**[Que-44] - How would you ceate a RESTful API endpoint in Flask that returns JSON data**

Creating a RESTful API endpoint in Flask that returns JSON data involves defining a route and using Flask's jsonify function to return the data in JSON format. Here's a step-by-step guide to creating a simple RESTful API in Flask:

### **Step 1: Install Flask**

If you haven't already installed Flask, you can do so using pip:

pip install flask

### **Step 2: Set Up the Flask Application**

Create a new directory for your project and within it, create a file named app.py.

### **Step 3: Define the RESTful API Endpoint**

In app.py, set up a simple Flask application and define a route that returns JSON data.

from flask import Flask, jsonify  
  
app = Flask(\_\_name\_\_)  
  
# Sample data  
data = [  
 {'id': 1, 'name': 'John Doe', 'email': '[john@example.com](mailto:john@example.com)'},  
 {'id': 2, 'name': 'Jane Doe', 'email': '[jane@example.com](mailto:jane@example.com)'}  
]  
  
@app.route('/api/users', methods=['GET'])  
def get\_users():  
 return jsonify(data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

### **Explanation:**

1. **Import Flask and jsonify**:

from flask import Flask, jsonify

* + Flask: The core Flask class.
  + jsonify: A Flask function that converts Python dictionaries to JSON format.

1. **Create an instance of the Flask class**:

app = Flask(\_\_name\_\_)

* + This creates an instance of the Flask class, initializing the Flask application.

1. **Define sample data**:

python

Copy code

data = [  
 {'id': 1, 'name': 'John Doe', 'email': '[john@example.com](mailto:john@example.com)'},  
 {'id': 2, 'name': 'Jane Doe', 'email': '[jane@example.com](mailto:jane@example.com)'}  
]

* + This is a list of dictionaries that represents the sample data to be returned by the API.

1. **Define the RESTful API endpoint**:

@app.route('/api/users', methods=['GET'])  
def get\_users():  
 return jsonify(data)

* + @app.route('/api/users', methods=['GET']): This decorator defines the route /api/users and specifies that it accepts GET requests.
  + get\_users(): This function is executed when the /api/users endpoint is accessed. It uses jsonify(data) to return the sample data in JSON format.

1. **Run the Flask application**:

if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

* + This condition checks if the script is executed directly (not imported as a module), and if so, it runs the Flask application with debugging enabled.

### **Step 4: Running the Flask Application**

1. Open a terminal or command prompt.
2. Navigate to the directory where app.py is located.
3. Run the following command:

python app.py

1. Open a web browser and go to <http://127.0.0.1:5000/api/users>.

You should see the JSON data displayed in the browser.

### **Step 5: Enhancing the API**

You can add more endpoints and functionalities to your API. For example, you can create endpoints to get a single user by ID, add new users, update users, and delete users.

#### **Get a Single User by ID**

@app.route('/api/users/<int:user\_id>', methods=['GET'])  
def get\_user(user\_id):  
 user = next((item for item in data if item['id'] == user\_id), None)  
 if user:  
 return jsonify(user)  
 else:  
 return jsonify({'message': 'User not found'}), 404

#### **Add a New User**

from flask import request  
  
@app.route('/api/users', methods=['POST'])  
def add\_user():  
 new\_user = request.get\_json()  
 data.append(new\_user)  
 return jsonify(new\_user), 201

#### **Update a User**

@app.route('/api/users/<int:user\_id>', methods=['PUT'])  
def update\_user(user\_id):  
 user = next((item for item in data if item['id'] == user\_id), None)  
 if user:  
 updates = request.get\_json()  
 user.update(updates)  
 return jsonify(user)  
 else:  
 return jsonify({'message': 'User not found'}), 404

#### **Delete a User**

@app.route('/api/users/<int:user\_id>', methods=['DELETE'])  
def delete\_user(user\_id):  
 global data  
 data = [item for item in data if item['id'] != user\_id]  
 return jsonify({'message': 'User deleted'})

### **Full Example with All CRUD Operations**

from flask import Flask, jsonify, request  
  
app = Flask(\_\_name\_\_)  
  
# Sample data  
data = [  
 {'id': 1, 'name': 'John Doe', 'email': '[john@example.com](mailto:john@example.com)'},  
 {'id': 2, 'name': 'Jane Doe', 'email': '[jane@example.com](mailto:jane@example.com)'}  
]  
  
@app.route('/api/users', methods=['GET'])  
def get\_users():  
 return jsonify(data)  
  
@app.route('/api/users/<int:user\_id>', methods=['GET'])  
def get\_user(user\_id):  
 user = next((item for item in data if item['id'] == user\_id), None)  
 if user:  
 return jsonify(user)  
 else:  
 return jsonify({'message': 'User not found'}), 404  
  
@app.route('/api/users', methods=['POST'])  
def add\_user():  
 new\_user = request.get\_json()  
 data.append(new\_user)  
 return jsonify(new\_user), 201  
  
@app.route('/api/users/<int:user\_id>', methods=['PUT'])  
def update\_user(user\_id):  
 user = next((item for item in data if item['id'] == user\_id), None)  
 if user:  
 updates = request.get\_json()  
 user.update(updates)  
 return jsonify(user)  
 else:  
 return jsonify({'message': 'User not found'}), 404  
  
@app.route('/api/users/<int:user\_id>', methods=['DELETE'])  
def delete\_user(user\_id):  
 global data  
 data = [item for item in data if item['id'] != user\_id]  
 return jsonify({'message': 'User deleted'})  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

With this setup, you have a fully functional RESTful API with CRUD operations using Flask and JSON data.