

Follow



Tech with
Jatin



How to Test **API Effectively**

in



POSTMAN

Index

Section 1

Functionality

Section 2

Compatibility

Section 3

Performance

Section 4

Error Handling

Section 5

Security



Functionality



Check for :

- ✓ Correct **Response**
- ✓ Correct **Status Code**
- ✓ Schema **Validation**



Snippet :

```
pm.test("Status code is 200", function () {  
    pm.response.to.have.status(200);  
});  
  
// Example: Check if response time is within limit (optional)  
pm.test("Response time is less than 500ms", function () {  
    pm.expect(pm.response.responseTime).to.be.below(500);  
});
```

Correct Response & Status Code



Snippet :

```
const schema = {
  "type": "object",
  "properties": {
    "id": {
      "type": "integer"
    },
    "name": {
      "type": "string"
    },
    "email": {
      "type": "string",
      "format": "email"
    }
  },
  "required": ["id", "name", "email"]
};

pm.test("Response schema is valid", function () {
  pm.response.to.have.jsonSchema(schema);
});

// Example: Check specific field in the response
pm.test("Response contains the field 'name'", function () {
  const jsonData = pm.response.json();
  pm.expect(jsonData).to.have.property("name");
});

// Example: Check if field matches a specific value
pm.test("Name field is correct", function () {
  const jsonData = pm.response.json();
  pm.expect(jsonData.name).to.eql("John Doe");
});
```

Schema Validation



Compatibility



Check for :



API with different client libraries and platforms to ensure that it works as expected.



Snippet :

```
pm.test("Check API version compatibility", function () {
  var version = pm.request.headers.get('Version') || pm.request.url.path[1];
  // Check header or URL for version
  var response = pm.response.json(); // Assuming the response is JSON

  console.log("Testing with API version: " + version);

  // Test based on API version
  if (version === "v1") {
    pm.expect(response).to.have.property('data_v1'); // Ensure v1 response format
  } else if (version === "v2") {
    pm.expect(response).to.have.property('data_v2'); // Ensure v2 response format
  }

  // Additional response status and format checks
  pm.expect(pm.response.code).to.eql(200); // Ensure successful response
  pm.expect(pm.response.headers.get('Content-Type')).to.include('application/json');
  // Ensure JSON response
});
```



Performances



Check for :



The API's response times.



Snippet :

```
// Initialize or increment the count of requests
var requestCount = pm.environment.get("requestCount") || 0;
requestCount++;
pm.environment.set("requestCount", requestCount);

// Add the current response time to the total response time
var totalTime = pm.environment.get("totalTime") || 0;
totalTime += pm.response.responseTime;
pm.environment.set("totalTime", totalTime);

// Calculate the average response time
if (requestCount > 1) {
  var averageTime = totalTime / requestCount;
  console.log("Average response time after " + requestCount + " requests: " + averageTime + " ms");
}

if (requestCount > 1) {
  pm.test("Average response time is less than 500ms", function () {
    pm.expect(averageTime).to.be.below(500);
  });
}
```




Error Handling



Check for :



API handles errors gracefully and provides useful error messages to its users



Snippet :

```
pm.test("Error response contains message and code", function () {
  var jsonData = pm.response.json();

  pm.expect(jsonData).to.have.property('error');
  pm.expect(jsonData).to.have.property('message');
  pm.expect(jsonData).to.have.property('code');
});
```

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

pm.test("Status code is 400 (Bad Request)", function () {
  pm.response.to.have.status(400); // For bad requests, such as invalid input
});

pm.test("Status code is 401 (Unauthorized)", function () {
  pm.response.to.have.status(401); // For unauthorized access
});

pm.test("Status code is 404 (Not Found)", function () {
  pm.response.to.have.status(404); // For missing resources
});

pm.test("Status code is 500 (Internal Server Error)", function () {
  pm.response.to.have.status(500); // For server errors
});
```



Security



Check for :

- ✓ Injection attacks and unauthorized access.
- ✓ Authentication and authorization mechanisms



Snippet :

```
// Malicious SQL payloads
let sqlInjectionPayloads = [
  "' OR '1'='1'",
  "'; DROP TABLE users; --",
  "' OR '1'='1' --",
  "' OR 'x'='x'",
  "' UNION SELECT NULL, NULL, NULL --"
];

// Iterate through payloads to test SQL injection
pm.test("SQL Injection Check", function () {
  let response = pm.response.json();

  // Look for typical error responses that may indicate vulnerability
  let errorPatterns = [
    "syntax error", // Common SQL syntax error
    "unhandled exception",
    "SQLSTATE", // SQL error codes
    "ORA-", // Oracle DB error codes
    "Microsoft OLE DB", // MS SQL error
    "You have an error in your SQL syntax", // MySQL
    "Warning: mysql", // MySQL
    "PostgreSQL", // PostgreSQL
  ];

  // Loop through error patterns and assert that they do not appear in the response
  errorPatterns.forEach(function(pattern) {
    pm.expect(pm.response.text()).to.not.include(pattern, `Response should not contain SQL error: ${pattern}`);
  });
});
```