API and Postman

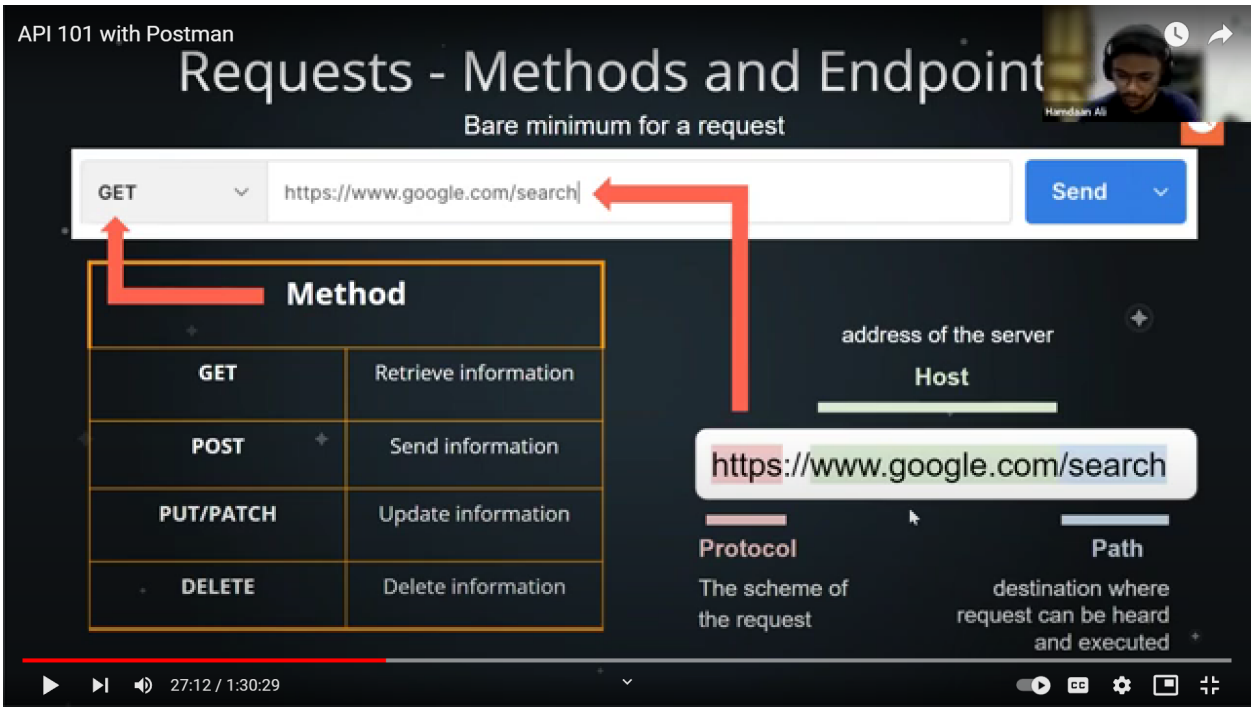
Application Programming Interfaces allow services to communicate with each other.

Developers no longer need to create every service from scratch.

APIs allow developers to access data from a service like google or twitter without any knowledge of how the codebase has been implemented.

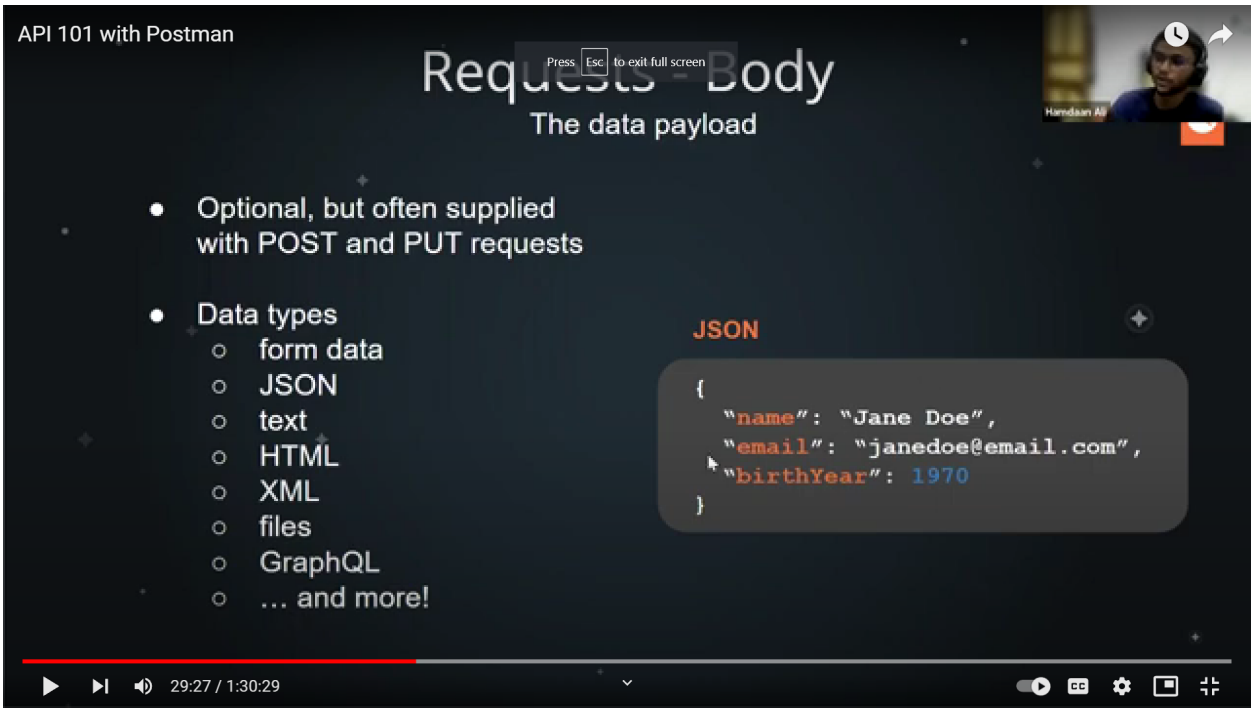
You can breakdown program into different parts and each part will interact with each other through an API:

Making Requests



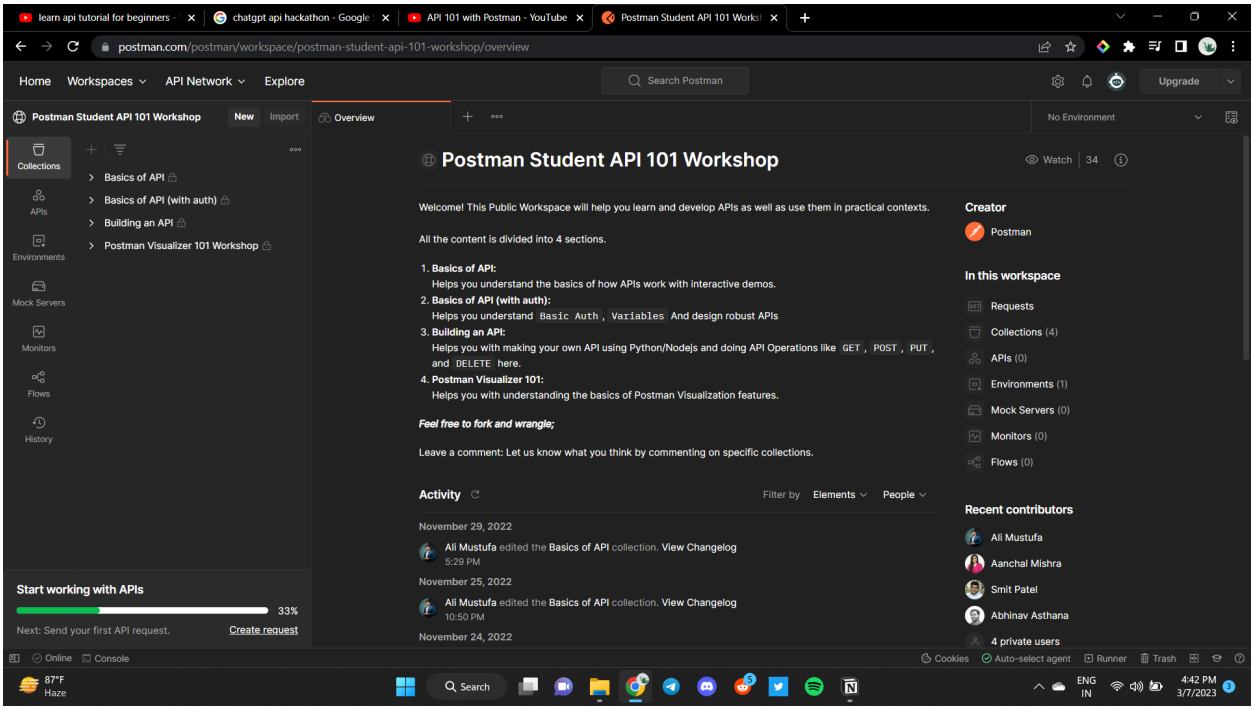
Note: you cannot send a put/post request without body tag but you can send a get request without a body tag.

without body tag your put or post request might not get stored in the server of service provider.

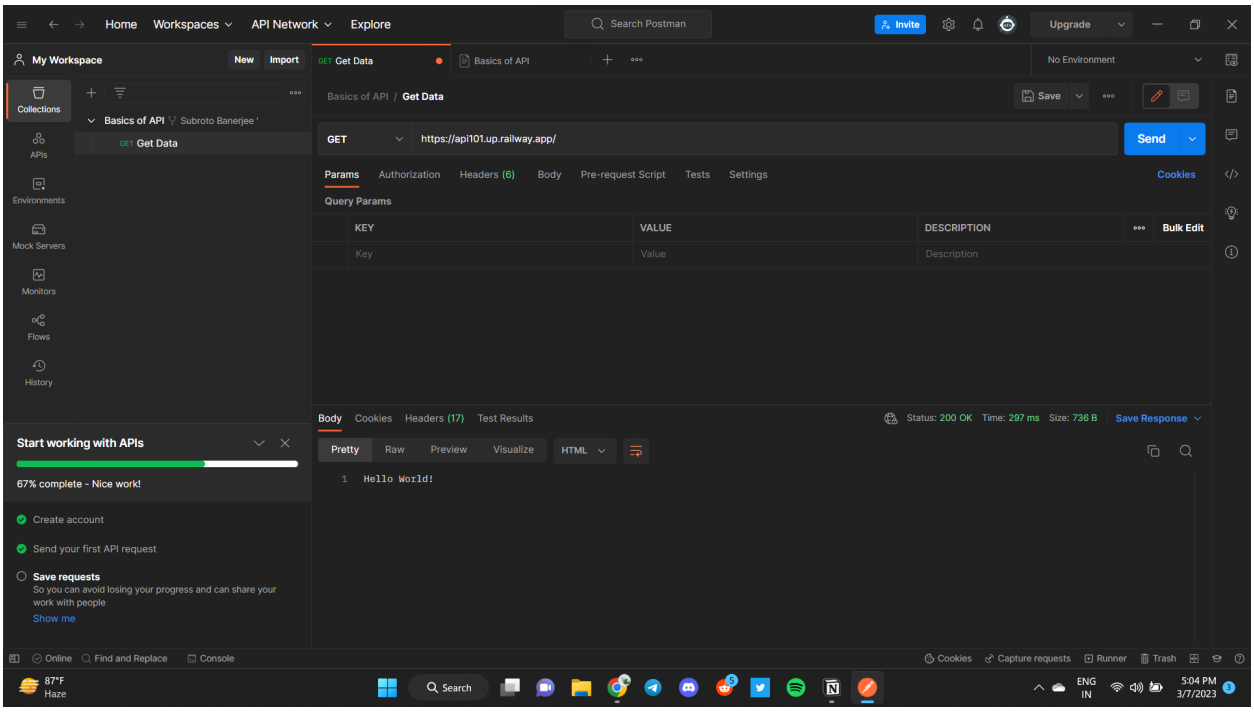


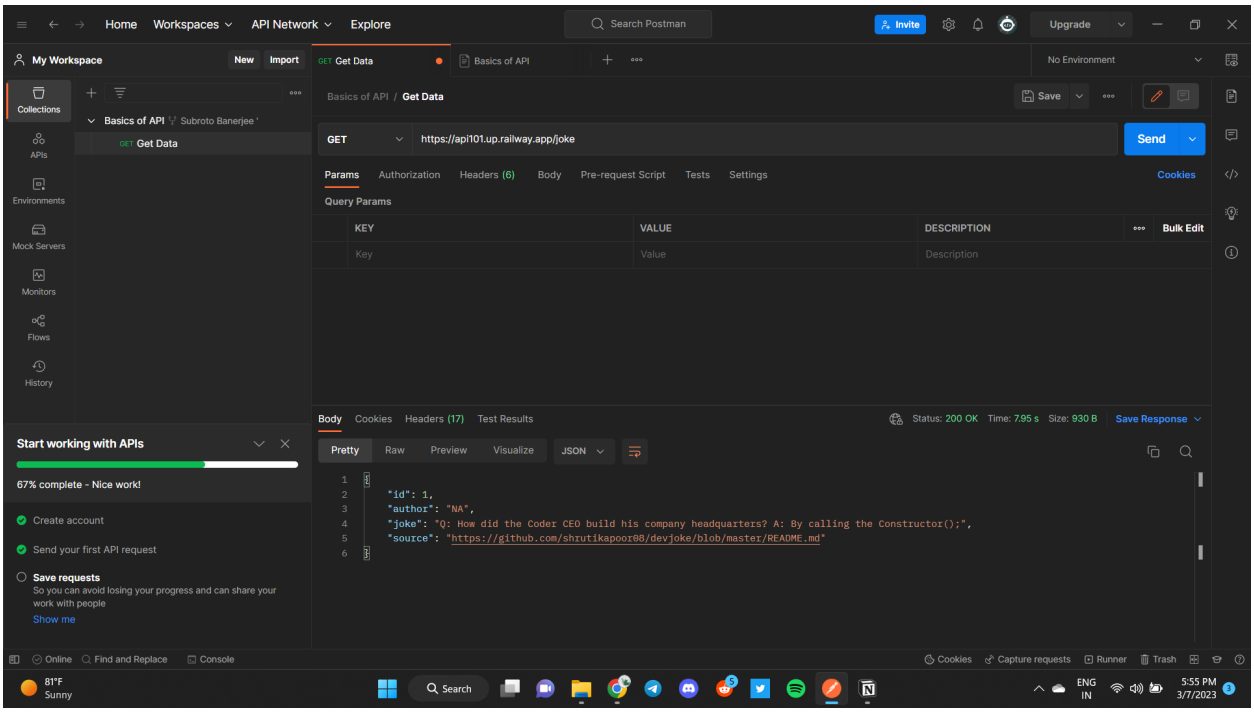
Receiving responses

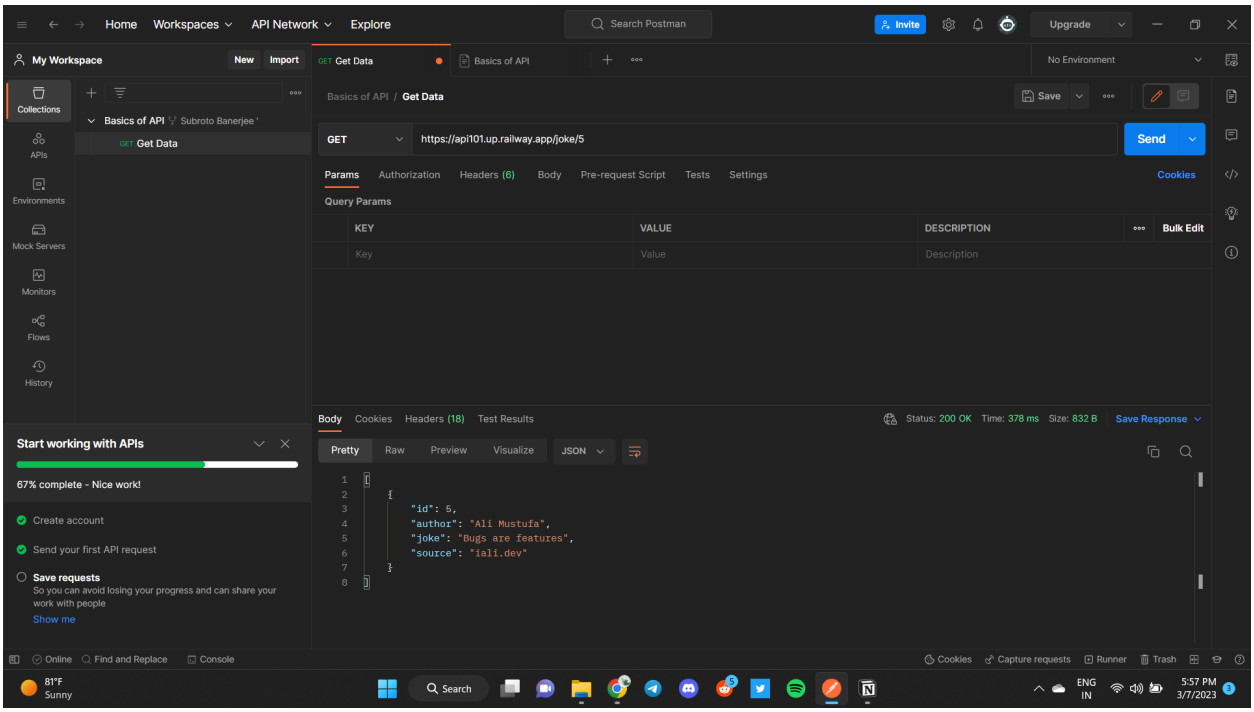
Documentation



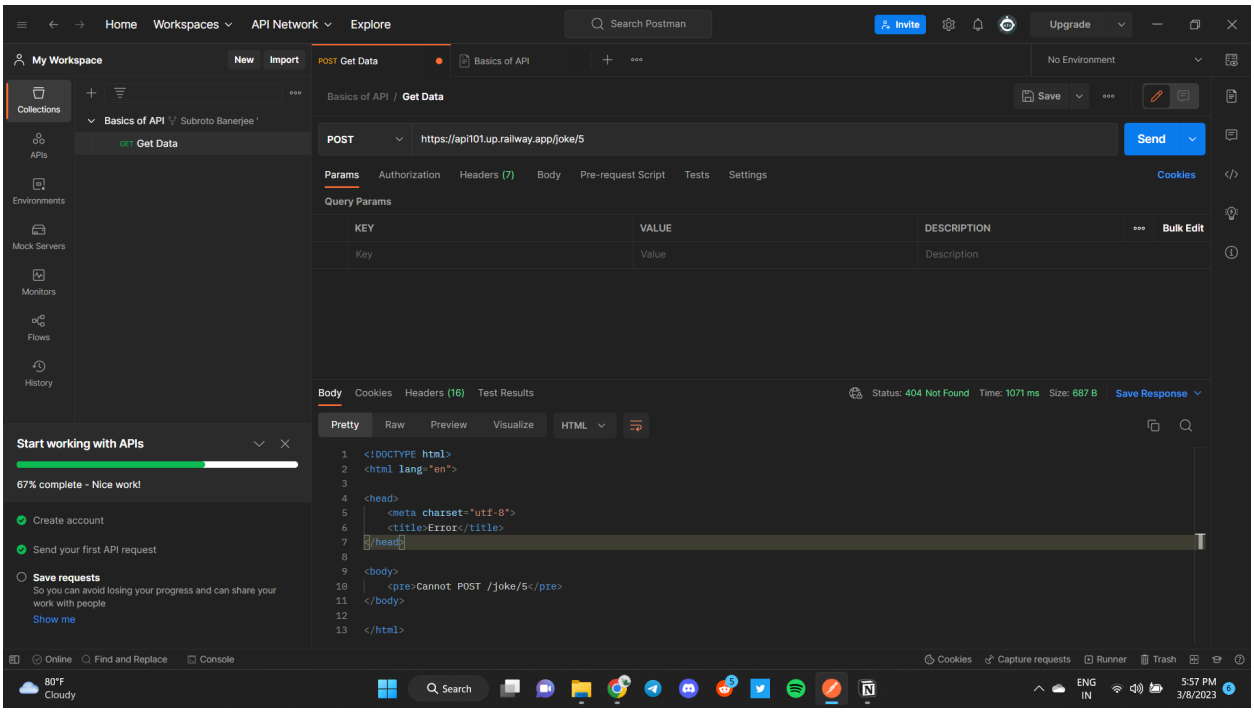
Requesting




Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Get Data
Basics of API
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Basics of API / Get Data
Save
Basics of API Subroto Banerjee
GET Get Data
GET
https://api101.up.railway.app/joke
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (17)
Test Results
Status: 200 OK
Time: 7.95 s
Size: 930 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"author": "NA",
"joke": "Q: How did the Coder CEO build his company headquarters? A: By calling the Constructor();",
"source": "https://github.com/shrutikapoor08/devjoke/blob/master/README.md"
}
Start working with APIs
67% complete - Nice work!
Create account
Send your first API request
Save requests
So you can avoid losing your progress and can share your work with people
Show me
Online
Find and Replace
Console
Cookies
Capture requests
Runner
Trash
81°F
Sunny
Search
ENG
IN
5:55 PM
3/7/2023

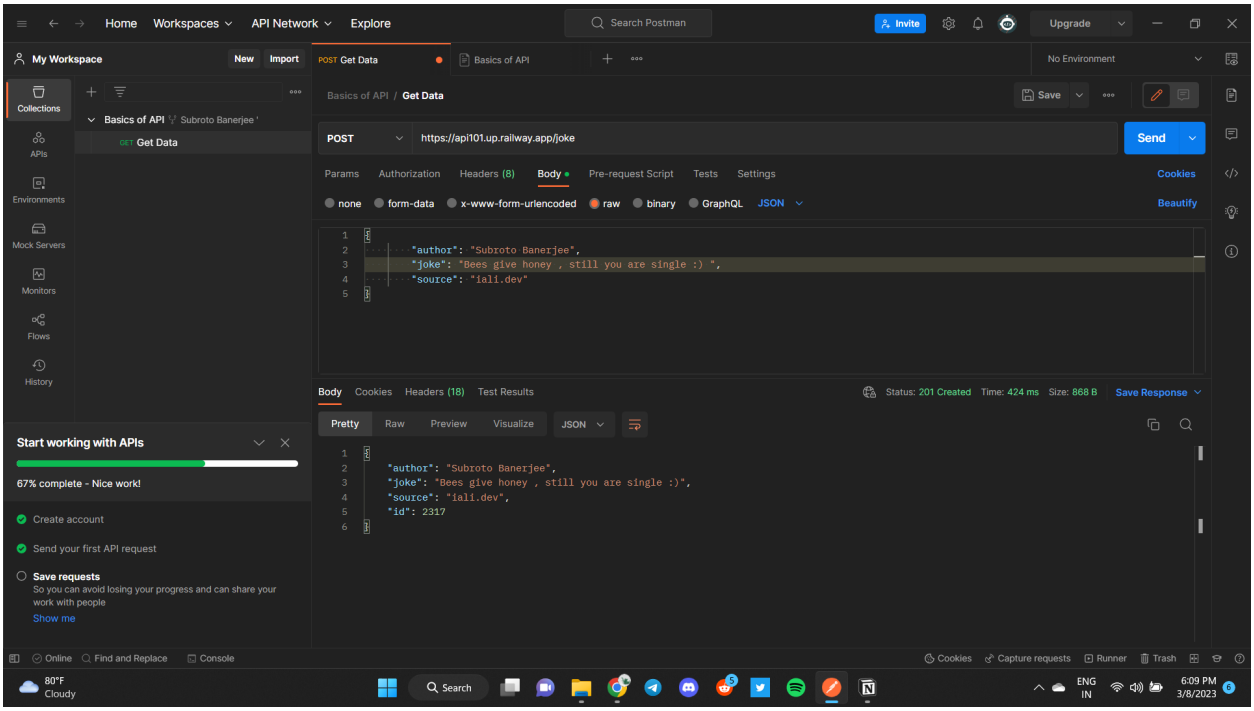

Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Get Data
Basics of API
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Basics of API / Get Data
Save
Basics of API Subroto Banerjee
GET Get Data
GET
https://api101.up.railway.app/joke/5
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (18)
Test Results
Status: 200 OK
Time: 378 ms
Size: 832 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 5,
"author": "Ali Mustufa",
"joke": "Bugs are features",
"source": "iali.dev"
}
]
Start working with APIs
67% complete - Nice work!
Create account
Send your first API request
Save requests
So you can avoid losing your progress and can share your work with people
Show me
Online
Find and Replace
Console
Cookies
Capture requests
Runner
Trash
81°F
Sunny
Search
ENG
IN
5:57 PM
3/7/2023

using POST :



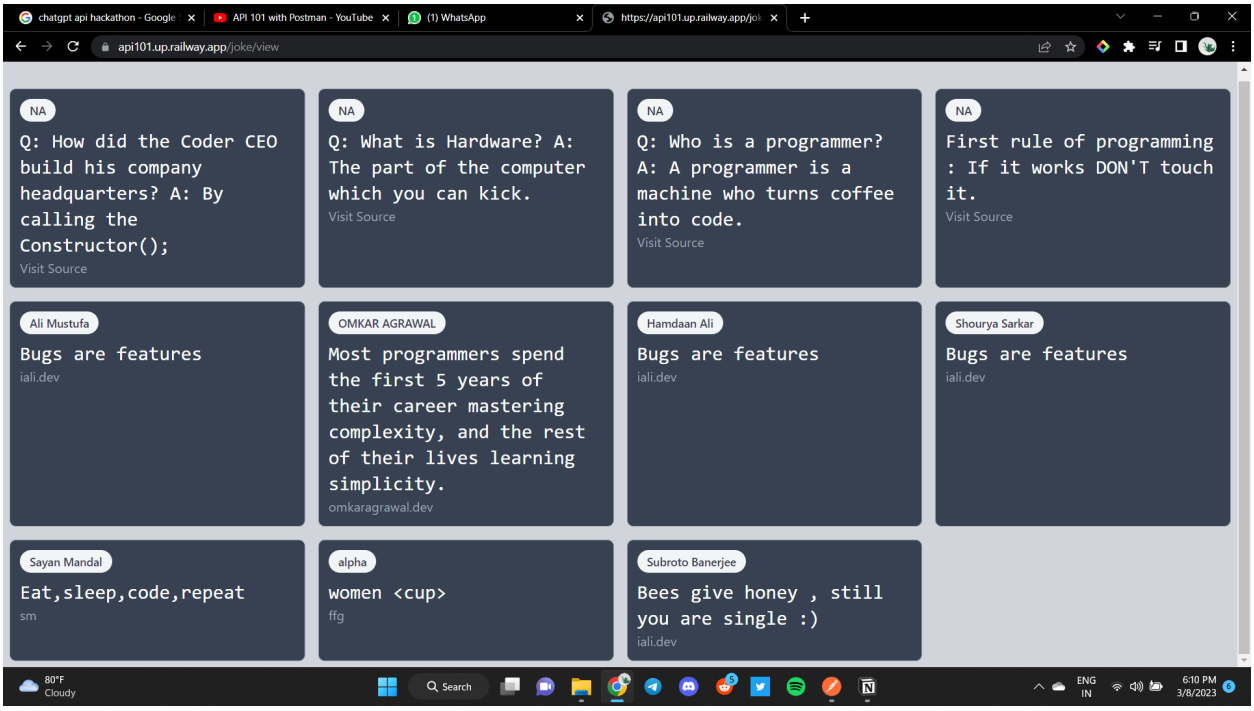
by doing this it is showing status code 404 not found because we did not add a body tag.

hence request is not sent. so we have to add body tag :-

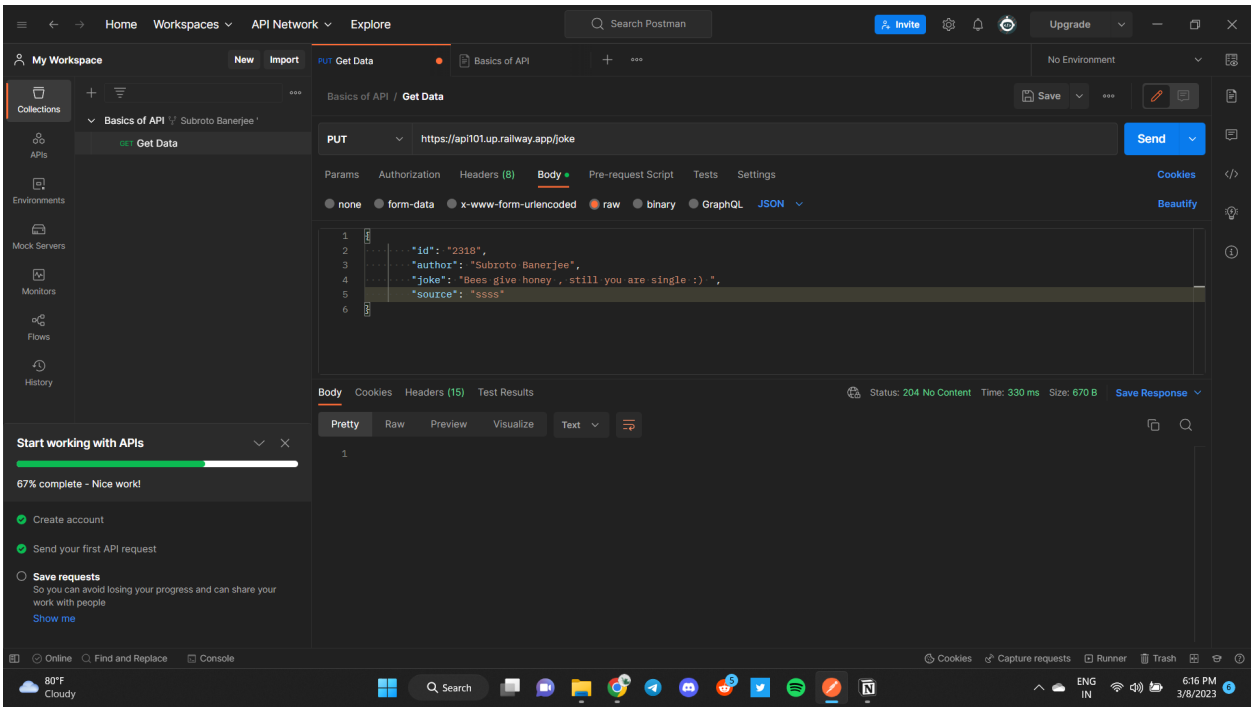


this is how you create a body tag, copy and paste it from the documentation of the API for now only.

Then your sent request will be shown in the original website as follows:-

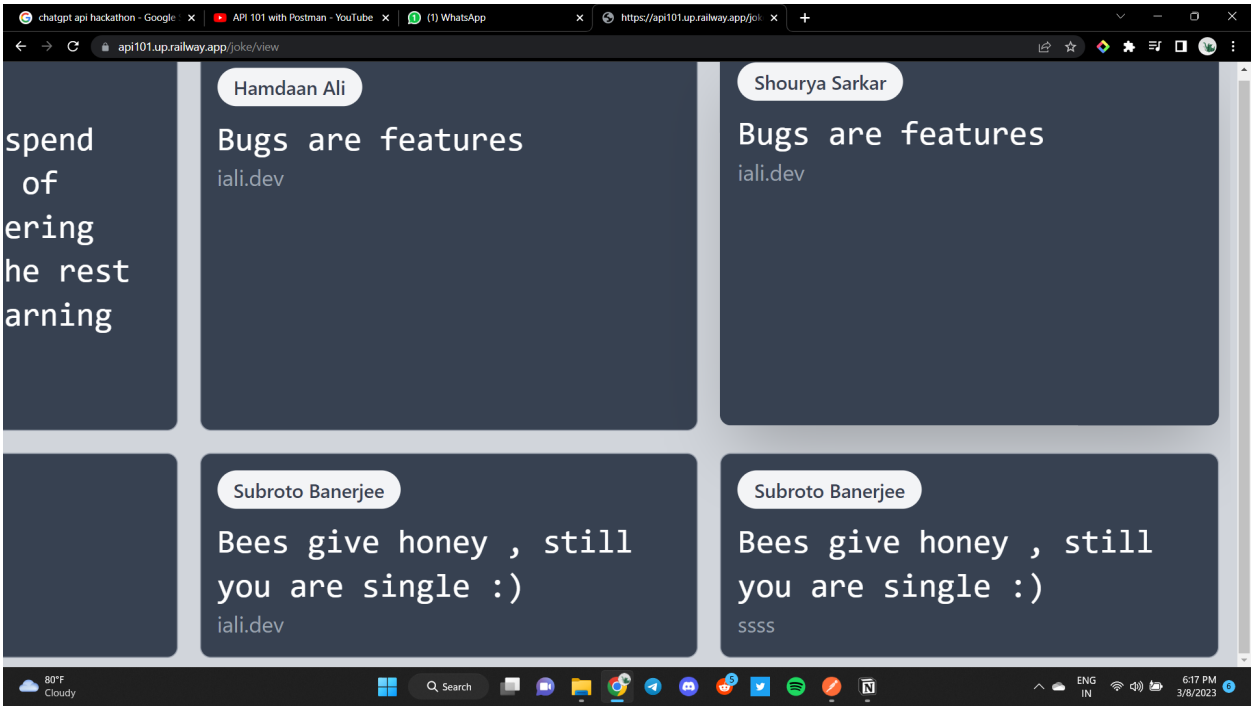


using PUT :



use the id which showed in post method to update info using put method.

The results will show as follows:



look here on left the source is “iali.dev” and on right it is “ssss”.

Note : Any status code starting from 2 means successfully request sent.

using DELETE :

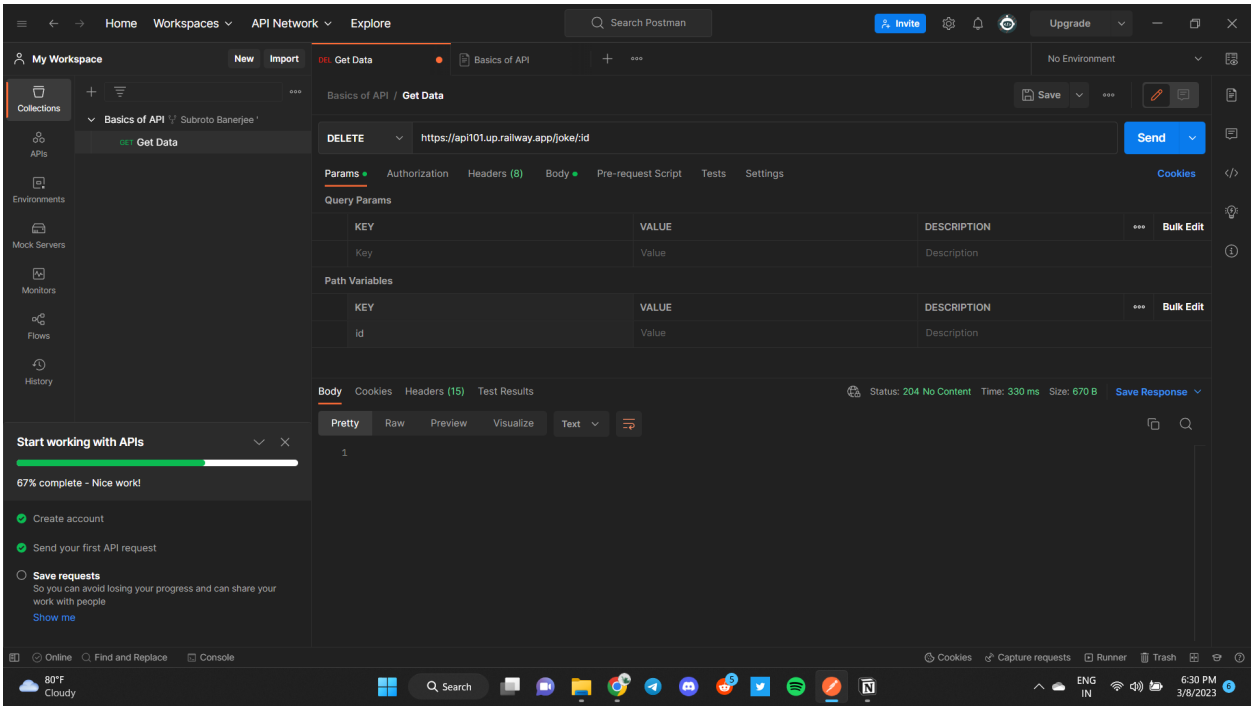
we dont need body tag for this , but have to specify id to delete a specific code from server.

so remembering id is very important.

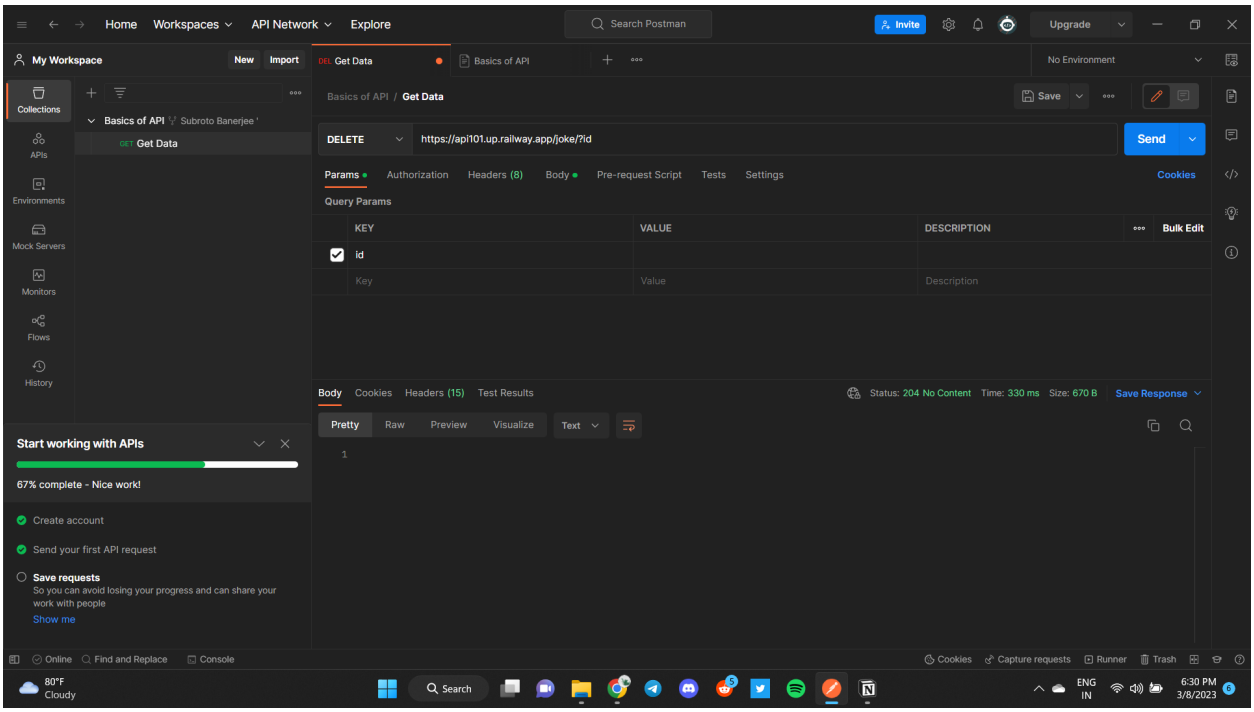
forming a query paramter :-

Note : if it is after “?” it is query parameter , if it after a “:” it is a path variable :

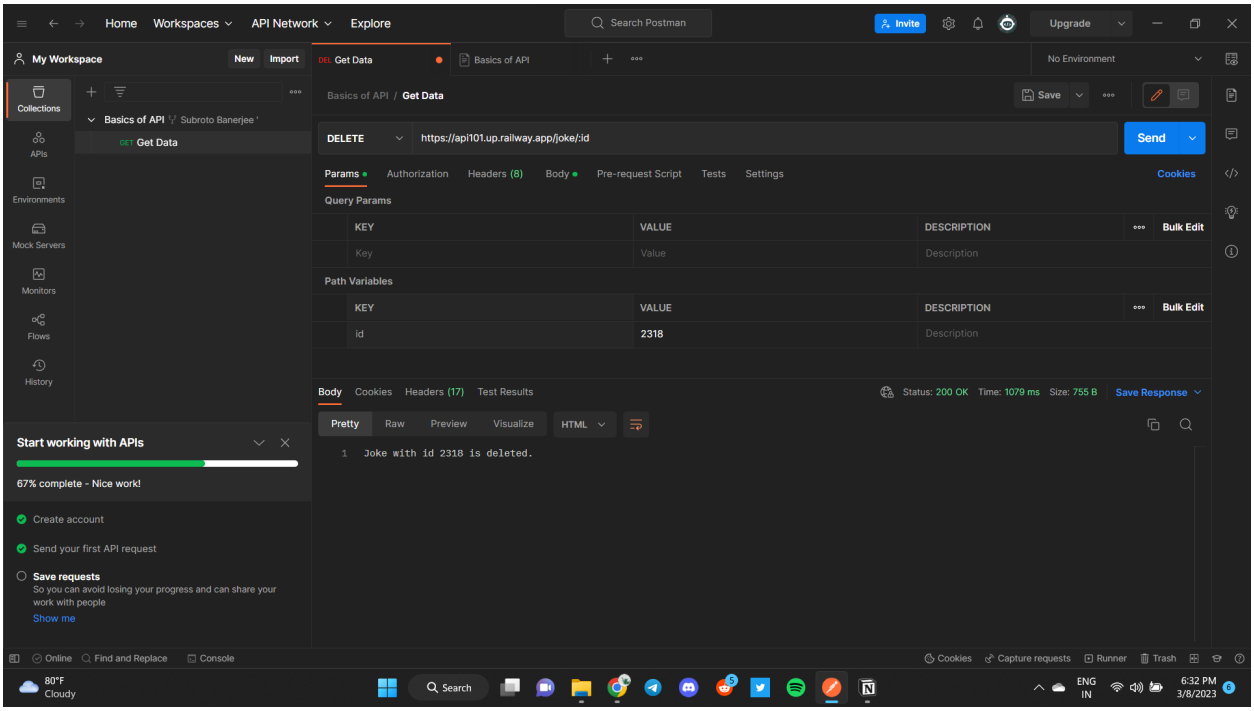
path variable:



query parameter :


Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
Get Data
Basics of API
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Basics of API / Get Data
Save
Basics of API
Subroto Banerjee
GET Get Data
DELETE
https://api101.up.railway.app/joke/?id
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
id
Key
Value
Description
Body
Cookies
Headers (15)
Test Results
Status: 204 No Content
Time: 330 ms
Size: 670 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1
Start working with APIs
67% complete - Nice work!
Create account
Send your first API request
Save requests
So you can avoid losing your progress and can share your work with people
Show me
Online
Find and Replace
Console
Cookies
Capture requests
Runner
Trash
80°F
Cloudy
Search
ENG
IN
6:30 PM
3/8/2023

deletion :


Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
My Workspace
New
Import
Get Data
Basics of API
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Basics of API / Get Data
Save
Basics of API
Subroto Banerjee
GET Get Data
DELETE
https://api101.up.railway.app/joke/:id
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Path Variables
KEY
VALUE
DESCRIPTION
Bulk Edit
id
2318
Description
Body
Cookies
Headers (17)
Test Results
Status: 200 OK
Time: 1079 ms
Size: 755 B
Save Response
Pretty
Raw
Preview
Visualize
HTML
1
Joke with id 2318 is deleted.
Start working with APIs
67% complete - Nice work!
Create account
Send your first API request
Save requests
So you can avoid losing your progress and can share your work with people
Show me
Online
Find and Replace
Console
Cookies
Capture requests
Runner
Trash
80°F
Cloudy
Search
ENG
IN
6:32 PM
3/8/2023

Types of APIs

Medium

While this course will focus on Web APIs, it is important to know that the term "API" can apply to a broad range of interfaces.

- **Hardware APIs**Interface for software to talk to hardware.*Example: How your phone's camera talks to the operating system.*
- **Software Library APIs**Interface for directly consuming code from another code base.*Example: Using methods from a library you import into your application.*
- **Web APIs**Interface for communicating across code bases over a network.*Example: Fetching current stock prices from a finance API over the internet.*

Multiple API types may be used to achieve a task. For example, uploading a photo to Instagram makes use of various APIs:

1. Hardware API for the app to talk to your camera
2. Software library API for the image to be processed with filters
3. Web API for sending your image to Instagram's servers so your friends can like it!

Architectures

There is more than one way to build and consume APIs. Some architecture types you may come across are:

- REST (Representational State Transfer)
- GraphQL
- WebSockets
- webhooks
- SOAP (Simple Object Access Protocol)
- gRPC (Google Remote Procedure Call)
- MQTT (MQ Telemetry Transport)

😱 Don't worry about knowing all of these! **We will focus on REST APIs in this course** since this is the most widely adopted API architecture.

REST APIs Some traits of REST APIs include not storing session state between requests, the ability to cache, and ability to send and receive various data types. Still confused? Don't worry, we will learn hands-on very soon in this course!

Accessibility

APIs also vary in the scope of who can access them.

- **Public APIs (aka Open APIs)**Consumed by anyone who discovers the API
- **Private APIs**Consumed only within an organization and not made public
- **Partner APIs**Consumed between one or more organizations that have an established relationship

The API we will use in this course will be a **Public, REST, Web API** 👀

Method name	Operation
`GET`	Retrieve data (**R**ead)
`POST`	Send data (**C**reate)
`PUT/PATCH`	Update data (**U**pdate)* `PUT` usually replaces an entire resource, whereas `PATCH` usually is for partial updates
`DELETE`	Delete data (**D**elete)

Response status codes

The Postman Library API v2 has sent back a **response status code** of "**200 OK**". Status codes are indicators of whether a request failed or succeeded.Status codes have conventions. For example, any status code starting with a "2xx" (a "200-level response") represents a successful call. Get familiar with other status code categories:

Code range	Meaning	Example
`2xx`		

	Success	`200` - OK `201` - Created `204` - No content (silent OK)
`3xx`	Redirection	`301` - Moved (path changed)
`4xx`	Client error	`400` - Bad request `401` - Unauthorized `403` - Not permitted `404` - Not found
`5xx`	Server error	`500` - Internal server error `502` - Bad gateway `504` - Gateway timeout

That's a lot to remember 😱! Have no fear - in Postman you can hover on any response code to see what it means.

Query parameters

Remember that the minimum ingredients you need to make a request are:

- a request method (`GET` / `POST` / `PUT` / `PATCH` / `DELETE`, etc)
- a request URL

Some APIs allow you to refine your request further with key-value pairs called **query parameters**.

Query parameter syntax

Query parameters are added to the end of the path. They start with a question mark `?`, followed by the key value pairs in the format: `<key>=<value>`. For example, this request might fetch all photos that have landscape orientation:

```
GET https://some-api.com/photos?orientation=landscape
```

If there are multiple query parameters, each is separated by an ampersand `&`. Below, two query parameters to specify the orientation and size of photos to be returned:

```
GET https://some-api.com/photos?orientation=landscape&size=500x400
```

Search Google - with query parameters!

Try pasting this URL into your browser or as a GET request in Postman to make a Google search for "postman". *(If you use Postman, click the "Preview" tab in the response to view the rendered HTML!)*

`https://www.google.com/search` `?q=postman`

This request adds a search term as a query parameter `q=postman` ("q" refers to "query" here) to the `GET /search` path on Google's server.

Because this parameter is in our request, the server returns an HTML document that is a search results page with hits for "postman". The search bar is pre-populated with our query "postman".

You can change your search directly from the URL by changing the value for the query parameter `q=<something else!>`

When to use query parameters?

The answer is always: read the API documentation!

Sometimes query parameters are optional and allow you to add filters or extra data to your responses. Sometimes they are required in order for the server to process your request. APIs are implemented differently to fulfill different needs.

The Postman Library API v2 allows you to add optional query parameters on requests to `GET /books` to filter the books that come back in response.

Path Parameters

Another way of passing request data to an API is via **path parameters**. A path parameter (or "path variable") is a dynamic section of a path, and is often used for IDs and entity names such as usernames.

Path parameter syntax

Path parameters come immediately after a slash in the path. For example, the GitHub API allows you to search for GitHub users by providing a username in the path in place of `{username}` below:

```
GET https://api.github.com/users/{username}
```

Making this API call with a value for `{username}` will fetch data about that user:

```
GET https://api.github.com/users/postmanlabs
```

You can have multiple path parameters in a single request, such as this endpoint for getting a user's GitHub code repository:

`GET https://api.github.com/repos/{owner}/{repoName}`

For example, to get information about the `newman` code repository from `postmanlabs`: `GET https://api.github.com/repos/postmanlabs/newman`

Path vs. query parameters

At first it is easy to confuse these two parameter types. Let's compare them side by side.

Path parameters	**Query parameters**
ex: `/books/abc123`	ex: `/books?search=borges&checkedOut=false`
Located **directly after a slash** in the path. **Can be anywhere in the path**	Located only at the **end of a path**, right after a question mark `?`
Accepts **dynamic values**	Accepts **defined query keys with potentially dynamic values**.
* Often used for IDs or entity names	* Often used for options and filters

- *These are just conventions! Some APIs might ask you to pass an ID or username in a query parameter like this:* `/users?username=getpostman`

When to use path parameters?

Always read the API documentation! If a path parameter is required, the documentation will mention this.

Note that some API documentation uses **colon syntax** to represent a wildcard in the path, like `/users/:username`, while some use curly braces like `/users/{username}`. They both mean the same thing: that part of the path is dynamic!Next we will use path parameters with the Postman Library API v2 to get a specific book.

Debugging requests in the Postman Console

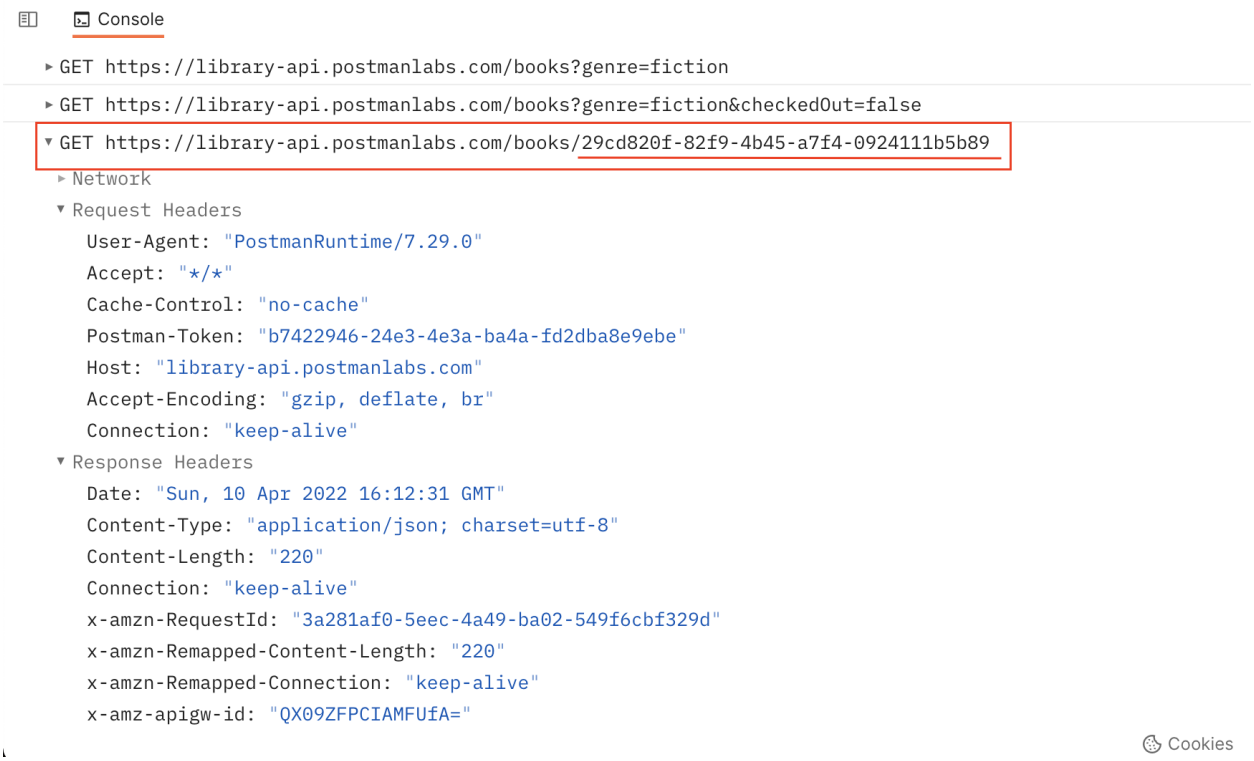
You used Postman's path variable helper in the **Params** tab of the request to add a path variable nicknamed `:id` to the request URL in a human-friendly way. Postman replaces `:id` with the value you specify for `id` in the Path Variables editor.

You can always view the raw request sent to the API by opening the

Postman Console

in the lower left of Postman.

All requests you make and their responses are logged in the Postman Console. Scroll to the bottom to expand the most recent request.



You can see that Postman has inserted the book

```
id
```

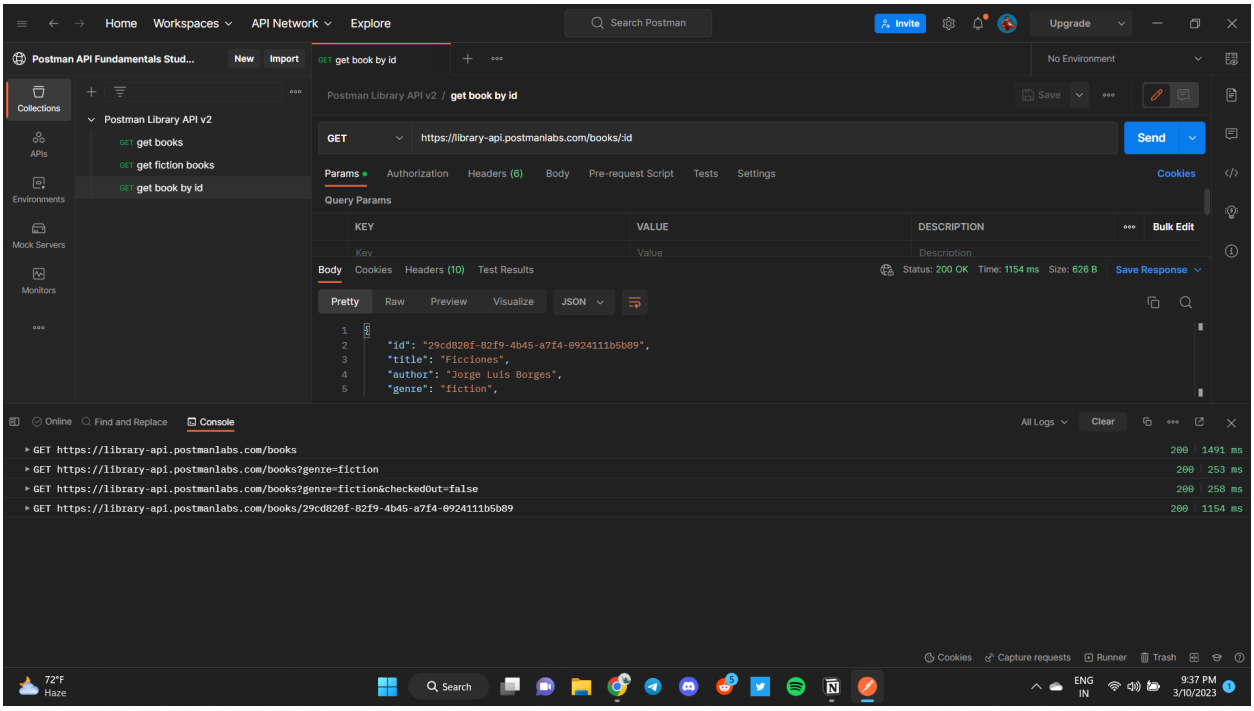
as a path parameter in place of the

```
:id
```

placeholder when making the request. Cool!

If you run into any errors when making API calls, always check the Postman Console and make sure the raw request was sent as you expected.

A common error is adding accidental white space in your query or path parameter values.



Task: Add an authorization header

Some APIs require **Authorization** (aka **Auth**) for certain endpoint in order to permit a request.

Authorization

Think about why you might not want an API to have completely open endpoints that anyone can access publicly. It would allow unauthorized people to access data they shouldn't see, or allow bots to flood an API with thousands of calls per second and shut it down.

There are multiple methods for authorizing a request. Some examples are **Basic Auth** (username and password), **OAuth** (password-less authorization), and **API Keys** (secret strings registered to a developer from an API).

Getting an API Key

APIs that use API Key auth usually allow developers to sign up in a developer portal, where they will receive a random API Key that can be used to authorize their requests to the API. The API Key allows the API to track who is making calls and how often.

The Postman Library API v2 uses very light protection and does not require you to register for an API Key. You simply have to know it:

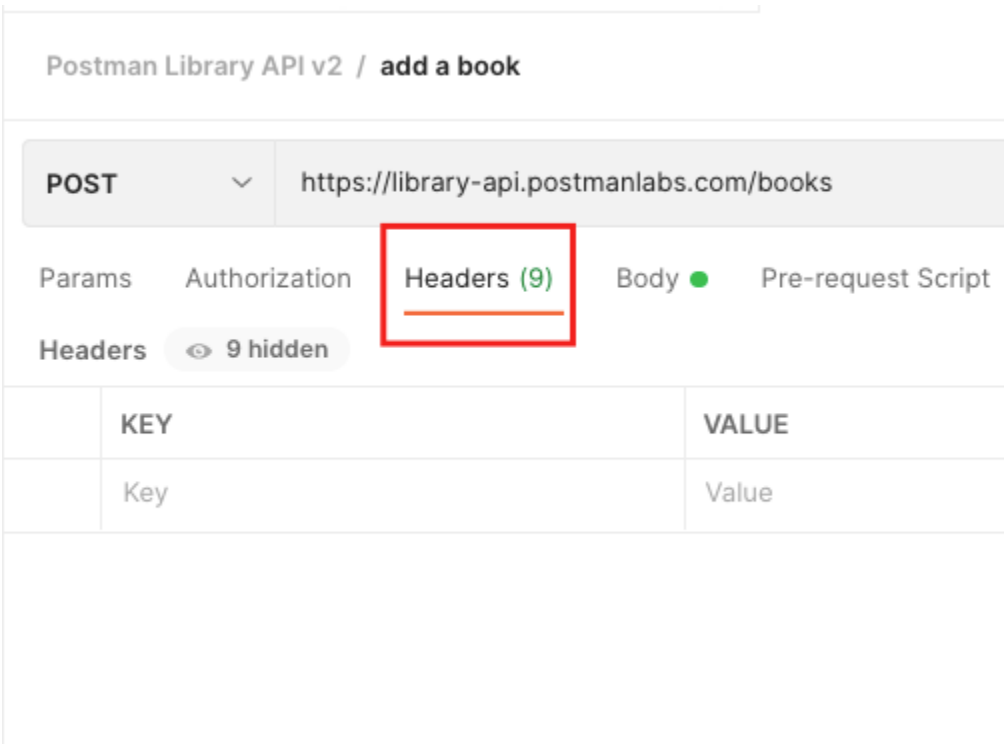
Header name: `api-key` Header value: `postmanrulz` As the documentation shows, the Postman Library API v2 requires adding this **header** to any requests for adding, updating and deleting books, since these operations actually change data in the database as opposed to simply reading them.

Headers

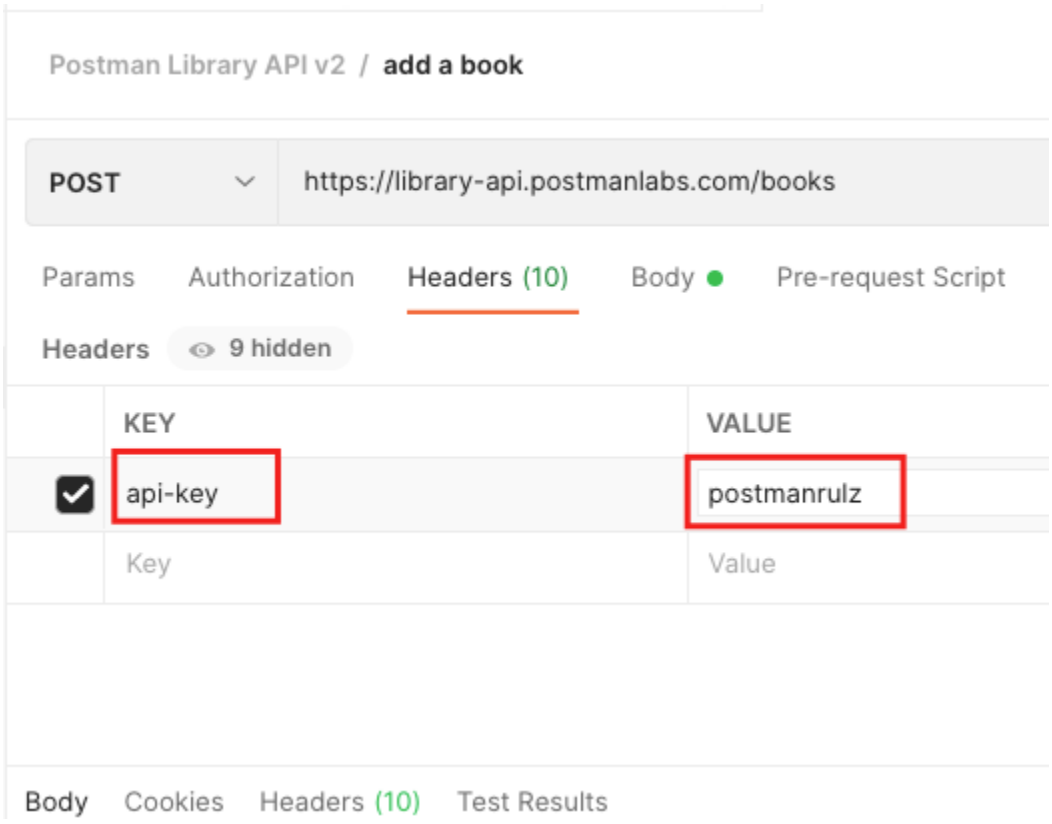
Headers are how we can add **metadata** about our requests, such as authorization information or specify the data type we want to receive in a response. This is different than the actual payload data we send in the body of a request, such as our new book information.You can think of headers like the outside of an envelope when you send a letter. The envelope has information about delivering the letter, like proof that you've paid for postage. The actual data "payload" is the letter inside the envelope.

Add the API Key to the request header

1. On your "**add a book**" request, click the **Headers** tab



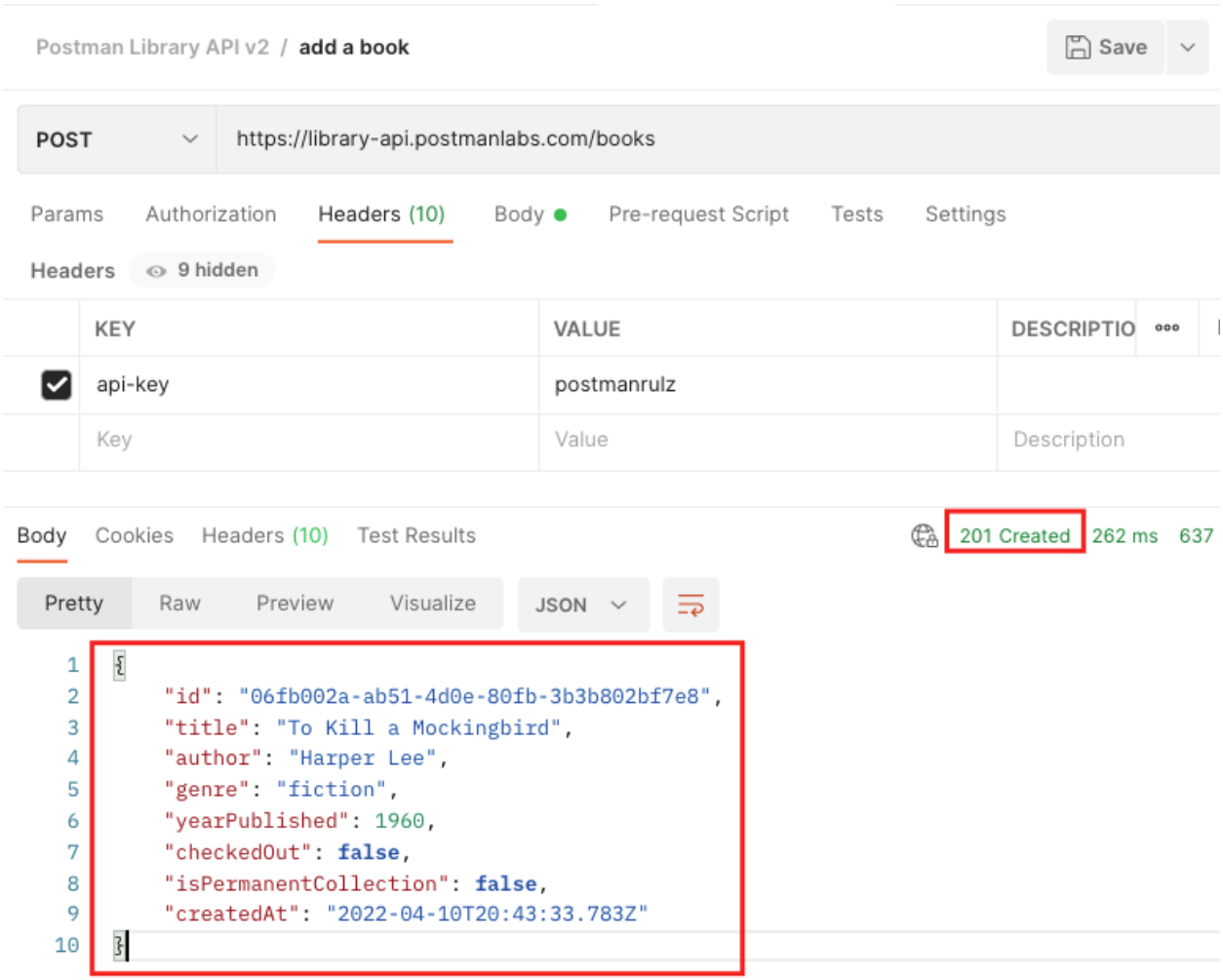
2. In the Headers helper table, add the **key** `api-key` with a **value** of `postmanrulz`



3. **Save** your request
4. **Send** your request

🚀 Success!

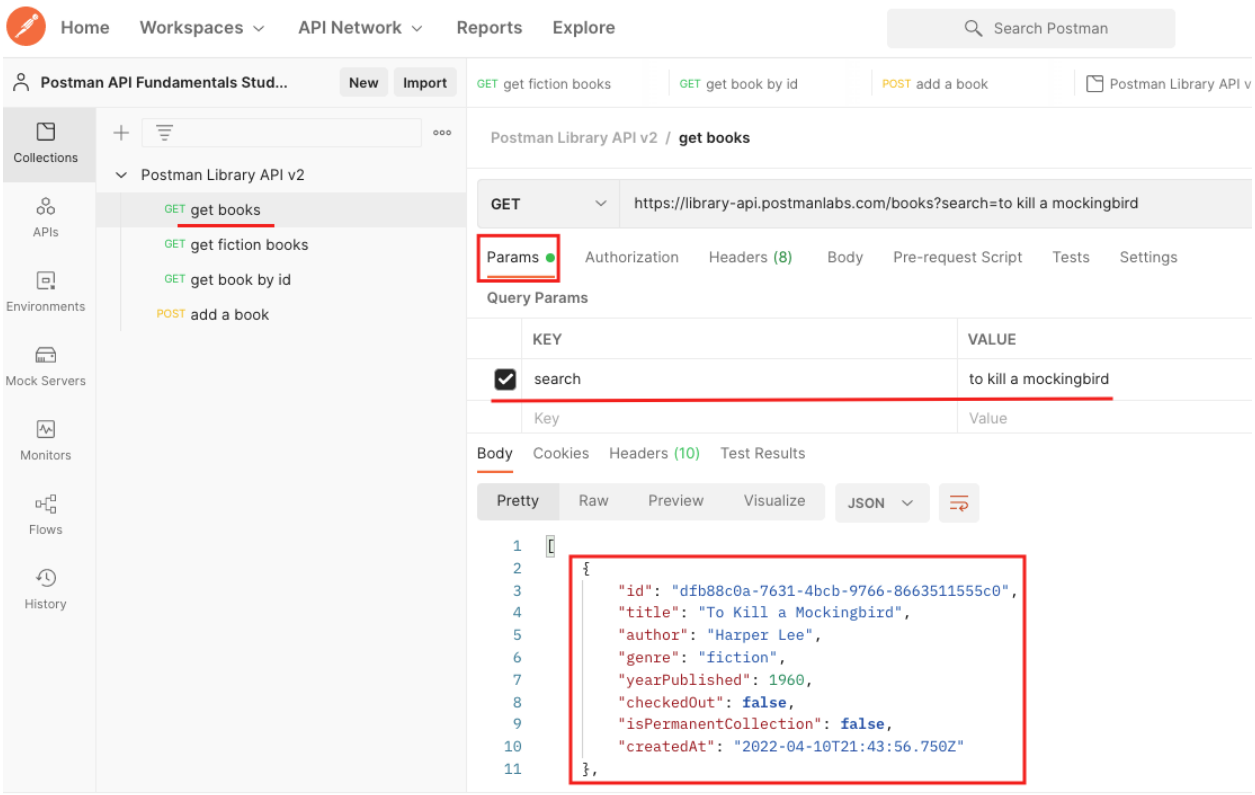
Your book was added! Now that your request is properly authorized in the header, you should get a **201 Created** response with a response body that is an object representing your newly added book!



Your new book has been assigned a random unique `id`, and has extra information now such as it's `checkedOut` status and when it was added to the library (`createdAt`)

View your new book

You can now return to your "get books" request, add the query parameter `search` with a value of the title of the book you added.Anyone can now see your book when they fetch books!



After confirming, you can **uncheck the `search` query parameter** to disable it and **Save** your request.

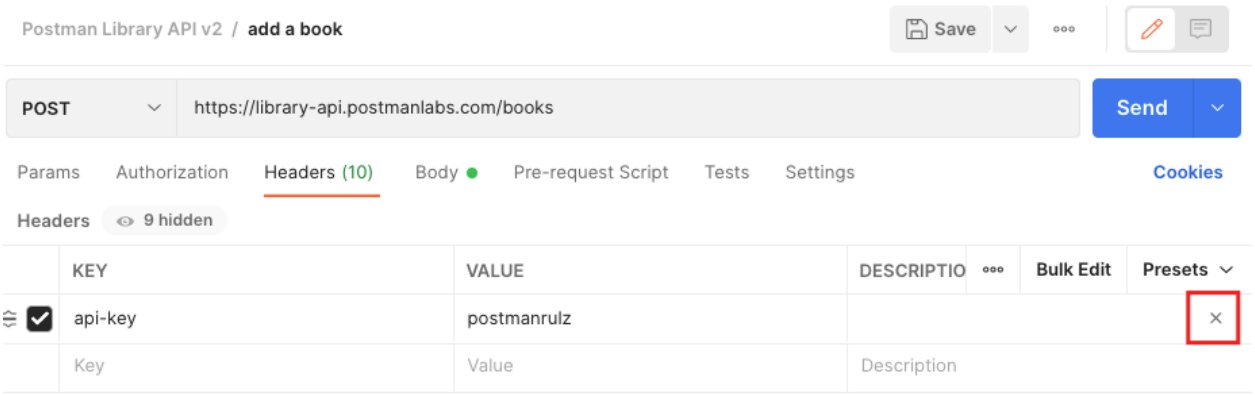
Task: Use Postman Auth instead!

Postman has an Auth helper that makes authorizing requests even easier!

Delete the `api-key` header

Before we use the Postman Auth helper, let's remove the hard-coded header we just added on the "add a book" request.

Hover over the `api-key` header in the **Headers** tab and click the "x" icon at the right to **delete the header**. **Save** your request.

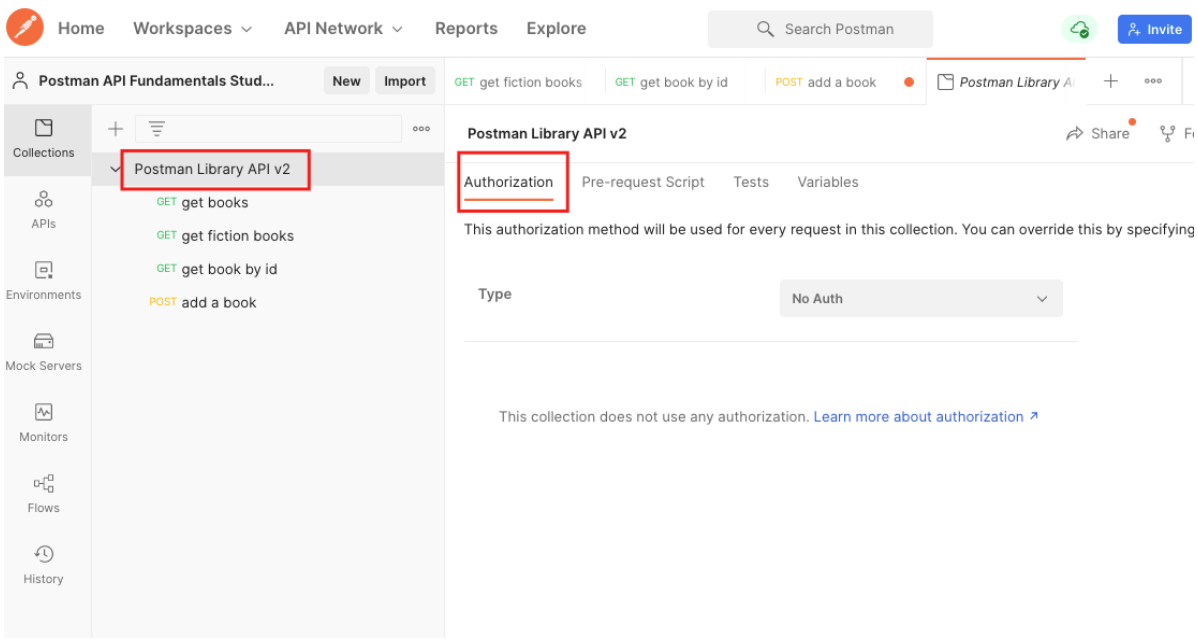


Add Auth to the Collection

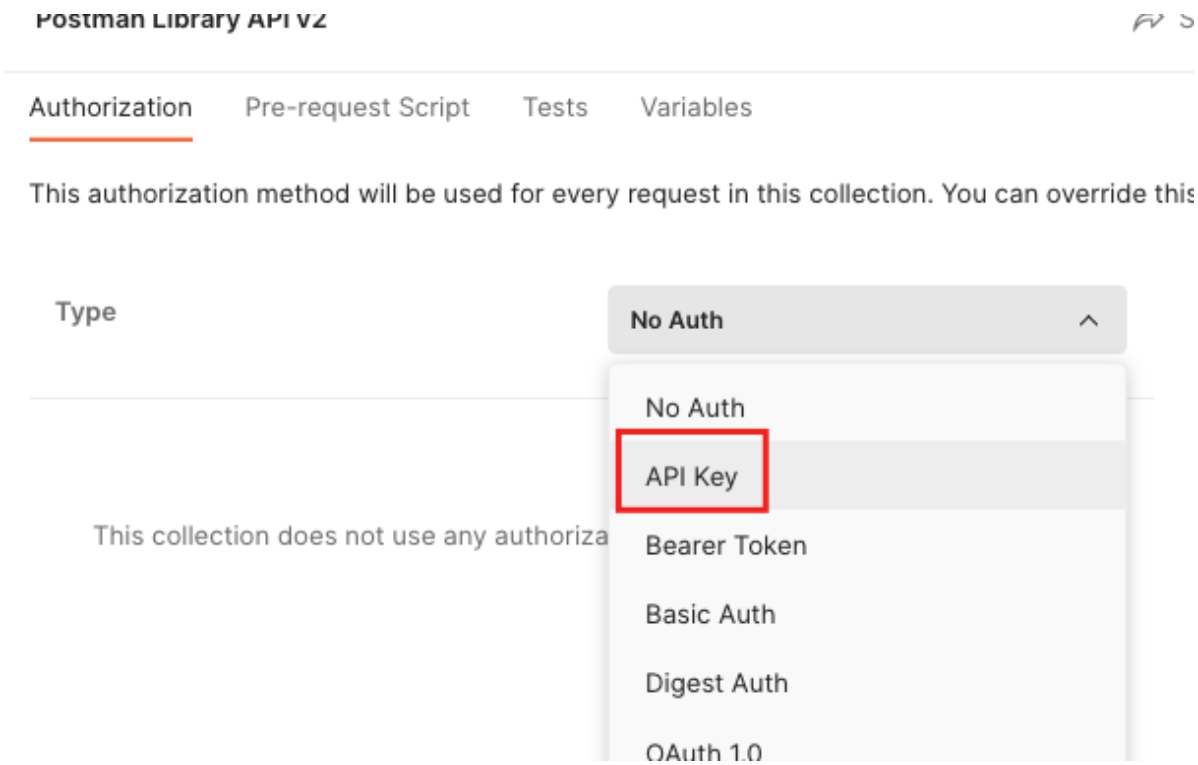
The Postman Auth helper can help you add authorization at the request, folder or collection level.

Let's add the api-key to our entire collection so that all requests will send the key.

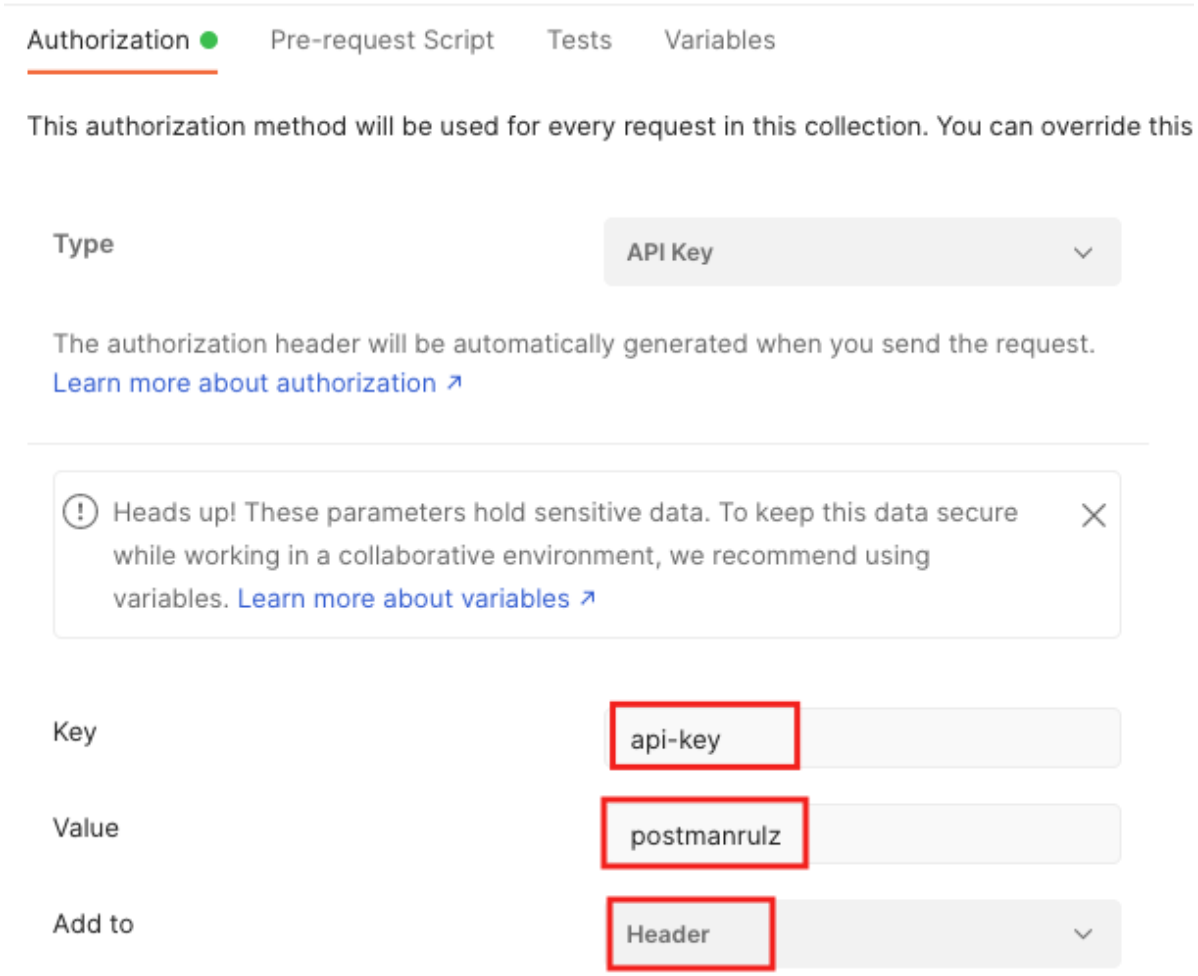
1. Click on your collection "**Postman Library API v2**" and select the **Authorization** (or **Auth**) tab



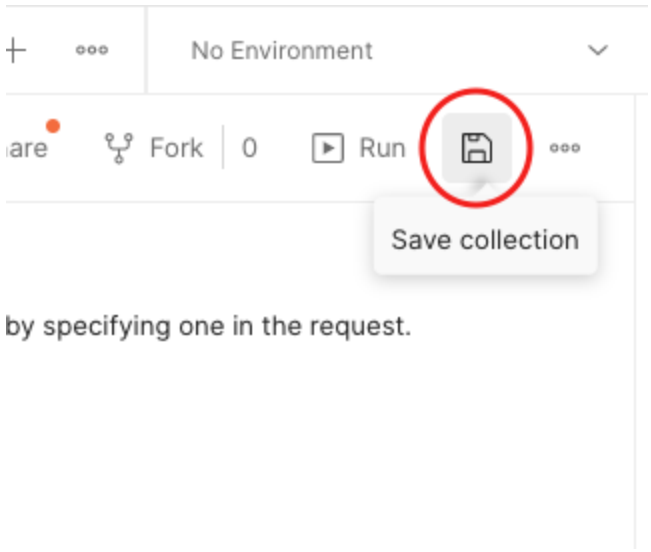
2. Select **API Key** as the auth **Type**



3. Enter the API Key details in the fields below. **Key:** `api-key`, **Value:** `postmanrulz`, **Add to: Header**



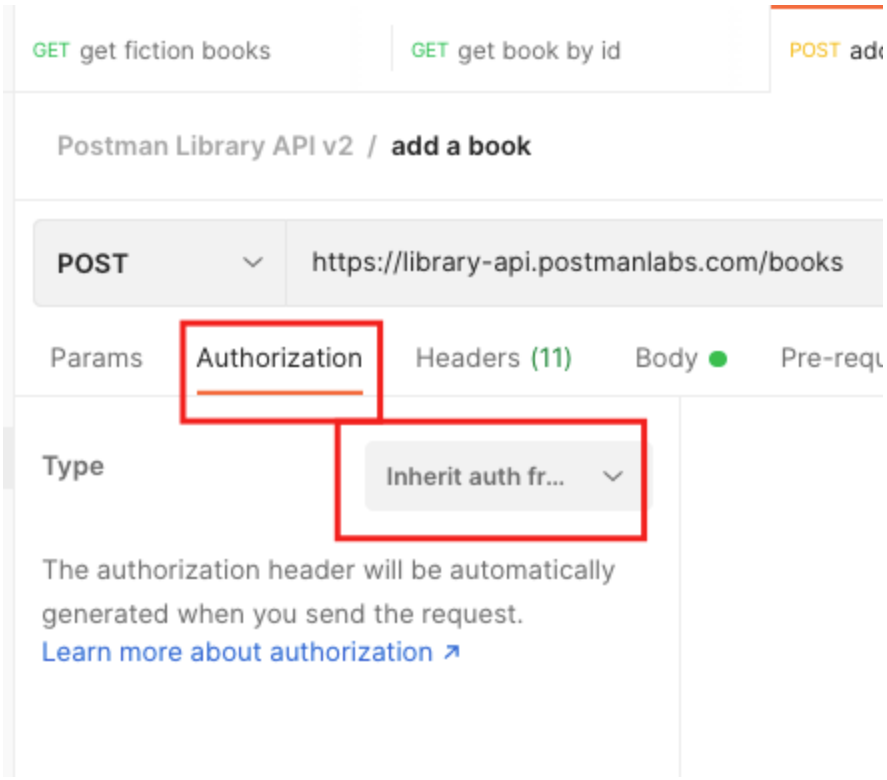
4. **Save** the changes to your collection by clicking the floppy disk icon in the upper right (important!)



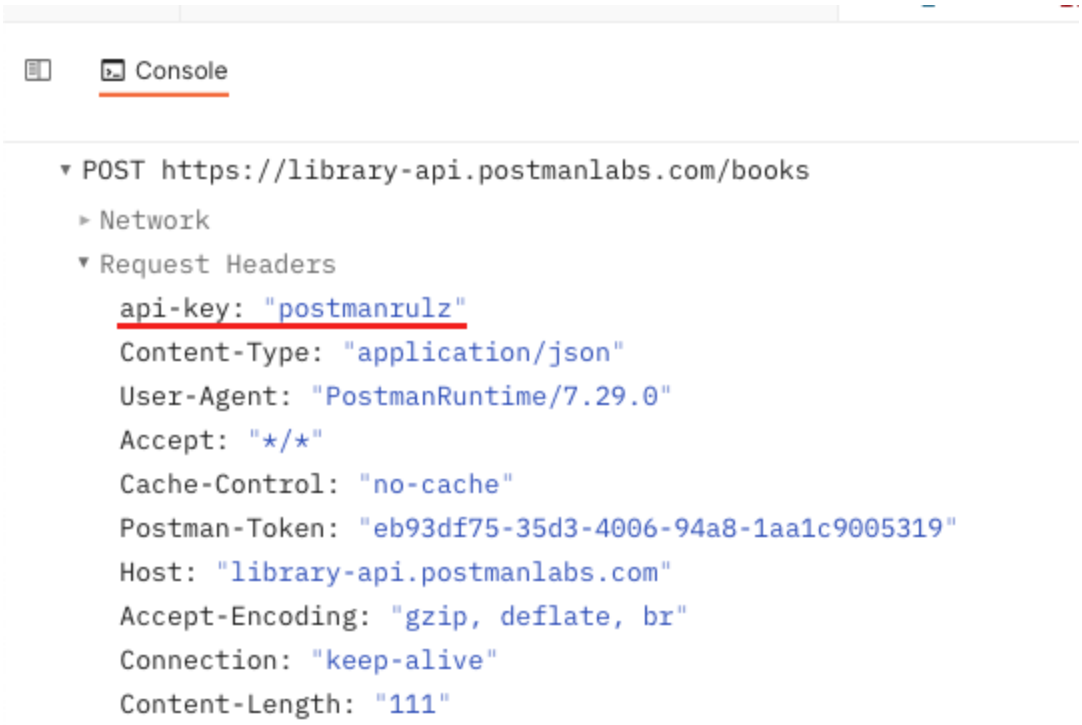
Now all requests inside this collection that use the auth method “Inherit from parent” will have this header attached, and therefore be authorized.

Add a new book

1. Go back to your "add a book" request and **add another book by changing the body in the Body tab**
2. Make sure the Auth method in the **Authorization** tab of your request is set to "Inherit from parent" in order to use the API Key we set at the collection level. This is the default behavior for requests.



3. **Save** your request and hit **Send**!
4. Open up the Postman Console in the lower left and you'll see that the API Key has been added as a header `api-key: postmanrulz`, which is why we were authorized to add a book!



You can use send the "get books" request again to see your new book

Variables in Postman

Postman allows you to save values as variables so that you can:

1. Reuse values to keep your work DRY (Don't Repeat Yourself)Hide sensitive values like API keys from being shared publicly

In this section we will add variables to our collection to introduce better practices and allow us to make dynamic requests.

Variable scopes

You can set variables that live at various scopes. Postman will resolve to the value at the nearest and narrowest scope.In order from broadest to narrowest, these scopes are: **global**, **collection**, **environment**, **data**, and **local**.

We will work with **collection variables** today, which are variables that live at the collection level and be can accessed anywhere inside the collection.

Variable syntax

Once a variable is defined, you can access its value using double curly brace syntax like this: `{{variableName}}`

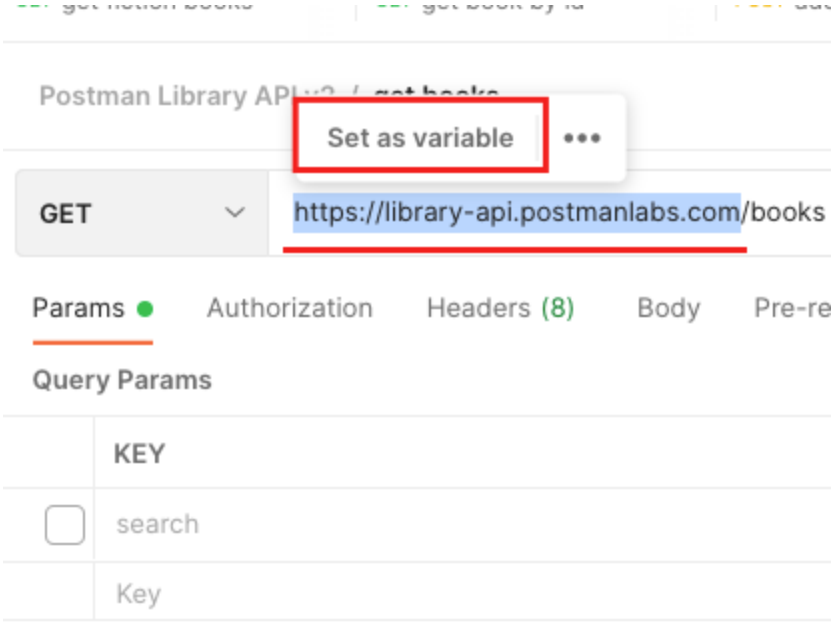
Task: Set "baseUrl" variable

One way to set a variable is to highlight the input text you'd like to convert into a variable, and follow the "Set as variable" popup instructions.

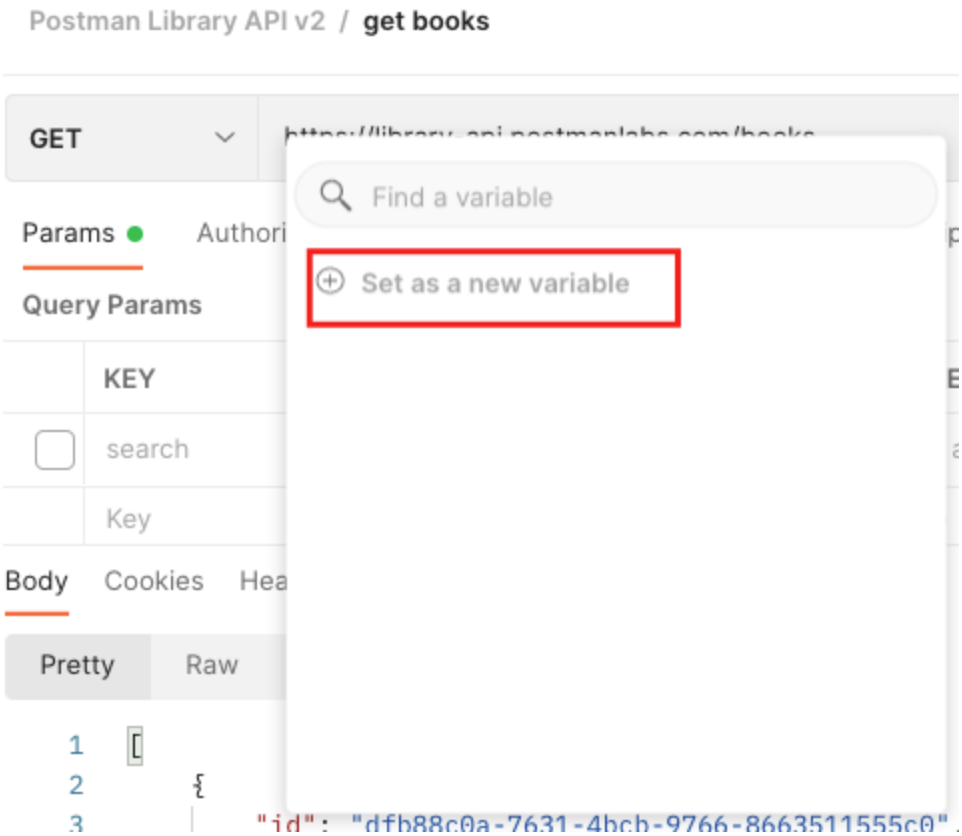
Make a `baseUrl` variable

We use the same base URL for all requests to the library API. We can simplify our requests by replacing `https://library-api.postmanlabs.com` with a variable called `{{baseUrl}}`

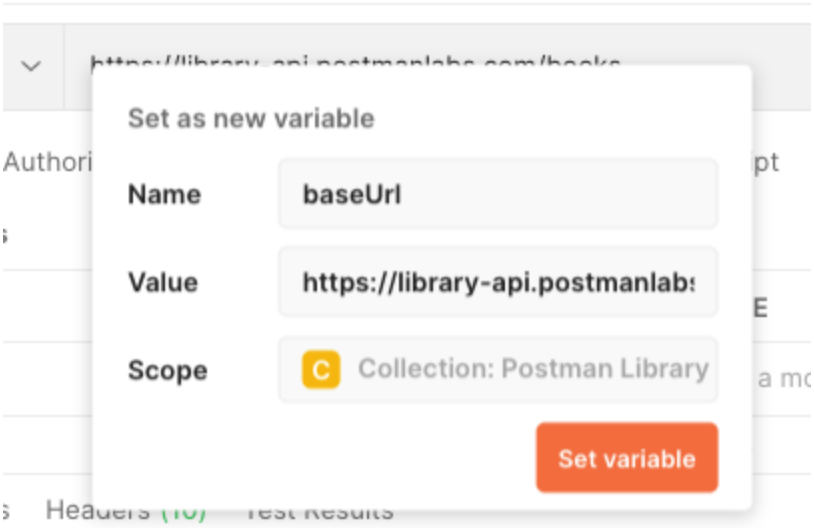
1. Go to the first "**get books**" request in your collection.
2. With your cursor, select the entire base URL of the API (`https://library-api.postmanlabs.com`). Do not include the slash `/` after `.com`.Click **Set as variable** to save the base URL to a variable.



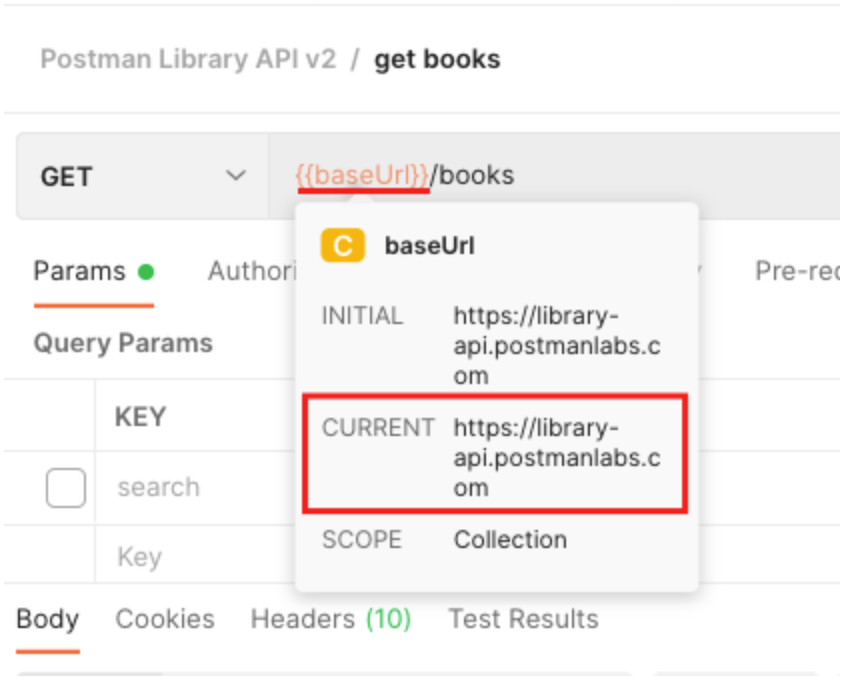
3. Click Set as a new variable



4. Name your new variable “**baseUrl**” and select “**Collection**” as the scope, then click **Set variable**Now that the variable is set, you can access the value anywhere in your collection by typing `{{baseUrl}}`



5. Hover over `{{baseUrl}}`. You will see its current value is set to `https://library-api.postmanlabs.com`

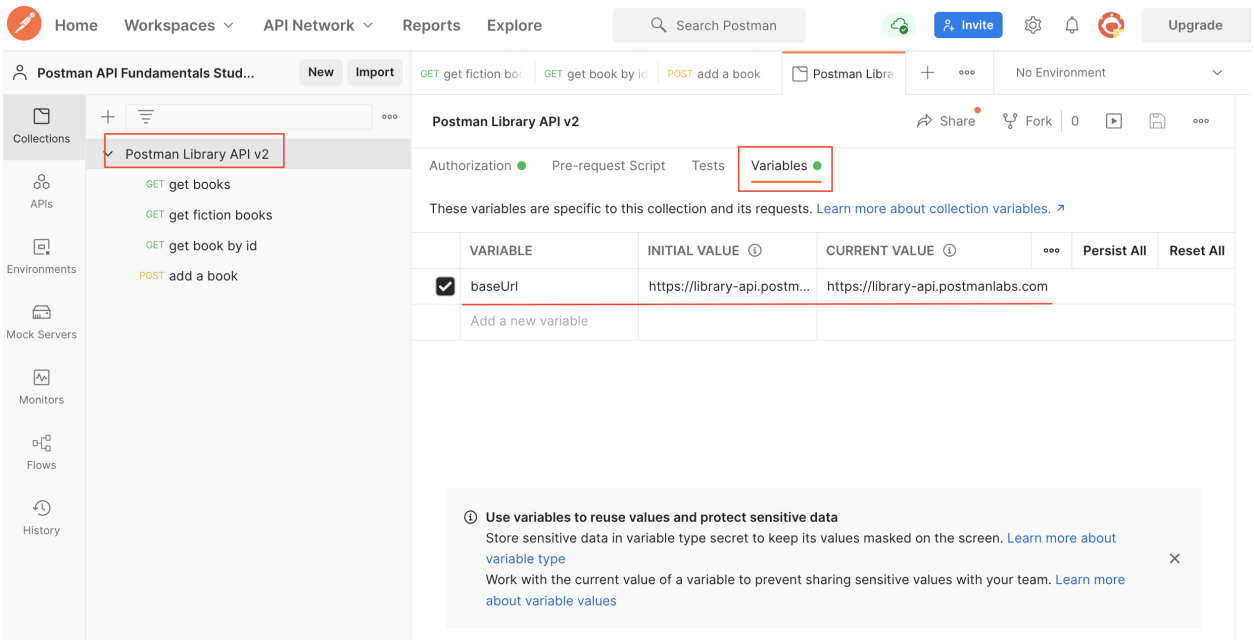


6. Now you can **send your request** and see that it works just like before! You should get a status `200 OK` response with a list of books.

Where are my variables?

You can find Collection variables on your collection.

Click on your collection, then the **Variables** tab. Here you can view and edit your variables.



Note that there are two columns:

Initial Value - the value initially set when someone forks or imports your collection. Note that if you share your collection with others they will see this value, so don't put any secrets here!

Current Value - Postman always resolves the variable to this value. This is local to your Postman account, and not public. It is good to keep secrets like API Keys ONLY in this column and not include in the Initial Value column.

Setting variables programmatically

Scripting in Postman

Postman gives you the ability to add automations and dynamic behaviors to your collections with scripting.Any Node.js code in the **Tests** tab of a request will be executed after a response comes back from the API.

The `pm` object

Postman has a helper object named `pm` that gives you access data about your postman environment, requests, responses, variables and testing utilities.

For example, you can access the JSON response body from the API with: `pm.response.json()`

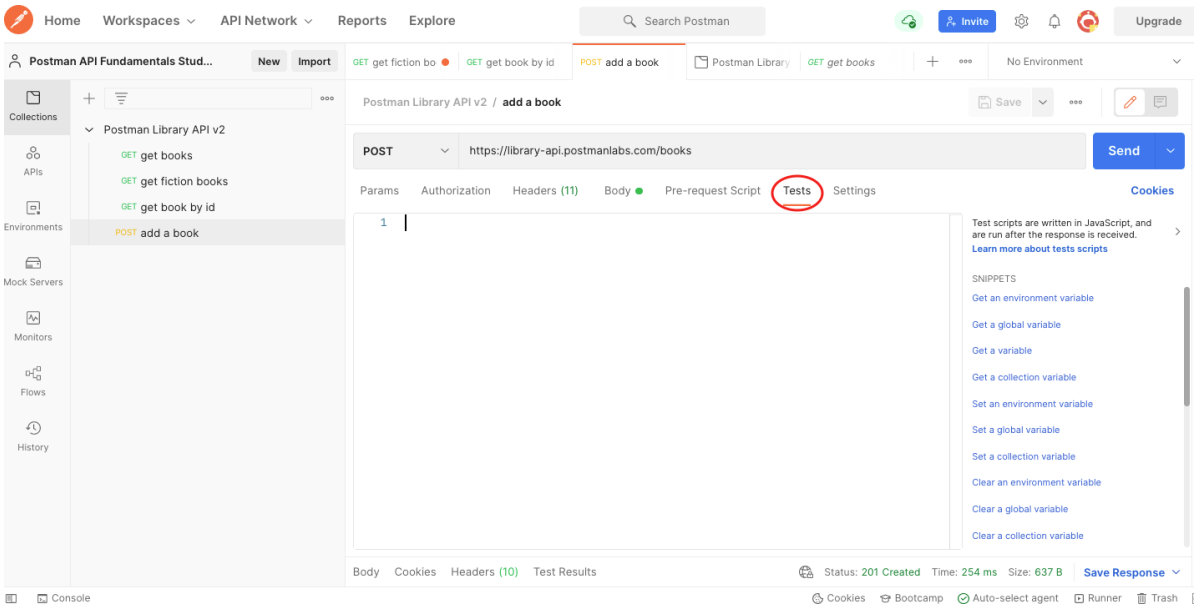
You can also programmatically get collection variables like the value of baseUrl with:

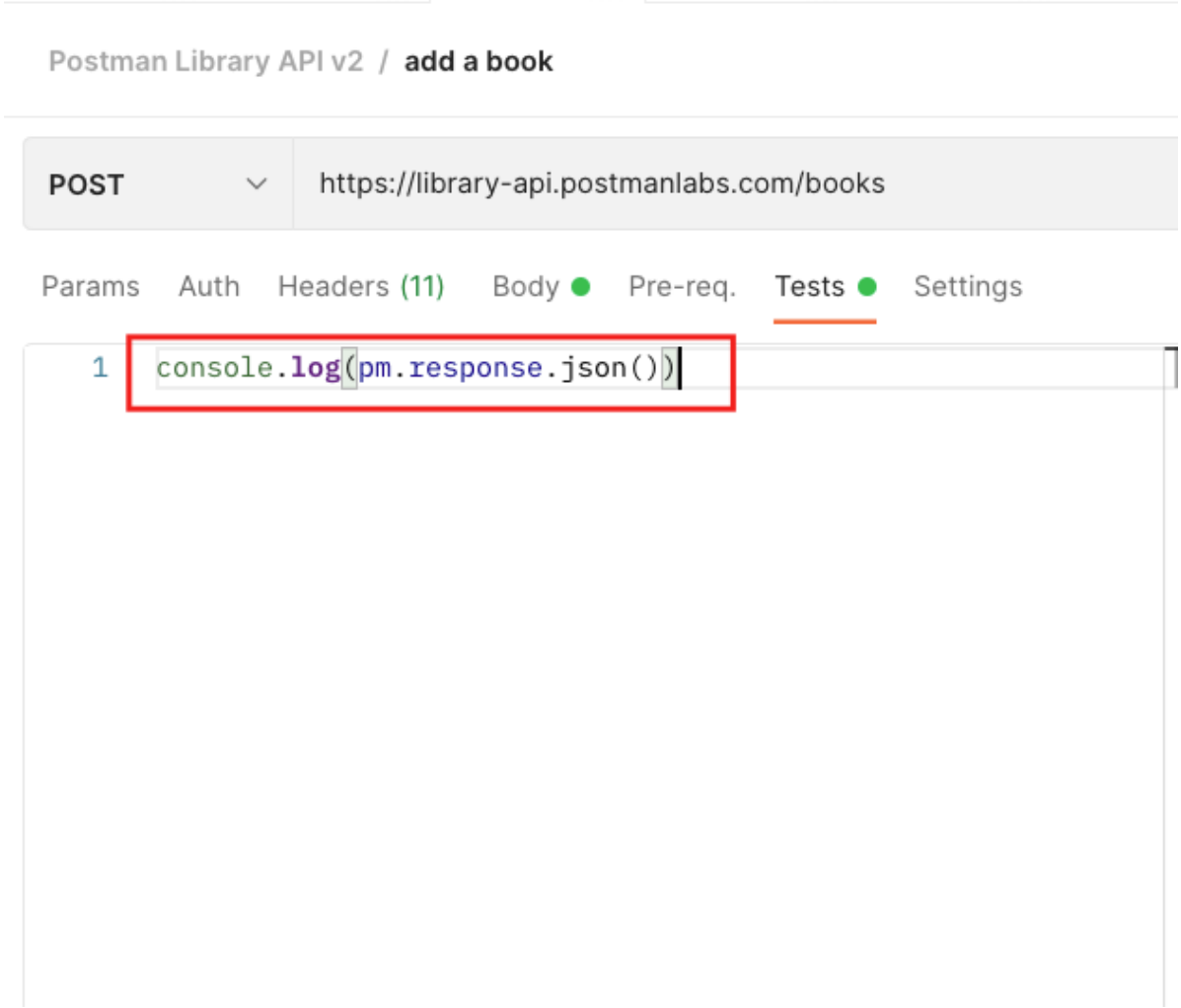
`pm.collectionVariables.get(“baseUrl”)`

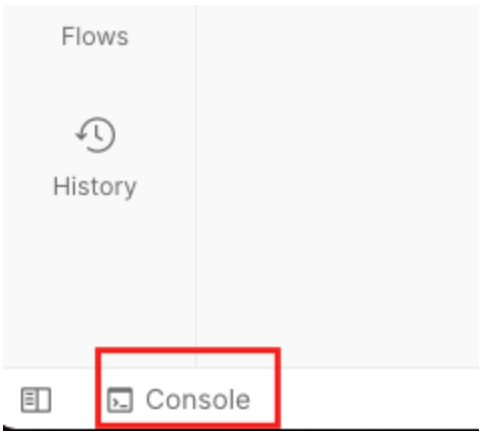
In addition to getting variables you can also set them with `pm.collectionVariables.set("variableName", "variableValue")` like this:`pm.collectionVariables.set(“myVar”, “foo”)`

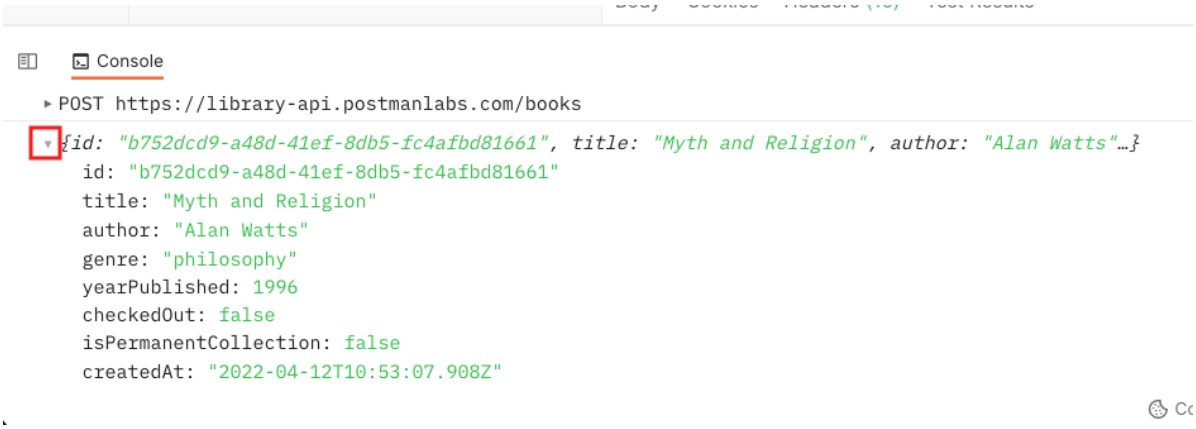
Add a script to your request

1. In your "**add a book**" request, change the book data in your **Body** to a new book you like.Open the **Tests** tab of the requestInside the Tests editor, **add this JavaScript code** to log the JSON response from the API:`console.log(pm.response.json())`**Save** your request**Send** your request. This will trigger the script in the Tests tab to run after the response comes back from the API**Open the Postman Console** in the lower left of the window Scroll to the bottom of the logs in the console. You will see your most recent request `POST https://library-api.poistmanlabs.com/books`. The response data from the API is logged in the console because of the code in our **Tests** tab! You can **expand the data** by clicking on the small arrow to the left


Home
Workspaces
API Network
Reports
Explore
Search Postman
Invite
Upgrade
Postman API Fundamentals Stud...
New
Import
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
Postman Library API v2
GET get books
GET get fiction books
GET get book by id
POST add a book
Postman Library API v2 / add a book
Save
POST
https://library-api.postmanlabs.com/books
Send
Params
Authorization
Headers (11)
Body
Pre-request Script
Tests
Settings
Cookies
1
Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests scripts
SNIPPETS
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Set a collection variable
Clear an environment variable
Clear a global variable
Clear a collection variable
Body
Cookies
Headers (10)
Test Results
Status: 201 Created
Time: 254 ms
Size: 637 B
Save Response
Console
Cookies
Bootcamp
Auto-select agent
Runner
Trash


Postman Library API v2 / add a book
POST
https://library-api.postmanlabs.com/books
Params
Auth
Headers (11)
Body
Pre-req.
Tests
Settings
1
console.log(pm.response.json())





Task: Grab the new book id

Saving a value as a variable allows you to use it in other requests. Let's grab the `id` of a newly added book and save it so we can use it in future requests.

Setting and getting collection variables

The `pm` object allows you to set and get collection variables.

To **set** a collection variable use the `.set()` method with two parameters: the variable name and the variable value

```
pm.collectionVariables.set("variableName", value)
```

To **get** a collection variable use the `.get()` method and specify the name of the variable you want to retrieve:

```
pm.collectionVariables.get("variableName")
```

Local variables

We can also store local variables inside our **Tests** script using JavaScript.

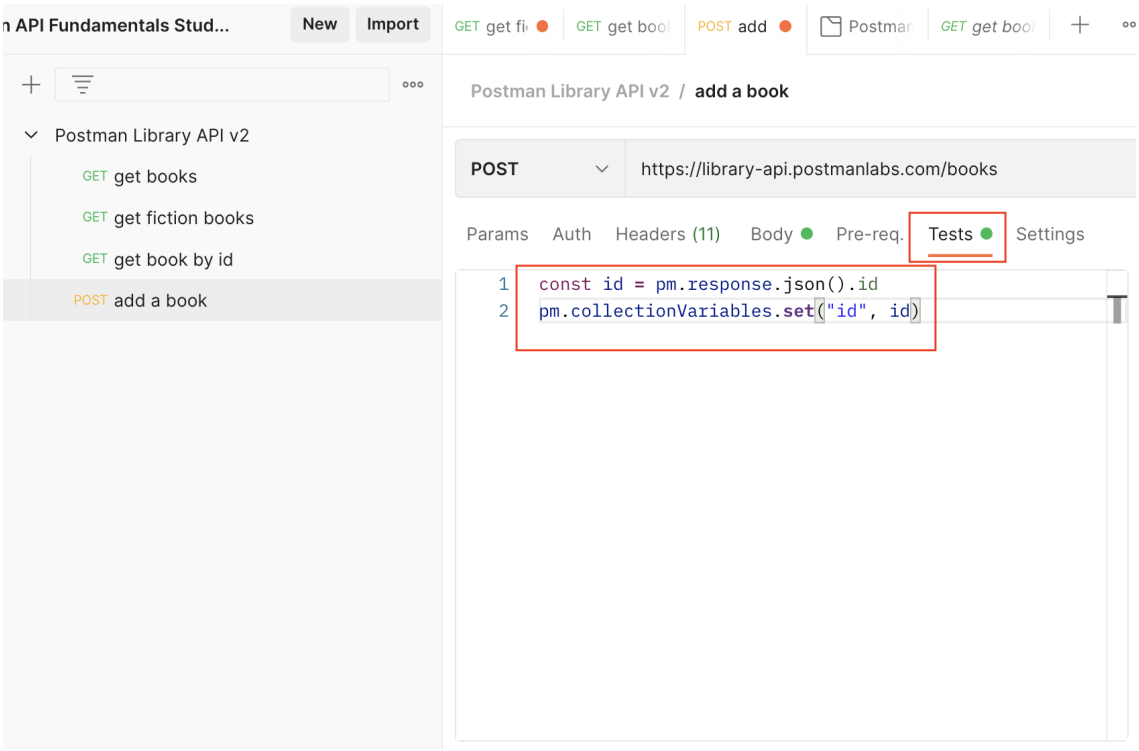
There are two ways to define a variable in JavaScript: using the `const` or `let` keywords. `const` is for variables that won't change value, whereas `let` allows you to reassign the value later.

```
// -- Defining variables with const --
const myVar = "This variable can't be reassigned"
console.log(myVar) // => This variable can't be reassigned

// attempt to reassign the value of myVar
myVar = "foo"
//=> [ERROR!] Uncaught TypeError: Assignment to constant variable.

// -- Defining variables with let --
let myVar2 = "I can change!"
console.log(myVar2) // => I can change!

myVar2 = "See, I changed!"
console.log(myVar2) // => See, I changed!
```

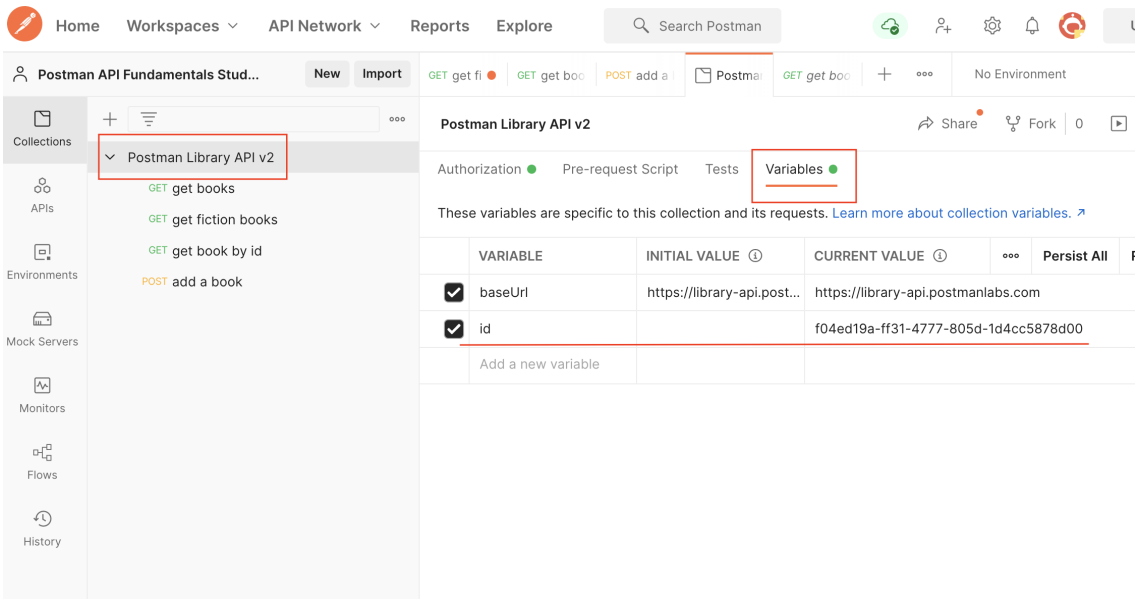
Set the new book `id` as a variable

1. In the **Body** tab of the "**add a book**" request, **change the details to yet another new book to add**!
2. In the **Tests** tab of the "add a book" request, replace the console.log() statement with this code:First we assign the `id` value from our API response to a local `const` variable named `id` (we use `const` because this value doesn't change in our script). The second line sets this value to a collection variable called `id`.

```
const id = pm.response.json().id
pm.collectionVariables.set("id", id)
```



3. **Save** your request
4. **Send** your request.When the `201` response comes back from the API with you newly created book, the Tests script will run and save the book's `id` as a collection variable automatically.
5. View your collection variables by clicking on your **Postman Library API v2** collection, then the **Variables** tab.The `id` variable has been automatically assigned the id of your new book as its Current Value!



You can now use `{{id}}` anywhere in your collection to access this value! Which will come in handy for what we do next...

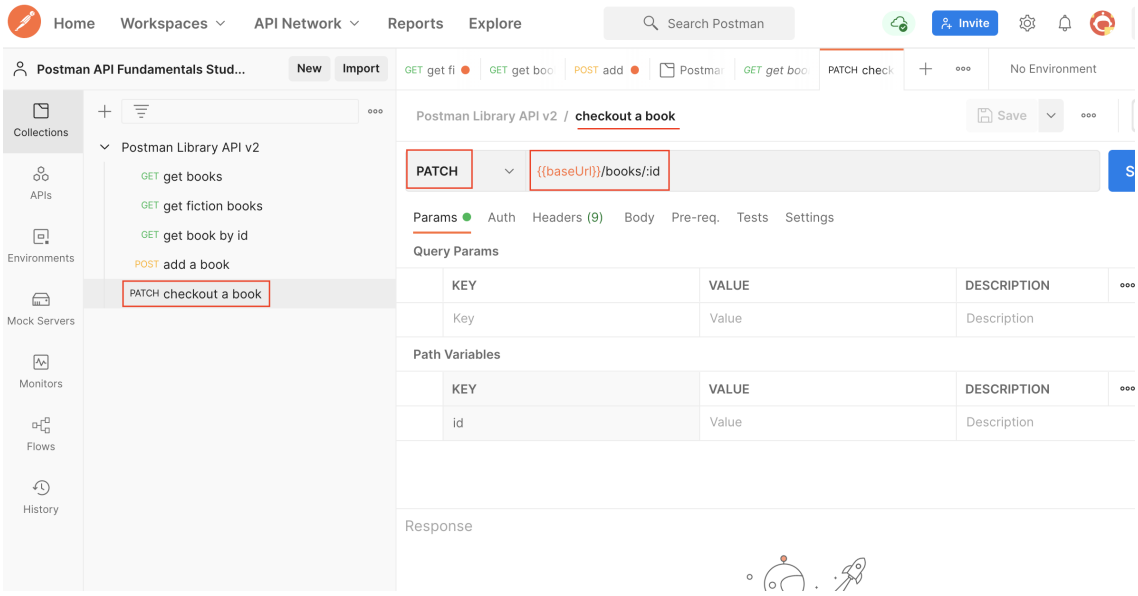
Task: Checkout your book

Someone wants to check out the book you just added!As librarian, you will update the library database via the API to mark the book's `checkedOut` status from `false` to `true`.

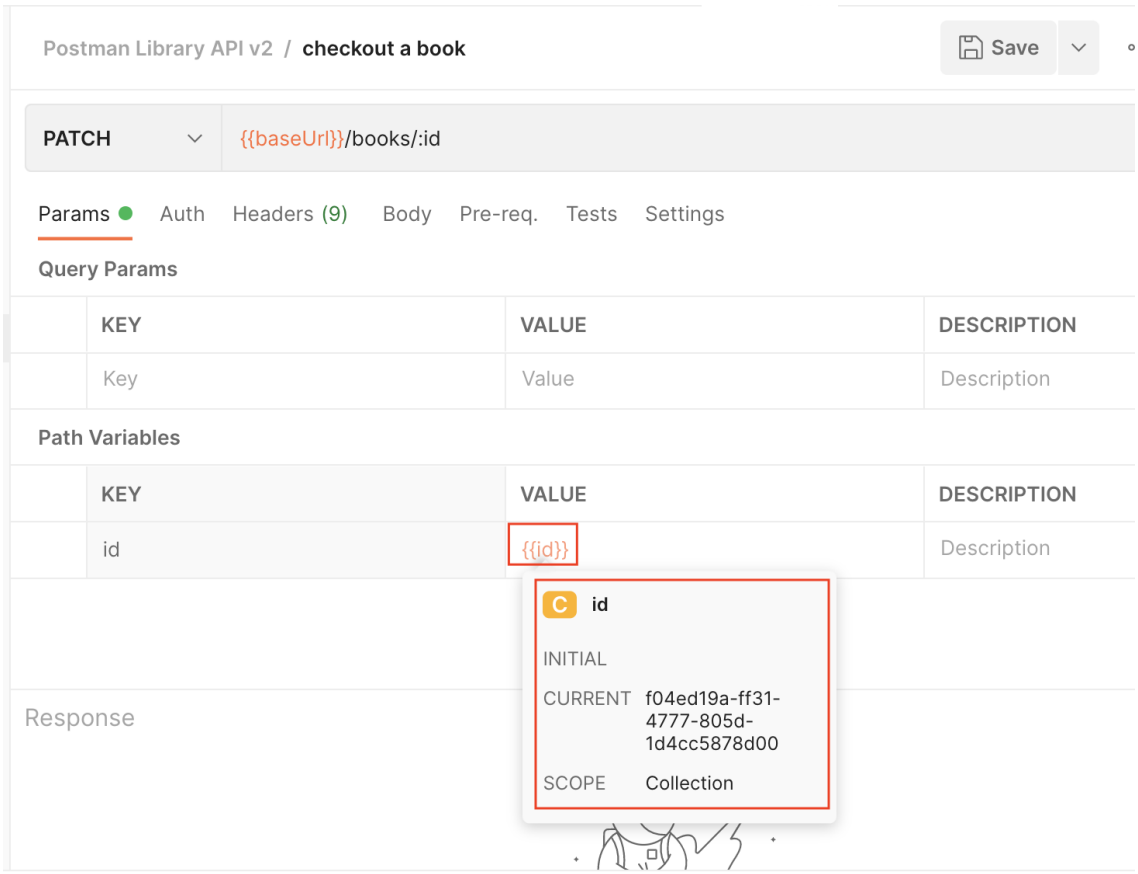
The API documentation shows we can **update a book by id** by making a request (authorized with the API Key) with the updated information to: `PATCH https://library-api.postmanlabs.com/books/:id`

Make a request to update the book

1. Hover on your **Postman Library API v2** collection, click the three dots, and select **Add request.** Name your new request **"checkout a book"**
2. Set the request method to **PATCH** and the request URL to `{{baseUrl}}/books/:id`

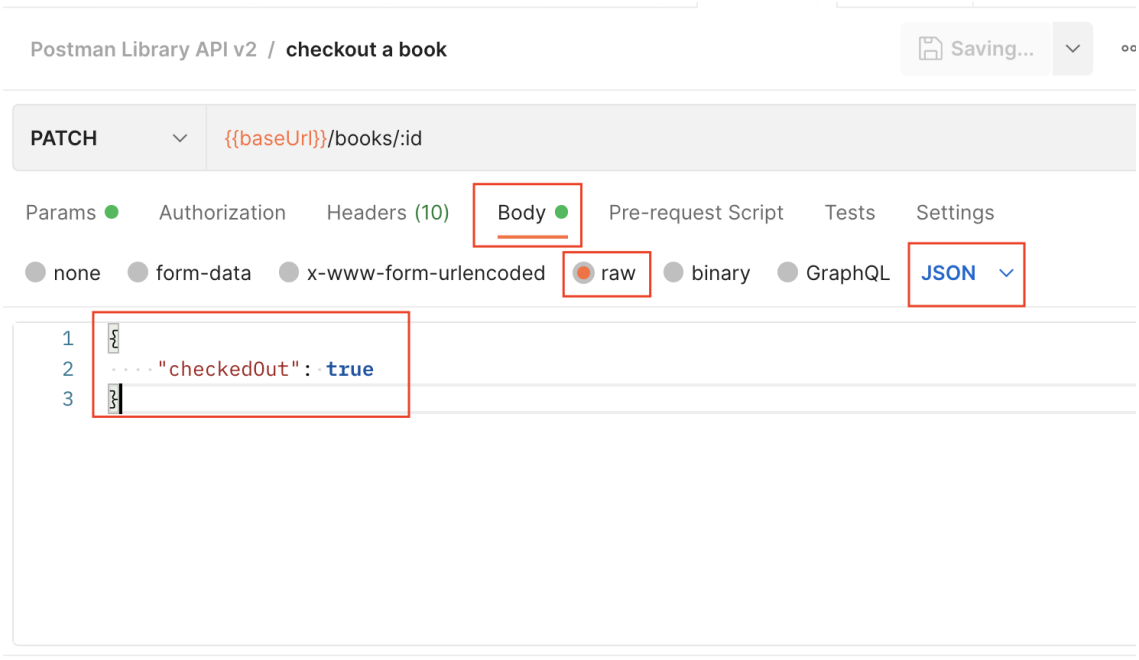


3. Set the value of path variable `id` to `{{id}}`.This will use the value of our collection variable named `id` that was set in the script of the"add a book" request

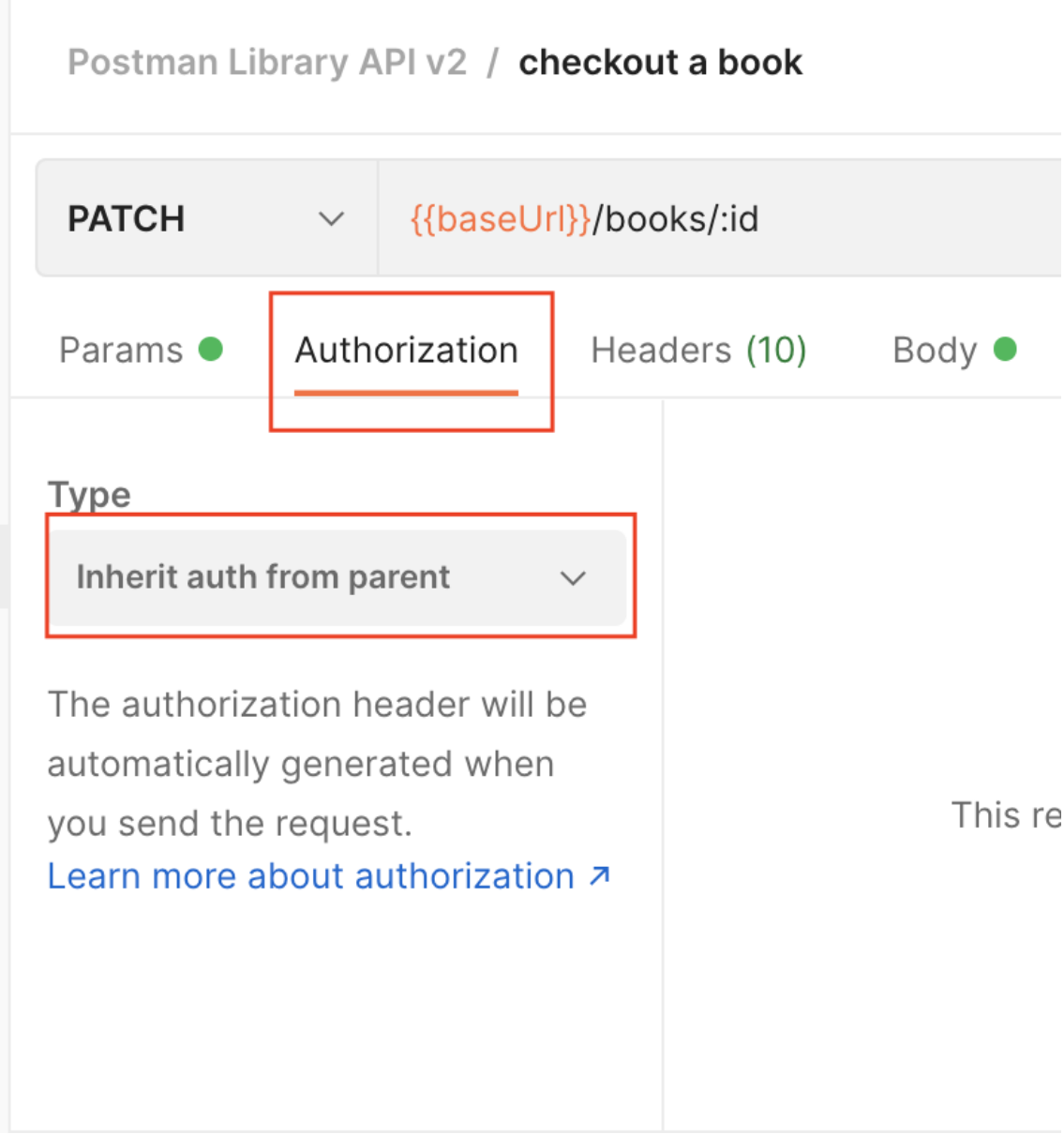


4. Add a **raw JSON** body in the **Body** tab for updating the `checkedOut` property to `true`:

```
{
  "checkedOut": true
}
```



5. Make sure in the **Authorization (Auth) tab** that the Auth type is set to "Inherit from parent". This will use the API Key set at the collection level on our PATCH request.



6. **Save** your request
7. **Send** your request

You should get a

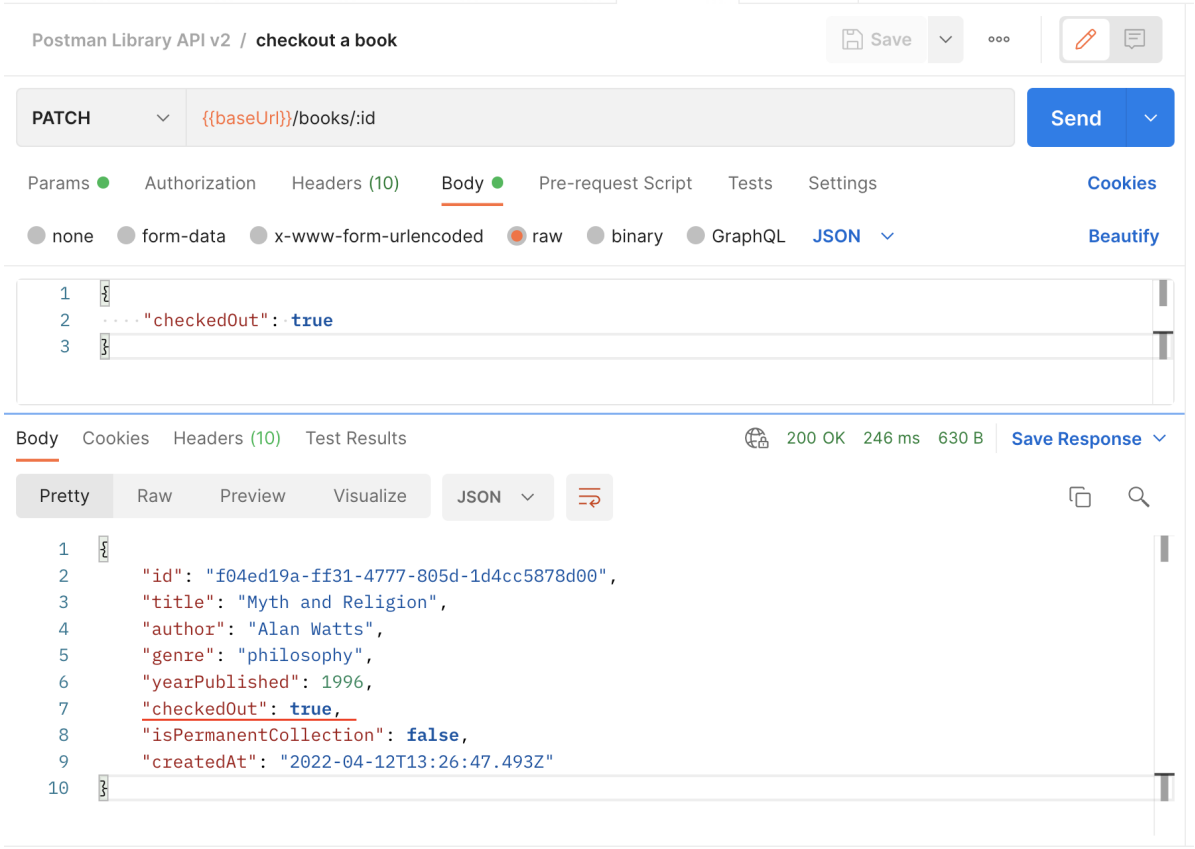
```
200 OK
```

response that shows the updated data about your book. Notice how

```
checkedOut
```

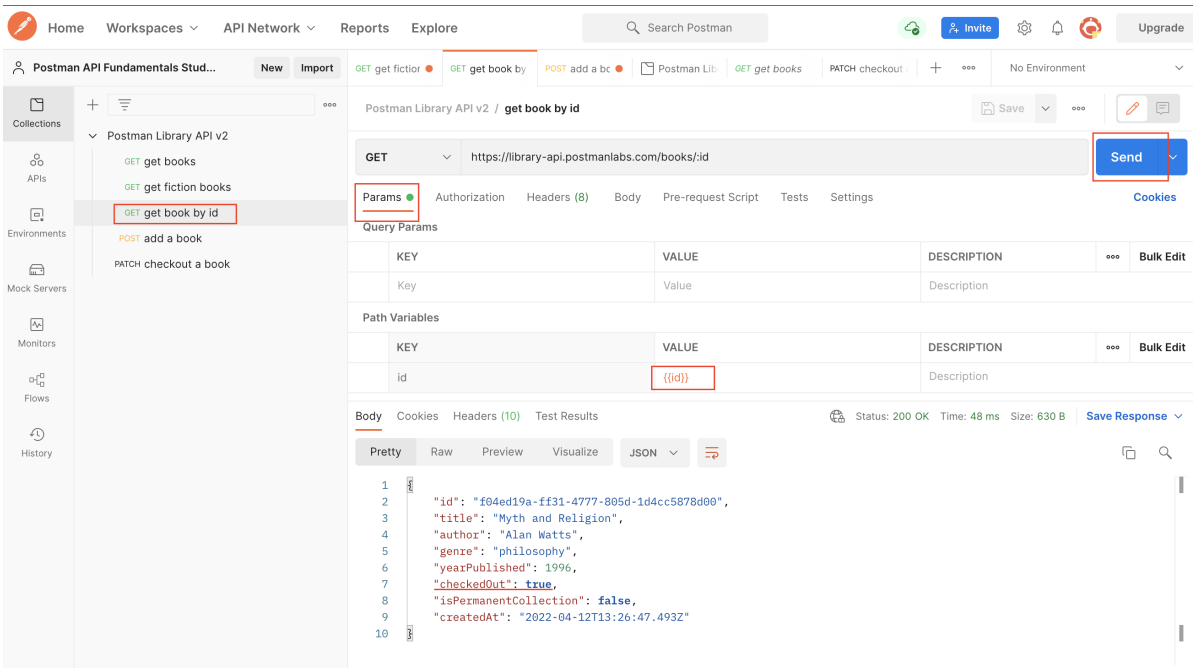
is now

```
true
```



Your book is updated!

Now if you return to your "get book by id" request, update the id path variable value to `{{id}}` , **Save** and **Send**, you will see the same updated data!



Task: Delete your book

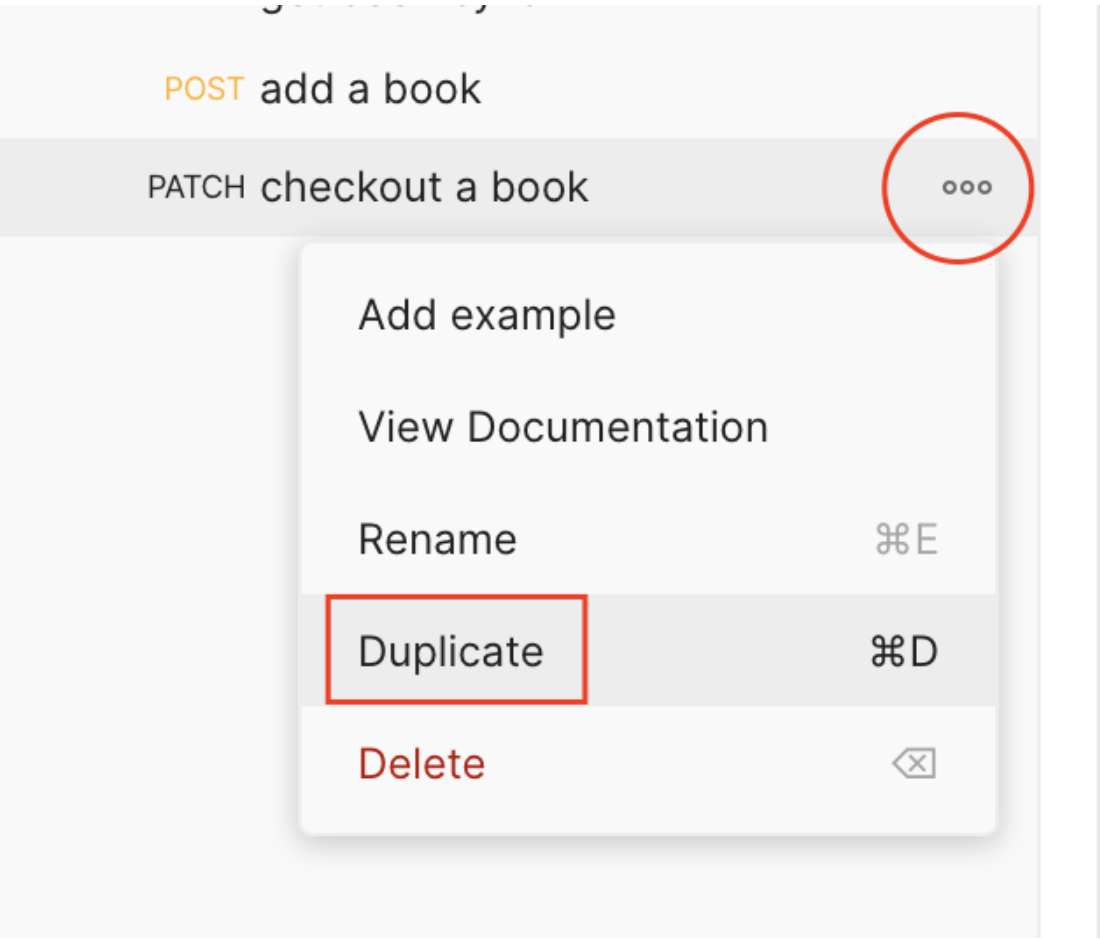
Oops! The person that checked out your book accidentally lost it... you will need to delete it from the library database.

The API documentation shows how we can delete books with the `DELETE /books/:id path`

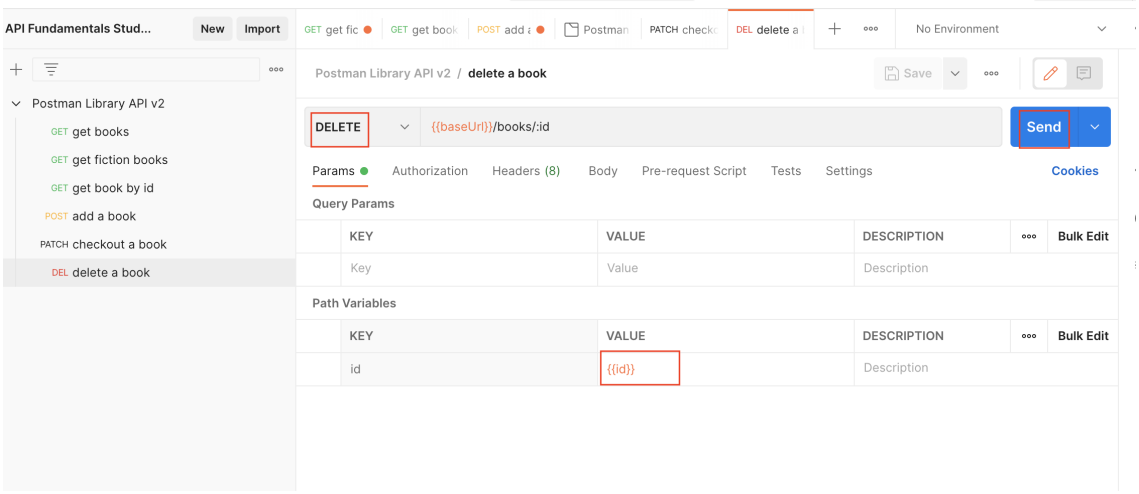
Make a new request

The `DELETE` request has a similar format to the `PATCH` request, so let's copy the `PATCH` request to make our new request.

1. Hover over your "checkout a book" requet, click the three dots icon, then select **Duplicate** to create a copy of the request. Rename your new request "**delete a book**"



2. Set the request method of the "delete a book" request to `DELETE`
3. Make sure the request **Body** is empty. This endpoint does not require a body.
4. In the **Params** tab**,** make sure the path variable id is set to `{{id}}` Your request should now look like this:



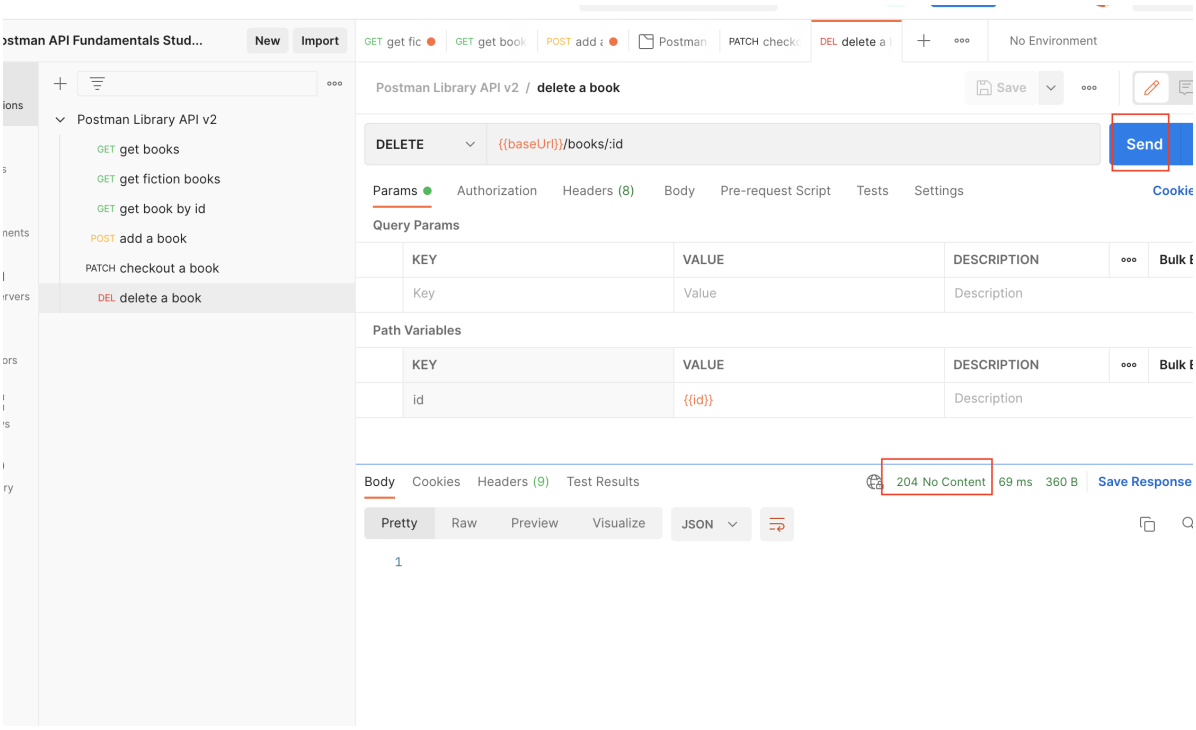
5. **Save** your request
6. **Send** your request

You should get a

```
204 No Content
```

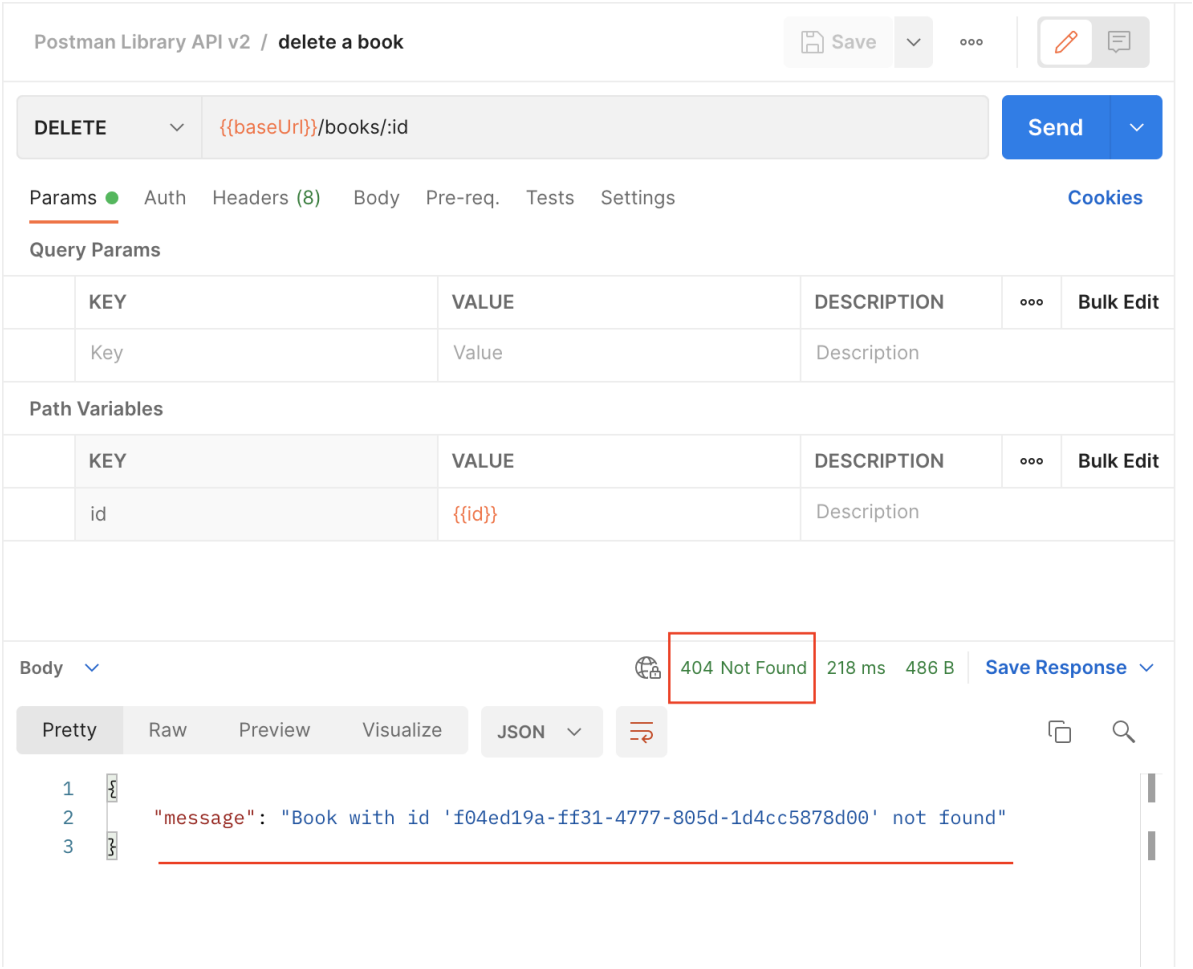
response from the API. This means the server successfully deleted the book, and won't send any response body back.

Remember: if you ever wonder what a status code means, you can hover on it in Postman for an explanation!



Is it really gone?

Without changing anything, try sending your request again.Since you are sending a request to delete a book with an id that no longer exists, you get a `404` error!



Previous - Task: Checkout your bookPostman's codegen feature Next

Postman's codegen feature

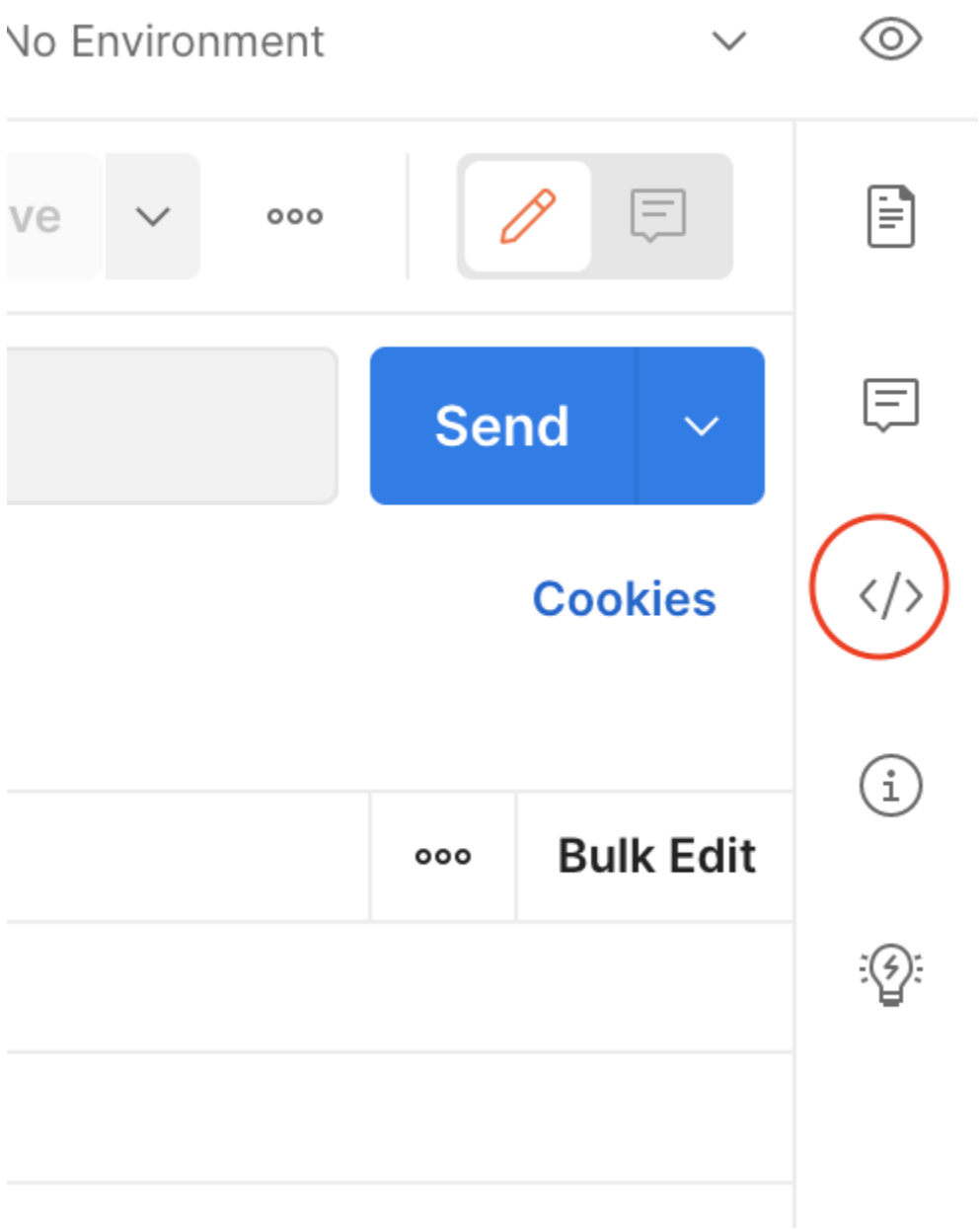
Postman can help you integrate API calls into your applications by generating code snippets in various coding languages.

Generating code snippets

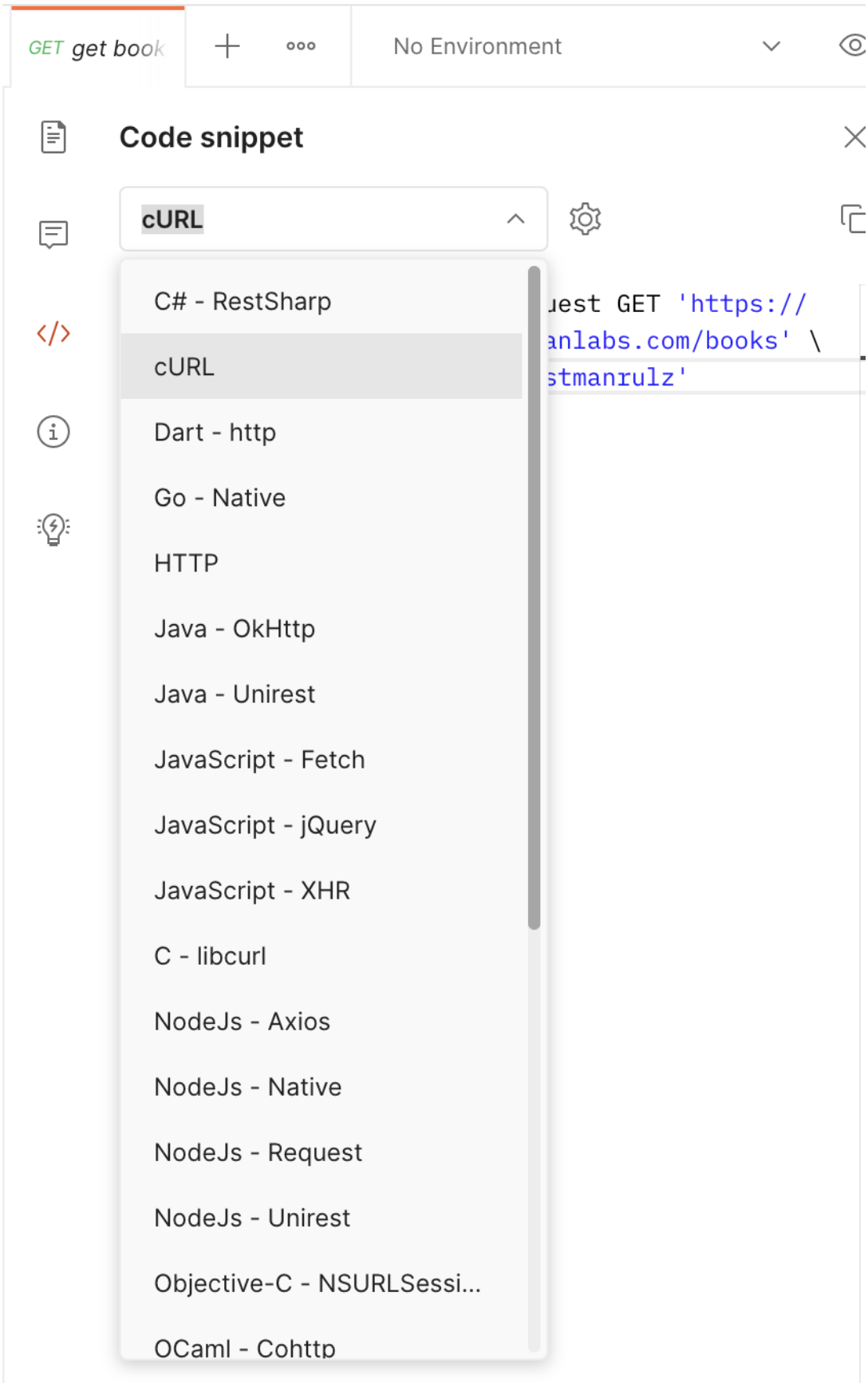
Every request you made has a code generation tab you can access from the code

```
</>
```

icon in the far right



Once the code snippet generator pane is open, you can select from a dropdown to generate the API request in common coding languages like Python, JavaScript, C, NodeJS and more.


GET get book
No Environment
Code snippet
cURL
C# - RestSharp
cURL
Dart - http
Go - Native
HTTP
Java - OkHttp
Java - Unirest
JavaScript - Fetch
JavaScript - jQuery
JavaScript - XHR
C - libcurl
NodeJs - Axios
NodeJs - Native
NodeJs - Request
NodeJs - Unirest
Objective-C - NSURLSessi...
OCaml - Cohttp
uest GET 'https://
anlabs.com/books' \
stmanrulz'

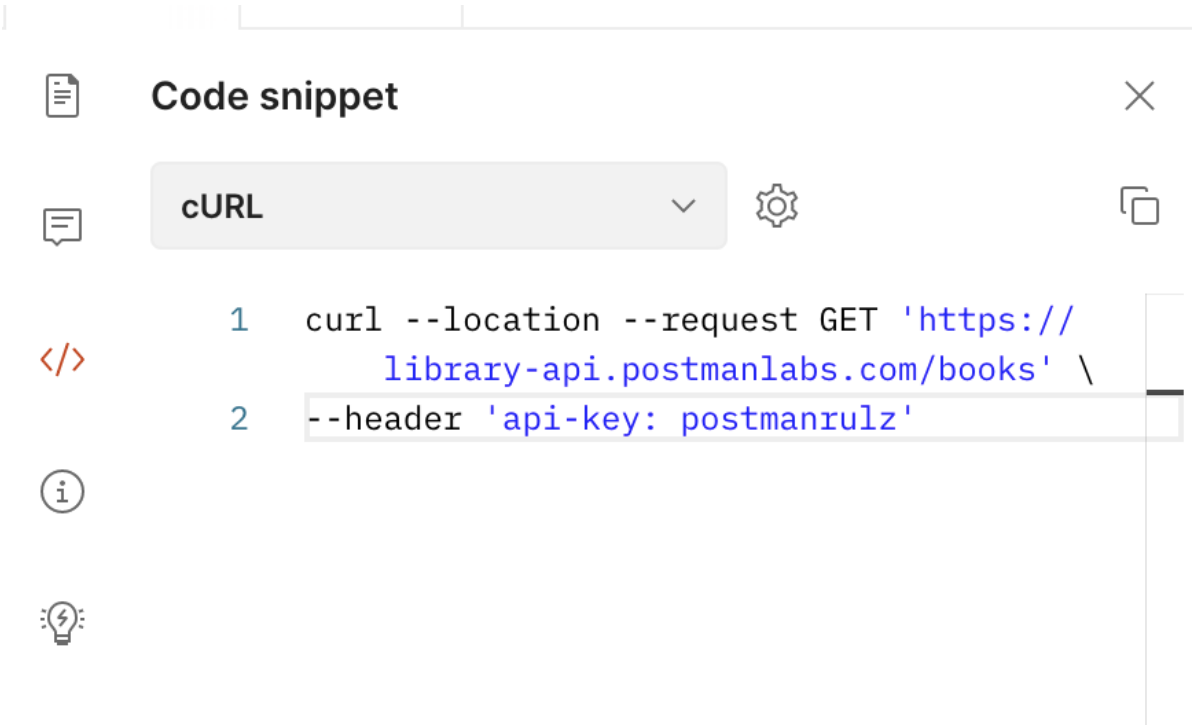
For example, we can generate the

```
GET /books
```

request in

```
cURL
```

syntax:



If we copy this snippet into the terminal and press Enter, we make the API call and the response body is printed:

```
➜  ~ curl --location --request GET 'https://library-api.postmanlabs.com/books' \
--header 'api-key: postmanrulz'
[{"id":"b752dcd9-a48d-41ef-8db5-fc4afbd81661","title":"Myth and Religion","author":
"Alan Watts","genre":"philosophy","yearPublished":1996,"checkedOut":false,"isPerman
entCollection":false,"createdAt":"2022-04-12T10:53:07.908Z"},{"id":"7ad72643-67af-4
e0d-8d96-551d85327fd6","title":"To Kill a Mockingbird","author":"Harper Lee","genre
":"fiction","yearPublished":1960,"checkedOut":false,"isPermanentCollection":false,"
createdAt":"2022-04-12T10:51:17.541Z"},{"id":"dfb88c0a-7631-4bcb-9766-8663511555c0"
,"title":"To Kill a Mockingbird","author":"Harper Lee","genre":"fiction","yearPubli
shed":1960,"checkedOut":false,"isPermanentCollection":false,"createdAt":"2022-04-10
T21:43:56.750Z"},{"id":"06fb002a-ab51-4d0e-80fb-3b3b802bf7e8","title":"To Kill a Mo
ckingbird","author":"Harper Lee","genre":"fiction","yearPublished":1960,"checkedOut
":false,"isPermanentCollection":false,"createdAt":"2022-04-10T20:43:33.783Z"},{"id"
:"9c49e2cd-738d-4387-ba7f-4b7bc4911807","title":"Myth and Religion","author":"Alan
Watts","genre":"philosophy","yearPublished":1997,"checkedOut":false,"isPermanentCol
lection":false,"createdAt":"2022-03-31T15:47:53.093Z"},{"id":"85a5424d-535d-48a9-a1
63-8627b84662d7","title":"Continuous API Management: Making the Right Decisions in
an Evolving Landscape","author":"Mehdi Medjaoui, Erik Wilde, Ronnie Mitra, Mike Amu
ndsen","genre":"computers","yearPublished":2018,"checkedOut":false,"isPermanentColl
ection":true,"createdAt":"2022-03-30T00:54:52.606Z"},{"id":"03daae04-c778-45fb-9acb
-4c6d19067a9c","title":"Go Design Patterns","author":"Mario Castro Contreras","genr
e":"computers","yearPublished":2017,"checkedOut":false,"isPermanentCollection":true
,"createdAt":"2022-03-30T00:54:52.606Z"},{"id":"539f89ed-df63-469c-94ce-157e75366de
c","title":"Domain-Driven Design: Tackling Complexity in the Heart of Software","au
thor":"Eric Evans","genre":"computers","yearPublished":2003,"checkedOut":false,"isP
ermanentCollection":true,"createdAt":"2022-03-30T00:54:52.606Z"},{"id":"29cd820f-82
f9-4b45-a7f4-0924111b5b89","title":"Ficciones","author":"Jorge Luis Borges","genre"
:"fiction","yearPublished":1944,"checkedOut":false,"isPermanentCollection":true,"cr
eatedAt":"2022-03-30T00:54:52.606Z"},{"id":"14b533eb-5ef3-422a-b037-7b6161ee7020","
```

It's definitely easier to read API responses as a human in Postman... 😅