**MODULE 1(ASSIGNMENT-2)**

**Develop a basic API to handle user registration and retrieval using Flask. The user data will be stored in memory for this assignment, setting the groundwork for more advanced database operations.**

To develop a basic API to handle user registration and retrieval using Flask with in-memory storage, follow these comprehensive steps:

**Step-by-Step Approach**

**Step 1: Set Up the Development Environment**

1. **Create a Project Directory**:
   * Create a directory for your project to keep all files and dependencies organized.
2. **Set Up a Virtual Environment**:
   * Use a virtual environment to manage your project's dependencies.

**Steps to create and activate a virtual environment**:

* + On **Windows**:
    1. Open a command prompt or terminal.
    2. Navigate to your project directory.
    3. Run: python -m venv venv
    4. Activate with: venv\Scripts\activate
  + On **macOS/Linux**:
    1. Open a terminal.
    2. Navigate to your project directory.
    3. Run: python3 -m venv venv
    4. Activate with: source venv/bin/activate

1. **Install Flask**:
   * Install Flask within the virtual environment using pip.
   * Command: pip install flask

**Step 2: Create the Flask Application**

1. **Create the Main Application File**:
   * In your project directory, create a file named app.py.
2. **Initialize the Flask App**:
   * Import Flask from the flask package.
   * Create an instance of the Flask class.

**Step 3: Create In-Memory Storage**

1. **Set Up In-Memory Data Structure**:
   * Use a Python dictionary to store user data. The keys can be user IDs or usernames, and the values can be another dictionary containing user details like name and email.

**Step 4: Define API Routes**

1. **User Registration Route (/register)**:
   * Define a POST route for /register that accepts user details (e.g., username, name, email) in the request body.
   * Validate the input data to ensure required fields are provided.
   * Check for duplicate users before adding to the in-memory storage.
   * Add the user to the in-memory storage and return a success response.
2. **User Retrieval Route (/users/<username>)**:
   * Define a GET route for /users/<username> that retrieves user details by username.
   * Check if the user exists in the in-memory storage.
   * Return the user details in a JSON response if the user exists; otherwise, return an appropriate error message.

**Step 5: Main Entry Point**

1. **Create Main Entry Point**:
   * Ensure the Flask application can be run directly by including an if \_\_name\_\_ == "\_\_main\_\_": block.
   * Include the Flask application run method within this block.

**Step 6: Run and Test the Application**

1. **Run the Flask Development Server**:
   * Start the development server by navigating to your project directory in the terminal and running the command: flask run
   * The server will host your application locally.
2. **Test the Routes**:
   * Use an API testing tool (like Postman) to test the endpoints:
     + **User Registration Route (/register)**:
       - Send a POST request with user details in the request body.
       - Verify that the user is added to the in-memory storage and that a success response is received.
     + **User Retrieval Route (/users/<username>)**:
       - Send a GET request with a username.
       - Verify that the correct user details are returned or an appropriate error message if the user doesn't exist.

**Summary of the Steps**

1. **Environment Setup**:
   * Create a project directory.
   * Set up and activate a virtual environment.
   * Install Flask.
2. **Flask Application Initialization**:
   * Create app.py.
   * Initialize Flask.
3. **In-Memory Storage**:
   * Set up a dictionary to store user data.
4. **API Routes**:
   * Define a POST route for user registration.
   * Define a GET route for user retrieval by username.
5. **Main Entry Point**:
   * Ensure the Flask application can run with if \_\_name\_\_ == "\_\_main\_\_":.
6. **Testing**:
   * Run the Flask development server.
   * Test the registration and retrieval endpoints using an API testing tool.

**Optimization Tips**

* **Validation**: Ensure thorough validation of input data to maintain data integrity.
* **Error Handling**: Handle errors gracefully, providing meaningful messages to the client.
* **Efficiency**: Use a dictionary for O(1) average time complexity for insertions and lookups.
* **Scalability**: This approach uses in-memory storage. For more advanced applications, consider integrating a database.