# Talk2DB: Talk to Your Data – Final Project Report

## Introduction:

The Talk2DB project is an innovative web application designed to bridge the gap between non-technical users and database systems by enabling the use of natural language queries. By leveraging Natural Language Processing (NLP) techniques, Talk2DB allows users to interact with both SQL (MySQL) and NoSQL (MongoDB) databases. This provides an intuitive and accessible platform for exploring and querying datasets without needing prior technical knowledge of SQL syntax or database management, making data interaction more inclusive and user-friendly.

The primary goal of the project is to create an interactive platform where users can simply type their database queries in natural language. The system processes these inputs, translating them into executable SQL or MongoDB queries. These queries are then executed to retrieve relevant data from the database, which is presented in a clear and understandable format. This approach simplifies database interaction for non-technical users and fosters better access to valuable data insights.

## Planned Implementation:

As described in the project proposal, the implementation of Talk2DB follows a modular approach that integrates several key components to streamline database interaction for users:

- **Natural Language Processing (NLP):** The system leverages nltk and re(regular expression) libraries to process user input, tokenize queries, and detect important patterns and keywords. These elements are then mapped to corresponding MySQL or MongoDB queries. The NLP module ensures that diverse natural language inputs are accurately interpreted, making the query system versatile and user-friendly.
- **Pattern-Based Query Generation:** Talk2DB employs predefined query templates, which are dynamically populated with keywords extracted from user input. This method provides flexibility, enabling the generation of a wide range of database queries while maintaining scalability and robustness. The modular design ensures that additional query patterns can be integrated seamlessly as the system evolves.
- **Database Integration:** MySQL is utilized for managing structured data in relational databases, while MongoDB supports unstructured or semi-structured data in NoSQL environments. This dual-database approach ensures compatibility with a variety of data types and user needs, allowing for comprehensive data exploration and management.

- **User Interface (UI):** The application features an intuitive, web-based interface built using Streamlit. The UI enables users to interact with the system effortlessly, choose query types, and view query results in a clear, user-friendly format. The design emphasizes accessibility, providing a seamless experience for users without technical expertise in database querying.

This modular and integrated approach ensures that Talk2DB delivers a robust and efficient solution for bridging the gap between natural language inputs and database systems.

**Implemented System Overview:**

Our project, Talk2DB, simplifies user interaction with databases by enabling natural language queries instead of complex SQL/NoSQL commands. The system processes natural language inputs, interprets user queries, and converts them into valid SQL/NoSQL queries that interact with a database. The result is a conversational interface that returns the desired SQL/NoSQL command and data in real-time.

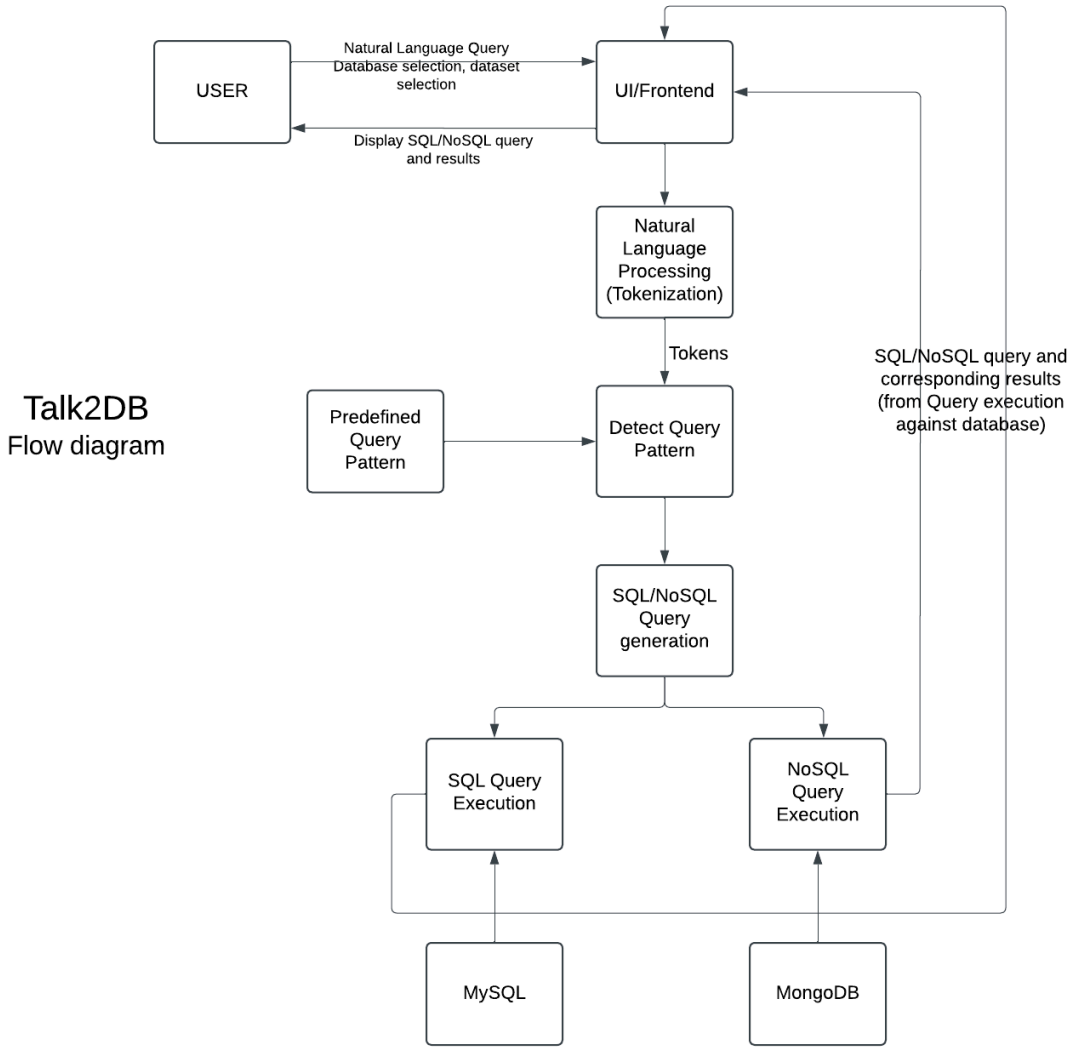
The system was developed as a web application, where users input queries into a chat-like interface. Natural Language Processing (NLP) techniques were used to tokenize the input and identify key components such as column names and table names. These key components are extracted from the given datasets. These include the table names, column names, dataset name and data type of each column. These identified elements were then used to generate the appropriate SQL commands. The results are returned to the user in an intuitive format. The system is designed to be efficient, user-friendly, and scalable, with a robust backend that handles various types of queries.

1. **Database System Selection:**
    - **SQL Database:** MySQL is used to store structured datasets in tables.
    - **NoSQL Database:** MongoDB is employed for more flexible, unstructured datasets, allowing users to work with collections.
2. **Pattern-Based Query Generation:** A query pattern system was developed to generate sample queries automatically using templates like “Total <A> by <B>.” This ensures that users can retrieve a variety of sample queries for both SQL and NoSQL databases without the need for hardcoding.
3. **Natural Language Query Detection:** A module was implemented to detect user-submitted natural language questions and map them to predefined query patterns. This module matches the query to the closest pattern and suggests the corresponding SQL or MongoDB query.
4. **Web-Based UI:** A web browser interface was developed using Python and Streamlit, allowing users to select databases, explore their structure, and submit both sample and natural language queries. The interface displays the generated queries and their results.

5. **Dataset Upload Feature:** Functionality was built to allow users to upload datasets into the selected SQL/NoSQL databases and use Talk2DB to query them interactively.

## Architecture Design

- **Flow Diagram**



**Description:**

1. User Input: The user submits a natural language query through the web interface.
2. UI/Frontend : The UI user interacts with which is also responsible for displaying their query results
3. NLP Processing: The system tokenizes the input using `nltk` and identifies key phrases or terms (such as column names, filters, or aggregation functions).
4. Pattern Recognition: Using regular expressions and NLP algorithms, the system maps the detected patterns to predefined SQL or MongoDB query templates.
5. Query Execution: The generated SQL or MongoDB query is executed on their respective databases.
6. Result Display: The results of the query are displayed to the user in the UI.

## Implementation

## 1. Functionalities

### 1.1 Database Support

The system supports both MySQL (SQL) and MongoDB (NoSQL). Users can explore the structure of both, including tables in MySQL and collections in MongoDB, with dataset information and sample data displayed for clarity.

### 1.2. Sample Query Generation

Talk2DB generates dynamic sample queries for SQL/NoSQL constructs like SELECT, WHERE, GROUP BY, and ORDER BY. The system uses flexible query patterns or templates, avoiding hard-coded statements, ensuring scalability.

### 1.3. Natural Language Support

Users can input natural language queries, which the system analyzes to detect predefined patterns and generate the corresponding SQL or NoSQL query. For example, asking for "total production cost by product category" triggers an aggregation(group by) query.

### 1.4. Query Execution

The generated SQL/NoSQL query is executed against the database, and results are returned to the user in an intuitive format for easy interaction.

### 1.5. Dataset Management

Talk2DB allows users to upload and manage datasets in both SQL and NoSQL databases, enabling interactive querying of uploaded data.

### 1.6. Web-based User Interface

A web interface, built with Python and Streamlit, allows users to select databases, submit natural language queries, and view corresponding SQL/NoSQL query and its results in an easy-to-use format.

### 1.7. Multiple Dataset Testing

Talk2DB was tested with three different datasets each, covering both MySQL and MongoDB, to ensure effective performance across diverse data types and structures.

## 2. Tech Stack

### 2.1. pymysql:

Pymysql is the interface with MySQL, enabling the system to manage connections, retrieve and query data, and support transactions.

### 2.2. pymongo:

Pymongo is the interface with MongoDB, allowing the system to interact with collections, execute queries, and perform aggregation or filter operations on NoSQL data.

### 2.3. nltk (Natural Language Toolkit):

NLTK is used to parse and process natural language input, tokenize queries, and detect patterns that can be mapped to SQL or MongoDB queries.

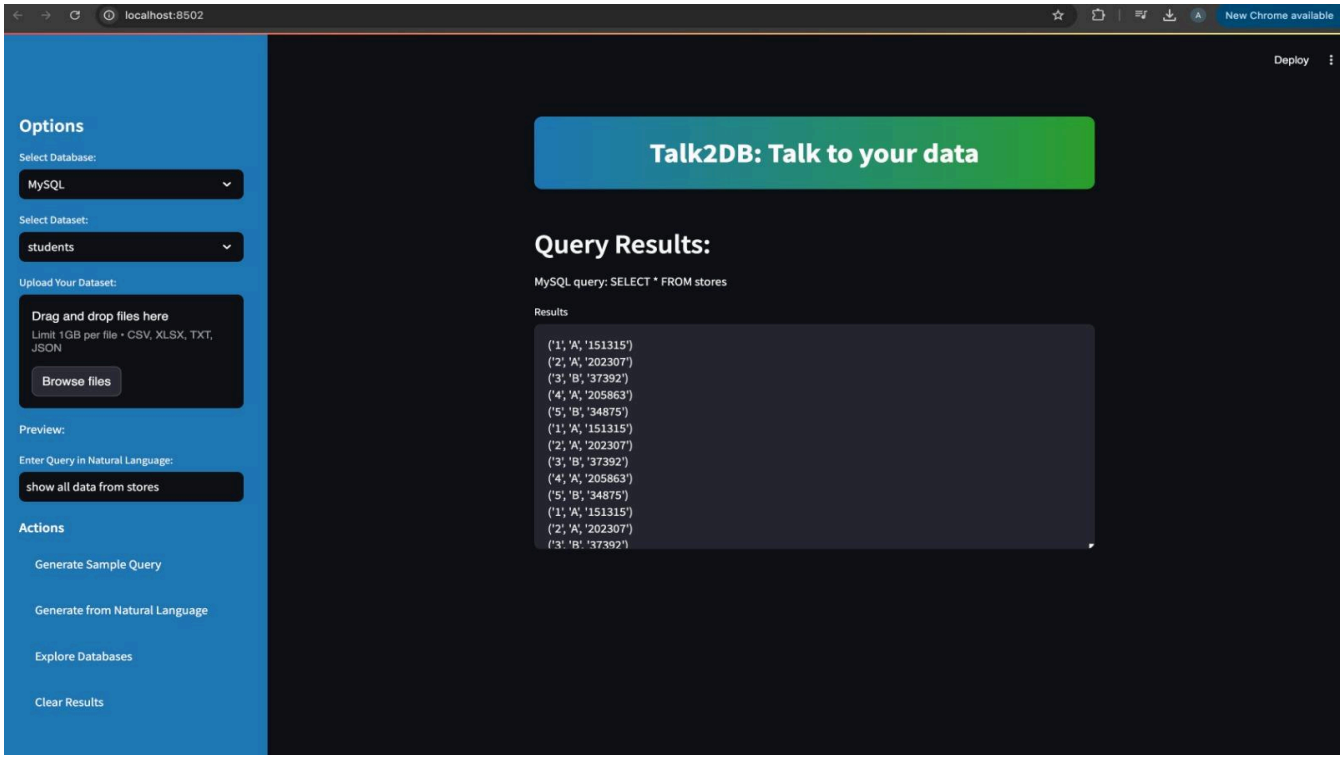
### 2.4. re (Regular Expressions):

Re is essential for pattern matching within strings, especially for identifying specific components (e.g., column names, filters, and aggregations) in the user's input.

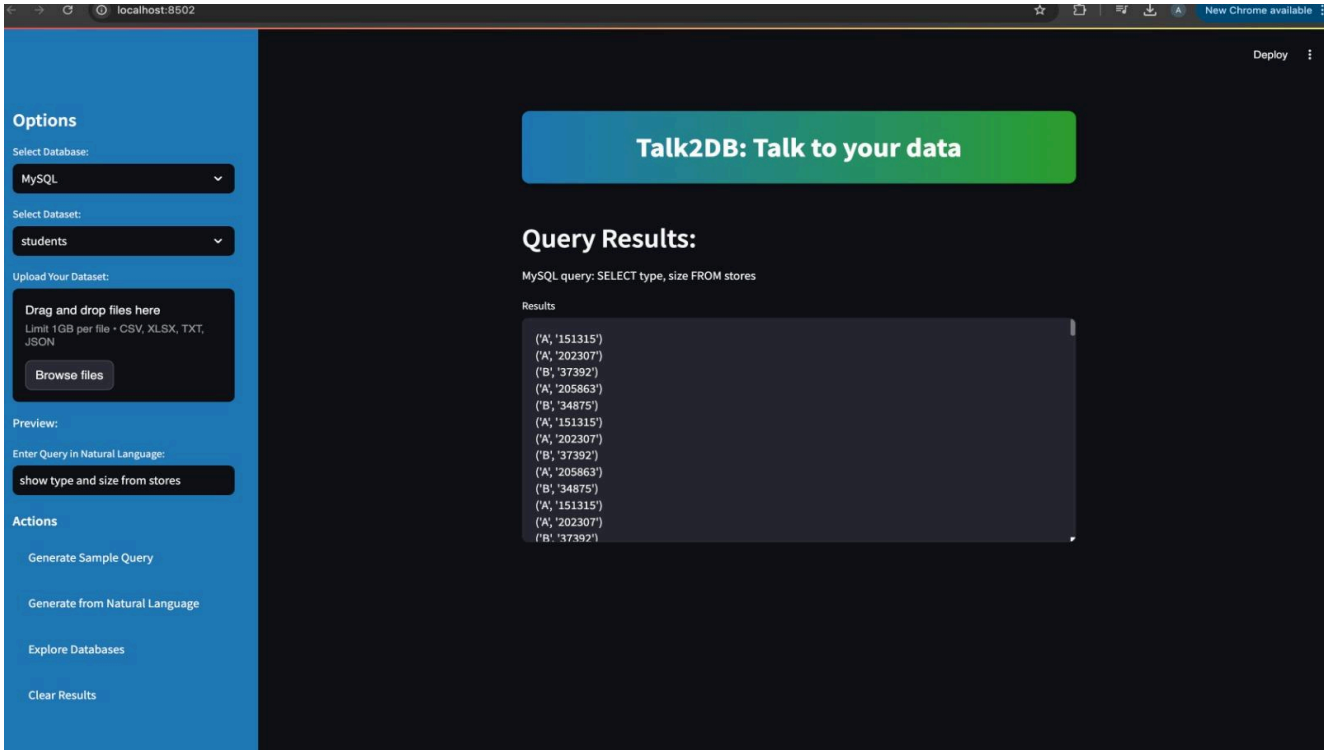
## Implementation Screenshots

(Include a few screenshots showing key functionality such as database exploration, query generation, and result display.)

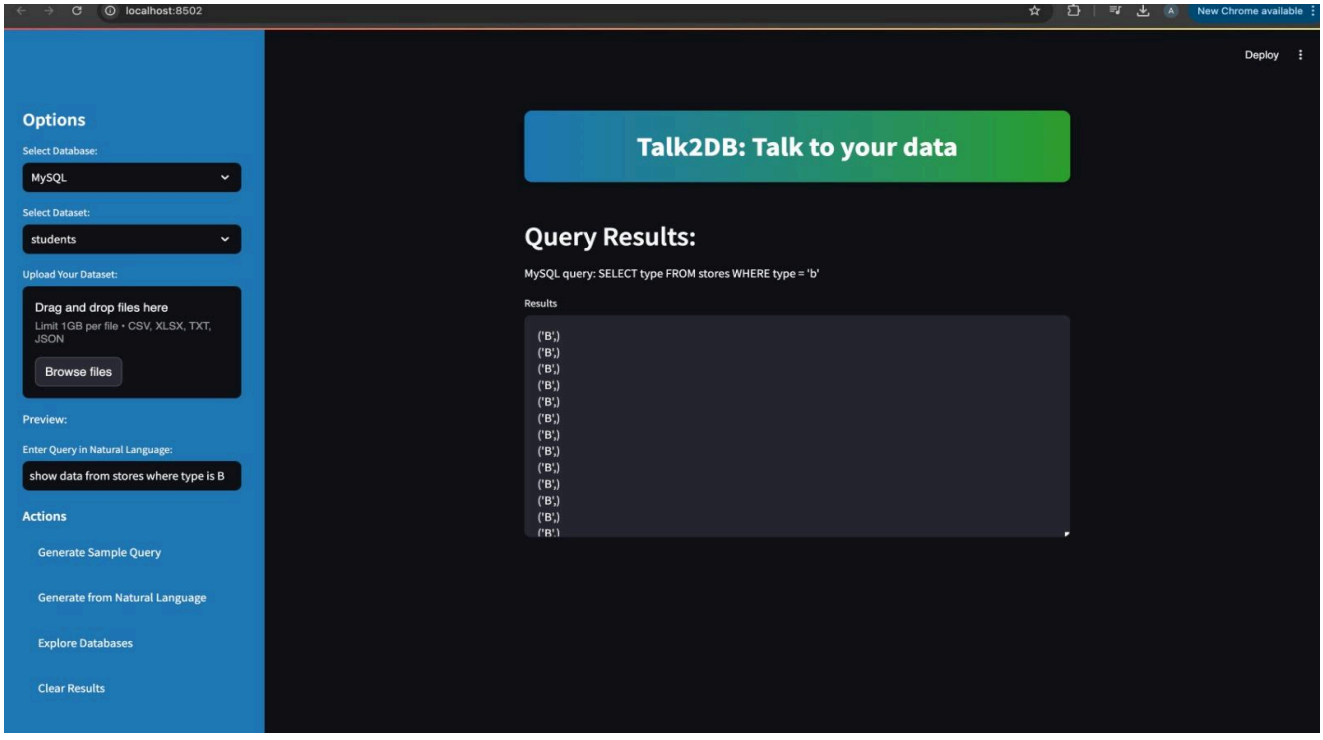
## MySQL:



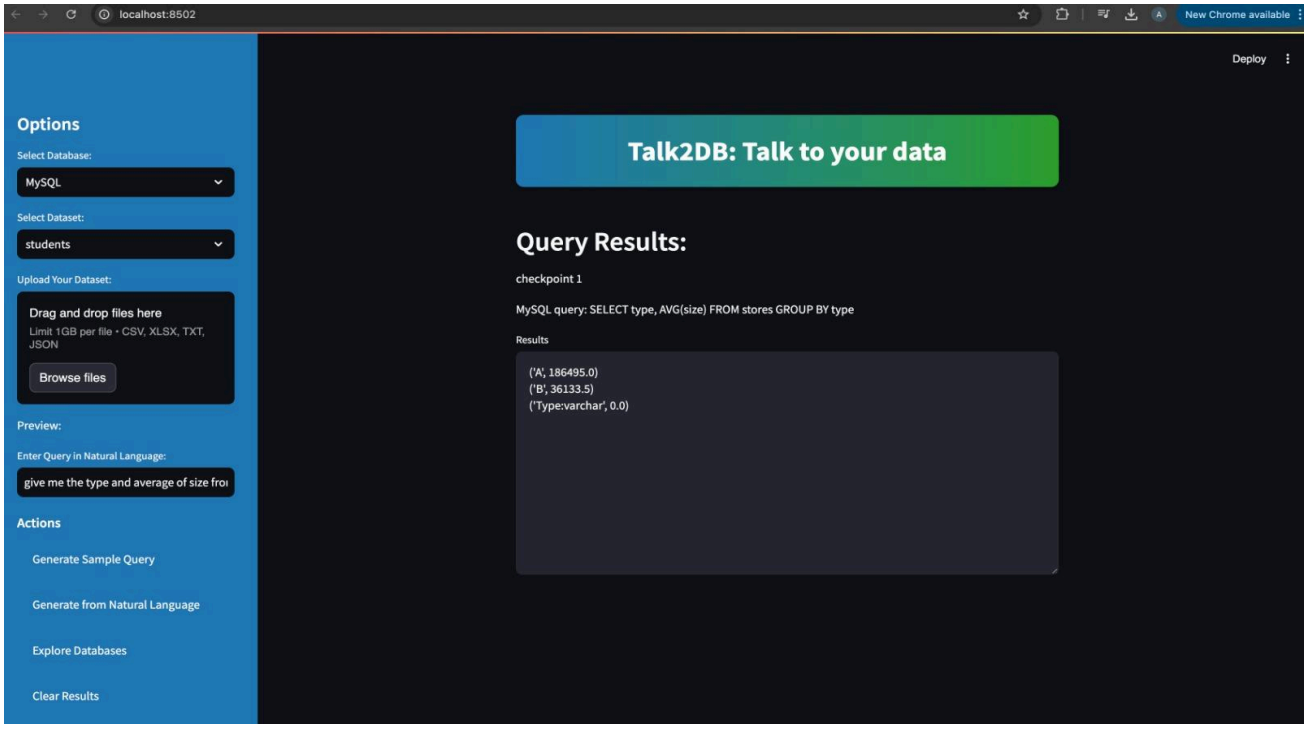

**Natural language query : Show all data from stores**



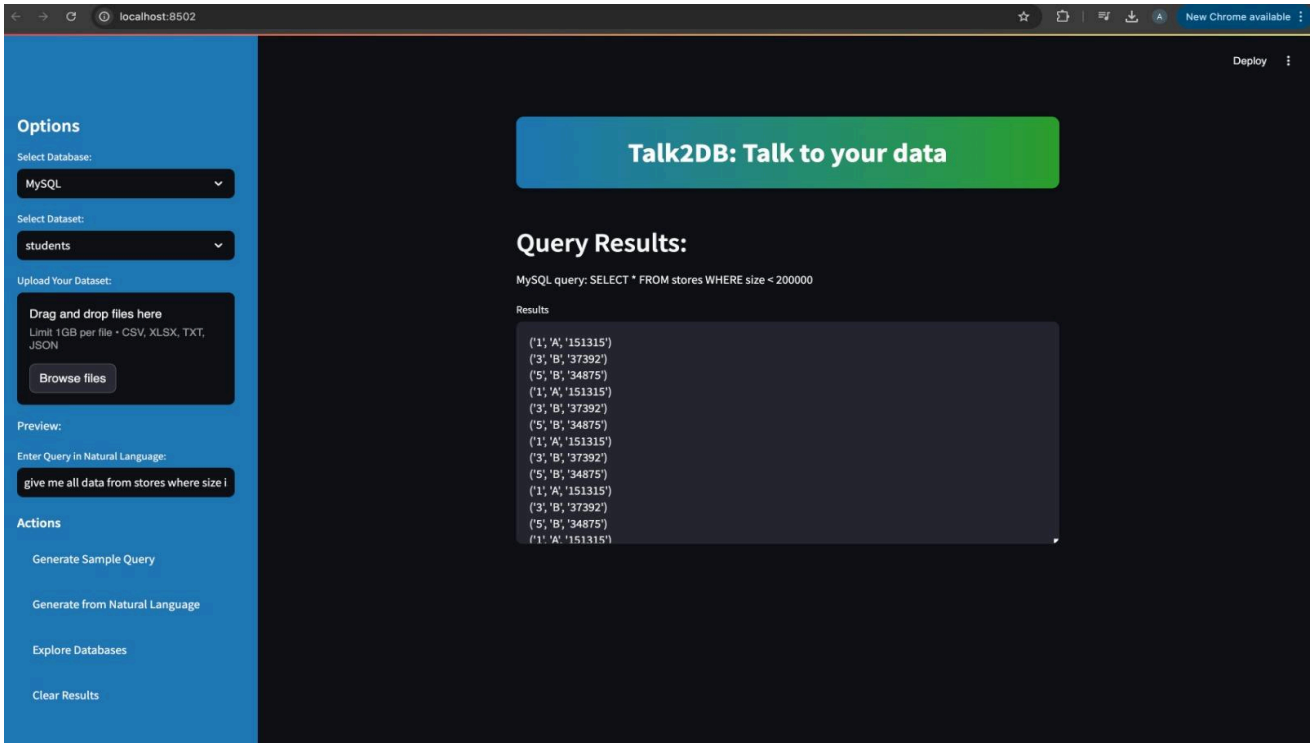

**Natural language query : Show type and size from stores**




**Natural language query : Show data from stores where type is b**



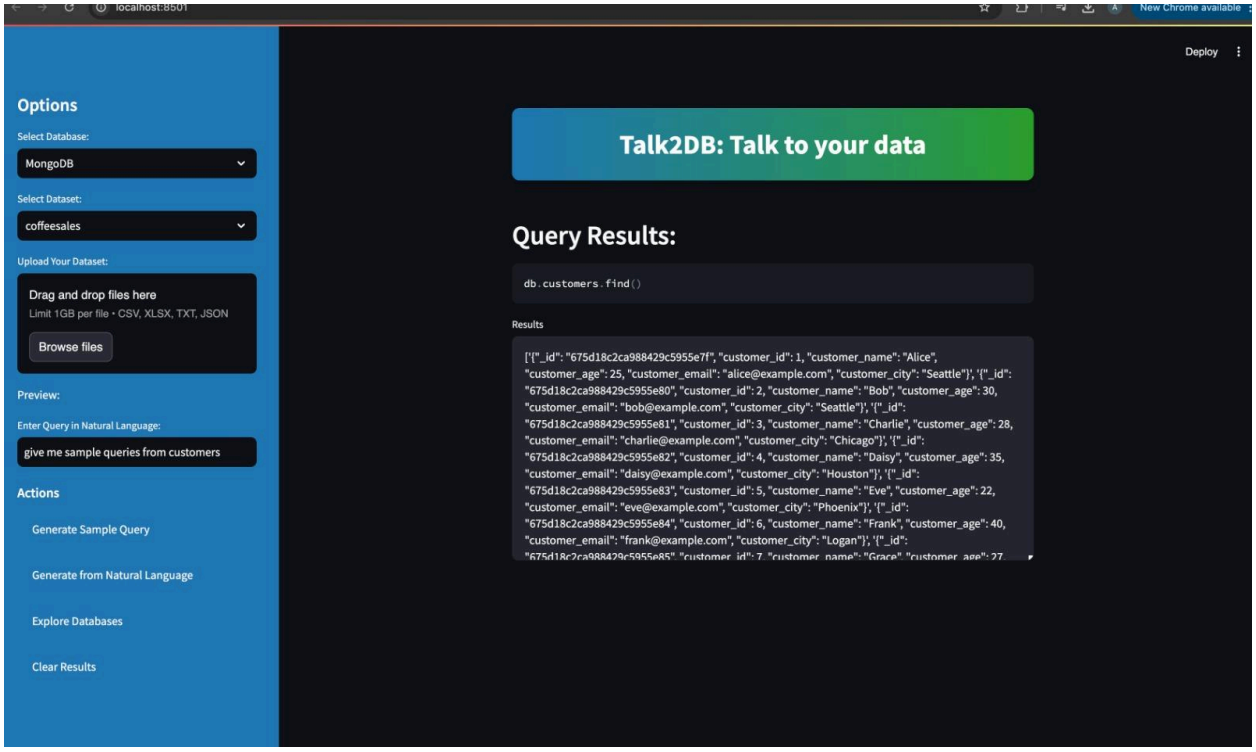

**Natural language query : give me the type and average of size from stores and group it by type**



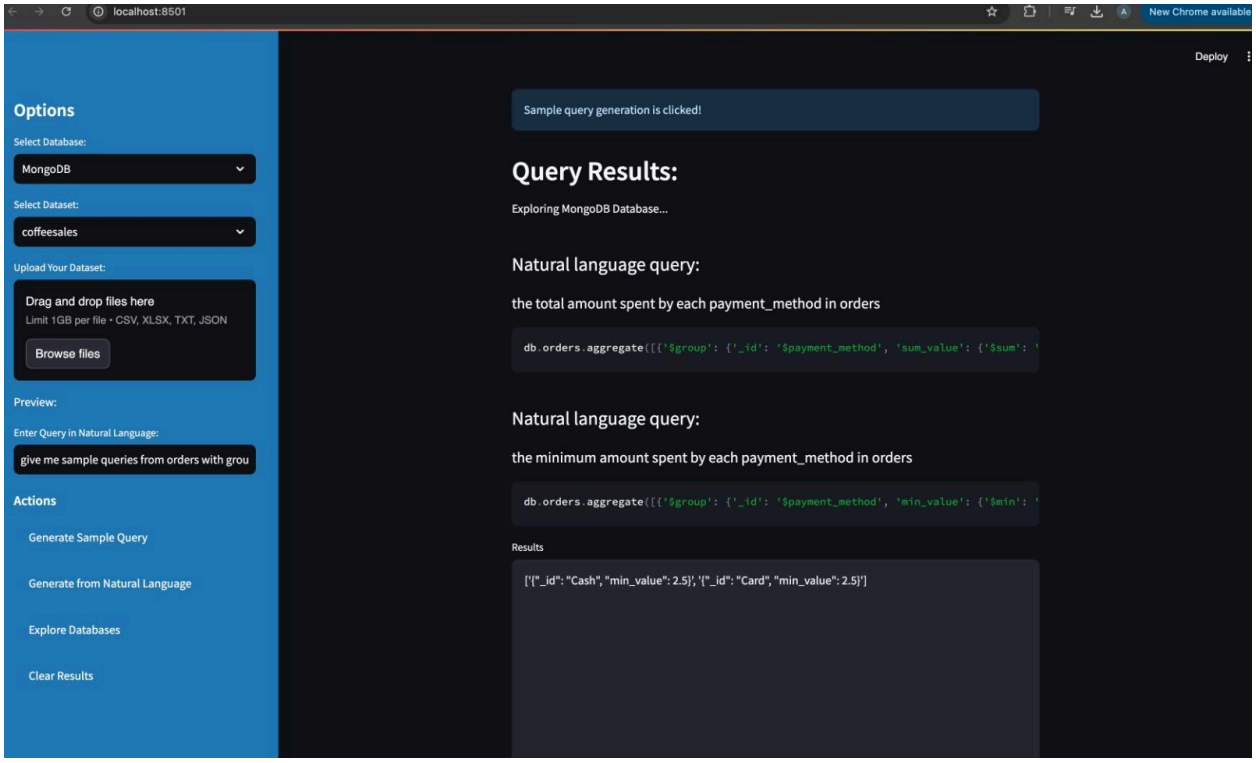

**Natural language query : give me all data from stores where size is less than 200000**

# MongoDB

## coffeesales - collections: customers, orders, products




**Natural language query: give me sample queries from customers**




**Natural language query:** give me sample queries from orders with group by

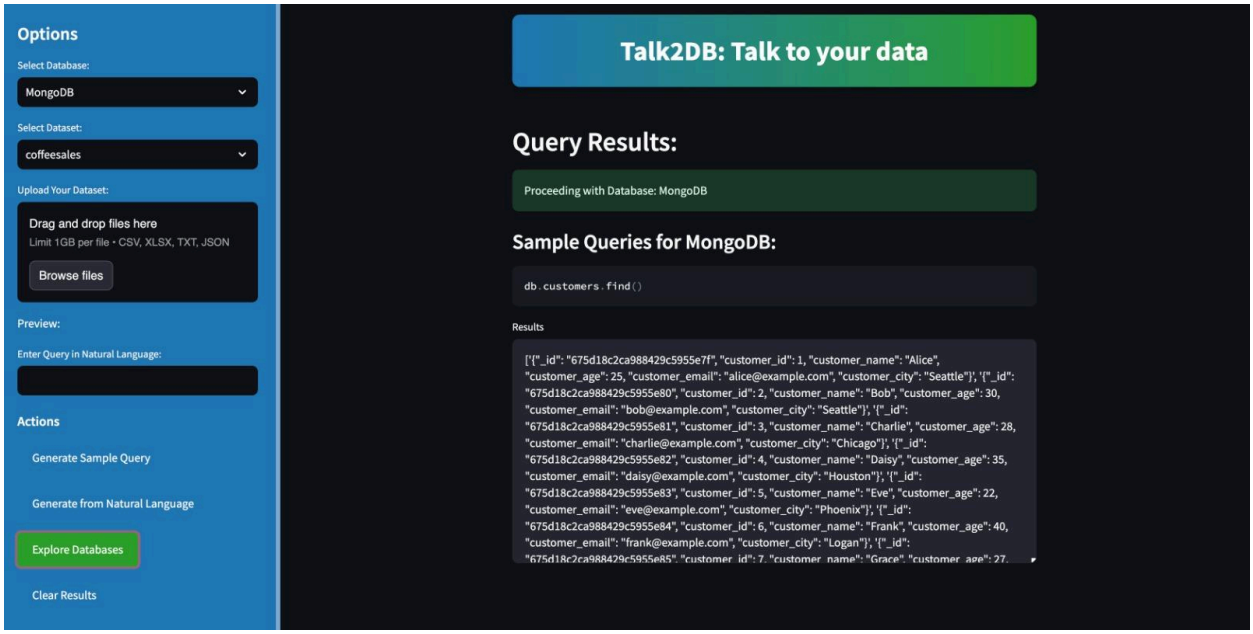
Results:

the total amount spent by each payment_method in orders

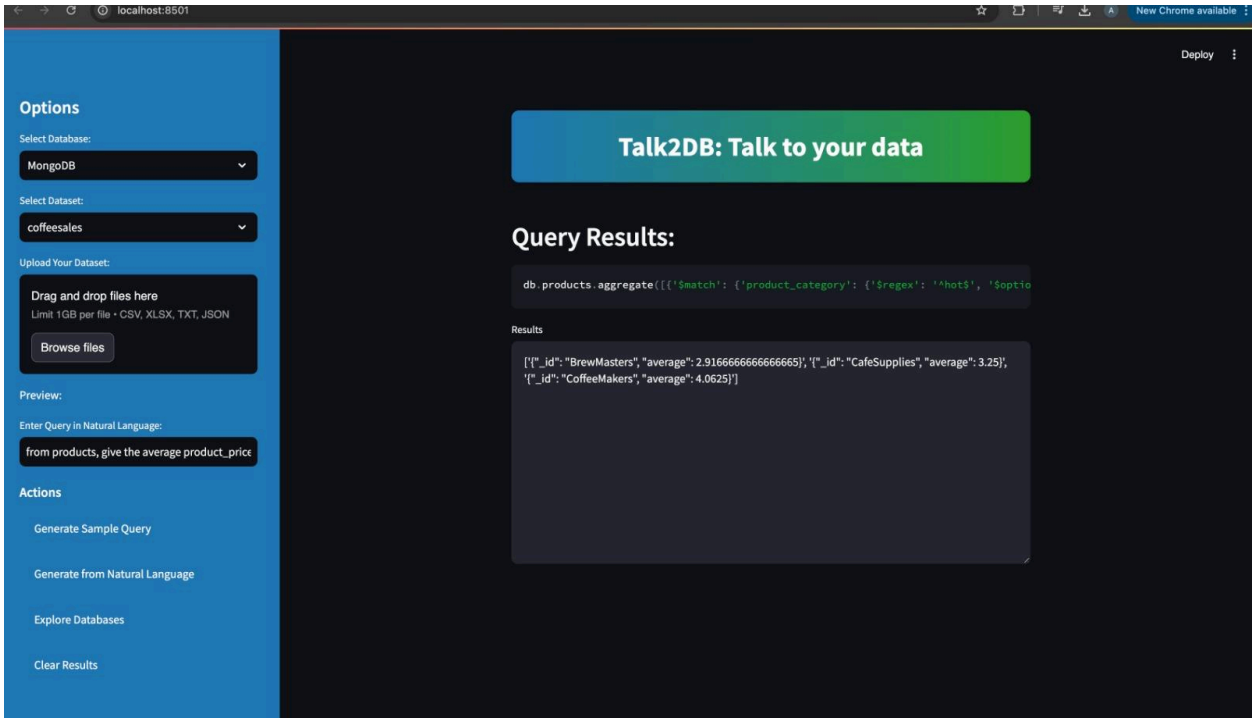
db.orders.aggregate([{'$group': {'_id': '$payment_method', 'sum_value': {'$sum': '$amount'}}}])

the minimum amount spent by each payment_method in orders

db.orders.aggregate([{'$group': {'_id': '$payment_method', 'min_value': {'$min': '$amount'}}}])

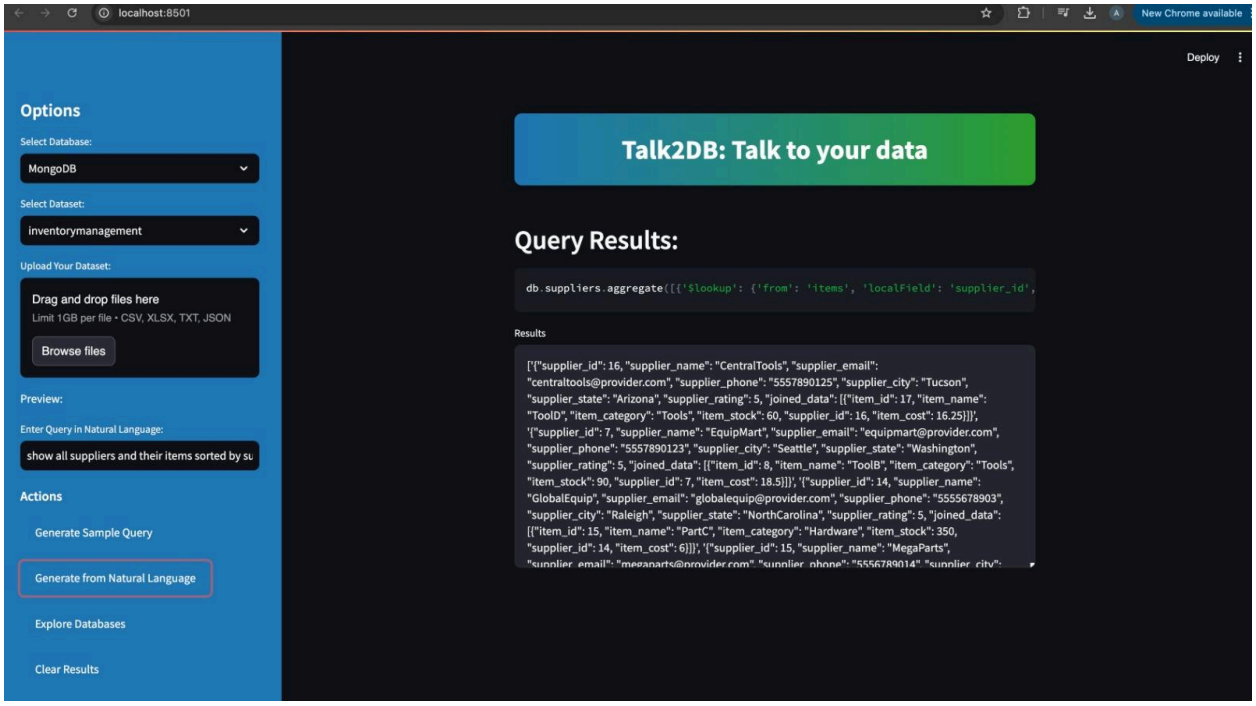



**Explore database(coffeesales - collection: customers)**




**Natural language query:** from products, give the average product_price per product_supplier sorted by product_supplier in ascending order where product_category is hot and display only those rows having average product_price greater than 2

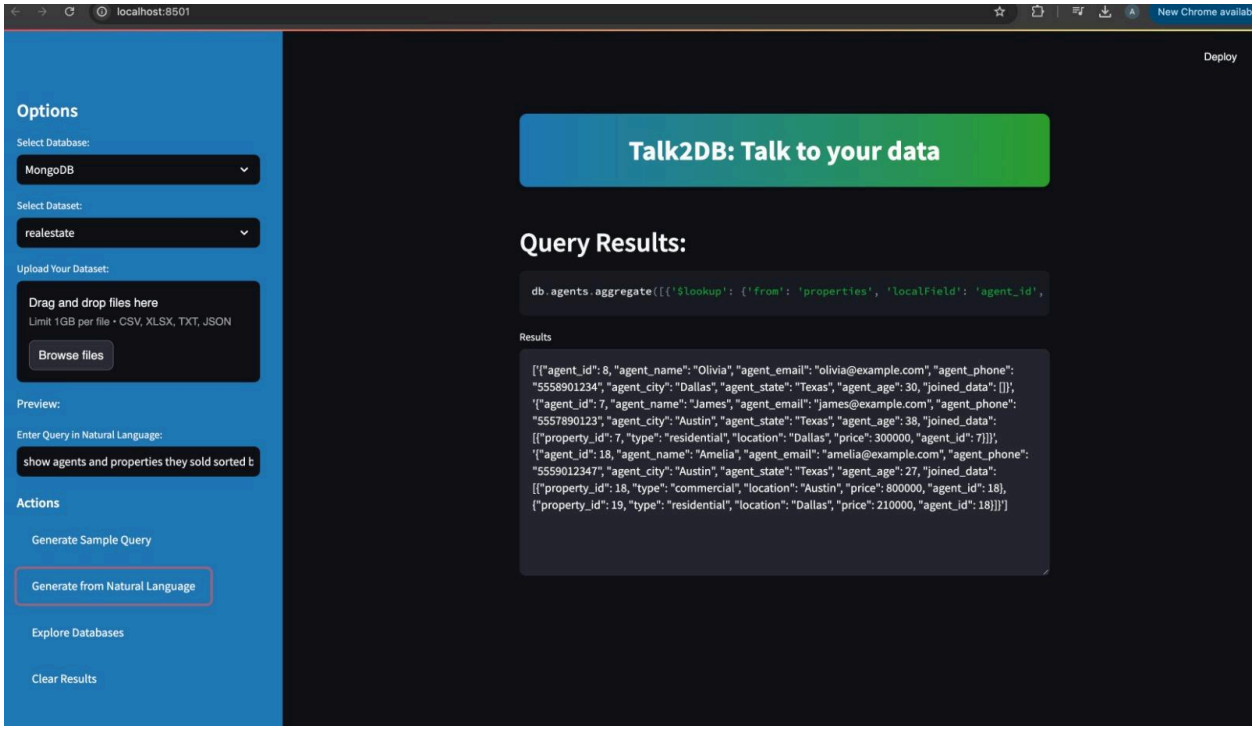
db.products.aggregate([{'$match': {'product_category': {'$regex': '^hot$', '$options': 'i'}}}, {'$group': {'_id': '$product_supplier', 'average': {'$avg': '$product_price'}}}, {'$match': {'average': {'$gt': 2}}}, {'$sort': {'_id': 1}}])




**Inventory management: collections - suppliers, items, warehouses**

**Natural language query:** show all suppliers and their items sorted by supplier_name in ascending order, where supplier_rating is 5

db.suppliers.aggregate([{'$lookup': {'from': 'items', 'localField': 'supplier_id', 'foreignField': 'supplier_id', 'as': 'joined_data'}}, {'$project': {'_id': 0, 'joined_data._id': 0}}, {'$match': {'supplier_rating': {'$eq': 5.0}}}, {'$sort': {'supplier_name': 1}}])




**Real estate: collections - properties, clients, agents**
**Natural language query:** show agents and properties they sold sorted by agent_name in descending order, where agent_state is Texas

db.agents.aggregate([{'$lookup': {'from': 'properties', 'localField': 'agent_id', 'foreignField': 'agent_id', 'as': 'joined_data'}}, {'$project': {'_id': 0, 'joined_data._id': 0}}, {'$match': {'agent_state': {'$regex': 'texas', '$options': 'i'}}}, {'$sort': {'agent_name': -1}}])

## Learning Outcomes

- NLP and Query Mapping: The integration of `nltk` and `re` for processing natural language queries and mapping them to SQL or MongoDB queries provided practical experience in applying NLP techniques for database query generation.

- SQL and NoSQL Integration: Working with both MySQL and MongoDB allowed us to understand the different approaches required for querying structured versus unstructured data, ensuring that the system could handle both types of databases.

- Web Development: Developing the front-end interface using Streamlit enhanced our understanding of user interaction design, especially in the context of dynamic data-driven applications.

## Challenges Faced

**1. NLP Query Parsing**: Accurately interpreting user inputs posed a significant challenge, particularly in recognizing different ways users phrase their queries. To tackle this, we refined the NLP model by adjusting the tokenization process and identifying patterns in query structures.

**2. Compatibility Between SQL and NoSQL**: Ensuring that the query generation system could handle both MySQL and MongoDB queries required adapting the templates to account for differences in query syntax. This was particularly challenging with aggregations and joins, which are handled differently in SQL and NoSQL environments.

**3. Pattern Recognition Accuracy**: Detecting the full range of potential query constructs and keywords (like filters, aggregations, and conditions) required careful design and testing of regular expressions and NLP models to ensure robust pattern matching.

## Individual Contribution

- **Abhishek Nagare**: Handled MySQL database integration, query generation logic, and frontend integration. Abhishek ensured that the user interface was intuitive, and users could seamlessly interact with the backend to run SQL queries.

- **Poornashree Nagaraju**: Focused on MongoDB integration and handled backend tasks for MongoDB, including designing sample query generation, defining query patterns, and developing a pattern detection system. She built and optimized MongoDB queries for efficient data retrieval.

- **Diksha Pardeshi**: Worked on MySQL integration and handled query pattern detection. Diksha was responsible for implementing the NLP logic to parse and understand natural language queries, as well as mapping these queries to SQL templates.

## Conclusion

The Talk2DB system successfully provides a user-friendly interface for querying both SQL and NoSQL databases using natural language. This allows users without a technical background to explore and analyze data, making database interaction more accessible. The system's core functionality, including NLP processing, query generation, and integration with MySQL and MongoDB, has been implemented and is operational.

## Future Scope

- **Enhance NLP Model**: Future work will focus on improving the NLP model to handle more complex queries and improve the accuracy of pattern recognition.
- **Advanced UI Features**: Additional features such as query editing, error handling, and improved result formatting will be added to enhance user experience.
- **Expanded Database Support**: The system will support additional databases (e.g., PostgreSQL, SQLite) and more complex query templates (e.g., nested queries, subqueries) for greater versatility.

## Code and Link:

Github link for codebase