B06 - AI-Assisted MCP Server: Project Report

Student: Michele Sagone **Project:** B06 - MCP Server Exposing CSV Directory **Date:** 19/12/2025

1. Introduction and Objectives

The goal of this project was to develop a server compatible with the Model Context Protocol (MCP) capable of exposing local CSV files as queryable tables. The role of AI was that of a **"Lead Developer"** for generating Python code, while I acted as the **"Technical Project Manager"**, defining specifications, managing the execution environment (WSL), and validating the results.

2. Methodology and Prompts (Chain of Thought)

Development took place through a series of targeted iterations. Below are the 5 main prompts used to define the server architecture:

Prompt 1: Basic Infrastructure Setup

Objective: Create the server skeleton and file scanning logic.

> "Act as a Senior Python Developer. I want to create an MCP server using the `fastmcp` library. The server must scan a local folder `./data`, find all `.csv` files, and automatically create a `list_tables` tool to list them. Use `os` for path management in a way that is compatible with Linux/WSL."

Prompt 2: Data Retrieval & Formatting

Objective: Implement data reading and resolve formatting issues.

> "I need a tool `query_data(table_name, limit)` that reads a CSV using Pandas and returns the rows in Markdown format to make them readable for the LLM. If I get the `Missing optional dependency 'tabulate'` error, tell me exactly how to fix it in my virtual environment."

Prompt 3: Dynamic Resource Implementation

Objective: Expose files as passive resources (not just tools).

> "I want every CSV file in the folder to appear in the Inspector's 'Resources' tab as `csv://filename.csv`. Write a function that iterates over the files and uses `@mcp.resource` to register them dynamically. Warning: ensure that the reading functions do not overwrite each other in the for loop (lambda closure issue)."

Prompt 4: Architecture Change (From STDIO to SSE)

Objective: Make the server accessible via network/web.

> "Modify the `server.py` file. Instead of using standard transport (STDIO), I want to use SSE (Server-Sent Events) over HTTP. Configure `mcp.run()` to use `uvicorn` on port 8000 and explain how I should change the startup command in the terminal."

Prompt 5: Advanced Data Analytics

Objective: Add intelligence to the server (Analytics).

> "Raw data is too much for the LLM to process. Create two new advanced tools:
>
> 1. `get_stats(table_name)`: uses `pandas.describe()` to give me the mean, min, and max of numeric columns.
> 2. `search_in_table(table_name, column, value)`: to search for specific rows, ignoring case sensitivity. The code must be robust and handle errors if the column does not exist."

3. Verification and Corrections (QA & Debugging)

During development, the code generated by the AI was syntactically correct, but integration with the local environment required critical manual interventions:

- **Dependencies:** Manual installation of `tabulate` and `uvicorn`, which were missing in the AI's initial outputs.
- **WSL vs. Windows:** Resolution of absolute paths to allow Node.js (Inspector) to communicate with the Python (Server) through the Linux subsystem.

4. Project Evolution: Analytics and SSE

In the final phase of development, we extended the basic functionalities to make the server more robust:

Improvement 1: From STDIO to SSE

We switched from standard input/output (STDIO) transport to **SSE (Server-Sent Events)** over HTTP.

- **Motivation:** This allows decoupling the server from the client, facilitating debugging via the Web Inspector and preparing the system for future integrations.

Improvement 2: Analytics Tools

We noticed that the AI struggled to calculate statistics by reading raw rows. We therefore implemented two new native Python tools:

1. `get_stats`: Delegates mathematical calculations to Pandas, returning a reliable summary to the AI.
2. `search_in_table`: Allows searching for specific records (e.g., orders for a user) without having to read the entire file.

Improvement 3: Prompt Templates

Predefined templates (e.g., `audit_data_quality`) were added directly to the server to standardize common requests and automate CSV analysis.

5. Conclusion

The project demonstrates how a well-configured MCP server can transform a simple folder of static files into a "smart database." The use of specific tools for analysis (`get_stats`) instead of just reading raw data (`query_data`) significantly improved LLM performance, reducing calculation errors and token consumption.