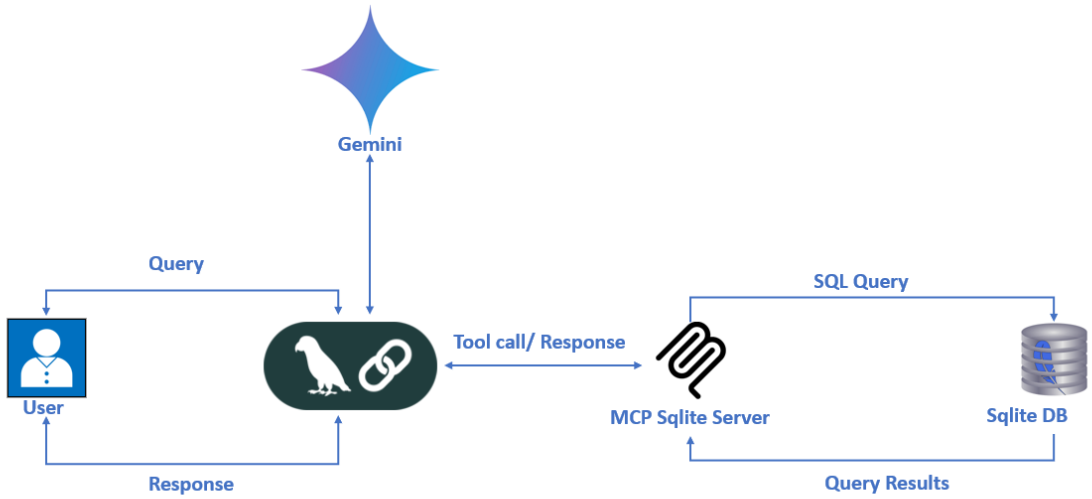
LangChain with SQLite MCP Server

Overview



Prerequisites

Before you begin, ensure you have the following installed:

- Python 3.8 or higher
- Git
- A Google API key for accessing the Gemini model
- Nodejs, npm, npx
- Copy the database.db to a location and note the path

Project Setup

Installing uv Package Manager

`uv` is a fast Python package installer and virtual environment manager, significantly faster than pip for package installations and environment management.

```
pip install uv
```

Setting Up the Project Directory

Create a new project directory and initialize it:

```
uv init sqlite_mcp_langchain
cd sqlite_mcp_langchain
```

Installing Dependencies

Install all required packages using `uv`:

```
uv add mcp==1.21.0 langchain==1.0.5 langchain-mcp-adapters==0.1.12 langgraph==1.0.2
langchain-google-genai==3.0.1
```

Setting Up the MCP SQLite Database Server

The application requires the MCP SQLite server to be available:

```
git clone https://github.com/modelcontextprotocol/servers-archived/
```

This repository contains various MCP server implementations, including the SQLite server we need.

Configuring Google API Key

To use the Gemini model, you'll need to set up a Google API key:

1. Go to https://aistudio.google.com/welcome
2. Sign in or create a Google Cloud account
3. Create an API key in the Google AI Studio
4. Set the API key as an environment variable:

Windows:

```
set GOOGLE_API_KEY=your_gemini_api_key_here
```

macOS/Linux:

```
export GOOGLE_API_KEY=your_gemini_api_key_here
```

To verify that your API key is correctly set:

Windows:

```
echo %GOOGLE_API_KEY%
```

macOS/Linux:

```
echo $GOOGLE_API_KEY
```

Example queries:

- "List all users in the database"
- "Show me the schema of the database"
- "How many entries are in each table?"
- "Find users older than 30"

To exit the application, type `/q` at the query prompt.

Understanding the Code

Let's break down the main components of the `sqlite_assistant.py` script:

1. Imports and Initialization

```
import asyncio
import os
from dotenv import load_dotenv
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client
from langchain_mcp_adapters.tools import load_mcp_tools
from langchain.agents import create_agent
from langchain_google_genai import ChatGoogleGenerativeAI
```

This section imports necessary libraries and initializes the Gemini model. The `ChatGoogleGenerativeAI` class provides an interface to Google's Gemini model.

2. Server Configuration

```
# Configure SQLite server parameters
server_params = StdioServerParameters(
    command="uv",
    args=[
        "--directory",
        "path_to_the_cloned_repo/src/sqlite/src/mcp_server_sqlite",
        "run",
        "mcp-server-sqlite",
        "--db-path",
         "path_to_the_db"
    ],
```

```
)
```

This section configures the SQLite server parameters. Make sure to:

- Replace `path_to_the_cloned_repo` with the actual path to the cloned MCP servers repository
- Replace `path_to_the_db` with the path to your SQLite database file.

3. Query Processing

The `process_query` function handles the processing of natural language queries:

```
async def process_query(agent, query, history):
    # Function implementation details...
```

This function:

- Takes a user query and conversation history
- Formats the query with context from previous conversations
- Executes the query using the LangChain agent
- Processes and displays tool calls and results
- Returns the final response

4. Main Program Loop

The `main` function establishes connections and handles user interaction:

```
async def main():
    # Function implementation details...
```