POSTMAN

Postman Expert

PRESENTED BY

Johnny Tu

POSTMAN

Welcome & Introductions

Instructor: Johnny Tu

- Training Manager
- Content author
  - Postman
  - Docker
  - NGINX

POSTMAN

Housekeeping

Duration

- 3 hours
- 10 minute break every hour
- Ask questions anytime

Format

- Slides
- Demonstrations
- Exercises

Course Prerequisites

Prior Training and/or knowledge

You should have either taken the **Postman Fundamentals** course or be familiar with the following topics

- Postman Requests, Collections, Variables and Environments, Workspaces
- Basic scripting to test API's
- Simple JavaScript code
- HTTP Request structure (HTTP methods, common headers etc…)

Required Tools

- Postman Desktop application and account

Course Pre-requisites (cont'd)

If you do not have Postman installed already:

1. Go to https://www.getpostman.com/apps, download and install the app.
2. If you do not have a Postman account, sign up for a free account.
3. Open the Postman desktop application and make sure you are signed into your account

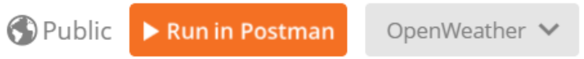
POSTMAN

EXERCISE

Get course materials

1. Open your Postman desktop application and create a new personal Workspace
2. Open a web browser and go to

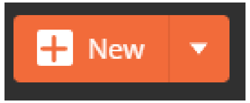
   https://documenter.getpostman.com/view/4805376/RznEKJNG
3. Click the Run in Postman button
4. Select "Postman for Windows / Mac" option
5. At the end of this, you should have:
   - **Collection:** Ex1.1 – Using Environments
   - **Environment:** testEnv



POSTMAN

EXERCISE

Get course materials (cont'd)



6. Click the "New" button on the top left of the Postman App
7. Select the Templates tab
8. Use the search function to search for "Postman Workshop"
9. You should see a template called "Postman Workshop Restful Booker"
10. Select the template and click "Run in Postman"
11. Click on "Create"
12. You should now have a collection called "Restful Booker"

POSTMAN

Agenda

1. Collection Runs
2. Advanced Scripting
3. API Documentation
4. Mock Servers
5. Monitors

POSTMAN

Module 1 – Collection Runs

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Start a collection run
- Use environments in a collection run
- Run multiple iterations of a collection
- Upload a data file for a collection run
- Debug a collection run

POSTMAN

Start a Collection Run

Intro to Collection runs

- A collection run is an execution of all requests in a collection
- Requests are run one after another
- Collections can be run by
  - ➢ Postman application collection runner
  - ➢ Newman (CLI tool)
  - ➢ Postman monitors

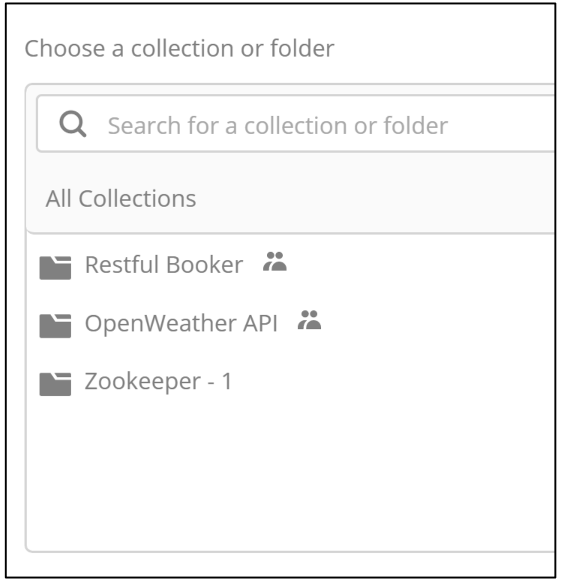
Why run Collections

- Useful in automated API testing
- Scripts can be used to build test suites and pass data between API requests
    - Helps to mirror a workflow
    - Replicate a user experience
- Test for hundreds of scenarios by using a data file

POSTMAN

Starting a Collection Run

- Click on "Runner" to open the Postman Application Collection Runner
- Select the Collection or folder to run
  - ➢ Selecting a collection will run all requests in every folder sequentially in the order listed
  - ➢ Selecting a folder will run only the requests in that folder
- Click "Run"



Choose a collection or folder

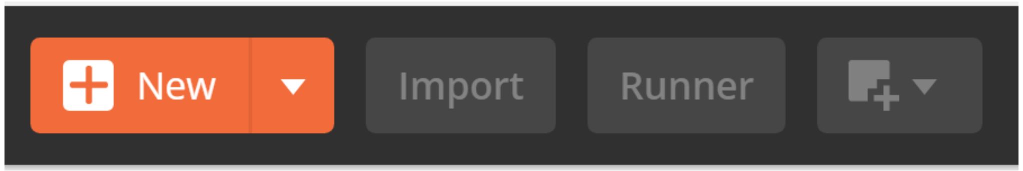
Search for a collection or folder

All Collections

Restful Booker

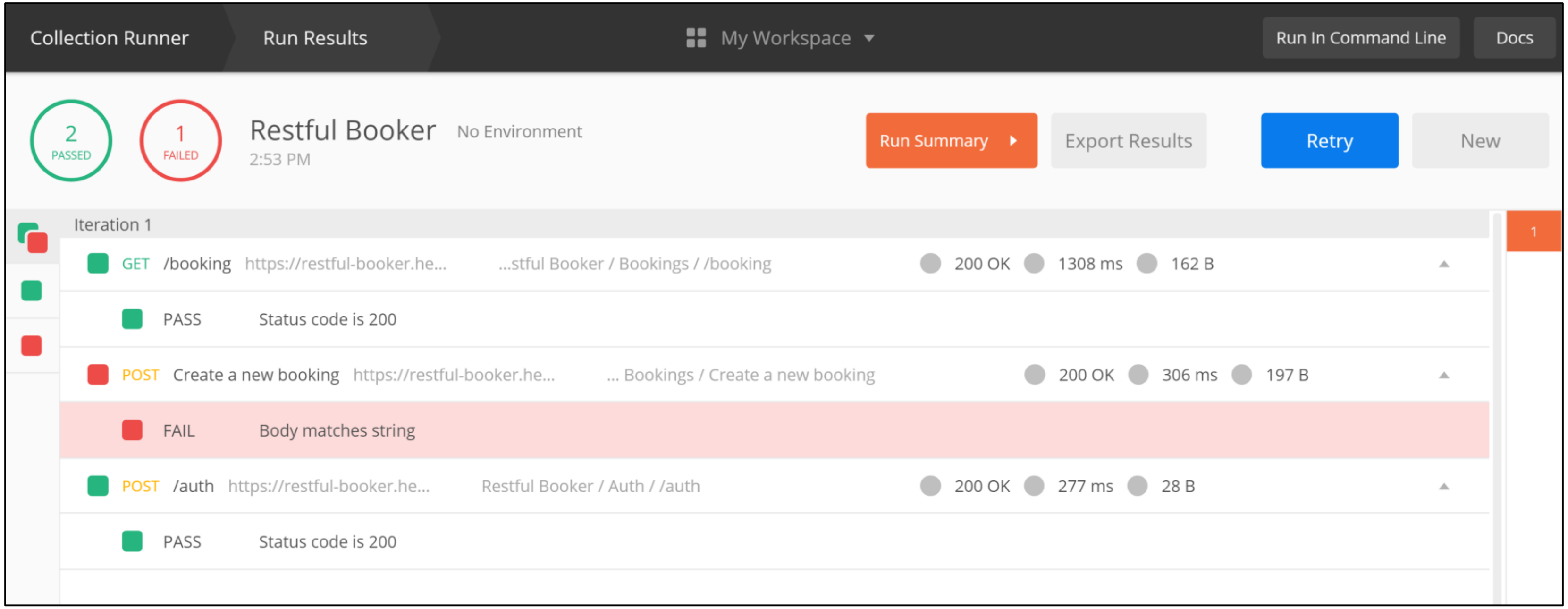
OpenWeather API

Zookeeper - 1



POSTMAN

Viewing Results



2 PASSED 1 FAILED

Restful Booker No Environment
2:53 PM

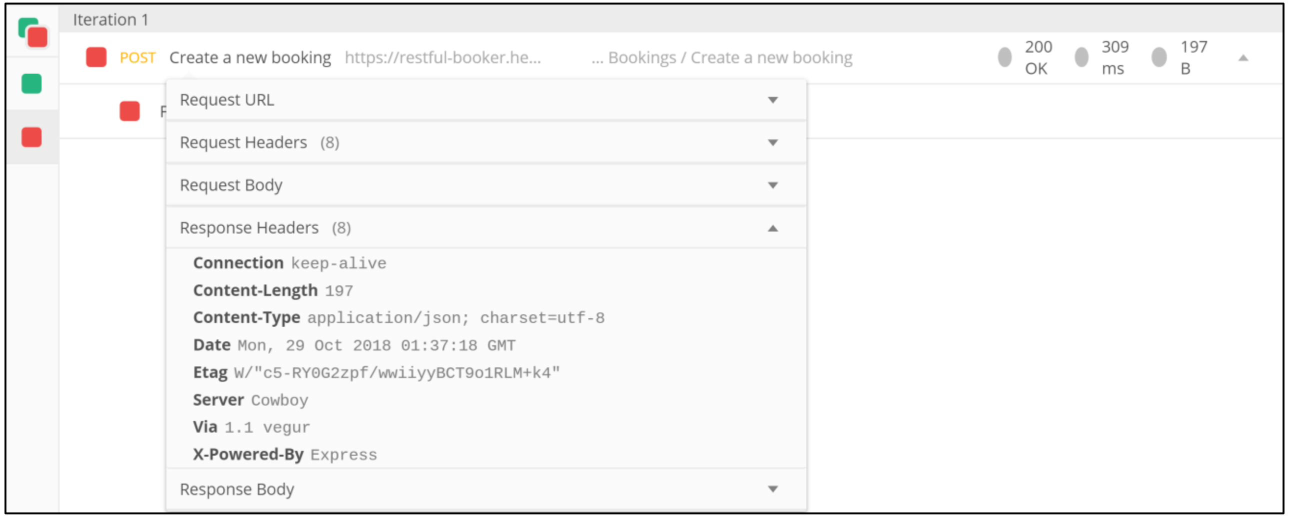
Iteration 1

🟩	GET	/booking	https://restful-booker.he...	...stful Booker / Bookings / /booking	200 OK	1308 ms	162 B
🟩	PASS	Status code is 200					
🟥	POST	Create a new booking	https://restful-booker.he...	... Bookings / Create a new booking	200 OK	306 ms	197 B
🟥	FAIL	Body matches string					
🟩	POST	/auth	https://restful-booker.he...	Restful Booker / Auth / /auth	200 OK	277 ms	28 B
🟩	PASS	Status code is 200					

POSTMAN

Drill down in Results

- Can inspect the request and response details for each Request in the collection run
- Useful for troubleshooting failed requests

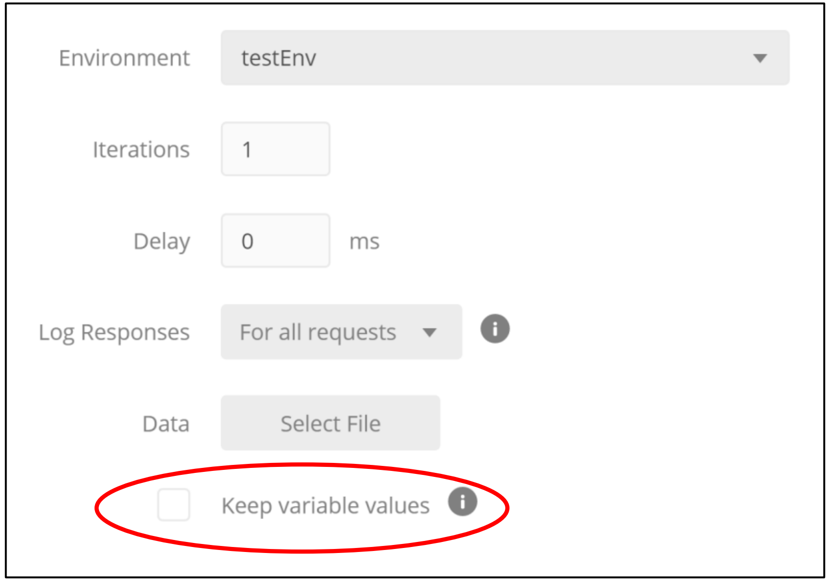


Connection keep-alive
Content-Length 197
Content-Type application/json; charset=utf-8
Date Mon, 29 Oct 2018 01:37:18 GMT
Etag W/"c5-RY0G2zpf/wwiiyyBCT9o1RLM+k4"
Server Cowboy
Via 1.1 vegur
X-Powered-By Express

POSTMAN

Using variables in a collection run

- Select environment to use the variable values in a collection run
- “Keep variable values” will preserve the “current value” of variables if they are changed during the collection run



Environment	testEnv
Iterations	1
Delay	0 ms
Log Responses	For all requests
Data	Select File
☐	Keep variable values

POSTMAN

EXERCISE

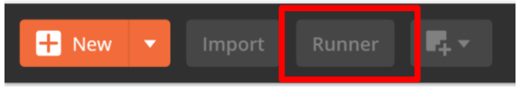
Ex1.1 – Run a collection

1. Import the sample Ex1.1 collection and environment provided by the instructor. You should have a collection called "Using Environments" and an environment called "testEnv"
2. Open the request in the "**Using Environments**" collection
3. Select the "**testEnv**" environment on the request builder
4. Click on the "**Environment quick look**" button
5. Check the current value of the "**foo**" variable. It should be "**bar**"
6. Send the request and then check the current value of the "**foo**" variable again. Can you identify the cause of the change in the value?
7. Change the current value of "foo" back to "bar"

POSTMAN

EXERCISE

Ex1.1 – Run a collection (cont’d)



8. Click the “Runner” button
9. Select the “Using Environments” collection and the “testEnv” environment
10. Run the collection and then check the value of the “foo” variable. What do you notice?
11. Start a new Collection run using the same settings as before but this time, tick the “Keep variable values” option
12. What do you notice about the current value of the “foo” variable afterwards?

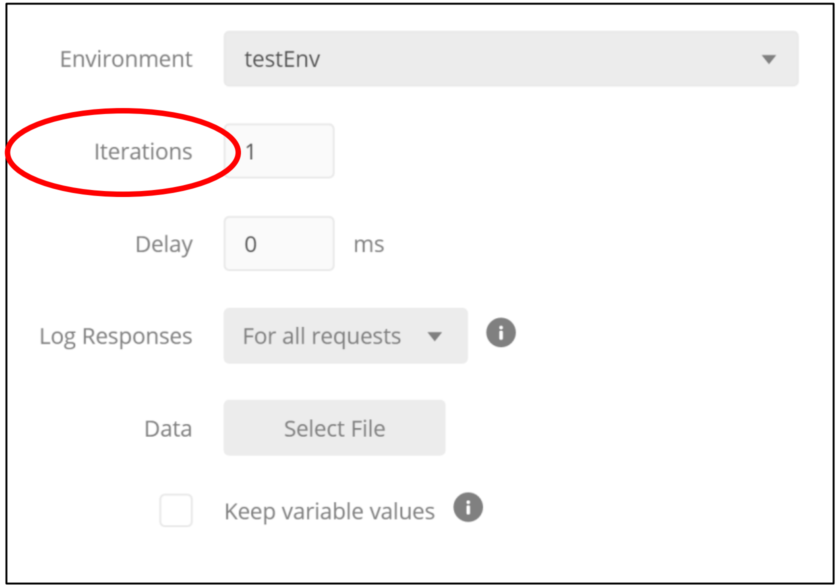
POSTMAN

Multiple iterations run

POSTMAN

Running multiple iterations

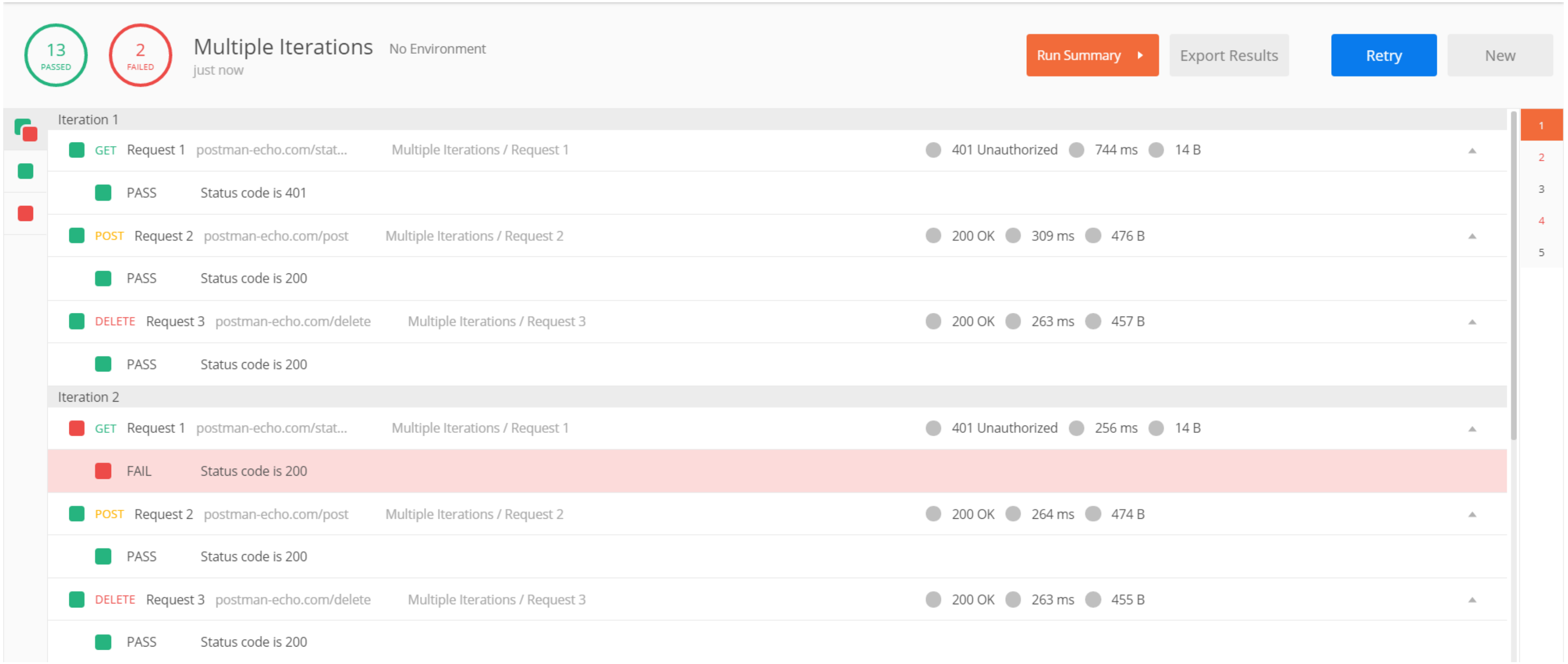
- An iteration represents the number of times a collection will run
- Run multiple iterations to test for consistent behaviour



Environment	testEnv
Iterations	1
Delay	0 ms
Log Responses	For all requests
Data	Select File
☐	Keep variable values

POSTMAN

Multiple iteration results



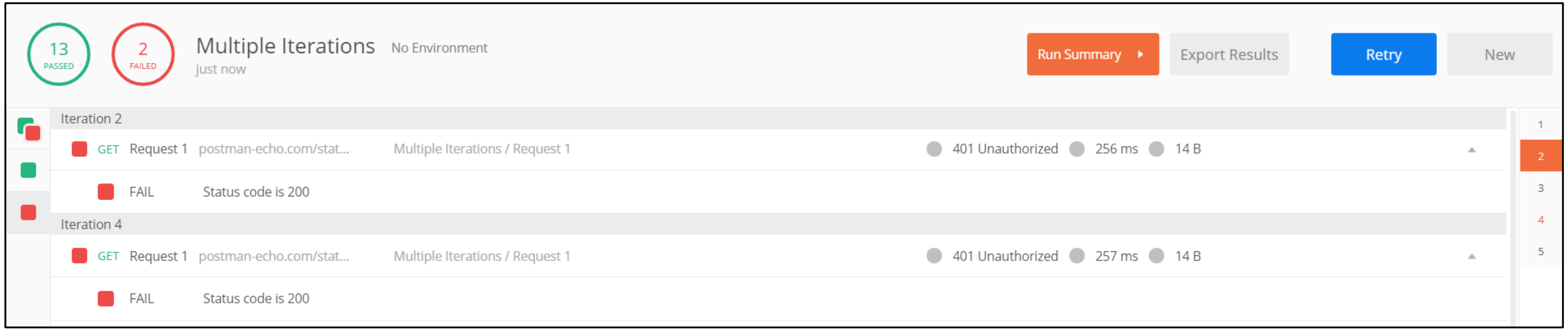
13 PASSED | 2 FAILED | Multiple Iterations | No Environment | just now

```
Iteration 1
GET     Request 1  postman-echo.com/stat...    Multiple Iterations / Request 1   401 Unauthorized  744 ms  14 B
        PASS       Status code is 401
POST    Request 2  postman-echo.com/post       Multiple Iterations / Request 2   200 OK  309 ms  476 B
        PASS       Status code is 200
DELETE  Request 3  postman-echo.com/delete     Multiple Iterations / Request 3   200 OK  263 ms  457 B
        PASS       Status code is 200
Iteration 2
GET     Request 1  postman-echo.com/stat...    Multiple Iterations / Request 1   401 Unauthorized  256 ms  14 B
        FAIL       Status code is 200
POST    Request 2  postman-echo.com/post       Multiple Iterations / Request 2   200 OK  264 ms  474 B
        PASS       Status code is 200
DELETE  Request 3  postman-echo.com/delete     Multiple Iterations / Request 3   200 OK  263 ms  455 B
        PASS       Status code is 200
```

POSTMAN

Using red and green filters

- Useful for filtering results on large collections
- A request where one or more test cases failed will be marked red



Multiple Iterations No Environment
just now

Iteration 2

🟥 GET Request 1 postman-echo.com/stat... Multiple Iterations / Request 1 401 Unauthorized 256 ms 14 B

🟥 FAIL Status code is 200

Iteration 4

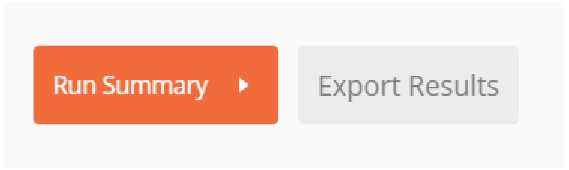
🟥 GET Request 1 postman-echo.com/stat... Multiple Iterations / Request 1 401 Unauthorized 257 ms 14 B

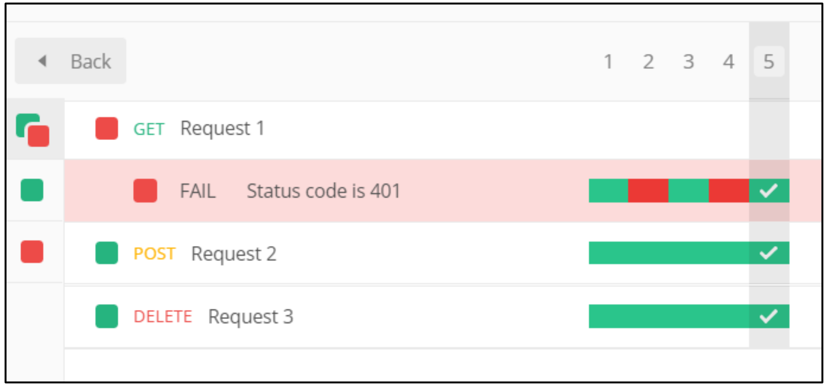
🟥 FAIL Status code is 200

POSTMAN

Run summary

- Shows each request and iteration in a timeline format
- Can easily see which request in the collection failed and which iteration the request failed in





POSTMAN

Uploading a data file

POSTMAN

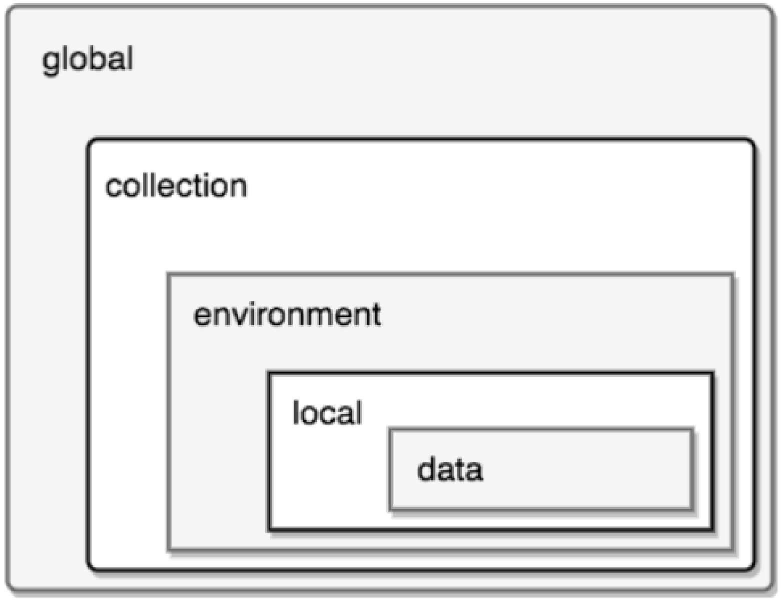
Using a data file

- Upload a data file and use the values defined in the file in your HTTP requests and scripts
- Supported formats are CSV and JSON
- Each record in the data file is treated as a separate iteration
- Benefits:
    - ➢ More robust testing of API's by testing for many variations in request parameters
    - ➢ Can be used to perform database initialization and
    - ➢ Setup and teardown for testing

POSTMAN

Data variables

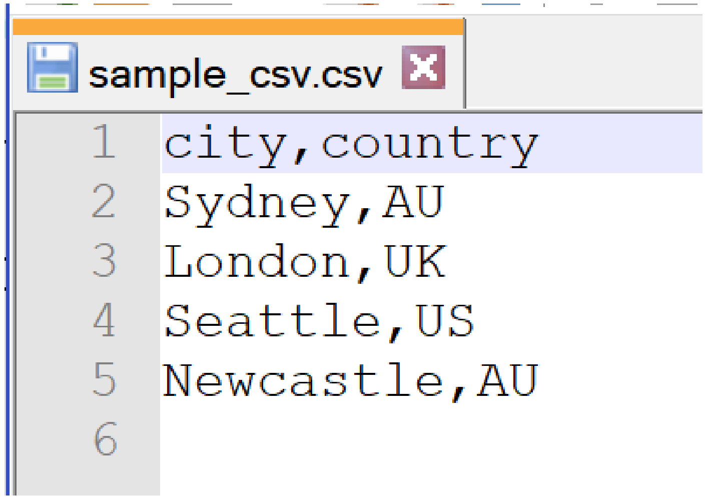
- Values from an uploaded CSV or JSON file during a Collection run are treated as data variables
- Can be used in the same manner as global, collection and environment variables



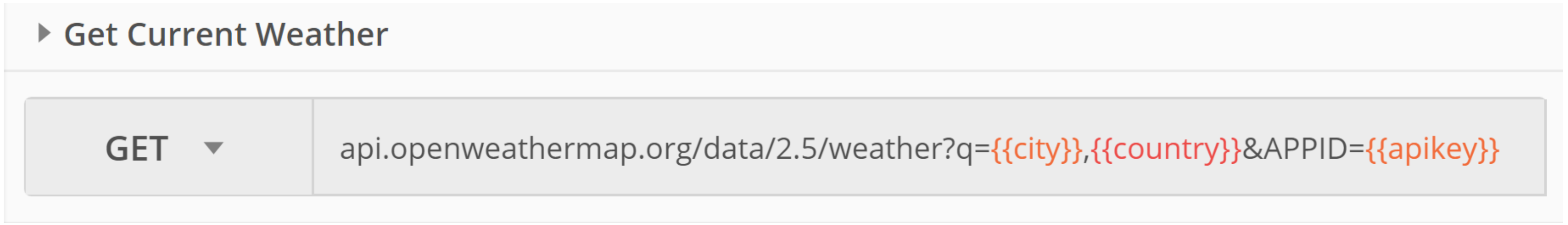
POSTMAN

CSV file format

- First row contains the variable names to be used in the request
- Every row after is used as a data row
- Rows should have the same number of columns


sample_csv.csv

```
city,country
Sydney,AU
London,UK
Seattle,US
Newcastle,AU
```

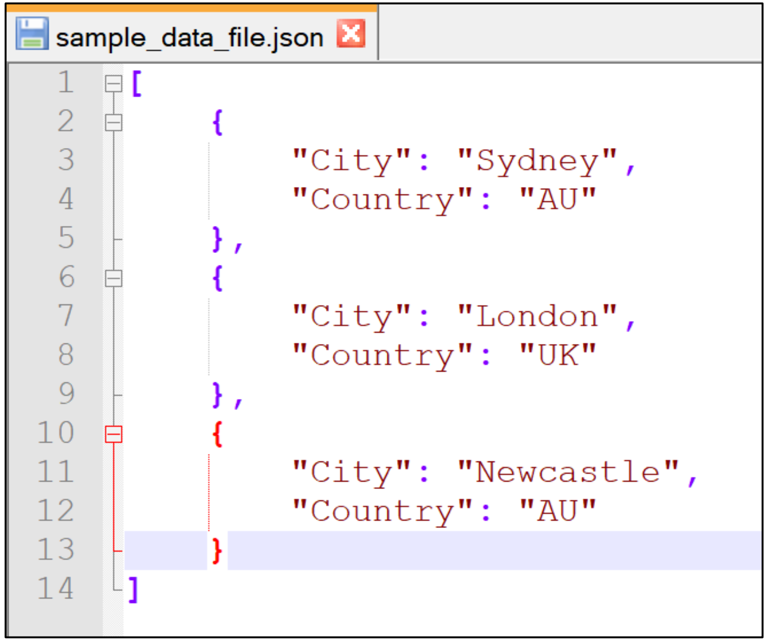

Get Current Weather

GET api.openweathermap.org/data/2.5/weather?q={{city}},{{country}}&APPID={{apikey}}

POSTMAN

JSON file format

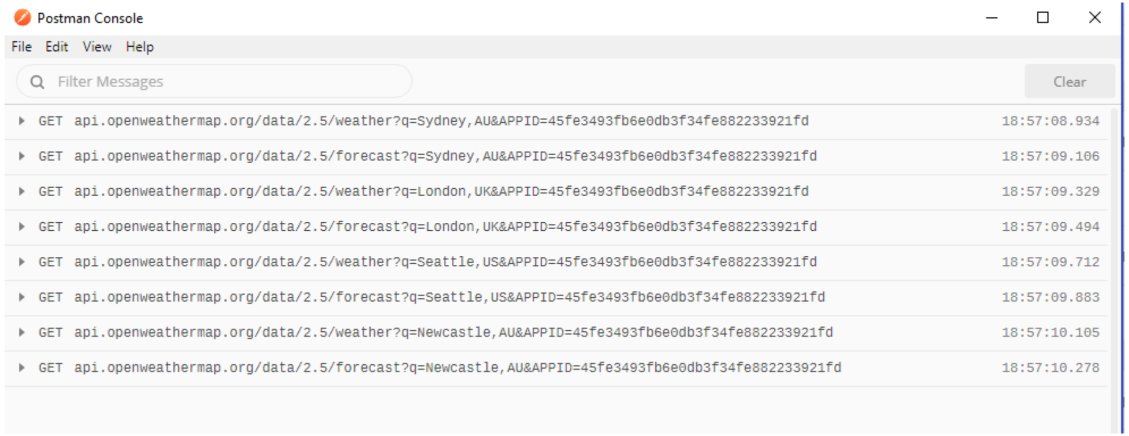
- Array of key value pairs
- Each key represents the name of the variable


sample_data_file.json

```
[
	{
		"City": "Sydney",
		"Country": "AU"
	},
	{
		"City": "London",
		"Country": "UK"
	},
	{
		"City": "Newcastle",
		"Country": "AU"
	}
]
```

Debugging requests

- Drill down on the results to inspect the request and response headers and body
- Use the Postman console
- Console must be open prior to running the collection



```
▸ GET  api.openweathermap.org/data/2.5/weather?q=Sydney,AU&APPID=45fe3493fb6e0db3f34fe882233921fd      18:57:08.934
▸ GET  api.openweathermap.org/data/2.5/forecast?q=Sydney,AU&APPID=45fe3493fb6e0db3f34fe882233921fd     18:57:09.106
▸ GET  api.openweathermap.org/data/2.5/weather?q=London,UK&APPID=45fe3493fb6e0db3f34fe882233921fd      18:57:09.329
▸ GET  api.openweathermap.org/data/2.5/forecast?q=London,UK&APPID=45fe3493fb6e0db3f34fe882233921fd     18:57:09.494
▸ GET  api.openweathermap.org/data/2.5/weather?q=Seattle,US&APPID=45fe3493fb6e0db3f34fe882233921fd     18:57:09.712
▸ GET  api.openweathermap.org/data/2.5/forecast?q=Seattle,US&APPID=45fe3493fb6e0db3f34fe882233921fd    18:57:09.883
▸ GET  api.openweathermap.org/data/2.5/weather?q=Newcastle,AU&APPID=45fe3493fb6e0db3f34fe882233921fd   18:57:10.105
▸ GET  api.openweathermap.org/data/2.5/forecast?q=Newcastle,AU&APPID=45fe3493fb6e0db3f34fe882233921fd  18:57:10.278
```

POSTMAN

Newman

POSTMAN

Running collections with Newman

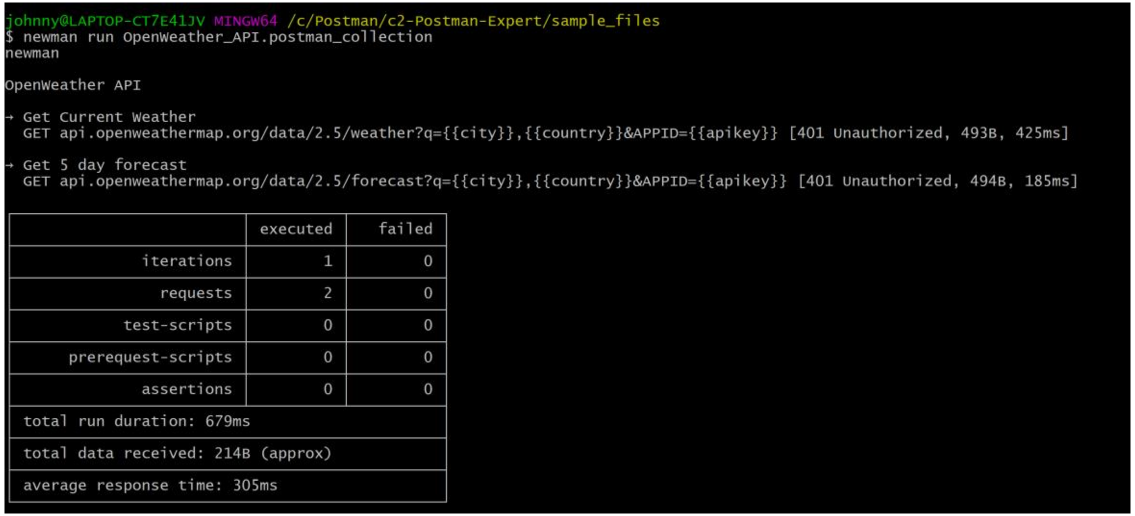
- Newman is the command line tool used to run Postman collections
- Built on Node.js
- Easily integrated with CI/CD servers
- Maintains feature parity with the Postman App
- For more detailed information and examples of usage, see https://github.com/postmanlabs/newman

POSTMAN

Running a Collection with Newman

- Need to export the Collection and any environment it uses to a JSON file
- Run Newman against the file

```
newman run <collection file>
```



```
johnny@LAPTOP-CT7E41JV MINGW64 /c/Postman/c2-Postman-Expert/sample_files
$ newman run OpenWeather_API.postman_collection
newman

OpenWeather API

→ Get Current Weather
  GET api.openweathermap.org/data/2.5/weather?q={{city}},{{country}}&APPID={{apikey}} [401 Unauthorized, 493B, 425ms]

→ Get 5 day forecast
  GET api.openweathermap.org/data/2.5/forecast?q={{city}},{{country}}&APPID={{apikey}} [401 Unauthorized, 494B, 185ms]

┌─────────────────────────┬──────────┬──────────┐
│                         │ executed │   failed │
├─────────────────────────┼──────────┼──────────┤
│              iterations │        1 │        0 │
├─────────────────────────┼──────────┼──────────┤
│                requests │        2 │        0 │
├─────────────────────────┼──────────┼──────────┤
│            test-scripts │        0 │        0 │
├─────────────────────────┼──────────┼──────────┤
│      prerequest-scripts │        0 │        0 │
├─────────────────────────┼──────────┼──────────┤
│              assertions │        0 │        0 │
├─────────────────────────┴──────────┴──────────┤
│ total run duration: 679ms                     │
├───────────────────────────────────────────────┤
│ total data received: 214B (approx)            │
├───────────────────────────────────────────────┤
│ average response time: 305ms                  │
└───────────────────────────────────────────────┘
```

POSTMAN

Newman run parameters

Display run options

```
newman run -h
```

Run the collection using an environment (environment must first be exported)

```
newman run <collection_file> -e <environment_file.json>
```

Run the collection over 10 iterations

```
newman run <collection_file> -n 10
```

POSTMAN

Module Review

- Collections can be run via the Postman app collection runner or via Newman
- Changes in variable values during a collection run, can be persisted
- Using a data file allows us to run a collection over many iterations to test for as many variations in input as needed

Module 2 – Advanced Scripting

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Describe the difference between pre-request and test scripts
- Describe the execution order of Postman scripts
- Write a pre-request script
- Set the script execution order to create a workflow

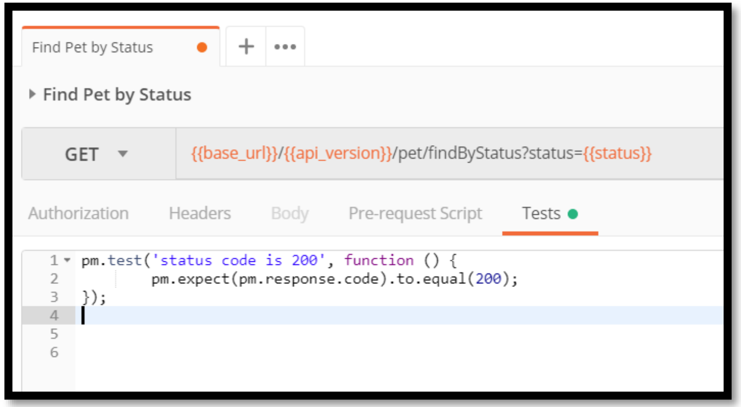
POSTMAN

Intro to Scripting

POSTMAN

Recall: Postman API Testing

- JavaScript code can be executed after receiving the response to a request
- Access request, response, and variable data
- Postman Sandbox API provides a set of JavaScript variables and functions to access the request and response data
- The **Tests Results** tab in the response area to show the results of each test



Find Pet by Status

Find Pet by Status

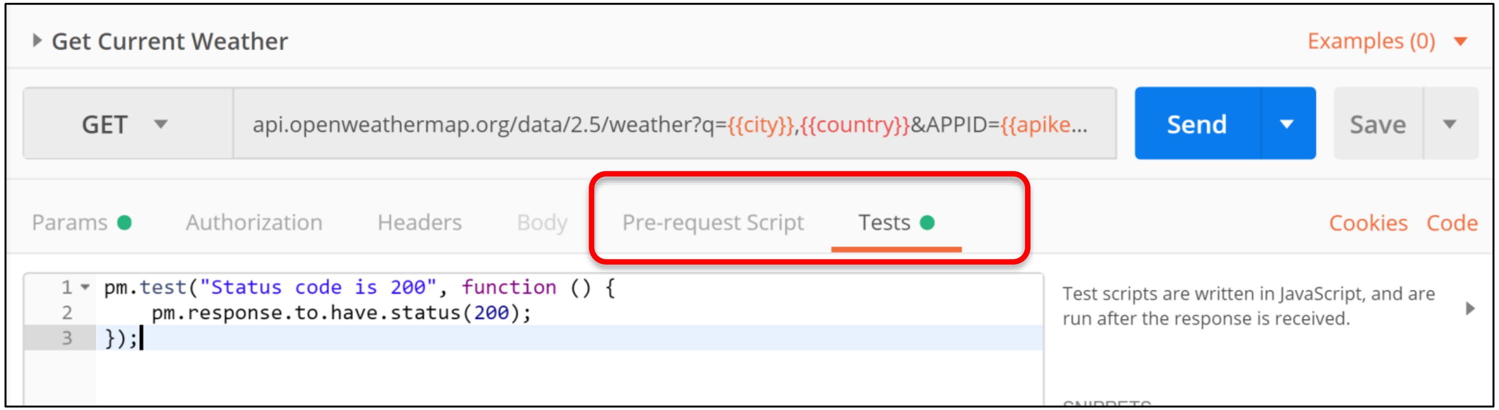
GET {{base_url}}/{{api_version}}/pet/findByStatus?status={{status}}

```
pm.test('status code is 200', function () {
        pm.expect(pm.response.code).to.equal(200);
});
```

POSTMAN

Types of Postman scripts

- Pre-request script
  - Executed before the postman request is sent
- Test script
  - Executed after the response has been received



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

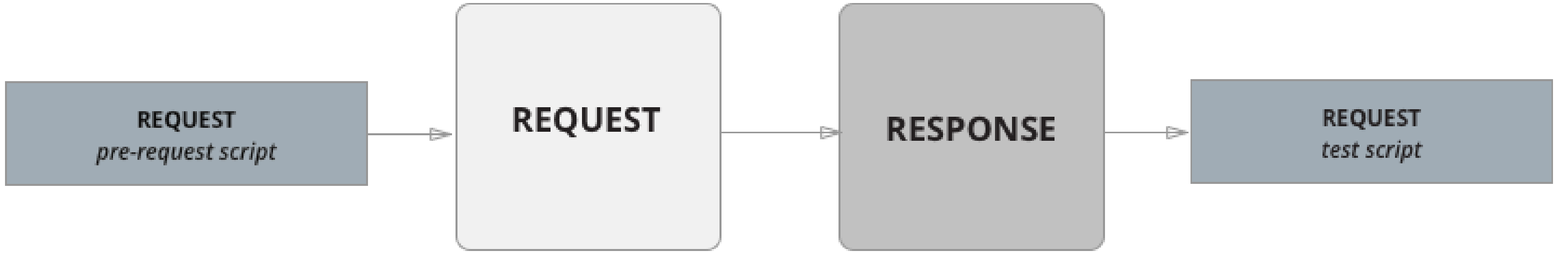
Test scripts are written in JavaScript, and are run after the response is received.

POSTMAN

Script execution order

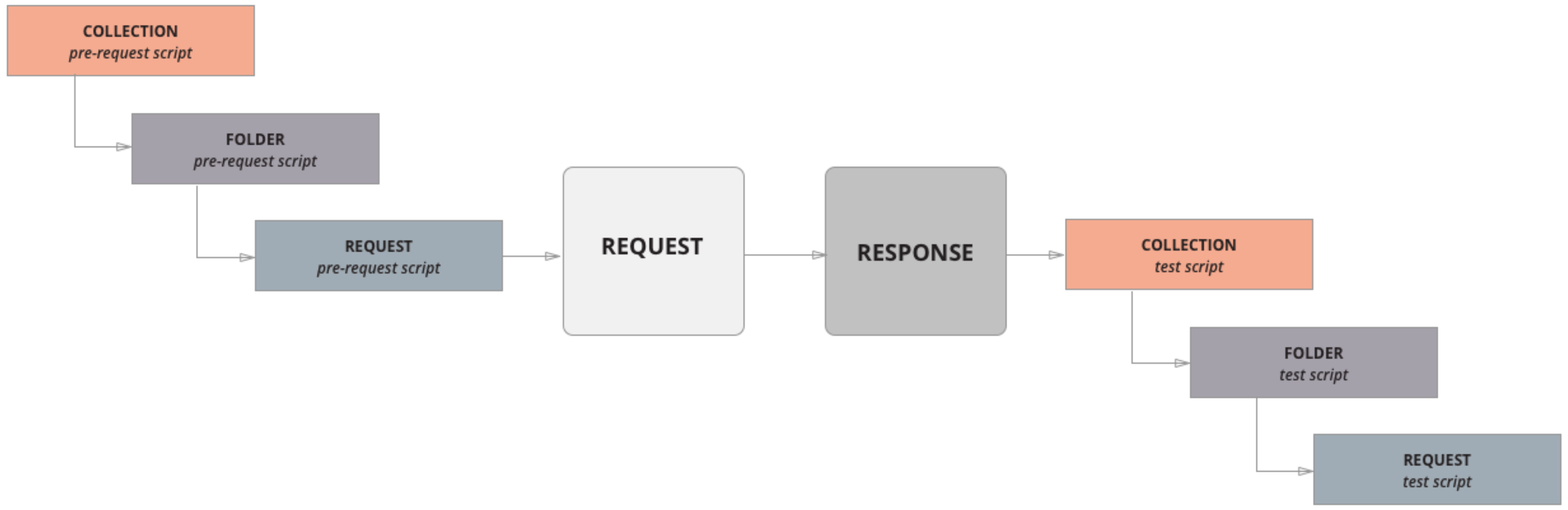
For an individual request:

- Pre-request scripts are executed before the request is sent
- Test scripts executes after the request is sent



POSTMAN

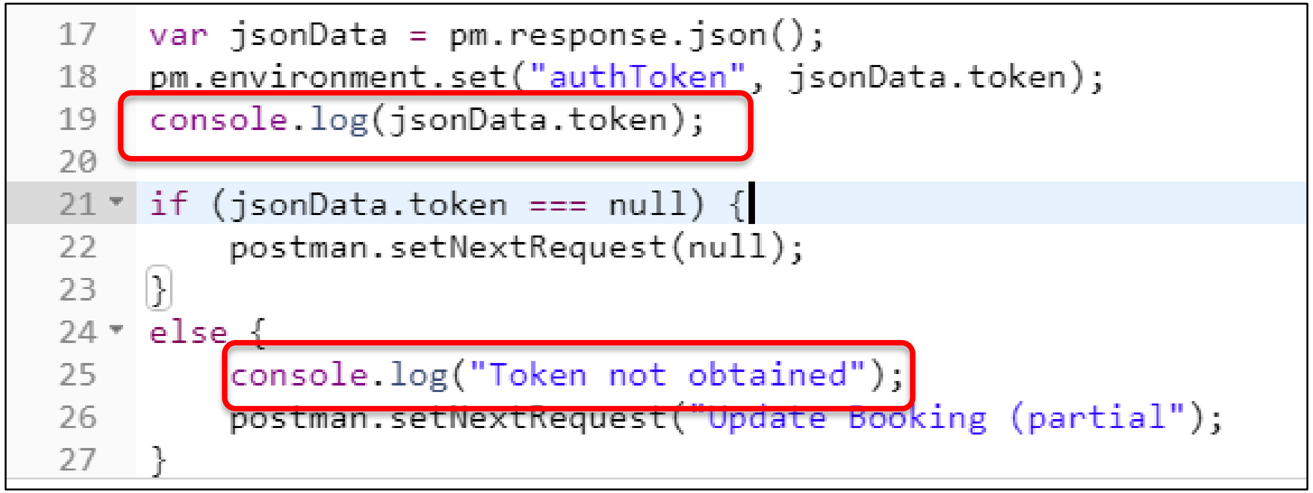
Collection script execution order



POSTMAN

Script debugging

- Use `console.log()` function to log values to the console
- Useful for inspect variable and response data and script flow
- Postman Console must be open before running the request



```
var jsonData = pm.response.json();
pm.environment.set("authToken", jsonData.token);
console.log(jsonData.token);

if (jsonData.token === null) {
    postman.setNextRequest(null);
}
else {
    console.log("Token not obtained");
    postman.setNextRequest("Update Booking (partial");
}
```

Pre-request scripts

POSTMAN

Writing a pre-request script

- Written in JavaScript
- Same as test scripts but without access to the response object
- Typically used to manipulate data before a request is sent
  - Set variable values based on a function call
  - Add headers such as timestamp

POSTMAN

EXERCISE

EX2.1 – Write a pre-request script

1. Create a Collection called “Forex”
2. Create a new request with the following URL: https://api.exchangeratesapi.io/{{date}}?base={{currency}} and save it to the Forex collection. Name the request “Get Historical Exchange Rate”
3. Create an environment called “Forex” with the “currency” and “date” variables defined
4. Set the initial and current value of the “currency” variable to “**USD**”
5. Set a placeholder value for the “date” variable

POSTMAN

EXERCISE

EX2.1 – (cont’d)

6. Switch to the “Pre-request” tab and write the following script:

```
const moment = require('moment’);
var previousWeek = moment().subtract(7, 'days').format('YYYY-MM-DD’);
pm.environment.set("date", previousWeek);
```

7. Run the request with the “Forex” environment and check the date on the response body. The date we should see should be exactly 7 days ago from today.

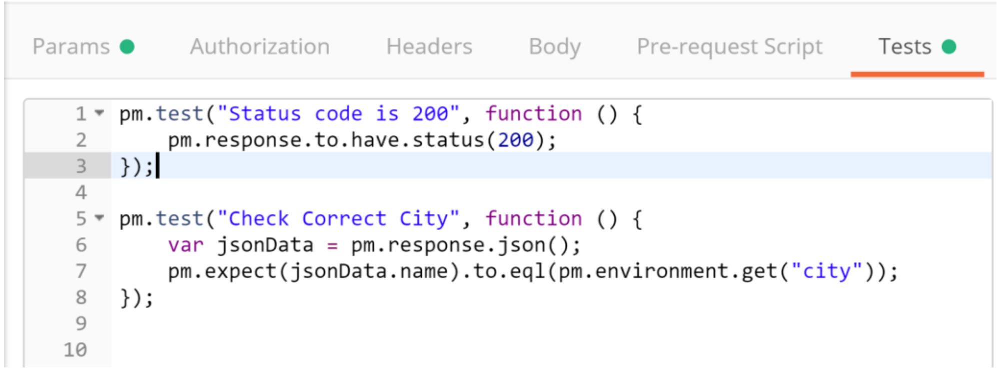
POSTMAN

Test Scripts

POSTMAN

Recall – Examples of Test Scripts

- Many types of tests are available in snippets
  - Check response code
  - Check JSON value
  - Check response headers and body



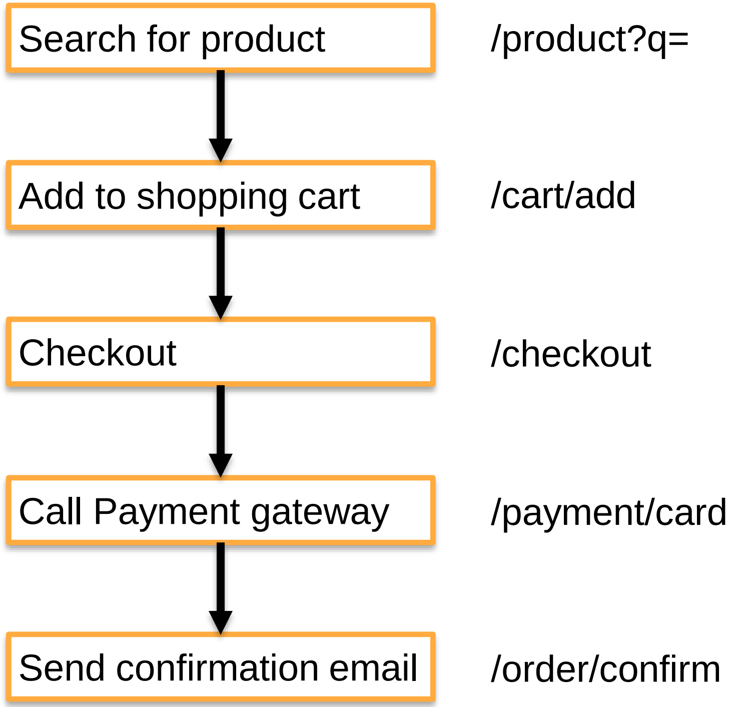
```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Check Correct City", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql(pm.environment.get("city"));
});
```

Building Workflows

Intro to Workflows

- A workflow is a sequence of requests in a collection, which represents a particular user story / application feature
- Each sequence in the workflow is a separate API call
- **For example: Purchase an item from a store**



Controlling the request order

- Requests in a collection are executed in linear order (in the order that you see them listed)
- To build a workflow, we must control the sequencing of our requests.
- Use the `postman.setNextRequest()` function in our test script
  - postman.setNextRequest(“request name”);
  - postman.setNextRequest(null);

POSTMAN

postman.setNextRequest() function

- Can be used in pre-request or test script
  - Last set value is considered
- Always executed at the end of the current script.
- Function scope is limited to the source of your collection run
  - If running from Collection level, you can set any request as the next request
  - If running from a folder level, you can only set a request in the same folder as the next request
- If function is not present in any script in a request, the collection runner defaults to the next request in linear order

POSTMAN

Our example workflow

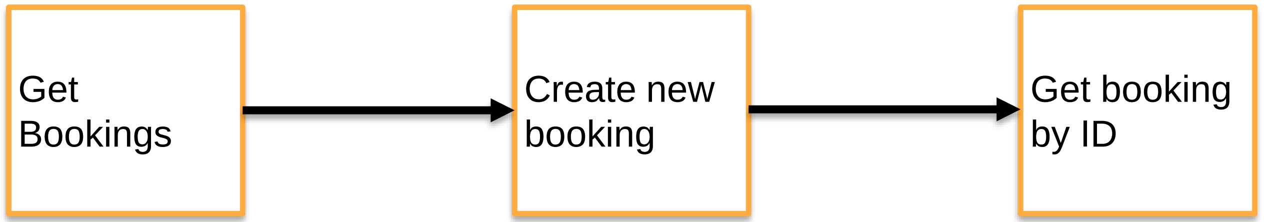
- Restful booker API hosted on a server
- Instructor will provide server URL
- Collection is already defined and ready for import

Get Bookings	/booking
Create new booking	/booking
Update Booking	/booking/{{id}}
Get booking by ID	/booking/{{id}}
Delete booking	/booking/{{id}}
Get auth token	/auth

POSTMAN

Building our example workflow (part 1)

Step 1 **Step 2** **Step 3**



Get list of bookings

Create new hotel booking for “Sally”.

- Returns bookingID (_Id field)

Use the bookingID from step 2, to check
That the new booking is in the system

POSTMAN

EXERCISE

EX2.2 – Build Example Workflow (part 1)

1. Open the **Restful Booker** collection
2. Create an environment call **Restful Booker** and define a variable called “url”. Your instructor will provide the URL value.
3. In the same environment, define a variable called “bookingID”. Do not set a value
4. Open the **Create new booking** request inside the **Restful Booker** collection.
5. Go to the “Tests” tab and add the following lines at the end:

```
var bookingID = jsonData._Id;
console.log(bookingID);
pm.environment.set("bookingID", bookingID);
postman.setNextRequest("Get Booking by ID");
```

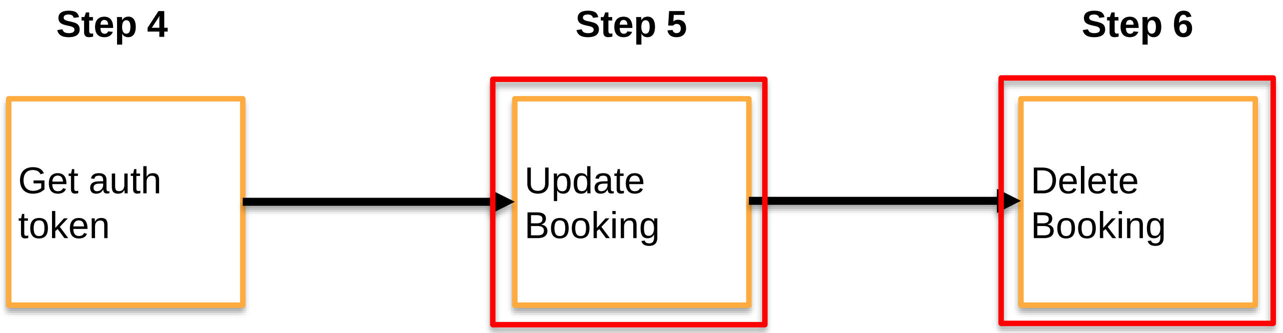
POSTMAN

EXERCISE

EX2.2 – (cont’d)

6. Open the Postman Console
7. Select the **Restful Booker** environment
8. Run the **Create New Booking** request. Take note of the booking ID in the JSON response data and also in the Console log
9. Open the **Restful Booker** environment and check the current value of the booking ID. Does this match the value in step 4?
10. Run the **Get Booking by ID** request. You should get the same record returned as the one you just created.

Building our example workflow (part 2)



Get auth token and store in environment variable

Change the name of the booking from "Sally" to "James".

- Returns updated record

Requires auth token

Use the bookingID from step 2, to delete the booking

Requires auth token

POSTMAN

EXERCISE

EX2.3 - Build Example Workflow (part 2)

1. Open the **Get Booking by ID** request and go to the “Tests” tab
2. Write a line to set the next request to **Get Auth Token**
3. Open the **Restful Booker** environment and define a new variable called “authToken”. Set a placeholder value in the Initial Value field
4. Open the **Get Auth Token** request and go to the “Tests” tab
5. Add the following lines:

   ```
   var jsonData = pm.response.json();
   pm.environment.set("authToken", jsonData.token);
   ```
6. Add a line to set the next request to **Update Booking**

POSTMAN

EXERCISE

EX2.3 – (cont'd)

7. In the **Restful Booker environment,** define a variable called "password" and set its current value to "password123"
8. Run the **Get Auth Token** request. Check the token on the response and then check the value of the "authToken" environment variable. They should match
9. Open the **Update Booking** request and go to the "Headers" tab
10. Add the "Cookie" header with value of "`token={{authToken}}`"
11. Go to the "Tests" tab and set the next request to Delete Booking
12. Open the **Delete Booking** request and repeat Step 10

POSTMAN

EXERCISE

EX2.3 – (cont’d)

12. Run the **Update Booking** request. You should see that the booking name is now “James”
13. Confirm step 12 by running **Get booking by ID** again
14. Run the **Delete Booking** request
15. Run **Get booking by ID** again. This time you should get a response saying “Not Found”

Our example workflow (correct order)

- Now we are ready to run the collection

Get Bookings	/booking
Create new booking	/booking
Get booking by ID	/booking/{{id}}
Get auth token	/auth
Update Booking	/booking/{{id}}
Delete booking	/booking/{{id}}

POSTMAN

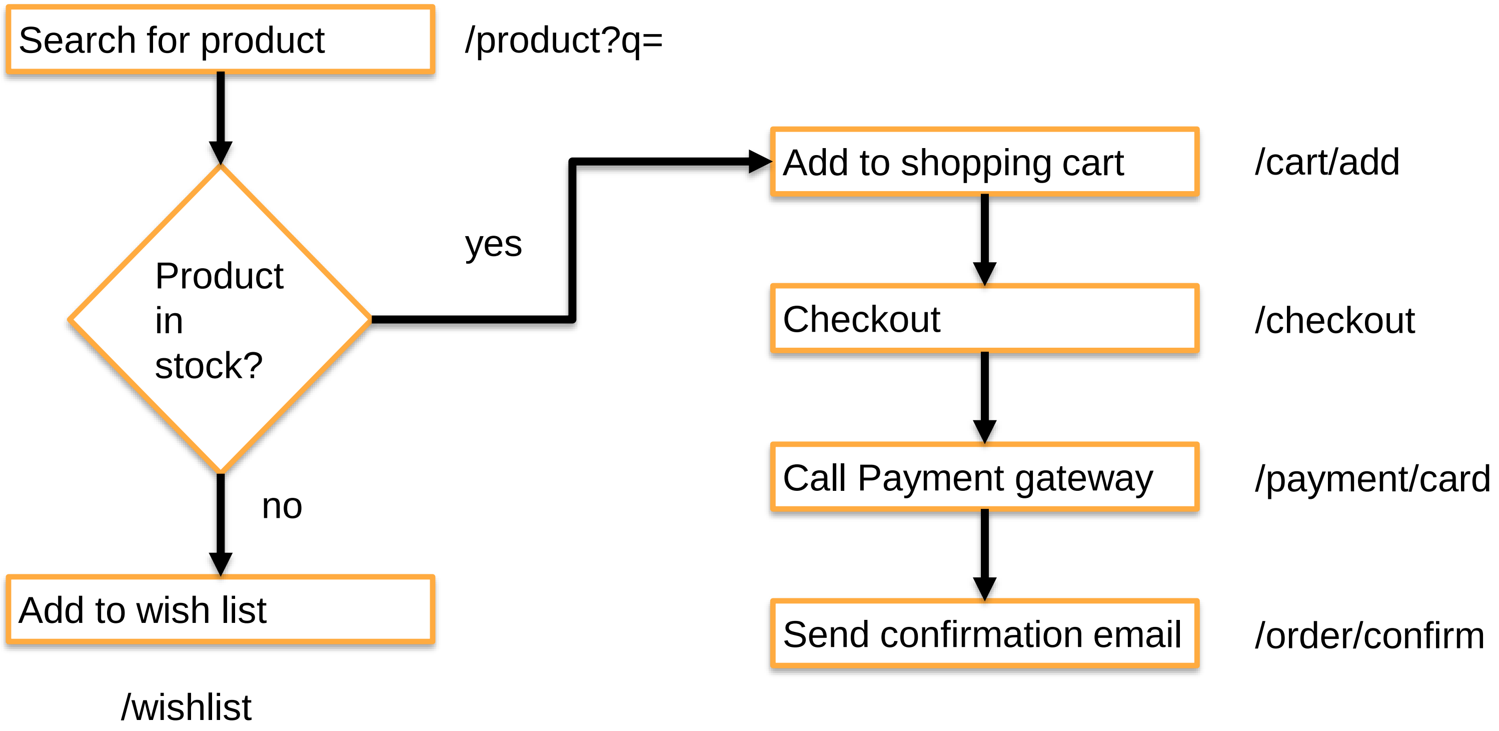
EXERCISE

EX2.4 – Run workflow

1. Open the Collection Runner
2. Select the **Restful Booker** collection
3. Select the **Restful Booker** environment
4. Click Run. What do you notice? Why do you think this is occurring?
5. Open the **Delete Booking** request from the collection and go to the "Tests" tab
6. Add the line

   ```
   postman.setNextRequest(null);
   ```
7. Run the Collection again. What do you observe this time?

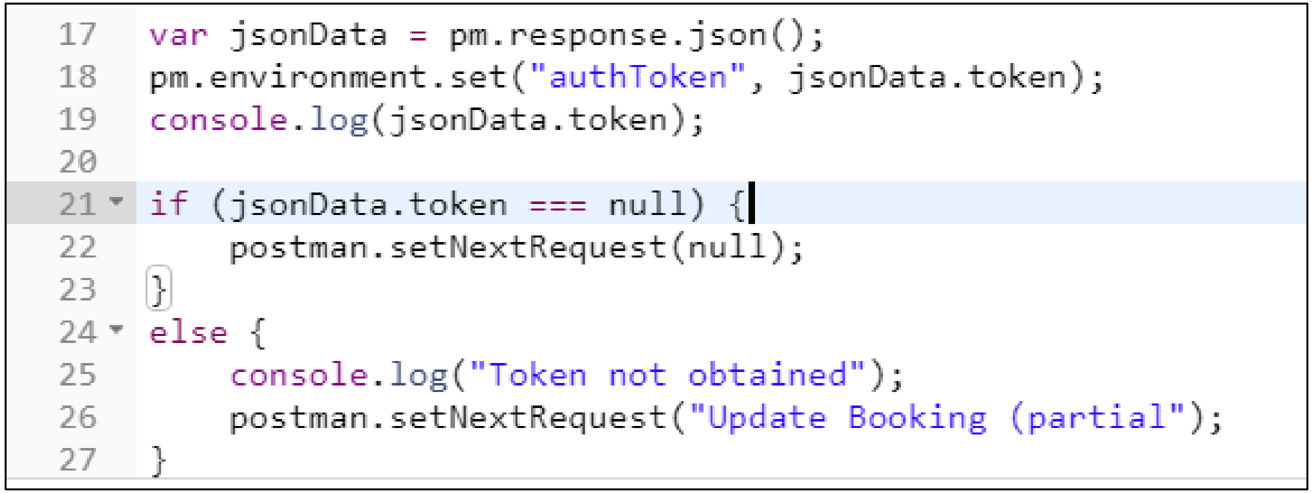
Workflow conditional logic



POSTMAN

Implementing conditional logic

- Use ‘if’ statement to check a condition and decide which request to execute next

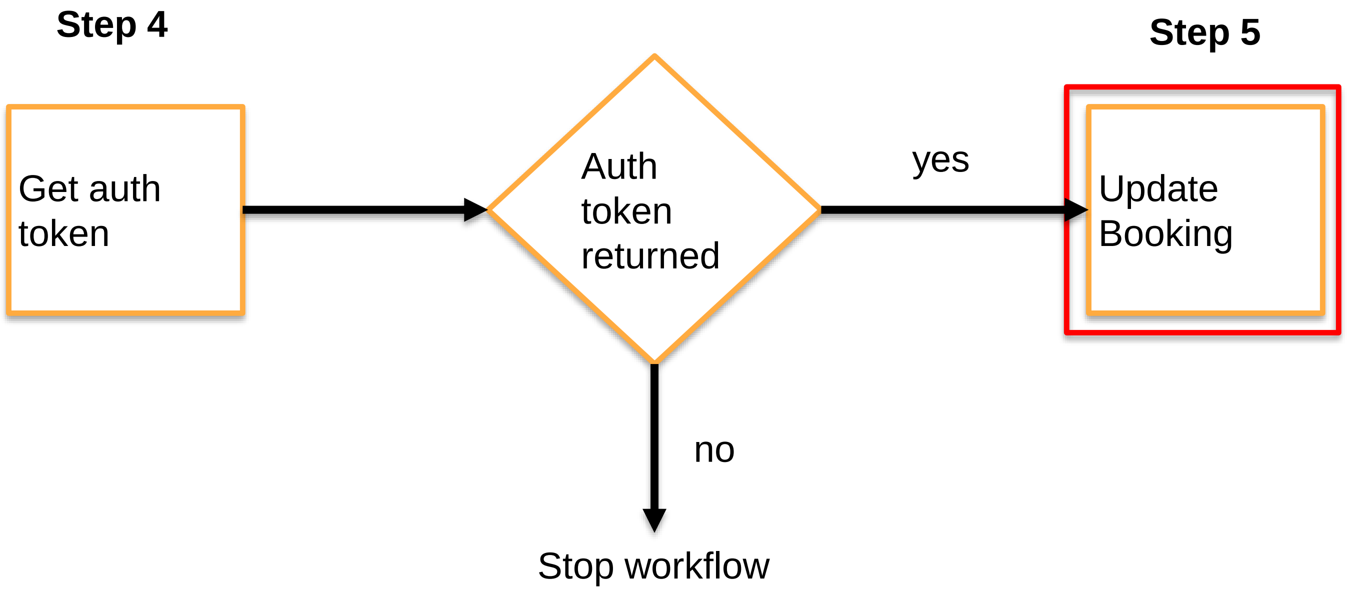


```
var jsonData = pm.response.json();
pm.environment.set("authToken", jsonData.token);
console.log(jsonData.token);

if (jsonData.token === null) {
    postman.setNextRequest(null);
}
else {
    console.log("Token not obtained");
    postman.setNextRequest("Update Booking (partial");
}
```

EXERCISE

EX2.5 – Implement condition logic



POSTMAN

EXERCISE

EX2.5 - (cont’d)

1. Open the **Get Auth Token** request and replace the “`postman.setNextRequest`” line with the following:

```
if (jsonData.token === undefined) {
    postman.setNextRequest(null);
} else {
    postman.setNextRequest("Update Booking (partial)”;
}
```

2. Change the value of the password variable to anything and run the collection.
3. What do you observe?

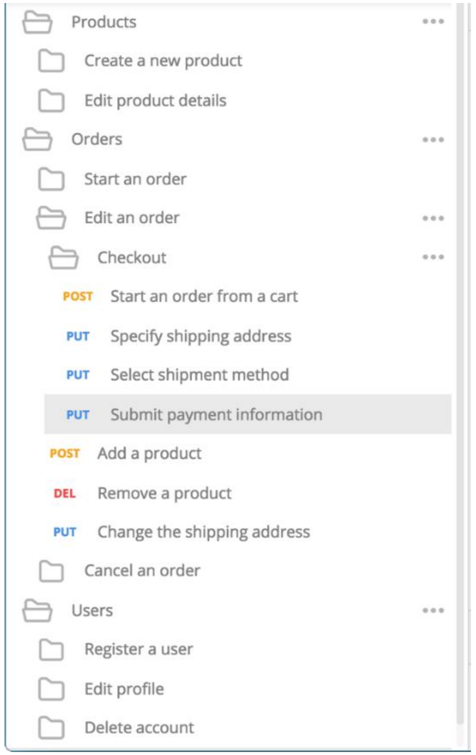
POSTMAN

Postman Sandbox API

- The JavaScript execution environment for pre-request and test scripts
- Gives access to the request, response and variable objects
- Functions from common utility libraries such as
  - cheerio
  - tv4 JSON schema validation
- https://www.getpostman.com/docs/v6/postman/scripts/postman_sandbox
- API reference at https://www.getpostman.com/docs/v6/postman/scripts/postman_sandbox_api_reference

Organizing collections

- Suggestions for organizing requests into folders
- Three level folder hierarchy
- Group API resources in top level folder (users, orders, products etc…)
- Second level folder contains test suites for those resources
  - Create new user
  - Submit order
- Third level folder for complex tests which require a workflow through multiple requests



Products
Create a new product
Edit product details
Orders
Start an order
Edit an order
Checkout
POST Start an order from a cart
PUT Specify shipping address
PUT Select shipment method
PUT Submit payment information
POST Add a product
DEL Remove a product
PUT Change the shipping address
Cancel an order
Users
Register a user
Edit profile
Delete account

Module Review

Key Points:

- Pre-request scripts are effective for manipulating data
- The response of a request can be used as input in a subsequent request
- Scripts at the collection level are executed first, followed by the folder level and then individual request

POSTMAN

Module 3 – API Documentation

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Create public and private documentation for your API
- Outline the content that is automatically generated in documentation
- Write documentation content using markdown

POSTMAN

Creating Private Documentation

POSTMAN

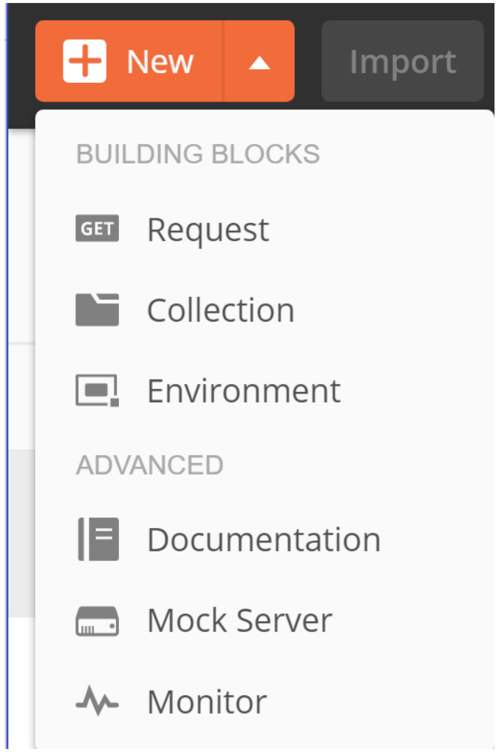
Intro to API documentation

- Postman can generate and host web based documentation for your API
- Documentation is based on the requests defined in a collection
- Markdown support for formatting
- Private view for you and your team
- Publish docs to make them visible to the public

POSTMAN

Creating Documentation

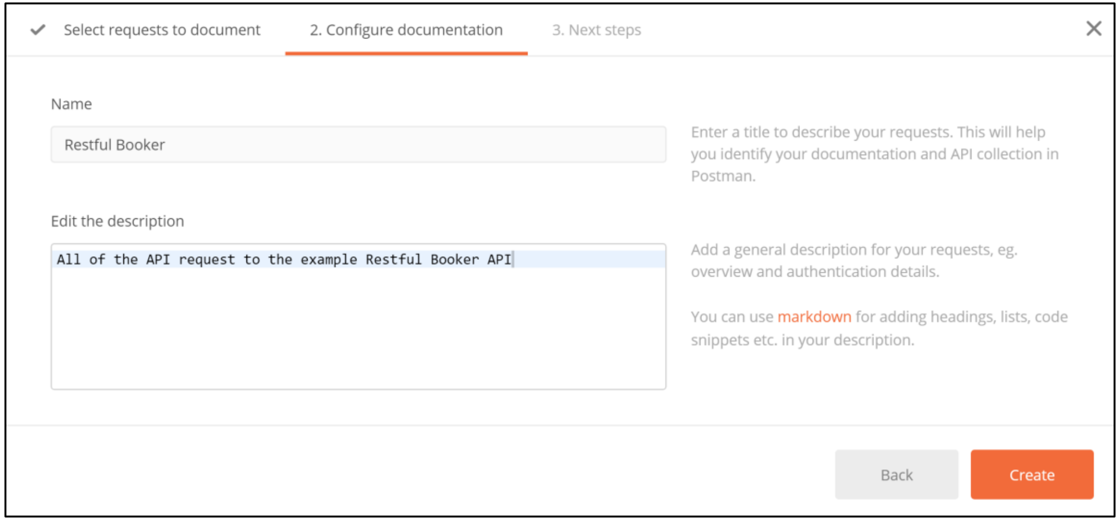
- Different ways to create documentation
  - New button
  - View the private documentation of an existing collection
  - Publishing public documentation of an existing collection
  - From the Postman app launch screen



POSTMAN

Create documentation from New Button

- Choose the “Use Collection from this workspace” tab
- Select the collection
- Configure name and description and click “Create”



Name

Restful Booker

Enter a title to describe your requests. This will help you identify your documentation and API collection in Postman.

Edit the description

All of the API request to the example Restful Booker API

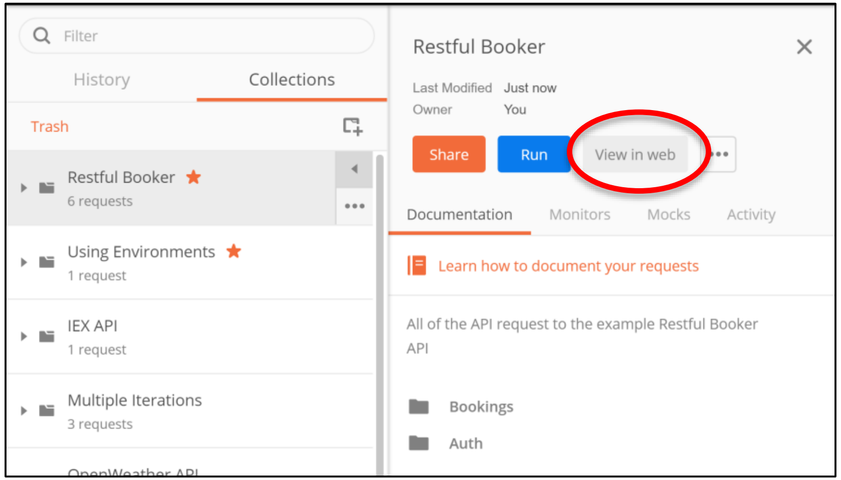
Add a general description for your requests, eg. overview and authentication details.

You can use markdown for adding headings, lists, code snippets etc. in your description.

POSTMAN

Viewing private documentation

- Click the “Play” button of a collection
- Click “View in web”
- If there’s no existing documentation on the collection, it will be created
- Visible only to author of collection unless the collection is shared in a team workspace



All of the API request to the example Restful Booker API

POSTMAN

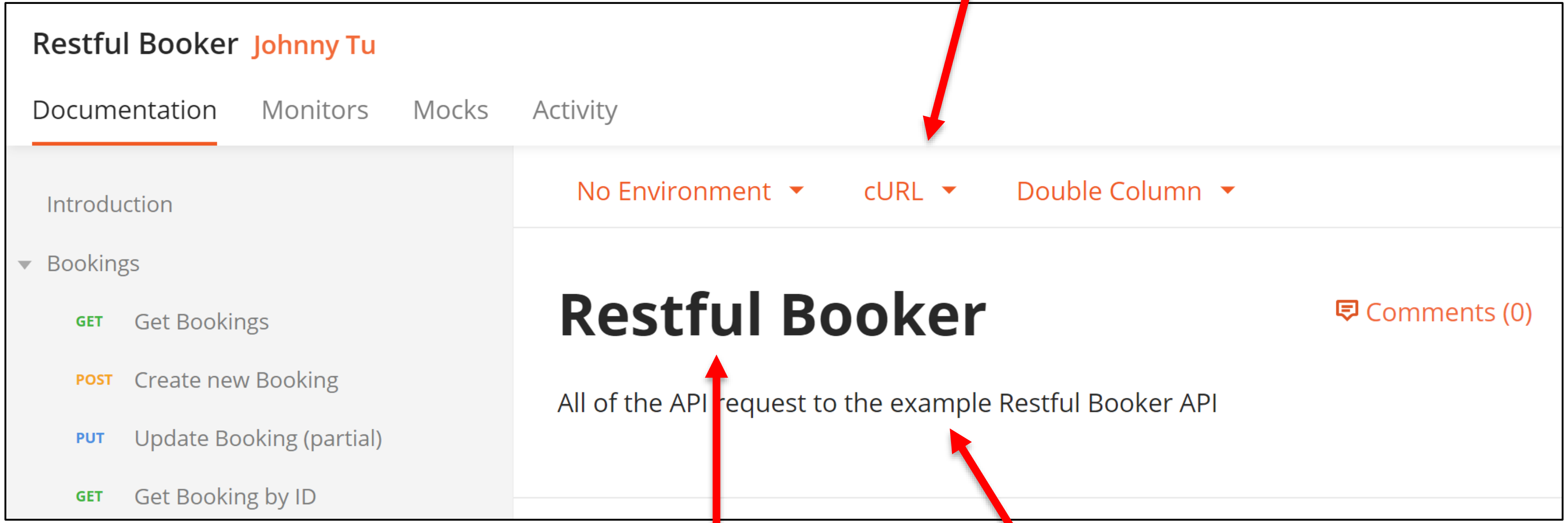
Documentation content

- Postman automatically generates the documentation content by using the contents of the folders and requests in your collection
- Descriptions for the collection, folders and requests
- For each request the docs will show:
  - Configured headers and parameters
  - Request body (if applicable)
  - Generated cope snippets in various programming languages

POSTMAN

Example Documentation page

Selected programming language



Restful Booker Johnny Tu

Documentation Monitors Mocks Activity

Introduction

Bookings

GET Get Bookings

POST Create new Booking

PUT Update Booking (partial)

GET Get Booking by ID

No Environment cURL Double Column

Restful Booker

Comments (0)

All of the API request to the example Restful Booker API

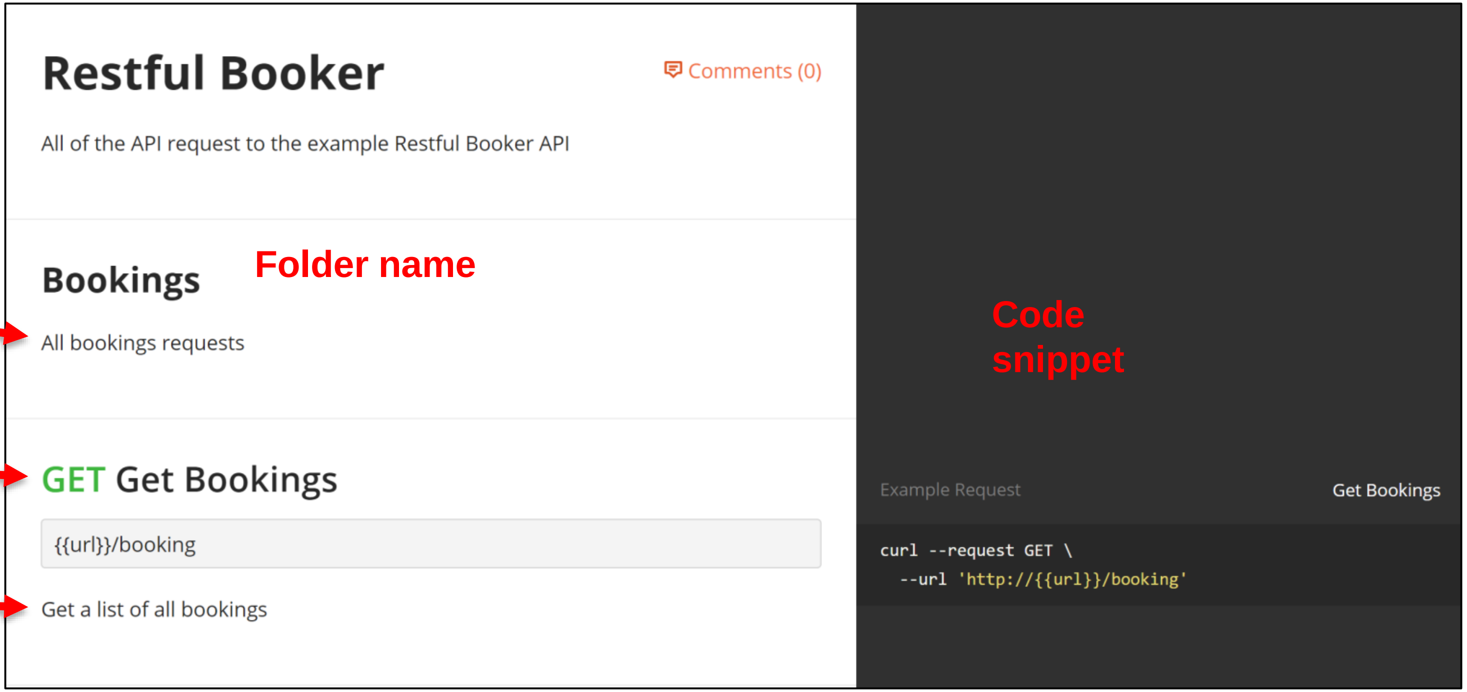
Folders and requests

Collection name

Collection description

POSTMAN

Documented Requests



Restful Booker

Comments (0)

All of the API request to the example Restful Booker API

Bookings

All bookings requests

GET Get Bookings

{{url}}/booking

Get a list of all bookings

Example Request

Get Bookings

```
curl --request GET \
  --url 'http://{{url}}/booking'
```

Folder description

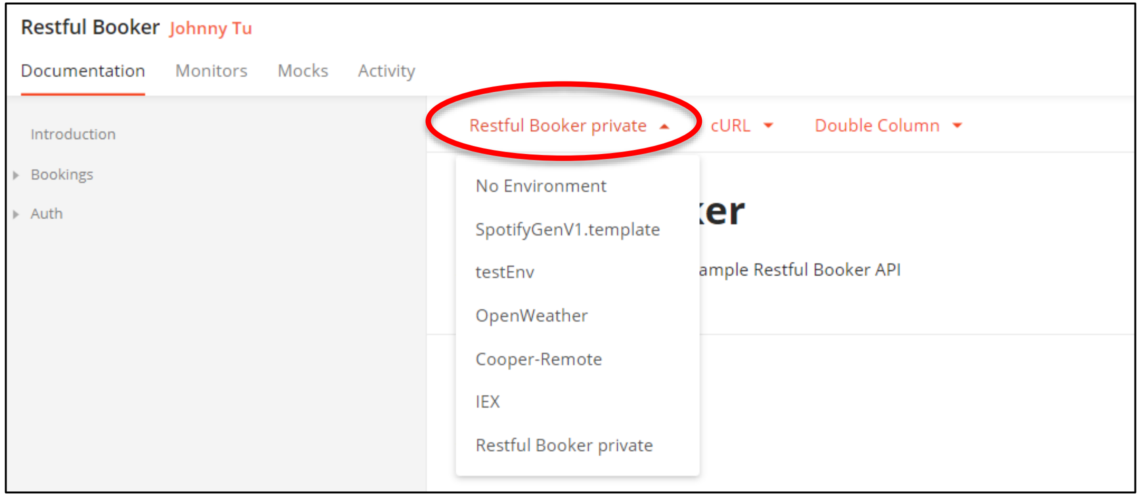
Request name

Request description

POSTMAN

Selecting an environment

- Selecting an environment will assign the environment values to any variables in the documentation
  - Only the variable “Initial Value” is assigned
- Can select from all local and shared environments


Restful Booker Johnny Tu

Documentation Monitors Mocks Activity

Introduction
Bookings
Auth

Restful Booker private ▲ cURL ▼ Double Column ▼

No Environment
SpotifyGenV1.template
testEnv
OpenWeather
Cooper-Remote
IEX
Restful Booker private

ker

ample Restful Booker API

POSTMAN

EXERCISE

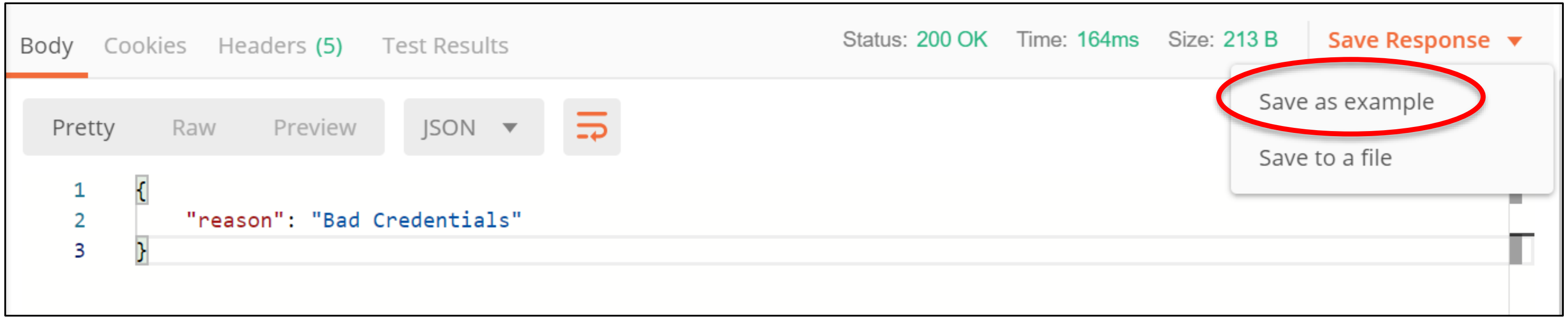
EX3.1 – Generate private documentation

1. Click the “New” dropdown and select “Documentation”
2. Select the Restful Booker collection and create the documentation for it
3. Open the Restful Booker collection documentation
4. Browse through the requests in the docs
5. Select the Restful Booker environment and observe how variable values are substituted in the docs

POSTMAN

Example requests and responses

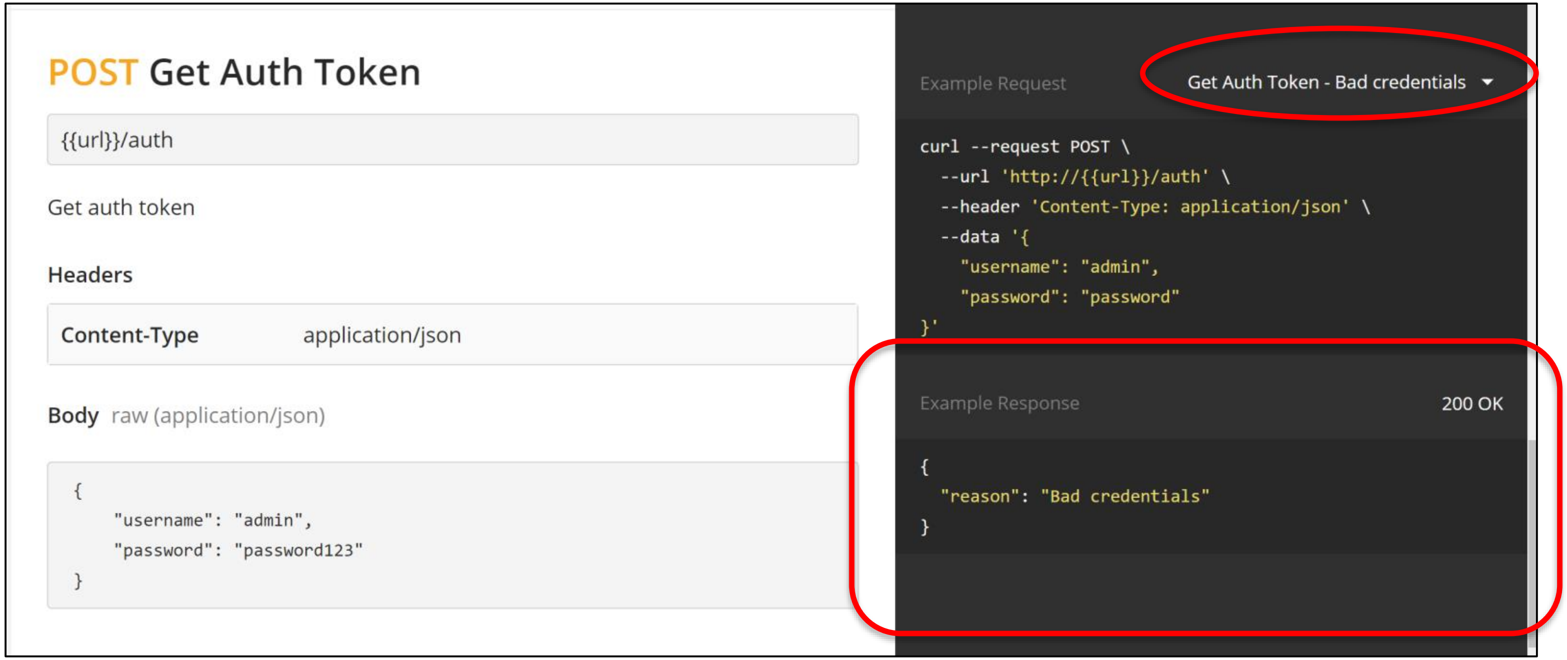
- It’s useful to show example responses in the documentation to assist developers in understanding what to expect
- To include example responses, run the request and save the response
- Can include as many examples as needed
  - Name them effectively to make it easy for users to understand what each example illustrates



```
{
    "reason": "Bad Credentials"
}
```

POSTMAN

Viewing Example responses



POST Get Auth Token

{{url}}/auth

Get auth token

Headers

Content-Type	application/json

Body raw (application/json)

```
{
    "username": "admin",
    "password": "password123"
}
```

Example Request   Get Auth Token - Bad credentials

```
curl --request POST \
  --url 'http://{{url}}/auth' \
  --header 'Content-Type: application/json' \
  --data '{
    "username": "admin",
    "password": "password"
}'
```

Example Response   200 OK

```
{
  "reason": "Bad credentials"
}
```

POSTMAN

EXERCISE

EX3.2 – Save example responses

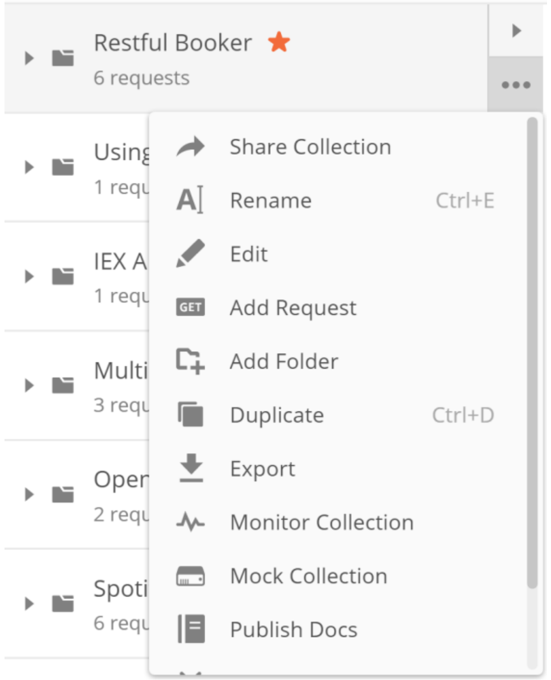
1. Open the **Get Auth Token** request in the Restful Booker collection
2. Change the CURRENT VALUE of the “password” variable in the Restful Booker environment to “wrongpw” and send the request. You should get a response saying “Bad Credentials”
3. Save the response and call it “**Get Auth Token – Bad credentials**”
4. Change the CURRENT VALUE of the “password” variable in the Restful Booker environment to “password123” and send the request. This time you should receive an auth token in the response
5. Save the response and call it “**Get Auth Token – Token returned**”
6. Go to the web browser and refresh your documentation page
7. Scroll down to the “Get Auth Token” endpoint. You should be able to select from your two example responses

Publishing Documentation

POSTMAN

Creating public documentation

- Click the "Publish" button on the private documentation page
- Click "Publish Docs" from collection drop down
- Must be collection author or have write permissions to publish the docs
- Changes made in the collection will be automatically reflected in the published docs



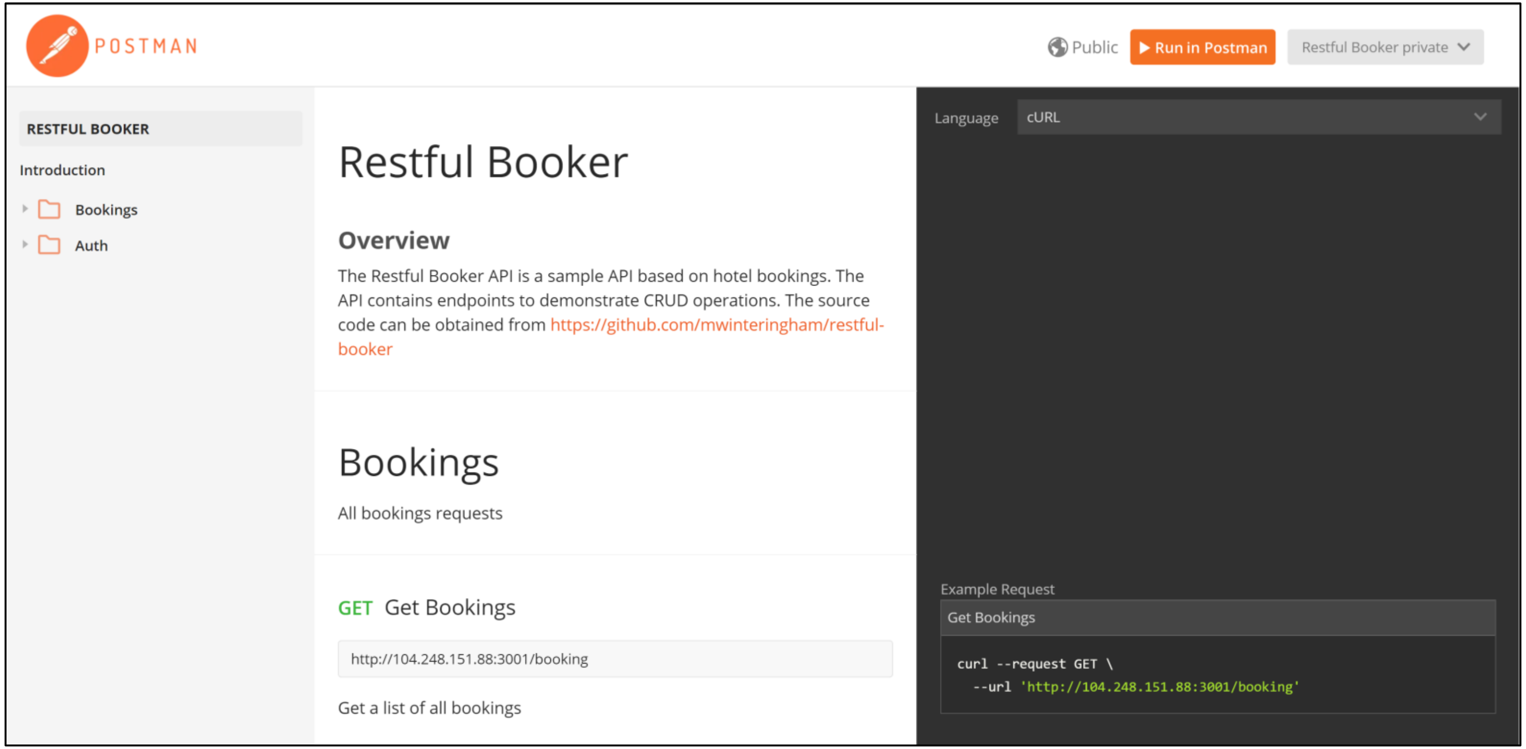
POSTMAN

Publishing options

- Select the appropriate environment so references to variables will be replaced with the value from the environment
  - Remember not to include any sensitive data in the environment values
- Custom page styling options
- Make your API discoverable through the API network (Postman Pro and Enterprise only)
  - https://www.getpostman.com/docs/v6/postman/launching_postman/newbutton#api-network
- Share as a template

POSTMAN

Public documentation view



Restful Booker

Overview

The Restful Booker API is a sample API based on hotel bookings. The API contains endpoints to demonstrate CRUD operations. The source code can be obtained from https://github.com/mwinteringham/restful-booker

Bookings

All bookings requests

GET Get Bookings

```
http://104.248.151.88:3001/booking
```

Get a list of all bookings

Example Request

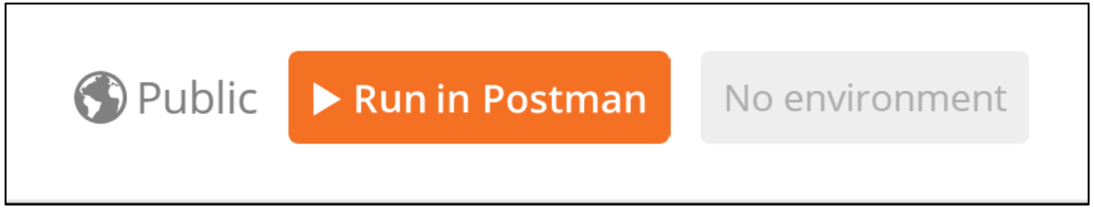
Get Bookings

```
curl --request GET \
  --url 'http://104.248.151.88:3001/booking'
```

POSTMAN

Run in Postman button

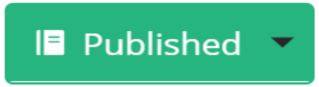
- Allows anyone to import a collection and its associated environment with one click
- Included on public documentation pages
- Can generate your own "Run in Postman" embed code and embed it onto any web page
- Great for developer onboarding
- Examples:
  - https://developer.okta.com/reference/postman_collections/
  - https://apidocs.imgur.com/



POSTMAN

EXERCISE

EX3.3 – Publish documentation

1. On your web browser, go to the private documentation view you created in Exercise 3.1
2. Click **Publish**
3. Select the Restful Booker environment and then publish the documentation
4. Go back to the Postman App and open the Restful Booker docs in the private view (use the “View in Web” option)
5. Click on the “Published” button on the top right 

6. Select “Edit Published Documentation” and then click on the “Unpublish” button to remove your documentation from public view

POSTMAN

Module Review

Key Points:

- Private documentation is created based on a collection and it’s configured folders and requests
- Publishing as collection’s private documentation makes it available for public viewing

POSTMAN

Module 4 – Mock Servers

POSTMAN

Module Objectives

After completing this module, you will be able to:

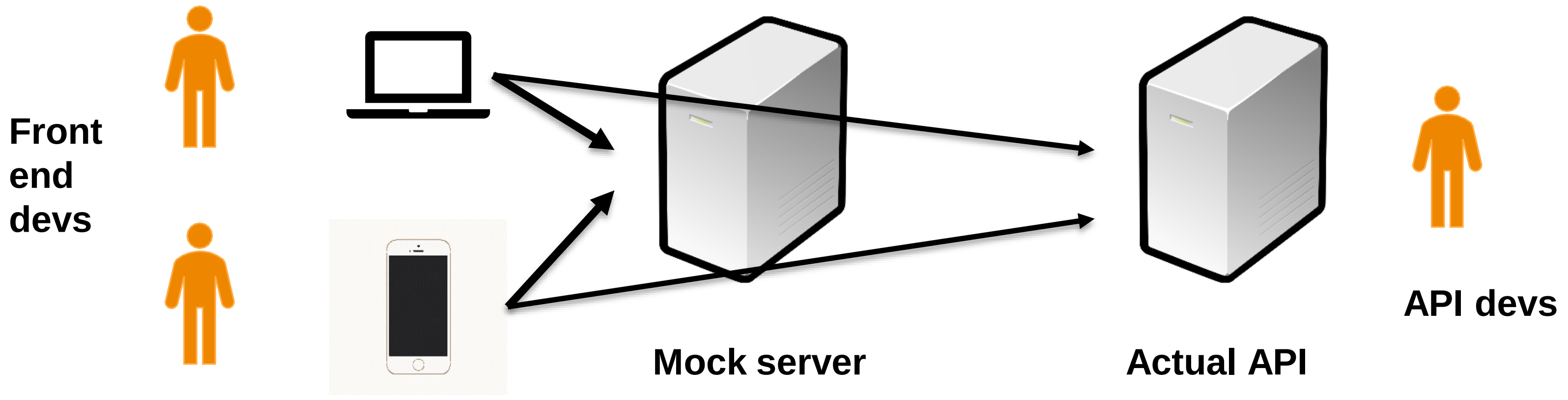
- Create a mock server for a new API
- Define example responses for mock requests
- Describe how Postman selects which response to use for a given mock request

POSTMAN

Intro to Mock servers

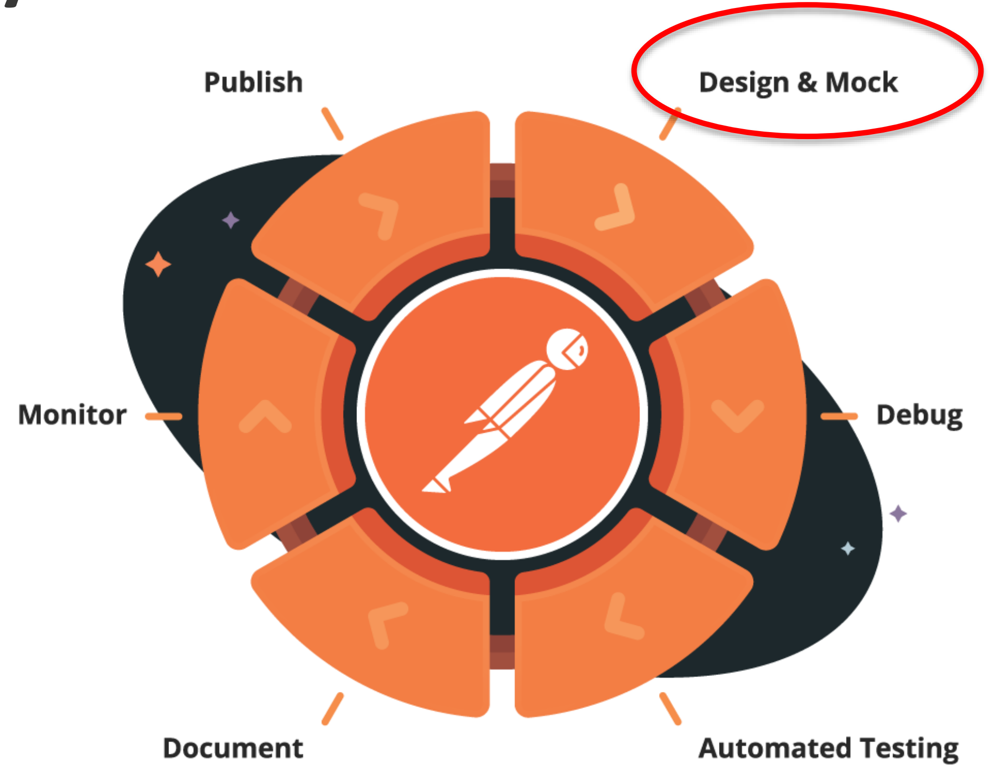
What is a mock

- A mock is a fake API that simulates a server response
- Postman mocks are associated with an underlying collection



POSTMAN

API Lifecycle



Public vs Private mocks

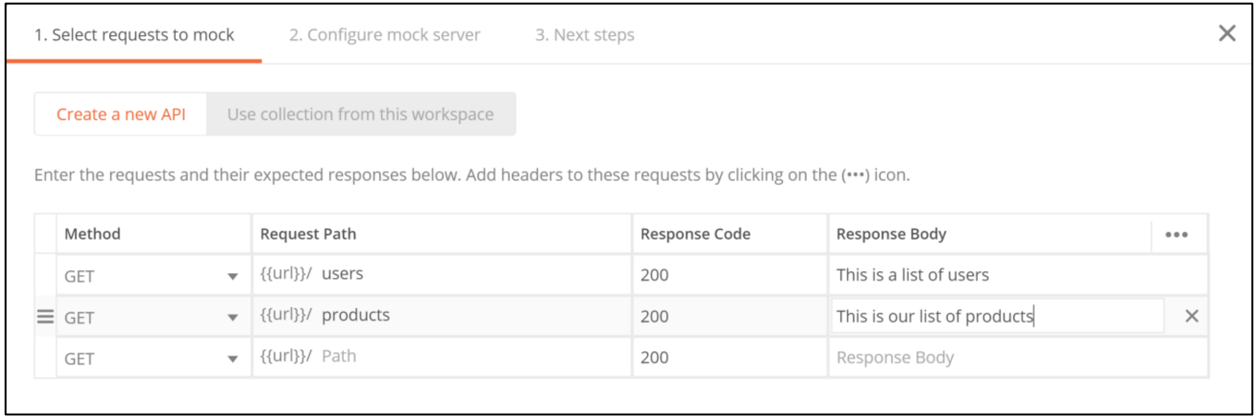
- Public mock
  - Mock server is accessible by anyone
  - No need for Postman API key
  - Mock servers are by default, publicly available
- Private mock
  - Only accessible to users who have access to the underlying Collection
  - Users must use their Postman API key when sending requests to the mock endpoints

POSTMAN

Create a mock server

Create a mock

- Two ways to create a Mock
  - Create a new API
  - Create from an existing Collection

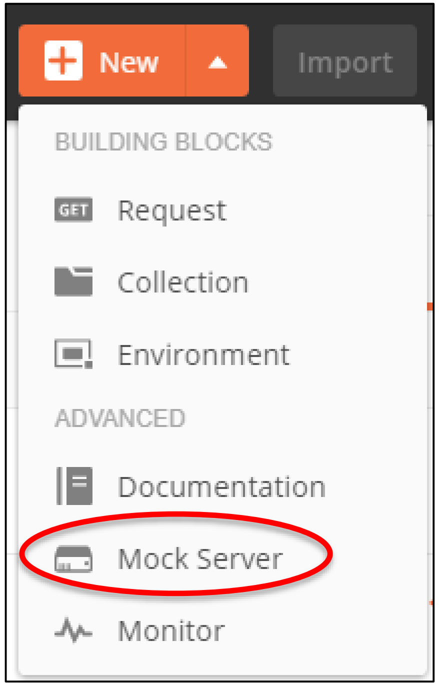


1. Select requests to mock | 2. Configure mock server | 3. Next steps

Create a new API | Use collection from this workspace

Enter the requests and their expected responses below. Add headers to these requests by clicking on the (•••) icon.

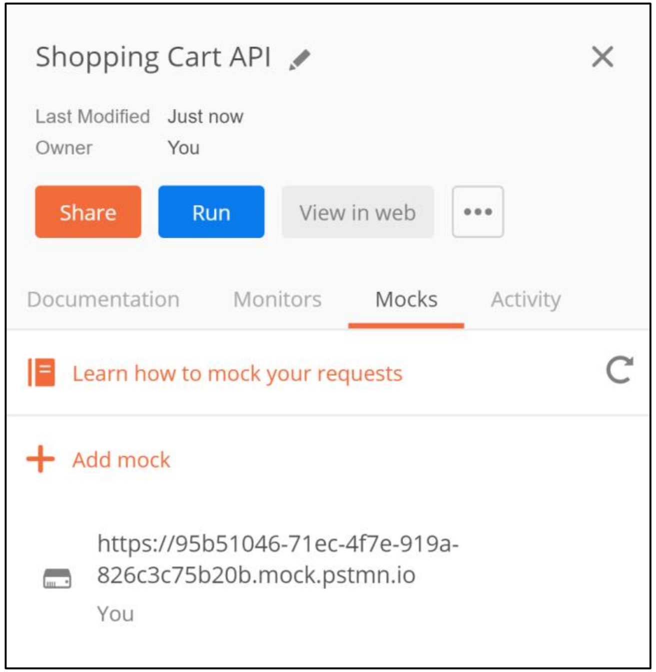
Method	Request Path	Response Code	Response Body
GET	{{url}}/ users	200	This is a list of users
GET	{{url}}/ products	200	This is our list of products
GET	{{url}}/ Path	200	Response Body



POSTMAN

Create a mock (cont'd)

- Create from API
  - Define the endpoints you want to mock
  - Specify the HTTP method and response code for each endpoint
  - Define an example response for each endpoint
- Postman will create a new mock server and collection for your API
- An environment will be created with a "url" variable, which contains the URL to the mock server
- Open the Collection and go to the "Mocks" tab to see the mock server


Shopping Cart API

Last Modified Just now
Owner You

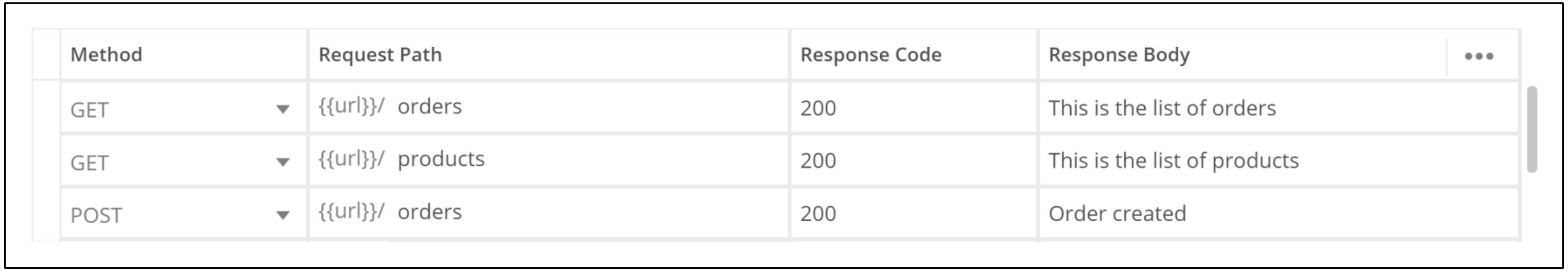
https://95b51046-71ec-4f7e-919a-826c3c75b20b.mock.pstmn.io
You

POSTMAN

EXERCISE

EX4.1 – Create a mock server

1. Click the “New” dropdown in the header and select “Mock Server”
2. Select the “Create a new API” tab
3. Fill in the details as indicated on the screenshot below and click “Next”



Method	Request Path	Response Code	Response Body
GET	{{url}}/ orders	200	This is the list of orders
GET	{{url}}/ products	200	This is the list of products
POST	{{url}}/ orders	200	Order created

POSTMAN

EXERCISE

EX4.1 – (cont'd)

4. On the "Name" field, type "Shopping Cart API"
5. Do not select an environment and leave the "Make this mock server private" checkout blank
6. Click "Create"
7. You should now see the mock server URL
8. Find the Shopping Cart API on your Collection's list and expand it
9. Click the "Mocks" tab to see your mock server listed
10. Find and select the "Shopping Cart API" on the environment drop down list
11. Click on the "Environment quick look" button and confirm that you can see the mock URL
12. Open your browser and sends a request to <mock url>/orders. What do you see in the response?

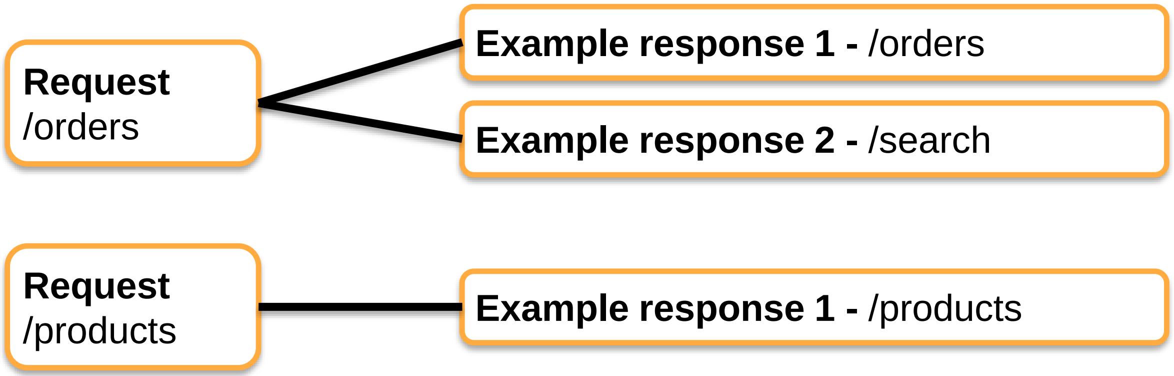
POSTMAN

Mock with Examples

POSTMAN

Mock responses

- Mock responses are dependant on your saved examples
- Examples are matched against the request URL and method type
- Multiple examples can be saved against each request in a Collection
- Example can have a unique endpoint



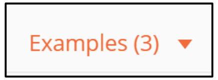
Request /orders
Example response 1 - /orders
Example response 2 - /search
Request /products
Example response 1 - /products

POSTMAN

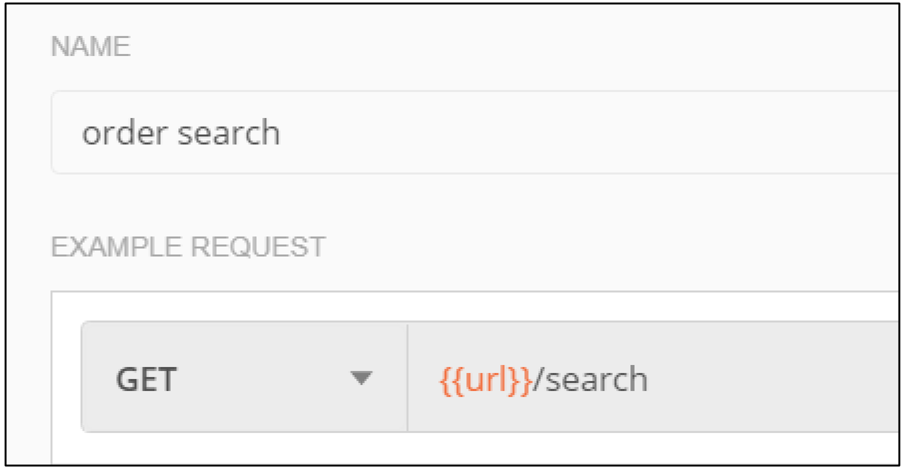
EXERCISE

EX4.2 – Create examples

1. Open the GET **Orders** request in the in the Shopping Cart API
2. Click the “Examples” dropdown and select “Add Example”


Examples (3)

3. Configure the name and example request as shown:


NAME

order search

EXAMPLE REQUEST

GET

{{url}}/search

POSTMAN

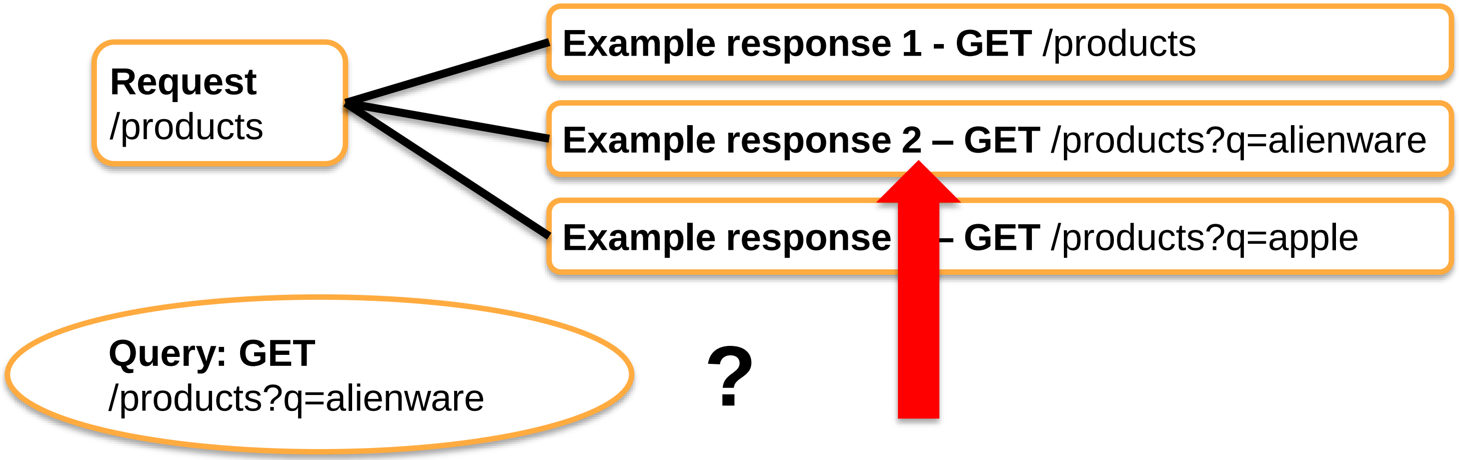
EXERCISE

EX4.2 – (cont'd)

4. In the example response body, type "This is the list of orders based on our search"
5. Click "Save Example" on the top right corner
6. Use Postman to send a request to the mock server on the "/search" endpoint
7. What can you observe in your response?
8. Change the request to /orders and observe the response

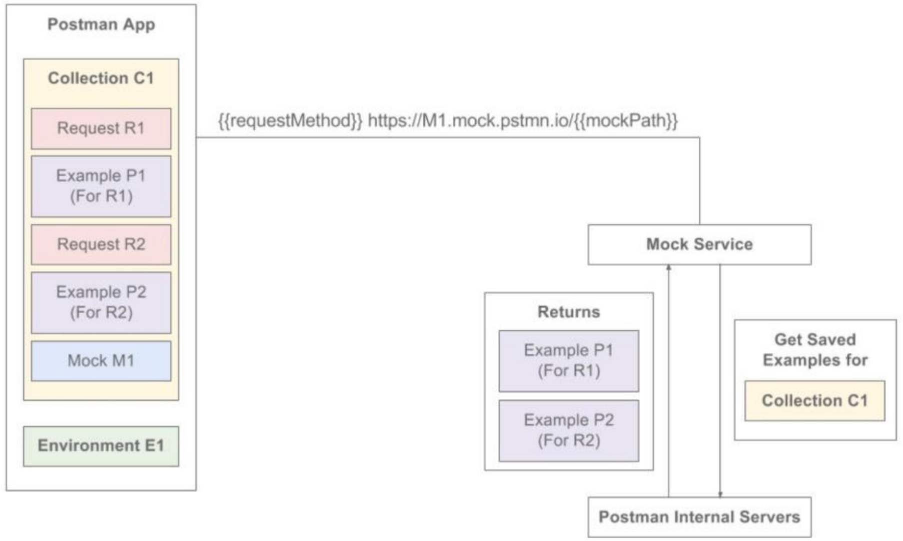
Using query parameters

- Different example responses can be returned based on matching query parameters
- Allows mocking of different responses for different query parameters on the same request path


Request /products
Example response 1 - GET /products
Example response 2 – GET /products?q=alienware
Example response – GET /products?q=apple
Query: GET /products?q=alienware
?

Matching algorithm

- When multiple examples are configured for a request, how does the mock service know which one to return?
- When a mock call is made, the mock service retrieves all saved examples for that collection, from the Postman servers and begins the matching process.



POSTMAN

Matching algorithm (cont’d)

- The incoming request is paired with the closest matching example
- The request method, mock path and various request headers are checked
- The matching algorithm will remove example responses from the list of options it can return, based on the following logic

1. Properly formatted responses
   - Responses not in the expected format are removed

Matching algorithm – (cont'd)

2. HTTP method
   - Responses that don't correspond to the request HTTP method are removed
3. URL filter
   - Iterate over the remaining examples to compare the mock path of the request to the example and set a threshold value
   - Step by step matching in the following order
     1. Exact match
     2. Strip out trailing slashes
     3. Lower case the example path and the request mock path
     4. Strip out alphanumeric ID's from both input and example path
     5. If all the steps have failed, the example is not eligible

Matching algorithm – (cont'd)

4. Response code
   - Check if the request has the `x-mock-response-code` header and filter out the examples with a non matching response code
5. Highest threshold value
   - Remaining response are sorted in descending order and the one with the highest threshold value is returned.

POSTMAN

Module Review

Key Points:

- Mock servers help speed up overall application development
- Mock server calls are dependent on configured example responses
- Postman supports multiple responses per mock endpoint

Module 5 – Monitors

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Describe the process of monitoring a collection
- Setup and configure a monitor for a collection

POSTMAN

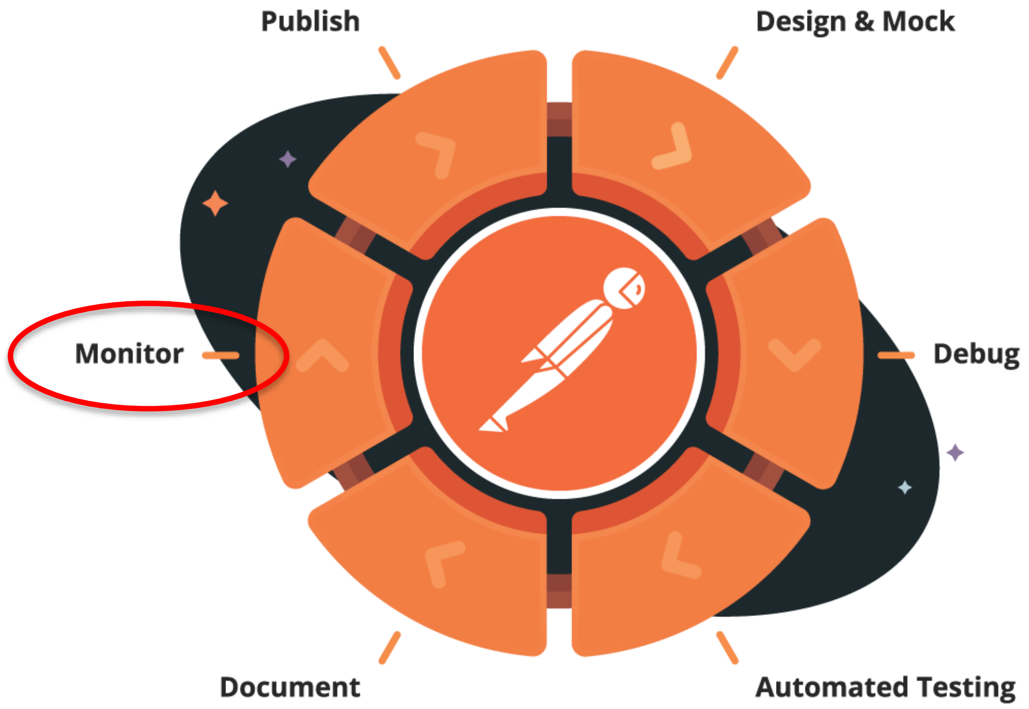
Intro to Monitors

What is monitoring

- A **monitor** is a scheduled run of a Collection
- A monitor allows you to run a collection periodically to check for performance
- Checks are done to ensure all requests in a collection are healthy
- Tests in each request are also run
- Monitor’s can be configured to send notification alerts when there are test failures

POSTMAN

API lifecycle



Collection Runner vs Monitor

- Differences exists between running collections with the collection runner and with a monitor
- Variables
  - No import of global variables
  - No variable persistence
- Console output
  - Request and response bodies and sensitive headers are not logged by default
- Monitors only run 1 iteration and the max run time is 5 minutes
- Data files
  - Cannot upload a data file but can use data files from API's such as Google sheets or Dropbox
- All API's endpoints must be publicly available

Pricing

- Postman monitors are billed per-request made
- Free Postman users are allowed 1000 free monitoring calls per month
- Postman Pro teams allowed 10,000 free monitoring calls per month
- Postman Enterprise teams allowed 100,000 free monitoring calls per month

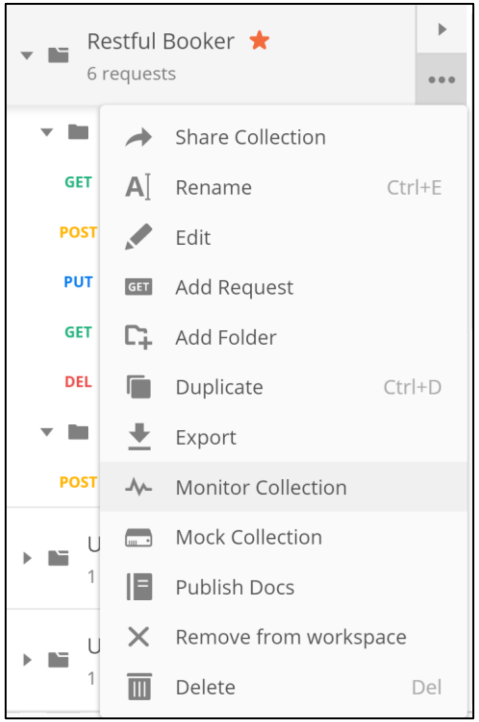
POSTMAN

Setup a Monitor

POSTMAN

Create a monitor

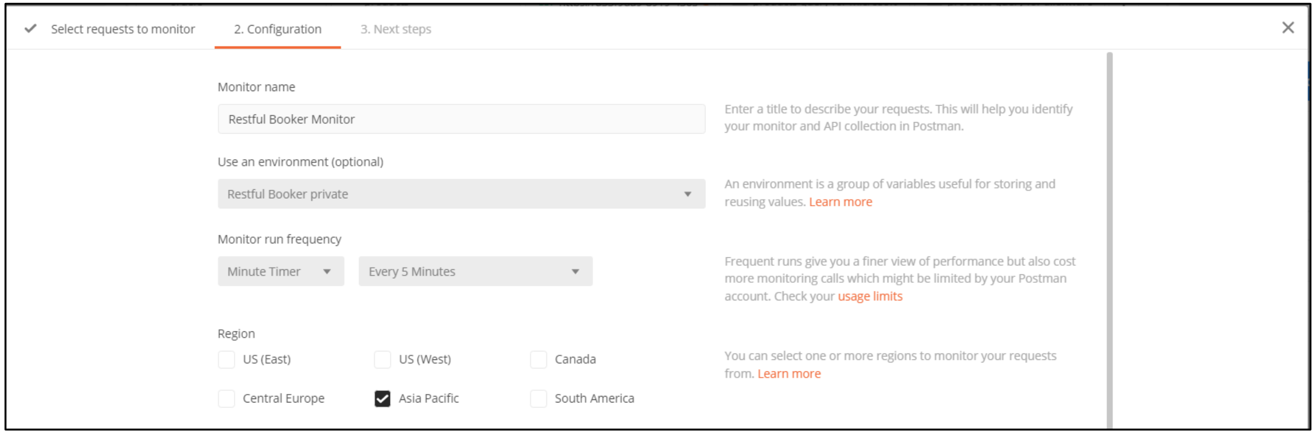
- Can create a monitor from an existing Collection or create a new API in the process
- Select environment to use when running the collection
- Define the frequency
  - Lowest interval is 5 minutes
- Select region where the monitor will run



POSTMAN

Create a monitor – (cont'd)

- Monitors created on a shared collection will be visible to the team
- Monitors created on a private collection will only be visible to you
- Cannot share monitor's across workspaces



Monitor name

Restful Booker Monitor

Enter a title to describe your requests. This will help you identify your monitor and API collection in Postman.

Use an environment (optional)

Restful Booker private

An environment is a group of variables useful for storing and reusing values. Learn more

Monitor run frequency

Minute Timer | Every 5 Minutes

Frequent runs give you a finer view of performance but also cost more monitoring calls which might be limited by your Postman account. Check your usage limits

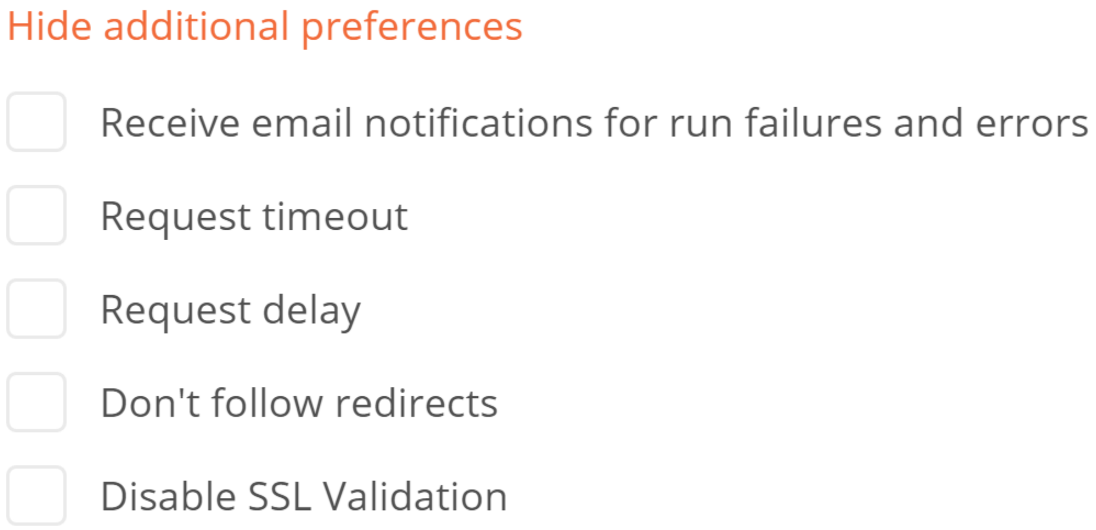
Region

☐ US (East) ☐ US (West) ☐ Canada

☐ Central Europe ☑ Asia Pacific ☐ South America

You can select one or more regions to monitor your requests from. Learn more

Additional preferences



Hide additional preferences

- ☐ Receive email notifications for run failures and errors
- ☐ Request timeout
- ☐ Request delay
- ☐ Don't follow redirects
- ☐ Disable SSL Validation

Multi region monitoring

- When select the region for your monitor, you can choose multiple regions
- The region defines where the monitoring calls to the API are made from
- The monitor will run the collection once for each region specified
- Results in more monitoring calls
- Failures will be duplicated

EX5.1 – Create a monitor

1. Open the **Restful Booker environment** and set the initial value of the **password** variable to "password123"
2. Click the "New" dropdown and select "Monitor"
3. Select the "Use Collection from this Workspace" option and choose the Restful Booker collection
4. Name the monitor "Restful Booker monitor"
5. Select the Restful Booker environment
6. Set the monitor frequency to be once per day
7. Select any region of your choice
8. Add yourself to receive email notifications on run failures and errors
9. Click Create

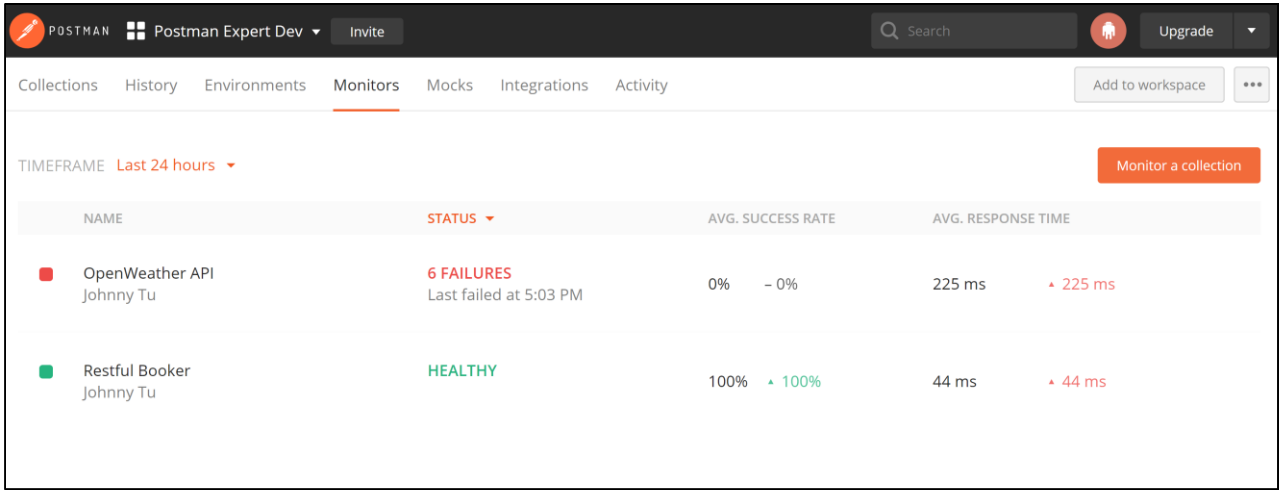
POSTMAN

Viewing monitor results

POSTMAN

Monitors page

- Monitor results are viewed through the web portal of your Postman account
- Monitors page displays the list of monitors you have access to (individual and team)

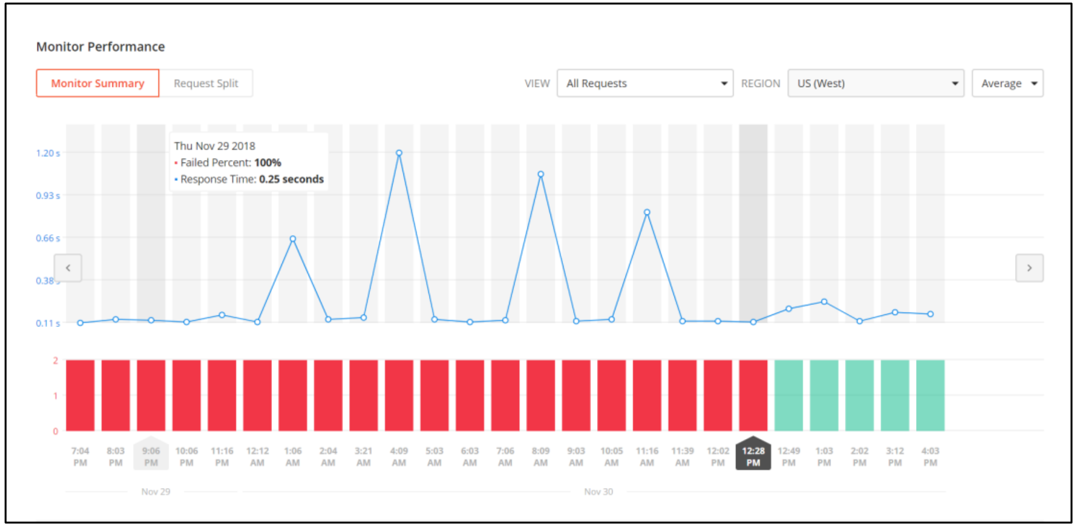


TIMEFRAME Last 24 hours

	NAME	STATUS	AVG. SUCCESS RATE		AVG. RESPONSE TIME	
🟥	OpenWeather API Johnny Tu	6 FAILURES Last failed at 5:03 PM	0%	– 0%	225 ms	▲ 225 ms
🟩	Restful Booker Johnny Tu	HEALTHY	100%	▲ 100%	44 ms	▲ 44 ms

Monitor details page

- Shows all past runs of the monitor
- Red columns indicate failed runs, green indicates successful runs
- Click on the column to show the full run details


Monitor Performance

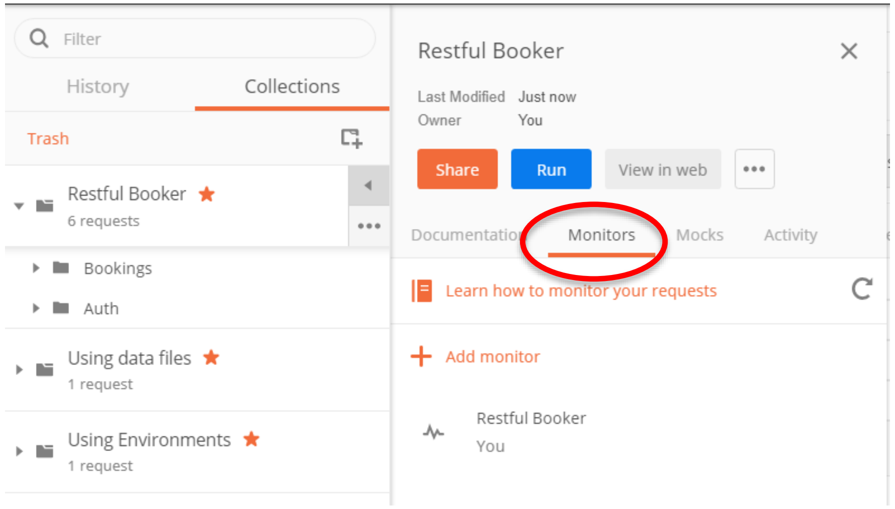
VIEW All Requests REGION US (West) Average

Thu Nov 29 2018
- Failed Percent: **100%**
- Response Time: **0.25 seconds**

POSTMAN

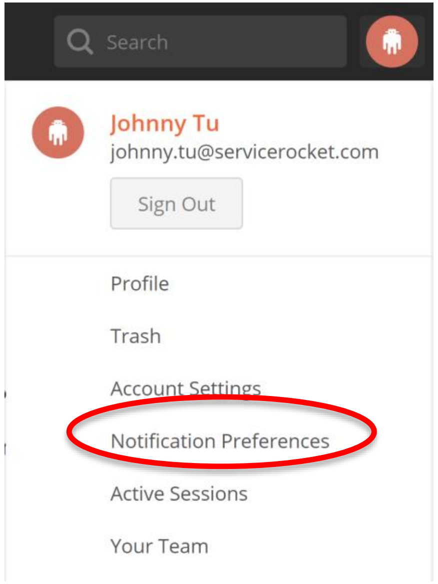
Viewing your monitor results

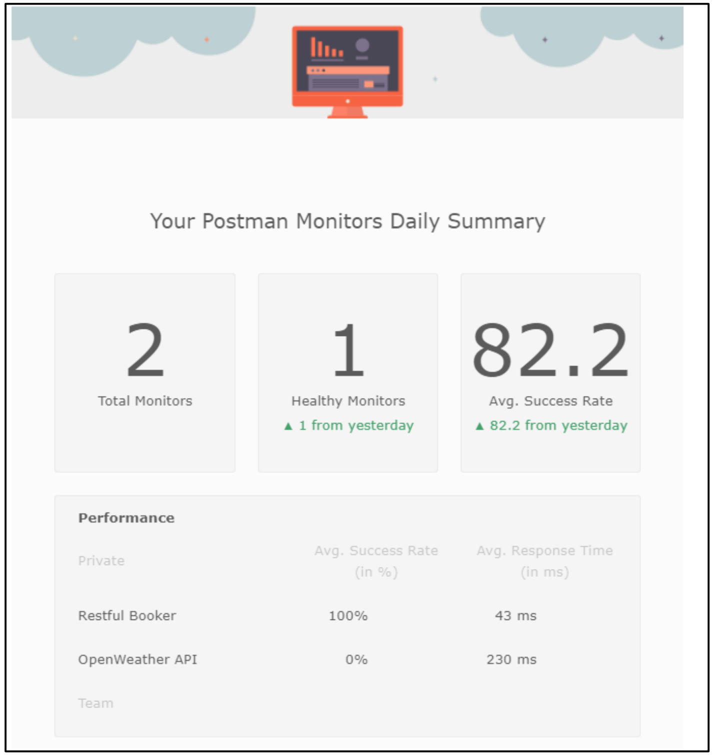
- Monitor results are shown in the web portal of your Postman account
- Can have multiple monitors in a collection
- Expand the collection view in the Postman App and go to the Monitors tab to find the list of monitors for that collection



Monitor email summary

- Receive a daily or weekly email summary of your monitors
- Configured in your account Notification preferences




Your Postman Monitors Daily Summary

2	1	82.2
Total Monitors	Healthy Monitors	Avg. Success Rate
	▲ 1 from yesterday	▲ 82.2 from yesterday

Performance

Private	Avg. Success Rate (in %)	Avg. Response Time (in ms)
Restful Booker	100%	43 ms
OpenWeather API	0%	230 ms
Team		

POSTMAN

EXERCISE

EX5.2 – View monitor results

1. On the Postman App, click on the Restful Booker collection and click on the Monitors tab
2. Click the monitor listed to open up the monitor details page
3. Click the “Run” button to run the collection and then refresh the page
4. Open up the details of the latest run and alternate between the “Test Results” tab and the “Console Log” tab

POSTMAN

Module Review

Key Points:

- Monitors run a collection in the cloud on a scheduled basis to test our API’s
- Some differences exist when running a collection on the collection runner compared to running in a monitor
- Email notifications can be sent out in the result of a failed collection run

POSTMAN

Further references

- Documentation - https://learning.getpostman.com
- Blog - http://blog.getpostman.com
- Postman API docs - https://docs.api.getpostman.com/
- Restful Booker sample API - https://restful-booker.herokuapp.com/apidoc/index.html
  - https://github.com/johnny-tu/restful-booker (API ported over to Java)

POSTMAN

You have reached the end of the Postman Expert course!