Mastering API Testing with Postman 2025

- **Pdf Content Priview:**
- What is Postman?
- Creating Requests: GET, POST, PUT, DELETE
- Adding Tests with pm.test()
- Using Variables & Environments
- Data-Driven Testing with CSV/JSON
- Automating with Newman + Jenkins
- Real Test Cases (Login, Auth, CRUD)
- 15+ Interview Questions (With Answers)
- Best Practices
- Bonus: Postman vs Other API Tools

1. What is Postman?

Ans - **Postman** is a powerful collaboration platform for API development, testing, and automation. It simplifies the process of developing APIs by providing:

A **user-friendly interface** to send requests and view responses
Support for multiple request types: GET, POST, PUT, DELETE, etc.
Automation features like test scripts, environment variables, and collection runners
Integrations with CI/CD tools (Newman, Jenkins, GitHub, etc.)

Where it's used:

API testing (manual and automation)

Backend validation

Regression suite for services

Mocking and simulating APIs

Data-driven testing

Example Use Case:

If your QA team needs to test a login API that accepts a username and password and returns a token, Postman can:

Send the request

Extract the token from the response

Save it for the next API (e.g., /user/profile)

2. Creating Requests GET, POST, PUT, DELETE in Postman

Ans - **A. GET Request – Fetch data from the server**

Use case: Get a list of users from an API.

Steps:

Open Postman → Click on **"New" → Request**

Set method to **GET**

Enter URL: https://reqres.in/api/users?page=2

Click **Send**

Response will show JSON with user data

{

"page": 2,

"data": [

{

"id": 7,

"first_name": "Michael",

...

}

]

}

B. POST Request – Submit data to the server

Use case: User signup or login.

Steps:

Method: **POST**

URL: https://reqres.in/api/users

Go to **Body → raw → JSON**

Json file

```
{
"name": "Archana",
"job": "QA Engineer“
}
```

Click **Send**

✔ You’ll get a 201 response with an ID and timestamp.

C. PUT Request – Update existing resource

Use case: Update user profile.

Steps:

Method: **PUT**

URL: https://reqres.in/api/users/2

Body → JSON:

Json file

```
{
"name": "Archana",
"job": "Senior QA"
}
```

Send → You’ll receive an updated timestamp in the response.

D. DELETE Request – Remove a resource

Use case: Delete a user record.

Steps:

Method: **DELETE**

URL: https://reqres.in/api/users/2

Click **Send**

✔ You'll get status 204 No Content — means deleted.

Tips:

Always check **status codes** (200, 201, 204, 400, 401, 500) to verify results.

Use **Headers** (e.g., Content-Type: application/json) when working with POST/PUT.

1. **Adding Tests with pm.test() in Postman**

Ans - Postman allows you to write JavaScript-based tests using the pm (Postman) object after a request is sent. This helps validate API responses automatically.

Why Use Tests in Postman?

Automatically check if status codes, response times, or data fields are correct

Create assertions to ensure your APIs behave as expected

Used in automation pipelines with Newman/Jenkins

Basic Syntax of a Test

Javascript file -

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

This test will pass only if the response status is 200.

Common Test Examples

1. Status Code Test

Javascript file -

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

2. Response Body Test

Javascript file -

```
pm.test("Response has user name", function ()
{
var jsonData = pm.response.json();
pm.expect(jsonData.name).to.eql("Archana");
});
```

3. Check Response Time

Javascript file -

```
pm.test("Response time is less than 1000ms", function ()
 {
pm.expect(pm.response.responseTime).to.be.below(1000);
});
```

4. Content-Type Header

Javascript file -

```
pm.test("Content-Type is JSON", function ()
 {
pm.response.to.have.header("Content-Type", "application/json; charset=utf-8");
 });
```

5. Store Token from Response

Javascript file -

var jsonData = pm.response.json();

pm.environment.set("authToken", jsonData.token);

This stores the token from the login response into an **environment variable** for future requests.

Where to Add Tests in Postman:

Click on a request

Go to the **"Tests"** tab

Paste your test script

Click **Send**

See test results in the **"Test Results"** tab

Pro Tip: Use tests in **Collection Runner** for running multiple requests together and verifying them.

Using Variables & Environments in Postman

Ans - Postman supports **variables** and **environments** to help you write flexible, reusable, and maintainable tests across different setups (dev, QA, staging, prod).

What are Variables?

Variables let you avoid hardcoding values like base URLs, tokens, IDs, etc.

Example:

Instead of writing:

arduino

https://api.staging.example.com/users

You can write:

bash

{{base_url}}/users

Types of Variables

Type	Scope	Example Usage
Global	Available everywhere	{{global_token}}
Environment	Specific to environment	{{env_token}}
Collection	Specific to collection	{{collection_token}}
Local	Within a single request	let myVar = "value"
Data (Runner)	From CSV/JSON	{{data.username}}

Setting Up Environments

Click the **Environment dropdown** (top right)

Click **Manage Environments**

Add a new environment:

base_url = https://api.dev.example.com

authToken = <your token>

Click **Set as active**

Now you can use {{base_url}} and {{authToken}} in your requests.

Using Variables in Requests

In URL:

Bash file -

{{base_url}}/users/2

In Headers:

Css file -

Authorization: Bearer {{authToken}}

In Body:

Json file -

{ "username": "{{user_name}}", "password": "{{password}}" }

Setting Variables from Test Script

Javascript file -

var jsonData = pm.response.json();

pm.environment.set("userId", jsonData.id);

Stores the ID from response into environment for later use.

Tips:

Use environments to switch between **DEV, QA, STAGE, PROD**

Use pm.variables.get("var_name") to access any variable in tests

Keep tokens and secrets out of public/global environments

Data-Driven Testing with CSV/JSON in Postman

Ans - Data-driven testing means running the same request multiple times with different inputs. Postman supports this using the Collection Runner and input files in CSV or JSON format.

Why Use Data-Driven Testing?

Test login API with 10 different usernames/passwords

Send bulk registration or update requests

Validate APIs with positive and negative data

Tools Required:

Postman Collection (with test scripts)

A CSV or JSON file with test data

Collection Runner

 Sample CSV File (loginData.csv)

Csv file -

username,password

archana01,pass123

qa_user,test456

admin_user,admin789

 Sample JSON File (loginData.json)

Json file -

[

{ "username": "archana01", "password": "pass123" },

{ "username": "qa_user", "password": "test456" },

{ "username": "admin_user", "password": "admin789" }

]

 How to Use Data in Postman Request

Example API Body (POST /login):

1. In request body, use variable format:

Json file -

{ "username": "{{username}}",

"password": "{{password}}" }

2. Save the request to a **collection**

▶ **Run in Collection Runner**

Click the **Runner icon**

Choose the **Collection**

Click **Select File** → Upload loginData.csv or loginData.json

Click **Run**

Postman will run the same request **for each row of data**.

Add Validation Test (in Tests tab)

Javascript file -

```
pm.test("Login is successful", function ()

{

pm.response.to.have.status(200);

var res = pm.response.json();

pm.expect(res.token).to.not.be.null;

});
```

This ensures every iteration gives a successful login.

Pro Tips:

Use **CSV** when testing simple values

Use **JSON** for nested or complex data

Combine this with **Newman** to automate in pipelines

2. **6. Automating Postman Tests with Newman + Jenkins**

Ans - Postman is great for manual API testing — but for automation, we use:

Newman: Command-line collection runner for Postman

Jenkins: CI/CD tool to run tests automatically (e.g., after each code push)

Why Automate?

sRun tests on every build/deploy

Generate reports

Catch regressions early

Integrate with tools like GitHub, GitLab, Bitbucket

A. Installing Newman (Locally)

You need Node.js installed first.

Steps:

Bash

npm install -g newman

▶ B. Run a Postman Collection with Newman

bash

newman run MyCollection.json

Add environment file:

bash

newman run MyCollection.json -e MyEnvironment.json

Add data file (for data-driven testing):

bash

newman run MyCollection.json -d loginData.csv

C. Export Postman Files

From Postman:

Go to **Collections**

Click ••• **> Export**

Export as Collection v2.1

Do the same for **Environment**

D. Generate HTML Reports with Newman

bash

newman run MyCollection.json -r cli,html

This creates a beautiful report in newman/ folder.

E. Integrate with Jenkins (CI/CD)

Step-by-Step:

1. Install Jenkins
2. Install Node.js & Newman on the Jenkins server
3. Create a **new Jenkins job**
4. In "Build" section → Add this shell command:

bash
newman run /path/to/collection.json -e /path/to/environment.json -r cli,html

5. (Optional) Configure:
    - **Email reports**
    - **Publish HTML Report plugin**
    - **Trigger builds** on GitHub commits

JJenkins + Newman Benefits:

- Daily smoke/regression runs
- Scheduled health checks for APIs
- Runs on multiple environments (QA, Stage, UAT)

Pro Tip:
You can also integrate with **GitHub Actions**, **GitLab CI**, or **Azure DevOps** using Newman the same way!

6. **Real Test Cases in Postman (Login, Auth, and CRUD APIs)**

    Ans -

Below are **step-by-step examples** of how you can test real-world API scenarios in Postman.

A. Login API Test (POST)

Endpoint: https://reqres.in/api/login

Request Type: POST
Body (raw → JSON):

Json file -
```
{
  "email": "eve.holt@reqres.in",
  "password": "cityslicka"
}
```

Test Script:

Javascript file –

```
pm.test("Login successful", () => {
  pm.response.to.have.status(200);
  var jsonData = pm.response.json();
  pm.expect(jsonData.token).to.not.be.undefined;
  pm.environment.set("authToken", jsonData.token);
});
```

 Stores the token in the environment for the next request.

B. Get User Profile (GET)

Endpoint: https://reqres.in/api/users/2
Headers:

Css file -
Authorization: Bearer {{authToken}}

Test Script:

Javascript file

```
pm.test("User found", () => {
  pm.response.to.have.status(200);
  var jsonData = pm.response.json();
  pm.expect(jsonData.data.id).to.eql(2);
});
```

C. Create a New User (POST)

Endpoint: https://reqres.in/api/users
Body:

Json file -

```
{
  "name": "Archana",
  "job": "QA Engineer"
}
```

Test:

Javascript file -

```
pm.test("User created", () => {
```

```
  pm.response.to.have.status(201);
});
```

✏ D. Update User Info (PUT)

Endpoint: https://reqres.in/api/users/2
Method: PUT
Body:

Json file
```
{
  "name": "Archana",
  "job": "Senior QA"
}
```

Test:

Javascript file -
```
pm.test("User updated", () => {
  pm.response.to.have.status(200);
});
```

E. Delete User (DELETE)

Endpoint: https://reqres.in/api/users/2
Method: DELETE

Test:

Javascript file -
```
pm.test("User deleted", () => {
  pm.response.to.have.status(204);
});
```

These 5 examples give you a **complete CRUD + Auth flow** in Postman:

1. Login and save token
2. Fetch profile using token
3. Create user
4. Update user
5. Delete user

8. 15+ Postman Interview Questions (with Answers)

These questions are commonly asked in QA and API automation interviews and will help you confidently explain your Postman knowledge.

? 1. What is Postman?

Answer:
Postman is a collaboration platform and API testing tool that allows users to send HTTP requests, validate responses, automate tests, and integrate with CI/CD pipelines.

? 2. What types of HTTP requests can Postman send?

Answer:
GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS

? 3. What are collections in Postman?

Answer:
A **collection** is a group of saved requests (like folders). You can organize APIs, add pre-request/test scripts, and run them together using the collection runner.

? 4. What is the purpose of environments in Postman?

Answer:
Environments store variables like {{base_url}}, {{authToken}}, etc., allowing you to switch between DEV, QA, STAGE, and PROD without editing each request.

? 5. How do you write test scripts in Postman?

Answer:
Using JavaScript in the **Tests** tab:

Javascript file -

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

❓ 6. How can you share a Postman collection with others?

Answer:
You can:

- Export the collection as a .json file
- Share a public link via Postman workspace
- Use GitHub or CI pipelines

❓ 7. What is Newman?

Answer:
Newman is Postman's command-line tool that lets you run Postman collections automatically from the terminal or CI/CD tools like Jenkins.

❓ 8. How do you run data-driven tests in Postman?

Answer:
Using the Collection Runner and uploading a CSV or JSON file with test data. Variables like {{username}} and {{password}} are replaced in each iteration.

❓ 9. Can Postman test response time and headers?

Answer:
Yes. Example:

Javascript file -

```
pm.test("Response time < 1000ms", () => {
  pm.expect(pm.response.responseTime).to.be.below(1000);
});

pm.test("Content-Type is JSON", () => {
  pm.response.to.have.header("Content-Type");
});
```

❓ 10. How can you capture and reuse a token?

Answer:

1. Save the login token using:

Javascript file -
pm.environment.set("authToken", pm.response.json().token);

2. Use it in the next request's **Authorization** header:

Bearer {{authToken}}

---

11. What is the difference between pm.environment.set and pm.variables.set?

Answer:

- pm.environment.set: Saves a variable to the selected environment
- pm.variables.set: Sets a temporary local variable for one request

---

12. Can Postman test GraphQL APIs?

Answer:
Yes. Postman supports **GraphQL** queries and mutations via the POST method.

---

13. What status codes do you check in API testing?

Answer:

- 200 (OK)
- 201 (Created)
- 204 (No Content)
- 400 (Bad Request)
- 401 (Unauthorized)
- 403 (Forbidden)
- 500 (Server Error)

---

14. What is pre-request script vs. test script?

Answer:

- **Pre-request Script**: Runs **before** the request (e.g., to set timestamp, auth headers)
- **Test Script**: Runs **after** the response (e.g., to check status code, body data)

❓ 15. How to run Postman collection automatically every day?

Answer:
Use **Newman + Jenkins** (or GitHub Actions) to schedule runs via cron jobs or on pull request triggers.

9. Best Practices in Postman for Efficient API Testing

Following best practices will improve your productivity, maintainability, and automation readiness in Postman.

✅ 1. Organize with Collections & Folders

- Group related APIs into collections (e.g., *User APIs*, *Payment APIs*)
- Use folders for modules (e.g., *Login*, *Signup*, *Admin*)
- Add descriptions for each request for clarity

✅ 2. Use Environments Effectively

- Create environments for DEV, QA, UAT, and PROD
- Store common variables:
  - base_url
  - authToken
  - userId
- Avoid hardcoding sensitive values in the body or headers

✅ 3. Write Modular Tests

Avoid copy-pasting test code — create reusable functions or snippets.

javascript
```
pm.test("Valid token", () => {
  const token = pm.response.json().token;
  pm.expect(token).to.not.be.undefined;
});
```

✅ 4. Use Pre-request Scripts for Dynamic Data

You can generate values before requests:

javascript
```
pm.environment.set("timestamp", new Date().toISOString());
```

Or generate random test data:

Javascript
pm.envir vcfgvonment.set("randomEmail", `user${Math.random()}@mail.com`);

---

5. Keep Tests Focused and Simple

- Validate:
  - Status code
  - Key JSON fields
  - Response time
- Avoid overly complex test logic inside Postman

---

6. Use Data Files for Bulk Testing

Use **CSV/JSON** files for testing:

- Login with multiple users
- Submit forms with varied inputs
- Automate negative test cases (e.g., wrong password, blank fields)

---

7. Automate with Newman & CI/CD

- Always export collections and environments
- Use Newman CLI for local test runs and in Jenkins or GitHub Actions
- Add HTML reports to check results visually

---

8. Use Comments & Naming Conventions

- Add comments in test scripts
- Name your requests and variables clearly (getUserById, updateProfile)
- Add changelogs or version notes in collection descriptions

---

9. Leverage Monitoring (Optional)

Postman provides monitoring (on Pro plans) to:

- Schedule runs every hour/day

- Monitor uptime
- Send alerts on failures

---

10. Keep Postman Updated

Postman adds frequent updates (like Visualizer, GraphQL support). Keeping it updated gives you access to new testing features and better security.

---

10. Bonus: Postman vs Other API Testing Tools

Postman is one of the most popular API tools—but there are others like **Rest Assured**, **SoapUI**, **Swagger**, and **Insomnia**. Here's how Postman compares:

Postman

Feature	Description
Type	GUI-based tool
Ideal For	Manual + automated API testing
Scripting	JavaScript (pm.* APIs)
Automation Support	Yes (via Newman + Jenkins)
Data-Driven Tests	Yes (CSV/JSON + Runner)
Learning Curve	Easy
Reporting	Built-in CLI + HTML (via Newman)

Best For: QA Engineers, API Explorers, Test Automation with minimal coding.

Rest Assured

Feature	Description
Type	Java Library

Feature	Description
Ideal For	Developers and coders
Scripting	Java
Automation Support	Yes (Jenkins, Maven)
Data-Driven Tests	With test frameworks like TestNG
Learning Curve	Medium to High
Reporting	Custom with ExtentReports or Allure

 Best For: Java automation testers doing API + UI testing in one framework.

SoapUI

Feature	Description
Type	Desktop app (GUI)
Ideal For	SOAP and REST testing
Scripting	Groovy
Automation Support	Yes (Pro version)
Data-Driven Tests	Yes (Excel, DB)
Learning Curve	Medium
Reporting	Rich reports in Pro

 Best For: Enterprise-level testing, especially SOAP APIs.

Swagger / Swagger UI / SwaggerHub

Feature	Description
Type	API documentation + mock server
Ideal For	API design and mocking
Scripting	Not a test tool directly
Automation Support	No direct test execution
Data-Driven Tests	Not supported
Learning Curve	Easy
Reporting	NA

Best For: API designers, developers documenting and mocking APIs.

Insomnia

Feature	Description
Type	GUI-based
Ideal For	Devs, quick testing
Scripting	Minimal
Automation Support	Limited
Data-Driven Tests	No
Reporting	No

Best For: Lightweight alternative to Postman for developers.

Feature	Description
Type	GUI-based
Ideal For	Devs, quick testing
Scripting	Minimal
Automation Support	Limited
Data-Driven Tests	No
Reporting	No

- **Summary Table:**

Tool	Manual	Automation	Scripting	Data Testing	Best Use Case
Postman	✅	✅ (Newman)	JS	✅	Manual + Semi-Auto Testing
Rest Assured	✅	✅	Java	✅ (TestNG)	Java-based Test Automation
SoapUI	✅	✅ (Pro)	Groovy	✅	SOAP & REST Enterprise APIs
Swagger	✅	✅	✅	✅	API Design & Mocking
Insomnia	✅	✅	✅	✅	Lightweight Manual Testing