5COSC022W.2 Client-Server Architectures

Tutorial Week 05: Socket Programming

EXERCISE 1: WEB SERVICES

This lab is a hands-on introduction to RESTful web services using Java. You'll work with a simplified program that simulates a movie RESTful service.

You will:

- Understand the core concepts of RESTful APIs.
- Use HTTP methods (GET and POST) to interact with the API.
- Learn how to use JSON (JavaScript Object Notation) for data exchange.
- Practice using the ReqBin Chrome extension to test API requests.
- Use Maven to manage dependencies in a NetBeans project.
- Complete parts of a pre-built Java application (filling in missing code).

PART 1

- Download the week05.zip file from the Week 5 section on Blackboard.
- Extract (unzip) the contents of the ZIP file to a convenient location on your computer (e.g., your Documents or Desktop folder).
- Open NetBeans.
- Go to File -> Open Project....
- Navigate to the folder where you extracted week05.zip. You should see a folder named tutorial_week05_ex1 (or similar). This is the NetBeans project folder.
- Select the tutorial_week05_ex1 folder and click Open Project.

PART 2

- In the Projects window (usually on the left), expand the tutorial_week05_ex1 project.
- Double-click on the pom.xml file to open it. This file manages project dependencies.
- Find the <dependencies> section. If it's not there, create it inside the <project> element, like this:

```
<project ...>

    <dependencies>
    </dependencies>

</project>
```

Inside the **<dependencies>** section, add the following dependency for the **org.json** library:

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20231013</version>
</dependency>
```

- Important: Check https://mvnrepository.com/artifact/org.json/json for the latest version number and use that if it's different.
- Save the **pom.xml** file. NetBeans should automatically download and install the org.json library. You might see a progress bar.

PART 3

- Open Tutorial_Week05_EX01.java
- In the Projects window, expand tutorial_week05_ex1 -> Source Packages -> com.example.tutorial_week05_ex1.
- Double-click on Tutorial_Week05_EX01.java to open it in the code editor.
- Complete the required parts under the given comments.
- After completing the code go to Build -> Clean and Build Project.
- Compile your Java code.
- Check for any errors.
- Prepare the project for running.
- If you see any errors in the "Output" window, carefully review the error messages and go back to the code to fix them. Common errors include typos, incorrect syntax, or missing semicolons.

PART 4

- In NetBeans, right-click on the Tutorial_Week05_EX01.java file in the Projects window.
- Choose Run File. This starts the embedded HTTP server.
- Look in the NetBeans console (usually at the bottom). You should see a message like **Server started on port 8080**. This confirms that the server is running. If you see errors here, go back to Part 1 and double-check your code and dependencies.

PART 5

- Open the Google Chrome browser.
- Click on the **ReqBin** extension icon in your Chrome toolbar (it looks like a small terminal). If you can't find it, go to **chrome://extensions/** and make sure the ReqBin extension is enabled.

PART 6

TEST THE API (INTERACTIVE TASKS):

- Task 1: Get All Movies
- In ReqBin, set the HTTP method to **GET**.
- Enter the **URL: http://localhost:8080/movies**
- Click **Send**.
- Observe and Record:
    - What is the HTTP status code? (It should be 200 OK if everything is working).
    - What is the response body? (It should be a JSON array of movie objects).

PART 7

GET A SPECIFIC MOVIE

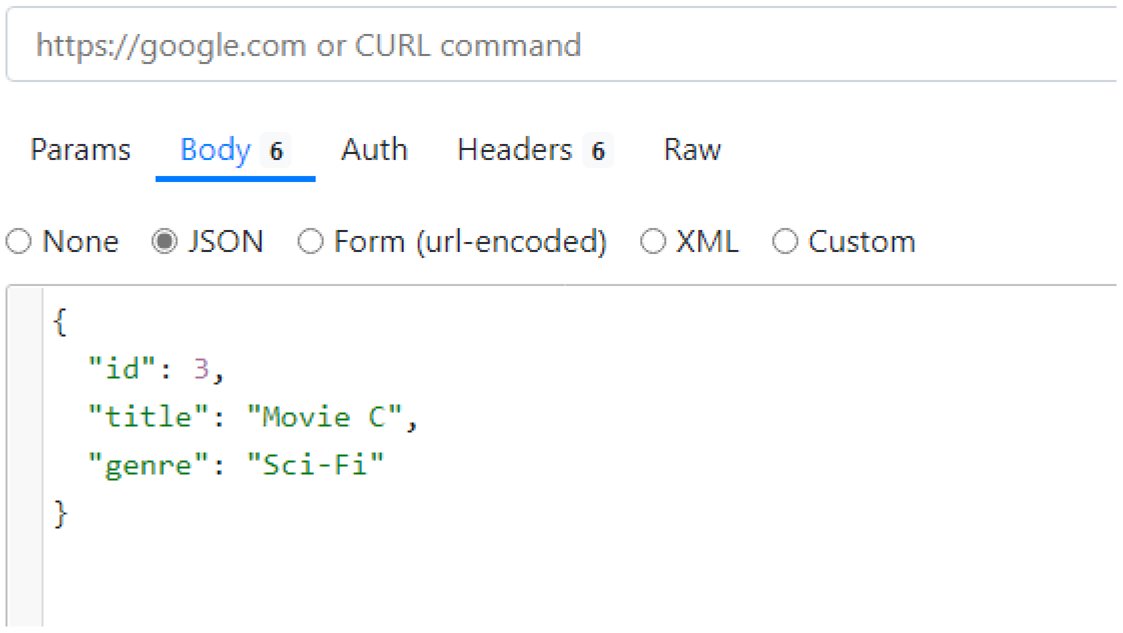
- In ReqBin, set the HTTP method to **GET**.
- Enter the **URL: http://localhost:8080/movies/1** (this requests the movie with ID 1).

- Click **Send**.
- Observe and Record:
    - What is the status code?
    - What is the response body? How is it different from Task 1?
- **Experiment:** Now try requesting a movie ID that *doesn't* exist (e.g., http://localhost:8080/movies/999). What status code and response body do you get *now*?

ADD A NEW MOVIE

- In ReqBin, set the HTTP method to **POST**.
- Enter the **URL: http://localhost:8080/movies**
- Click the Body tab (below the URL bar).
- Select JSON for the Content Type. This tells the server that you're sending JSON data.
- In the request body area, enter the following JSON:

```
{
  "id": 3,
  "title": "Movie C",
  "genre": "Sci-Fi"
}
```



None [✓ JSON] Form (url-encoded) XML Custom

```
{
  "id": 3,
  "title": "Movie C",
  "genre": "Sci-Fi"
}
```

- Click Send.
- Observe and Record:
    - What is the status code? (It should be 201 Created or 200 OK).
    - What is the response body?
    - Then, please change the request to GET, and you will see the new resource is shown in the output!

EXERCISE 2: CHAT PROGRAM

This exercise will guide you through building a simple chat application using Java Sockets. You'll learn how to implement a client-server architecture, handle multiple clients, manage usernames, broadcast messages, and implement private messaging.

INSTRUCTIONS

1. **Download and Import:** Download the ZIP file provided under "Week 05" on Blackboard named Tutorial_Week05_EX02_Questions.zip. This ZIP file contains the skeleton code for the chat application. Import the project into your NetBeans IDE.
2. **Complete the Code:** The provided code has some missing sections. Carefully examine both the Client.java and Server.java files. Your task is to complete the missing code in each class, following the comments and instructions provided within the files. Pay close attention to how the client interacts with the server and how the server manages the clients.
3. **Run the Server:** First, compile and run the Server.java file. This will start the server and make it ready to accept client connections. The server will print a message to the console indicating that it has started. **Keep the server running.**
4. **Run Multiple Clients:** Next, compile and run the Client.java file. You will need to run multiple instances of the client. Each client instance will represent a different user in the chat application. You can do this by right-clicking on the Client.java file in NetBeans and selecting "Run File" multiple times.

HOW TO USE THE CHAT PROGRAM

Once you have the server running and multiple clients connected, you can start chatting:

1. **Enter Username:** When a client starts, a dialog box will appear asking you to enter your username. Choose a unique username and click "OK". This username will be displayed to other users in the chat. If you enter a duplicate username, the server will reject your connection.

2. **View User List:** The client window is divided into two main areas. The larger area in the center displays the chat messages. On the left side, you'll see a list of currently online users. This list is automatically updated as users join or leave the chat.

3. **Send a Message to All:** To send a message to all users in the chat, type your message in the text field at the bottom of the client window and click the "Send" button. Your message will be broadcast to all connected clients, including yourself. The message will be displayed in the chat area along with your username.

4. **Send a Private Message:** To send a private message to a specific user, first select the recipient's username from the user list on the left. Then, type your message in the text field at the bottom and click "Send". The private message will only be delivered to the selected user. The message will be displayed in your chat area indicating that it's a private message to that user.

5. **Exit the Chat:** To leave the chat, click the "Exit Chat" button located in the top right corner of the client window. This will disconnect you from the server, and your username will be removed from the user list of other clients.

IMPORTANT CONSIDERATIONS

- The server needs to be running *before* you start any clients.
- Each client must choose a unique username.
- Private messages are only visible to the sender and the recipient.
- The user list is dynamically updated as users join and leave.

CHAT WITH EACH OTHER!

The provided code uses localhost (127.0.0.1) as the server address. This means that, by default, the server and clients must be running on the *same* machine. To allow multiple students to chat with each other, with one student acting as the server and others as clients, you need to use the actual IP address of the server machine. Here's how to arrange the IP addresses for multi-user chat:

- **Windows:** Open the Command Prompt (cmd) and type ipconfig. Look for the "IPv4 Address" associated with your active network adapter (e.g., Wi-Fi or Ethernet).
- **macOS:** Open the Terminal and type ifconfig. Look for the "inet" address associated with your network interface (e.g., en0 or wlan0).

This IP address is what the clients will use to connect to the server. It will be something like 192.168.x.x (for a local network) or a public IP address if you're connecting over the internet (less common for this type of exercise).

- **Start the Server:** The student hosting the server should run the Server.java code *on their machine*. They do *not* need to change the server code itself; it will listen on all available interfaces.
- **Configure the Clients:** The other students (who will be clients) need to modify the Client.java code *before* running it. Specifically, they need to change the server address in the Client constructor from "localhost" to the IP address of the server machine that they identified in step 1.

```
socket = new Socket("server_ip_address", 12345);
```

For example, if the server's IP address is 192.168.1.100, the client code should look like this:

```
socket = new Socket("192.168.1.100", 12345);
```

- **Run the Clients:** After making this change, the client students can compile and run their Client.java code. They should now be able to connect to the server running on the other student's machine.