

Project: B06 - AI-Assisted MCP Server Exposing a CSV Directory as Data Tables

### Goal

Use AI to help design and implement an MCP-compatible server that scans a directory of CSV files and exposes each file as a data source/table via MCP tools/endpoints (e.g. list tables, describe schema, run simple queries or filtered reads).

Tasks

Design & Implement

Use AI to assist in designing and implementing an MCP server (language of your choice) that:

Reads a configurable directory of CSV files

Exposes each CSV as a named resource/table

Provides at least tools/endpoints to list tables, inspect columns, and retrieve filtered rows (e.g. by column equality or simple predicates).

Test & Demonstrate

Create a small set of example CSV files, connect your MCP server from a compatible client, and demonstrate interactions (listing tables, inspecting schemas, fetching filtered data), documenting any limitations or edge cases.

AI Usage & Verification Report

Collect and submit the prompts you used with AI and write a short report explaining how you evaluated and corrected the AI-generated code (e.g. validating CSV parsing, checking protocol compliance, handling malformed input).

### Architecture

Java project using the Spring Boot framework. Maven is used for dependency management. Maven is a build automation and project management tool mainly used for Java applications.

Dependences:

Java 25 -- Oracle Open JDK 25.0.1

Spring Boot -- 3.5.8

Spring AI -- 1.1.1

OpenCSV – 5.12.0

Lombok – 1.18.42

Spring Boot Test

Maven – 3.9.11

### Build and compilation

If you want to build the project, you must do so via Maven. Using the `-- clean install --` command, you can optionally add the `-DskipTests` parameter to skip tests (junit). After the build, you can run the Spring Boot application using the command (also via Maven) `-- spring-boot:run --`

**For simplicity, if you just want to run the application**, a "fat-jar" has been created. You can find it in the git repository under the `jar` folder. To run it, you just need the `jvm` and run the command `-- java -jar ServerMCP-0.0.1.jar --`

## TEST

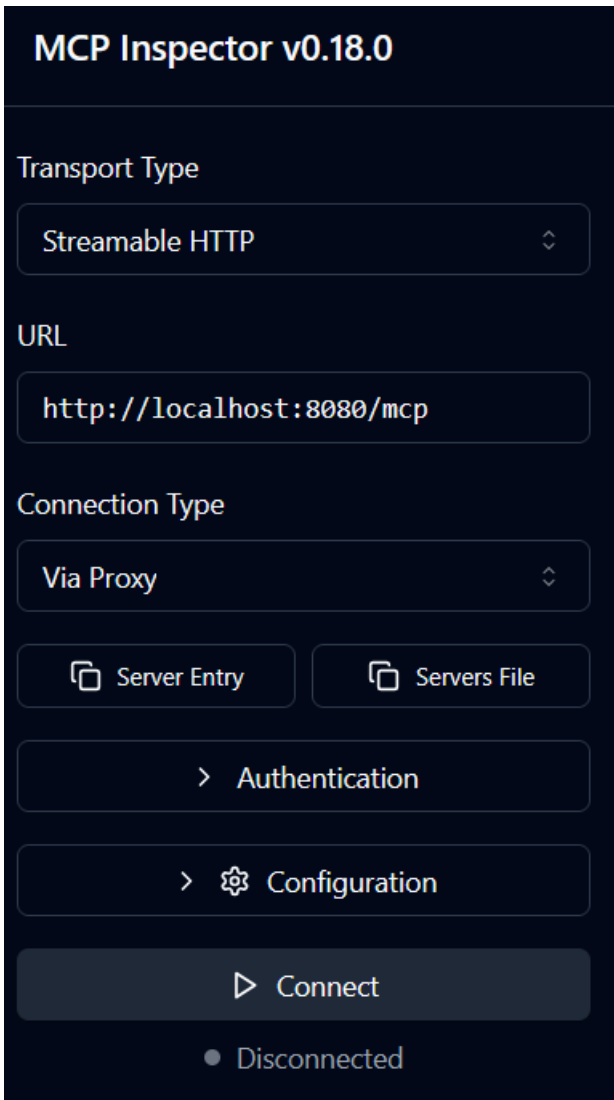
MCP-Inspector was used for end-to-end testing.

**Important:** If you want to repeat the tests using MCP-Inspector, you must enable the proxy integrated into MCP-Inspector. The URL is `http://localhost:8080/mcp`.

**Important:** By default, if you don't specify it in the tool call, the path where the CSV files will be searched is the same as the one where the jar is launched.

Below are some test examples;

Configuring MCP-Inspector connection:



The screenshot shows the MCP Inspector v0.18.0 configuration window. It has a dark theme. At the top, the title 'MCP Inspector v0.18.0' is displayed. Below the title, there are several configuration sections: 'Transport Type' with a dropdown menu set to 'Streamable HTTP'; 'URL' with a text input field containing 'http://localhost:8080/mcp'; 'Connection Type' with a dropdown menu set to 'Via Proxy'; two buttons labeled 'Server Entry' and 'Servers File'; a section for 'Authentication' with a right-pointing chevron; a section for 'Configuration' with a right-pointing chevron and a gear icon; a large 'Connect' button with a play icon; and a status indicator showing a grey dot and the text 'Disconnected'.

**MCP Inspector v0.18.0**

Transport Type

Streamable HTTP

URL

`http://localhost:8080/mcp`

Connection Type

Via Proxy

Server Entry Servers File

> Authentication

> ⚙ Configuration

▶ Connect

● Disconnected

Tool list:

A screenshot of a tool list interface. It features a dark blue background with a rounded rectangle containing four tool entries. Each entry consists of a tool name in bold, a description in a lighter font, and a right-pointing chevron icon.

- filterCsv**  
filterCSv
- openCsv**  
Opens the CSV file and returns max the first 20 rows. Possible parameters are: the required file path, an optional column separator (default is ,), an optional...
- hello**  
Send hello
- csvList**  
List CSV file in directory the default is ..\ directory

CSV list without passing the path as a parameter, as previously said by default it is the path where the jar is launched :

A screenshot of the 'csvList' tool configuration and execution interface. The interface is dark-themed and includes a title bar, a parameter input field, a metadata section, action buttons, and a results display.

List CSV file in directory the default is ..\ directory

path \*

**Tool-specific Metadata:** Add Pair

No metadata pairs.

Run Tool

Copy Input

**Tool Result: Success**

```
[
  0: "ordini.csv"
  1: "prodotti.csv"
  2: "Sensore.csv"
  3: "separator.csv"
  4: "studenti.csv"
  5: "testData.csv"
]
```

CSV list by passing the path C:\Users\PC\Desktop


List CSV file in directory the default is ..\ directory


path \*

C:\Users\PC\Desktop

**Tool-specific Metadata:** Add Pair


No metadata pairs.

 Run Tool

 Copy Input

**Tool Result: Success**

```
[
  0: "studenti.csv"
  1: "testData.csv"
]
```



Test opening a CSV with a specified path, with field names in the first row. Showing the first five rows:

path \*

C:\Users\PC\Desktop\studenti.csv


separator \*

nameInFirstLine \*

☒ Toggle this option

number \*

5



Response:

```
{
  "colunsName": [
    "id",
    "nome",
    "cognome",
    "corso",
    "anno"
  ],
  "lines": [
    [
      "1",
      "Luca",
      "Rossi",
      "Informatica",
      "1"
    ],
    [
      "2",
      "Giulia",
      "Bianchi",
      "Matematica",
      "2"
    ],
    [
      "3",
      "Marco",
      "Verdi",
      "Fisica",
      "3"
    ],
    [
      "4",
      "Sara",
      "Neri",
      "Informatica",
      "2"
    ],
    [
      "5",
      "Paolo",
      "Gallo",
      "Chimica",
      "1"
    ]
  ]
}
```

Test opening CSV with separator passed as parameter, in this case it is the semicolon ";". The default is the colon ":"

path \*

separator.csv

separator \*

;

nameInFirstLine \*

☒ Toggle this option

number \*

1

Tool-specific Metadata:

Add Pair

No metadata pairs.

Response:

```
{
  "colunsName": [
    "id",
    "prodotto",
    "categoria",
    "quantita",
    "scaffale"
  ],
  "lines": [
    [
      "1",
      "Viti 3mm",
      "Ferramenta",
      "500",
      "A1"
    ]
  ]
}
```

CSV Filter Test, As you can see below you have the possibility to apply multiple filters simultaneously per request.

path \*

Sensore.csv

filters \*

Copy JSON

Format JSON

Switch to Form

```
[
  {
    "column": "umidita",
    "operator": "GREATER_THAN",
    "value": "40"
  },
  {
    "column": "stato",
    "operator": "EQUALS",
    "value": "ok"
  }
]
```

separator \*

nameInFirstLine \*

☒ Toggle this option

Response:

```
{
  "filtersLines": [
    {
      "stato": "ok",
      "umidita": "45",
      "id": "1",
      "temperatura": "21.5",
      "timestamp": "2024-02-01T10:00"
    },
    {
      "stato": "ok",
      "umidita": "44",
      "id": "2",
      "temperatura": "21.7",

```

```
    "timestamp": "2024-02-01T10:05"
  },
  {
    "stato": "ok",
    "umidita": "43",
    "id": "3",
    "temperatura": "22.0",
    "timestamp": "2024-02-01T10:10"
  },
  {
    "stato": "ok",
    "umidita": "42",
    "id": "4",
    "temperatura": "22.3",
    "timestamp": "2024-02-01T10:15"
  },
  {
    "stato": "ok",
    "umidita": "41",
    "id": "5",
    "temperatura": "22.8",
    "timestamp": "2024-02-01T10:20"
  }
]
```