POSTMAN

Postman Fundamentals

PRESENTED BY

Johnny Tu

POSTMAN

Welcome
&
Introductions

Instructor: Johnny Tu

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

POSTMAN

Housekeeping

Duration

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

Format

- Slides
- Demonstrations
- Exercises

Teleconference

- Please mute when possible
- Stay on the line during break
- Please use headset and microphone

POSTMAN

Course Prerequisites

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.

OpenWeather API account is needed for later exercises:

1. Go to https://home.openweathermap.org/users/sign_up and register for a free account.
2. Sign in to your newly registered account.
3. Click the **API Keys** tab and verify that an API key was created.

POSTMAN

Agenda

1. Postman Concepts & Postman UI
2. Creating & Sending API Requests
3. Organizing Requests into Postman Collections
4. Configuring Postman Variables & Environments
5. Performing Basic API Testing with JavaScript
6. Collaborating through Postman Workspaces

POSTMAN

Module 1
Postman Concepts & User Interface

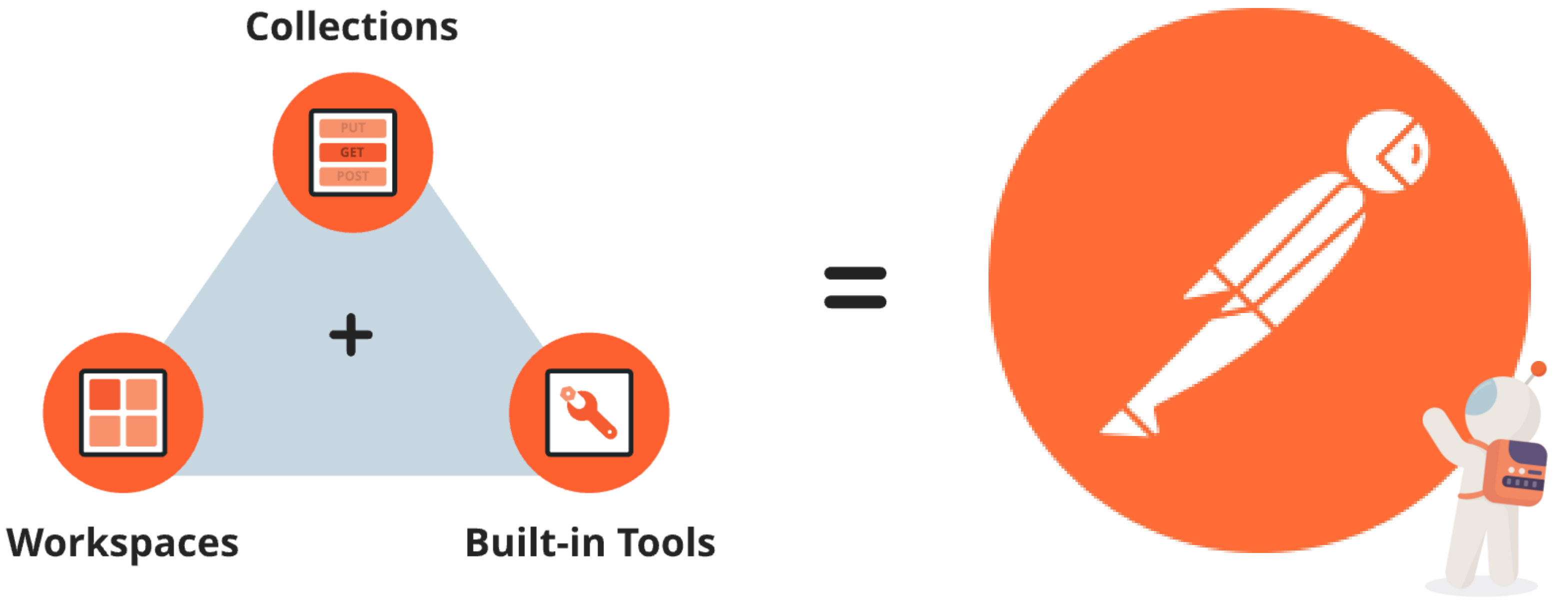
Module Objectives

After completing this module, you will be able to:

- Describe Postman Concepts including: Requests, Collections, Variables, Environments, and Workspaces
- Describe Postman User Interface (UI)
- Locate Postman UI components including: Request Builder, Sidebar, and Header

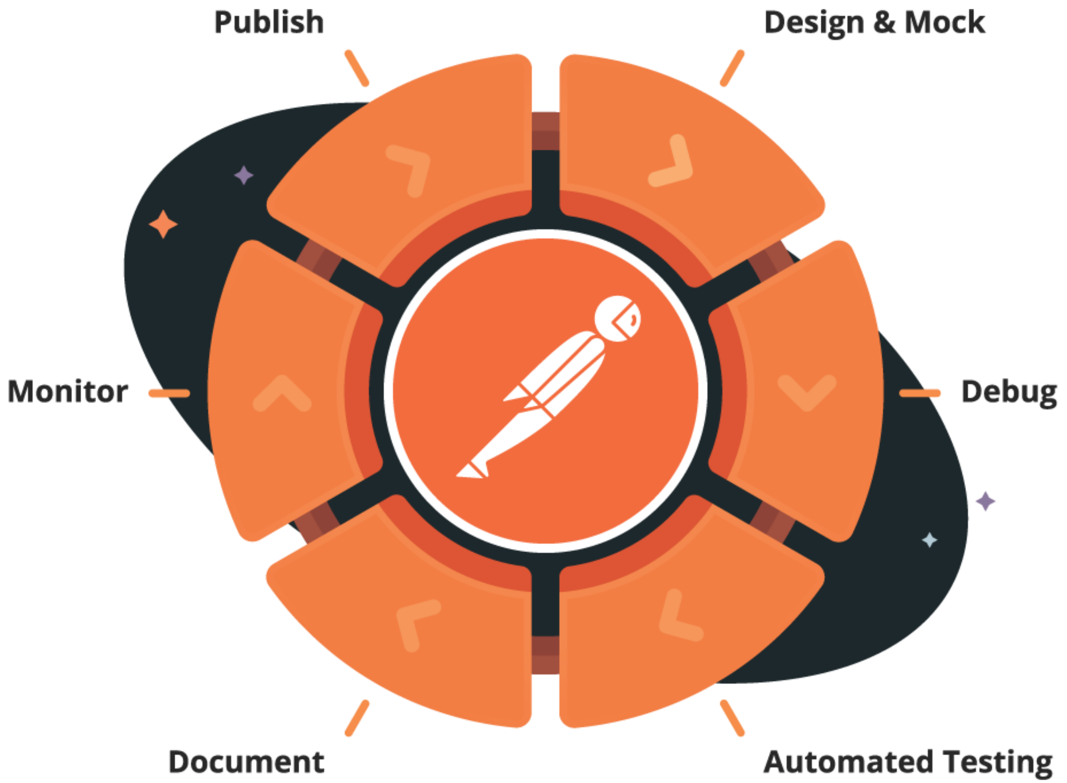
Postman Concepts

Collaborative API Development Environment



POSTMAN

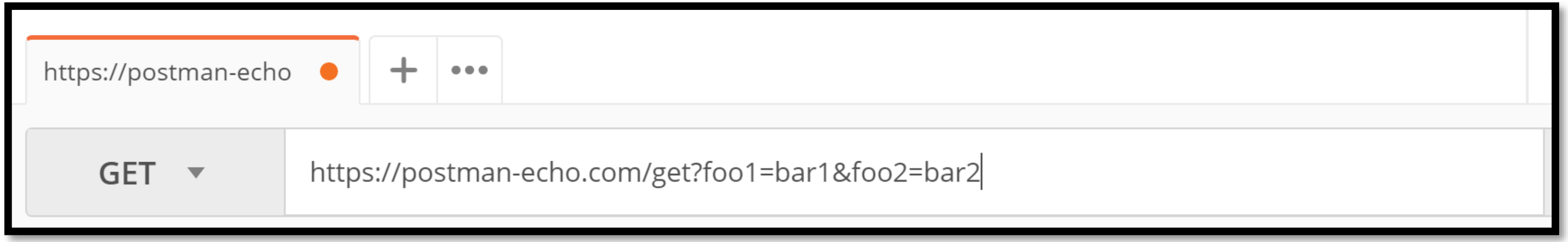
Collaborate Across API Lifecycle Using Postman Tools



POSTMAN

Requests

- A **Request** is a core function in Postman
- Send a HTTP Request to an API endpoint and perform some action
- Support for all HTTP methods (GET, POST, PUT, DELETE, etc.)
- Postman displays the results of the request in the UI
- Can be saved for re-use in future

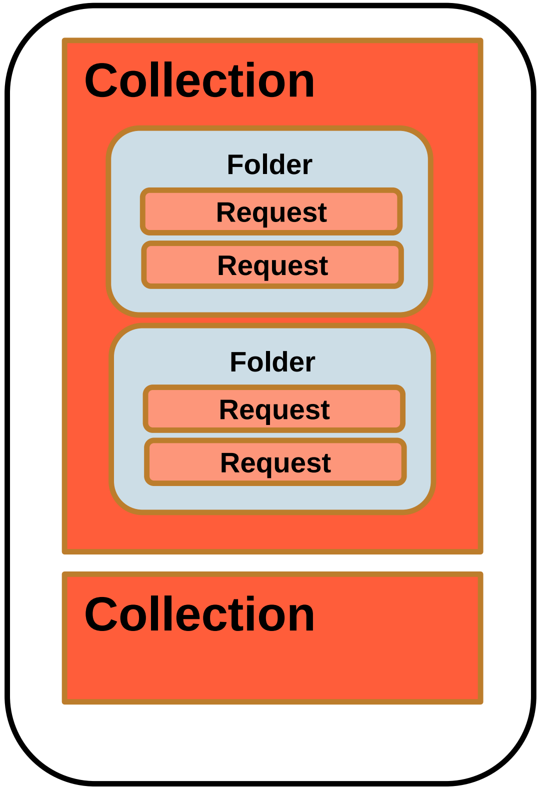


https://postman-echo.com/get?foo1=bar1&foo2=bar2

POSTMAN

Collections

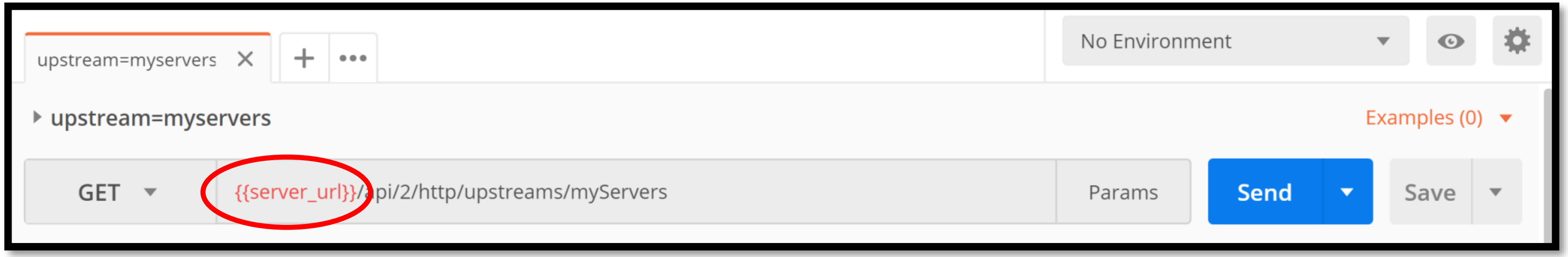
- A **Collection** is a group of saved requests
- Requests in a Collection can be further organized into Folders
- Reasons to use Collections:
  - Organization
  - Documentation
  - Test Suites
  - Conditional Workflows



POSTMAN

Variables

- **Variables** allow for re-use of values to avoid repetition
- Change value in one place
- Can be used in the Request Builder and in scripts
- Variable values can be defined globally, in a collection or in an Environment
- Configure different Environments for the same Request



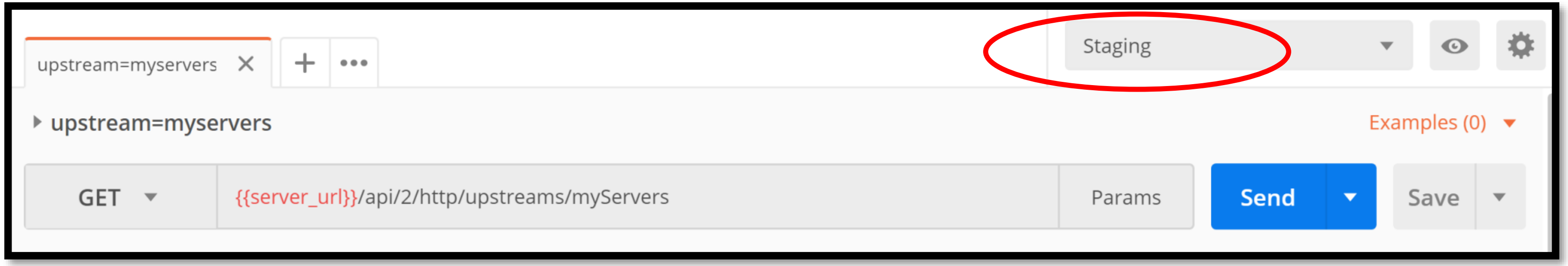
upstream=myservers

GET {{server_url}}/api/2/http/upstreams/myServers

POSTMAN

Environments

- An **Environment** is a set of key-value pairs
- Each key represents a Variable that can be used somewhere in a Request
- Allows easy switching between setups without having to change the Request

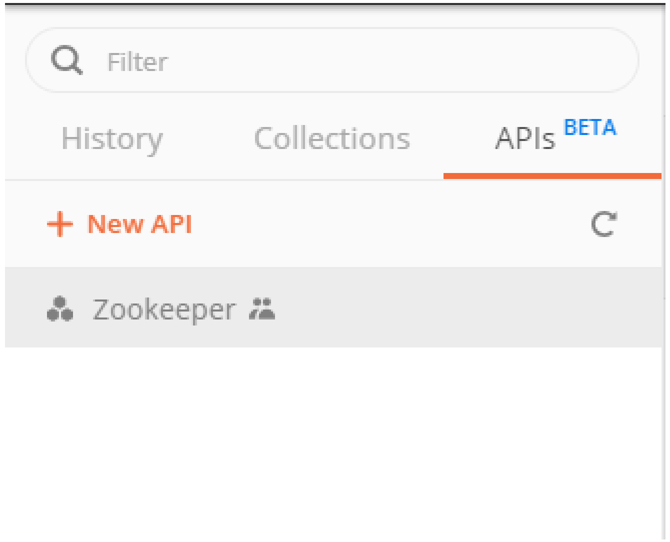


```
{{server_url}}/api/2/http/upstreams/myServers
```

POSTMAN

APIs

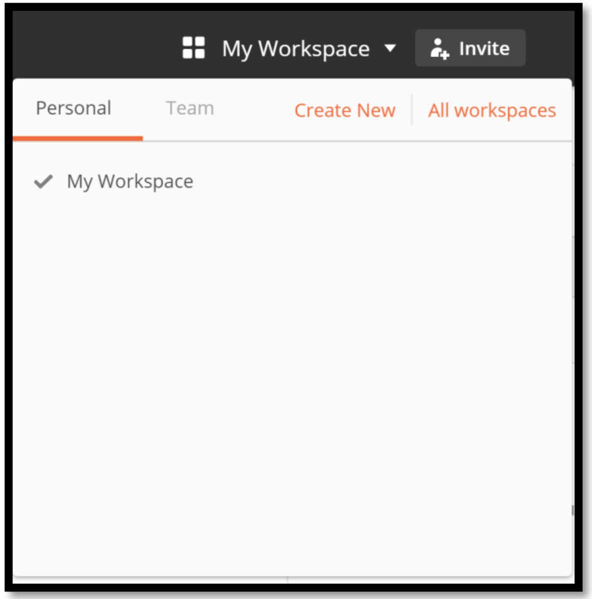
- Postman's API feature allows you to design and develop APIs in your workspace
- Define or import an API schema in Postman
- Version control of APIs
- Generate collections from an API or link an existing collection to an API version



POSTMAN

Workspaces

- A **Workspace** is a view of Postman Collections, Environments, etc.
- Provides another layer of organizing your work
- Team vs Personal Workspaces
- Collections can be shared across Workspaces
- Team Workspaces enhances collaboration

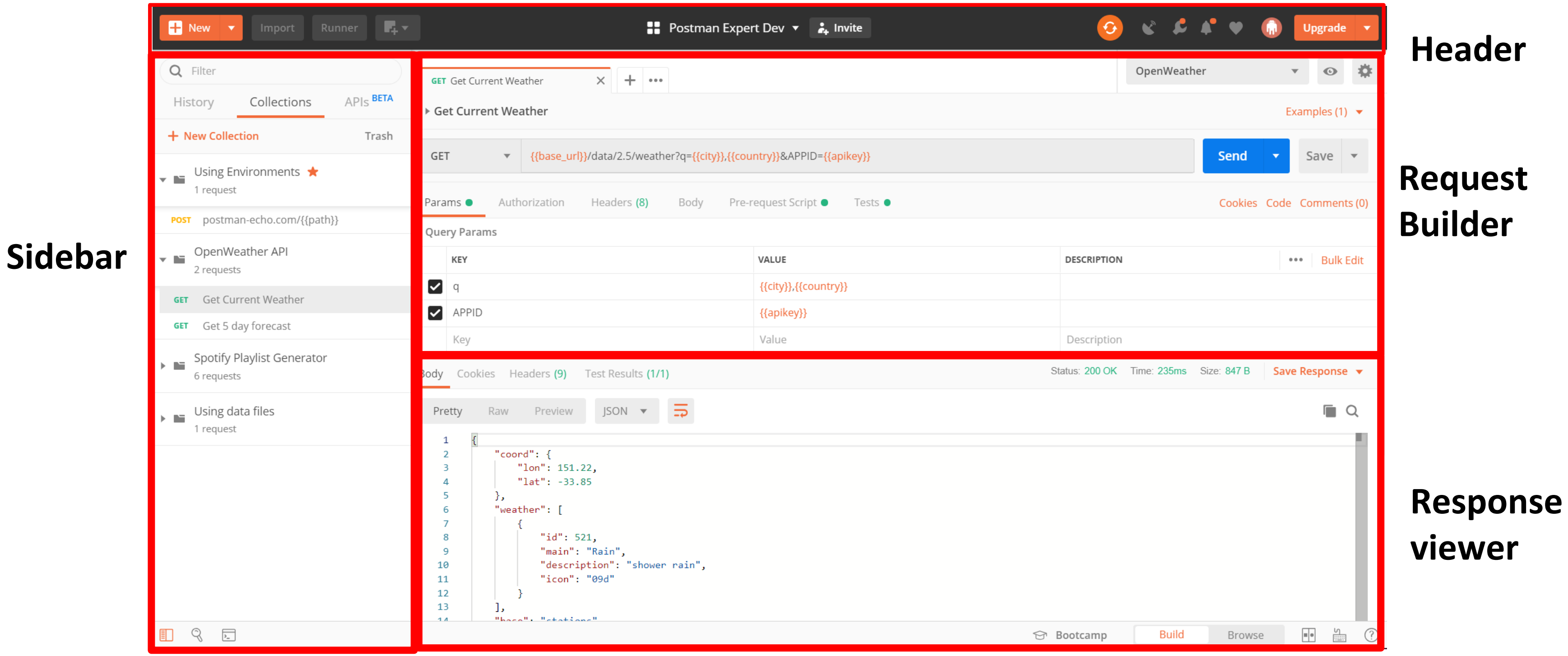


POSTMAN

Postman User Interface

POSTMAN

Postman Application UI



GET {{base_url}}/data/2.5/weather?q={{city}},{{country}}&APPID={{apikey}}

KEY	VALUE	DESCRIPTION
✓ q	{{city}},{{country}}	
✓ APPID	{{apikey}}	
Key	Value	Description

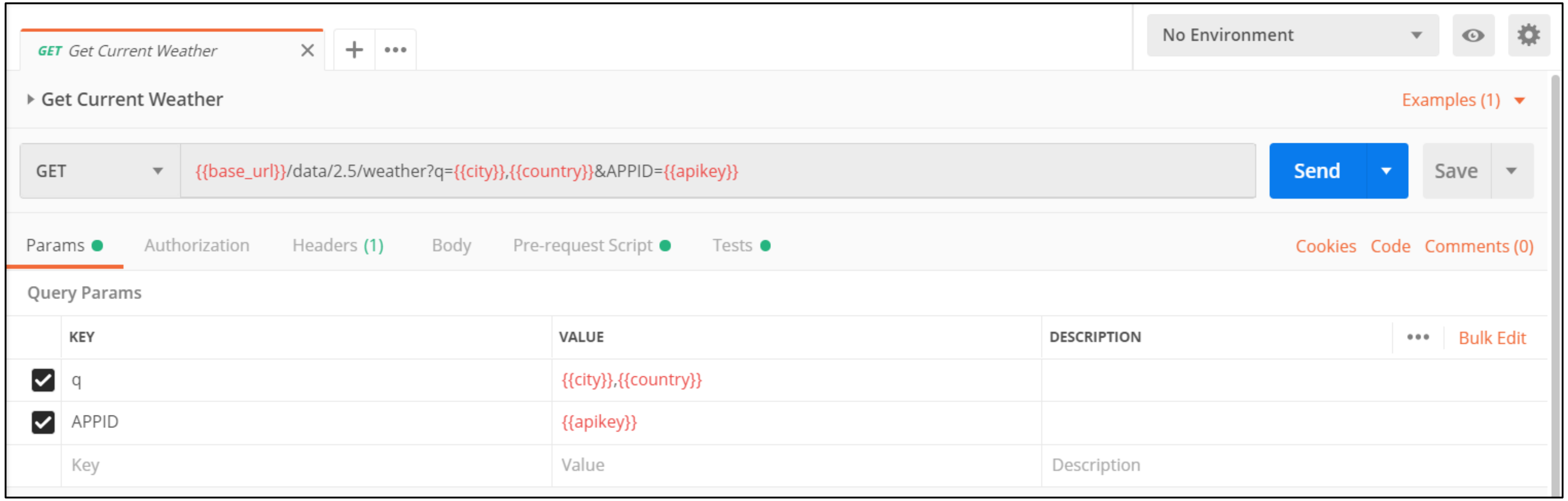
Status: 200 OK Time: 235ms Size: 847 B

```
{
    "coord": {
        "lon": 151.22,
        "lat": -33.85
    },
    "weather": [
        {
            "id": 521,
            "main": "Rain",
            "description": "shower rain",
            "icon": "09d"
        }
    ],
    "base": "stations",
```

POSTMAN

Request Builder

- The **Request Builder** contains all the elements needed to build a request



GET Get Current Weather

Get Current Weather

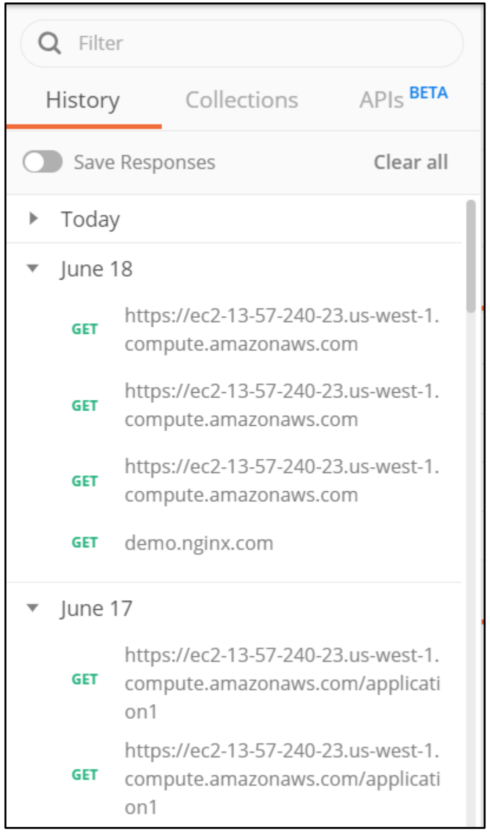
GET `{{base_url}}/data/2.5/weather?q={{city}},{{country}}&APPID={{apikey}}`

Query Params

	KEY	VALUE	DESCRIPTION
✓	q	{{city}},{{country}}	
✓	APPID	{{apikey}}	
	Key	Value	Description

Sidebar

- Contains the **History** of previous requests, the list of Collections and APIs
- Can be collapsed and expanded



GET https://ec2-13-57-240-23.us-west-1.compute.amazonaws.com
GET https://ec2-13-57-240-23.us-west-1.compute.amazonaws.com
GET https://ec2-13-57-240-23.us-west-1.compute.amazonaws.com
GET demo.nginx.com

GET https://ec2-13-57-240-23.us-west-1.compute.amazonaws.com/application1
GET https://ec2-13-57-240-23.us-west-1.compute.amazonaws.com/application1

POSTMAN

Module Review

Key Points:

- Requests are used to test API endpoints
- Collections help to organize requests
- Variables and Environments allow reuse of values
- Application UI main elements include the Request Builder and Sidebar

POSTMAN

Module 2
Creating & Sending API Requests

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Create a request
- Describe headers and parameters
- View a response
- Create/ Send GET requests
- Create/ Send POST requests
- Input API Authorization credentials

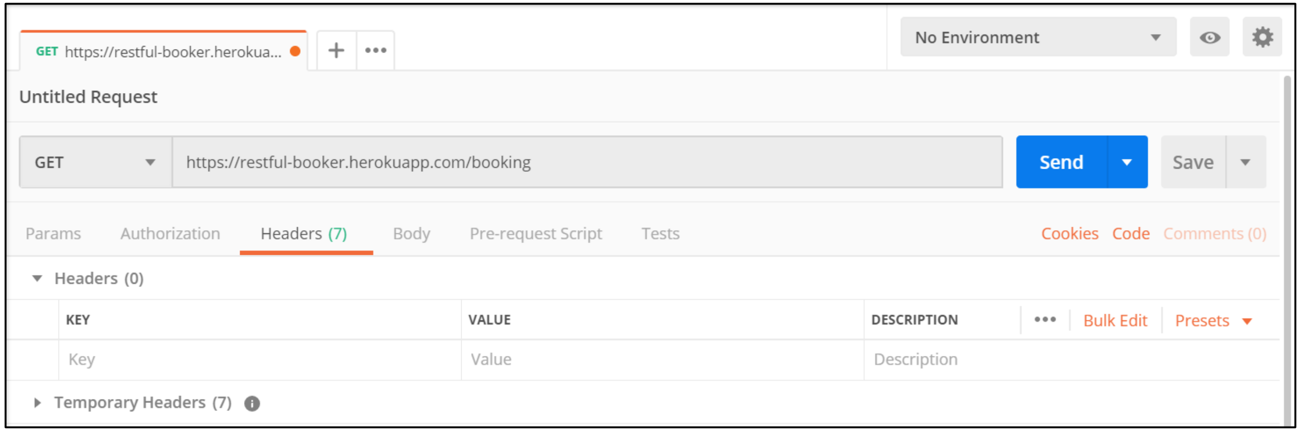
POSTMAN

Creating Requests

POSTMAN

Creating a Request

- Select **HTTP method.**
- Enter **URL** and press **Send**.
- Edit request body.
- Add request Headers.
- Edit Parameters.



GET https://restful-booker.herokua...

No Environment

Untitled Request

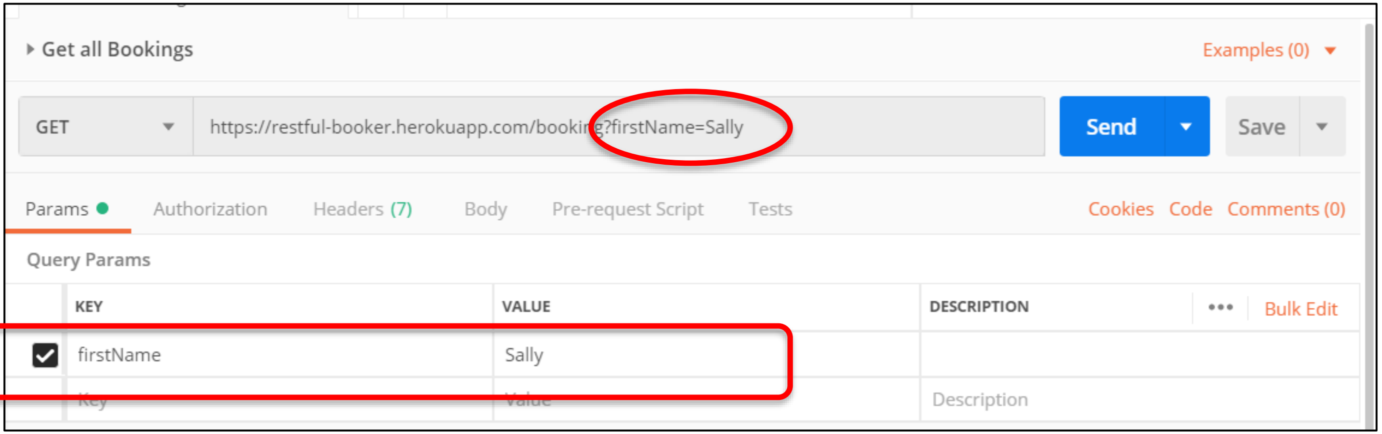
GET https://restful-booker.herokuapp.com/booking

KEY	VALUE	DESCRIPTION
Key	Value	Description

POSTMAN

Headers & Parameters

- Key value pairs are used to define the HTTP request **Headers**
- Auto-complete function when entering Headers
- Temporary headers are auto generated for every request
- **Parameters** can be entered in the request URI or in its own key value editor



Get all Bookings

GET https://restful-booker.herokuapp.com/booking?firstName=Sally

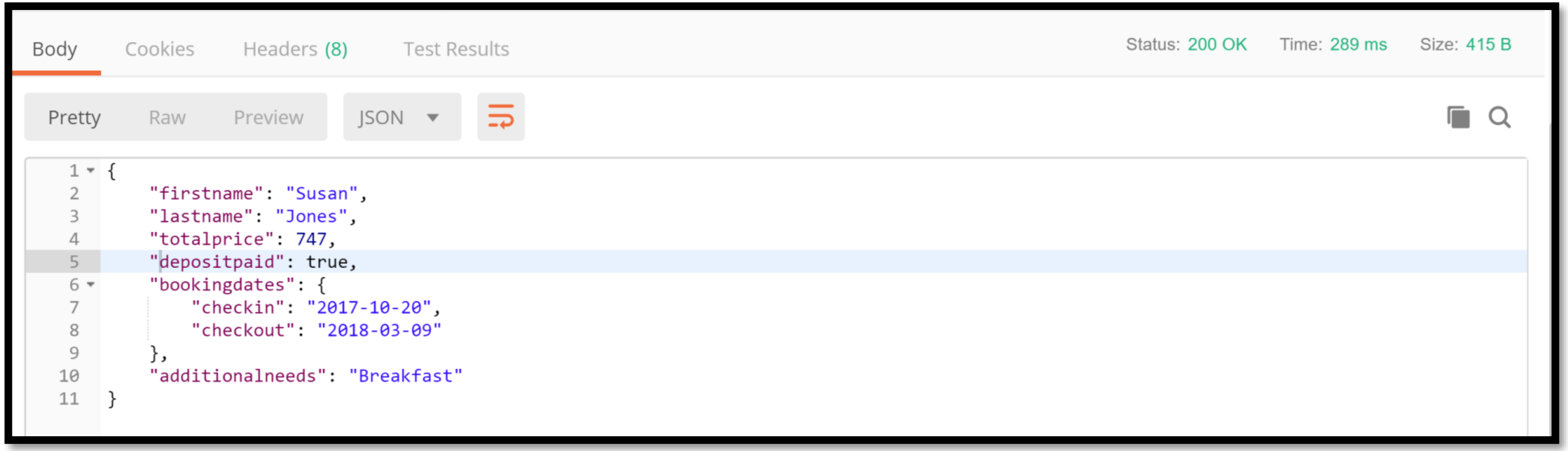
Query Params

	KEY	VALUE	DESCRIPTION
✓	firstName	Sally	
	Key	Value	Description

POSTMAN

Viewing Response

- Response body can be displayed in different formats
- Find feature to look for specific strings
- HTML responses can be previewed but without CSS rendering



```
{
    "firstname": "Susan",
    "lastname": "Jones",
    "totalprice": 747,
    "depositpaid": true,
    "bookingdates": {
        "checkin": "2017-10-20",
        "checkout": "2018-03-09"
    },
    "additionalneeds": "Breakfast"
}
```

POSTMAN

EXERCISE

EX2.1 – Create a GET Request

1. In the Request Builder, enter the **URL** .

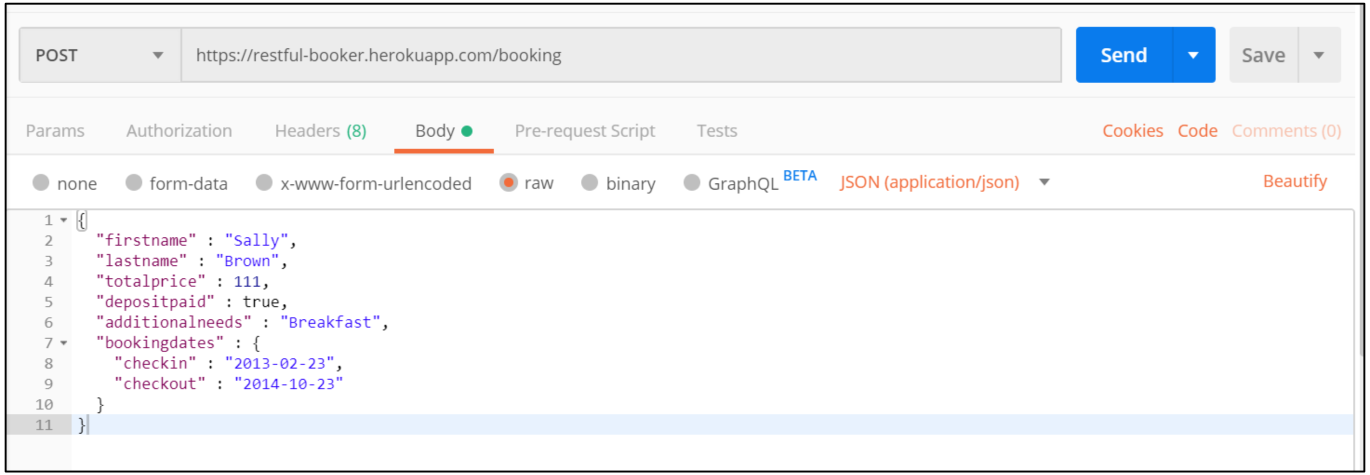
   https://restful-booker.herokuapp.com/booking

2. Select **GET** as the HTTP method, then press the **Send** button.
3. Check the response body. What do you see?
4. Now try the URL https://www.getpostman.com and press **Send.**
5. What do you notice in the response body?
6. Click on the **Preview** tab on the response and check the HTML render.

POSTMAN

POST Request

- For POST requests, data sent to API is defined in the request body as:
  - Form data
  - Raw text, JSON, XML, etc.
- Set the **Content-Type** Header to the type of content being sent to the server.



POST https://restful-booker.herokuapp.com/booking

Body: [✓ raw] JSON (application/json)

```
{
  "firstname" : "Sally",
  "lastname" : "Brown",
  "totalprice" : 111,
  "depositpaid" : true,
  "additionalneeds" : "Breakfast",
  "bookingdates" : {
    "checkin" : "2013-02-23",
    "checkout" : "2014-10-23"
  }
}
```

POSTMAN

EXERCISE

EX2.2 – Create a POST Request

1. In the Request Builder, enter the **URL** https://restful-booker.herokuapp.com/booking
2. Select **POST** as the method.
3. Click on the **Body** tab, select the **Raw** option and type in the JSON string shown on the right
4. Select **JSON** on the dropdown list and check that this has added the **Content-Type** header on the **Headers** tab
5. Press **Send**.

```
{
  "firstname" : "Sally",
  "lastname" : "Brown",
  "totalprice" : 111,
  "depositpaid" : true,
  "additionalneeds" : "Breakfast",
  "bookingdates" : {
    "checkin" : "2013-02-23",
    "checkout" : "2014-10-23"
  }
}
```

POSTMAN

EXERCISE

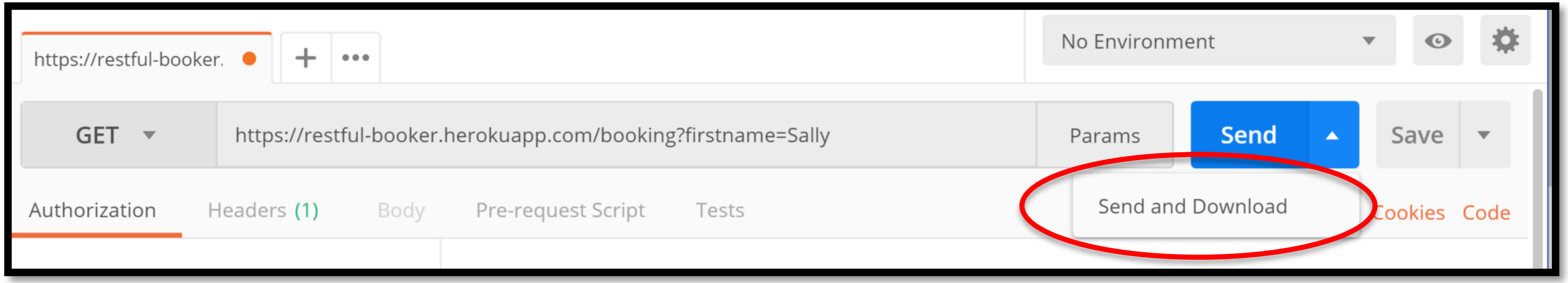
EX2.2 – Create a POST Request (cont’d)

6. Check the response body and make note of the “**bookingid**” attribute.
7. Change the request method to **GET** and click **Params** to open the key value editor for request Parameters.
8. Enter the key “**firstname**” with value “**Sally**”.
9. Press **Send**.
10. The result should display the “**bookingid**” observed in Step 6, along with additional **bookingids** .

POSTMAN

Send & Download

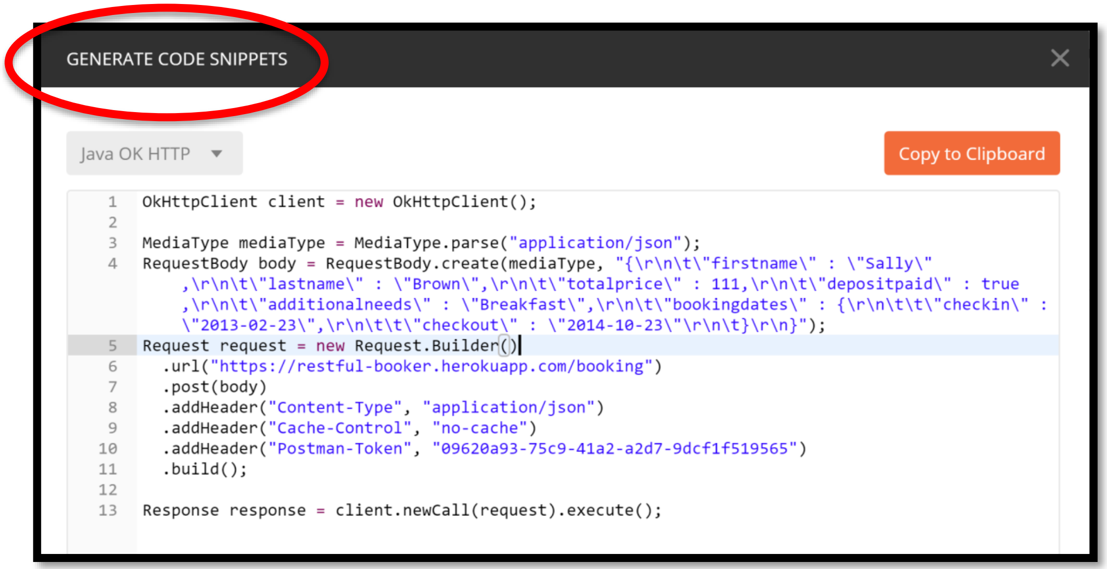
- Allows you to save the response to a file
- Useful if the service returns a response that is not JSON, HTML or text. e.g.
  - Image file
  - Audio and video files



https://restful-booker.herokuapp.com/booking?firstname=Sally

Generate Request Code Snippets

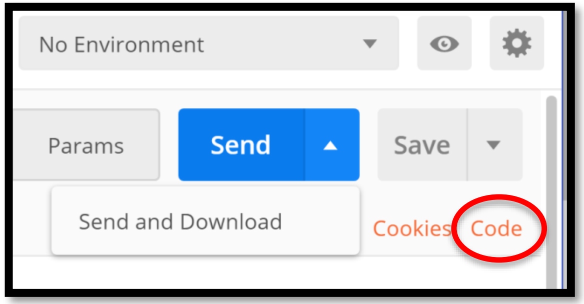
- Convert a request into a **Code Snippet** to aid application development
- Support for 15 languages and various libraries with certain languages



```
OkHttpClient client = new OkHttpClient();

MediaType mediaType = MediaType.parse("application/json");
RequestBody body = RequestBody.create(mediaType, "{\r\n\t\"firstname\" : \"Sally\"
    ,\r\n\t\"lastname\" : \"Brown\",\r\n\t\"totalprice\" : 111,\r\n\t\"depositpaid\" : true
    ,\r\n\t\"additionalneeds\" : \"Breakfast\",\r\n\t\"bookingdates\" : {\r\n\t\t\"checkin\" :
    \"2013-02-23\",\r\n\t\t\"checkout\" : \"2014-10-23\"\r\n\t}\r\n}");
Request request = new Request.Builder()
  .url("https://restful-booker.herokuapp.com/booking")
  .post(body)
  .addHeader("Content-Type", "application/json")
  .addHeader("Cache-Control", "no-cache")
  .addHeader("Postman-Token", "09620a93-75c9-41a2-a2d7-9dcf1f519565")
  .build();

Response response = client.newCall(request).execute();
```



POSTMAN

DEMO

Generate Request Code Snippets

POSTMAN

Inputting API Authorization Credentials

POSTMAN

API Authorization Methods

- Production APIs don’t let just anyone make a request
- Authentication credentials need to be supplied in order for an API to determine what resources you can access
- Postman app contains authorization helpers to simplify the process of supplying your API credentials

Authentication
Who you are
Authorization
What you can do

POSTMAN

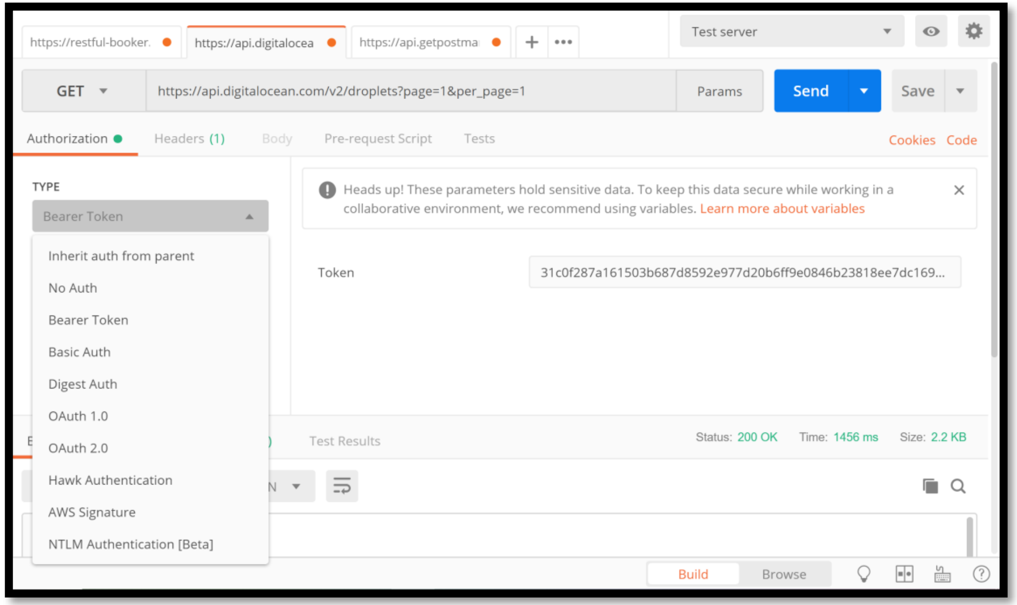
API Authorization Methods (cont’d)

- Many **Authentication methods** are supported by **Authorization helpers** such as:
  - OAuth
  - Basic Auth
  - Bearer Token
  - Digest
  - AWS Signature
  - More …

POSTMAN

API Authorization Methods (cont'd)

- Full details for each Authentication method at: https://www.getpostman.com/docs/v6/postman/sending_api_requests/authorization


GET https://api.digitalocean.com/v2/droplets?page=1&per_page=1

TYPE: Bearer Token

- Inherit auth from parent
- No Auth
- Bearer Token
- Basic Auth
- Digest Auth
- OAuth 1.0
- OAuth 2.0
- Hawk Authentication
- AWS Signature
- NTLM Authentication [Beta]

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

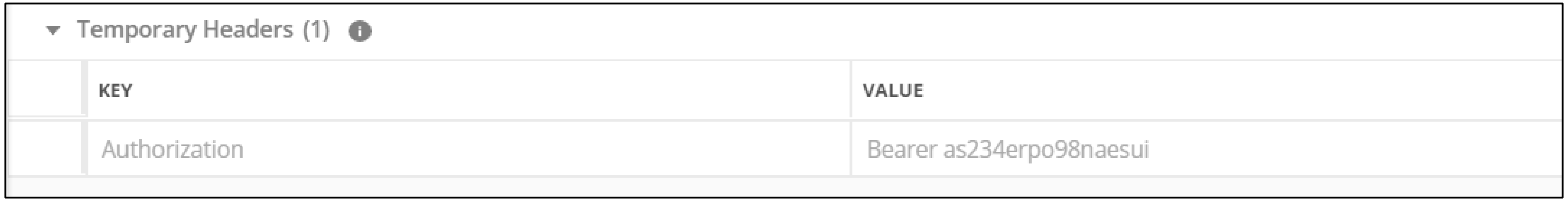
Token: 31c0f287a161503b687d8592e977d20b6ff9e0846b23818ee7dc169...

Status: 200 OK Time: 1456 ms Size: 2.2 KB

POSTMAN

Authorization Header

- Credentials specified in the Authorization helper are converted into temporary request headers
- Some APIs require a custom **Authorization Header** or query string parameter
  - **Example:** Postman API (https://api.getpostman.com/collections?apikey=96d02f....)



Temporary Headers (1)

KEY	VALUE
Authorization	Bearer as234erpo98naesui

POSTMAN

DEMO

API Authorization

POSTMAN

EXERCISE

EX2.3 – Authorize API Request

1. Login to your OpenWeatherMap account on the web browser. (https://home.openweathermap.org/users/sign_in)
2. Click on the API Key tab in your account profile and copy the API key to the clipboard
3. Switch to the Postman desktop app and create a new request with the request URL api.openweathermap.org/data/2.5/weather?q=Sydney&APPID=<your api key>
4. Send the request and observe the response.

POSTMAN

Module Review

Key Points:

- Use Request Builder to define request headers, parameters, body, etc.
- Use Authorization helper to specify API credentials
- View formatted JSON responses
- Binary responses can be downloaded
- Option to generate request code snippet for application development

POSTMAN

Module 3:
Organizing Requests into Postman Collections

POSTMAN

Module Objectives

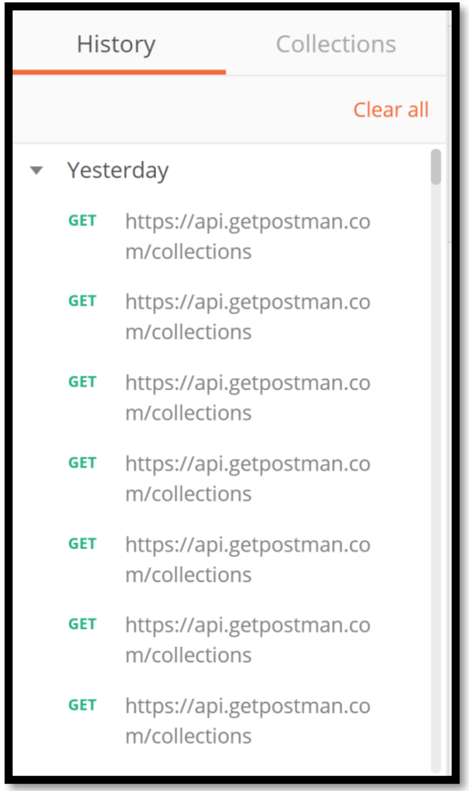
After completing this module, you will be able to:

- Describe request History
- Create a Collection
- Save requests into a Collection
- Describe Collection Authorization
- Describe Collection use cases
- Organize request Collections in Folders
- Perform other Collection operations
- Export and Import Collections

POSTMAN

Request History

- Sent requests are recorded in the request **History**
- No cohesive organization
- Difficult to find a certain request
- Use Collections for better organization



GET https://api.getpostman.com/collections

GET https://api.getpostman.com/collections

GET https://api.getpostman.com/collections

GET https://api.getpostman.com/collections

GET https://api.getpostman.com/collections

GET https://api.getpostman.com/collections

GET https://api.getpostman.com/collections

POSTMAN

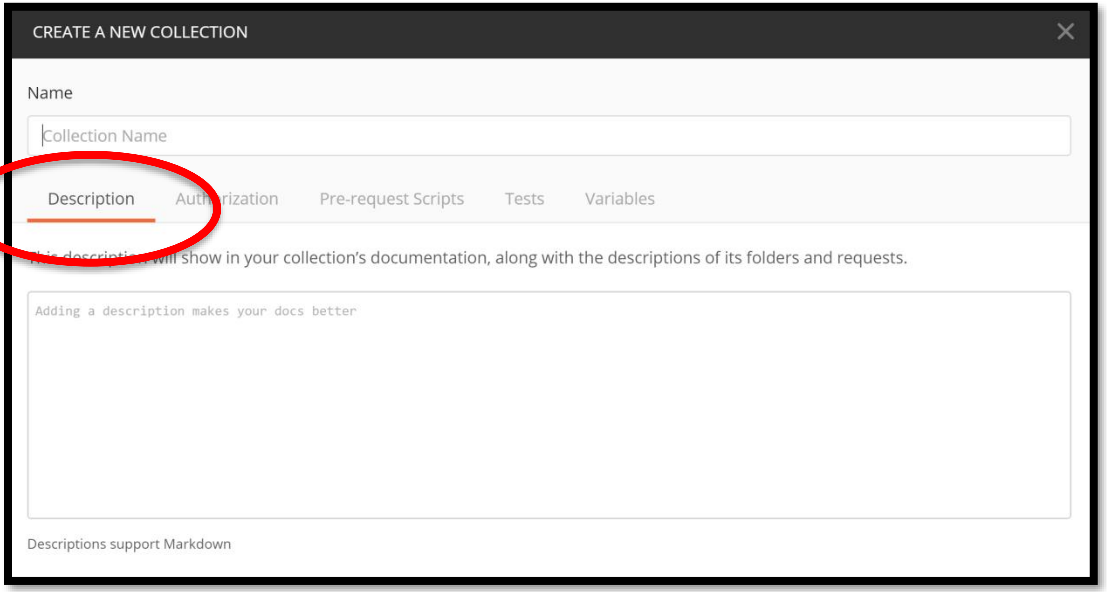
Using Collections

- Most of Postman’s features depend on **Collections**:
  - Mocks
  - Tests
  - Documentation
  - Collaboration through Workspaces
  - Monitoring

POSTMAN

Creating a Collection

- Collections can be created from scratch or when saving a request
- Write a good description for your Collection for documentation purposes


CREATE A NEW COLLECTION

Name

Collection Name

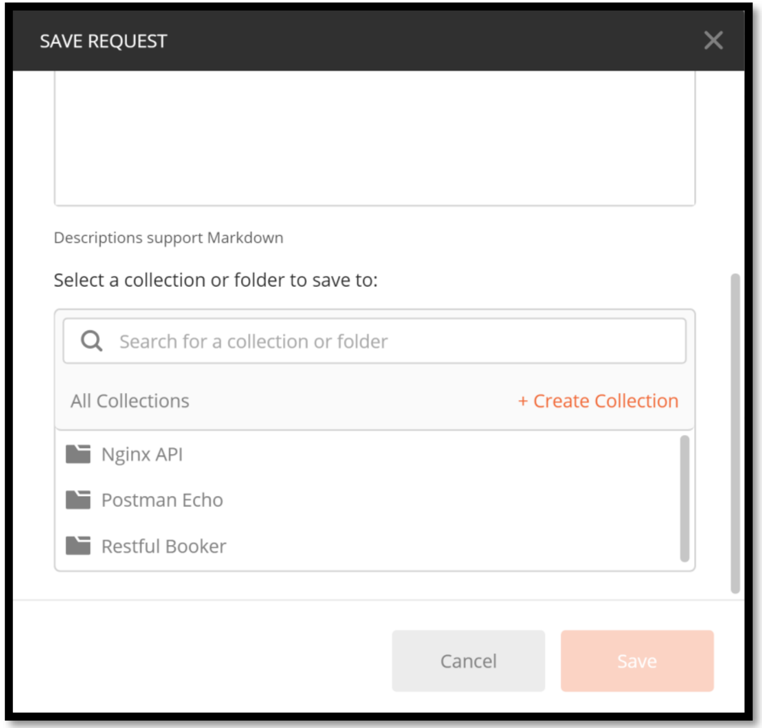
This description will show in your collection's documentation, along with the descriptions of its folders and requests.

Adding a description makes your docs better

Descriptions support Markdown

Saving a Request

- Requests can be saved into an existing Collection
- Can create a new Collection in the process of saving a request
- Request description helps make documentation look better


SAVE REQUEST

Descriptions support Markdown

Select a collection or folder to save to:

POSTMAN

EXERCISE

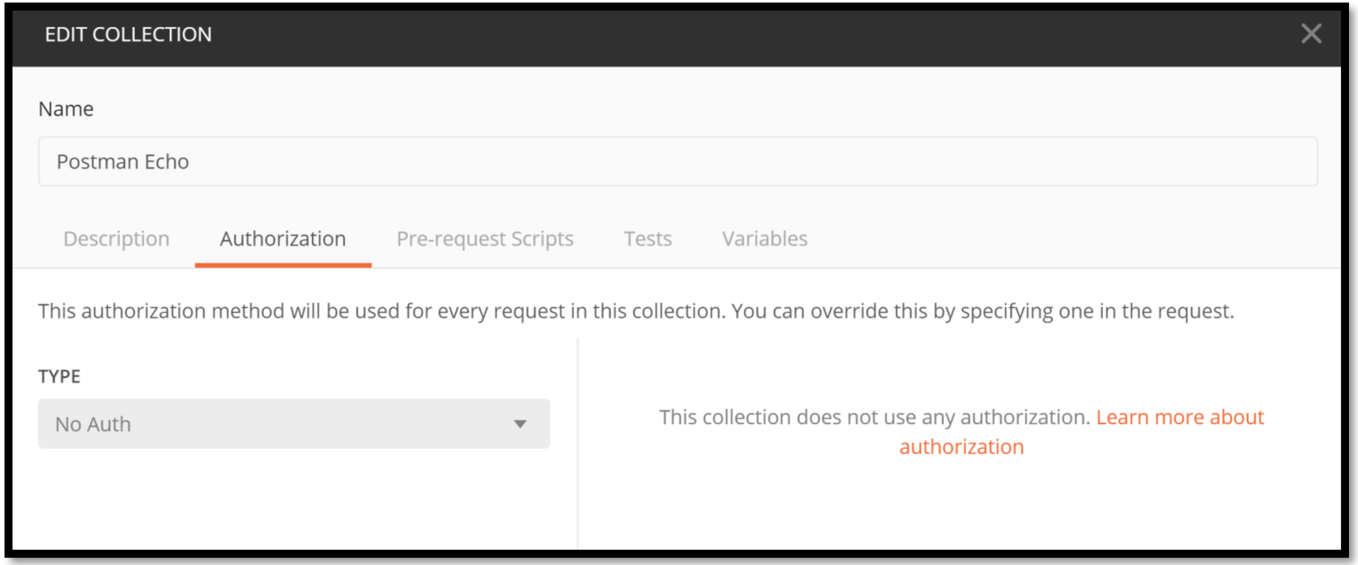
EX3.1 – Create a Collection

1. Click on the **Collection** tab on the sidebar.
2. Click **New Collection** to bring up the Create Collection screen.
3. Type in **Restful Booker** in the **Collection Name** field.
4. Click the **Create** button.
5. Click the **History** tab and look for your previous GET request to the endpoint https://restful-booker.herokuapp.com/booking.
6. Save the request to the newly created **Restful Booker** Collection with the following details:
   - **Name:** Get all Bookings
   - **Description:** Get a list of all bookings

POSTMAN

Collection Authorization

- Defines an **Authorization method** to be used for every request in the Collection
- Individual requests can still have their own independent authorization details


EDIT COLLECTION

Name

Postman Echo

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

TYPE

No Auth

This collection does not use any authorization. Learn more about authorization

POSTMAN

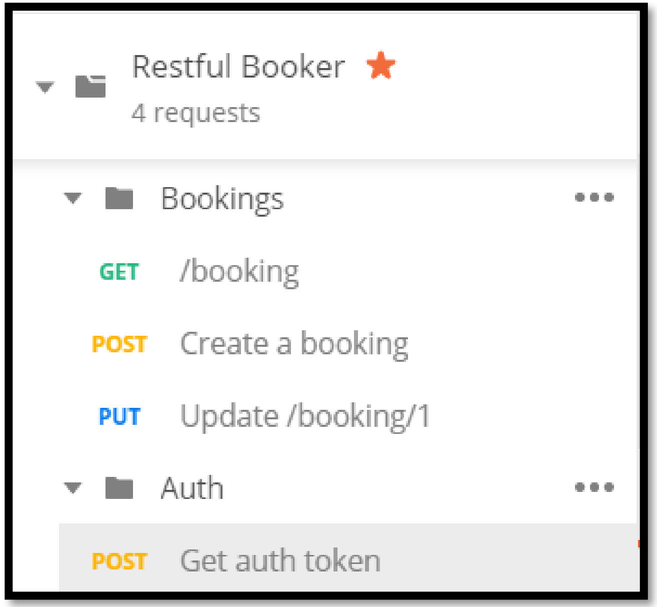
Collection Use Cases

- Organize your APIs into Collections
- Single API = Single Collection
- Larger Collections should be split into multiple Collections or organized with Folders

POSTMAN

Organizing with Folders

- Requests in a Collection can be grouped into **Folders**
- Group of requests which can be run individually, sequentially or conditionally
- Logical way to organize Collections would be to base them on:
  - Related endpoints
  - Steps in a process
  - Common authorization
  - Request method
- Each Folder can have a common Authorization configuration for all requests


Restful Booker
4 requests
Bookings
GET /booking
POST Create a booking
PUT Update /booking/1
Auth
POST Get auth token

POSTMAN

EXERCISE

EX3.2 – Create Folders

1. Create two Folders called **Bookings** and **Auth** in the **Restful Booker** Collection.
2. Move the **Get All Bookings** request into the Bookings Folder.
3. Use the **History** tab to find the **POST** request that was created earlier in the course.
4. Save the request to your **Restful Booker** Collection under the **Bookings** Folder. Give it the name **Create New Booking**

POSTMAN

EXERCISE

EX3.2 – (cont'd)

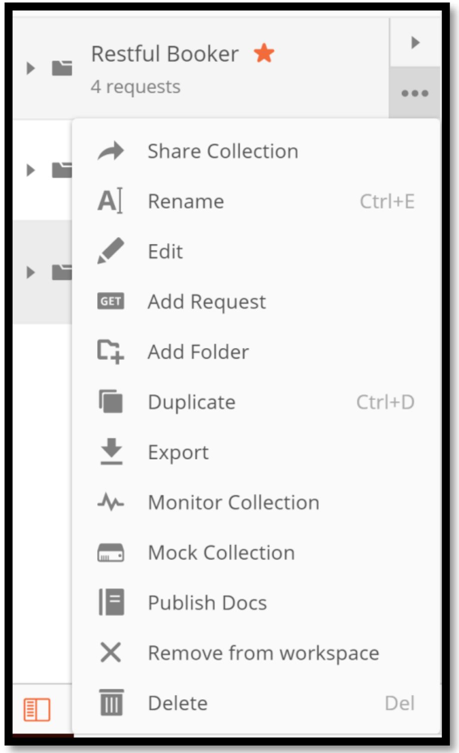
5. Create a new POST request to the end point https://restful-booker.herokuapp.com/auth
6. Use the **Raw** option to configure the request body as shown on the right.
7. Send the request and verify that you received a string token as the response.
8. Save the request to the "**Auth**" folder in your **Restful Booker** Collection. Give it the name **Get Auth Token**

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

POSTMAN

Other Collection Operations

- **Delete** sends the Collection to the trash area of your account
- **Favourite** Collections to move them to the top of the list
- **Filter** searches for a particular request
- **Add** scripts for dynamic behaviour
- **Run** the Collection to test the whole API
- **Export** and **Import**



POSTMAN

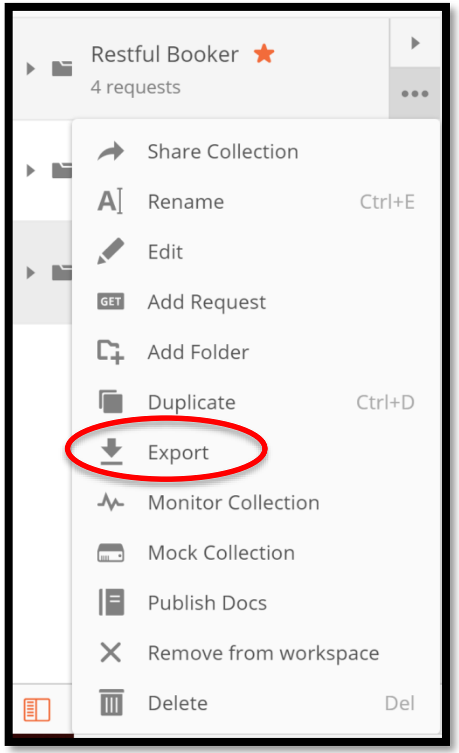
DEMO

Performing Other Collection Operations

POSTMAN

Exporting & Importing Collections

- Exports a Collection into a JSON file
- File contains all data and metadata needed to Import a Collection
- Simple way to share a Collection
- For a more detailed dive into the different data formats, see http://blog.getpostman.com/2015/06/05/travelogue-of-postman-collection-format-v2



Module Review

Key Points:

- Collections help organize requests to each API
- Folders provide further level of breakdown of requests in a Collection
- Can define common request authorization credentials at Folder and Collection level
- Collections can be exported and imported

POSTMAN

Module 4
Configuring Postman Variables & Environments

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Describe different levels of scope for Variables
- Describe where Variables can be used
- Configure Postman Variables
- Configure Postman Environments
- Describe best practices for using Environments
- Describe the benefits of Variables
- Describe Global Variables

POSTMAN

Configuring Postman Variables

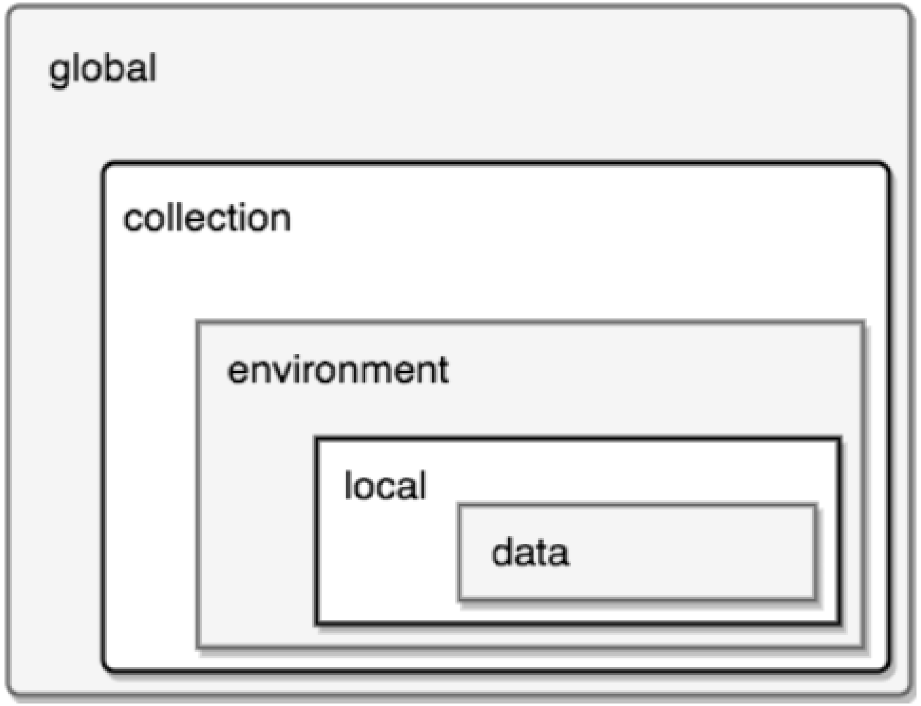
POSTMAN

Levels of Scope for Variables

- **Global** – this Variable has a value within a particular Workspace
- **Collection** – this Variable is specific to an individual Collection
- **Environment** – provides portable scope for Variables
- **Local** – is used during the execution of scripts
- **Data** – is used with Collection runner

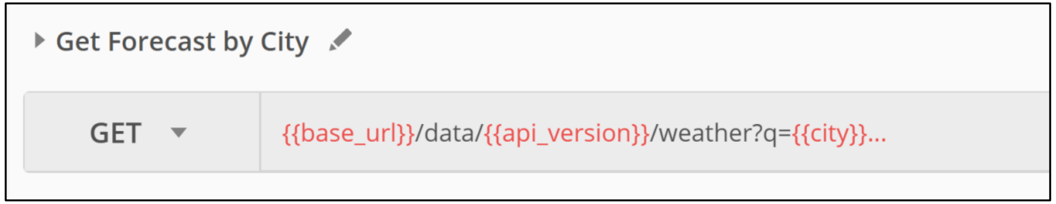
POSTMAN

Levels of Scope for Variables (cont'd)



Variables

- Variables can be used in most areas of the Postman Request Builder that uses text:
  - URL
  - Parameters
  - Header values
  - Form data / url-encoded values
  - Raw body content
- Syntax: {{variable_name}}


Get Forecast by City

GET

```
{{base_url}}/data/{{api_version}}/weather?q={{city}}...
```

POSTMAN

EXERCISE

EX4.1 – Define Variables

1. Use the history view and open the OpenWeather API request that was created in exercise 3.3
2. Save the request into a new Collection called **OpenWeather API.** Name the request **Get Weather by City**
3. Replace the "api.openweathermap.org" portion of the URL with **{{base_url}}**
4. Replace the "2.5" portion of the URL with **{{api_version}}**
5. Replace "Sydney" with **{{city}}**
6. Replace the part where you specify your API key with {{api_key}}
7. The final request URL should be
   {{base_url}}/data/{{api_version}}/weather?q={{city}}&APPID={{api_key}}
8. Save the request

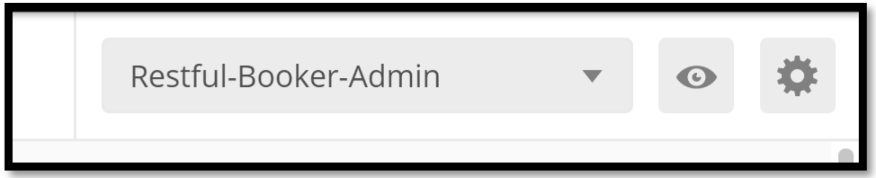
POSTMAN

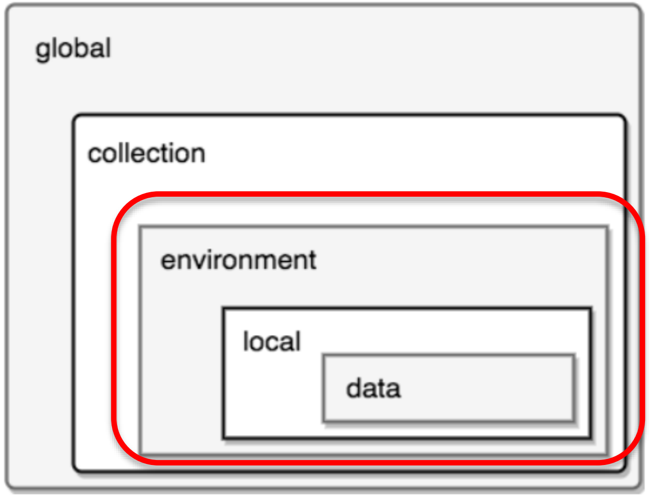
Configuring Postman Environments

POSTMAN

Environments

- An **Environment** is a portable scope for Variables
- Environments can be shared, synced, and exported
- Data in an Environment is secured
- Environments are ideal for data that needs to be moved around





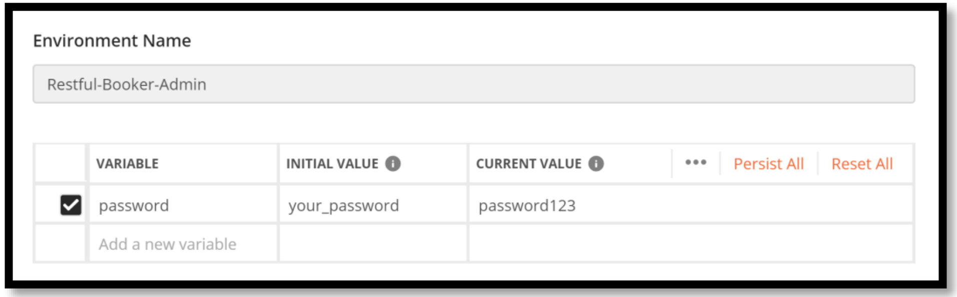
Session Variables

- **Sessions** provide a way to store variable values that are unique to every Postman user and stays local to that users instance of Postman
- Applies to Global, Collection and Environment variables
- Sessions are useful when sharing an Environment or Collection with Variables that contain sensitive data
- Enables users working on a shared Collection within a team to make requests using their own variable values without overwriting the original Environment value

POSTMAN

Defining Session Variables

- A variable's **INITIAL VALUE** will sync across the team workspace, collection or environment
- Use placeholder text on the INITIAL VALUE to avoid giving away sensitive data
- Users then set the variable value using the **CURRENT VALUE** field
- The **CURRENT VALUE** remains local to your instance of Postman
- The **CURRENT VALUE** can be persisted; this will replace the **INITIAL VALUE** with the **CURRENT VALUE** for that workspace, collection or environment



Environment Name

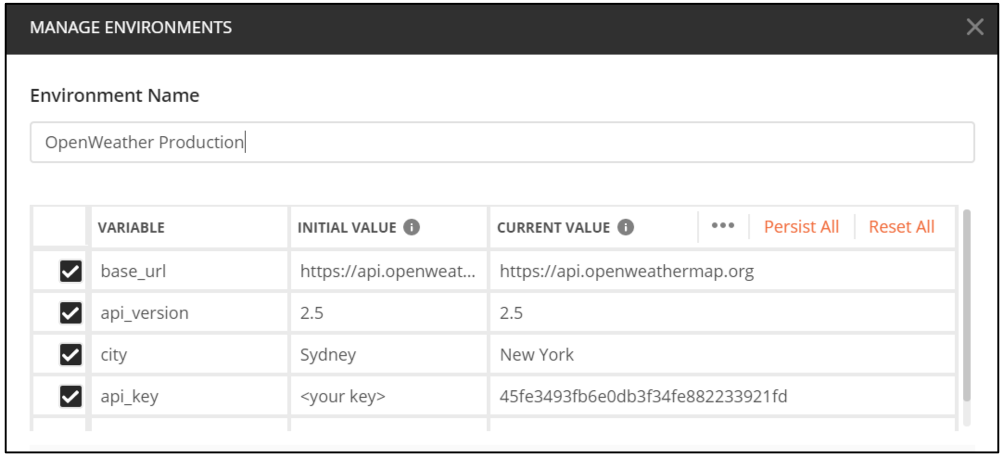
Restful-Booker-Admin

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	password	your_password	password123
	Add a new variable		

POSTMAN

Defining Environment Variables

- Initial value is shared with your team if you share the Environment
- Current value is used when sending a request and can be changed for each request
- Persist option to replace initial value with current value


MANAGE ENVIRONMENTS

Environment Name

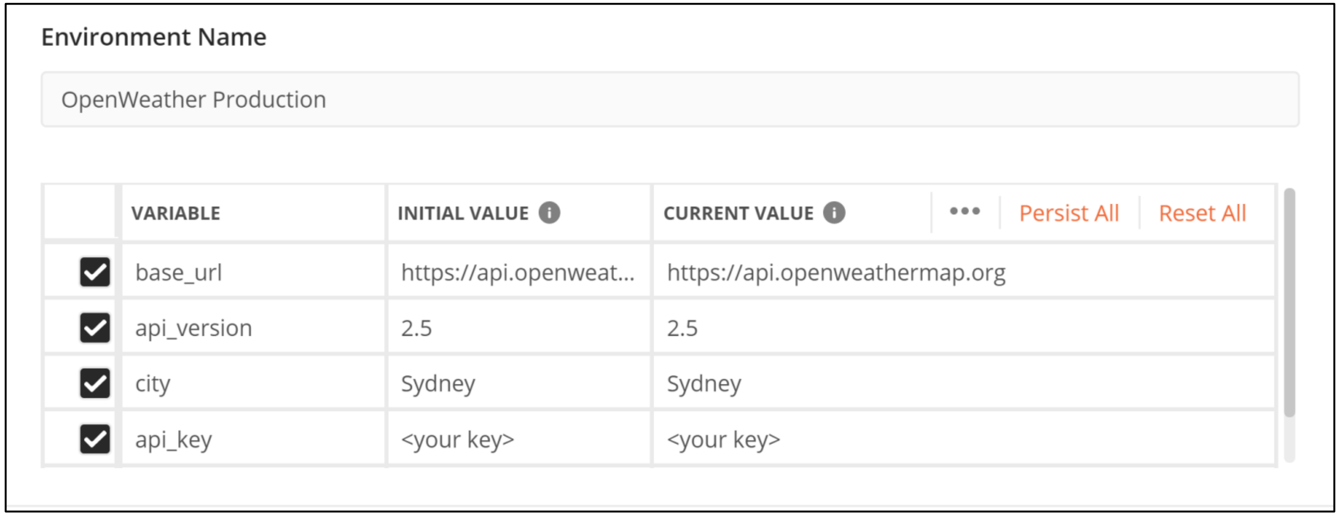
OpenWeather Production

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	base_url	https://api.openweat...	https://api.openweathermap.org
✓	api_version	2.5	2.5
✓	city	Sydney	New York
✓	api_key	<your key>	45fe3493fb6e0db3f34fe882233921fd

POSTMAN

EX4.2 – Create an Environment

1. Click the **Manage Environments** button.
2. Add a new Environment called **OpenWeather Production**.
3. Define four **Variables** as shown.



Environment Name

OpenWeather Production

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	base_url	https://api.openweat...	https://api.openweathermap.org
✓	api_version	2.5	2.5
✓	city	Sydney	Sydney
✓	api_key	<your key>	<your key>

POSTMAN

EXERCISE

EX4.2 – (cont’d)

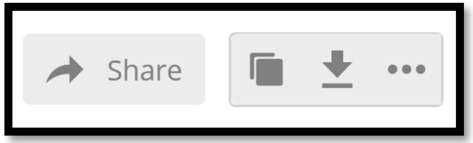
4. Click **Update** to update the Environment.
5. Select your newly created **OpenWeather Production** environment in the Environment drop-down list of the Request Builder.
6. Click the **Environment Quick Look** button.
7. Input your OpenWeather API key into the “current value” of the **api_key** variable.
8. Send the **Get Weather by City** request and observe the response.
9. Change the value of the **city** variable to “Perth” and send the request again. You should see differences in the response data.

POSTMAN

EXERCISE

EX4.2 – (cont'd)

10. Click the **Manage Environments** button and duplicate the **OpenWeather Production** Environment.



11. You should now see a new Environment called **OpenWeather Production Copy**. Click on this Environment.
12. Change name of the Environment to **OpenWeather Staging**.

POSTMAN

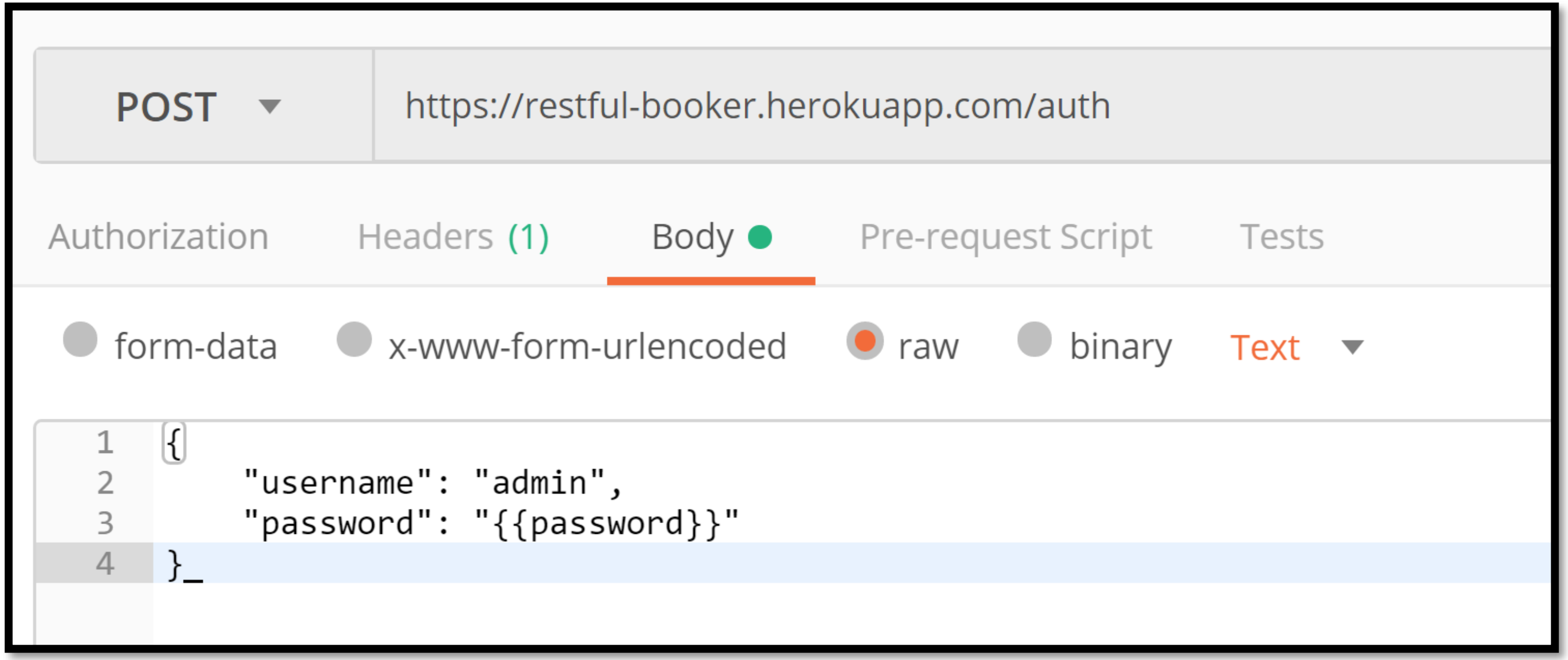
EXERCISE

EX4.2 – (cont'd)

13. Modify the current value of the "**base_url**" variable to "https://staging-api.openweathermap.org"
14. Click on the row drop-down and click on **Persist**. This will modify the initial value to the current value.
15. On the Request Builder, select the **OpenWeather Staging** Environment and send the request.

POSTMAN

Form Usage Example



POST https://restful-booker.herokuapp.com/auth

[✓ raw] Text

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

POSTMAN

EXERCISE

EX4.3 – Define Variables in Request Body

1. Open the “**Get Auth Token**” request from your Restful Booker collection.
2. Modify the body of the request to match the example on the right side of this slide.
3. Add an environment called “Restful-Booker Admin”.
4. Define a variable called “password” in the environment and set the **CURRENT value** to “**password123**”.
5. Select the “**Restful Booker Admin**”. environment and send the request.
6. Verify that the response is an auth token.

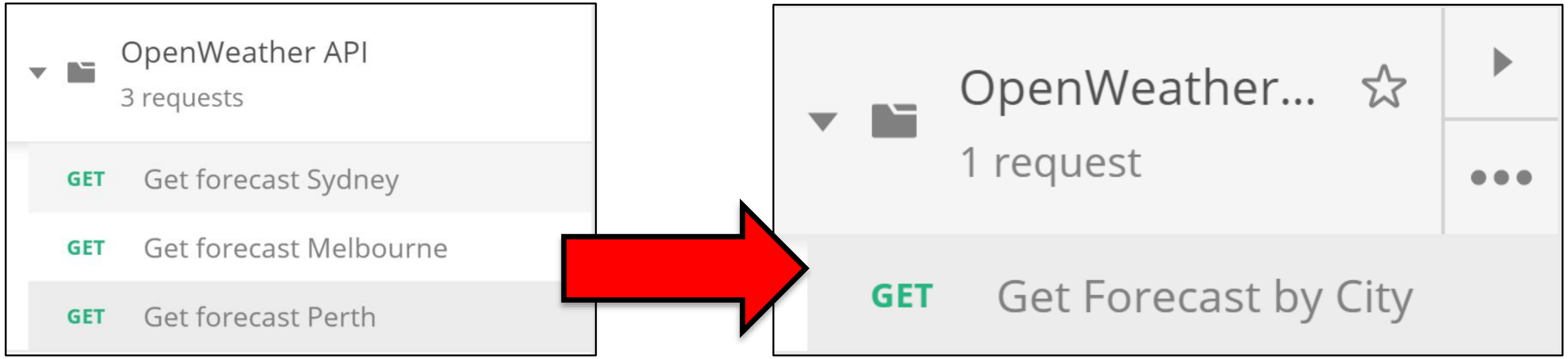
```
{
    "username": "admin",
    "password": "{{password}}"
}
```

POSTMAN

Best Practices for Using Environments

- Environments should be used for data that changes frequently but needs to be shared across people
- Environments are encrypted at rest and at storage and therefore passwords, secrets and API keys should be stored in an Environment
- Use Variables to minimise the number of requests in a Collection
- Avoid having requests of the same endpoint but a different parameter

Benefits of Using Variables


OpenWeather API
3 requests
GET Get forecast Sydney
GET Get forecast Melbourne
GET Get forecast Perth

OpenWeather...
1 request
GET Get Forecast by City

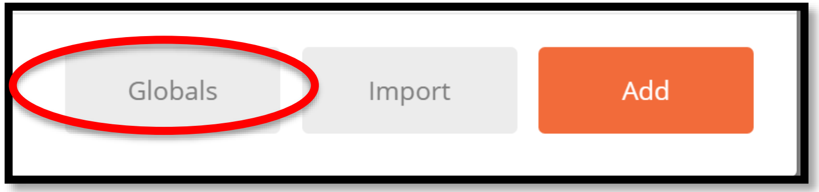
- {{url}}/data/{{api_version}}/weather?q=Sydney
- {{url}}/data/{{api_version}}/weather?q=Melbourne
- {{url}}/data/{{api_version}}/weather?q=Perth

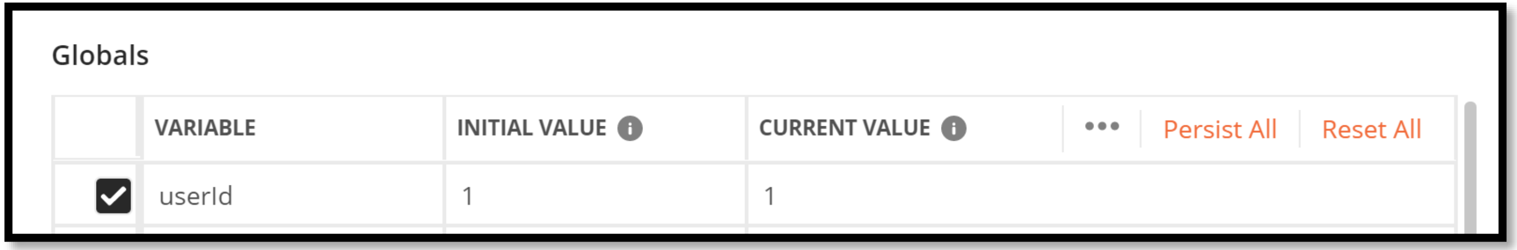
- {{url}}/data/{{api_version}}/weather?q={{city}}

POSTMAN

Global Variables

- **Global** variables contain a value that is defined across the entire Workspace
- Good for storing information that does not change but is used repetitively
- To define:
  - Click **Manage Environments** button.
  - Then click **Globals**.




Globals

	VARIABLE	INITIAL VALUE	CURRENT VALUE	••• Persist All Reset All
✓	userId	1	1	

POSTMAN

Module Review

Key Points:

- Variables minimize the number of requests that need to be defined
- Environments variables are ideal for changing data in a request
- When the same variable is defined in multiple scopes, the value of the narrower scope will be used (Environment over Global)

POSTMAN

Module 5
Performing Basic API Testing with JavaScript

Module Objectives

After completing this module, you will be able to:

- Describe basic API testing
- Describe Postman API testing
- Write a basic test in JavaScript
- Describe/Use test Assertions
- Describe/Use Variables in scripts
- Describe/Use Code Snippets
- Describe Postman Sandbox API

POSTMAN

Basic API Testing

Why test APIs?

- Interface to application logic
- APIs form a critical contract in the systems we build
- Bugs in an API can affect many API consumers
- Testing helps to identify failures earlier in the development cycle and fix them before the API consumers are affected

POSTMAN

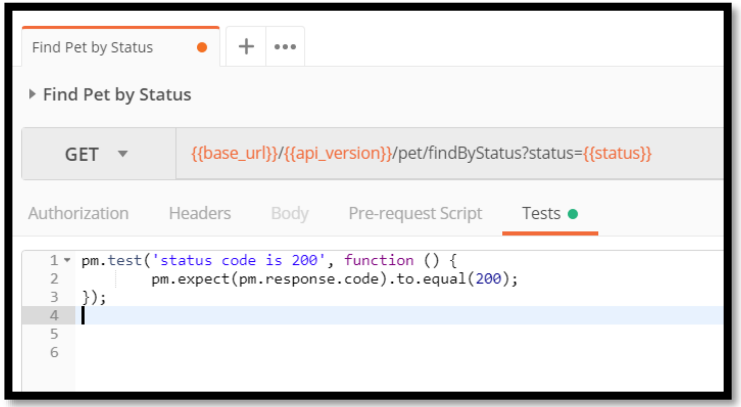
What is being tested?

1. **Functionality**
    - Check whether API behaves as expected
    - Send request and verify response
2. **Performance**
    - Verify that API gives a response within a certain time constraint
3. **Reliability**
    - Verify that the API is up and available for consumers
4. **Security**
    - Ensure that API does not allow unauthorised access and usage
    - For example, check for permissions of different sets of users

POSTMAN

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



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

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

POSTMAN

Writing a Basic API Test in JavaScript

POSTMAN

Writing a Test

- Test cases are defined by the **pm.test()** function
- Each test requires a **test name** and a set of **test assertions**

First argument is the test name, second argument is a function which will contain our assertions

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

POSTMAN

Test Assertions

- A **Test Assertion** is a Boolean expression at a specific point in a program which will be true unless the program behaves in a different way to what we were expecting
- Assertions are expressions which encapsulate testable logic
- Assertion functions are used to test expected behavior

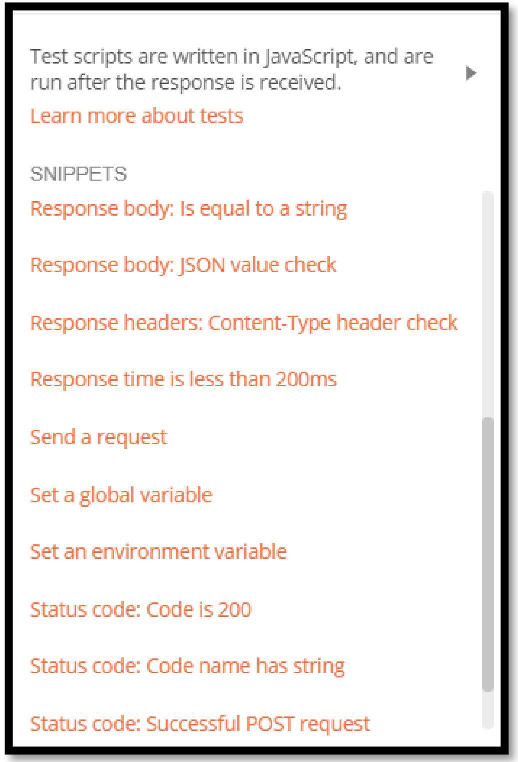
Test Assertions (cont'd)

- Use **pm.expect()** function
- Generic assertion function based on **ChaijS** expect BDD library (http://www.chaijs.com/api/bdd/)
- Handles assertions of data from a response or variables.
- For example:
  - Check the status code of a response
  - Check for values of JSON fields
  - Check for response header values

POSTMAN

Code Snippets

- Snippets of code for commonly used test logic
- Click on line to automatically append the JavaScript code



Test scripts are written in JavaScript, and are run after the response is received.
Learn more about tests

SNIPPETS
Response body: Is equal to a string
Response body: JSON value check
Response headers: Content-Type header check
Response time is less than 200ms
Send a request
Set a global variable
Set an environment variable
Status code: Code is 200
Status code: Code name has string
Status code: Successful POST request

POSTMAN

EXERCISE

EX5.1 – Write a Test

1. Open the **Get Weather by City** request from the OpenWeather API Collection
2. Click on the **Test** tab and look for the code snippet called **"Status code: Code is 200"**
3. Click on the snippet to insert the code
4. Send the request and check the **Test Results** tab to ensure that the test case has passed.

POSTMAN

EXERCISE

EX5.2 – Use Test Assertions

1. Open **Tests** tab of the **Get Forecast by City** request.
2. Look for the code snippet called **“Response body: JSON value check”** and insert the code.
3. Name the test case **“Check correct city”**
4. Change the **pm.expect()** function to the following:

   ```
   pm.expect(jsonData.name).to.eql("Sydney");
   ```
5. Select the **OpenWeather Production** environment and make sure the value of the **“city”** variable is set to “**Sydney**”.
6. Send the request and verify that both test cases pass.
7. Change the value of the environment **“city”** variable to “**Perth**” and send the request again.
8. What do you notice in the test results and why is that the case?

Using Variables in Testing

POSTMAN

Variables in Scripts

Get the value of the environment variable “Status”

```
pm.environment.get("status");
```

Set the value of an environment variable

```
pm.environment.set("variable_key", "variable_value");
```

Get the value of a global variable

```
pm.globals.get("variable_key");
```

Set the value of a global variable

```
pm.globals.set("variable_key", "variable_value");
```

POSTMAN

EXERCISE

EX5.3 – Use Variables in Testing

1. Go to the **Check correct city** test that was defined in Exercise 6.2.
2. Modify the **pm.expect()** function so that we assert that the value of the **“name”** key in the JSON data is equal to the value that we set on our environment **“city”** variable.

   Hint: Use the code snippet to “get an environment variable”
4. Send the request and verify that both test cases pass.
5. Change the value of the environment **“city”** variable to “**New York**” and send the request again.
6. Verify that both test cases pass.

POSTMAN

Postman Sandbox API

- JavaScript execution environment that is available when writing pre-request and test scripts
- Code is executed inside the sandbox
- Full API reference at https://www.getpostman.com/docs/v6/postman/scripts/postman_sandbox_api_reference

POSTMAN

Module Review

Key Points:

- Test APIs for functionality, performance, reliability and security
- JavaScript code is executed against the response received from a request
- Test the response by checking values, status codes, headers, etc.

POSTMAN

Module 6
Collaborating Through Postman Workspaces

POSTMAN

Module Objectives

After completing this module, you will be able to:

- Describe Postman Workspaces
- Create Workspaces
- Browse Workspaces
- Use Workspaces to share and organize

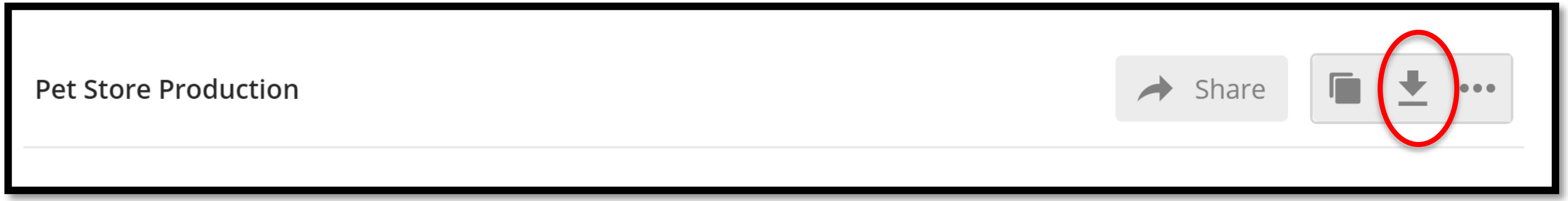
POSTMAN

Postman Workspaces

POSTMAN

Basic Collaboration

- Collections can be exported to a JSON file and imported
- Environments can be downloaded into a file and imported
- Useful for sharing Collections and Environments with team members
- Very basic method of collaboration
- No syncing occurs



POSTMAN

Postman Workspaces

- A view of all Postman elements:
  - Collections
  - Environments
  - Mocks
  - Monitors
  - Integrations
  - Activity feed
  - History, pre-set Headers, global Variables
- Collections and Environments can be shared between Workspaces

POSTMAN

Personal vs Team Workspaces

- **Personal Workspaces**
  - Workspace that only you can see
  - Unlimited personal Workspaces allowed if you have a Postman account
- **Team Workspaces**
  - Can be shared with members of your team
  - Invite users to the Workspace and grant them permissions to view and/or edit

POSTMAN

Why use Workspaces?

- Personal organization
- Team organization
- Source of truth
- Team permissions
- Discovery
- Up to data activity feed
- Real time debugging with history

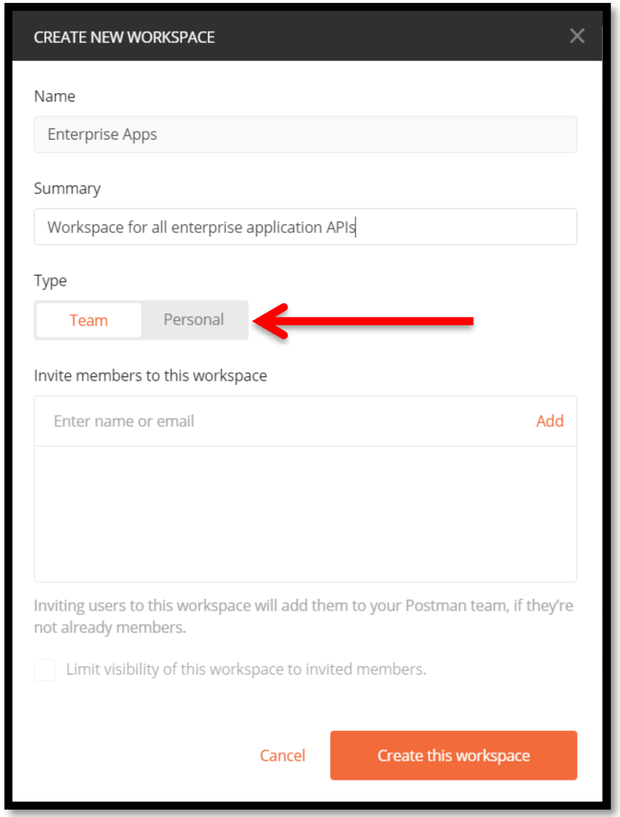
POSTMAN

Creating Workspaces

POSTMAN

Creating a Workspace

- Select a Workspace **Type** (**Team** or **Personal**)
- Creating a **Team Workspace** gives you the option of inviting users
- Users can be invited to a Personal Workspace; it then converts to a Team workspace.


CREATE NEW WORKSPACE

Name
Enterprise Apps

Summary
Workspace for all enterprise application APIs

Type
[✓ Team] Personal

Invite members to this workspace
Enter name or email — Add

Inviting users to this workspace will add them to your Postman team, if they're not already members.

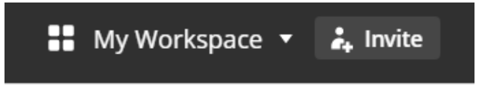
Limit visibility of this workspace to invited members.

Cancel | Create this workspace

POSTMAN

EXERCISE

EX6.1 – Create a Workspace



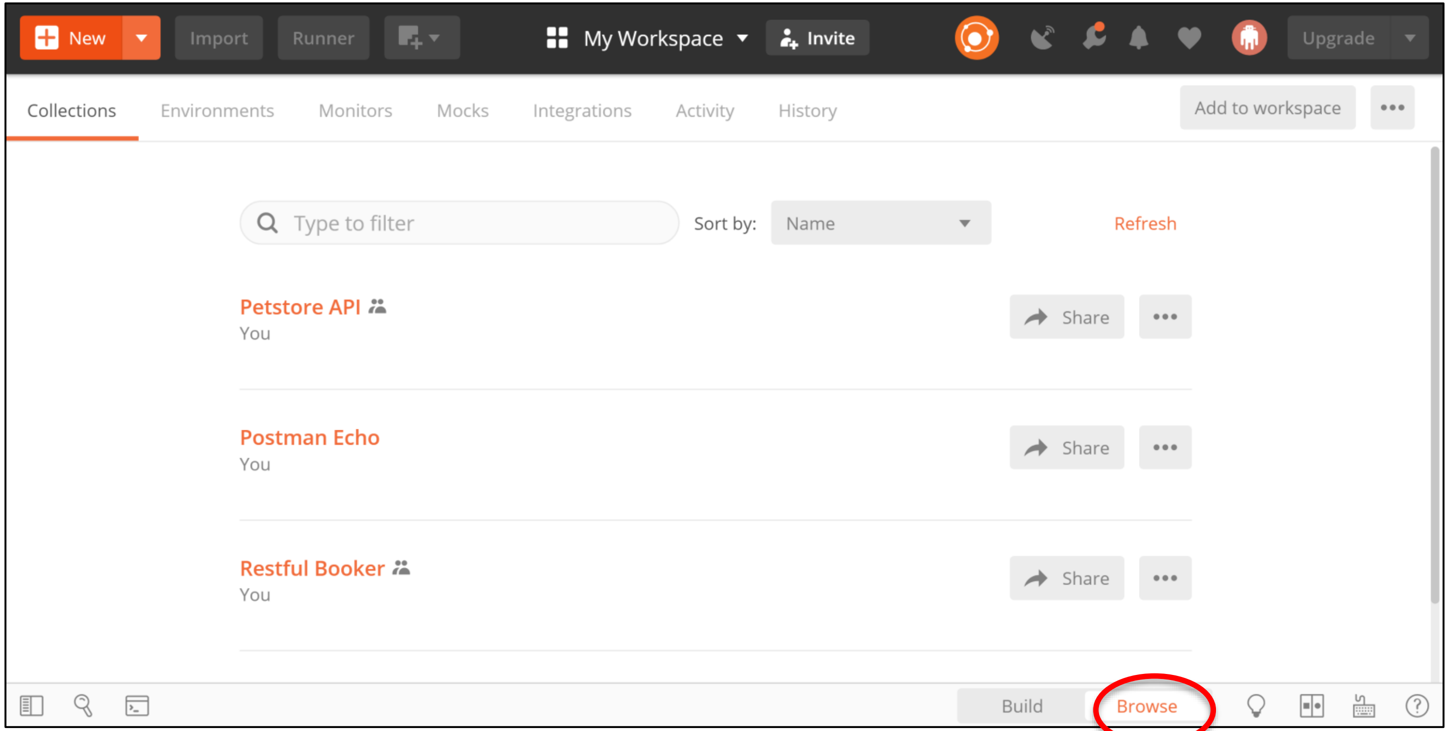
1. Click the **My Workspace** drop-down menu in the Header.
2. Click on **Create new**.
3. Give the Workspace a **name** and **summary**.
4. Select the **Personal** workspace option.
5. Click the **Create this workspace** button.
6. Switch over to the new Workspace. Notice how there are no Collections, Requests, Environments, etc.
7. Create another Workspace but this time select the **Team** option

POSTMAN

Browsing Workspaces

POSTMAN

Browse Workspaces



Petstore API
You

Postman Echo
You

Restful Booker
You

POSTMAN

DEMO

Browse Workspaces

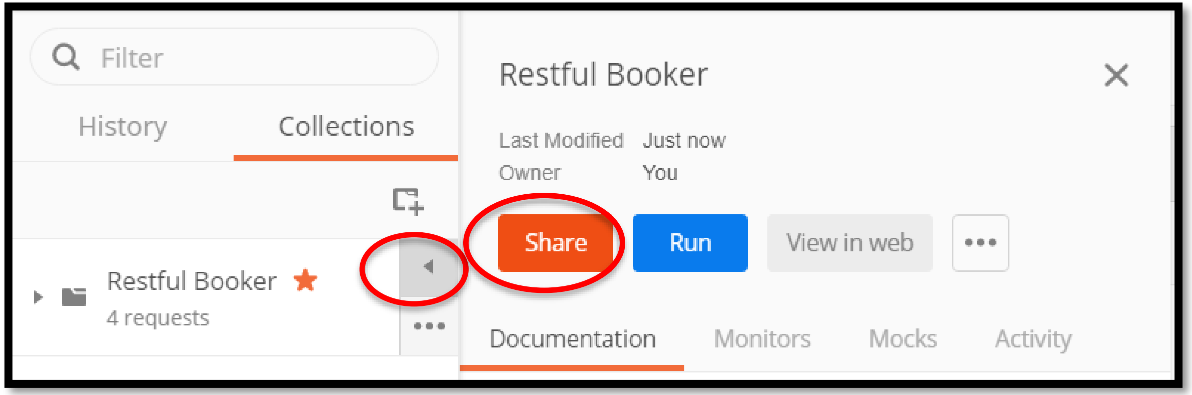
POSTMAN

Using Workspaces to Share & Organize

POSTMAN

Sharing Collections

- Collections can be shared between Workspaces
- Shared Collections are automatically synced
- Allows other team members to view and/or edit the Collection



POSTMAN

EXERCISE

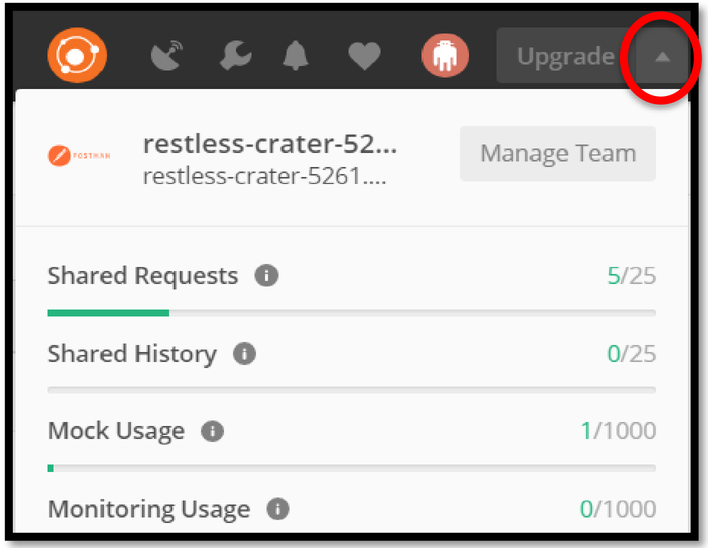
EX6.2 – Share a Collection

1. Return to your default workspace called **My Workspace**.
2. Open the **Restful Booker** Collection and click “Share”.
3. Select the team workspace you created previously and click “Share and Continue”
4. On the Manage Roles screen, leave the permissions as they are and click “Save Roles”
5. Switch to the team workspace and verify that you can see the Restful Booker Collection
6. Create a new **Request** in your **Restful Booker Collection**
7. Switch back to your previous workspace and verify you can see the new request inside the Restful Booker Collection

POSTMAN

Shared Usage Limits

- Accounts on free Postman plan are allocated a limit of shared resources
- Click the drop-down next to the Upgrade button on the header to check usage.
- Upgrade to Postman Pro or Enterprise for unlimited usage.



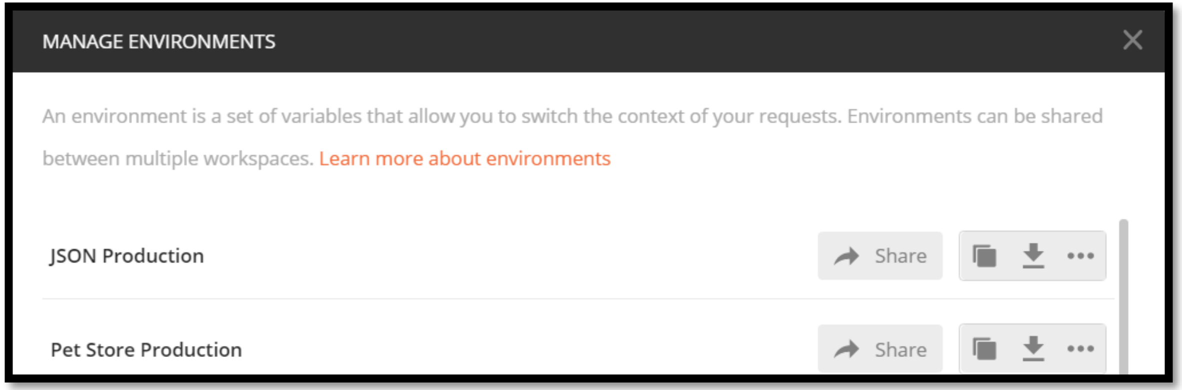
restless-crater-52...
restless-crater-5261....

Shared Requests	5/25
Shared History	0/25
Mock Usage	1/1000
Monitoring Usage	0/1000

POSTMAN

Sharing Environments

- Environments can be shared across Workspaces
- Shared Environments are auto synced
- Do not share Environments containing sensitive data (e.g. passwords, API keys)
- Variables for sensitive data should use placeholder values


MANAGE ENVIRONMENTS

An environment is a set of variables that allow you to switch the context of your requests. Environments can be shared between multiple workspaces. Learn more about environments

JSON Production — Share

Pet Store Production — Share

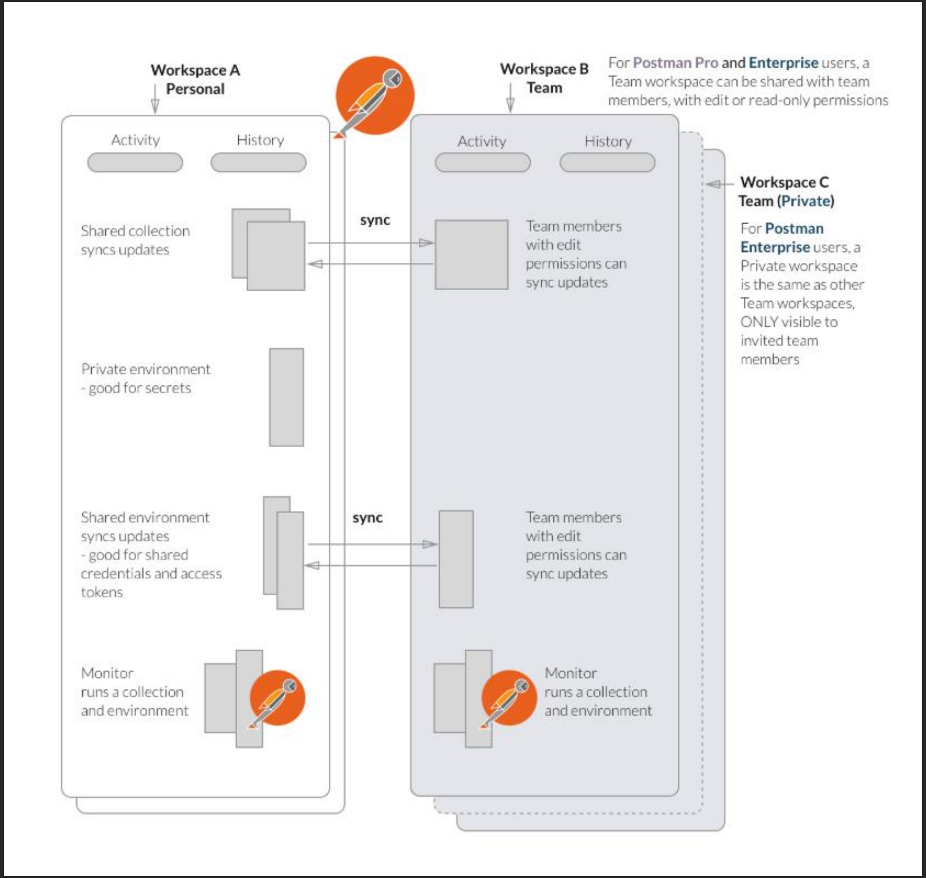
EXERCISE

EX6.3 – Share an Environment

1. Click **Manage Environments** button.
2. Click on the **Restful Booker Admin** Environment.
3. Edit the **INITIAL VALUE** of the “password” variable to “your_password”.
4. Set the **CURRENT_VALUE** to anything you like and click “Update”
5. Return to **Manage Environments** and **share** the new Environment with the team Workspace you created in EX7.1.
6. Switch into the team workspace and check Restful Booker Admin environment. The current value should be the placeholder text “your_password”.
7. Input a password on the current value.

Workspace Sharing

- From Postman version 6.2 onwards, you no longer need Postman Pro or Enterprise to create team workspaces


Workspace A
Personal
Workspace B
Team
For Postman Pro and Enterprise users, a Team workspace can be shared with team members, with edit or read-only permissions
Activity
History
Activity
History
Workspace C
Team (Private)
For Postman Enterprise users, a Private workspace is the same as other Team workspaces, ONLY visible to invited team members
Shared collection syncs updates
sync
Team members with edit permissions can sync updates
Private environment - good for secrets
Shared environment syncs updates - good for shared credentials and access tokens
sync
Team members with edit permissions can sync updates
Monitor runs a collection and environment
Monitor runs a collection and environment

POSTMAN

Using Workspaces to Organize

- Workspaces are a free form organization principle (it is completely up to you, how you want to organize them)
- Some suggestions for organizing:
  - By **function** (documentation, testing, etc.)
  - By **product**
  - By **project**
  - By **partner**

POSTMAN

Module Review

Key Points:

- Team workspaces are important for effective collaboration
- Environments and Collections can be shared and synced across workspaces
- Use session variables to avoid sharing sensitive data in an Environment
- Many ways to organize workspaces (e.g. by function, product, project)

POSTMAN

Additional 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

POSTMAN

You have reached the end of the Postman Fundamentals course!