# ****MCP Client-Server Implementation Summary****

## ****Approach to Solve the Task****

The objective was to build an **MCP-based Client-Server system** where the client sends a query, the server processes it using **Ollama LLM**, generates a system command, executes it, and returns the result. The implementation was based on **FastAPI** for the server and **aiohttp** for client communication.

## ****Server Implementation (****server.py****)****

A **FastAPI server** was set up with **FastMCP** to handle MCP tools. The process\_query function was registered as an MCP tool, which took user input, sent it to **Ollama**, received a system command, and executed it using subprocess.run(). The server was expected to expose an **MCP API endpoint** to handle client queries. However, the **tool was not properly registered**, leading to **404 errors and connection failures**.

## ****Client Implementation (****client.py****)****

The client was implemented using **aiohttp** to send queries to the MCP server. Initially, incorrect imports (mcp.client.http, mcp.client.transport) caused issues, so we switched to direct HTTP requests. The client successfully sent requests, but the server **failed to recognize the tool’s endpoint**, leading to 404 errors.

## ****Challenges Faced****

MCP's **limited documentation and unstable API** made it difficult to register tools dynamically. The **server failed to expose the required endpoint**, making communication impossible. Connection issues like **SSL errors and refusal of connection** also persisted. Additionally, Ollama’s response format required adjustments.

## ****Current Status & Next Steps****

The implementation is **not fully functional** due to MCP's tool registration issues. Since **MCP is still evolving**, debugging is ongoing. The task is being submitted for review while further investigation continues to resolve these issues.