Documentation on SQL Generation Project with Gemini Project

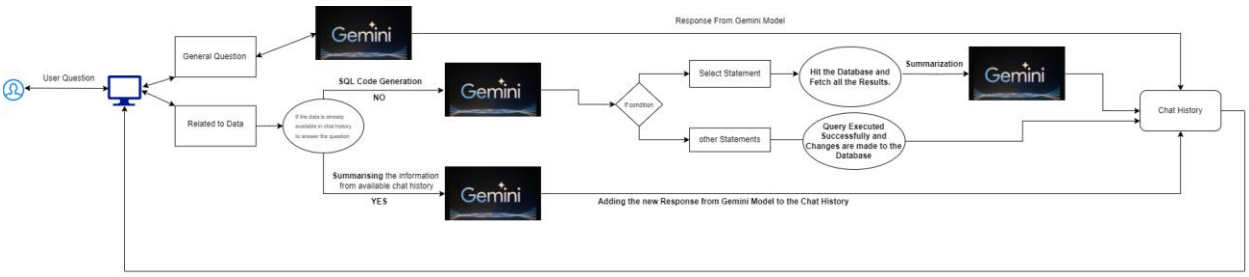
Steps to get Google API Key for Using Gemini LLM:

1. Open this website - https://aistudio.google.com/app/apikey
2. Click on **Create API Key**
3. Accept the terms and conditions and click on "**Create API Key for New Project**".
4. Now project is created, click on "**Create API Key**".
5. Select the created project and click on "**Create API Key on existing project**".
6. API key is created **copy the secret API Key**.

Steps to Setup environment before creating project:

1. Create a virtual environment in VS Code.
2. Download required libraries using the command – **pip install streamlit python-dotenv mysql-connector-python google-generativeai pandas**
3. Create a new **.env** file in the virtual environment.
4. Inside the .env file create a variable with random name and assign the variable with Secrete API Key.
5. Create a main.py folder for the original workflow.

Workflow of the SQL Generation and Summarization:



Explanation:

1. Identify if the question is general.
2. If general:
    - Generate a response from Gemini model and display a general response.
3. If not general:

- Check for relevant data from the Chat History.
- If relevant data is found:
  - Generate a summarization using Gemini model and display it through Chat History.
- If no relevant data is found:
  - Generate an SQL query from Gemini model.
  - Execute the query and display the results through Chat History.
  - Generate a summarization based on the results if the query is a SELECT statement.

Explanation with code:

The Gemini Chatbot is a Streamlit-based application that integrates with Google's Generative AI model to provide conversational responses, SQL query generation, and data summarization based on user input. It utilizes a MySQL database to fetch and display data as per the generated SQL queries. The chatbot can handle general questions and interact with the database schema to generate meaningful SQL queries and summaries.

Environment Variables

Create a `.env` file in your project root and add your Google API key:`GOOGLE_API_KEY = your_google_api_key`

Configure Generative AI

```
import google.generativeai as genai
# Configure GenerativeAI
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
```

Helper Functions

Get Gemini Response

This function interacts with the Generative AI model to get a response based on the conversation history and user prompt. This will take Conversation History, prompt, retries

and delay. If conversation history is found, then new history is appended with the old history and sent the total history to LLM. If not, only User Question is sent to LLM.

```python
def get_gemini_response(conversation_history, prompt, retries=3, delay=2):
    model = genai.GenerativeModel('gemini-pro')
    attempts = 0
    while attempts < retries:
        try:
            conversation = "\n".join(
                [
                    f"User: {entry['user']}\nBot: {entry.get('bot', '')}"
                    for entry in conversation_history
                    if 'user' in entry
                ]
            )

            summarized_history = "\n".join(
                [
                    f"Previous Summary: {entry['bot']}"
                    for entry in conversation_history
                    if 'summary' in entry
                ]
            )

            if not conversation:
                conversation = "No previous conversation available."

            final_prompt = f"{prompt}\n\nConversation History:\n{conversation}\n{summarized_history}\nUser: {conversation_history[-1].get('user'
            response = model.generate_content([final_prompt])
            return response.text

        except Exception as e:
            st.write(f"Attempt {attempts+1} failed with error: {e}")
            attempts += 1
            if attempts == retries:
                raise e
            time.sleep(delay)
```

Read SQL Queries

Function to execute SQL queries and fetch results from the MySQL database. This works only for SELECT Statement related Queries. Once the query is hit to Database then it fetches all the data and converts the table data into csv format and returns the data in text format.

```python
def read_sql_query(sql):
    host = 'localhost'
    user = 'root'
    password = 'root'
    database = 'Project_Main'

    conn = mysql.connector.connect(
        host = host,
        user = user,
        password = password,
        database = database
    )
    cur = conn.cursor()
    cur.execute("SET SESSION sql_mode=(SELECT REPLACE(@@SESSION.sql_mode, 'ONLY_FULL_GROUP_BY', ''))")

    all_results = []

    for result in cur.execute(sql, multi=True):
        if result.with_rows:
            rows = result.fetchall()
            column_names = [i[0] for i in result.description]
            df = pd.DataFrame(rows, columns=column_names)
            all_results.append(df)

    cur.close()
    conn.close()

    return all_results
```

For all other Statements,

```python
def read_sql_all_query(sql):
    host = 'localhost'
    user = 'root'
    password = 'root'
    database = 'Project_Main'

    conn = mysql.connector.connect(
        host=host,
        user=user,
        password=password,
        database=database
    )

    cur = conn.cursor()
    cur.execute(sql)
    conn.commit()
    cur.close()
    conn.close()
```

Question Handling

Functions to identify if the question is general or specific and find relevant data in the provided dataframes.

```python
def is_general_question(question):
    general_keywords = ['hello', 'hi', 'who are you', 'introduce', 'help', 'thanks', 'thank you', 'Good Morning', 'Good Afternoon']
    return any(keyword in question.lower() for keyword in general_keywords)

def find_relevant_data(question, dataframes):
    for df in dataframes:
        if not df.empty and any(col in question.lower() for col in df.columns.str.lower()):
            return df
    return None
```

Main function

Main function handles all the flow of the data and use all the functions created above. This function contains all the prompts required, Conditions for flow of data and contains the code for Web Interface (Streamlit).

```
def main():
    MAX_HISTORY_LENGTH = 100

    st.set_page_config(layout="wide")
    st.markdown(
        """
        <style>
        .main {
        background-color: #FFFFFF;
        color: #0e1117;
        }
        .css-1v0mbdj.e1tzin5v0 {
            background-color: #FFFFFF;
        }
        body {
            background-image: url('https://images.rawpixel.com/image_800/cHJpdmF0ZS9sci9pbWFnZXMvd2Vic2l0ZS8yMDIyLTA1L3B4MTU4NDgwMy1pbWFnZS1rd3Z
            background-size: cover;
            color: white;
        }
        .chat-history {
            height: 400px;
            overflow-y: auto;
            padding: 10px;
            background-color: rgba(255, 255, 255, 0.8);
            border-radius: 10px;
        }
        .user-question {
            background: rgba(0, 123, 255, 0.1);
            color: #ffffff;
            padding: 10px;
            border-radius: 10px;
            margin-bottom: 10px;

        }
        .bot-response {
```

```
    prompt_1 = """
        You will generate a valid SQL query that fetches the information the user wants from the database. User will be generic in framing his/h
        For example: User asks "Give me the details of the Product with max price"
        Reasoning: Products details will be in Product table and if you read the table details you will find that the columns to pick out are Pr

        Here is the Schema Description of the database. You will find all the tables and column details. Use those names to insert into your SQL
        CompanyX Database Schema (Stores information on customers, products, sales and orders)
        1. Product:
           - Product_ID (VARCHAR, PK)
           - Product_Name (VARCHAR)
           - Price_MRP (DECIMAL(10,2))
           - Price_SALE (DECIMAL(10,2))
           - Quantity (FLOAT)

        2. Customers:
           - Index1 (INT)
           - Customer_Id (VARCHAR, PK)
           - First_Name (VARCHAR)
           - Last_Name (VARCHAR)
           - Company (VARCHAR)
           - City (VARCHAR)
           - Country (VARCHAR)
           - Email (VARCHAR)
           - Subscription_Date (DATE)
           - Sex (CHAR(7))
           - Date_of_birth (DATE)

        3. Orders:
           - Order_ID (VARCHAR, PK)
           - Customer_ID (VARCHAR, FK to Customers)
           - Customer_Status (VARCHAR)
           - Date_Order_Placed (DATE)
           - Delivery_Date (DATE)
```

```
prompt_2 = """
   The table information above was retrieved from the database to answer the user query. Use only that information and construct a meaningfu
    You are an expert in analyzing table information and providing insightful textual summaries by understanding the context of the user que
    Always try to understand how the table information can answer the user's question, if not respond saying the information is not availabl

    Response Instructions

    1. Always answer in the context of the user's query.
    2. Analyze the table to understand the key points, statistics, and trends relevant to the query.
    3. every time Provide a concise yet informative summary, including any relevant statistical insights, mathematical operations, and meani
    4. If you have encounter some random question then answer if it is related to Schema or if it is a genral question like introduction or
    6. Explain the statistics like mean median mode if needed.
    7. you need to explain table information in proper english text.
    Example:
    What is the most expensive product?
    The most expensive product in the given table is the “Premia Kashmiri Chilli Powder” with a maximum retail price (MRP) of 880.00.

    What are the top 3 products in terms of pricing?
    The “Premia Kashmiri Chilli Powder” stands out as the most expensive product with an MRP of 880.00.
    The “Premia Pista Plain (Pistachios)” follows with an MRP of 358.00.
    The “Date Crown Fard (Khajur)” is the least expensive among the top 3, priced at 257.00.
"""

prompt_general = """
You are a conversational chatbot designed to answer general questions, provide introductions, and assist users with basic information about

Example questions:
    Who are you?
    How can you help me?
    Tell me about this service.
    Can you help me summarize my data?
    Can you generate SQL queries for my database?
```

```
st.title('Gemini Chatbot with Streamlit')

col1, col2 = st.columns([2, 3])

with col1:
    st.header("User Interaction")
    if 'history' not in st.session_state:
        st.session_state.history = []
    if 'dataframes' not in st.session_state:
        st.session_state.dataframes = []

    question = st.text_input('Enter your question here:')

    if st.button('Get Response'):
        st.session_state.history.append({'user': question})

        if len(st.session_state.history) > MAX_HISTORY_LENGTH:
            st.session_state.history = st.session_state.history[-MAX_HISTORY_LENGTH:]

        if is_general_question(question):
            try:
                general_response = get_gemini_response(st.session_state.history, prompt_general)
                st.session_state.history.append({'bot': general_response})
                st.write(general_response)
            except Exception as e:
                error_message = "Error generating response from Gemini API: " + str(e)
                st.session_state.history.append({'bot': error_message})
                st.write(error_message)
        else:
            relevant_data = find_relevant_data(question, st.session_state.dataframes)
            if relevant_data is not None:
                try:
                    summarize_prompt = relevant_data.to_string(index=False) + "\n" + prompt_2
                    summary_response = get_gemini_response(st.session_state.history, summarize_prompt)
```

```
            if relevant_data is not None:
                try:
                    summarize_prompt = relevant_data.to_string(index=False) + "\n" + prompt_2
                    summary_response = get_gemini_response(st.session_state.history, summarize_prompt)
                    st.session_state.history.append({'bot': summary_response, 'summary': summary_response})
                    st.write("Summary Response:")
                    st.write(summary_response)
                except Exception as e:
                    error_message = "Error generating response from Gemini API: " + str(e)
                    st.session_state.history.append({'bot': error_message})
                    st.write(error_message)
            else:
                try:
                    generated_query = get_gemini_response(st.session_state.history, prompt_1)
                    st.session_state.history.append({'bot': "Generated SQL Query:\n" + generated_query})
                    st.write("Generated SQL Query:")
                    st.write(generated_query)

                    if generated_query.strip().lower().startswith("select"):
                        try:
                            response_dfs = read_sql_query(generated_query)
                            if response_dfs:
                                for response_df in response_dfs:
                                    st.session_state.dataframes.append(response_df)
                                    st.write(response_df)
                                    df_summary = response_df.to_string(index=False)
                                    summarize_prompt = df_summary + "\n" + prompt_2
                                    summary_response = get_gemini_response(st.session_state.history, summarize_prompt)
                                    st.session_state.history.append({'bot': "Summary Response:\n" + summary_response, 'summary': summary_res
                                    st.write("Summary Response:")
                                    st.write(summary_response)
                            else:
                                no_data_message = "No data found."
                                st.session_state.history.append({'bot': no_data_message})
```

```
                        except Exception as e:
                            error_message = "Summarization error" + str(e)
                            st.session_state.history.append({'bot': error_message})
                            st.write(error_message)
                    else:
                        try:
                            read_sql_all_query(generated_query)
                            success_message = "Query executed successfully."
                            st.session_state.history.append({'bot': success_message})
                            st.write(success_message)
                        except Exception as e:
                            error_message = "Error executing SQL query: " + str(e)
                            st.session_state.history.append({'bot': error_message})
                            st.write(error_message)

                except Exception as e:
                    error_message = "Error generating response from Gemini API: " + str(e)
                    st.session_state.history.append({'bot': error_message})
                    st.write(error_message)

    with col2:
        st.header("Chat History")
        if st.session_state.history:
            for entry in st.session_state.history:
                if 'user' in entry:
                    st.markdown(f'<div class="user-question">{entry["user"]}</div>', unsafe_allow_html=True)
                if 'bot' in entry:
                    st.markdown(f'<div class="bot-response">{entry["bot"]}</div>', unsafe_allow_html=True)

f __name__ == '__main__':
    main()
```

In the above code, Prompt_1 and Prompt_2 is for SQL code generation prompt and summarization prompt. General_prompt is used to answer the general questions to the user like hi, hello, bye, etc. Chat history is maintained throughout the session. If reloaded, then the session ends and a new session is opened.

User Manual:

- Run the Streamlit application: "`streamlit run main.py`"
- Open your web browser and navigate to the URL provided by Streamlit.
- Interact with the chatbot by typing your questions and receiving responses. The chatbot will also generate and execute SQL queries based on your questions.