

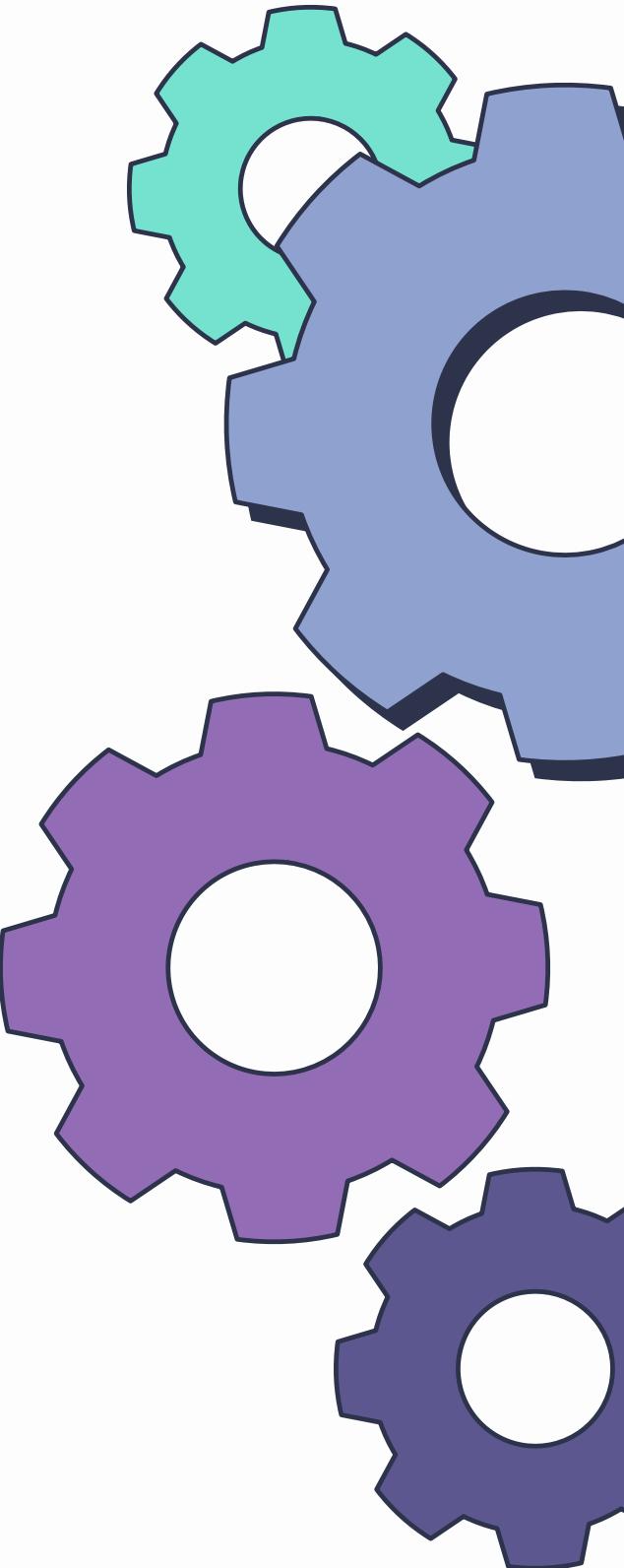
API mocking & testing tools

Group 1:

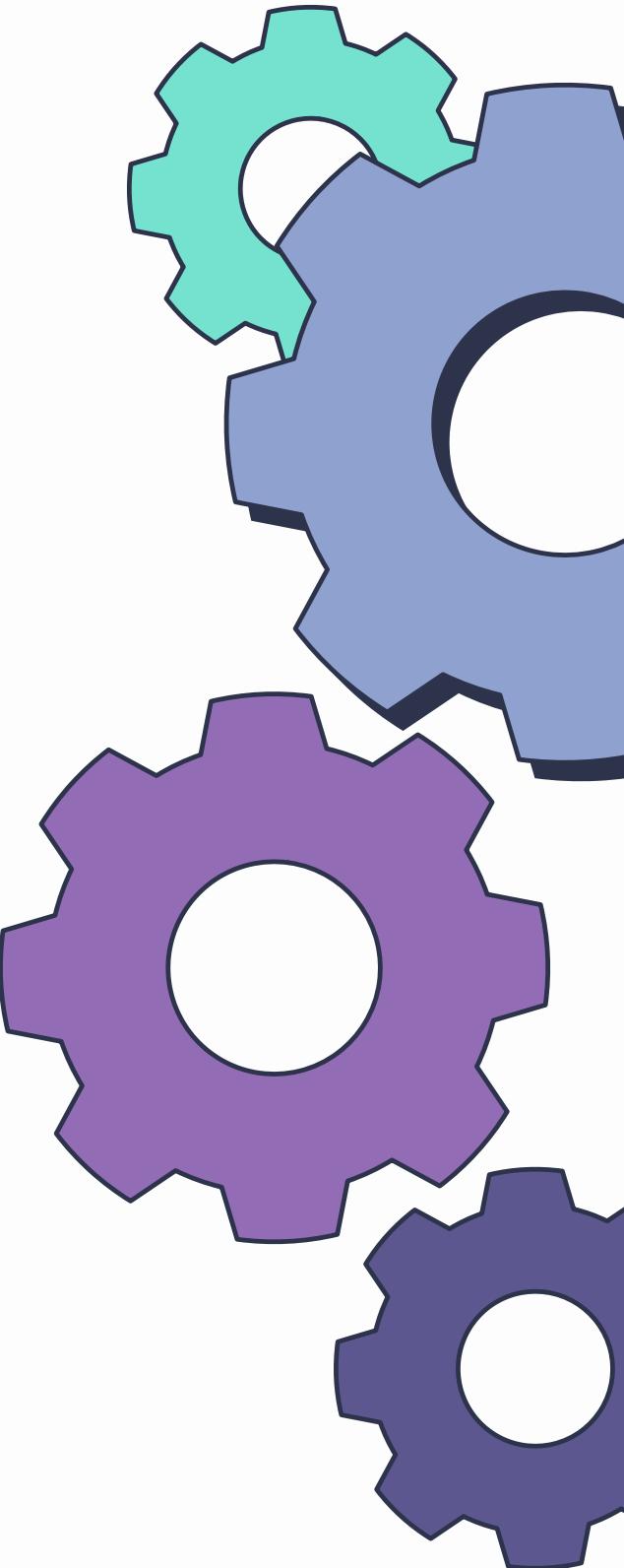
22125017 - Nguyễn Tiến Dũng

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

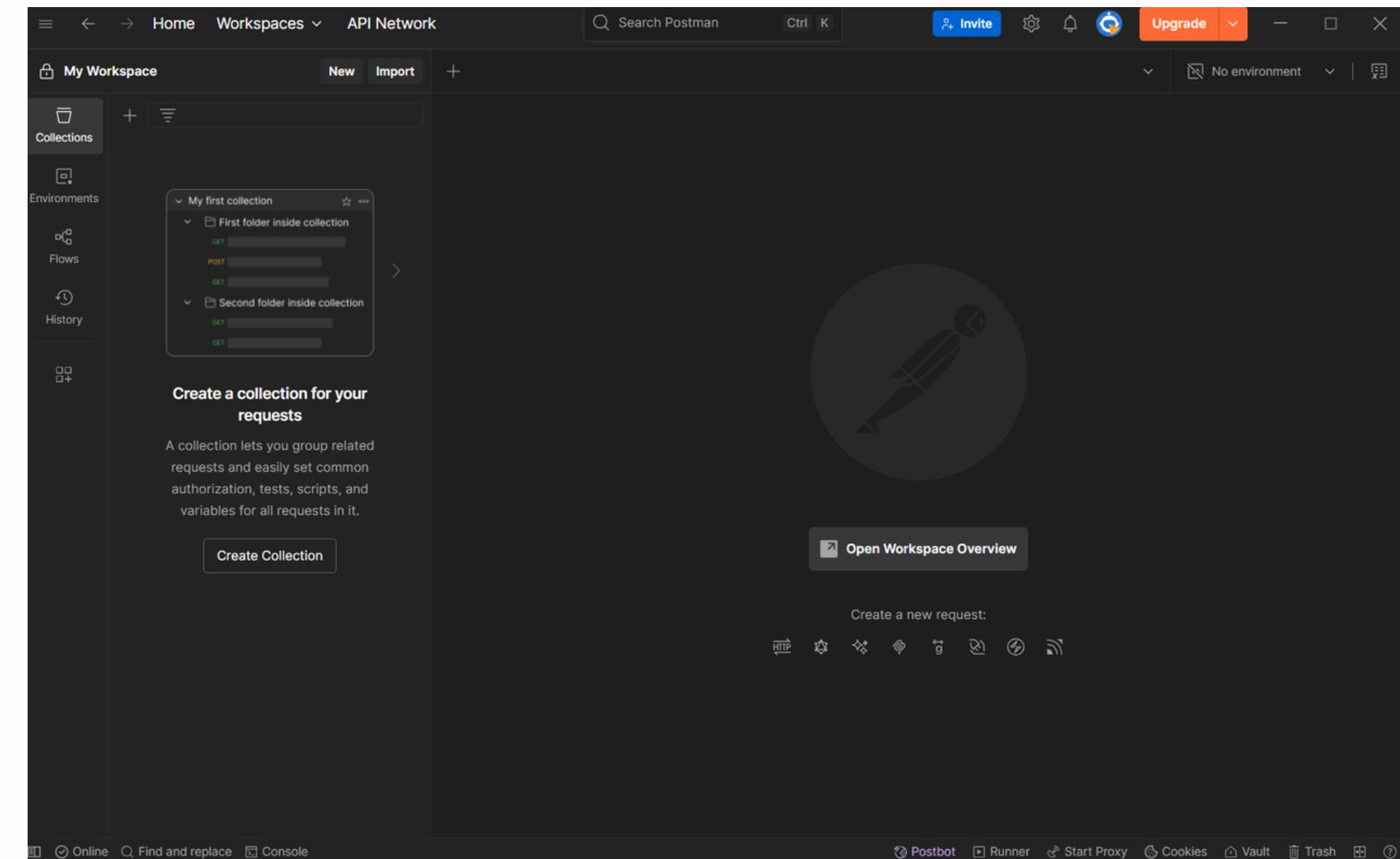
Why Postman?

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

GET

POST

PUT

PATCH

DELETE

HEAD

OPTIONS

Type a new method

API requests - GET

The screenshot shows the Postman application interface. The left sidebar includes sections for My Workspace, Collections, Environments, History, Flows, and Files (BETA). The main workspace displays a collection named "toolshop" containing a single API endpoint named "get cate tree". The endpoint is defined with a GET method and the URL <https://api-with-bugs.practicesoftwaretesting.com/categories/tree>. The "Params" tab is selected, showing a table for Query Params:

Key	Value	Description

Below the table, the response status is shown as 200 OK with a response time of 867 ms and a size of 623 B. The response body is displayed in JSON format, which is currently empty: {}.

At the bottom of the interface, there are navigation links for Cloud View, Find and replace, Console, Terminal, Runner, Start Proxy, Cookies, Vault, Trash, and Help.

API requests - POST

Screenshot of the Postman application interface showing a POST request setup.

The interface includes:

- Header Bar:** Home, Workspaces, API Network, Search Postman, Invite, Settings, Upgrade, Minimize, Close.
- Left Sidebar:** My Workspace, Collections, toolshop, Environments, toolshop, GET get cate tree, POST post cate (selected), History, Flows, Files (BETA).
- Request Overview:** Method: POST, URL: https://api-with-bugs.practicesoftwaretesting.com/categories, Headers: (8), Body (raw JSON):

```
1 {  
2   "name": "new category",  
3   "slug": "new-category"  
4 }
```
- Buttons:** Save, Send, Cookies, Beautify.
- Visuals:** A cartoon astronaut launching a rocket from a small planet.
- Text:** Click Send to get a response.
- Bottom Navigation:** Cloud View, Find and replace, Console, Terminal, Runner, Start Proxy, Cookies, Vault, Trash, Help.

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

The screenshot shows the Postman application interface. At the top, there's a navigation bar with Home, Workspaces, API Network, a search bar, and various tool icons. Below the navigation is a header for "My Workspace" with tabs for New and Import, and a search bar for "toolshop". A "GET script" tab is selected.

The main workspace displays an API endpoint: `https://api.practicesoftwaretesting.com/users`. The method is set to GET. The "Headers (6)" tab is selected in the header section. The "Scripts" tab is also selected, showing two sections: Pre-request and Post-response. The Pre-request script contains a large token value. The Post-response script contains the following code:

```
//  
// Build Authorization header dynamically  
// pm.request.headers.add({  
//   key: "Authorization",  
//   value: "Bearer " + pm.environment.get("token")  
//});
```

Below the script editor, the "Body" tab is selected, showing a JSON response with a single key-value pair: "message": "Unauthorized". The status bar at the bottom indicates a 401 Unauthorized response with 572 ms duration and 343 B size.

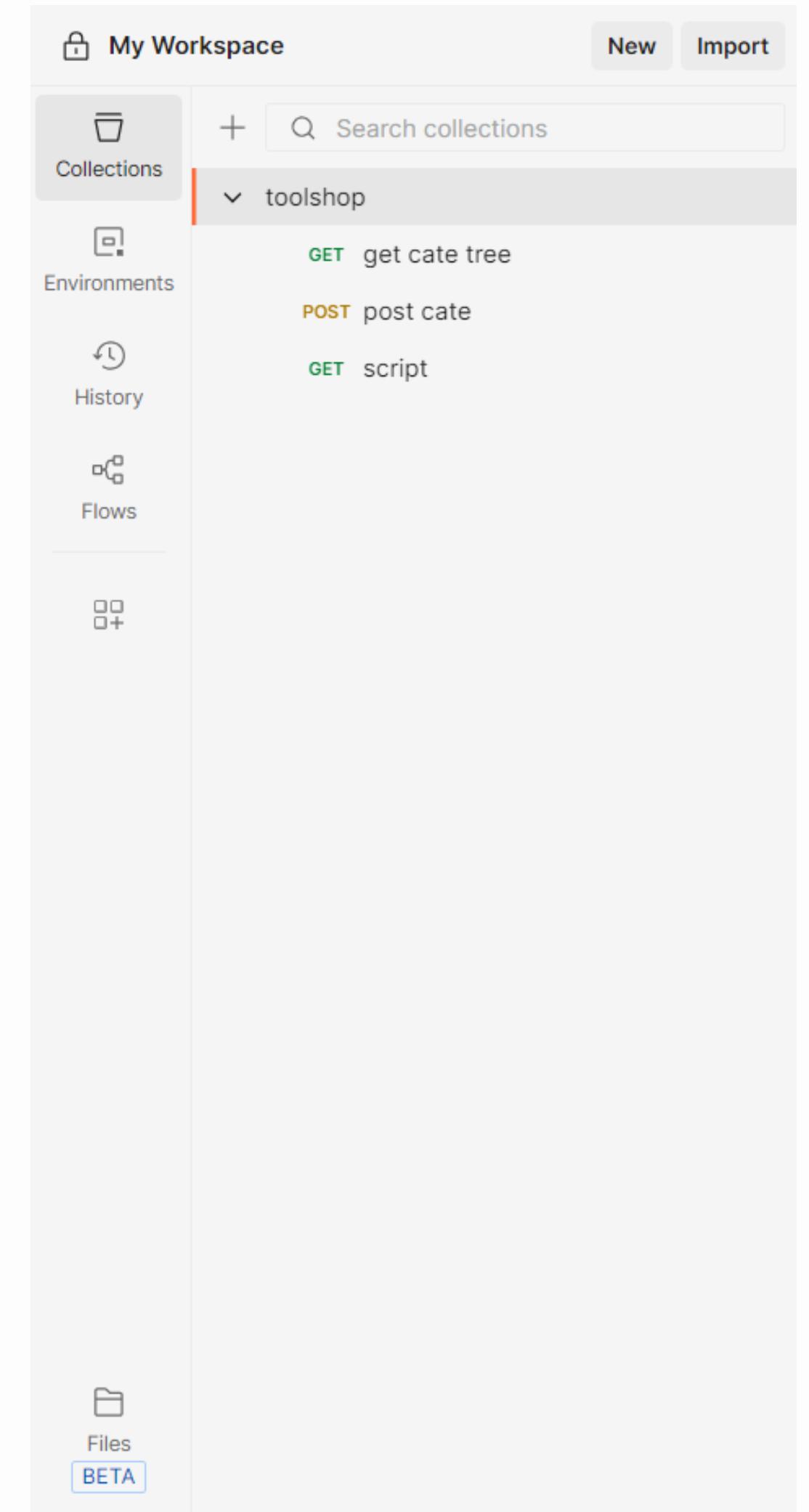
Collections

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



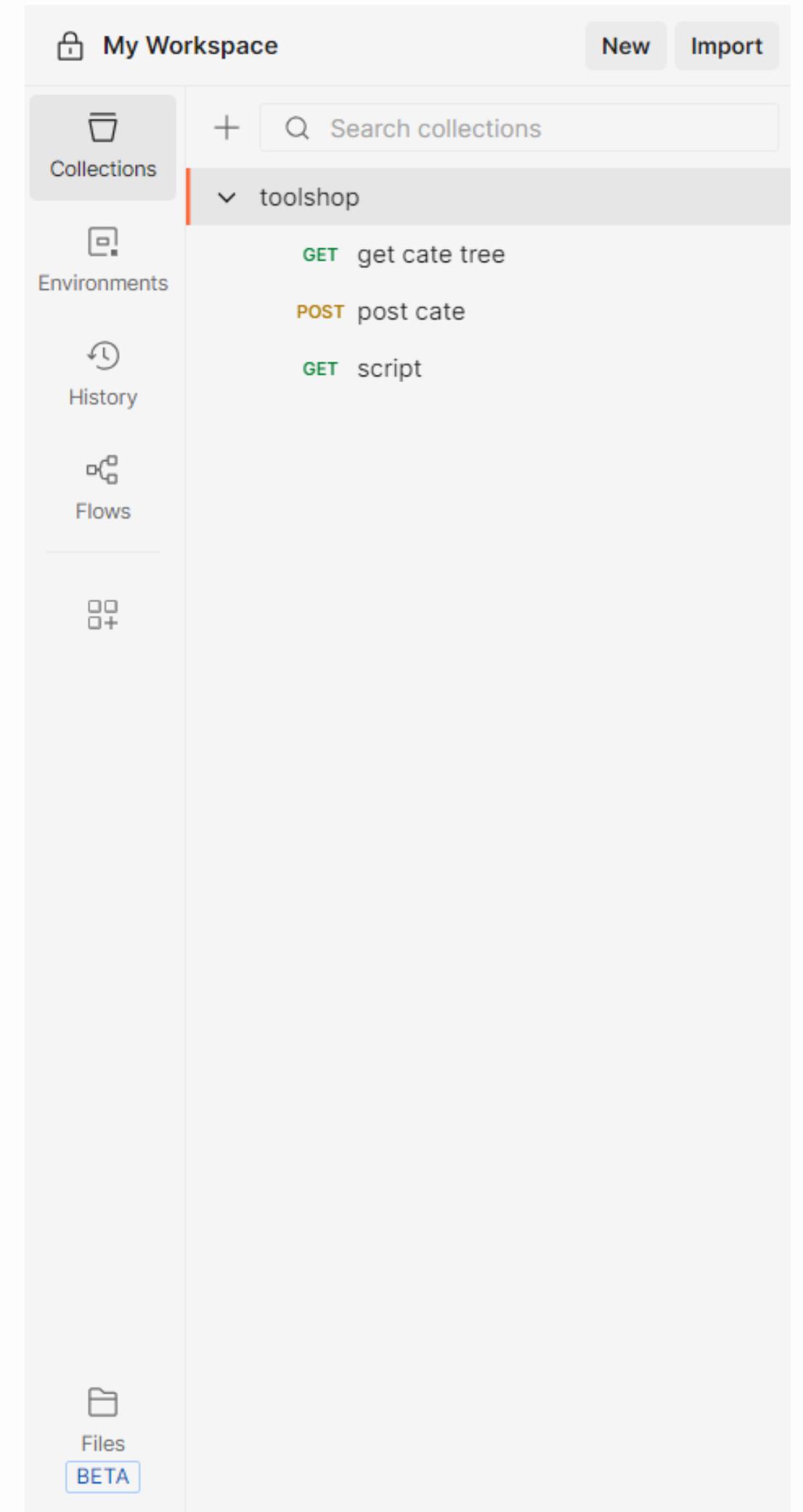
Collections

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

Globals		 Export
Variable	Value	 
BASE_BUG_URL	https://api-with-bugs.practicesoftwaretesting.com	
BASE_PROD_URL	https://api.practicesoftwaretesting.com	
Add variable		

Environment Variables

The screenshot shows the toolshop interface with a search bar containing '{B'. A dropdown menu is open, listing various environment variables starting with 'B':
- BASE_BUG_URL
- BASE_PROD_URL
- \$randomAbbreviation
- \$randomAbstractImage
- \$randomBankAccount
- \$randomBankAccountBic
- \$randomBankAccountIban
- \$randomBankAccountName

The screenshot shows the toolshop interface with a search bar containing '{{BASE_BUG_URL}}'. The 'Body' tab is selected, showing the placeholder text: "This request does not have a body".

Environment Variables

The screenshot shows the Postman application interface. At the top, the URL is `toolshop / script + env`. Below the URL, there's a search bar with the placeholder `{BASE_BUG_URL} /categories/tree`. To the right of the search bar are "Save" and "Share" buttons. Underneath the search bar, the method is set to "GET". A "Send" button is located to the right of the URL input.

The "Headers" tab is selected, showing the value `6`. Below the headers, there are several options for "Body": "none" (selected), "form-data", "x-www-form-urlencoded", "raw", "binary", and "GraphQL". A note below says "This request does not have a body".

The "Cookies" tab is also visible. On the left side, there are tabs for "Body", "Cookies", "Headers (12)", and "Test Results". The "Body" tab is selected, showing the response content. The response status is "200 OK" with a green background. The response time is "789 ms", size is "623 B", and it includes a "Save Response" button.

The response body is displayed as JSON, numbered from 1 to 21. The JSON structure represents a tree of categories:

```
1 [  
2 {  
3   "id": 1,  
4   "parent_id": null,  
5   "name": "Hand Tools",  
6   "slug": "hand-tools",  
7   "sub_categories": [  
8     {  
9       "id": 3,  
10      "parent_id": 1,  
11      "name": "Hammer",  
12      "slug": "hammer",  
13      "sub_categories": []  
14    },  
15    {  
16      "id": 4,  
17      "parent_id": 1,  
18      "name": "Hand Saw",  
19      "slug": "hand-saw",  
20      "sub_categories": []  
21    }  
].
```

At the bottom of the interface, there are buttons for "Runner", "Start Proxy", "Cookies", "Vault", "Trash", and a help icon.

Environment Variables

Screenshot of the Postman application interface showing a request setup and an error message.

The top navigation bar includes Home, Workspaces, API Network, Search Postman, Invite, Settings, and Upgrade.

The left sidebar shows My Workspace, Collections (toolshop), Environments, History, Flows, and Files (BETA).

The main workspace displays a collection named "toolshop" with a "script + env" request. The request method is GET, URL is {{api_url}}, and the body contains the script:

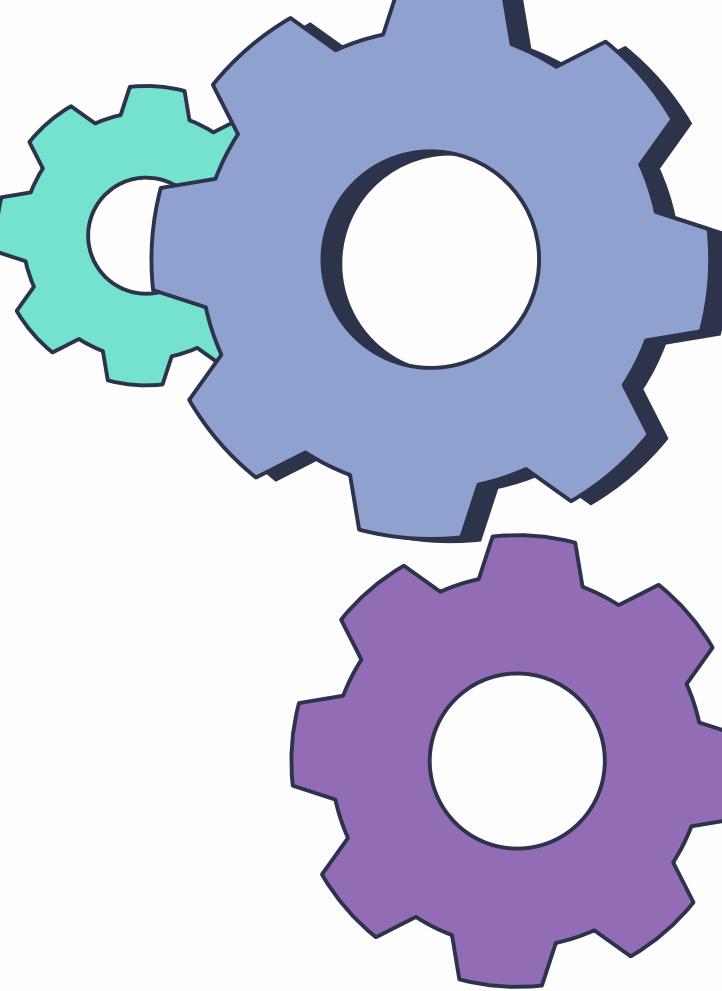
```
// // Set API URL  
// pm.environment.set("api_url", "https://api-with-bugs.practicesoftwaretesting.com/categories/tree");
```

The "Headers" tab shows 6 items. The "Response" tab indicates "Could not send request".

An error message at the bottom states: "Error: getaddrinfo ENOTFOUND {{api_url}} | Debug with AI".

The bottom navigation bar includes Cloud View, Find and replace, Console, Terminal, Runner, Start Proxy, Cookies, Vault, Trash, and Help.

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

+ Search collections

Linear Execution / Post req

Save Share

Collections Environments History Flows

POST https://api-with-bugs.practicesoftwaretesting.com/products

Send

Docs Params Authorization Headers (8) Body Scripts Settings Cookies Schema Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2 "name": "{{name}}",
3 "description": "string",
4 "price": {{price}},
5 "category_id": {{category_id}},
6 "brand_id": {{brand_id}},
7 "product_image_id": {{product_image_id}},
8 "is_location_offer": {{is_location_offer}},
9 "is_rental": {{is_rental}},
10 "co2_rating": "B"
11
```

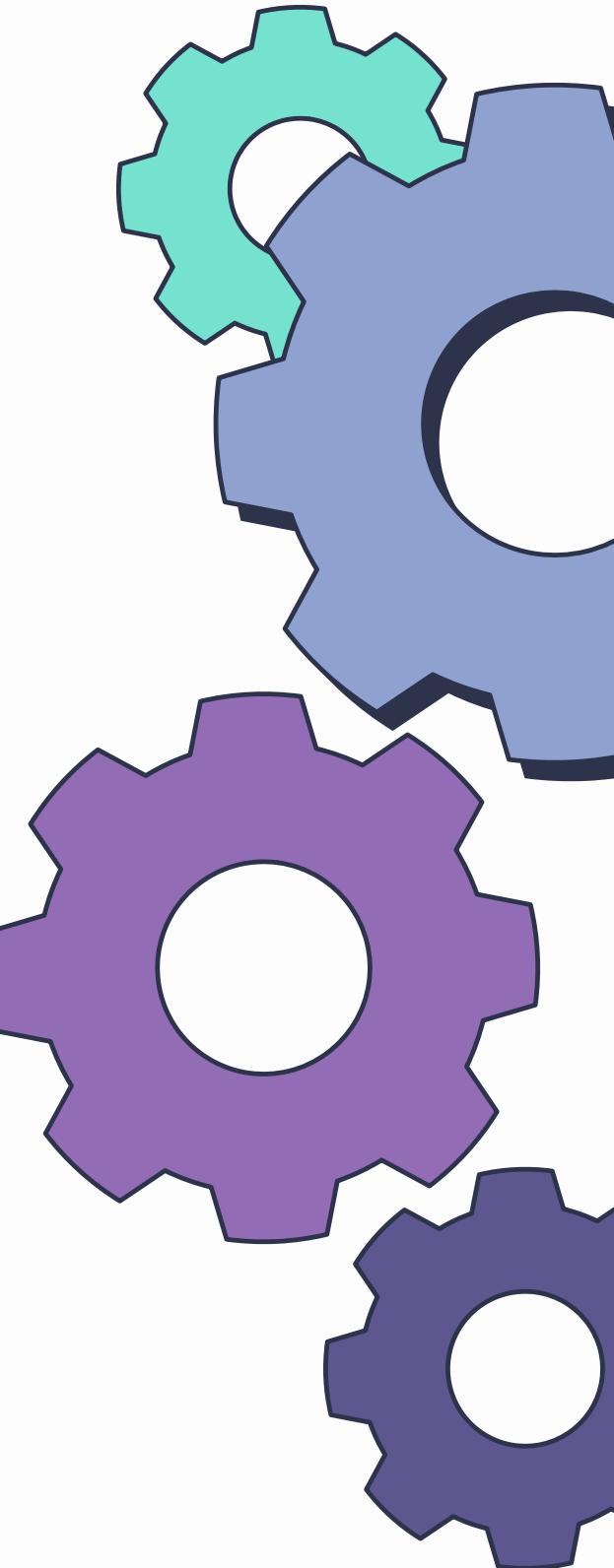
Body Cookies Headers (9) Test Results

201 Created 243 ms 507 B Save Response

{ JSON Preview Visualize

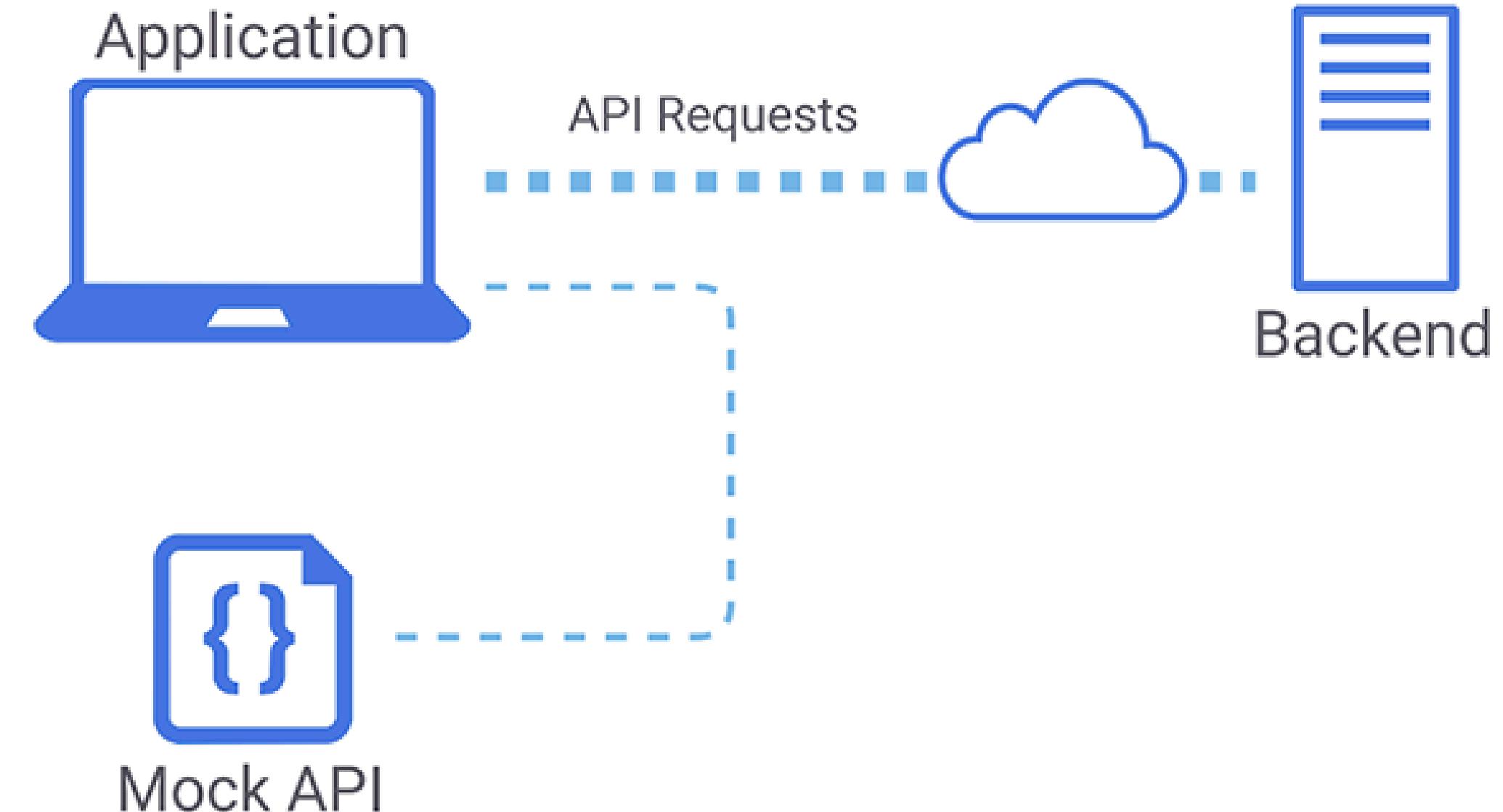
```
1 {
2   "name": "partnerships Account",
3   "description": "string",
4   "price": 21.93,
5   "category_id": 6,
6   "brand_id": 7,
7   "product_image_id": 8,
8   "is_location_offer": 1,
9   "is_rental": 1,
10  "co2_rating": "B",
11}
```

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:

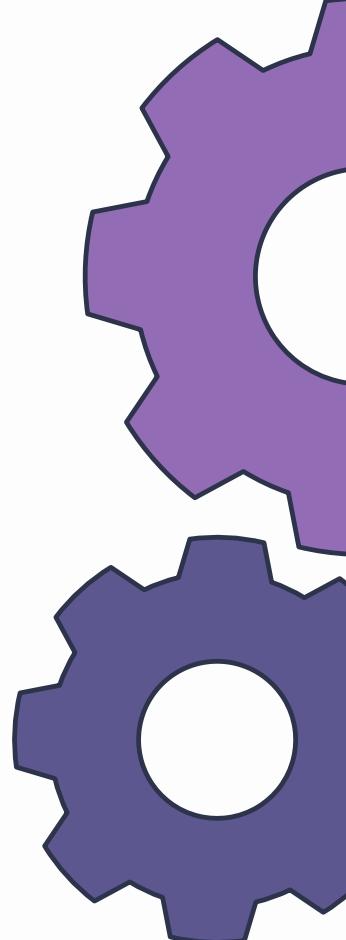
- Easily simulate rare error conditions or unusual/unexpected data responses.
- Ensures the application handles extreme scenarios gracefully.

3. Supports Stable Testing

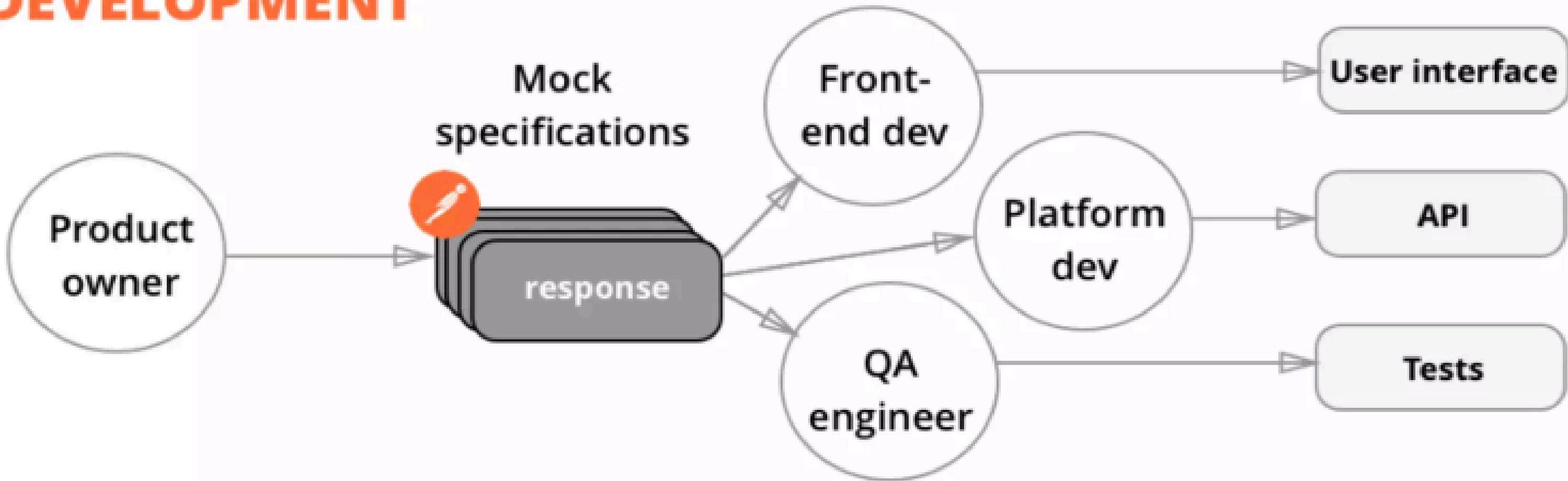
- Provides consistent, controlled responses for reliable, repeatable tests.
- Testers aren't affected by real API downtime or sudden changes

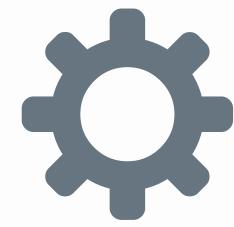
4. Improves Early Bug Detection

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



Mocks for continuous **DEVELOPMENT**





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.

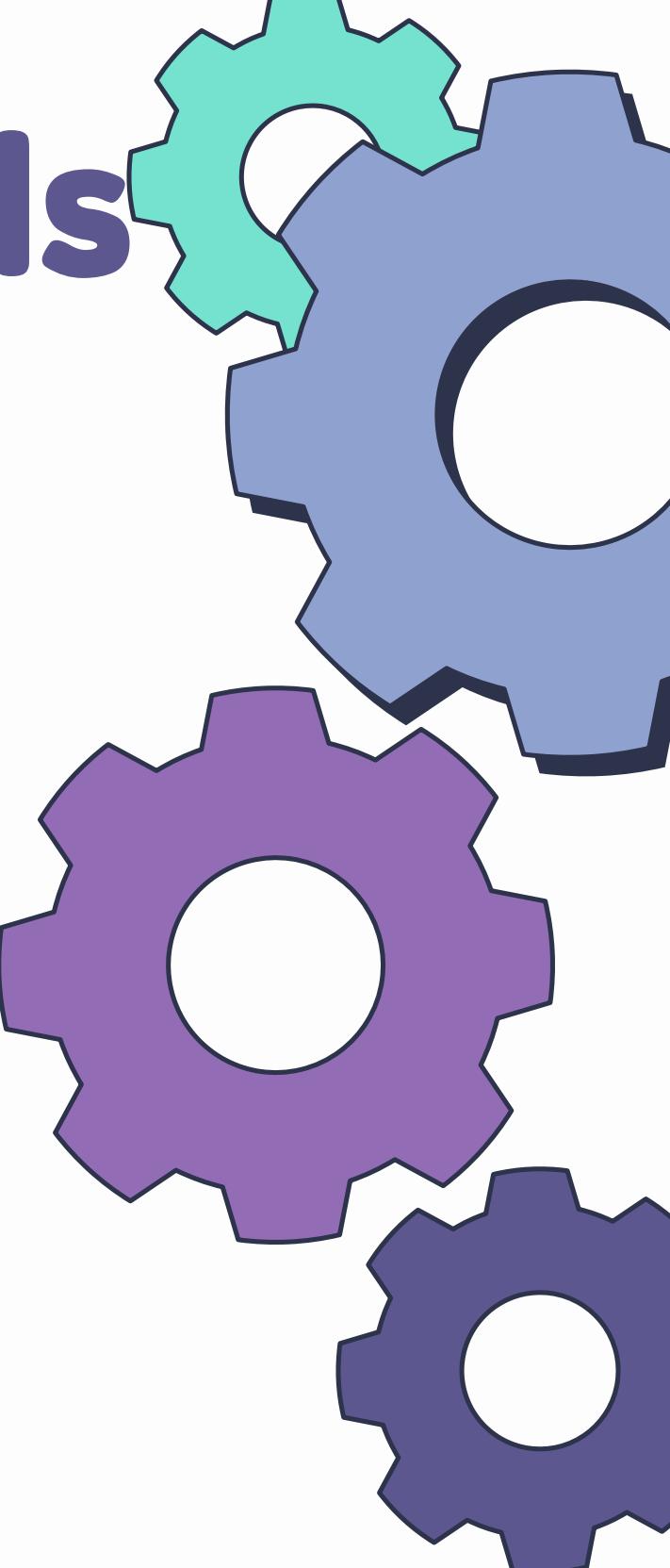
4. Behavior-driven Mocks:

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



Popular Tools for Mocking APIs

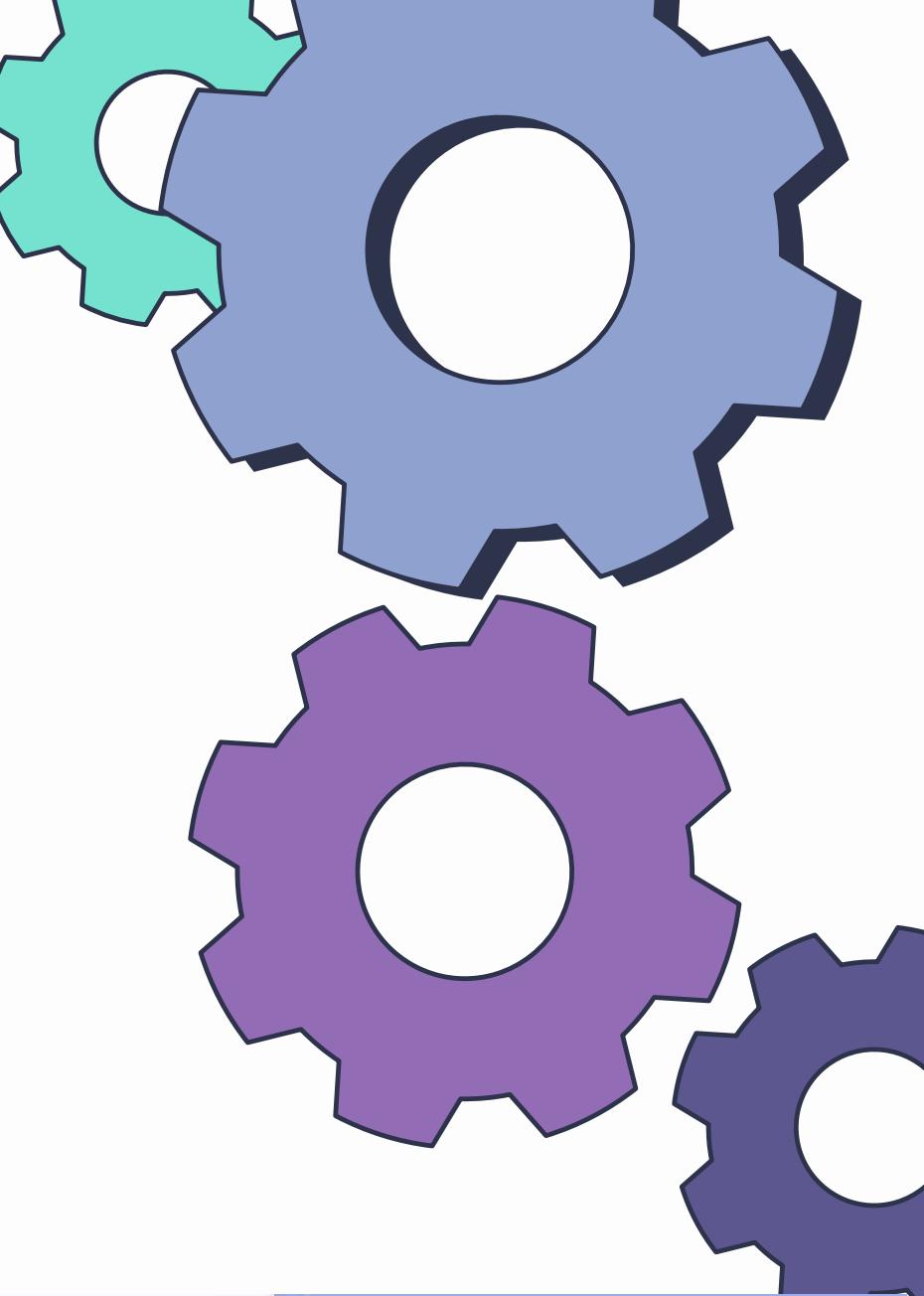
- **Postman** : a popular API development environment that supports creating mock servers
- **WireMock** : a flexible, open-source API mocking tool that runs as a standalone server or embedded in tests.
- **Mockoon** : a desktop application that allows you to create mock APIs quickly.
- **JSON Server**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



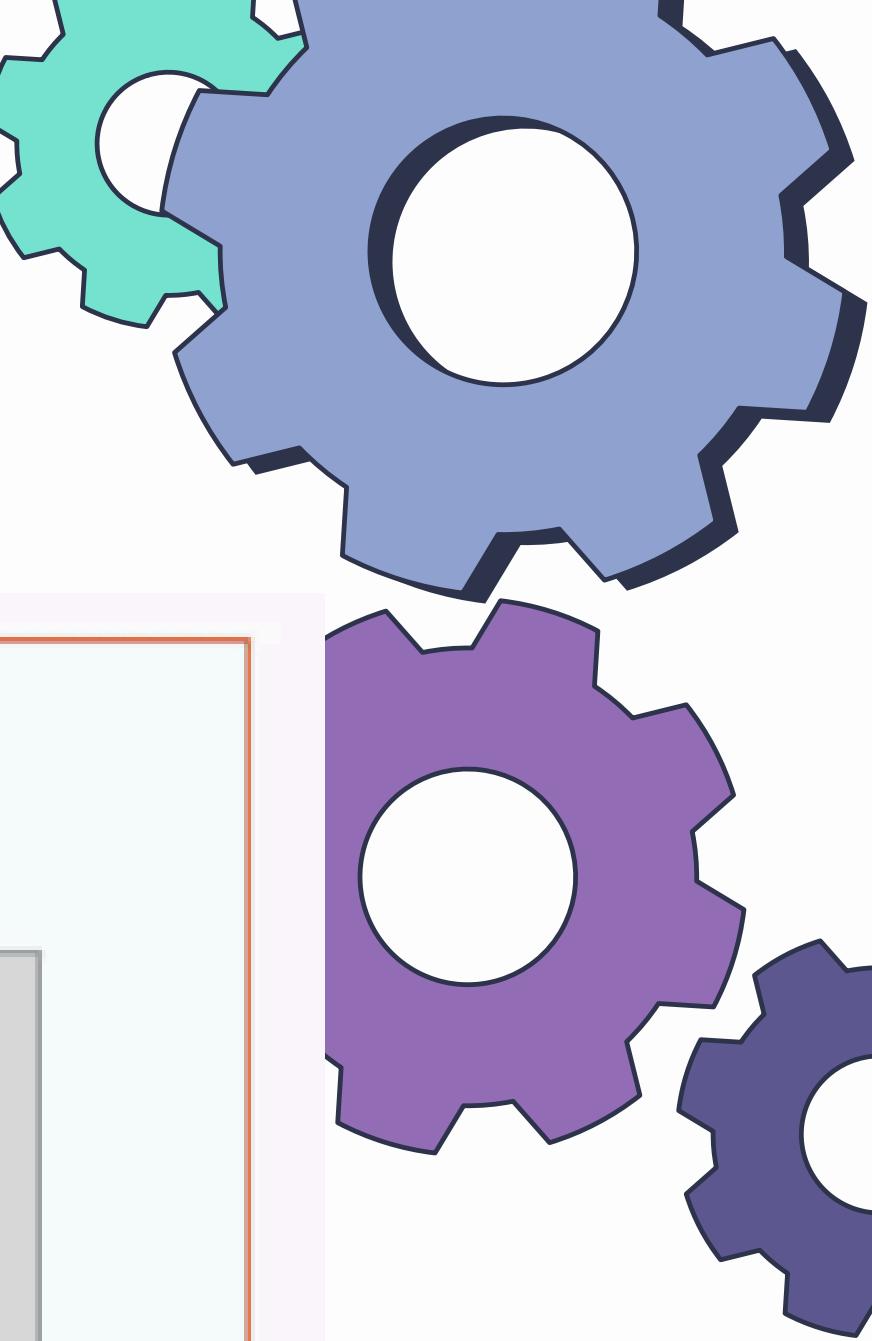
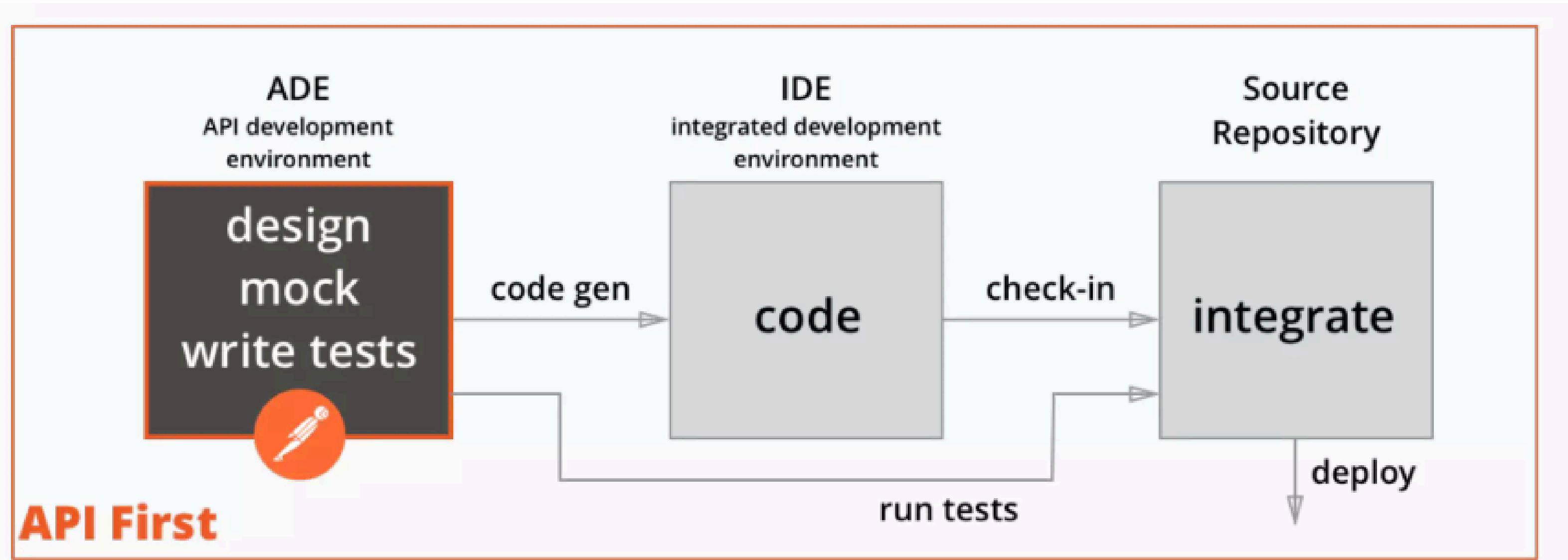
Postman mock server

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

POSTMAN
MOCK SERVER



How we use mocks



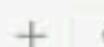
My Workspace

New Import

Create mock server • +

▼

No environment ▼



Search mock servers

Collections



Environments



Flows



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.

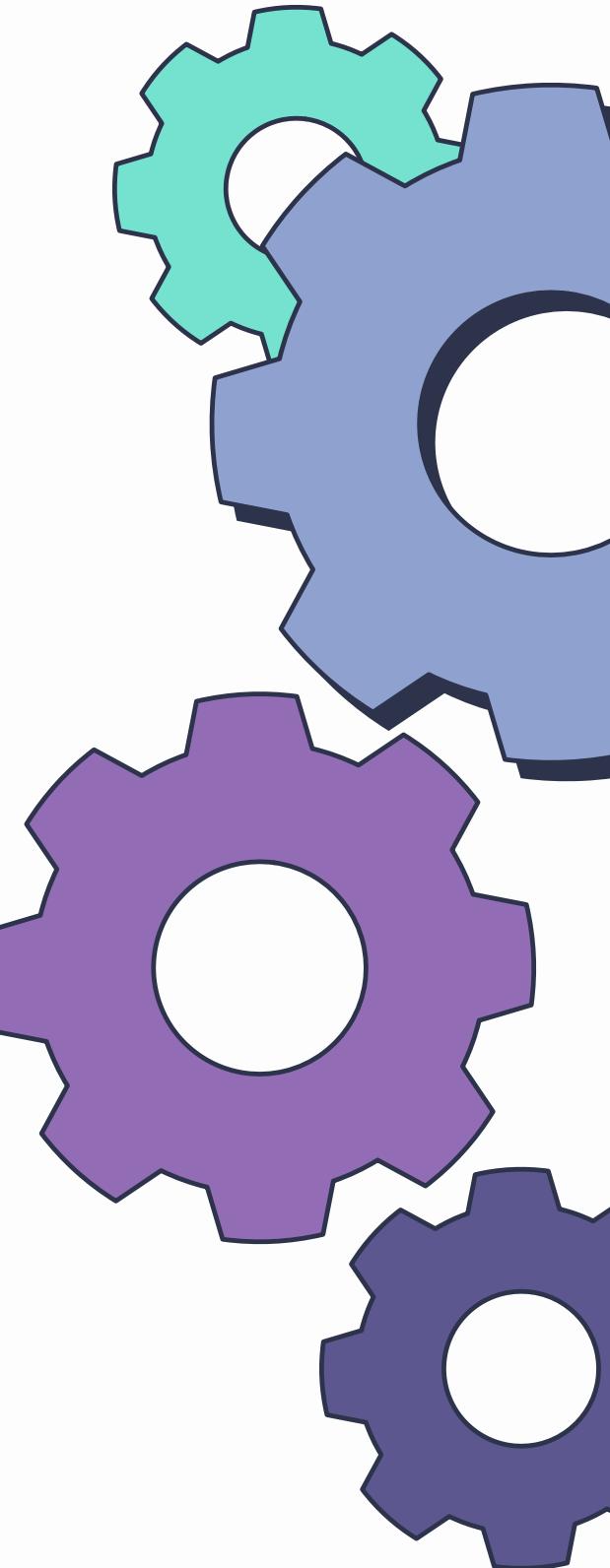
 Make mock server private

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

3. AI-Approach for API Testing



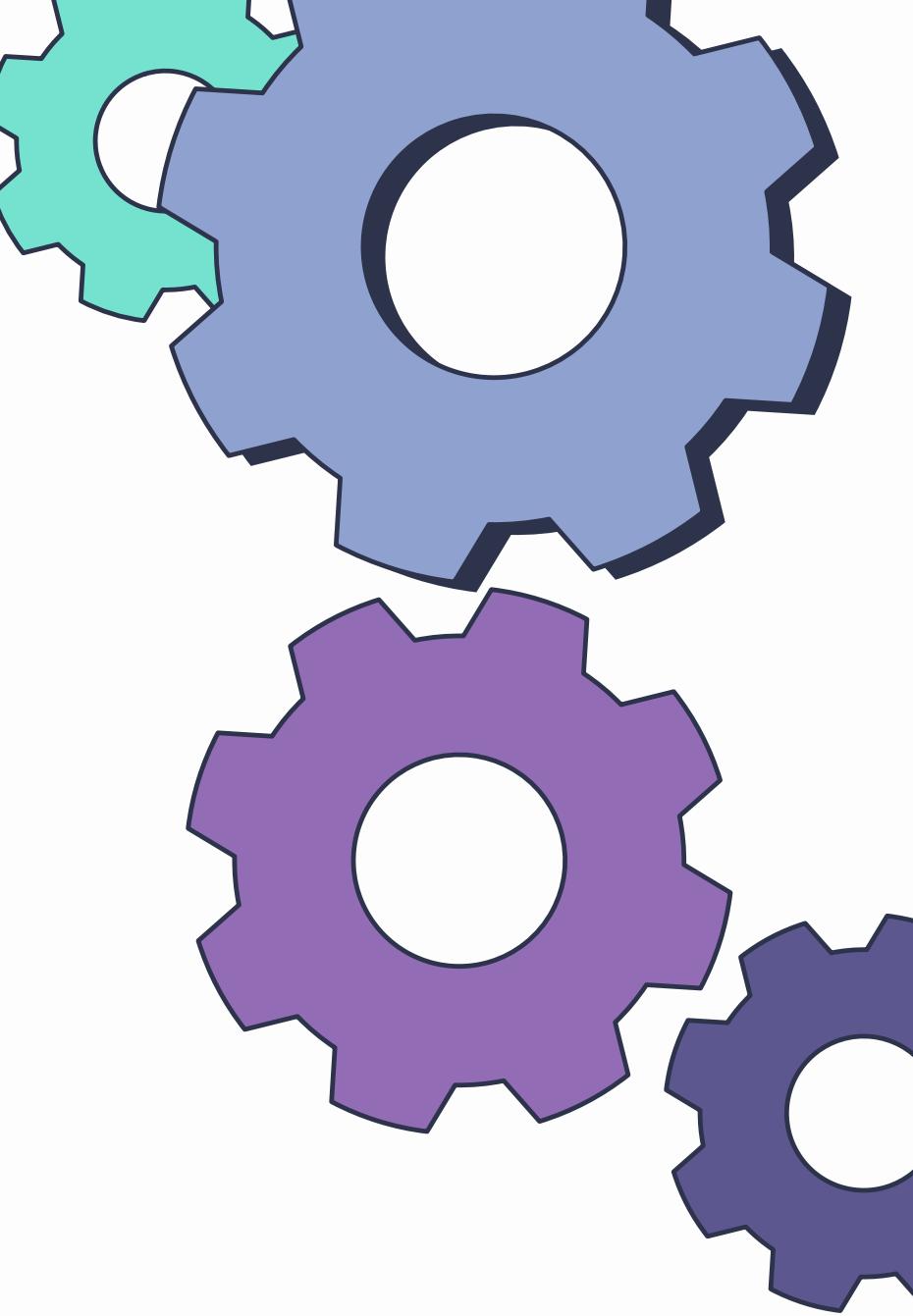
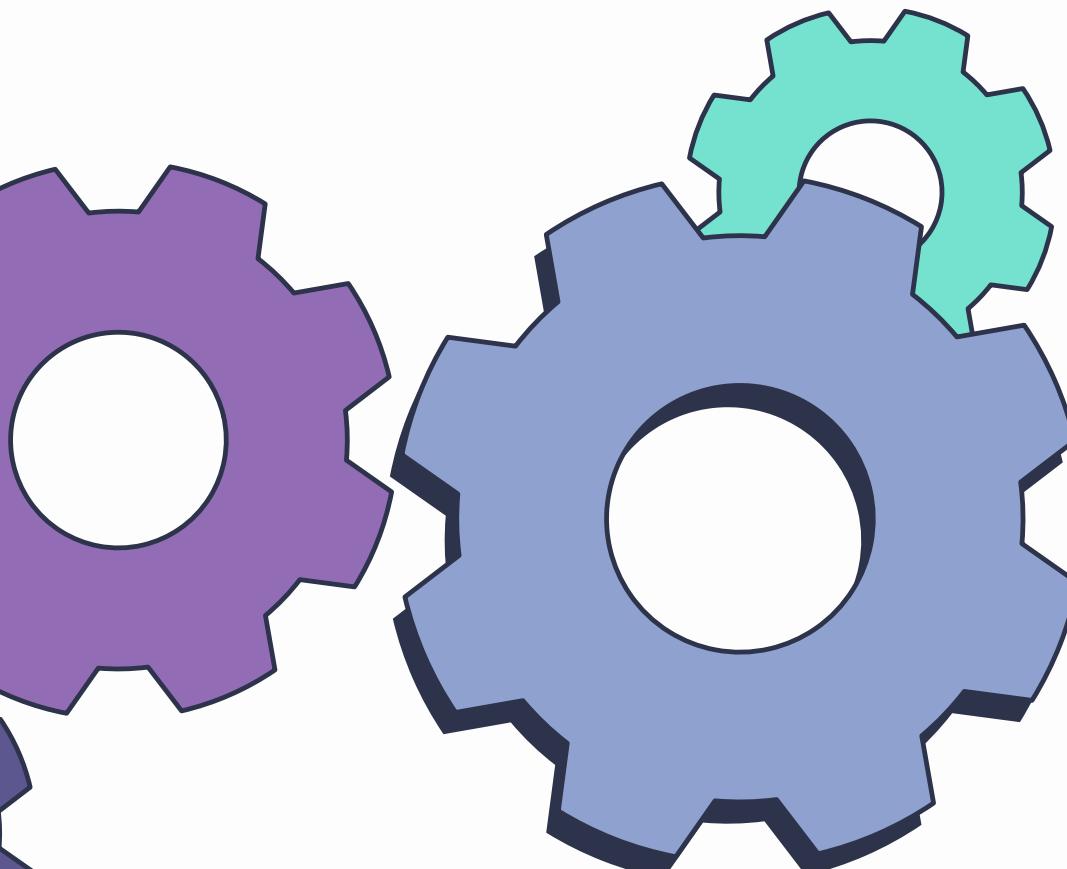
Why AI for API Testing?

- 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)

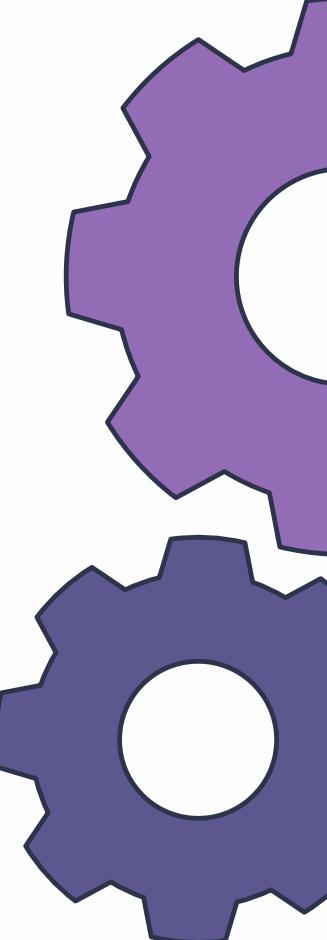
3. AI-Enhanced Automation Scripts

- 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

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, Environments, History, Flows, and Files (BETA). The main area shows a collection named "My Collection" containing three items: "GET Get cate", "GET Get all product", and "GET Get product". A specific test step for "GET Get cate" is selected, with the URL set to "https://api.practicesoftwaretesting.com/categories". The "Scripts" tab is active in the request editor. Below the request, there are sections for "Pre-request" and "Post-response". The response pane at the bottom shows a successful "200 OK" status with a response time of 764 ms and a body size of 775 B. The response body contains several headers: Date, Server, and Charset. A large red annotation with a red arrow points to the "Save Response" button in the response summary bar, with the text "Click the icon at the bottom to access the AI." overlaid.

d dün's Workspace

New Import GET Get all product GET Get product My Collection GET GET cate + No environment

Collections + Search collections My Collection GET Get cate GET Get all product GET Get product

Environments

History

Flows

GET https://api.practicesoftwaretesting.com/categories

Send

Docs Params Authorization Headers (6) Body Scripts Settings Cookies

Pre-request

1 Use JavaScript to write tests, visualize response, and more. Ctrl+Alt+P to Ask AI

Post-response

Packages Snippets

Body Cookies Headers (12) Test Results 200 OK 764 ms 775 B Save Response ...

Key	Value
Date	Sun, 07 Dec 2025 18:12:17 GMT
Server	Apache/2.4.52 (Ubuntu)
Charset	utf-8

Click the icon at the bottom to access the AI.

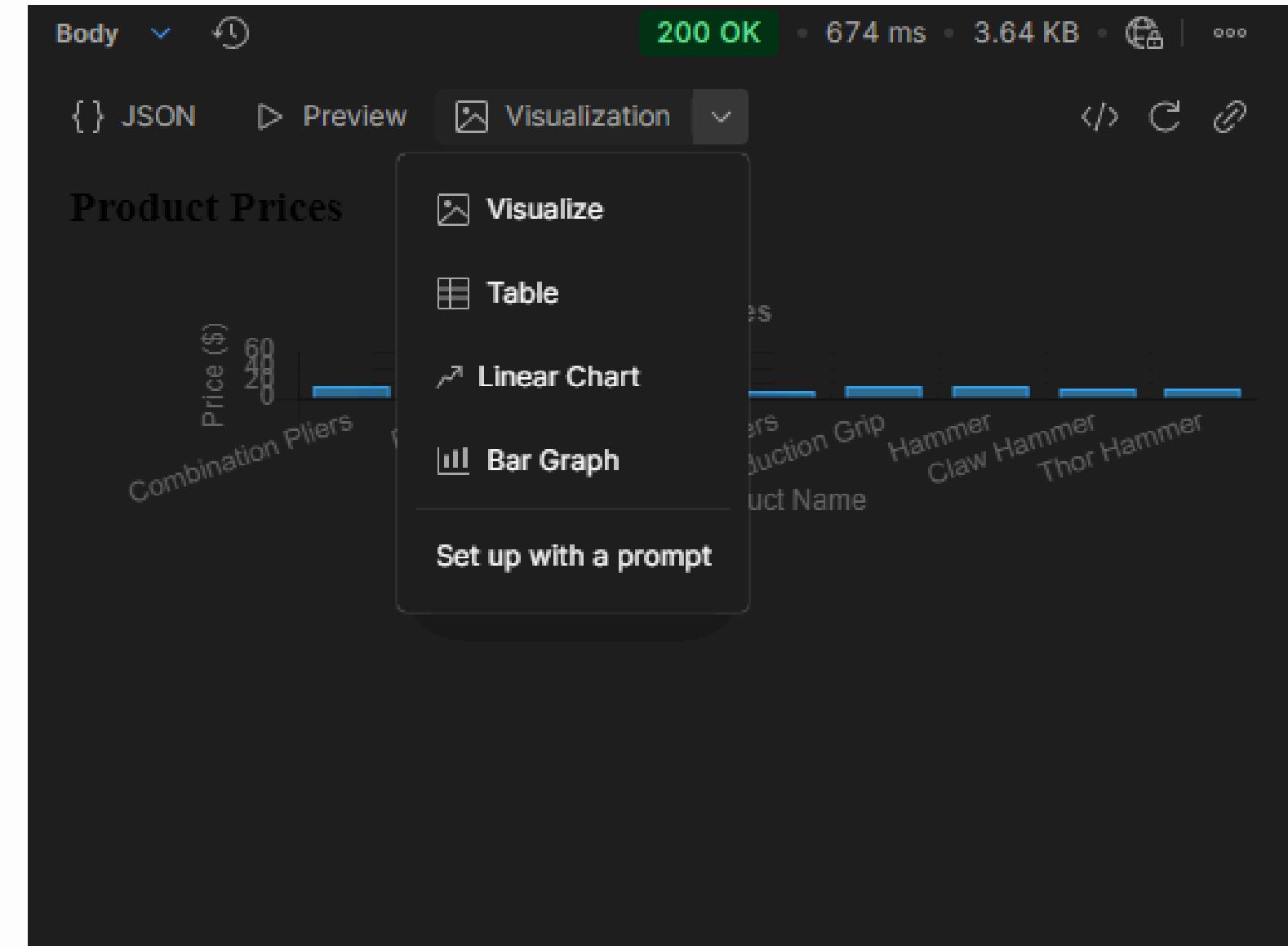
Open Agent Ctrl+Alt+P

Postbot Visualize responses

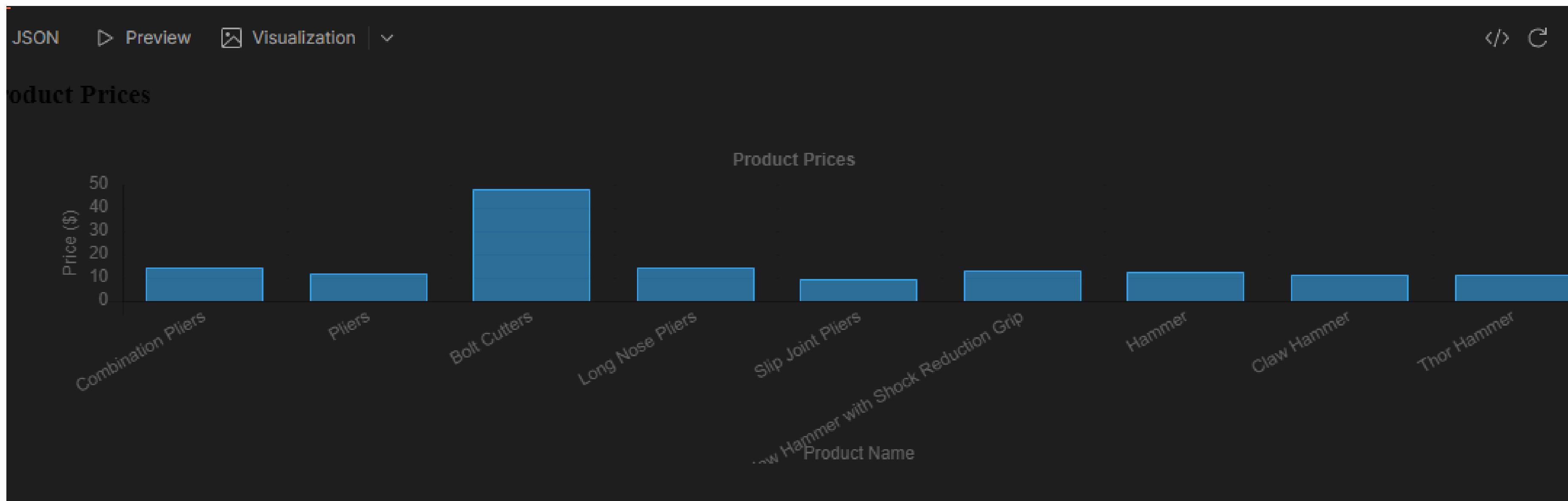
You can use a prompt like:

- 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



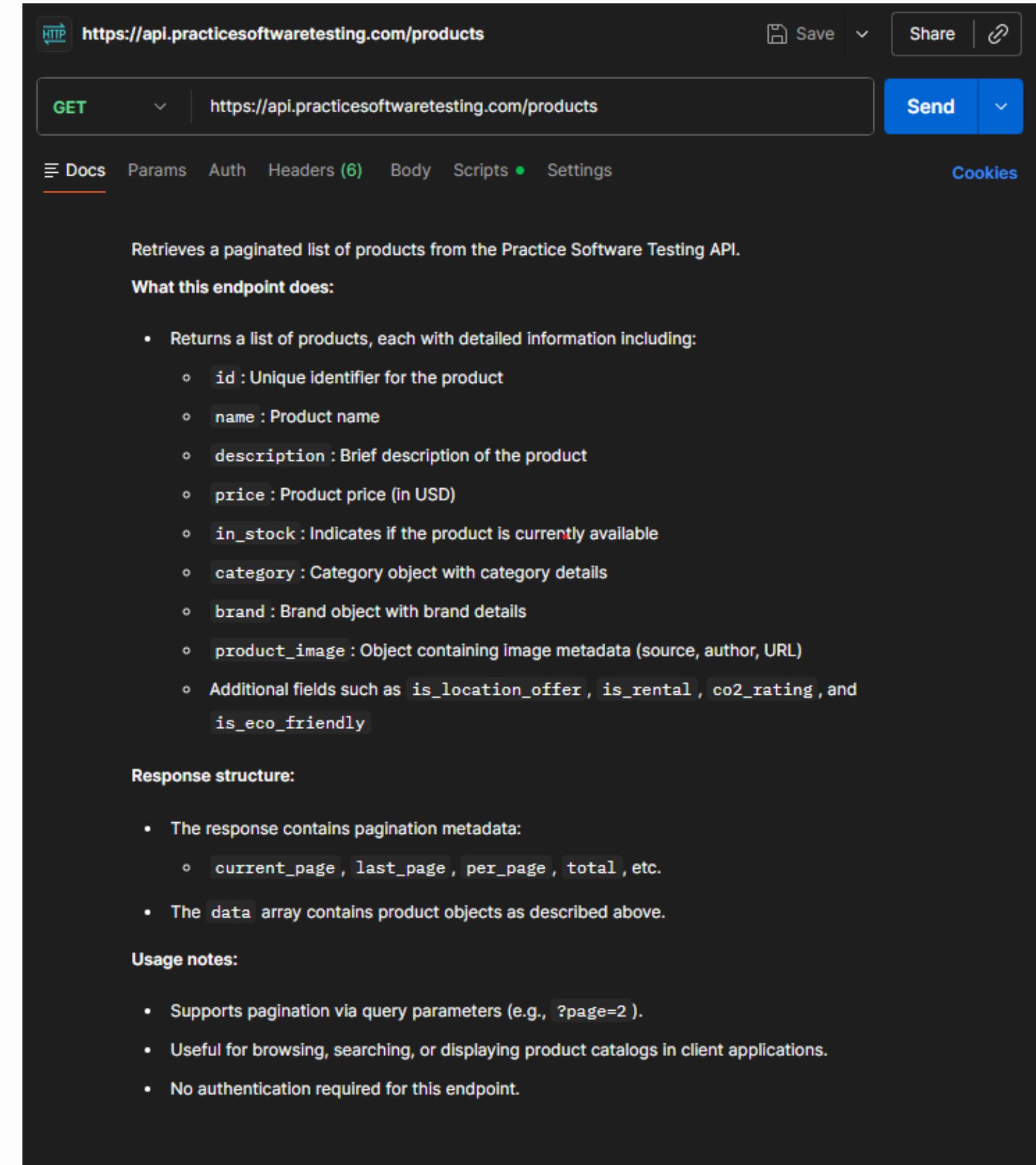
Visualization result



Postbot Write documentation

Simply click a button

Alternatively, using natural language prompt
for more detail docs



The screenshot shows a dark-themed API documentation interface. At the top, there's a header with 'HTTP' and the URL 'https://api.practicesoftwaretesting.com/products'. To the right are 'Save', 'Share', and 'Send' buttons. Below the header, the URL 'https://api.practicesoftwaretesting.com/products' is repeated in a search bar. A navigation bar below the search bar includes 'Docs' (which is underlined in orange), 'Params', 'Auth', 'Headers (6)', 'Body', 'Scripts', and 'Settings'. On the far right is a 'Cookies' link. The main content area starts with a description: 'Retrieves a paginated list of products from the Practice Software Testing API.' followed by 'What this endpoint does:' and a bulleted list of product details. It then moves on to 'Response structure:' and 'Usage notes:' sections.

Retrieves a paginated list of products from the Practice Software Testing API.

What this endpoint does:

- Returns a list of products, each with detailed information including:
 - `id` : Unique identifier for the product
 - `name` : Product name
 - `description` : Brief description of the product
 - `price` : Product price (in USD)
 - `in_stock` : Indicates if the product is currently available
 - `category` : Category object with category details
 - `brand` : Brand object with brand details
 - `product_image` : Object containing image metadata (source, author, URL)
 - Additional fields such as `is_location_offer`, `is_rental`, `co2_rating`, and `is_eco_friendly`

Response structure:

- The response contains pagination metadata:
 - `current_page`, `last_page`, `per_page`, `total`, etc.
- The `data` array contains product objects as described above.

Usage notes:

- Supports pagination via query parameters (e.g., `?page=2`).
- Useful for browsing, searching, or displaying product catalogs in client applications.
- No authentication required for this endpoint.

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

No awareness of system environment

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

AI may misinterpret poorly written docs

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

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

