



# API and Postman

Application Programming Interfaces allow services to communicate with each other.

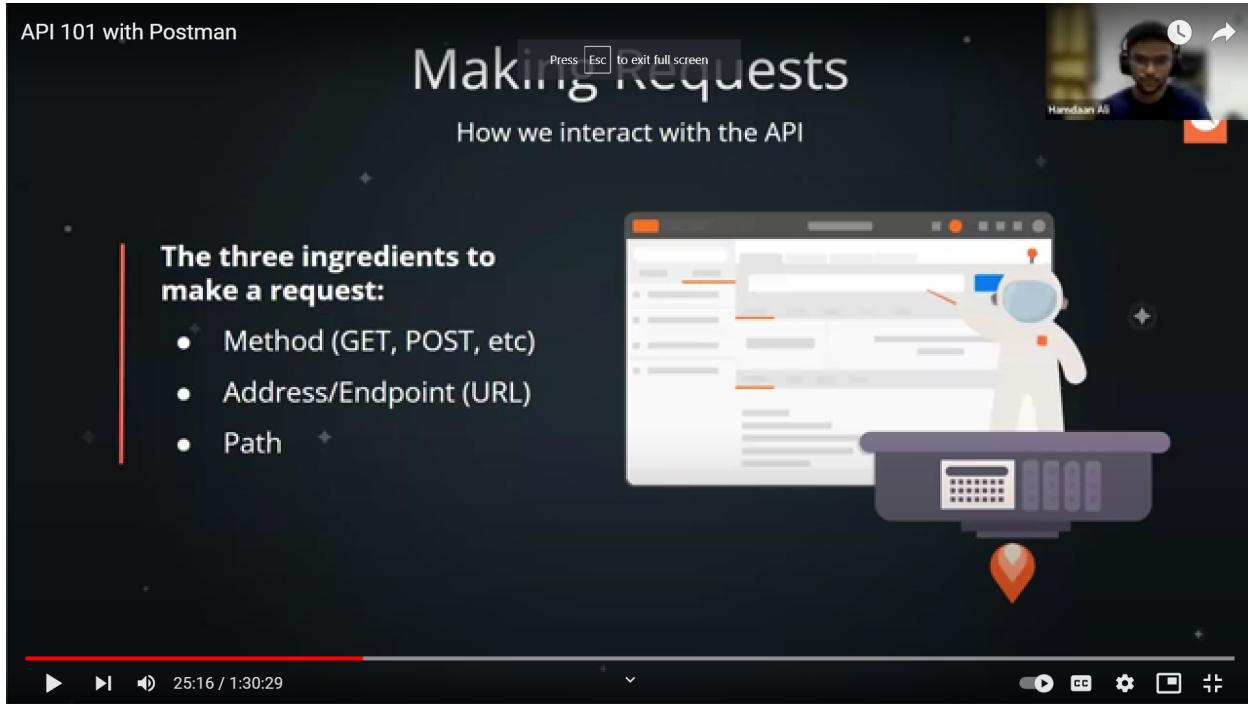
Developers no longer need to create every service from scratch.

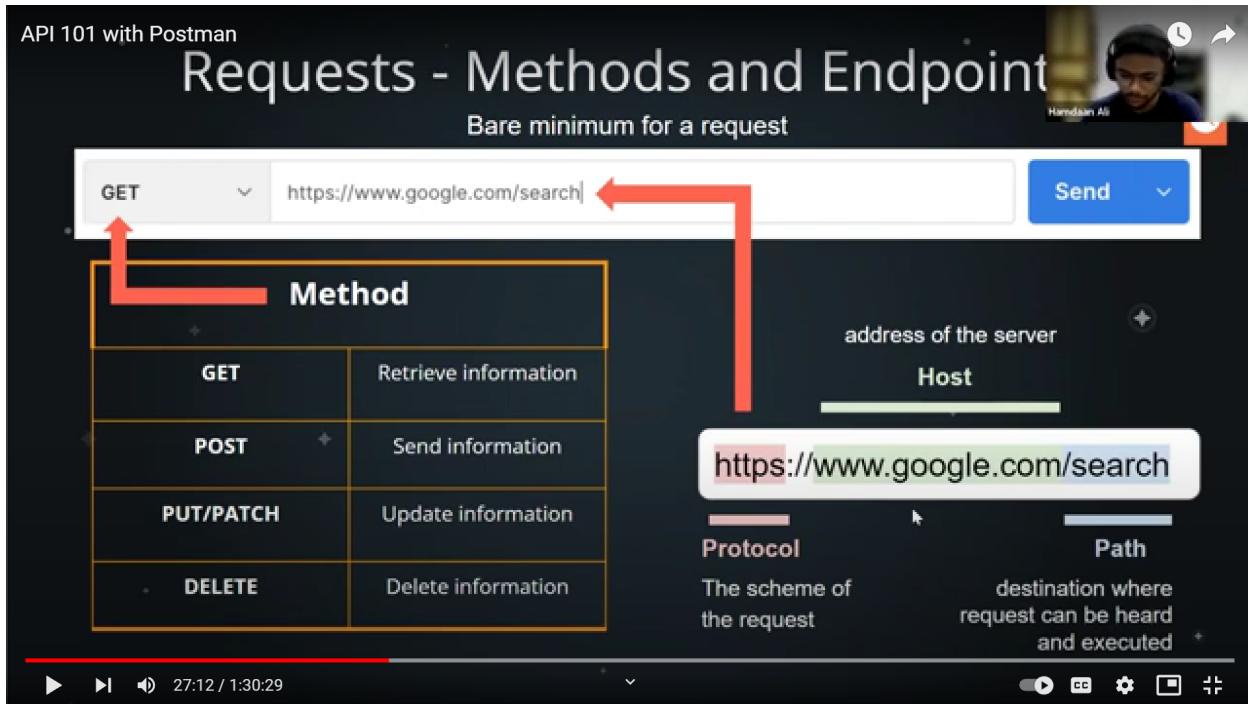
APIs allow developers to access data from a service like google or twitter without any knowledge of how the codebase has been implemented.

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.

The data payload

- Optional, but often supplied with POST and PUT requests
- Data types
  - form data
  - JSON
  - text
  - HTML
  - XML
  - files
  - GraphQL
  - ... and more!

**JSON**

```
{  
  "name": "Jane Doe",  
  "email": "janedoe@email.com",  
  "birthYear": 1970  
}
```

# Receiving responses

The image is a screenshot of a presentation slide titled "Receiving Responses". The title is displayed prominently at the top center in a large, white, sans-serif font. Below the title, there is a small instruction "Press Esc to exit full screen". In the top right corner, there is a video player interface showing a man named Harshavardhan Aji wearing headphones and a blue shirt. The video player includes standard controls like play, pause, volume, and a progress bar indicating the video is at 30:58 / 1:30:29. On the left side of the slide, there is a vertical red bar. To the right of this bar, the section title "Response elements" is written in bold black text. Below the title, there is a bulleted list of three items: "Status codes (200 OK, 201 Created, 404 Not found)", "Headers", and "Accessing body data". To the right of the list, there is a graphic illustration of a purple satellite dish antenna pointing towards a bright orange sphere with yellow lightning bolts around it. A small white character with a backpack and a camera is standing next to the dish. The overall background is dark gray.

# Documentation

The screenshot shows the Postman Student API 101 Workshop overview page. The left sidebar contains sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main content area displays the workshop's purpose, sections, and activity feed. The activity feed shows contributions from Ali Mustafa on November 29, 2022, and November 25, 2022. The right sidebar lists workspace components like Requests, Collections, APIs, Environments, Mock Servers, Monitors, and Flows. The bottom navigation bar includes links for Cookies, Auto-select agent, Runner, Trash, and a timestamp of 4:42 PM on 3/7/2023.

# Requesting

The screenshot shows the Postman interface with a request for "Get Data" from the "Basics of API" collection. The request method is GET, and the URL is https://api01.up.railway.app/. The "Params" tab is selected, showing a single query parameter "Key" with value "Value". The "Body" tab shows the response body: "Hello World!". The status bar at the bottom indicates a 200 OK status, 297 ms time, and 736 B size. The left sidebar shows the workspace structure and a progress bar for "Start working with APIs" at 67% complete. The bottom navigation bar includes links for Cookies, Capture requests, Runner, Trash, and a timestamp of 5:04 PM on 3/7/2023.

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area displays a collection named "Basics of API" by Subroto Banerjee. A specific request titled "Get Data" is selected, showing a GET method pointing to <https://api101.up.railway.app/joke>. The "Body" tab of the request details section shows the following JSON response:

```
1  {
2   "id": 1,
3   "author": "NA",
4   "joke": "Q: How did the Coder CEO build his company headquarters? A: By calling the Constructor();",
5   "source": "https://github.com/shrutikapoor08/devjoke/blob/master/README.md"
6 }
```

The "Body" tab also indicates a Status: 200 OK, Time: 7.95 s, and Size: 930 B. Below the request details, there's a "Start working with APIs" section with a progress bar at 67% complete. The bottom of the screen shows the Windows taskbar with various pinned icons.

This screenshot is nearly identical to the one above, showing the same Postman interface and collection. The difference is in the request URL, which is now <https://api101.up.railway.app/joke/5>. The response body is different, reflecting the specific joke ID 5:

```
1  [
2   {
3     "id": 5,
4     "author": "Ali Mustafa",
5     "joke": "Bugs are features",
6     "source": "iali.dev"
7   }
8 ]
```

The rest of the interface, including the sidebar, requests list, and status bar, remains the same.

## using POST :

The screenshot shows the Postman interface with a collection named "Basics of API". A POST request is made to <https://api101.up.railway.app/joke/5>. The Headers tab shows "Content-Type: application/json". The Body tab is empty. The response status is 404 Not Found. The response body is:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8">
6   <title>Error</title>
7 </head>
8
9 <body>
10  <p>Cannot POST /joke/5</p>
11 </body>
12
13 </html>
```

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 :-

The screenshot shows the Postman interface with a collection named "Basics of API". A POST request is made to <https://api101.up.railway.app/joke>. The Headers tab shows "Content-Type: application/json". The Body tab has a JSON payload:

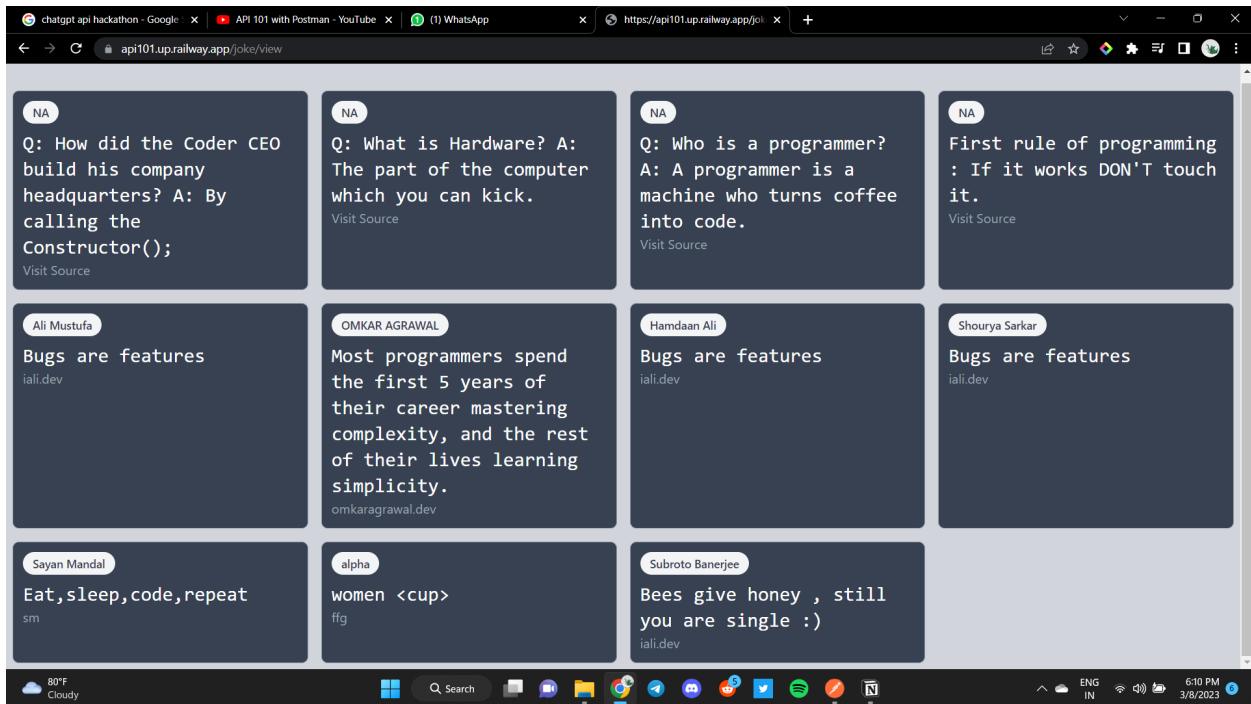
```
1 {
2   "author": "Subroto Banerjee",
3   "joke": "Bees give honey , still you are single :)",
4   "source": "iall.dev"
5 }
```

The response status is 201 Created. The response body is:

```
1 {
2   "id": 2317,
3   "author": "Subroto Banerjee",
4   "joke": "Bees give honey , still you are single :)",
5   "source": "iall.dev",
6   "id": 2317
7 }
```

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 :

The screenshot shows the Postman application interface. In the top navigation bar, 'Home', 'Workspaces', 'API Network', and 'Explore' are visible. The search bar contains 'Search Postman'. On the right, there are buttons for 'Invite', 'Upgrade', and environment dropdowns. The left sidebar has sections for 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. A 'Start working with APIs' section is present with a progress bar at 67% complete. The main workspace shows a collection named 'Basics of API' by Subroto Banerjee. A 'Get Data' endpoint is selected, showing a PUT method with the URL <https://api101.up.railway.app/joke>. The 'Body' tab is selected, displaying a JSON payload:

```
1 *id: "2338",
2 *autho: "Subroto Banerjee",
3 *joke": "Bees give honey , still you are single :)",
4 *source": "ssss"
```

The response status is 204 No Content, time 330 ms, size 670 B. Below the body, there are tabs for Body, Cookies, Headers (15), and Test Results. The test results show a single entry with value 1. The bottom of the screen shows system icons for weather (80°F Cloudy), search, file, browser, and social media, along with a taskbar showing various open applications.

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

The results will show as follows:

The screenshot shows a web browser window with multiple tabs. The active tab is <https://api101.up.railway.app/joke/view>. The page displays two joke cards side-by-side. The left card, associated with user 'Hamdaan Ali', shows the joke 'Bugs are features' from 'iali.dev'. The right card, associated with user 'Shourya Sarkar', also shows the same joke. Both cards have identical content: 'Bugs are features' and 'iali.dev'. Below these cards, another row shows two more cards, both attributed to 'Subroto Banerjee'. These cards also display the same joke: 'Bees give honey , still you are single :)'. The source for these cards is 'iali.dev'. The bottom of the browser window shows the taskbar with various application icons and system status indicators.

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:

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace shows a collection named 'Basics of API' with a single GET request named 'Get Data'. The request method is set to 'DELETE'. The URL is 'https://api101.up.railway.app/joke/:id'. In the 'Params' section, there is one entry: 'Key' with 'Value' and 'Description'. In the 'Path Variables' section, there is one entry: 'id' with 'Value' and 'Description'. Below the request details, the 'Body' tab shows a response with a status of '204 No Content'. At the bottom, there are various system icons and a status bar indicating '80°F Cloudy' and the date '3/8/2023'.

### query parameter :

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' with a 'Collections' section containing 'Basics of API' by Subroto Banerjee, which has a 'Get Data' item. Below this is a 'Start working with APIs' section with a progress bar at 67% complete. The main workspace shows a 'DELETE' request to the URL <https://api101.up.railway.app/joke?id>. The 'Params' tab is selected, showing a single parameter 'id' with a value of 'Key'. The 'Body' tab shows a response with the number '1'. The bottom status bar indicates a status of 204 No Content, time 330 ms, and size 670 B.

## deletion :

This screenshot is similar to the previous one but shows a different request. The URL is now <https://api101.up.railway.app/joke:id>. The 'Path Variables' tab is selected, showing a path variable 'id' with a value of '2318'. The 'Body' tab shows a response message: '1 Joke with id 2318 is deleted.' The bottom status bar indicates a status of 200 OK, time 1079 ms, and size 755 B.

# 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 ( <b>Read</b> )
POST	Send data ( <b>Create</b> )
PUT/PATCH	Update data ( <b>Update</b> )* <b>PUT</b> usually replaces an entire resource, whereas <b>PATCH</b> usually is for partial updates
DELETE	Delete data ( <b>Delete</b> )

## 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	<b>200</b> - OK <b>201</b> - Created <b>204</b> - No content (silent OK)
<b>3xx</b>	Redirection	<b>301</b> - Moved (path changed)
<b>4xx</b>	Client error	<b>400</b> - Bad request <b>401</b> - Unauthorized <b>403</b> - Not permitted <b>404</b> - Not found
<b>5xx</b>	Server error	<b>500</b> - Internal server error <b>502</b> - Bad gateway <b>504</b> - 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: <code>/books/abc123</code>	ex: <code>/books?search=borges&amp;checkedOut=false</code>
Located <b>directly after a slash</b> in the path. <b>Can be anywhere in the path</b>	Located only at the <b>end of a path</b> , right after a question mark <code>?</code>
Accepts <b>dynamic values</b>	Accepts <b>defined query keys with potentially dynamic values</b> .
* 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

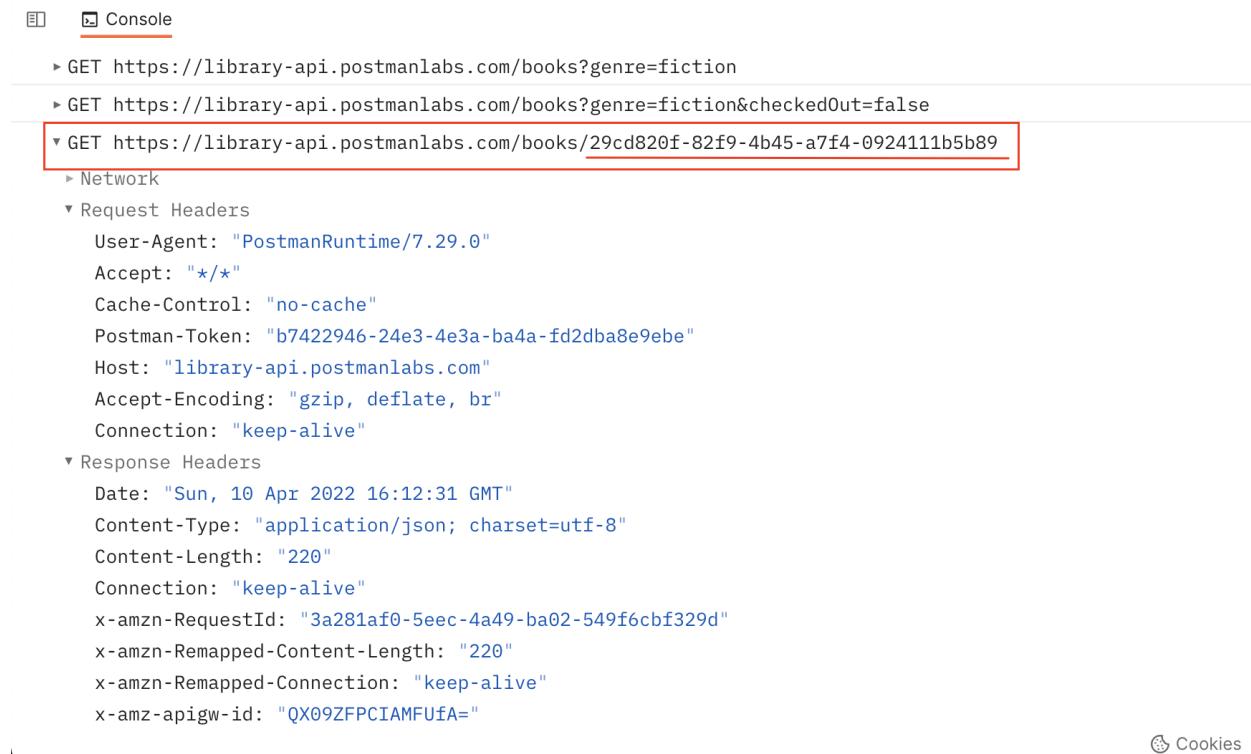
You used Postman's path variable helper in the **Params** tab of the request to add a path variable nicknamed `:id` to the request URL in a human-friendly way. Postman replaces `:id` with the value you specify for `id` in the Path Variables editor.

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.



```
Console
▶ GET https://library-api.postmanlabs.com/books?genre=fiction
▶ GET https://library-api.postmanlabs.com/books?genre=fiction&checkedOut=false
▼ GET https://library-api.postmanlabs.com/books/29cd820f-82f9-4b45-a7f4-0924111b5b89
  ▶ Network
  ▶ Request Headers
    User-Agent: "PostmanRuntime/7.29.0"
    Accept: "*/*"
    Cache-Control: "no-cache"
    Postman-Token: "b7422946-24e3-4e3a-ba4a-fd2dba8e9ebe"
    Host: "library-api.postmanlabs.com"
    Accept-Encoding: "gzip, deflate, br"
    Connection: "keep-alive"
  ▶ Response Headers
    Date: "Sun, 10 Apr 2022 16:12:31 GMT"
    Content-Type: "application/json; charset=utf-8"
    Content-Length: "220"
    Connection: "keep-alive"
    x-amzn-RequestId: "3a281af0-5eec-4a49-ba02-549f6cbf329d"
    x-amzn-Remapped-Content-Length: "220"
    x-amzn-Remapped-Connection: "keep-alive"
    x-amz-apigw-id: "QX09ZFPCIAMFUfA="
```

 Cookies

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.*

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections', 'APIs', 'Environments', 'Mock Servers', and 'Monitors'. The main area displays a collection named 'Postman Library API v2' containing three endpoints: 'get books', 'get fiction books', and 'get book by id'. The 'get book by id' endpoint is currently selected. The 'Params' tab is active, showing a 'Query Params' table with one entry: 'genre' with value 'fiction'. Below it, the 'Body' tab shows a JSON response with fields: id, title, author, and genre. The 'Console' tab at the bottom shows the raw requests and their responses, including a successful GET request for the book with id '29cd820f-82f9-4b45-a7f4-0924111b5b89'.

## 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

Postman Library API v2 / add a book

POST https://library-api.postmanlabs.com/books

Params Authorization Headers (9) Body Pre-request Script

Headers (9 hidden)

	KEY	VALUE
	Key	Value

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

Postman Library API v2 / add a book

POST https://library-api.postmanlabs.com/books

Params Authorization Headers (10) Body Pre-request Script

Headers (9 hidden)

	KEY	VALUE
<input checked="" type="checkbox"/>	api-key	postmanrulz
	Key	Value

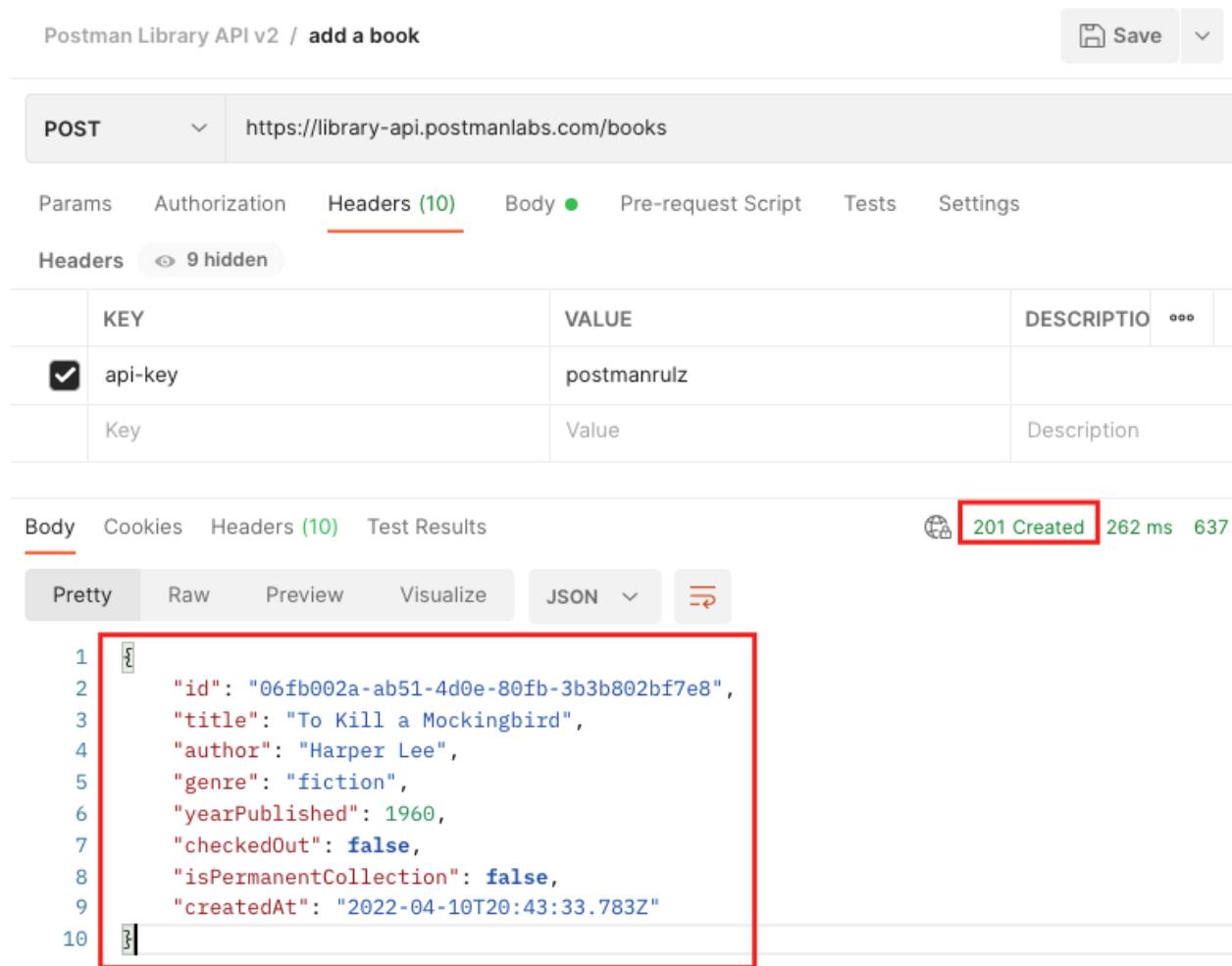
Body Cookies Headers (10) Test Results

3. Save your request

4. Send your request



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!



A screenshot of the Postman application interface showing a successful API request. The request is a POST to `https://library-api.postmanlabs.com/books`. The Headers tab is selected, showing an `api-key` header with value `postmanrulz`. The Body tab is selected, displaying a JSON response object with fields: `id`, `title`, `author`, `genre`, `yearPublished`, `checkedOut`, `isPermanentCollection`, and `createdAt`. The response status is **201 Created**.

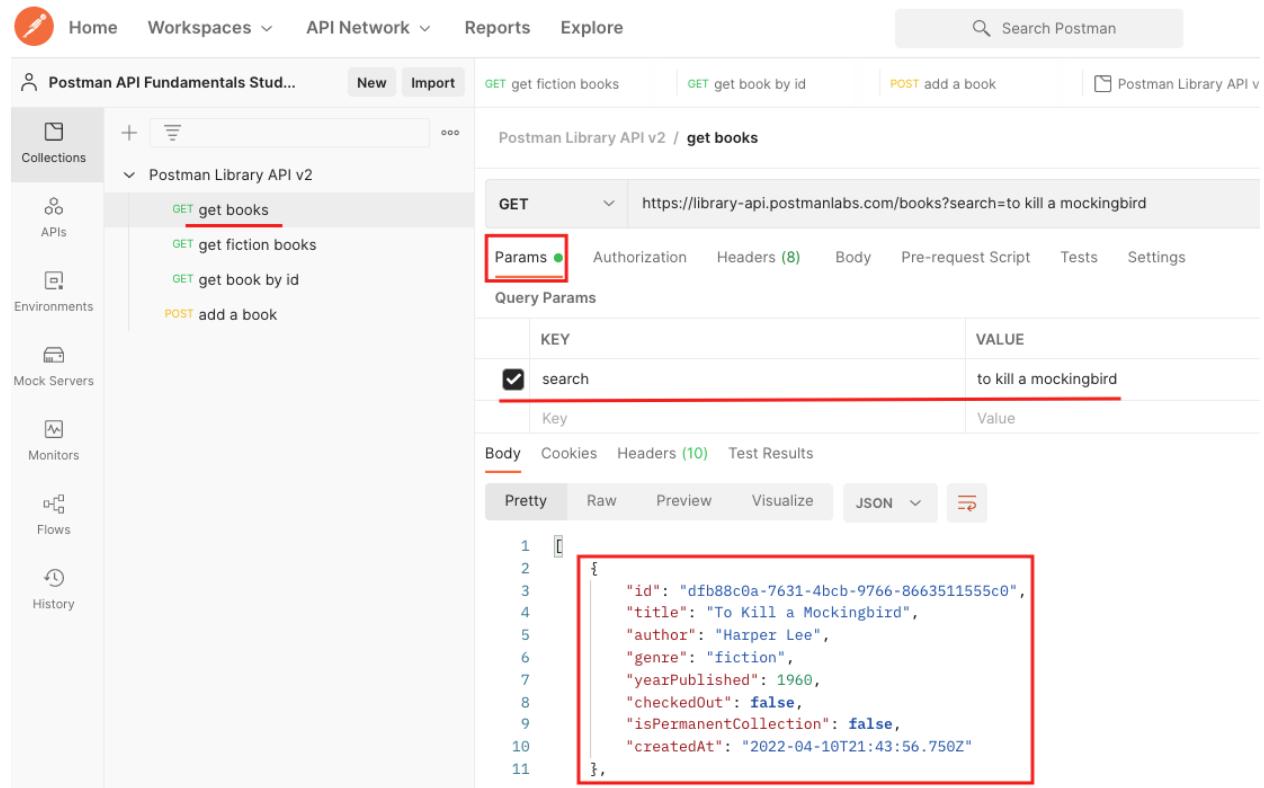
KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> api-key	postmanrulz	
Key	Value	Description

```
1  {
2      "id": "06fb002a-ab51-4d0e-80fb-3b3b802bf7e8",
3      "title": "To Kill a Mockingbird",
4      "author": "Harper Lee",
5      "genre": "fiction",
6      "yearPublished": 1960,
7      "checkedOut": false,
8      "isPermanentCollection": false,
9      "createdAt": "2022-04-10T20:43:33.783Z"
10 }
```

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!



The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Header:** Home, Workspaces, API Network, Reports, Explore, Search Postman.
- Request List:** Postman Library API v2 / `get books`.
  - Method: GET, URL: <https://library-api.postmanlabs.com/books?search=to kill a mockingbird>.
  - Params tab (highlighted with a red box):
    - Key: search, Value: to kill a mockingbird
  - Body tab (selected):
    - Pretty: JSON (highlighted with a red box)
    - Raw: JSON
    - Preview: JSON
    - Visualize: JSON
  - Response Body (JSON):

```
1  [
2   {
3     "id": "dfb88c0a-7631-4bcb-9766-8663511555c0",
4     "title": "To Kill a Mockingbird",
5     "author": "Harper Lee",
6     "genre": "fiction",
7     "yearPublished": 1960,
8     "checkedOut": false,
9     "isPermanentCollection": false,
10    "createdAt": "2022-04-10T21:43:56.750Z"
11  },
```

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

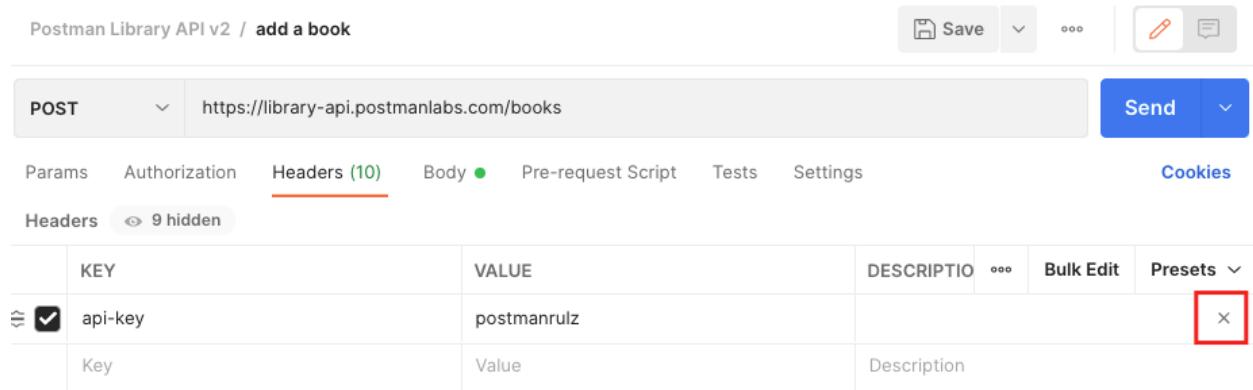
## Task: Use Postman Auth instead!

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.



The screenshot shows the Postman interface for a collection named "Postman Library API v2 / add a book". A specific POST request is selected with the URL <https://library-api.postmanlabs.com/books>. The "Headers" tab is active, displaying a table with one row. The first column is labeled "KEY" and the second is "VALUE". The row contains "api-key" in the KEY column and "postmanrulz" in the VALUE column. To the right of the VALUE column is a red-bordered "X" button used for deletion. The "Send" button is visible in the top right corner.

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
api-key	postmanrulz			X
Key	Value	Description		

## 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

The screenshot shows the Postman interface with the 'Postman Library API v2' collection selected. The left sidebar lists 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. The main area displays the collection's structure with three GET requests and one POST request. The 'Authorization' tab is highlighted with a red box. A tooltip states: 'This authorization method will be used for every request in this collection. You can override this by specifying'. Below it, another tooltip says: 'This collection does not use any authorization. [Learn more about authorization](#)'.

## 2. Select API Key as the auth Type

The screenshot shows the 'Authorization' tab selected in the 'Postman Library API v2' collection settings. A dropdown menu is open under the 'Type' field, listing 'No Auth', 'API Key' (which is highlighted with a red box), 'Bearer Token', 'Basic Auth', 'Digest Auth', and 'OAuth 1.0'. A tooltip at the bottom left says: 'This collection does not use any authorization'.

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

This authorization method will be used for every request in this collection. You can override this

Type

API Key



The authorization header will be automatically generated when you send the request.

[Learn more about authorization ↗](#)

ⓘ Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables ↗](#) X

Key

api-key

Value

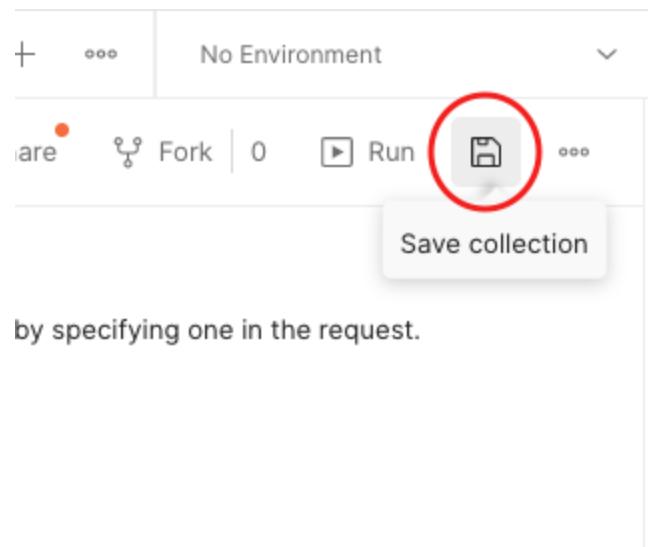
postmanrulz

Add to

Header



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



by specifying one in the request.

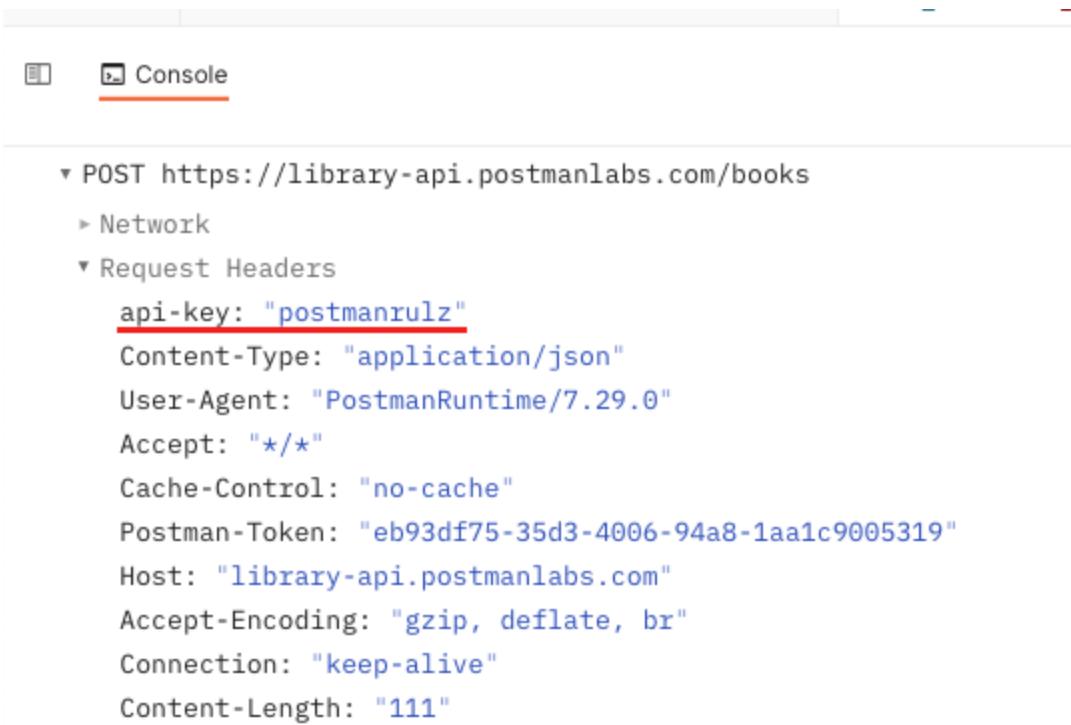
Now all requests inside this collection that use the auth method “Inherit from parent” will have this header attached, and therefore be authorized.

## 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.

The screenshot shows the Postman Library API v2 / add a book interface. At the top, there are three tabs: GET get fiction books, GET get book by id, and POST add a book. The POST add a book tab is selected. Below the tabs, the URL https://library-api.postmanlabs.com/books is displayed. The request method is POST. The Headers section shows 11 items. A red box highlights the 'Authorization' field under 'Params'. Another red box highlights the dropdown menu 'Inherit auth fr...' under 'Type'. Below these fields, a note states: 'The authorization header will be automatically generated when you send the request.' followed by a link 'Learn more about authorization'.

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!



```
▼ POST https://library-api.postmanlabs.com/books
  ▶ Network
  ▼ Request Headers
    api-key: "postmanrulz"
    Content-Type: "application/json"
    User-Agent: "PostmanRuntime/7.29.0"
    Accept: "*/*"
    Cache-Control: "no-cache"
    Postman-Token: "eb93df75-35d3-4006-94a8-1aa1c9005319"
    Host: "library-api.postmanlabs.com"
    Accept-Encoding: "gzip, deflate, br"
    Connection: "keep-alive"
    Content-Length: "111"
```

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 can be 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}}`

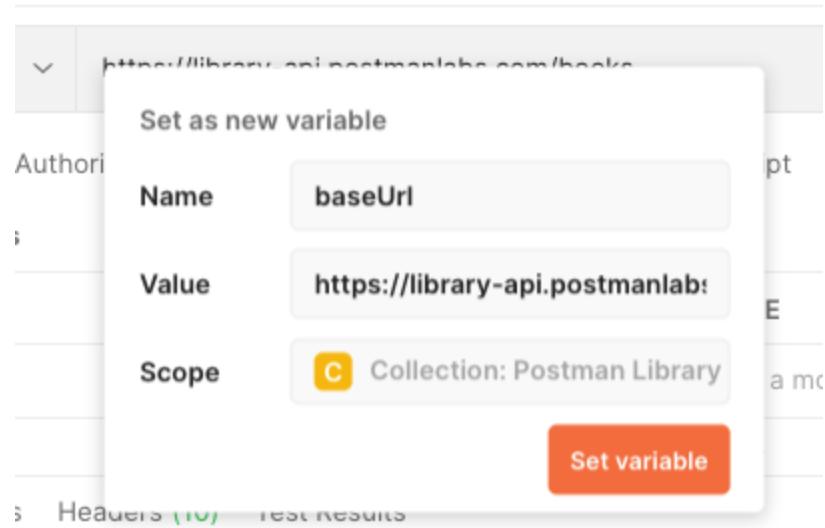
1. Go to the first "get books" request in your collection.
2. With your cursor, select the entire base URL of the API (`https://library-api.postmanlabs.com`). Do not include the slash `/` after `.com`. Click **Set as variable** to save the base URL to a variable.

The screenshot shows the Postman Library API v2 / get books interface. At the top, there is a red box around the "Set as variable" button. Below it, the method is set to "GET" and the URL is "https://library-api.postmanlabs.com/books". The "Params" tab is selected, showing a table with one row. The first column is labeled "KEY" and the second column contains "search" and "Key". Other tabs like "Authorization", "Headers (8)", "Body", and "Pre-request" are visible but not selected.

### 3. Click Set as a new variable

The screenshot shows the same Postman interface after clicking the "Set as variable" button. A red box highlights the "Set as a new variable" button in the "Query Params" table. The table has two rows: one for "search" and one for "Key". The "Params" tab is still selected. The "Pretty" tab at the bottom is active, displaying a JSON response with three lines of code: "1 [", "2 {", and "3 | "id": "dfb88c0a-7631-4bcb-9766-8663511555c0".

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>

The screenshot shows a 'Postman Library API v2 / get books' request. The method is 'GET' and the URL is  `{{baseUrl}}/books`. In the 'Params' section, there is a 'search' parameter. A tooltip for the  `{{baseUrl}}`  placeholder is displayed, showing 'CURRENT' value: <https://library-api.postmanlabs.com>. This tooltip is highlighted with a red box. Other parts of the tooltip show 'INITIAL' value: <https://library-api.postmanlabs.com> and 'SCOPE' Collection.

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.

The screenshot shows the Postman interface with the 'Postman API Fundamentals Stud...' collection selected. The left sidebar shows 'Collections' with 'Postman Library API v2' expanded, revealing requests like 'get books', 'get fiction books', 'get book by id', and 'add a book'. The top navigation bar includes 'Home', 'Workspaces', 'API Network', 'Reports', 'Explore', and a search bar. The main area displays the 'Postman Library API v2' collection details, including tabs for 'Authorization', 'Pre-request Script', 'Tests', and 'Variables'. The 'Variables' tab is highlighted with a red box. A table lists variables with columns for 'VARIABLE', 'INITIAL VALUE', 'CURRENT VALUE', and actions. One variable, 'baseUrl', is shown with its initial value as 'https://library-api.postm...' and current value as 'https://library-api.postmanlabs.com'. A tooltip at the bottom left provides information about using variables to reuse values and protect sensitive data.

VARIABLE	INITIAL VALUE	CURRENT VALUE	...	Persist All	Reset All
<input checked="" type="checkbox"/> baseUrl	https://library-api.postm...	https://library-api.postmanlabs.com			
Add a new variable					

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")`

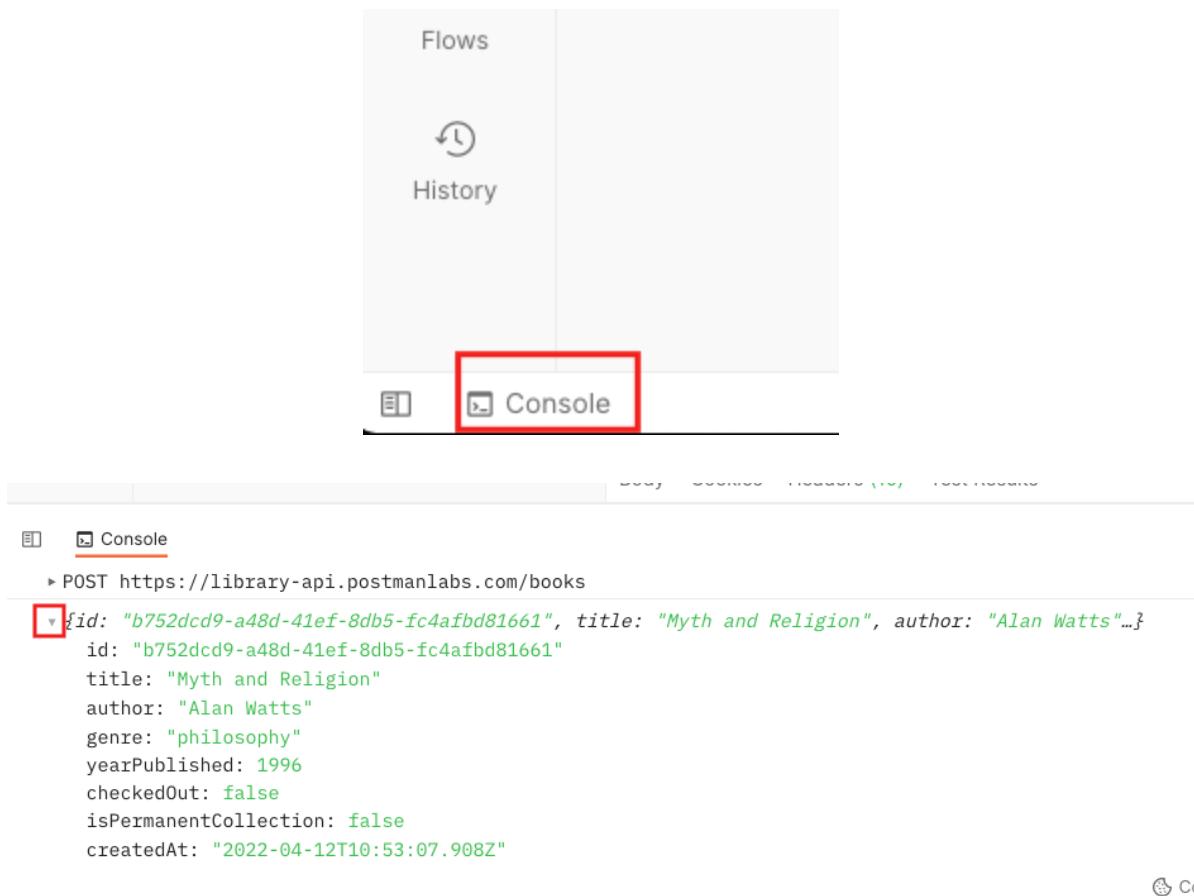
## Add a script to your request

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 request. Inside 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

The screenshot shows the Postman application interface. On the left, there's a sidebar with various sections like Home, Workspaces, API Network, Reports, and Explore. Below these are collections, APIs, environments, mock servers, monitors, flows, and history. The main area shows a collection named "Postman API Fundamentals Stud...". Under this, there's a "Postman Library API v2" folder containing three GET requests: "get books", "get fiction books", and "get book by id". A new POST request titled "add a book" is selected. The request details show a POST method and the URL "https://library-api.postmanlabs.com/books". The "Tests" tab is active, indicated by a red circle. The test script contains the line "1 console.log(pm.response.json())". Other tabs include Params, Authorization, Headers (11), Body, Pre-request Script, and Settings. To the right of the tests tab is a sidebar with snippets for setting environment variables. At the bottom, status information shows "Status: 201 Created Time: 254 ms Size: 637 B".

## Postman Library API v2 / add a book

This is a zoomed-in view of the "Tests" tab from the previous screenshot. The tab is highlighted with a red border. The test script editor contains the line "1 console.log(pm.response.json())". The rest of the interface elements like POST method, URL, and other tabs are visible but not highlighted.



```
POST https://library-api.postmanlabs.com/books
{
  "id": "b752dcd9-a48d-41ef-8db5-fc4afbd81661",
  "title": "Myth and Religion",
  "author": "Alan Watts",
  "id": "b752dcd9-a48d-41ef-8db5-fc4afbd81661",
  "title": "Myth and Religion",
  "author": "Alan Watts",
  "genre": "philosophy",
  "yearPublished": 1996,
  "checkedOut": false,
  "isPermanentCollection": false,
  "createdAt": "2022-04-12T10:53:07.908Z"
}
```

## 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)
```

The screenshot shows the Postman application interface. On the left, there's a sidebar titled "Postman Library API v2" containing several API endpoints: "GET get books", "GET get fiction books", "GET get book by id", and "POST add a book". The "POST add a book" endpoint is currently selected. The main panel displays a POST request to "https://library-api.postmanlabs.com/books". Below the method and URL, there are tabs for "Params", "Auth", "Headers (11)", "Body", "Pre-req.", "Tests", and "Settings". The "Tests" tab is active and highlighted with a red border. Inside the "Tests" section, there are two lines of JavaScript code:

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

3. **Save** your request
4. **Send** your request. When the **201** response comes back from the API with your 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

KEY	VALUE	DESCRIPTION
id	<code>{{{id}}}</code>	Description

**C id**

INITIAL

CURRENT f04ed19a-ff31-4777-805d-1d4cc5878d00

SCOPE Collection

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

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

Postman Library API v2 / **checkout a book**

Saving...

PATCH

`{{baseUrl}}/books/:id`

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1 {  
2   "checkedOut": true  
3 }
```

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.

## Postman Library API v2 / checkout a book

PATCH ▼ `{{baseUrl}}/books/:id`

Params • Authorization Headers (10) Body •

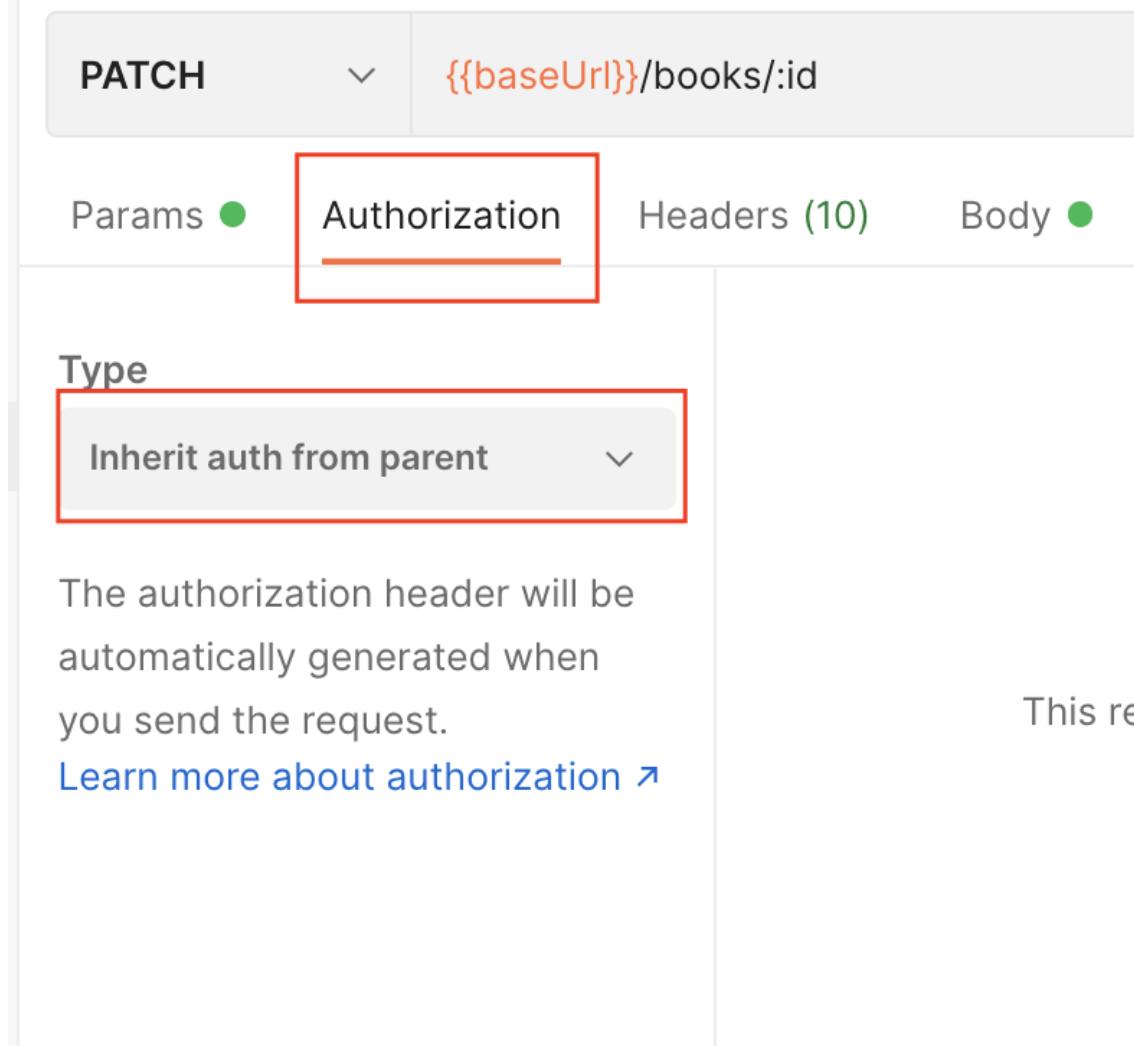
Type ▼

Inherit auth from parent

The authorization header will be automatically generated when you send the request.

This re

Learn more about authorization ↗



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

Postman Library API v2 / checkout a book

PATCH {{baseUrl}}/books/:id

Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

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

1 ... "checkedOut": true  
2  
3

Body Cookies Headers (10) Test Results 200 OK 246 ms 630 B Save Response

Pretty Raw Preview Visualize JSON

1 "id": "f04ed19a-ff31-4777-805d-1d4cc5878d00",  
2 "title": "Myth and Religion",  
3 "author": "Alan Watts",  
4 "genre": "philosophy",  
5 "yearPublished": 1996,  
6 "checkedOut": true,  
7 "isPermanentCollection": false,  
8 "createdAt": "2022-04-12T13:26:47.493Z"  
9  
10

# 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!

The screenshot shows the Postman application interface. On the left, there's a sidebar with various sections like Home, Workspaces, API Network, Reports, and Explore. Below these are collections, APIs, environments, mock servers, monitors, flows, and history. The main area displays a collection named 'Postman API Fundamentals Stud...'. Under this collection, there are several requests: 'get books', 'get fiction books', 'get book by id' (which is selected and highlighted with a red box), 'POST add a book', and 'PATCH checkout a book'. The 'get book by id' request is currently being viewed. It has a 'GET' method, a URL 'https://library-api.postmanlabs.com/books/:id', and a 'Send' button which is also highlighted with a red box. Below the method and URL, there are tabs for Params, Authorization, Headers (8), Body, Pre-request Script, Tests, and Settings. The 'Params' tab is active, showing a table with a single row: 'Key' (Value) and 'Value' (Description). The 'Body' tab is also active, showing a JSON response with the following content:

```

1
2   "id": "f04ed19a-ff31-4777-805d-1d4cc5878d00",
3   "title": "Myth and Religion",
4   "author": "Alan Watts",
5   "genre": "philosophy",
6   "yearPublished": 1996,
7   "checkedOut": true,
8   "isPermanentCollection": false,
9   "createdAt": "2022-04-12T13:26:47.493Z"
10

```

At the bottom of the body panel, there are tabs for Pretty, Raw, Preview, Visualize, and JSON, with 'JSON' currently selected. To the right of the body panel, there are status details: Status: 200 OK, Time: 48 ms, Size: 630 B, and a 'Save Response' button.

## Task: Delete your book

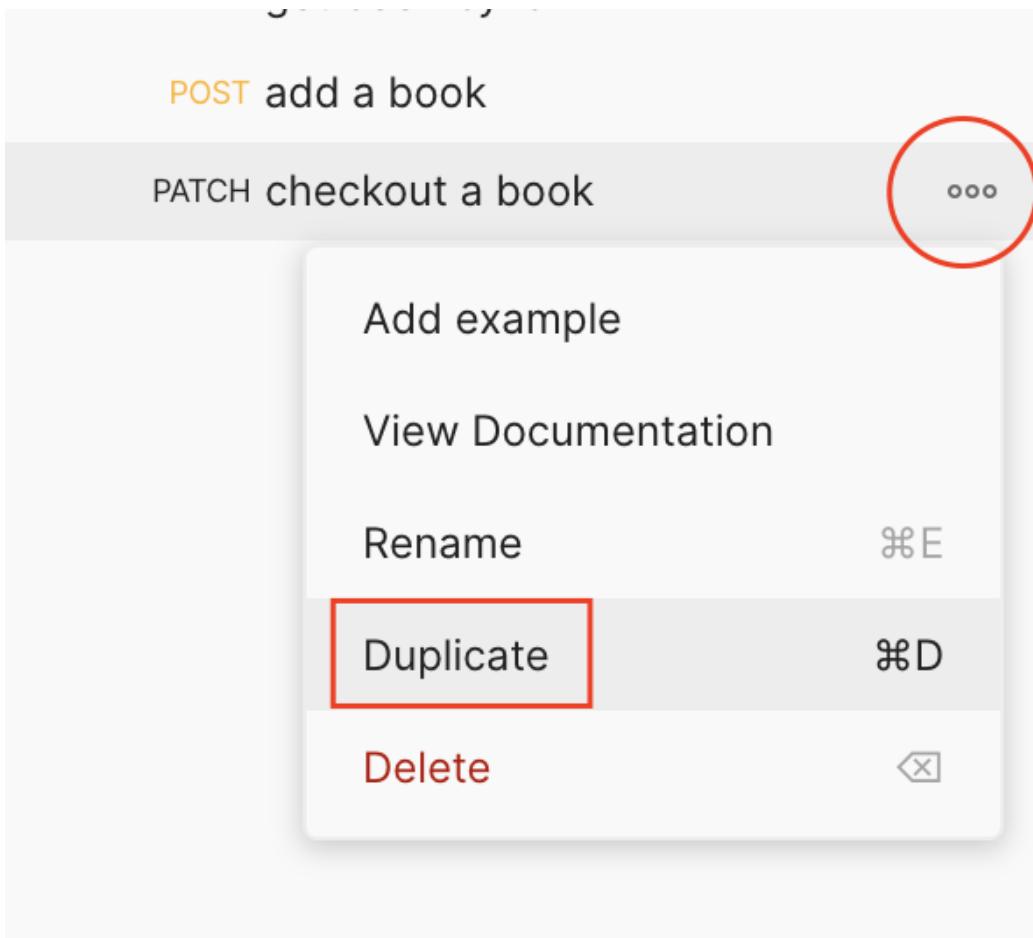
Oops! The person that checked out your book accidentally lost it... you will need to delete it from the library database.

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

## 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" request, 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:

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for 'New' and 'Import'. Below the tabs, a list of collections is shown: 'Postman Library API v2' (selected), 'Postman', and 'No Environment'. Under 'Postman Library API v2', there are several requests listed: 'get books', 'get fiction books', 'get book by id', 'add a book', 'checkout a book', and 'delete a book'. The 'delete a book' request is selected. The main workspace shows a 'DELETE' method with the URL {{baseUrl}}/books/:id. The 'Params' tab is selected, showing a 'Key' column with 'Key' and a 'Value' column with 'Value'. The 'Path Variables' tab is also visible, showing a 'KEY' column with 'id' and a 'VALUE' column with '{{id}}'. At the bottom right of the workspace, there is a red box around the 'Send' button.

## 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!*

The screenshot shows the Postman application interface. On the left, the sidebar lists various environments and collections. The main workspace displays a DELETE request to 'delete a book' with the URL `((baseUrl))/books/:id`. The 'Params' tab is selected, showing a single parameter 'id' with the value `{:id}`. Below the URL, the 'Path Variables' section also shows the variable `id` with the value `{:id}`. The 'Body' tab is active, showing a status message '204 No Content'. Other tabs include 'Cookies', 'Headers (9)', and 'Test Results'. At the bottom, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', 'JSON', and a 'Save Response' button.

## 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!

The screenshot shows the Postman interface for a DELETE request to delete a book. The URL is `{{baseUrl}}/books/:id`. The response status is 404 Not Found, with the message "Book with id 'f04ed19a-ff31-4777-805d-1d4cc5878d00' not found".

**DELETE** `{{baseUrl}}/books/:id` **Send**

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Path Variables

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	id	<code>{{id}}</code>	Description		

Body **404 Not Found** 218 ms 486 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Book with id 'f04ed19a-ff31-4777-805d-1d4cc5878d00' not found"  
3 }
```

[Previous - Task: Checkout your book](#) [Postman's codegen feature](#) [Next](#)

## Postman's codegen feature

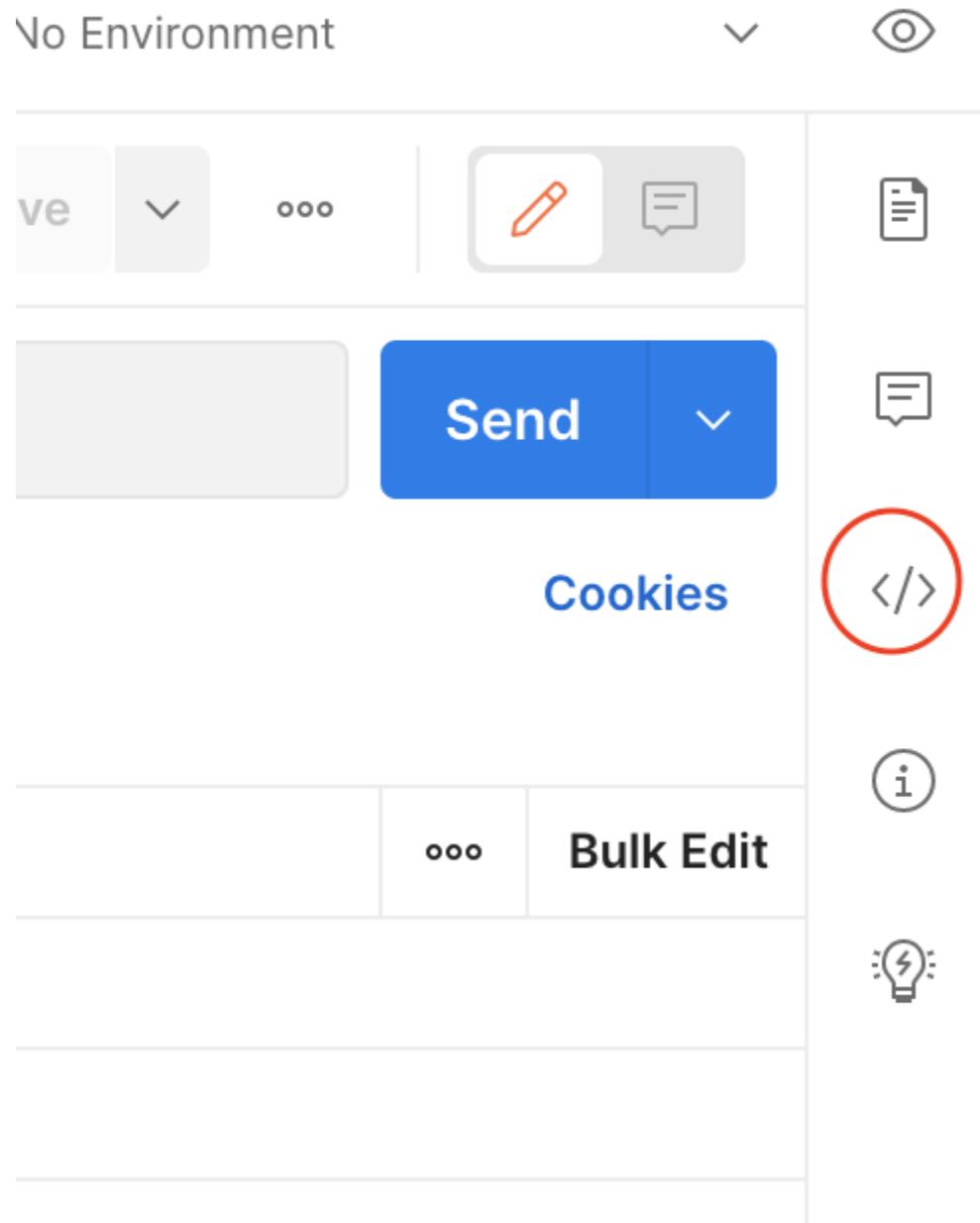
Postman can help you integrate API calls into your applications by [generating code snippets](#) in various coding languages.

## 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 + ooo 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 - NSURLConnection
- OCaml - Cohttp

```
curl GET 'https://manlabs.com/books' \stmanrulz'
```

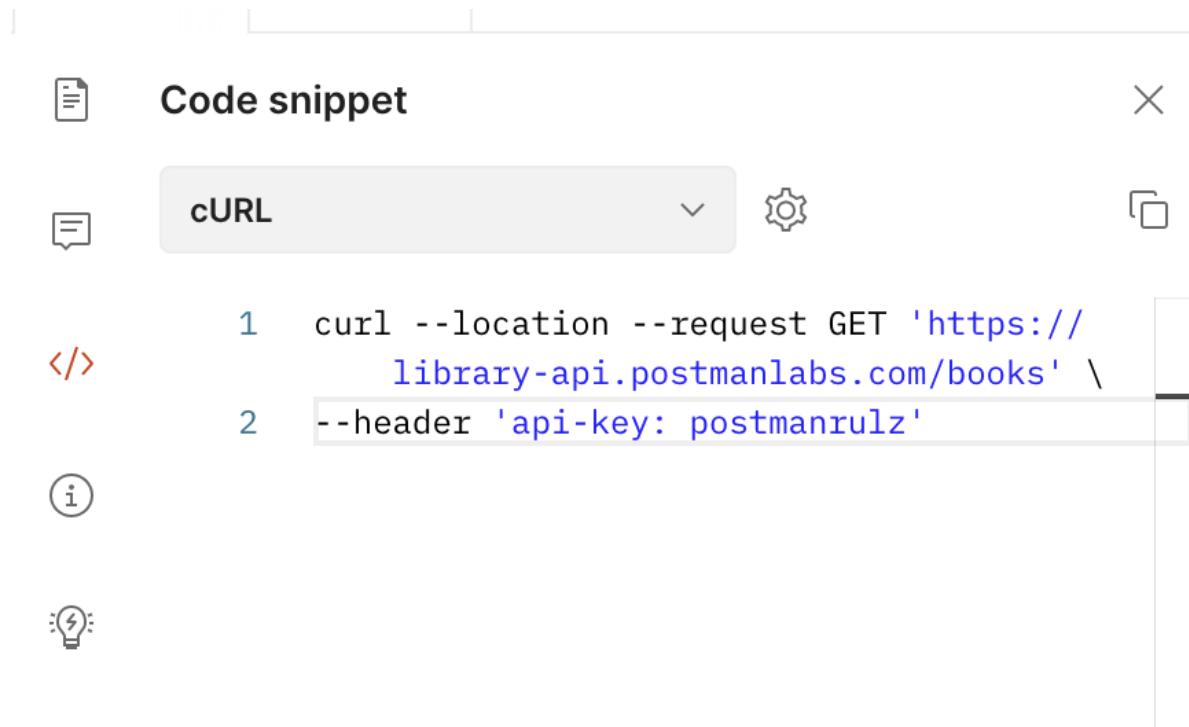
For example, we can generate the

```
GET /books
```

request in

```
CURL
```

syntax:



The screenshot shows the Postman Code snippet editor. The title bar says "Code snippet". The dropdown menu is set to "CURL". The code area contains two lines of a CURL command:

```
curl --location --request GET 'https://library-api.postmanlabs.com/books' \
      --header 'api-key: postmanrulz'
```

Below the code are two icons: an info icon (i) and a lightbulb icon.

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": "b752dc9-a48d-41ef-8db5-fc4afbd81661", "title": "Myth and Religion", "author": "Alan Watts", "genre": "philosophy", "yearPublished": 1996, "checkedOut": false, "isPermanentCollection": false, "createdAt": "2022-04-12T10:53:07.908Z"}, {"id": "7ad72643-67af-4e0d-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", "yearPublished": 1960, "checkedOut": false, "isPermanentCollection": false, "createdAt": "2022-04-10T21:43:56.750Z"}, {"id": "06fb002a-ab51-4d0e-80fb-3b3b802bf7e8", "title": "To Kill a Mockingbird", "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, "isPermanentCollection": false, "createdAt": "2022-03-31T15:47:53.093Z"}, {"id": "85a5424d-535d-48a9-a163-8627b84662d7", "title": "Continuous API Management: Making the Right Decisions in an Evolving Landscape", "author": "Mehdi Medjaoui, Erik Wilde, Ronnie Mitra, Mike Amundsen", "genre": "computers", "yearPublished": 2018, "checkedOut": false, "isPermanentCollection": true, "createdAt": "2022-03-30T00:54:52.606Z"}, {"id": "03daae04-c778-45fb-9acb-4c6d19067a9c", "title": "Go Design Patterns", "author": "Mario Castro Contreras", "genre": "computers", "yearPublished": 2017, "checkedOut": false, "isPermanentCollection": true, "createdAt": "2022-03-30T00:54:52.606Z"}, {"id": "539f89ed-df63-469c-94ce-157e75366dec", "title": "Domain-Driven Design: Tackling Complexity in the Heart of Software", "author": "Eric Evans", "genre": "computers", "yearPublished": 2003, "checkedOut": false, "isPermanentCollection": true, "createdAt": "2022-03-30T00:54:52.606Z"}, {"id": "29cd820f-82f9-4b45-a7f4-0924111b5b89", "title": "Ficciones", "author": "Jorge Luis Borges", "genre": "fiction", "yearPublished": 1944, "checkedOut": false, "isPermanentCollection": true, "createdAt": "2022-03-30T00:54:52.606Z"}, {"id": "14b533eb-5ef3-422a-b037-7b6161ee7020", "title": "The Art of War", "author": "Sun Tzu", "genre": "military", "yearPublished": 490 BC, "checkedOut": false, "isPermanentCollection": true, "createdAt": "2022-03-30T00:54:52.606Z"}]
```

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