>
13:       <App />
14:     </ThemeProvider>
15:   </React.StrictMode>
16: );
17:
18: // If you want to start measuring performance in your app, pass a function
19: // to log results (for example: reportWebVitals(console.log))
20: // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
21: reportWebVitals();
22:
```

File: theme.ts

```
01: import { createTheme } from "@mui/material";
02:
03: export const theme = createTheme({
04:   typography: {
05:     fontFamily: [
06:       "-apple-system",
07:       "BlinkMacSystemFont",
08:       "Segoe UI",
09:       "Roboto",
10:       "Oxygen",
11:       "Ubuntu",
12:       "Cantarell",
13:       "Fira Sans",
14:       "Droid Sans",
15:       "Helvetica Neue",
16:       "sans-serif",
17:     ].join(","),
18:   },
19: });
20:
```

File: package.json

```
01: {
02:   "name": "natural-db",
03:   "version": "0.1.0",
04:   "private": true,
05:   "dependencies": {
06:     "@emotion/react": "^11.10.5",
07:     "@emotion/styled": "^11.10.5",
08:     "@mui/icons-material": "^5.10.9",
09:     "@mui/material": "^5.10.12",
10:     "react": "^18.2.0",
11:     "react-dom": "^18.2.0",
12:     "react-scripts": "5.0.1",
13:     "recorder-js": "^1.0.7",
14:     "web-vitals": "^2.1.4"
15:   },
16:   "scripts": {
17:     "start": "react-scripts start",
18:     "build": "react-scripts build",
19:     "test": "react-scripts test",
20:     "eject": "react-scripts eject",
21:     "lint": "tsc --noEmit && eslint --ext .js,.jsx,.ts,.tsx src/",
22:     "lint-fix": "eslint --ext .js,.jsx,.ts,.tsx --fix src/"
23:   },
24:   "browserslist": {
25:     "production": [
26:       ">0.2%",
27:       "not dead",
28:       "not op_mini all"
29:     ],
30:     "development": [
31:       "last 1 chrome version",
```

```
32:     "last 1 firefox version",
33:     "last 1 safari version"
34:   ]
35: },
36: "devDependencies": {
37:   "@testing-library/jest-dom": "^5.16.5",
38:   "@testing-library/react": "^13.4.0",
39:   "@testing-library/user-event": "^13.5.0",
40:   "@types/jest": "^27.5.2",
41:   "@types/node": "^16.18.3",
42:   "@types/react": "^18.0.24",
43:   "@types/react-dom": "^18.0.8",
44:   "@types/recorder-js": "^1.0.2",
45:   "@typescript-eslint/eslint-plugin": "^5.42.0",
46:   "@typescript-eslint/parser": "^5.42.0",
47:   "eslint": "^8.26.0",
48:   "eslint-config-airbnb-typescript": "^17.0.0",
49:   "eslint-config-prettier": "^8.5.0",
50:   "eslint-config-standard-with-typescript": "^23.0.0",
51:   "eslint-plugin-import": "^2.26.0",
52:   "eslint-plugin-jest": "^27.1.3",
53:   "eslint-plugin-n": "^15.4.0",
54:   "eslint-plugin-prettier": "^4.2.1",
55:   "eslint-plugin-promise": "^6.1.1",
56:   "eslint-plugin-react": "^7.31.10",
57:   "prettier": "^2.7.1",
58:   "typescript": "^4.8.4"
59: }
60: }
61:
```

ДОДАТОК Д

Результати перевірки роботи на співпадіння