Postman

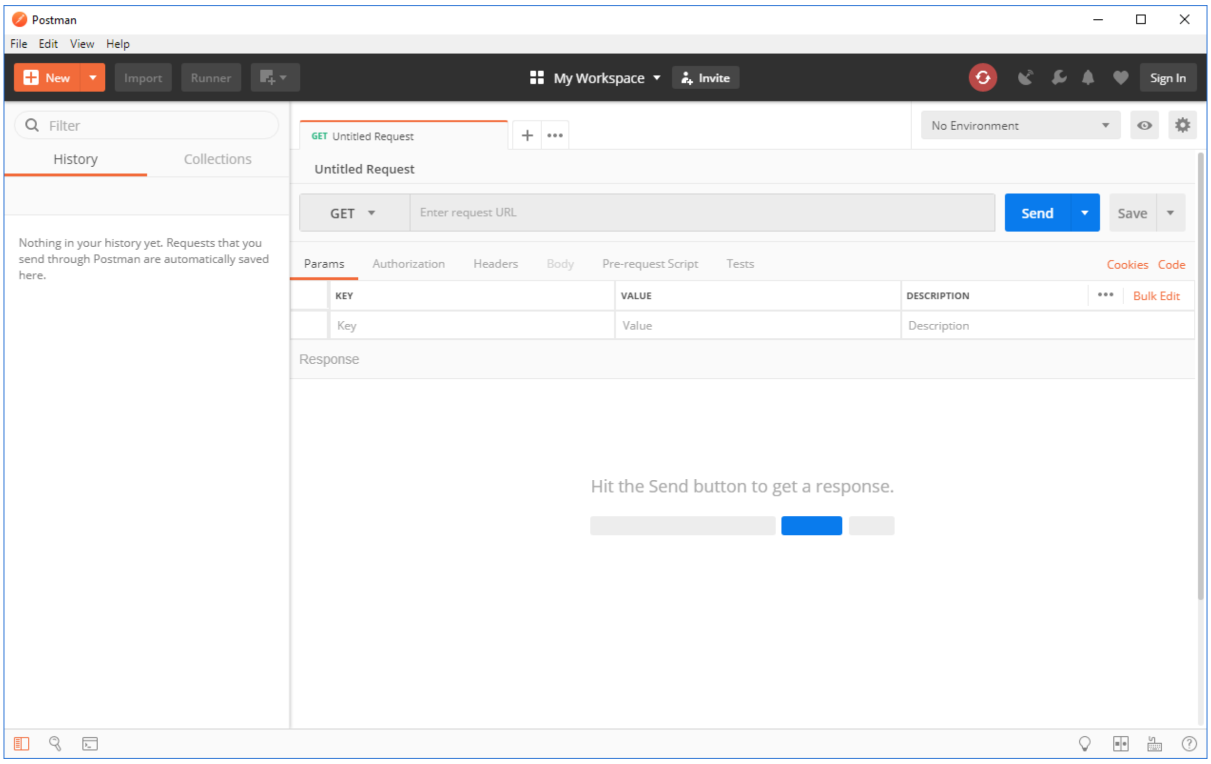
https://www.getpostman.com/

Postman is a powerful all-in-one API development tool that offers a range of services useful when developing and testing web-based applications. HTTP requests lie at the heart of its operation.

Sending a HTTP Request

https://learning.getpostman.com/docs/postman/sending_api_requests/requests

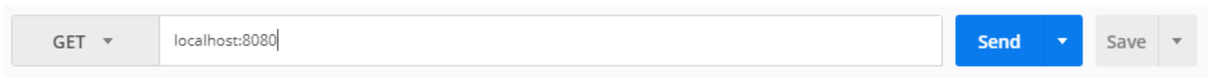
We can use Postman to send any sort of HTTP request we would like, from a simple GET to a data-driven POST. When you first open the application, you will be presented with the following window.



Nothing in your history yet. Requests that you send through Postman are automatically saved here.

Hit the Send button to get a response.

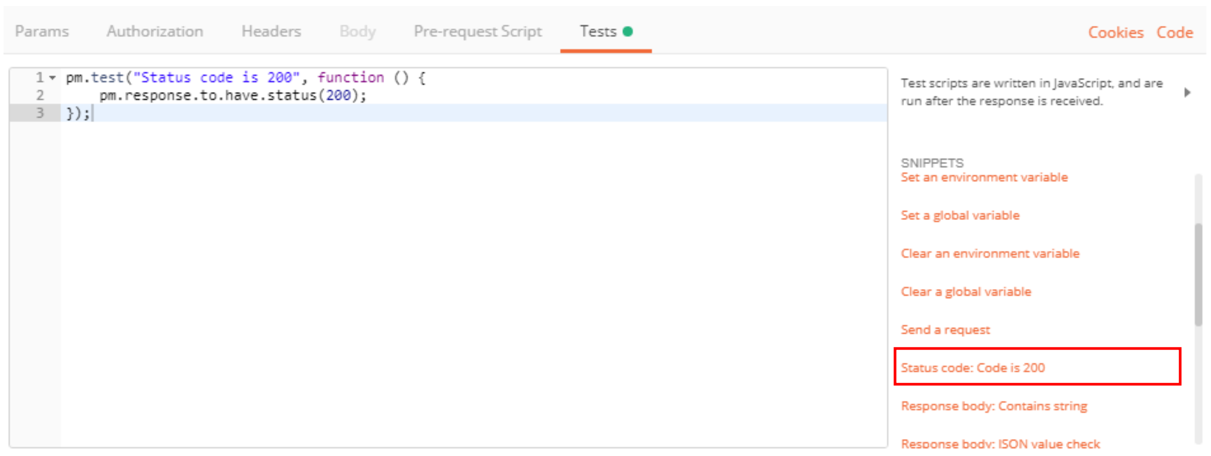
A brief look over the interface and you will find elements to be rather self-explanatory. I'll start by sending a basic GET request to the index route of my web server, currently running locally. I will define in as follows:



localhost:8080

You will also notice the various tabs under the request URL input, allowing you to define querystring parameters, authorization and other custom headers, as well as body data. We need nothing special for our basic GET request. We will however add a test to ensure the status code of the response is 200. These tests are also used for Postman's Collection Runner, covered later in this document.

Under the *Tests* tab of the request, we can add a simple status code test using the handy snippets that are provided in a list on the right. We'll find and select the "*Status code: Code is 200*" snippet:



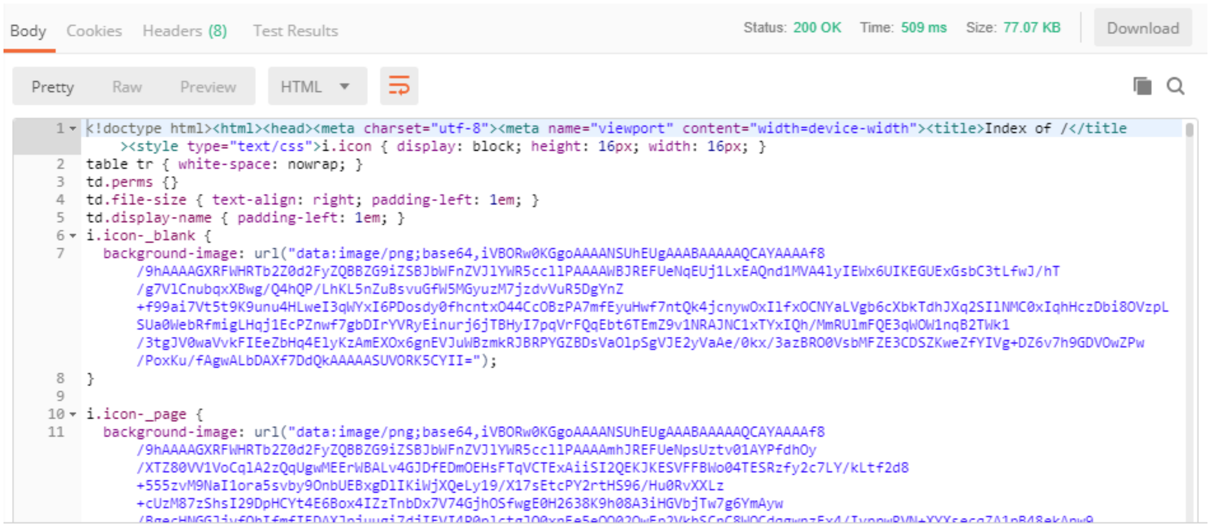
```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Test scripts are written in JavaScript, and are run after the response is received.

SNIPPETS
Set an environment variable
Set a global variable
Clear an environment variable
Clear a global variable
Send a request
Status code: Code is 200
Response body: Contains string
Response body: JSON value check

You'll see it adds the test to our request. Postman scripts use Javascript, and can be used to define *pre-request scripts* as well as our tests. You can read more on how to use them here.

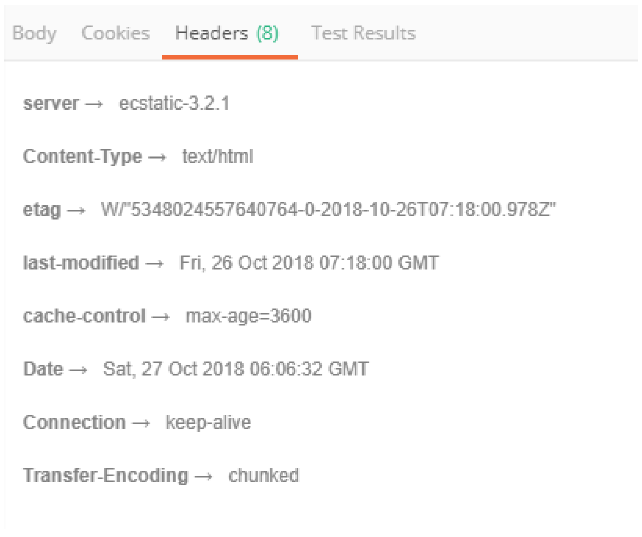
Sending the request will present the data that was received from the response. As below, we can see a body of HTML was received…



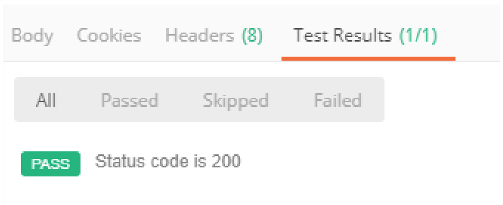
```
<!doctype html><html><head><meta charset="utf-8"><meta name="viewport" content="width=device-width"><title>Index of /</title
    ><style type="text/css">i.icon { display: block; height: 16px; width: 16px; }
table tr { white-space: nowrap; }
td.perms {}
td.file-size { text-align: right; padding-left: 1em; }
td.display-name { padding-left: 1em; }
i.icon-_blank {
  background-image: url("data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8
      /9hAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAWBJREFUeNqEUj1LxEAQnd1MVA4lyIEWx6UIKEGUExGsbC3tLfwJ/hT
      /g7VlCnubqxXBwg/Q4hQP/LhKL5nZuBsvuGfW5MGyuzM7jzdvVuR5DgYnZ
      +f99ai7Vt5t9K9unu4HLweI3qWYxI6PDosdy0fhcntxO44CcOBzPA7mfEyuHwf7ntQk4jcnywOxIlfxOCNYaLVgb6cXbkTdhJXq2SIlNMC0xIqhHczDbi8OVzpL
      SUa0WebRfmigLHqj1EcPZnwf7gbDIrYVRyEinurj6jTBHyI7pqVrFQqEbt6TEmZ9v1NRAJNC1xTYxIQh/MmRU1mFQE3qWOW1nqB2TWk1
      /3tgJV0waVvkFIEeZbHq4ElyKzAmEXOx6gnEVJuWBzmkRJBRPYGZBDsVaOlpSgVJE2yVaAe/0kx/3azBRO0VsbMFZE3CDSZKweZfYIVg+DZ6v7h9GDVOwZPw
      /PoxKu/fAgwALbDAXf7DdQkAAAAASUVORK5CYII=");
}

i.icon-_page {
  background-image: url("data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8
      /9hAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAmhJREFUeNpsUztv01AYPfdhOy
      /XTZ80VV1VoCqlA2zQqUgwMEErWBALv4GJDfEDmOEHsFTqVCTExAiiSI2QEKJKESVFFBWo04TESRzfy2c7LY/kLtf2d8
      +555zvM9NaI1ora5svby9OnbUEBxgDlIKiWjXQeLy19/X17sEtcPY2rtHS96/Hu0RvXXLz
      +cUzM87zShsI29DpHCYt4E6Box4IZzTnbDx7V74GjhOSfwgE0H2638K9h08A3iHGVbjTw7g6YmAyw
```

We can view other information such as the response headers, and our test results…



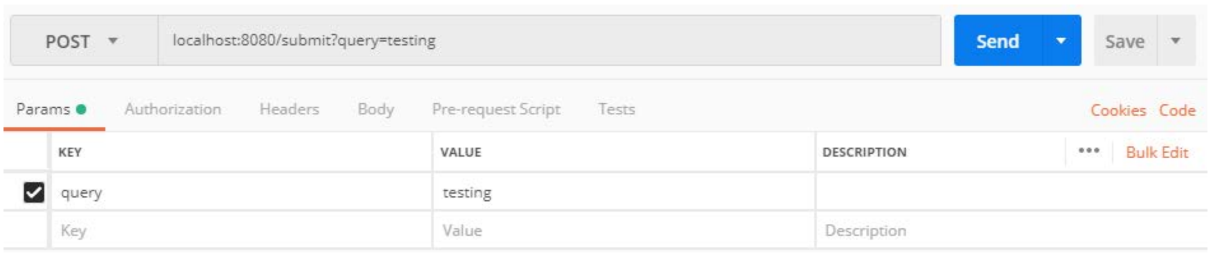
```
server → ecstatic-3.2.1
Content-Type → text/html
etag → W/"5348024557640764-0-2018-10-26T07:18:00.978Z"
last-modified → Fri, 26 Oct 2018 07:18:00 GMT
cache-control → max-age=3600
Date → Sat, 27 Oct 2018 06:06:32 GMT
Connection → keep-alive
Transfer-Encoding → chunked
```



PASS Status code is 200

You can start to see how useful this process can become when testing your applications.

I'll also try a POST that will hit a defined `/submit` route, with some data formatted as querystring parameters. As below.



localhost:8080/submit?query=testing

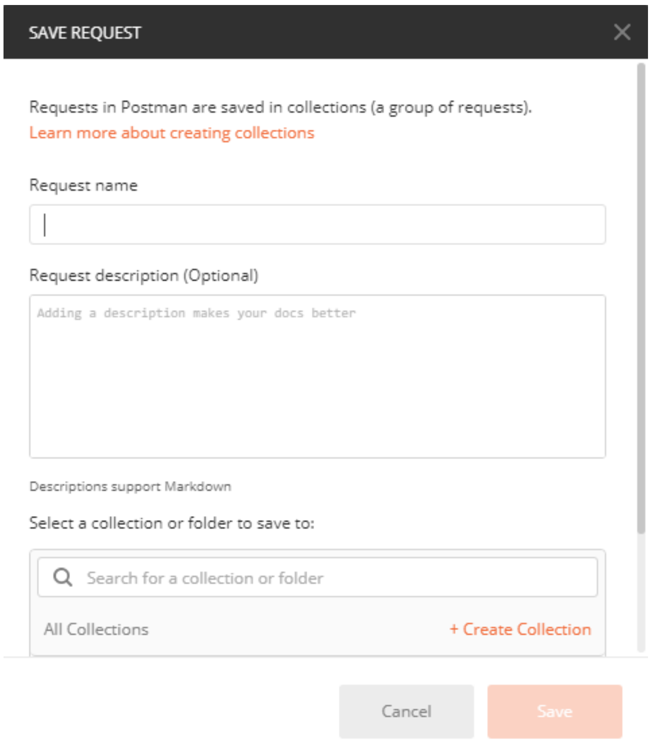
	KEY	VALUE	DESCRIPTION
✓	query	testing	
	Key	Value	Description

Collections

https://learning.getpostman.com/docs/postman/collections/intro_to_collections

Given our set of HTTP requests, we can create a collection that both saves and groups them.

You can use the *Save* button (next to the *Send* button) to save your request.



SAVE REQUEST

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

Request name

Request description (Optional)

Adding a description makes your docs better

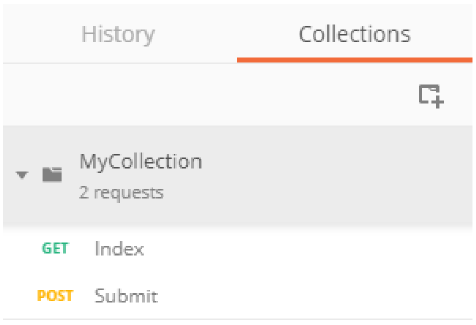
Descriptions support Markdown

Select a collection or folder to save to:

Search for a collection or folder

All Collections
+ Create Collection

Give your request a name and optional description, you can then create and save your request to a Collection. I've gone ahead and saved both my GET and POST requests, which I can then view on the left sidebar:

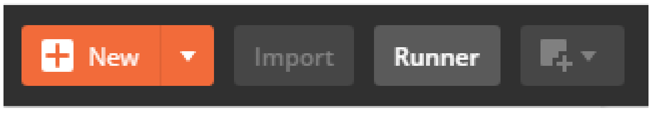


MyCollection
2 requests
GET Index
POST Submit

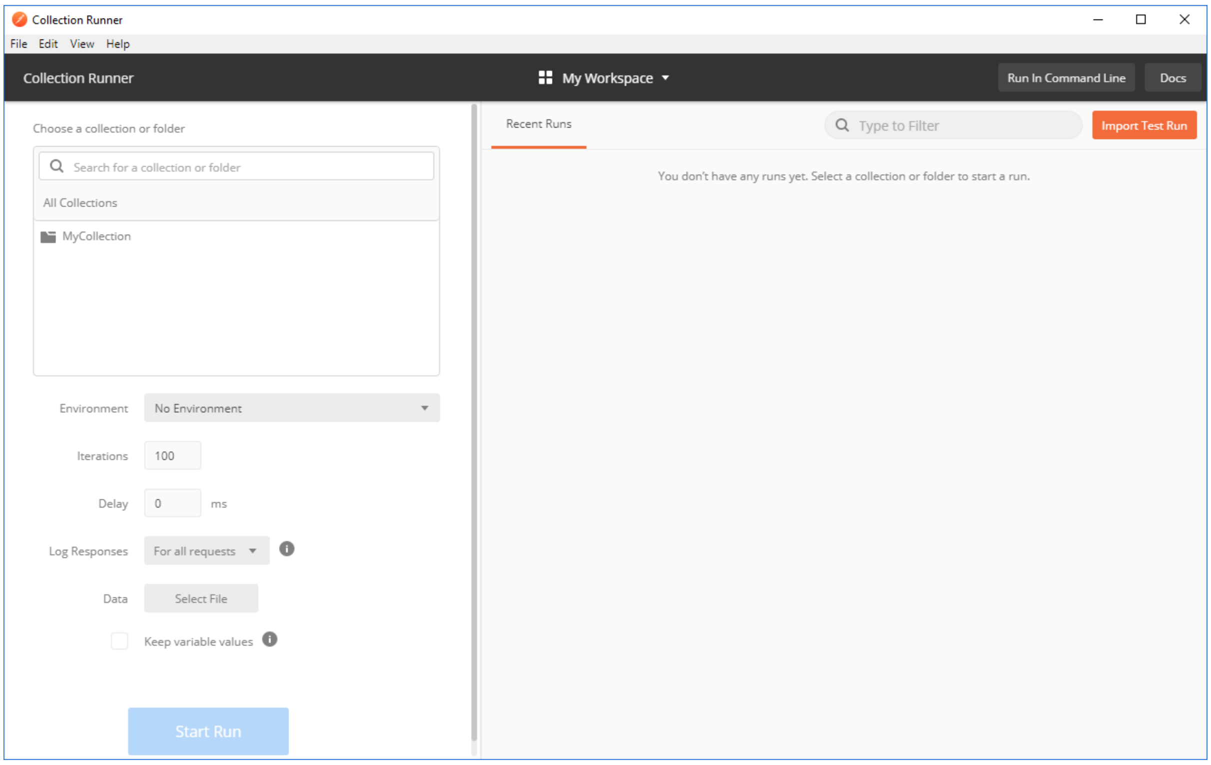
Collection Runner

https://learning.getpostman.com/docs/postman/collection_runs/intro_to_collection_runs

Given a Collection of requests, we can use Postman's Collection Runner to automatically send the set of requests one after another. Such a process can be useful for tasks such as load and responsiveness testing. We can open the Collection Runner by clicking the *Runner* button at the top left:



You will be presented with the following window:

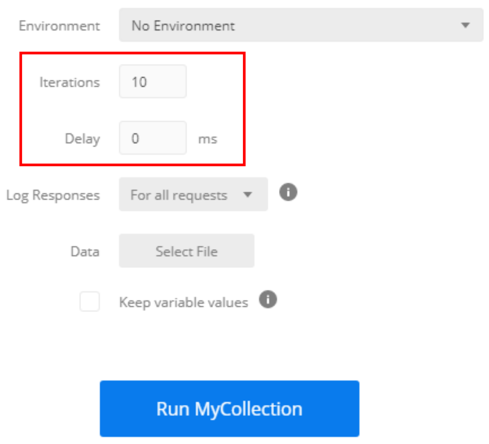


You don't have any runs yet. Select a collection or folder to start a run.

Choose a collection or folder

Environment	No Environment
Iterations	100
Delay	0 ms
Log Responses	For all requests
Data	Select File
	☐ Keep variable values

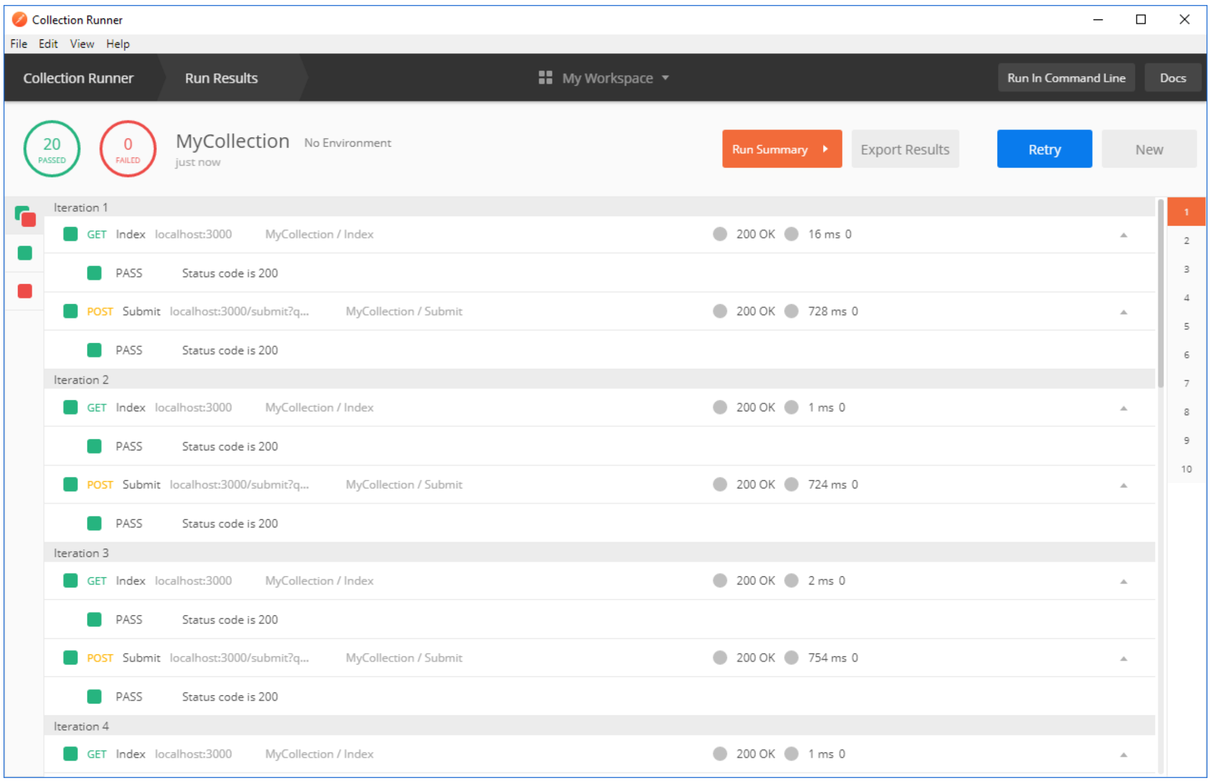
We'll get started by selecting the Collection of requests we would like to run, on the left. At which point we will specify we want to run 10 iterations, with no delay between requests:



Environment	No Environment
Iterations	10
Delay	0 ms
Log Responses	For all requests
Data	Select File
	☐ Keep variable values

Upon running our Collection, you will see that the requests begin to send…

Upon completion, we see the results for our run, as below. For each iteration, the two requests within our Collection have been run.



20 PASSED 0 FAILED MyCollection No Environment just now

Iteration 1
GET Index localhost:3000 MyCollection / Index 200 OK 16 ms 0
PASS Status code is 200
POST Submit localhost:3000/submit?q... MyCollection / Submit 200 OK 728 ms 0
PASS Status code is 200
Iteration 2
GET Index localhost:3000 MyCollection / Index 200 OK 1 ms 0
PASS Status code is 200
POST Submit localhost:3000/submit?q... MyCollection / Submit 200 OK 724 ms 0
PASS Status code is 200
Iteration 3
GET Index localhost:3000 MyCollection / Index 200 OK 2 ms 0
PASS Status code is 200
POST Submit localhost:3000/submit?q... MyCollection / Submit 200 OK 754 ms 0
PASS Status code is 200
Iteration 4
GET Index localhost:3000 MyCollection / Index 200 OK 1 ms 0

1 2 3 4 5 6 7 8 9 10

We see that of the 20 requests we sent, all 20 passed their defined tests.