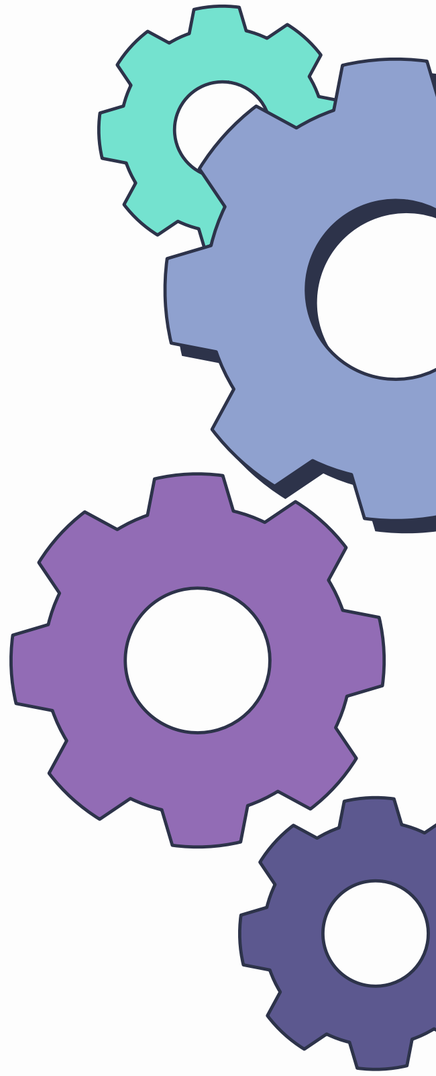
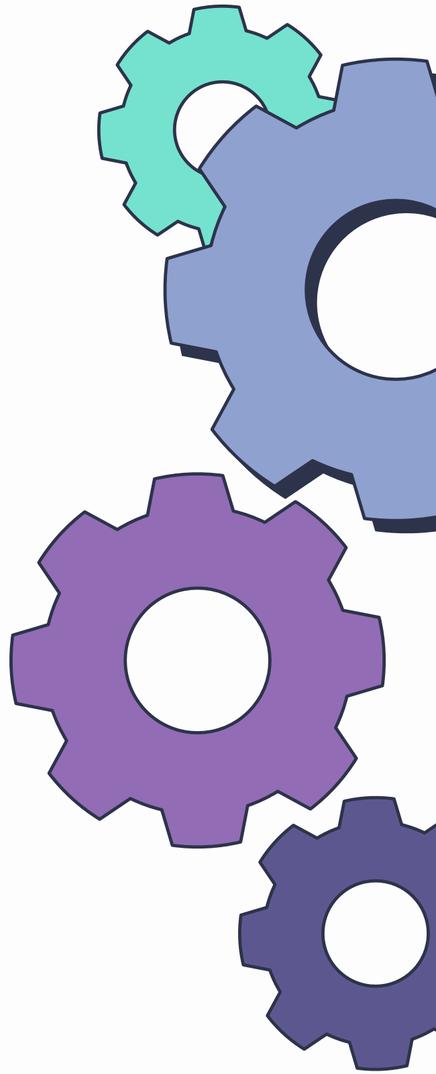
API mocking & testing tools



Group 1:
22125017 - Nguyễn Tiến Dũng
22125019 - Nguyễn Đức Duy
22125097 - Võ Như Thiện

1.API testing



What is API Testing?

API Testing is a software testing technique that focuses on verifying the functionality, reliability, performance, and security of application programming interfaces (APIs) by sending requests and checking responses.

- Ensures that APIs produce accurate outputs and behave as intended across various scenarios.
- Often examines HTTP actions such as GET, POST, PUT, and DELETE, emphasizing how data is retrieved, created, updated, and removed.

Some types of API Testing

1

Functional Testing

Validates that the API’s actions align with its defined requirements and documented specifications.

2

Load Testing

Emulates heavy usage to assess how well the API performs and scales when placed under intense load.

3

Security Testing

Confirms that the API’s authentication and authorization work correctly and identifies any weaknesses that could expose sensitive data.

4

Contract Testing

Ensures that the API adheres to its expected data structure, like JSON schema, preventing breaking changes.

5

Integration Testing

Ensures that multiple systems communicating through APIs work together properly as a unified whole.

Functional Testing - Example

Scenario: Login API

Request: POST /login

Payload

```
{
  "email": "user@example.com",
  "password": "Correct123"
}
```

Expected Functional Behavior:

- Status code: 200 OK
- Response contains: accessToken, refreshToken
- User role and profile returned correctly
- Invalid credentials → 401 Unauthorized

Load Testing – Example

Scenario: Get Products API under 1000 users

Request: GET/products

Test Setup:

- Simulate 1000 concurrent users
- Each user sends 5 requests/sec

Expected Load Behavior:

- 95% responses < 300ms
- Error rate < 1%
- API does not crash under heavy traffic

Security Testing – Example

Scenario: Accessing a protected endpoint without a token

Request: GET /orders

Security Checks:

- Request without token → 401 Unauthorized
- Invalid token → 403 Forbidden
- Token tampering → rejected
- Sensitive fields (password, credit card) NOT returned
- API uses HTTPS, not HTTP

Contract Testing – Example

Scenario: Create User API must follow JSON schema

Expected Response Schema:

```
{
  "id": "string",
  "name": "string",
  "email": "string",
  "createdAt": "string"
}
```

Contract Testing Example:

- id must always be a string, not number
- email must be present
- No extra unexpected fields
- Response matches OpenAPI/Swagger spec

Integration Testing – Example

Scenario: Order → Payment → Inventory

Workflow:

- POST /orders → creates order
- POST /payment → processes payment
- PATCH /inventory → reduces stock

Expected Integrated Behavior:

- Order created only if payment succeeds
- Payment rollback if inventory fails
- All systems update consistently
- Data flows correctly across services

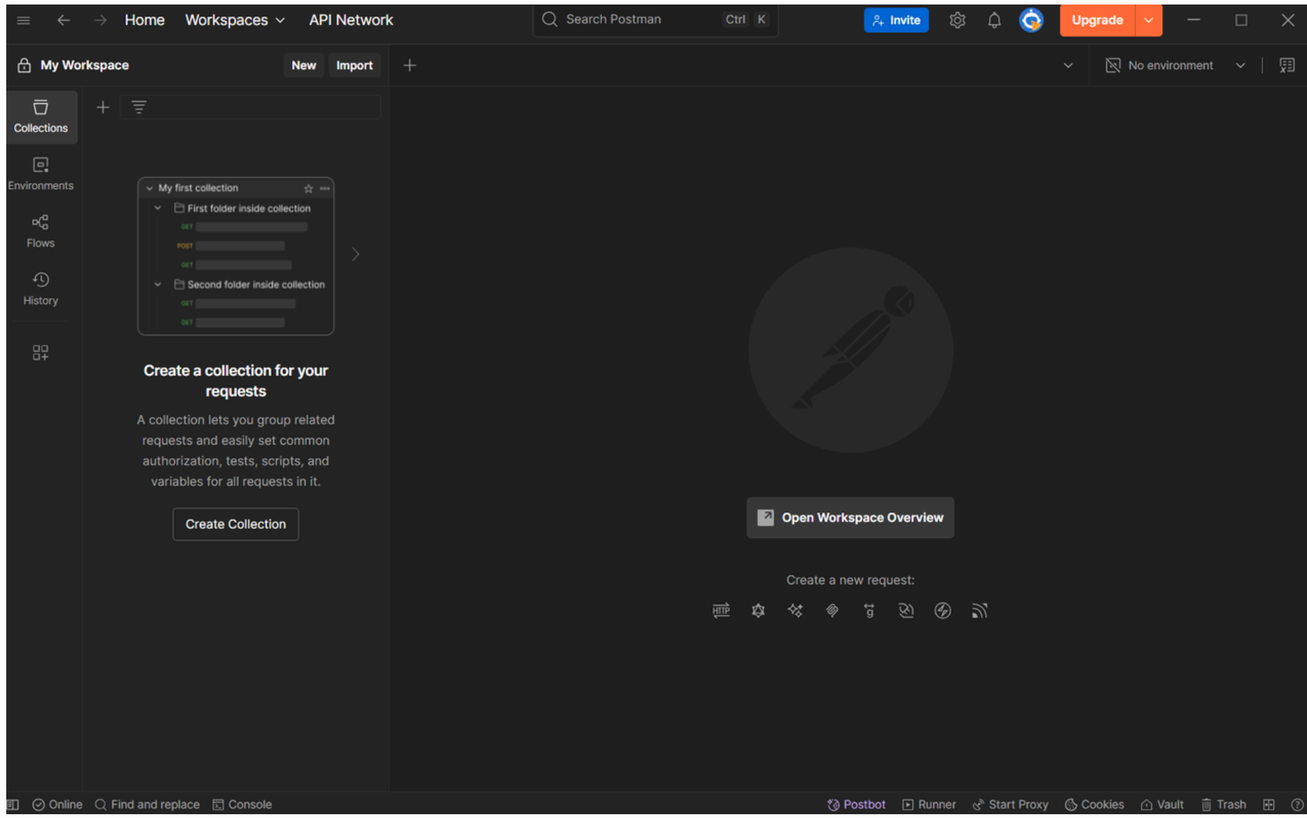
Why Postman?

User-Friendly Interface: A clear, easy-to-use GUI makes it simple to send requests and review responses.

Scripting Support: Allows using JavaScript for pre-request logic and writing test assertions.

Environments & Variables: Enables quick management of environments and variables, eliminating the need for code modifications.

CI/CD Integration: Allows test collections to run seamlessly at any stage of the build, test, or deployment pipeline.

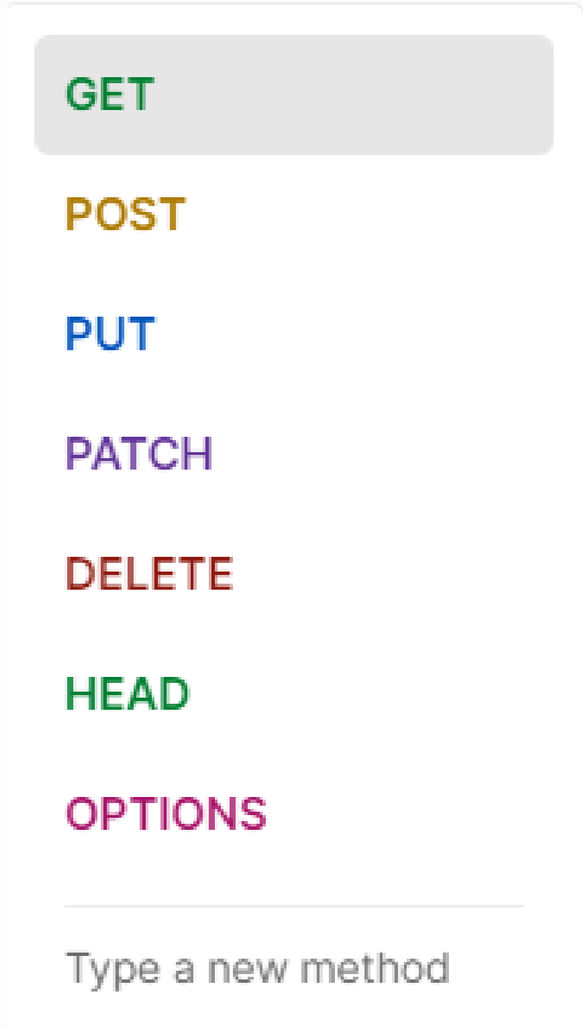


How to use Postman?

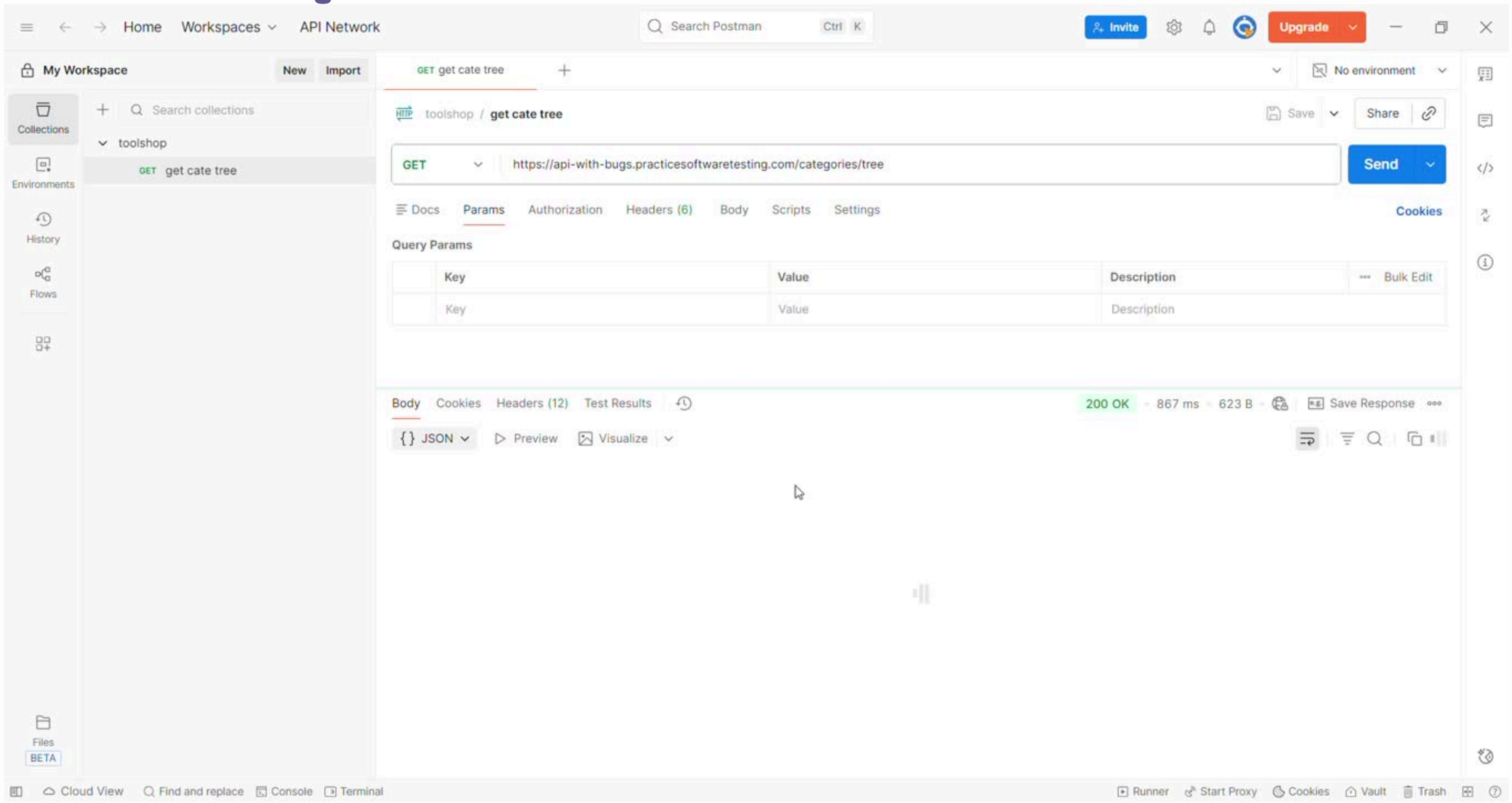
1. **Sending API Requests**
2. **Test Scripts**
3. **Collections**
4. **Environment Variables**

API requests

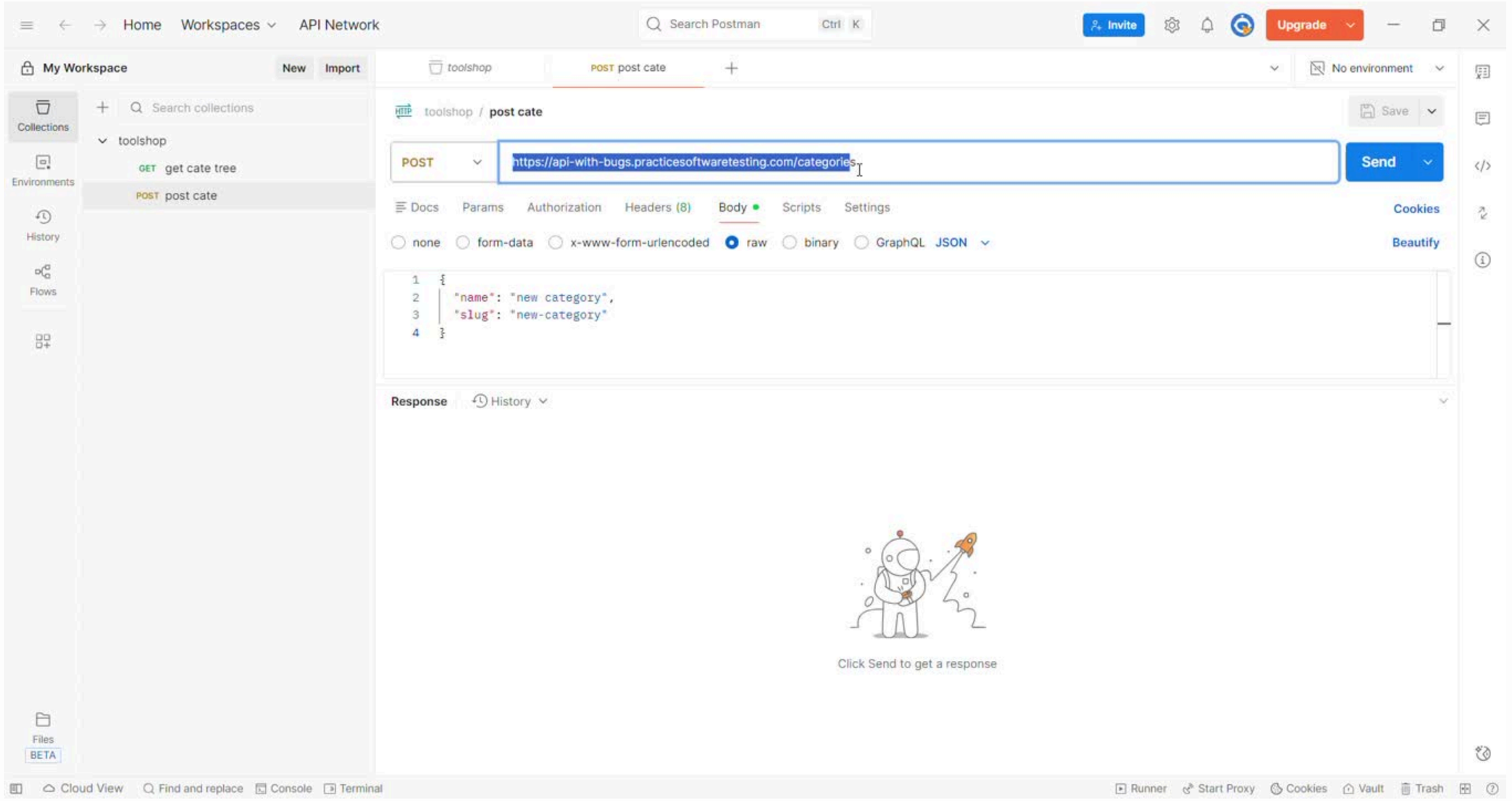
- GET – Retrieve data from the server
- POST – Send new data to the server
- DELETE – Remove data from the server
- PUT/PATCH – Update existing data on the server



API requests - GET



API requests - POST



Test scripts

Pre-request Script

Runs before the request is sent.

Used to:

- Generate dynamic data (timestamp, UUID, signatures)
- Set or update environment variables
- Prepare headers/body before sending the request

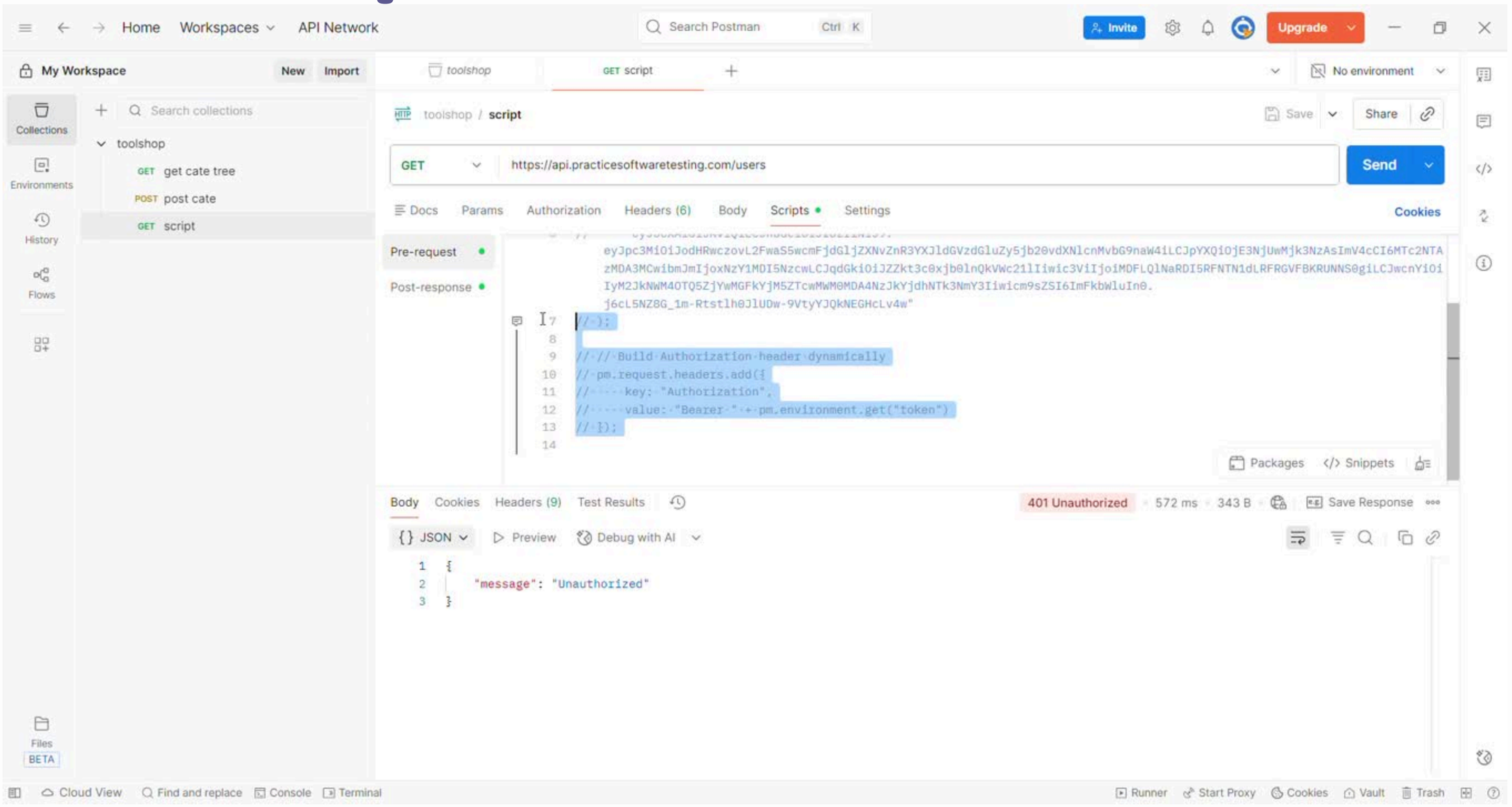
Post-request Script

Runs after the response is received.

Used to:

- Validate response status, body, and headers
- Extract values from the response
- Save data (e.g., token) for later requests

Test scripts



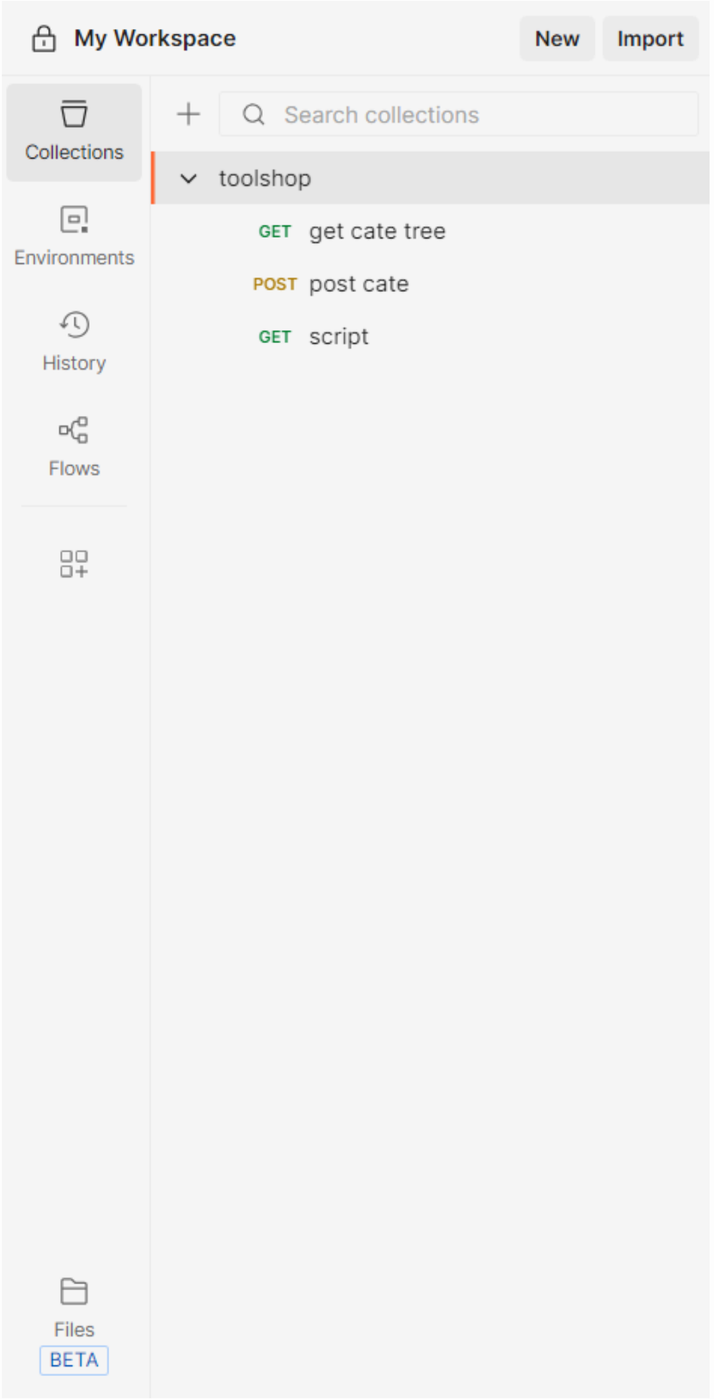
Collections

What is a Collection?

A Collection is a group of saved API requests organized in one place.

Why Use a Collection?

- Organize API endpoints by feature or module
- Reuse authentication, variables, and headers
- Add Pre-request Scripts and Tests
- Share APIs easily with team members
- Enable automation and documentation



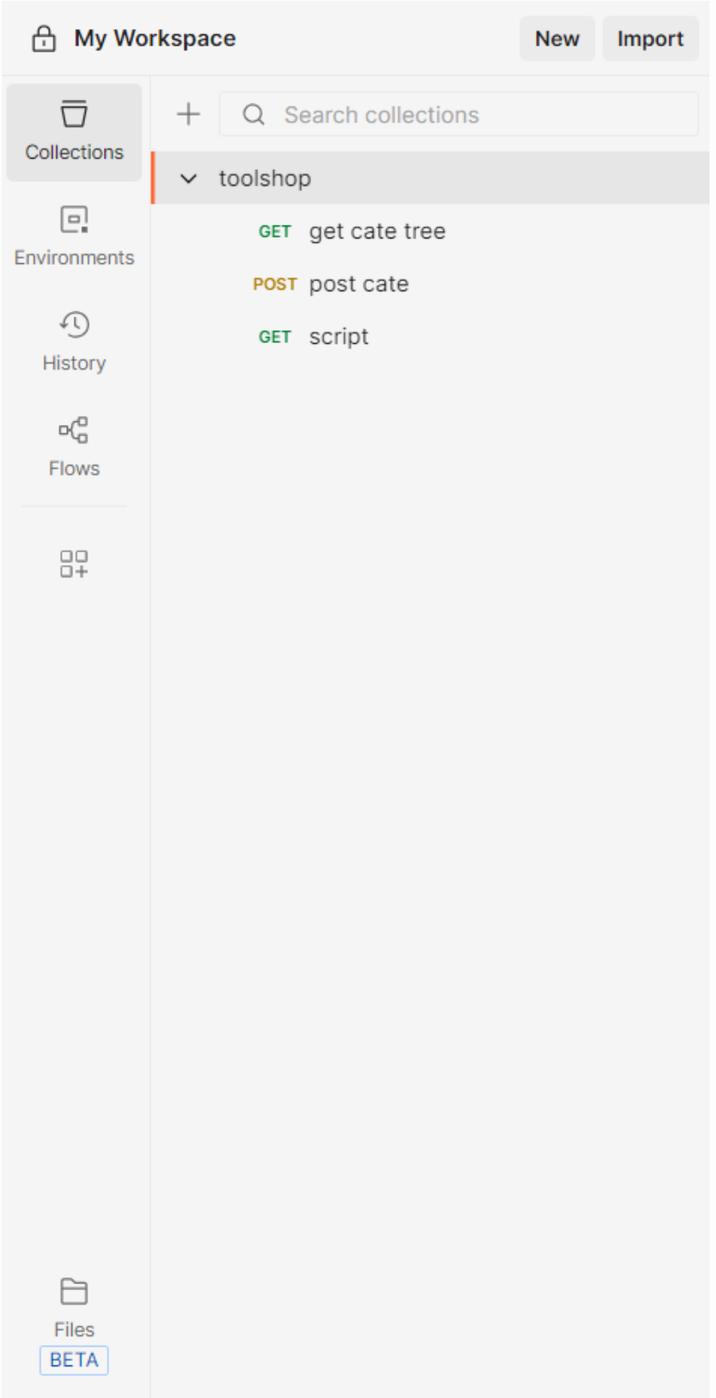
Collections

Key Features

- Supports folders and subfolders
- Stores multiple request types (GET/POST/PUT/DELETE)
- Allows collection-level variables
- Can be exported/imported
- Works with Collection Runner for test automation

Benefits

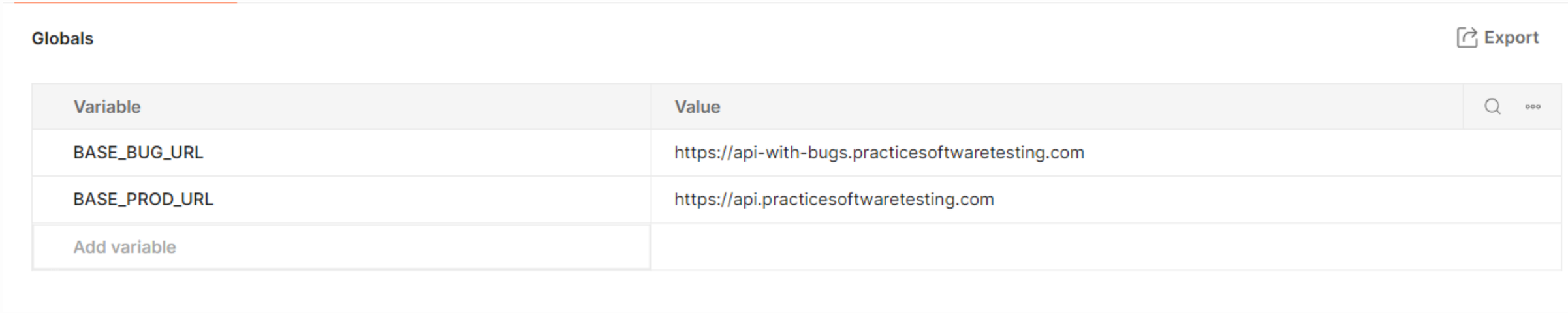
- Clear API structure
- Faster testing and development
- Consistent request setup
- Easy collaboration
- Ready for automated workflows



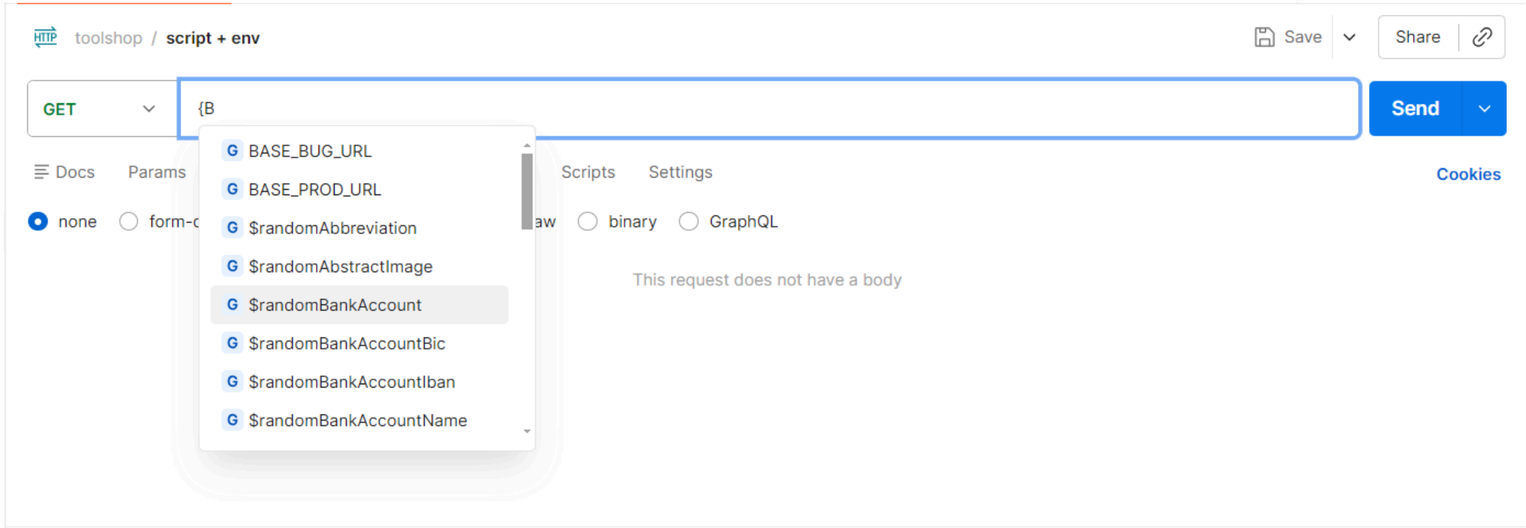
Environment Variables

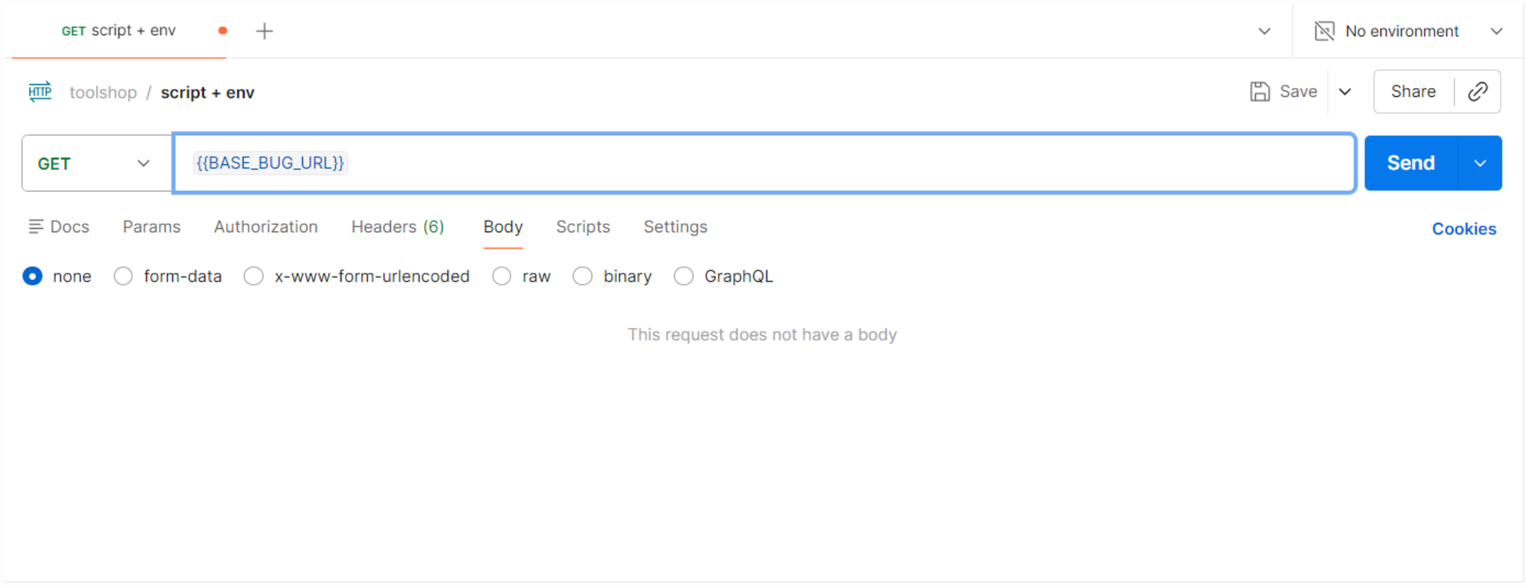
Environment Variables are key–value pairs used to store dynamic or reusable data in Postman.

- Avoid hardcoding values (URLs, tokens...)
- Switch between environments easily (Dev / Test / Prod)
- Reuse variables across multiple requests
- Keep sensitive data separate from requests
- Make collections portable and maintainable

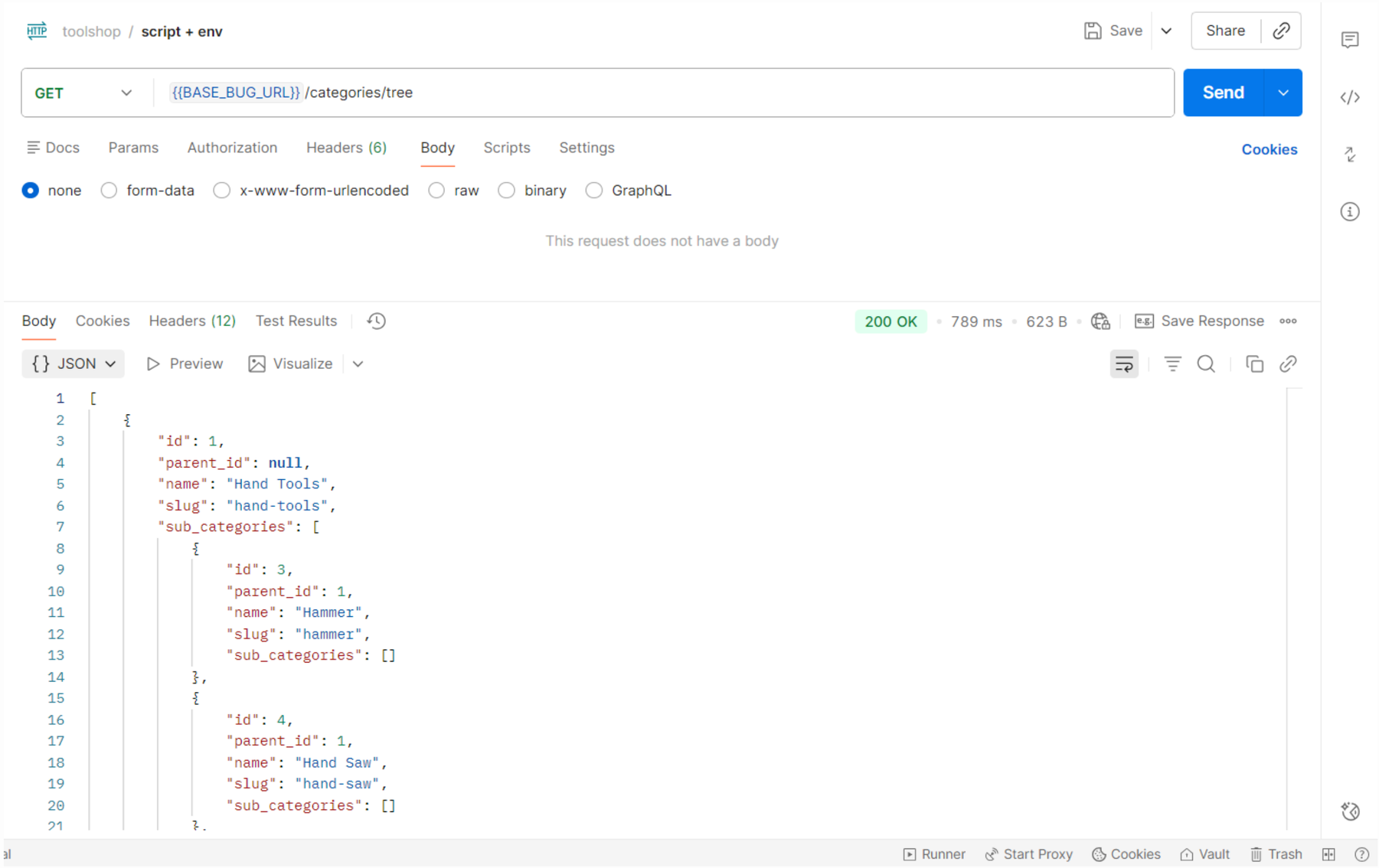


Environment Variables

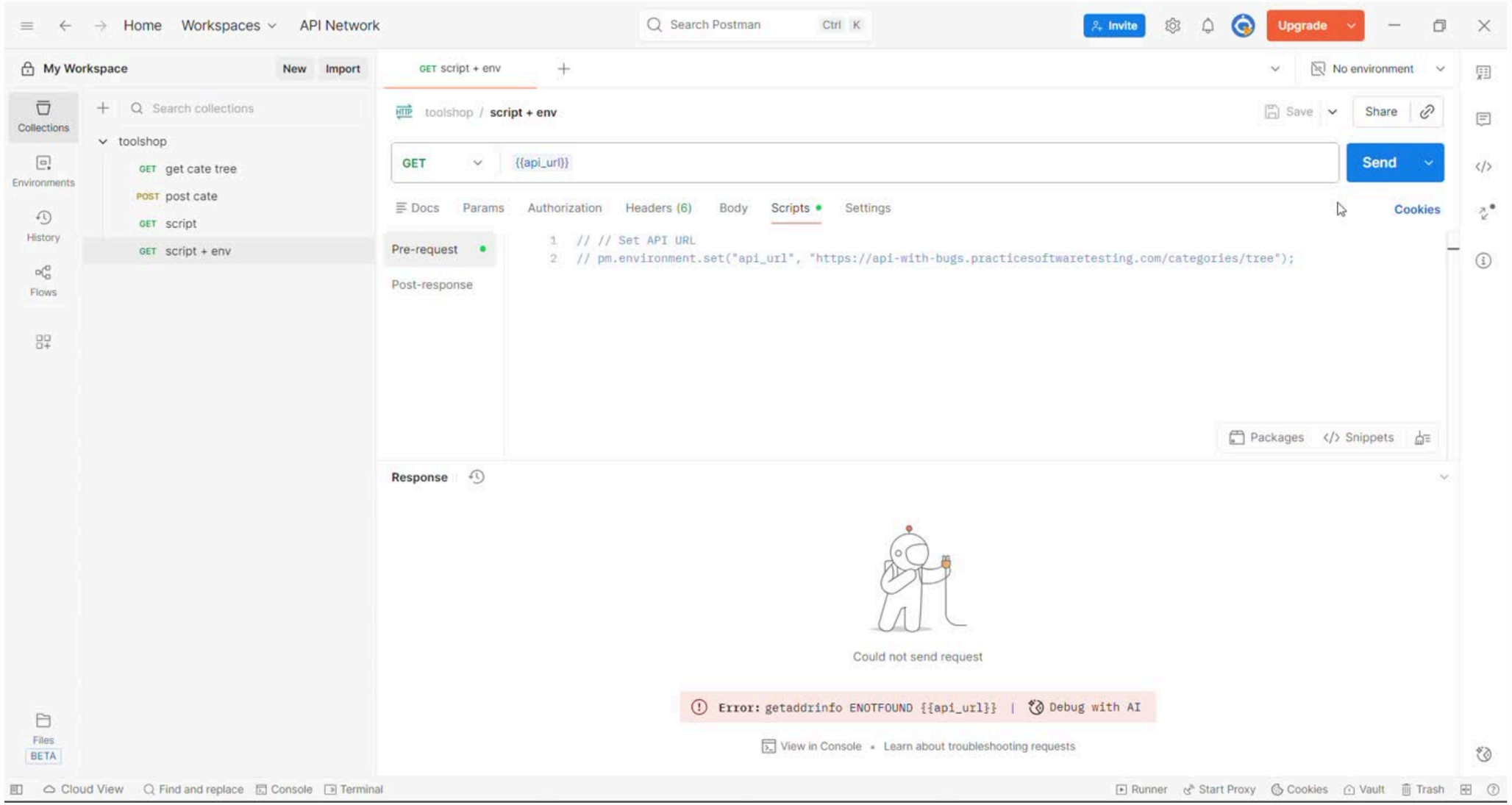




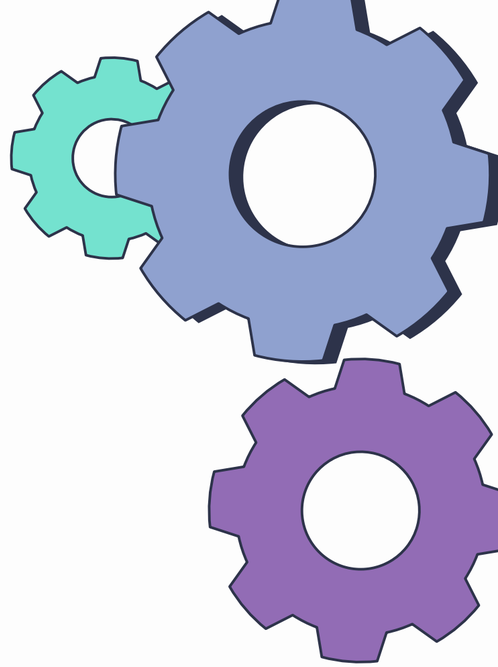
Environment Variables



Environment Variables



API Testing Automation



What is Automate API Testing:

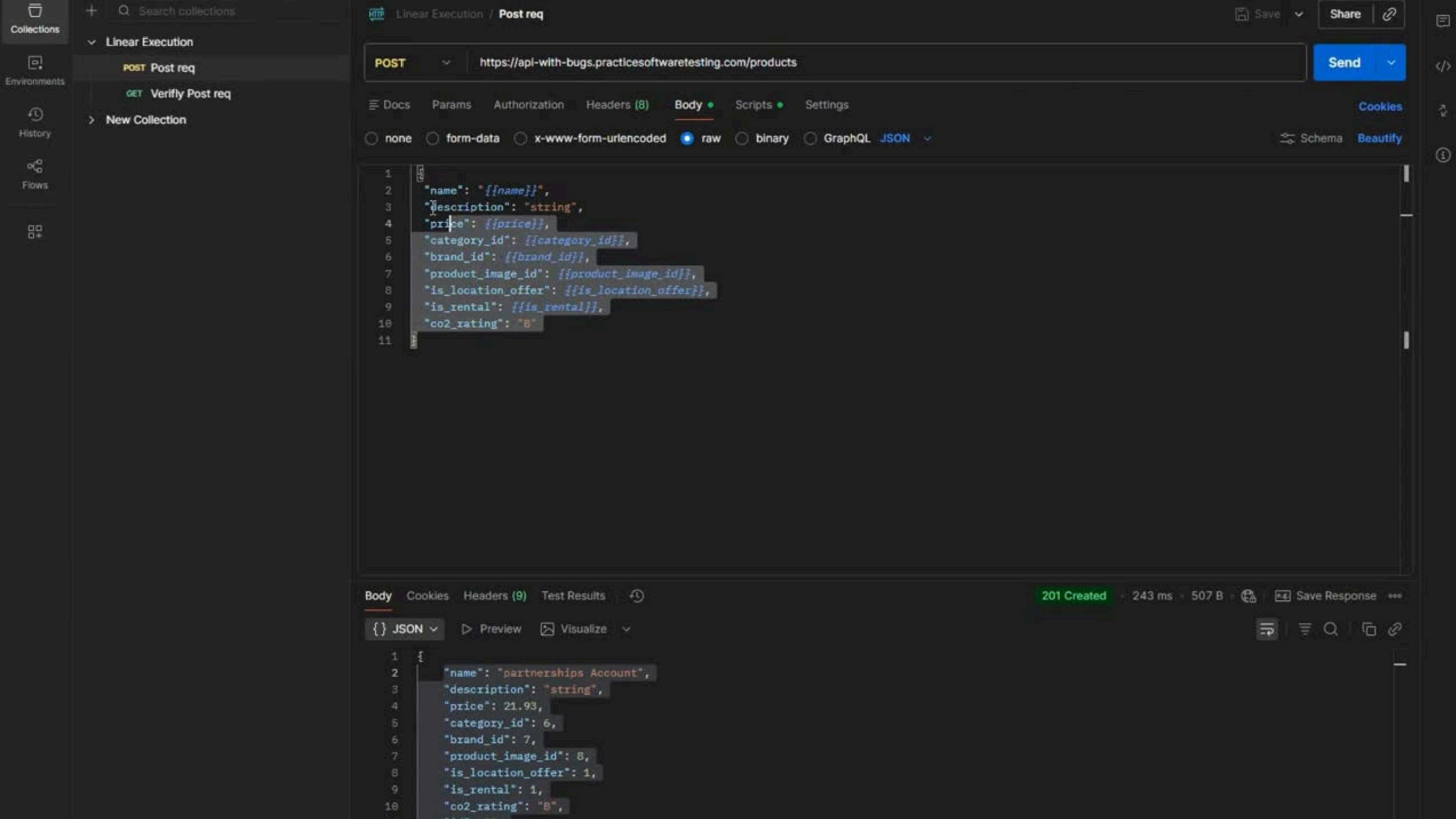
- Execute API tests at certain times or frequencies
- Execute API tests in CI/CD pipelines

Why Automate API Testing:

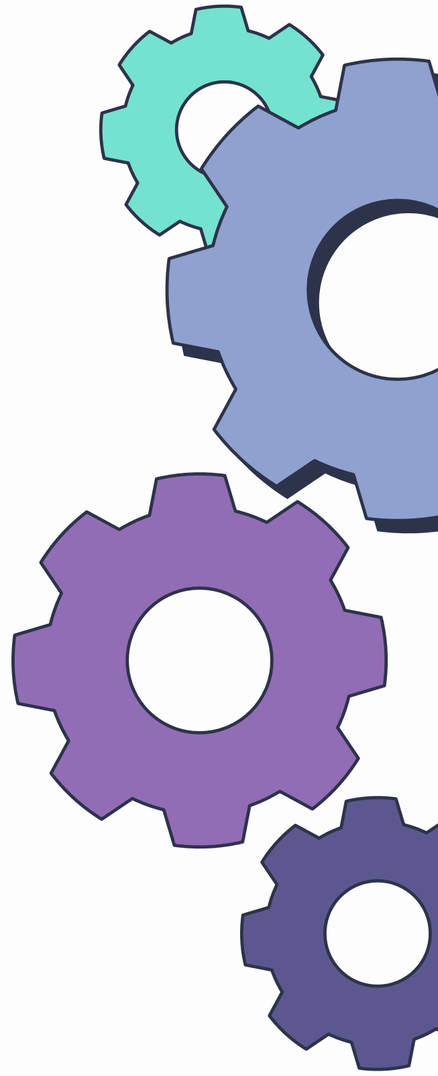
- Faster regression cycles
- Consistent & repeatable tests
- Early bug detection in development

Means of Test Automation

Test Automation	Collection runner	Postman CLI / Newman	Webhooks (custom, flows)	Scheduled runs / Monitors / live collections
Interface	Postman app	CLI / container / script	Postman servers	Postman servers
Use cases	Local development Debugging Functional Performance*	CI/CD On-demand	Event-based trigger On-demand Integrations	Ongoing Health / status

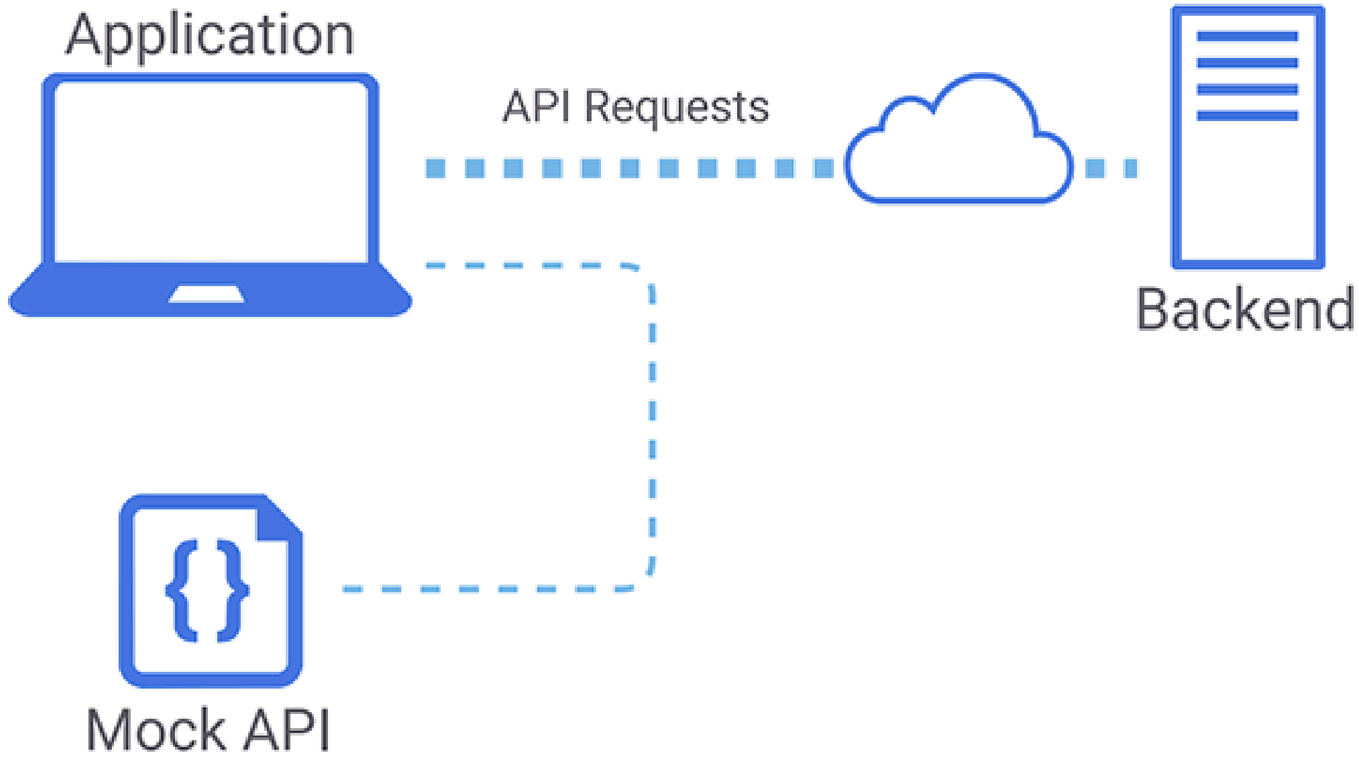

Collections
Environments
History
Flows
Search collections
Linear Execution
POST Post req
GET Verifly Post req
New Collection
Linear Execution / Post req
Save
Share
POST
https://api-with-bugs.practicesoftwaretesting.com/products
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"name": "{{name}}",
"description": "string",
"price": {{price}},
"category_id": {{category_id}},
"brand_id": {{brand_id}},
"product_image_id": {{product_image_id}},
"is_location_offer": {{is_location_offer}},
"is_rental": {{is_rental}},
"co2_rating": "B"
}
Body
Cookies
Headers (9)
Test Results
201 Created
243 ms
507 B
Save Response
{} JSON
Preview
Visualize
{
"name": "partnerships Account",
"description": "string",
"price": 21.93,
"category_id": 6,
"brand_id": 7,
"product_image_id": 8,
"is_location_offer": 1,
"is_rental": 1,
"co2_rating": "B",

2.API mocking



What is API mocking?

- API Mocking is simulating the behavior of a real API service.
- It uses fake responses (data, status codes, errors) instead of connecting to the actual backend.



Why We Need API Mocking

1.Enables Parallel Development

- Frontend team can build the UI/UX without waiting for the backend API to be complete.
- Reduces idle time and accelerates project delivery.

2.Allows Edge Case Testing:

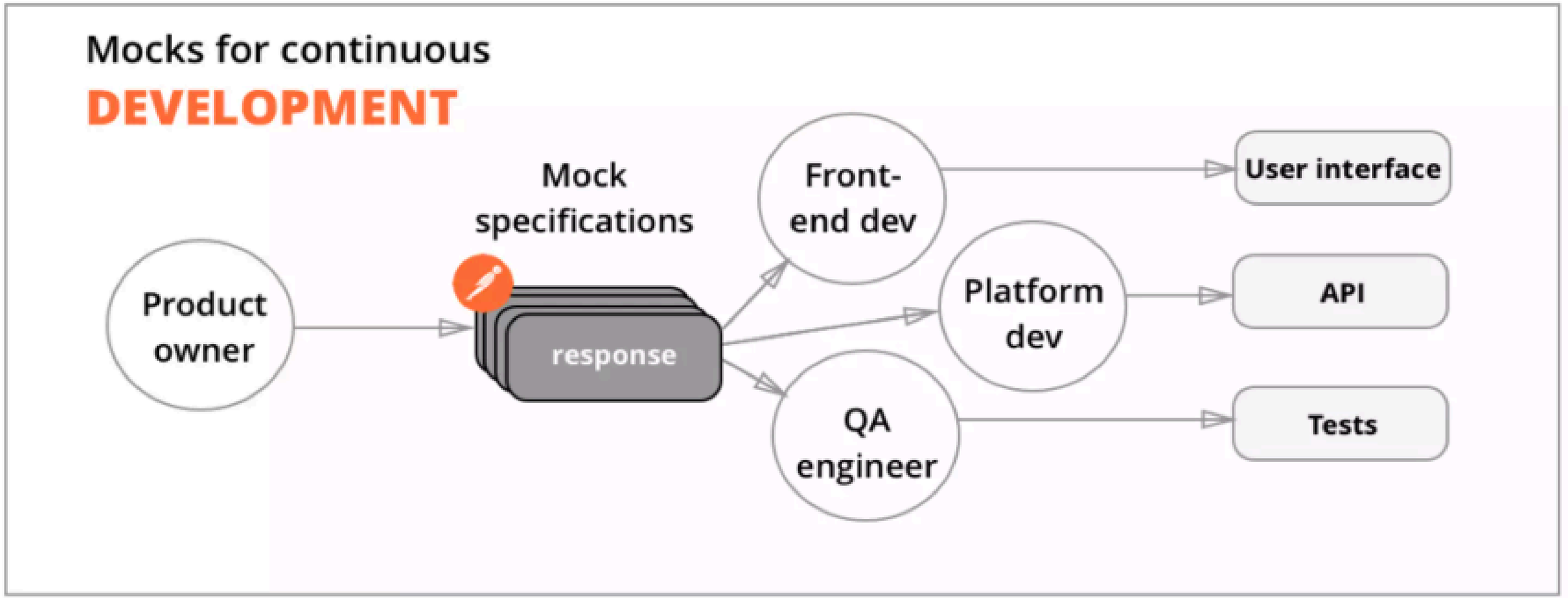
- Easily simulate rare error conditions or unusual/unexpected data responses.
- Ensures the application handles extreme scenarios gracefully.

3.Supports Stable Testing

- Provides consistent, controlled responses for reliable, repeatable tests.
- Testers aren't affected by real API downtime or sudden changes

4.Improves Early Bug Detection

- Catch integration issues sooner in the development cycle by simulating responses.
- Reduces the risk of discovering critical problems late.


Mocks for continuous
DEVELOPMENT
Mock
specifications
Product
owner
response
Front-
end dev
Platform
dev
QA
engineer
User interface
API
Tests

Types of API Mocking

1.Static Mocks: Always return the same fixed response for a specific request.Best for early-stage UI development when API details are minimal.

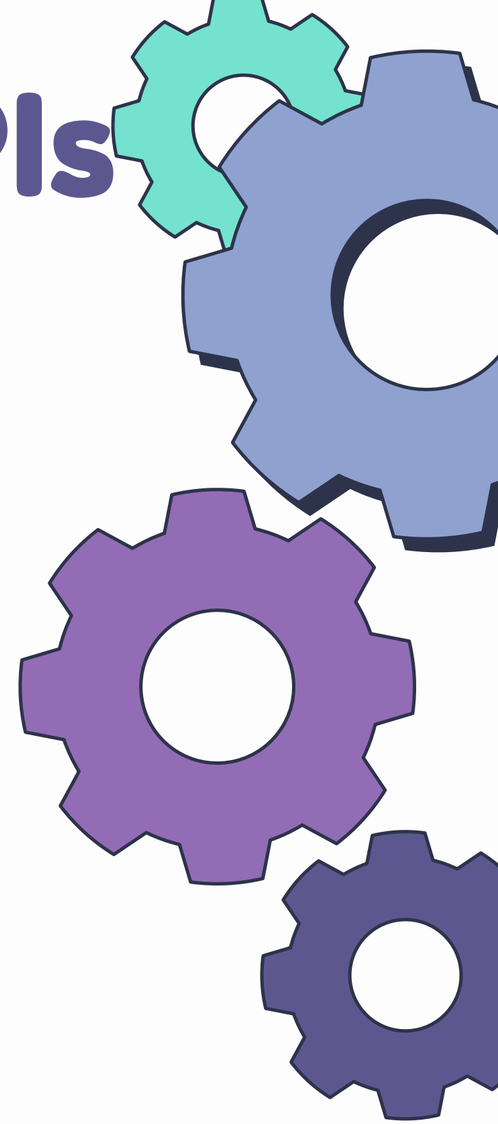
2.Dynamic Mocks: Responses change based on inputs (e.g., parameters or headers).Useful for simulating different workflows without changing code.

3. Contract-based Mocks: Generate responses based on API specifications (e.g., OpenAPI).Helps catch schema mismatches and ensures alignment between teams.

4.Behavior-driven Mocks:

- Simulate specific conditions like timeouts, throttling, or failures.
- Used to validate client-side error handling logic (retry, fallbacks).

Popular Tools for Mocking APIs

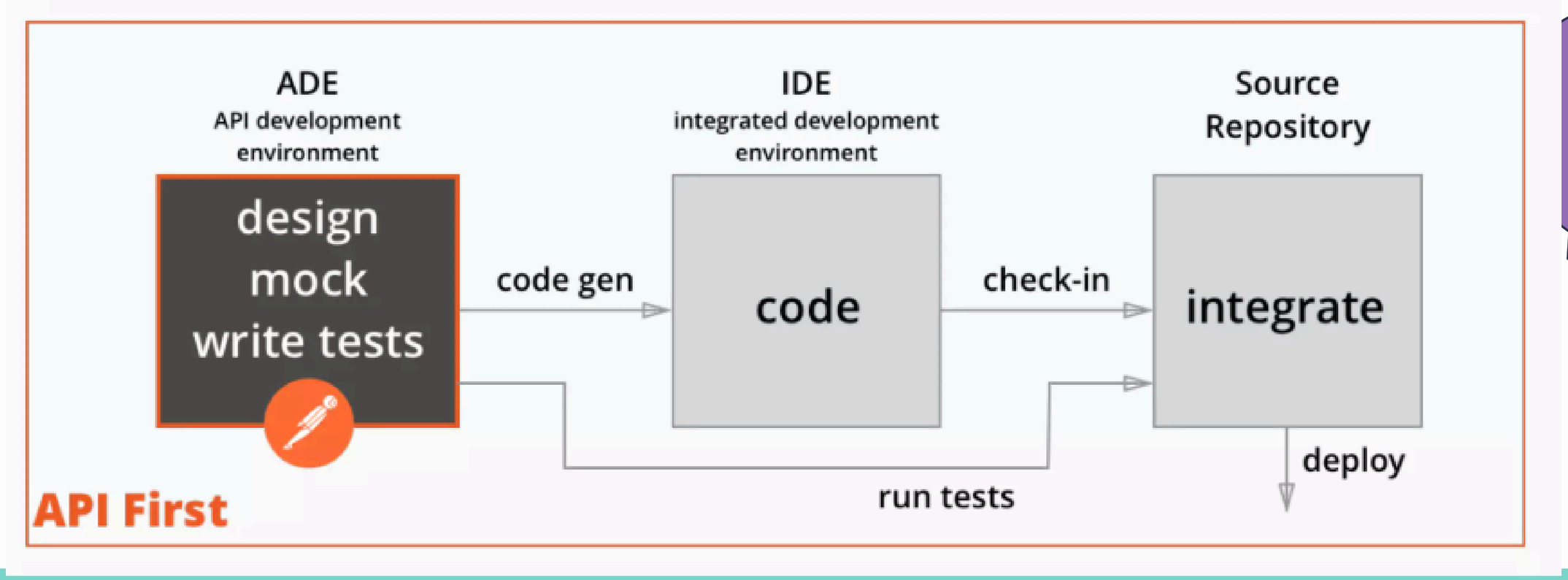


- **Postman** : a popular API development environment that supports creating mock servers
- **WireMock** : a flexible, open-source API mocking tool that runs as a standalone server or embedded in tests.
- **Mockoon** : a desktop application that allows you to create mock APIs quickly.
- **JSON Server** : an open-source tool that allows you to create a full fake REST API with a simple JSON file.

Postman mock server

- Simulated APIs that mimic real server responses.
- Easily customizable to deliver any response you need, all without coding.
- Conveniently hosted on the Postman platform.

How we use mocks



Home Workspaces API Network Search Postman Ctrl K Invite Upgrade

My Workspace New Import

Create mock server

No environment

Collections
Environments
Flows
Mock servers
History

Search mock servers

You don't have any mock servers.

Mock servers let you simulate endpoints and their corresponding responses in a collection without actually setting up a back end.

Create mock server

Create a mock server

Select a collection to mock

An existing collection Create a new collection

Mock server name

demo api mocking

Add requests

Enter the requests you want to mock. Optionally, add a request body by clicking on the (...) icon.

Request Method	Request URL	Response Code	Response Body
GET	{{url}}/ Path	200	Response Body

Environment

An environment is a group of variables useful for storing and reusing values.

No Environment

Simulate a fixed network delay

No delay

Save the mock server URL as an new environment variable

This will create a new environment containing URL.

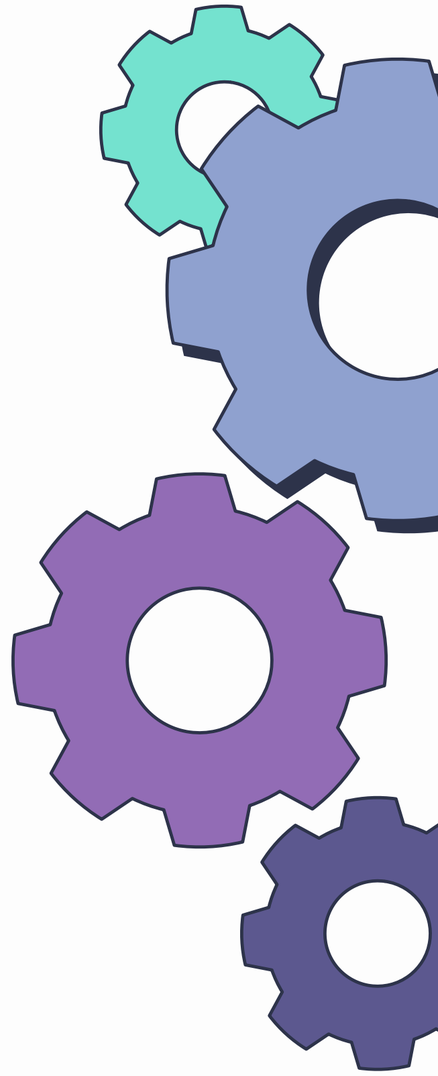
Make mock server private

To call a private mock server, you'll need to add an `x-api-key` header to your requests. See how to generate a Postman API key ↗

Create Mock Server Cancel

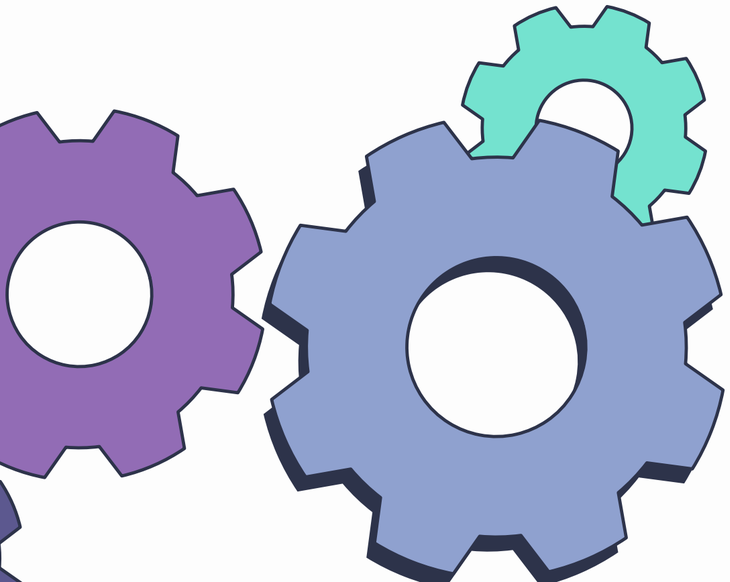
Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

3.AI-Approach for API Testing



Why AI for API Testing?

- Reduces manual effort in test case design
- Accelerates testing timelines
- Improves test coverage and accuracy
- Helps detect issues earlier in development
- Enhances security and anomaly detection



For short:

AI drastically reduces:

- Time
- Cost
- Human error

AI can assist in four major areas:

1.AI-Generated Test Cases

AI can automatically generate:

- Positive & negative test cases
- Boundary cases
- ...

2.AI-Generated API Mocking

AI examines API structures and generates:

- Mock servers
- Fake responses
- Fault injection scenarios (500s, timeouts, bad data)

3.AI-Enhanced Automation Scripts

- AI can write or fix your automation test scripts
- Suggest schema validation

4.AI-Driven API Anomaly Detection

Machine learning models monitor APIs can detect anomalies such as:

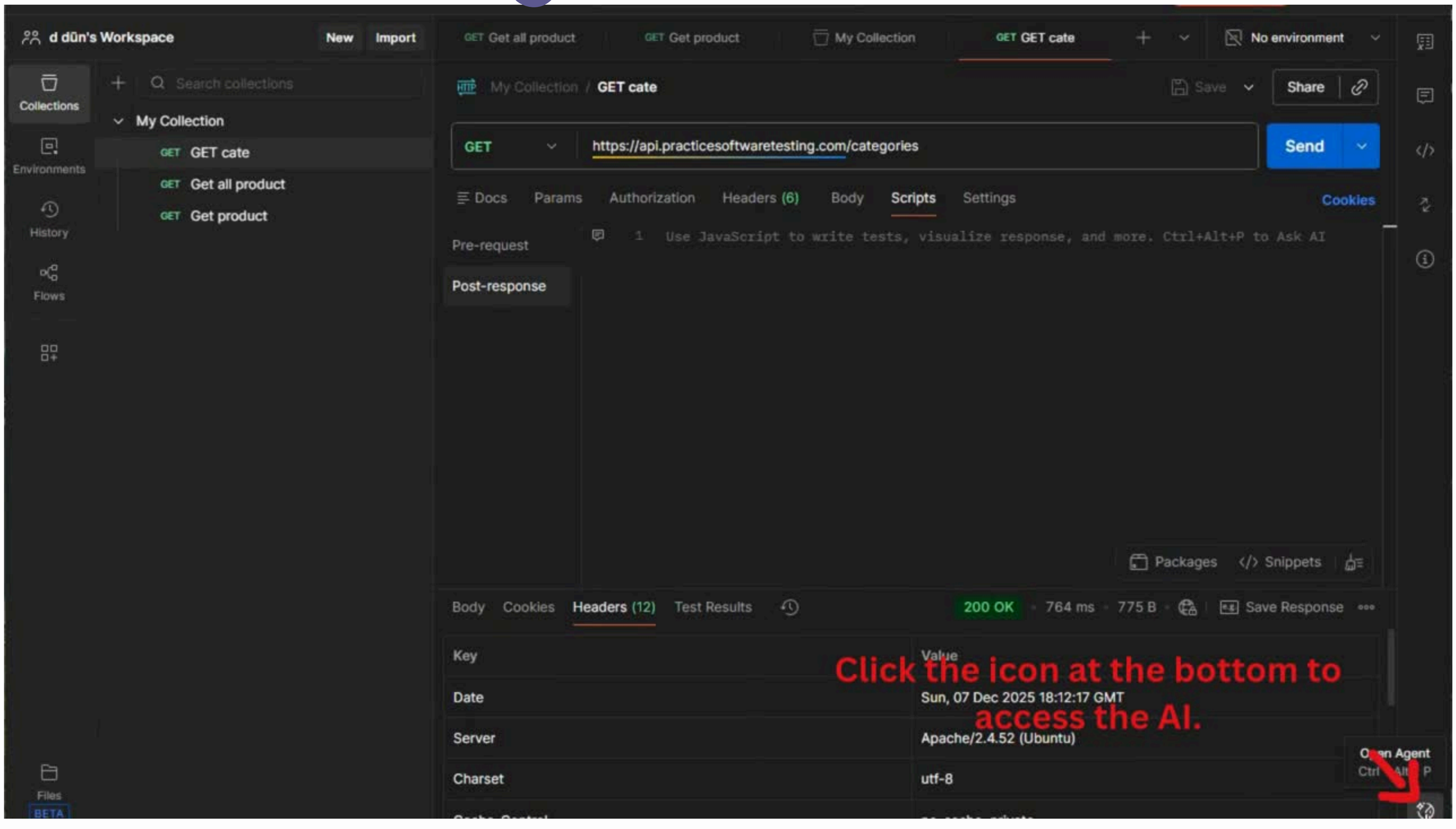
- Unusual traffic
- Response delays
- Error spikes
- Suspicious payloads or unauthorized access
-

Postbot - Postman AI assistant

It can help:

- Write tests
- Visualize responses
- Write FQL in Postman Flows
- Write documentation

Postbot test generate demo

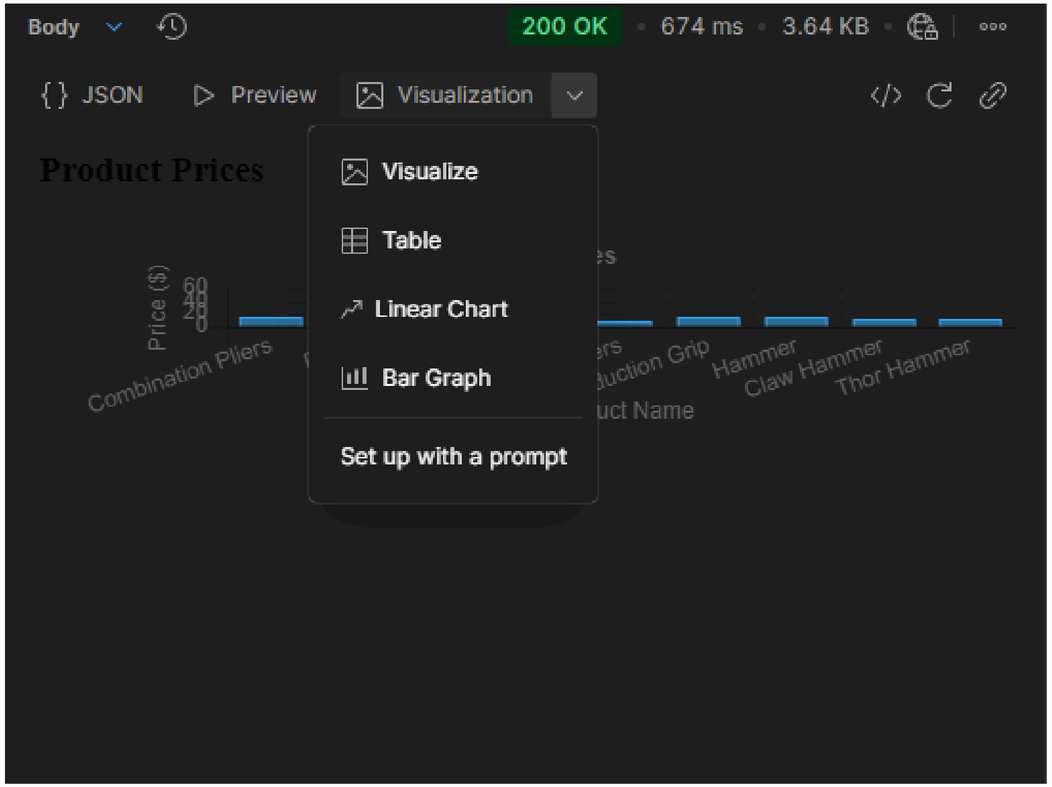


Postbot Visualize responses

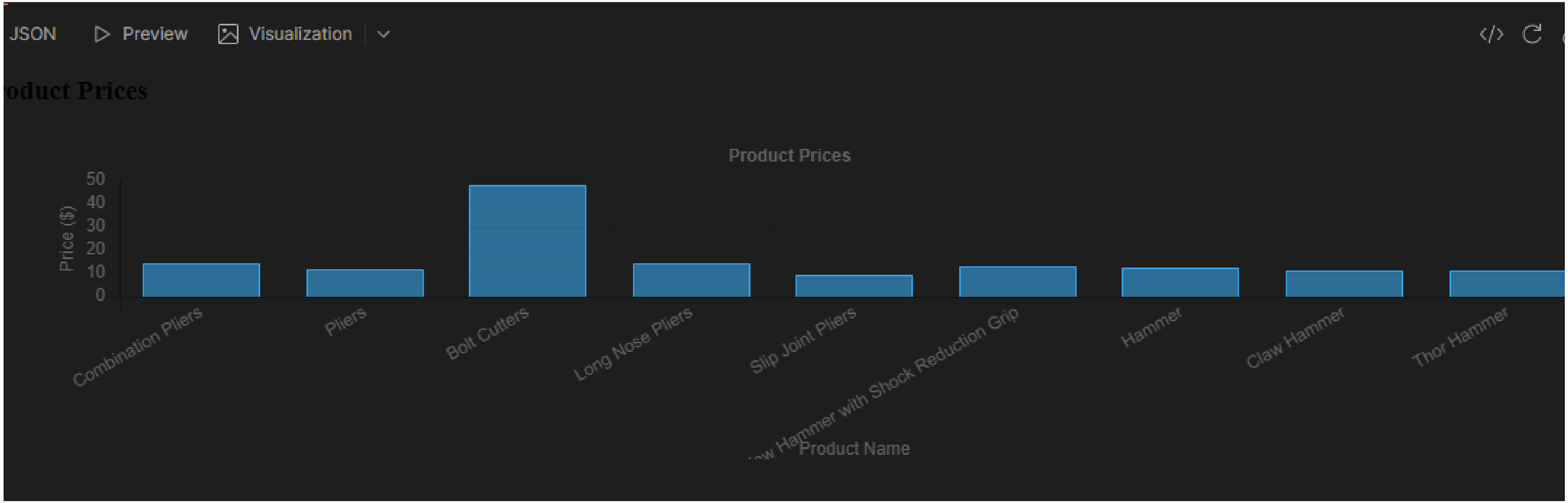
You can using prompt like:

- Visualize response as bar graph with prices are shown on the y-axis.

Alternatively, you can choose from several types of visualizations, and the AI will generate the response based on your schema



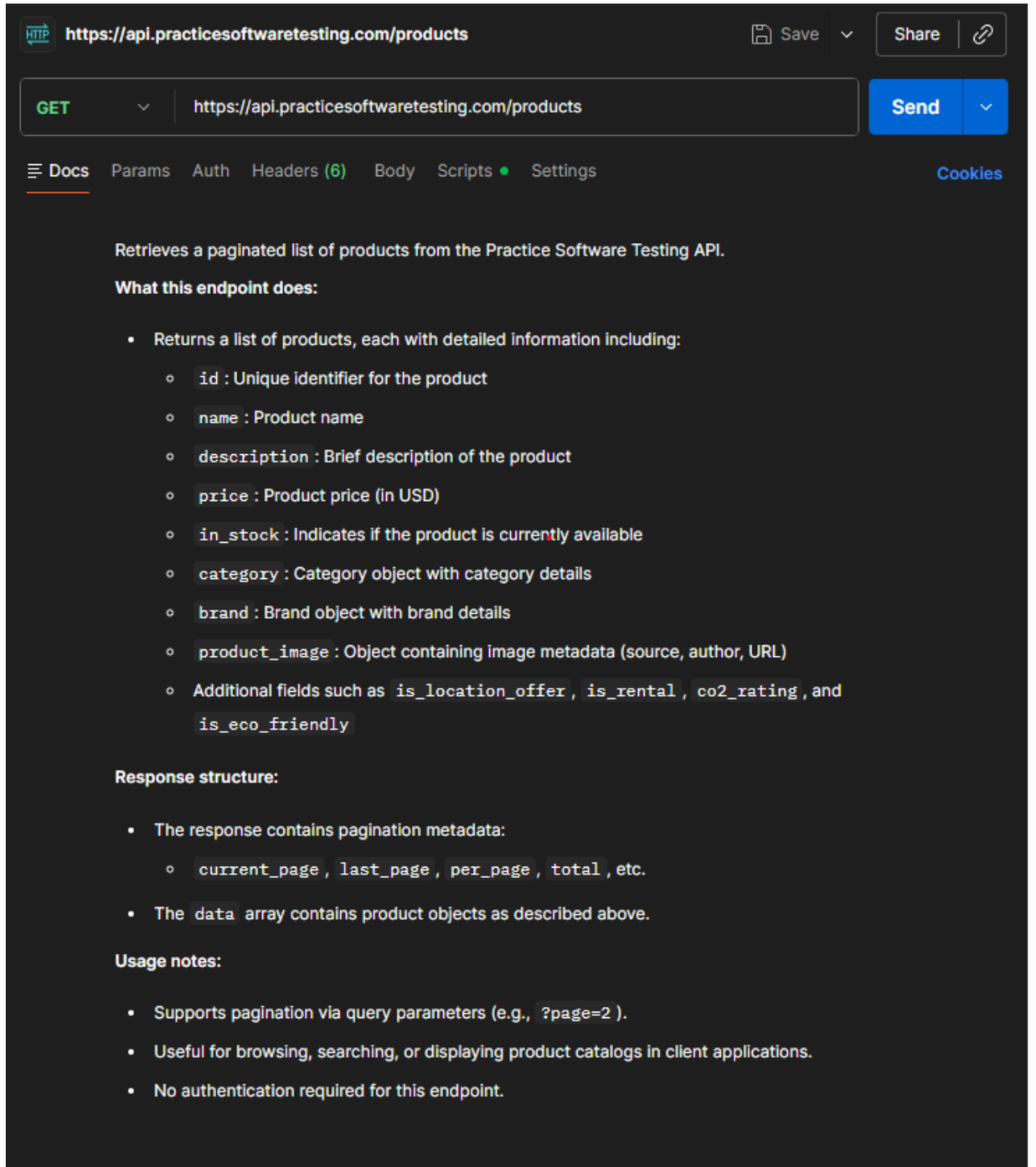
Visualization result



Postbot Write documentation

Simply click a button

Alternatively, using natural language prompt for more detail docs



Limitations of AI

AI still needs human supervision

- Remember our previous demo?
- Actually, there is nothing wrong with API
- The AI is one that make mistake

AI may misinterpret poorly written docs

- Why the AI failed on our demo?
- My poorly prompt
- No API docs was given

No awareness of system environment

- Why the AI failed on our demo?
- Can the collection variable be set?
- What if the variable is bounded?

There are many other limitations

- Limited understanding of business rules
- Not reliable for security validation
- Difficulty handling edge cases
- etc,

Conclusion

AI can be helpful, but it should never be trusted blindly.

Thanks