Automated Testing With Postman

http://5d77c6d31e31aa00149a3635.mockapi.io/api/abc/

Jenelle C - W2019

Contents

Before testing APIs, you need:

1. An API to test

● OPTION 1: Make your own API	https://www.mockapi.io
● OPTION 2: Use a prebuilt-api	https://github.com/toddmotto/public-apis

2. Software for API testing

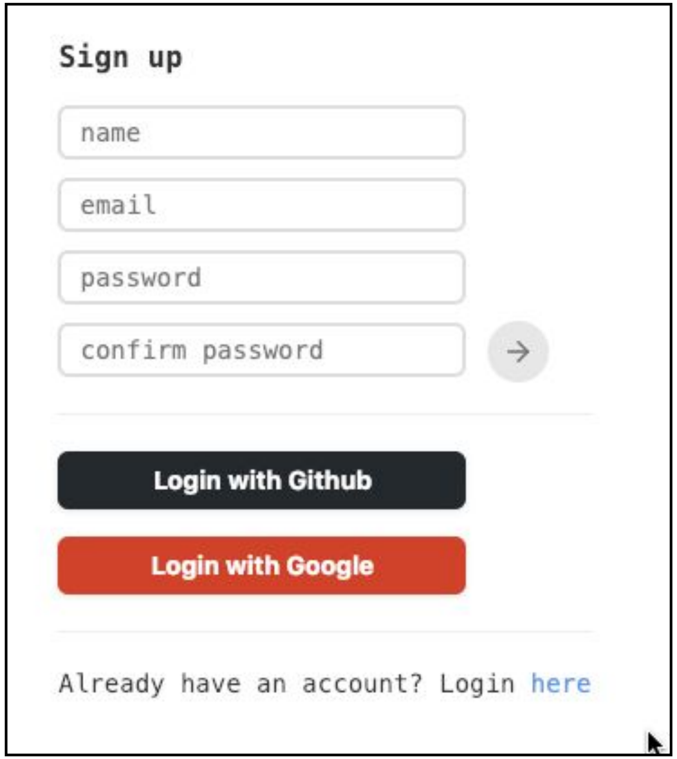
● OPTION 1: Postman	https://www.getpostman.com/
● OPTION 2: SoapUI	https://www.soapui.org/

Creating a Custom API with MockApi

How to create your own API

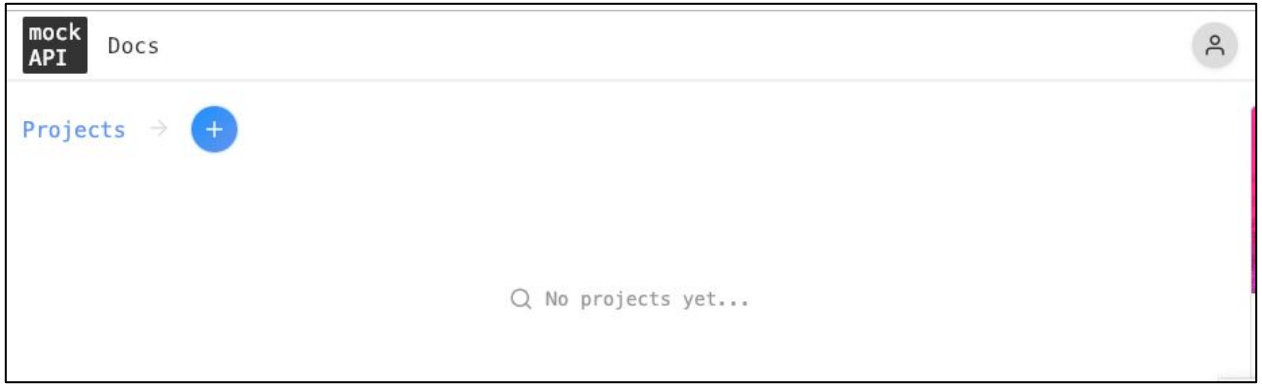
Signup for an account here:

https://www.mockapi.io/signup



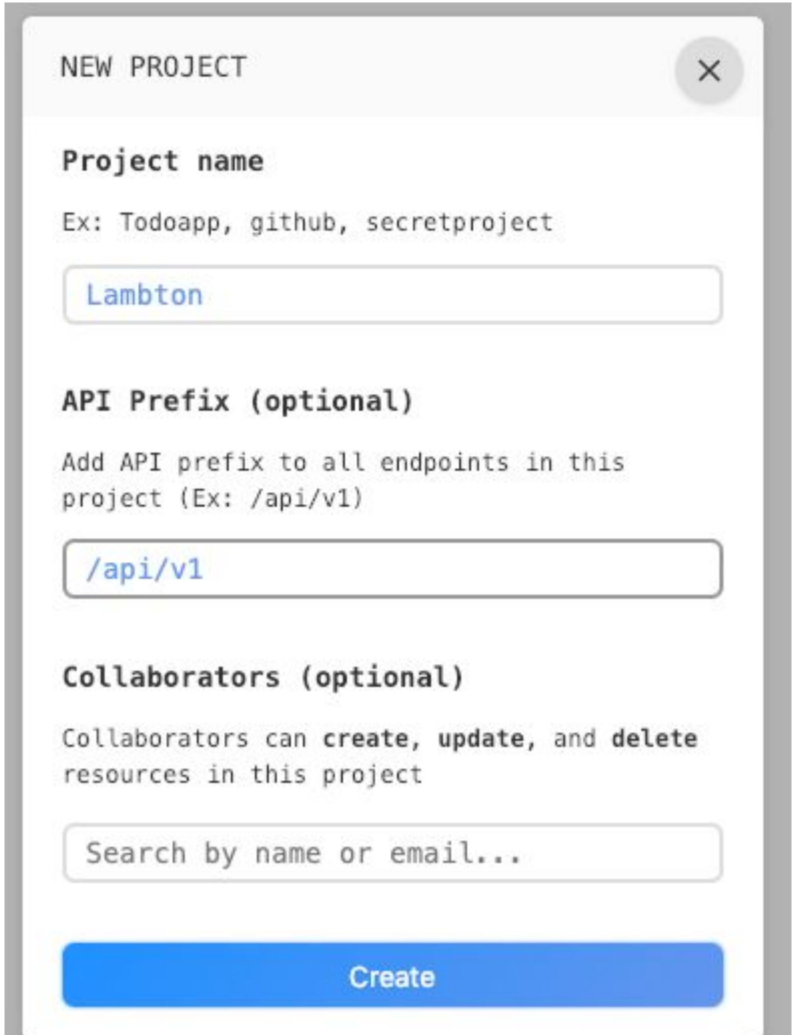
Already have an account? Login here

Click projects:



No projects yet...

Give project a name


NEW PROJECT

Project name

Ex: Todoapp, github, secretproject

Lambton

API Prefix (optional)

Add API prefix to all endpoints in this project (Ex: /api/v1)

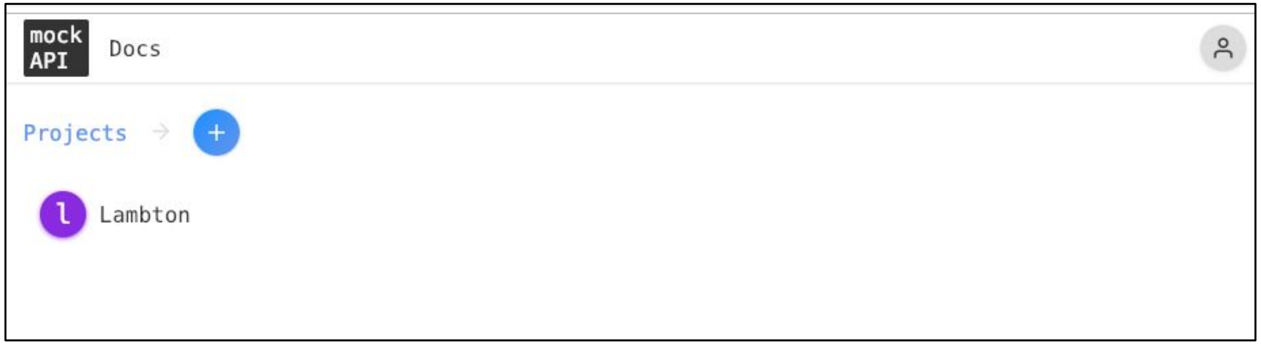
/api/v1

Collaborators (optional)

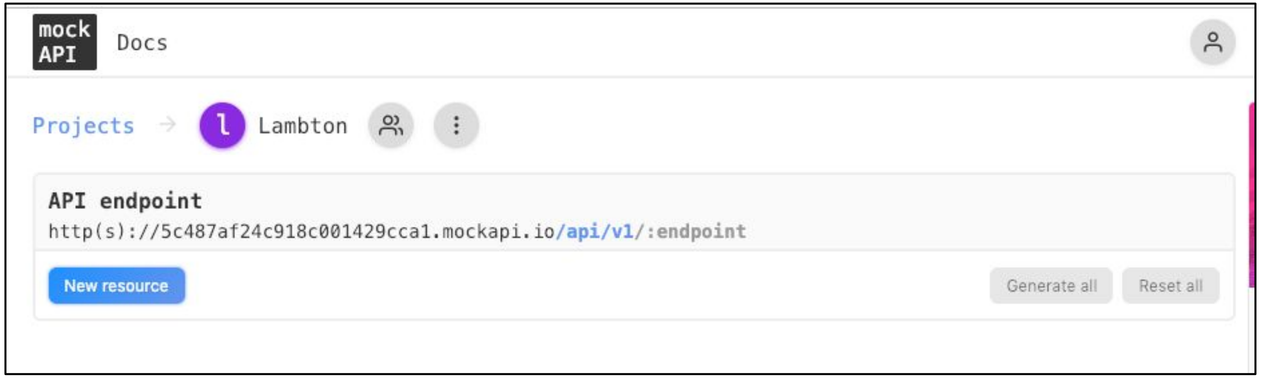
Collaborators can **create**, **update**, and **delete** resources in this project

Search by name or email...

Click on project

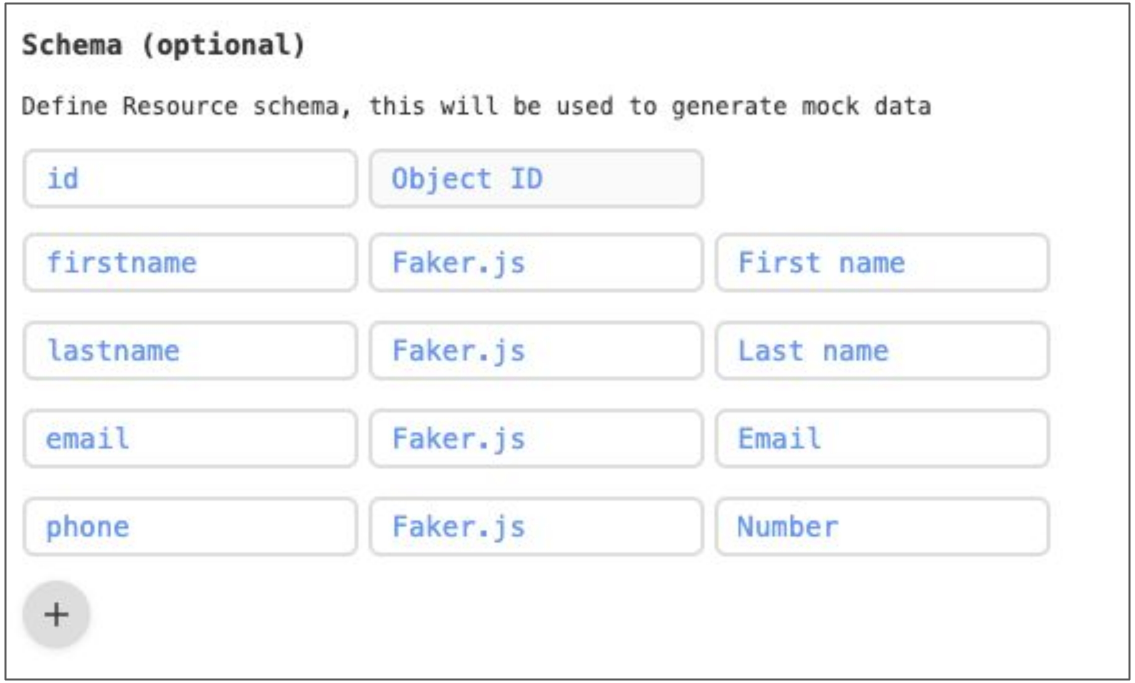


Click NEW Resource



API endpoint

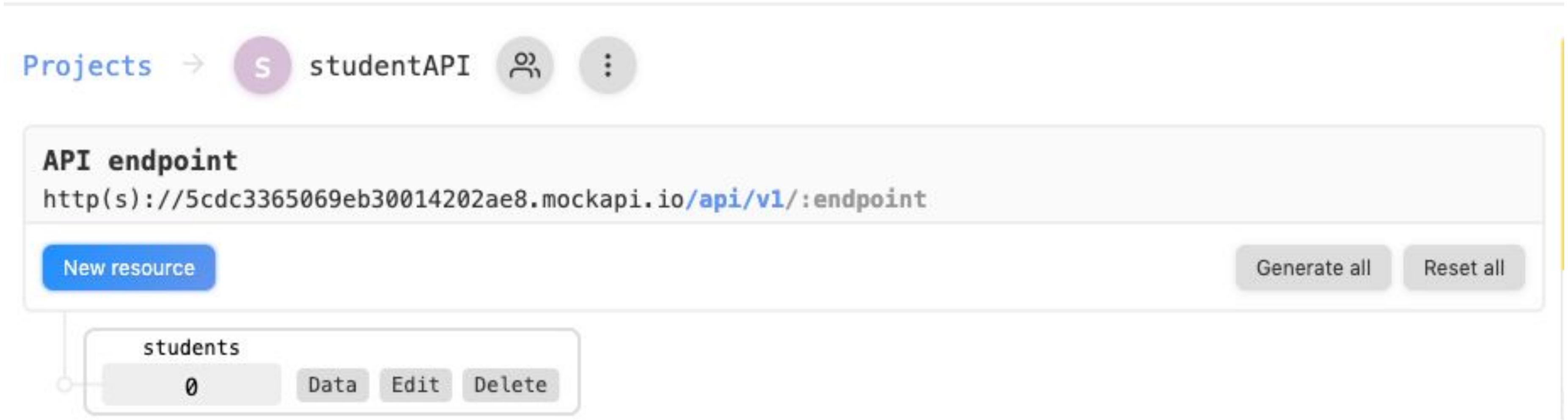
http(s)://5c487af24c918c001429cca1.mockapi.io/api/v1/:endpoint



Schema (optional)

Define Resource schema, this will be used to generate mock data

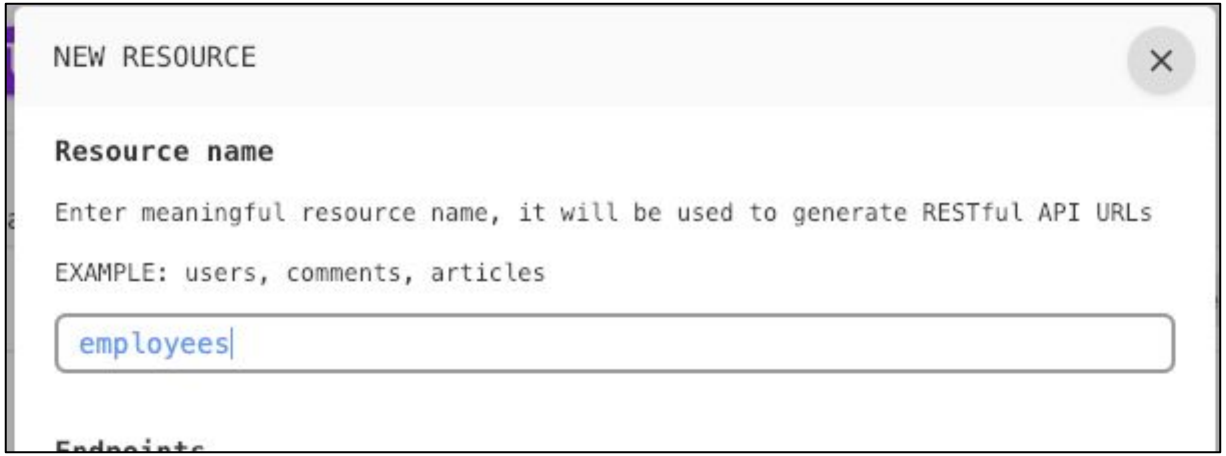
id	Object ID	
firstname	Faker.js	First name
lastname	Faker.js	Last name
email	Faker.js	Email
phone	Faker.js	Number



API endpoint

```
http(s)://5cdc3365069eb30014202ae8.mockapi.io/api/v1/:endpoint
```

Give resource a name


NEW RESOURCE

Resource name

Enter meaningful resource name, it will be used to generate RESTful API URLs

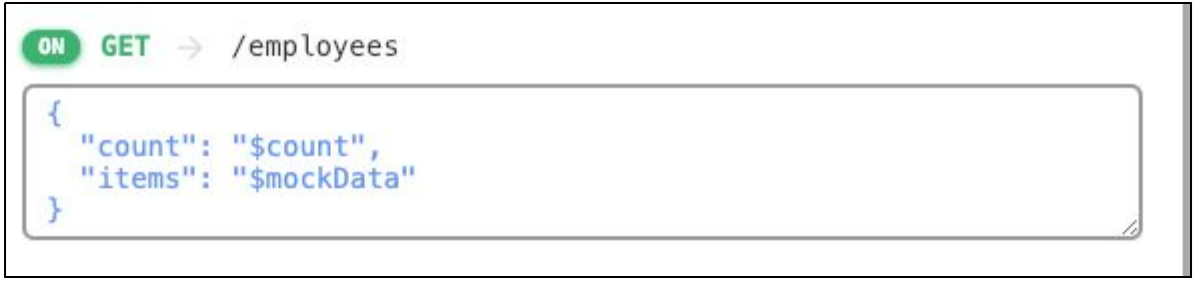
EXAMPLE: users, comments, articles

employees

For each request, type in the response format

GET:

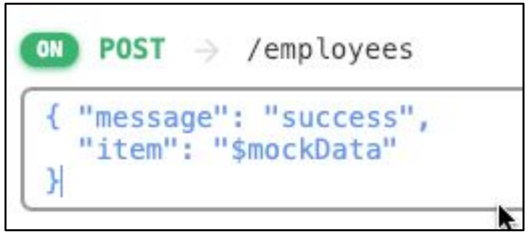
```
{ "count": "$count",
  "items": "$mockData"
}
```



```
{
  "count": "$count",
  "items": "$mockData"
}
```

POST

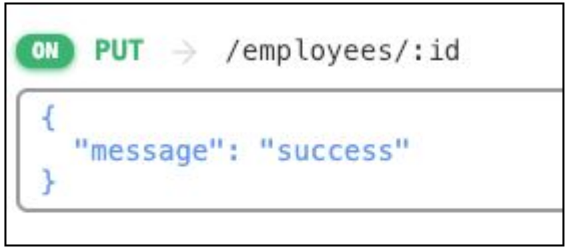
```
{
  "message": "success",
  "item": "$mockData"
}
```



```
{ "message": "success",
  "item": "$mockData"
}
```

PUT

```
{
  "message": "success"
}
```

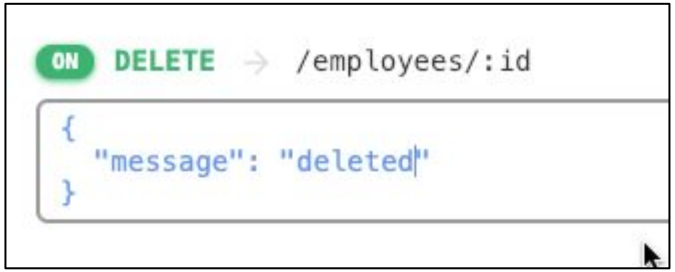


ON PUT → /employees/:id

```
{
  "message": "success"
}
```

DELETE

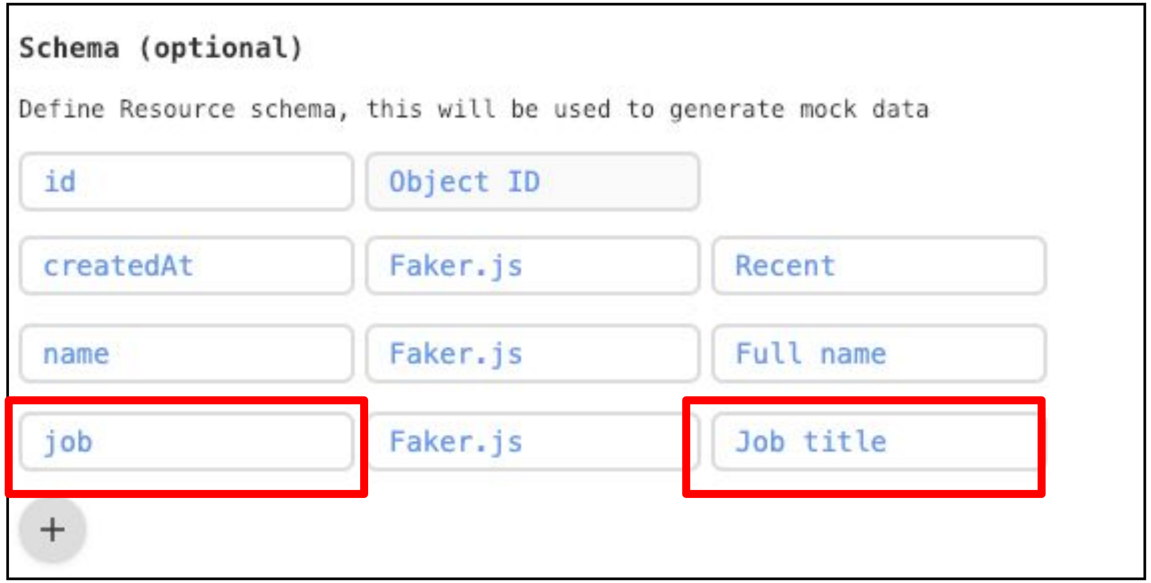
```
{
  "message": "deleted"
}
```



ON DELETE → /employees/:id

```
{
  "message": "deleted"
}
```

Add the schema



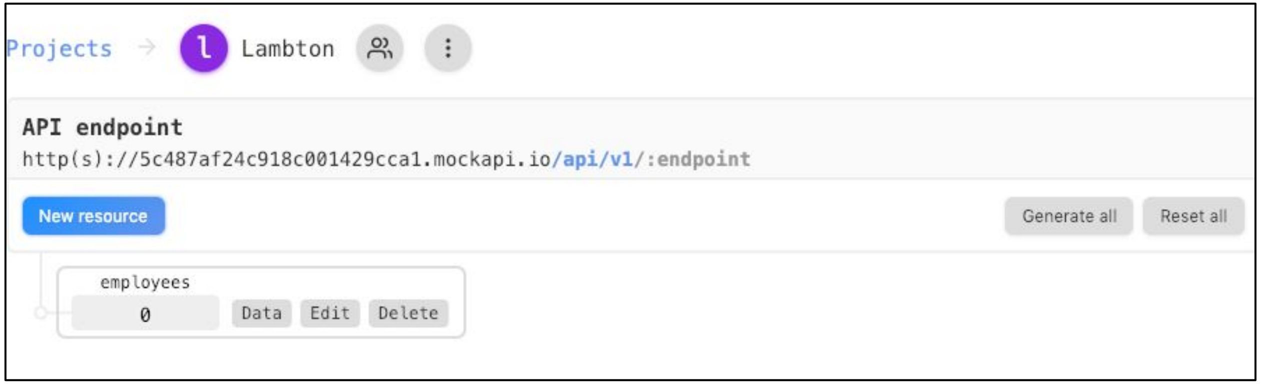
Schema (optional)

Define Resource schema, this will be used to generate mock data

id	Object ID	
createdAt	Faker.js	Recent
name	Faker.js	Full name
job	Faker.js	Job title

Press CREATE.

Expected result:

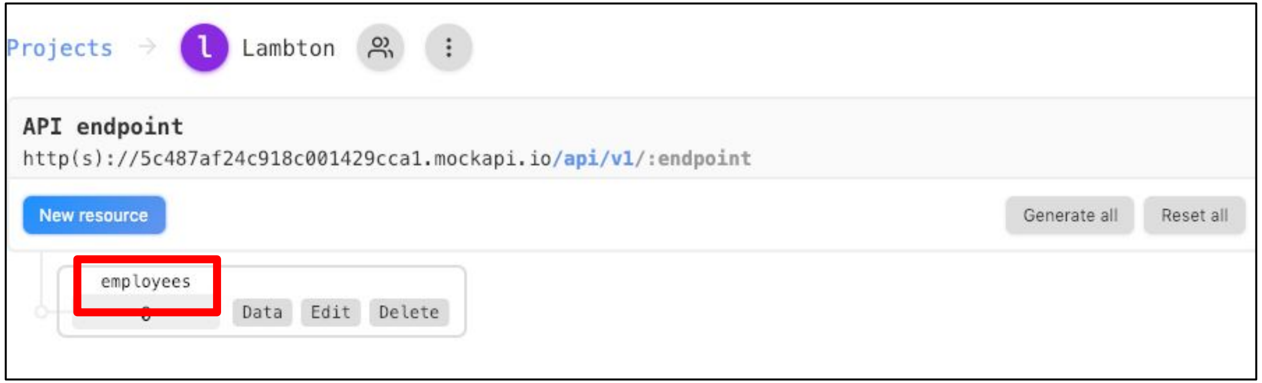


API endpoint

http(s)://5c487af24c918c001429cca1.mockapi.io/api/v1/:endpoint

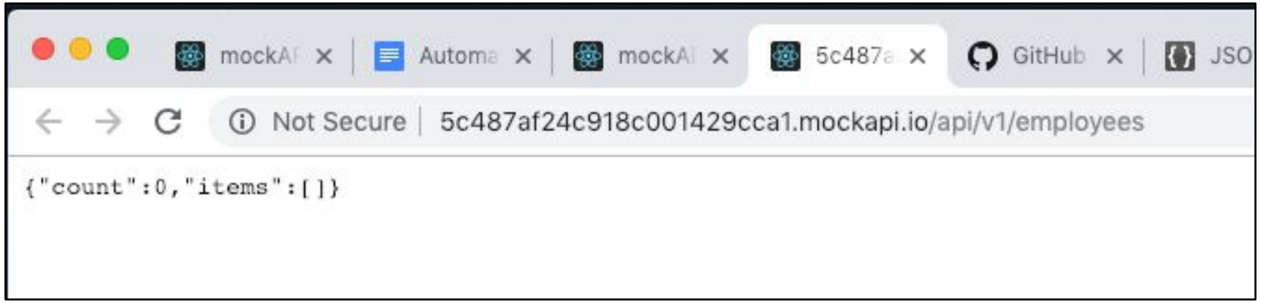
Test your api

Click on **employees**



http(s)://5c487af24c918c001429cca1.mockapi.io/api/v1/:endpoint

Expected result:

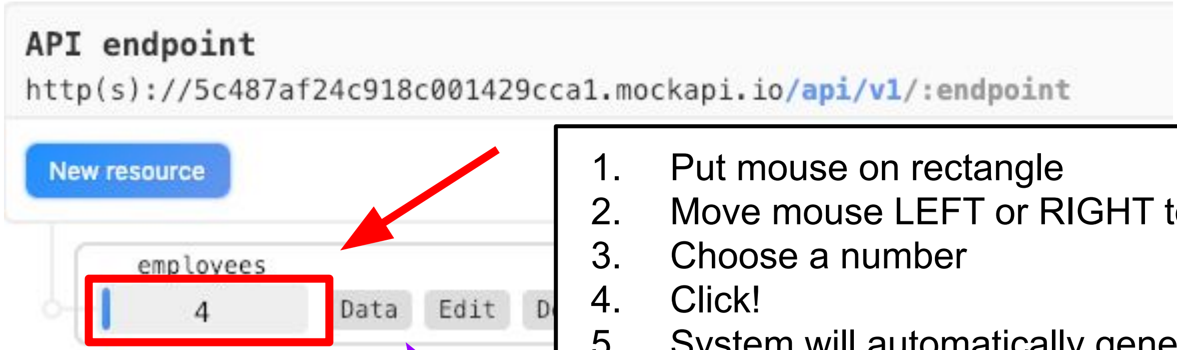


5c487af24c918c001429cca1.mockapi.io/api/v1/employees

```
{"count":0,"items":[]}
```

Add some data to your API

Click on the rectangle, choose a number


API endpoint

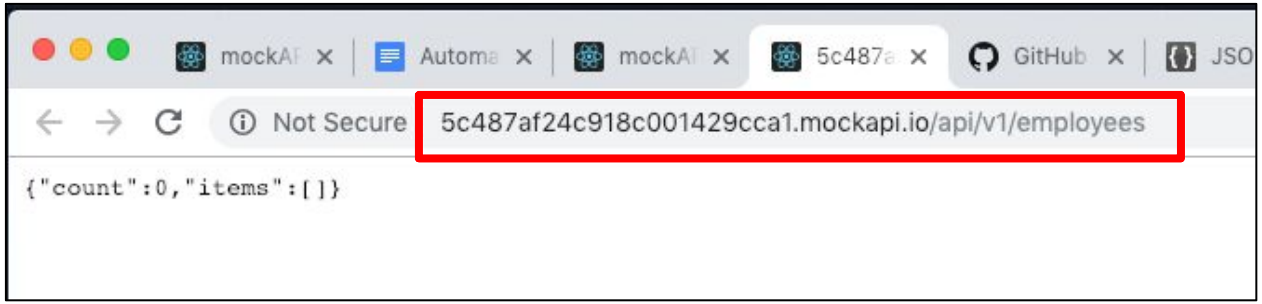
http(s)://5c487af24c918c001429cca1.mockapi.io/api/v1/:endpoint

1. Put mouse on rectangle
2. Move mouse LEFT or RIGHT to see a number
3. Choose a number
4. Click!
5. System will automatically generate that number of data
6. In this example, system has created 4 items

7. Click DATA button to see the data

Done!

Access your API by visiting here:



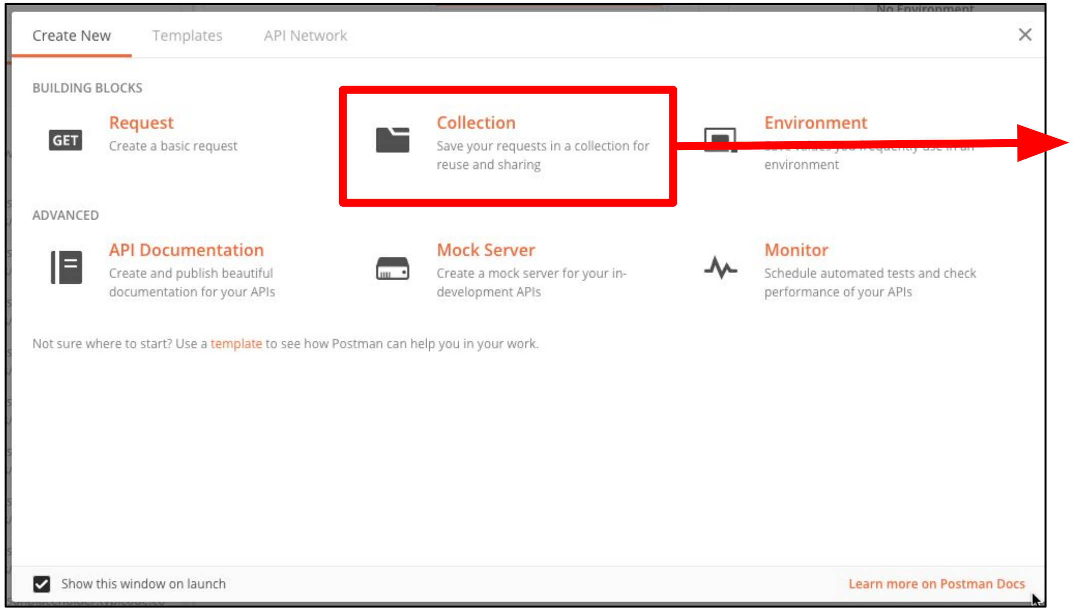
5c487af24c918c001429cca1.mockapi.io/api/v1/employees

```
{"count":0,"items":[]}
```

To test the endpoint, use POSTMAN to send GET/POST/PUT/DELETE requests to the URL .   How? See next slides

Use POSTMAN to test API

Start POSTMAN, click Collections



Create New Templates API Network

BUILDING BLOCKS

Request
Create a basic request

Collection
Save your requests in a collection for reuse and sharing

Environment
environment

ADVANCED

API Documentation
Create and publish beautiful documentation for your APIs

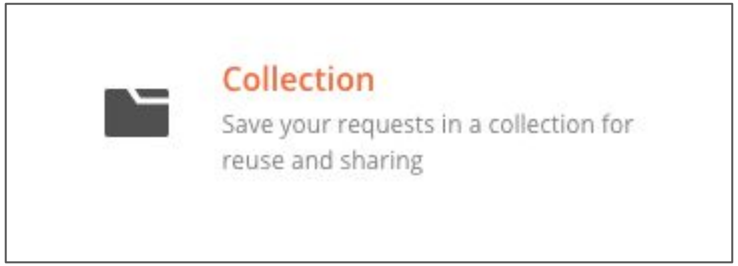
Mock Server
Create a mock server for your in-development APIs

Monitor
Schedule automated tests and check performance of your APIs

Not sure where to start? Use a template to see how Postman can help you in your work.

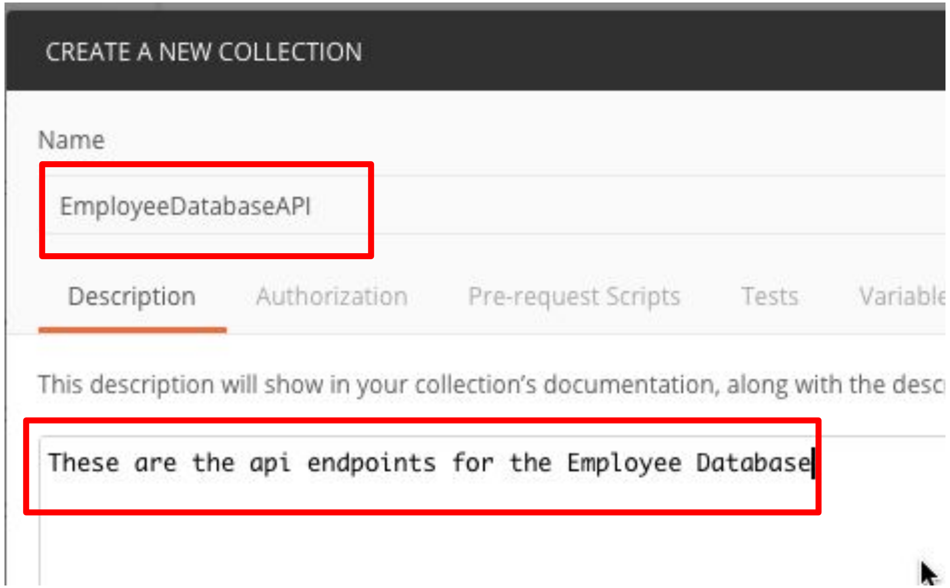
Show this window on launch

Learn more on Postman Docs



Collection
Save your requests in a collection for reuse and sharing

Give collection a name and description



CREATE A NEW COLLECTION

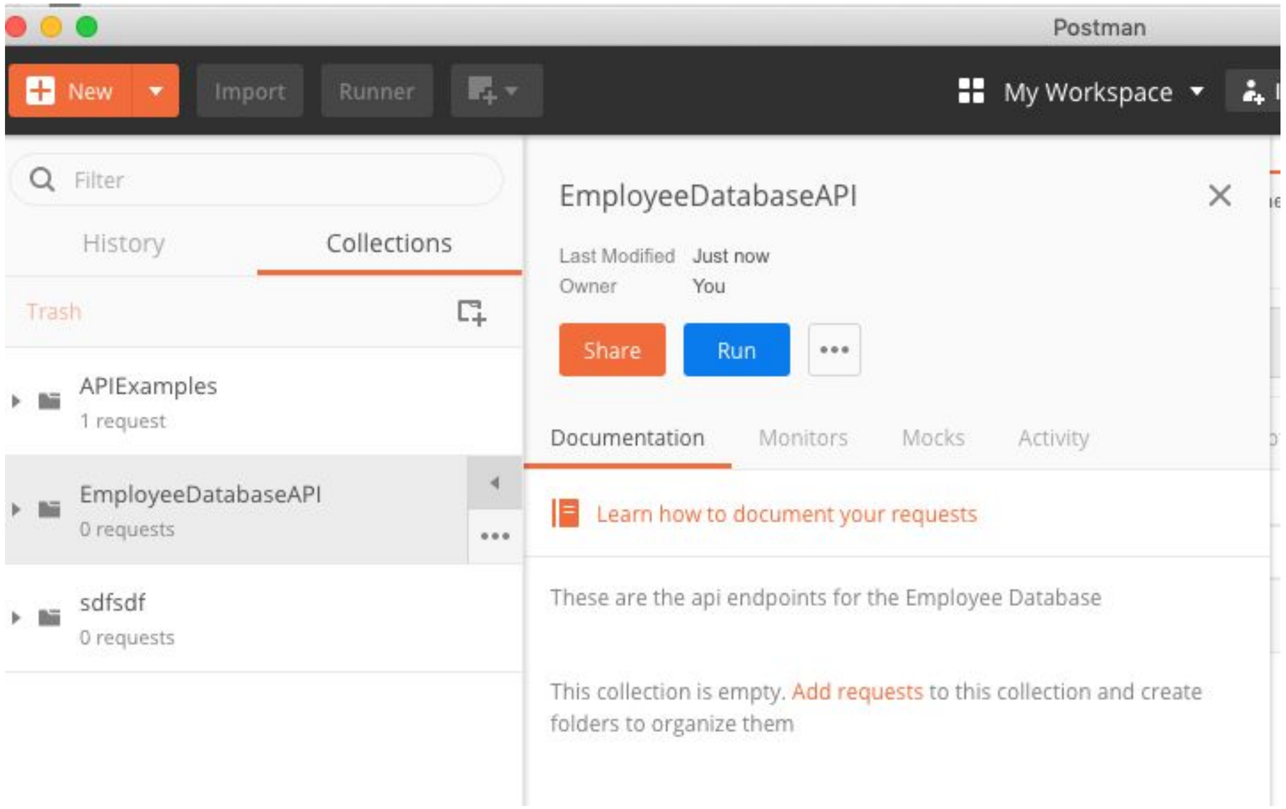
Name

EmployeeDatabaseAPI

This description will show in your collection's documentation, along with the desc

These are the api endpoints for the Employee Database

You will see this:



EmployeeDatabaseAPI

Last Modified Just now
Owner You

Learn how to document your requests

These are the api endpoints for the Employee Database

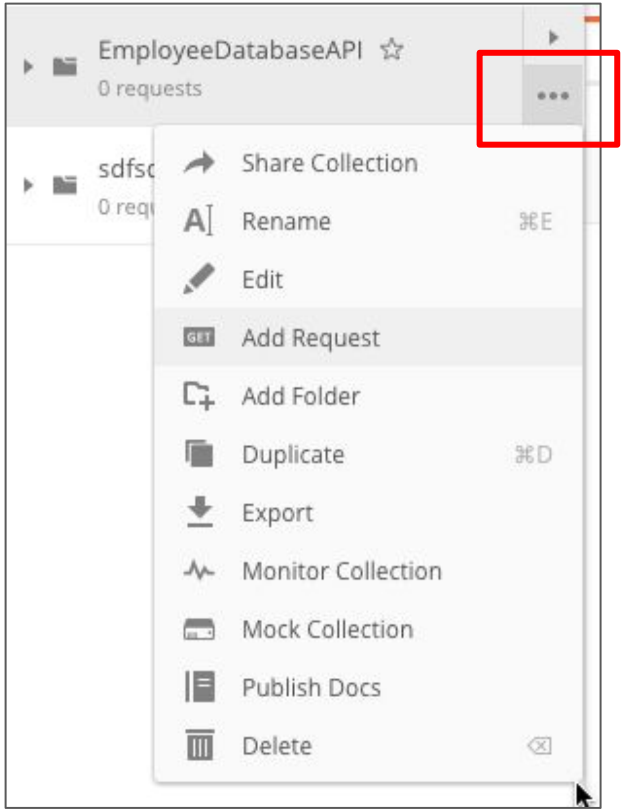
This collection is empty. Add requests to this collection and create folders to organize them

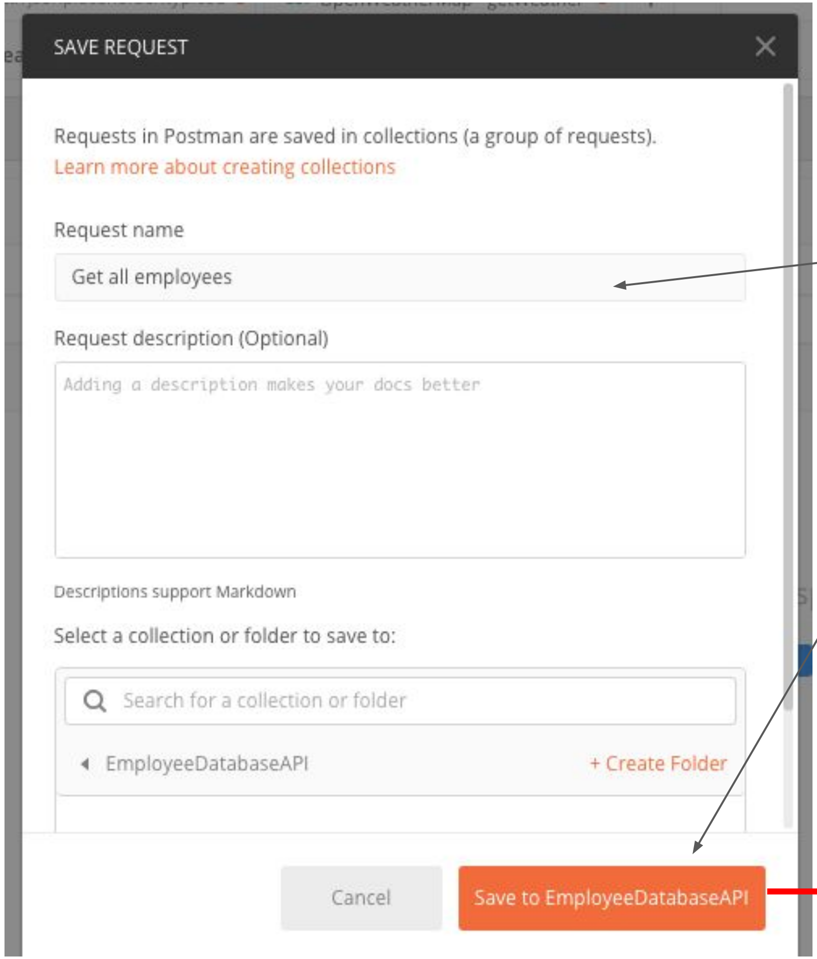
Create some requests

Look at your collection

Click on … button

In popup menu, select **Add Request**





SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
Learn more about creating collections

Request name

Get all employees

Request description (Optional)

Adding a description makes your docs better

Descriptions support Markdown

Select a collection or folder to save to:

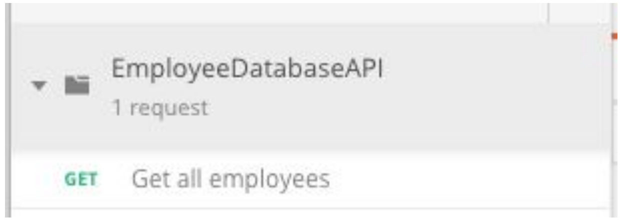
Search for a collection or folder

EmployeeDatabaseAPI + Create Folder

Cancel | Save to EmployeeDatabaseAPI

Give request a name

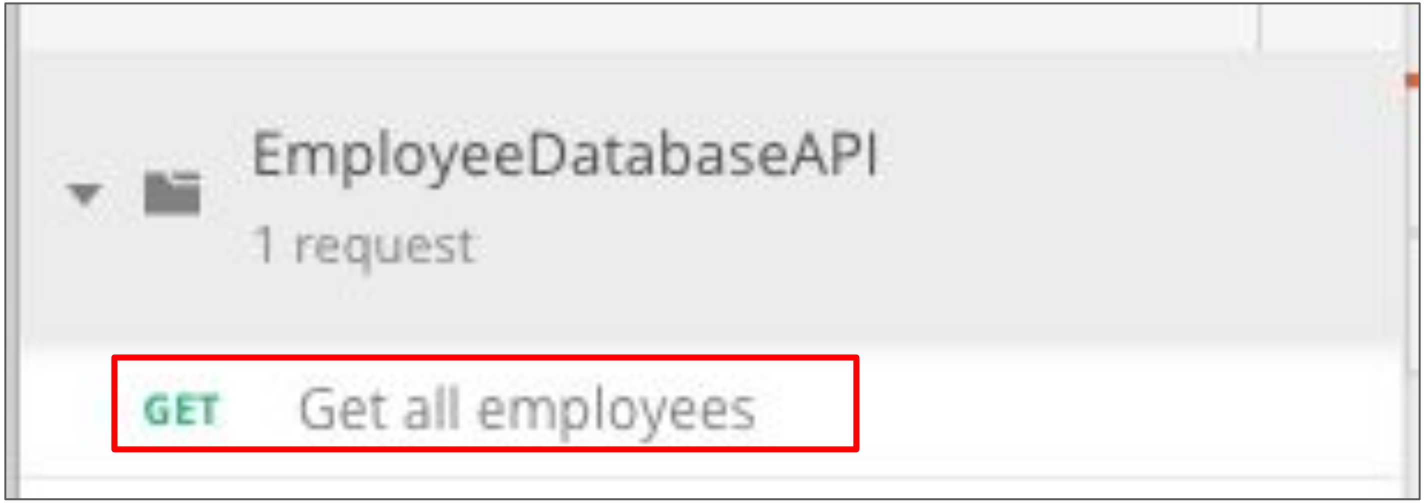
Then press **Save**



EmployeeDatabaseAPI
1 request

GET Get all employees

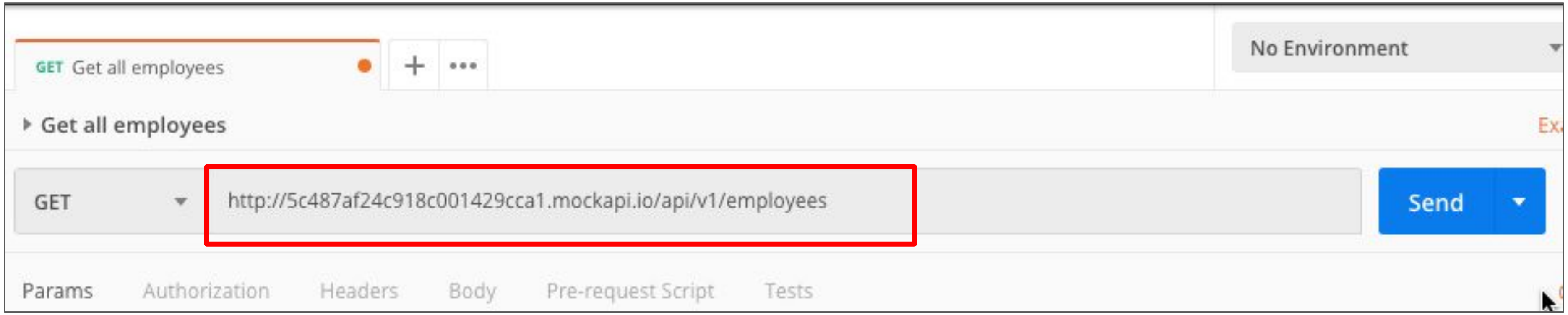
Click on the GET request



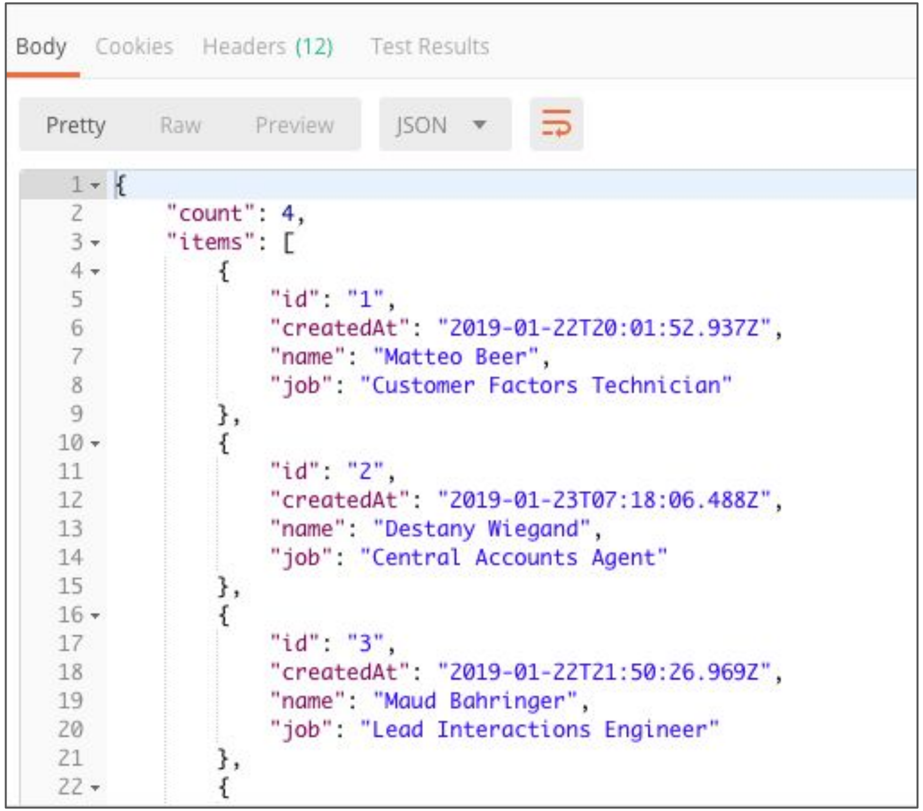
1. Put your URL into the box

https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22

2. Press SEND

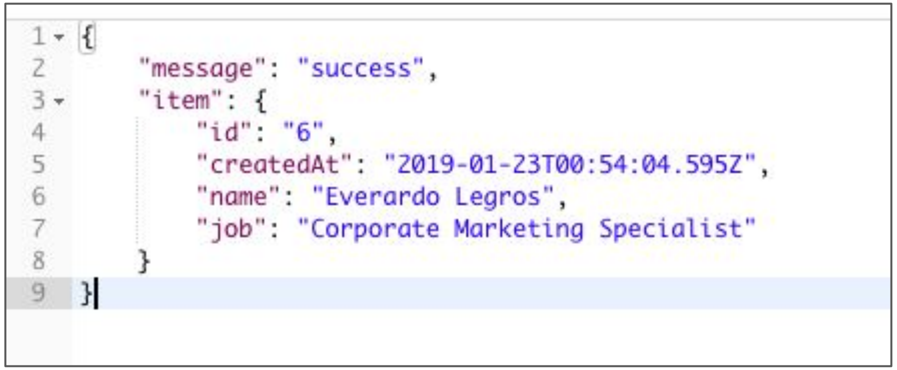


Look at result - it should match the api



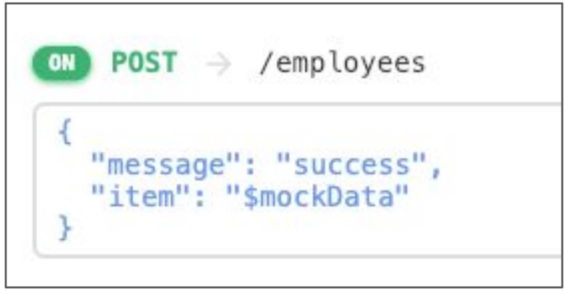
```
{
    "count": 4,
    "items": [
        {
            "id": "1",
            "createdAt": "2019-01-22T20:01:52.937Z",
            "name": "Matteo Beer",
            "job": "Customer Factors Technician"
        },
        {
            "id": "2",
            "createdAt": "2019-01-23T07:18:06.488Z",
            "name": "Destany Wiegand",
            "job": "Central Accounts Agent"
        },
        {
            "id": "3",
            "createdAt": "2019-01-22T21:50:26.969Z",
            "name": "Maud Bahringer",
            "job": "Lead Interactions Engineer"
        },
        {
```

Why does the JSON response look like this?



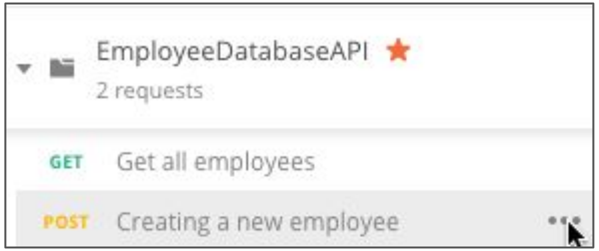
```
{
      "message": "success",
      "item": {
          "id": "6",
          "createdAt": "2019-01-23T00:54:04.595Z",
          "name": "Everardo Legros",
          "job": "Corporate Marketing Specialist"
      }
}
```

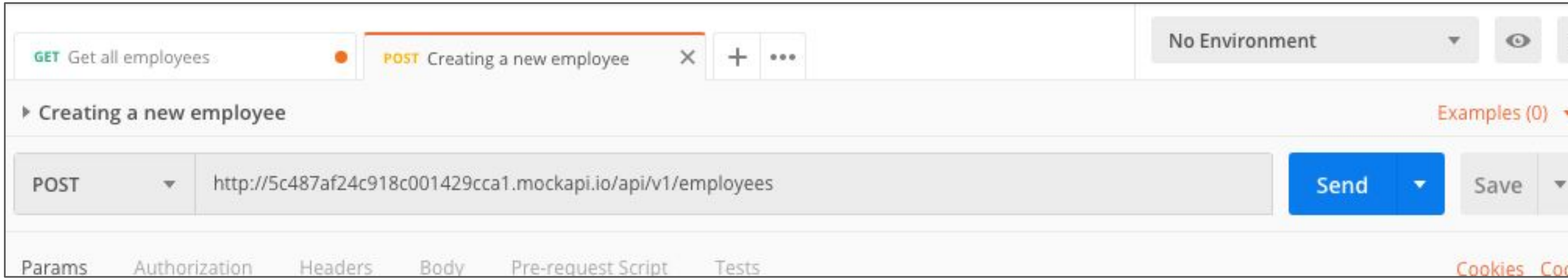
Because you specified in the MockApi.io website that POST requests should look like this



```
POST → /employees
{
  "message": "success",
  "item": "$mockData"
}
```

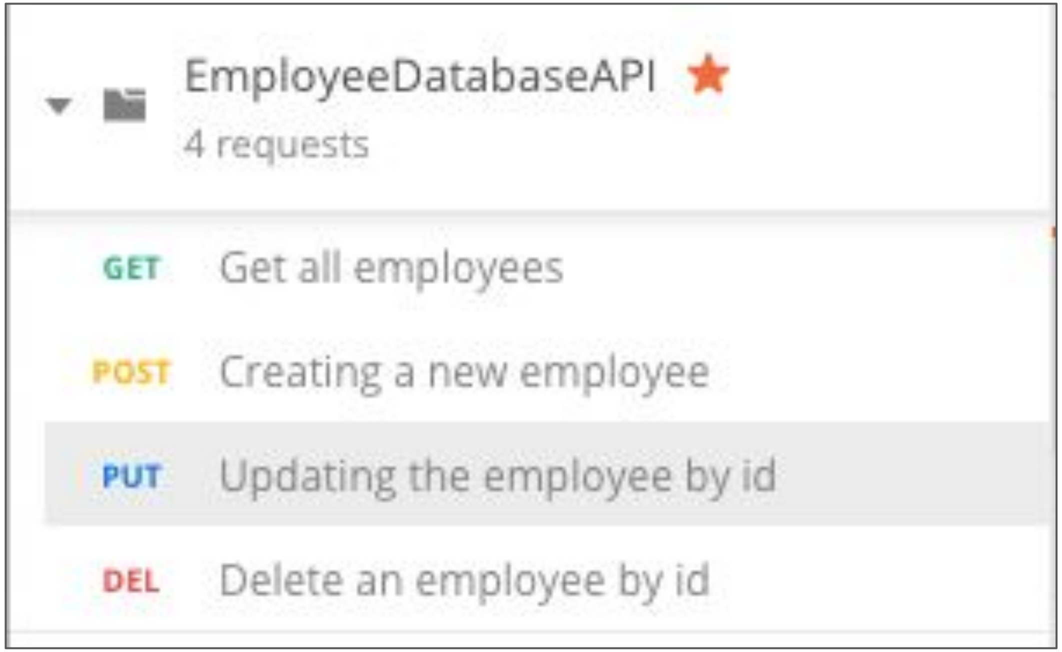
Repeat the same process for PUT , POST, DELETE





```
http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees
```

When done, you have this



EmployeeDatabaseAPI

4 requests

GET Get all employees

POST Creating a new employee

PUT Updating the employee by id

DEL Delete an employee by id

Now you can test!

Note: There is something different you must do to setup the PUT and DELETE requests.

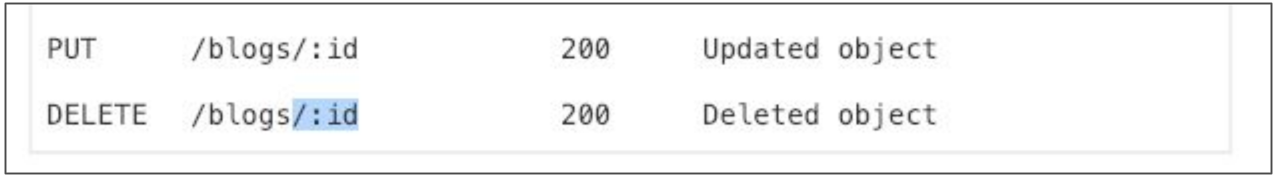
- PUT and DELETE use a different URL

http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees/:id

You must provide the url with the id of the employee you want to update / delete

- For PUT request only, you need to provide information in the **request body**

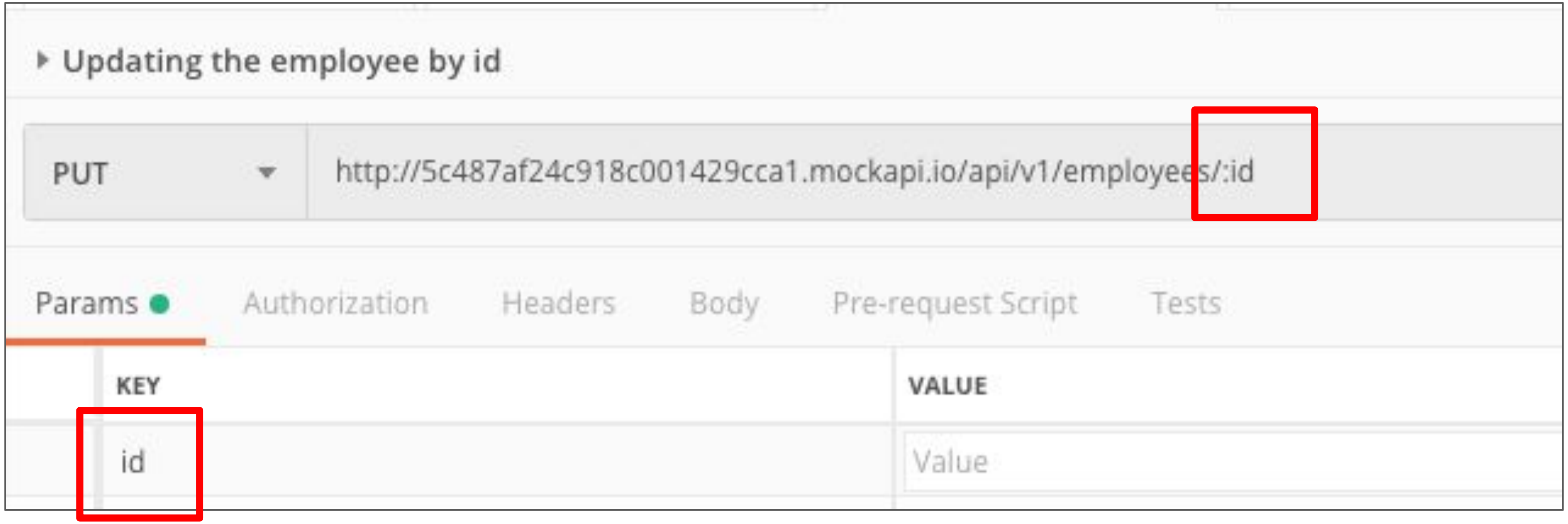
For the PUT and DELETE operations, update your url to have this at the end:



PUT	/blogs/:id	200	Updated object
DELETE	/blogs/:id	200	Deleted object

http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees/:id

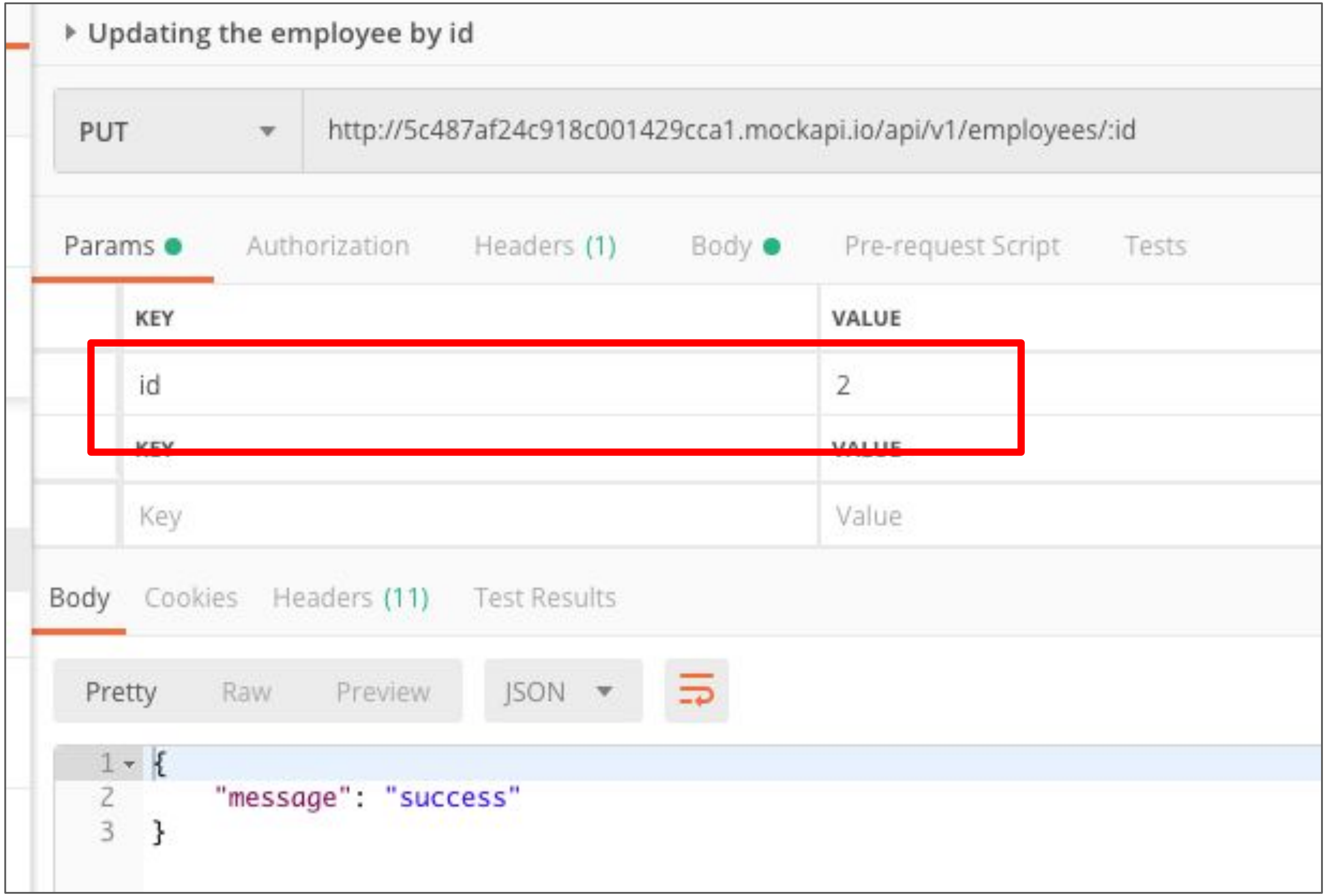
In POSTMAN, you can provide the id in the **params** panel



▸ Updating the employee by id

PUT http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees/:id

KEY	VALUE
id	Value



Updating the employee by id

PUT http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees/:id

Params ● Authorization Headers (1) Body ● Pre-request Script Tests

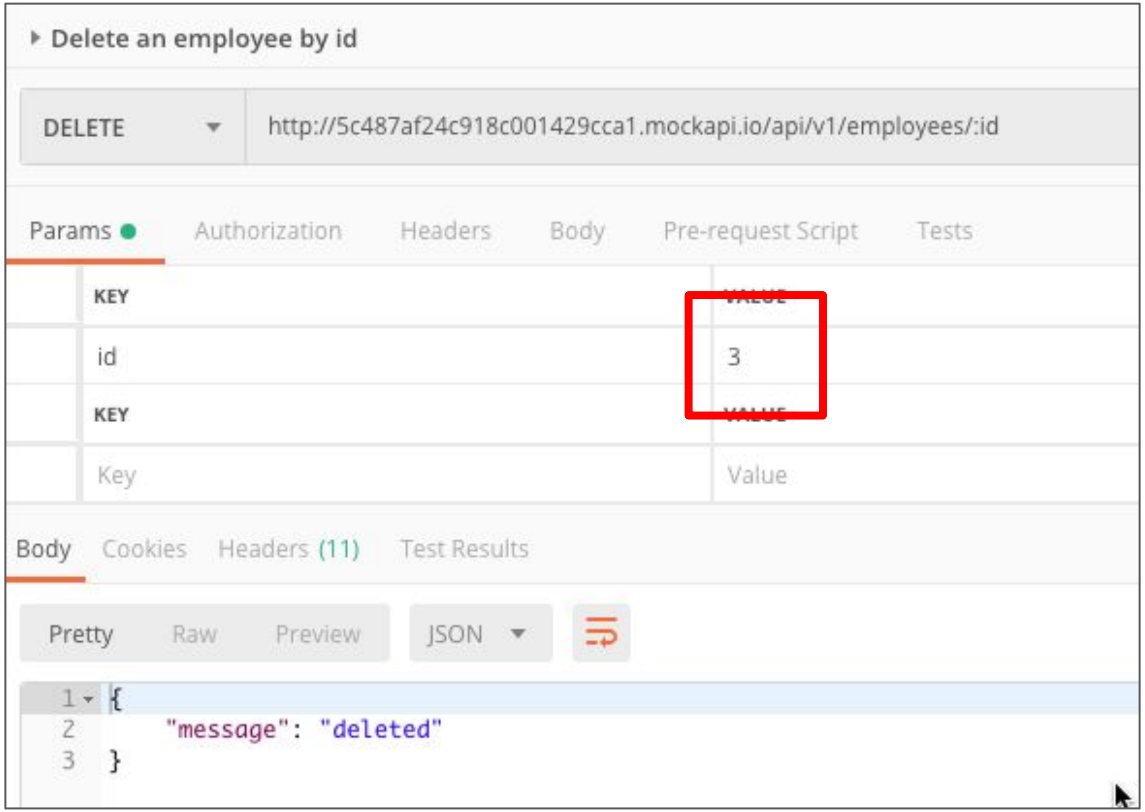
KEY	VALUE
id	2
Key	Value

Body Cookies Headers (11) Test Results

Pretty Raw Preview JSON

```
{
    "message": "success"
}
```

Example: Delete employee by id



Delete an employee by id

DELETE http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees/:id

Params Authorization Headers Body Pre-request Script Tests

KEY	VALUE
id	3

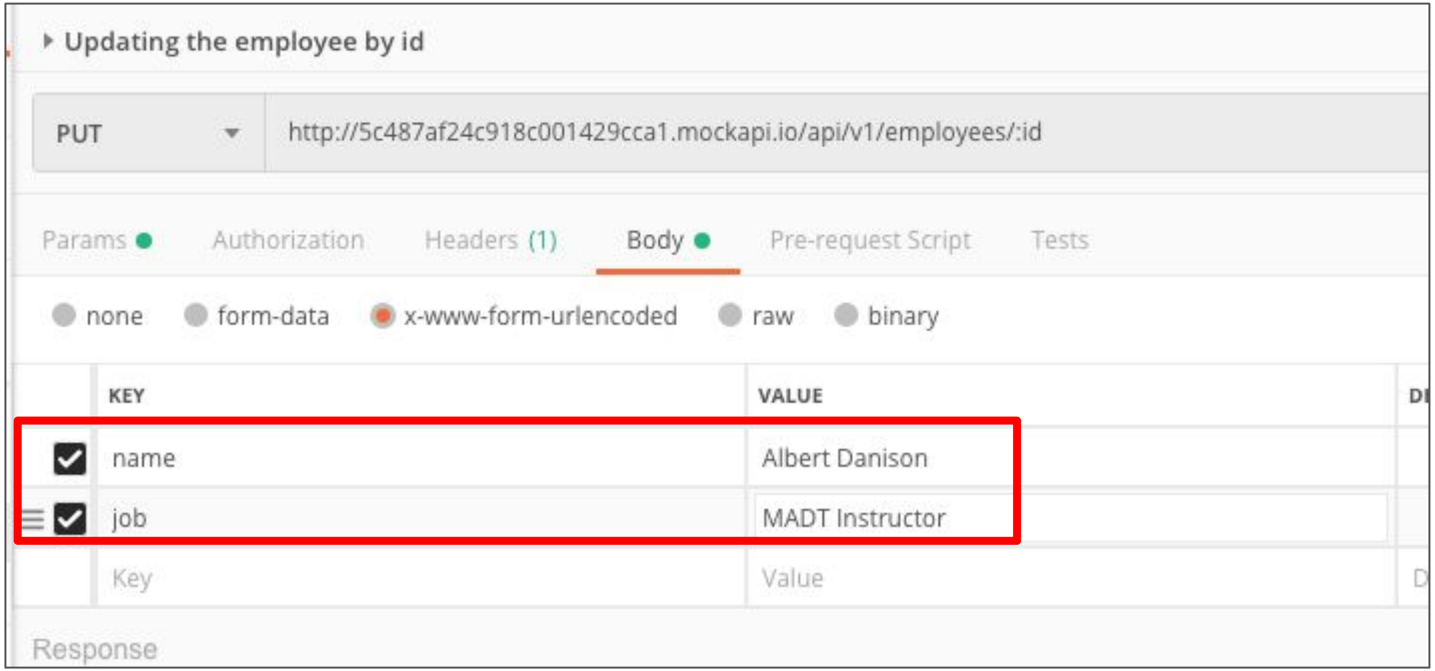
KEY	VALUE
Key	Value

Body Cookies Headers (11) Test Results

Pretty Raw Preview JSON

```
{
    "message": "deleted"
}
```

For PUT request, you also need to include information in **Body** panel



▸ Updating the employee by id

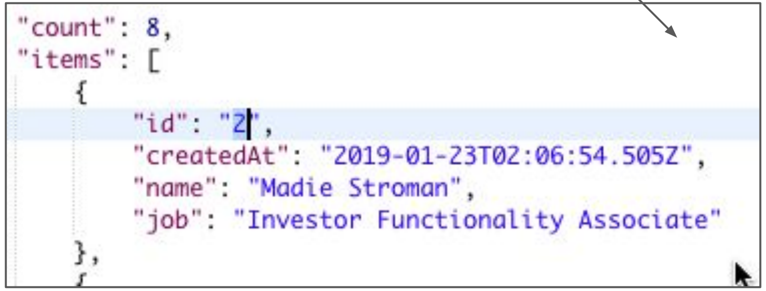
PUT ▾ http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees/:id

	KEY	VALUE
✓	name	Albert Danison
✓	job	MADT Instructor
	Key	Value

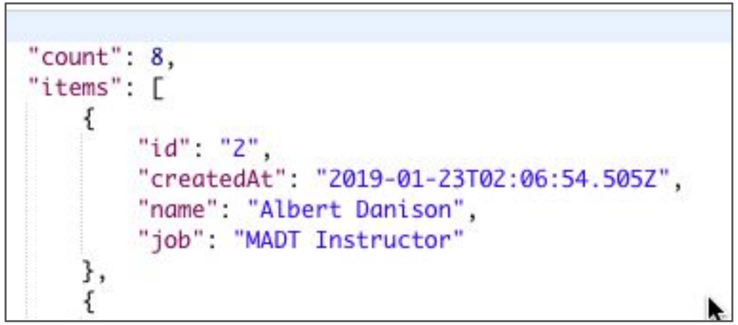
Response

Before PUT

After PUT



```
"count": 8,
"items": [
    {
        "id": "2",
        "createdAt": "2019-01-23T02:06:54.505Z",
        "name": "Madie Stroman",
        "job": "Investor Functionality Associate"
    },
```



```
"count": 8,
"items": [
    {
        "id": "2",
        "createdAt": "2019-01-23T02:06:54.505Z",
        "name": "Albert Danison",
        "job": "MADT Instructor"
    },
    {
```

Writing Automated Tests

What am I testing?

1. Test that correct status code is returned

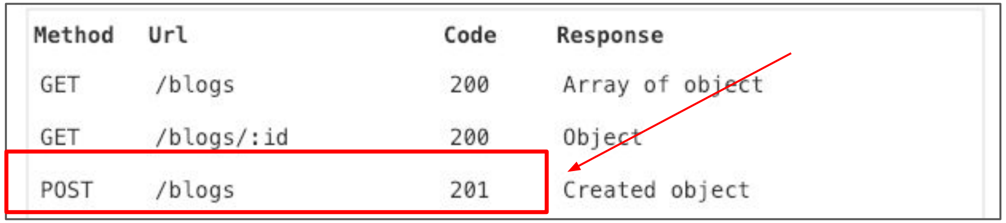
- Does the response have the correct status code?

2. Test that correct response is received

- Does response have correct keys / values?

For every request, you are testing

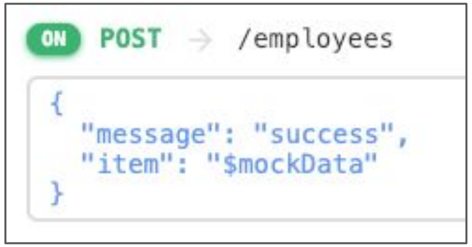
1. Test that correct status code is returned



Method	Url	Code	Response
GET	/blogs	200	Array of object
GET	/blogs/:id	200	Object
POST	/blogs	201	Created object

POST request should return status code `201`

2. Test that correct response is sent



POST → /employees

```
{
  "message": "success",
  "item": "$mockData"
}
```

Response should have `message` and `item` key-value pairs

Example - Testing GET Request

What are you testing?

TEST CASE 1: GET request produces correct status code

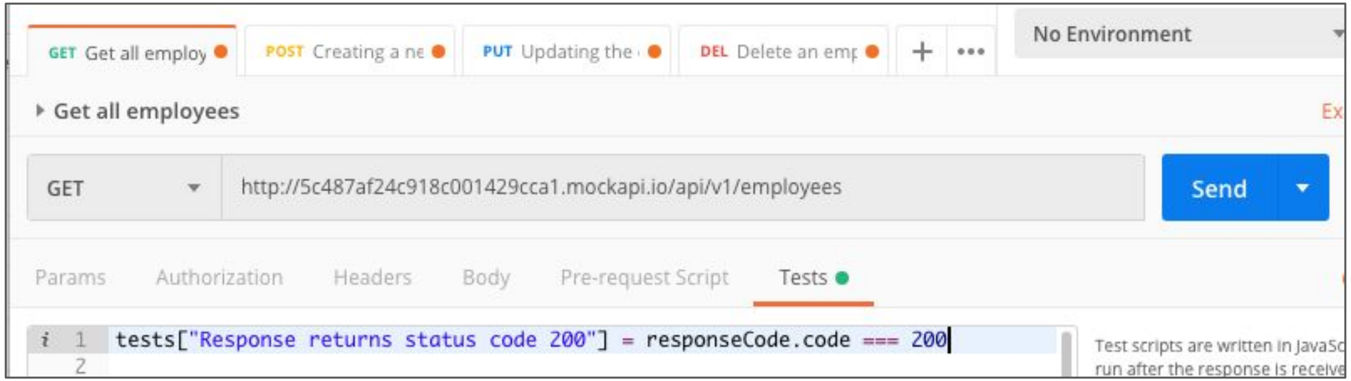
TEST CASE 2: Response contains correct content

Test Case 1: Test for correct response code

1. Copy and paste this code into the TEST panel

```
// Test if the response returns 200
// According to documentation, all GET requests should return status code 200
tests["Response returns status code 200"] = responseCode.code === 200
```

2. Press SEND, look at results

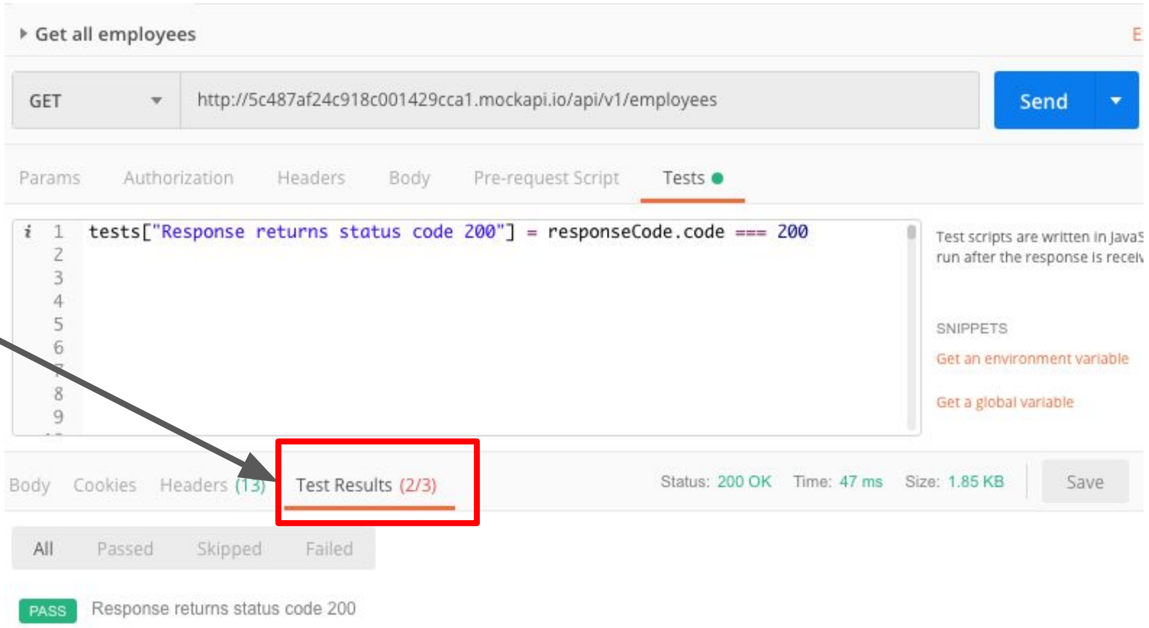


http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees

```
tests["Response returns status code 200"] = responseCode.code === 200
```

Test scripts are written in JavaSc
run after the response is receive

3. Look at result



http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees

```
tests["Response returns status code 200"] = responseCode.code === 200
```

Test scripts are written in JavaS
run after the response is receiv

Status: 200 OK Time: 47 ms Size: 1.85 KB

PASS Response returns status code 200

You can also write your test case like this:

Here is the “long” way to write the test case

```
// code you need to run to get to the test case
var results = false;
if (responseCode.code === 200) {
    results = true;
}
else {
    results = false;
}

// Here is the test case
tests["Response returns status code 200"] = results
```

Both codes produce same results

You can simplify the test case to look like this:

```
tests["Response returns status code 200"] = responseCode.code === 200
```

Test Case 2: Testing the Contents of the response

In this test case, you are checking that the JSON response matches the documentation.

Ways to do this:

- Check that it contains the correct key-value pairs
  - EXHAUSTIVE TESTING: Check all the key-value pairs
  - EQUIVALENCE TESTING: Check that **one** key exists

Exhaustive vs. Equivalence

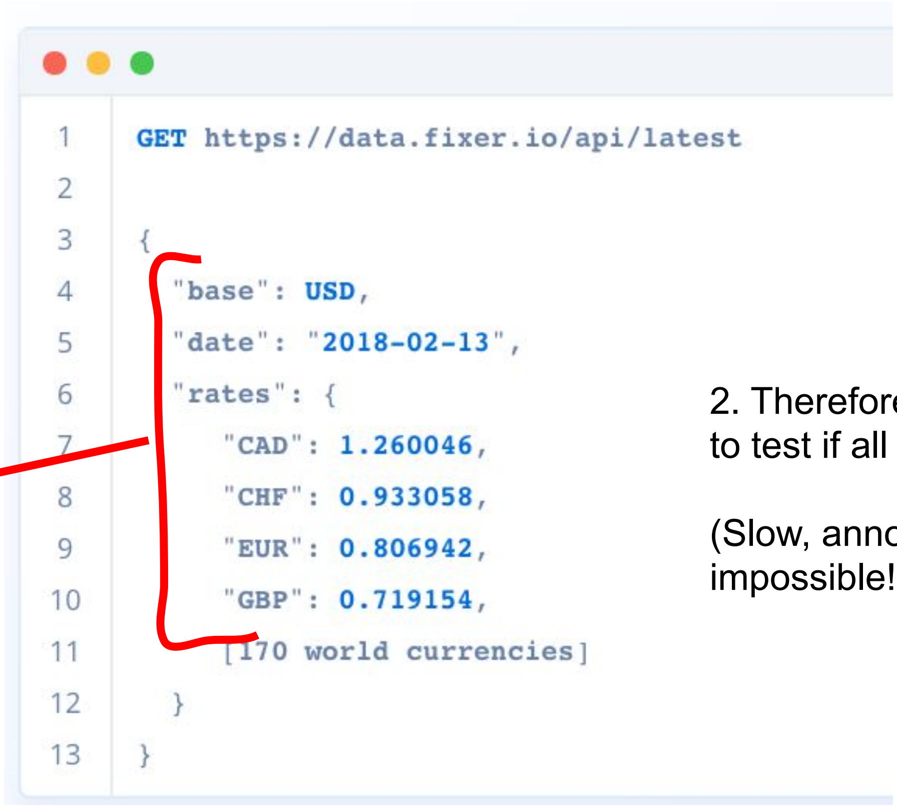
In this class, you are doing BLACK BOX testing on the APIs.

Why? Because you have NO access to the code that is generating the JSON responses.

Therefore - think back to black box testing lecture! You must SELECT the correct inputs to test!

For Exhaustive testing, check that all keys exist

This is the API documentatation for www.fixer.io



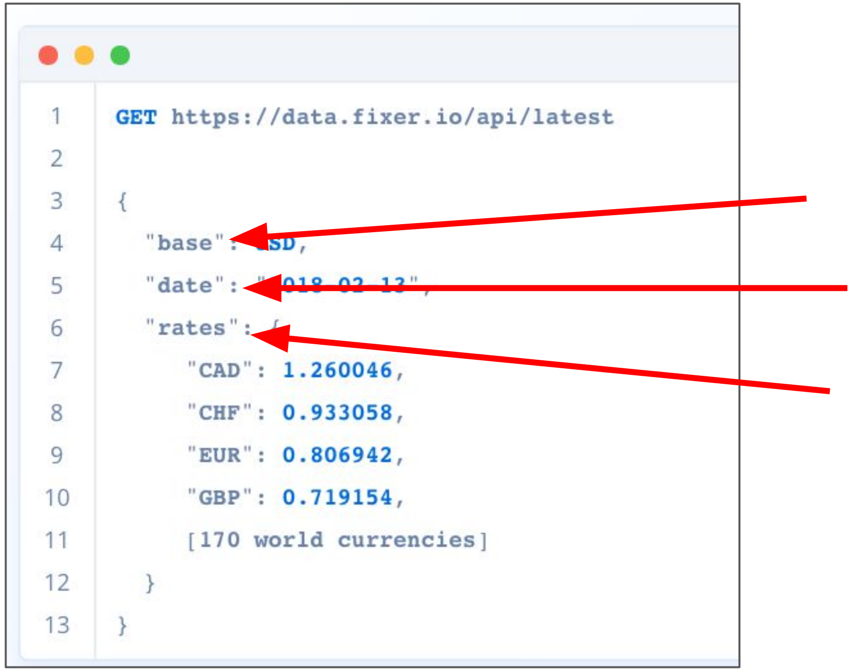
```
GET https://data.fixer.io/api/latest

{
  "base": USD,
  "date": "2018-02-13",
  "rates": {
     "CAD": 1.260046,
     "CHF": 0.933058,
     "EUR": 0.806942,
     "GBP": 0.719154,
     [170 world currencies]
  }
}
```

1. Documentation says GET response should have all these keys

2. Therefore, write test cases to test if all keys exist!

(Slow, annoying, and impossible!)

For equivalence testing, just check that ONE key exists



```
GET https://data.fixer.io/api/latest

{
  "base": "USD",
  "date": "2018-02-13",
  "rates": {
     "CAD": 1.260046,
     "CHF": 0.933058,
     "EUR": 0.806942,
     "GBP": 0.719154,
     [170 world currencies]
  }
}
```

Example cases:

If “`base`” exists, then all other keys must exist

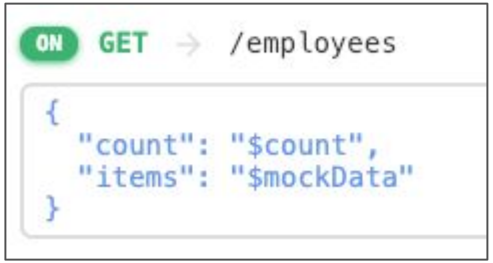
If “`date`” exists, then all other keys must exist

If “`rates`” exists, then all other keys must exist

Example: Testing that "count" key exists

Documentation says that a GET request will contain a response with 2 keys:

- count
- items



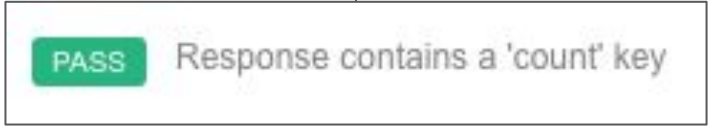
ON GET → /employees

```
{
  "count": "$count",
  "items": "$mockData"
}
```

Therefore → write a test case that checks for ONE of the keys

If one key exists, then all keys must exist (ha! Or so says the theory!)

```
tests["Response contains a 'count' key"] = responseBody.has("count");
```



PASS Response contains a 'count' key

Testing that the "count" key exists in the response

Testing the contents of Response, part 2

You can also test if the VALUES in the keys are what you expect!

See next page for examples

TEST CASE 2: Testing the contents of the response

Convert the JSON response to a Javascript dictionary, so you can work with it

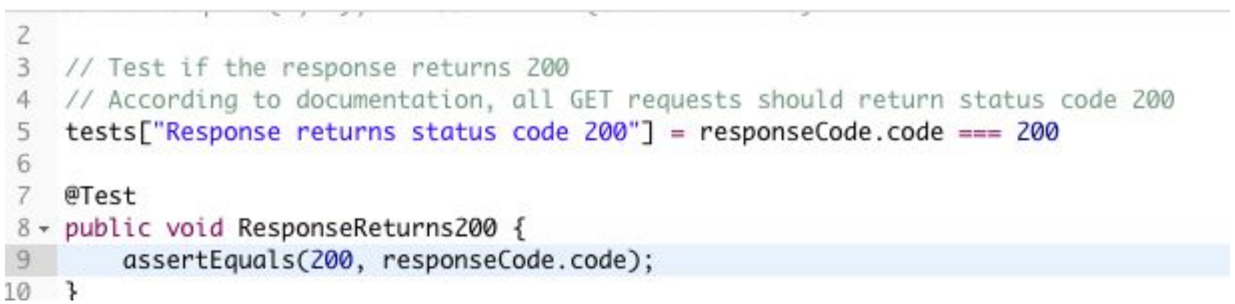
Example of debugging your code.

See `DEBUGGING WITH POSTMAN` section for more info

Here is the test case

```
// Test that the API sends back the correct information
// ------------------------
// Convert the response to a Javascript dictionary
var data = JSON.parse(responseBody);

// this is just debugging nonsense, it's not required for the test
console.log(data);
console.log("How many items? " + data.count);

// Check if the "count" key is equal to 8
tests["Response contains 8 items"] = data.count === 8;
```

Meaning of a Postman Test case



```

// Test if the response returns 200
// According to documentation, all GET requests should return status code 200
tests["Response returns status code 200"] = responseCode.code === 200

@Test
public void ResponseReturns200 {
    assertEquals(200, responseCode.code);
}
```

Debugging With Postman

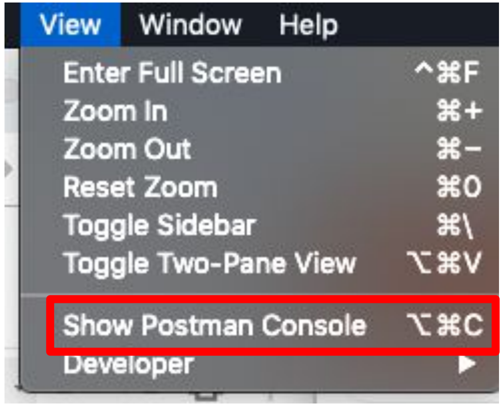
How to debug your test cases

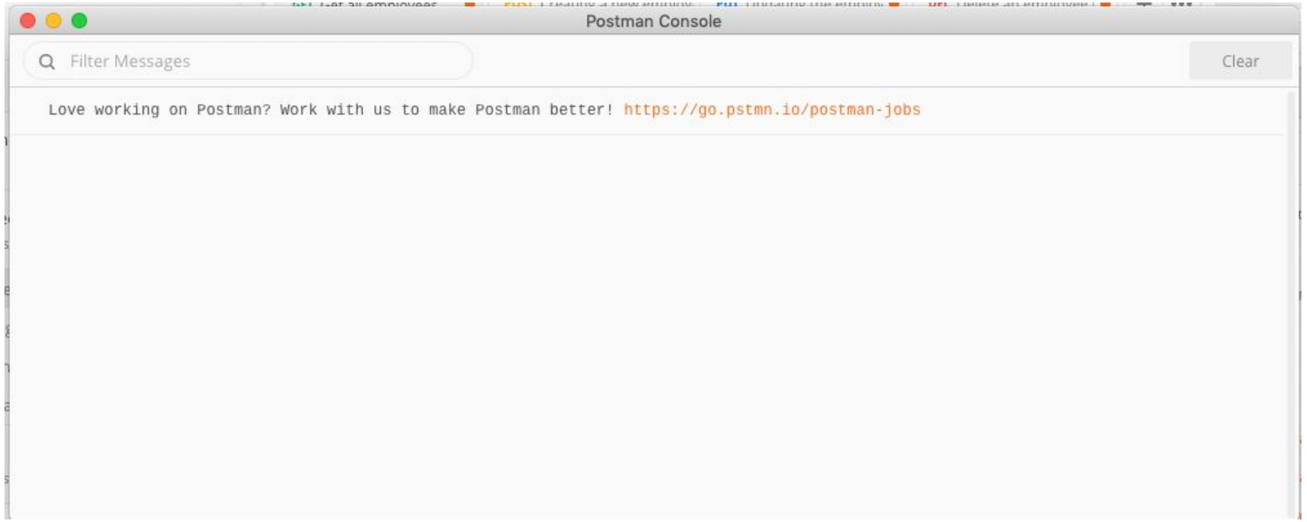
In Javascript, you can debug by outputting things to the console.

Example:

```javascript
// Output some text
console.log("here is some information");

// Output the contents of an object
var m = {"LHR":"London Airport", "YYZ":"Toronto Pearson Airport"}
console.log(m);
```

You can “view” your output by using the Postman Console.





```
Love working on Postman? Work with us to make Postman better! https://go.pstmn.io/postman-jobs
```

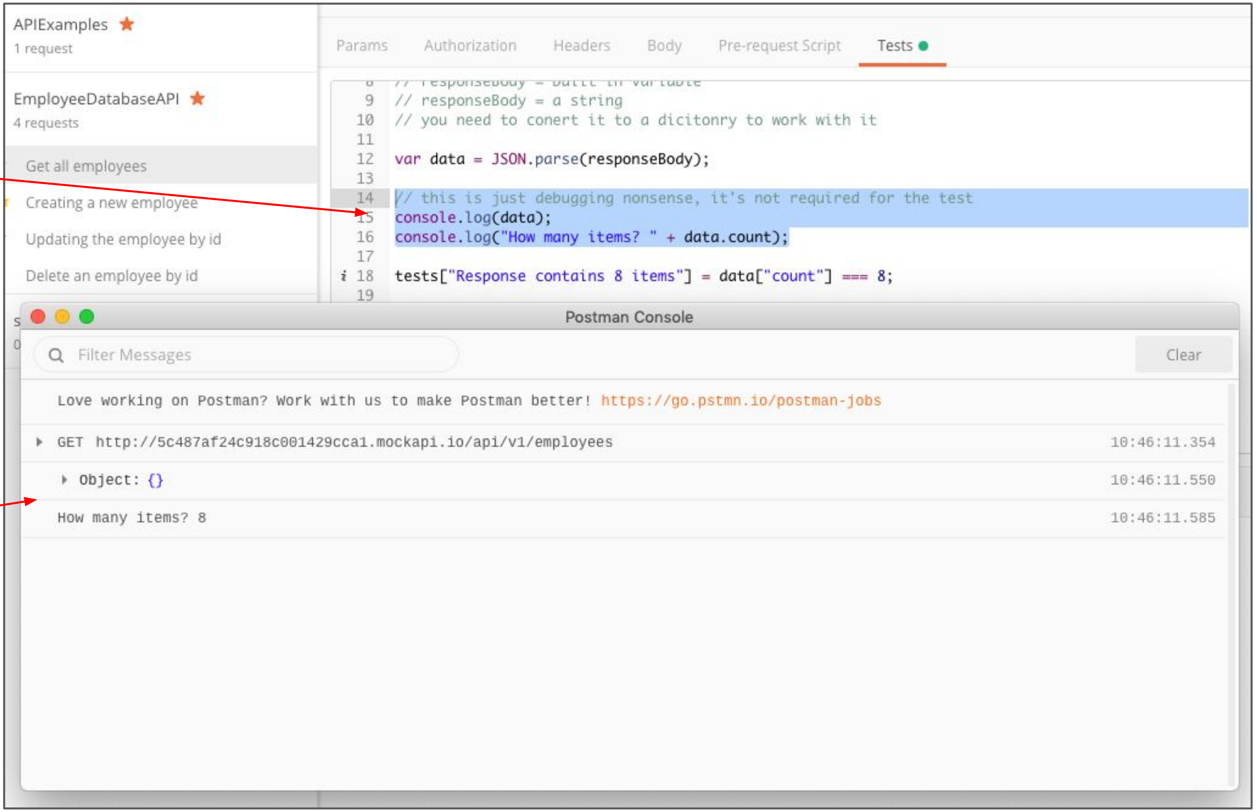
How to use Postman Console

1. Open Postman Console
2. Write your test cases
3. Press SEND
4. Look at output in Postman Console

Run test and look at postman console

Test case

Postman console output



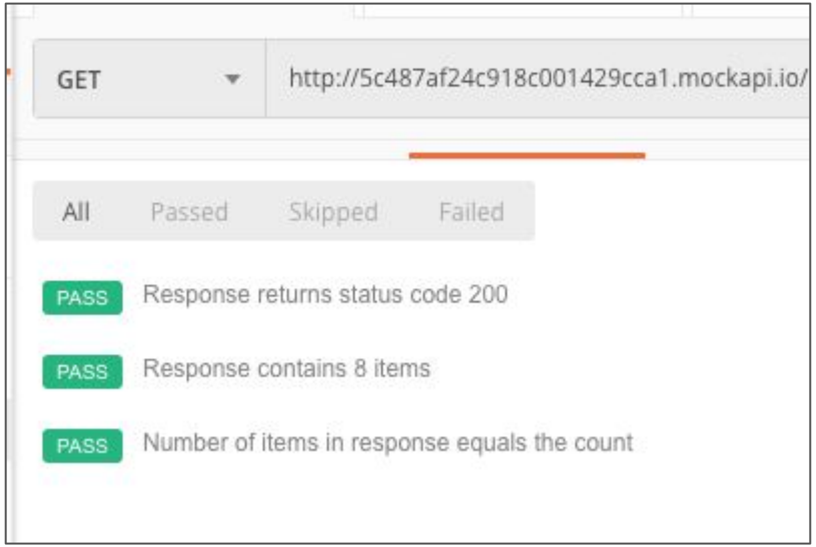
```
// responseBody = a string
// you need to conert it to a dicitonry to work with it

var data = JSON.parse(responseBody);

// this is just debugging nonsense, it's not required for the test
console.log(data);
console.log("How many items? " + data.count);

tests["Response contains 8 items"] = data["count"] === 8;
```

```
Love working on Postman? Work with us to make Postman better! https://go.pstmn.io/postman-jobs
▸ GET http://5c487af24c918c001429cca1.mockapi.io/api/v1/employees     10:46:11.354
  ▸ Object: {}                                                        10:46:11.550
How many items? 8                                                     10:46:11.585
```

More Examples of Test Scripts

Example 3

```
// Test case 3 - Test if number of items in ‘count’ matches
// the number of items in ‘items’ array
// ------------------------------------------
// OPTION 1: This is the more but easier to understand version
var numItems = data["items"].length

var results = false;
if (data["count"] === numItems) {
    results = true;
}
else {
    results = false;
}

tests["Number of items in response equals the count"] = results;

// ------------------------------------------
// OPTION 2: This is the shortcut way of writing your test case - For this
option, you don’t need to do results, or if-else statement. Just do this line!
tests["Number of items in response equals the count"] = data["count"] ===
numItems;
```



http://5c487af24c918c001429cca1.mockapi.io/

PASS Response returns status code 200

PASS Response contains 8 items

PASS Number of items in response equals the count

Two ways to write the test cases

OPTION 1 - Long but easier to understand

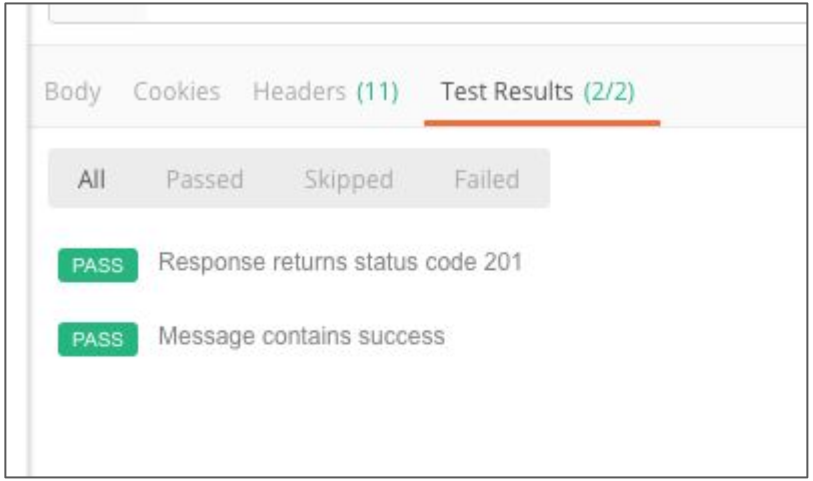
```
var numItems = data["items"].length

var results = false;
if (data["count"] === numItems) {
    results = true;
}
else {
    results = false;
}

tests["Number of items in response equals the count"] = results;
```

```
var numItems = data["items"].length

tests["Number of items in response equals the
count"] = data["count"] === numItems;
```

OPTION 2 - shorter but maybe harder to understand

Test cases for POST

```
// 1. check that status code = 201
tests["Response returns status code 201"] = responseCode.code === 201

// 2. check that data contains a success message
var data = JSON.parse(responseBody);

// get the mesage
var results = false;
var msg = data["message"];
if (msg === "success") {
    results = true;
}
else {
    results = false;
}
tests["Message contains success"] = results;
```

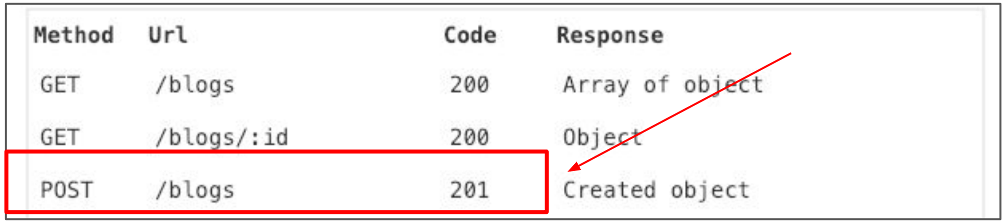


PASS Response returns status code 201

PASS Message contains success

Test cases for POST request

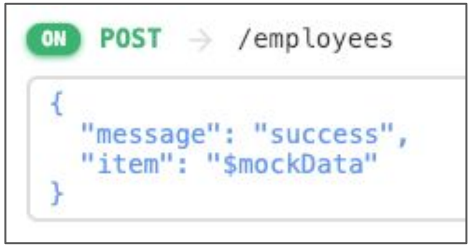
1. Test that correct status code is returned



Method	Url	Code	Response
GET	/blogs	200	Array of object
GET	/blogs/:id	200	Object
POST	/blogs	201	Created object

POST request should return status code `201`

2. Test that correct response is sent

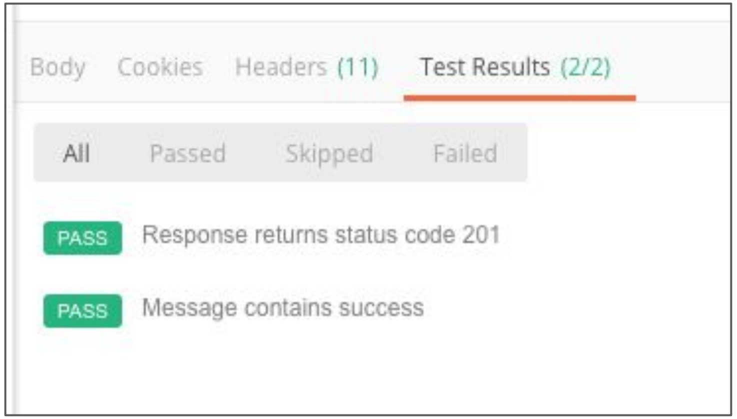


```
POST → /employees
{
  "message": "success",
  "item": "$mockData"
}
```

Response should have `message` and `item` key-value pairs

POST REQUEST - Test Case Code

```
// TEST CASE 1. check that status code = 201
// -------------------------------
tests["Response returns status code 201"] = responseCode.code === 201


// TEST CASE 2. check that data contains a success message
// -------------------------------

// 1. Convert the response to a Javascript dictionary
var data = JSON.parse(responseBody);

// 2. Parse out the “message” (key,value) pair from dictionary
var results = false;
var msg = data["message"];
if (msg === "success") {
    results = true;
}
else {
    results = false;
}
tests["Message contains success"] = results;
```

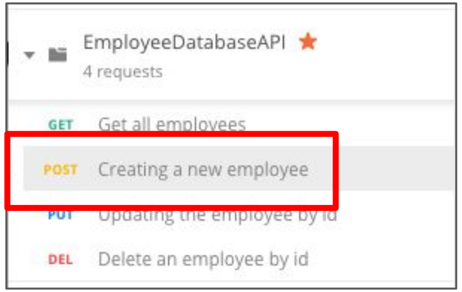


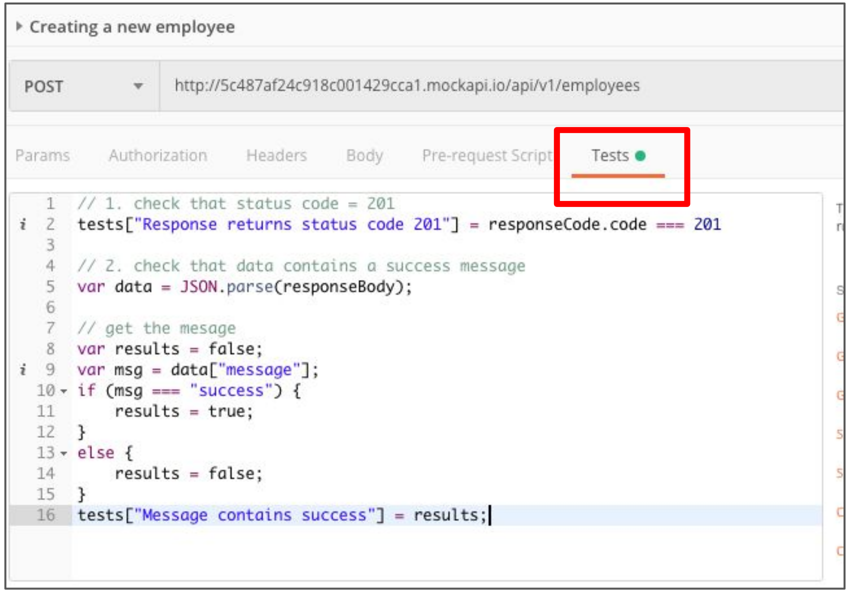
PASS Response returns status code 201

PASS Message contains success

How to do it?

1. Switch to POST request

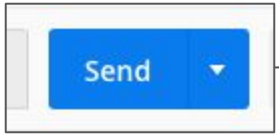

EmployeeDatabaseAPI
4 requests
GET Get all employees
POST Creating a new employee
PUT Updating the employee by id
DEL Delete an employee by id

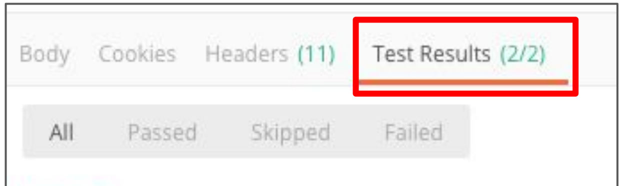


```
// 1. check that status code = 201
tests["Response returns status code 201"] = responseCode.code === 201

// 2. check that data contains a success message
var data = JSON.parse(responseBody);

// get the mesage
var results = false;
var msg = data["message"];
if (msg === "success") {
    results = true;
}
else {
    results = false;
}
tests["Message contains success"] = results;
```

2. Copy and paste code into Tests panel

3. Press SEND, then look at results




Body
Cookies
Headers (11)
Test Results (2/2)
All
Passed
Skipped
Failed