A red text on a white background

Description automatically generated

|  |  |
| --- | --- |
| Project Title | **SQL Query Generator** |
| Project Estimated Start Date | **27/09/2024** |
| Project Estimated End Date | **06/11/2024** |
| Candidate Name | Roshini Panjwani |
| Mentor Name | Ankan Chowdhury |

**GEBS GET Training Project Document**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Doc Version | Submitted Date | Reviewed By and Date | Comment/Remarks |
|  |  |  |  |

Reviewer(s)

|  |  |
| --- | --- |
| Name | Title |
|  |  |

Approver(s)

|  |  |
| --- | --- |
| Name | Title |
|  |  |

Document Reference(s)

|  |
| --- |
| Releated Documents : |
| **OpenAI API** [**https://platform.openai.com/docs/guides/text-generation**](https://platform.openai.com/docs/guides/text-generation)  **Microsoft Azure OpenAI API :** [**https://learn.microsoft.com/en-us/azure/ai-services/openai/overview**](https://learn.microsoft.com/en-us/azure/ai-services/openai/overview)  **Django :** [**https://www.w3schools.com/django/**](https://www.w3schools.com/django/) |

1.1 Introduction

In today’s data-driven world, efficient SQL (Structured Query Language) is vital for managing and querying databases. SQL developers often struggle with crafting optimized queries that ensure performance and accuracy, which can be both time-consuming and error prone. To streamline this, an SQL Query Generator Application integrating the Azure OpenAI API can generate, explain, and fix SQL queries based on user inputs. This application offers a user-friendly interface to automate query creation, explanation and correction reducing manual effort and boosting developer productivity.

**Objective:**

The primary objective of this application is to assist SQL developers in:

* Generating well-structured SQL queries based on their input or file uploads, such as in Excel,.CSV file.
* Explaining SQL queries to provide insights into the query's operations and structure.
* Fixing invalid SQL queries, automatically correcting errors

**Scope/Goal:**

1. User-Friendly Interface: A clean, intuitive interface where users can easily input their data, upload files and select the type of action (generate, explain or fix the query).
2. Query Generation: The core functionality of generating SQL queries based on user inputs or file uploads.
3. Query Explanation: Providing detailed explanations of SQL queries for better understanding and learning.
4. Fixing Invalid Queries: Correcting any errors in user-input SQL queries and generating a fixed query.
5. Error Handling: Robust error handling ensuring smooth operation without requiring users to debug the system themselves.
6. Chat History: The application will offer users the ability to view the generated queries through a time-stamped chat history.
7. File Uploads: Users can upload Excel files or CSV files, from which the application will generate queries based on the data provided.

These users will benefit from the application’s ability to combine query generation and error handling into one seamless tool without requiring extensive technical expertise. The application will make it easier for both technical and non-technical users to efficiently generate and improve SQL queries, reducing the need for deep knowledge of SQL or database performance tuning.

1.2 Technology Stack

Software Requirements:

**Frontend**: Django

**Backend**: Python ,Django RestApI Framework, Azure OpenAI API, Pandas

**Database**: SQLite

1.3 Glossary or Terminology

1. Django ORM (Object-Relational Mapping): A system that lets you interact with your database by writing Python code instead of raw SQL, making data queries simpler and more intuitive.
2. DRF (Django Rest Framework): An extension of Django specifically designed for building APIs, offering tools like serializers, views and routers to simplify API development.
3. Serializers: Used in Django Rest Framework (DRF) to convert complex data types to JSON, allowing the backend to send data in a format the frontend can use.
4. CSRF (Cross-Site Request Forgery): A security measure Django implements to prevent unauthorized actions on your application by requiring a CSRF token for form submissions.

1.4 High Level Design

A diagram of a data flow

Description automatically generated

1.5 Programming Standards

Programming Standards Followed:

1. **Separation of Frontend and Backend:** The frontend and backend are kept separate to maintain modularity and improve maintainability. The **frontend** is developed using HTML and CSS to ensure a clean and responsive user interface.

The **backend** is built using **Django Rest Framework (DRF)**, which serves as the API provider for the frontend. This separation ensures that the presentation layer and data processing logic are decoupled, promoting a clean architecture.

**Code Snippet:**

**Frontend:**

A screen shot of a computer program

Description automatically generated

**Backend**

: A screen shot of a computer screen

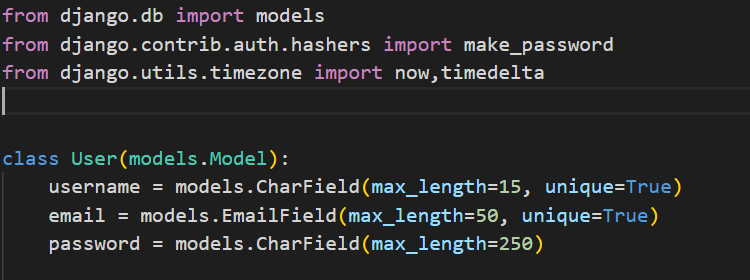
Description automatically generated

**2.** **MVC Pattern (Model-View-Controller):** Django follows the MVC (Model-View-Controller) design pattern (in Django it’s called MVT(Model-View-Template). This ensures separation of concerns by properly dividing data (Model), user interface (Template), and logic (View) in the project.

* + **Models** handle the database schema.
  + **Views** handle business logic and connect models to templates.
  + **Templates** handle the frontend presentation.

**Code Snippet:**

Model:



Views:

A screen shot of a computer program

Description automatically generated

Template:

A computer screen with many colorful text

Description automatically generated

**3.Function Based Views and Class Based Views:** In the frontend, function-based views are used for simplicity, directly handling user actions through HTML and JavaScript, which makes it easy to customize and manage the user interface. Class-based views are used in the backend with Django Rest Framework to keep the code organized, reusable, and scalable, as class-based views offer built-in methods for handling tasks like data retrieval and creation.

**Code Snippet:**

**Function Based Views:**

A computer screen shot of text

Description automatically generated

**Class Based Views:**

A computer screen shot of a program code

Description automatically generated

**4.** **Validation and Error Handling:**

Form Validations: User inputs such as unique username, valid email, password strength, and OTP validation are implemented using Django’s built-in validators and custom validation logic.

A screenshot of a computer program

Description automatically generated

File Validations: For file uploads (CSV/Excel), the system checks file size limits (under 5MB), correct formatting, and prevents corrupted files from being processed.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Error Handling: Proper error messages and feedback are provided for invalid inputs, authentication failures, and API errors, ensuring the user understands the issue.

A screen shot of a computer program

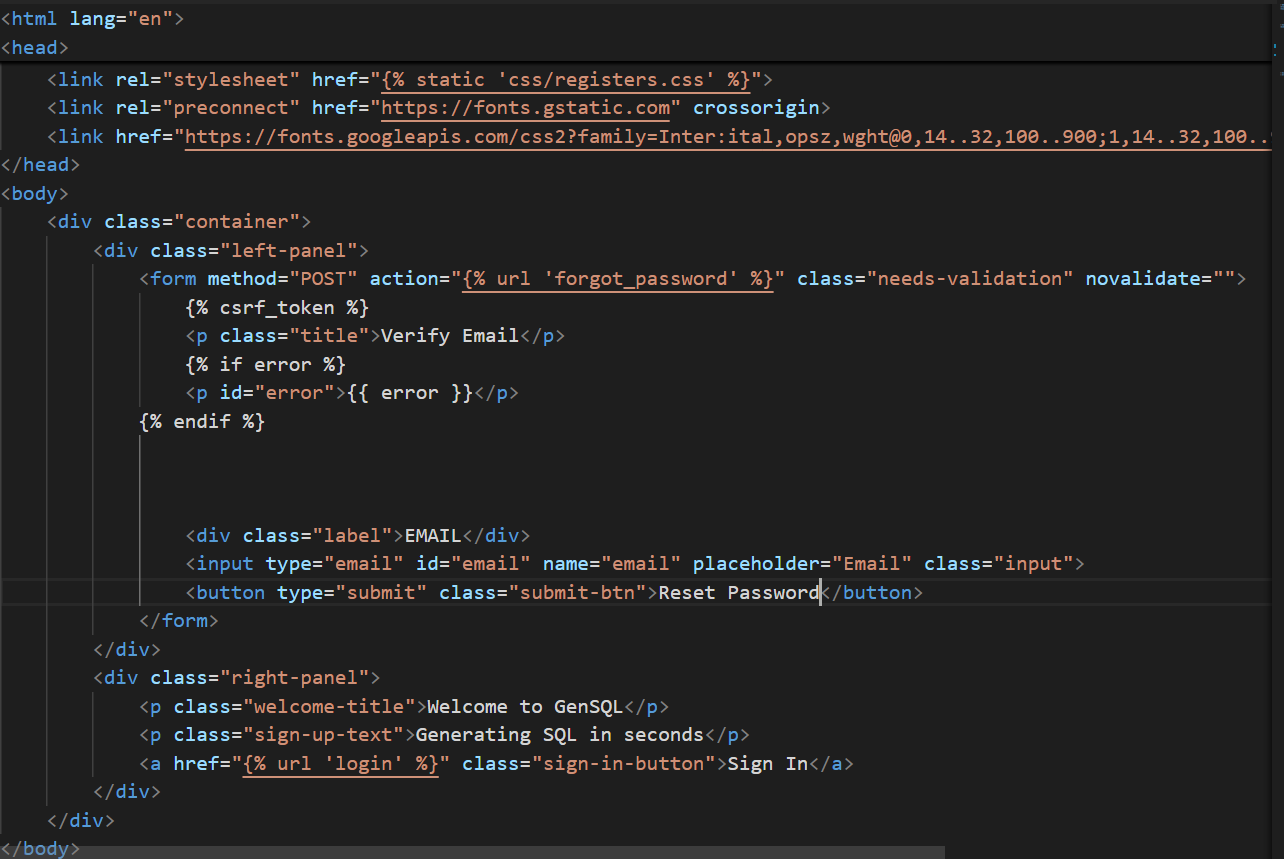
Description automatically generated

A screen shot of a computer program

Description automatically generated

**5.Security Best Practices:**

* CSRF Protection: The project includes CSRF tokens for forms to mitigate cross-site request forgery attacks.



* Password Security: Passwords are stored in an encrypted format using Django’s hashing system, and password strength is enforced by requiring at least 8 characters, with one uppercase, one lowercase, one special character, and one numeric value.

A screen shot of a computer program

Description automatically generated

* Authentication: Authentication flows include OTP-based verification for both registration and password recovery, ensuring secure access and reset processes.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

* Rockwell’s Azure OpenAI models are maintained within the Rokfusion Firewall protection

The model governance, management and security policies are governed by the policies of CISO and Rockwell's AI Governance council. Due to these infrastructure steps taken by Rockwell, any possibility of data leakage or backdoor leak is prevented.

1.6 Components

Project Workflow:

Wireframe/UI:<https://drive.google.com/file/d/1hw6DIzb89bKFi-uvTgAFjqKR6gTSAHCH/view?usp=drive_link>

1. User Registration and Login:

* Register: New users create an account by providing necessary details like username, email, and password. This information is securely stored in the user database. Users receive an OTP verification on their registered email to complete the process.
* Login: Existing users log in with their credentials. If authentication is successful, users are redirected to the Home Page where they can begin generating SQL queries.
* Validations: The registration and login process includes validations such as unique username, password validation, email checks, and numeric-only OTP validation.

1. Input Data:

* Text Input: On the Home Page, users can input a SQL query request directly into a text field. This is a compulsory input to generate a response.
* File Upload: Alternatively, users can upload data files in CSV or Excel (.xlsx,.xls formats. Uploaded files are parsed to extract table structures or other relevant data that will be used to generate SQL responses.

1. Action Selection: A dropdown menu allows users to select the type of action they want to perform:

* Generate – Generate an SQL query from user input.
* Explain – Provide an explanation of a given SQL query.
* Fix – Correct any errors in an SQL query and provide a fixed query.

1. Query Generation/Explanation/Fixing:

* Triggering the Process: Once the input is provided (through text or file upload) and the desired action is selected from the dropdown menu, the user clicks the Generate Response button. This sends a POST request to the backend.
* Backend Processing: The backend (built with Django Rest Framework) processes the input and forwards the request to the Azure OpenAI API, which performs the specified action (generate, explain, or fix).

1. Execution and Database Interaction:

The OpenAI API returns the response (query, explanation, or corrected query) to the backend. The backend also interacts with the SQLite database to save the input and the corresponding generated result for future reference.

1. Returning the Result to the Frontend:

Once the SQL response (or explanation/fix) is generated and saved, the backend sends it back to the frontend. The user sees the generated result displayed on the interface.

1. Chat History:

Users can view a history of their interactions, including input queries and the generated results. Each interaction is timestamped and includes details of the action performed (generate, explain, or fix) and the SQL server selected.

1. Error Handling:

The application implements comprehensive error handling mechanisms:

* Invalid Inputs: If users provide incorrect or malformed inputs (e.g., corrupted files or invalid queries), the system returns appropriate error messages.
* API Errors: If any issues arise with the Azure OpenAI API or during query generation, error messages are displayed on the frontend, ensuring that users receive clear and informative feedback about the issue.

1.7 Pre-requisite

1. Basic Knowledge of SQL: Users should have basic knowledge of Structured Query Language (SQL) and how to interact with databases. They should be familiar with SQL syntax such as WHERE, GROUP BY, JOIN, and common functions like (COUNT (), SUM () etc ) in order to better understand the generated response.
2. Understanding of Databases: Users should be familiar with various databases such as MSSQL, MYSQL, POSTGRESQL.
3. CSV/Excel Data Format: Users should understand the structure of CSV or Excel files if they plan to upload files, ensuring that the headers and data rows are properly formatted. Files should be free of errors, as corrupted or incorrectly structured files may be rejected by the system.
4. Proficiency with Natural Language Inputs: The project utilizes the Azure OpenAI GPT model so the users need to understand how to formulate natural language inputs that yield the most accurate and relevant SQL queries. This includes providing clear and specific instructions, mentioning necessary columns, tables, and relationships in their query request, avoiding ambiguous or overly broad statements to minimize incorrect or incomplete SQL generation.
   1. User Guide

1) Registration Process: To create a new account, follow the below steps

* Navigate to the Register page. Enter the following required details:
* Username (must be unique).
* Email Address (must be unique and valid).
* Password and Confirm Password: Both fields must match and meet the password criteria (at least 8 characters long, containing one uppercase letter, one lowercase letter, and one special character).

An OTP (One-Time Password) will be sent to the registered email address. Enter the OTP to complete the registration process. Ensure the OTP is numeric and matches the one sent to your email.

2) Login Process: For existing users, log in with the following steps

* Navigate to the Login page.
* Enter your Username and Password.
* Click Login to access the system. Ensure your credentials are correct. If not, appropriate validation errors will be displayed.

3) Forgot Password: If you've forgotten your password, go to the Forgot Password page.

* Enter your registered Email Address.
* An OTP will be sent to your email to verify your identity.
* Enter the OTP and follow the instructions to reset your password. Ensure the OTP is numeric and correct.

4) Query Input and Generation:

1. After logging in, you will be directed to the Home Page.

2. On the home page, there are two methods for inputting data:

* Query Prompt: Enter a natural language query in the prompt provided (this field is mandatory).
* File Upload: You can alternatively upload a CSV or Excel file with file size limit less than or equal to 5MB.Ensure the file is properly formatted and not corrupted. If a file is uploaded but not in the correct format (or if corrupted), an error message will be triggered.

3. Action Selection: Use the dropdown menu to select the action you want to perform

* Generate: Generate an SQL query based on input.
* Explain: Explain the structure and logic of the provided SQL query.
* Fix: Fix any errors in the provided SQL query.

4. Generate Response:

After inputting your data (either through the query prompt or file upload), click the Generate Response button to generate the SQL query (or explain or fix it, depending on your action selection).

5) Chat History and Timestamping:

* After generating a query, the system will save both the user input and the generated SQL query in the Chat History.
* Each query and response will be time-stamped for future reference.
* Users can revisit past queries via the chat history to review or reuse them as needed.

2.Testing

2.1 Test Scenario(s)

**1) Use Case: Register a new user**

**1.Test Scenario:** Unique username

**Expected Result:** If the username already exists in the system, an error message is displayed to prompt the user to choose a different username.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**2.Test Scenario:** Unique email (email exists or not)

**Expected Result:** If the email already exists, the system displays an error message indicating that the email is already in use.

**Actual Result:**

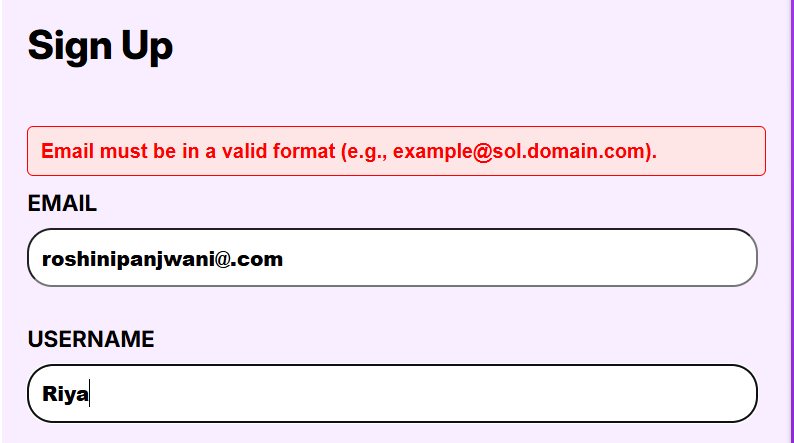
A screenshot of a login box

Description automatically generated

**3. Test Scenario:** Email format validation

**Expected Result:** If the email format is invalid, an error message is displayed, prompting the user to enter a valid email address.

**Actual Result:**



**4. Test Scenario:** Password format validation

**Expected Result**: If the password does not meet the required format (e.g., length, special character requirements), an error message is displayed.

**Actual Result:**

A screenshot of a login form

Description automatically generated

**5. Test Scenario:** OTP verification for new registration

**Expected Result:** After successful input, an OTP is sent to the user’s email for verification.

**Actual Result:**

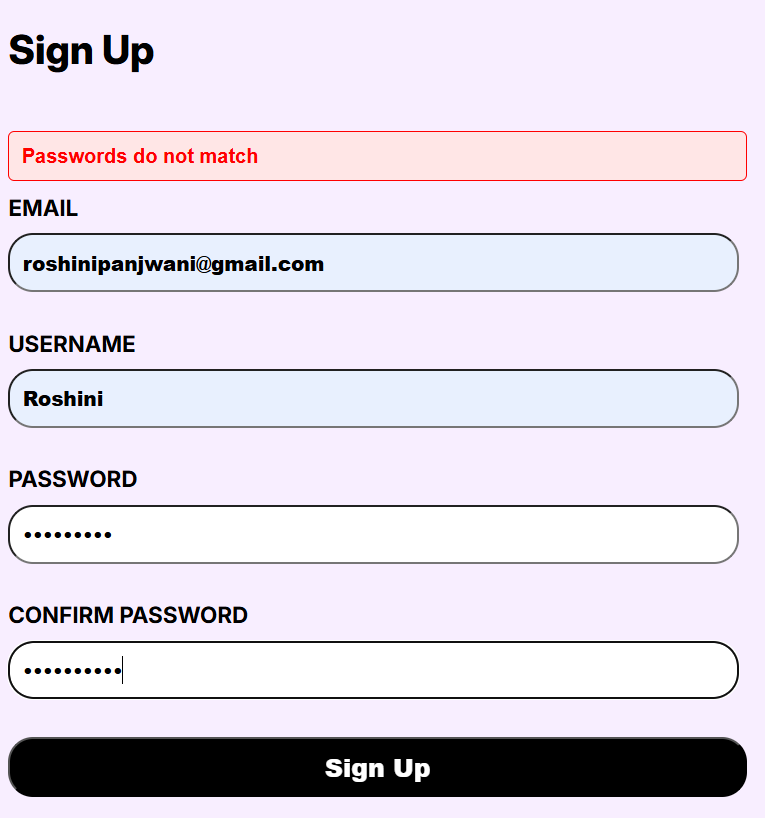
A screenshot of a computer error

Description automatically generated

**6. Test Scenario:** Password and Confirm Password match

**Expected Result:** If the password and confirm password fields do not match, an error message is displayed that password do not match.

**Actual Result:**



**7. Test Scenario:** OTP validation

**Expected Result:** If the entered OTP is incorrect, an error message is displayed. If correct, registration is completed, and user redirected to login page.

**Actual Result:**

A screenshot of a computer error

Description automatically generated

**2) Use Case: User login**

**1. Test Scenario:** Verify login with correct username and password

**Expected Result:** If both username and password are correct, the user successfully logs in and is redirected to the home page else an error message generates of invalid credentials.

**Actual Result:**

A screenshot of a login screen

Description automatically generated

**2. Test Scenario:** Non-existent user

**Expected Result:** If the user does not exist, an error message is displayed with a prompt to sign up.

**Actual Result:**

A login screen with black and white boxes

Description automatically generated

**3) Use Case: Forgot Password**

**1. Test Scenario:** Verify email existence

**Expected Result:** If the email does not exist in the system, an error message is displayed, prompting the user to register.

**Actual Result:**

A screenshot of a computer screen

Description automatically generated

**2. Test Scenario:** OTP validation for password reset

**Expected Result:** If the OTP is incorrect, an error message is displayed. If correct, the user is allowed to proceed to reset the password.

**Actual Result:**

A screenshot of a login screen

Description automatically generated

**3. Test Scenario:** New Password and Confirm Password match

**Expected Result:** If the new password and confirm password fields do not match, an error message is displayed.

**Actual Result:**

A screen shot of a login screen

Description automatically generated

**4) Use Case: Upload a file (CSV or Excel)**

**1. Test Scenario:** File format validation

**Expected Result:** If a non-CSV or non-Excel file is uploaded, an error message "Invalid file format" is displayed.

**Actual Result:**

A white background with black text

Description automatically generated

**2.Test Scenario:** Corrupted file validation

**Expected Result**: If the uploaded file is corrupted, an error message related to the error is displayed.

**Actual Result**

A purple and blue rectangle with text

Description automatically generated

**3. Test Scenario:** Successful file upload

**Expected Result :** If the file format and size are valid and if its error-free, a success message File uploaded successfully is displayed.

**Actual Result.**

A screenshot of a computer

Description automatically generated

**5) Use Case: Prompt validation**

**1. Test Scenario**: Generate query without entering a prompt

**Expected Result:** If the user clicks Generate Response button without entering a prompt, an error message “Please enter an input prompt" is displayed**.**

**Actual Result:**

A white background with black text

Description automatically generated

**6) Use Case: SQL Query Generation**

**1. Test Scenario:** Generate query for a valid SQL prompt

**Expected Result:** The query is generated based on the input prompt and displayed.

**Actual Result:**

A screenshot of a computer program

Description automatically generated

**2. Test Scenario:** Non-SQL prompt handling

**Expected Result:** If the input prompt is unrelated to SQL, a polite message "Please ask SQL-related questions" is displayed.

**Actual Result :**

A screenshot of a computer

Description automatically generated

**7) Use Case: SQL Query Explanation**

**1. Test Scenario: Selecting Explain query action with valid SQL query**

**Expected Result:** If a user selects "Explain" and provides a valid SQL query, the system explains the query.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**2. Test Scenario: Selecting Explain query action with invalid SQL query**

**Expected Result:** If a user selects "Explain" and provides a invalid SQL query, the system gives a message to provide a valid SQL query to be explained.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**3. Test Scenario: Selecting Explain query action with non-SQL related prompt**

**Expected Result:** If a user selects "Explain" and provides a non-SQL related prompt the system gives a message to provide a SQL related question to be explained.

**Actual Result :**

A screenshot of a computer

Description automatically generated

**8) Use Case: SQL Query Fixing**

**1. Test Scenario: Fix incorrect query**

**Expected Result:** If an incorrect query is provided, it is corrected by the system.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**2. Test Scenario: Fix correct query**

**Expected Result: If the query is already correct, a message "Your SQL query is already correct" is displayed.**

**Actual Result:**

A screenshot of a computer

Description automatically generated

**9) Use Case: File-based SQL Query Generation**

**1. Test Scenario: Generate query related to file content**

**Expected Result:** If a valid file is uploaded and the user requests a query related to the file’s schema, the query is generated based on the file content.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**2. Test Scenario: Unrelated SQL prompt with file uploaded**

**Expected Result:** If the prompt is unrelated to SQL, a polite message "Please ask SQL-related questions" is displayed.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**3. Test Scenario:** Query related to unsupported schema

**Expected Result:** If the prompt is SQL-related but not relevant to the uploaded file schema, a message "Please provide a relevant file" is displayed.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**10)Use Case: File-based SQL Query Explanation**

**1. Test Scenario: Explain query related to file schema**

**Expected Result:** If the query is related to the file's schema, it is explained based on the file’s structure.

**Actual Result:**

A screenshot of a computer

Description automatically generated

**2. Test Scenario: Explain query related to file schema**

**Expected Result:** If the query is not related to the file's schema, a message "Please provide a relevant file" is displayed.

**Actual Result:**

A screenshot of a computer

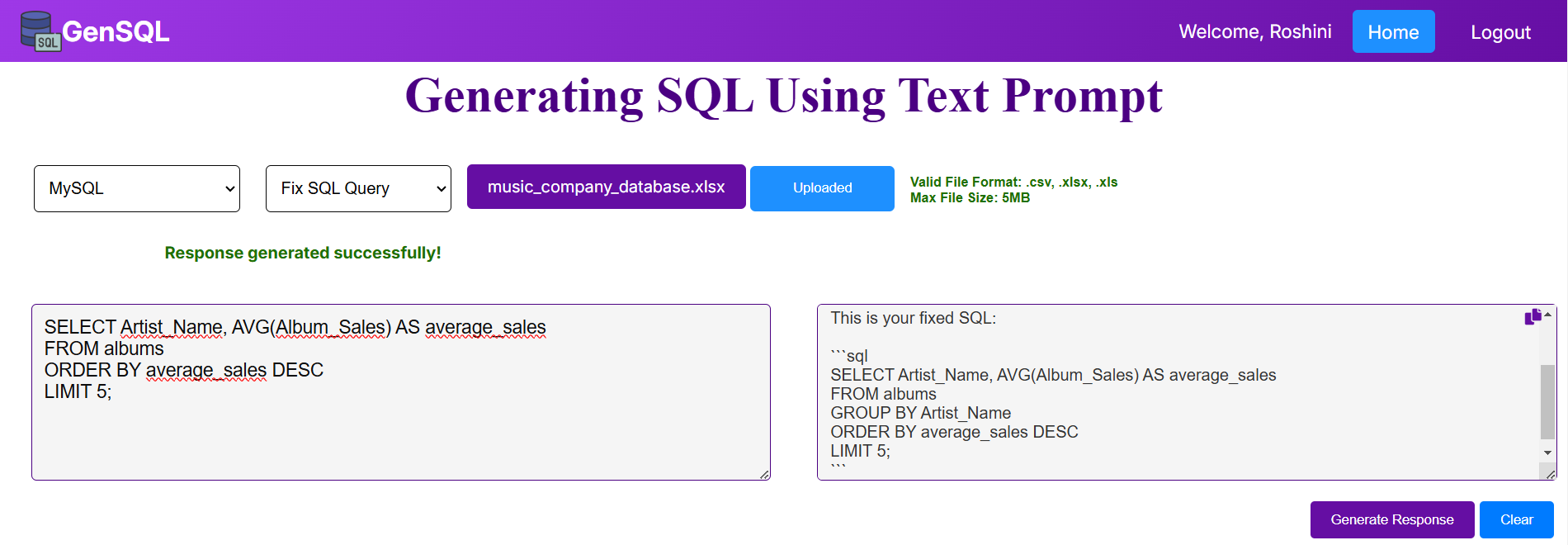
Description automatically generated

**11)Use Case: File-based SQL Query Fixing**

**1. Test Scenario: Fix query related to file schema**

**Expected Result:** If the query is incorrect and related to the file schema, it is corrected based on the file’s structure.

**Actual Result:**



**2. Test Scenario: Fix query related to file schema**

**Expected Result:** If the query is incorrect but not related to the file schema, a message is received to upload a relevant file.

A screenshot of a computer

Description automatically generated

**12) Use Case: Chat History Filtering and Deletion**

**1. Test Scenario: SQL type filtering in chat history**

**Expected Result**: When filtered by SQL type, only relevant records are displayed in the chat history.

**Actual Result**

A screenshot of a computer

Description automatically generated

**2. Test Scenario: Deleting a record from chat history**

**Expected Result:** Upon clicking Delete button the corresponding record is deleted from the chat history.

**Actual Result:**

Before clicking on delete button:

A screenshot of a computer

Description automatically generated

After clicking on delete button:

A purple and white rectangle with text

Description automatically generated