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
K.Bharathi Venkata Satyaveni	FrontEnd
Kalisetti Prem Kumar	BackEnd
Karra Santosh	AI Integration
Kesani Phani Sekhar	DataBase

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

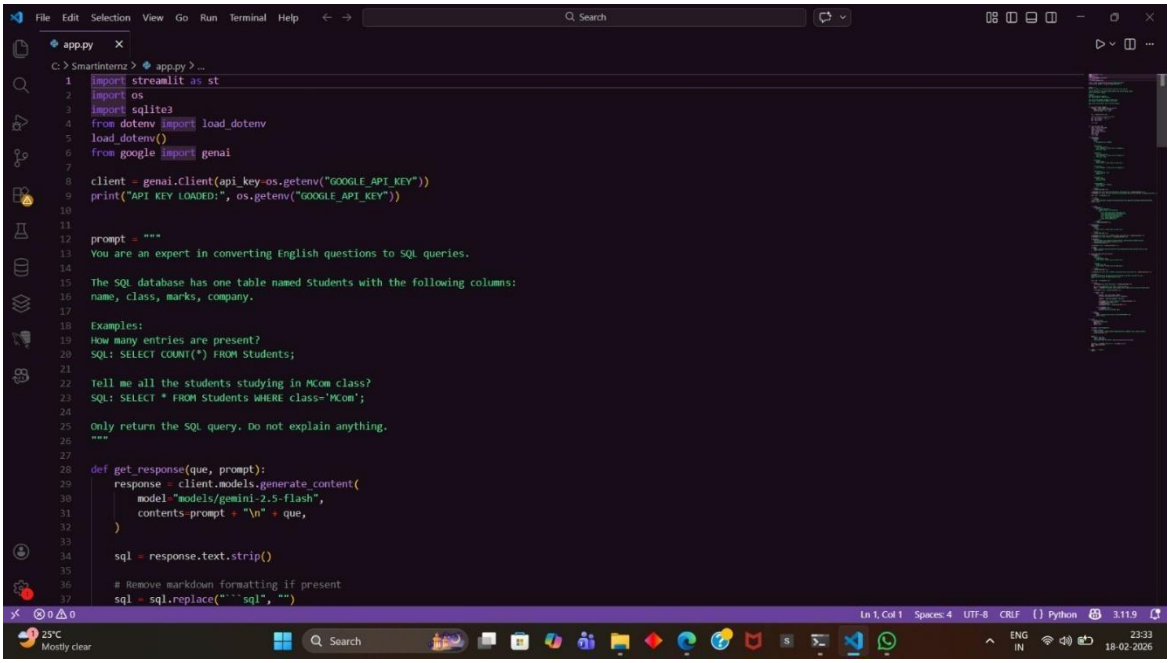
- Includes login page, dashboard for query input, SQL output display, results table, and graphical visualization section.

10. Testing

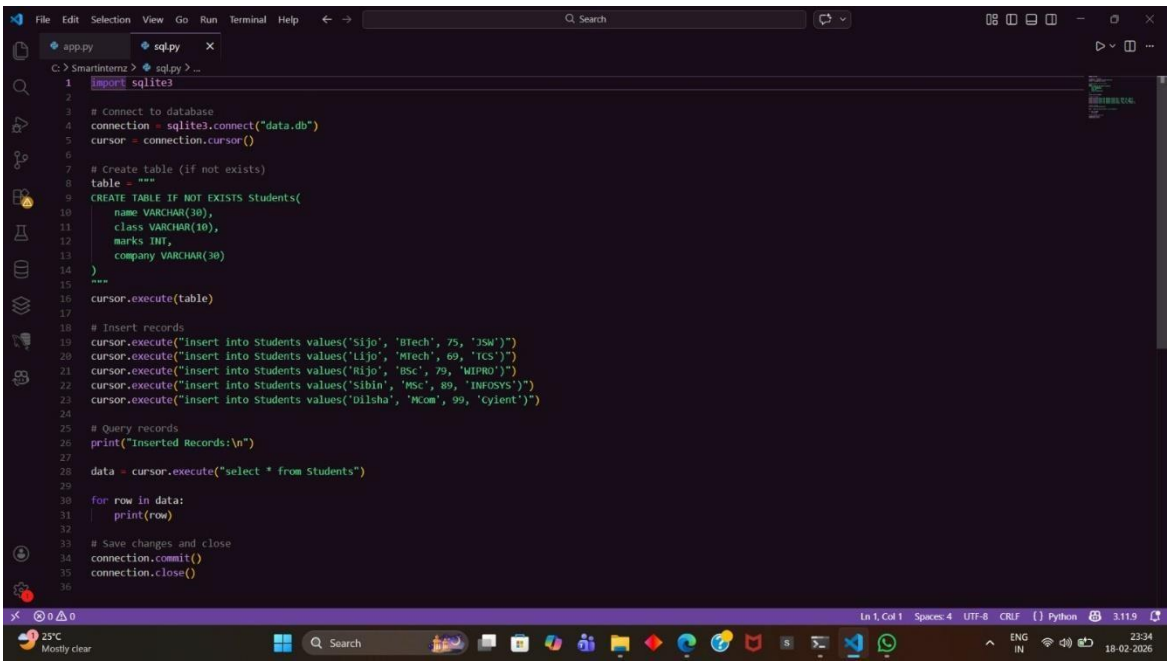
- Unit testing for APIs and integration testing for NL-to-SQL workflow.

Performance testing for large datasets.

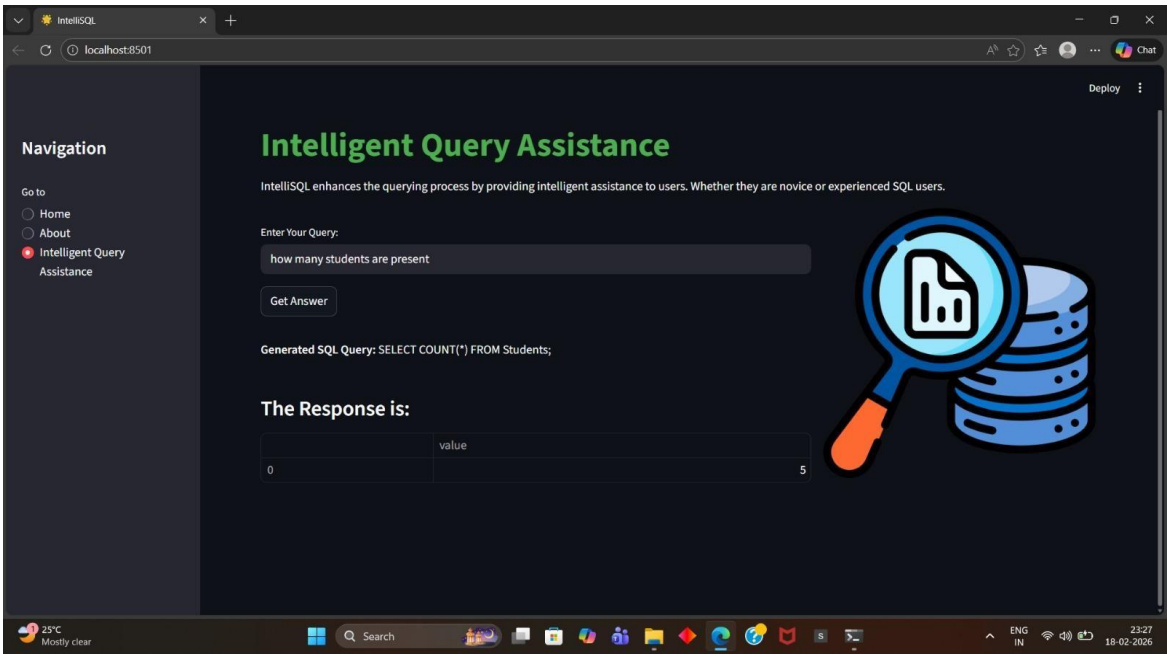
11. Screenshots or Demo



```python
import streamlit as st
import os
import sqlite3
from dotenv import load_dotenv
load_dotenv()
from google import genai

client = genai.Client(api_key=os.getenv("GOOGLE_API_KEY"))
print("API KEY LOADED:", os.getenv("GOOGLE_API_KEY"))


prompt = """
You are an expert in converting English questions to SQL queries.

The SQL database has one table named Students with the following columns:
name, class, marks, company.

Examples:
How many entries are present?
SQL: SELECT COUNT(*) FROM Students;

Tell me all the students studying in MCom class?
SQL: SELECT * FROM Students WHERE class='MCom';

Only return the SQL query. Do not explain anything.
"""

def get_response(que, prompt):
    response = client.models.generate_content(
        model="models/gemini-2.5-flash",
        contents=prompt + "\n" + que,
    )

    sql = response.text.strip()

    # Remove markdown formatting if present
    sql = sql.replace("```sql", "")
```



```python
import sqlite3

# Connect to database
connection = sqlite3.connect("data.db")
cursor = connection.cursor()

# Create table (if not exists)
table = """
CREATE TABLE IF NOT EXISTS Students(
    name VARCHAR(30),
    class VARCHAR(10),
    marks INT,
    company VARCHAR(30)
)
"""
cursor.execute(table)

# Insert records
cursor.execute("insert into Students values('Sijo', 'BTech', 75, 'JSW')")
cursor.execute("insert into Students values('Lijo', 'MTech', 69, 'TCS')")
cursor.execute("insert into Students values('Rijo', 'BSc', 79, 'WIPRO')")
cursor.execute("insert into Students values('Sibin', 'MSc', 89, 'INFOSYS')")
cursor.execute("insert into Students values('Dilsha', 'MCom', 99, 'Cyient')")

# Query records
print("Inserted Records:\n")

data = cursor.execute("select * from Students")

for row in data:
    print(row)

# Save changes and close
connection.commit()
connection.close()
```



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.