API Testing Documentation

This document outlines testing a RESTful API using Postman and automated scripts. The API used for demonstration is JSONPlaceholder..

Objectives
✔ Verify API endpoints work as expected.
✔ Test error handling and edge cases.
✔ Validate response structure and status codes.
✔ Ensure security and performance best practices.

1. API Details

Field	Value
Base URL	https://jsonplaceholder.typicode.com
Authentication	None (Public API)
Format	JSON

---

2. Test Cases

3.1 `GET /posts`

Description: Retrieve all posts.

Request: GET /posts

Expected Response:

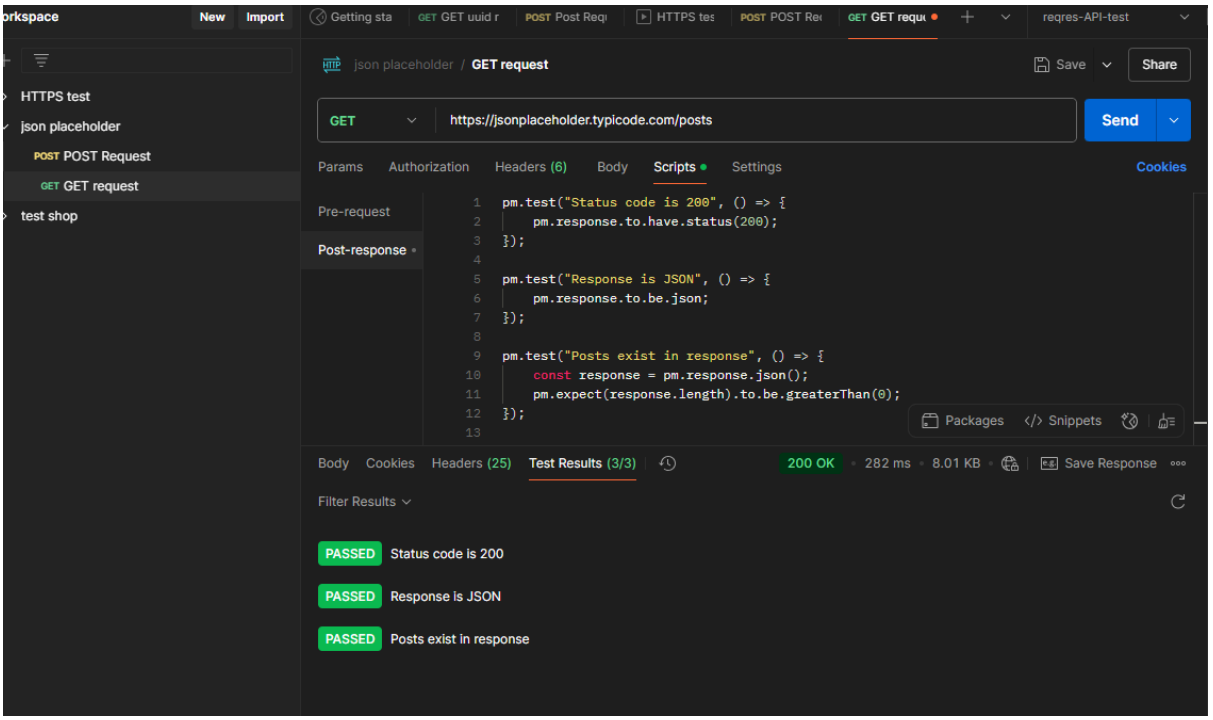
- **Status Code:** `200 OK`
- **Body:** JSON array of posts.

Postman Test Script:

```
pm.test("Status code is 200", () => {
    pm.response.to.have.status(200);
});

pm.test("Response is JSON", () => {
    pm.response.to.be.json;
});

pm.test("Posts exist in response", () => {
    const response = pm.response.json();
    pm.expect(response.length).to.be.greaterThan(0);
});
```



3.2 `GET /posts/{id}`

Description: Fetch a single post by ID.

Request: GET /posts/1

Expected Response:

- **Status Code:** `200 OK`

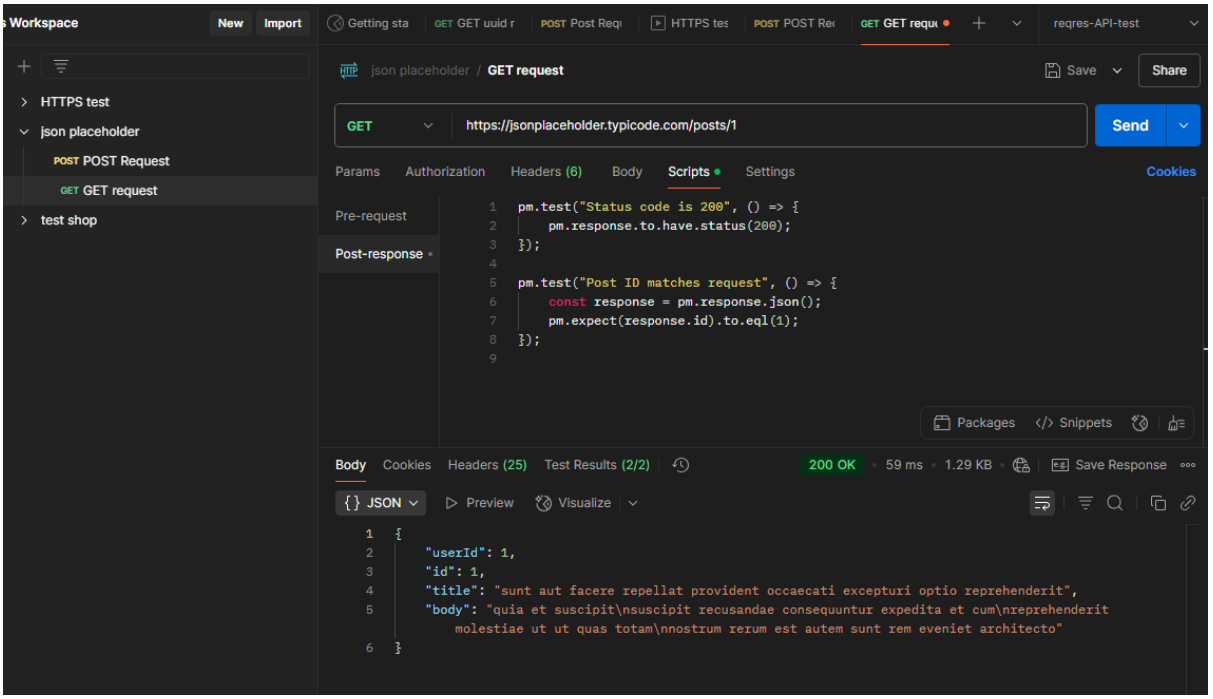
- **Body:**


```
{
   "userId": 1,
   "id": 1,
   "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
   "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
}
```


Postman Test Script:

```
pm.test("Status code is 200", () => {
    pm.response.to.have.status(200);
});

pm.test("Post ID matches request", () => {
    const response = pm.response.json();
    pm.expect(response.id).to.eql(1);
});
```



3.3 `POST /posts`

Description: Create a new post.

Request: POST /posts

Headers: { "Content-Type": "application/json" }
Body:
{
  "title": "New Post",
  "body": "shopping cart.",
  "userId": 101
}

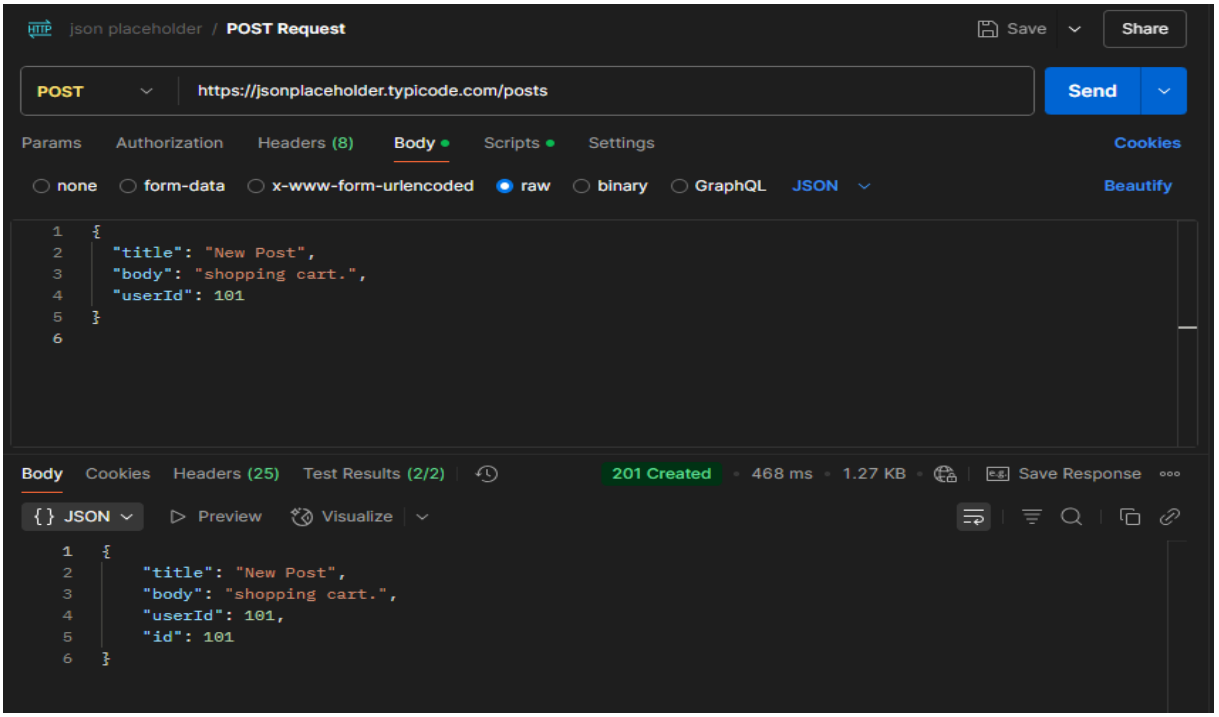
Expected Response:

- **Status Code:** `201 Created`
- **Body:** The newly created post with a unique ID.

Postman Test Script:

```
pm.test("Status code is 201", () => {
    pm.response.to.have.status(201);
});

pm.test("New post has an ID", () => {
    const response = pm.response.json();
    pm.expect(response.id).to.exist;
});
```



3.4 `PUT /posts/{id}`

Description: Update an existing post.

Request: PUT /posts/101

Headers: { "Content-Type": "application/json" }
Body:

```
{
  "id": 101,
  "title": "Updated Title",
  "body": "Updated body text.",
  "userId": 5
}
```

Expected Response:

- **Status Code:** `200 OK`
- **Body:** The updated post.

Postman Test Script:

```
pm.test("Status code is 200", () => {
    pm.response.to.have.status(200);
});

pm.test("Title was updated", () => {
    const response = pm.response.json();
    pm.expect(response.title).to.eql("Updated Title");
});
```

3.5 `DELETE /posts/{id}`

Description: Delete a post.

Request: DELETE /posts/101

Expected Response:

- **Status Code:** `204 No Content`
- **Body:** Empty or confirmation message.

Postman Test Script:

```
pm.test("Status code is 200", () => {
    pm.response.to.have.status(200);
});
```

4. Negative Test Cases

Test Case	Request	Expected Status
Invalid Endpoint	`GET /invalid`	`404 Not Found`
Missing Required Field	`POST /posts` (without `userId`)	`400 Bad Request`
Invalid ID Format	`GET /posts/abc`	`404 Not Found`

Next Steps

- **Add Performance Testing:**
    - Use tools like **JMeter** to evaluate:
        - Response time under load
        - Throughput and latency

- **Implement Security Testing:**
    - Use **OWASP ZAP** to test for:
        - SQL Injection
        - Broken Authentication
        - Sensitive Data Exposure