CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+
Accredited University

Worksheet No. - 4

Student Name: Devanand Utkarsh **UID:** 24MCA20454
Branch: MCA **Section/Group:** 6 (B)
Semester: II **Date of Performance:** 25-02-2024
Subject Name: Software Testing Lab **Subject Code:** 24CAH-654

Aim/Overview of the practical: Use Rest API using POSTMAN for API testing and development.

Objective: The objective of this guide is to demonstrate how to use Postman for testing and developing REST APIs. Postman is a powerful tool for sending HTTP requests, receiving responses, and validating the behavior of REST APIs. It allows you to perform various operations such as GET, POST, PUT, DELETE, and more, making it suitable for functional testing, debugging, and API development.

Input/Apparatus Used: Postman for testing Rest API, API URL with endpoints.

Procedure/Algorithm/Code:

1. Download and Install Postman:

- Download and install Postman from the official website.
- Open Postman after installation.

2. Create a New Request:

➢ Click the dropdown next to the URL field and select:
- **GET** → Retrieve data from the server.
- **POST** → Send data to create a new resource.
- **PUT** → Update an existing resource.
- **DELETE** → Remove a resource.

⇨ **Enter the Request URL:**
○ Type the API endpoint URL that you want to test in the URL bar.

- **GET -** https://api.restful-api.dev/objects
- **POST-** https://api.restful-api.dev/objects

- **UPDATE-** https://api.restful-api.dev/objects/ff808181932badb601953b9e591814cf

- **DELETE-** https://api.restful-api.dev/objects/ff808181932badb601953b9b777214cb

Add Request Parameters (Optional)

- **For GET requests, you can add query parameters in the "Params" tab.**
- **For POST/PUT requests, go to the "Body" tab:**
  - **Select raw and choose JSON.**
  - **Enter JSON data (example for creating a user):**

⇨ **Example**

for a POST request:	**for a UPDATE request:**
{ "name": "Apple MacBook Pro 110", "data": { "year": 20192, "price": 1849.99, "CPU model": "Intel Core i9", "Hard disk size": "1 TB" } }	{ "name": "Apple MacBook Pro 16666", "data": { "year": 2019, "price": 2049.99, "CPU model": "Intel Core i9", "Hard disk size": "1 TB", "color": "silver" } }

1. Send the Request:

- After configuring your request, click on the Send button in Postman.
- Postman will send the request to the server and display the response in the bottom section of the window. Status code (e.g., **200 OK**, **201 Created**, **404 Not Found**)Response body (JSON or other format).

2. Review the Response:

Example of a Response for a GET request:

```
[
 {
        "id": "1",
        "name": "Google Pixel 6 Pro",
        "data": {
           "color": "Cloudy White",
           "capacity": "128 GB"
        }
    },
    {
        "id": "2",
        "name": "Apple iPhone 12 Mini, 256GB, Blue",
        "data": null
    },
]
```

6. Add Assertions and Tests:

Postman allows you to add tests and assertions to verify the API behaviour.

- Go to the Tests tab:
- This is where you can write JavaScript code to assert the expected results.

Example –

GET

```
// Updated response time test to 300ms
pm.test("Response status code is 200", function () {
   pm.response.to.have.status(200);
});

pm.test("Response time is less than 300ms", function () {
   pm.expect(pm.response.responseTime).to.be.below(3000);
});

pm.test("Response schema validation", function () {
   const responseData = pm.response.json();
```

```
  pm.expect(responseData).to.be.an('array').that.is.not.empty;

  responseData.forEach(function(object) {
    pm.expect(object).to.be.an('object');
    pm.expect(object).to.have.property('id').that.is.a('string');
    pm.expect(object).to.have.property('name').that.is.a('string');



  });
});
```

POST

```
pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Content-Type header is application/json", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});

pm.test("Response time is within an acceptable range", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Response has the required fields", function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('id');
    pm.expect(responseData).to.have.property('name');
    pm.expect(responseData).to.have.nested.property('data.price');
    pm.expect(responseData).to.have.nested.property('data.CPU model');
});
pm.test("Year in data object is a non-negative integer", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.data.year).to.be.a('number');
    pm.expect(responseData.data.year).to.be.at.least(0, "Year should be a non-negative integer");
});

pm.test("Price in data object is a non-negative integer", function () {
    const responseData = pm.response.json();

});
```

UPDATE

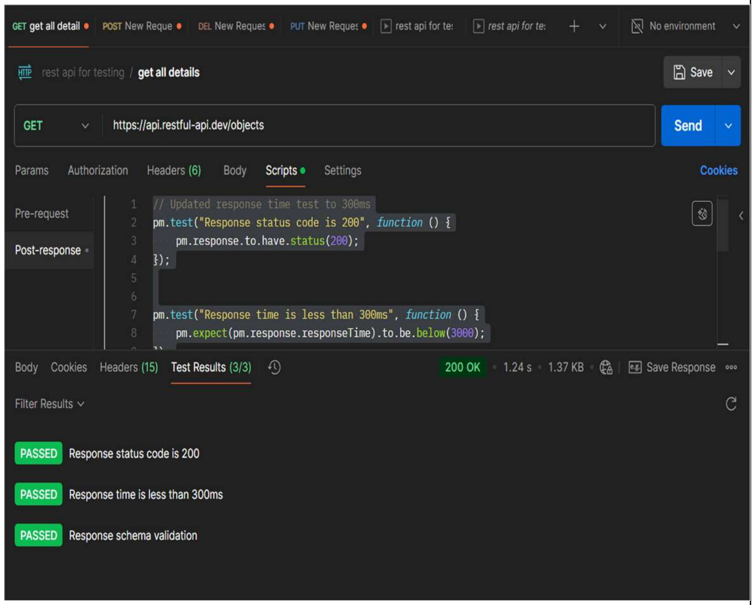
```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Validate the response schema for required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData).to.have.property('id');
    pm.expect(responseData).to.have.property('name');
    pm.expect(responseData).to.have.property('updatedAt');
    pm.expect(responseData).to.have.property('data');
});

pm.test("Year and price in data object are non-negative integers", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.data).to.exist.and.to.be.an('object');
    pm.expect(responseData.data.year).to.exist.and.to.be.a('number').and.to.be.at.least(0, "Year should be
non-negative");
    pm.expect(responseData.data.price).to.exist.and.to.be.a('number').and.to.be.at.least(0, "Price should be
non-negative");
});

pm.test("Data object properties should be non-empty strings", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.data).to.exist.and.to.be.an('object');
    pm.expect(responseData.data).to.have.property('CPU
model').that.is.a('string').and.to.have.lengthOf.at.least(1, "Value should not be empty");
    pm.expect(responseData.data).to.have.property('Hard disk
size').that.is.a('string').and.to.have.lengthOf.at.least(1, "Value should not be empty");
    pm.expect(responseData.data).to.have.property('color').that.is.a('string').and.to.have.lengthOf.at.least(1,
"Value should not be empty");
});
```

DELETE

```
pm.test("Response status code is 404", function () {
  pm.expect(pm.response.code).to.equal(404);
});

pm.test("Response has the required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.error).to.exist;
});

pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});

pm.test("Content-Type header is application/json", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});

pm.test("Error field is a non-empty string", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.error).to.be.a('string').and.to.have.lengthOf.at.least(1, "Error field should be
a non-empty string");
});
```

Output:

GET

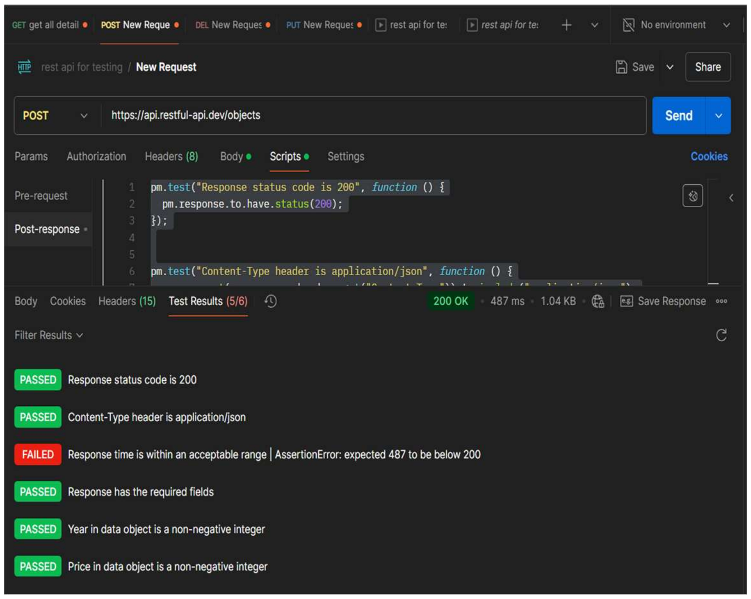


```
GET  https://api.restful-api.dev/objects

// Updated response time test to 300ms
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});


pm.test("Response time is less than 300ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(3000);
```

Test Results (3/3)   200 OK · 1.24 s · 1.37 KB

```
PASSED Response status code is 200
PASSED Response time is less than 300ms
PASSED Response schema validation
```

POST

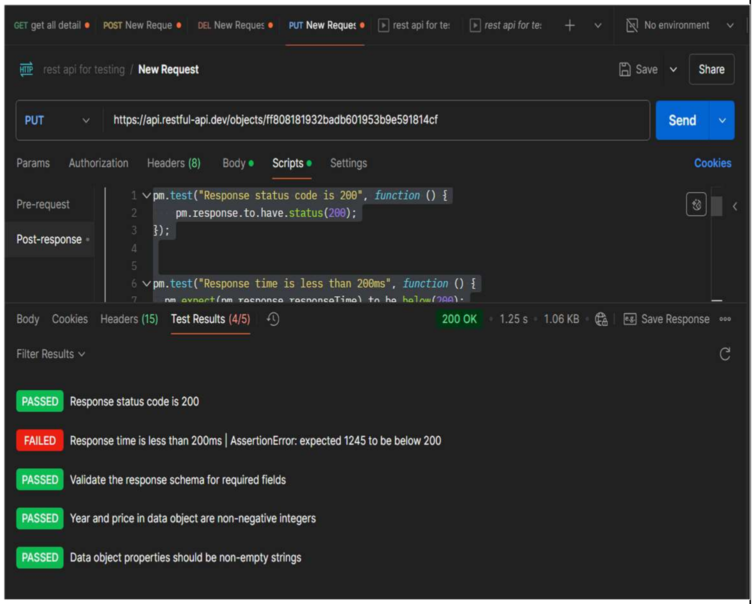


```
POST  https://api.restful-api.dev/objects

pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});


pm.test("Content-Type header is application/json", function () {
```

Test Results (5/6)   200 OK · 487 ms · 1.04 KB

```
PASSED Response status code is 200
PASSED Content-Type header is application/json
FAILED Response time is within an acceptable range | AssertionError: expected 487 to be below 200
PASSED Response has the required fields
PASSED Year in data object is a non-negative integer
PASSED Price in data object is a non-negative integer
```

UPDATE

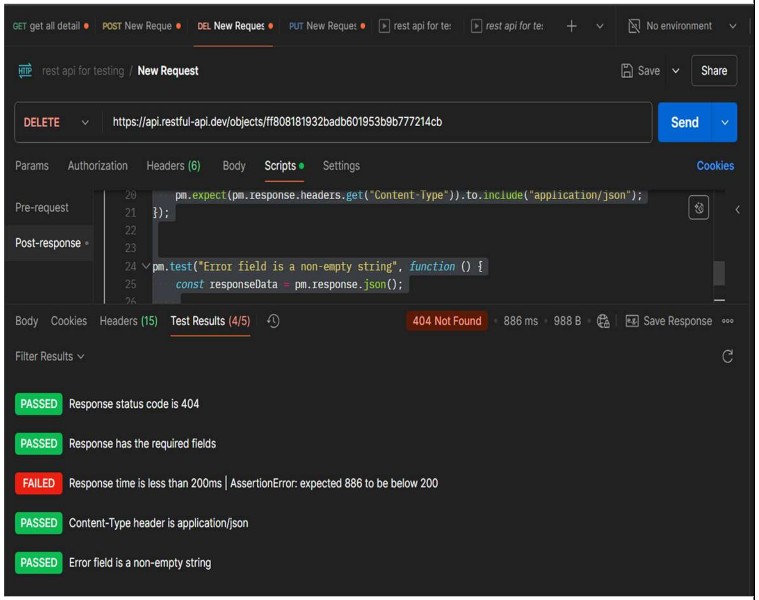


```
PUT  https://api.restful-api.dev/objects/ff808181932badb601953b9e591814cf

pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});


pm.test("Response time is less than 200ms", function () {
```

Test Results (4/5)   200 OK · 1.25 s · 1.06 KB

```
PASSED Response status code is 200
FAILED Response time is less than 200ms | AssertionError: expected 1245 to be below 200
PASSED Validate the response schema for required fields
PASSED Year and price in data object are non-negative integers
PASSED Data object properties should be non-empty strings
```

DELETE

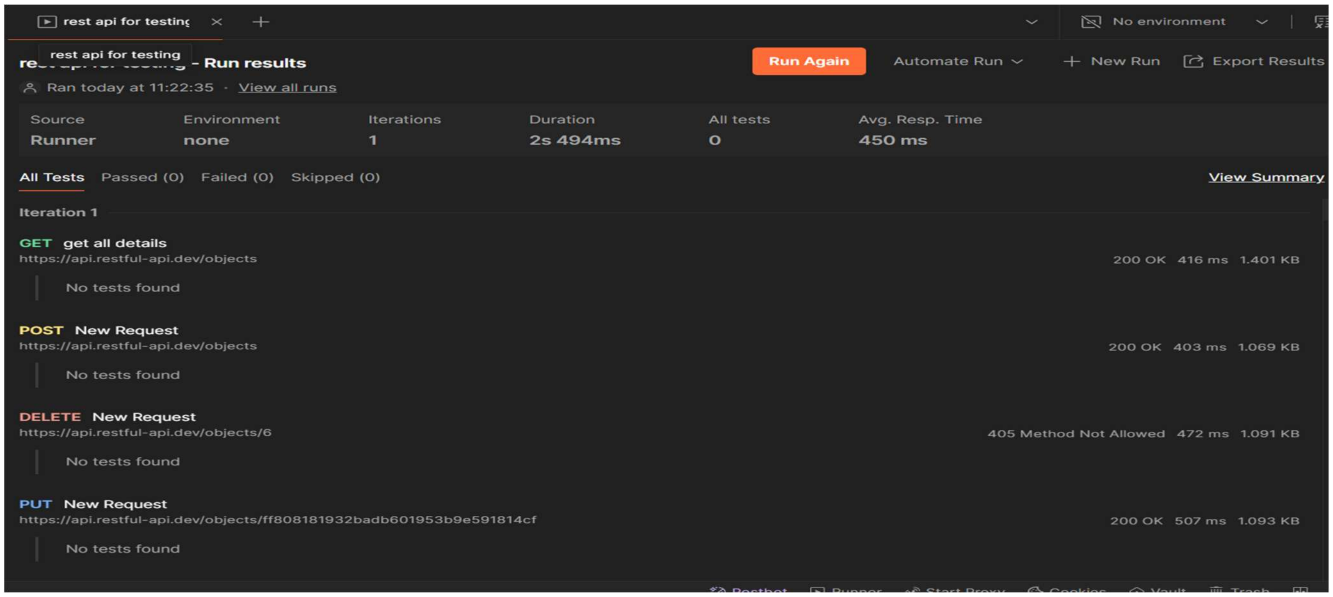


```
DELETE  https://api.restful-api.dev/objects/ff808181932badb601953b9b777214cb

    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});


pm.test("Error field is a non-empty string", function () {
    const responseData = pm.response.json();
```

Test Results (4/5)   404 Not Found · 886 ms · 988 B

```
PASSED Response status code is 404
PASSED Response has the required fields
FAILED Response time is less than 200ms | AssertionError: expected 886 to be below 200
PASSED Content-Type header is application/json
PASSED Error field is a non-empty string
```

⇨ Test Results:



rest api for testing - Run results
Ran today at 11:22:35 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	2s 494ms	0	450 ms

All Tests Passed (0) Failed (0) Skipped (0) View Summary

Iteration 1

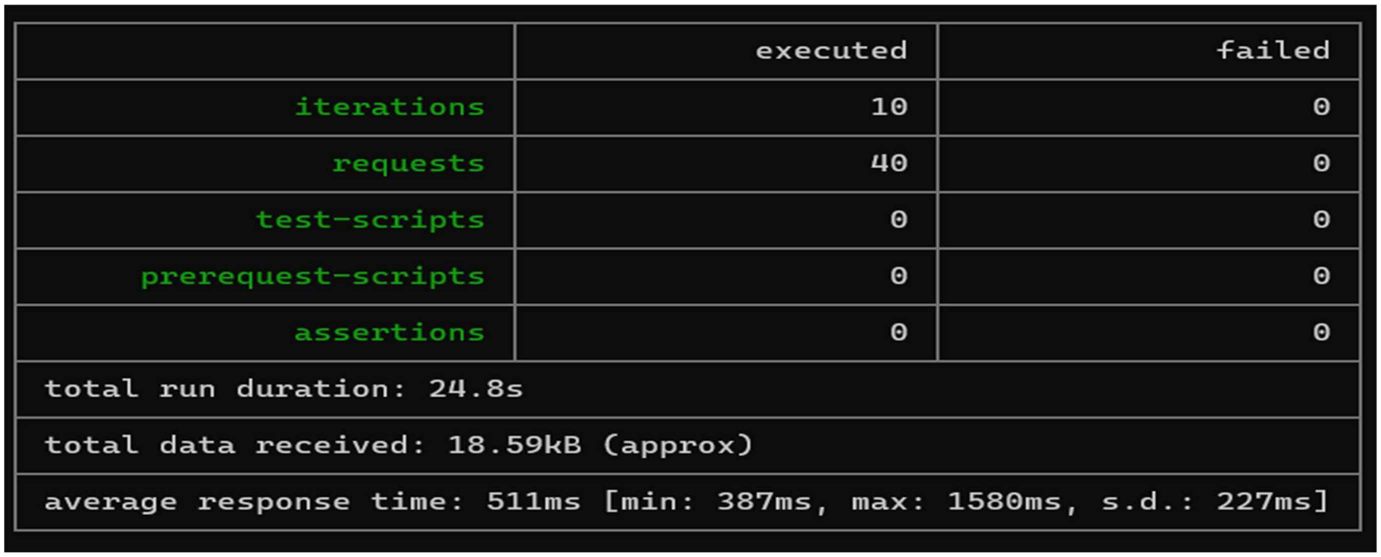
GET get all details
https://api.restful-api.dev/objects — 200 OK 416 ms 1.401 KB
No tests found

POST New Request
https://api.restful-api.dev/objects — 200 OK 403 ms 1.069 KB
No tests found

DELETE New Request
https://api.restful-api.dev/objects/6 — 405 Method Not Allowed 472 ms 1.091 KB
No tests found

PUT New Request
https://api.restful-api.dev/objects/ff808181932badb601953b9e591814cf — 200 OK 507 ms 1.093 KB
No tests found

Load_Test Result:



```
┌─────────────────────────┬────────────────────┬───────────────────┐
│                         │           executed │            failed │
├─────────────────────────┼────────────────────┼───────────────────┤
│              iterations │                 10 │                 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│                requests │                 40 │                 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│            test-scripts │                  0 │                 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│      prerequest-scripts │                  0 │                 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│              assertions │                  0 │                 0 │
├─────────────────────────┴────────────────────┴───────────────────┤
│ total run duration: 24.8s                                        │
├──────────────────────────────────────────────────────────────────┤
│ total data received: 18.59kB (approx)                            │
├──────────────────────────────────────────────────────────────────┤
│ average response time: 511ms [min: 387ms, max: 1580ms, s.d.: 227ms] │
└──────────────────────────────────────────────────────────────────┘
```

Learning outcome:

1. **Purpose of the Guide:** Demonstrates how to use Postman for testing and developing REST APIs.

2. **Operations Supported:** Used GET, POST, PUT, DELETE, and other HTTP methods.

3. **Postman as a Tool:** A powerful tool for sending HTTP requests and receiving responses.