



## Introduction of API Web Services

## Introduction of API Web Services

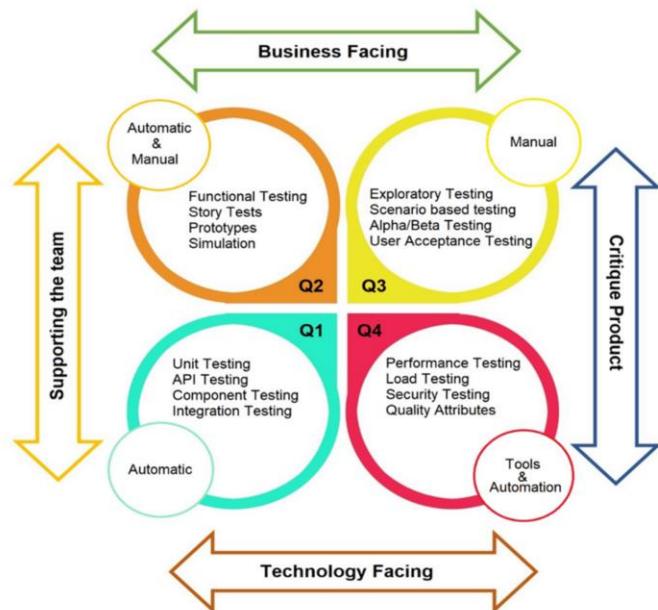


**RUBRIC**

2

- In this chapter, you will learn about:
  - What is a Web Service?
  - Basic concept behind web services
  - How web services work
  - Types of Web Services
    - REST Web Service
    - SOAP Web Service
  - API Testing
  - How to Test APIs?
  - Importance of API Testing
  - Types of API tests

## Agile Testing quadrants



3

Quadrant 1: Technology-Facing Tests that Support the Team as it focuses on verifying the functionality and correctness of the API at a technical level, usually through automated tests, providing quick feedback to the development team during the development process.

## Web Service (1)

- A **web service** is a collection of **open protocols** and standards used for **exchanging data between applications or systems**.
- Software applications written in various programming languages and running on various platforms can use **web services** to exchange data over computer networks like the Internet in a manner like **inter-process communication on a single computer**.



Simple definition: Web service is a service available over web.

There can be an application written in Java over a Linux platform and it uses Oracle DB. There is another application written in C++ over Windows platform and it uses SQL Server DB. These 2 applications can communicate with each other with the help of web service

## Web Service (2)

- In plain English:
  - Allow a program to talk to a web page, instead of using a browser to open a web page.
  - The key is providing data as a service instead of U.I
  - Independent application components which are available over the web

## Web Service (3) – How Skyscanner Fetches Real-Time Flight Information?



6

### Use Case Scenario: User Searches for a Flight

A traveler visits Skyscanner and searches for flights from Mauritius to Brussels on [Skyscanner](#). Within seconds, a list of available flights appears, showing different airlines, prices, and schedules.

### Key Question:

1. How does Skyscanner fetch real-time flight information from multiple airlines so quickly?
2. Does Skyscanner have access to all airline databases?

The answer is **NO**. Airlines do not expose their databases directly. There must be security measures in place.

### Possible Solution:

One might assume that airlines dump flight data every five minutes into a shared repository, which Skyscanner retrieves.

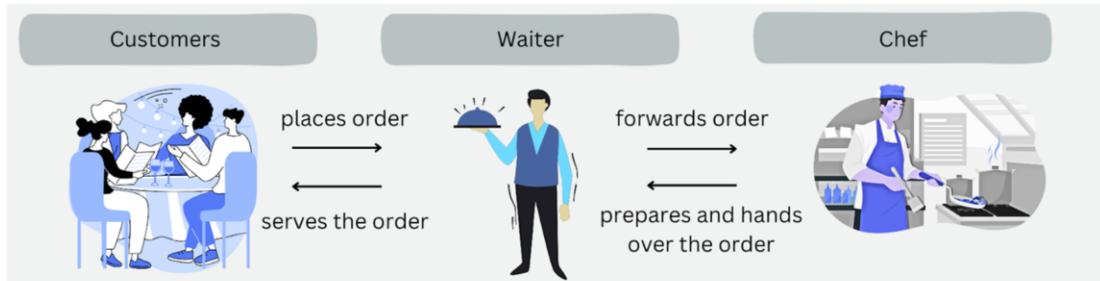
However, this approach has a flaw:

If a flight is **canceled** or **fully booked** within those five minutes, the displayed data would be **outdated**, leading to booking errors and customer dissatisfaction.

### The Real Solution – Web Services and APIs:

Instead of relying on outdated data dumps, airlines expose their flight information through **secure web services (APIs)**. Skyscanner interacts with these APIs in real-time to get the latest flight availability, prices, and schedules.

## Web Service (4) – Restaurant



7

Let us say we got to a hotel or a restaurant for a dinner:

### 1. Customers

- This is **you** when you walk into a restaurant and order food.

### 2. Waiter

- The **waiter** takes your order, delivers it to the kitchen, and brings the food back.
- The waiter acts as a **messenger** between you and the kitchen/chef.
- The waiter needs to make sure that the kitchen prepares the order and serve you back.

### 3. Chef/Kitchen

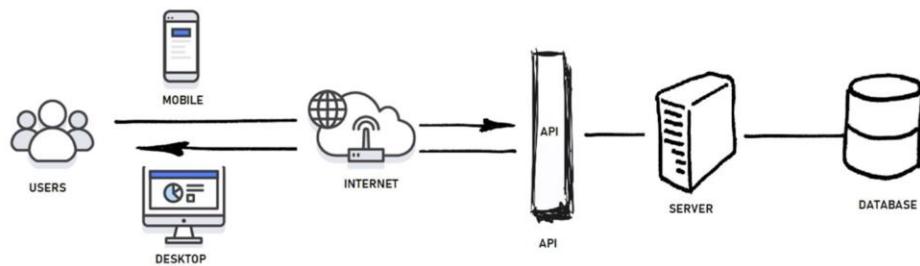
- The **kitchen** is where the food (data) is prepared by the chef.
- The waiter then picks the food and delivers it back to you.

## What is an API?

- An API, which stands for **Application Programming Interface**, is a set of **protocols** that enable different software components to **communicate** and **transfer data**.
- Developers use APIs to bridge the gaps between small, discrete chunks of code to **create** applications that are powerful, resilient, secure, and able to meet user needs.
- Even though you can't see them, APIs are everywhere—working continuously in the background to power the digital experiences that are essential to our modern lives.

## What is an API?

- APIs work by sharing data between applications, systems, and devices. This happens through a **request** and **response** cycle. The request is sent to the API, which retrieves the data and returns it to the user.
- Here's a high-level overview of how that process works:



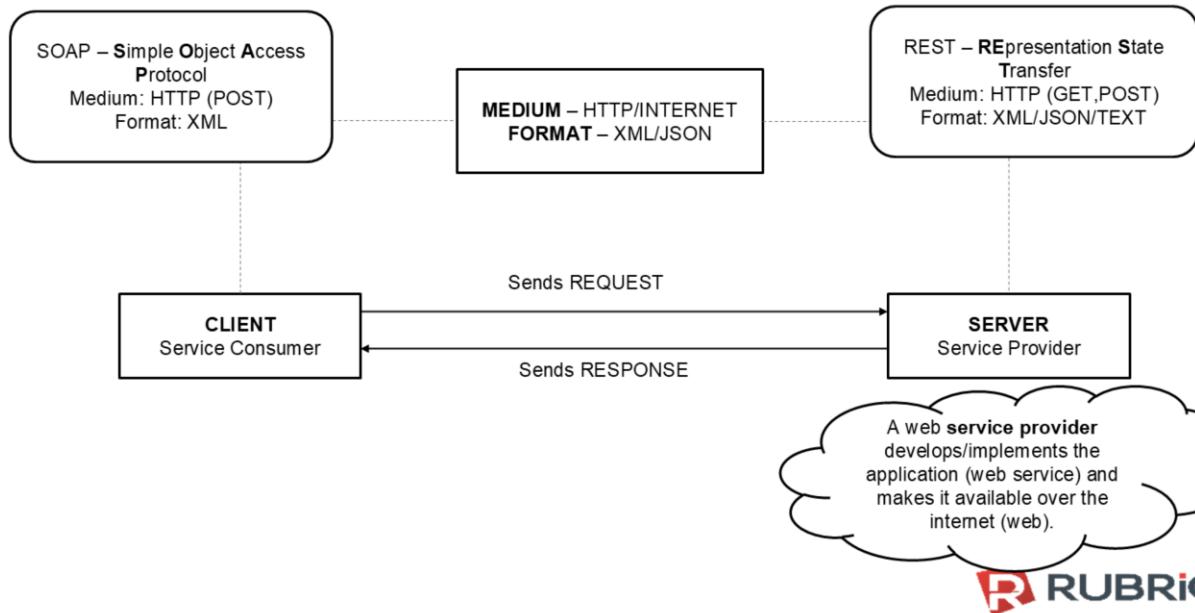
## What is an API Request?

- An API Request consists of:
  - Endpoints
  - Header
  - Method
  - Data
  - Authentication

10

- **Endpoints**
  - There are two key parts to an endpoint that are used when making an API request.
  - One of which is the URL.
  - The second part is the path. The path will vary depending on what you are trying to accomplish.
- **Header**
  - Some APIs require you to send headers along with requests, typically to provide additional metadata about the operation you are performing.
- **Method**
  - Methods are the actions taken when sending a request which are the API methods.
- **Data**
  - The request data, also commonly referred to as the “body,” is information that will be either sent to or returned by a server.
- **Authentication**
  - Some APIs require auth details. Authentication involves verifying the identity of the client sending a request, and authorization involves verifying that the client has permission to carry out the endpoint operation.

## How does API Web Service works?



11

A web service facilitates communication between two applications. But how does this communication take place?

In web service, there are **two** key components:

1. **Consumer** (the application requesting the service)
2. **Service Provider** (the application offering the service).

To enable communication between them, **two** essential elements are required:

1. **Medium** – In the case of web services, the internet serves as the medium.
2. **Format** – A common data format, such as JSON or XML, allows web services to interact and exchange information effectively.

Think of it like a phone conversation with a friend: the **phone** acts as the **medium**, while the **language** (e.g., English) serves as the **format** that both participants understand.

There are 2 types of Web Services, or it can be said that web services are implemented into 2 types:

1. **SOAP** (simple object access protocol)
2. **REST** (Representational state transfer) web service

**REST** stands for “Representational State Transfer” and is the set of rules that developers follow when creating an API. **Rest APIs** that relies on the **HTTP protocol** and **JSON** data format to send and receive messages

**SOAP** or Simple Object Access Protocol is another design modal for web services. SOAP uses a language known as **Extensible Markup Language (XML)**. XML is designed to be **machine- and human-readable**. SOAP follows a strong standard of rules, such as **messaging structure** and **convention** for providing request or responses.

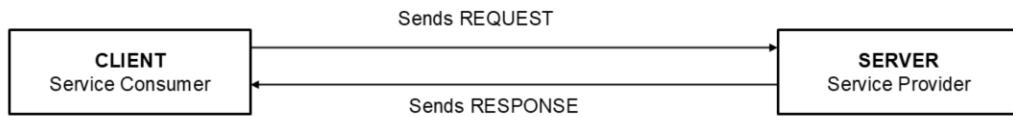
## What is WSDL and UDDI?

Consumer needs to know:

- What the web service is?
- What are the various functionalities?
- What are the parameters and return types?
- How to call a web service?

Service provider publishes an interface for its web service that describes all the attributes of the web services.

This is XML based interface, and it is called –  
**Web Services Description Language (WSDL)**



12

The Web Services Description Language (WSDL) is an XML-based interface used to describe the functionalities of web services.

**Why XML?**

XML is chosen for WSDL because:

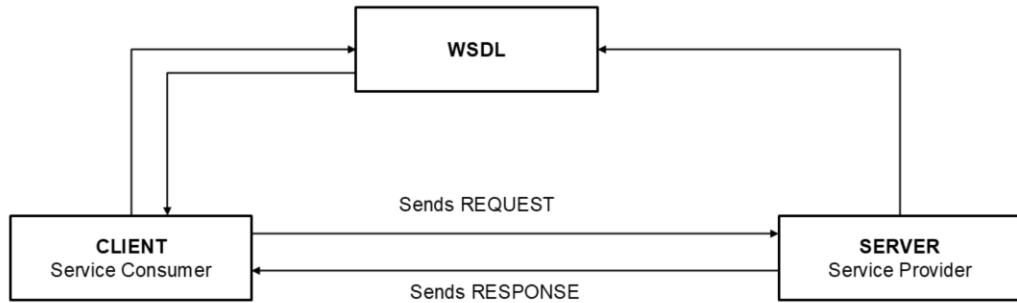
- It is **machine-readable**, allowing systems to interpret and process it efficiently.
- It can be easily **parsed** using various tools and programming languages.
- It provides a **structured format**, ensuring consistency in data exchange.

This structured nature makes XML an ideal choice for defining web service contracts in a standardized and platform-independent way.

**Example:**

<http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wsdl>

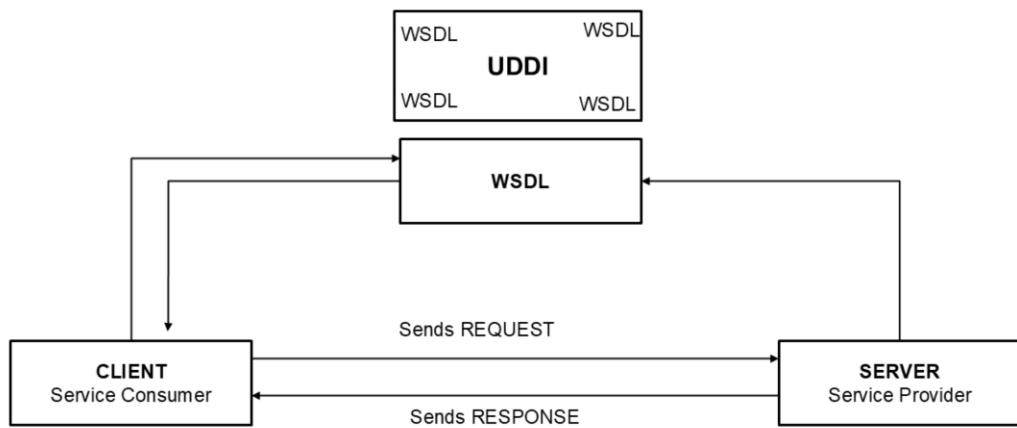
## What is WSDL and UDDI?



13

- Service provider creates the interface (WSDL) for its web service
- Service Consumer get the this WSDL
- Service Consumer use those web services and all the requests of the web services

## What is WSDL and UDDI?



14

What happens when the Service Consumer and Service Provider do not know each other?

In cases where the Service Consumer and Service Provider are not directly connected or aware of each other, the consumer needs a way to discover the service and obtain its WSDL document.

How does the Service Consumer get the WSDL document?

To facilitate this, the Service Provider publishes its web service (along with the WSDL) in an online registry or directory, where consumers can search, query, and discover available web services.

This online registry is called UDDI (Universal Description, Discovery, and Integration).

What is UDDI?

UDDI is a centralized directory that helps businesses and applications find and interact with web services by providing details such as:

- Service descriptions (WSDL)
- Endpoints (where the service can be accessed)
- Technical information for integration

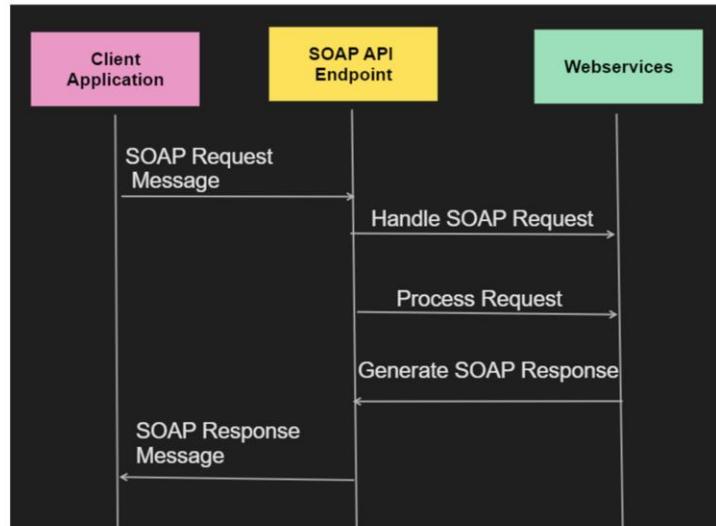
By using UDDI, a Service Consumer can dynamically locate a Service Provider, retrieve its WSDL, and interact with the web service without prior knowledge of its existence.

## What are SOAP Web Services?

- **SOAP (Simple Object Access Protocol)** is a messaging protocol used for exchanging structured data between web services over the internet.
- It's based on the **XML** language and defines a standard format for messages and their contents, including headers, bodies, and optional attachments.



## How does SOAP works?



16

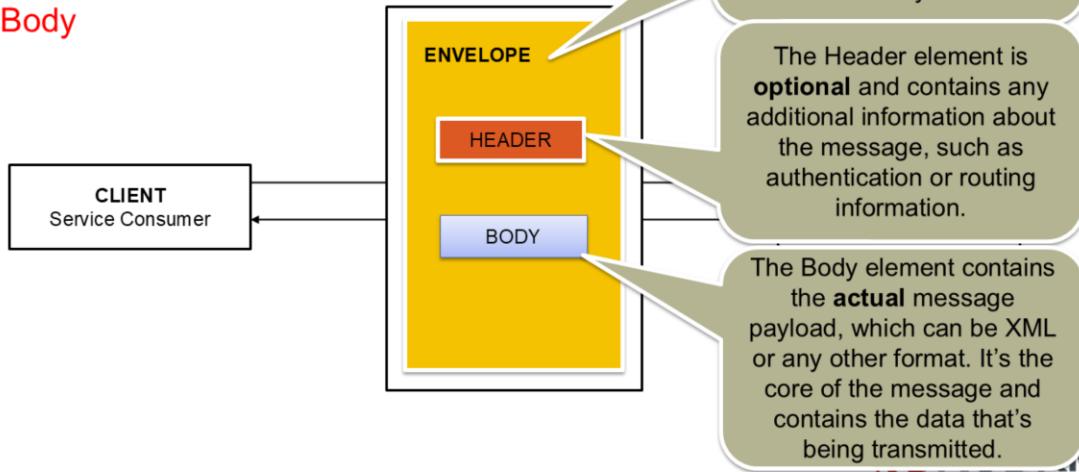
**SOAP messages** are typically sent over **HTTP** (Hypertext Transfer Protocol), but other transport protocols such as **SMTP** (Simple Mail Transfer Protocol) and **TCP** (Transmission Control Protocol) can also be used.

The protocol is based on a request-response model, where the client sends a **request** message to the server and waits for a **response** message in return.

**SOAP messages** are typically used for integrating web services in enterprise applications and other large-scale software systems. They provide a standardized and reliable way to exchange data between software components, ensuring that messages are properly formatted and processed by **SOAP-compatible applications**.

## SOAP Message

- The SOAP Message typically consists of:
  - Envelope
  - Header
  - Body



17

The SOAP envelope structure is designed to ensure that messages are properly formatted and processed by SOAP-compatible applications. This is achieved using well-defined **XML** schema that specifies the structure and format of the message, as well as the rules for processing it. This makes it possible for different web services to communicate with each other seamlessly, regardless of the programming language or platform they're using.

## SOAP Example: Request and Response

```

POST http://www.webservicex.com/globalweather.asmx HTTP/1.1
Host: www.webservicex.com
Content-Length: 341
Content-Type: application/soap+xml

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header></soap:Header>
  <soap:Body>
    <ns1:GetWeather xmlns:ns1='http://www.webserviceX.NET'>
      <ns1:CityName>Montreal</ns1:CityName>
      <ns1:CountryName>Canada</ns1:CountryName>
    </ns1:GetWeather>
  </soap:Body>
</soap:Envelope>
```

```

HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Type: application/soap+xml; charset=utf-8
Server: Microsoft-IIS/7.0
Date: Wed, 28 Oct 2015 13:36:31 GMT
Content-Length: 802

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <GetWeatherResponse xmlns="http://www.webserviceX.NET">
      <GetWeatherResult><?xml version="1.0" encoding="utf-16"?>
        &lt;CurrentWeather&gt; &lt;Location&gt;Montreal-Est, Canada
        (CWPQ) 45-38N 070-33W&lt;/Location&gt; &lt;Time&gt;Mar 26, 2008 -
        12:00 PM EST / 2008.03.26 1700 UTC&lt;/Time&gt; &lt;Wind&gt;
        from the W (260 degrees) at 16 MPH (14 KT) gusting to 23 MPH (20
        KT):&lt;/Wind&gt; &lt;Temperature&gt; 39 F (4
        C)&lt;/Temperature&gt;
        &lt;Status&gt;Success&lt;/Status&gt;
        &lt;CurrentWeather&gt;
      </GetWeatherResult>
    </GetWeatherResponse>
  </soap:Body>
</soap:Envelope>
```

18

The soap:Body element contains a **GetWeather** element with **CityName** and **CountryName** child elements, which hold the values "**Montreal**" and "**Canada**," respectively. This forms the actual payload of the message, indicating that the client is requesting the **weather** record for Montreal, Canada.

## SOAP Key Components - Service Endpoints

- SOAP APIs expose **services**.
- Each service has a specific endpoint (URL) where clients send requests.
- Example: This endpoint provides operations like **GetOrder**, **CreateOrder**, **UpdateOrder**:

`https://api.example.com/OrderService`

## SOAP Key Components - SOAP Methods (Operations)

- SOAP APIs use operations defined in a **WSDL (Web Services Description Language)** file.
- Example SOAP Operations:
  - GetOrderDetails → Fetch order information.
  - CreateOrder → Place a new order.
  - CancelOrder → Cancel an existing order.
- Each operation is wrapped in a SOAP envelope.

## SOAP Key Components - Representations (XML-Based Messaging)

- SOAP messages must be in **XML** format.
- The message structure consists of **Envelope**, **Header**, **Body**, and **Fault** (for errors).
- Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
                  xmlns:ord="http://example.com/orders">  
    <soapenv:Header/>  
    <soapenv:Body>  
        <ord:GetOrderDetails>  
            <ord:OrderID>12345</ord:OrderID>  
        </ord:GetOrderDetails>  
    </soapenv:Body>  
</soapenv:Envelope>
```

## SOAP Key Components - WSDL (Web Services Description Language)

- SOAP APIs use WSDL files to describe available operations and data formats.
- Clients use WSDL to understand how to interact with the API. This WSDL file defines the **structure** of SOAP **requests** and **responses**.

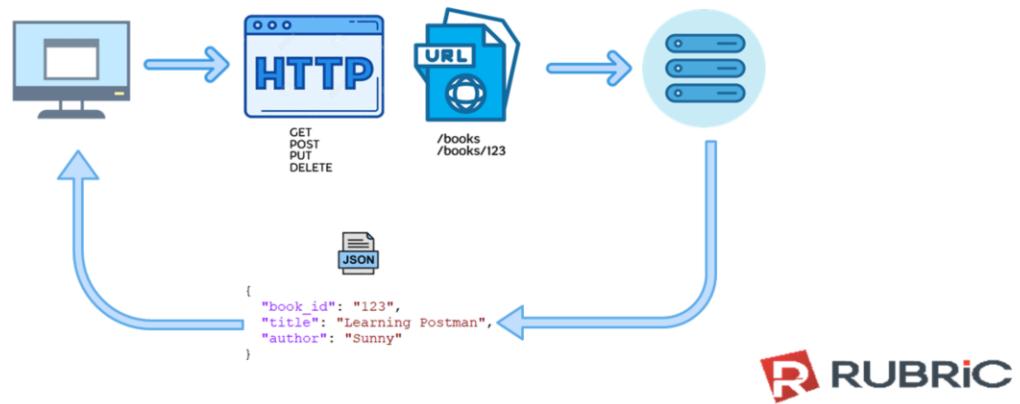
## SOAP Key Components - Status Codes (SOAP Faults)

- SOAP returns **SOAP Faults** inside an **XML response** when **errors** occur.
- Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>soapenv:Client</faultcode>
            <faultstring>Invalid Order ID</faultstring>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

## REST API(Representational State Transfer)

- REST API architecture refers to the design and structure of a web service that follows the principles of REST (Representational State Transfer).
- Here's an overview of the components and considerations involved in REST API architecture



## REST Key Components - Resources

- Resources are the main entities that an API exposes, like books, users, products, or orders.
- Each resource has a unique URI (Uniform Resource Identifier).
- Example: A book can be retrieved with a unique ID “123”:

<https://api.example.com/books/123>

## REST Key Components - HTTP Methods (CRUD Operations)

- REST APIs use HTTP methods to perform actions on resources:
  - GET → Retrieve data
  - POST → Create new data
  - PUT → Update existing data
  - DELETE → Remove data
- Example:
  - Fetch a user's details (GET):  
`GET https://api.example.com/books/123`
  - Create a new user (POST):  
`POST https://api.example.com/books`

## REST Key Components - Representations

- Resources can be represented in different formats, like **JSON** or **XML**.
- **JSON** is the most commonly used format in REST APIs.

```
{  
  "book_id": "123",  
  "title": "Learning Postman",  
  "author": "Sunny"  
}
```

## REST Key Components - Status Codes

- APIs return HTTP status codes to indicate the success or failure of a request.
- They are **three-digit numbers** that indicate the **outcome** of an HTTP request. They are grouped into five categories:
  - Informational responses (100–199)
  - Successful responses (200–299)
  - Redirects (300–399)
  - Client errors (400–499)
  - Server errors (500–599)

28

### 1xx - Informational

- **100 Continue** – When you upload a large file to a cloud service, the server may respond with 100 Continue to indicate that it is ready to receive the full file.

### 2xx - Success

- **200 OK** – When you successfully log into your email, the server responds with 200 OK, delivering your inbox.
- **201 Created** – When you sign up for a new account on an e-commerce website, the server creates a user profile and responds with 201 Created.
- **204 No Content** – When you delete an email from your inbox and the request is processed successfully, but the server does not return a message body, you get a 204 No Content response.

### 3xx - Redirection

- **301 Moved Permanently** – When you visit an old website URL (e.g., <http://example.com/old-page>), and it redirects to <http://example.com/new-page> because the page was permanently moved.
- **302 Found** – If you enter <http://www.example.com>, the server might temporarily redirect you to <https://www.example.com>, using 302 Found.
- **304 Not Modified** – When you revisit a website, your browser checks if the cached version is still valid. If it is, the server responds with 304 Not Modified, telling your browser to use the cached content instead of re-downloading it.

### 4xx - Client Errors

- **400 Bad Request** – If you try to enter an invalid search query (e.g., [example.com/search?query=%%](http://example.com/search?query=%%)), the server might return 400 Bad Request because the syntax is incorrect.
- **401 Unauthorized** – If you try to access a banking portal without logging in, the server responds with 401 Unauthorized.
- **403 Forbidden** – If you try to access a restricted company database without the right permissions, you get 403 Forbidden.
- **404 Not Found** – If you visit a broken link or mistype a URL (e.g., [example.com/nonexistent-page](http://example.com/nonexistent-page)), you get 404 Not Found.
- **405 Method Not Allowed** – If a website allows only GET requests but you try to send a POST request, you get 405 Method Not Allowed.
- **409 Conflict** – If two users try to edit the same document at the same time, the server might return 409 Conflict, preventing data overwrites.

### 5xx - Server Errors

- **500 Internal Server Error** – If an online shopping website crashes due to a bug in the checkout process, you might see 500 Internal Server Error.
- **502 Bad Gateway** – If an API gateway (e.g., a payment processor) tries to connect to a third-party service, but the response is invalid, it returns 502 Bad Gateway.
- **503 Service Unavailable** – If an online streaming service (like Netflix) is undergoing maintenance, trying to access it may result in 503 Service Unavailable.
- **504 Gateway Timeout** – If you try to load a webpage and the request takes too long because the backend server is slow, you might see 504 Gateway Timeout.

## REST Key Components - Hypermedia Links

- Responses can include links to related actions, making APIs self-explanatory and navigable.
- Example:
  - This allows the client to **dynamically discover** actions it can perform.

```
{  
    "book_id": "123",  
    "title": "Learning Postman",  
    "author": "Sunny",  
    "links": {  
        "self":"/books/123",  
        "edit":"/books/123/edit",  
        "delete":"/books/123/delete"  
    }  
}
```

## REST API Principles

- These are the six principles REST APIs follow:
  - Client-server architecture
  - Layered System
  - Uniform interface
  - Stateless
  - Cacheable
  - Code on demand

## Principle 1: Client-Server Architecture

- The client (user) and the server (system) are independent of each other.
- The client sends a request, and the server responds, but they don't need to be on the same platform or use the same technology.
- Example:
  - You open a mobile banking app (client) on your phone, which connects to a bank's server.
  - The app could be built using Swift (for iPhone) or Kotlin (for Android), while the bank's server could be running Java or Python. They still communicate smoothly.

## Principle 2: Layered System

- The server can have multiple layers (e.g., security, caching, authentication), but the client doesn't know about them.
- Each layer has a specific role, making the system more scalable and secure.
- Example:
  - When you order food from a delivery app, the app contacts a restaurant server.
  - The restaurant's server might use other servers for payment processing, order tracking, and customer support.
  - You, as the client, don't see these layers—only the final result (your order status).

## Principle 3: Uniform Interface

- The API sends responses in a consistent format, usually JSON or XML.
- This makes it easy for different clients (web apps, mobile apps, other servers) to use the data.
- Example:

- When you check the weather on your phone, the API response might be in JSON.

```
{  
    "temperature": "20°C",  
    "humidity": "60%",  
    "condition": "Sunny"  
}
```

- Whether you're using a web browser, mobile app, or smartwatch, they can all understand and display this data because it's in a uniform format.

## Principle 4: Stateless

- Each API request is independent; the server doesn't remember past requests.
- This makes APIs scalable since they don't need to store session data.
- Example:
  - If you check your bank balance now and then check it again later, the API treats both requests separately.
  - It doesn't remember that you asked for your balance five minutes ago.

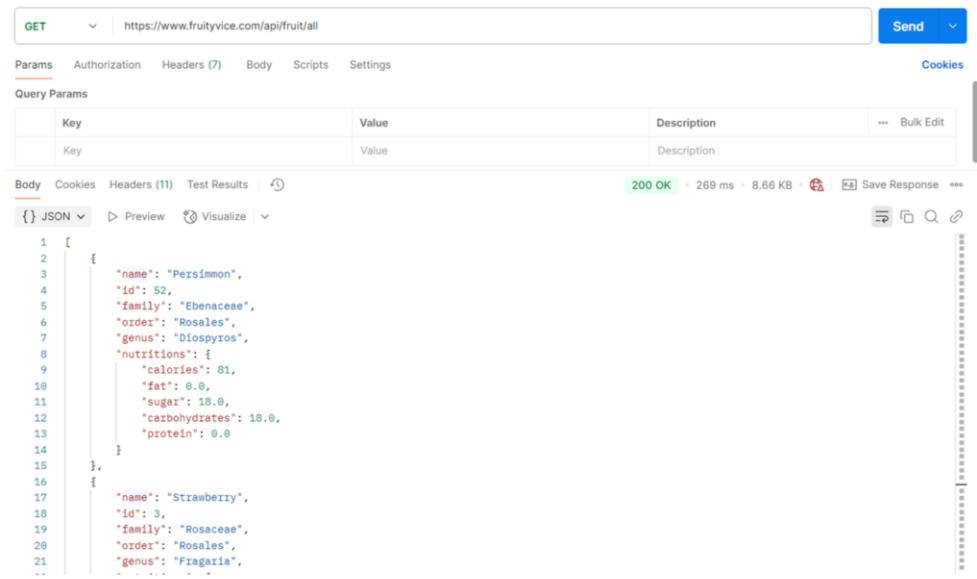
## Principle 5: Cacheable

- Some API responses can be stored (cached) so they don't have to be fetched again.
- This improves speed and reduces server load.
- Example:
  - When you search for a movie on a streaming app, the app might save (cache) the movie details for a while.
  - If you search for the same movie again, it loads instantly instead of asking the server again.

## Principle 6: Code on demand (Optional feature)

- This means the server can send executable code (like JavaScript) to the client, allowing it to extend its functionality dynamically.
- It's the only **optional** REST principle, as not all APIs need to do this.
- Example:
  - Web Applications:
    - When you visit a website, the server might send a JavaScript file along with the API response.
    - Your browser executes this JavaScript, enabling interactive elements like pop-ups or dynamic forms.
  - Google reCAPTCHA:
    - When you submit a form, sometimes the server sends a JavaScript-based CAPTCHA challenge to check if you're human.
    - This JavaScript is "on demand" and runs only when necessary.

## REST Example



The screenshot shows a REST API testing interface. At the top, there's a header bar with 'GET' selected, the URL 'https://www.fruityvice.com/api/fruit/all', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Scripts', and 'Settings'. A 'Cookies' tab is also present. Under 'Query Params', there's a table with columns 'Key', 'Value', 'Description', and 'Bulk Edit'. The 'Body' tab is active, showing the JSON response from the API. The response is a list of fruit objects:

```
1 [  
2   {  
3     "name": "Persimmon",  
4     "id": 52,  
5     "family": "Ebenaceae",  
6     "order": "Rosales",  
7     "genus": "Diospyros",  
8     "nutritions": {  
9       "calories": 81,  
10      "fat": 0.0,  
11      "sugar": 18.0,  
12      "carbohydrates": 18.0,  
13      "protein": 0.0  
14    }  
15  },  
16  {  
17    "name": "Strawberry",  
18    "id": 3,  
19    "family": "Rosaceae",  
20    "order": "Rosales",  
21    "genus": "Fragaria",  
22  }]
```

37

## Summary of differences between SOAP vs REST

	<b>SOAP</b>	<b>REST</b>
<b>Stands for</b>	Simple Object Access Protocol	Representational State Transfer
<b>What is it?</b>	SOAP is a protocol for communication between applications	REST is an architecture style for designing communication interfaces.
<b>Design</b>	SOAP API exposes the operation.	REST API exposes the data.
<b>Transport Protocol</b>	SOAP is independent and can work with any transport protocol.	REST works only with HTTPS.
<b>Data format</b>	SOAP supports only XML data exchange.	REST supports XML, JSON, plain text, HTML.
<b>Performance</b>	SOAP messages are larger, which makes communication slower.	REST has faster performance due to smaller messages and caching support.
<b>Scalability</b>	SOAP is difficult to scale. The server maintains state by storing all previous messages exchanged with a client.	REST is easy to scale. It's stateless, so every message is processed independently of previous messages.
<b>Security</b>	SOAP supports encryption with additional overheads.	REST supports encryption without affecting performance.
<b>Use case</b>	SOAP is useful in legacy applications and private APIs.	REST is useful in modern applications and public APIs.

38

### Design

The SOAP API exposes functions or operations, while REST APIs are data-driven. For example, consider an application with employee data that other applications can manipulate. The application's SOAP API could expose a function called CreateEmployee. To create an employee, you would specify the function name in your SOAP message when sending a request.

However, the application's REST API could expose a URL called /employees, and a POST request to that URL would create a new employee record.

### Flexibility

SOAP APIs are rigid and only allow XML messaging between applications. The application server also has to maintain the state of each client. This means it has to remember all previous requests when processing a new request.

REST is more flexible and allows applications to transfer data as plain text, HTML, XML, and JSON. REST is also stateless, so the REST API treats every new request independently of previous requests.

### Performance

SOAP messages are larger and more complex, which makes them slower to transmit and process. This can increase page load times.

REST is faster and more efficient than SOAP due to the smaller message sizes of REST. REST responses are also cacheable, so the server can store frequently accessed data in a cache for even shorter page load times.

### Scalability

The SOAP protocol requires applications to store the state between requests, which increases bandwidth and memory requirements. As a result, it makes applications expensive and challenging to scale.

Unlike SOAP, REST permits stateless and layered architecture, which makes it more scalable. For example, the application server can pass the request to other servers or allow an intermediary (like a content delivery network) to handle it.

### Security

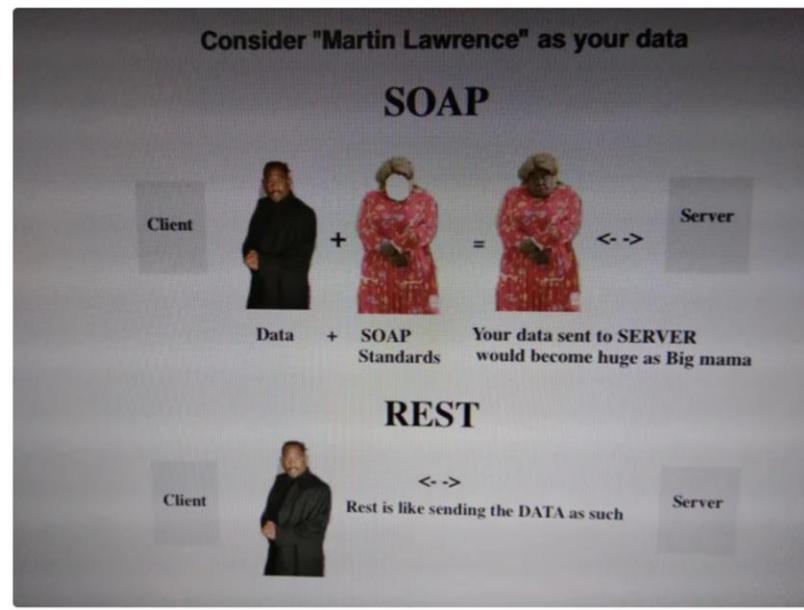
SOAP requires an additional layer of WS-Security to work with HTTPS. WS-Security uses additional header content to ensure only the designated process in the specified server reads the SOAP message content. This adds communication overheads and negatively impacts performance.

REST supports HTTPS without additional overheads.

### Reliability

SOAP has error handling logic built into it, and it provides more reliability. On the other hand, REST requires you to try again in case of communication failures, and it's less reliable.

## SOAP vs REST



39



POSTMAN

API Testing

40

## API Testing

- API testing test whether the API return the **correct response or output** under varying conditions whenever a request is made.
- API testing allows the user to test **headless technologies** like **JMS, HTTP Databases, and Web Services**.
- It **bypasses** the **GUI** and communicate directly with the application by making calls to its **API**.

## How to Test APIs?



Send calls/commands to the API



Wait for a response



Review the log of the system's response to the call

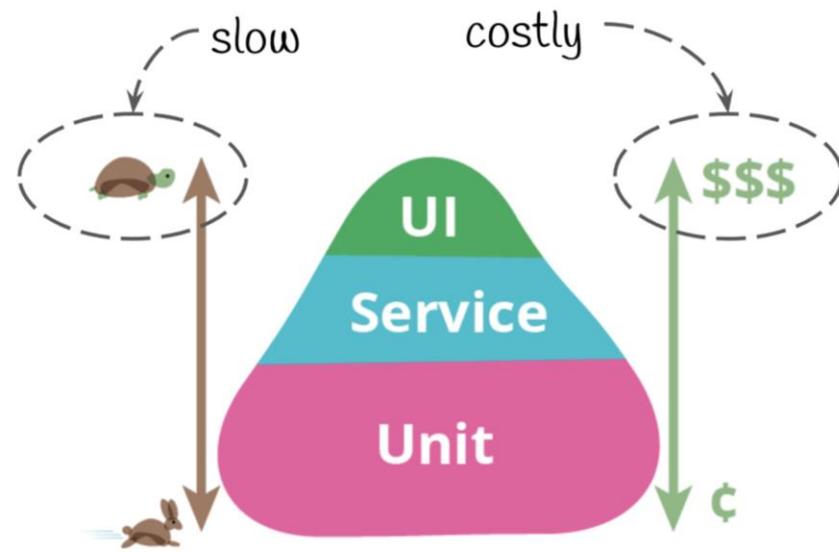


Verify that the output matches the expected output values

The test cases of API testing are based on:

1. Return value based on input condition
  - It is relatively easy to test, as input can be defined, and results can be authenticated.
2. Does not return anything
  - When there is no return value, behaviour of API on the system needs to be checked.
3. Trigger some other API/event/interrupt
  - If output of an API triggers some event or interrupt, then those events and interrupt listeners should be tracked.
4. Update data structure
  - Updating data structure will have some outcome or effect on the system, and that should be authenticated.
5. Modify certain resources
  - If API call modifies some resources, then it should be validated by accessing respective resources.

## Why API testing is important?



43

In an Agile environment, the way software and automated tests are being developed has changed dramatically.

GUI tests tend to take a long time to run. For certain Agile practices like continuous builds, when new code is checked in, the amount of time it takes to receive feedback from a GUI regression suite of tests is unacceptable. In those cases, quicker feedback is needed. The sooner bugs are found the better, since a developer instantly knows the code changes they made have broken the build and need to be looked at. In test driven processes, users need a large percentage of test sets to run fast and frequently and must be able to integrate them into the development lifecycle. Since API tests bypass the user interface, they tend to be quicker and much more reliable than GUI tests. And most importantly -- since API tests don't rely on a UI to be ready, they can be created early in the development cycle.

### GUI Test

GUI testing focuses on testing an application user interface to ensure that its functionality is correct. IDE GUI testing is at the top of the pyramid and represents a small piece of the total number of automation test types that should be created.

### Unit Test

Unit testing makes up the largest section of the pyramid, forming a solid base. A unit test is created to verify a single unit of source code, like a method. By doing this, developers can isolate the smallest testable parts of their code. Unit tests are the easiest to create and yield the biggest "bang for the buck." Since unit tests are usually written in the same language that an application is written in, developers should have an easy time adding them to their development process.

### Service - API Test

The middle Service layer is the "sweet spot". Service testing is also known as integration testing. Integration testing focuses on verifying that the interactions of many small components can integrate without issue.

## Types of API Tests



Functionality testing



Reliability testing



Load testing



Security testing



Negative Testing

44

- 1. Functionality testing** — the API works and does exactly what it's supposed to do.
- 2. Reliability testing** — the API can be consistently connected to and lead to consistent results
- 3. Load testing** — the API can handle a large amount of calls
- 4. Security testing** — the API has defined security requirements including authentication, permissions and access controls
- 5. Negative Testing** — checking for every kind of wrong input the user can possibly supply

## Some Tools for API Testing



REST-assured

## Exercise:

- **User Story:**

- As a pet store owner,
  - I want to add new pets to the system
    - so that they are available for customers to view and purchase.

- **Acceptance Criteria:**

- Ensure a pet can be added only when all required details (**ID, name, category, status**) are provided.
- Ensure customers can view the newly added pet.
- Ensure that error message "**Pet ID is required**" is returned, when required field ID is not filled in.
- Ensure that error message "**Pet name is required**" is returned, when required field name is not filled in.
- Ensure that error message "**Pet category is required**" is returned, when required field category is not filled in.
- Ensure that error message "**Pet status is required**" is returned, when required field status is not filled in.

46

<https://petstore.swagger.io/>

## Answer:

### 1. Valid Scenario:

#### 1. Successfully add a pet with valid details

- Send a request with a **valid pet ID, name, category, and status**.
- Expected result:** API should return **200 OK**, and the pet should be retrievable using **GET /pet/{petId}**.

#### 2. Retrieve an added pet using **GET** request

- Send a request with a valid pet ID using **GET /pet/{petId}**.
- Expected result:** API should return **200 OK** with the **correct** pet details.

### 2. Negative Scenarios:

#### 1. Missing required field name

- Send a request with a missing pet name in the request body.
- Expected result:** API should return **400 Bad Request** with an error message "**Pet name is required**".

#### 2. Missing required field category

- Send a request with missing category in the request body.
- Expected result:** API should return **400 Bad Request** with an error message "**Pet category is required**".

## Exercise:

- **User Story:**
  - As a banking app customer,
  - I want to check my account balance through the bank's API
  - so that I can keep track of my available funds.
- **Acceptance criteria:**
  - Ensure that the **correct balance** is displayed when a **valid account number** is provided.
  - Ensure that account verification is **not possible** when an **invalid account number** is provided.
  - Ensure that a user who is **not logged in cannot check** the account balance.
  - Ensure that balance retrieval takes **less than 2 seconds** under normal load.

48

**Endpoint:**

GET /account/balance

**Request Headers:**

GET /account/balance?accountId=123456789

Authorization: Bearer abcdef123456

## Answer:

1.  **Valid Scenario:**
  - Send a request with a **valid accountId** and a **correct Authorization token**
  - **Expected result:** API should return the correct balance.
2.  **Invalid Account ID:**
  - Send a request with an **invalid or missing accountId**
  - **Expected result:** API should return a **400 Bad Request** error.
3.  **Unauthorized Access:**
  - Send a request **without an Authorization token**.
  - **Expected result:** API should return a **401 Unauthorized** error.
4.  **Performance Test:**
  - Measure API response time under **normal and heavy load**.
  - **Expected result:** API should respond in **less than 2 seconds**.



## API Testing using Postman

50

## Course Overview



**RUBRIC**

- Postman Introduction
- Postman Requests
- Postman Collections: Import, Export And Generate Code Samples
- Variable Scopes And Environment Files
- Pre-Request And Post Request Scripts
- Automating Response Validations with Assertions
- Postman Advanced Scripting
- Postman – Command-Line integration with Newman
- Postman – Reporting templates with Newman

## Introduction to Postman



- In this chapter, you will learn about:
  - API Testing using Postman
  - Basics of Postman
  - Components of Postman



**RUBRIC**

## What is Postman?

- Postman is a standalone software testing **API** (Application Programming Interface) platform to build, test, design, modify, and document APIs.
- This tool has the ability to make various types of **HTTP** requests like **GET**, **POST**, **PUT**, **PATCH**, and convert the API to code for languages like **JavaScript** and **Python**.
- While using Postman, for testing purposes, one doesn't need to write any HTTP client network code. Instead, we build test suites called collections and let Postman interact with the API.

## Terminologies Related to Postman

- **API**
  - Application Programming Interface (API) is software that acts as an intermediary for two apps to communicate with each other.
- **HTTP**
  - HTTP (Hypertext Transfer Protocol) is the collection of rules for the transmission of data on the **World Wide Web**, like **graphic images, text, video, sound**, and other multimedia data.
  - The Web users implicitly make use of HTTP as soon as they open their Web browser.

## When Postman can be used?

- It is used for backend testing where we enter the end-point URL, it sends the request to the server and receives the response back from the server.

## Why Use Postman?

- **Postman offers a lot of advanced features like:**
  - Accessibility
  - Use of Collections
  - Test development
  - Automation Testing
  - Creating Environments
  - Debugging
  - Collaboration
  - Continuous integration

56

- 1. Accessibility-** One can use it anywhere after installing Postman into the device by simply logging in to the account.
- 2. Use Collections-** Postman allows users to build collections for their API-calls. Every set can create multiple requests and subfolders. It will help to organize the test suites.
- 3. Test development-** To test checkpoints, verification of successful HTTP response status shall be added to every API- calls.
- 4. Automation Testing-** Tests can be performed in several repetitions or iterations by using the Collection Runner or Newman, which saves time for repeated tests.
- 5. Creating Environments-** The design of multiple environments results in less replication of tests as one can use the same collection but for a different setting.
- 6. Debugging-** To effectively debug the tests, the postman console helps to track what data is being retrieved.
- 7. Collaboration-** Collections and environments can be imported and exported to enhance the sharing of files. You may also use a direct connection to share the collections.
- 8. Continuous integration-** It can support continuous integration.

## Installation

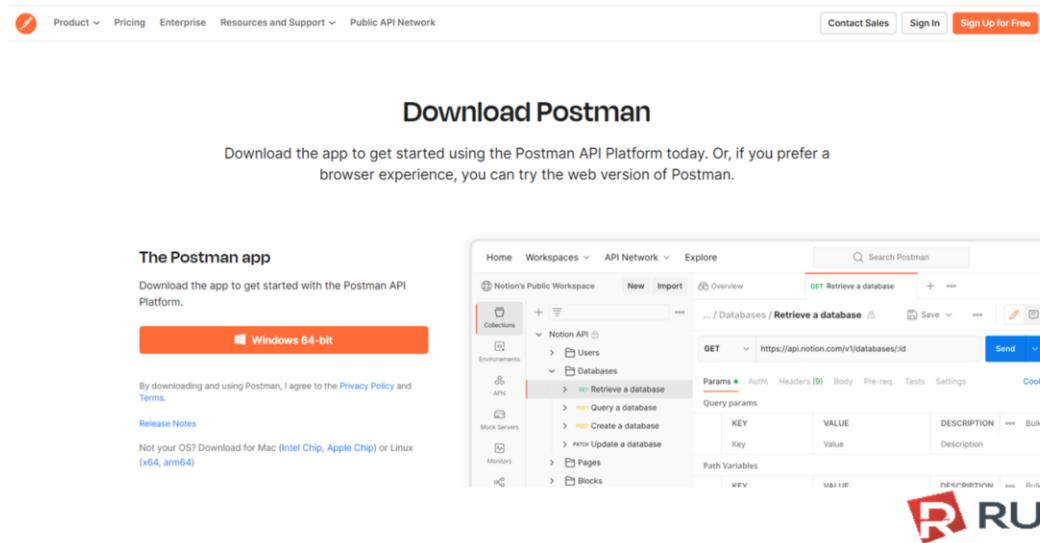
- Postman is available in 2 options:
  - As a **Chrome extension application** (this is already deprecated and has no support from the Postman developers)
  - **Native App** (standalone application) for different platforms like Windows, Mac OS, Linux, etc.

57

**NOTE:** However, it is better to install and use the native app because Postman chrome extension does not support all the features that the native app has.

## How to Download POSTMAN (1)

- Being an open-source tool, Postman can be easily downloaded from <https://www.postman.com/downloads/>.



The screenshot shows the Postman download page and the Postman application interface side-by-side.

**Postman Download Page:**

- The top navigation bar includes links for Product, Pricing, Enterprise, Resources and Support, and Public API Network.
- On the right, there are buttons for Contact Sales, Sign In, and Sign Up for Free.
- The main heading is "Download Postman".
- A sub-headline says: "Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman."
- A large orange button labeled "Windows 64-bit" is prominently displayed.
- Below the button, there's a note about accepting the Privacy Policy and Terms, and a Release Notes link.
- At the bottom, it says "Not your OS? Download for Mac (Intel Chip, Apple Chip) or Linux (x64, arm64)".

**Postman Application Interface:**

- The interface has a top navigation bar with Home, Workspaces, API Network, and Explore tabs.
- The search bar at the top right contains the placeholder "Search Postman".
- The left sidebar shows sections for Collections, Environments, APIs, Mock Servers, and Monitors. The "Notion API" collection is currently selected, showing items like "Retrieve a database", "Query a database", "Create a database", "Update a database", "Pages", and "Blocks".
- The main workspace shows a "GET Retrieve a database" request. The URL is "https://api.notion.com/v1/databases/{id}".
- The request details panel includes sections for Params, Auth, Headers (9), Body, Pre-req, Tests, Settings, and Cookies.
- Below the request details, there are tables for "Query params" and "Path Variables".

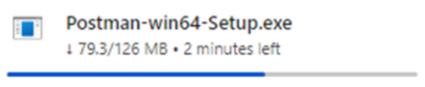
58

 RUBRIC

- Go to <https://www.postman.com/downloads/>
- Click **Download** for Mac or Windows or Linux based on your operating system.

## How to Download POSTMAN (2)

- You can check the download progress download history tab if you are using the Chrome browser.



- Once the .exe file is downloaded, you need to install the application.

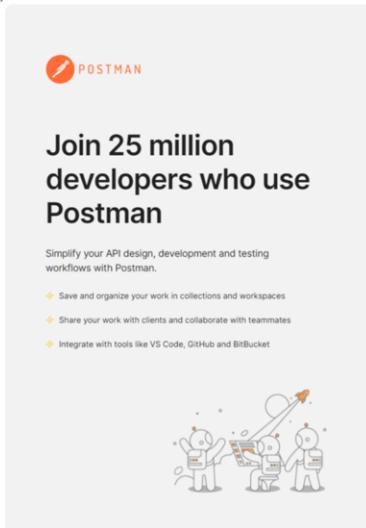
## How to Install POSTMAN (1)

- Click on the .exe file to open it. Installation starts when you see the picture below:



## How to Install POSTMAN (2)

- Once the installation completes, you will be redirected to a window where you need to sign up for a Postman account.



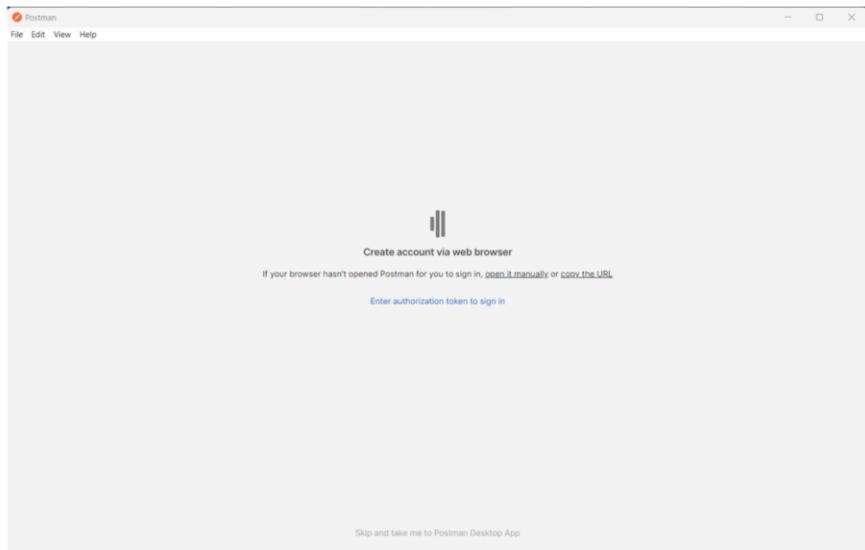
61

When you open the Postman desktop app for the first time, you can create a free Postman account or sign into your Postman account.

**NOTE:** If you choose to skip creating or signing into an account, you will enter the lightweight API Client. You'll also enter the lightweight API Client if you sign out of Postman. Though Postman allows users to use the tool without logging in, signing up ensures that your collection is saved and can be accessed for later use.

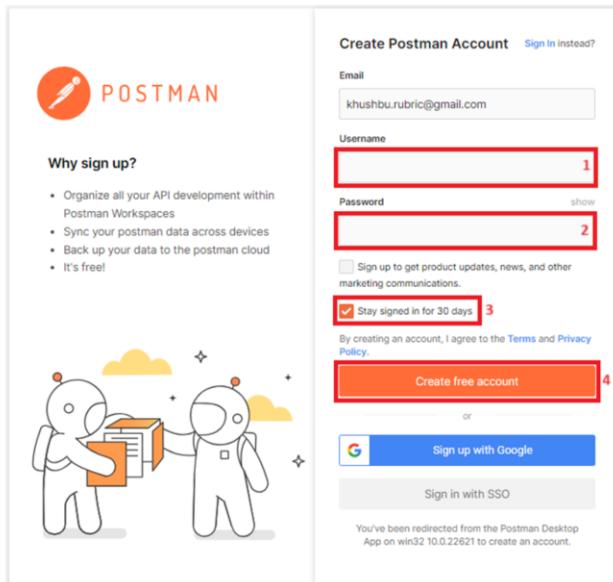
## How to Install POSTMAN (3)

- Upon entering email address, you will get the screen below:



## How to Install POSTMAN (4)

- On your browser, you will see the Postman – Sign Up screen.



63

- Enter **Username**.
- Enter **Password**.
- Make sure the **Stay signed in for 30 days** checkbox is checked.
- Click on **Create free account**.

## How to Install POSTMAN (5)



Welcome to Postman! Tell us a bit about yourself.

Your name

Khushbu Jowaheer

1

What is your role?

Knowing your role will help us offer you a better experience

Analyst

2

Continue

3

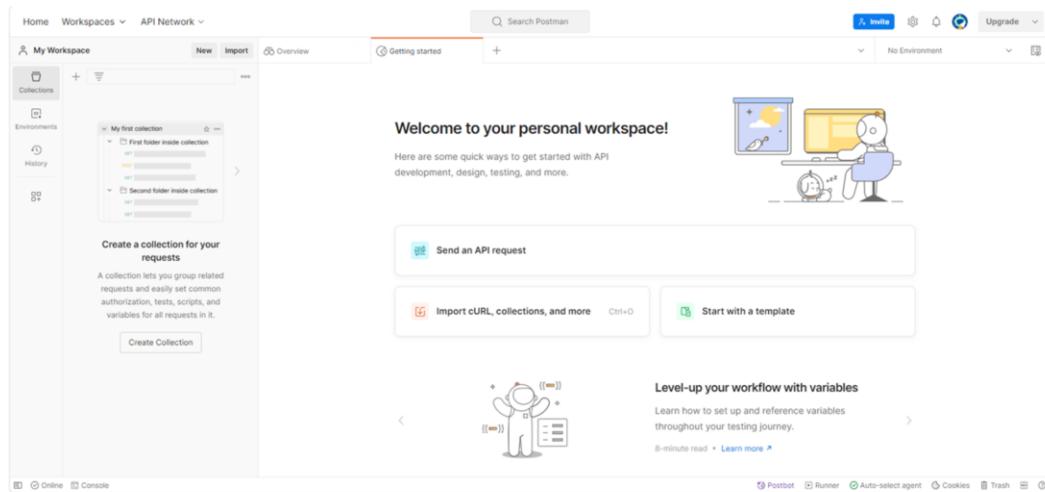


64

1. Enter your **name**.
2. Select **Analyst** as your role.
3. Click **Continue**.

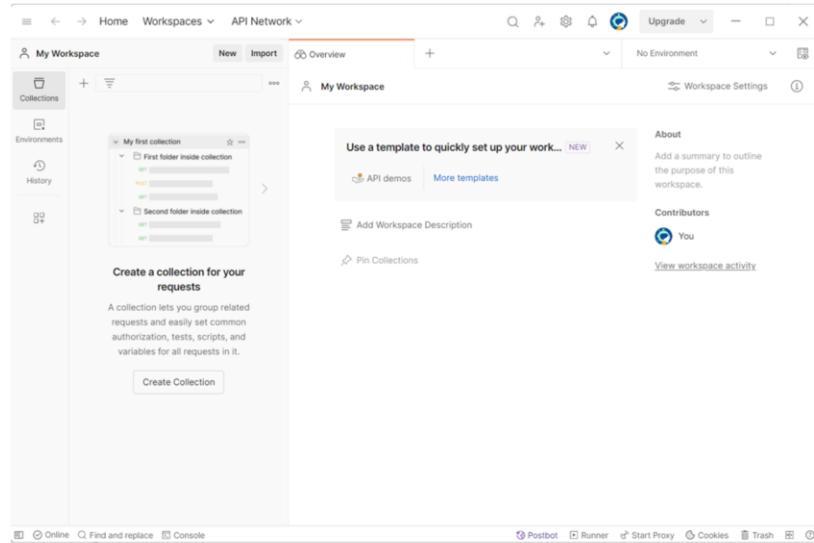
## Postman - Browser Startup Screen

- Once finish, you will see the Startup Screen and then you are ready to use Postman.



## Postman - Standalone application Startup Screen

- After creation of Postman account, open postman and Log in with the credentials you used when setting up postman.



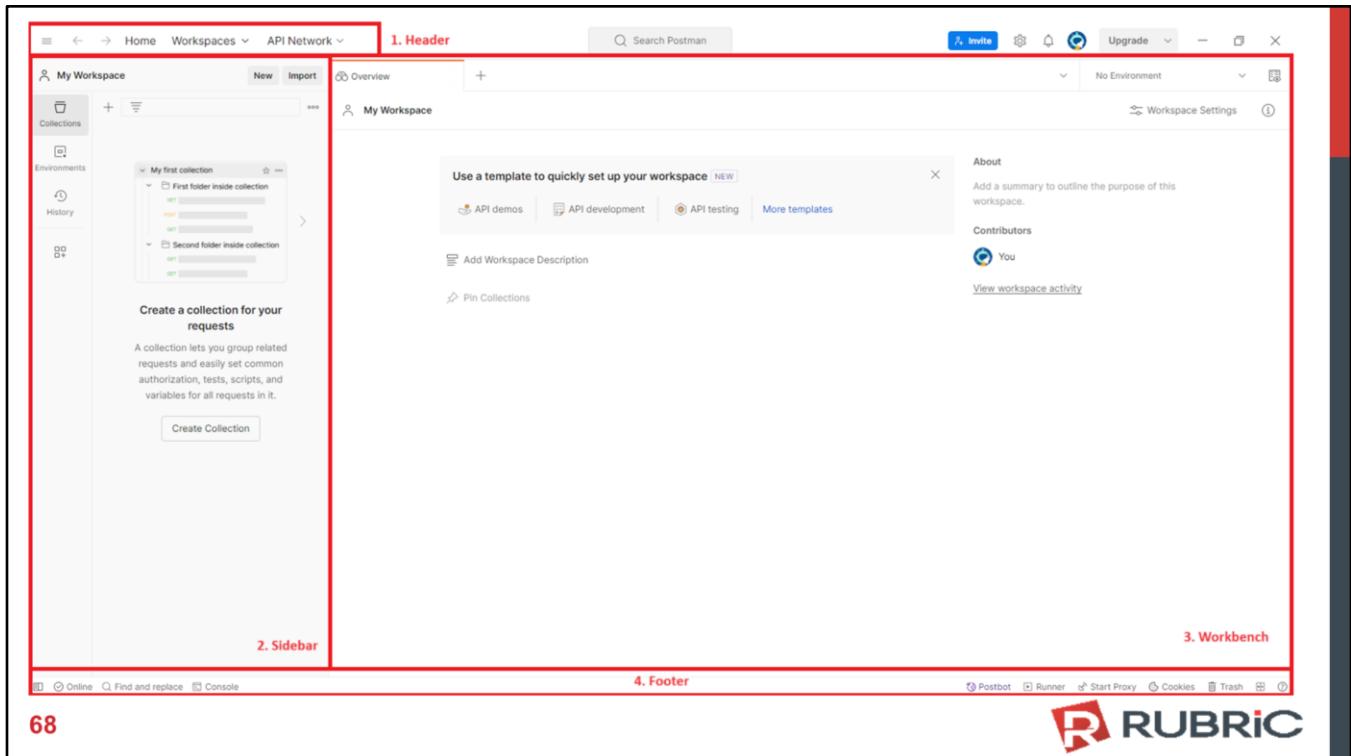
66

 RUBRIC

**NOTE:** When using the Log in option, you are redirected to browser to confirm the email that has been used for the Postman account creation and you will get a pop up to open the Postman standalone application.

## Postman Navigation

- Postman has a variety of tools, views, and controls to help you manage your API projects. This guide is a high-level overview of Postman's primary interface areas:
  - Header
  - Sidebar
  - Workbench
    - Tabs
    - Right sidebar
    - Environment selector and environment quick look
    - Quick Help
  - Footer



The screenshot shows the Postman application interface. A red box labeled "1. Header" highlights the top navigation bar which includes Home, Workspaces, API Network, a search bar, and various user and workspace settings. A red box labeled "2. Sidebar" highlights the left sidebar containing sections for Collections, Environments, History, and a "Create a collection for your requests" section. A red box labeled "3. Workbench" highlights the main workspace area showing an "Overview" card with workspace details like "About", "Contributors", and "View workspace activity". A red box labeled "4. Footer" highlights the bottom footer with links for Online, Find and replace, Console, and various system icons.

## Header

- The header enables you to create workspaces, access reports, explore the public API network, search in Postman, view sync status and notifications, and access your settings, account, and Postman plan.

← → Home Workspaces ▾ API Network ▾

Search Postman

Invite



Upgrade ▾

## Header (1)

← → Home Workspaces ▾ API Network ▾

- ← → - You can navigate backward and forward through pages you've visited within Postman. (*Available on the Postman desktop app*)
- **Home** – You can go to your personal home page, which includes your recently visited workspaces, and links to resources for your team if applicable.
- **Workspaces** - Search for workspaces, view your recently visited workspaces, or create a new workspace.
- **API Network** - Explore the Public API Network and access your team's Private API Network.

## Header (2)

 Search Postman

- **Search Postman** - Search all workspaces, collections, requests, APIs, Flows, and teams in Postman. For more details on searching in Postman, see [Search Postman](#).

71

- To search in Postman, select **Search Postman** in the header then enter your search terms. You can also use the keyboard shortcut **Ctrl+K**. To change the scope of your search, select the scope dropdown list to the left of the search bar. You can search all of Postman, your team, the Private API Network, and the Public API Network. You can also specify the element type you'd like to search for, such as **Workspaces**, **Collections**, or **Teams**.
- You can search by **tag names** that team members have added to collections, APIs, and workspaces (Enterprise plans only). To search by tag names, select **Search Postman** in the header then enter your search using the **tag:tag-name** format.
  - For example, if an API has the tag "production", enter **tag:production** to return the API in your search results.
- The **Search results** page lists the scope and element type. You can change these options to further filter your results.
- You can also sort results using **Sort by** on the **right** by selecting *Most relevant* (default), *Most views*, or *Most recent*.
- Depending on the element type, the search results contain different information:
  - For **workspaces**, the search result shows the workspace type, summary, who published it, and when was it published.
  - For **collections**, the search result shows the workspace type, whether the collection is a fork or not, who published it, and when was it published.
  - For **APIs**, the search result shows the name and summary of the API, the API's owner (either an individual user or a team), and the workspace type.
  - For **teams**, the search result shows the name and summary of the team. Selecting the team name redirects you to the team profile.

## Header (3)

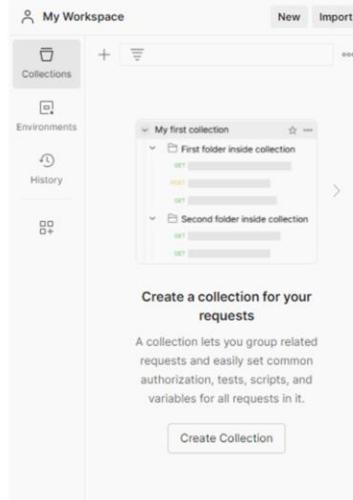


- **Invite** - If you have an **Admin role** on a workspace, you can invite other users to collaborate.
- **⚙️ Settings** - Access Postman settings and other Postman resources.
- **🔔 Notifications** - View recent activity from your team, get notifications about Postman updates, and see pull requests, comment activity, and other important information.
- **Your avatar** - View your profile, access your account and notification settings, see all active sessions for your account, or sign out of your account.
- **Upgrade (free plan) or Team (paid plans)** - View resource usage and access your billing dashboard and other account management tools.

## Sidebar (1)

- The Postman sidebar provides access to the fundamental elements of Postman:

- Collections
- APIs
- Environments
- Mock servers
- Monitors
- Flows
- History



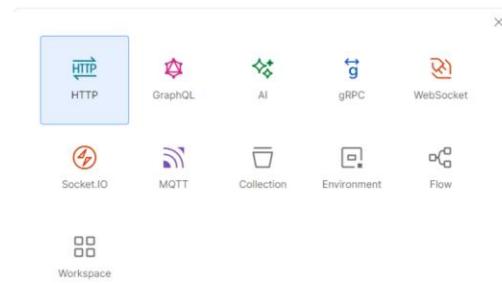
73

To see the task options that are available for elements in the sidebar, select the element's name then select the more options icon More actions icon . The task options will vary depending on the type of element you've selected.

For collections, folders and requests inside a collection, and requests in History, you can select more than one element at a time. Press and hold **Ctrl**, then select the desired elements. For elements that are next to each other, press and hold **Shift**, then select the desired elements. For collections and their contents, you can also use *keyboard shortcuts* for tasks like copying, pasting, and deleting.

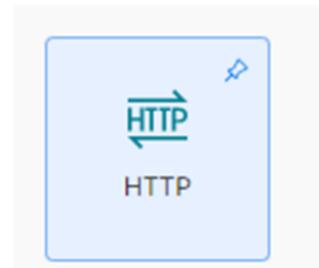
## Sidebar (2)

- You can select **New** in the sidebar to create requests, collections, APIs, and other common Postman elements.



## Sidebar (3)

- You can also pin the elements you use most often:
  - To pin an element, hover over an element and select the pin element icon.
  - To unpin an element, hover over the pinned element and deselect the pin element icon Unpin element icon .



75

## Sidebar (4)

- To hide the sidebar, select the hide sidebar icon  from the footer or right-click in an empty part of the sidebar and select **Collapse sidebar**.
- By default, the sidebar hides empty elements other than **Collections**, **Environments**, and **History**.
- To add elements to sidebar you can click  to open **Workspace settings** where you can specify which sidebar elements to show or hide.
- Right-click on the sidebar for options to show/hide labels, collapse the sidebar, and configure the sidebar.

## Sidebar - History

- To access the requests you've made, select **History** in the sidebar. When you're signed into Postman, your history **syncs** across your devices. The history also includes **collection runs**. These remain as the summarized version of the run and aren't logged as single requests.
- To remove all requests from your history, select the more actions icon next to the History search bar, then select **Clear all**.
- To save request responses in your history, select the more actions icon next to the History search bar, then turn on **Save Responses**.



### NOTE:

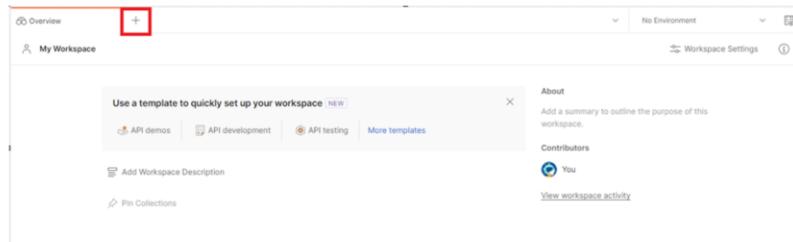
- When you make requests in a **shared workspace**, your request history is visible to you but **not** to other team members in the workspace.
- You can't use the **Save Responses** option with requests from **Collection Runner**.

## Workbench

- Whether you're working with a collection, an API, or another element type, the Postman workbench is where you do majority of your work.
  - **Tabs** enable you to organise your work.
  - The **right sidebar** gives you access to element-specific tools like documentation.
  - The **environment selector and environment quick look** enable you to manage variables.

## Tabs – Opening a new tab

- As mentioned earlier, tabs allow you to organise and work between requests. To open a tab, you can select + in the workbench.



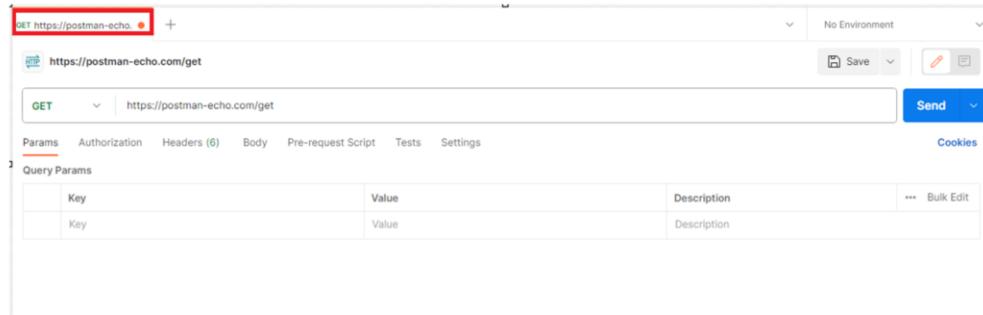
79

### NOTE:

- You can also select **Ctrl+T** to open a new tab.
- If you open a request and don't edit or send it, then open another request, the second tab replaces the first tab. When the tab is in preview mode, it displays in italics.
- You can set whether Postman opens requests and other sidebar items in new tabs. Select the settings icon **Settings** icon in the header and select **Settings**. Under **User interface**, select **Always open sidebar items in new tab** to turn this option on or off.

## Tabs – Saving or discarding changes

- If a tab has unsaved changes, Postman displays a dot next to the tab name. Select **Save** to save the changes.



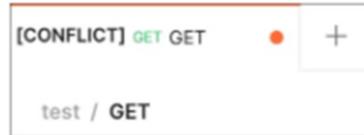
The screenshot shows the Postman application interface. A tab titled "GET https://postman-echo." is open, indicated by a red box around the tab name and a small red dot to its right. The URL "https://postman-echo.com/get" is entered in the address bar. Below the address bar, there are tabs for "Params", "Authorization", "Headers (6)", "Body", "Pre-request Script", "Tests", and "Settings". The "Params" tab is selected. Under "Query Params", there is a table with one row:

Key	Value	Description	...	Bulk Edit
Key	Value	Description	...	

- To close the tab and discard changes, select the close icon **Close** icon then select **Don't save**.

## Viewing conflicts

- A tab will alert you to a conflict if you or a collaborator changes its contents in another tab or workspace. Postman prompts you to resolve any conflicts that happen.



## Tabs – Managing tabs (1)

- You can have many tabs open at the same time. To rearrange your open tabs, select and drag them in the desired order.
- To manage your open tabs, right-click the open tab to activate a menu with action options:
  - **Duplicate Tab** - (Available on the Postman desktop app) Duplicates the current tab. This doesn't create a copy of the request, so when you duplicate a tab any edits you make are reflected in the original request.
  - **Close Tab** - Closes the current tab. If the tab has unsaved changes, Postman asks if you want to save before closing.
  - **Force Close Tab** - Closes the current tab without saving any changes.
  - **Close Other Tabs** - Closes all tabs except the one you're working in.
  - **Close All Tabs** - Closes all tabs. If any tabs contain unsaved changes, Postman will ask if you want to save before closing.
  - **Force Close All Tabs** - Closes all tabs without saving any changes.

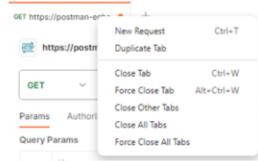
82

### NOTE:

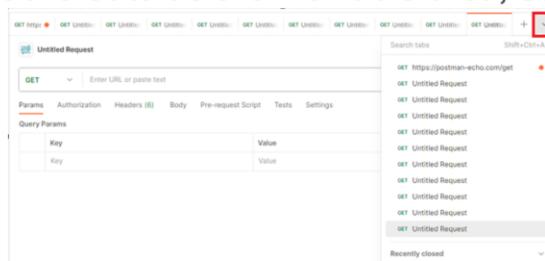
Closing unsaved tabs - You can set whether Postman asks you to save when you close a tab that has changes. Select the **Settings** icon in the header and select **Settings**. Under **General > Request**, select **Always ask when closing unsaved tabs** to turn this option on or off.

## Tabs – Managing tabs (2)

- Tab menu action options:

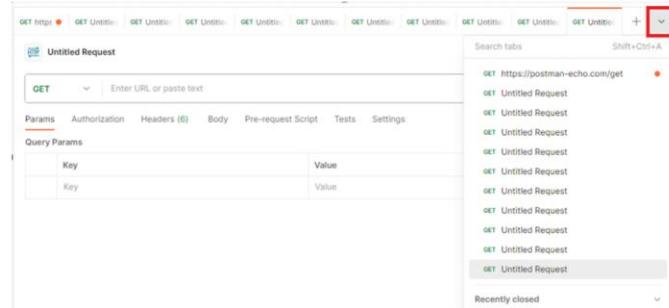


- If you have a lot of tabs open, they might overflow the area of the tab bar. To go to tabs that are outside the viewable area, select the arrows next to the tab bar.



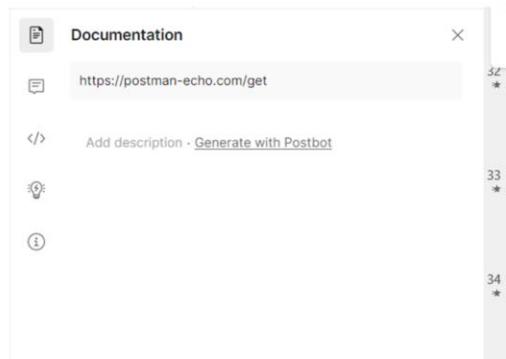
## Tabs – Tab search

- To search open tabs or access recently closed tabs, select the tab search dropdown list.



## Right sidebar (1)

- The right sidebar gives you access to more tools, including documentation, comments, code snippets, and request information, based on which kind of Postman element you select.



85

## Right sidebar (2)

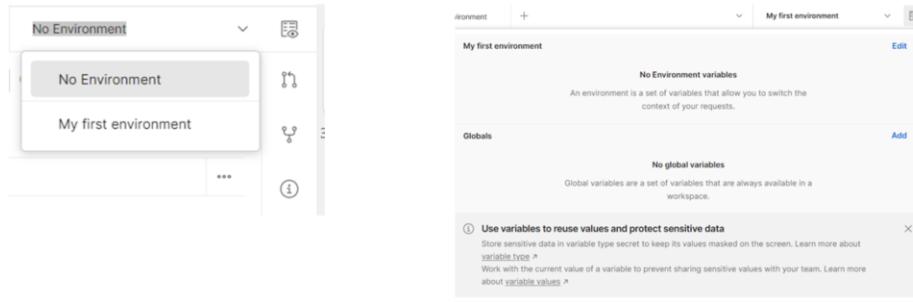
Tool	Available for	Description
Documentation 	Requests	View the documentation for a specific request.
Comments 	Collections, requests, APIs	Leave comments on a specific part of a request or an API.
Code 	Requests, APIs, history	Generate code snippets in a variety of languages and frameworks that you can use in other apps.
Live Collections 	Collections	Keep collections up-to-date based on your application using Live Collections.
Related collections 	Requests, history	View public collections from the Public API Network that share the same base URL as your request and include API documentation.
Info 	Collections, requests, APIs, environments, mock servers, monitors, Flows	See details about the element, including its ID, when it was created, who created it, and more.

## Right sidebar (3)

Tool	Available for	Description
Changelog ⓘ	Collections, APIs	Use the changelog to see changes that you and your collaborators have made.
Pull requests ⓘ	Collections, environments	View any pull requests for a Postman element.
Forks ⓘ	Collections, environments, Flows	View any forks of a Postman element.
Activity feed ⓘ	Monitors	View the activity feed for a monitor.
Flow Element Info ↑↓	Flows	View the input and output of the selected block.
Released Versions 🎉	Flows	View your released Flows.
Execution issues ⚠️	Flows	View a Flow's execution issues, if any.

## Environment selector and environment quick look

- The environment selector enables you to choose which **environment** to use in your work.
- You can select an environment from the menu to set it as the **active** environment, which gives you access to the variables in that environment.



## Footer

- The footer on the bottom of Postman enables you to find and replace text, open the Console, capture requests and cookies, and access several other tools.

  Online  Find and replace  Console

 Postbot  Runner  Start Proxy  Cookies  Trash  

## Footer (1)

  Online  Find and replace  Console

-  **Hide sidebar** - Close or reopen the sidebar.
-  **Sync status** – To see your data is syncing if you are connected to Postman's servers.
-  **Find and replace** - Search the current workspace. You can also use the shortcuts **Ctrl+Shift+F**. Enter your search string then select Find. Limit your search to a specific element type by selecting **Collections**, **Environments**, **Global**, or **Open tabs**. To replace your search term in a selected element, select **Replace** in selected. (*Available on the Postman desktop app*)
-  **Console** - Inspect and debug your Postman requests.

## Footer (2)

 Postbot  Runner  Start Proxy  Cookies  Trash  

- **Postbot** - Open the Postbot AI assistant to ask questions and search for answers.
- **Runner** - Open the Collection Runner.
- **Start Proxy** - Start the Postman proxy. (*Available on the Postman desktop app*)
- **Cookies** - Cookies in Postman work just like they do in a browser. They store session data, authentication tokens, or user preferences when making API requests..
- **Two-pane view** - Toggle between a single pane view and a two-pane view.
- **Help** - Access more resources, including release notes and Postman Support.

## Postman Requests



**RUBRIC**

92

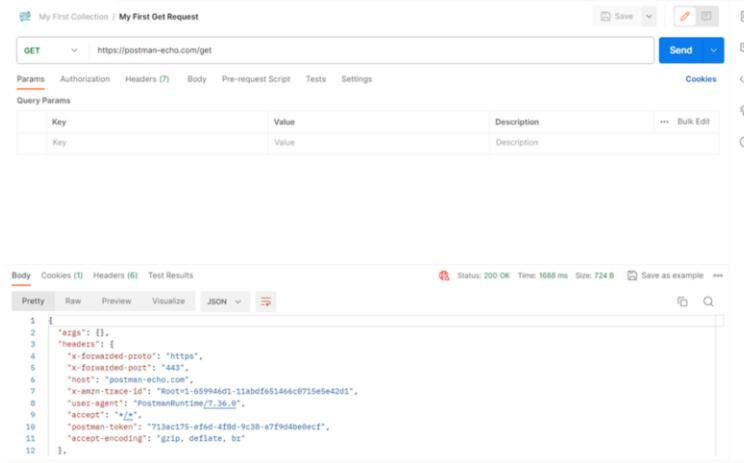
- In this chapter, you will learn about:
  - Overview of Postman Requests
  - API response structure
  - Generate code snippets
  - Debug API requests using the Postman Console

## API requests in Postman

- You can send requests in Postman to connect to APIs you are working with. Your requests can **retrieve, add, delete, and update** data.
- Whether you are building or testing your own API, or integrating with a third-party API, you can send your requests in Postman. Your requests can send **parameters, authorisation details, and any body data** you require.
  - For example, if you're building a client application (such as a mobile or web app) for a store, you might send one request to retrieve the list of available products, another request to create a new order (including the selected product details), and a different request to log a customer in to their account.

## Create and send API requests in Postman

- When you send a request, Postman displays the response received from the API server in a way that lets you examine, visualise, and if necessary, troubleshoot it.



The screenshot shows the Postman interface with a GET request to <https://postman-echo.com/get>. The request tab shows the URL and method. Below it, the 'Params' tab is selected, showing a single entry 'Key'. The 'Headers' tab shows several forwarded headers. The 'Body' tab is empty. The 'Test Results' tab at the bottom shows a JSON response with various keys like 'args', 'headers', 'x-forwarded-proto', 'x-forwarded-port', 'host', 'x-forwarded-for', 'user-agent', 'postman-token', and 'accept-encoding'.

```

1 {
2   "args": {},
3   "headers": {
4     "x-forwarded-proto": "https",
5     "x-forwarded-port": "443",
6     "host": "postman-echo.com",
7     "x-forwarded-for": "127.0.0.1-659944d1-11abdf65144c0715e9e42d1",
8     "user-agent": "PostmanRuntime/7.36.0",
9     "accept": "*/*",
10    "postman-token": "713ac175-e1fd-4f8d-9c38-a7ff9d4be0ecf",
11    "accept-encoding": "gzip, deflate, br"
12  },

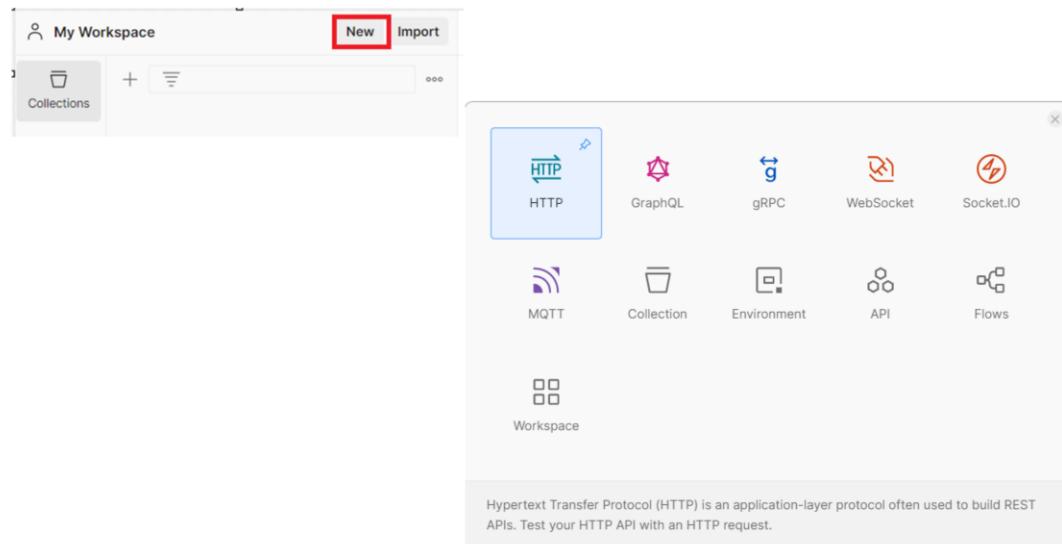
```

94

## Create Postman requests

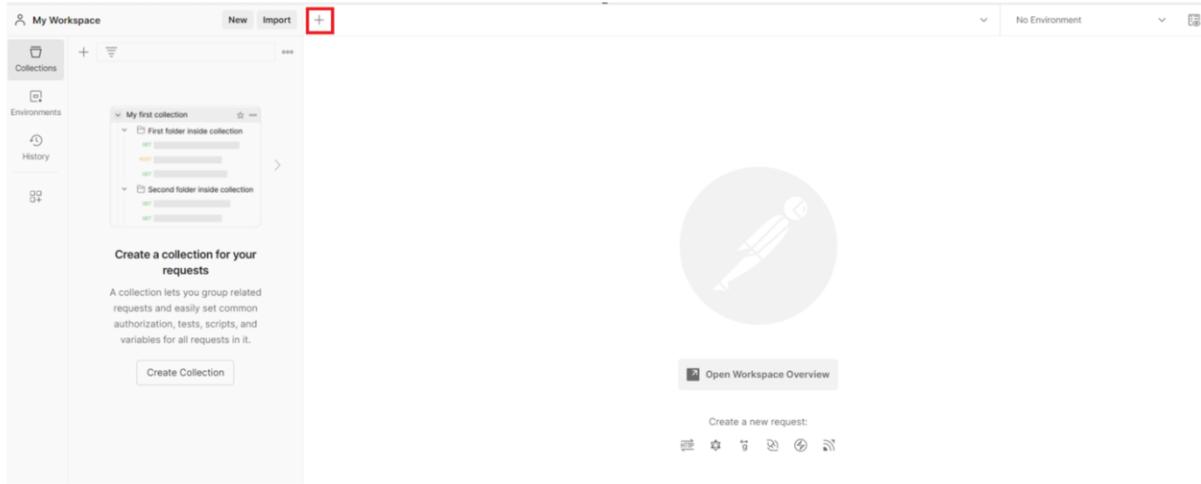
- Postman requests can include multiple details determining the data Postman will send to the API you are working with.
  1. Enter a URL.
  2. Choose an HTTP method.
  3. Specify a variety of other details. (*optional*)
- You can create a new request from a workspace, by using **New > HTTP**, or by selecting **+** to open a new tab.

## Create request: Option 1



96

## Create request: Option 2



The screenshot shows the RUBRIC workspace interface. On the left, there's a sidebar with 'My Workspace' at the top, followed by 'Collections' (with a red box around the '+' icon), 'Environments', 'History', and 'UV'. Below this is a section titled 'Create a collection for your requests' with a sub-section about collections. At the bottom of this sidebar is a 'Create Collection' button. The main area shows a collection named 'My first collection' containing two folders: 'First folder inside collection' and 'Second folder inside collection', each with several items. To the right of the collection list is a large circular icon with a pen nib. Below the icon is a 'Open Workspace Overview' button. At the bottom of the workspace area is a 'Create a new request:' section with various icons.

97

## Create request

GET Untitled Request X + No Environment

Untitled Request

Save Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description	...	

Response

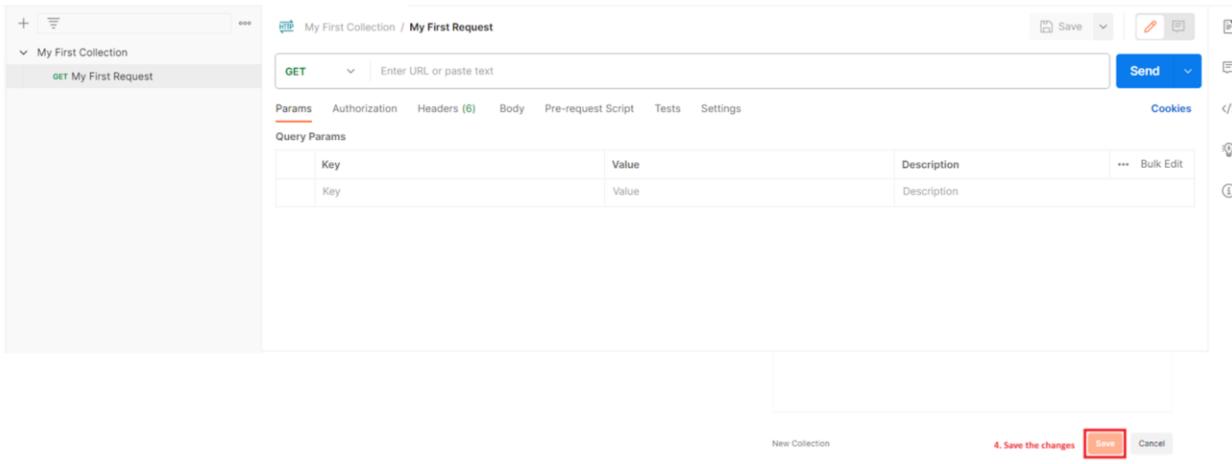
Enter the URL and click Send to get a response



98

## Save Request in new collection

- Select Save to create your request. You can give your request a **name** and **description**, and **choose or create a collection** to save it in.



The screenshot shows the Postman interface. On the left, there's a sidebar with a collection named "My First Collection" containing a request named "My First Request". The main area shows a "My First Request" with a "GET" method. The "Params" tab is selected, showing a single query parameter "Key" with a value "Value". At the bottom right of the main area, there's a "Save" button with a red border and the text "4. Save the changes". To the right of the main area, there are various icons for file operations like copy, paste, and delete.

99

## Adding request detail

- To execute a request, you need to know the **URL**, **method**, and other optional values such as **auth** and **parameters**.
- You can also follow these steps to achieve the above:
  1. Selecting protocols
  2. Setting request URLs
  3. Selecting request methods
  4. Sending parameters
  5. Sending body data
  6. Authenticating requests
  7. Configuring request headers
  8. Using cookies

10  
0

## Test a simple GET request

1. Set the URL to the Postman Echo sample API endpoint:

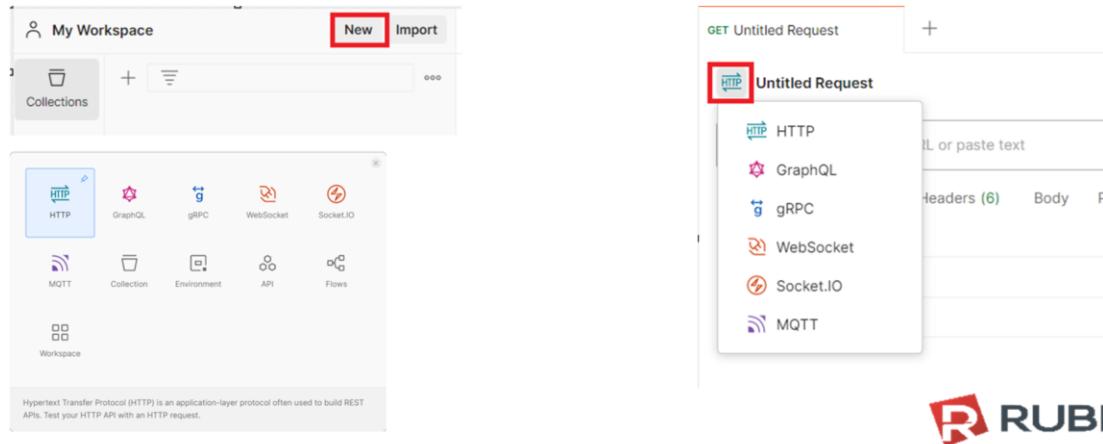
*https://postman-echo.com/get*

2. Select HTTP method GET.
3. Select Send.

The screenshot shows the Postman application interface. At the top, it says "My First Collection / My First Request". Below that, there's a search bar with "GET" and a dropdown arrow, and a URL input field containing "https://postman-echo.com/get". To the right of the URL field is a "Send" button with a blue background and white text. On the left side, there's a sidebar with various HTTP methods: GET (selected), POST, PUT, PATCH, DELETE, HEAD, and OPTIONS. Below the sidebar, it says "Type a new method". The main area has tabs for Headers (7), Body, Pre-request Script, Tests, and Settings. On the far right, there are several icons for saving, deleting, and editing. A red box highlights the "Send" button, and another red box highlights the URL input field.

## 1. Selecting protocols

- You can change the protocol for a new request. Select **New** in the sidebar and then select a request protocol or select **+** to open a new tab. Select the **protocol dropdown menu** to the left of the request's name, and then select a different request protocol.



10  
2

 RUBRIC

**NOTE:** You can't change the request protocol after you select **Save**.

## 2. Setting request URLs

- Each request you send in Postman requires a **URL** representing the API endpoint you are working with. Each operation you can perform using an API is typically associated with an **endpoint**. Each endpoint in an API is available at a particular URL.
  - If you're building an API, the URL will typically be the base location plus path. For example, in the request <https://postman-echo.com/get>, https://postman-echo.com is the **base URL**, and /get is the **endpoint path**.
  - If you're using a third-party API, your API provider will supply the URLs you need, for example within their developer documentation.



10  
3

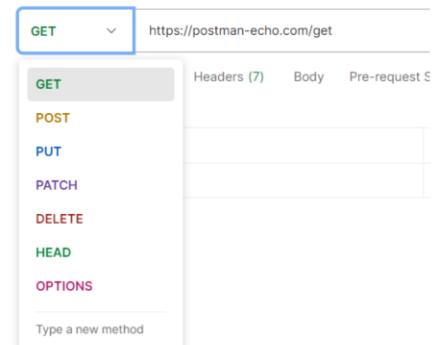
 RUBRIC

### NOTE:

- When you start typing in the URL input field, Postman will present a dropdown list of previously used locations you can use to autocomplete.
- Postman will automatically add **http://** to the start of your URL if you don't specify a protocol.
- You can optionally enter **query parameters** into the URL field, or you can enter them in the **Params** tab. If your request uses **path parameters**, you can enter them directly into the URL field.

### 3. Selecting request methods (1)

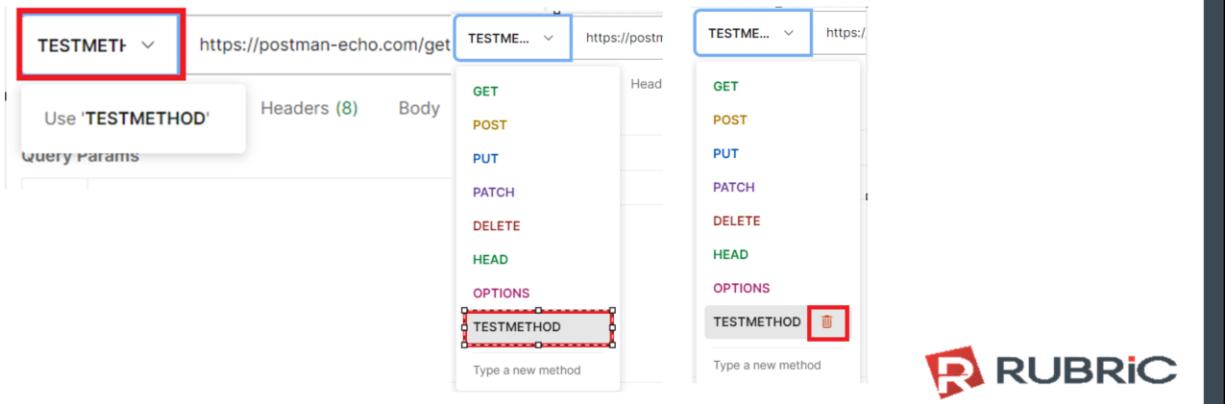
- By default, Postman will select the **GET** method for new request. **GET** method is typically for retrieving data from an API. You can use a variety of other methods to send data to your APIs, including the following most common options:
  - **POST** - add new data
  - **PUT** - replace existing data
  - **PATCH** - update some existing data fields
  - **DELETE** - delete existing data

10  
4

For example, if you're working with an API for a To Do list application, you might use a **GET** method to **retrieve** the current list of tasks, a **POST** method to **create** a new task, and a **PUT** or **PATCH** method to **edit** an **existing** task.

### 3. Selecting request methods (2)

- Postman supports several extra request methods by default, and you can use **custom** methods.
- Select the **method** dropdown list, edit the method name text, and save your new method. To delete a method, hover over it in the list and select the delete icon Delete icon .



The screenshot shows the Postman interface with two tabs open: 'https://postman-echo.com/get' and 'https://postman-test.com/post'. Both tabs have their method dropdowns set to 'TESTMETHOD'. The left tab's dropdown has a red box around it, and its list includes 'TESTMETHOD' at the bottom. The right tab's dropdown also has a red box around it, and its list includes 'TESTMETHOD' at the bottom, with a small trash icon next to it. A red box highlights the 'TESTMETHOD' entry in both lists. Below each dropdown is a text input field: 'Type a new method' on the left and 'Type a new method' on the right, each with a small trash icon next to it.

10  
5

### 3. Selecting request methods (3)

- The same location (sometimes called "route") can provide more than one endpoint by accepting different methods.
  - For example, an API might have a **POST /customer** endpoint for adding a new customer, and a **GET /customer** endpoint for retrieving an existing customer.

10  
6

## 4. Sending parameters (1)

- You can send **path** and **query** parameters with your requests using the **URL** field and the **Params** tab.
- **Query** parameters are used for filtering or sorting data. They are **appended** to the **end** of the request URL, following **?** and listed in **key value pairs**, separated by **&** using the following syntax:

`?id=1&type=new`

- **Path parameters** form **part of the request URL** and they are used to specify a particular resource (e.g., a specific user). They are referenced using **placeholders** preceded by **:** as in the following example:

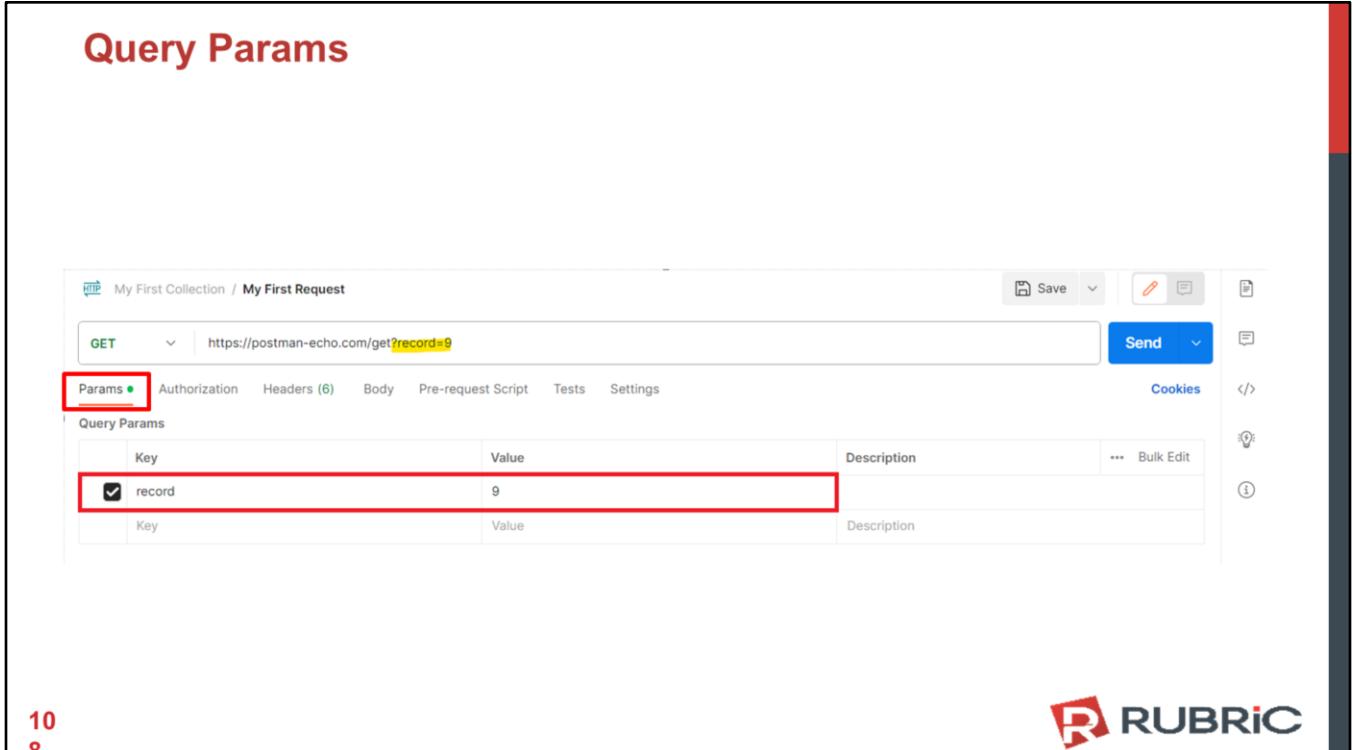
`/customer/:id`

10  
7

### NOTE:

- When you enter your **query parameters** in either the URL or the Params fields, these values will update everywhere they're used in Postman.
- When you enter a **path parameter** in the URL , Postman will populate it in the Params tab, where you can also edit it.

## Query Params



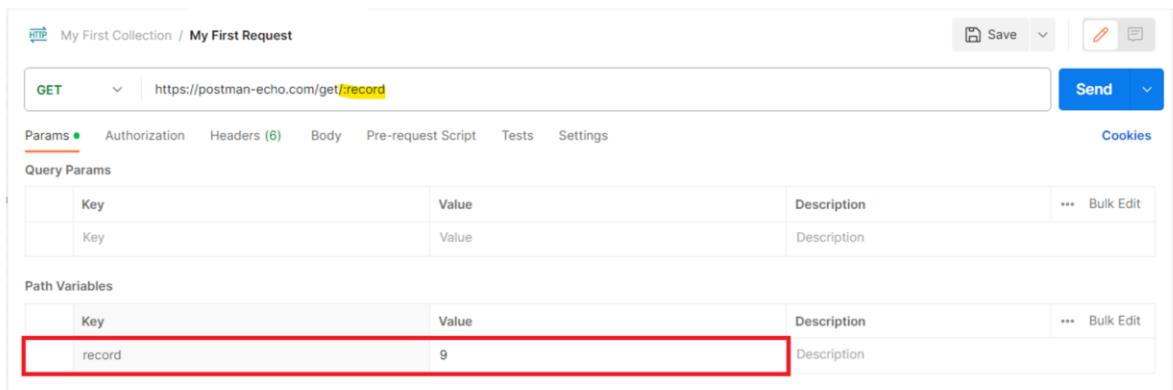
The screenshot shows the Postman interface with a 'My First Collection / My First Request' tab. The request method is 'GET' and the URL is 'https://postman-echo.com/get?record=9'. The 'Params' tab is selected, showing a table with one row: 'record' (Key) with value '9'. Other tabs like 'Authorization', 'Headers', 'Body', etc., are visible but not selected.

10  
8

### NOTE:

- You can add descriptions to your parameters, and they'll appear for anyone sharing the request (for example in your workspace) or viewing your API documentation.
- You can use the **Bulk Edit** option if you prefer to enter your parameters in plain text instead of using the request builder.

## Path Params



HTTP My First Collection / My First Request

GET https://postman-echo.com/get/**record**

Params • Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description	...	Bulk Edit

Path Variables

Key	Value	Description	...	Bulk Edit
record	9	Description	...	Bulk Edit

10  
9

## 5. Sending body data (1)

- You need to send **body data** with requests whenever you need to **add** or **update** structured data.
  - For example, if you're sending a request to **add** a new customer to a database, you might include the customer details in **JSON**.
- You will use body data with **PUT**, **POST**, and **PATCH** requests.
- The **Body** tab in Postman enables you to specify the data you need to send with a request. You can send various types of body data to suit your API.

## 5. Sending body data (2)

- There are various data type you can use to send the body data:
  - form data
  - URL-encoded
  - raw,
  - binary
  - GraphQL

The screenshot shows the Postman interface for a 'My First Request' in 'My First Collection'. The request method is 'POST' and the URL is 'https://postman-echo.com/record'. The 'Body' tab is highlighted with a red box. Below it, there are five options: 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', and 'GraphQL'. A note at the bottom says 'This request does not have a body'.

11  
1

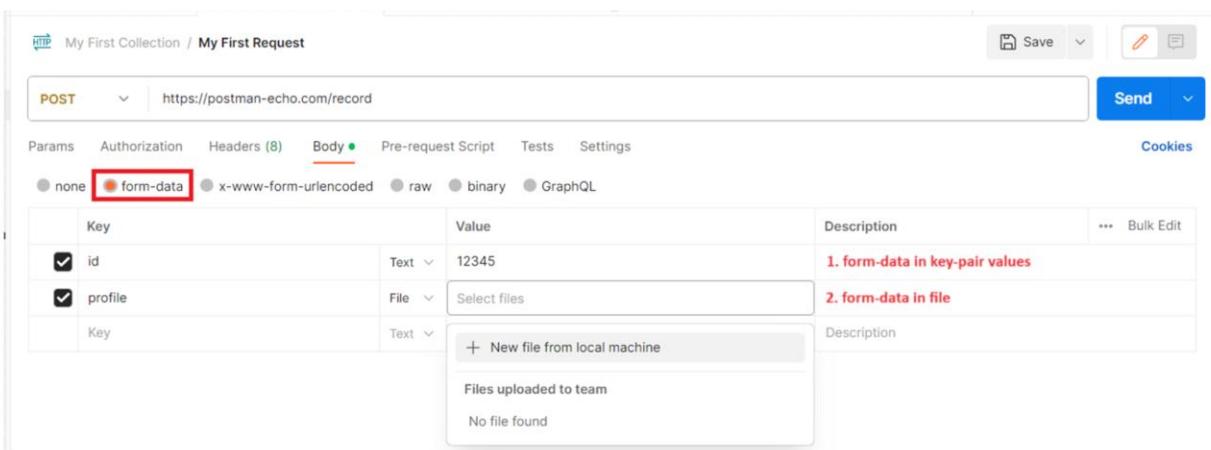
### NOTE:

By default, Postman will select **None**—leave it selected if you don't need to send a body with your request.

## Form data (1)

- Website forms often send data to APIs as **multipart/form-data**. You can replicate this in Postman using the **form-data Body tab**.
- Form data enables you to send **key-value pairs** and specify the **content type**.
- You can also attach a **file** using form data and send it with your request. Select **File** in the dropdown list next to a key name, select the file you want to send.

## Form data (2)



The screenshot shows the Postman interface for a POST request to <https://postman-echo.com/record>. The 'Body' tab is selected, showing the 'form-data' option highlighted with a red box. The table below lists three key-value pairs:

Key	Type	Description
<input checked="" type="checkbox"/> id	Text	1. form-data in key-pair values
<input checked="" type="checkbox"/> profile	File	2. form-data in file
Key	Text	Description

The 'profile' row has a dropdown menu open, showing options like 'Select files', 'New file from local machine', 'Files uploaded to team', and 'No file found'. The 'Send' button is visible at the top right.

11  
3

### NOTE:

You can select a file from your local system, and Postman saves the file path in the request. The saved file path is relative to your local working directory.

## Upload files for shared requests and cloud runs (1)

- You can attach a file with test data to a request as form data or binary data. Postman will save the file path relative to your local working directory and use the file when sending the request. However, if you share the request in a workspace, the local file **isn't** shared.
- This means other team members won't be able to send the request unless they put a copy of the same file in their own local working directory. Also, a local file won't be available when sending a request from a monitor or a scheduled collection run, which are run in the Postman cloud and not locally.

11  
4

### NOTE:

Test data storage is available on **Postman Free**, **Basic**, and **Professional** plans.

## Upload files for shared requests and cloud runs (2)

- To enable sharing requests that use test data files, you can **upload** the files to your Postman team. Uploaded files are **available** to all members of your team and can be used to send requests that are shared in a workspace.
- Uploaded files are also available to requests sent from monitors and scheduled collection runs and can be used from Postman Flows and the Postman CLI (but not Newman).

## Upload file to Postman team (1)

1. Open a request and select the **Body** tab.
2. Select **form-data** or binary depending on the type of data you want to send with the request.
3. If you're attaching form data, select **File** in the dropdown list next to a key name.

The screenshot shows the Postman interface with a POST request to <https://postman-echo.com/record>. The 'Body' tab is selected (indicated by a red number 1). Under the 'Body' tab, the 'form-data' option is selected (indicated by a red number 2). In the data table, there is one row with a checked checkbox for the 'profile' key (indicated by a red number 3). The 'Value' column for this row has a dropdown menu open, showing 'Text' and 'File' options, with 'File' being the selected option.

11  
6

## Upload file to Postman team (2)

### 4. Select the test data file you want to use for the request:

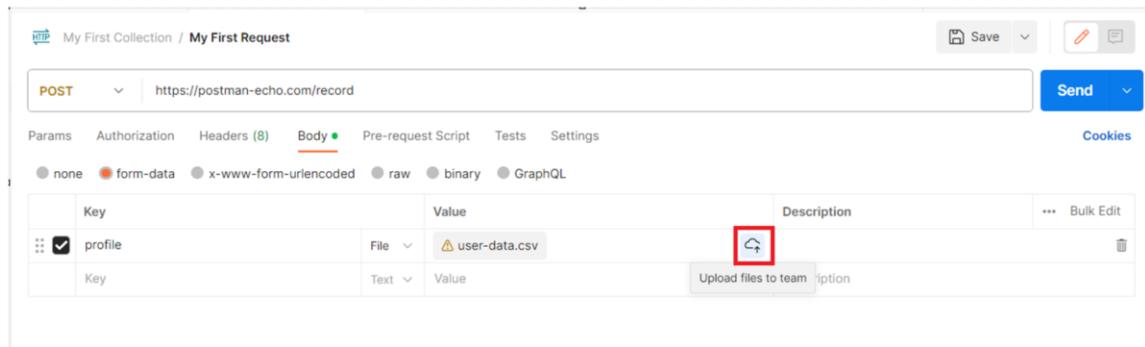
- To use a local file, select **+ New file** from local machine. Select a file and select **Open**. Supported file types are **CSV**, **JSON**, and **binary**.
- To use a file that was uploaded before, select the file in the list. You can use any file uploaded by a member of your team. To search for a file, start typing the file name.

The screenshot shows the 'Body' tab in Postman for a POST request to 'https://postman-echo.com/record'. The 'form-data' option is selected. A table lists a single key 'profile' with a checked checkbox. The 'Value' column contains a 'File' input field with a dropdown menu showing 'Select files' and '+ New file from local machine'. Below this, a message says 'Files uploaded to team' and 'No file found'.

11  
7

## Upload file to Postman team (3)

- 5. To upload a file, select the upload icon next to the file and select **Upload**. Uploaded files can't exceed **5 MB** in size.**



The screenshot shows the Postman interface with a request to 'https://postman-echo.com/record'. The 'Body' tab is selected, showing a table with a row for 'profile'. The 'Value' column for 'profile' contains a file named 'user-data.csv'. To the right of this file, there is an 'Upload files to team' button with a red box drawn around it. The 'Body' tab has a green dot next to it, indicating it's selected.

11  
8

After uploading the file, other team members can send the request without needing to place a copy of the file in their local working directory. Instead, the request will use the uploaded file. Also, the uploaded file is used if the request is sent from a monitor or a scheduled collection run.

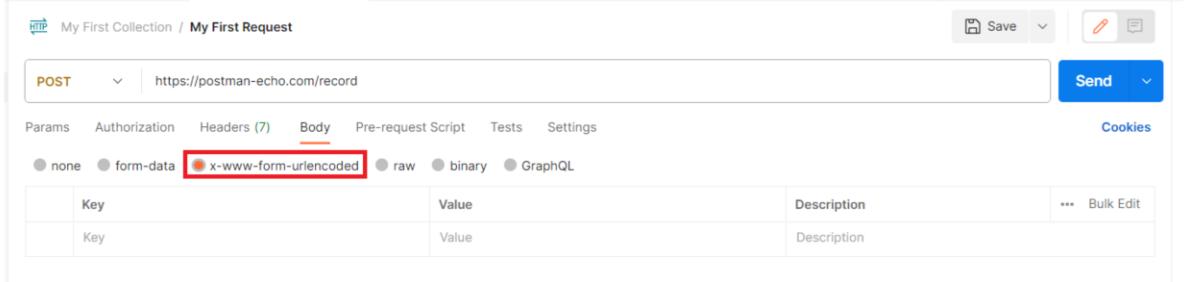
Uploaded files can be used in any requests you or other members of your team create (except in public workspaces). Uploaded files can also be accessed by Postman Flows and the Postman CLI when automating runs. If you fork a collection that uses test data files, you may need to upload any files that haven't been uploaded to your team.

### NOTE:

Your Postman plan gives you a limited amount of storage space you can use for uploaded test data files. Your plan also gives you a limited number of retrievals of uploaded files. Learn more about test data usage.

## URL-encoded

- URL-encoded data uses the same encoding as URL parameters. If your API requires url-encoded data, select **x-www-form-urlencoded** in the **Body** tab of your request. Enter your **key-value pairs** to send with the request and Postman will encode them before sending.



My First Collection / My First Request

POST https://postman-echo.com/record

Save Send

Params Authorization Headers (7) **Body** Pre-request Script Tests Settings Cookies

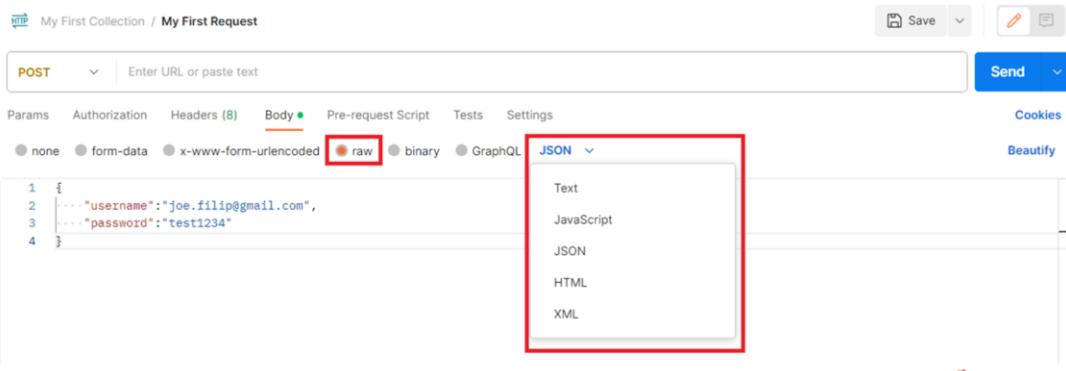
none  form-data  **x-www-form-urlencoded**  raw  binary  GraphQL

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

11  
9

## Raw data

- You can use raw body data to send anything you can enter as text. Use the raw tab, and the type dropdown list to indicate the format of your data (**Text**, **JavaScript**, **JSON**, **HTML**, or **XML**) and Postman will enable syntax-highlighting and appending the relevant headers to your request.



The screenshot shows the Postman interface for a 'My First Request'. The 'Body' tab is selected, showing a dropdown menu with options: none, form-data, x-www-form-urlencoded, raw (which is highlighted with a red box), binary, and GraphQL. Below this, there is a code editor containing JSON data:

```

1 {
2   "username": "joe.filip@gmail.com",
3   "password": "test1234"
4 }

```

The 'raw' option in the dropdown menu is also highlighted with a red box. To the right of the dropdown, there is a 'JSON' dropdown with options: Text, JavaScript, JSON, HTML, and XML. The 'JSON' option is currently selected.

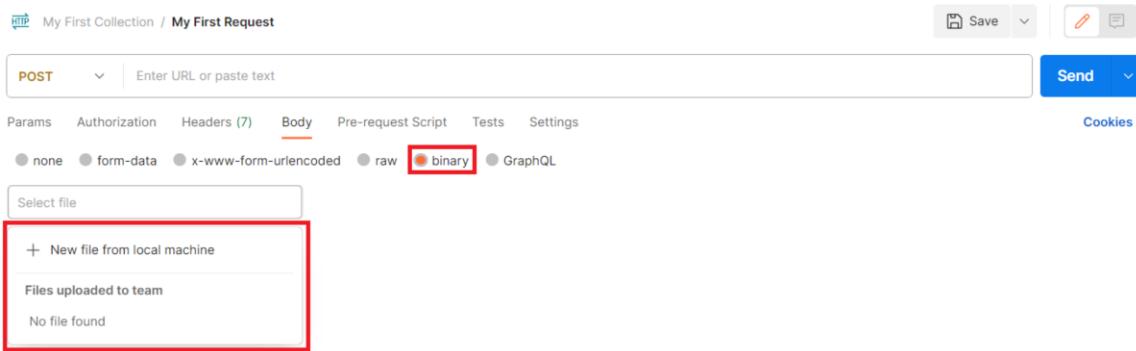
12  
0

### NOTE:

- You can set a **content type header** manually if you need to override the one Postman sends automatically.
- You can use **variables** in your body data and Postman will populate their current values when sending your request.
- For JSON raw body data, you can **add** comments, and they will be stripped out when the request is sent. Single-line comments delimited with // and multi-line comments delimited with /\* \*/ will be removed in the request.
- To beautify your **XML** or **JSON**, select the text in the **editor** and then select **Ctrl+Alt+B**.

## Binary data

- You can use binary data to send information you can't enter manually in the Postman editor with your request body, such as **image, audio, and video files.** (*You can also send text files.*)



The screenshot shows the Postman interface with a 'POST' request selected. The 'Body' tab is active. In the file selection dropdown, the 'binary' option is highlighted with a red box. Below the dropdown, a red box highlights the '+ New file from local machine' button. The interface also shows 'Params', 'Authorization', 'Headers (7)', 'Pre-request Script', 'Tests', and 'Settings' tabs. On the right, there are 'Save', 'Send', and 'Cookies' buttons.

12  
1

### NOTE:

You can also upload a file with test data to your Postman team. This is useful if you want to share the request with others on your team or use the request in a monitor or scheduled collection run. (*same as uploading file in form-data*)

## GraphQL

- GraphQL is **schema-driven**, providing insight into an API's functionality and reducing dependency between teams building the client and the server.
- A client can introspect the schema from the server to see the **available data fields** and then send **queries** specifying which fields to retrieve or manipulate.
- The server returns or modifies only the data requested in the query, which prevents **overfetching** or **underfetching** data.



APIs that support **GraphQL** enable clients to ask the server for only the data they need, using **GraphQL's** powerful **query language** and runtime. Unlike REST HTTP, which uses multiple endpoints to access different data sets, GraphQL streamlines querying by accessing all data through a **single endpoint**.

## GraphQL requests (1)

- Every GraphQL request has a **URL** and a **query**. The URL is the endpoint where the data is hosted, and the query defines what data to retrieve or manipulate.
- The request can also contain **authentication**, **headers**, and **settings** based on the requirements specified by the API.

12  
3

## GraphQL requests (2)

- **GraphQL requests can perform three types of operations:**
  - **Query** - Retrieves data from the server. Queries specify the required data fields and can include arguments for more precise data retrieval.
  - **Mutation** - Manipulates data on the server, including creating, updating, or deleting records. Mutations specify the fields to be returned after the operation and use arguments to detail the manipulation.
  - **Subscription** - Gets real-time data updates from the server. Subscriptions enable clients to listen to specific data fields and receive updates automatically over a persistent connection.

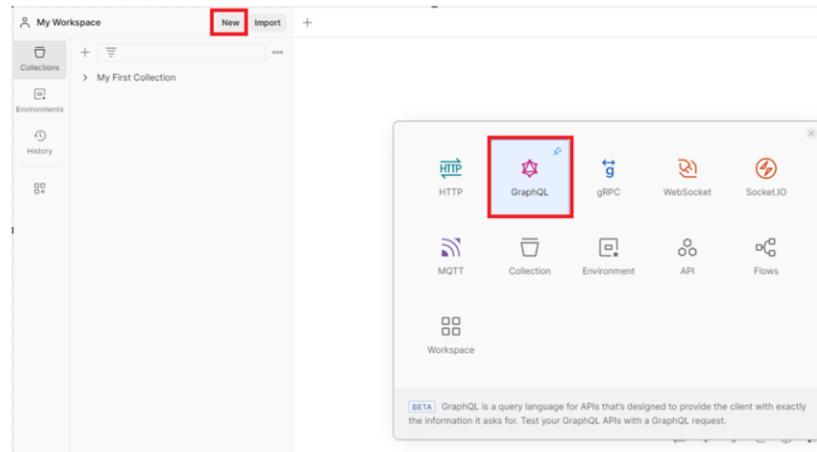
## GraphQL requests (2)

- Let's try to understand how to execute GraphQL based API using the POSTMAN tool.
- For example, we will test with a simple request that would query all the continents list against the given endpoint:

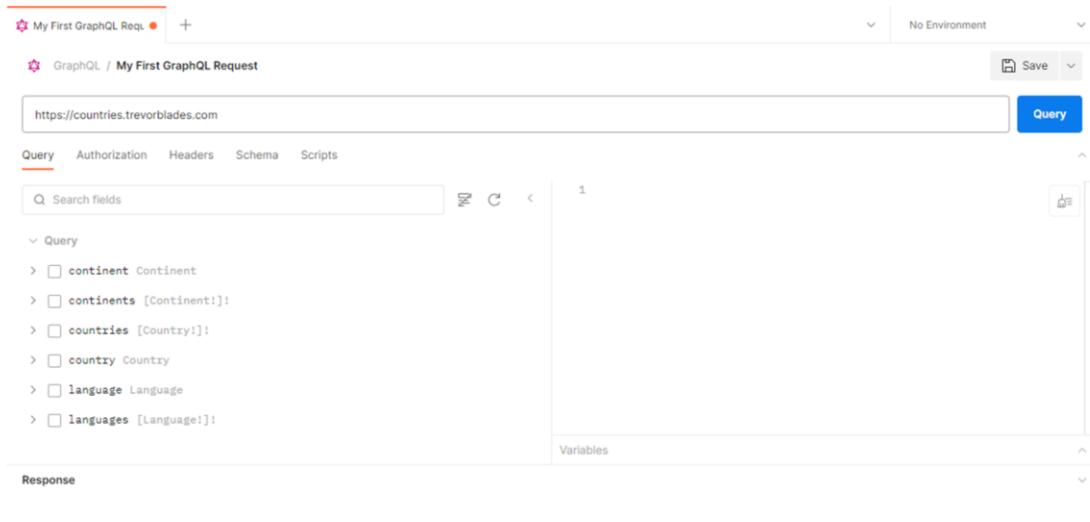
<https://countries.trevorblades.com>

## Create GraphQL Request

- You can create GraphQL request by selecting **New** in the sidebar and then select **GraphQL** or by selecting **+** to open a new tab.



## Configure GraphQL Request



My First GraphQL Req. +

No Environment

GraphQL / My First GraphQL Request

https://countries.trevorblades.com

Save Query

Query Authorization Headers Schema Scripts

Search fields

Variables

Response

continent

continents

countries

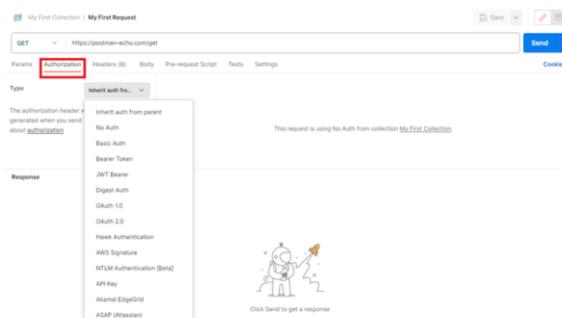
country

language

languages

## 6. Authenticating requests

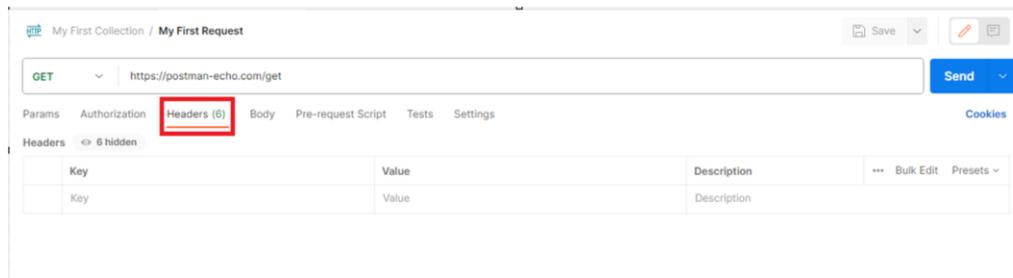
- Some APIs require auth details you can send in Postman.
- Authentication involves **confirming the identity** of the client sending a request, and authorization involves confirming that the client has **permission** to carry out the endpoint operation.
- You can configure the access details under **Authorization** tab to configure your access details.

**NOTE:**

Postman will automatically include your authentication details in the relevant part of the request, for example in Headers.

## 7. Configuring request headers

- Some APIs require you to send **headers** along with requests, typically to provide more **metadata** about the **operation** you are performing.
- You can set these up in the **Headers** tab. Enter **any key-value pairs** you need, and Postman will send them along with your request. As you enter text, Postman **prompts** you with common options you can use to **autocomplete** your setup, such as **Content-Type**.



The screenshot shows the Postman application interface. At the top, there's a header bar with 'My First Collection / My First Request'. Below that is a toolbar with 'Save', 'Edit', and 'Send' buttons. The main area has tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Headers' tab is active and highlighted with a red box. Below the tabs is a table with columns 'Key', 'Value', 'Description', and buttons for 'Bulk Edit' and 'Presets'. There are 6 hidden headers listed. The URL in the address bar is 'https://postman-echo.com/get'.

Think of **headers** in Postman as **labels** on a package you send by mail. These labels help the recipient (server) understand what's inside and how to handle it.

### Content-Type (What kind of data are you sending?)

Example:

- If sending **JSON data**, use → Content-Type: application/json
- If sending a file (like an image), use → Content-Type: multipart/form-data

### Accept (What kind of response do you want?)

Example:

- If you want the response in **JSON format**, use → Accept: application/json
- If you want **XML**, use → Accept: application/xml

### Cookie (Sending stored login info)

Example:

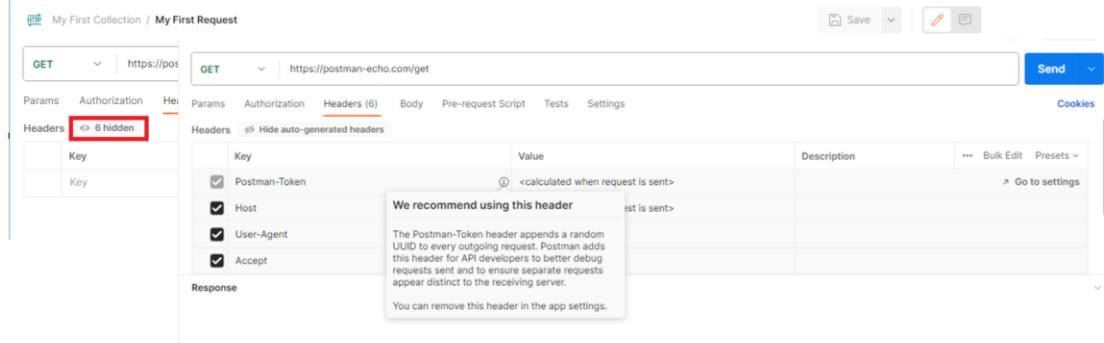
If you're logged into a website, your browser sends a **cookie** like:

Cookie: sessionId=abc123

This keeps you logged in.

## Autogenerated headers

- Postman automatically adds certain headers to your requests based on your request selections and settings. Select **hidden** at the top of the **headers** tab for information about what Postman will send with your request.

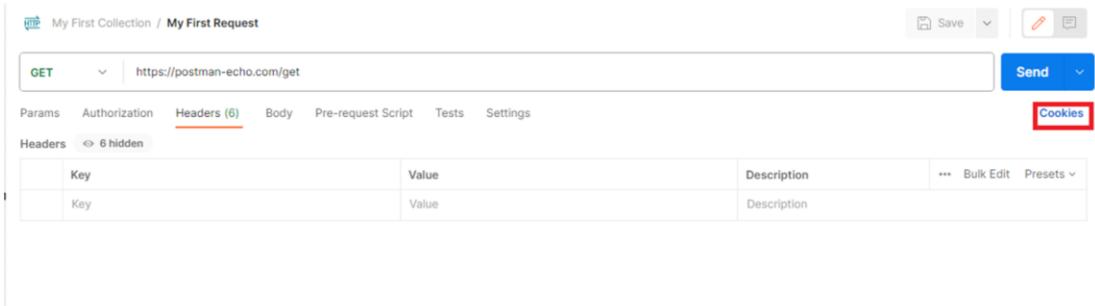


The screenshot shows the Postman interface with a collection named "My First Collection" and a request named "My First Request". The request method is GET and the URL is https://postman-echo.com/get. The "Headers" tab is selected, showing a table with columns: Key, Value, Description, Bulk Edit, and Presets. There are four rows: "Postman-Token" (Value: <calculated when request is sent>, Description: "The Postman-Token header appends a random UUID to every outgoing request. Postman adds this header for API developers to better debug requests sent and to ensure separate requests appear distinct to the receiving server."), "Host" (Value: host), "User-Agent" (Value: user-agent), and "Accept" (Value: accept). A note below the table says "We recommend using this header". A red box highlights the "6 hidden" link in the Headers sidebar.

13  
0

## 8. Using cookies

- You can manage **Cookies** for your domains from Postman. Select **Cookies** (under Send).



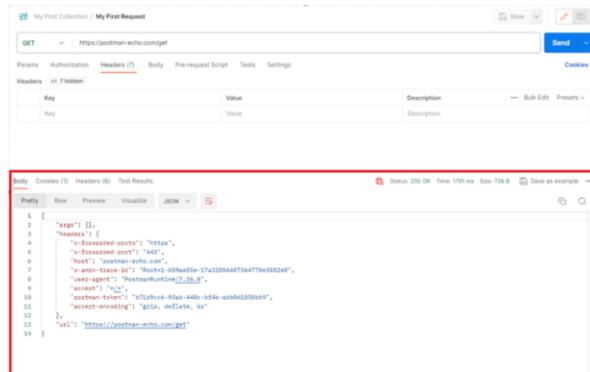
The screenshot shows the Postman interface for a 'My First Request' collection. The request method is 'GET' and the URL is 'https://postman-echo.com/get'. The 'Headers' tab is selected, showing a table with one row: 'Key' (Value) and 'Value' (Description). A red box highlights the 'Cookies' button in the top right corner of the interface.

13  
1

Cookies in Postman work just like they do in a browser. They store **session data**, authentication tokens, or user preferences when making API requests.

## API response structure in Postman

- The **Postman response** viewer helps you visualise and check the correctness of **API responses**. An API response consists of the **response body**, **headers**, **cookies**, and the **HTTP status code**. You can view details about the response, including **test results**, **network information**, **response size**, **response time**, and **security warnings**. You can also save responses as examples or files.



The screenshot shows the Postman interface with a request to `https://postman-echo.com/get`. The response body is displayed in JSON format:

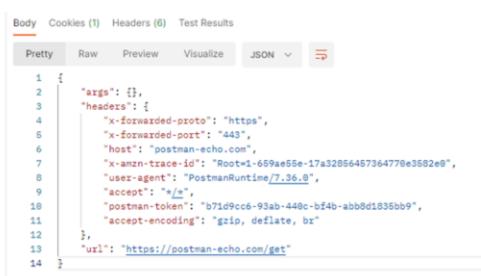
```

1 {
2   "args": {},
3   "headers": {
4     "x-forwarded-port": "443",
5     "x-forwarded-proto": "https",
6     "host": "postman-echo.com",
7     "user-agent": "PostmanRuntime/7.30.0",
8     "accept": "*/*",
9     "postman-token": "97d6c6-93a0-44b5-bd4b-a0dd0d339009",
10    "accept-encoding": "gzip, deflate, br"
11  },
12  "url": "https://postman-echo.com/get"
13 }
14

```

## Response body (1)

- The Postman **Body** tab gives you several tools to help you understand the **response**. You can view the body in one of four views: **Pretty**, **Raw**, **Preview**, and **Visualise**.
- Pretty**
  - The **Pretty** view formats **JSON** or **XML** responses so they're easier to view. Links inside the **Pretty** view are highlighted and by selecting them can load a GET request in Postman with the link URL.



```

1 {
2   "args": {},
3   "headers": {
4     "x-forwarded-proto": "https",
5     "x-forwarded-port": "443",
6     "host": "postman-echo.com",
7     "x-amzn-trace-id": "Root=1-69ae55e-17a32856487364770e3682e0",
8     "user-agent": "PostmanRuntime/7.36.0",
9     "accept": "*/*",
10    "postman-token": "b71d9cc6-93ab-448c-bf4b-abb8d1836bb9",
11    "accept-encoding": "gzip, deflate, br"
12  },
13  "url": "https://postman-echo.com/get"
14 }

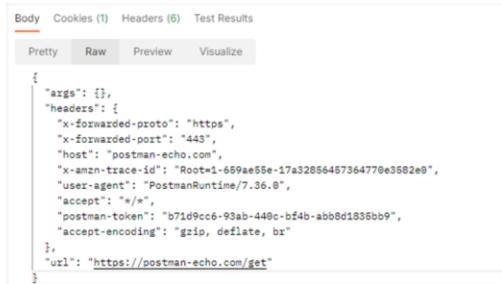
```

### NOTE:

- If the response's Content-Type header indicates that the response is an **image**, Postman will detect and render the **image** automatically.
- For navigating large responses, select the down arrows next to a line to collapse large sections of the response.

## Response body (2)

- **Raw**
  - The **Raw** view is a large text area with the response body. It can indicate whether your response is minified.



The screenshot shows the 'Raw' tab of a Postman request. The response body is a JSON object:

```
{  
  "args": {},  
  "headers": [  
    "x-forwarded-proto": "https",  
    "x-forwarded-port": "443",  
    "host": "postman-echo.com",  
    "x-amzn-trace-id": "Root=1-669ae55e-17a32856487364770e3882e9",  
    "user-agent": "PostmanRuntime/7.36.0",  
    "accept": "*/*",  
    "postman-token": "b71d9cc6-93ab-448c-bf4b-abb8d1835bb9",  
    "accept-encoding": "gzip, deflate, br"  
  ],  
  "url": "https://postman-echo.com/get"  
}
```

13  
4

## Response body (3)

### ▪ Preview

- The Preview view renders the response in an iframe sandbox. Some web frameworks by default return HTML errors, and Preview can be helpful for debugging in those cases.
- Due to iframe sandbox restrictions, JavaScript and images are turned off in the iframe. For **binary** response types, you can select the down arrow next to Send and select Send and Download to save the response locally. You can then view it using the appropriate viewer. This gives you the flexibility to test **audio files**, **PDFs**, **zip files**, or any other file types the API returns.



A screenshot of a Postman API response interface. At the top, there are tabs for Body, Cookies (1), Headers (6), and Test Results. The Body tab is selected and contains a JSON object. Below the tabs, there are buttons for Pretty, Raw, Preview (which is selected), and Visualize. To the right, there is status information: Status: 200 OK, Time: 1791 ms, Size: 726 B, and a Save as example button. The JSON response is:

```
{"args": {}, "headers": { "x-forwarded-proto": "https", "x-forwarded-port": "443", "host": "postman-echo.com", "x-amzn-trace-id": "Root=1-659ae55e-17a32856457364770e3582e0", "user-agent": "PostmanRuntime/7.36.0", "accept": "*/*", "postman-token": "b71d9cc6-93ab-440c-bf4b-abb8d1835bb9", "accept-encoding": "gzip, deflate, br" }, "url": "https://postman-echo.com/get" }
```

13  
5

## Response body (4)

- **Visualize**
  - The Visualize view renders the data in the API response according to visualization code that you add to the requests Tests. For details on how to add, use, and debug visualization code, see Visualizing responses.

13  
6

## Cookies

- You can select **Cookies** to inspect cookies sent by the server. A cookie's entry includes its **name**, **value**, the **associated domain** and **path**, and other information about the cookie.



Name	Value	Expires	HttpOnly	Secure
sails.sid	s%3AtYIUFQv...	Session	true	false

## Headers

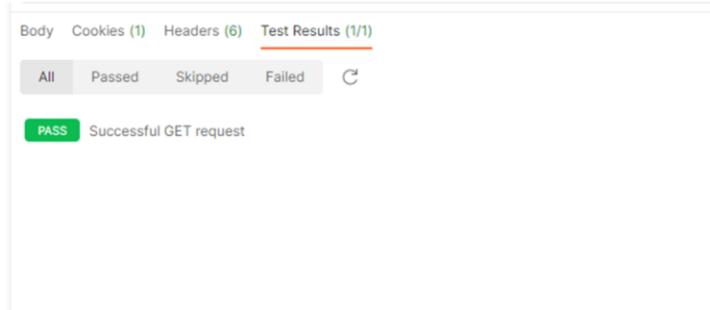
- **Headers** are displayed as **key-value pairs** under the **Headers** tab. Hover over the **information icon** next to the header name to get a description of the header according to the HTTP specification.

Body	Cookies (1)	Headers (6)	Test Results
Key		Value	
		① Sun, 07 Jan 2024 17:54:38 GMT	
		① application/json; charset=utf-8	
		① keep-alive	
		① W/"199-h4gEFwrNLkpjjSCt9l9mXvedFU"	
		① sails.sid=s%3AtYIUFQw4QBOi57gzTk7W1A8fkpsNsOJx.jH9y0%2BD8G5vRsD5MdxN2Oo94KQA...	
		① 409	

13  
8

## Test results

- If the API request you are viewing had any test scripts, the results are displayed in the **Test Results** tab.



The screenshot shows the 'Test Results' tab selected in the navigation bar. Below it, there are four filter buttons: 'All' (selected), 'Passed', 'Skipped', and 'Failed'. A green 'PASS' button indicates success, followed by the message 'Successful GET request'.

13  
9

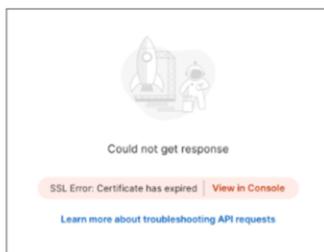
## Network information

- Postman displays network information when your API returns a response. Hover over the network icon to get the local and remote IP addresses for the request you sent.
- When you make an https request, the network icon includes a padlock. When you hover over the icon, the network information will show more information including certificate verification details.

Network	
Local Address	192.168.100.76
Remote Address	3.224.43.57
TLS Protocol	TLSv1.3
Cipher Name	TLS_AES_256_GCM_SHA384
Certificate CN	postman-echo.com
Issuer CN	ca.sdworx.eu.goskope.com
Valid Until	Jan 6 17:54:36 2025 GMT
 Self signed certificate in certificate chain	
 Status: 200 OK Time: 2.71 s	

## SSL verification errors

- SSL (Secure Sockets Layer) verification ensures that Postman only communicates with secure and trusted servers by checking their SSL/TLS certificate.
- If SSL verification is enabled and verification fails, the response area displays an error message. Select the link to open the Console and view more information about the error.



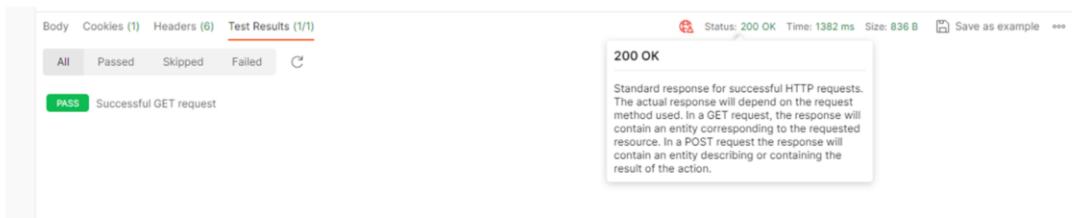
If needed, you can turn off SSL verification for the request or turn it off globally in Postman:

- To turn off SSL verification for a **request**, select the **Settings** tab in the request and **turn off Enable SSL certificate verification**.
- To turn off SSL verification **globally**, select the **settings** icon in the header and select **Settings**. Under **Request**, **turn off SSL certificate verification**.

If you have SSL verification turned off and your request returns a certificate verification error, you can hover over the network information for details about the error.

## Response code

- Postman displays the **response code** returned by the API. Hover over the response code to get a short description of the code and what it means.



The screenshot shows a Postman interface with the following details:

- Header tabs: Body, Cookies (1), Headers (6), Test Results (1/1).
- Status bar: Status: 200 OK, Time: 1382 ms, Size: 836 B, Save as example, more options.
- Test Results section:
  - Filter buttons: All, Passed, Skipped, Failed.
  - Result count: 1/1.
  - Result details: PASS, Successful GET request.
  - Description: 200 OK
  - Description content: Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.

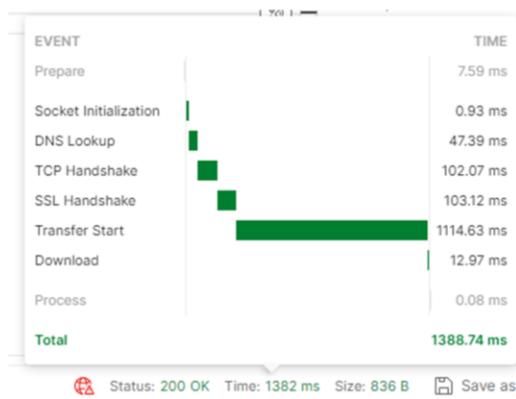
14  
2

### NOTE:

Some API responses also contain **custom messages** that can help you understand response codes. For example, if you receive a **401 Unauthorized** response, the message might tell you to **check the token** you used in the request. If custom messages are returned, they're displayed in the **Body** of the response.

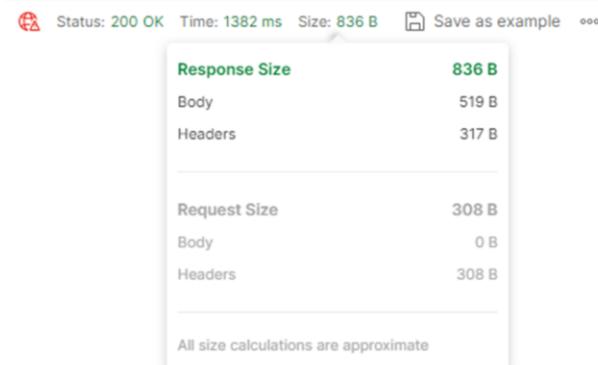
## Response time

- Postman automatically calculates the time in milliseconds it took for the response to arrive from the server. This information can be useful for some preliminary performance testing. Hover over the response time for a graph with information on how long each event in the process took.



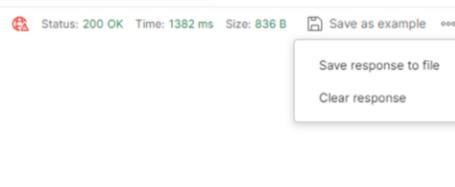
## Response size

- Postman displays the **size of the response**. Hover over the response size to get a breakdown by body and header sizes.



## Saving responses

- If a request has been saved in a collection, you can save responses for that request. Once the response has been returned, you can:
  - Select **Save as Example** to save the response as an example that you can access later.
  - Select the **more** actions icon then **Save response to file** to save the response as a JSON file.
  - Select the **more** actions icon then **Clear response** to remove any data in the response viewer. Note that for event-based requests, this is available after the stream is closed.



14  
5

## Generate code snippets from API requests

- Postman can convert an **API request** into a **code snippet**, and you can choose the **programming language** or framework. You can use this generated code snippet in your front-end applications.

## How to generate code snippet in Postman? (1)

1. Open the request you want to use for a code snippet, then select the **code icon** in the **right pane**.
2. Select a **language or framework** from the dropdown list.
3. Select the **copy icon** to copy the code snippet to your clipboard.

14  
7

### NOTE:

For more configuration options, like the indentation type and count, select the settings icon Settings icon next to the dropdown list. The settings vary based on the chosen language or framework.

## How to generate code snippet in Postman? (2)

The screenshot shows the Postman interface with two requests listed in the sidebar: "GET My First Request" and "GET My First Request". The main area displays the details for the first request, which is a GET method to "https://postman-echo.com/get". The "Code snippet" panel is open on the right, showing various language options. The "cURL" option is selected and highlighted with a red box. A red box also highlights the "Copy" icon in the top right corner of the panel. The URL "://postman-echo.com/" is visible in the background of the panel.

14  
8

## Supported languages and frameworks (1)

- Postman supports the following:

Language	Framework
C	LibCurl
C#	HttpClient
C#	RestSharp
cURL	cURL
Dart	Dart
HTTP	(Raw HTTP request)
Java	OkHttp
Java	Unirest
JavaScript	Fetch
JavaScript	jQuery
JavaScript	XHR
NodeJS	Axios
NodeJS	Native
NodeJS	Request

## Supported languages and frameworks

- Postman supports the following:

Language	Framework
NodeJS	Unirest
Objective-C	NSURLSession
OCaml	Cohttp
PHP	cURL
PHP	Guzzle
PHP	Http_Request2
PHP	pecl_http
PowerShell	RestMethod
Python	http.client (Python 3)
Python	Requests
R	httr
R	RCurl
Ruby	NET::Http
Shell	Httpie
Shell	wget
Swift	URLSession

15  
0

## Debug API requests using the Postman Console

- Every request sent by Postman is logged in the **Postman Console**, so you can view the detail of what happened when you sent a request. This means you can use the Console to help **debug** your requests when an API isn't behaving as you expect. Keeping the Console open while you work will increase the **visibility** of your **network calls** and **log messages** while debugging.
- The Postman Console logs the following information:
  - The primary request that was sent, including all underlying **request headers**, **variable values**, and **redirects**.
  - The **proxy configuration** and **certificates** used for the request.
  - **Network information** such as **IP addresses**, **ciphers**, and **protocols used**.
  - **Log statements** and **asynchronous requests** from **test** or **pre-request scripts**.
  - The **raw response** sent by the server before it's processed by Postman.

## Opening the Console

- Open the **Console** by selecting  in the Postman footer. In the Postman desktop app, you can use **Ctrl+Alt+C** to open the Postman Console in a new window.



Filter by log message type under **All Logs**. Select the **more actions icon** to turn **timestamps** and **network information** **on** or **off**.

## Using log statements

- Using log statements at appropriate locations in your test scripts can help you debug your requests. Postman accepts the following log statements:

- `console.log()`
- `console.info()`
- `console.warn()`
- `console.error()`
- `console.clear()`



## Postman Collections



RUBRIC

- In this chapter, you will learn about:
  - What Is A Postman Collection?
  - Creating Postman Collections
  - Exporting/Importing A Postman Collection
  - Exporting Postman Request As Code
  - Importing Postman Collections
  - Executing Postman Collections

## What Is A Postman Collection?

- **Postman collection** is nothing but a **container or folder** for **storing Postman requests**.
- Putting similar requests into folders and collections helps the client in better organization and documentation of their requests.
- These collections can be **shared** amongst the team in the Postman workspace.

## Advantages of creating collections

- Easy APIs import and export
- Request categorisation
- Passing data among requests
- Running collections/folders
- API documentation
- Building Test suites
- Time-saving

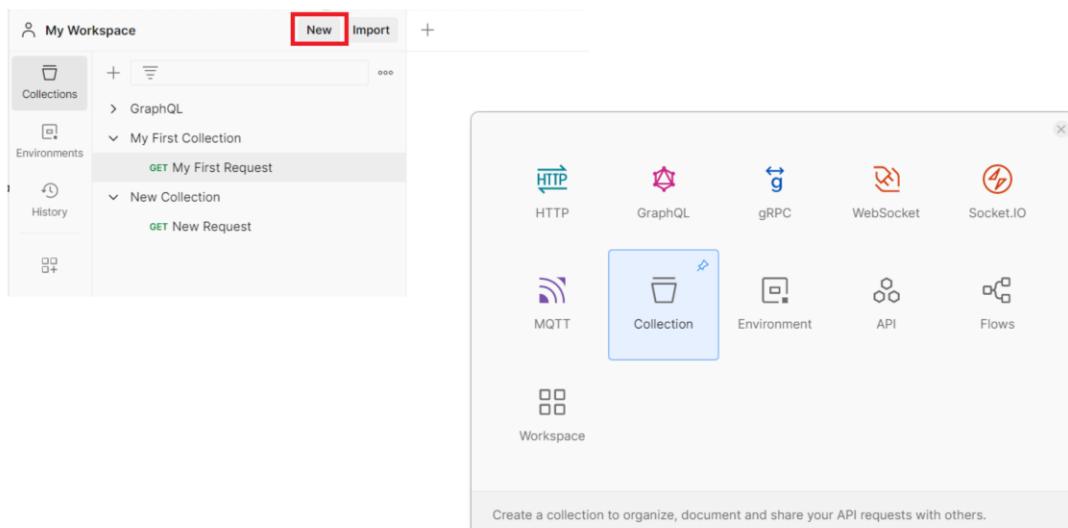
15  
6

- Easy APIs import and export - A collection can be imported or exported. Thus, it can save time for transferring the requests.
- Request categorization - Requests can be grouped into folders and collections for easy access.
- Passing data among requests - Scripts can be used to pass data between API requests.
- Running collections/folders - You can run individual requests or folder or collection in the Postman.
- API documentation - Postman provides you an option to add a name and description to your request, folder, and collections.
- Building Test suites - Test scripts can be added to requests, and integration test suites can be built.
- Time-saving - Setting any variable for collections will automatically apply the same to the folders generated under the collection and requests; thus, it saves time.

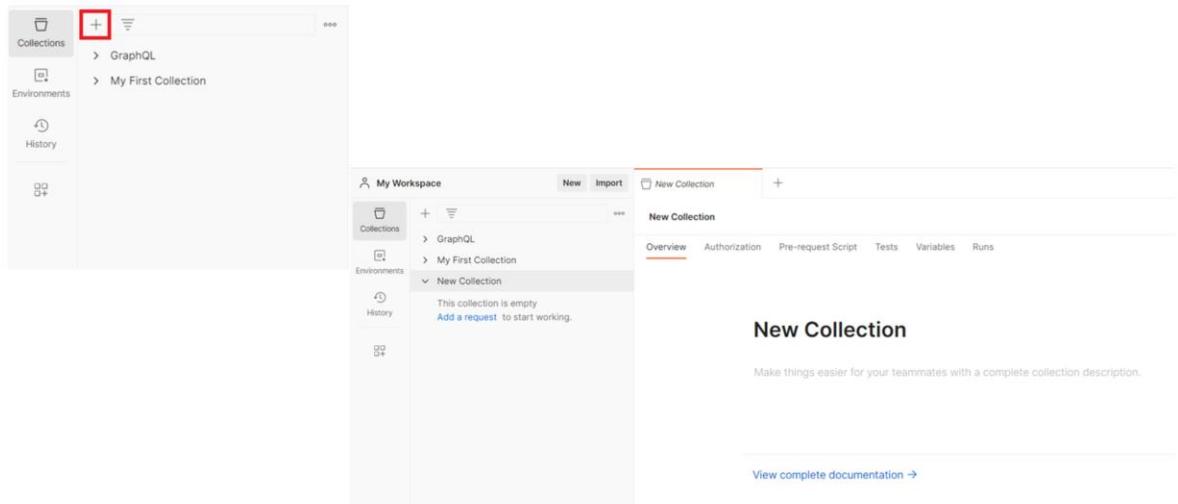
## Creating Postman Collections

- A new collection can be created from the following options in Postman:
  - New button in the sidebar
  - Using + from Collection tab in Sidebar

## Create Collection – Option 1

15  
8

## Create Collection – Option 2



The screenshot illustrates the 'Create Collection – Option 2' process in the RUBRIC interface. On the left, the sidebar includes 'Collections' (highlighted with a red box), 'Environments', and 'History'. The main workspace shows 'My Workspace' with 'GraphQL' and 'My First Collection'. A 'New Collection' button is also highlighted with a red box. The right panel displays the 'New Collection' creation dialog, which includes tabs for 'Overview' (selected), 'Authorization', 'Pre-request Script', 'Tests', 'Variables', and 'Runs'. The 'Overview' tab contains a placeholder message: 'This collection is empty' and 'Add a request to start working.' Below the tabs, there's a link to 'View complete documentation →'.

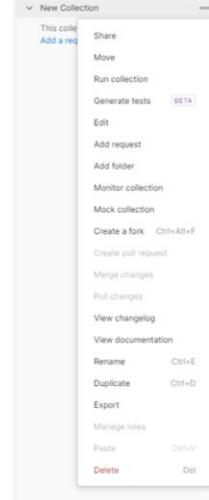
15  
9

### NOTE:

- The Collections Tab should be open to be able to create collection using this option.
- Upon creating the of the collection, a new tab is opened with these different tabs:
  - Overview
  - Authorization
  - Pre-request Script
  - Tests
  - Variables
  - Runs

## Managing Postman Collections

- To manage postman collections, select the collection and click on the More icon  . You will get the following:
  - Share
  - Move
  - Run Collection
  - Generate tests
  - Edit
  - Add Request
  - Add folder
  - Monitor collection
  - Mock Collection
  - .....



## Basic actions

- From the list of collections in your workspace, you can:
  - Select a collection to open its overview in a tab.
  - Open and close collection content by selecting the arrow next to the collection name.
  - Hover over a collection name and select the star icon ★ to move that collection to the top of the list.
  - Use the collection search bar to filter through your collections.
  - Reorder the requests, folders, and examples inside a collection by dragging and dropping them.

## Adding folders to a collection

- Select the more actions icon  next to the collection name.
- Select **Add folder**.
- You can also add **sub-folders** to create extra levels of nesting for the collection's requests and examples.

## Deleting a collection

- Select the more actions icon  next to the collection name.
- Select **Delete**.
- You can also select the collection and press **Delete** on your keyboard.

16  
3

### NOTE:

If a deleted collection is larger than 30 MB, you won't be able to recover it. To avoid this, you can split the collection into smaller pieces before deleting it, or back up the collection first.

## Recovering a deleted collection

- Select the more actions icon  next to the collection name.
- Select **Open Trash**.
- In the **Trash page**, select the **restore** icon icon next to the collection you want to recover.

16  
4

### NOTE:

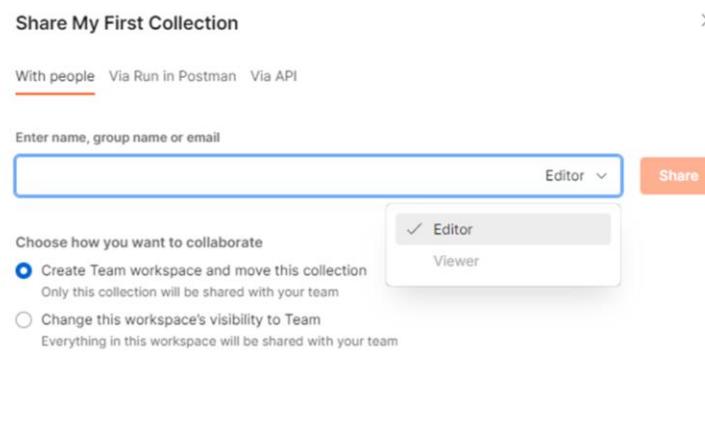
- Alternatively, select **Trash** from the Postman footer.
- Recovery options depend on your Postman plan:
  - Free account: Recover collections for up to one day.
  - Postman Basic: Recover collections for up to 30 days.
  - Postman Professional and Enterprise: Recover collections for up to 90 days.

## Sharing a collection

- There are 3 options that collections can be shared:
  - Share to a Workspace
  - Share via Run in Postman button
  - Share via API

## Share to a Workspace

- Select the more actions icon  next to the collection name.
- Select Share.



16  
6

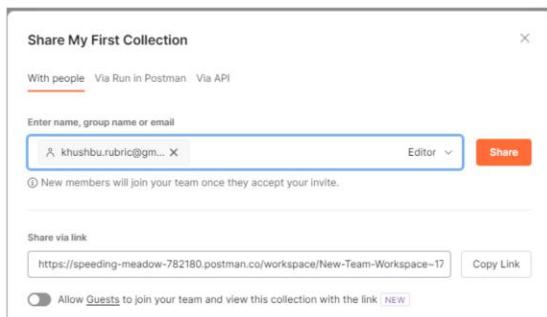
### NOTE:

The above screenshot is mainly for a first-time user.

- Adding a name or group name or email and sharing will create a new workspace and will move the collection to that workspace. (*If the Create Team workspace and move this collection option is checked*)
- Or you can also change the visibility of your workspace and then invite team members to the workspace.
- You can also specify which right you want to give the team member, either editor or viewer.

## Share to a Workspace – Team Workspace

- Select the more actions icon  next to the collection name.
- Select **Share**.

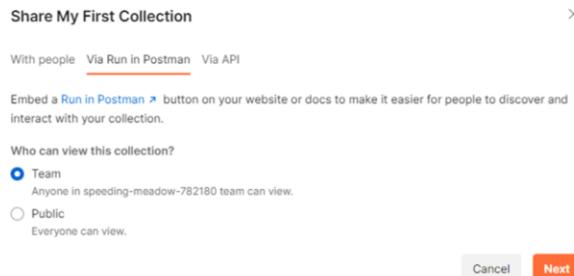
16  
7

Allow external users who aren't in your Postman team to view collections by selecting the more actions icon next to the collection name, selecting **Share**, then turning on the toggle next to **Allow Guests to join your team and view this collection with the link**.

Upon clicking **Share**, the person will get a mail to accept the invite for the workspace and by accepting the invite, the person can see the workspace in his/her list of workspaces. Underneath the workspace the person will also get to see the collection that has been shared to him/her.

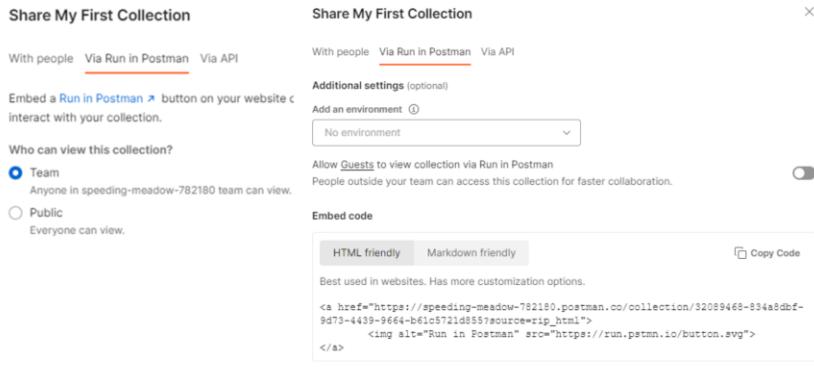
## Share via Run in Postman button

- One way to share your Postman collections is to create a **standalone Run in Postman button**. The **Run in Postman** button allows users to fork the collections. You can **embed** the button in your website or a **README** to let developers interact with your API more quickly.



## Creating a Run in Postman button (1)

- Select the more actions icon  next to the collection name.
- Select **Share**.
- Select the **Via Run in Postman** tab.



### NOTE:

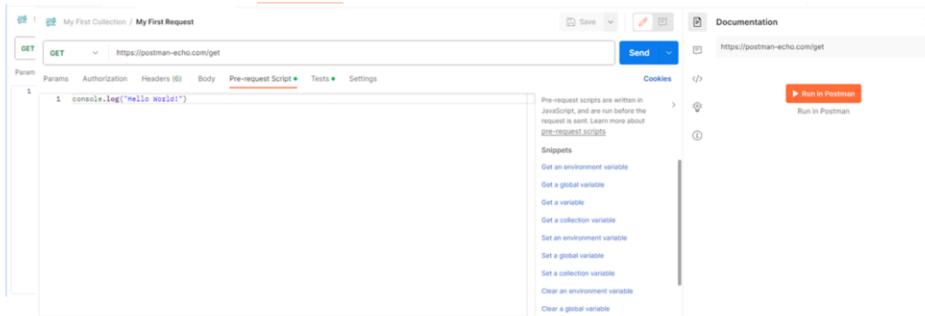
You can optionally include an environment to embed with your collection. Select the Add an environment dropdown list and choose the environment.

## Creating a Run in Postman button (2)

- Select **Copy Code** icon.



- Embed the code where you would like the button to display.



The screenshot shows the Postman interface. On the left, there's a "My First Collection : My First Request" screen with a GET request to "https://postman-echo.com/get". The "Pre-request Script" tab is selected, containing the code: "1 console.log('Hello world!')". On the right, there's a "Documentation" panel for the same endpoint. At the bottom of this panel is a red "Run in Postman" button.

17  
0

Your embedded code will include your collection's ID.

In the examples below, `:collection_id` is a placeholder for that ID and `:collection_url` is a placeholder for url. If you choose to include an environment in your button, the code will also have the environment parameter.

**Sample Markdown snippet:**

`[](https://god.gw.postman.com/run-collection/:collection_id)`

Select the Run in Postman button to open the page where you can fork the collection to your workspace. Forking the collection into your workspace will enable you to contribute to the source collection using pull requests. You can also view the collection in a public workspace if you like and even import a copy of the collection using the links present on the screen. All collections shared with the new Run in Postman buttons come with Fork counts, that help you and your consumers understand how developers use the API.

## Share via API

### Share My First Collection

With people Via Run in Postman Via API

#### Get collection JSON using Collection Access Key

Grant read-only access to this collection via the Postman API. The collection JSON served via this endpoint always reflects the current state of the collection. [Manage keys](#)

##### Generate key?

This will create a publicly-accessible link. Make certain that your collection does not contain any sensitive information before you share this link.

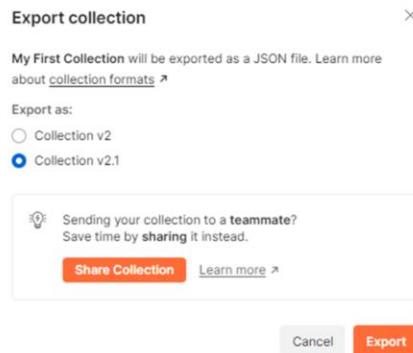
[https://api.postman.com/collections/32089468-834a8dbf-9d73-4439-9664-b61c5721d855?access\\_key=PMAT-01HCKMH7RJ1J77DFQBXEEK608CX](https://api.postman.com/collections/32089468-834a8dbf-9d73-4439-9664-b61c5721d855?access_key=PMAT-01HCKMH7RJ1J77DFQBXEEK608CX) 

#### Link to collection in public workspace

You can also import the collection in another workspace via URL.

## Exporting a Postman Collection (1)

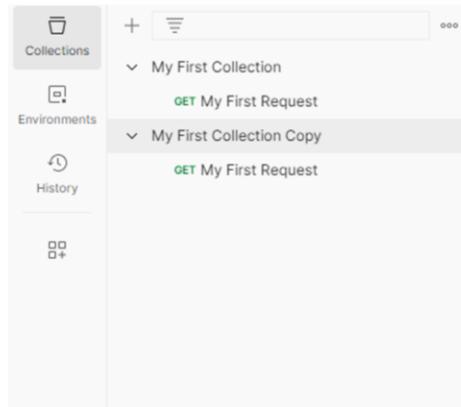
- Select the more actions icon  next to the collection name.
- Select **Export**.
- Select the Collectionv2.1 format for export option.



A JSON file is downloaded in your local directory.

## Duplicating an existing collection

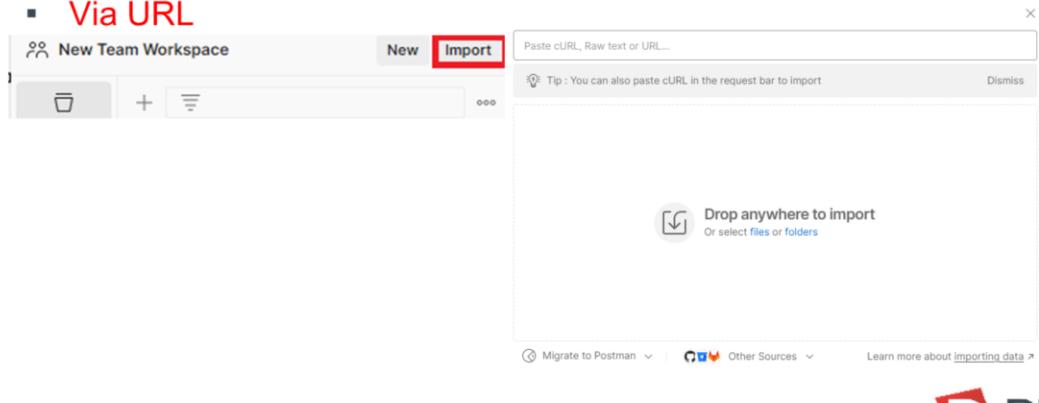
- Select the more actions icon  next to the collection name.
- Select **Duplicate**.



17  
3

## Importing a Postman Collection

- You can import a postman collection by selecting Import from the sidebar.
- There are 2 ways a postman collection can be imported:
  - Via JSON file
  - Via URL

17  
4

## Executing Postman Collections via Runner

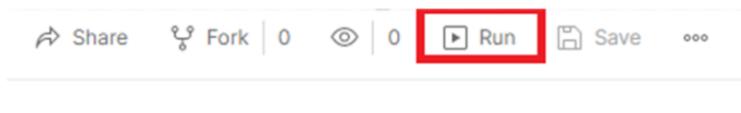
- The **Collection Runner** enables you to run a collection's requests in a **specified sequence** to test the functionality of your API. It **logs** your request test results and can use scripts to pass data between requests and alter the request workflow.
- You can configure the Collection Runner to meet your development needs. You can run collections using a **specific environment** and can **pass data files** into a run. Collection runs enable you to **automate** your functional API testing.

17  
5

By default, your requests run in the sequence they're listed in the collection. If you need to change the order of execution, select and drag a request to its new location in the order. You can also remove an individual request from the run by clearing the checkbox next to its name.

## Configuring a collection run (1)

- Select **Collections** in the sidebar and select the **collection** or folder you want to run.
- On the **Overview** tab, select **Runner icon**  .

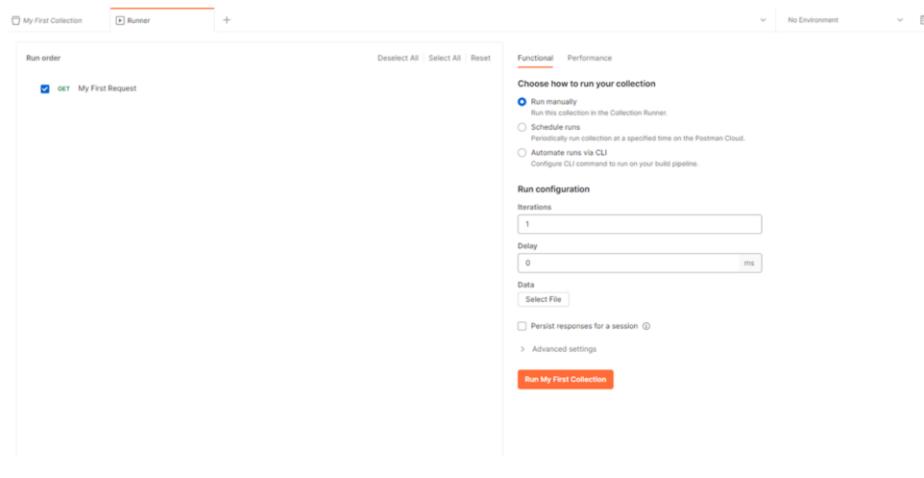
17  
6

Another way to run the collection:

- Select the more actions icon next to the collection name.
- Select **Run collection**.

## Configuring a collection run (2)

- On the **Functional** tab, select **Run manually**.

17  
7

You can also schedule runs and automate runs with the CLI.

If you want your collection to run with an **environment**, select it using the **environment selector** at the top right of Postman. You can also select **Environments** in the sidebar, then select the environment you want to use.

## Configuring a collection run (3)

- **Choose any configuration options:**
  - **Iterations** - The number of iterations for your collection run. You can also run collections multiple times with different data sets to build workflows.
  - **Delay** - An interval delay in milliseconds between each request.
  - **Data** - A data file for the collection run.
  - **Persist responses for a session** - Log the response headers and bodies so you can review them after running the collection. For large collections, persisting responses may affect performance.

17  
8

Advanced settings:

**Stop run if an error occurs** - By default, the collection run stops if an exception is encountered within a script or if there's a problem sending a request. Clear this checkbox if you want the collection run to continue after an error occurs.

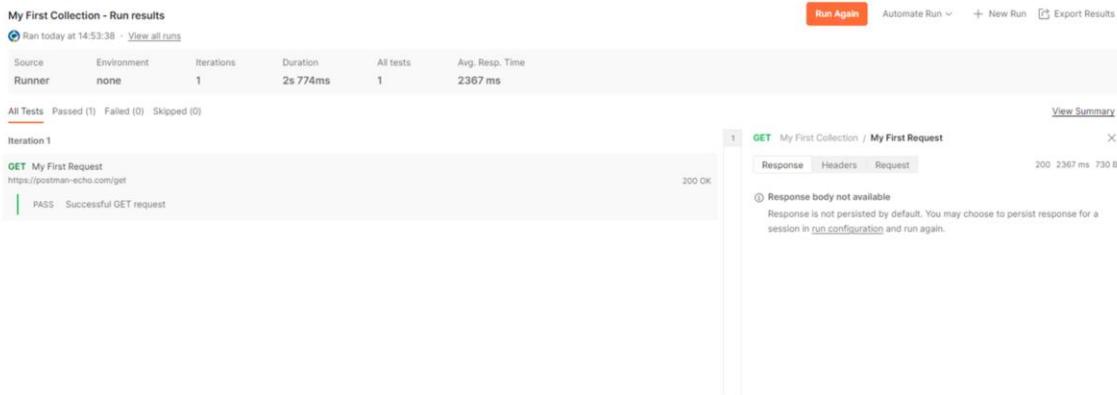
**Keep variable values** - Persist the variables used in the run, so that any variables updated by the run will remain changed after it completes. If you don't persist variables, changes aren't saved after the run completes. Note that persisting variables in the collection run will update the current value only.

**Run collection without using stored cookies** - If your requests use cookies, you can optionally deactivate them for a collection run.

**Save cookies after collection run** - Save the cookies used in this session to the cookie manager. Any values changed by requests during the run will remain after it completes.

## Executing Runner

- When you've completed your configuration, select **Run (collection name)**.



17  
9

When running collections **manually**, Postman displays the results of your request executions and test results in real time. You can view the source of the **collection run**, **selected environment**, **number of iterations**, **total duration**, **number of tests**, and **average response time**.

To learn more about what happened during the collection run, do any of the following:

- Select a request to view details about the request. You can view general information about the request and the request headers and body. You can also view the response headers and body if you selected the Persist responses for a session option when configuring the collection run.
- Select the name of a request to open the request in a new tab. You can view any test scripts or select Send to send the request again.
- Select the Passed, Failed, or Skipped tabs to filter the results by test status. To show all requests, select the All Tests tab. If any tests in a request script fail during a collection run, the whole request fails.
- If your collection run included more than one iteration, select an iteration number to jump to the results of a specific iteration.
- Select **View all runs** to view a list of past runs. Learn more about viewing run history.
- Select **View Summary** to view a summary of the collection run, including test results. To return to the full results, select View Results.

## Running the collection again

- After reviewing the results of the collection run, you can run the collection again. For example, you can edit the code for a failed test and run the collection again to check if the test succeeds.
- To run the collection again from the run results, do one of the following:
  - Select **Run Again** to run the collection again using the same settings.
  - Select **+ New Run** to configure a new run for the collection. Make changes to any settings, and then select **Run (collection name)** to run the collection again.



## Sharing collection runs

- You can share collection run results with others by **exporting** the results from the Collection Runner.
- To export a collection run, do the following:
  - Open the collection run in the Runner. You can also access the collection run using History in the sidebar if you don't have the run open.
  - Select **Export Results** at the top right to download the run.
  - Choose a location to save your downloaded collection run, then select **Save**.



18  
1

### NOTE:

The Export Results button is **available** in the **Postman desktop app** but not in the web version.

## Exercise:

- The PetStore API (Swagger Docs) provides endpoints to manage **pet-related data, store orders, and user management**.
- Task:
  - You are required to design a **comprehensive regression test pack** in Postman using **collections** and **folders** to verify the **functionality** of the PetStore API.
  - Your test pack should include both **positive and negative test scenarios** to ensure the API works correctly under different conditions.

**RUBRIC**18  
2

## Understanding Variable Scopes And Environment Files In Postman

- In this chapter, you will learn how to use Postman:
  - What Are Variables In Postman?
  - Variable Types In Postman
  - Global
  - Collection
  - Environment
  - Local
  - Data
  - Accessing Postman Variables

## What are variables?

- Variables, like any other programming language, are nothing but placeholders to hold some value or result of some expressions.

**Var testVar = "hello world"**

- To use Postman variables as a part of the request builder, simply use the double curly brace syntax as shown below:

**{{testVar}}**

- For example:

- Consider a variable declaration in C#/Javascript that holds string value "hello world!". Now whenever this string is required, someone can just use the name of the variable to get the value replaced as the actual string.



## What are variables in Postman?

- **Variables here are typically used in a context where:**
  - you need to replace values in request bodies
  - make assertions for the response body
  - use them in pre-request scripts to handle pre-processing logic
- **Variables scopes in Postman:**
  - Scope is lifetime and accessibility of a variable.
  - For Example, a variable can have a global scope.

18  
5

Firstly, let's try to understand different variable scopes in Postman. Scope is nothing but the lifetime and accessibility of a variable. It's similar to the concept of scope that we have in programming languages like Java.

For example, a variable can have:

- a global scope i.e. any class/method can access/modify the variable until the method or program is active.
- Similarly, there can be various types that have a specific scope as local variables declared in the method can be used only till the time, that function is executing.
- Likewise, variables declared in for loops can only be used while that loop is executing.

## Types of variables in Postman

- Global
- Collection
- Environment
- Data
- Local

18  
6

Let's try to understand each of these variable types and scope in detail!!

## Global variables

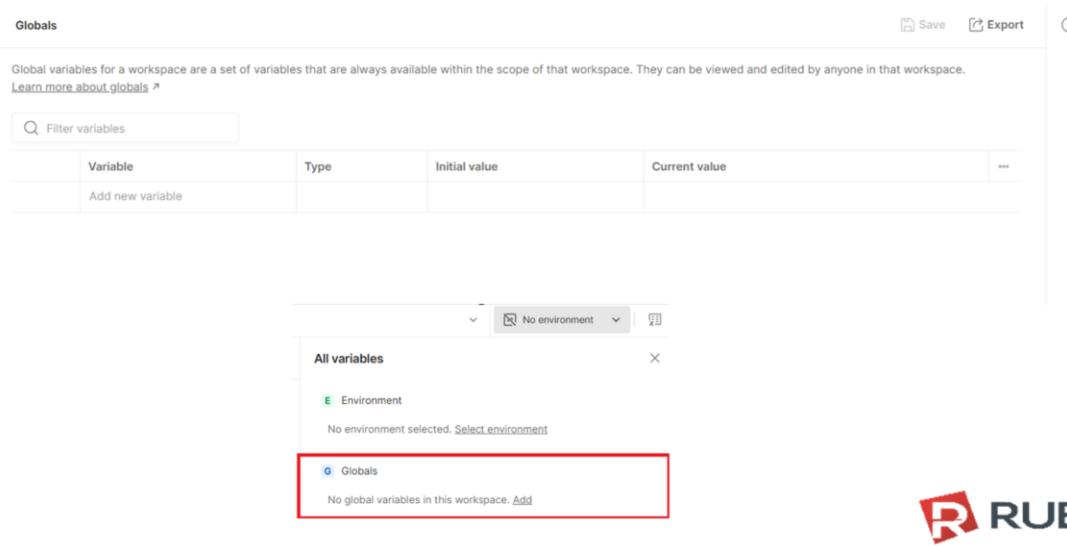
- **Global variables** enable you to access data between **collections**, **requests**, **test scripts**, and **environments**. Global variables are available throughout a workspace. Since global variables have the broadest scope available in Postman, they're well-suited for testing and prototyping. In later development phases, use more specific scopes.
- However, these variables are highly unreliable as each section of code can access/modify the global variable value.

18  
7

Global are general-purpose variables and are available to all requests in Postman console, irrespective of what collection they belong to.

## Create Global variables

- Global variables can be created and managed from the **Environment selector** or **Environment element tab** from the sidebar.



The screenshot shows the RUBRIC interface with the following elements:

- Top Bar:** Save, Export, and Help icons.
- Section Header:** Globals.
- Description:** Global variables for a workspace are a set of variables that are always available within the scope of that workspace. They can be viewed and edited by anyone in that workspace. [Learn more about globals](#).
- Search Bar:** Filter variables.
- Table:** A grid for managing variables, with columns: Variable, Type, Initial value, Current value, and ... . A button "Add new variable" is at the bottom left.
- Environment Sidebar:** Shows "No environment" selected. A dropdown menu lists "All variables" and "E Environment". Below it, a red-bordered box highlights the "G Globals" section, which displays "No global variables in this workspace. [Add](#)".
- Bottom Right:** RUBRIC logo.

## Accessing Global variables via Scripts

- Using the script, global variables can be accessed using “**pm.globals**”.
- Here are a few command to manipulate global variables:
  - Use the below command to get the value of a global variable named ‘testVar’.  
**pm.globals.get('testVar');**
  - Use the below command to set the value of a global variable named ‘testVar’.  
**pm.globals.set('testVar', 'Hello Postman tutorial!');**
  - If you want to remove a global variable through script, then you can use the unset function as mentioned below.  
**pm.globals.unset('testVar');**
  - To clear all the global variables through the script, you can use the below function.  
**pm.globals.clear();**

## Collection Variables

- **Collection variables** are available throughout the **requests in a collection** and are **independent of environments**. Collection variables **don't change** based on the selected environment. Collection variables are suitable if you're using a single environment, for example for auth or URL details.
- Using the script, collection variables can be accessed using "**pm.collectionVariables**"

19  
0

Collection variables are used to define variables at the collection scope. As we know, a collection in Postman is nothing but a group of Postman requests.

Collection variables do not change during the execution of a collection or request inside the given collection.

Essentially Collection variables could be just retrieved and not updated during request execution.

## Create Collection variables

- Select Collections in the sidebar.
- On the collection opened tab, select Variables tab.



My First Collection

Overview Authorization Pre-request Script Tests **Variables** Runs

These variables are specific to this collection and its requests. Learn more about [collection variables](#)

Variable	Initial value	Current value	...
name	John Doe	John Doe	...

19  
1

For using inside a request, collection variables can be referred using generic Postman script and depending upon the rule of closest scope,

if there is no other variable which is closer than the collection scope, then the collection variable gets returned.

## Environment Variables

- **Environment variables** enable you to scope your work to **different environments**, for example local development versus testing or production. One environment can be active at a time. If you have a single environment, using collection variables can be more efficient, but environments enable you to specify role-based access levels.
- Environment variables are the most heavily used kind of variables in Postman. They have a narrower scope than the Global variables but broader than the Collection variables.
- Using the script, environment variables can be accessed using “**pm.environment**”

19  
2

When there is a need for passing data or information from one request to another, environment variables are a good choice

## Local variables

- **Local variables** are temporary variables that are accessed in your request scripts. Local variable values are scoped to a single request or collection run and are no longer available when the run is complete. Local variables are suitable if you need a value to override all other variable scopes but don't want the value to persist once execution has ended.
- Using the script, local variables can be accessed using “**pm.variables**”

19  
3

This is in analogy to function level variables in modern programming languages like Java and C#.

These are exactly similar to Function variables in which the variables just have access within the context of the function execution.

One important use case of local variables is that they can be used when you want to override the values of a variable that is defined in any other scope like Global, Collection or Environment.

i.e. Suppose there's a global variable named 'testVar' and now you want to override the value without affecting the Global variable,  
you can create a local variable with the same name and use it with the value you like

## Data variables

- Data variables come from external CSV and JSON files to define data sets you can use when running collections with Newman or the Collection Runner. Data variables have current values, which don't persist beyond request or collection runs.
- Data variables can only be fetched but not updated/modified or added.
- Data variables are used while we are working with multiple data-sets and they exist only during the execution of an iteration from the data file.

## How To Use a Data Variable? (1)

- Change the request URL in the Postman request builder to <https://reqres.in/api/users/{{testDataVar}}>. Here, {{testDataVar}} is our data variable whose value will be fetched from the data source that will be used with the collection runner.
- Create a new data file in CSV format with the column name as “**testDataVar**” with values **1 – 5**
- Save the file onto the local file system and use this file while running the collection through the collection runner.

## How To Use a Data Variable? (2)

Scripts - Run results

Ran today at 20:31:18 · View all runs

Run	Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner		none	5	2s 201ms	0	121 ms

All Tests Passed (0) Failed (0) Skipped (0) [View Summary](#)

**Iteration 1**

**GET** Get User Copy  
https://reqres.in/api/users/1

No tests found

**Iteration 2**

**GET** Get User Copy  
https://reqres.in/api/users/2

No tests found

**Iteration 3**

**GET** Get User Copy  
https://reqres.in/api/users/3

No tests found

**Iteration 4**

**GET** Get User Copy  
https://reqres.in/api/users/4

No tests found

**Iteration 5**

**GET** Get User Copy  
https://reqres.in/api/users/5

No tests found

1  
2  
3  
4  
5

200 OK 137 ms 1.201 KB  
200 OK 116 ms 1.201 KB  
200 OK 115 ms 1.199 KB  
200 OK 114 ms 1.205 KB  
200 OK 124 ms 1.213 KB

19  
6

## Initial and current values

- **Initial value** is a value that's set in the element (collection, environment, or globals) where the variable is defined. This value is synced to Postman's servers and is shared with your team when you share that element. The initial value can be useful when collaborating with teammates.
- **Current value** is used when sending a request. These are local values and aren't synced to Postman's servers. If you change a current value, it won't be persisted in the original shared collection, environment, or globals.

19  
7

### NOTE:

Setting an initial value can be useful when sharing elements, but it's important to remember that sensitive data in an initial value will also be shared with others, and potentially with the world. Be careful setting initial values and consider using secret variables to mask sensitive data.

## Variable types

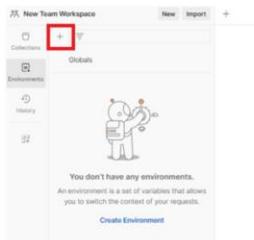
- Beyond scope, global and environment variables can also be defined by type. The two variable types that you can configure for global and environment variables are:
  - Default type is automatically assigned to variables. This type is shown as plain text and doesn't have extra properties.
  - Secret type masks the initial and current values for all workspace members and can be used to prevent unintentional disclosure of sensitive data, including API secrets, passwords, tokens, and keys.

## Group sets of variables in Postman using environments

- In Postman, an **environment** is a set of one or more variables that you can reference when sending requests or writing test scripts. You can create environments for the different types of work you do in Postman. When you switch between environments, all of the variables in your requests and scripts will use the values from the current environment.
- This is helpful if you need to use different values in your requests depending on the context, for example, if you're sending a request to a test server or a production server.

## Create an environment

- Select **Environments** in the sidebar and select **+**.



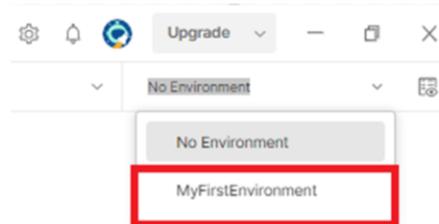
- Enter a name for your new environment.
- Add any variables you want to the environment. You can also add variables later.
- Select Save icon Save to save any environment variables you added.

### NOTE:

You can also create environment using the **New** option from the sidebar.

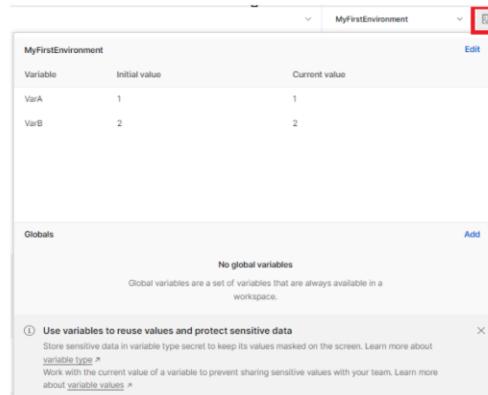
## Use the environment (1)

- To use the new environment, select it from the **environment selector** at the top right of the workbench. This makes it the **active** environment and sets all variables to the values specified in the environment.



## Use the environment (2)

- To check a variable's value at a glance, select the **environment quick look icon** next to the environment selector. The environment quick look lists the initial and current values for all variables in the **active** environment.



The screenshot shows the 'MyFirstEnvironment' environment quick look interface. It displays two variables: VarA (Initial value: 1, Current value: 1) and VarB (Initial value: 2, Current value: 2). Below this, there is a 'Globals' section with a note stating 'No global variables'. A tooltip provides information about using variables to reuse values and protect sensitive data.

Variable	Initial value	Current value
VarA	1	1
VarB	2	2

Globals

No global variables

Global variables are a set of variables that are always available in a workspace.

ⓘ Use variables to reuse values and protect sensitive data

Store sensitive data in variable type secret to keep its values masked on the screen. Learn more about variable type.

Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values.

## Manage environment

- To edit an environment, select Environments in the sidebar and select an environment. From here you can take the following actions:
  - To rename an environment, select the environment's name and enter a new name.
  - To duplicate an environment, select the **more actions icon** and select **Duplicate**.
  - To delete an environment, select the **more actions icon** and select **Delete**. Deleting an environment also deletes all variables in the environment.

### Exercise:

- In the previous task, you created a Postman regression test pack for the PetStore API, covering both positive and negative scenarios. Now, you need to enhance your test pack by using **variables** to make it **dynamic** and **reusable**.

20  
4

## Scripting in Postman



**RUBRIC**

20  
5

- In this chapter, you will learn how to use Postman:
- What are scripts in postman.?
- Execution order of scripts
- Pre-Request Script
- When to use Pre-Request Scripts?
- Postman Request Flow
- Using Pre-Request & Post-Request Scripts with Collections

## What are Scripts In Postman? (1)

- Scripts are a piece of code written in **Javascript** and are executed at specific points in your test Lifecycle. They are used in Postman to enable **dynamic behaviour** to request and collections.
- It allows you to write **tests**, **change parameters** and even **pass data** between the requests.
- A script can be added to the **request**, **collection**, **folder** or an independent request
- Scripts in Postman are written in **Postman Sandbox**, a runtime based on **Node.js**



## What are Scripts In Postman? (2)

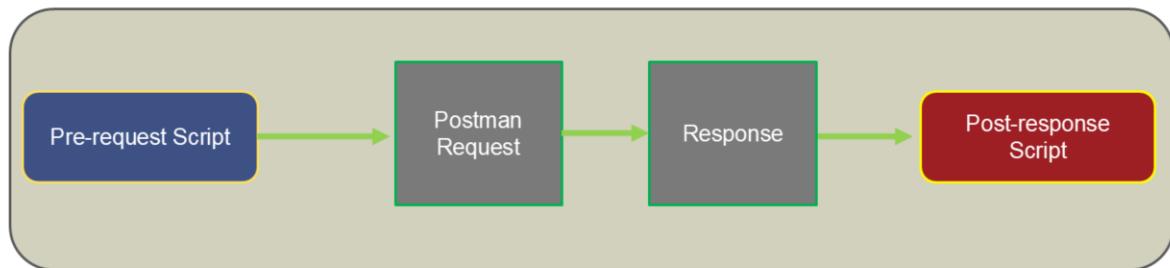
- You can add JavaScript code to execute during two events in the flow:
  - Before a request is sent to the server, as a **pre-request script**.
  - After a response is received, as a **post-response script**.



- The **Pre-request** and **Post-Response** are found under the **Scripts** tab.

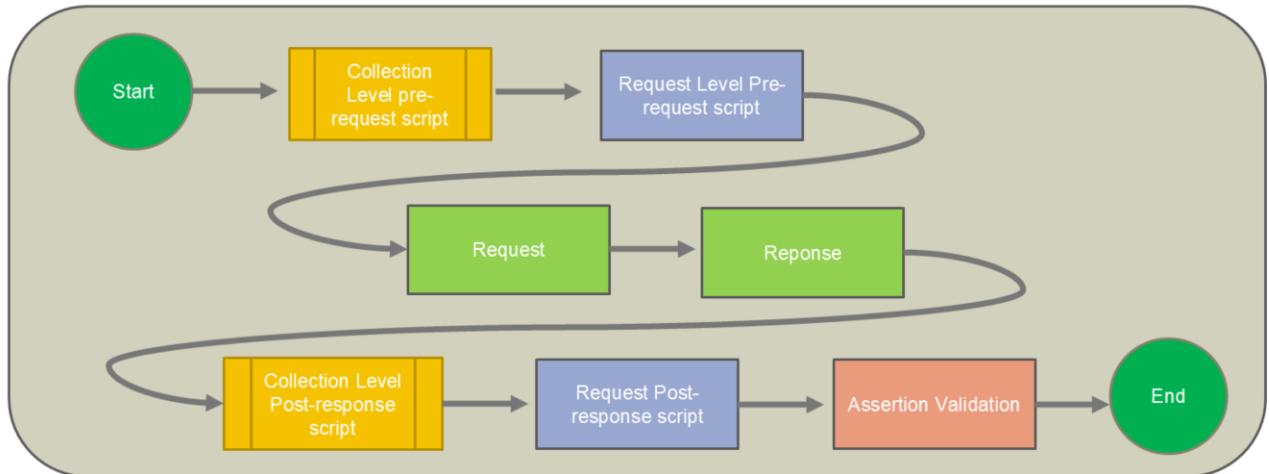
## Execution order of scripts (1)

- Let's try to understand how Postman enables or allows the pre and test scripts to get executed in the context of request execution.



## Execution order of scripts (2)

- Please refer to the below figure to see the Request flow when the collection level pre-request script and post-response script are there.



## Pre-Request Scripts

- An example usage of pre-request scripting could be as follows:
  - You have a series of requests in a collection and are running them in a sequence, such as when using the collection runner. The **second request** is dependent on a **value returned** from the **first request**. The value needs to be **processed before** you pass it to the second request.
  - The first request **sets** the data value from a **response field** to a variable in its **Post-response** script.
  - The second request **retrieves** the value and processes it in its **Pre-request Script**, then **sets** the processed value to a variable (which is referenced in the second request, for example in its parameters).

21  
0

Suppose your request expects a security token that needs to be retrieved from a third-party server and as this value changes with every request execution, it could not be persisted with the environment/global variables as well.

Request header expecting a session ID that needs to be randomly generated and needs some other conversions like base 64 encoding or processing in general.

## Write Pre-Request Scripts (1)

- We will be using the following GET API endpoint:  
`https://reqres.in/api/users/{{randomVal}}`
- Here {{randomVal}} is a random integer between 1 to 10 that would be calculated in the pre-request script.
- Script:

```
var random = Math.floor(Math.random() * 10);
pm.variables.set('randomVal',random)
```

21  
1

Let's see an example of how to use the pre-request script in this case:

### Here is the script flow:

- Add logic in the pre-request script tab. Generate a random number between 1 to 10.
- `var random = Math.floor(Math.random() * 10); pm.variables.set('randomVal',random)`
- Store the generated random number in an environment or local variable. In the above code snippet, you can see that we have generated a random value between 1 and 10 and stored it in a local variable named 'randomVal'.
- Use the environment variable as a part of the request body.
- Execute the request.
- Validate the result. You can try hitting the request multiple times and see the request getting hit for different values of userIds that got generated through the random variable pre-script.

## Write Pre-Request Scripts (2)



GET https://reqres.in/api/users/{{randomVal}}

Params    Authorization    Headers (7)    Body    **Scripts**    Settings

Pre-request

```
1 var random = Math.floor(Math.random() * 10); pm.variables.set('randomVal',random)
2 |
```

Post-response

21  
2

## Pre-Request Scripts Example (1)

- We will use a Postman collection with 2 requests with the following test endpoints.
  - **GET** https://reqres.in/api/users/{{randomVal}}
  - **POST** https://reqres.in/api/register
- We will add the following collection level scripts for these requests
  - Pre-request: Add a new header named “Content-Type” and set the value of the header to “application/json”.

```
pm.request.headers.add({  
    key: 'Content-Type', value:'application/json'  
})
```

## Pre-Request Scripts Example (2)

- Now, let's try running the requests inside the collection and see.

Scripts - Run results

Ran today at 19:29:52 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	1s 294ms	0	457 ms

All Tests Passed (0) Failed (0) Skipped (0)

Iteration 1

**GET Get User**  
https://regres.in/api/users/4  
No tests found

**POST Register User**  
https://regres.in/api/register  
No tests found

1 GET Scripts / Get User

Response Headers Request

200 488 ms 1.195 KB

General

URL: https://regres.in/api/users/4  
Method: GET  
Status Code: 200 OK

Request Headers

```
Content-Type: application/json
User-Agent: PostmanRuntime/7.36.0
Accept: */*
Postman-Token: 1948addc-cf84-4df6-a9a8-fecfcfa1022d
Host: regres.in
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Response headers not available

Response is not persisted by default. You may choose to persist response for a session in run configuration and run again.

21  
4



## Automating Response Validation With Assertions In Postman

217



**RUBRIC**

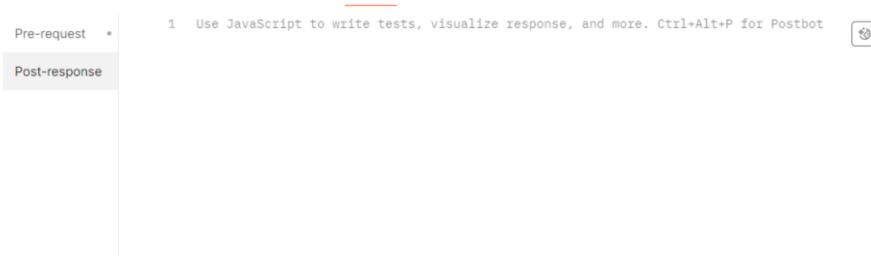
21  
9

## Automating Response Validation With Assertions

- In this chapter, you will learn about:
- Creating The First Assertion
  - The Postman Sandbox
  - The 'pm' Object
  - Tests For Postman Request
- Advanced Assertions -Validating A JSON Field In Response
- Tips & Notes
- Pros & Cons Of Assertions

## The Postman Sandbox

- The Postman Sandbox is a JavaScript execution environment that's available to you while writing pre-request and post-response scripts for requests (both in Postman and Newman).
- The code you write in these sections runs in this sandbox.

21  
9

## The 'pm' Object

- pm is the object that contains the script that's running in the Postman sandbox. It allows you to get request context and response context as well once the request is executed.
- It allows you to access requests and responses as read-only objects and helps in using different kinds of Postman variables by getting or setting their values.

22  
0

## Postman Assertions (1)

- An **assertion** is a code written in **Javascript** in Postman that is executed after receiving the response.
- Assertions make Postman a great utility tool to create integration tests for rest API endpoints. Depending on the output of these assertions, we can determine if a test can either **pass** or **fail**.
- Postman uses the **Chai assertion library** for creating assertions. It is based on the **BDD** style of assertions which make it highly readable and understandable.

22  
1

Chai – Please refer <https://www.chaijs.com/api/bdd/>

Chai assertion library is an external javascript library used to write assertions. Compared to what we write directly in javascript, this assertion library needs less time & effort and easy use.

Chai assertion library is available by default in the Postman. So, when you are writing chai assertions, then don't worry about other processes of installation. The main feature of Postman assertions is that they write the tests with English sentences, which is human readable. So, it is very easy to read and user friendly.

## Postman Assertions (2)

- In Postman, we can write the assertion in many ways. One of the simplest ways is the **snippets**, which are nothing but a block of codes that have some unique functions inside it and are available in the postman app.
- Users can easily access the snippets and can get the code inside the tests editor and run the test.

## Postman Assertions (3)

```
pm.expect(response).to.not.equal(null)
```

- In the example above, the assertion is trying to assert that the response is not null. Like this, the Chai library provides a lot of BDD operators.
- Likewise, Postman has a lot of prebuilt scripts to create basic assertions like checking the response to be non-null, validating the HTTP status code of the request, etc.

## Understanding The Postman Sandbox

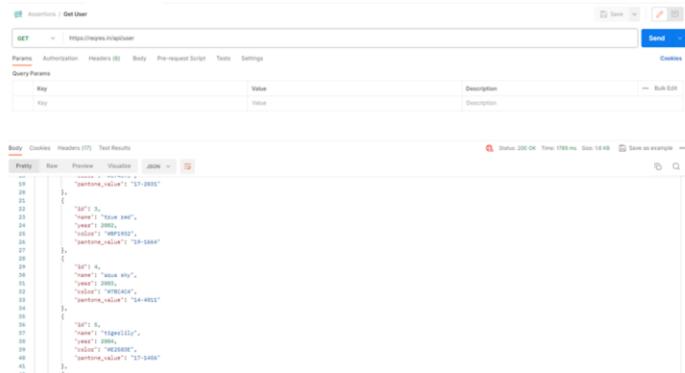
- Postman sandbox has been written in Javascript. It is an execution environment.
- It exposes the Postman object named 'pm' with which we can access various things like:
  - Request and response parameters
  - Set and get environment variable and global variables
  - Get request and response headers
  - Get cookie values

## Tests For Postman Request

- To create an assertion for a given request, you first need to have a test or expectation against which you would assert that.
- You can contrast similar to the Unit test analogy, where we follow the **3As** rule for each test :
  1. **Arrange** is simply the request execution with all the variables like URL, Request path, Request body set properly.
  2. **Act** is actually executing the request through the Postman console (or through any command-line utilities that support such execution)
  3. **Assert** is validating an exception and is done inside the “Tests” tab for a request.

## Writing your first assertion (1)

- Open Postman and create a GET request for the following URL:  
**<https://reqres.in/api/user>**
- Once the request is configured, click **Send** to ensure that the request is correctly set up and you are getting some response.



```

19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
      "id": 1,
      "name": "Leanne Graham",
      "username": "Bret",
      "email": "Sincere@april.biz",
      "phone": "+1-770-736-8292",
      "website": "hildegard.org",
      "company": {
        "name": "Schulist LLC",
        "catchPhrase": "User-centric犹太语单字典",
        "bs": "honestamet"
      }
    },
    {
      "id": 2,
      "name": "Ervin Howell",
      "username": "Antonette",
      "email": "Shanna@melissa.tv",
      "phone": "+1-542-177-0093",
      "website": "anastasia.net",
      "company": {
        "name": "Deckow-Crona",
        "catchPhrase": "Proactiveツイッターリード",
        "bs": "honestamet"
      }
    },
    {
      "id": 3,
      "name": "Clementine Bauch",
      "username": "Samantha",
      "email": "Natalie@kelly.ca",
      "phone": "+1-464-559-5467",
      "website": "moriah.org",
      "company": {
        "name": "Keebler LLC",
        "catchPhrase": "Proactiveツイッターリード",
        "bs": "honestamet"
      }
    }
  ]
}
  
```

Here we will discuss assertions that are primarily based on responses in the format of .json.

In Postman, an assertion can be applied on different attributes such as an object, arrays, etc.

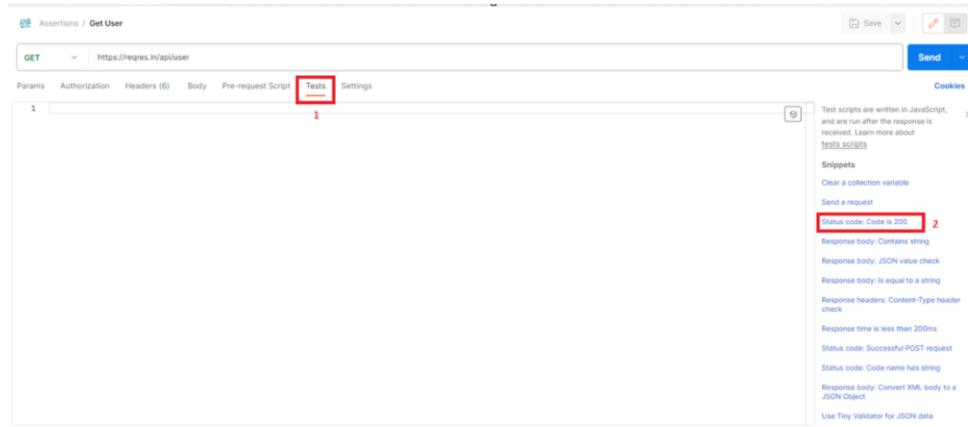
As we know by default in Postman, the received response from the server is displayed in the JSON format, or even we can select the JSON application and get the response converted into the JSON format.

We will be using a sample hosted API for writing our assertion.

**<https://reqres.in/api/users>** is the sample endpoint against which we will write tests and add assertions.

## Writing your first assertion (2)

- Now, let's try adding a test to this request where we will check whether the request return an **HTTP 200 OK Status code**.



The screenshot shows the Postman interface for a 'Get User' request. The 'Tests' tab is selected (highlighted with a red box). A dropdown menu is open, listing various assertions. The 'Status code: Code is 200' option is highlighted with a red box.

22  
7

The first one will be to just check whether the requests return an HTTP 200 OK Status code.

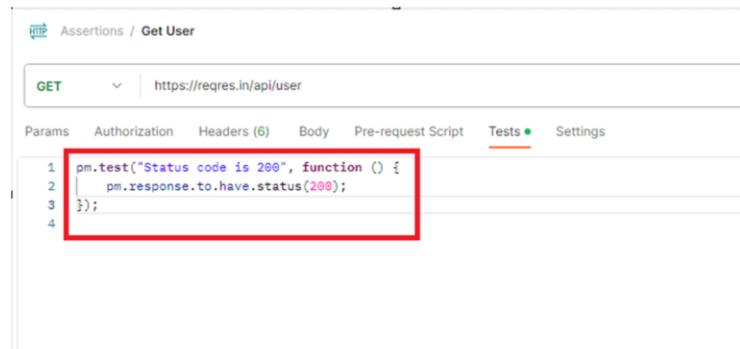
For that we can use Postman's easy templates to validate, else we can simply type-in as below.

Steps:

1. Navigate to Tests tab.
2. Add assertion here by selecting a template from the existing code snippet list.

## Writing your first assertion (3)

- Once the test is added, the following code is displayed:



The screenshot shows the Postman interface with a GET request to <https://reqres.in/api/user>. The 'Tests' tab is selected, displaying the following JavaScript code:

```
1 pm.test("Status code is 200", function () {  
2 | pm.response.to.have.status(200);  
3});  
4
```

## Writing your first assertion (4)

- In the previous code snippet, let's try to break it into 2 components i.e. test and assertion and try understanding each of them one by one.
- **pm.test()** is a closure function that allows you to write tests for a Postman request that's being executed
- **pm.response.to.have.status(200)** is the actual assertion that is trying to validate the response to have a status code of 200.

22  
9

### NOTE:

1. pm.test() Please note for writing multiple tests for the same request, there should be multiple blocks of pm.test() closure or function.

## Writing your first assertion (5)

- Click **Send** to execute the request and check the **Test Result**.

The screenshot shows the Postman interface with a red box highlighting the 'Test Results (1/1)' tab. The results table shows one row with a green 'PASS' status and the message 'Status code is 200'. At the bottom left, there is a red box containing the numbers '23' and '0'.

All	Passed	Skipped	Failed
	PASS	Status code is 200	

Click on the “**Test Results**” tab in the response section and see the test/assertion validation message..

In the example above, the test passed successfully.

## Failing the test (1)

- In this example we will try to fail the test by updating the expectation i.e., **HTTP 202** rather than **HTTP 200**.



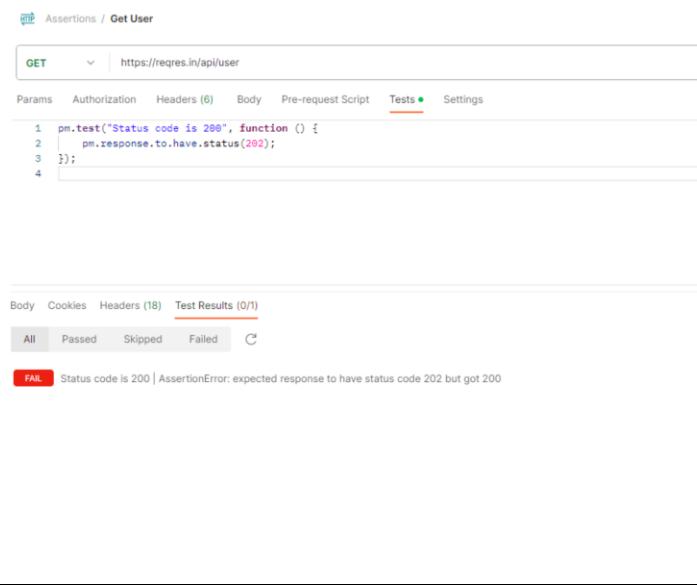
The screenshot shows the Postman interface with an HTTP request to 'https://reqres.in/api/user'. The 'Tests' tab is selected, displaying the following JavaScript code:

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(202);  
3 });  
4
```

23  
1

## Failing the test (2)

- Click **Send** and see the result.



The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** https://reqres.in/api/user
- Tests Tab:** The script contains the following code:

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(202);  
3 });  
4
```
- Test Results:** 0/1 failed. The status is **FAIL**. The error message is: "Status code is 200 | Assertion Error: expected response to have status code 202 but got 200".

23  
2

The test should fail and there should be an appropriate error message displayed.

## Example of Assertions

- There are multiple libraries supported for assertions in Postman. Using `pm.expect()` as a generic assertion library we can make use of chaiJs library assertions.
- For example, if we want to add an assertion that checks the status code as one of **200** or **201**, then we can simply write it as shown below:



The screenshot shows a Postman collection named "Assertions / Get User". A GET request is defined with the URL `https://reqres.in/api/user`. In the "Tests" tab, the following JavaScript code is written:

```
1 pm.test("Status code is 200", function () {  
2   pm.expect(pm.response.code).to.be.oneOf([200,201]);  
3 });  
4
```

23  
3

## Advanced Assertions

- In Postman, there is possibility to write advanced assertions where verification can be done for specific data.
- **For example**, validating a response body against a JSON schema, asserting the value of a particular field in the response, etc.

## Validating A JSON Field In Response (1)

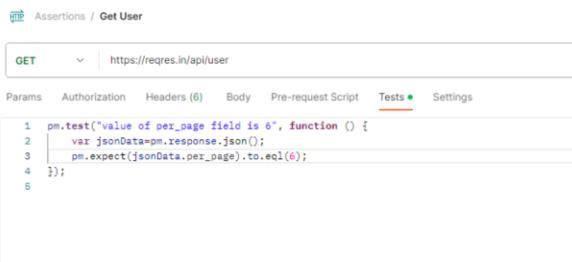
- In the previous example, the below **JSON response** was returned upon API call. We want to assert the value of **per\_page** field to always equal to **6**.

```
1  {
2   "page": 1,
3   "per_page": 6, [Red box]
4   "total": 12,
5   "total_pages": 2,
6   "data": [
7     {
8       "id": 1,
9       "email": "george.bluth@reqres.in",
10      "first_name": "George",
11      "last_name": "Bluth",
12      "avatar": "https://reqres.in/img/faces/1-image.jpg"[Blue underline]
13    },
14    {
15      "id": 2,
16      "email": "janet.weaver@reqres.in",
17      "first_name": "Janet",
18      "last_name": "Weaver",
19      "avatar": "https://reqres.in/img/faces/2-image.jpg"[Blue underline]
20    },
21  ]
```

23  
5

## Validating A JSON Field In Response (2)

- The assertion can be written as shown below:



A screenshot of the Postman interface. At the top, it says "Assertions / Get User". Below that, a "GET" method is selected with the URL "https://reqres.in/api/user". Under the "Tests" tab, there is a script:

```
1 pm.test("value of per_page field is 6", function () {  
2     var jsonData=pm.response.json();  
3     pm.expect(jsonData.per_page).to.eql(6);  
4 };  
5 );
```

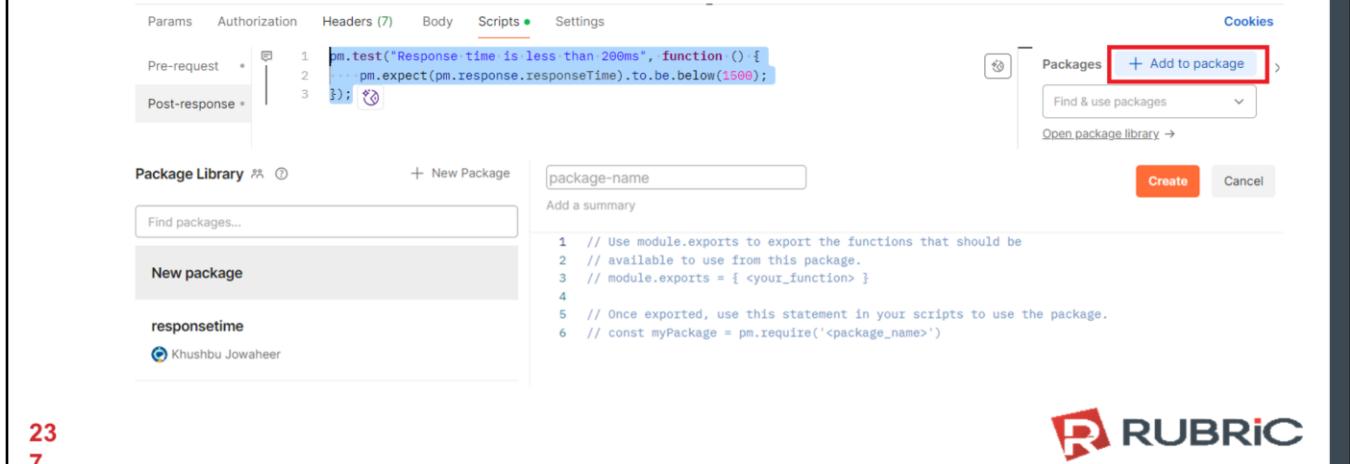
- As shown above, we have first stored the JSON response of the request in the **jsonData** (line 2) local variable and then added the assertion in the **pm.expect()** block (line 3).

23  
6

After executing the API call, click on the “**Test Results**” tab to view the result where it shows that the test passed successfully.

## Reuse scripts

- You can add commonly used scripts and tests to packages in your team's Package Library. This enables you to reuse scripts in the Pre-request and Post-response tabs in your team's HTTP requests, collections, and folders.



Params   Authorization   Headers (7)   Body   **Scripts**   Settings

Pre-request   •   Post-response

```
1 pm.test("Response-time is less than 200ms", function() {
2   ... pm.expect(pm.response.responseTime).to.be.below(1500);
3 });
```

**Cookies**

Packages **+ Add to package**

Find & use packages

Open package library →

**Create**   Cancel

Package Library

+ New Package

Find packages...

New package

responsetime

Khushbu Jowaheer

package-name

Add a summary

```
1 // Use module.exports to export the functions that should be
2 // available to use from this package.
3 // module.exports = [ <your_function> ]
4
5 // Once exported, use this statement in your scripts to use the package.
6 // const myPackage = pm.require('<package_name>')
```

**R** RUBRIC

23  
7

## Tips & Notes

- A single Postman test can contain as many assertions as you would like to have. They are exactly analogous to the unit testing world where we can have multiple assertions in a single unit test.
- In a single test, if one assertion fails, subsequent assertions will not be validated
  - For example, suppose a test had 5 assertions and 3rd assertion did not pass, then the 4th and 5th assertions will not be evaluated, and the Postman test will be marked as failed.
- Unlike assertions, if a test fails, the rest of the tests are still executed, and each test is either marked Pass or Fail once the request execution is complete.

## Pros & Cons Of Assertions

- Writing tests/assertions with the Postman provides a lot of power and flexibility in the hands of developers or QA who is working on performing end to end test for a particular API endpoint.
- But, just like the saying “too much of anything is not good”, too much of granular assertions for the response received can sometimes result in the Postman tests being flaky, as a slight change to dev code will make the tests fail.
- Thus, simply said, before adding a test into Postman, it is wise to understand the implication of the test/assertion being added, and we must ensure, that only absolute essential fields or validations should be covered as a part of assertions to avoid flakiness in the test results for future executions.



## Advanced Scripting For Complex Testing Workflows In Postman

240

## Advanced Scripting For Complex Testing Workflows In Postman



- In this chapter, you will learn about:
  - Passing Data Between Requests In Postman
  - Request Chaining In Postman
  - Advanced Workflow Chaining With Postman



RUBRIC

## Passing Data Between Requests

- Passing data between requests is an extension of using different types of Postman variables. It's not very uncommon where an API request depends upon data from the previous request's response.
- To build such kind of capabilities using Postman, we can simply use **Postman variables** to set the value depending on the response that was received from the preceding or previous requests.

24  
2

## Example: Register user scenario

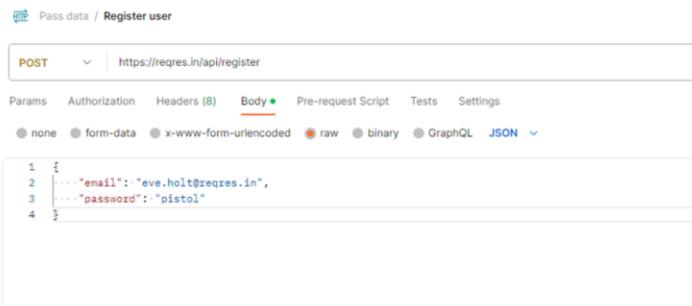
- We will be using the same API <https://reqres.in/api> with its 2 endpoints i.e. **Register user** and **Get user details**.
- Scenario:
  - We will use the value of the **user-id** returned from the registering endpoint and use it to get the user details method.
  - This will be achieved by storing the **userId** value in an **environment variable** and using that environment variable in the consequent request.

24  
3

### NOTE:

All such scripts will be a part of the “Tests” tab of the Postman request builder console.

## Example: Configuring the Body of request

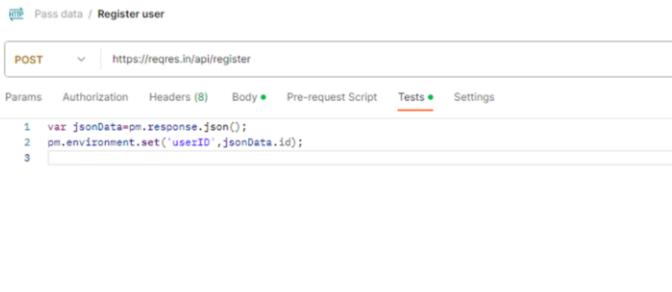


The screenshot shows the Postman interface with a POST request to <https://reqres.in/api/register>. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "email": "eve.holt@reqres.in",  
3   "password": "pistol"  
4 }
```

- POST request to register the user JSON body with fields having email and password as below:
  - Email: [eve.holt@reqres.in](mailto:eve.holt@reqres.in)
  - Password: [pistol](#)

## Example: Configuring the test script



```
1 var jsonData=pm.response.json();
2 pm.environment.set('userId',jsonData.id);
3
```

- 1: Fetch data from **JSON** response
- 2: Store the value of the **id** from this **JSON** into an environment variable named **userId**.

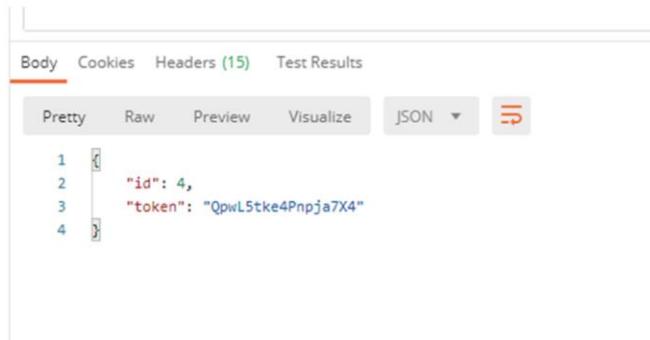
24  
5

### NOTE:

In the above script, the value of the **ID** will be now stored in the **userId** environment variable and the same can be used while executing the **GET** user request endpoint.

## Example: Checking Test Results

- After clicking on **Send**, the response of this API will look as shown below:



The screenshot shows a REST API testing interface with the following details:

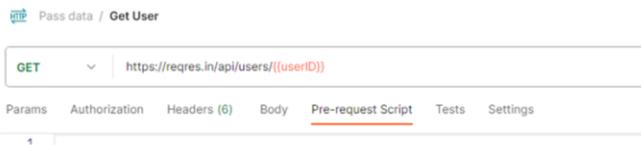
- Header bar: Body, Cookies, Headers (15), Test Results.
- Tool buttons: Pretty, Raw, Preview, Visualize, JSON dropdown, copy icon.
- JSON response content:

```
1  {
2   "id": 4,
3   "token": "QpwL5tke4Pnpja7X4"
4 }
```

24  
6

## Example: Configuration of GET Request

- The **GET** user request is configured as shown below:
  - <https://reqres.in/api/users/{{userId}}>



- Once the **POST register request** is executed, it will fetch the value of the **UserID** from the response and update the value of the environment variable, so that it could be used in the other requests.

## Request Chaining (1)

- With Postman, controlling the order of request execution is not straight forward. The default execution order is from top to bottom i.e. the order in which requests are declared or created in the Postman collection.
- Request chaining script are added in **pre-request script** or **post-request script** (or the **Tests** tab in the Postman request builder) which triggers the workflow once the request under execution completes.

## Request Chaining (2)

- **For example:**
  - Consider the login flow of an eCommerce site and validate the logged-in user. For a user to be able to log in, he or she must be first registered with the site and only then they will be able to log in. That is the order in which the API calls are executed.
- **Let's look at it from an integration test perspective. For an API test, we first need to:**
  1. Call the registering endpoint of the API for the user to successfully register.
  2. Then call the login endpoint and validate the details of the logged-in user.

## Request Chaining (3)

- Request chaining or the order of request execution is configured using the following script:

```
pm.execution.setNextRequest('{{RequestName}});
```

- In the example above the  **{{RequestName}}**  is the actual request name that is configured or set in the Postman collection.

## Configuration of request chaining in Postman (1)

- Let's configure a Postman collection with 3 different requests to illustrate the Request chaining concept.

1. **POST** request to register the user (<https://reqres.in/api/register>) with JSON body with fields having email and password as below:

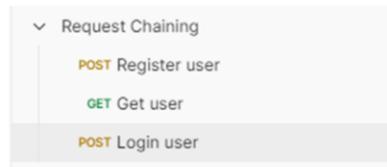
```
{  
  "email": "eve.holt@reqres.in",  
  "password": "pistol"  
}
```

2. **POST** request with **login** endpoint (<https://reqres.in/api/login>) with the same request body as above.

3. **GET** request with **userID 4**.

## Configuration of request chaining in Postman (2)

- The workflow that we will try to achieve is:
  - Register (**POST**) -> Login (**POST**) -> User details (**GET**)



25  
2

No matter how these requests are configured, we will create a workflow to have them executed in this order.

**But note** for the execution to start with the correct request, the request should be first in the collection. So that the runner knows the starting point

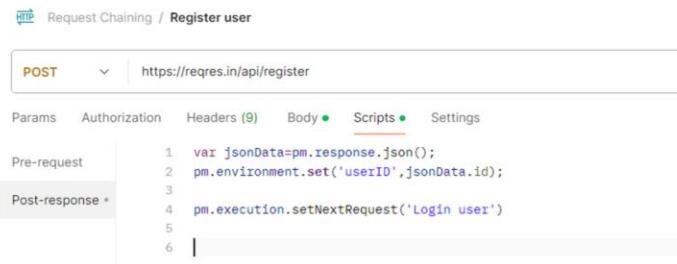
Please note that the requests are named as Register User, Get User and Login User respectively. It is important to have these request names exactly and correctly specified in the script, else the workflow will get stuck

## Configuration of request chaining in Postman (3)

- Let's see how the scripts look now:

### 1. POST request to register.

```
pm.execution.setNextRequest('Login user');
```



The screenshot shows the Postman interface with a POST request to `https://reqres.in/api/register`. The 'Scripts' tab is selected under the 'Body' section. A 'Post-response' script is defined with the following code:

```
1 var jsonData=pm.response.json();
2 pm.environment.set('userID',jsonData.id);
3
4 pm.execution.setNextRequest('Login user')
5
6
```

25  
3

## Configuration of request chaining in Postman (4)

### 2. POST request to login.

```
pm.execution.setNextRequest('Get user');
```

The screenshot shows a Postman collection named "Request Chaining / Login user". It contains two requests: "Login user" (POST) and "Get user" (GET). The "Login user" request has its "Pre-request" script set to `pm.execution.setNextRequest('Get user');`. The "Get user" request has its "Post-response" script set to `3`.

25  
4

## Termination of request chaining (1)

- The last is the **GET request** to get user details. It is important here to specify that nothing should happen after this request. If we don't add any workflow script here, the Postman collection is designed to resume the next request in its logical or default order.
- As shown earlier the original order of requests in the collection is **Register**, **Get User** and **Login** and our workflow is at Get User as the control does not know which request to go next, it will go to the next logical request that is Login and will ultimately result in an **infinite loop**.

## Termination of request chaining (2)

- To prevent such a situation, it is useful to terminate the workflow using:  
`pm.execution.setNextRequest(null)`
- Whenever the above script is encountered during a collection execution, the workflow execution will stop, and the collection runner will terminate.

25  
6

### NOTE:

1. In the cases where there are circular references of the next request, the collection runner will get stuck in an infinite loop thereby causing a memory leak sooner or later.
2. To execute the workflow, you will need to execute the first or starting request manually, post that, it will follow the workflow as defined in the post-request scripts. In other words, the first request that's required to be run as a part of the workflow should also be the first request in the **collection** or **collection folder**, so that the workflow execution starts with the first request which is also the first request of the workflow.

## Execution of workflow

- In the execution results/summary, you can see the order in which the requests have been executed and their results.

Request Chaining - Run results

Ran today at 12:02:30 · [View all runs](#)

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	MyFirstEnvironment	1	5s 140ms	0	1350 ms

[Run Again](#) [Automate Run](#) [+ New Run](#) [Export Results](#)

All Tests Passed (0) Failed (0) Skipped (0) [View Summary](#)

Iteration 1

**POST** Register user  
<https://regrès.in/api/register> 200 OK 1033 ms 873 B  
 No tests found

**POST** Login user  
<https://regrès.in/api/login> 200 OK 2202 ms 866 B  
 No tests found

**GET** Get user  
<https://regrès.in/api/users/4> 200 OK 815 ms 1.205 KB  
 No tests found

25  
7

## Advanced Workflow Chaining (1)

- The concept of request chaining can also be used for looping over a request multiple times based on some response values or environment variables.
- For example:
  - Consider integration test for a shopping cart app, where you need to test for a scenario where a user searches for a product and adds it to cart and has to perform the same operation 5 times i.e. till the cart has a total of 5 items and then finally checkout.

25  
8

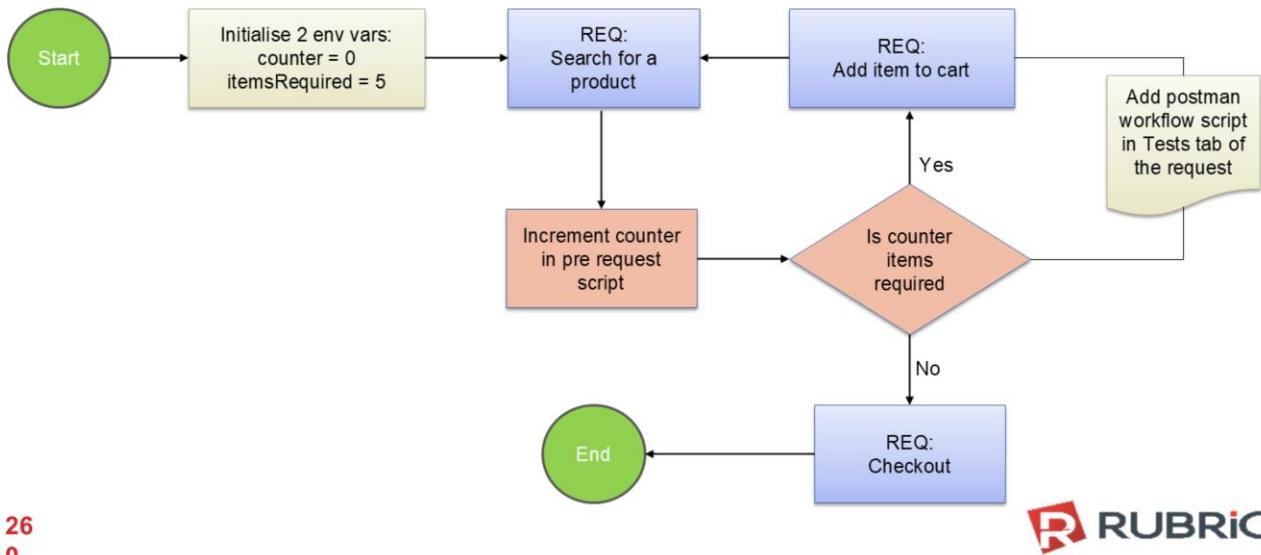
The above example that we discussed, is more of a linear workflow where we just configured workflow between a set of requests in the same collection.

## Advanced Workflow Chaining (2)

- Based on the previous example, if you were to write a linear flow for this kind of test, you would repeat individual requests in the collection and essentially the collection would have 5 requests for searching an item, and 5 requests for adding products to the cart and 1 request to checkout.
- With this workflow functionality and environment variables, we can avoid repeating the same requests in the collection and use the workflows to loop between the requests.

## Advanced Workflow Chaining (3)

- Let's see a flow sequence for such a scenario:



## Important tips

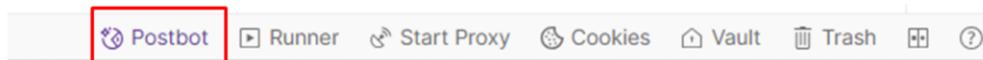
- While running with collections:
  - It is highly recommended to have **Postman.setNextRequest** in either all requests or in none of them. If few requests don't have **postment.setNextRequest** being set, the execution continues in the default order of collection.
  - If there are folders within a collection, then **Postman.setNextRequest** can be used just for requests belonging to the same folder.
- The **Postman.setNextRequest** is the last statement that gets executed in the post-request or pre-request script irrespective of where and what order it's mentioned.

## Conclusion

- In this tutorial, we covered few advanced scripting topics like combining environment and local variables to pass data between different requests in the Postman and how can we control the execution order of the requests using Postman Request chaining that allows advanced capabilities like looping and branching.
- It's a cool feature to mimic the behaviour of an application in the way in which it would interact with different APIs and is helpful to write end to end workflows using integration tests with API endpoints.

## What is Postbot ?

- **Postman Postbot** is an AI-driven testing assistant seamlessly integrated within the widely used Postman platform.
- It is designed to streamline and enhance the often-laborious process of API testing. Postbot will aid you in designing better test cases for API calls, leveraging its understanding of the deep context of the API.



## Features of Postbot

- **Automated Test Script Generation**
- **Test Script Refinement**
- **Natural Language Interaction**
- **Test Data Report Summarisation**
- **Automated API Debugging**

26  
4

### **Automated Test Script Generation:**

- Creates test scripts from API schemas and examples.
- Generates status code checks, data validation, and basic functional tests.
- Suggests valid parameter values for API requests.
- Aids in exploring API functionality and identifying edge cases.
- Compares response data against expected schemas.
- Highlights discrepancies and potential bugs, ensuring data integrity.

### **Test Script Refinement:**

- Identifies and corrects redundant or inefficient code.
- Proposes improvements to enhance test quality and maintainability.

### **Natural Language Interaction:**

- Enables test creation and modification using simple commands.
- Lowers the barrier to entry for users with limited coding experience.
- Interprets natural language queries for test generation.

### **Test Data Report Summarisation:**

- Summarizes test data reports to help you focus on the most critical aspects.

### **Automated API Debugging:**

- Debug API calls on your behalf, based on your intended outcomes.

## Advantages of Postbot

- Accelerated Test Creation
- Increased Test Coverage
- Improved Test Accuracy
- Enhanced Productivity
- Reduced Learning Curve
- Faster Feedback Loops
- Automated Debugging
- Test Data Report Summarisation

26  
5

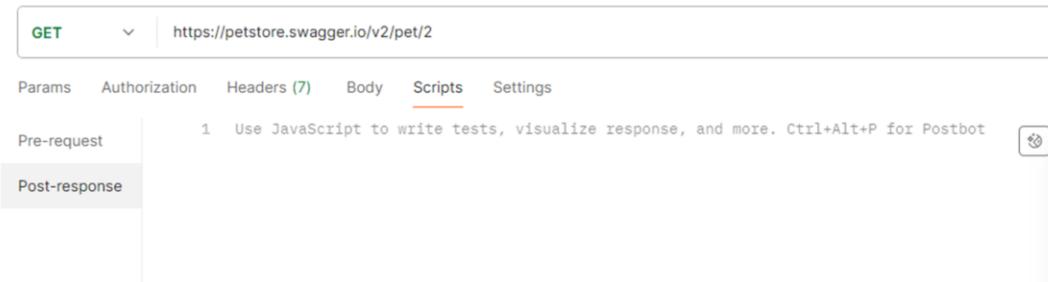
## Disadvantages of Postbot

- Limitations with Complex Logic
- Potential for Inaccuracy
- Dependence on AI
- Data Privacy Concerns
- Documentation Dependence
- Cost Considerations

26  
6

## Automated Test Script Generation (1)

- Here is a simple GET request to get pet with ID 4
- Before: There are **NO tests present** in this API request

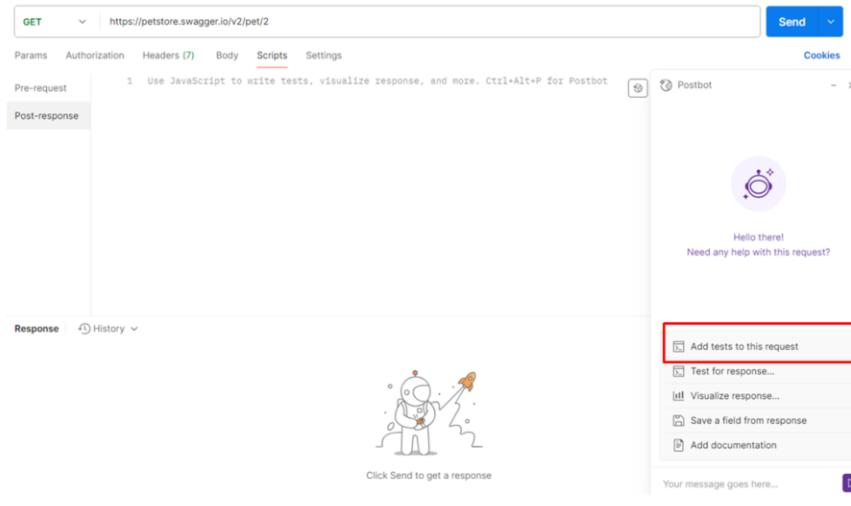


The screenshot shows the Postman interface with a GET request to <https://petstore.swagger.io/v2/pet/2>. The Scripts tab is selected. A note at the top right says "1 Use JavaScript to write tests, visualize response, and more. Ctrl+Alt+P for Postbot".

26  
7

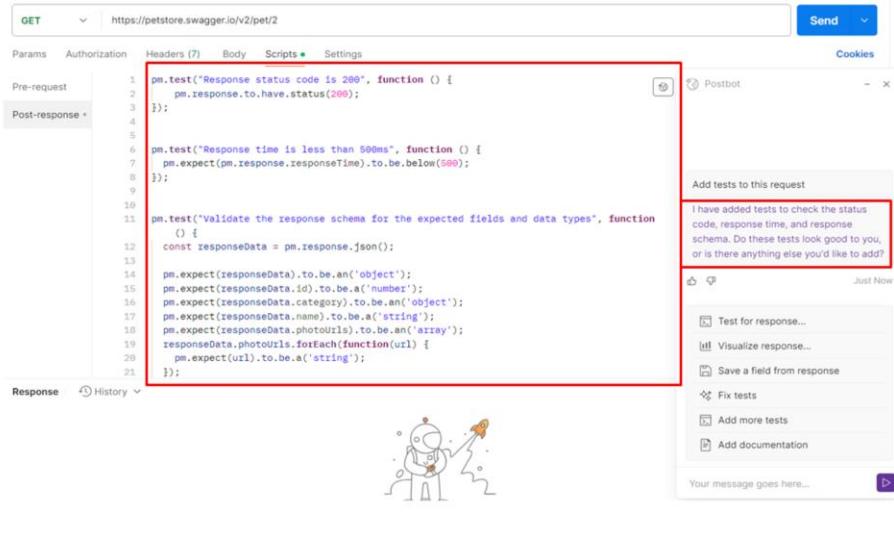
## Automated Test Script Generation (2)

- Postbot can generate a set of tests that are appropriate for this API call.

26  
8

## Automated Test Script Generation (3)

- After: Postbot has generated 5 tests for this API call.



The screenshot shows the Postman interface with a red box highlighting the generated test script in the 'Scripts' tab. The script content is:

```

1 pm.test("Response status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5
6 pm.test("Response time is less than 500ms", function () {
7     pm.expect(pm.response.responseTime).to.be.below(500);
8 });
9
10
11 pm.test("Validate the response schema for the expected fields and data types", function () {
12     const responseData = pm.response.json();
13
14     pm.expect(responseData).to.be.an('object');
15     pm.expect(responseData.id).to.be.a('number');
16     pm.expect(responseData.category).to.be.an('object');
17     pm.expect(responseData.name).to.be.a('string');
18     pm.expect(responseData.photosUrls).to.be.an('array');
19     responseData.photosUrls.forEach(function(url) {
20         pm.expect(url).to.be.a('string');
21     });
22 });

```

To the right, a 'Postbot' window is open with a message: "I have added tests to check the status code, response time, and response schema. Do these tests look good to you, or is there anything else you'd like to add?". Below the message are several options: Test for response..., Visualize response..., Save a field from response, Fix tests, Add more tests, and Add documentation.

26  
9

## Fixing test scripts (1)

- Postbot can **fix test scripts** that are broken or NOT working.
- **Before:**



The screenshot shows the Postman interface for a GET request to <https://petstore.swagger.io/v2/pet/2>. The 'Scripts' tab is selected. The 'Post-response' section contains the following code:

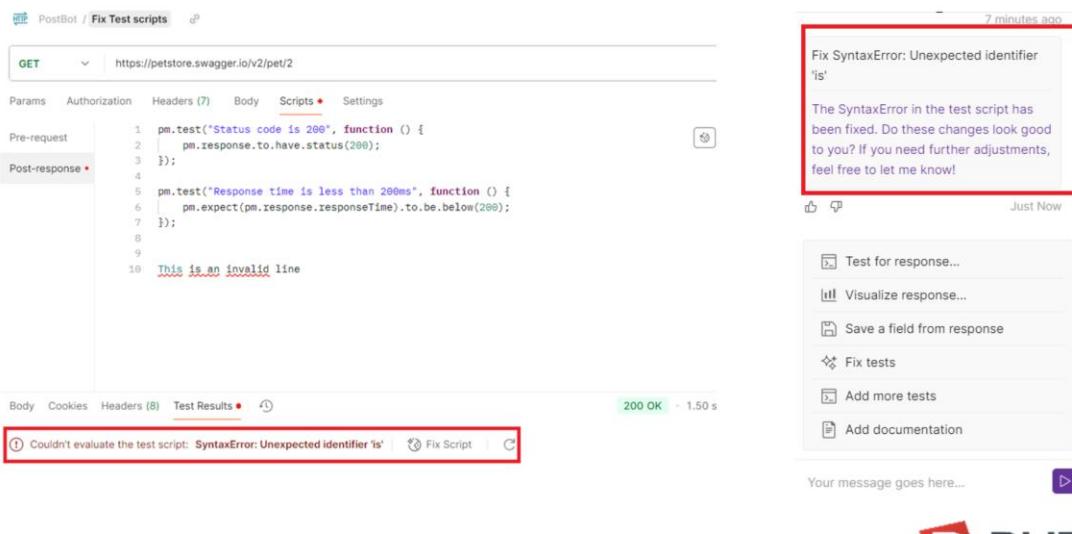
```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3 });  
4  
5 pm.test("Response time is less than 200ms", function () {  
6     pm.expect(pm.response.responseTime).to.be.below(200);  
7 });  
8  
9  
10 This is an invalid line
```

The line 'This is an invalid line' is highlighted in red, indicating it is causing an error.

27  
0

## Fixing test scripts (2)

- After:



The screenshot shows the Postman interface with a test script for a GET request to <https://petstore.swagger.io/v2/pet/2>. The 'Scripts' tab is selected. The script contains the following code:

```

1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("Response time is less than 200ms", function () {
6     pm.expect(pm.response.responseTime).to.be.below(200);
7 });
8
9
10 This is an invalid line

```

A red box highlights the error message at the bottom left: "Couldn't evaluate the test script: SyntaxError: Unexpected identifier 'is'".

A red box highlights a message from a collaborator:

**Fix SyntaxError: Unexpected identifier 'is'**

The SyntaxError in the test script has been fixed. Do these changes look good to you? If you need further adjustments, feel free to let me know!

Below the message are several buttons: Test for response..., Visualize response..., Save a field from response, Fix tests, Add more tests, and Add documentation.

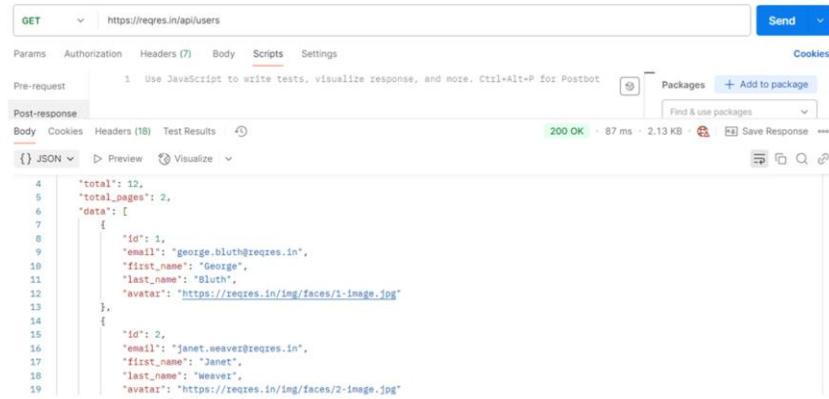
At the bottom right, there is a text input field labeled "Your message goes here..." with a send button.

27  
1

## Visualising Response Data (1)

- Postbot can generate code to neatly visualise JSON data in a table, making it easier to understand.

- Before:**



The screenshot shows the Postbot interface with a GET request to <https://reqres.in/api/users>. The response is displayed in a JSON viewer. The JSON data represents two users:

```

4   "total": 12,
5   "total_pages": 2,
6   "data": [
7     {
8       "id": 1,
9       "email": "george.bluth@reqres.in",
10      "first_name": "George",
11      "last_name": "Bluth",
12      "avatar": "https://reqres.in/img/faces/1-image.jpg"
13    },
14    {
15      "id": 2,
16      "email": "janet.weaver@reqres.in",
17      "first_name": "Janet",
18      "last_name": "Weaver",
19      "avatar": "https://reqres.in/img/faces/2-image.jpg"

```

27  
2

## Visualising Response Data (2)

- After:

GET <https://reqres.in/api/users> Send

Params Authorization Headers (7) Body Scripts Settings

Pre-request Post-response

```

1 var template = ` 
2 <style type="text/css">
3   .tfftable {font-size:14px;color:#333333;width:100%;border-width: 1px; border-color: #07ceeb; border-collapse: collapse;}
4   .tfftable th {font-size:18px;background-color:#07ceeb; border-width: 1px; padding: 8px; border-style: solid; border-color: #07ceeb; text-align:left;}
5   .tfftable tr {background-color:#ffffff;}
6   .tfftable td {font-size:14px; border-width: 1px; padding: 8px; border-style: solid; border-color: #07ceeb; }
7   .tfftable tr:hover {background-color:#e0ffff;}
8 </style>
9 
10 <table class="tfftable" border="1">
11   <tr>

```

Body Cookies Headers (19) Test Results

200 OK · 494 ms · 1.51 KB ·  Save Response 

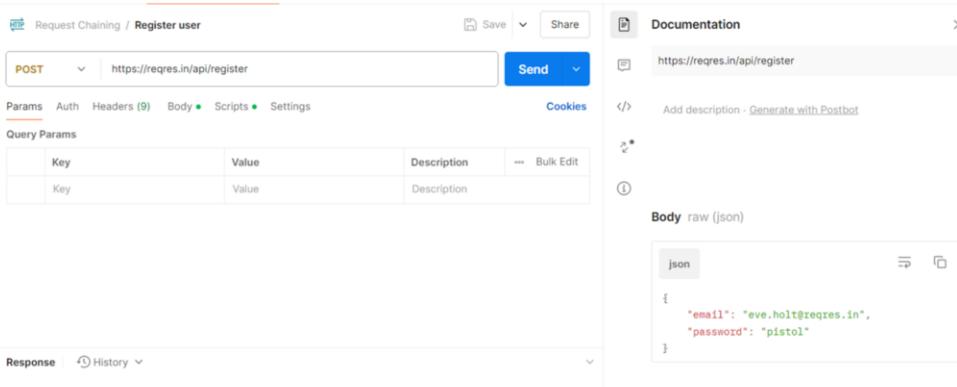
{ } JSON  Visualization  

ID	Email	First Name	Last Name	Avatar
1	george.bluth@reqres.in	George	Bluth	<a href="https://reqres.in/img/faces/1-image.jpg">https://reqres.in/img/faces/1-image.jpg</a>
2	janet.weaver@reqres.in	Janet	Weaver	<a href="https://reqres.in/img/faces/2-image.jpg">https://reqres.in/img/faces/2-image.jpg</a>
3	emma.wong@reqres.in	Emma	Wong	<a href="https://reqres.in/img/faces/3-image.jpg">https://reqres.in/img/faces/3-image.jpg</a>
4	eve.holt@reqres.in	Eve	Holt	<a href="https://reqres.in/img/faces/4-image.jpg">https://reqres.in/img/faces/4-image.jpg</a>
5	charles.morris@reqres.in	Charles	Morris	<a href="https://reqres.in/img/faces/5-image.jpg">https://reqres.in/img/faces/5-image.jpg</a>

27  
3

## Generating documentation for a Request (1)

- Postbot can generate a documentation for your API request. This is the quickest way to add simple and informative documentation to requests.
- Before:



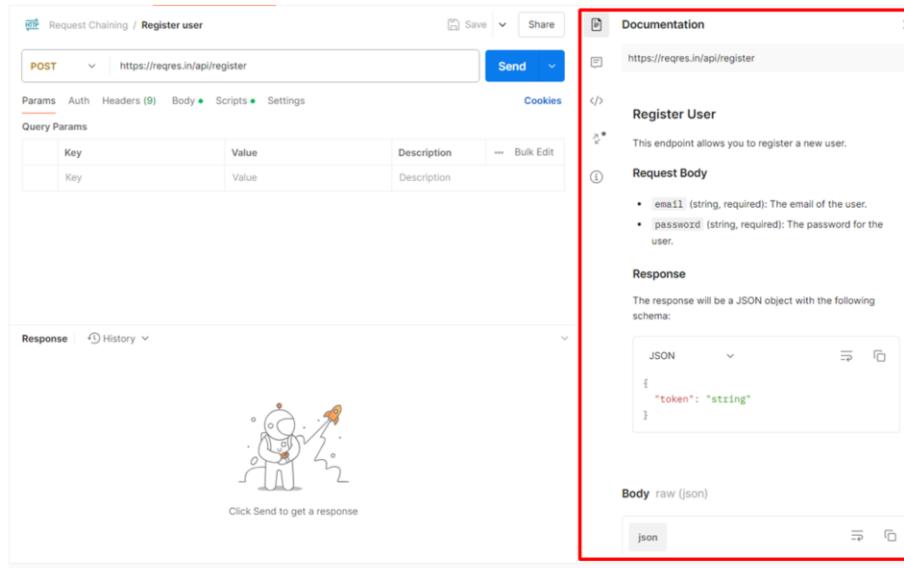
The screenshot shows the Postbot interface. On the left, there's a main panel for "Request Chaining / Register user" with a POST method selected and the URL <https://reqres.in/api/register>. Below this are sections for Params, Auth, Headers (9), Body, Scripts, and Settings. A "Cookies" section is also present. On the right, a "Documentation" panel is open with the URL <https://reqres.in/api/register>. It contains a note "Add description - Generate with Postbot" and a "Body raw (json)" section. The JSON content is:

```
json
{
  "email": "eve.holt@reqres.in",
  "password": "pistol"
}
```

27  
4

## Generating documentation for a Request (1)

- After:



The screenshot shows the Postman interface with a request to `https://reqres.in/api/register`. The request method is `POST`. The documentation panel on the right is highlighted with a red border and contains the following information:

**Documentation**  
`https://reqres.in/api/register`

**Register User**  
This endpoint allows you to register a new user.

**Request Body**

- `email` (string, required): The email of the user.
- `password` (string, required): The password for the user.

**Response**  
The response will be a JSON object with the following schema:

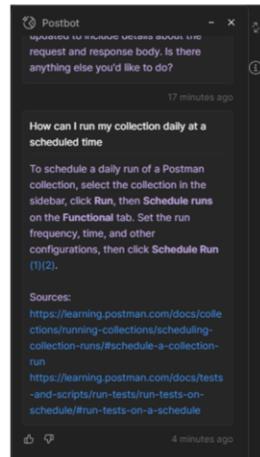
```
JSON
{
  "token": "string"
}
```

**Body** raw (json)

27  
5

## Postman Support using AI

- Instead of searching for **Postman documentation** online. You can ask Postbot directly. It will provide you with step-by-step instructions along with the related sources as well.





## API Testing Using Postman and Newman

277

## How To Use Command Line Integration With Newman In Postman?



**RUBRIC**

27  
8

- In this chapter, you will learn about:
  - What Is Newman
  - Installing Newman
  - Running Collections Using Newman
  - Newman Integration With Environment Variables
  - Assertion Results Using Newman
  - Report Generation Using Newman
  - More Options With Newman

In this tutorial, we will see how we can integrate or execute Postman collections through the command line using Newman which is a command-line integration tool for Postman.

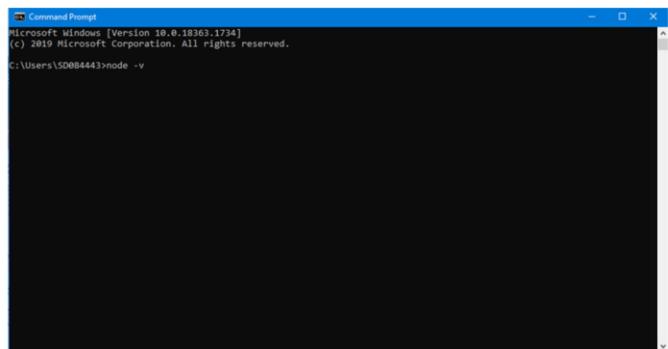
Newman is a free and open-source tool. It provides powerful capabilities to run the Postman collections, leveraging super-useful capabilities of Postman like Tests, Assertions, Pre-request scripts, etc and running the collection through the command line.

## What Is Newman?

- Newman is a command-line runner for Postman collections. In other words, it allows a user to run an existing Postman collection through the command line.
- It expects or consumes the JSON version of the collection that can be obtained by simply exporting the collection in JSON collection format or the URL of the collection which is nothing but the same JSON that's obtained by the collection export.

## Installing node.js (1)

- Newman is built on **node.js** which you can learn here. As it is built on node.js, it uses the default package manager for node.js which is **npm**.
- For the installation, you need to make sure that you don't have node.js previously installed. To do so:
  1. Open your Command Prompt
  2. Type **node -v**
  3. Press Enter



28  
0

### What is NPM?

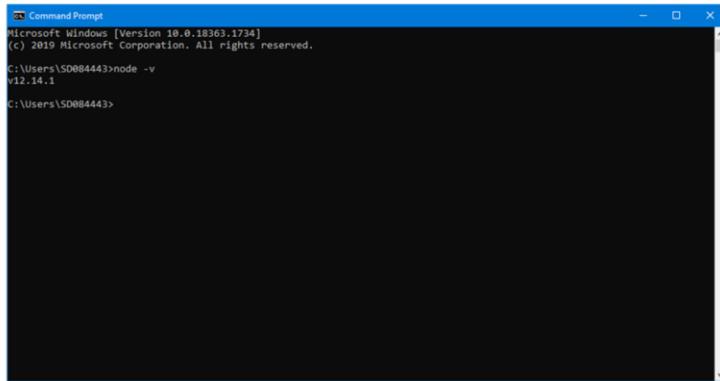
Node Package Manager or NPM is a package manager for Javascript programming language and is the default package manager for Node.js. It is like a repository of projects and has knowledge of what requirements each project has. According to the official website of npm, It is the world's largest software registry, with approximately 3 billion downloads per week. The registry contains over 600,000 packages (building blocks of code).

NPM makes it easy for the JS (Javascript) developers to share the code and problems on a repository. This code can then be reused by you in your next project or by anyone who wants the same feature that you have already developed. This makes it super easy for the developers to code better and in a less time.

Although too much knowledge about NPM is not necessary for us but there is much more to NPM than packages and registry. If you are interested in the same, you can visit their website here. We will try to install Newman now and as we discussed above, Newman requires node.js and NPM. So first we will try to install both of these things by following the steps.

## Installing node.js (2)

- If you see a version number, then you have node.js previously installed and do not need anything else to do. If not, you can install node by downloading it from <https://nodejs.org/en/download/>



```
Windows Command Prompt [Version 10.0.18363.1734]
(c) 2019 Microsoft Corporation. All rights reserved.

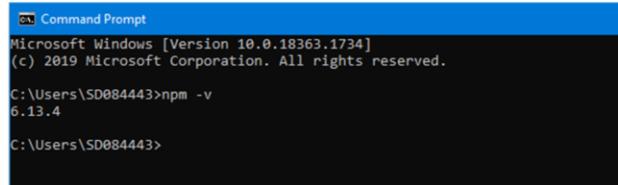
C:\Users\S0884443>node -v
v12.14.1

C:\Users\S0884443>
```

## Installing Node Package Manager (1)

- For installation, first we check if it is already installed:

- Open your Command Prompt
- Type **npm -v**
- Press Enter



```
Command Prompt
Microsoft Windows [Version 10.0.18363.1734]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SD084443>npm -v
6.13.4

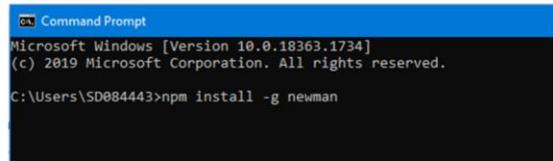
C:\Users\SD084443>
```

- Similarly, If you see a version number as you press enter, then you already have npm installed and you can proceed further for installing Newman.

## Installing Node Package Manager (2)

- If newman is not already installed, then use the below command:

```
npm install -g newman
```



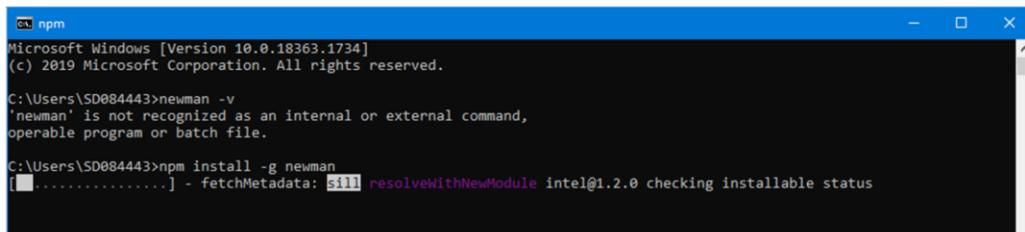
A screenshot of a Microsoft Windows Command Prompt window. The title bar says 'Command Prompt'. The window shows the following text:  
Microsoft Windows [Version 10.0.18363.1734]  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\SD084443>npm install -g newman

- Here '-g' denotes global installation which means that Newman package will be accessible from any folder/location on the file system.

For doing a local install, you can remove the '-g' flag from the above, where the Newman package will be accessible only from the installed location or folder.

## Installing Node Package Manager (2)

- This will install a new dependency through NPM. You will see the following screen after pressing enter (if npm is successfully fetched and installed).



```
npm
Microsoft Windows [Version 10.0.18363.1734]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SD084443>newman -v
'newman' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\SD084443>npm install -g newman
[0] .....] - fetchMetadata: sill resolveWithNewModule intel@1.2.0 checking installable status
```

28  
4

For doing a local install, you can remove the '-g' flag from the above, where the Newman package will be accessible only from the installed location or folder.

It will take a few minutes to install Newman. Once installed you will be indicated with the following line.

*newman @3.9.4*

*added 196 packages in 187.889s (Time may vary).*

For confirmation, you can also check the version of Newman.

## Newman installation

- To validate the successful installation of Newman, you can simply check its version using the below command:

```
newman -v
```

```
C:\Users\SD084443>newman -v  
5.3.0
```

## Running Collections Using Newman

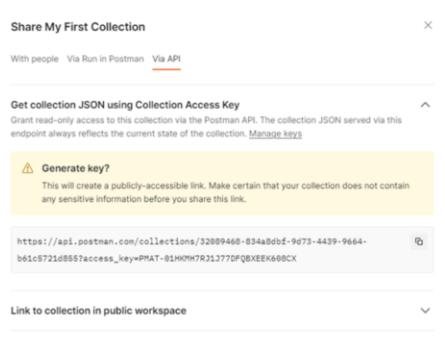
- To run collections using Newman, you should have any one of the two:
  - The collection in JSON format.
  - URL of the hosted collection.
- The command used to run the Postman collection using Newman is:
  - `newman run {{collectionJsonPath}}`
  - `newman run {{collectionUrl}}`

28  
6

To start with running a collection with Newman, first you need to have a collection in your Postman. We will be using the same collection that we created for the request chaining example.

## Running the collection using Newman through API link (1)

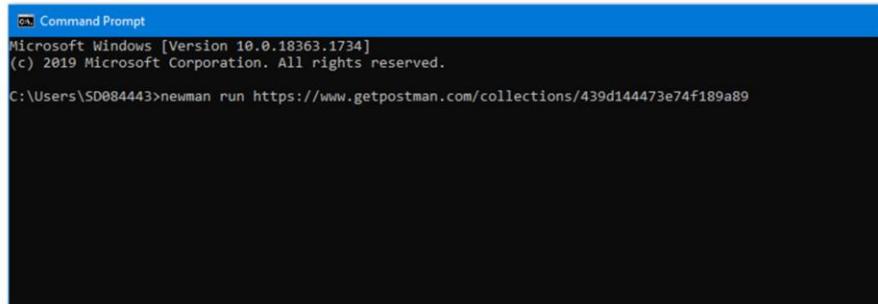
- Select the more actions icon  next to the collection name.
- Select **Share**.
- Select the **Via API** tab.



## Running the collection using Newman through API link (2)

- Open your Command Prompt and type:

```
newman run <<link>>
```

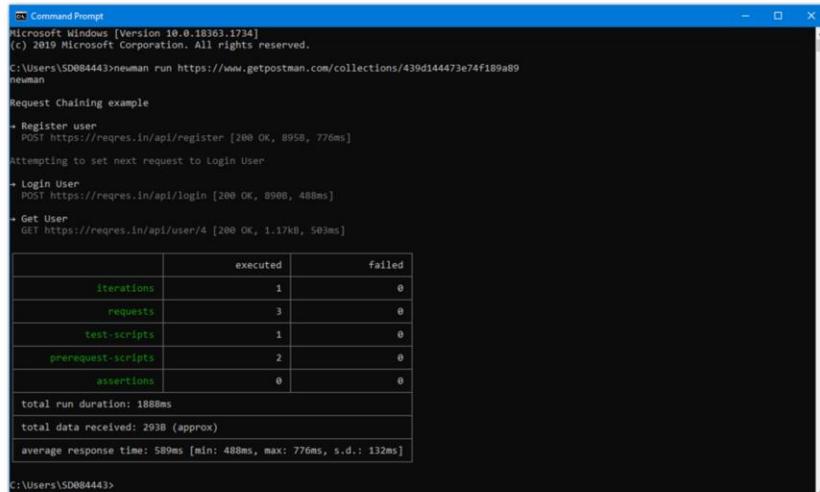


A screenshot of a Microsoft Windows Command Prompt window. The title bar says "Command Prompt". The window shows the following text:  
Microsoft Windows [Version 10.0.18363.1734]  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\SD084443>newman run https://www.getpostman.com/collections/439d144473e74f189a89

- Press **Enter**.

## Running the collection using Newman through share link (3)

- Your collection has successfully executed if you see the following screen:



```

Microsoft Windows [Version 10.0.18363.1734]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SD084443\neaman run https://www.getpostman.com/collections/439d144473e74f189a89
neaman

Request Chaining example

- Register user
  POST https://reqres.in/api/register [200 OK, 8958, 776ms]
  attempting to set next request to Login User

- Login User
  POST https://reqres.in/api/login [200 OK, 8968, 488ms]

- Get User
  GET https://reqres.in/api/user/4 [200 OK, 1.17kB, 503ms]

      iterations | executed | failed
      requests   | 1         | 0
      test-scripts | 1         | 0
      prerequest-scripts | 2         | 0
      assertions  | 0         | 0

total run duration: 1888ms
total data received: 293B (approx)
average response time: 589ms [min: 488ms, max: 776ms, s.d.: 132ms]

C:\Users\SD084443>

```

Once the collection has executed you will see the tests details as we saw in collection runner in Postman. We had one test for each of our requests, hence we see the results accordingly. It can also be seen from the image above, the details are similar to collection runner. We can see response status (time, size and status code) of each requests along with the test scripts that we executed.

## Running the collection using Newman through JSON file (1)

- Select the more actions icon  next to the collection name.
- Select **Export**.
- Save file to **Downloads → APITesting** (*folder name*)

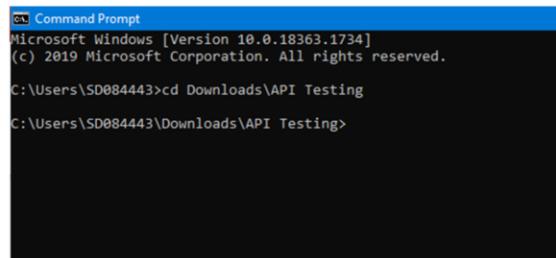
29  
0

### NOTE:

Always use **collection v2.1** which is recommended as discussed in the Collections in Postman chapter.

## Running the collection using Newman through JSON file (2)

- Once you have saved the json file, you need to change the working directory to the location where the file has been saved.
- For example:
  - The JSON file has been saved to **Downloads→API Testing** folder.
- To do so, execute the command **cd** followed by an absolute or relative path of the directory:



```
Command Prompt
Microsoft Windows [Version 10.0.18363.1734]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SD084443>cd Downloads\API Testing

C:\Users\SD084443\Downloads\API Testing>
```

29  
1

### NOTE:

Always use **collection v2.1** which is recommended as discussed in the Collections in Postman chapter.

Always use ‘\’ when specifying the absolute/relative path

## Running the collection using Newman through JSON file (3)

- Open your Command Prompt and type:

```
newman run <<name of the file>>
```

- For example:

```
newman run "RequestChainingExample.postman_collection.json"
```

Note: Remember to place the file name in inverted commas otherwise the shell will consider it as a directory name.

## Running the collection using Newman through JSON file (4)

- You will see the expected results of your collection Newman collection as shown below:

```
Command Prompt
Microsoft Windows [Version 10.0.18363.1734]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SD084443>cd Downloads\API Testing
C:\Users\SD084443\Downloads\API Testing>newman run "Request Chaining example.postman_collection.json"
newman
Request Chaining example
- Register user
  POST https://reqres.in/api/register [200 OK, 893B, 861ms]
Attempting to set next request to Login User
- Login User
  POST https://reqres.in/api/login [200 OK, 884B, 487ms]
- Get User
  GET https://reqres.in/api/user/4 [200 OK, 1.17kB, 103ms]

          iterations    executed    failed
  requests          1            0
  test-scripts      2            0
  prerequest-scripts 2            0
  assertions        0            0
total run duration: 1543ms
total data received: 293B (approx)
average response time: 483ms [min: 103ms, max: 861ms, s.d.: 309ms]

C:\Users\SD084443\Downloads\API Testing>
```

## Newman Integration With Environment Variables

- For a collection that does not rely on any environment variables, the collection could be simply executed using the Newman run command.
- But for collections, using the environment variables, we need to provide the environment variable JSON as well along with the collection JSON.

29  
4

Just as we used the environment variables in Postman, we can also set the environment variables in Newman.

## Creating and exporting environment variable (1)

- Create one environment called **Newman\_Env** with one environment variable **userId** and the value should be set to '4' as shown:

The screenshot shows a table for managing environment variables. The table has columns for VARIABLE, TYPE, INITIAL VALUE, and CURRENT VALUE. A single row is present, showing 'userId' as the variable, 'default' as the type, '4' as the initial and current value, and a checked checkbox next to it. There is also a placeholder 'Add a new variable' at the bottom.

VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE
<input checked="" type="checkbox"/> userId	default	4	4
Add a new variable			

29  
5

## Creating and exporting environment variable (2)

- Select the more actions icon  next to the environment name.
- Select **Export**.
- Save file to **Downloads → APITesting** (*folder name*)

29  
6

### NOTE:

Remember to save the environment file in the same location as you have your collection.

## Running collection with Environment Variables in Newman (1)

- For specifying environment --environment option is used, so the complete syntax becomes:

```
newman run <collection> --environment <file>
```

- Let's export the collection again with the changed request, and try running the same collection along with environment file with the command as below:

```
newman run "Request Chaining example.postman_collection.json" --  
environment "Newman_Env.postman_environment.json"
```

### NOTE:

Remember to use the complete name of the environment json file

Remember to place the file name in inverted commas otherwise the shell will consider it as a directory name.

## Running collection with Environment Variables in Newman (2)

- Once the command is executed, the output remains the same as the direct collection run, with the only change being the 'GET User' request now fetches the value from the environment JSON file.

```
C:\Users\SD004443\Downloads\API Testing>newman run "Request Chaining example.postman_collection.json" --environment "Newman_Env.postman_environment.json"
Newman
Request Chaining example
  • Register user
    POST https://reqres.in/api/register [200 OK, 8918, 983ms]
  Attempting to set next request to Login User
  • Login User
    POST https://reqres.in/api/login [200 OK, 8888, 474ms]
  • Get User
    GET https://reqres.in/api/user/4 [200 OK, 1.17kB, 562ms]

    iterations          |   executed |   failed |
    :-----|-----:|-----:|
    requests           |       1 |       0 |
    test-scripts        |       3 |       0 |
    prerequest-scripts |       2 |       0 |
    assertions          |       0 |       0 |

  total run duration: 2.3s
  total data received: 2938 (approx)
  average response time: 673ms [min: 474ms, max: 983ms, s.d.: 222ms]

C:\Users\SD004443\Downloads\API Testing>
```

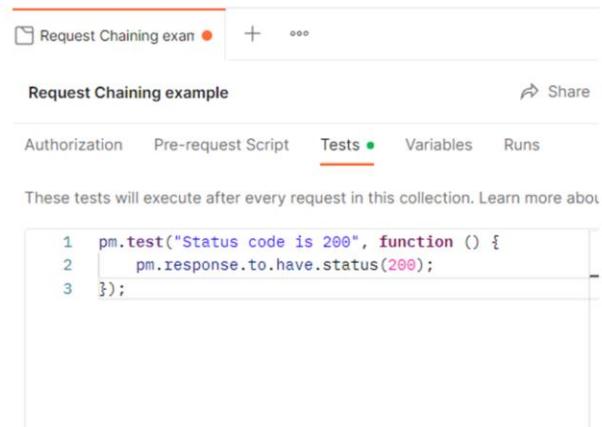
## Assertion Results Using Newman

- Since the Postman requests can contain assertions, we will now walkthrough, how the assertion results are displayed when the Postman collections are executed through a Newman.
- The Newman collection runner works the same as the Postman Collection runner and Request executor. For requests having assertions, the assertions get evaluated as and when the request execution completes, and the summary of the assertion execution is displayed in the test summary at the end of the test run.

## Adding assertion to collection level

- Open the collection in a new tab by clicking the collection name.
- Add the following code under **Tests** tab:

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



The screenshot shows the 'Tests' tab of a Postman collection named 'Request Chaining example'. The tab bar includes 'Authorization', 'Pre-request Script', 'Tests' (which is active), 'Variables', and 'Runs'. Below the tabs, a note says 'These tests will execute after every request in this collection. Learn more about tests'. The test code is displayed in a code editor:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3});
```

30  
0

Let's add a collection level assertion to check the status code of the response to be **200** i.e. for every request that's a part of the collection there should be this assertion associated.

So if this collection had 3 requests, it means that there should be a total of 3 assertions that should have got executed.

## Running collection with assertions in Newman

- Download the collection file again and execute the **newman run collection** command:

```
C:\Users\SD084443\Downloads\API Testing>newman run "Request Chaining example.postman_collection.json" --environment "Newman_Env.postman_environment.json"
newman
Request Chaining example
+ Register user
  ✓ POST https://reqres.in/api/register [200 OK, 8918, 1019ms]
    ✓ Status code is 200
Attempting to set next request to Login User
+ Login User
  ✓ POST https://reqres.in/api/login [200 OK, 8828, 61ms]
    ✓ Status code is 200
+ Get User
  ✓ GET https://reqres.in/api/user/2 [200 OK, 1.17kB, 513ms]
    ✓ Status code is 200
      Assertion execution details
      ┌─────────────────────────────────────────────────────────────────────────┐
      │          executed          failed          |
      └─────────────────────────────────────────────────┘
      iterations           1            0
      requests             3            0
      test-scripts         5            0
      prerequest-scripts  5            0
      assertions           3            0
      ┌─────────────────────────────────────────────────────────────────┐
      total run duration: 2.5s
      total data received: 293B (approx)
      average response time: 714ms [min: 513ms, max: 1019ms, s.d.: 219ms]
C:\Users\SD084443\Downloads\API Testing>
```

## Running a folder inside a collection using Newman

- In real and big project we have huge number of APIs inside a collection, so we segregate those into different folders. There can be a need when someone like to just test one set of APIs which are in single folder.
- In that case there is no point to execute all the collection as a whole, as it will execute all the folder which comes under the collection. Just like Postman, Newman also gives us the ability to run just a folder from collections.
- The following command is used:

```
newman run <collection_name> --folder <folder name>
```

## Report Generation Using Newman (1)

- With Newman along with test logs and test execution summary formatted reports can also be generated.
- There are some custom node modules available for generating Newman test execution reports such as **newman-reporter-html**. This reporter is again a node module and has to be separately installed using the below command:

```
npm install -g newman-reporter-html
```

```
C:\Users\SD084443\Downloads\API Testing>npm install -g newman-reporter-html
npm WARN newman-reporter-html@1.0.5 requires a peer of newman@4 but none is installed. You must install peer dependencies yourself.

+ newman-reporter-html@1.0.5
added 12 packages from 44 contributors in 11.711s
```

## Report Generation Using Newman (2)

- Once the module is installed, this can be used along with the Newman run command.

```
newman run <collection> --environment <file> -r html
```

- For example:

```
newman run "Request Chaining example.postman_collection.json" --environment "Newman_Env.postman_environment.json" -r html
```

```
C:\Users\SD084443\Downloads\API Testing>newman run "Request Chaining example.postman_collection.json" --environment "Newman_Env.postman_environment.json" -r html
```

## Report Generation Using Newman (3)

- Once the command is executed, an html report is generated in the working directory.

Newman Report				
Collection	Request Chaining example			
Time	Sun Sep 26 2021 15:05:26 GMT+0200 (Central European Summer Time)			
Exported with	Newman v5.0.0			
Iterations	Total	Failed		
Requests	1	0		
Preset Scripts	3	0		
Test Scripts	5	0		
Assertions	3	0		
Total run duration	2.5s			
Total data received	290B (approx)			
Average response time	679ms			
<b>Total Failures</b>	<b>0</b>			
Requests				
Register user				
Method	POST			
URL	https://reqres.in/api/register			
Mean time per request	976ms			
Mean size per request	368			
Total passed tests	1			
Total failed tests	0			
Status code	200			
Tests	Name	Pass count	Fail count	
	Status code is 200	1	0	
Login User				
Method	POST			
URL	https://reqres.in/api/login			

30  
5

Execution summary

Individual test details

## More Options With Newman

- For complete details of commands and switches that Newman supports, simply open command-line help for Newman using the below command:

**newman run -h**

```
C:\Users\15088443\Downloads\API Testing>newman run -h
Usage: newman run [collection] [options]

Initiate a Postman Collection run from a given URL or path

Options:
-e, --environment <path>          Specify a file or path to a Postman Environment
-f, --env-var <key>                Specify a file or path to a file containing Newman Globals
-r, --reporters [reporters]         Specify the reporters to use for this run (default: ["cli"])
-n, --iteration-count <ns>        Define the number of iterations to run
-d, --data <data-file-or-path>     Specify a data file or path to collections (either JSON or CSV)
--folder <path>                  Specify the folder to run from a collection. Can be specified multiple times to run multiple folders (default: [])
--global-env-var <key>            Allow the specification of environment variables via the command line, in a key-value format (default: [])
--export-environment <path>       Exports the final environment to a file after completing the run
--export <path>                   Exports the executed collection to a file after completing the run
--export-collection <path>        Exports the executed collection from the Postman API
--postman-api-key <apiKey>       API Key used to log the resources from the Postman API
--strictSSL                        Strict SSL validation. Disables certificate validation for the run
--ignore-redirects                 Prevents Newman from automatically following 3XX redirect responses
-x, --suppress-exit-code          Specify whether or not to override the default exit code for the current run
--allow-insecure                   Prevents Newman from validating certificates
--disable-unicode                 Forces Unicode compliant symbols to be replaced by their plain text equivalents
--user-validation                Enables user validation for requests (default: false)
--delay-between-request <n>       Specify the extent of delay between requests (milliseconds) (default: 0)
--timeout <n>                     Specify a timeout for collection run (milliseconds) (default: 0)
--timeout-script <n>              Specify a timeout for scripts (milliseconds) (default: 0)
--working-dir <path>              Specify the path to the working directory
--no-ssl                          Disables SSL validation
-k, --insecure                    Disables SSL validations
--ssl-client-cert-opts <path>    Specify the path to a client certificate configurations (JSON)
--ssl-client-key <path>           Specify the path to a client certificate (PEM)
--ssl-client-key-passphrase <passphrase> Specify the path to a client certificate private key
--ssl-extra-ca-certs <path>      Specify additionally trusted CA certificates (PEM)
--cookie-jar <path>               Specify the path to a custom cookie jar (serializing tough-cookie JSON)
--remove-cookie-jar <path>         Specify the path to a file to remove cookies
--verbose                         Show detailed information of collection run and each request sent
-h, --help                          display help for command
```

## Conclusion

- In this tutorial, we walked through the command line integration of Postman called Newman, which allows running Postman collections through a command-line interface.
- It's simply a node package and any command-line that has node installed along with Newman should be able to run the Postman collection and generate good looking reports of the collection execution.



## Reporting Templates With Newman

308

## Reporting Templates With Newman Command Line Runner In Postman



- In this chapter, you will learn about:
  - Standard HTML Reporter
  - Creating And Using Custom Reporters
  - Other Custom Reporters



**RUBRIC**

30  
9

In this tutorial, we will cover reporting templates that can be used along with the Newman command line runner to generate templated reports from the test execution of Postman collection.

## Standard HTML Reporter

- Given below is the image of how the standard HTML Report will look.

Newman Report																						
Collection	Postman-Newman Integration With Environment																					
Description	## Sample API collection This collection contains sample requests from this [API](https://regres.in) It contains following requests *																					
Time	Register a user * Login a user with given credentials * Get details for a user Mon Oct 07 2019 09:26:51 GMT+0530 (India Standard Time)																					
Exported with	Newman v4.5.5																					
	<a href="#">Collection details</a>																					
Iterations	Total	Failed																				
Requests	1	0																				
Preflight Scripts	3																					
Test Scripts	4																					
Assertions	6																					
Total run duration																						
Total data received																						
Average response time																						
Total Failures	0																					
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2">Requests</th> </tr> </thead> <tbody> <tr> <td colspan="2">Register User</td> </tr> <tr> <td>Description</td><td>API endpoint to "Register" a user in the system. &gt; A successful registration will result in a "HTTP 200" Status code</td></tr> <tr> <td>Method</td><td>POST</td></tr> <tr> <td>URL</td><td><a href="https://regres.in/api/register">https://regres.in/api/register</a></td></tr> <tr> <td>Mean time per request</td><td>736ms</td></tr> <tr> <td>Mean size per request</td><td>368</td></tr> <tr> <td>Total passed tests</td><td>1</td></tr> <tr> <td>Total failed tests</td><td>0</td></tr> <tr> <td>Status code</td><td>200</td></tr> </tbody> </table>			Requests		Register User		Description	API endpoint to "Register" a user in the system. > A successful registration will result in a "HTTP 200" Status code	Method	POST	URL	<a href="https://regres.in/api/register">https://regres.in/api/register</a>	Mean time per request	736ms	Mean size per request	368	Total passed tests	1	Total failed tests	0	Status code	200
Requests																						
Register User																						
Description	API endpoint to "Register" a user in the system. > A successful registration will result in a "HTTP 200" Status code																					
Method	POST																					
URL	<a href="https://regres.in/api/register">https://regres.in/api/register</a>																					
Mean time per request	736ms																					
Mean size per request	368																					
Total passed tests	1																					
Total failed tests	0																					
Status code	200																					
Individual request execution metrics																						

31  
0

The Standard HTML Report format that gets created through the Newman Html reporter is good but not great. It does not have a very engaging UI and the information is not succinct and is kind of scattered.

For reporting needs, it is required that the information is available at first glance and at the same time its pleasing to the eyes of the viewers as well. Thus the standard HTML Reporter template can be configured or changed in the way as desired, to make the reports look better as well as more UI friendly.

## Creating And Using Custom Reporters (1)

- Newman provides a highly customized template for formatting the Html reports that get generated by Newman-html-reporter by default.
- You can specify the location of the custom template using the ‘–reporter-html-template’ switch:
- The command will look as shown below:

```
newman run {{path to collection json}} -e {{path to environment json}} -r html --reporter-html-template {{path to custom template}}
```

## Creating And Using Custom Reporters (2)

- Follow the below steps for running a Postman collection using a custom reporter template.
  - Download template from the gist file.
  - Save the file on the local file system.
  - Now we can use any existing collection to run using Newman with the reporter template like the one that we downloaded.
  - Let us see the below command.

```
newman run postman-newman-integration2.json -e testEnvNewman.json -r html --reporter-html-template colored-template.hbs
```

31  
2

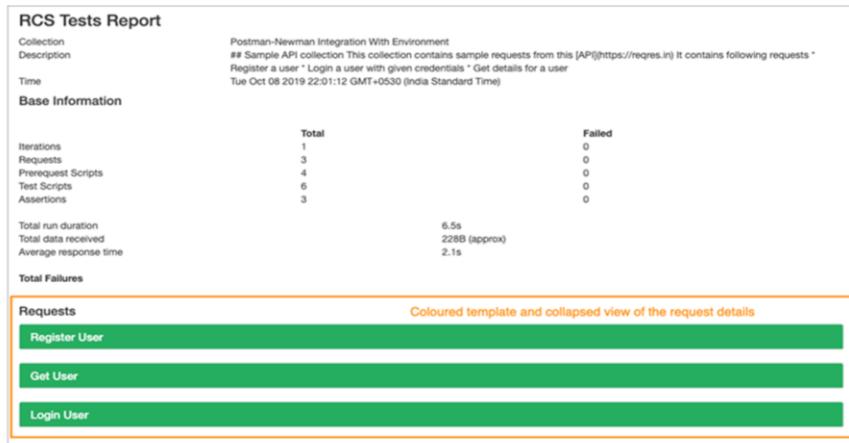
here.

<https://gist.githubusercontent.com/tegomass/fd67fa22f39a7ebe33a533862ff09d88/raw/89f179d8f9ff3acc38392d271b47f832c5076325/template-default-colored.hbs>

=

## Creating And Using Custom Reporters (3)

- Navigate to the Newman directory.
- Now open the generated HTML report. The below screenshot will show how the newly generated Html report will look like.

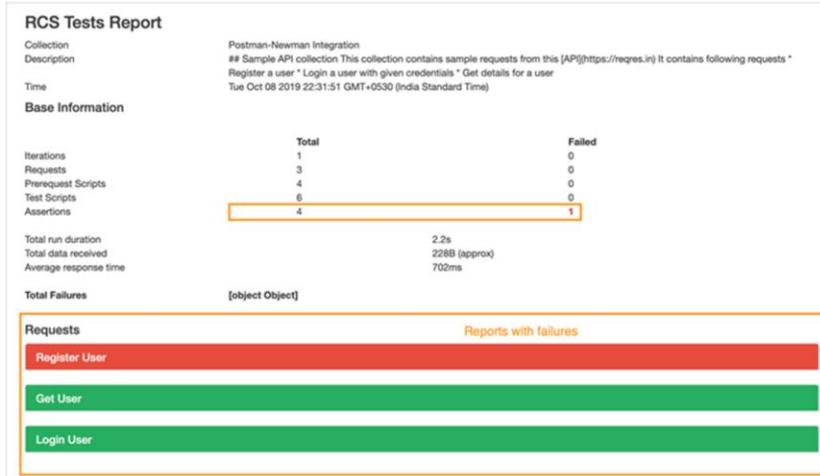


31  
3

As we have not specified location for generating Html reports, the default location will be inside the Newman folder that gets created in the directory where tests get executed from.

## Creating And Using Custom Reporters (3)

- Let us change an assertion in our test to make it fail and generate the report using the same template again and see how it looks.



31  
4

Contrast it with the generic or default report that gets generated with a newman-html-reporter. In the default view, there are no colored markers or styling that make it difficult to distinguish failures and successes and overall, it does not look very pleasing.

## Other Custom Reporters

- There is another Newman package that is available. It is named as Newman-reporter-htmlextra
- It also generates custom reports with more colored interfaces with distinguished success and failures. Let's see how the reports look with this template.

```
npm install -g newman-reporter-htmlextra
```

31  
5

Let's see the other custom reporters that are available (and are created by the online community), as they are free to use, we can choose as per our need and use it to customize the default reports that get generated by Newman.

### NOTE:

To use this template, install it using the node package manager with the below command.

## Other Custom Reporters

- Now earlier, with Newman collection runner, we were using reporter as just HTML using the -r switch. Now for this reporter, we will have to use the reporter name as htmlextra.
- Hence, to use this reporter, you can use the below command to execute a Postman collection.

```
newman run postman_collection_2.json -e testEnvNewman.json -r htmlextra
```

## Other Custom Reporters

- Navigate to the Newman folder in the same directory to find the generated HTML report.
- The salient features of the Reporter include:
  - Separate views for summary and all requests.
  - Lots of customizations.
  - Lots of color-coding to distinguish failures and success
  - More interactive with tabbed views as well as expand/collapse options for request details
  - All the request responses are captured fully including the header information.

31  
7

- Separate views for summary and all requests. There's also a separate tab to just view the failed requests.
- Lots of customizations are available with the reporter like Showing console logs, Not showing skipped tests, Showing only failures in the reports, etc. All the details are available at the GitHub page here for this module.
- Lots of color-coding to distinguish failures and success just by looking at the report.
- More interactive with tabbed views as well as expand/collapse options for request details.
- All the request responses are captured fully including the header information.

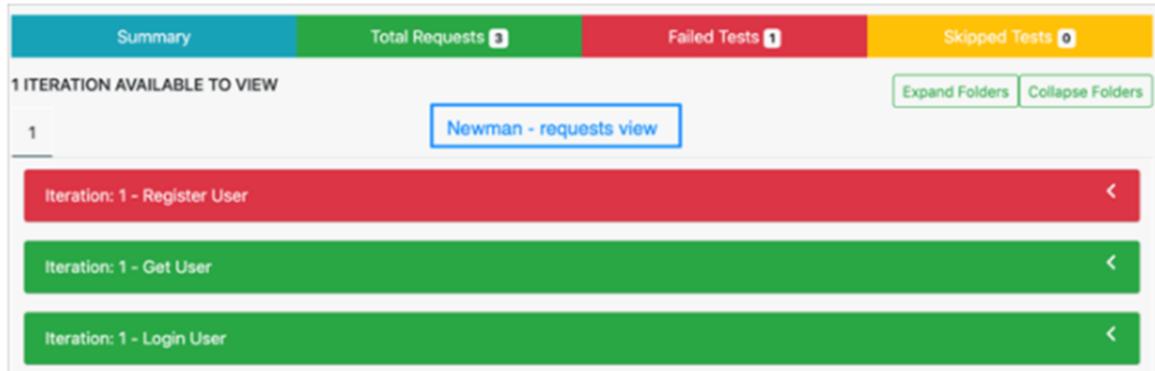
## Other Custom Reporters

- Requests view



## Other Custom Reporters

- Requests view



Summary      Total Requests 3      Failed Tests 1      Skipped Tests 0

1 ITERATION AVAILABLE TO VIEW

Newman - requests view

Iteration: 1 - Register User

Iteration: 1 - Get User

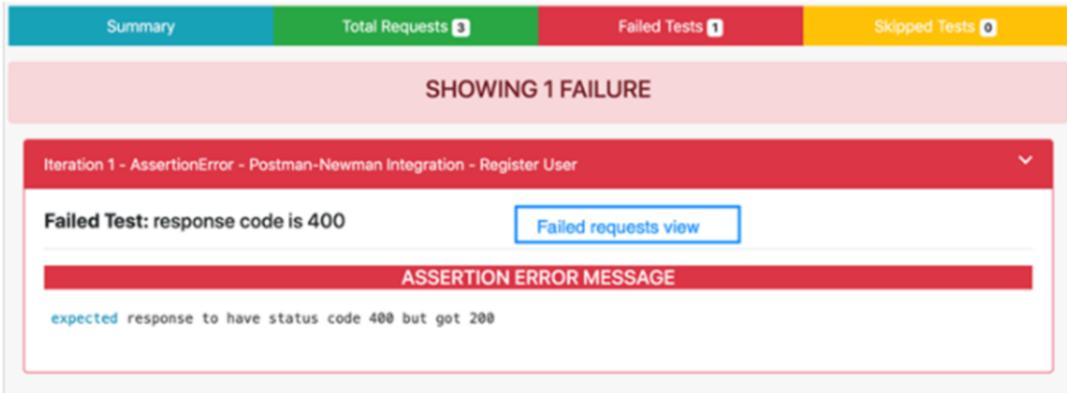
Iteration: 1 - Login User

Expand Folders   Collapse Folders

31  
9

## Other Custom Reporters

- Failed Requests View



The screenshot shows a reporter interface with the following structure:

- Summary Bar:** A horizontal bar at the top with four colored segments: teal (Summary), green (Total Requests 3), red (Failed Tests 1), and yellow (Skipped Tests 0).
- Section Header:** "SHOWING 1 FAILURE" in a pink header area.
- Iteration Details:** "Iteration 1 - Assertion Error - Postman-Newman Integration - Register User" in a red section.
- Failed Test Message:** "Failed Test: response code is 400" in a white box with a red border.
- Link:** "Failed requests view" in a blue button-like box.
- Assertion Error Message:** "ASSERTION ERROR MESSAGE" in a red header and "expected response to have status code 400 but got 200" in a white box below it.

32  
0

## Conclusion

- In this tutorial, we learned about customizing the HTML reports that get generated while using Newman-html-template.
- We also looked at another html reporter named Newman-reporter-htmlextra that generates better-styled reports with different views for Summary requests as well as failed requests.

32  
1

The customizations can be done, as per the requirements, but there are already a lot of readily available templates created by the online community and open source contributors that could be used out of the box.



## Mock Server in Postman

322

## Mock Server in Postman



- In this chapter, you will learn about:
  - What is Mock Server?
  - Why use Mock Server?
  - How to create a mock server in Postman?
  - How to get the response in a different format in Mock Server?

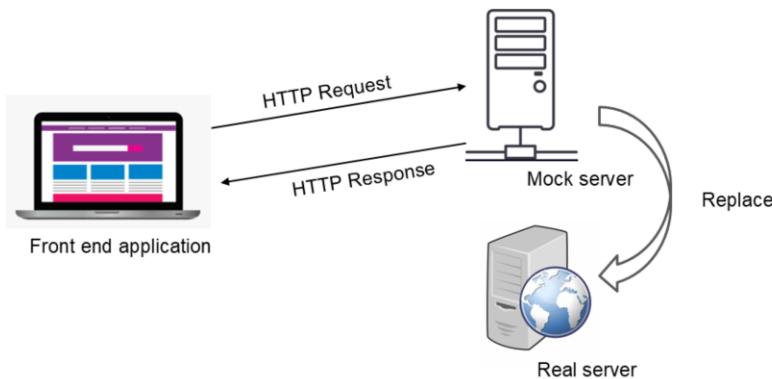


**RUBRIC**

32  
2

## What is Mock Server? (1)

- A mock server is a server that is not a real server. It is just a fake server that is simulated to work as a real server so that we can test our APIs and check the response or errors.

32  
4

This server is set up in such a way that we get a particular response for a particular request that we desire to see. A mock server behaves like a real server and uses fake APIs, of course, for testing and development purpose.

## What is Mock Server? (2)

- There are a number of reasons for which we require a mock server. Along with the case given above, it is also required in today's testing world.
- Such requirement is in Agile methodology which is recent and better than waterfall methodology. In this method, testing and development go side by side. For this, a tester needs to have the same requirement as the developer to work simultaneously.

## Why do we need a mock server?

To test your own API while developing and before making that API live on the server of your organisation.

To get feedbacks and bugs quicker.

For checking the dependencies in your API before it is available to everyone.

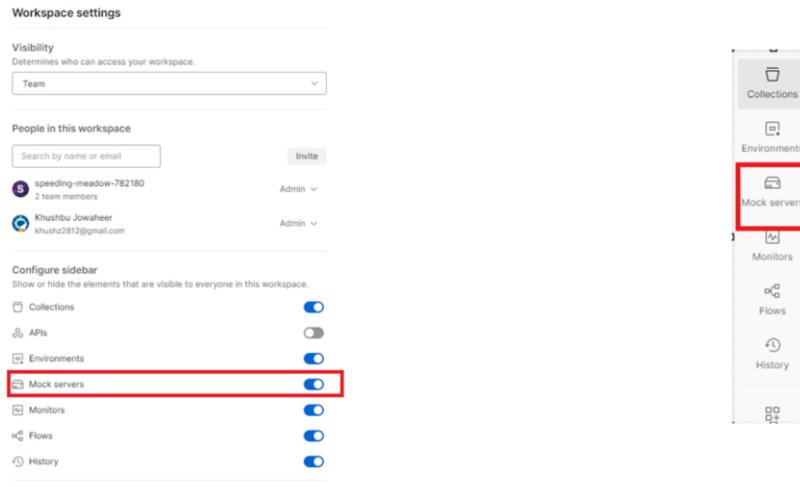
For QA engineers to use this for testing/isolating external dependencies and integration testing.

By front end developers to use it before actual endpoints are available. This is done while designing the UI so that you don't need to wait for the time till actual endpoints are developed. It saves a lot of time.

For engineers to develop a prototype of their idea to present it to the investors for funding.

## How to create a mock server in Postman (1)

- Click on the  button to **Mock Server** on the sidebar.



32  
7

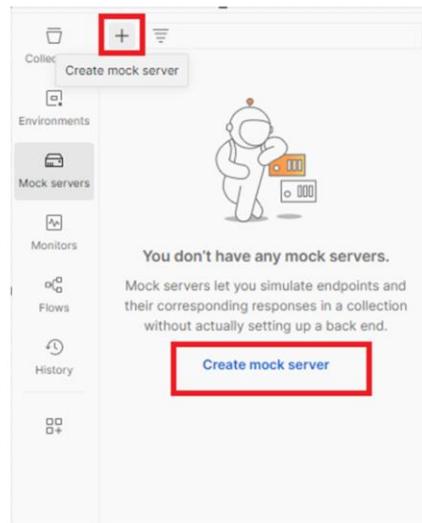
 RUBRIC

In this section, we will create our first mock server in Postman but before that, you must know a few things about the mock server

1. The mock server is already integrated inside the postman app and is not required externally.
2. The mock server also has CORS (Cross-Origin Resource Sharing) enabled. It means that you won't get any cross-origin errors while using the mock server.
3. The mock server is free to use i.e. it is available in free tier of Postman.

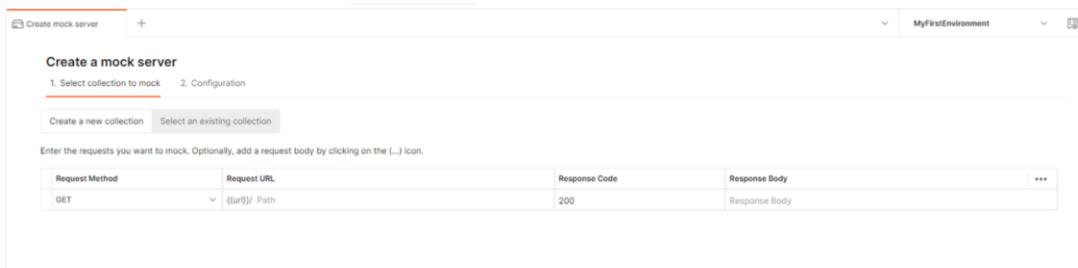
## How to create a mock server in Postman (2)

- Click to create mock server using **+** or **Create mock server** as shown below:

32  
8

## How to create a mock server in Postman (3)

- A new tab will be opened which will enable us to create mock server.
- There are different columns in this mock server panel which stands for:
  1. Method
  2. Request URL
  3. Response Code
  4. Response Body



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

32  
9

1. The first column **Method** is for the request type methods like GET, Post etc.
2. The second column **Request URL** will create the **URL/Endpoint** for your API
3. **Response code** will define the code you wish to get in response.
4. **Response Body** will have the response body that you want to show.

## How to create a mock server in Postman (4)

- Fill up the columns as shown in the image and click on **Next**.

Create a mock server

1. Select collection to mock    2. Configuration

[Create a new collection](#) [Select an existing collection](#)

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

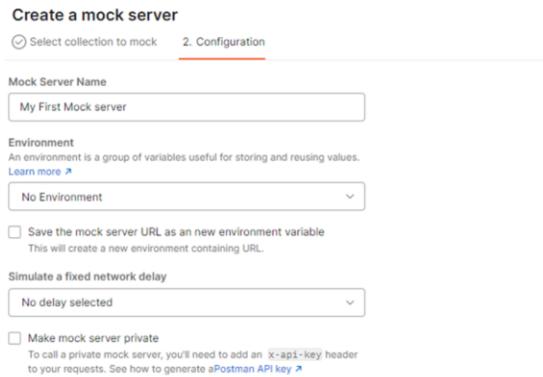
Request Method	Request URL	Response Code	Response Body	...
GET	(({url})) / getOrder	200 OK	This is a GET order response body.	
POST	(({url})) / createOrder	201 Created	This is a POST order response body.	
PUT	(({url})) / updateOrder	200 OK	This is a UPDATE order response body.	X
GET	(({url})) / Path	200	Response Body	

33  
0

In the server creation panel, instead of writing plain text, you can also write the response body in JSON format to get a different response format.

## How to create a mock server in Postman (5)

- Name your Mock Server as per your choice.



Create a mock server

Select collection to mock 2. Configuration

Mock Server Name  
My First Mock server

Environment  
An environment is a group of variables useful for storing and reusing values.  
Learn more ↗  
No Environment

Save the mock server URL as a new environment variable  
This will create a new environment containing URL.

Simulate a fixed network delay  
No delay selected

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 ↗

33  
1

### NOTE:

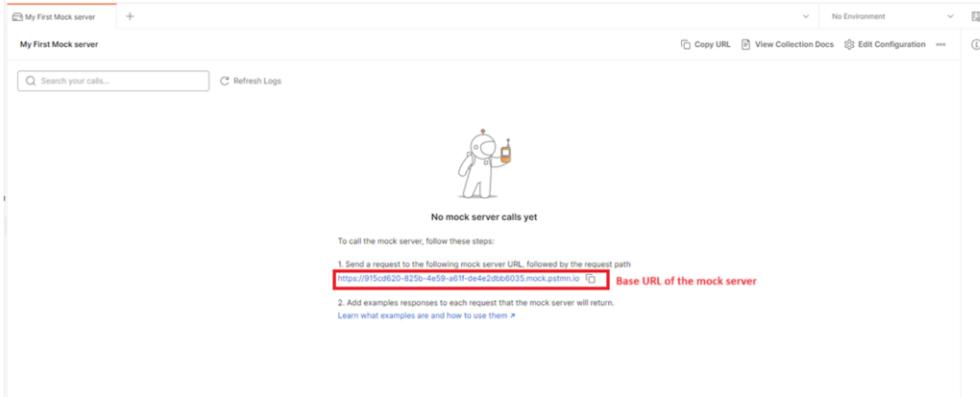
You can make the server private also if you don't want to make your information accessible by everyone, but it would require Postman API key to access the server.

For the beginning, we will keep our server public to reduce complexity.

If you want to save the URL as an environment variable, you can select the environment where you want to save the URL variable from the Environment dropdown and check the option **Save the mock server URL as a new environment variable**.

## How to create a mock server in Postman (6)

- After creation you will be directed to a tab where it gives you an overview of the mock server created where you can find the base URL of the mock server.



My First Mock server

My First Mock server

Search your calls... Refresh Logs

No mock server calls yet

To call the mock server, follow these steps:

- Send a request to the following mock server URL, followed by the request path  
<https://915ca620-8250-4e59-a1f1-de4e2bb6025.mock.postman.io>  Base URL of the mock server
- Add examples responses to each request that the mock server will return.  
Learn what examples are and how to use them

33  
2

## Mock server in Postman

- Along with the creation of the mock server, a **new collection** with the **same name** is created and with your **APIs** underneath it.



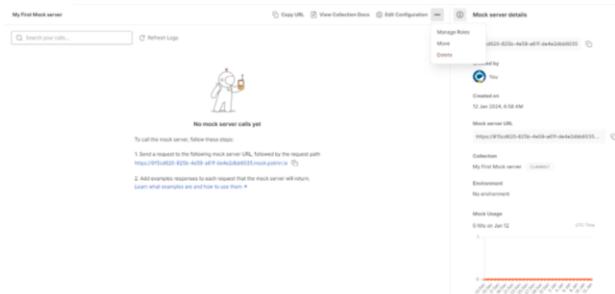
- You can also notice that a new collection variable called **url** has been created as well.



Variable	Initial value	Current value
url	https://915c6f20-823b-4e59-a1ff-de4a2bb6025...	https://915c6f20-823b-4e59-a1ff-de4a2bb6025.mock.prisma.io

## Managing mock server

- The following actions can be done for a mock server:
  - Rename
  - Move
  - Manage roles
  - Delete
  - Copy URL
  - Edit Configuration
  - View Collection Docs
  - View mock server details from the right sidebar



The screenshot shows the RUBRIC interface with a modal window titled "Mock server details" open. The main area displays a placeholder for a logo and instructions on how to call the mock server. The sidebar contains the following information:

- Mock server URL:** https://localhost:4200/mock-server
- Collection:** My First Mock server
- Environment:** No environment
- Mock Usage:** 0 ms on Jan 12 UTC time

## How to get the response in a different format in Mock Server? (1)

- It is very easy to get the response of Mock Server in other formats also.
  - In the server creation panel, instead of writing plain text or you can write the response body in JSON format, or you can also edit the default example under existing collection.

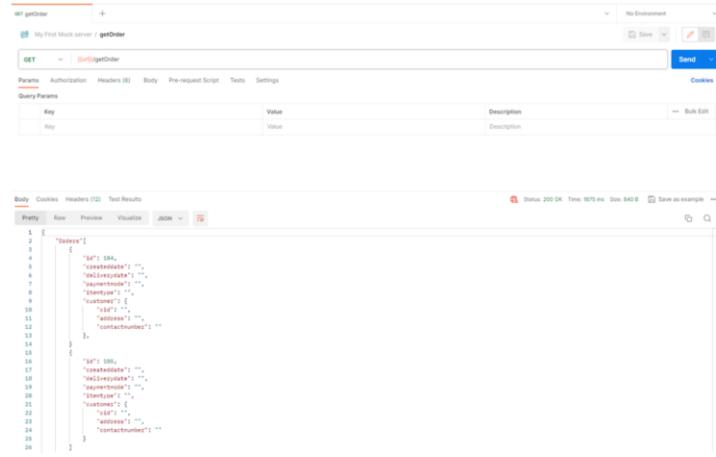
The screenshot shows the Postman interface with the following details:

- Collection:** My First Collection
- Request Type:** GET
- URL:** https://97.106.42.42:4439/api/v1/deals/20846035.mock.parms/superOrderInfo
- Headers:** Content-Type: application/json
- Body:** A JSON object representing a customer order with fields like id, name, address, and contactDetails.
- Status Code:** 200 OK

```
{
  "id": "1",
  "name": "John Doe",
  "address": "123 Main St",
  "contactDetails": {
    "phone": "+1234567890",
    "email": "john.doe@example.com"
  }
}
```

## How to get the response in a different format in Mock Server? (2)

- You will receive the response in **JSON** format now. You might get **HTML** directly but change the format to **JSON** from the dropdown to beautify the response as shown:



The screenshot shows a Mock Server interface with a request configuration and a response preview.

**Request Configuration:**

- Method: GET
- Path: /getOrder
- Headers: Content-Type: application/json
- Body: (empty)

**Response Preview:**

```

1   [
2     "Orders": [
3       {
4         "id": "584",
5         "orderdate": "",
6         "orderstatus": "",
7         "customer": "",
8         "customer": {
9           "name": "John Doe",
10          "address": "",
11          "contactname": ""
12        }
13      },
14      {
15        "id": "585",
16        "orderdate": "",
17        "orderstatus": "",
18        "customer": {
19          "name": "Jane Doe",
20          "address": "",
21          "customer": {
22            "name": "John Doe",
23            "address": "",
24            "contactname": ""
25          }
26        }
27      }
28    ]
29  ]

```

Status: 200 OK | Time: 1079 ms | Size: 840 B | Save as example