**Task 1 Build a Simple HTTP Server (HTTP Server Task)**

In this task, you will build a simple HTTP server that responds to basic GET, POST, and PUT requests.

**Task: Build an HTTP Server**

* **Objective**: Create a Java HTTP server that listens on a specific port and responds to GET, POST, and PUT requests.

**Instructions:**

1. **Create an HTTP Server**:
   * Write a simple HTTP server that listens on port 8000.
   * The server should handle three types of requests:
     + **GET**: Respond with a "Hello, World!" message.
     + **POST**: Accept a JSON payload and respond with a message that includes the received data.
     + **PUT**: Accept a JSON payload, update some internal state (like a variable), and return the updated value.
     + **PATCH**:update concrete field
2. **Create Endpoints**:
   1. GET /hello – Respond with a "Hello, World!" message in plain text.

Output: *Hello, World!*

* 1. POST /user – Accept JSON data (e.g.,

{

“id”: "e63d82e7-1b4a-4b96-8f96-bf39d04e2508"

"name": "John",

"age": 30,

"email": “ [john@example.com](mailto:john@example.com)“,

“status”: “processing”

}

and respond with Received object

* 1. PUT /user– Update internal data based on the received payload and respond with the updated state

Output:

{

"id": null,

"name": null,

"age": 0,

"email": null,

"status": "ready"

}

* 1. PATCH /user

Body: {  
 "status": "active"  
}

Output:

{

"id": " e63d82e7-1b4a-4b96-8f96-bf39d04e2508",

"name": "John",

"age": 30,

"email": "john@example.com",

"status": "active"

}

1. **Implement Error Handling**:
   1. Handle invalid HTTP methods by responding with 405 Method Not Allowed.
   2. Respond with 404 Not Found for invalid paths.

**Task 3 Upload and Save a File (HTTP UPLOAD Task)**

Implement File Upload Handling

* Objective: Create an endpoint /upload that allows clients to upload files to the server.

Instructions:

1. Create the /upload Endpoint:
   * Accept multipart form data or a binary file in the body of the POST request.
   * Save the uploaded file to a directory on the server (e.g., /uploads).
   * Respond with a success message once the file is saved.
2. Add Validation:
   * Ensure only specific file types are allowed (e.g., txt, jpg, png).
   * Return an error for unsupported file types.
3. Respond with Download Link:
   * After uploading the file, provide a download link (e.g., /files/{filename}) that allows users to download the uploaded file.