Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** IntelliSQL: Intelligent SQL Querying with LLMs Using Gemini Pro
- **Team Members:**

Name	Role
B. Sravani	FrontEnd
J. Nivya	BackEnd
Praveena Reddy	AI Integration
M. Naveena	Database
B.Jahnavi	Testing

2. Project Overview

- **Purpose:**

IntelliSQL enables users to convert natural language queries into SQL statements using Gemini Pro LLM. It reduces the complexity of SQL writing and improves data accessibility for non-technical users.

Features:

- Natural Language to SQL conversion using Gemini Pro
- Secure relational database connectivity
- Query validation and execution
- Data visualization (tables and charts)
- Query history management
- JWT-based authentication

3. Architecture

- **Frontend(React):**

Built using React.js for creating an interactive dashboard that allows users to input queries, view generated SQL, and analyze results visually.

- **Backend(Node.js & Express.js):**

Handles API requests, integrates with Gemini Pro API for NL-to-SQL conversion, validates SQL queries, and manages secure communication with databases.

Database:

MongoDB stores user credentials and query history. MySQL/PostgreSQL databases are dynamically connected for executing generated SQL queries.

4. Setup Instructions

- **Prerequisites:**

- Node.js
- MongoDB
- MySQL or PostgreSQL
- Gemini Pro API Key
- npm package manager

Installation:

1. Clone the repository.
2. Run npm install in client and server directories.
3. Configure environment variables (DB credentials, API key).
4. Start backend and frontend servers.

5. Folder Structure

- **Client:** Contains React components, pages, services, and UI assets
- **Server:** Contains routes, controllers, models, middleware, and Gemini integration logic.

6. Running the Application

- Frontend: npm start (inside client folder).

- Backend: npm start (inside server folder).

7. API Documentation

- POST /api/generate-sql – Convert natural language to SQL.
- POST /api/execute-query – Execute SQL and return results.

GET /api/history – Retrieve past queries

8. Authentication

- JWT-based authentication ensures secure access. Role-based authorization controls database query permissions.

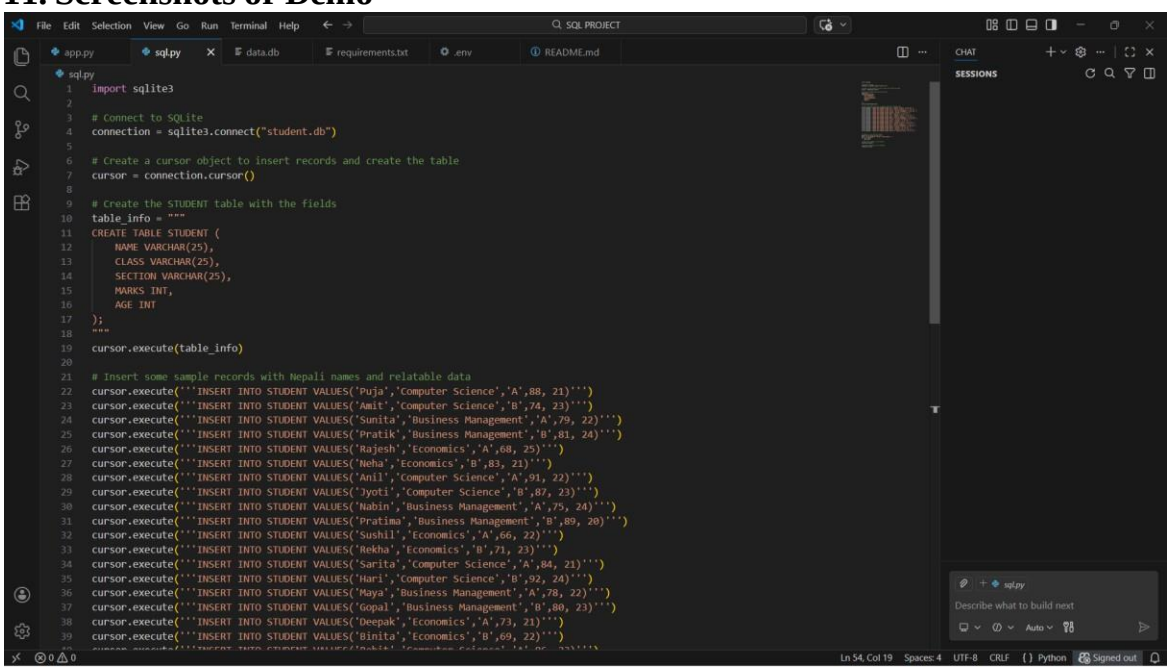
9. User Interface

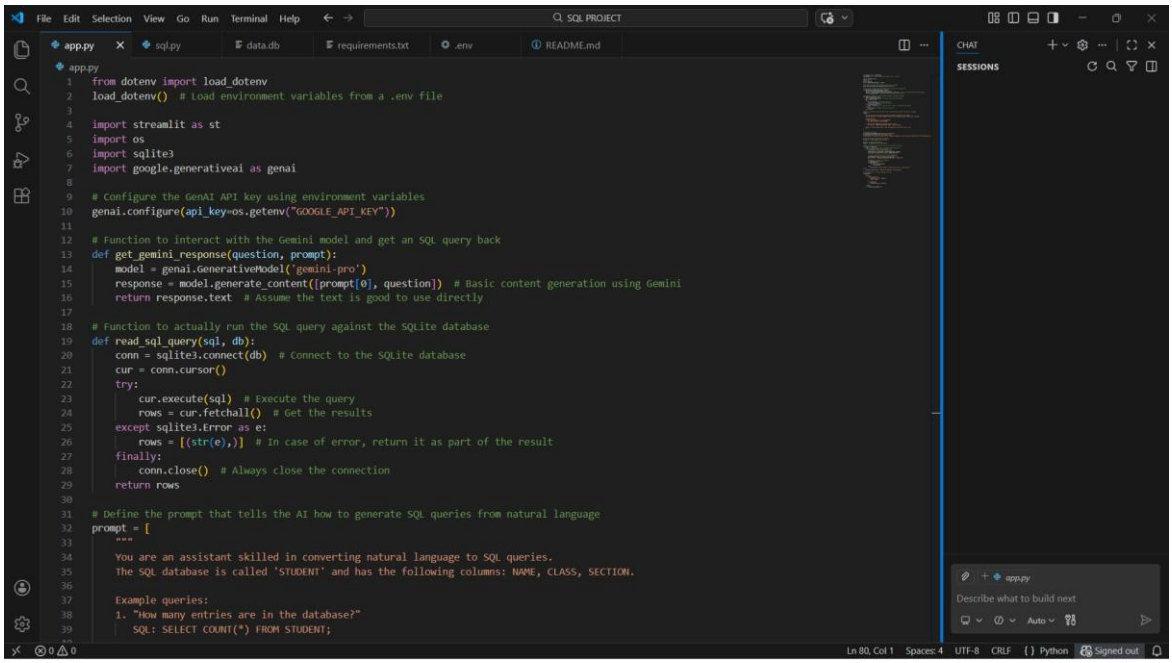
- Includes login page, dashboard for query input, SQL output display, results table, and graphical visualization section.

10. Testing

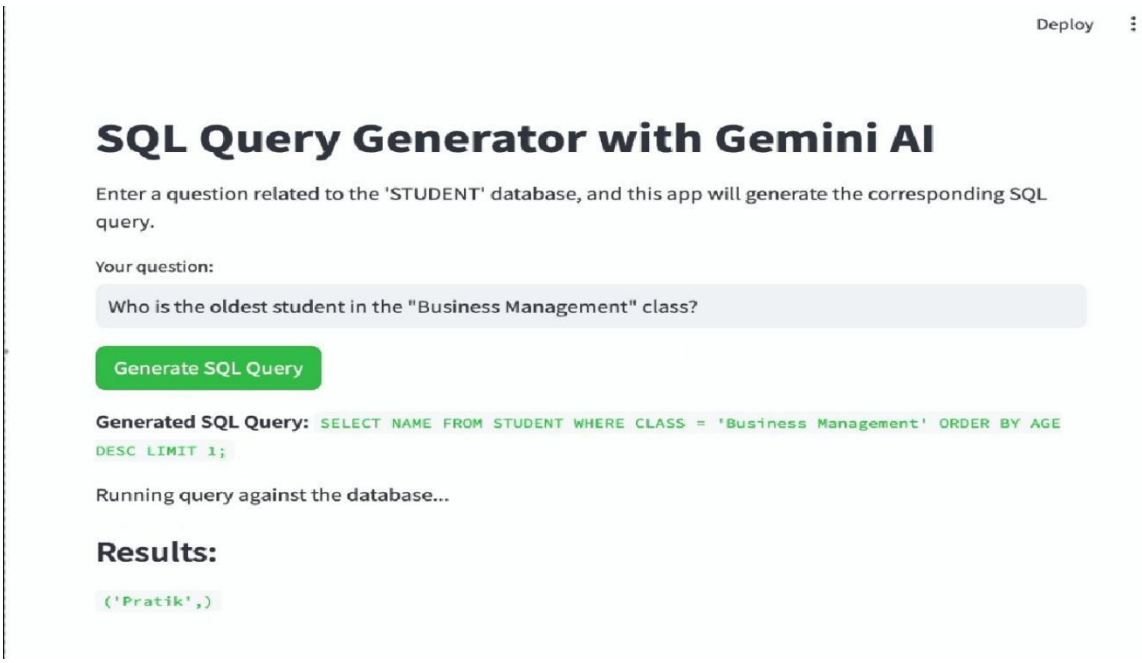
- Unit testing for APIs and integration testing for NL-to-SQL workflow. Performance testing for large datasets.

11. Screenshots or Demo





```python
from dotenv import load_dotenv
load_dotenv()  # Load environment variables from a .env file

import streamlit as st
import os
import sqlite3
import google.generativeai as genai

# Configure the GenAI API key using environment variables
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

# Function to interact with the Gemini model and get an SQL query back
def get_gemini_response(question, prompt):
    model = genai.GenerativeModel('gemini-pro')
    response = model.generate_content([prompt[0], question])  # Basic content generation using Gemini
    return response.text  # Assume the text is good to use directly

# Function to actually run the SQL query against the SQLite database
def read_sql_query(sql, db):
    conn = sqlite3.connect(db)  # Connect to the SQLite database
    cur = conn.cursor()
    try:
        cur.execute(sql)  # Execute the query
        rows = cur.fetchall()  # Get the results
    except sqlite3.Error as e:
        rows = [(str(e),)]  # In case of error, return it as part of the result
    finally:
        conn.close()  # Always close the connection
    return rows

# Define the prompt that tells the AI how to generate SQL queries from natural language
prompt = [
    """
    You are an assistant skilled in converting natural language to SQL queries.
    The SQL database is called 'STUDENT' and has the following columns: NAME, CLASS, SECTION.

    Example queries:
    1. "How many entries are in the database?"
       SQL: SELECT COUNT(*) FROM STUDENT;
```



Deploy

SQL Query Generator with Gemini AI

Enter a question related to the 'STUDENT' database, and this app will generate the corresponding SQL query.

Your question:

Who is the oldest student in the "Business Management" class?

Generate SQL Query

Generated SQL Query: `SELECT NAME FROM STUDENT WHERE CLASS = 'Business Management' ORDER BY AGE DESC LIMIT 1;`

Running query against the database...

Results:

```
('Pratik',)
```

12. Known Issues

• SQL accuracy depends on database schema clarity.

• Complex nested queries may need refinement.

• Performance varies with dataset size

13. Future Enhancements

• Multi-LLM support.

- Advanced optimization engine.
- Voice-based querying.
- BI tool integration.