# ****Approach to Developing the MCP Client-Server System****

## ****1. Introduction****

The CLI-based **MCP (Model Context Protocol) Client-Server system** has developed using a **Flask-based server** with an **LLM (Ollama)** to generate and execute system commands dynamically. The key focus was on **maintaining conversation context, ensuring security, and handling OS-specific commands** while maintaining a structured interaction flow.

## ****2. Development Approach****

### ****2.1 MCP Client-Server Architecture****

The system follows a **client-server model**, where:

1. The **MCP Client** sends user queries.
2. The **MCP Server (Flask)** processes the queries, interacts with **Ollama**, and executes the generated commands.
3. The server returns the execution results to the client.

### ****2.2 Conversation Context Handling****

To maintain **chat history**, we implemented:

* **Session-based storage** to retain previous queries.
* **Appending past messages** to new requests for better LLM context.
* Ensuring the LLM generates **context-aware responses** instead of isolated commands.

### ****2.3 Command Processing Strategy****

* Queries are passed to the **Ollama LLM** for interpretation.
* The LLM **generates the most suitable system command** based on user input.
* The generated command is checked against **restricted commands** for security.
* The system detects the **OS type** and modifies commands accordingly (e.g., ls for Linux, dir for Windows).
* The **command is executed securely** using Python’s subprocess.run() with shell=False to prevent shell injection.

### ****2.4 Coding Strategy****

#### ****2.4.1 MCP Server (****app.py****)****

* Handles **Flask routes** for client communication.
* Calls the **LLM via Ollama** to generate system commands.
* Ensures safe execution of generated commands.
* Returns results while maintaining **chat history**.

#### ****2.4.2 MCP Client (****client.py****)****

* Takes user input and **sends queries to the server**.
* Receives **commands and execution results** from the server.
* Displays the interaction in a structured manner.

#### ****2.4.3 LLM Interaction (****handler.py****)****

* Formats queries into **structured MCP messages**.
* Sends user input (along with previous context) to the **Ollama LLM**.
* Extracts and refines **system commands** from the response.
* Returns the final command for execution.

#### ****2.4.4 Logging (****logger.py****)****

* Stores **all interactions (queries, commands, results)** for auditing.
* Helps **debug issues** related to command generation or execution.