Postman API Testing Cheat Sheet - By Naveen Automation Labs

Basic Request Methods

HTTP Methods

- **GET** - Retrieve data
- **POST** - Create new resource
- **PUT** - Update entire resource
- **PATCH** - Partial update
- **DELETE** - Remove resource
- **HEAD** - Get headers only
- **OPTIONS** - Check allowed methods

Request Structure

```
Method: GET/POST/PUT/DELETE
URL: https://api.naveenautomationlabs.com/users
Headers: Content-Type, Authorization, etc.
Body: JSON/Form data/Raw text
```

Environment Variables

Setting Variables

- **Global**: Available across all collections
- **Collection**: Available within specific collection
- **Environment**: Switch between dev/staging/prod
- **Local**: Current request session only

Variable Syntax

javascript

```javascript
// Set variable
pm.environment.set("token", "abc123");
pm.globals.set("baseUrl", "https://api.naveenautomationlabs.com");
pm.collectionVariables.set("userId", "12345");

// Get variable
{{variableName}}
pm.environment.get("token")
```

Authentication

Bearer Token

```
Authorization: Bearer {{token}}
```

API Key

```
Header: X-API-Key: {{apiKey}}
Query Param: ?api_key={{apiKey}}
```

Basic Auth

```
Authorization: Basic base64(username:password)
```

OAuth 2.0

```javascript
// In pre-request script
pm.sendRequest({
  url: '{{authUrl}}',
  method: 'POST',
  header: { 'Content-Type': 'application/x-www-form-urlencoded' },
  body: {
    mode: 'urlencoded',
    urlencoded: [
      {key: 'grant_type', value: 'client_credentials'},
      {key: 'client_id', value: '{{clientId}}'},
      {key: 'client_secret', value: '{{clientSecret}}'}
    ]
  }
}, function (err, res) {
  pm.environment.set("token", res.json().access_token);
});
```

Pre-Request Scripts

Common Use Cases

```javascript
// Generate random data
pm.environment.set("randomEmail", pm.variables.replaceIn("{{$randomEmail}}"));
pm.environment.set("timestamp", Date.now());

// Generate UUID
const uuid = require('uuid');
pm.environment.set("requestId", uuid.v4());

// Set dynamic headers
pm.request.headers.add({
  key: 'X-Request-ID',
  value: pm.environment.get("requestId")
});

// Conditional logic
if (pm.environment.get("env") === "production") {
  pm.environment.set("baseUrl", "https://api.prod.com");
} else {
  pm.environment.set("baseUrl", "https://api.dev.com");
}
```

Test Scripts (Assertions)

Status Code Tests

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

pm.test("Status code is in 200 range", function () {
  pm.expect(pm.response.code).to.be.within(200, 299);
});
```

Response Time Tests

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

Header Tests

javascript

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

pm.test("Has required headers", function () {
    pm.expect(pm.response.headers.has("X-RateLimit-Limit")).to.be.true;
});
```

JSON Response Tests

javascript

```javascript
pm.test("Response is valid JSON", function () {
    pm.response.to.be.json;
});

pm.test("Response has required fields", function () {
    const jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property('id');
    pm.expect(jsonData).to.have.property('name');
    pm.expect(jsonData.email).to.be.a('string');
});

pm.test("Array contains items", function () {
    const jsonData = pm.response.json();
    pm.expect(jsonData.users).to.be.an('array');
    pm.expect(jsonData.users).to.have.length.above(0);
});
```

Schema Validation

javascript

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

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

Advanced Assertions

javascript

```javascript
pm.test("User ID matches request", function () {
  const jsonData = pm.response.json();
  pm.expect(jsonData.id).to.equal(parseInt(pm.environment.get("userId")));
});

pm.test("Response contains no sensitive data", function () {
  const responseText = pm.response.text();
  pm.expect(responseText).to.not.include("password");
  pm.expect(responseText).to.not.include("ssn");
});
```

Data Extraction

Save Response Data

```javascript
// Save to environment variables
const jsonData = pm.response.json();
pm.environment.set("userId", jsonData.id);
pm.environment.set("userName", jsonData.name);

// Save array item
pm.environment.set("firstUserId", jsonData.users[0].id);

// Save nested object data
pm.environment.set("addressId", jsonData.user.address.id);
```

Extract from Headers

```javascript
const authToken = pm.response.headers.get("Authorization");
pm.environment.set("token", authToken.replace("Bearer ", ""));
```

Dynamic Data Generation

Built-in Variables

```
{{$guid}}          - Generate GUID
{{$timestamp}}      - Current timestamp
{{$randomInt}}      - Random integer
{{$randomEmail}}    - Random email
{{$randomFirstName}} - Random first name
{{$randomLastName}}  - Random last name
{{$randomPhoneNumber}} - Random phone
{{$randomColor}}    - Random color
{{$randomUrl}}      - Random URL
```

Custom Dynamic Data

```javascript
// In pre-request script
const names = ["John", "Jane", "Bob", "Alice"];
const randomName = names[Math.floor(Math.random() * names.length)];
pm.environment.set("testName", randomName);

// Generate custom format
const customId = `USER_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
pm.environment.set("customId", customId);
```

Request Chaining

Sequential API Calls

javascript

```javascript
// Test 1: Create user
pm.test("User created successfully", function () {
    pm.response.to.have.status(201);
    const user = pm.response.json();
    pm.environment.set("newUserId", user.id);
});

// Test 2: Use the ID in next request
// URL: {{baseUrl}}/users/{{newUserId}}
```

Conditional Requests

javascript

```javascript
// Skip request if condition not met
if (!pm.environment.get("authToken")) {
    pm.execution.setNextRequest(null);
}

// Set next request dynamically
if (pm.response.code === 401) {
    pm.execution.setNextRequest("Get Auth Token");
} else {
    pm.execution.setNextRequest("Next API Call");
}
```

Error Handling

Robust Error Checking

```javascript
pm.test("Request completed successfully", function () {
    pm.response.to.not.be.error;
    pm.response.to.not.have.jsonBody('error');
});

pm.test("Error response format", function () {
    if (pm.response.code >= 400) {
        const errorData = pm.response.json();
        pm.expect(errorData).to.have.property('error');
        pm.expect(errorData).to.have.property('message');
    }
});
```

Try-Catch for Scripts

```javascript
try {
    const jsonData = pm.response.json();
    pm.environment.set("data", jsonData.result);
} catch (error) {
    console.log("Failed to parse JSON:", error);
    pm.environment.set("data", "");
}
```

Collection Runner & Monitoring

Data-Driven Testing

```javascript
// Use CSV/JSON file with test data
// CSV format: name,email,age
// Access in request: {{name}}, {{email}}, {{age}}

pm.test("User data is valid", function () {
    const name = pm.iterationData.get("name");
    const email = pm.iterationData.get("email");
    pm.expect(name).to.be.a('string');
    pm.expect(email).to.include('@');
});
```

Performance Testing

```javascript
pm.test("API performance check", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);

    // Log performance metrics
    console.log(`Response time: ${pm.response.responseTime}ms`);
    console.log(`Response size: ${pm.response.size()} bytes`);
});
```

Best Practices

Organization

- Use descriptive request names
- Group related requests in folders
- Use consistent naming conventions
- Add request descriptions and examples

Variables Management

- Use environment variables for URLs and tokens
- Keep sensitive data in environment variables
- Use collection variables for test data
- Clear temporary variables after use

Testing Strategy

- Test happy path and error scenarios
- Validate response structure and data types
- Check for security vulnerabilities
- Test edge cases and boundary conditions

Documentation

- Add request descriptions
- Include example responses
- Document required headers and parameters
- Maintain up-to-date environment templates

Common Patterns

CRUD Operations Testing

```javascript
// CREATE
pm.test("POST /users creates user", function () {
    pm.response.to.have.status(201);
    const user = pm.response.json();
    pm.environment.set("testUserId", user.id);
});

// READ
pm.test("GET /users/{id} returns user", function () {
    pm.response.to.have.status(200);
    const user = pm.response.json();
    pm.expect(user.id).to.equal(pm.environment.get("testUserId"));
});

// UPDATE
pm.test("PUT /users/{id} updates user", function () {
    pm.response.to.have.status(200);
});

// DELETE
pm.test("DELETE /users/{id} removes user", function () {
    pm.response.to.have.status(204);
});
```

Authentication Flow

javascript

```javascript
// 1. Login request
pm.test("Login successful", function () {
  pm.response.to.have.status(200);
  const response = pm.response.json();
  pm.environment.set("accessToken", response.access_token);
  pm.environment.set("refreshToken", response.refresh_token);
});

// 2. Authenticated request
// Header: Authorization: Bearer {{accessToken}}

// 3. Token refresh
pm.test("Token refreshed", function () {
  if (pm.response.code === 401) {
    // Trigger refresh token request
    pm.execution.setNextRequest("Refresh Token");
  }
});
```

Troubleshooting Tips

Debug Console Usage

javascript

```javascript
// Log variables
console.log("Current token:", pm.environment.get("token"));

// Log request details
console.log("Request URL:", pm.request.url);
console.log("Request method:", pm.request.method);

// Log response details
console.log("Response status:", pm.response.status);
console.log("Response body:", pm.response.text());
```

Common Issues

- **401 Unauthorized**: Check token expiration and format
- **CORS errors**: Use Postman agent or disable web security
- **SSL errors**: Disable SSL verification in settings
- **Timeout**: Increase timeout in collection settings
- **Variable not found**: Check variable scope and spelling

This cheat sheet covers the essential Postman features for comprehensive API testing. Keep it handy for quick reference during your testing workflows!