DATA-236 Sec 12 - Distributed Systems for Data Engineering

HOMEWORK 4

Total Point: 30

Instructions:

- Include screenshots of the code for each component, as well as the code for app.js, along with the corresponding output. Ensure that the code and its output are displayed together, one below the other.
- Submission should be in PDF Format.
- Please name your submission file as {last_name}_HW4.pdf

Q1. React (10 Points)

Create a Book Management App where users can add, update, and delete books.

I. Home Page **(1 Point)**
Write the necessary code to creat the 'Home.jsx' component which will display all the books. The Home component should be rendered when the user is on the root route '/'.

II. Add a New Book **(2 Points)**
Write the necessary code to create the 'CreateBook.jsx' component. The component should be rendered when the user is on the '/create' route.
The component should accept props (similar to demo) to add the new book and have the following:

- An input field for entering the Book Title.
- An input field for entering the Author Name.
- A submit button labeled "Add Book".

When the user clicks the Add book button:

- A new book should be added to the book list with an auto-incremented book ID.
- The user should be redirected to the home page to see the updated book list.

III. Update Book **(2 Points)**

Write the necessary code to create the 'UpdateBook.jsx' component. The component should be rendered when the user is on the '/update' route.
The component should accept props (similar to demo) to update the new book and have the following:

- An input field for Book Title.

- An input field for Author Name.
- A submit button labeled "Update Book".

When the user clicks the Update Book button the book within context should be updated and the user should be redirected to the home page showing updated book list.

IV. Delete the Book **(2 Points)**

Write the necessary code to create the 'DeleteBook.jsx' component. The component should be rendered when the user is on the '/delete' route.
The component should accept props (similar to demo) to delete a book and have the following:

- A button labeled "Delete Book".

When the user clicks the Delete Book button the book in context should be deleted and the user should be redirected to the home page showing the updated book list.

V. For all the above questions make sure to:

- Pass props for the Create, Update, and Delete components. **(1 Point)**
- Use hooks like useState and useEffect wherever necessary. **(1 Point)**
- User React-Router-Dom for routing. **(1 Point)**

Q2. MySQL (10 points)
In this question, you will apply the concepts learned in class to build a simple CRUD (Create, Read, Update, Delete) application using Node.js and MySQL using Sequelize

Create an appropriate book Model, *View(Optional*), and Controller.

You are required to create a Book Management System that allows users to:

- Add a new book (POST)
- View all books (GET)
- View a book by ID (GET)
- Update book details (PUT)
- Delete a book (DELETE)

Create a MySQL database named **book_db** with a table called **books** for this task.
You are free to name your API endpoints appropriately.

Submit the Postman API response screenshots for each operation. Each completed operation will get 2 points. Also, submit a screenshot of the database and project folder structure.

Homework: Build a MCP server that fetches recipes from TheMealDB (10 points)

Your task is to write a **local MCP server** (in Python) that exposes a few tools Claude can call to **search and fetch recipes** from the public **TheMealDB** API. You'll run and debug it with the MCP Inspector, then wire it into **Claude Desktop** so you can ask for recipes in plain English.

This assignment uses **no API keys you need to create** TheMealDB allows a test key of 1 for education/dev, and it publishes simple JSON endpoints. (themealdb.com)

What you'll build

A local MCP server named meals that provides these tools:

1. **search_meals_by_name(query, limit=5)**
    - Call TheMealDB **Search meal by name** endpoint: …/search.php?s=<query>
    - Return up to limit meals with: name, area (cuisine), category, thumbnail, and the meal **id** for drill-down. (themealdb.com)
2. **meals_by_ingredient(ingredient, limit=12)**
    - Call **Filter by main ingredient**: …/filter.php?i=<ingredient>
    - Return small cards (name, thumbnail, id). (themealdb.com)
3. **random_meal()**
    - Call **Lookup a single random meal**: …/random.php. (themealdb.com)

The API base is https://www.themealdb.com/api/json/v1/1/… and the published "test key" is 1, which is sufficient for this homework. (themealdb.com)

Set up your environment

1. **Install the MCP Python SDK with CLI** :
    - With **pip**: pip install "mcp[cli]" (GitHub)

2. You'll write a single Python file (e.g., meals_server.py) that defines your tools and runs FastMCP over **STDIO**.

Important logging rule: For STDIO servers, **never write to stdout** (that corrupts the JSON-RPC stream). Log to **stderr** using a logger. (Model Context Protocol)

I/O:

Input → Output shapes you should implement:

1. search_meals_by_name(query, limit)
    - **Input:** query: str, limit: int (1–25)
    - **Output (list):** objects with { id, name, area, category, thumb }
2. meals_by_ingredient(ingredient, limit)
    - **Input:** ingredient: str, limit: int
    - **Output (list):** objects with { id, name, thumb }
3. meal_details(id)
    - **Input:** id: str | int
    - **Output (object):** { id, name, category, area, instructions, image, source, youtube, ingredients: [{name, measure}] }
4. random_meal()
    - **Input:** none
    - **Output:** same shape as meal_details.

- **Error handling:**
    - If an endpoint returns "meals": null (no results), return a **clear, empty result** with a short message (e.g., "no matches").
    - Network or JSON errors should raise a clean error to the client (MCP Inspector will display it).

Where in the API to look

- Search by **name**: …/search.php?s=Arrabiata
- List by **first letter**: …/search.php?f=a (optional bonus)
- **Lookup by id**: …/lookup.php?i=52772
- **Random**: …/random.php
- **Filter by main ingredient**: …/filter.php?i=chicken_breast
- **List categories**: …/categories.php (optional bonus)
  All of the above are documented on the official API page and work with the dev key 1. (themealdb.com)

Run and debug locally with MCP Inspector

1. In your project folder, open a terminal and start **development mode** with the server file you wrote:

```
None
mcp dev <your_file_name.py>
```

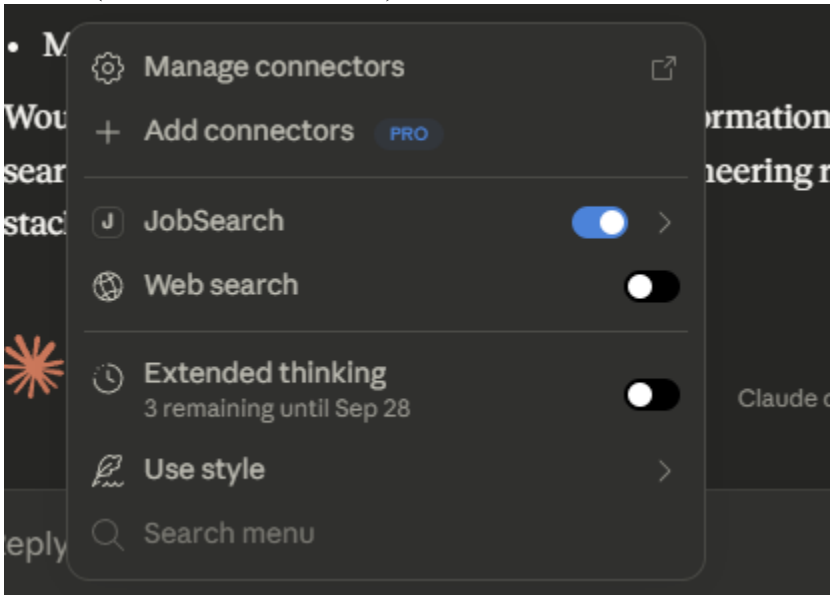
2. The Inspector opens in your browser. Use it to:
   - Call each tool with test inputs (e.g., "Arrabiata", ingredient "chicken").
   - Verify outputs look like your designed schema.
   - Fix any errors before moving on.
     The Inspector is the official debugging UI for MCP servers. (Model Context Protocol)

Connect your server to Claude Desktop

Claude for Desktop can launch and talk to your local MCP server once you add it to the **Claude config**. The official "Build an MCP server" guide walks through this flow and shows the exact config file location and format. Key points:

- Edit claude_desktop_config.json (path shown in the docs for Windows and macOS).

- Add an entry under mcpServers with a command (e.g., python or uv) and the **absolute path** to your project and server file in args.
- Save the file and **restart Claude Desktop**; look for the tools slider icon to confirm the server is loaded. (Model Context Protocol)



Something like this (Jobsearch in this case)

> The doc also notes Windows path tips (use absolute paths; escape backslashes) and includes screenshots of the Claude UI showing tools once the server is configured. (Model Context Protocol)

What to show (prompts to try in Claude)

Once your server is connected and visible in Claude:

- "Find 3 **Italian** pasta recipes by name and show short summaries with images."
- "List meals that use **chicken** as a main ingredient; include meal IDs so I can ask for details."
- "Show full details for meal id **52772**."
- "Give me one **random** meal with ingredients and measures."

Claude will choose the right tool based on your natural-language question; you can inspect tool calls in the Claude UI. (The workflow of client → tools → results is summarized in the MCP quickstart.) (Model Context Protocol)

1. **Screenshot(s) and code snippet**:
    - Claude Desktop showing your server's tools and a sample query/answer.

- **TheMealDB API (official docs & endpoints)** — test key 1, sample URLs. (themealdb.com)
- **Build an MCP server (official guide)** — end-to-end tutorial, Claude Desktop config, logging rules, and screenshots. (Model Context Protocol)
- **MCP Python SDK (GitHub README)** — installing mcp[cli], and using **mcp dev** with Inspector. (GitHub)
- **MCP Inspector (docs)** — features and usage for debugging servers. (Model Context Protocol)