# **API Testing using POSTMAN**



APIs are everywhere, whether you realize it or not. They play a crucial role in simplifying communication between different software systems. But what exactly is an API?

[Canada post](https://www.linkedin.com/in/bhargav-s-363b362a/)

[e-commerce](https://www.linkedin.com/in/bhargav-s-363b362a/)

[application](https://www.linkedin.com/in/bhargav-s-363b362a/)

Customer,products,review, orders information

Order

Products

Customer

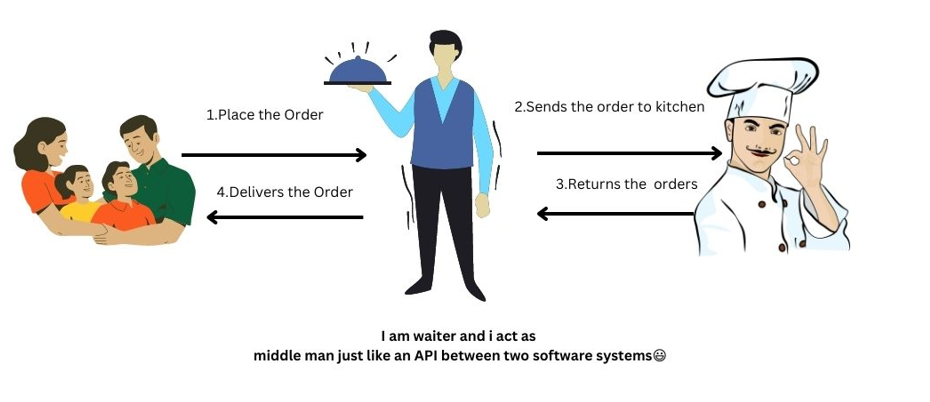
An API, or **A**pplication **P**rogramming **I**nterface, is like a language that software applications use to talk to each other. It's a set of rules and tools that enable different programs to exchange information and work together seamlessly.

Think of it this way: APIs are like the waiters at a restaurant. When you go to a restaurant, you don't directly communicate with the chefs in the kitchen. Instead, you place your order with the waiter, who then conveys your request to the kitchen staff. The waiter acts as an intermediary, ensuring that your order is understood and delivered correctly.

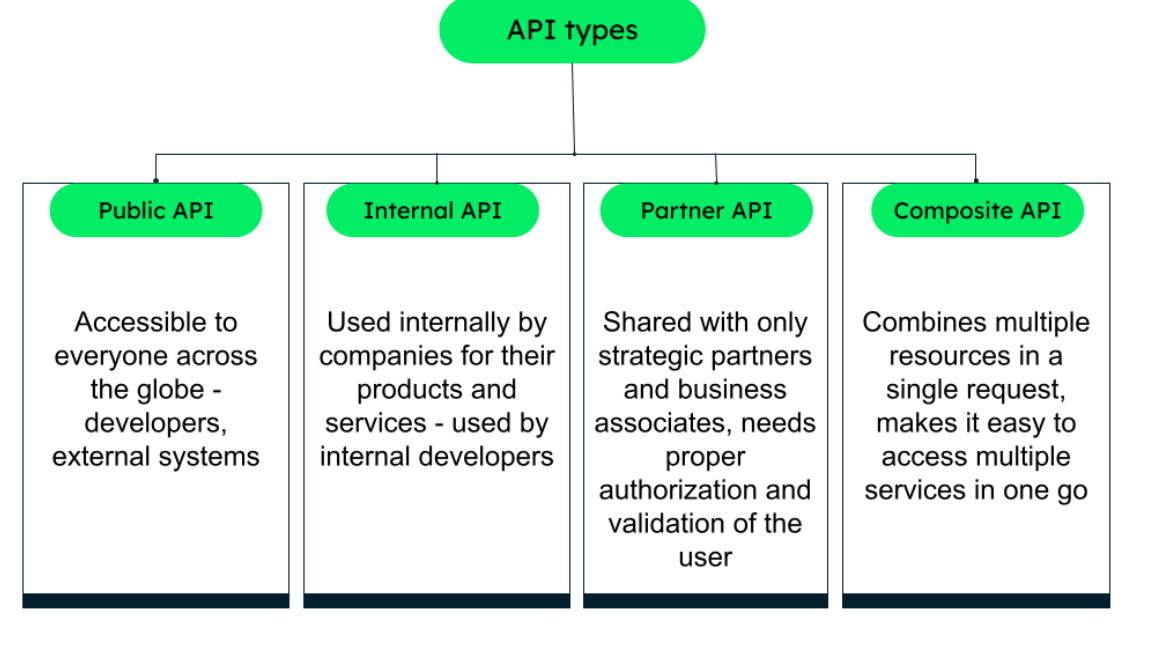
Similarly, **an API acts as a middleman between software applications**. It allows them to exchange data and instructions without needing to know the nitty-gritty details of how the other application works. APIs define the "language" that applications understand, making it easier for them to communicate effectively.

You've likely encountered APIs in various ways, such as when you use social media integration in an app, make an online payment, or access weather information. In each of these cases, APIs facilitate the communication between different software systems, making your experience smoother and more seamless.

So, APIs are the secret sauce that simplifies communication between software systems, allowing them to work together harmoniously and provide you with the services and features you enjoy.

~~~~

**Types of APIs**

****

**Internal API:**

**Short Answer:** An internal API refers to an Application Programming Interface (API) that is designed and intended to be used within a specific organization or system, typically for communication between different components, services, or modules within that organization or system.

**Detailed explanation:**

Consider a large e-commerce company with multiple departments and systems, such as inventory management, order processing, customer support, and shipping logistics. Each department has its own software applications and databases tailored to their specific needs. However, these systems need to interact and share information with each other efficiently.

To enable smooth communication and data exchange between these internal systems, the company can develop and utilize internal APIs. Internal APIs are APIs that are designed and used exclusively within an organization or company, allowing different internal systems to interact seamlessly.

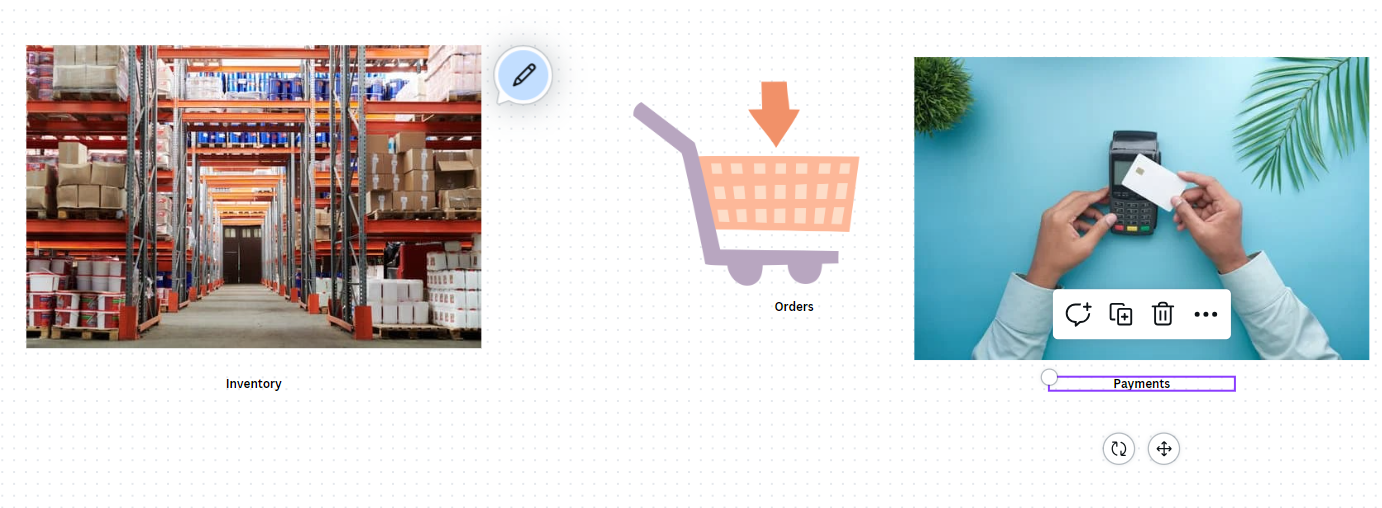
Here's an example use case to illustrate the concept:

Let's say a customer places an order on the e-commerce website. Once the order is confirmed, several internal processes need to kick in to fulfill the order, including inventory management, payment processing, and shipping logistics.

1. The e-commerce website's order management system receives the order details from the customer.
2. The order management system uses an internal API to communicate with the inventory management system, checking the availability of the ordered items.
3. The inventory management system responds to the API request, indicating whether the items are in stock or not.
4. If the items are available, the order management system proceeds to the next step.
5. The order management system uses another internal API to communicate with the payment processing system, initiating the payment transaction.
6. The payment processing system processes the payment and sends a confirmation response via the API.
7. Once the payment is confirmed, the order management system triggers an API call to the shipping logistics system, providing the shipping details and instructions.
8. The shipping logistics system receives the API request, generates a shipping label, and schedules the shipment.
9. The shipping logistics system responds to the API call with the shipment details and tracking information.
10. The order management system updates the order status and notifies the customer.

In this use case, internal APIs play a crucial role in enabling smooth coordination between different internal systems. They allow the order management system to interact with the inventory management, payment processing, and shipping logistics systems seamlessly, exchanging data and triggering the necessary actions to fulfill the customer's order.

Internal APIs provide a standardized and efficient way for internal systems to communicate, reducing dependencies and simplifying the integration process. They promote modularity, flexibility, and reusability of software components within the organization, making it easier to maintain and enhance the overall system.

****

**Public API**

**Short answer:**   
Public APIs are interfaces provided by companies or organizations that allow external developers to access and utilize their services, data, or functionality in their own applications.

**Explanation:**

Suppose you have an e-commerce application that sells products online and ships them to customers. To provide accurate shipment time estimates, you decide to integrate the Google Maps API into your application.

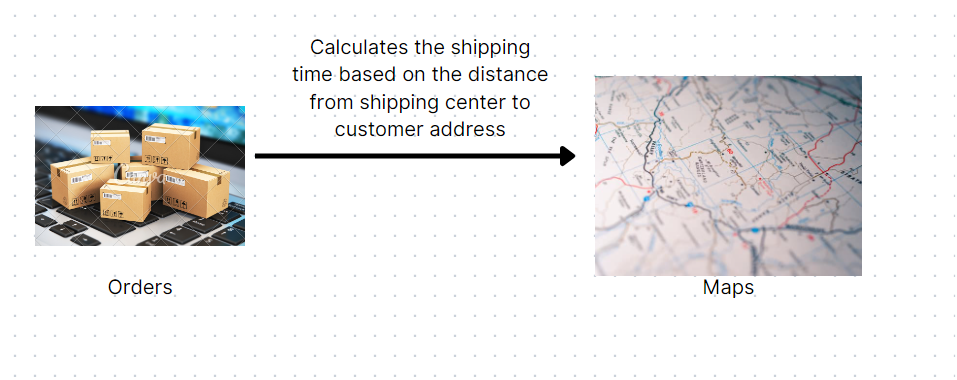
Here's how the usage of the public API would work:

1. When a customer places an order on your e-commerce application, you collect their shipping address.
2. Your application utilizes the Google Maps API by sending a request that includes the customer's address and the destination address (your warehouse or the customer's location).
3. The Google Maps API receives the request and calculates the distance and estimated travel time between the two addresses, taking into account factors like traffic and route options.
4. The Google Maps API generates a response containing the estimated travel time for the shipment.
5. Your e-commerce application receives the response from the API and extracts the estimated shipment time.
6. Your application displays the estimated shipment time to the customer during the checkout process or order confirmation page.

By leveraging the Google Maps API, your e-commerce application can provide customers with valuable information about when they can expect their order to be delivered. This enhances the transparency and customer experience by setting realistic expectations and helping customers plan accordingly.

Public APIs, like the Google Maps API, provide access to a wealth of functionality and data that can be leveraged by developers. In this case, the API enables your e-commerce application to calculate accurate shipment times based on real-time traffic and distance calculations.

Please note that although public API typically refers to APIs that are publicly available for developers to integrate into their applications but it requires developers to obtain credentials (such as an API key) to authenticate and track their usage

****

**Some common public apis**

1. Google Maps API: Provides mapping and location-based services, allowing developers to integrate interactive maps, geocoding, and route calculations into their applications.
2. Twitter API: Enables developers to interact with Twitter's functionalities, such as accessing user timelines, posting tweets, retrieving tweets, and searching for tweets based on specific criteria.
3. Facebook Graph API: Allows developers to integrate Facebook's features into their applications, including user authentication, retrieving user information, posting on timelines, and accessing social graph data.
4. OpenWeatherMap API: Provides access to real-time and forecast weather data for locations worldwide, allowing developers to retrieve weather information such as temperature, humidity, and conditions.
5. GitHub API: Allows developers to interact with the GitHub platform, enabling tasks such as managing repositories, accessing user profiles, creating issues, and retrieving code snippets.
6. Stripe API: Facilitates online payment processing by providing a secure and easy-to-use interface for developers to handle payment transactions, manage customers, and handle subscriptions.
7. Spotify API: Enables developers to access the vast music catalog of Spotify, providing functionalities like searching for songs, retrieving artist information, creating playlists, and streaming music.
8. YouTube Data API: Allows developers to integrate YouTube's features into their applications, including searching for videos, retrieving video details, uploading videos, and managing playlists.
9. Twilio API: Enables developers to incorporate messaging, voice, and video communication capabilities into their applications, allowing tasks such as sending SMS messages, making phone calls, and handling two-factor authentication.
10. NASA API: Provides access to a wide range of NASA data, including images, videos, space mission information, and astronomy datasets.

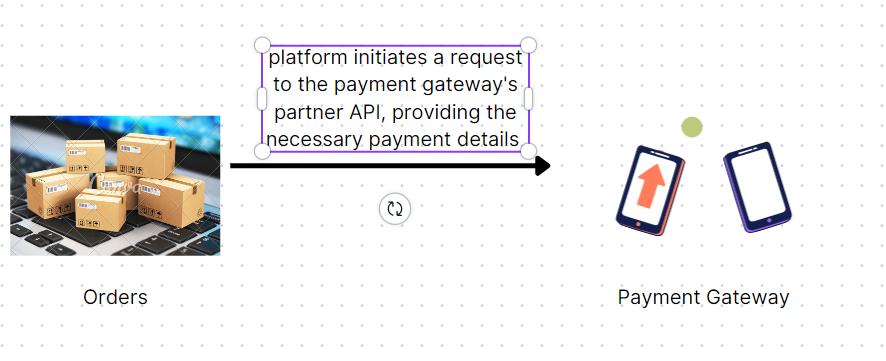
**Partner API use case1**

**Short Explanation**: facilitate communication and integration between a company's platform or services and trusted partner organizations, granting them controlled access to certain functionalities or data.

A common use case of a partner API is seen in the integration between e-commerce platforms and payment gateways. Let's consider an example:Suppose you run an e-commerce platform that allows businesses to sell products online. To process online payments securely and efficiently, you partner with a payment gateway provider. The payment gateway provider offers a partner API that allows your platform to communicate and exchange information with their payment processing system. Here's how the partner API would be utilized in this scenario:

1. customer places an order on a seller's online store hosted on your e-commerce platform.
2. When the customer proceeds to the payment step during checkout, your such as the order amount and customer information.
3. The payment gateway's partner API receives the request and securely communicates with the payment gateway's system to process the payment transaction.
4. The payment gateway's system performs the necessary validation, authentication, and transaction processing based on the received data.
5. The payment gateway's system generates a response indicating the status of the transaction, such as success or failure, along with any additional relevant details.
6. The partner API sends the response back to your e-commerce platform, allowing you to handle the payment status and update the order accordingly.
7. Based on the response, your platform notifies the customer about the success or failure of the payment transaction and proceeds with order fulfillment accordingly.

In this use case, the partner API provided by the payment gateway enables seamless integration between your e-commerce platform and their payment processing system. It allows your platform to securely transmit payment data, receive transaction status updates, and provide a smooth payment experience to customers.

****

**Partner API use case 2**

Consider a travel booking platform that partners with airlines to offer flight bookings to its customers. The travel booking platform relies on the partner API provided by the airlines to access their flight availability, pricing, and booking systems.

Here's how the partner API would be used in this scenario:

1. A customer visits the travel booking platform and searches for flights for their desired destination and travel dates.
2. The travel booking platform utilizes the partner API of its airline partners to make requests for flight availability and pricing.
3. The partner API receives the request and communicates with the airline's systems to retrieve the available flights, fares, and other relevant details.
4. The partner API sends a response back to the travel booking platform, containing the flight options, prices, and any additional information provided by the airline.
5. The travel booking platform processes the response and presents the flight options to the customer, including details like departure times, durations, layovers, and prices.
6. The customer selects a flight and proceeds to book it on the travel booking platform.
7. The travel booking platform uses the partner API to initiate the booking request with the airline, providing the customer's details and payment information.
8. The partner API communicates with the airline's booking system to confirm the reservation and complete the booking process.
9. The partner API returns a response to the travel booking platform, indicating the status of the booking (e.g., success, pending, or failure).
10. The travel booking platform notifies the customer about the booking status and provides the necessary booking details, such as the flight itinerary and confirmation number.

In this use case, the partner API allows the travel booking platform to connect with airline systems and provide a seamless flight booking experience to its customers. It enables the platform to search for available flights, retrieve pricing information, and initiate the booking process, all through the integration with the partner airline's API.

Partner APIs enable businesses to collaborate with external partners, such as airlines, to expand their services and offerings. They streamline processes, provide access to partner functionalities and data, and enhance the overall user experience by integrating different systems seamlessly.

**Composite API**

**Short Explanation:**

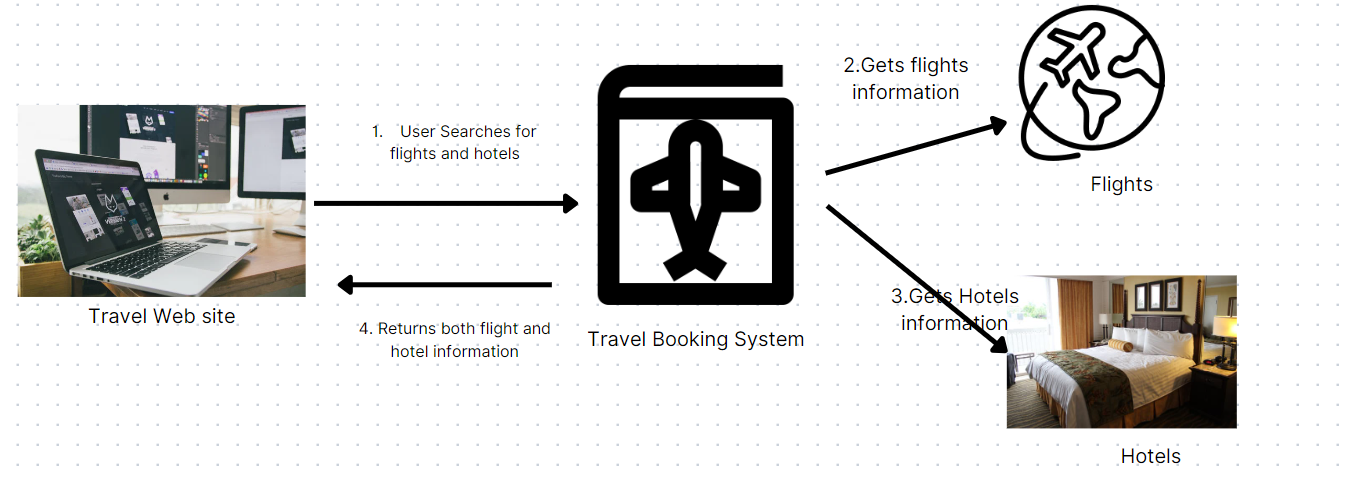
Composite APIs are valuable in scenarios where multiple services or providers need to be integrated to offer a comprehensive solution. They consolidate data and functionalities from various sources, simplify the integration process, and provide a cohesive experience to both the platform and its users.

Consider a travel booking platform that aims to provide a comprehensive booking experience by integrating with various travel service providers, such as airlines, hotels, and car rental agencies. To streamline the integration process and offer a unified interface, the travel booking platform can leverage a composite API.

Here's how the composite API would be used in this scenario:

1. Flight Search and Booking: The travel booking platform utilizes the composite API to search for flights across multiple airlines. The composite API interacts with the individual APIs of different airlines, retrieves flight schedules, prices, and availability, and presents the consolidated results to the platform. It also handles the booking process by communicating with the airline APIs to reserve seats and generate booking confirmations.
2. Hotel Search and Booking: When a user searches for accommodation, the travel booking platform uses the composite API to access hotel availability and pricing from various hotel providers. The composite API interacts with the hotel APIs, retrieves information about available rooms, rates, and amenities, and presents a unified view of hotel options to the platform. It also handles the booking process by communicating with the hotel APIs to reserve rooms and generate booking confirmations.
3. Car Rental: To provide car rental services, the travel booking platform integrates with multiple car rental agencies. The composite API interacts with the APIs of these agencies, retrieves information about available vehicles, rental rates, and pick-up/drop-off locations, and presents the consolidated options to the platform. It also handles the booking process by communicating with the car rental APIs to reserve vehicles and generate booking confirmations.
4. Trip Itinerary: The travel booking platform utilizes the composite API to generate a comprehensive trip itinerary for users. It combines flight details, hotel reservations, and car rental information obtained from the respective APIs. The composite API aggregates this data, organizes it into a coherent itinerary, and presents it to the platform and the users.

By utilizing a composite API in this travel booking use case, the platform simplifies the integration process with multiple travel service providers. It abstracts the complexities of interacting with individual APIs, allowing the platform to provide a unified and seamless booking experience to its users. The composite API streamlines the search, booking, and itinerary generation processes, enhancing the functionality and user experience of the travel booking platform.

****

**What is API testing**

API testing refers to the process of testing the functionality, reliability, performance, and security of an API (Application Programming Interface). It involves sending requests to an API endpoint and validating the responses received against expected results.

API testing focuses on examining the behavior and capabilities of the API itself, rather than testing the user interface or the entire system. It ensures that the API functions correctly, handles various input scenarios, and produces accurate and expected outputs.

The primary objectives of API testing include:

1. **Functional Testing**: Verifying that the API functions as intended, including proper handling of requests, validation of input data, and generation of appropriate responses.
2. **Integration Testing**: Testing the interaction between the API and other components or systems it communicates with, such as databases, external services, or third-party APIs.
3. **Performance Testing**: Assessing the performance and scalability of the API by simulating heavy loads, concurrent requests, and measuring response times, throughput, and resource utilization.
4. **Security Testing**: Evaluating the security measures implemented in the API, including authentication, authorization, encryption, and protection against common vulnerabilities like SQL injection or cross-site scripting (XSS).
5. **Error Handling**: Ensuring that the API handles error conditions gracefully by returning appropriate error codes, messages, and error response formats.

API testing can be performed manually or automated using specialized tools or frameworks. Automated API testing is particularly useful for repetitive or complex test scenarios, as it allows for efficient test execution and provides faster feedback on the API's functionality.

Common techniques used in API testing include:

1. Unit Testing: Testing individual API methods or functions in isolation to validate their behavior and functionality.
2. Integration Testing: Verifying the interaction between the API and other components or systems by testing end-to-end scenarios.
3. Parameter and Boundary Testing: Testing various input combinations, including valid and invalid inputs, to validate how the API handles different scenarios.
4. Performance Testing: Conducting load testing, stress testing, or endurance testing to evaluate the API's performance under different workloads and stress conditions.
5. Security Testing: Assessing the API's security features and vulnerabilities by performing security scans, penetration testing, and vulnerability assessments.

By thoroughly testing APIs, developers and testers can ensure that the APIs are robust, reliable, and compliant with the expected functionality and performance requirements. It helps identify and address issues early in the development lifecycle, leading to higher-quality API implementations and improved overall system reliability.

**Tools used for API testing**

There are several tools available to test APIs, ranging from specialized API testing frameworks to general-purpose testing tools. Here are some commonly used tools for API testing:

1. **Postman**: Postman is a widely-used tool for API testing and development. It provides a user-friendly interface to send requests, set up test cases, and validate API responses. Postman supports various request methods, authentication types, and allows for easy creation of test scripts.
2. REST Assured: REST Assured is a Java-based testing framework specifically designed for testing RESTful APIs. It provides a rich set of methods and features to easily construct API requests, extract response data, and perform assertions. REST Assured integrates well with popular testing frameworks like JUnit and TestNG.
3. SoapUI: SoapUI is a comprehensive API testing tool that supports both REST and SOAP protocols. It offers a graphical interface to create and execute test cases, generate test data, and validate API responses. SoapUI also provides features like data-driven testing, security testing, and load testing.
4. JMeter: Apache JMeter is a versatile tool primarily used for performance testing, but it can also be used for API testing. It allows you to create test scenarios with various requests, simulate concurrent users, measure response times, and analyze performance metrics for APIs.
5. Newman: Newman is a command-line tool for running Postman collections as part of automated API testing. It allows you to integrate API testing into continuous integration and deployment pipelines, making it useful for automating API tests and incorporating them into your build process.
6. Karate: Karate is an open-source API testing framework that uses a Behavior-Driven Development (BDD) approach. It enables testers to write expressive and readable tests in a simple syntax. Karate supports both REST and SOAP APIs and provides features like request/response validation, test data management, and parallel test execution.
7. Insomnia: Insomnia is a powerful API testing and debugging tool that offers a visually appealing interface for constructing requests, inspecting responses, and managing API workspaces. It supports multiple authentication methods, environments, and allows for organizing and sharing API collections.

**API Types**

1. Web APIs: Web APIs, also known as **HTTP** APIs or **RESTful** APIs, are designed to enable communication and data exchange over the internet using the HTTP protocol. These APIs provide access to web-based services, resources, or functionalities and are typically based on standards like HTTP, JSON, or XML. Web APIs are widely used for building client-server applications, mobile apps, and integrating different systems.
2. SOAP APIs: SOAP (Simple Object Access Protocol) APIs are based on the XML-based messaging protocol called SOAP. These APIs define a set of rules for structuring messages, performing remote procedure calls (RPCs), and exchanging data between systems. SOAP APIs often use the XML Schema Definition (XSD) to define message structures and Web Services Description Language (WSDL) to describe the API's interface.
3. GraphQL APIs: GraphQL is a query language and runtime for APIs that was developed by Facebook. It allows clients to request specific data and shape the response according to their needs, reducing the problem of over-fetching or under-fetching data. GraphQL APIs provide a flexible and efficient way of fetching data from servers and have gained popularity for their ability to optimize data retrieval and minimize network traffic.
4. WebSocket APIs: WebSocket APIs facilitate real-time, bidirectional communication between a client and a server. Unlike traditional HTTP-based APIs, which follow a request-response model, WebSocket APIs establish a persistent connection that enables both the server and the client to send data to each other asynchronously. WebSocket APIs are commonly used in applications that require real-time updates, such as chat applications, collaborative tools, and live dashboards.
5. Library or Framework APIs: Library or framework APIs provide pre-built functions, classes, and methods that developers can utilize within their applications. These APIs are often specific to a particular programming language, framework, or library, and they abstract complex functionalities, making them easier to use and integrate into applications. Examples include the Python Standard Library, jQuery API, or Java Development Kit (JDK) APIs.
6. Internal/Private APIs: Internal or private APIs are designed for internal use within an organization or a specific system. These APIs are not exposed to external developers or public consumption. Internal APIs facilitate communication between different components, services, or modules within an application or across different applications within an organization.
7. Public APIs: Public APIs are exposed to external developers and the public for consumption. They allow developers to access and utilize the functionalities or data provided by a platform, service, or organization. Public APIs enable the integration of third-party applications, the development of add-ons or extensions, and the creation of new services or products that leverage the capabilities of the API provider.

These are some of the common types of APIs, each with its own characteristics, protocols, and intended usage. The choice of API type depends on the specific requirements, technologies involved, and the purpose of the API implementation.

Lets understand RESTFUL web services with postman which is most widely used across industry. HTTP protocol is used for communication for RESTFUL web service

**What is HTTP?**

HTTP (Hypertext Transfer Protocol) is a protocol that governs the communication between a client (such as a web browser) and a server over the internet. It is the foundation of data communication on the World Wide Web and plays a crucial role in API development and usage.

As a beginner, here are some key aspects of HTTP that you should know:

1. **Request-Response Model:** HTTP follows a request-response model, where the client sends a request to the server, and the server responds with a corresponding response. The request contains information such as the HTTP method (e.g., GET, POST, PUT, DELETE), URL, headers, and optionally, a body. The server processes the request and sends back a response with a status code, headers, and optionally, a response body.
2. **Stateless Protocol**: HTTP is a stateless protocol, which means that each request is independent of any previous requests. The server does not maintain any information or state about the client between requests. This design simplifies scalability and allows for better fault tolerance in distributed systems.
3. **URL Structure**: URLs (Uniform Resource Locators) are used to identify resources on the web. An HTTP request includes a URL that specifies the address and location of the resource being requested. URLs consist of a protocol (e.g., http://), domain name (e.g., [www.example.com](http://www.example.com/)), path (e.g., /api/users), and optional query parameters (e.g., ?name=John&age=25).
4. **Methods**: HTTP defines various methods, also known as verbs, that indicate the type of operation to be performed on a resource. The most commonly used methods in API development are:
   * GET: Retrieves data from a server.
   * POST: Sends data to a server to create a new resource.
   * PUT: Updates an existing resource with new data.
   * DELETE: Removes a resource from the server.
5. **Status Codes**: HTTP responses include status codes that indicate the outcome of the request. These three-digit codes provide information about whether the request was successful, encountered an error, or requires further action. For example, 200 OK signifies a successful request, while 404 Not Found indicates that the requested resource was not found.

The importance of HTTP in the context of APIs lies in its role as a foundation for communication. APIs typically expose functionality and data over HTTP, allowing clients to interact with the API's endpoints using HTTP requests and receive responses in a standard format like JSON or XML. HTTP provides a standardized and widely supported mechanism for making API calls and exchanging data between clients and servers.

Understanding HTTP is essential for working with APIs as it helps you grasp the fundamentals of how information is exchanged between systems. It enables you to construct API requests, interpret responses, handle errors, and troubleshoot issues. Having a basic understanding of HTTP will empower you to effectively utilize APIs and develop applications that leverage their capabilities.

**Most common http methods:**



1. GET : The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

2. POST : A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

3. PUT : PUT is used to send data to a server to create/update a resource. Replaces all the current representations of the target resource with the uploaded content.

4. PATCH : PATCH is used to update partial resources. For instance, when you only need to update one field of the resource, Putting a complete resource representation might be cumbersome and utilizes more bandwidth.

5. HEAD : HEAD is almost identical to GET, but without the response body. HEAD transfers the status line and the header section only.

6. DELETE : The DELETE method deletes the specified resource.

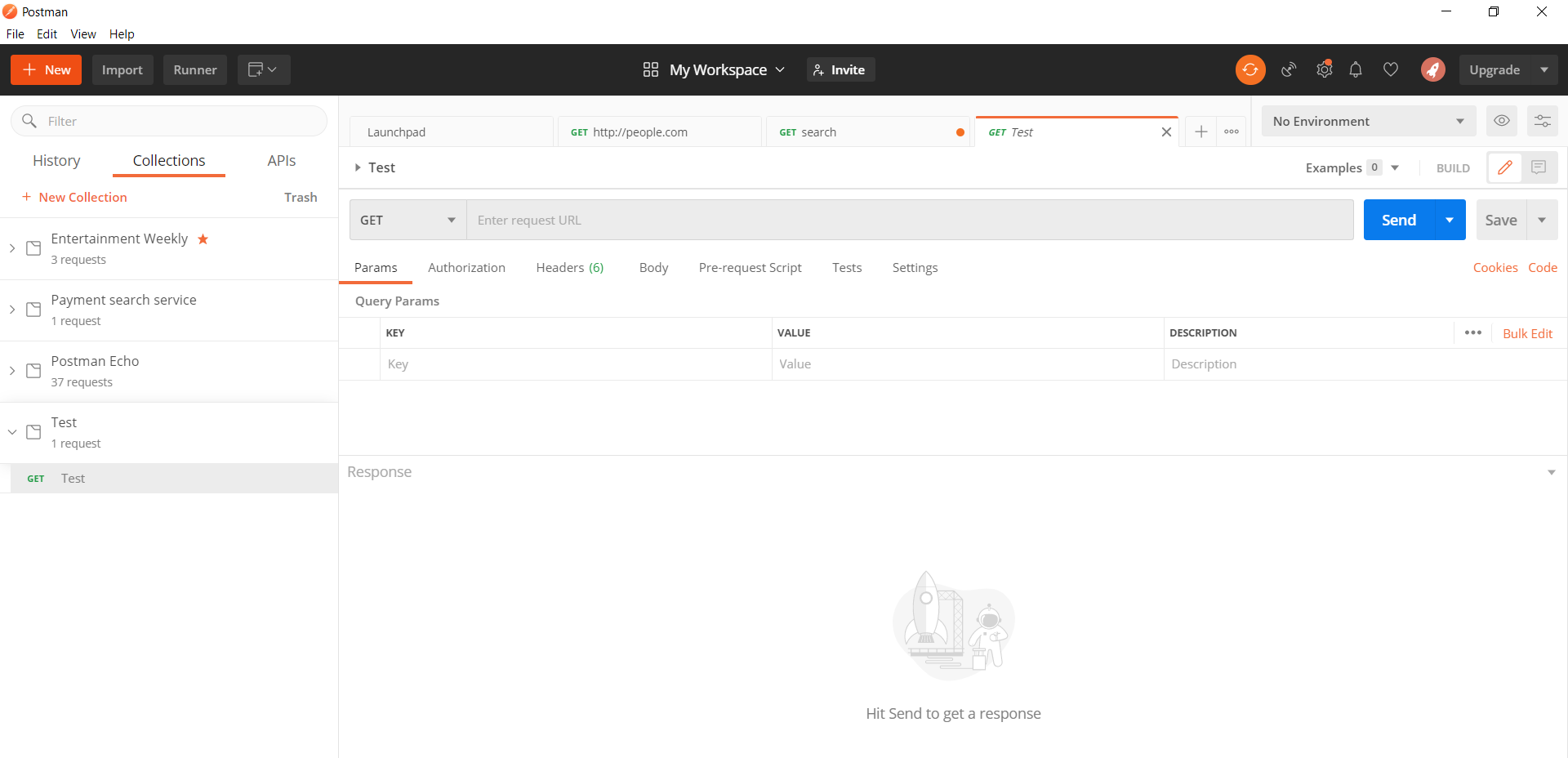
7. OPTIONS : The OPTIONS method describes the communication options for the target resource.

**Install native Postman Application**

Postman for Mac/Windows/Linux:

Go to <https://www.getpostman.com/apps> and download the application based on the OS you are using and follow the steps prompted to successfully install the Postman application.

After you have installed Postman successfully, your postman window should look like:



### **Making the first http request in Postman:**

Since we have installed the Postman app successfully, it is now time to start testing the API with Postman by making first ever HTTP request to server.

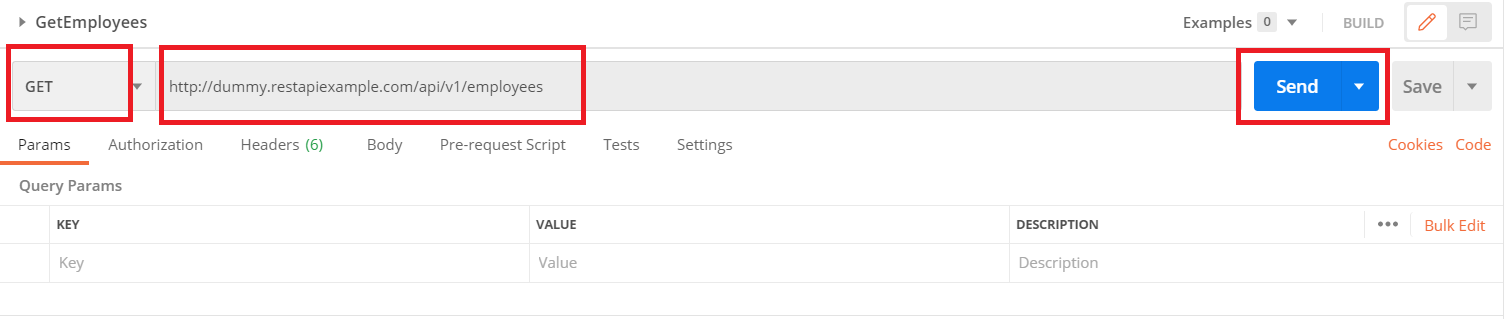
## **Testing GET Requests**

Let’s now jump directly to test those API’s. Suppose we have an API which fetches the user information of a particular application. To test this we will have to use GET request. The GET request is explained below:

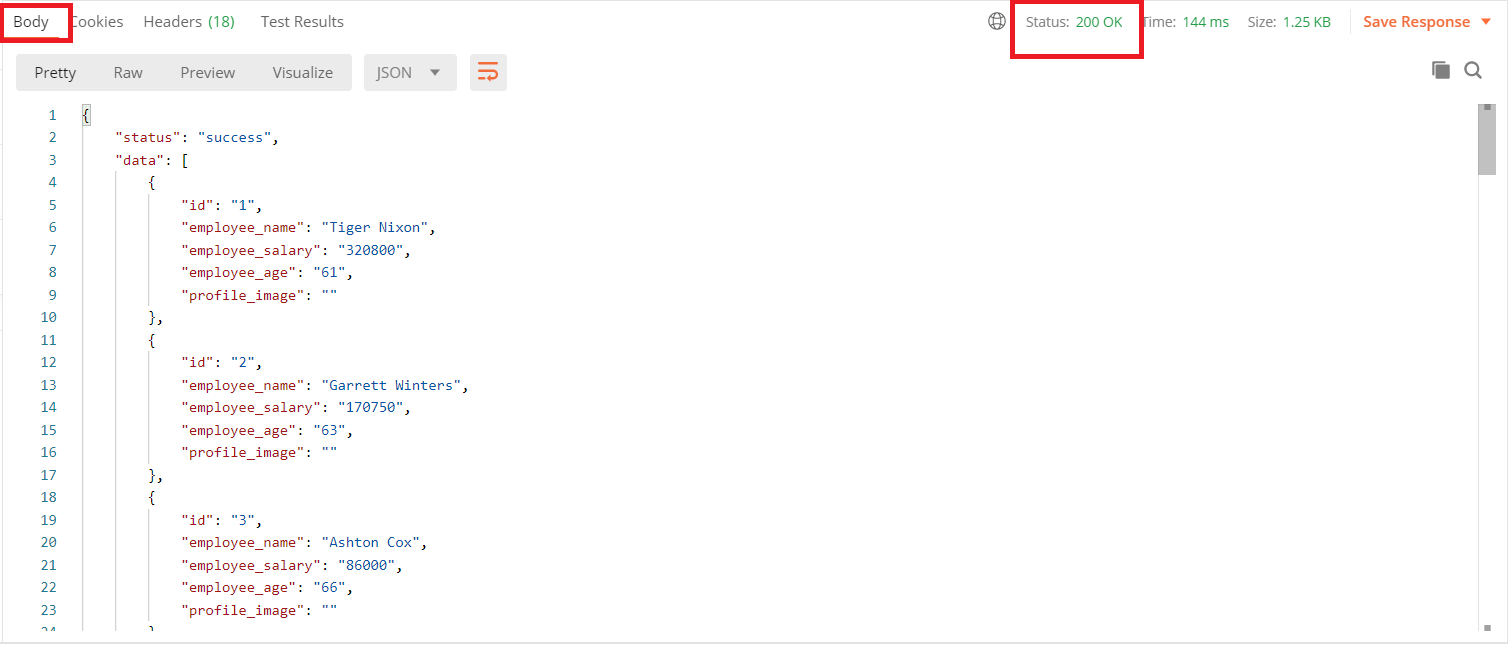
For sample requests, <http://dummy.restapiexample.com/api/v1/employees>

a. For making the first HTTP request(GET):

1. Make a collection in Postman - To make a collection in Postman, click on New-> Collection -> CollectionTest (Any Collection Name you wish) -> Create
2. Make a Request - To make a request, click on New->Request-> GetEmployees (Any request name you wish) -> Select the Collection you wish to save request in (Present in bottom of dialog box) -> Save to CollectionTest
3. By now, we have created our first request, now we need to pass different parameters in the request to get the expected response.
4. In the “Enter Request URL” text box type <http://dummy.restapiexample.com/api/v1/employees>
5. Click on “Send” Button



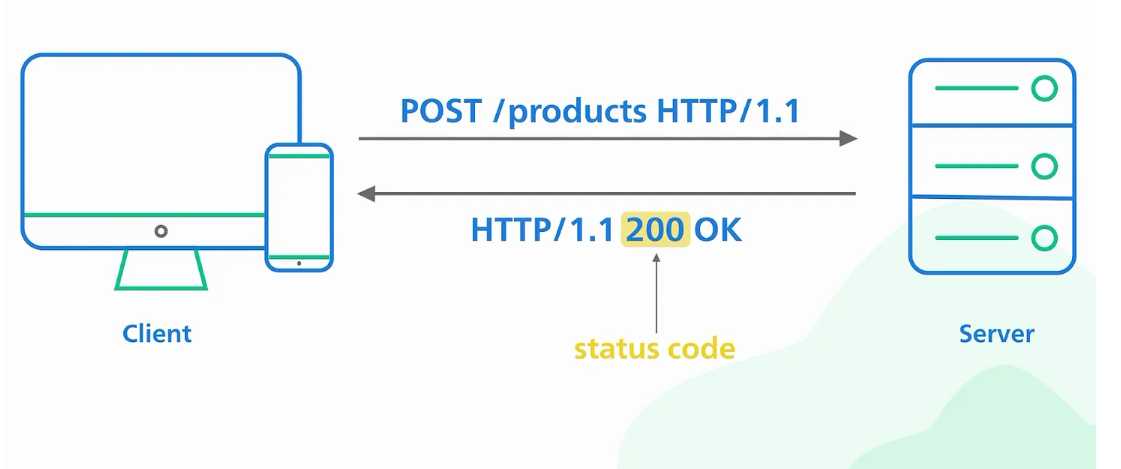
6. You should be able to see the below response in the Body section with status as 200 OK:



## **Testing POST Requests**

Now, suppose we need to create a user into a application that means we are sending data or feeding data to an application. For these type of requests we use POST request. In POST request we send data/parameter in the body of the request, and in response to that, API returns some data to us which validates the user has been created. The response can either be a success message or the id of the new user created and time when the user was created.

a. For making the first HTTP request(POST):



POST Request — To make a POST request, click on New->Request->CreateUser(Any request name you wish)->Select the Collection you wish to save request in(Present in bottom of dialog box)->Save to Collection Demo

From the Dropdown select POST

1. In the “Enter Request URL” text box, type : <http://dummy.restapiexample.com/api/v1/create>
2. Click on Body Tab and select “Raw” radio button
3. In the text box, paste :

{

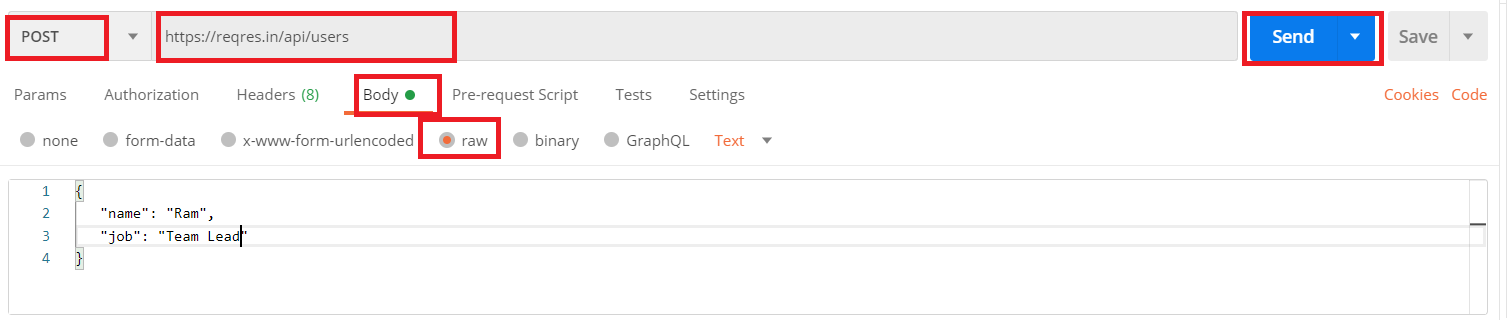
"name": "John Doe",

"salary": "5000",

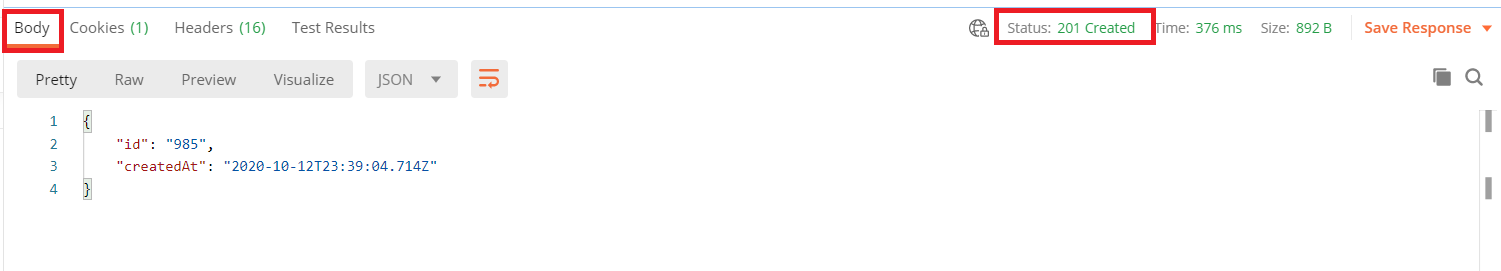
"age": "30"

}

1. Click on Send button



6. User should see the below response with status 201 Created:



You have successfully tested your POST request too, similarly you can try your hands with PUT, PATCH, DELETE etc.

1. Check for expected response.
2. Check for correct status code.
3. Check for Time (Response Time), it should be acceptable as per business.
4. Always perform negative tests to verify that the API doesn’t respond if data is tampered.

"idempotent" means that making multiple identical requests has the same effect as making a single request.

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Usage** | **Use case** |
| POST | create a new resource on the server. typically include the data for the new resource in the request body, and the server responds with the newly created resource and its associated identifier.**not idempotent** | * submitting an order * adding a product to a shopping cart * creating a new customer account |
| PUT | update or replace an entire existing resource on the server.you include the complete representation of the updated resource in the request body, **including any unchanged fields.**idempotent | * update the quantity of a product in a shopping cart |
| PATCH | update or modify specific attributes or fields of an existing resource. change a subset of the resource's **properties without sending the complete representation.**idempotent | * modifying customer information like phone number and email without all customer details |

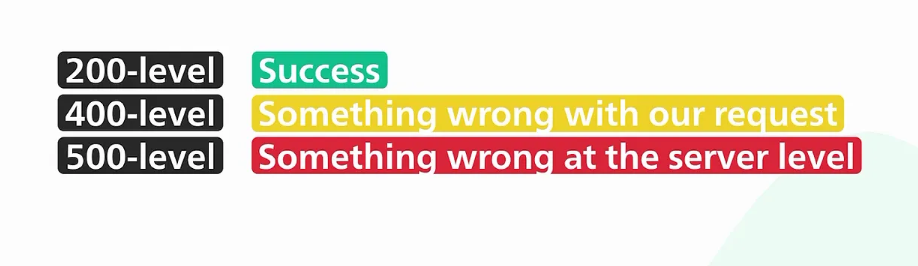
**Some public apis:**

<https://www.postman.com/cs-demo/workspace/public-rest-apis/collection/8854915-454a2dc7-dcbe-41cf-9bfa-da544fcd93a2>

|  |  |  |  |
| --- | --- | --- | --- |
| **Public URL** | **GET** | **POST URL & JSON** | **PUT Url & JSON** |
| https://jsonplaceholder.typicode.com/ | <https://jsonplaceholder.typicode.com/posts/1>  <https://jsonplaceholder.typicode.com/posts> | <https://jsonplaceholder.typicode.com/posts/2>  {  "title": "New Post",  "body": "This is a new post",  "userId": 1  } | <https://jsonplaceholder.typicode.com/posts/2>  {  "title": "New Post",  "body": "This is a new post",  "userId": 1  } |
| https://dummy.restapiexample.com/ | [**http://dummy.restapiexample.com/api/v1/employees**](http://dummy.restapiexample.com/api/v1/employees) | <http://dummy.restapiexample.com/api/v1/create>  {  "name": "John Doe",  "salary": "5000",  "age": "30"  } | [https://dummy.restapiexample.com/api/v1/update/21](https://dummy.restapiexample.com/api/v1/update/21/)  {  "name": "John Doe",  "salary": "5000",  "age": "30"  } |
| https://reqres.in/ | <https://reqres.in/api/users> | <https://reqres.in/api/users> | {  "name": "John Doe",  "salary": "5000",  "age": "30"  } |

Access github api:

Generate Access token(Fine grained tokens) using <https://github.com/settings/tokens?type=beta> It needs to have GITHUB user name and password.

HTTP Response Codes

Http Response Codes Summary

**200: OK. Everything worked as expected.**

**201: A resource was successfully created in response to a POST request. The Location header contains the URL pointing to the newly created resource.**

204: The request was handled successfully and the response contains no body content (like a DELETE request).

304: The resource was not modified. You can use the cached version.

**400: Bad request. This could be caused by various actions by the user, such as providing invalid JSON data in the request body, providing invalid action parameters, etc.**

**401: Authentication failed.**

**403: The authenticated user is not allowed to access the specified API endpoint.**

**404: The requested resource does not exist.**

405: Method not allowed. Please check the Allow header for the allowed HTTP methods.

415: Unsupported media type. The requested content type or version number is invalid.

422: Data validation failed (in response to a POST request, for example). Please check the response body for detailed error messages.

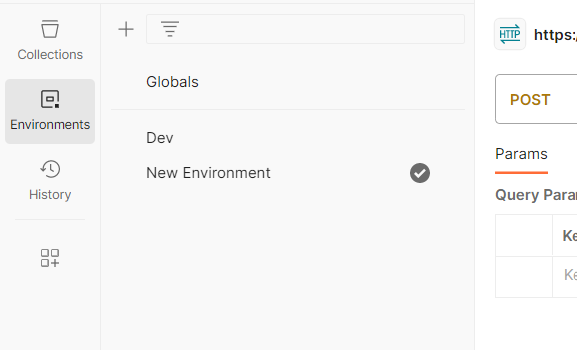
429: Too many requests. The request was rejected due to rate limiting.

500: Internal server error. This could be caused by internal program errors.

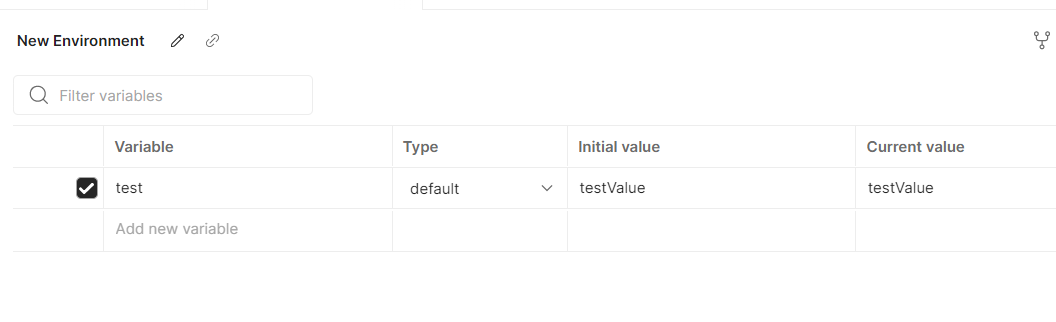
**Environment**

Lets see the why environment is needed first, when any application is developed it goes into the different stages like local(Developer’s setup),Development environment,Test environment,Staging and production. When we want to test the same end point with different url and parameters then we can create separate environment to each.

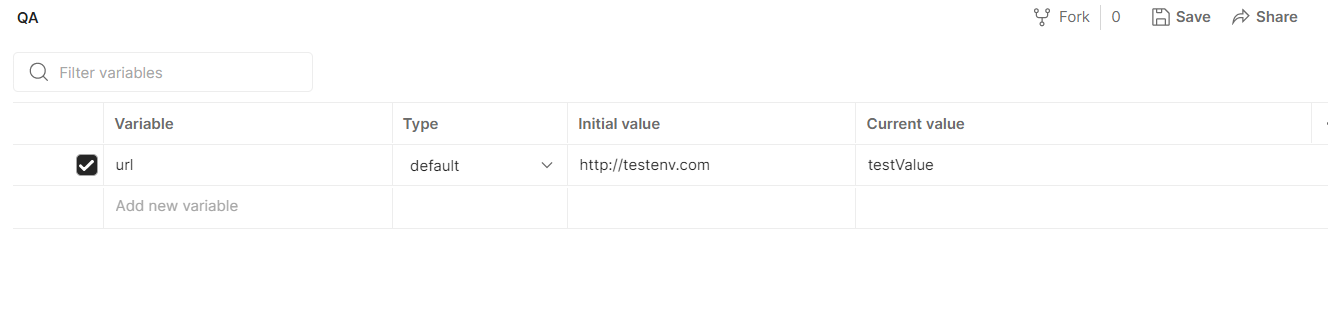
Click on “Environment” 🡪 New Environment ->



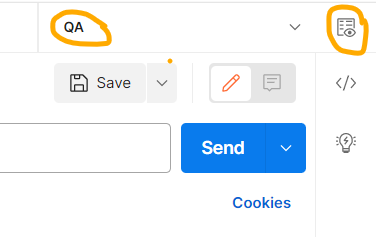
Click on Pencil icon next to “New Environment” to rename. In this case I am naming it as “Test”



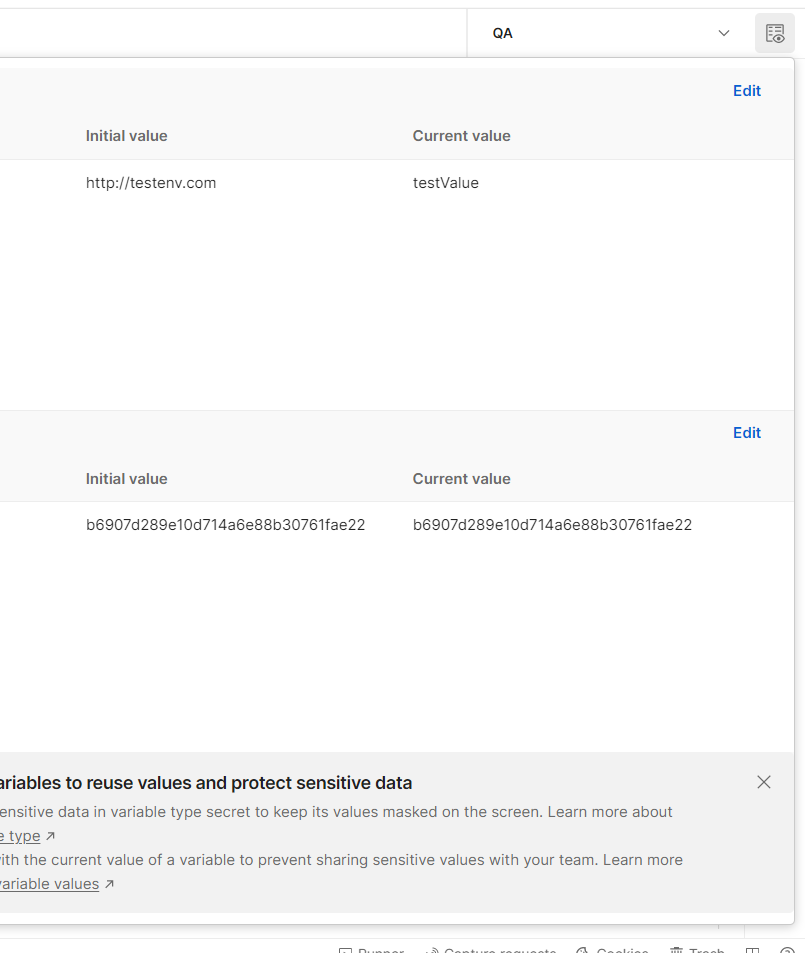
Here you can define desired variables, something like below and click on Save



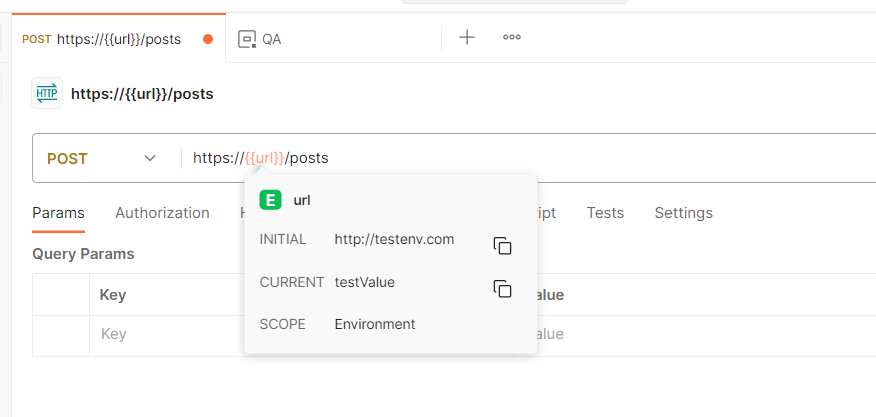
Now, select the environment on top right corner you will be able to see the newly created environment QA(picture is below)



Click on the eye icon to see the values inside QA environment



Now you can access the URL in any of the request using this environment



If you want to switch to test dev environment simply switch to environment where you have defined dev

In summary Environments helps to allow you to dynamically customize your requests by referencing their values in various parts of the request, such as headers, URLs, or request bodies

**Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Type** | **Usage** | **Example** | **Path to create** |
|  |  |  |  |
| Global Variables | Configure variables for different environments  Store values accessible across all environments and requests in the Postman workspace | **{{accessToken}}**  **{{username}}** | **Globals -> Edit** |
| Collection Variables | Share common values among different environments  Store values specific to a collection  Shareable across requests within that collection |  | Collection > edit > Variables tab |
| Environment Variables | Store values specific to an environment  Shareable across requests within an environment | **{{baseURL}}**  **{{apiKey}}** |  |
| Data Variables | Useful for data manipulation or scripting  Store data imported from external sources  Used in conjunction with data-driven testing  Enable iterating through different data sets | {{dataVar}}, {{dataFile}} |  |
| Local Variables | Shareable across requests within that collection  Store temporary values during request execution  Scope limited to a single request | **{{tempVar}}**  **{{response.body}}** |  |

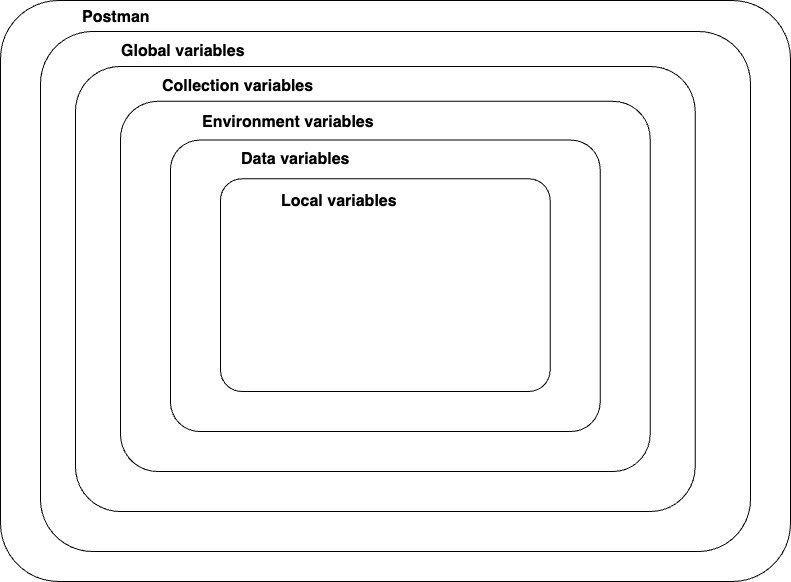
When creating variables you will have two values which are initial and current, below are the difference

1. Initial: The "Initial" column represents the initial value of a variable. This value is set when the variable is first created or initialized. The initial value can be manually set or automatically generated through a pre-request script, environment, or collection variable. The initial value remains constant unless it is explicitly changed during runtime.
2. Current: The "Current" column represents the current value of a variable. This value can be modified during the execution of requests or through scripts. The current value can be updated dynamically based on the responses received or through user-defined logic. The current value of a variable is subject to change throughout the execution of requests or test scripts.

The "Current" column reflects the most up-to-date value of a variable at any given point during the execution of requests or test scripts. It allows you to track and observe how the value of a variable may change as you work with requests and execute test scripts.

By differentiating between the "Initial" and "Current" values of variables, Postman enables you to manage and monitor the evolution of variables throughout the lifecycle of your API testing or development process.

Duplicate Variables



If a variable with the same name is declared in two different scopes, the value stored in the variable with narrowest scope will be used. For example, if there is a global variable named username and a local variable named username, the local value will be used when the request runs.

Dynamic variables: <https://learning.postman.com/docs/writing-scripts/script-references/variables-list/>

Sample tests:

//pm.globals.get("variable\_key");

/\*

pm.test("Status code is 200", function () {

    pm.response.to.have.status(200);

});

\*/

pm.**test**("Response time is less than 200ms", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(10);

});

pm.**test**("Not Found Expected", **function** () {

    pm.response.to.have.status(404);

});

pm.environment.**get**("variable\_key");

pm.globals.**get**("variable\_key");

pm.variables.**get**("variable\_key");

pm.collectionVariables.**get**("variable\_key");

pm.**test**("Status is 200", **function** () {

    pm.response.to.not.be.error;

    pm.response.to.have.jsonBody("");

    pm.response.to.not.have.jsonBody("error");

});

// Set array

var array **=** [1, 2, 3, 4];

pm.environment.**set**("array", JSON.**stringify**(array, **null**, 2));

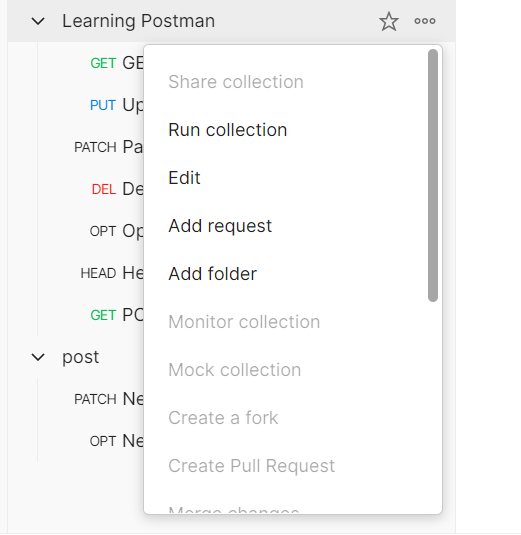
pm.test("Content-Type header is present", function () {

   pm.response.to.have.header("Content-Type");

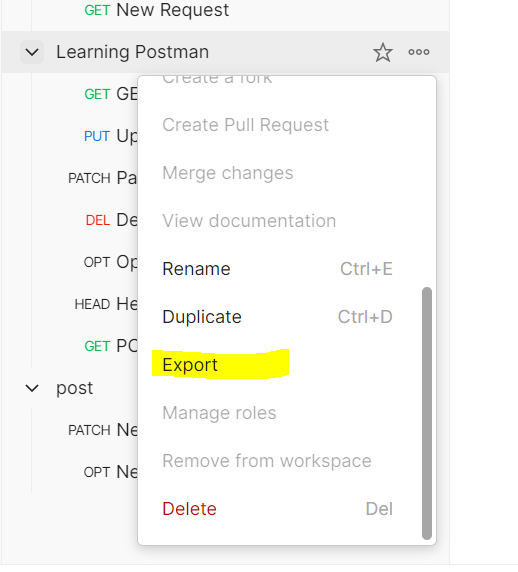
});

**Import and Export of postman collections:** You can share the postman collection by export and import.

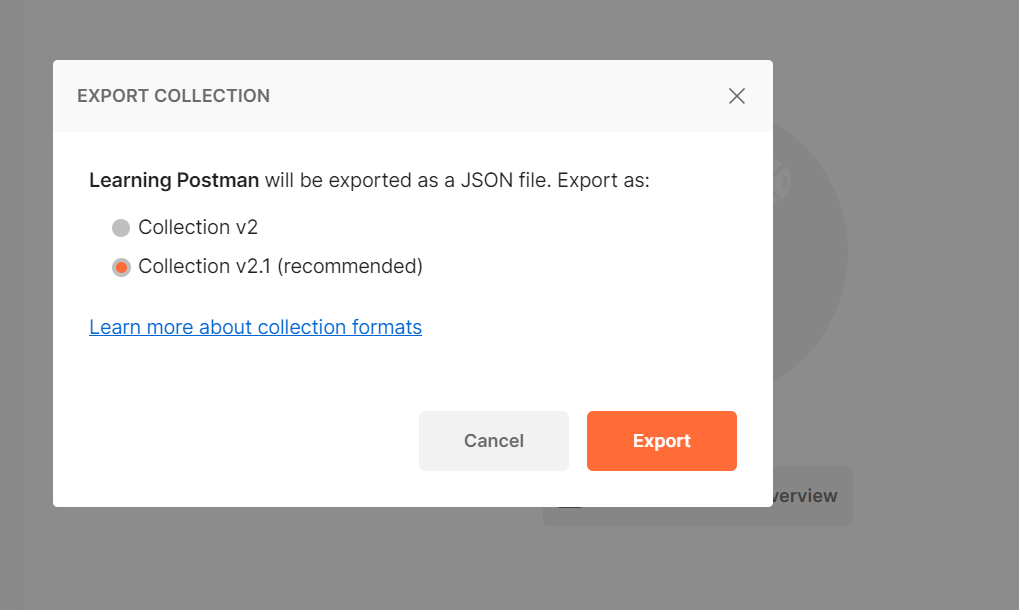
Click on three dots next to the postman collection you want to export



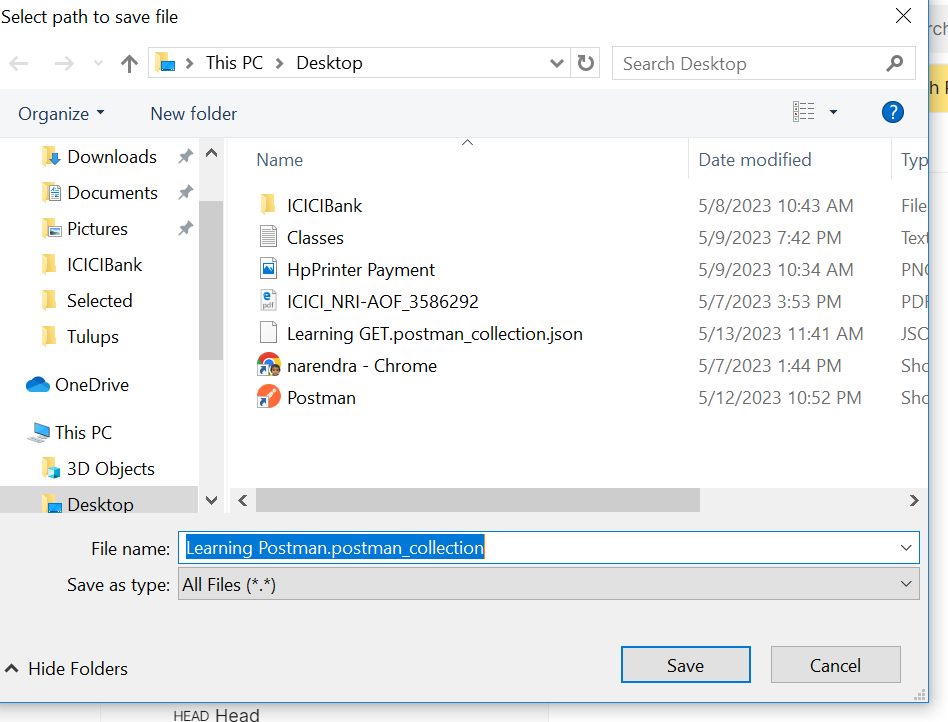
Scroll down to see “Export”



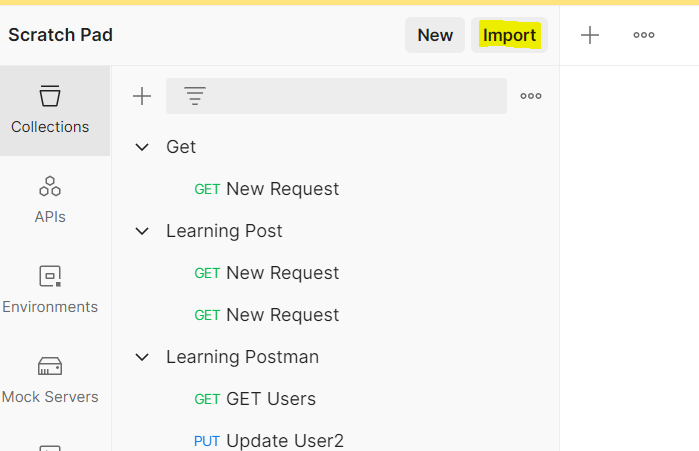
**Leave the default selection and click on export**

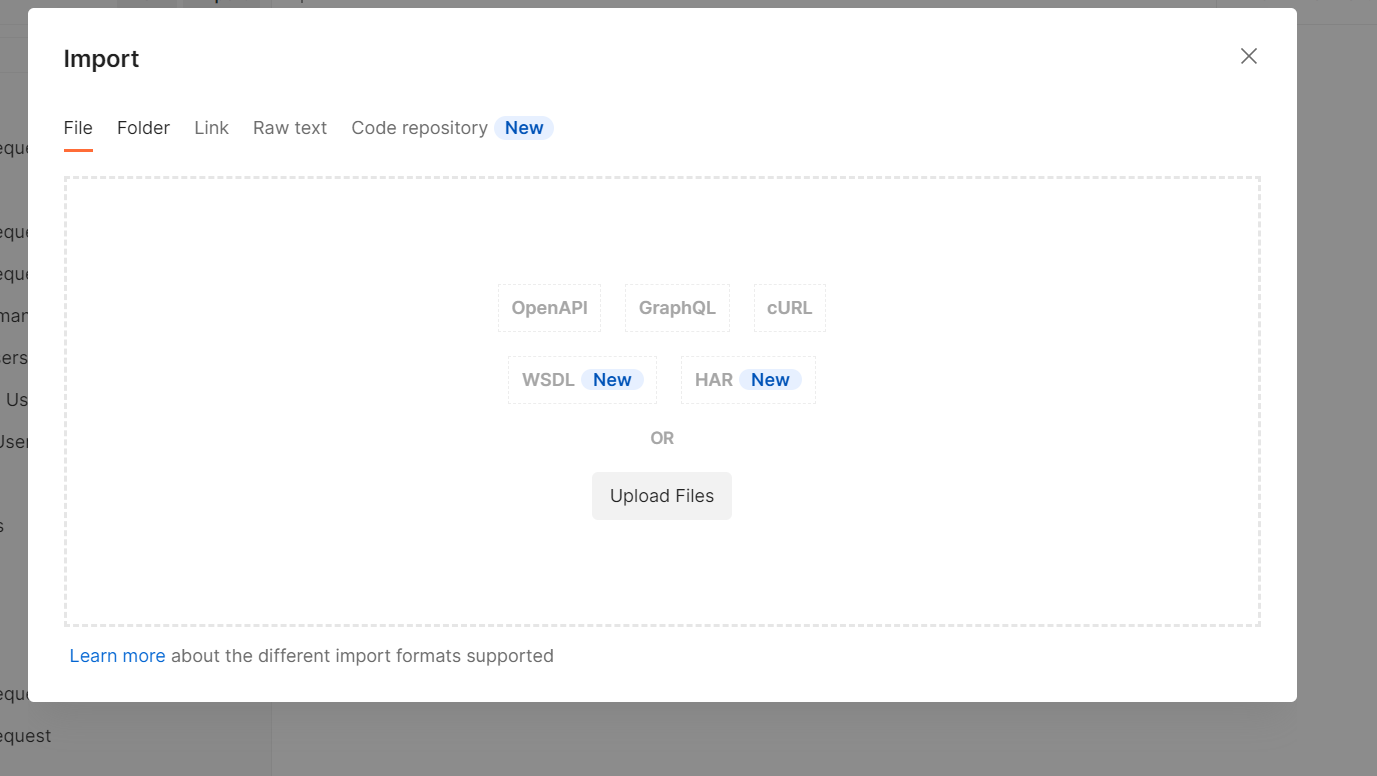
****

**Choose any desired path to save the postman collection(it is in JSON Format)**

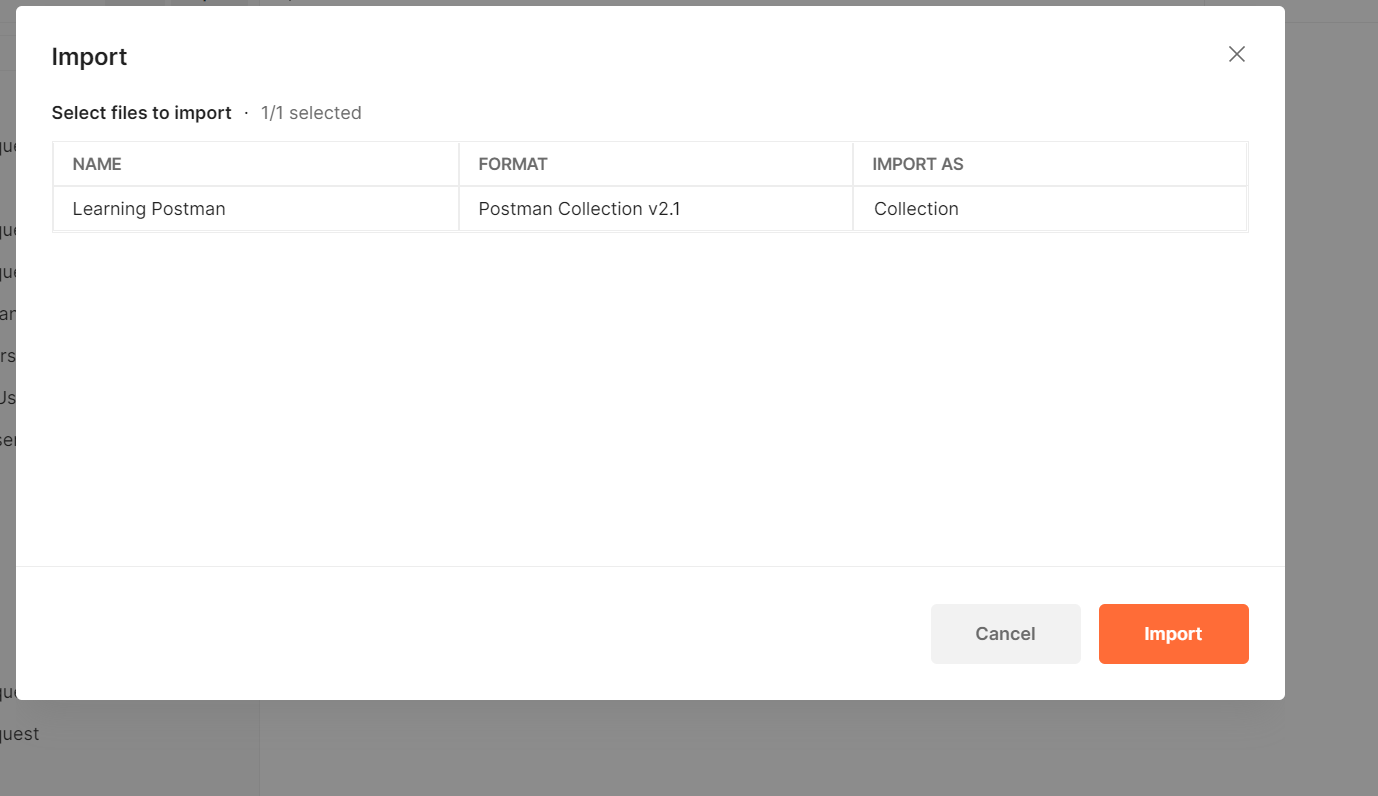
****

Import of Collection:

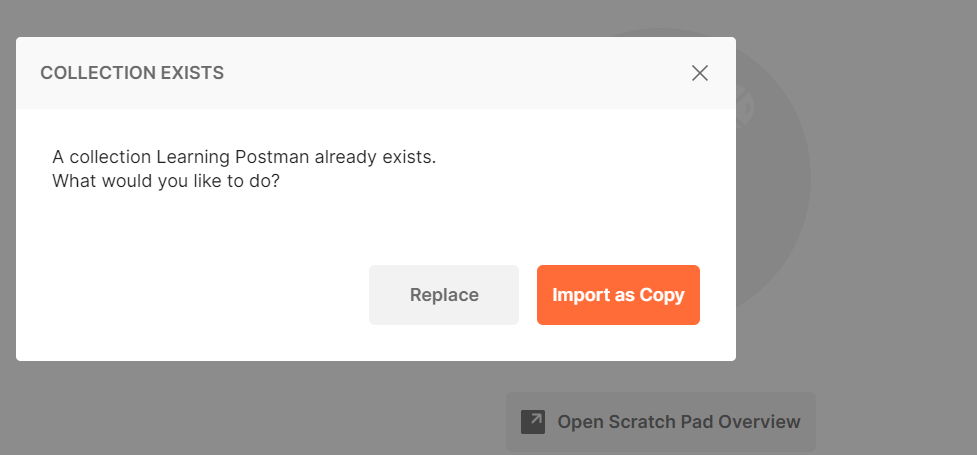
Click on Import, 



Click on “Upload Files” and select the desired file



Click on Import, If you already has a collection with same same name then it asks to import as copy or replace existing collection



If you want to replace choose as “Import as Copy‘