

# 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
V. Sreshta	FrontEnd
V.sirisha	BackEnd
K. Sai Sri Chaitra	AI Integration
B. Venkaiah Naidu	DataBase
V. Sreshta & V. Sirisha	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.

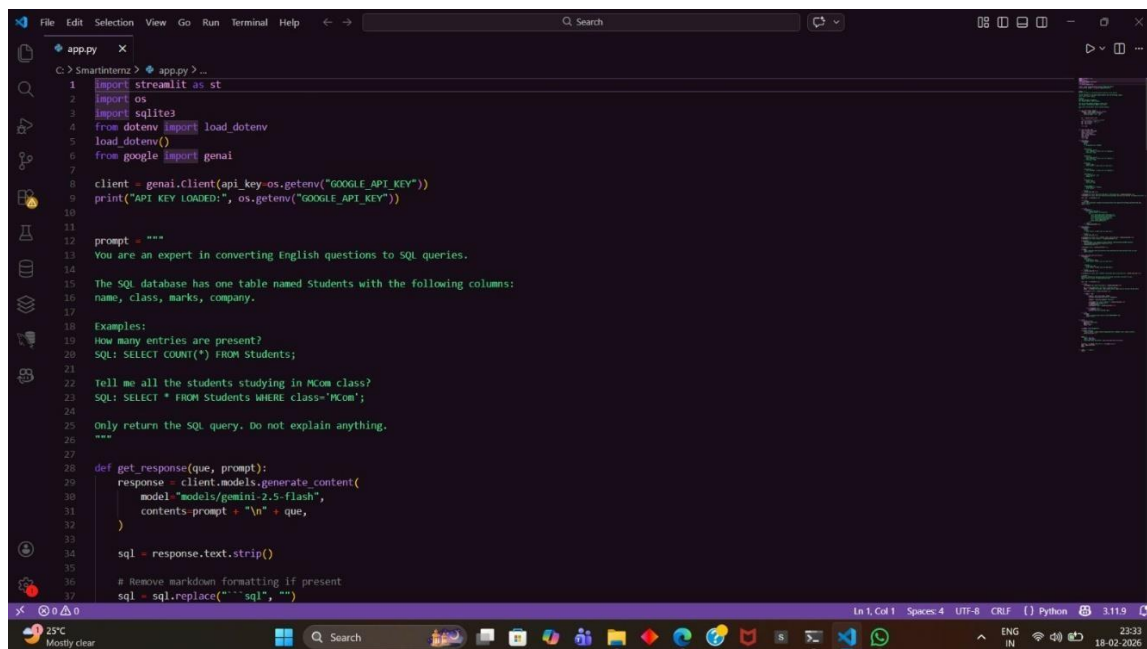
## 9. User Interface

- 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

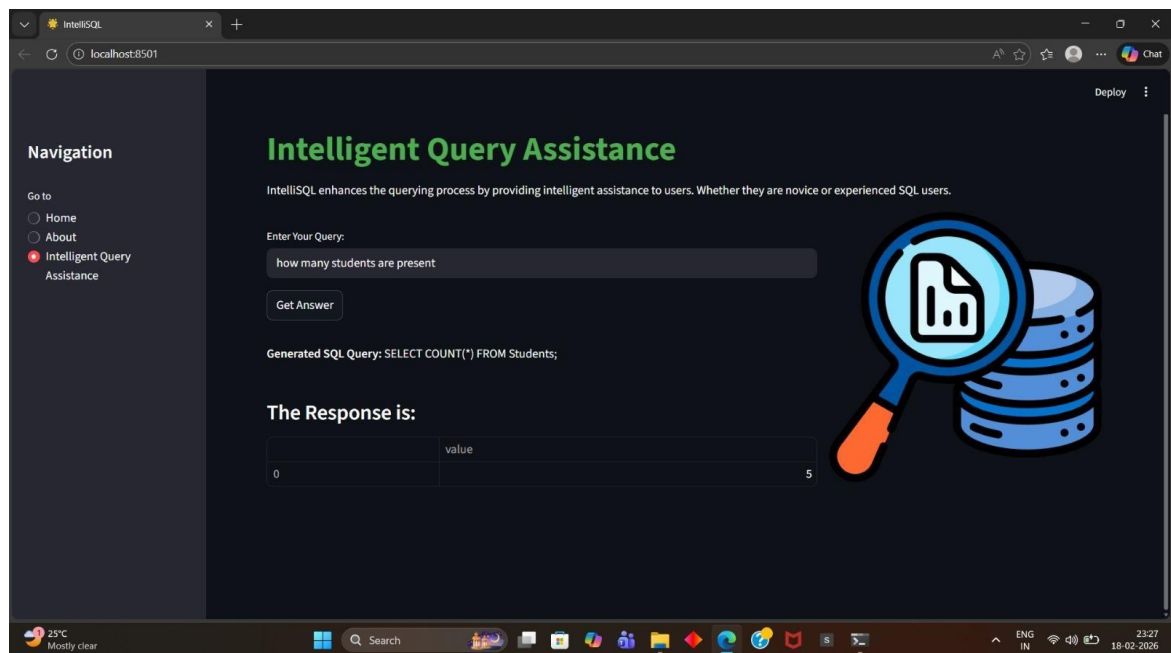


The screenshot shows a VS Code editor window with a Python script named `app.py`. The script is designed to interact with a Gemini API to convert natural language queries into SQL. It includes the following components:

- Imports:** `streamlit` as `st`, `os`, `sqlite3`, `dotenv` (with `load_dotenv`), and `google` (with `genai`).
- Configuration:** It loads environment variables and initializes a `genai` client using a `GOOGLE_API_KEY`.
- Prompt Template:** A multi-line prompt is defined, instructing the model to act as an expert in converting English questions to SQL queries. It also provides schema information for a table named `Students` with columns `name`, `class`, `marks`, and `company`. Examples of queries and their corresponding SQL are provided.
- Function:** A `get_response` function is defined, which takes a query and prompt as input, generates content using the `genai` client, and returns the SQL query.

The script is currently running in a terminal window, and the output shows the API key being loaded successfully.

```
File Edit Selection View Go Run Terminal Help ← → Search
app.py sqlpy x
C:\SmartInternz > sqlpy > ...
1 import sqlite3
2
3 # Connect to database
4 connection = sqlite3.connect("data.db")
5 cursor = connection.cursor()
6
7 # Create table (if not exists)
8 table = """
9 CREATE TABLE IF NOT EXISTS Students(
10     name VARCHAR(30),
11     class VARCHAR(10),
12     marks INT,
13     company VARCHAR(30)
14 )
15 """
16 cursor.execute(table)
17
18 # Insert records
19 cursor.execute("insert into Students values('Sijo', 'BTech', 75, 'JSW')")
20 cursor.execute("insert into Students values('Lijo', 'MTech', 69, 'TCS')")
21 cursor.execute("insert into Students values('Rijo', 'BSc', 79, 'WIPRO')")
22 cursor.execute("insert into Students values('Sibin', 'MSc', 89, 'INFOSYS')")
23 cursor.execute("insert into Students values('Dilisha', 'MCom', 99, 'Cyient')")
24
25 # Query records
26 print("Inserted Records:\n")
27
28 data = cursor.execute("select * from Students")
29
30 for row in data:
31     print(row)
32
33 # Save changes and close
34 connection.commit()
35 connection.close()
36
```



## 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.