NATURAL LANGUAGE QUERYING IN RELATIONAL AND SCHEMA-LESS DATABASES

A MAJOR PROJECT REPORT

by

ANTONY DOMINIC S (VJC21AD013)
JEEVAN SEN JOSEPH (VJC21AD035)
JOHANN ABY VANNILAM (VJC21AD038)
SANAL JOSE (VJC21AD052)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

VISWAJYOTHI COLLEGE OF ENGINEERING AND TECHNOLOGY, VAZHAKULAM

APRIL 2025

NATURAL LANGUAGE QUERYING IN RELATIONAL AND SCHEMA-LESS DATABASES

MAJOR PROJECT REPORT

by

ANTONY DOMINIC S (VJC21AD013)
JEEVAN SEN JOSEPH (VJC21AD035)
JOHANN ABY VANNILAM (VJC21AD038)
SANAL JOSE (VJC21AD052)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Under the guidance of

MRS. FEMY JOHN

ASSISTANT PROFESSOR

DEPT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE, VJCET

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

VISWAJYOTHI COLLEGE OF ENGINEERING AND TECHNOLOGY, VAZHAKULAM

APRIL 2025

VISWAJYOTHI COLLEGE OF ENGINEERING AND TECHNOLOGY, VAZHAKULAM

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that the major project report entitled '**NATURAL LANGUAGE QUERYING IN RELATIONAL AND SCHEMA-LESS DATABASES**' is a bonafide record of the project presented by **ANTONY DOMINIC S (VJC21AD013), JEEVAN SEN JOSEPH (VJC21AD035), JOHANN ABY VANNILAM (VJC21AD038), SANAL JOSE (VJC21AD052),** in partial fulfillment of the requirements for the award of the **Degree of Bachelor of Technology** in **Artificial Intelligence and Data Science** of APJ Abdul Kalam Technological University.

Internal Supervisor External Supervisor

Project Coordinator Head of the Department

VISWAJYOTHI COLLEGE OF ENGINEERING AND TECHNOLOGY, VAZHAKULAM

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Vision

Moulding professionals catering to research, innovation, and entrepreneurial developments for global digitalization.

Mission

1. To inculcate research culture for design and innovative development towards a better world.
2. To guide and mould the students to become globally competent professionals with ethical values.
3. To generate innovative ideas, and disseminate it through quality publications.

Program Educational Objectives

Our Graduates

1. Shall apply their skill-based knowledge attained during undergraduate education to adapt with the recent technological advancements and acquire a promising career.
2. Shall possess critical thinking, professional skills and strong work ethics, to solve real world problems required to cater the needs of industry and to serve the society.
3. Shall learn, apply and adapt to grow with the industrial needs thereby becoming a successful innovator in the challenging technological world.
4. Shall have the competency to seek higher professional degrees, certifications and to do quality research work as an individual or as a team.

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

1. Able to apply the computational knowledge of data science, machine learning and deep learning to analyse, design and model novel applications.
2. Ability to become an entrepreneur which provides engineering solutions for industrial and societal problems.
3. Up-skilling in advanced data science, machine learning and deep learning technologies.

ACKNOWLEDGEMENT

First and foremost, we thank God Almighty for his divine grace and blessings all throughout the project, without which the project would not have reached its completion successfully. It is our privilege to render our sincere gratitude to **Msgr. Dr. Pius Malekandathil**, our most beloved Manager, **Rev. Dr. Paul Parathazham**, our Director, and our Principal, **Dr. K.K. Rajan**, for providing us the opportunity to do this major project during the final year (2024-25) of our B. Tech degree course. We are deeply thankful to our Head of the Department, **Dr. Anita Brigit Mathew** for her support and encouragement. We would like to express our sincere gratitude and heartfelt thanks to our Major Project Guide and Major Project co-ordinator, **Mrs. Femy John**, Assistant Professor, Department of Artificial Intelligence and Data Science, for her motivation, assistance and help for the project. We express our deepest and sincere gratitude to **Mr. Nithin A.R.**, Senior Software Engineer, IBS Software, for his expert guidance, dedicated efforts, persistent motivation and constant support, all throughout the project. We also thank all the faculty members of the Department, including lab faculty for their support and encouragement. Finally, we also thank all our friends and family for their valuable feedback from time to time as well as their help and motivation.

DECLARATION

We undersigned hereby declare that the project report **NATURAL LANGUAGE QUERYING IN RELATIONAL AND SCHEMA-LESS DATABASES**, submitted for partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology of the APJ Abdul Kalam Technological University is a bonafide work done by us under the supervision of **Mrs. Femy John**, Assistant Professor, Department of Artificial Intelligence and Data Science. This submission represents ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or formed the basis for the award of any degree, diploma, or similar title of any other University.

Place: Vazhakulam

Date:

Antony Dominic S

Jeevan Sen Joseph

Johann Aby Vannilam

Sanal Jose

ABSTRACT

This project introduces a plug-and-play framework that enables developers to seamlessly convert natural language into database queries, whether for structured (SQL) or unstructured (NoSQL) data. Using NLP and Large Language Models (LLMs), the system intelligently generates precise SQL queries for relational databases and dynamic NoSQL queries for schema-less databases like MongoDB. By leveraging advanced prompting techniques such as chain of thought and prompt chaining, it ensures accuracy and adaptability across different database types. The framework simplifies database interactions by providing a unified, intuitive interface, reducing complexity for developers working with diverse data architectures. Initially focused on the travel industry, particularly airport marketplaces, the solution is designed for easy integration into various applications and industries. With its ability to streamline database operations and enhance developer productivity, this framework represents a significant step toward making natural language querying a standard feature in modern data-driven applications.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SQL	Structured Query Language
NoSQL	Not Only SQL
NLP	Natural language Processing
LLM	Large Language Models
AI	Artificial Intelligence
RDBMS	Relational Database Management System
GPT	Generative Pre-trained Transformers
API	Application Programming Interface
UI/UX	User Interface/User Experience
HTTP	Hyper Text Transfer Protocol
URL	Uniform Resource Locator
SPA	Single Page Application
REST	Representational State Transfer Application
DRF	Django REST Framework
JSON	JavaScript Object Notation
CRUD	Create Read Update Delete

CHAPTER 1

INTRODUCTION

In today’s data-centric world, facilitating seamless and intuitive interaction with databases is critical. The complexity and technical nature of traditional query languages like SQL can create barriers, limiting database accessibility to those with specialized skills. For non-technical users, especially in environments that demand fast access to complex information, this can hinder productivity and data-driven decision-making. However, recent advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) offer a revolutionary solution: natural language querying. This approach enables users to interact with databases using everyday language, breaking down technical barriers and democratizing access to valuable data insights.

This project, titled "Natural Language Querying in Relational and Schema-Less Databases," aims to develop a versatile framework that allows users to "talk to their data." The framework will accept natural language input from users, process it through advanced NLP techniques, and convert it into structured queries for relational databases like PostgreSQL or flexible expressions for schema-less systems such as MongoDB. By bridging natural language and database query languages, this framework empowers users to retrieve data seamlessly, without needing technical expertise in SQL or familiarity with specific database schemas.

At the heart of this solution is an intelligent system that combines NLP, LLMs, and advanced prompting techniques. These components work together to interpret user intent accurately, ensuring that even complex or multi-part questions are understood and processed correctly. The framework’s flexible design allows it to support structured relational databases alongside schema-less environments, making it applicable across a broad spectrum of data storage needs. Furthermore, the system has been designed as a plug-and-play solution, allowing developers to integrate natural language querying capabilities into their applications with ease.

While the project’s initial application focuses on the travel industry—specifically airport marketplaces—where the need for instant, accessible data is essential, its adaptable architecture makes it suitable for various industries. By offering an intuitive way to interact with data, this

framework can enhance productivity, streamline operations, and support decision-making across sectors.

Thus, this project represents an innovative approach to database querying by transforming natural language into actionable queries for both structured and unstructured data systems. It advances the vision of "talking to your data," enabling users to access, analyze and interact with information easily and intuitively, regardless of their technical background.

1.1 MOTIVATION

The growing need for user-friendly interaction with database systems is driving the development of solutions that allow both technical and non-technical users to query data using natural language. Complex databases often require expertise in SQL or NoSQL, making data retrieval challenging for many users. A framework that translates natural language inputs into accurate database commands would enable users across various skill levels to access information effortlessly. This simplification not only enhances accessibility but also aids in better decision-making by allowing users to interact with data more intuitively. Additionally, training agents to process natural language queries directly, instead of traditional database queries, can reduce operational costs and improve efficiency, making such a framework a valuable asset for businesses.

1.2 OBJECTIVE

The objective of this project is to develop a versatile, AI-powered framework that enables users to interact with relational and schema-less databases through natural language. By translating conversational queries into accurate database commands, the framework aims to make data access intuitive, bridging the gap between technical and non-technical users. This solution seeks to improve data accessibility, enhance productivity, and democratize data-driven decision-making across various industries and application with minimal configuration.

1.3 SCOPE

The scope of this project includes designing and implementing a framework that supports natural language querying across both relational and schema-less databases. It will leverage NLP and LLMs to process and interpret user queries, generating precise SQL and NoSQL commands for data retrieval. Initially focused on use cases within the airline marketplace industry, the framework is adaptable to various sectors, providing a versatile, plug-and-play solution to simplify data interactions for users of all technical levels.

CHAPTER 2

LITERATURE SURVEY

SL NO	TITLE	METHODOLOGY	ADVANTAGES	DISADVANTAGES
1.	**Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning**	Seq2SQL converts NLP questions into SQL queries using a reinforcement learning-based approach. Model is divided into modules that select SQL operations, columns, and conditions.	High accuracy for complex queries Modular approach simplifies query structure	Requires large training data Limited performance on highly complex database schemas
2.	**SQLNet: Generating Structured Queries from Natural Language Without Reinforcement Learning**	SQLNet uses sequence-to-sequence learning without reinforcement learning. It introduces a sketch-based approach to predict SQL queries, focusing on the structure.	Avoids complexities of reinforcement learning Faster training High accuracy on simple queries	Struggles with very complex or nested queries Less adaptable to dynamic schema
3.	**TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation**	TypeSQL enhances SQL generation by incorporating a type system and external knowledge (table schemas) to handle	Type-aware approach reduces errors Effective in handling database-specific keywords and	May struggle with highly ambiguous queries Requires additional external knowledge

		ambiguities in natural language queries. This approach improves the handling of database-specific attributes.	column types	(e.g., schemas)
4.	**Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation**	This paper introduces an Intermediate Representation (IR) approach, breaking down complex natural language queries for cross-domain databases, simplifying the query generation process.	Better generalization across domains Handles complex queries with multiple conditions	May struggle with very specific domain-specific queries More training steps required
5.	**UnifiedSKG: Unifying Structured Knowledge Grounding with Text-to-SQL Parsing**	UnifiedSKG is a framework that unifies multiple structured knowledge grounding tasks (e.g., text-to-SQL parsing), leveraging data from various sources and representations.	Unified approach across various tasks Effective for cross-domain applications	High resource requirements Complex to implement due to multi-task learning
6.	**Context-based Ontology Modelling for Database: Enabling ChatGPT for Semantic Database Management**	Context-based Ontology Modelling (COM-DB) to convert database schemas into natural language for easier interaction with ChatGPT.	Reduces required domain knowledge Speedy database management process Enhances privacy protection	Focuses on basic operations, lacks complex functions Relies on semantic schema for ChatGPT to understand

7.	**C3: Zero-shot Text-to-SQL with ChatGPT**	C3 uses clear prompting, calibration with hints, and consistency outputs to enable zero-shot Text-to-SQL querying using ChatGPT.	Achieves high accuracy (82.3%) in SQL query generation Does not require fine-tuning Efficient with fewer tokens	Struggles with complex queries Requires consistent refinement and calibration
8.	**A Feasibility Study on Automated SQL Exercise Generation with ChatGPT-3.5**	Investigates ChatGPT-3.5 for generating SQL exercises using prompt engineering.	Saves time for educators Can generate varying exercises based on different database schemas and mock data	May generate too easy or overly difficult questions Inconsistent quality and feasibility for complex questions
9.	**Natural Language Data Interfaces: From Keyword Search to ChatGPT, are we there yet?**	Reviews the evolution of natural language interfaces for databases and discusses the future of text-to-SQL systems.	Highlights research directions Explores the gaps in current systems for conversational data querying	Current systems are not widely adopted Most approaches fail with complex, real-world databases
10.	**Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language**	Proposes a framework for ChatGPT to handle geospatial SQL queries using a combination of training data and	Effective in geospatial data querying Demonstrates success in tasks like	Prone to hallucination errors Struggles with complex or unseen datasets and tasks

	into Structured Query Language within a Spatial Database	schema input.	spatial joins	
11.	**An Interpretation of Long Short-Term Memory Recurrent Neural Network for Approximating Roots of Polynomials**	Uses LSTM-RNN model combined with ADAM optimizer for gradient descent, approximating roots of polynomials by connecting polynomial coefficients to weights.	Highly efficient for handling sequences Computational efficiency Low memory needs Adaptive learning rates based on gradient history.	Struggles with complex polynomial relationships if not enough training data is provided. Risk of Overfitting in small datasets.
12.	**Robust Natural Language Processing: Recent Advances, Challenges, and Future Directions**	Systematic overview of NLP robustness research, focusing on adversarial attacks and defences	Provides a comprehensive taxonomy of NLP robustness Identifies gaps in current research Offers insights into various techniques and metrics used in NLP robustness	Limited by the existing literature's focus on specific aspects rather than a unified framework May not cover all recent advancements in NLP robustness comprehensively
13.	**Building a Natural Language Query and Control Interface for IoT Platforms**	Utilizes hierarchical semantic parsing algorithms and directed edge-tagged graph structures for command parsing.	Handles complex commands with multiple operations. Integrates with popular IoT platforms like AliCloud. Enhances user interaction through natural language processing.	Current methods may struggle with very long commands. Requires robust training data for optimal performance. Limited to specific IoT platforms in initial implementation.

14.	**Exploring Natural Language Processing in Model-to-Model Transformations**	Combines NLP techniques such as CoreNLP, Stanza, BERT, and Flair for extracting verb-noun relations from model elements in BPMN and UML models using M2M transformation processes.	NLP aids in detecting complex relations between model elements Enhances model-to-model transformations. Custom BERT models improve verb extraction Facilitates automatic generation of transformed models.	Ambiguity in text labels can affect accuracy. Lack of large, specialized datasets. Handling acronyms and abbreviations contextually in model transformations.
15.	**Natural Language Processing-Based Software Testing: A Systematic Literature Review**	Reviews 24 studies focusing on the integration of NLP in software testing, including test case generation, requirements analysis, and machine learning methods in the testing process.	Offers valuable insights into NLP's potential to automate and improve test generation Highlights state-of-the-art tools Showcases NLP's role in enhancing the analysis of requirements and software quality assurance.	Ambiguity in natural language requirements poses challenges. Limited domain adaptation of NLP models across different software testing scenarios. Scalability of frameworks for large systems is a recurrent issue.
16.	**An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering**	This research utilized ChatGPT to develop a web-based code generation platform with three key components: User Interface, Prompt Builder, and Backend	65.06% improvement in exact match (EM), 38.45% in BLEU, 15.70% in CodeBLEU, and 50.64% in Pass@1. 98.5% of test cases	Inherent shortcomings of ChatGPT, such as accuracy and timeliness of information. Tendency toward verbose outputs and

		Service. The Prompt Builder dynamically generated prompts to enhance model generation performance.	validated via manual validation, showing significant assistance in code generation.	potential misguidance.
17.	**Prompt-Based Label-Aware Framework for Few-Shot Multi-Label Text Classification**	Proposes a Prompt-based Label-Aware framework (PLAML) with three techniques: (i) token weighting algorithm, (ii) template for augmenting training samples, and (iii) dynamic threshold mechanism to improve multi-label text classification performance.	Outperforms baseline methods on few-shot text classification. Effectiveness of each technique was analyzed for the multi-label setting.	Exploration of tuneable continuous templates or multiple discrete templates could reduce input ambiguity. Experiments conducted only with the base version of PLMs; larger PLMs could improve performance.
18.	**Generation of Asset Administration Shell With Large Language Model Agents: Toward Semantic Interoperability in Digital Twins in the Context of Industry 4.0**	Introduces a novel approach for semantic interoperability in digital twins, assisting in creating Asset Administration Shell (AAS). The approach uses a 'semantic node' data structure to capture the semantic essence of textual data, allowing automatic translation of	Introduces a novel approach for semantic interoperability in digital twins, assisting in creating Asset Administration Shell (AAS). The approach uses a 'semantic node' data structure to capture the semantic essence of textual data, allowing automatic translation of raw textual data into AAS	Limited to technical data sub-models according to AAS specifications. Challenge in serializing diverse knowledge representations into a textual format suitable for LLMs.

		raw textual data into AAS models via large language models.	models via large language models.	
19.	**Prompting Large Language Models with Knowledge-Injection for Knowledge-Based Visual Question Answering**	Proposes a knowledge-based VQA framework called 'Prompting Large Language Models with Knowledge-Injection' (PLLMKI), using in-context learning, knowledge enhancement, and prediction with two LLMs for more robust inferential capability.	More robust knowledge representation than knowledge graphs. Cost-effective as it uses open LLMs. Improved accuracy on two knowledge-based VQA datasets.	Does not explore tunable continuous templates or multiple discrete templates. Experiments conducted only with the base version of PLMs; larger PLMs could yield better results.
20.	**Visual Prompt Engineering for Enhancing Facial Recognition Systems Robustness against Evasion Attacks**	Introduces an attack-defense framework using Visual Prompt Engineering (VPE) to enhance FRS robustness against evasion attacks. The attack uses StyleGAN2-generated synthetic images, while VPE detects them via deep feature extraction and classification.	Achieves high accuracy: 97.92% in average-match scenarios, 87.08% in best-match, and 91.96% on the Trueface dataset. Effectively tackles GAN attacks and enhances system robustness.	GAN-based attacks successfully deceive FRS, showing their vulnerability against both real-looking and partially constructed synthetic images. Human observers struggle to detect these evasion attacks.

TABLE 2.1 Literature Survey

CHAPTER 3

PROPOSED WORK

3.1 PROCESS OVERVIEW

The proposed framework introduces a streamlined approach to accessing both relational and schema-less databases through natural language, making data interaction accessible to a broad range of users, regardless of technical expertise. The process initiates with a user registration and login through the user interface. This interface functions as the entry point, allowing users to communicate their data needs in conversational language, such as asking questions about datasets or requesting specific information, without needing to understand query syntax or database structures. The user is given the option to select the type of database to work with. The user then enters a natural language input query in the user interface.

Once the query is submitted, it flows into an NLP module, where sophisticated query analysis and entity recognition are performed. This module plays a critical role in understanding the structure, context, and intent of the user's input. By identifying key entities, conditions, and relationships within the query, the NLP module ensures that user intentions are accurately captured. This phase is essential, as it translates vague or ambiguous natural language into a structured format that can be effectively processed in subsequent steps.

Following query analysis, the refined query information is passed to a language model, such as the GPT API, which is trained in natural language understanding and generation. Here, the language model interprets the processed input and generates precise SQL queries for relational databases or equivalent NoSQL commands for schema-less databases. Advanced prompting techniques, including chain of thought and prompt chaining, are employed to manage complex queries, breaking them down into manageable parts and ensuring accurate interpretation. This step transforms the user's question into a format that is compatible with different types of databases, ensuring the query will return relevant results.

The resulting query commands are then processed by the Query Executor, which organizes and optimizes them before directing them to the appropriate database. The framework supports both structured databases like PostgreSQL and unstructured systems like MongoDB, enabling flexibility in handling different data storage architectures. The Query Executor dynamically

adapts the query based on the database type, ensuring that each command is formatted correctly for efficient execution. This modular design makes the framework highly adaptable, allowing it to cater to various data environments and use cases.

Upon executing the query, the framework retrieves results from the database and processes them in the response handling phase. Here, data formatting and presentation adjustments are made to ensure the output is clear, structured, and easily understandable. This processing step adds an additional layer of user-friendliness by presenting data in an accessible manner, allowing users to quickly interpret and act on the information retrieved.

Finally, the processed data is displayed back to the user through the interface, completing the cycle of interaction. By transforming natural language inputs into structured database queries and delivering clear, actionable responses, this framework offers a robust solution for "talk to your data" functionality. It not only democratizes database access but also promotes efficiency and productivity, making it valuable for various applications across industries.

3.2 FLOW DIAGRAM

The following flow diagram illustrates the end-to-end process of the TalkToData framework, which enables users to interact with databases using natural language.

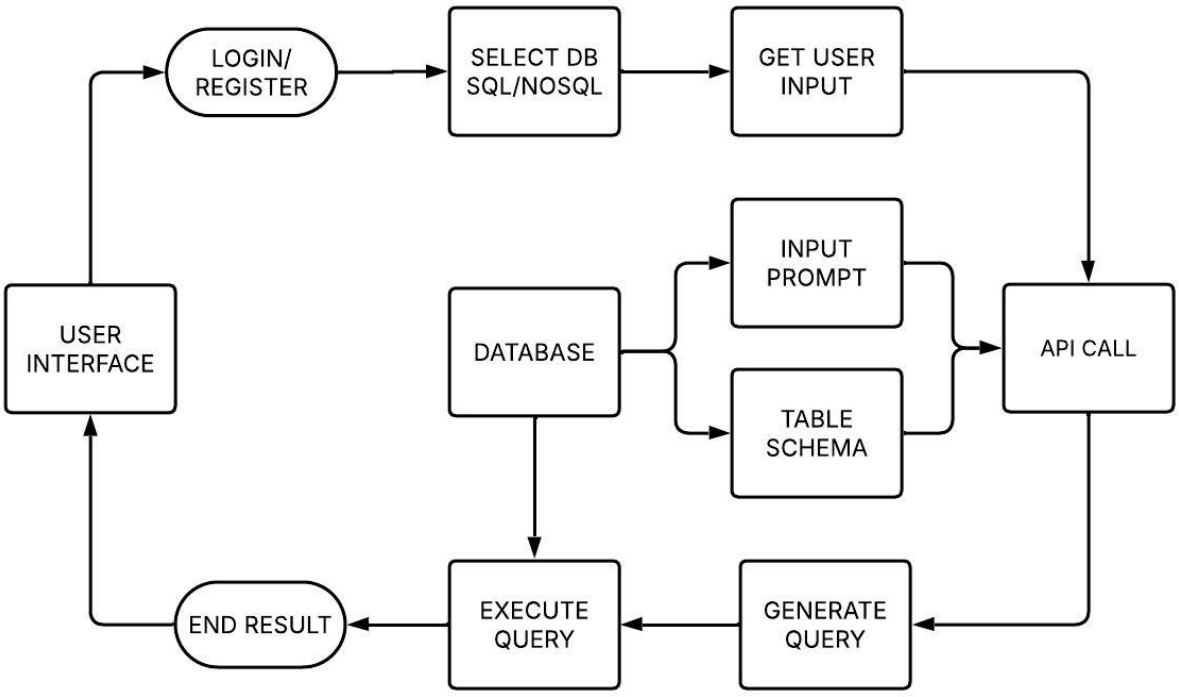

Fig 3.1 Flow diagram

The flow diagram outlines the sequential processes within the framework, showcasing each phase from initial user login to final data presentation. This system is structured to enable seamless, natural language interaction with databases, bridging the gap between conversational language and complex database querying across relational and schema-less systems.

3.2.1 Login/Register

The Login/Register module serves as the entry point of the system, enabling users to either create a new account or access an existing one. The flow begins when the user selects either the Login or Register option.

In the Registration Flow, the user provides essential details such as a username, email, and password. The system validates the input to ensure correctness and checks for duplicate accounts to prevent redundancy. If the validation is successful, the credentials are securely stored in the database. Upon successful registration, the user is redirected to the home page, confirming the completion of the process.

In the Login Flow, the user enters their credentials, including an email or username along with a password. The system then verifies these credentials against the stored records in the database. If authentication is successful, the user is granted access to the system; otherwise, an error message is displayed, prompting them to re-enter the correct details.

3.2.2 Select DB

The Database Selection module allows users to choose between SQL and NoSQL databases, enabling flexible data management based on their requirements. This module is a crucial step in the workflow, as it determines the subsequent query generation and execution process. When the user accesses this module, they are presented with two options: SQL and NoSQL.

If the SQL option is selected, the system establishes a connection with PostgreSQL, a relational database that handles structured data using SQL queries. The user is then directed to the next stage, where they can input natural language queries to generate and execute SQL statements on PostgreSQL. Alternatively, if the NoSQL option is chosen, the system connects to MongoDB, a document-based database designed for unstructured and semi-structured data. This allows the user to execute queries suited for NoSQL storage, such as retrieving or modifying JSON-like documents. Once a database type is selected, the backend dynamically configures the connection to the corresponding database engine, ensuring seamless query generation and execution.

3.2.3 Get User Input

Here the natural language input from the user is collected, processed and prepared for further operations. When a user enters input, the system first captures the raw data. The input then undergoes a cleaning process, which includes removing unwanted whitespace, handling special characters and converting it to a standardized format. Once cleaned, the refined user input is returned in a structured format, making it easier for the subsequent phase to process.

3.2.4 API Call

The API Call module is responsible for interfacing with the GPT model to generate database queries based on user input. This module processes the refined user input along with the dynamically fetched table schema and input prompt to construct an appropriate request for the API. Once the system receives the cleaned user input, it retrieves the relevant table schema and a predefined input prompt from the database. These elements help the API interpret the user's intent accurately. The module then formulates a structured prompt, combining the refined user query, schema details, and necessary contextual information to guide GPT in generating the correct SQL or NoSQL query. After sending the request to the GPT API, the response is processed to extract the generated query. The resulting query is then formatted to ensure it aligns with the expected database structure. The final output is a well-formed SQL query for PostgreSQL or a NoSQL query for MongoDB, which can then be executed in the next phase.

3.2.5 Execute Query

The Execute Query module is responsible for processing the generated query, executing it on the selected database, and retrieving the results for display. Once the query is generated by the API Call module, it is passed to this module for execution. The system establishes connection with the database. The query is then executed, and the system fetches the results from the database. After retrieving the results, they are processed and formatted for better readability before being displayed to the user.

The system provides a seamless workflow, starting from capturing and cleaning user input to generating and executing database queries. By dynamically fetching table schemas and input prompts, it ensures accurate query generation using a GPT-powered API. The integration of PostgreSQL for SQL queries and MongoDB for NoSQL queries allows flexibility in handling both structured and unstructured data. With proper result formatting and error handling mechanisms, the system provides an enhanced overall user experience in database interaction.

CHAPTER 4

IMPLEMENTATION

The system is implemented integrating multiple technologies to create a seamless, efficient, and scalable database querying framework. It follows a modular architecture comprising a React-based frontend, a Django REST framework backend, and a dual-database system using PostgreSQL and MongoDB.

4.1 ARCHITECTURE

The following figure shows the overall architecture of the system implementation.

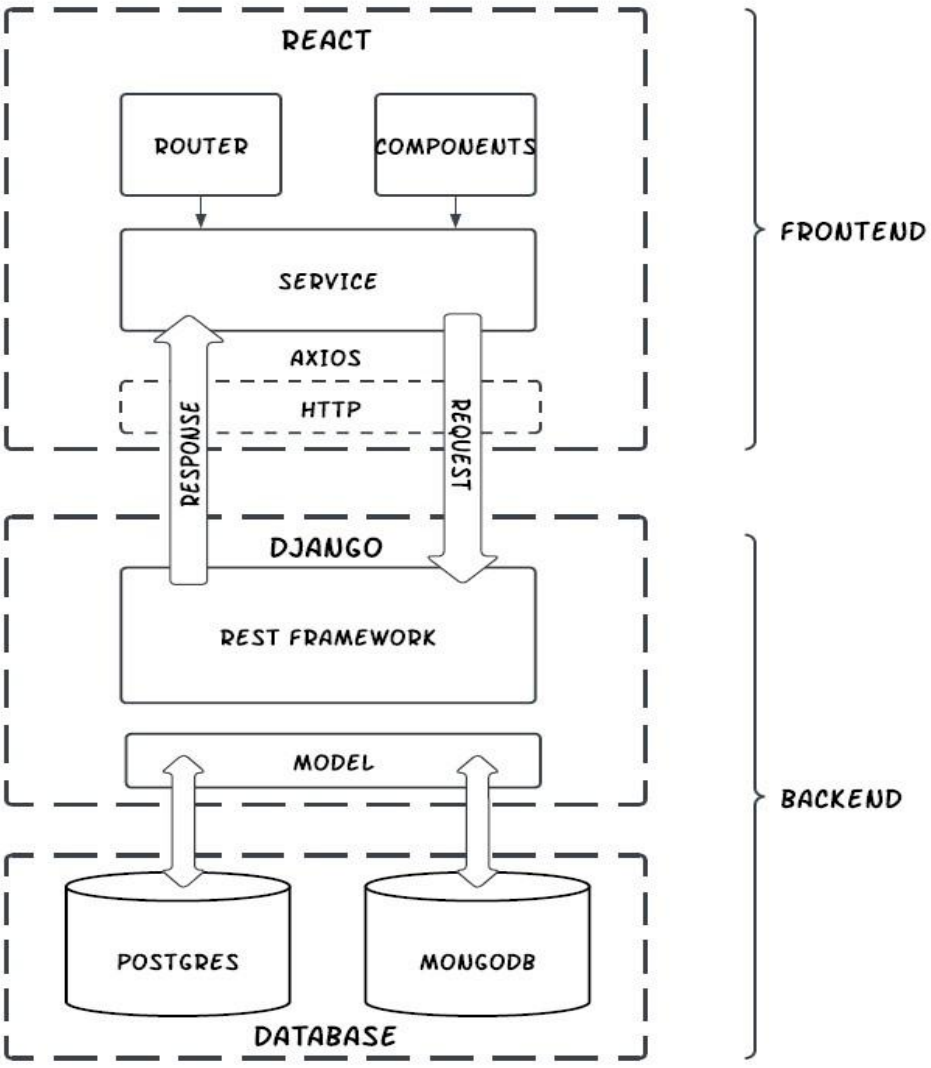

Fig. 4.1 System Architecture

The system implementation adopts a systematic architecture that combines a React frontend, a Django REST framework backend, and a two-database system based on PostgreSQL and MongoDB. The architecture is designed to provide effective communication between the user interface and the database, allowing for smooth query processing and data retrieval.

The frontend, created in React, comprises components and a routing system for user interface handling. The frontend sends HTTP requests and returns responses using Axios to communicate with the backend. The Django REST framework within the backend processes such requests, filters data, and deals with the correct database model as per the selection made by users. The backend dynamically uses PostgreSQL for SQL-type queries and MongoDB for NoSQL-type queries for managing flexibility.

This module-based implementation allows effective query generation, execution, and retrieval of results, with the provision of an easy-to-use and scalable database querying interface. The sections below present a detailed explanation of each of these components and how it contributes to the entire process.

4.2 FRONT END

This is the portion of the system that provides ample experience to the users. The components and the functionalities are described below.

4.2.1 REACT

React is a JavaScript library for constructing user interfaces, specifically for single-page applications (SPAs). It was created by Facebook. With React, developers can write fast and interactive web apps with a component-based approach. Components are reusable, independent UI elements used to construct React apps. Components are either functional (using hooks) or class-based. They keep track of data via state and get external inputs via props. React makes use of a Virtual DOM (VDOM) to update the UI in a performance-efficient manner by re-rendering the changed elements only rather than the whole page, thus enhancing performance. React maintains unidirectional data flow to make the state predictable.

For dealing with complex state, developers take advantage of libraries such as React Context API or Redux. React also offers hooks (useState, useEffect, etc.) to handle component logic without class components. Navigation within React applications is managed with React Router,

which provides smooth transitions between pages without page reloads. React integrates effortlessly with backend APIs (REST or GraphQL) to retrieve and update data. Owing to its efficiency, reusability, and strong community backing, React is used extensively in contemporary web development for developing scalable and high-performance applications.

4.2.1.1 Router

React Router is a popular routing library that facilitates navigation and page control in React apps. It supports client-side routing, where users can change views or pages without causing a full-page reload. This speeds up performance and delivers a more intuitive user experience, just like with classic multi-page apps.

Router takes a declarative design, with routes represented by special components. The BrowserRouter component encloses the application and provides proper URL management. Within it, the Routes and Route components specify the paths and the respective components to be displayed. Page navigation is supported through Link and NavLink, which avoid unnecessary reloads. Moreover, the useNavigate hook enables programmatic navigation. Router has enhanced capabilities like dynamic routing, through which parameters are passed in the URL, nested routes, used to organize complicated layouts, and protected routes, which limit access depending on user authentication. By handling navigation in an efficient manner, React Router assists developers in creating scalable, interactive, and user-friendly SPAs. It ensures that smooth transitions among views occur, enhancing overall usability and performance in React-based web applications.

4.2.1.2 Components

Components are the constructs of a React application. Components are reusable, modular, independent UI elements which assist in efficiently structuring and managing complex UIs. Any component can be considered as a self-contained module, and they are used for rendering a UI part, such as buttons, forms, tables, or parts of pages.

React has support for two primary types of components: Functional Components – Basic JavaScript functions that return JSX and can utilize React hooks (useState, useEffect) to handle state and side effects. Class Components – Legacy React components that inherit from React.Component and handle state using this.state. React supports the decomposition of the UI into small, reusable components, which encourages code reusability, maintainability, and effectiveness.

In React, UI (User Interface) components determine how components look on the screen. These encompass buttons, input, dropdowns, and navigation bars. They concern the appearance of the application. UX (User Experience) components, however, concern interaction and usability. These encompass loading spinners, modals, tooltips, and accessibility options that drive user engagement.

4.2.1.3 Service

A Service in a React application is designed to manage communication with the backend through HTTP requests and response processing. It provides an interface between the frontend and backend to facilitate smooth data exchange. Services are often implemented as individual modules so that API-related logic remains organized and reusable in various components.

React apps tend to utilize Fetch API or Axios to make requests to backend servers. These requests may be GET, POST, PUT, DELETE, and other HTTP request types to fetch, send, update, or delete data from a database or API. The service layer helps ensure all network requests are handled effectively and may involve error handling, authentication headers, and response transformations prior to passing data to components.

By localizing API calls in a service layer, React applications abide by the separation of concerns principle, making the codebase cleaner and easier to maintain. Components do not call the backend directly; rather, they depend on the service to retrieve or update data. In addition, services in React applications tend to be combined with state management libraries such as Redux, React Query, or Context API to handle and cache API responses, enhancing performance and minimizing unnecessary network requests. This improves the overall scalability and efficiency of the application.

4.3 BACKEND

The backend refers to the server-side of a web application, software, or system, responsible for managing and processing data, logic, and functionality. It is the behind-the-scenes infrastructure that powers the frontend and enables it to interact with the data and services.

4.3.1 DJANGO

Django is a high-level Python web framework that follows the Model-View-Template (MVT) architectural pattern. It aims to simplify web development by providing built-in functionality

such as authentication, ORM (Object-Relational Mapping), and an admin interface. Django is used extensively for developing scalable and secure applications.

The process usually begins in Django by defining models in models.py, which are essentially database tables. Django's ORM enables the use of Python to work with the database, rather than raw SQL. Database selection is project-dependent—PostgreSQL for intricate queries, MySQL for speed, or MongoDB (with third-party libraries) for NoSQL agility. Business logic, request processing, and response return are taken care of by the views.py file. Views communicate with models and pass data to templates or return JSON responses for APIs. Django REST Framework (DRF) is an extension of Django's abilities to create RESTful APIs, allowing for ease of communication between frontend and backend systems. URLs are mapped to views in urls.py, which directs user requests accordingly. Functionality is augmented by middleware and serializers, maintaining data validation and security. Authentication, caching, and scalability capabilities built into Django make it a solid option for contemporary web applications, accommodating traditional web pages and API-based architectures.

4.3.1.1 Django REST Framework

Django REST Framework (DRF) is a robust framework for creating RESTful APIs in Django. It has built-in functionalities such as serialization, authentication, pagination, and request handling, which simplify API development and scalability. DRF is commonly used to create backend services that communicate with web and mobile applications.

Key Components of Django REST Framework are

Serializers: Serializers turn Django model instances into JSON format to be used for API responses. They also parse and validate incoming JSON data into Python objects before saving them into the database. DRF offers ModelSerializer for auto-mapping of Django model fields and Serializer for manual serialization logic.

Views: DRF provides support for function-based views and class-based views to manage API logic. Generic views and views set, further ease API development by offering in-built CRUD operations.

Authentication & Permissions: DRF provides support for authentication schemes such as Token Authentication, JWT, and OAuth. Permission classes (e.g., IsAuthenticated, IsAdminUser) limit access to certain endpoints.

Routers & URL Routing: DRF eases API routing with routers, which generate URL patterns automatically for viewsets.

Pagination & Filtering: DRF includes in-built pagination, filtering, and throttling for handling large data sets and API request limits.

4.3.1.2 Models

In Django, a Model is a Python class that specifies the schema of a database table. It is the data layer of Django's framework and relies on Object-Relational Mapping (ORM) to communicate with the database without the use of raw SQL. Every model extends django.db.models.Model and specifies fields such as CharField, IntegerField, and ForeignKey to state data attributes. Models accommodate in-built database queries through the ORM like .all(), .filter(), and .create(). In order to implement changes, the developers execute makemigrations and migrate. Models come in handy in implementing database operations efficiently in Django applications. This project model is centered around dynamically creating SQL and NoSQL queries with GPT-3.5 Turbo based on user input, database schema, and pre-defined prompts. The system makes secure, context-aware query generation possible while facilitating interaction with relational and non-relational databases.

The process of generating the query starts with three main inputs: user input from the frontend, the schema of the table from the database (PostgreSQL for SQL queries and MongoDB for NoSQL queries), and the stored prompt template in PostgreSQL.

These inputs direct GPT-3.5 Turbo to create the correct database queries. The query that is generated then goes through two essential checks prior to execution. The out-of-context check ensures that the user’s input aligns with the database schema, preventing irrelevant or incorrect queries. The admin check restricts certain queries, allowing execution only if the user has admin privileges. These validations help maintain data security and accuracy. The frontend first shows the query that is generated. When the user presses the Execute button, the query is transmitted to the corresponding database, where it is run. The retrieved results are then shown on the frontend. This orderly method helps generate queries efficiently while imposing security practices. By combining AI-based query generation and validation checks, the system improves usability and protects database integrity.

4.4 DATABASE

A database is a systematic collection of organized data, stored in a way that allows for efficient retrieval, manipulation, and management. It is a centralized repository that stores data in a structured format, enabling easy access, updating, and analysis. We use PostgreSQL as SQL database and MongoDB for NoSQL database.

4.4.1 POSTGRESQL

PostgreSQL is a versatile, open-source relational database management system (RDBMS) with a reputation for sophisticated features, reliability, and support for intricate queries. It is meant to support diverse data types and provides rich support for managing both structured and unstructured data. Being an extensible platform, PostgreSQL enables programmers to define customized functions, data types, and operators, so it can be used in multiple applications.

In the TalkToData framework, PostgreSQL is the central database, which offers a robust and effective environment for running structured queries derived from natural language requests. Its functionalities allow users to access complex datasets with ease while ensuring data integrity and performance.

4.4.2 MONGODB

MongoDB is a feature-rich, open-source NoSQL database management system that is popular for its flexibility, scalability, and support for large amounts of unstructured data. MongoDB stores data in document-oriented form in the form of BSON (Binary JSON), and it supports schema-less data storage and dynamic querying. MongoDB is a highly scalable platform that supports horizontal scaling, sharding, and replication and is therefore used in a variety of applications. Within the TalkToData system, MongoDB is used as the central NoSQL database, which gives a highly efficient and flexible space for managing unstructured and semi-structured data.

Using these modules, the system provides a seamless interface where natural language inputs are parsed and translated into correct SQL queries, making relational databases easier to interact with for users.

CHAPTER 5

RESULTS

This chapter presents the outcomes and observations derived from the implementation of the project. This section aims to showcase the effectiveness of the developed framework in transforming natural language queries into accurate and usable SQL commands for relational databases.

5.1 DATASETS

This section provides an overview of the data sources utilized in testing and validating the project. It details the structure, type, and domain relevance of the datasets, as they play a crucial role in assessing the effectiveness of the natural language query generation framework.

5.1.1 CATEGORIES TABLE

The Categories table is a critical component of the airline marketplace database, designed to organize and categorize various aspects of airline services.

id	category_name
1	Beauty & Personal Care
2	Perfumes & Fragrances
3	Wearable Technology
4	Headphones & Earbuds
5	Boys' Clothing
6	Girls' Clothing
7	Backpacks
8	Men's Clothing
9	Men's Accessories
10	Men's Watches
11	Men's Shoes
12	Women's Clothing
13	Women's Accessories
14	Women's Watches

Table 5.1 Categories

This table comprises two primary columns:

id: A unique identifier for each category, serving as the primary key. This integer value ensures that each category can be distinctly referenced within the database.

category_name: A descriptive name for each category, which provides insight into the type of services or offerings available within the marketplace. This column contains a total of 17 distinct categories.

5.1.2 PRODUCTS TABLE

The Products table is an essential component of the airline marketplace database, designed to store information about various products associated with the 17 categories defined in the Categories table.

	A	B	C	D	E	F	G	H	I	J	K
1	asin	title	imgurl	stars	reviews	price	listprice	category_i	isbestselle	boughtinlastmonth	
2	B0B6336C	women's s	https://m.	4.4	0	25.99	0	12	f	200	
3	B0C9TGRX	30 pairs br	https://m.	0	0	35.99	0	12	f	0	
4	B09QYQN'	women's s	https://m.	3.9	0	40.99	0	12	f	50	
5	B0BVRFD6	womens s	https://m.	4	0	17.99	18.99	12	f	200	
6	B074QJHK	cotton cor	https://m.	4.4	0	23.99	0	12	f	100	
7	B075RWK7	women's l	https://m.	4.2	0	24.99	32.99	12	f	100	
8	B08YLNTD	women's r	https://m.	4.5	0	49.99	69	12	f	50	
9	B0BL6V18	super stick	https://m.	4	0	16.99	17.99	12	f	200	
10	B0BW2Z9)	women's b	https://m.	4	0	39.99	0	12	f	50	
11	B0C1Z7JD	women flo	https://m.	3.8	0	29.85	0	12	f	50	
12	B0BGCBZZ	women thi	https://m.	4.7	0	15.99	0	12	f	300	
13	B0B9S7FB'	women's p	https://m.	3.9	0	25.99	0	12	f	100	

Table 5.2 Products

5.1.3 CUSTOMERS TABLE

The Customers table is designed to manage information about customers who purchase products within the airline domain. It includes key columns such as customer_id, a unique identifier for each customer; first_name and last_name, which allow for personalized communication; and email and phone_number, serving as primary contact methods.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	customer_	first_name	last_name	email	phone_number	date_of_birt	nationality	address	freq_flyer_numb	loyalty_id	loyalty_tie	loyalty_po	account_status
2	1	John	Carlson	darrell81@gm	444-118-0533x1	29-07-1947	Japan	59753 Carol	FF57266510	LLTY3434114	Platinum	80001	Inactive
3	2	Michael	Spencer	richard39@gn	(815)327-8500x	24-10-1987	Luxembou	1262 Hanser	FF21357875	LLTY0933181	Bronze	45677	Active
4	3	Mark	Gilbert	morsemark@y	(629)646-8868	24-06-1996	Guinea	423 Jennifer	FF11079917	LLTY7944269	Platinum	92658	Suspended
5	4	Susan	Jones	robert30@col	(485)693-4659x	20-04-1982	Kiribati	1295 James	FF93799776	LLTY4537152	Bronze	7062	Inactive
6	5	Isaiah	Alvarez	cynthiarussell(	5930734917	18-07-1954	Monteneg	46482 Elizab	FF75482646	LLTY6294955	Silver	80002	Inactive
7	6	Colin	Thompson	anne20@hotn	-5381	14-06-1949	Congo	PSC 6601, Bo	FF82127811	LLTY0632386	Platinum	37529	Suspended
8	7	Shelia	Hopkins	russellgardner	+1-386-990-446	21-10-1956	Moldova	971 Humphr	FF11187485	LLTY4305286	Bronze	61193	Suspended
9	8	Debbie	Perry	angelica83@h	435-200-5660	18-07-1954	Greece	79988 Rebec	FF39109655	LLTY6394683	Bronze	14947	Inactive
10	9	Jennifer	Beard	coxfelicia@ya	294-566-6054x2	17-12-1982	Venezuela	Unit 6210 Bo	FF61594667	LLTY3356122	Gold	80359	Suspended
11	10	Austin	Doyle	jryan@little.in	850.923.3039	02-11-1959	Angola	852 Sandra F	FF66933011	LLTY9385856	Silver	7056	Inactive
12	11	Luis	Lambert	melissamorale	935.548.1546x9	31-12-1963	Guernsey	004 Nguyen	FF98387435	LLTY0129213	Platinum	10329	Active

Table 5.3 Customers

5.1.4 ORDERS TABLE

The Orders table serves as a comprehensive record of all transactions within the airline marketplace, capturing essential details for each order. Key columns include history_id, product_id, and customer_id, linking orders to specific products and customers. It tracks financial aspects such as quantity, total_price, and payment_status, along with order status and delivery information. Additionally, the table manages loyalty points, return statuses, and special instructions, providing a structured framework for efficient order processing and customer service.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	history_id	product_ic	customer_	FFN	quantity	price_per_	total_price	order_date	status	shipment_dat	mode_of_	delivery_lo	payment_	payment_	payment_	payment_dat	currency
2	1	B07Y479K(	5	FF7548264	2	11.99	23.98	23-05-2022	delivered	27-05-2022	gate delive	Gate A5	airport wa	failed	TXN63913	23-05-2022	EUR
3	2	B08RSSNW	81	FF4179233	5	40	200	13-08-2023	canceled	21-08-2023	gate delive	Terminal 1	PayPal	paid	TXN62616	13-08-2023	EUR
4	3	B08VZYN7	30	FF7349816	2	35.16	70.32	02-06-2022	canceled	09-06-2022	in-store pi	Terminal 2	credit card	failed	TXN74027	02-06-2022	USD
5	4	B08VJF52Y	3	FF1107991	4	29.99	119.96	12-11-2023	canceled	15-11-2023	gate delive	Gate A5	airport wa	refunded	TXN65447	12-11-2023	EUR
6	5	B0C4F6VD	85	FF9391443	2	14.99	29.98	07-06-2023	returned	11-06-2023	courier	Terminal 1	credit card	refunded	TXN54303	07-06-2023	EUR
7	6	B0C5HW1(	88	FF3308557	4	13.88	55.52	11-11-2023	returned	17-11-2023	in-store pi	Terminal 2	PayPal	refunded	TXN37004	11-11-2023	USD
8	7	B09WMV6	38	FF3051313	2	22.99	45.98	12-01-2022	returned	14-01-2022	courier	Gate A5	credit card	failed	TXN42796	12-01-2022	USD
9	8	B09WMV6	70	FF5286131	2	22.99	45.98	19-12-2023	delivered	24-12-2023	in-store pi	Terminal 2	PayPal	failed	TXN97879	19-12-2023	USD
10	9	B09YBGV4	49	FF1247833	4	33.99	135.96	07-07-2022	canceled	17-07-2022	gate delive	Terminal 1	airport wa	pending	TXN72107	07-07-2022	EUR
11	10	B07Y479K(	43	FF0028644	5	11.99	59.95	03-01-2022	shipped	05-01-2022	gate delive	Terminal 2	credit card	pending	TXN87956	03-01-2022	EUR
12	11	B09YK71C(	47	FF2498291	5	539	2695	21-01-2022	delivered	22-01-2022	gate delive	Terminal 2	PayPal	pending	TXN57820	21-01-2022	USD
13	12	B0C7ZRH3	46	FF0522644	3	12.99	38.97	08-07-2023	delivered	18-07-2023	courier	Terminal 1	PayPal	refunded	TXN81028	08-07-2023	USD
14	13	B09ZXXG4(	7	FF1118748	3	17.99	53.97	09-01-2023	returned	11-01-2023	in-store pi	Terminal 2	airport wa	pending	TXN33559	09-01-2023	USD
15	14	B09WMV6	67	FF4181497	1	22.99	22.99	06-05-2023	shipped	12-05-2023	courier	Terminal 1	airport wa	paid	TXN67321	06-05-2023	EUR
16	15	B07B5FTY!	67	FF4181497	4	16	64	27-06-2023	delivered	05-07-2023	in-store pi	Terminal 1	credit card	failed	TXN21926	27-06-2023	USD
17	16	B09P4JSB3	42	FF7348976	2	26.99	53.98	22-02-2022	returned	23-02-2022	courier	Terminal 2	credit card	failed	TXN31795	22-02-2022	EUR
18	17	B0BX74Z6:	58	FF8343223	4	25.95	103.8	04-06-2022	shipped	14-06-2022	courier	Terminal 2	credit card	pending	TXN19133	04-06-2022	USD
19	18	B0C6XQ6V	55	FF3001145	1	39.99	39.99	09-12-2023	shipped	18-12-2023	courier	Terminal 1	airport wa	paid	TXN39981	09-12-2023	USD
20	19	B0BL1PHK	46	FF0522644	4	28.73	114.92	15-02-2023	returned	20-02-2023	in-store pi	Terminal 2	airport wa	refunded	TXN74671	15-02-2023	USD

Table 5.4 Orders

5.1.5 FAQ TABLE

The FAQ table serves as a centralized repository for frequently asked questions related to products. Key columns include faq_id, customer_id and product_id, linking questions to specific customers and products. The faq_category classifies the type of inquiry, while faq_question and faq_answer provide the details of the questions and their responses. The faq_datetime records when each entry was created, ensuring timely updates and relevance.

	A	B	C	D	E	F	G	H
1	faq_id	customer_	product_id	product_nan	faq_category	faq_question	faq_answer	faq_datetime
2	10011	70	B00W88H62V	14mm long, (	Miscellaneous FAQs	What is the order stat	The order sta	12-01-2024 19:00
3	10015	72	B01L0KNGNQ	mens cargo s	Delivery and Shipping FAC	What is the delivery tir	Your product	08-06-2022 05:00
4	10021	73	B098KY45ZH	men's extrer	Delivery and Shipping FAC	What is the airport pic	The airport pi	08-11-2022 06:00
5	10189	7	B09XCQJSNT	ua sideline n	Technical and Support FA(	What is the product su	The product s	13-09-2022 23:00
6	10283	23	B07HP8YF4Q	romantic del	Technical and Support FA(	What is the warranty c	The warranty	11-12-2022 21:00
7	10351	39	B0BX7S37MF	jbu womens	Product-Related FAQs	What is the materials c	The materials	12-09-2022 16:00
8	10370	31	B098KY45ZH	men's extrer	Airport-Specific FAQs	What is the duty free c	The duty free	01-07-2023 09:00
9	10433	7	B09Q5T3CNG	rope sling ba	Delivery and Shipping FAC	What is the shipping cc	The shipping c	30-05-2022 15:00
10	10483	85	B073ZKZVHC	mouth guard	Miscellaneous FAQs	What is the operating	The operating	14-04-2023 15:00

Table 5.5 FAQ

5.2 DATABASES

5.2.1 POSTGRESQL DATABASE

The PostgreSQL database contains five primary tables: Categories, Products, Customers, Orders, and FAQ.

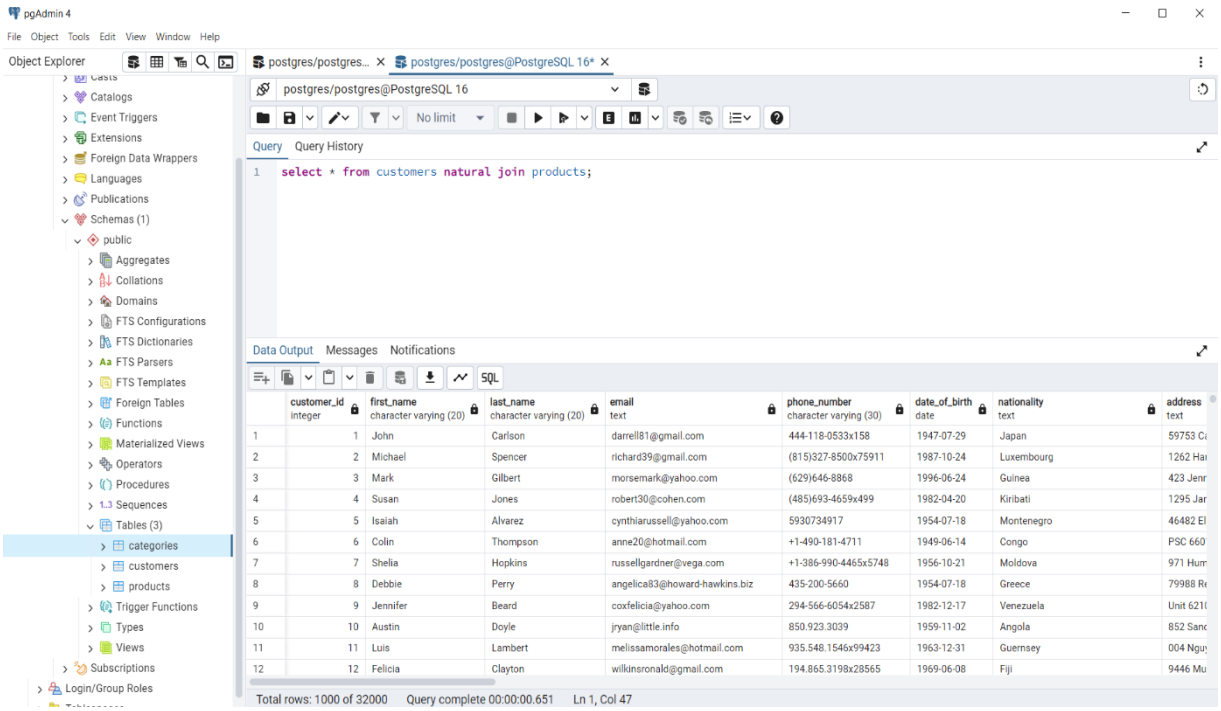

Fig. 5.1 Database in PostgreSQL

The Categories table stores different types of airlines services in organized form and can be considered the basis of the product categories. The Products table is associated with these categories in a direct reference, presenting customers with offerings as per them. The Customers table stores data important to note of users visiting or using the marketplace, while Orders table logs transactions made and tracks customer shopping activity. In order to improve the user experience and aid in proper customer service, the FAQ table has general questions and their corresponding answers regarding the products.

5.2.2 MONGODB DATABASE

This database is also created using the same tables that we used in PostgreSQL. The tables are stored in BSON format modelled in key-value pairs.

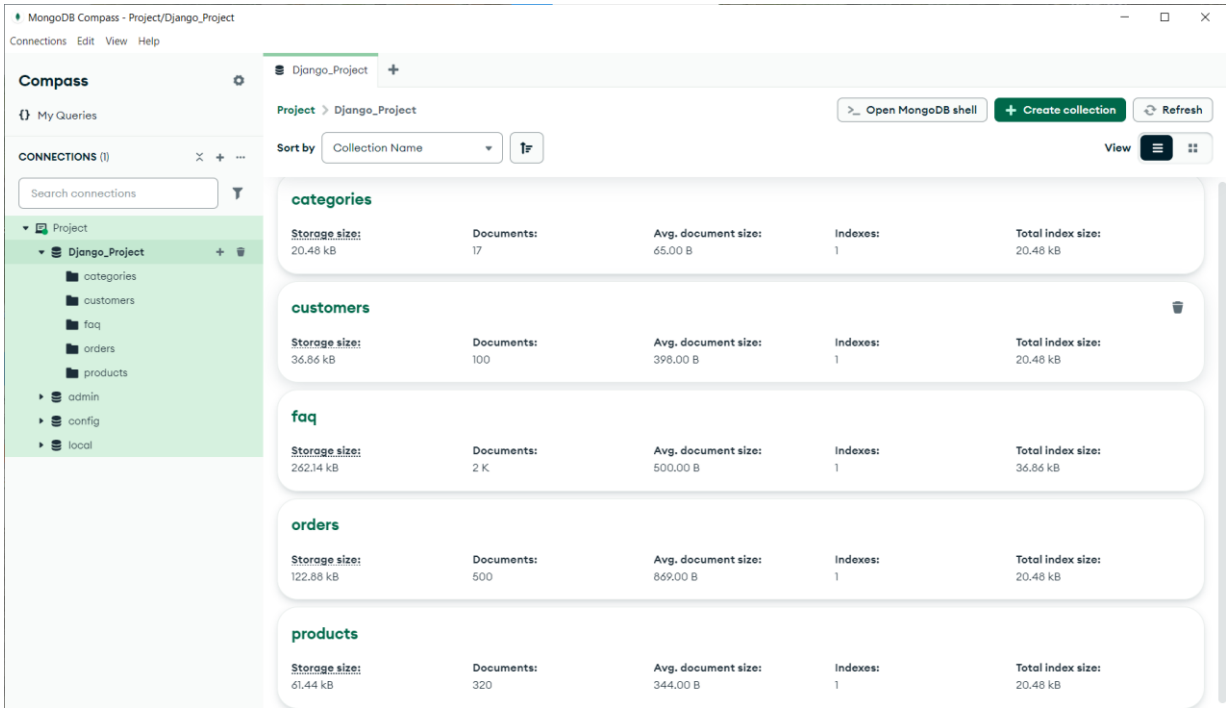

Fig 5.2 MongoDB Database

5.3 FRONT ENT PAGES

5.3.1 User Registration

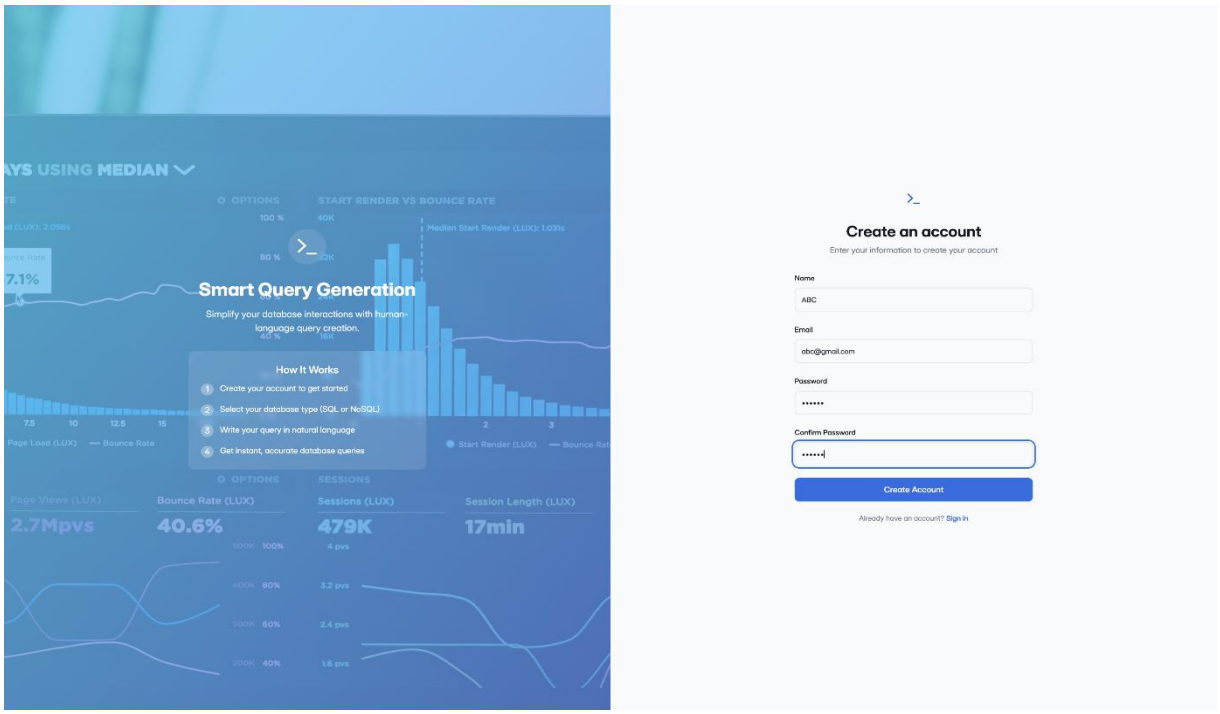

Fig 5.3 User Registration

The user registration page allows new users to create an account by providing required details

such as name, email, and password. It includes form validation to ensure accurate information, password strength checks, and error handling for duplicate emails. Upon successful registration, users gain access to the application's features.

5.3.2 User login

The SQL Generator login page enables registered users to access the platform by entering their email and password. It provides a "Create an account" option for new users. The page also highlights key functionalities, including NLP-powered SQL and NoSQL query generation, real-time query execution, and support for multiple database types.

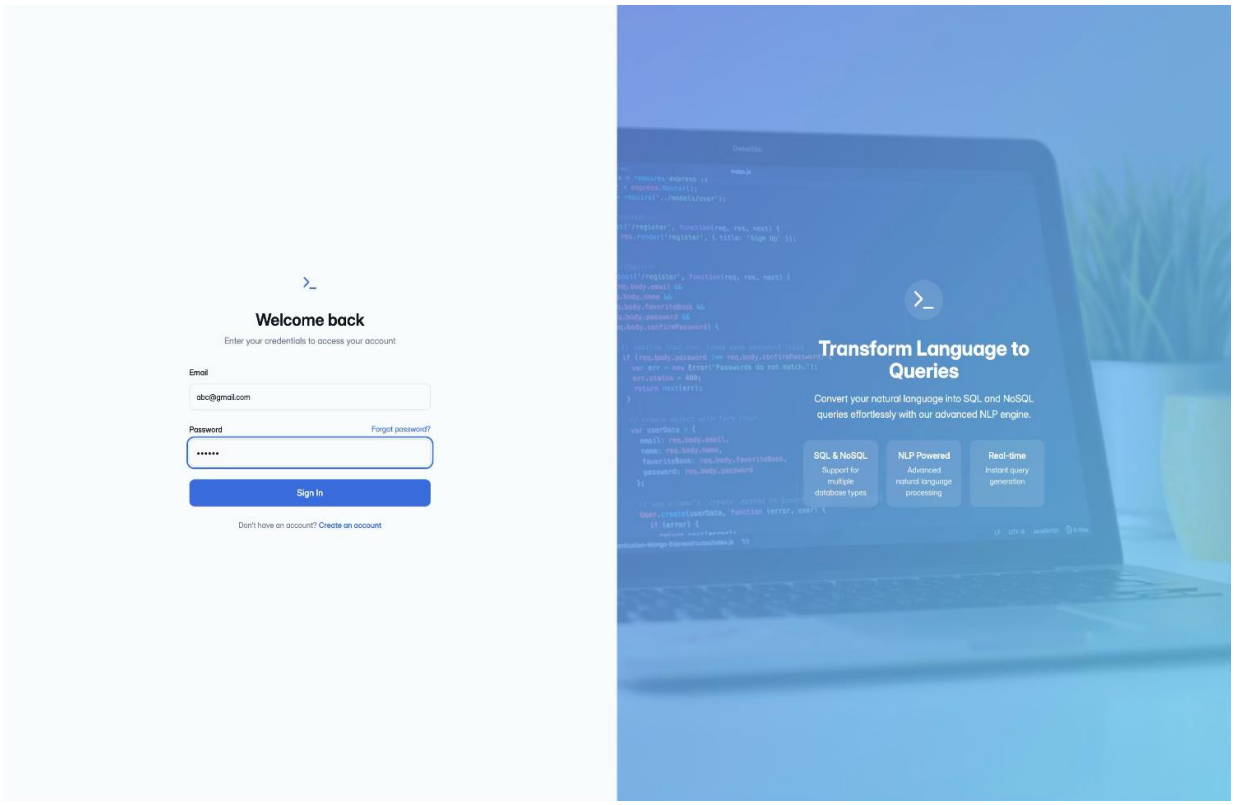

Fig 5.4 User Login

5.3.3 Database Selection page

The database selection interface provides options to select between NoSQL and SQL databases prior to generating queries. It maintains database compatibility and allows smooth query generation. The users can identify the target database and maintain correct execution of the

queries. It boosts flexibility and provides support for a variety of storage and retrieval demands of data.

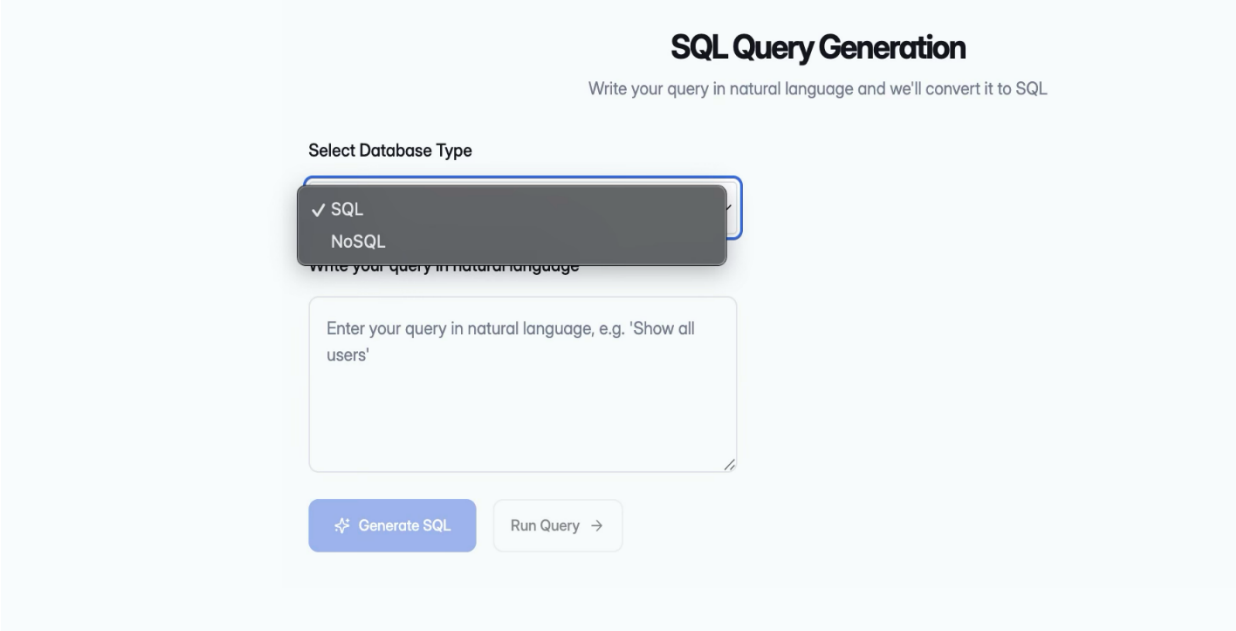

Fig 5.5 Database Selection

5.3.4 SQL Generation page

The SQL generation page allows users to generate SQL queries from plain English inputs.

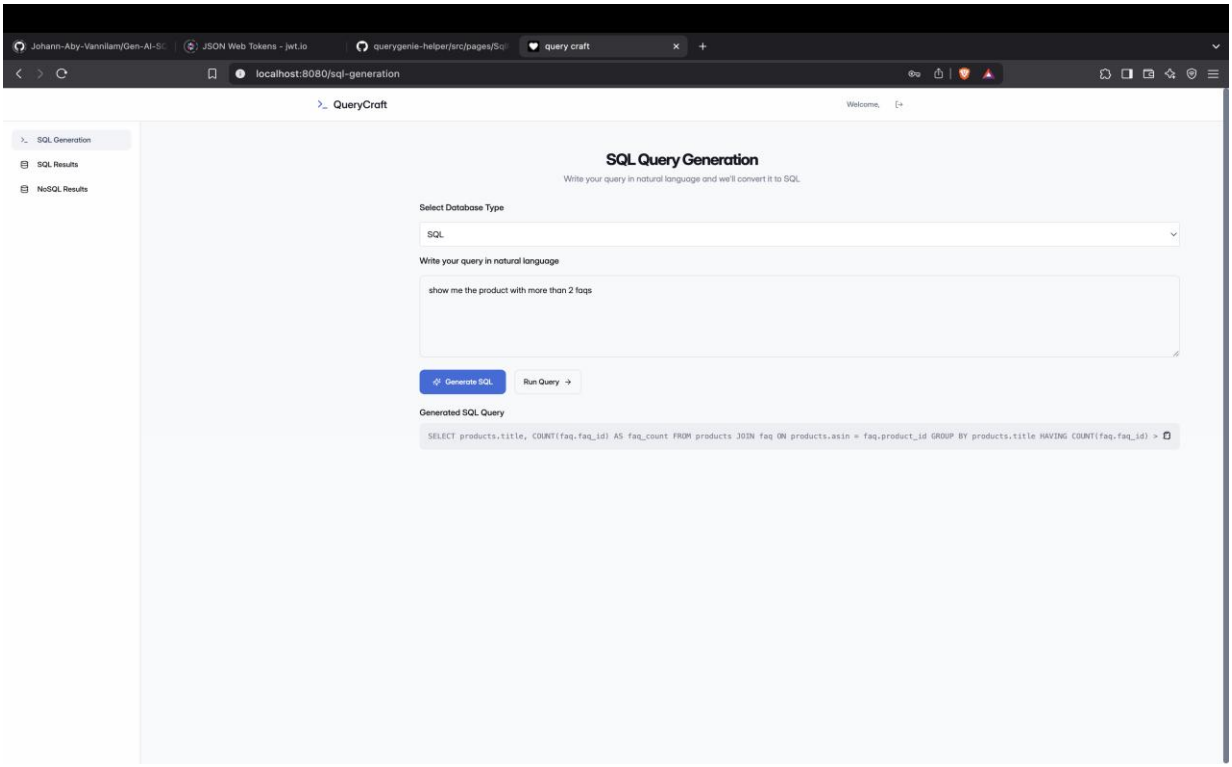

Fig 5.6 SQL Generation

It uses an enhanced NLP model to parse user requests and provide correct SQL queries based on the chosen database schema. Users can enter commands in English, and the system translates them into executable SQL commands. The produced queries are presented for inspection prior to execution for transparency and accuracy. This capability makes database interactions easier and query creation more accessible to both technical and non-technical users.

5.3.5 SQL Execution

This page enables users to see the output retrieved from the database via the SQL query run. It provides transparency through the exhibition of the retrieved information, enabling users to confirm the correctness of the query results and make the required changes where necessary. This is a feature that helps the user to see the data in the user interface rather than going to the database to run the sql query.

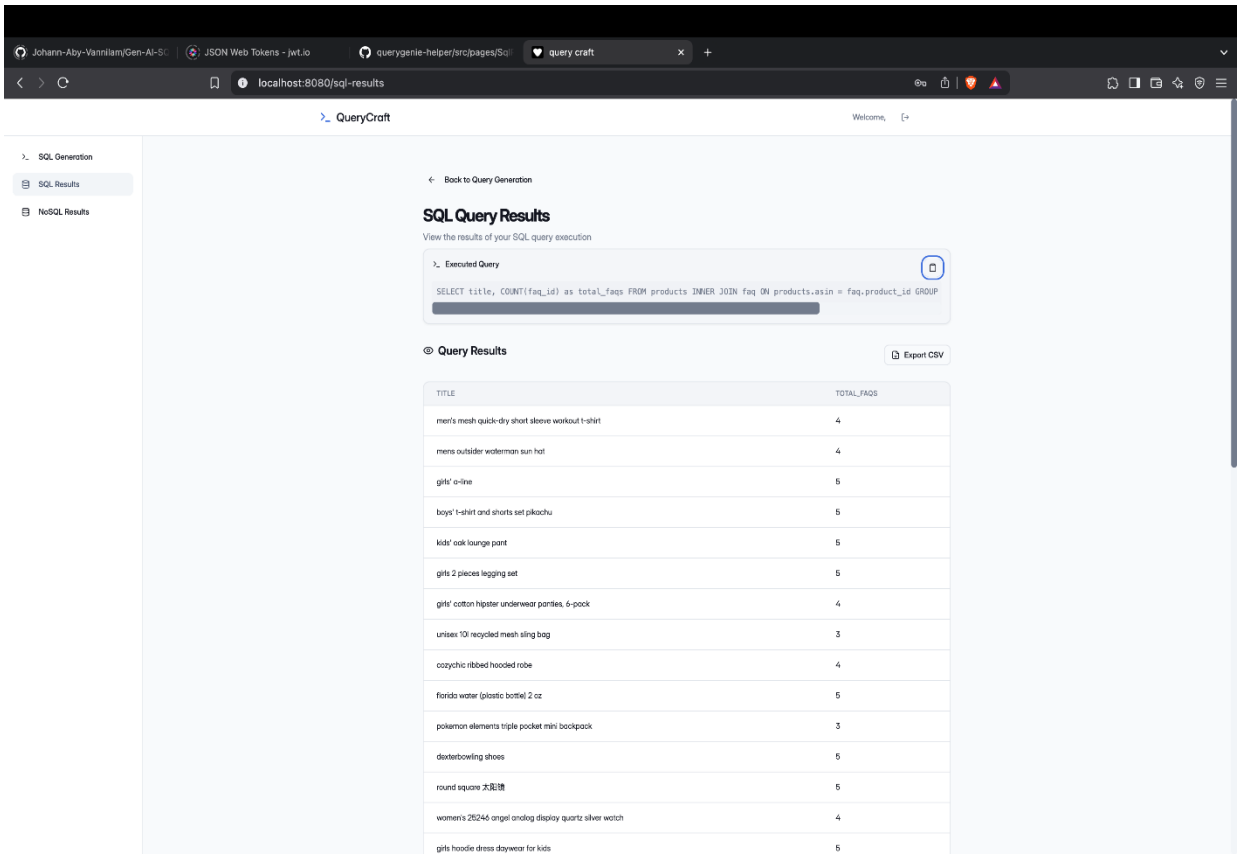

Fig 5.7 SQL Execution

5.3.6 NoSQL Generation

The NoSQL Generation page allows users to create NoSQL queries from natural language requests, making database interactions easier. It understands user requests, translates them into correct NoSQL queries, and presents the generated query for validation. This functionality eliminates the requirement of users knowing NoSQL syntax beforehand, and data retrieval and manipulation becomes easier. Users can fetch, update, and manipulate data with ease. By closing the gap between the human language and NoSQL query formats, this page increases efficiency and ease of use, enabling both technical and non-technical individuals to communicate easily and accurately with NoSQL databases.

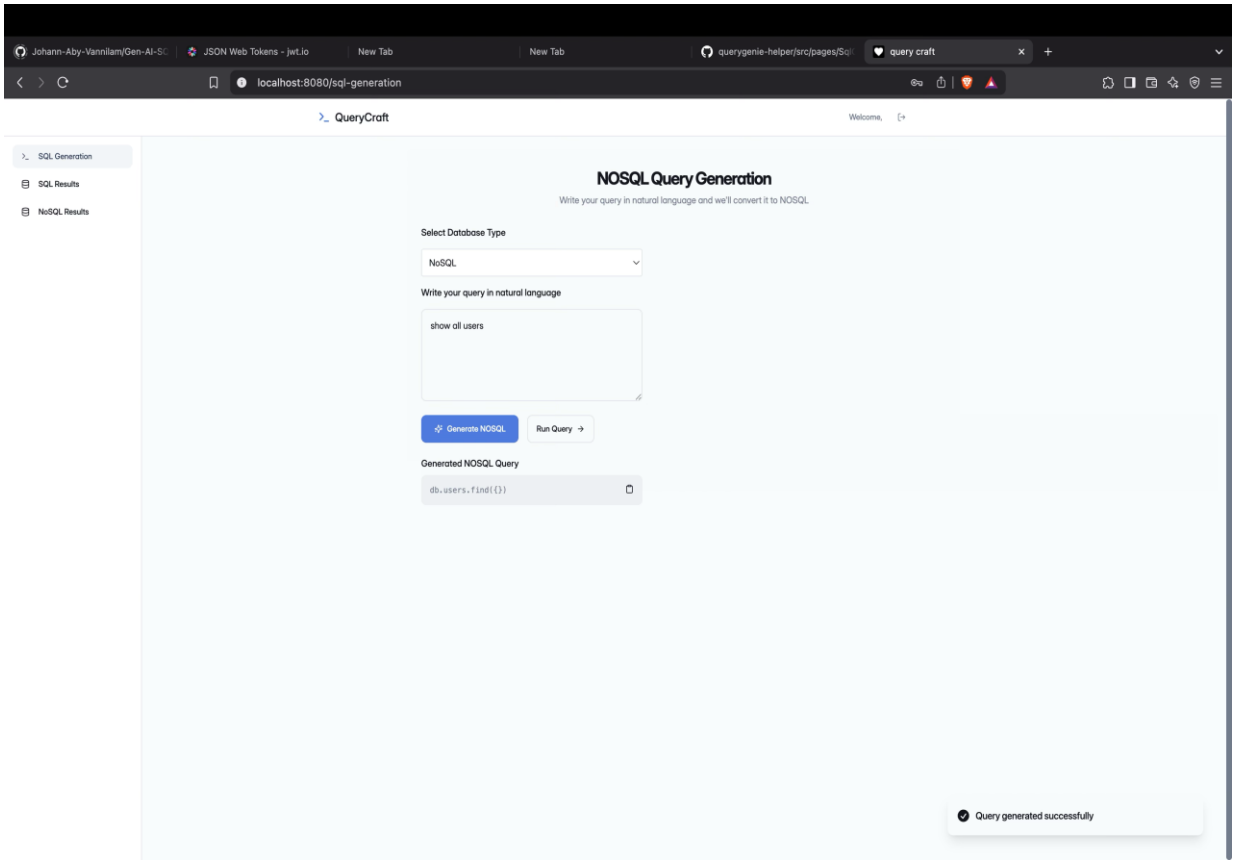

Fig 5.8 NoSQL Generation

5.3.7 NoSQL Execution

The NoSQL Executor page is used to execute the generated NoSQL queries and display the extracted results from the database. The page offers a way to implement the generated NoSQL query against the connected NoSQL database after the NoSQL query has been generated from the natural language input of the user. It shows the data retrieved in a formatted manner,

making it easy for users to check and analyze the results. The page provides smooth interaction with the database, allowing users to execute read and write operations effectively. The functionality improves usability by making it easy to query the database, even for individuals who lack profound knowledge of databases.

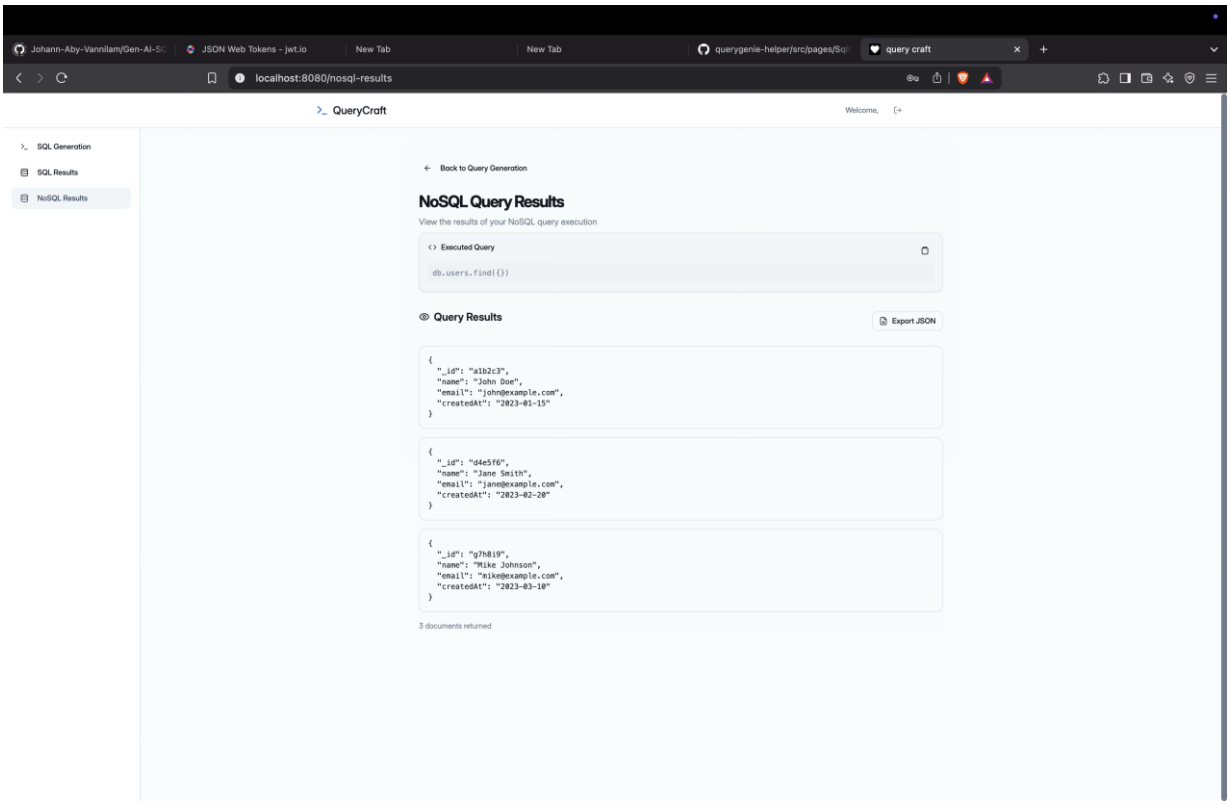

Fig 5.9 NoSQL Execution

CHAPTER 6

CONCLUSION AND FUTURE WORKS

This project successfully developed a natural language querying framework that enables seamless interaction with both relational (PostgreSQL) and schema-less (MongoDB) databases. By leveraging large language models (LLMs) and advanced prompting techniques, the system effectively translates user-friendly input into precise database queries, eliminating the need for manual query writing. This approach enhances accessibility, allowing both technical and non-technical users to interact with databases effortlessly. The results demonstrate that the framework significantly simplifies data retrieval, improving efficiency and reducing complexity in database interactions. With its modular, plug-and-play design, the system can be easily integrated into various applications across industries that require quick and intuitive access to structured and unstructured data.

Looking forward, several enhancements can further improve the framework's capabilities. Expanding NoSQL support will allow the system to handle more complex queries, making it more adaptable to diverse data environments. Multi-language processing will ensure accessibility for users across different linguistic backgrounds, broadening its usability. Integrating adaptive learning mechanisms will enable the system to refine query accuracy over time based on user feedback, making interactions smarter and more context-aware. Enhanced error handling will help identify and correct ambiguous or inaccurate queries, improving reliability and user experience. Additionally, optimizing the system for scalability will support high-volume query processing, making it suitable for enterprise applications handling large datasets.

By implementing these advancements, the framework has the potential to revolutionize database interactions, bridging the gap between natural language processing and structured data retrieval. It paves the way for a future where anyone, regardless of technical expertise, can retrieve, analyze, and utilize data with ease. This project serves as a foundation for making data-driven decision-making more intuitive, efficient, and widely accessible across industries.

REFERENCES

[1] Zhong, Victor, Caiming Xiong, and Richard Socher. "*Seq2sql: Generating structured queries from natural language using reinforcement learning*." arXiv preprint arXiv:1709.00103 (2017).

[2] Xu, Xiaojun, Chang Liu, and Dawn Song. "*Sqlnet: Generating structured queries from natural language without reinforcement learning.*" arXiv preprint arXiv:1711.04436 (2017).

[3] Yu, Tao, et al. "*Typesql: Knowledge-based type-aware neural text-to-sql generatio*n." arXiv preprint arXiv:1804.09769 (2018).

[4] Guo, Jiaqi, et al. "*Towards complex text-to-sql in cross-domain database with intermediate representation*." arXiv preprint arXiv:1905.08205 (2019).

[5] Xie, Tianbao, et al. "*Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models.*" arXiv preprint arXiv:2201.05966 (2022).

[6] W. Lin, P. Babyn, and W. Zhang, *"Context-based Ontology Modelling for Database: Enabling ChatGPT for Semantic Database Management,"* arXiv preprint arXiv:2303.07351, 2023.

[7] X. Dong *et al.*, *"C3: Zero-shot text-to-sql with chatgpt,"* arXiv preprint arXiv:2307.07306, 2023.

[8] W. Aerts, G. Fletcher, and D. Miedema, *"A Feasibility Study on Automated SQL Exercise Generation with ChatGPT-3.5,"* Proceedings of the 3rd International Workshop on Data Systems Education: Bridging Education Practice with Education Research, 2024.

[9] G. Katsogiannis-Meimarakis, C. Tsapelas, and G. Koutrika, *"Natural Language Data Interfaces: From Keyword Search to ChatGPT, are we there yet?,"* EDBT/ICDT Workshops, 2023.

[10] Y. Jiang and C. Yang, *"Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language into Structured Query Language within a Spatial Database,"* ISPRS International Journal of Geo-Information, vol. 13, no. 1, p. 26, 2024.

[11] M. Bukhsh *et al.*, *"An interpretation of long short-term memory recurrent neural network for approximating roots of polynomials," IEEE Access*, vol. 10, pp. 28194-28205, 2022.

[12] M. Omar *et al.*, *"Robust natural language processing: Recent advances, challenges, and future directions," IEEE Access*, vol. 10, pp. 86038-86056, 2022.

[13] Z. Xu *et al.*, *"Building a natural language query and control interface for IoT platforms," IEEE Access*, vol. 10, pp. 68655-68668, 2022.

[14] P. Danenas and T. Skersys, *"Exploring natural language processing in model-to-model transformations," IEEE Access*, vol. 10, pp. 116942-116958, 2022.

[15] M. Boukhlif *et al.*, *"Natural Language Processing-based Software Testing: A Systematic Literature Review," IEEE Access*, 2024.

[16] Y. Li, J. Shi, and Z. Zhang, *"An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering," IEEE Access*, 2024.

[17] T. Thaminkaew, P. Lertvittayakumjorn, and P. Vateekul, *"Prompt-Based Label-Aware Framework for Few-Shot Multi-Label Text Classification," IEEE Access*, 2024.

[18] Y. Xia *et al.*, *"Generation of Asset Administration Shell with Large Language Model Agents: Towards Semantic Interoperability in Digital Twins in the Context of Industry 4.0," IEEE Access*, 2024.

[19] Z. Hu *et al.*, *"Prompting Large Language Models with Knowledge-Injection for Knowledge-Based Visual Question Answering," Big Data Mining and Analytics*, vol. 7, no. 3, pp. 843-857, 2024.

[20] S. Gupta, K. Raja, and R. Passerone, *"Visual Prompt Engineering for Enhancing Facial Recognition Systems Robustness against Evasion Attacks," IEEE Access*, 2024.