

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.Jahnvi	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 code editor interface with multiple tabs. The active tab is 'sqlpy' containing Python code to interact with a SQLite database named 'student.db'. The code includes creating a table 'STUDENT' with fields NAME, CLASS, SECTION, MARKS, and AGE, and inserting sample data records. The code editor has syntax highlighting for Python and SQL. The status bar at the bottom indicates the code is in Line 54, Col 19, with 8 Spaces 4 UTF-8 CRLF, and it is signed out.

```
File Edit Selection View Go Run Terminal Help ↺ ↻ SQL PROJECT CHAT SESSIONS
```

```
app.py sqlpy data.db requirements.txt .env README.md
```

```
import sqlite3
connection = sqlite3.connect("student.db")

# Create a cursor object to insert records and create the table
cursor = connection.cursor()

# Create the STUDENT table with the fields
table_info = """
CREATE TABLE STUDENT (
    NAME VARCHAR(25),
    CLASS VARCHAR(25),
    SECTION VARCHAR(25),
    MARKS INT,
    AGE INT
);
"""
cursor.execute(table_info)

# Insert some sample records with Nepali names and relatable data
cursor.execute("INSERT INTO STUDENT VALUES('Puja','Computer Science','A',88, 21')")
cursor.execute("INSERT INTO STUDENT VALUES('Amit','Computer Science','B',74, 23')")
cursor.execute("INSERT INTO STUDENT VALUES('Sunita','Business Management','A',79, 22')")
cursor.execute("INSERT INTO STUDENT VALUES('Pratik','Business Management','B',81, 24')")
cursor.execute("INSERT INTO STUDENT VALUES('Rajesh','Economics','A',68, 25')")
cursor.execute("INSERT INTO STUDENT VALUES('Neha','Economics','B',83, 23')")
cursor.execute("INSERT INTO STUDENT VALUES('Anil','Computer Science','A',91, 22')")
cursor.execute("INSERT INTO STUDENT VALUES('Iyoti','Computer Science','B',87, 23')")
cursor.execute("INSERT INTO STUDENT VALUES('Nabin','Business Management','A',75, 24')")
cursor.execute("INSERT INTO STUDENT VALUES('Pratima','Business Management','B',89, 20')")
cursor.execute("INSERT INTO STUDENT VALUES('Sushil','Economics','A',66, 22')")
cursor.execute("INSERT INTO STUDENT VALUES('Rekha','Economics','B',71, 23')")
cursor.execute("INSERT INTO STUDENT VALUES('Sarita','Computer Science','A',84, 21')")
cursor.execute("INSERT INTO STUDENT VALUES('Hari','Computer Science','B',99, 24')")
cursor.execute("INSERT INTO STUDENT VALUES('Maya','Business Management','A',78, 23')")
cursor.execute("INSERT INTO STUDENT VALUES('Gopal','Business Management','B',80, 23')")
cursor.execute("INSERT INTO STUDENT VALUES('Deepak','Economics','A',73, 21')")
cursor.execute("INSERT INTO STUDENT VALUES('Binita','Economics','B',69, 22')")
```

```

app.py
1  #!/usr/bin/env python
2  from dotenv import load_dotenv
3  load_dotenv() # Load environment variables from a .env file
4
5  import streamlit as st
6  import os
7  import sqlite3
8  import google.generativeai as genai
9
10 # Configure the GenAI API key using environment variables
11 genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
12
13 # Function to interact with the Gemini model and get an SQL query back
14 def get_gemini_response(question, prompt):
15     model = genai.GenerativeModel("gemini-pro")
16     response = model.generate_content([prompt[0]], question) # Basic content generation using Gemini
17     return response.text # Assume the text is good to use directly
18
19 # Function to actually run the SQL query against the SQLite database
20 def read_sql_query(sql, db):
21     conn = sqlite3.connect(db) # Connect to the SQLite database
22     cur = conn.cursor()
23     try:
24         cur.execute(sql) # Execute the query
25         rows = cur.fetchall() # Get the results
26     except sqlite3.Error as e:
27         rows = [str(e),] # In case of error, return it as part of the result
28     finally:
29         conn.close() # Always close the connection
30     return rows
31
32 # Define the prompt that tells the AI how to generate SQL queries from natural language
33 prompt = [
34     "You are an assistant skilled in converting natural language to SQL queries.",
35     "The database is called 'STUDENT' and has the following columns: NAME, CLASS, SECTION."
36 ]
37
38 Example queries:
39 1. "How many entries are in the database?"
40 2. SQL: SELECT COUNT(*) FROM STUDENT;

```

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.