

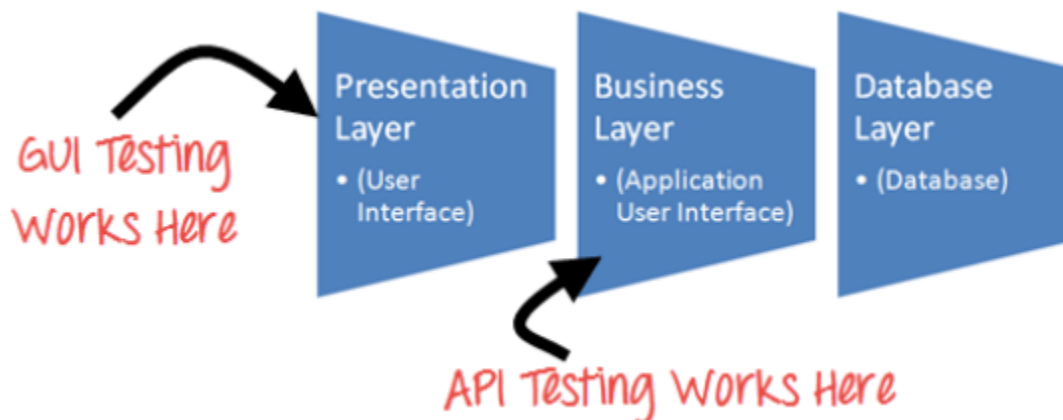


API Testing Using POSTMAN

*By Sabuj Kumar Modak
Jr. QA Engineer at Bdjobs.com Ltd*

API: An API, or Application Programming Interface, is a set of rules and protocols for building and interacting with software applications. It allows different software systems to communicate with each other by defining the methods and data structures they can use to request and exchange information.

API Testing is a software testing type that validates Application Programming Interfaces (APIs). The purpose of API Testing is to check the functionality, reliability, performance, and security of the programming interfaces. In API Testing, instead of using standard user inputs(keyboard) and outputs, you use software to send calls to the API, get output, and note down the system's response. API tests are very different from GUI Tests and won't concentrate on the look and feel of an application. It mainly concentrates on the business logic layer of the software architecture.



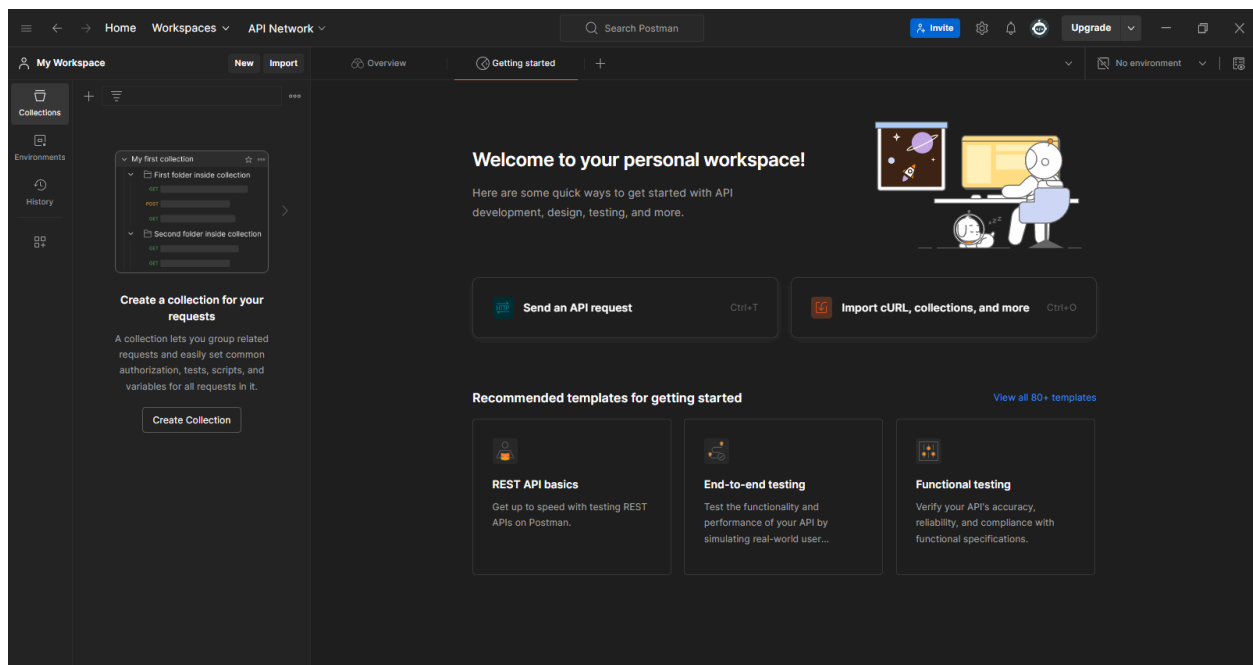
What are the Types of API Protocols and Standards?

Web APIs Web APIs are the most common type, enabling communication between servers and clients over the internet using HTTP/HTTPS protocols.

- **REST (Representational State Transfer):** Uses standard HTTP methods (GET, POST, PUT, DELETE). Stateless communication. Commonly used with JSON, but can use other formats like XML. Designed for scalable and straightforward web services.
- **SOAP (Simple Object Access Protocol):** Uses XML for message format. More rigid and protocol-heavy than REST. Provides built-in error handling and security features.

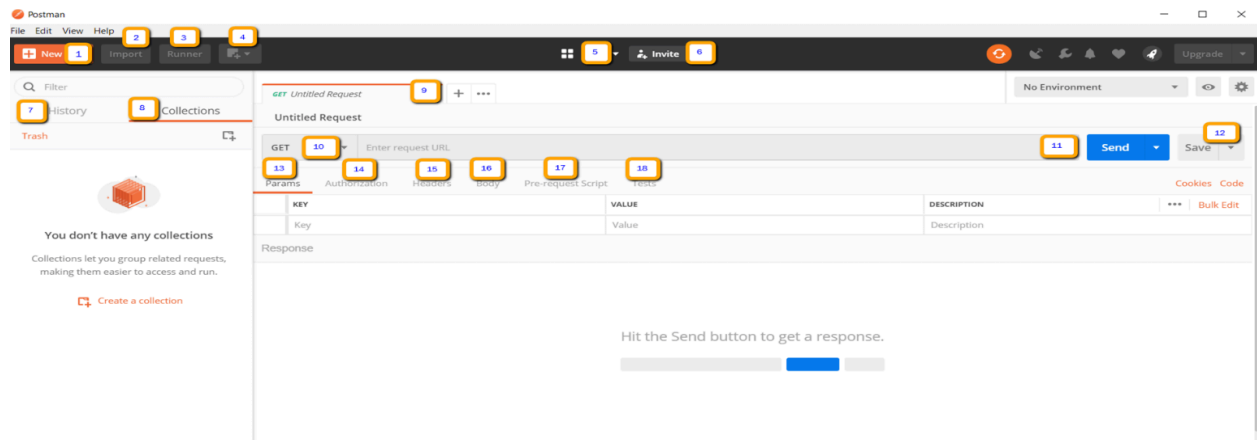
Postman is a scalable API testing tool that quickly integrates into CI/CD pipeline. It started in 2012 as a side project by Abhinav Asthana to simplify API workflow in testing and development. API stands for Application Programming Interface which allows software applications to communicate with each other via API calls.

Install Postman To get started with API testing using Postman, you first need to install it on your local machine. Postman is available for Windows, macOS, and Linux operating systems. You can download Postman from the official website (<https://www.postman.com/>) and follow the installation instructions for your specific operating system.



How to use Postman to execute APIs

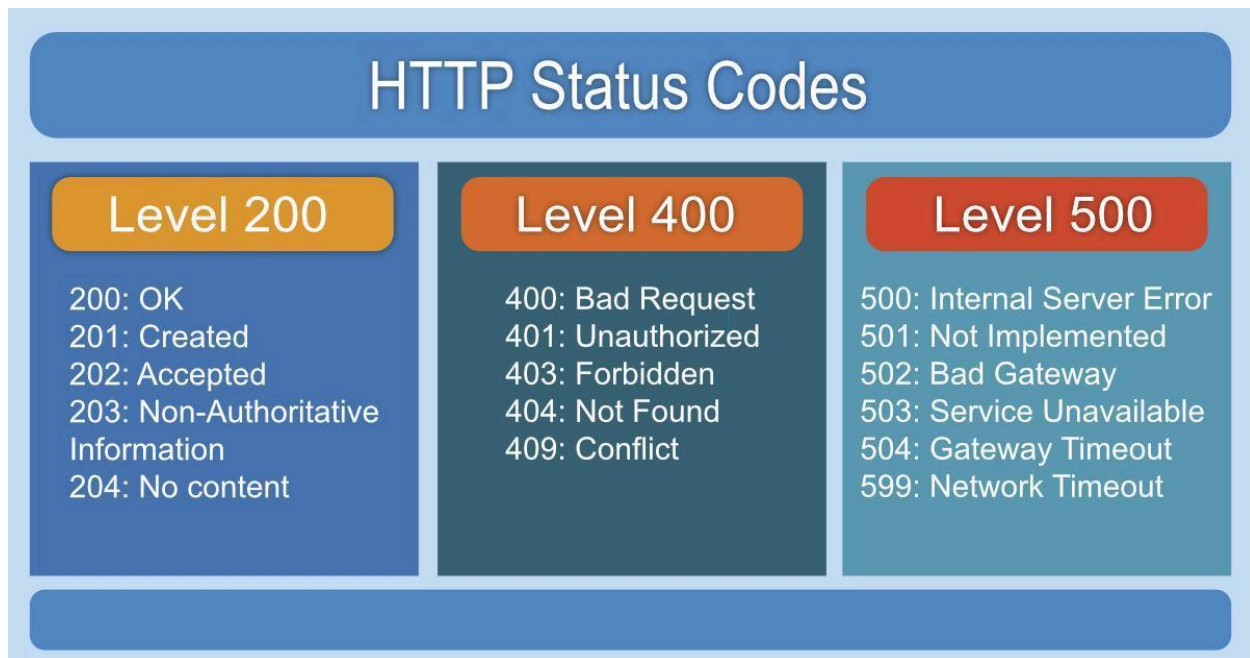
Below is the Postman Workspace. Let's explore the step by step process on **How to use Postman** and different features of the Postman tool!



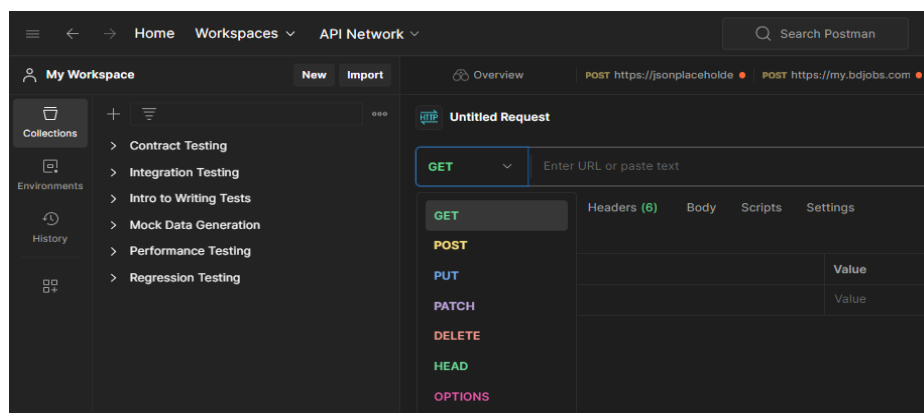
1. **New** – This is where you will create a new request, collection or environment.
2. **Import** – This is used to import a collection or environment. There are options such as import from file, folder, link or paste raw text.
3. **Runner** – Automation tests can be executed through the Collection Runner. This will be discussed further in the next lesson.
4. **Open New** – Open a new tab, Postman Window or Runner Window by clicking this button.
5. **My Workspace** – You can create a new workspace individually or as a team.
6. **Invite** – Collaborate on a workspace by inviting team members.
7. **History** – Past requests that you have sent will be displayed in History. This makes it easy to track actions that you have done.
8. **Collections** – Organize your test suite by creating collections. Each collection may have subfolders and multiple requests. A request or folder can also be duplicated as well.
9. **Request tab** – This displays the title of the request you are working on. By default, "Untitled Request" would be displayed for requests without titles.
10. **HTTP Request** – Clicking this would display a dropdown list of different requests such as GET, POST, COPY, DELETE, etc. In Postman API testing, the most commonly used requests are GET and POST.
11. **Request URL** – Also known as an endpoint, this is where you will identify the link to where the API will communicate with.
12. **Save** – If there are changes to a request, clicking save is a must so that new changes will not be lost or overwritten.
13. **Params** – This is where you will write parameters needed for a request such as key values.
14. **Authorization** – In order to access APIs, proper authorization is needed. It may be in the form of a username and password, bearer token, etc.
15. **Headers** – You can set headers such as content type JSON depending on the needs of the organization.

16. **Body** – This is where one can customize details in a request commonly used in POST requests.
17. **Pre-request Script** – These are scripts that will be executed before the request. Usually, pre-request scripts for the setting environment are used to ensure that tests will be run in the correct environment.
18. **Tests** – These are scripts executed during the request. It is important to have tests as it sets up checkpoints to verify if response status is ok, retrieved data is as expected and other tests.

HTTP status codes:



Types of HTTP:



- **GET-** Purpose: Retrieve information from the server.

Characteristics:

- Requests data from a specified resource.
- Should not change the state of the resource (idempotent).
- Can be cached.
- Can be bookmarked.
- Should not contain a request body.

- **POST-** Purpose: Submit data to be processed to a specified resource.

Characteristics:

- Often used to submit form data or upload a file.
- Can create a new resource or update an existing resource.
- The request body contains the data to be sent to the server.
- Not idempotent (multiple requests may have different effects).
- Cannot be cached.

- **PUT-** Purpose: Update a resource or create a new resource if it does not exist.

Characteristics:

- Replaces the current representation of the resource with the request payload.
- Idempotent (multiple requests with the same data should produce the same effect).
- The request body contains the new version of the resource.

- **DELETE-** Purpose: Remove a specified resource.

Characteristics:

- Deletes the resource identified by the URL.
- Idempotent (multiple requests should have the same effect as a single request).

- **PATCH-** Purpose: Apply partial modifications to a resource.

Characteristics:

- Unlike PUT, which replaces the entire resource, PATCH applies changes to a part of the resource.
- Not necessarily idempotent, depending on how the server handles the partial update.

- **HEAD-** Purpose: Retrieve the headers of a resource without the body.

Characteristics:

- Similar to GET but only transfers the status line and header section.
- Used to check what a GET request will return before making a GET request (e.g., for checking last-modified date or checking if the resource exists).

- **OPTIONS-** Purpose: Describe the communication options for the target resource.

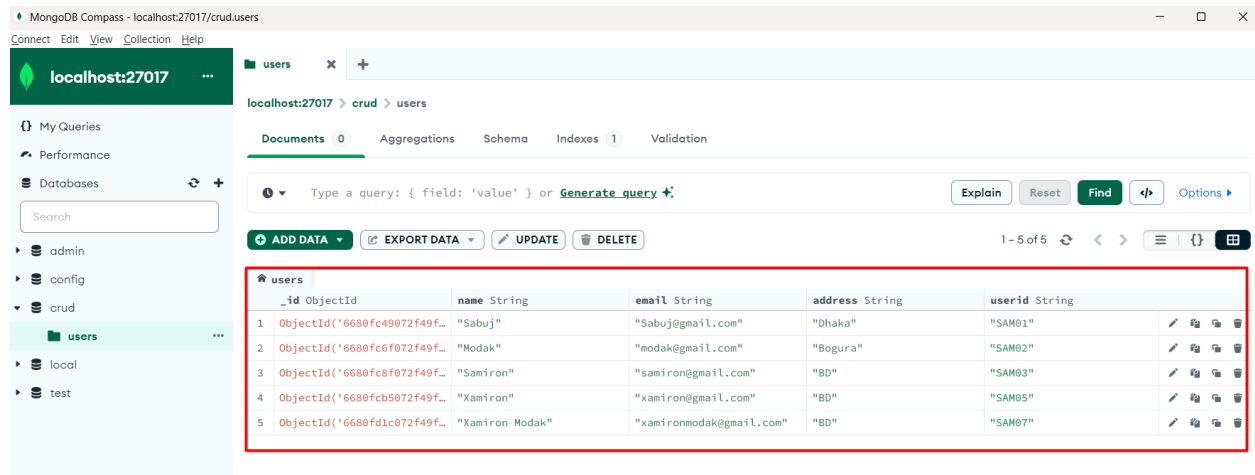
Characteristics:

- Used to determine the capabilities of a server or to test server functionality.
- Can be used to check the allowed HTTP methods for a resource.
- Does not change the resource state (safe and idempotent).

Implemented CRUD operations with Express.js, MongoDB, and Node.js. Now, I'm going to test my own developed API with Postman. Git Repository

Working with GET Requests:

Here are five examples of user data that I have previously created in my Mongo database. The following screenshot:

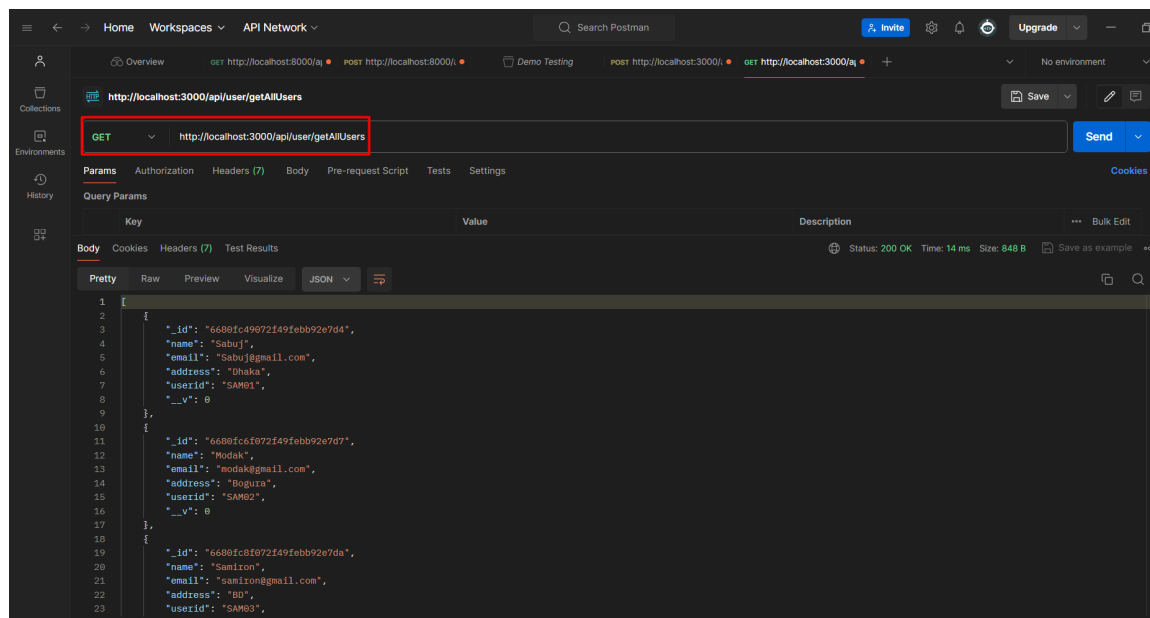


The screenshot shows the MongoDB Compass interface. The 'users' collection is selected, and the 'Documents' tab is active. A table of 5 documents is displayed, each with fields: _id, name, email, address, and userid. The table is highlighted with a red border.

	_id	name	email	address	userid
1	ObjectId('6680fc49072f49f...')	"Sabuj"	"Sabuj@gmail.com"	"Dhaka"	"SAM01"
2	ObjectId('6680fc6f072f49f...')	"Modak"	"modak@gmail.com"	"Bogura"	"SAM02"
3	ObjectId('6680fc8f072f49f...')	"Samiron"	"samiron@gmail.com"	"BD"	"SAM03"
4	ObjectId('6680fcb5072f49f...')	"Xamiron"	"xamiron@gmail.com"	"BD"	"SAM05"
5	ObjectId('6680fd1c072f49f...')	"Xamiron Modak"	"xamironmodak@gmail.com"	"BD"	"SAM07"

GET requests are to retrieve data from the API server.

The below screenshot will depict the process of creating a GET request.



In the workspace

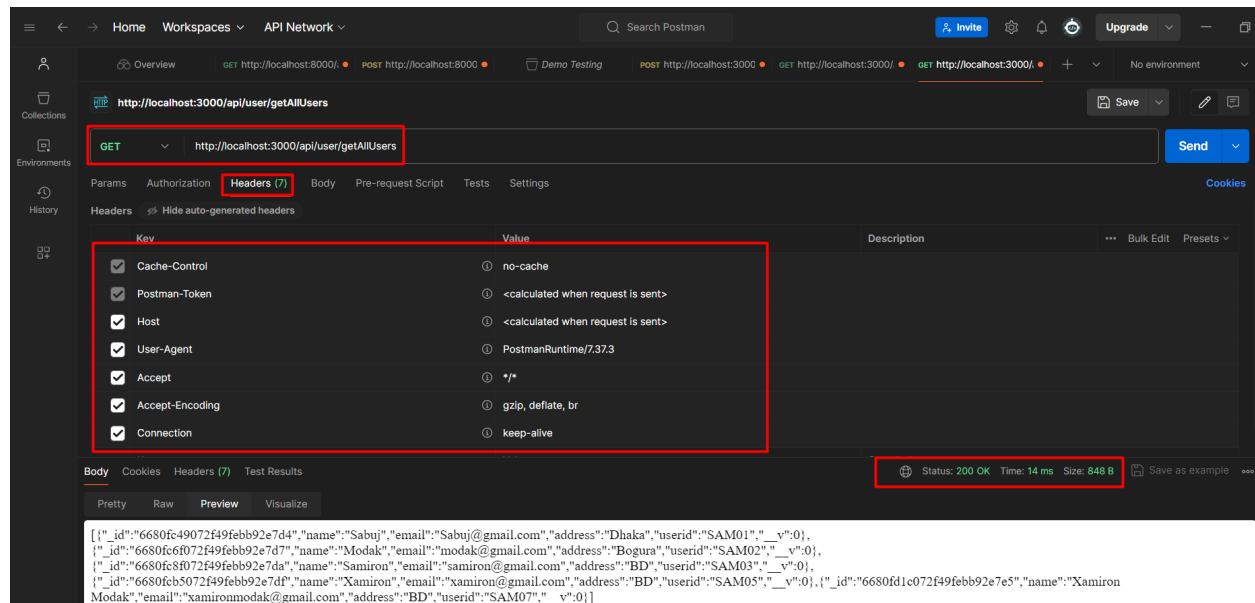
1. Set your HTTP request to GET.
2. In the request URL field, input link
3. Click Send

In this example, we want to retrieve information about 5 user results from my local server.

Request URL: localhost:3000/api/user/getAllUsers

Request Method: GET

Headers (optional, for additional information or authorization):

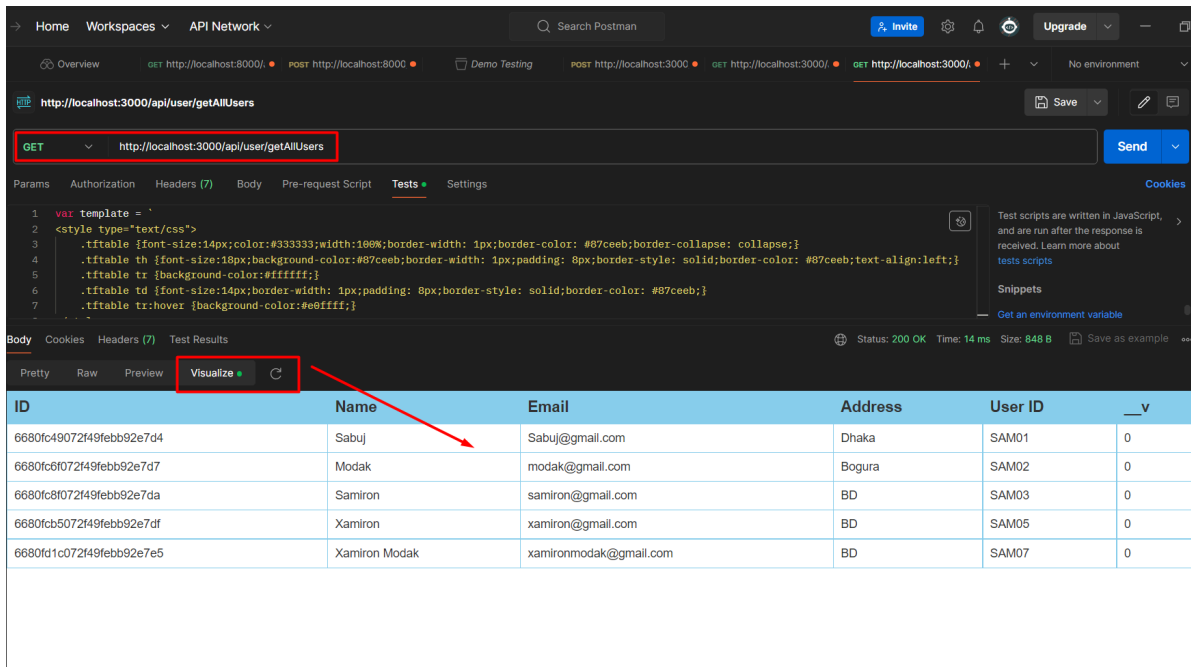


Visualization of GET Request Flow

Client (Browser or Application): Initiates a GET request to the server.

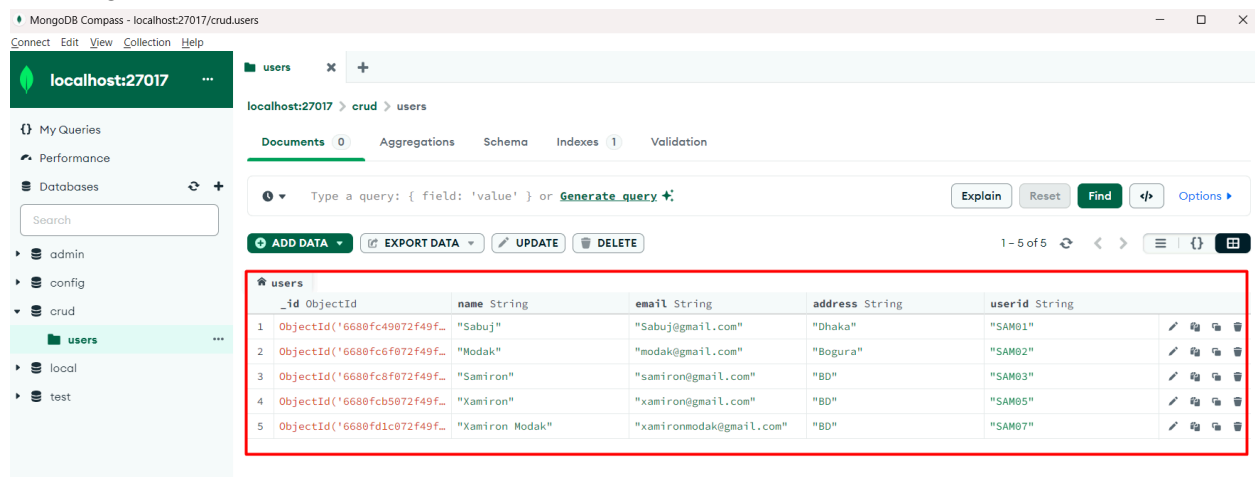
Server: Processes the GET request and retrieves the requested information.

Response: The server sends back the requested data.



Working with POST Requests:

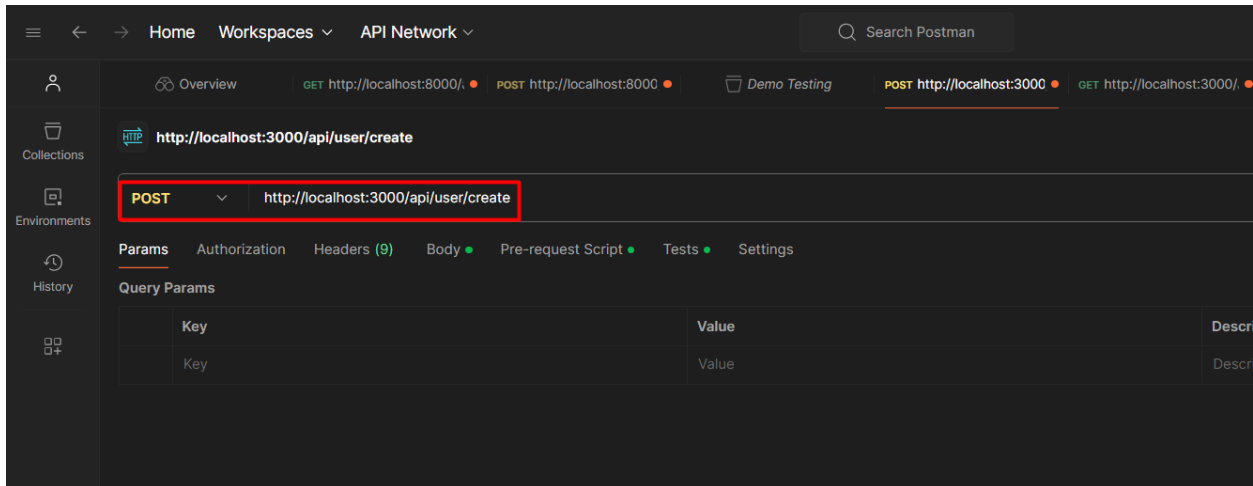
Here are five examples of user data that I have previously created in my MongoDB database. The following screenshot:



I can now add user data to my MongoDB using Postman's POST requests.

Post requests are different from Get requests as there is data manipulation with the user adding data to the endpoint.

The below screenshot will depict the process of creating a **POST** request.

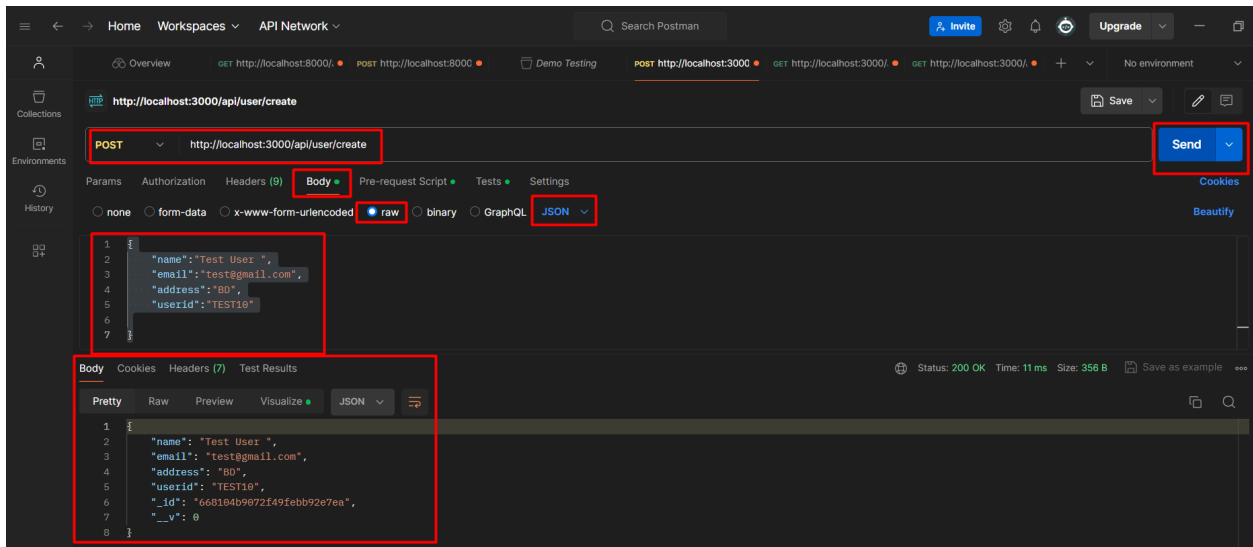


In the new tab

1. Set your HTTP request to POST.
2. Input the same link in request URL:
<http://localhost:3000/api/user/create>
3. switch to the Body tab
4. Click raw
5. Select JSON
6. Copy and paste just one user result from the previous get request, like below. Ensure that the code has been copied correctly with paired curly braces and brackets.

```
{
  "name": "Test User ",
  "email": "test@gmail.com",
  "address": "BD",
  "userid": "TEST10"
}
```

7. Click Send.
8. Status: 201 Created should be displayed
9. The posted data is showing up in the body.



Create a new user on the server. We can see response status code **201** which means created. Before sending a POST request, my Mongo database displays five users. Add a new user to my database after sending a POST request. The following screenshot:

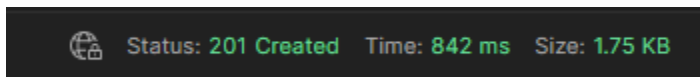
_id	ObjectId	name	String	email	String	address	String	userid	String
1	ObjectId('6680fc49072f49f...	"Sabuj"		"Sabuj@gmail.com"		"Dhaka"		"SAM01"	
2	ObjectId('6680fc6f072f49f...	"Modak"		"modak@gmail.com"		"Bogura"		"SAM02"	
3	ObjectId('6680fc8f072f49f...	"Samiron"		"samiron@gmail.com"		"BD"		"SAM03"	
4	ObjectId('6680fcb5072f49f...	"Xamiron"		"xamiron@gmail.com"		"BD"		"SAM05"	
5	ObjectId('6680fd1c072f49f...	"Xamiron Modak"		"xamironmodak@gmail.com"		"BD"		"SAM07"	
6	ObjectId('668104b9072f49f...	"Test User "		"test@gmail.com"		"BD"		"TEST10"	

Visualization of POST Request Flow

Client (Browser or Application): Initiates a POST request to the server with the new user's data.

ID	Name	Email	Address	User ID	_v
6680fc49072f49f9ebb92e7d4	Sabuj	Sabuj@gmail.com	Dhaka	SAM01	0
6680fc6f072f49f9ebb92e7d7	Modak	modak@gmail.com	Bogura	SAM02	0
6680fc8f072f49f9ebb92e7da	Samiron	samiron@gmail.com	BD	SAM03	0
6680fcb5072f49f9ebb92e7df	Xamiron	xamiron@gmail.com	BD	SAM05	0
6680fd1c072f49f9ebb92e7e5	Xamiron Modak	xamironmodak@gmail.com	BD	SAM07	0
668104b9072f49f9ebb92e7ea	Test User	test@gmail.com	BD	TEST10	0

Server: Processes the POST request and creates a new user.



Response: The server sends back a confirmation with the details of the newly created user.

```
Body  Cookies  Headers (7)  Test Results
Pretty  Raw  Preview  Visualize
{"name":"Test User ", "email":"test@gmail.com", "address":"BD", "userid":"TEST10", "_id":"668104b9072f49febb92e7ea", "__v":0}
```

Request URL: localhost:3000/api/user/create

Request Method: POST

Headers (optional, for additional information or authorization):

http://localhost:3000/api/user/create

POST http://localhost:3000/api/user/create

Params Authorization **Headers (9)** Body Pre-request Script Tests Settings

Hide auto-generated headers

Key	Value
<input checked="" type="checkbox"/> Cache-Control	no-cache
<input checked="" type="checkbox"/> Postman-Token	<calculated when request is sent>
<input checked="" type="checkbox"/> Content-Type	application/json
<input checked="" type="checkbox"/> Content-Length	<calculated when request is sent>
<input checked="" type="checkbox"/> Host	<calculated when request is sent>
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.37.3
<input checked="" type="checkbox"/> Accept	*/*
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br
<input checked="" type="checkbox"/> Connection	keep-alive

Test Result:

Body Cookies Headers (7) **Test Results (4/5)**

All Passed Skipped Failed

- PASS** Response status code is 200
- PASS** Response time is less than 200ms
- PASS** Response has the required fields
- PASS** Name, email, and address are non-empty strings
- FAIL** User ID and version (__v) should be non-negative integers | AssertionError: expected 'TEST10' to be a number

Working with PUT Requests:

Now let's update data from the database using PUT method. Now I update
"id": "668104b9072f49febb92e7ea"

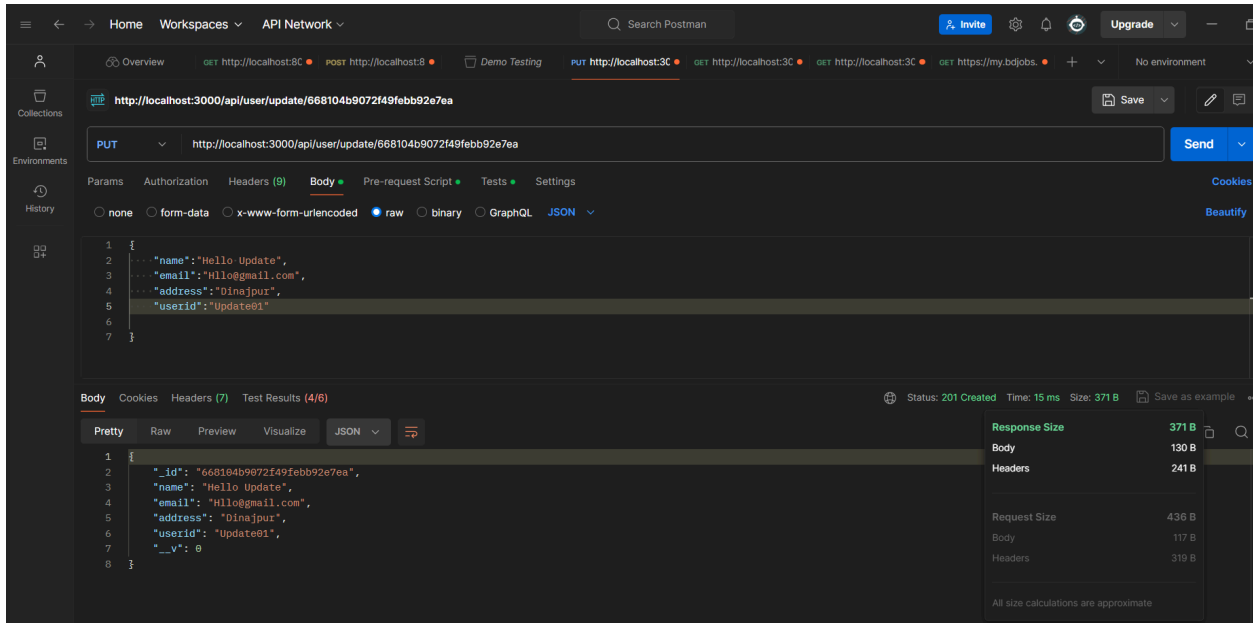
```
{
  "_id": {
    "$oid": "668104b9072f49febb92e7ea"
  },
  "name": "Test User ",
  "email": "test@gmail.com",
  "address": "BD",
  "userid": "TEST10",
  "__v": 0
}
```

First, Create a New PUT Request

- Then put URL : <http://localhost:3000/api/user/update/id>
- Then choose Body -> raw -> json
- And Write the json of name, email ,address and user id.

```
{
  "name": "Hello Update",
  "email": "Hllo@gmail.com",
  "address": "Dinajpur",
  "userid": "Update01"
}
```

- Now just press on Send and see the Body there is same json



Now, I checked whether my MongoDB PUT request worked or not. To check "id": "668104b9072f49febb92e7ea"

I will see that "id": "668104b9072f49febb92e7ea" is updated successfully. The following screenshot:



Working with DELETE Requests:

Now, for deleting data from data, you can just add the new DELETE request and API endpoint : `http://localhost:3000/api/user/delete/id`

Now let's delete data from the database using DELETE method. Now I delete "id": "668104b9072f49febb92e7ea"

```
{
  "_id": {
    "$oid": "668104b9072f49febb92e7ea"
  },
  "name": "Hello Update",
  "email": "Hllo@gmail.com",
  "address": "Dinajpur",
  "userid": "Update01",
  "__v": 0
}
```

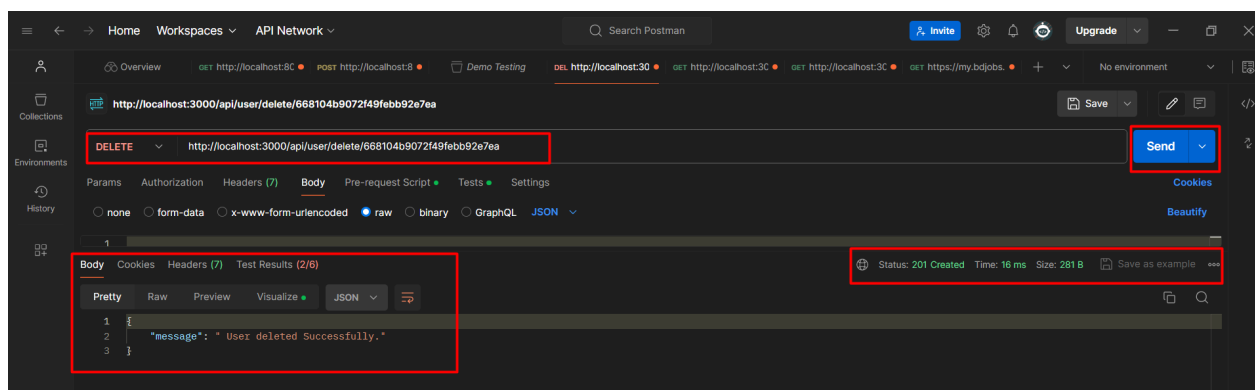
First, Create a New DELETE Request

- Then put URL :

<http://localhost:3000/api/user/delete/668104b9072f49febb92e7ea>

- Now just press on Send and see the response

If data is deleted successfully. It will show the message {"message ": "User deleted successfully"}



If you enjoyed it please do clap & let's collaborate.

Twitter: <https://x.com/xamiron>

Linkedin: <https://bd.linkedin.com/in/sabuj-modak>

Email: xamiron.modak@gmail.com

Thank You