Name : Tirth Chavda
Roll no : 20BCE043
Subject : Microservice Architecture and programming

- Practical-2:
  Aim: Submit here implementation of API calls with REST approach in any language, submit here code, commands and screen captures demonstrating API calls functionalities.

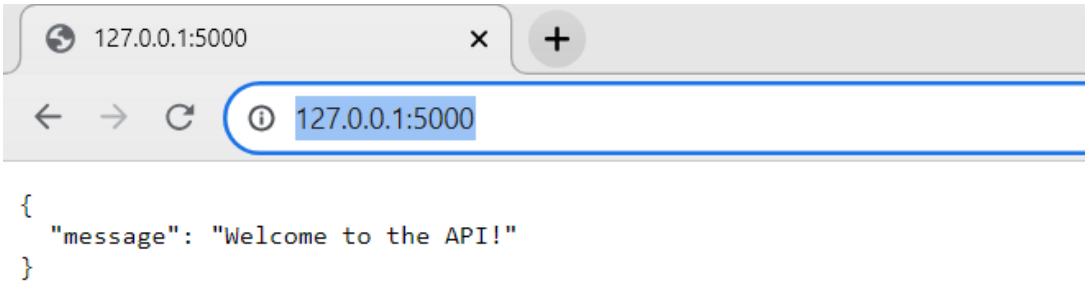
  Code:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# Sample data - you can replace this with your own data or database
integration
data = [
    {'id': 1, 'name': 'John Doe', 'age': 30},
    {'id': 2, 'name': 'Jane Smith', 'age': 25},
]

@app.route('/', methods=['GET'])
def home():
    return jsonify({'message': 'Welcome to the API!'}), 200


@app.route('/hello', methods=['GET'])
def ho():
    return jsonify({'message': 'hello from API'}), 200
# Route to get all items
@app.route('/api/items', methods=['GET'])
def get_items():
    return jsonify(data)

# Route to get a specific item by ID
@app.route('/api/items/<int:item_id>', methods=['GET'])
def get_item(item_id):
    item = next((item for item in data if item['id'] == item_id), None)
    if item:
        return jsonify(item)
    else:
        return jsonify({'message': 'Item not found'}), 404

# Route to create a new item
@app.route('/api/items', methods=['POST'])
def create_item():
    new_item = request.json
```

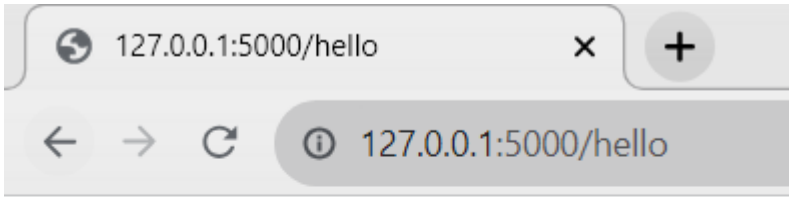
```
    new_item['id'] = len(data) + 1
    data.append(new_item)
    return jsonify(new_item), 201

# Route to update an existing item
@app.route('/api/items/<int:item_id>', methods=['PUT'])
def update_item(item_id):
    item = next((item for item in data if item['id'] == item_id), None)
    if item:
        item.update(request.json)
        return jsonify(item)
    else:
        return jsonify({'message': 'Item not found'}), 404

@app.errorhandler(404)
def not_found(error):
    return jsonify({'message': 'Resource not found'}), 404

# Route to delete an item by ID
@app.route('/api/items/<int:item_id>', methods=['DELETE'])
def delete_item(item_id):
    global data
    data = [item for item in data if item['id'] != item_id]
    return jsonify({'message': 'Item deleted successfully'}), 200

if __name__ == '__main__':
    app.run(debug=True)
```

output:

1. http://127.0.0.1:5000/

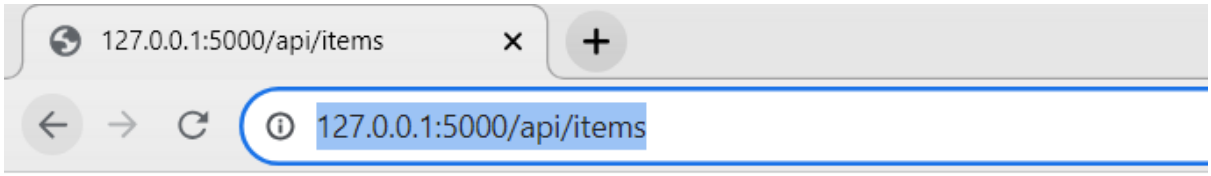


2. http://127.0.0.1:5000/hello



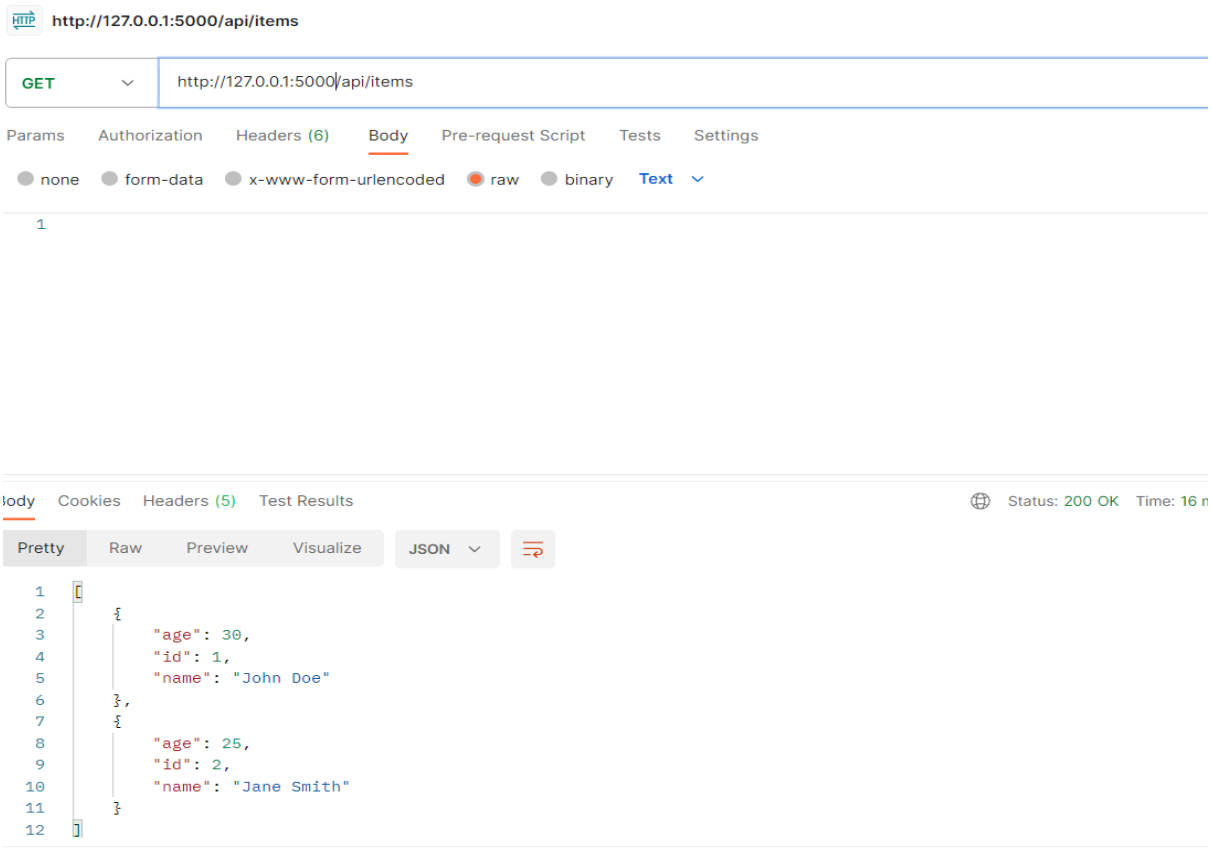

```
{
  "message": "hello from API"
}
```


3. http://127.0.0.1:5000/api/items

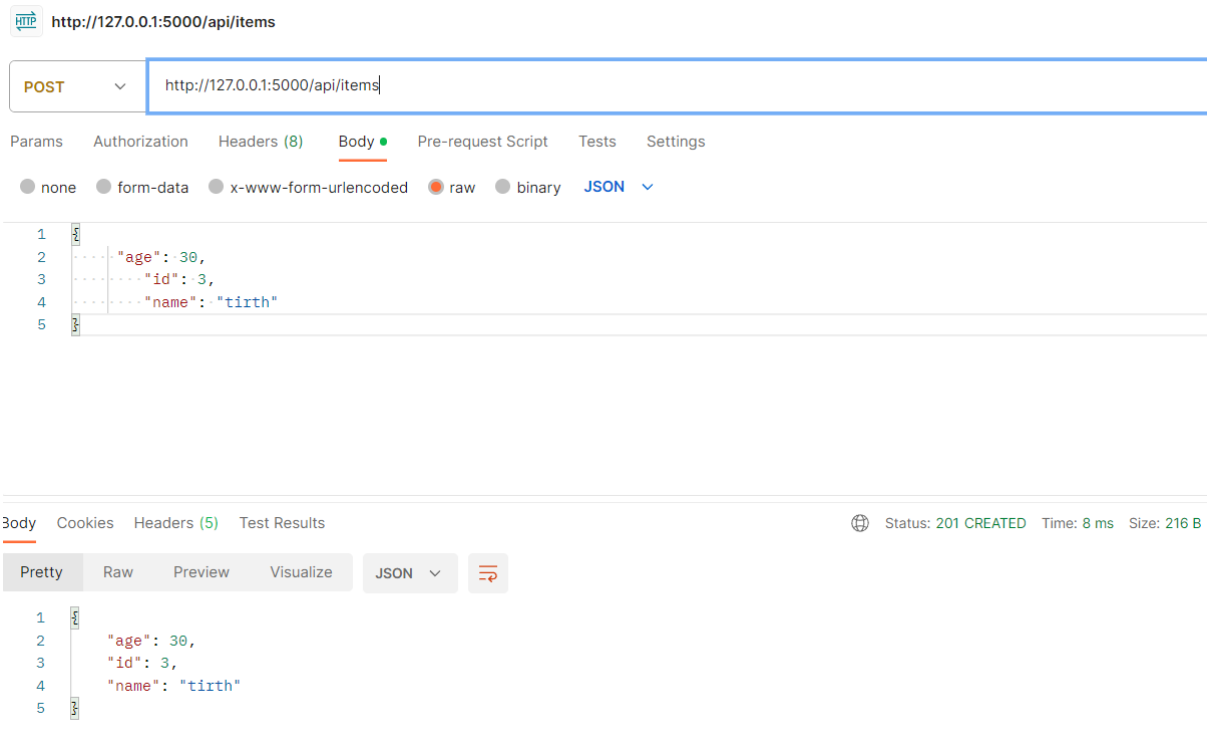



```
[
  {
    "age": 30,
    "id": 1,
    "name": "John Doe"
  },
  {
    "age": 25,
    "id": 2,
    "name": "Jane Smith"
  }
]
```

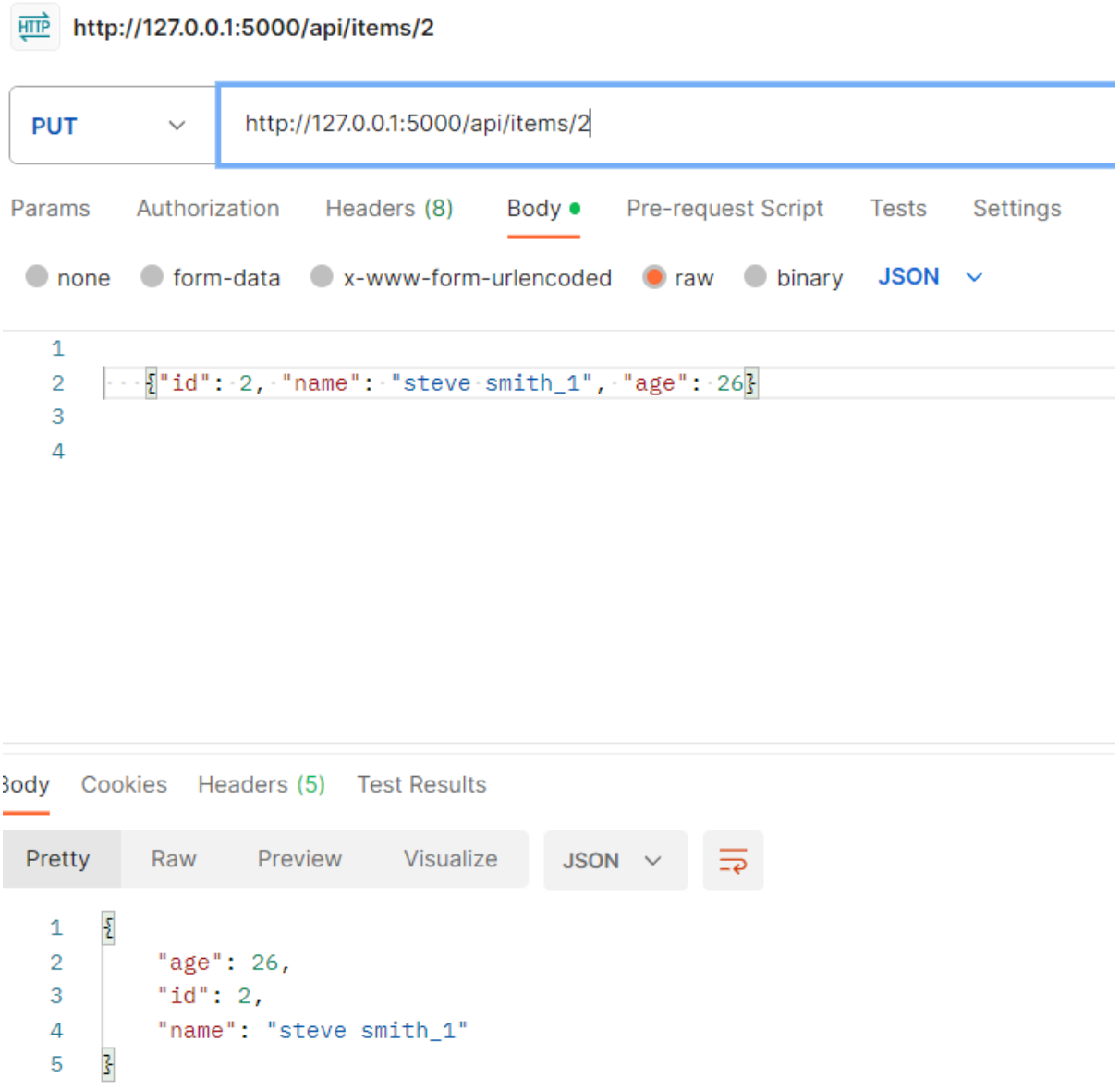

4. API Testing on Postman for http://127.0.0.1:5000/api/items (GET Request)

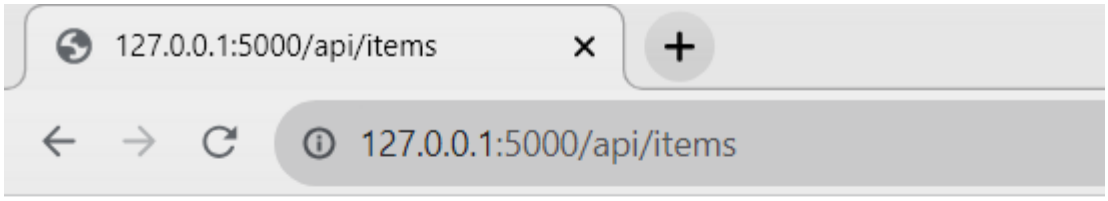


5. Post Request for http://127.0.0.1:5000/api/items to add new member.



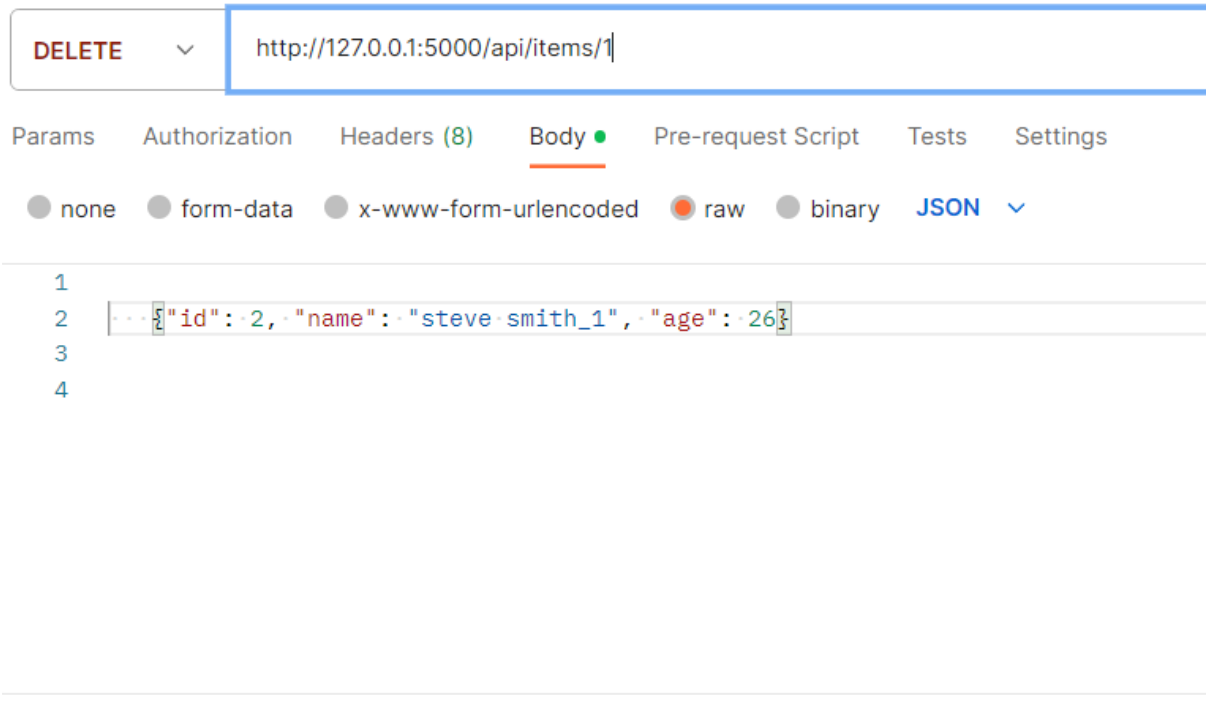
1. Put Request for http://127.0.0.1:5000/api/items/2 to update details of any member



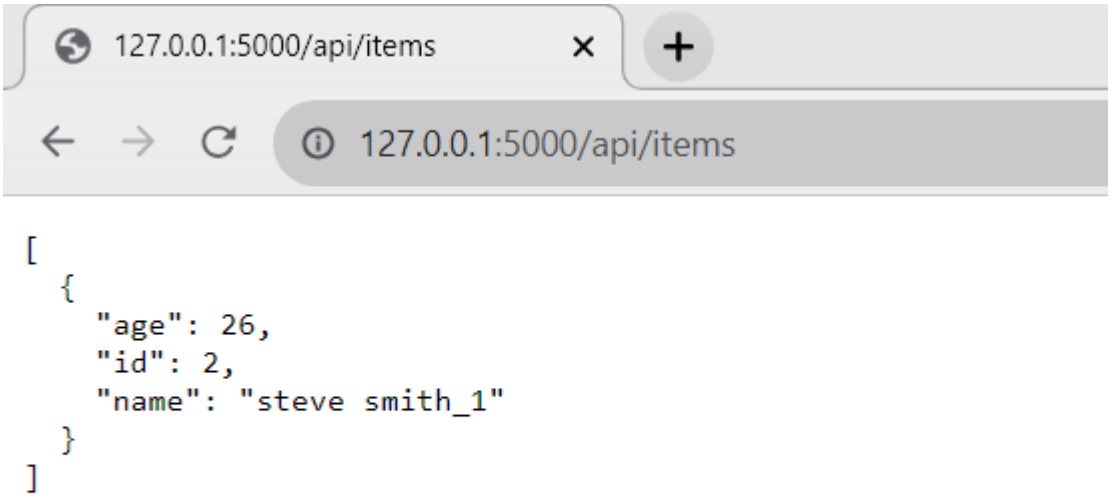



```
[
  {
    "age": 30,
    "id": 1,
    "name": "John Doe"
  },
  {
    "age": 26,
    "id": 2,
    "name": "steve smith_1"
  }
]
```


6. Delete Request for http://127.0.0.1:5000/api/items/1 to delete any member.


DELETE
http://127.0.0.1:5000/api/items/1
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
JSON
{"id": 2, "name": "steve smith_1", "age": 26}


Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Item deleted successfully"
}


127.0.0.1:5000/api/items
127.0.0.1:5000/api/items
[
{
"age": 26,
"id": 2,
"name": "steve smith_1"
}
]

Conclusion: After performing this practical, I learn about how to do API testing on POSTMAN platform and how to write code for the same.