



POSTMAN

Postman Fundamentals

PRESENTED BY

Johnny Tu

Welcome & Introductions



Instructor: Johnny Tu

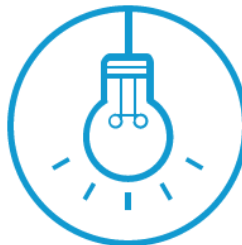
- Training Manager
- Content author
 - Postman
 - Docker
 - NGINX

Housekeeping



Duration

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



Format

- Slides
- Demonstrations
- Exercises

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.

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

Module 1

Postman Concepts & User Interface

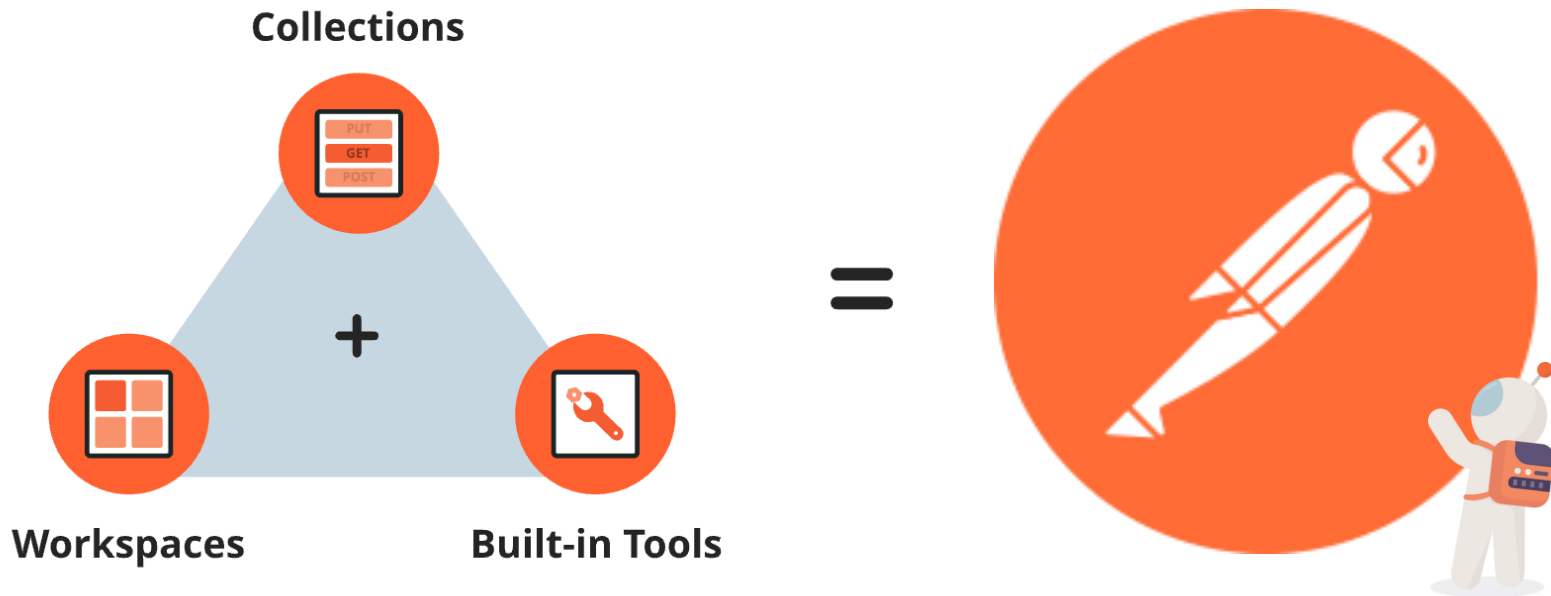
Module Objectives

After completing this module, you will be able to:

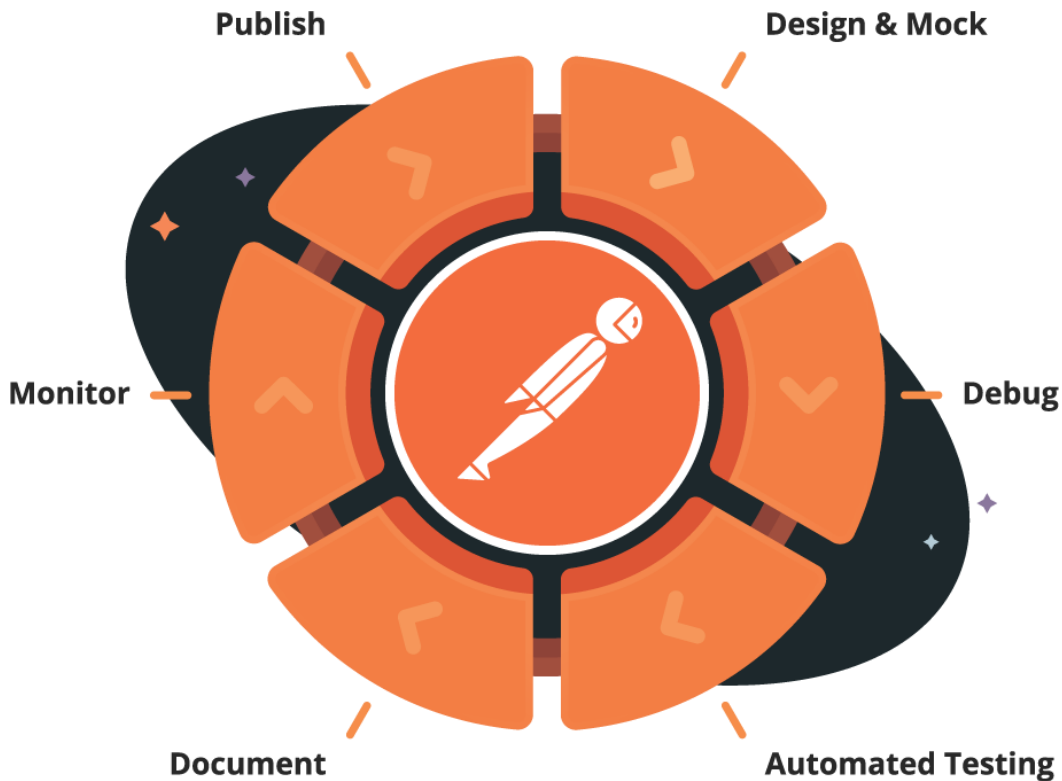
- Describe Postman Concepts including: Requests, Collections, Variables, Environments, and Workspaces
- Describe Postman User Interface (UI)
- Locate Postman UI components including: Request Builder, Sidebar, and Header

Postman Concepts

Collaborative API Development Environment

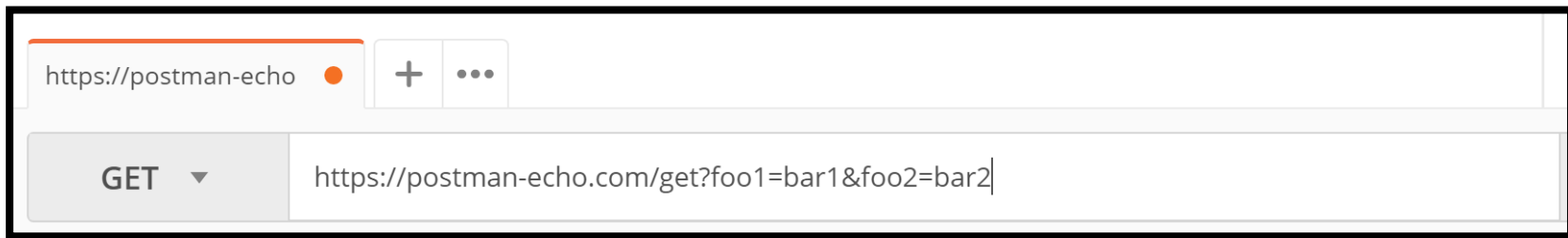


Collaborate Across API Lifecycle Using Postman Tools



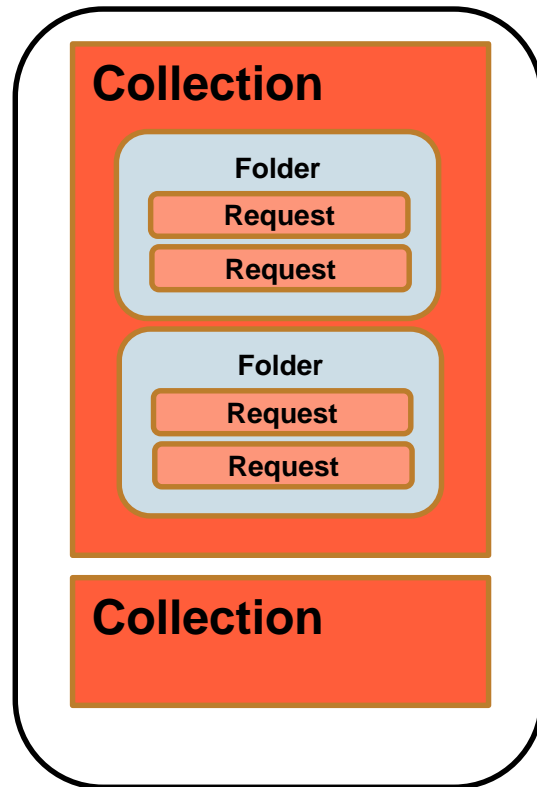
Requests

- 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



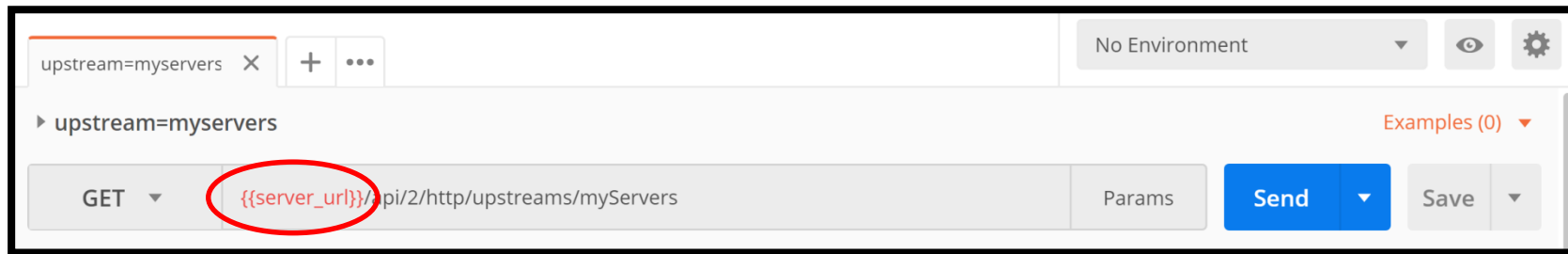
Collections

- 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



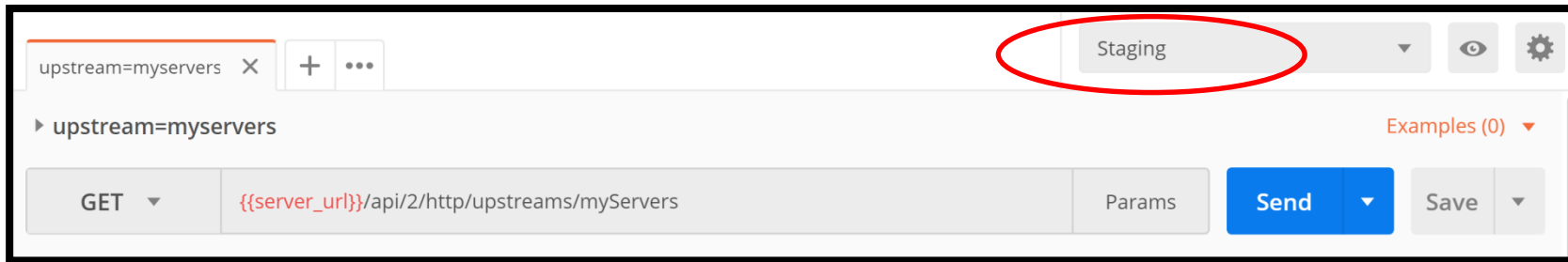
Variables

- **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



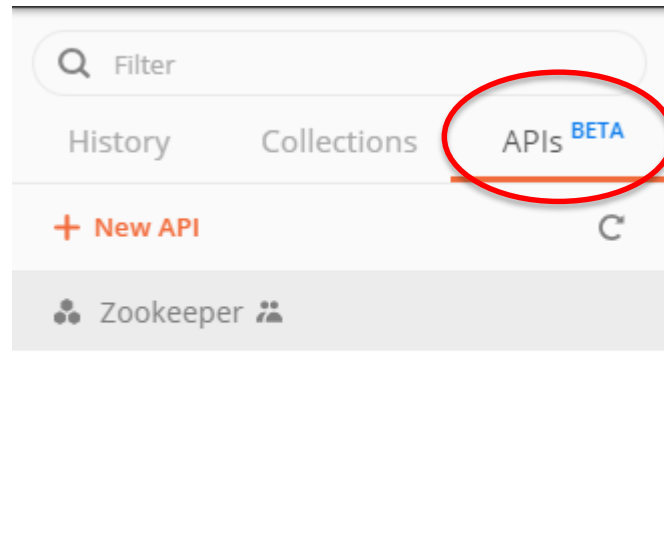
Environments

- 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



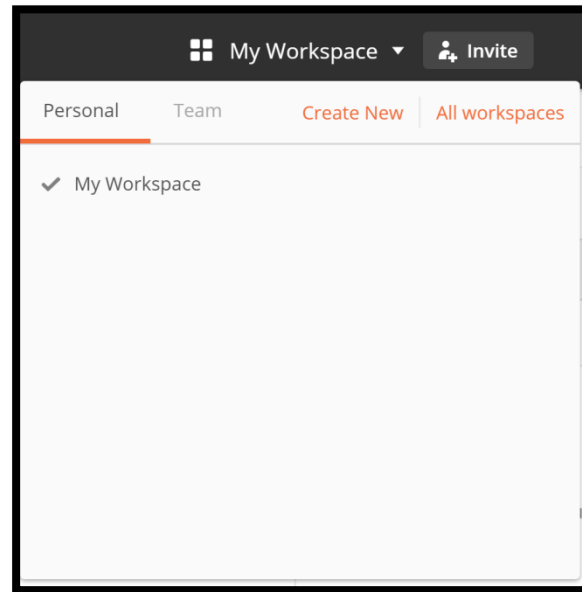
APIs

- 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



Workspaces

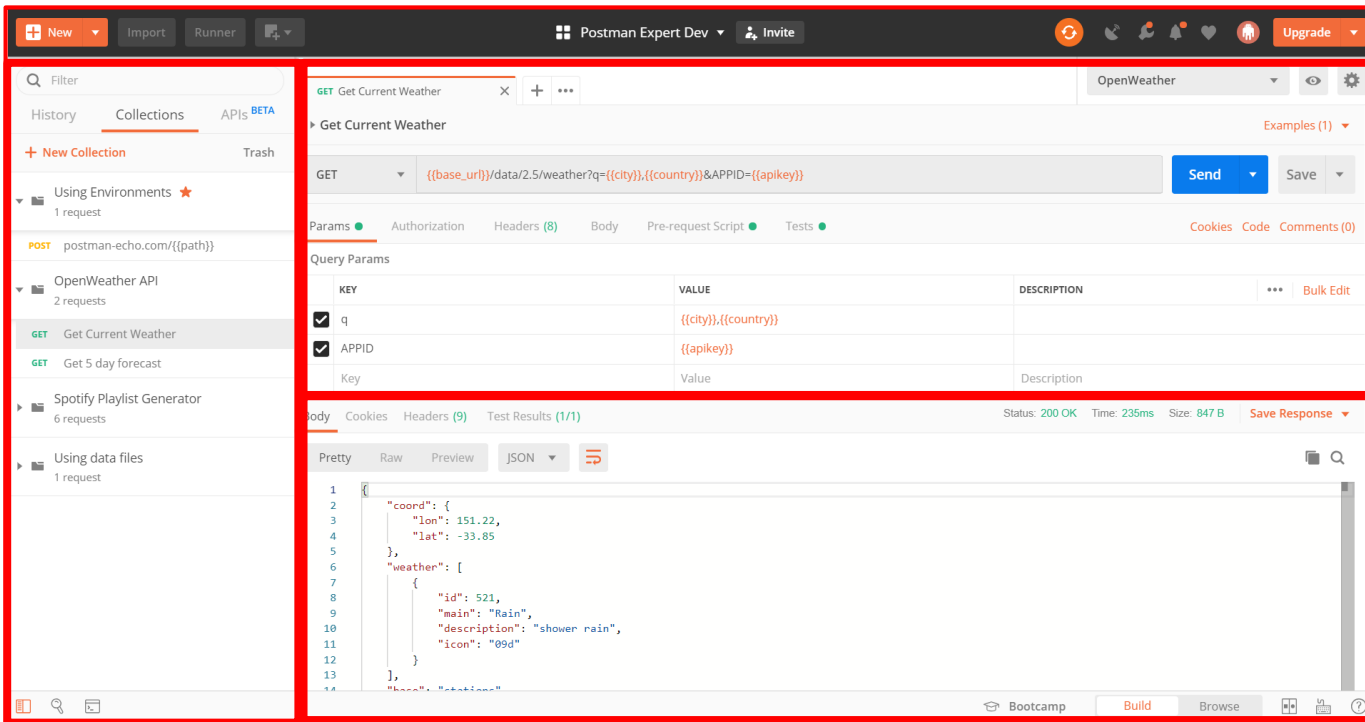
- A **Workspace** is a view of Postman Collections, Environments, etc.
- Provides another layer of organizing your work
- Team vs Personal Workspaces
- Collections can be shared across Workspaces
- Team Workspaces enhances collaboration



Postman User Interface

Postman Application UI

Sidebar



The screenshot displays the Postman application interface, which is divided into three main sections: the Sidebar, the Request Builder, and the Response viewer.

Header: The top bar of the application, featuring a dark background with various icons and buttons. It includes a 'New' button, 'Import', 'Runner', and 'Postman Expert Dev' text. On the right, there are icons for social media, a heart, and an 'Upgrade' button.

Request Builder: The central area where a request is configured. It shows a 'GET' request to 'OpenWeather' with a URL template: `{{base_url}}/data/2.5/weather?q={{city}},{{country}}&appid={{apikey}}`. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', and 'Tests'. The 'Params' tab is active, displaying a table of query parameters:

KEY	VALUE	DESCRIPTION
q	{{city}},{{country}}	
appid	{{apikey}}	

Below the table, there are tabs for 'Cookies', 'Headers (9)', and 'Test Results (1/1)'. The 'Body' tab is also visible.

Response viewer: The bottom section showing the response of the request. It displays a JSON response in a 'Pretty' format, with a status of '200 OK', time of '235ms', and size of '847 B'. The response content is:

```
1 {
2   "coord": {
3     "lon": 151.22,
4     "lat": -33.85
5   },
6   "weather": [
7     {
8       "id": 521,
9       "main": "Rain",
10      "description": "shower rain",
11      "icon": "09d"
12    }
13  ],
14   "base": "stations"
15 }
```

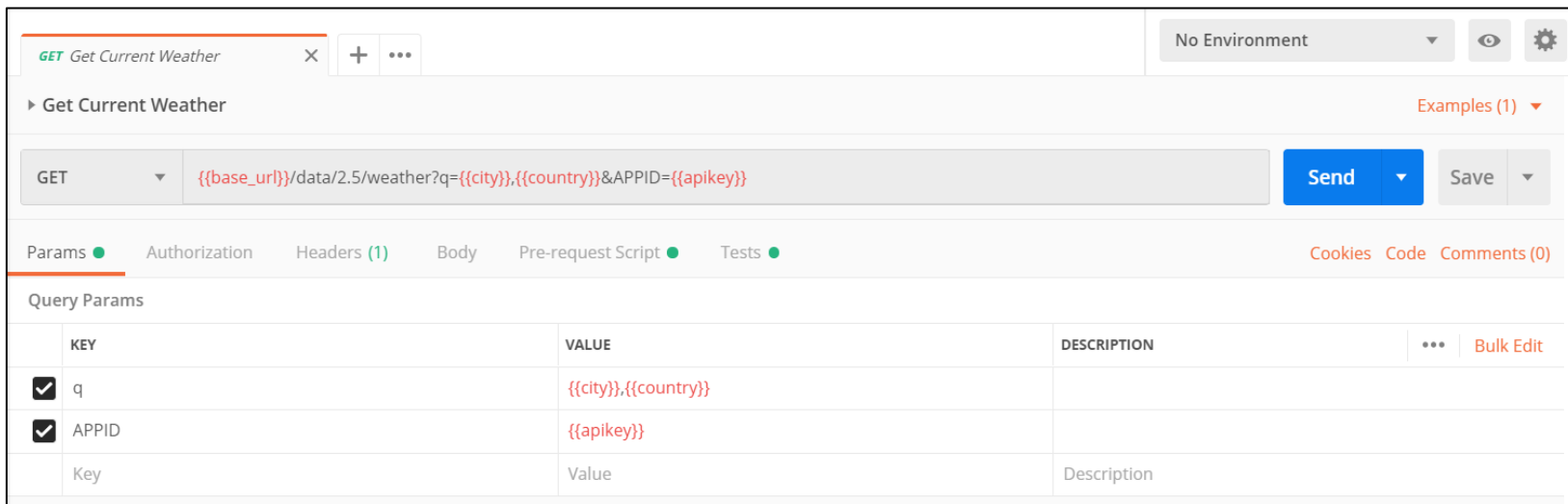
Header

Request
Builder

Response
viewer

Request Builder

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

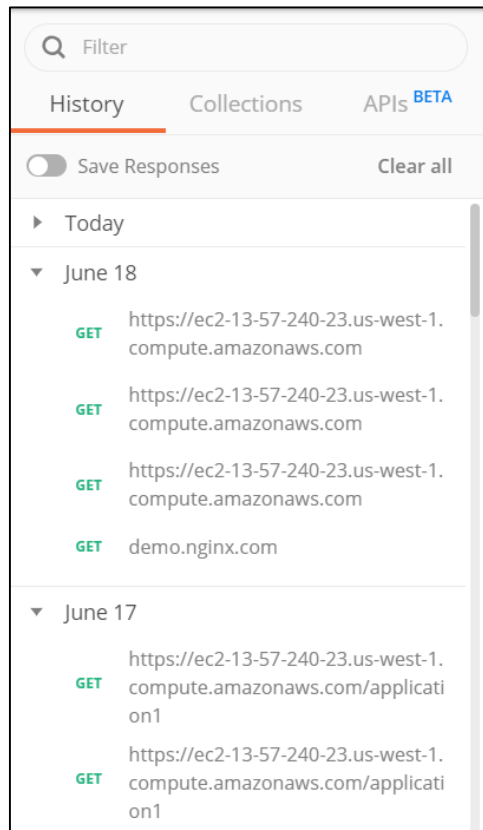


The screenshot shows the Postman Request Builder interface for a GET request named "Get Current Weather". The request URL is `{{base_url}}/data/2.5/weather?q={{city}},{{country}}&APPID={{apikey}}`. The interface includes tabs for Params, Authorization, Headers (1), Body, Pre-request Script, and Tests. The Params tab is active, showing a table of query parameters.

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	q	<code>{{city}},{{country}}</code>			
<input checked="" type="checkbox"/>	APPID	<code>{{apikey}}</code>			
	Key	Value	Description		

Sidebar

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



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

Module 2

Creating & Sending API Requests

Module Objectives

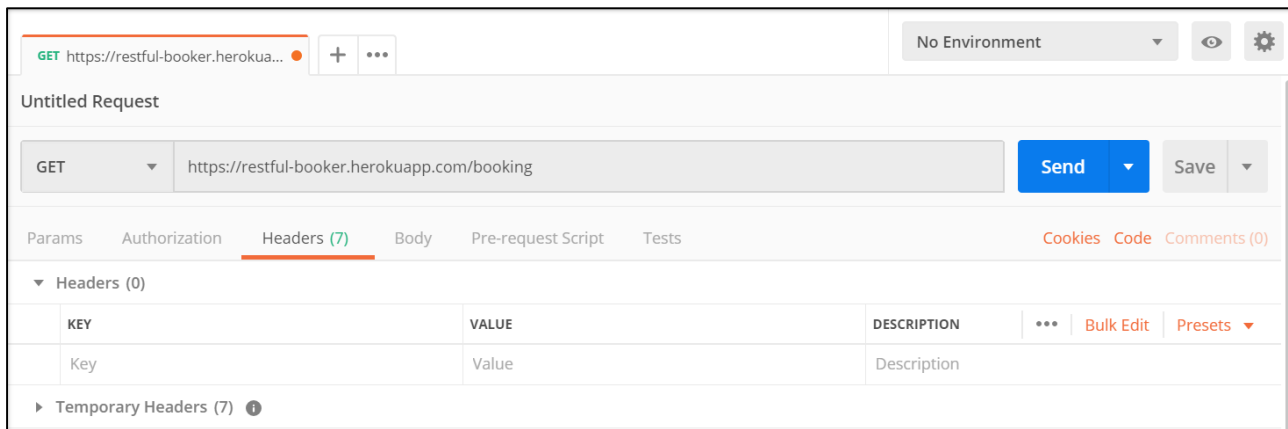
After completing this module, you will be able to:

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

Creating Requests

Creating a Request

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



The screenshot shows the Postman interface for creating a new request. At the top, there's a header bar with a dropdown menu showing 'GET https://restful-booker.herokuapp.com/booking', a '+', and a '...' menu. To the right, it says 'No Environment' with a dropdown arrow, an eye icon, and a settings gear icon.

Below the header, the title 'Untitled Request' is displayed. The main area has a dropdown menu set to 'GET' and a text input field containing 'https://restful-booker.herokuapp.com/booking'. To the right of the input field are two buttons: 'Send' (blue) and 'Save' (grey).

Below the main input area, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', and 'Tests'. The 'Headers (7)' tab is currently selected and highlighted with an orange underline. To the right of these tabs are links for 'Cookies', 'Code', and 'Comments (0)'.

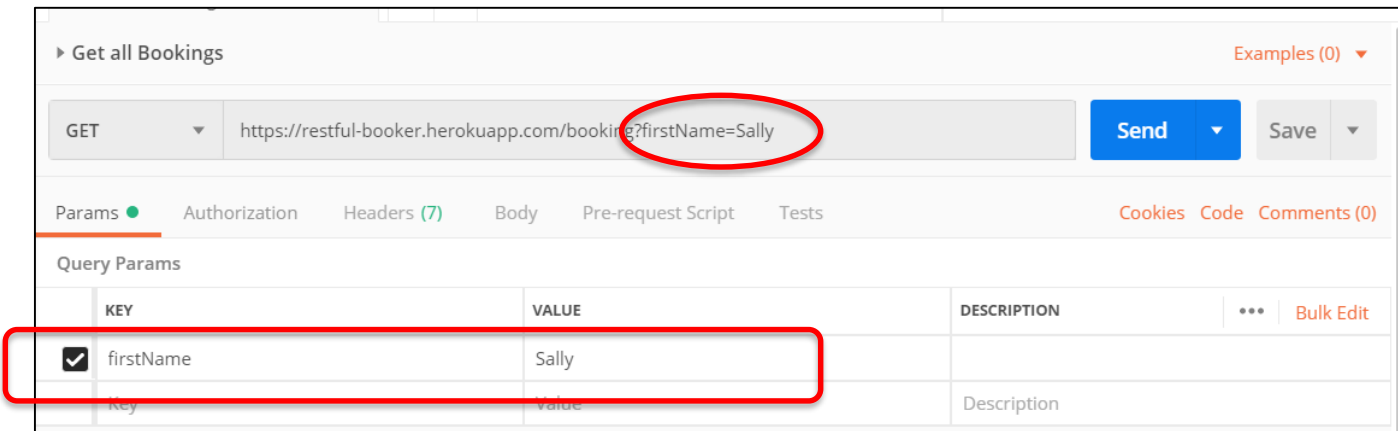
Under the 'Headers (7)' tab, there's a section titled '▼ Headers (0)'. Below this is a table with the following structure:

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets ▼
Key	Value	Description			

At the bottom of the headers section, there's a link '► Temporary Headers (7)' with an information icon.

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 Examples (0) ▾

GET ▾ `https://restful-booker.herokuapp.com/bookings?firstName=Sally` Send ▾ Save ▾

Params ● Authorization Headers (7) Body Pre-request Script Tests Cookies Code Comments (0)

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	firstName	Sally			
	Key	Value	Description		

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

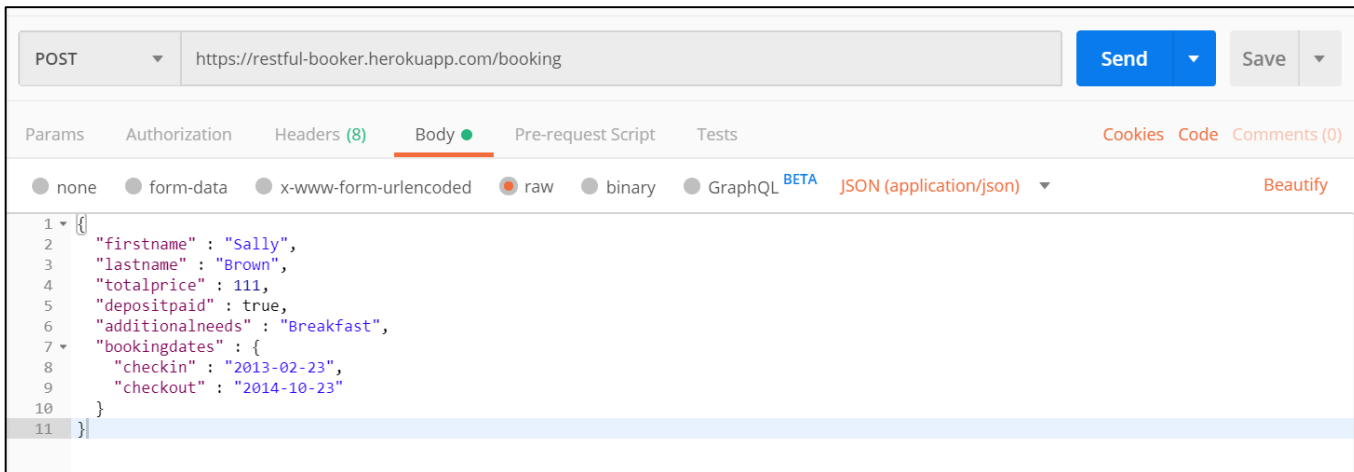


EX2.1 – Create a GET Request

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.

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.



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**.

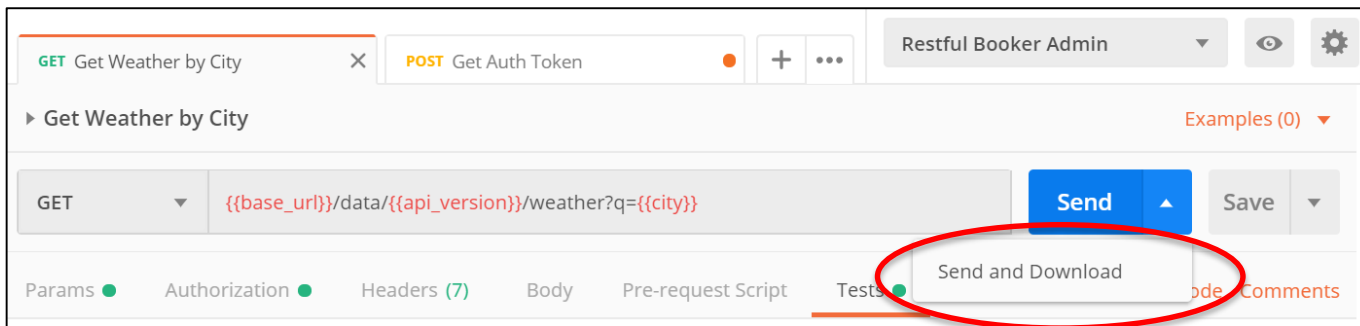
```
{
  "firstname" : "Sally",
  "lastname" : "Brown",
  "totalprice" : 111,
  "depositpaid" : true,
  "additionalneeds" : "Breakfast",
  "bookingdates" : {
    "checkin" : "2013-02-23",
    "checkout" : "2014-10-23"
  }
}
```

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** .

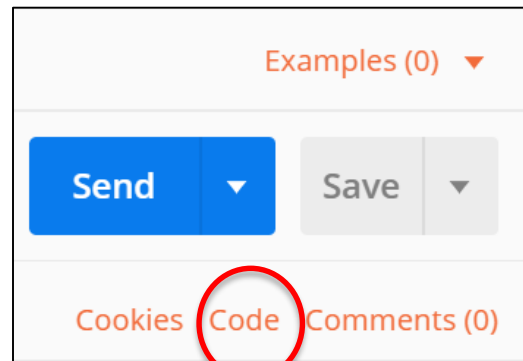
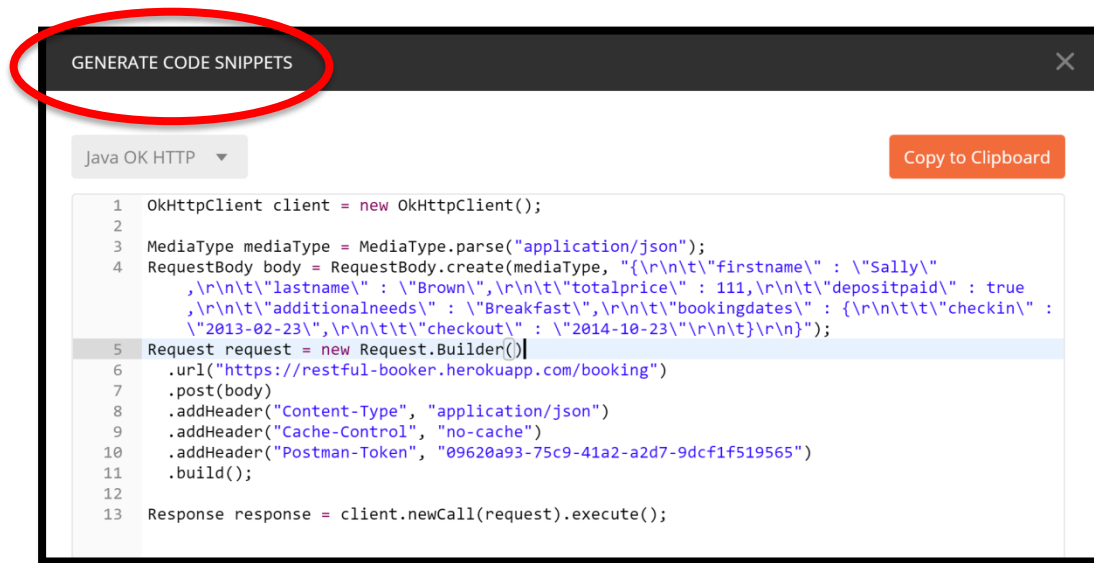
Send & Download

- 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



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



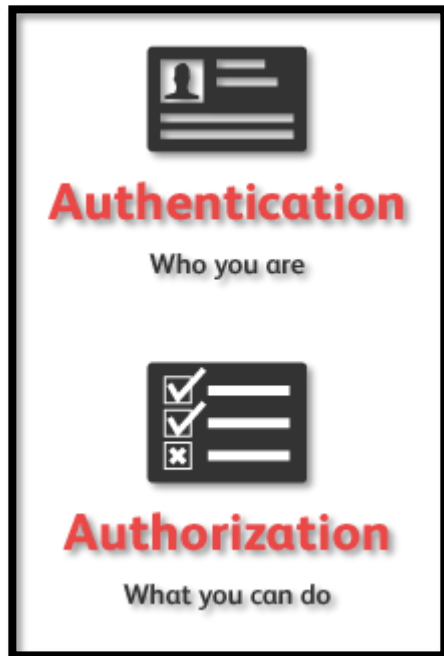
DEMO

Generate Request Code Snippets

Inputting API Authorization Credentials

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



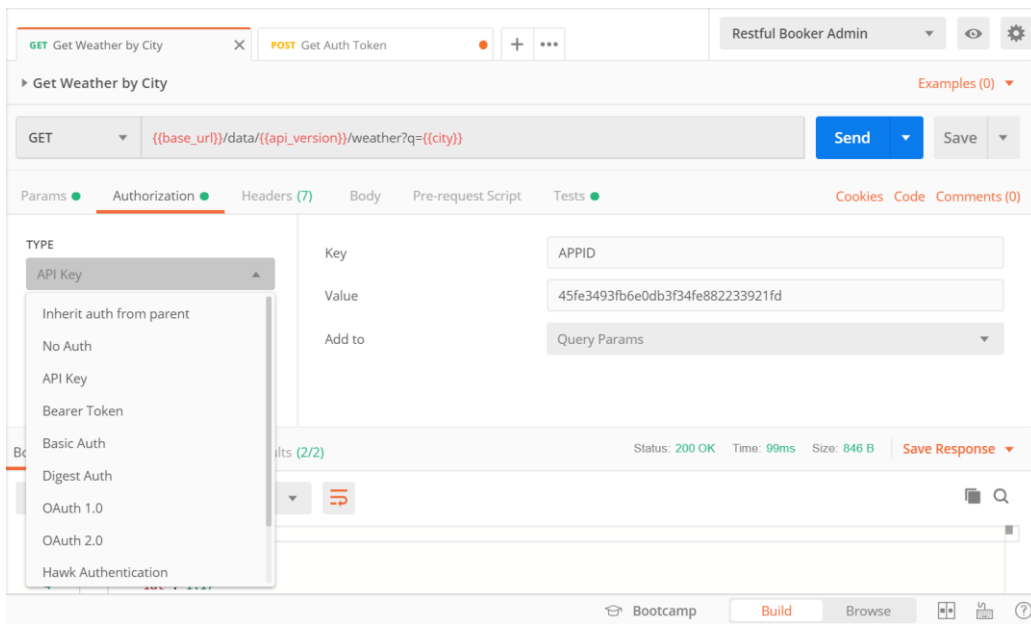
API Authorization Methods (cont'd)

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

API Authorization Methods (cont'd)

- Full details for each Authentication method at:

https://www.getpostman.com/docs/v6/postman/sending_api_requests/authorization



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

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.

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

Module 3:

Organizing Requests into Postman Collections

Module Objectives

After completing this module, you will be able to:

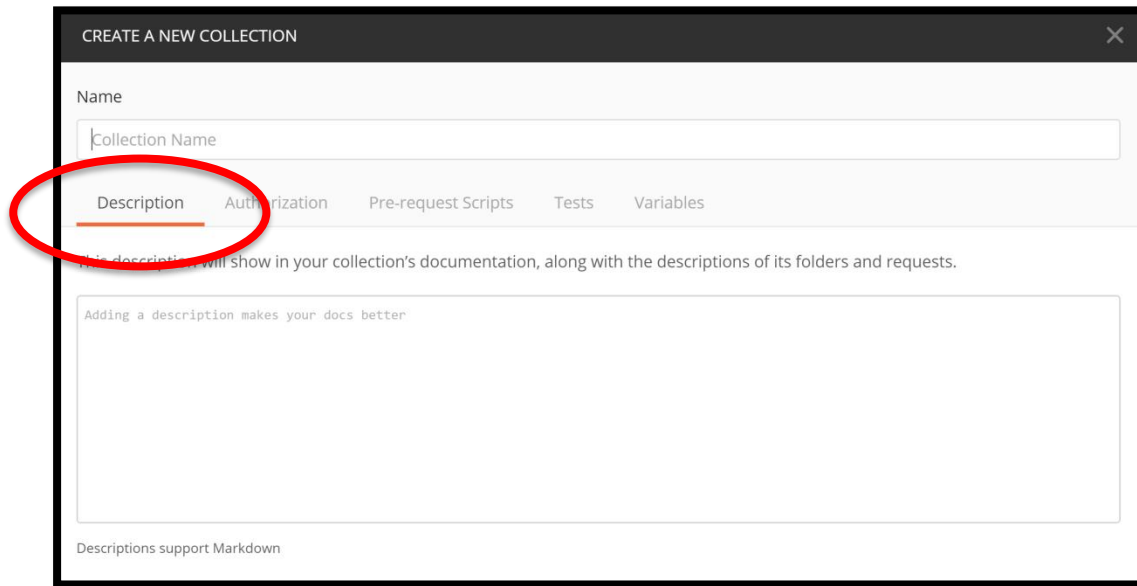
- Describe request History
- Create a Collection
- Save requests into a Collection
- Describe Collection Authorization
- Describe Collection use cases
- Organize request Collections in Folders
- Perform other Collection operations
- Export and Import Collections

Using Collections

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

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

Description Authorization Pre-request Scripts Tests Variables

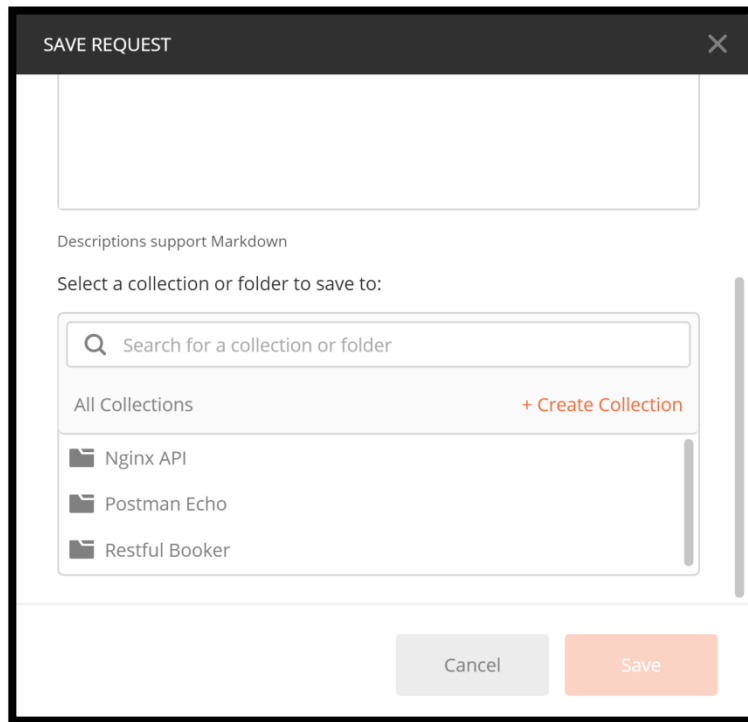
This description will show in your collection's documentation, along with the descriptions of its folders and requests.

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

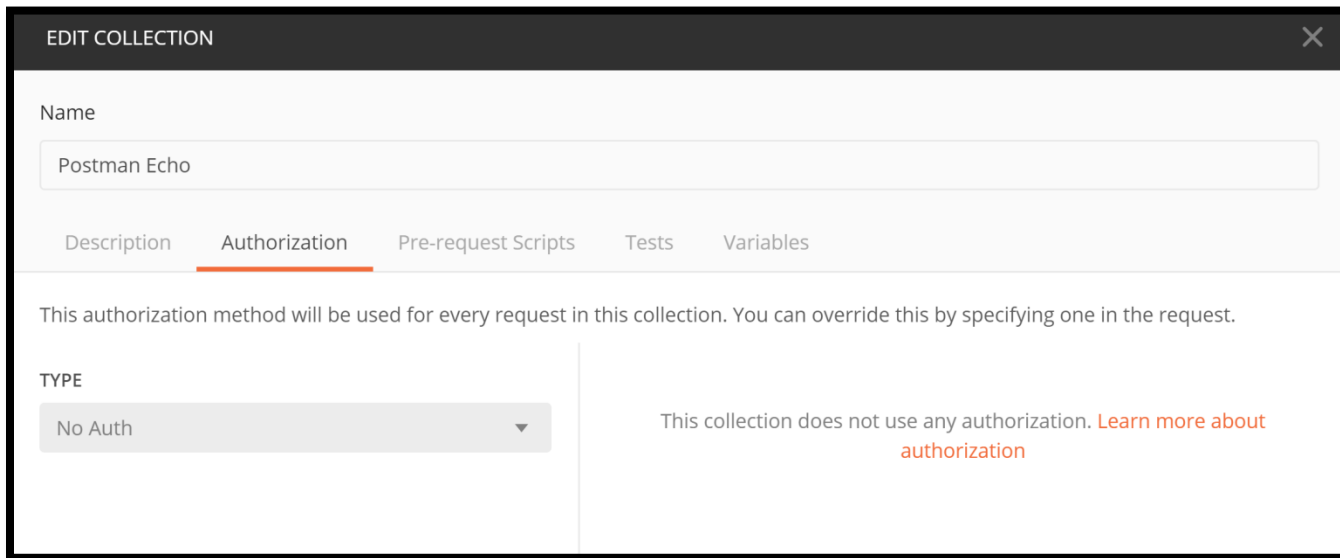


EX3.1 – Create a Collection

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

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



The screenshot shows the 'EDIT COLLECTION' dialog in Postman. The 'Name' field is set to 'Postman Echo'. The 'Authorization' tab is selected, showing a message: 'This authorization method will be used for every request in this collection. You can override this by specifying one in the request.' Below this, the 'TYPE' dropdown is set to 'No Auth'. To the right, a message states: 'This collection does not use any authorization. [Learn more about authorization](#)'.

EDIT COLLECTION

Name

Postman Echo

Description Authorization Pre-request Scripts Tests Variables

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

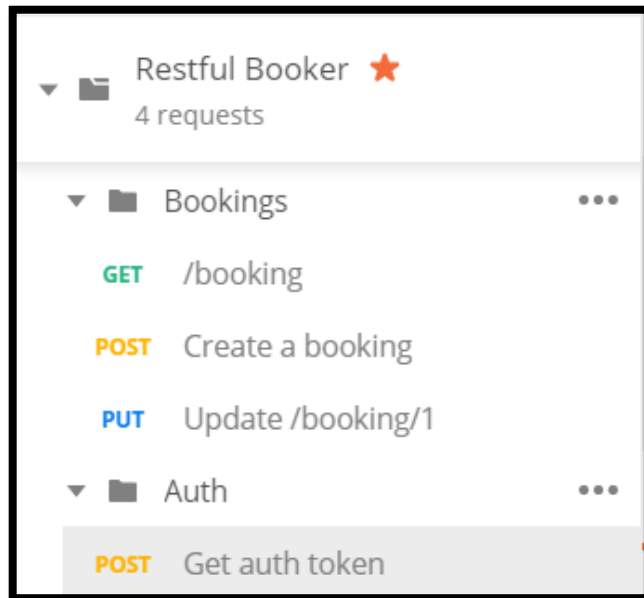
This collection does not use any authorization. [Learn more about authorization](#)

Collection Use Cases

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

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



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**

EX3.2 – (cont'd)

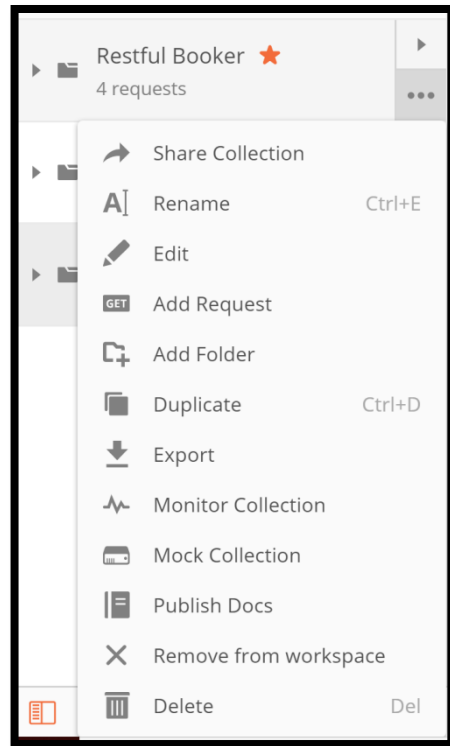
5. Create a new POST request to the end point `https://restful-booker.herokuapp.com/auth`
6. Use the **Raw** option to configure the request body as shown on the right.
7. Select JSON from the drop down list on the request body
8. Send the request and verify that you received a string token as the response.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

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

EXERCISE

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**



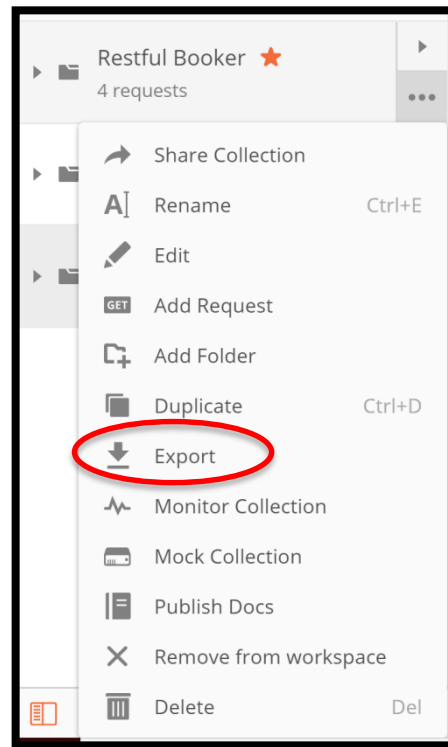
DEMO

Performing Other Collection Operations

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/travlogue-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

Module 4

Configuring Postman Variables & Environments

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

Configuring Postman Variables

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

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}}`



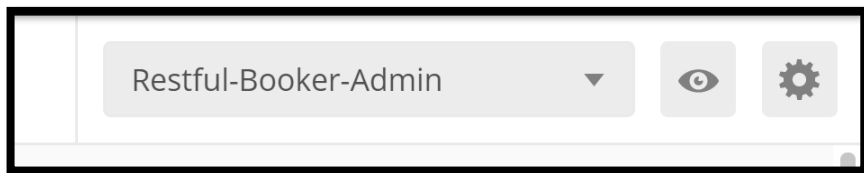
EX4.1 – Define Variables

1. Use the history view and open the OpenWeather API request that was created in exercise 2.3
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

Configuring Postman Environments

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

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 ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	password	your_password	password123			
	Add a new variable					

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 ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	base_url	https://api.openweat...	https://api.openweathermap.org			
<input checked="" type="checkbox"/>	api_version	2.5	2.5			
<input checked="" type="checkbox"/>	city	Sydney	New York			
<input checked="" type="checkbox"/>	api_key	<your key>	45fe3493fb6e0db3f34fe882233921fd			

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

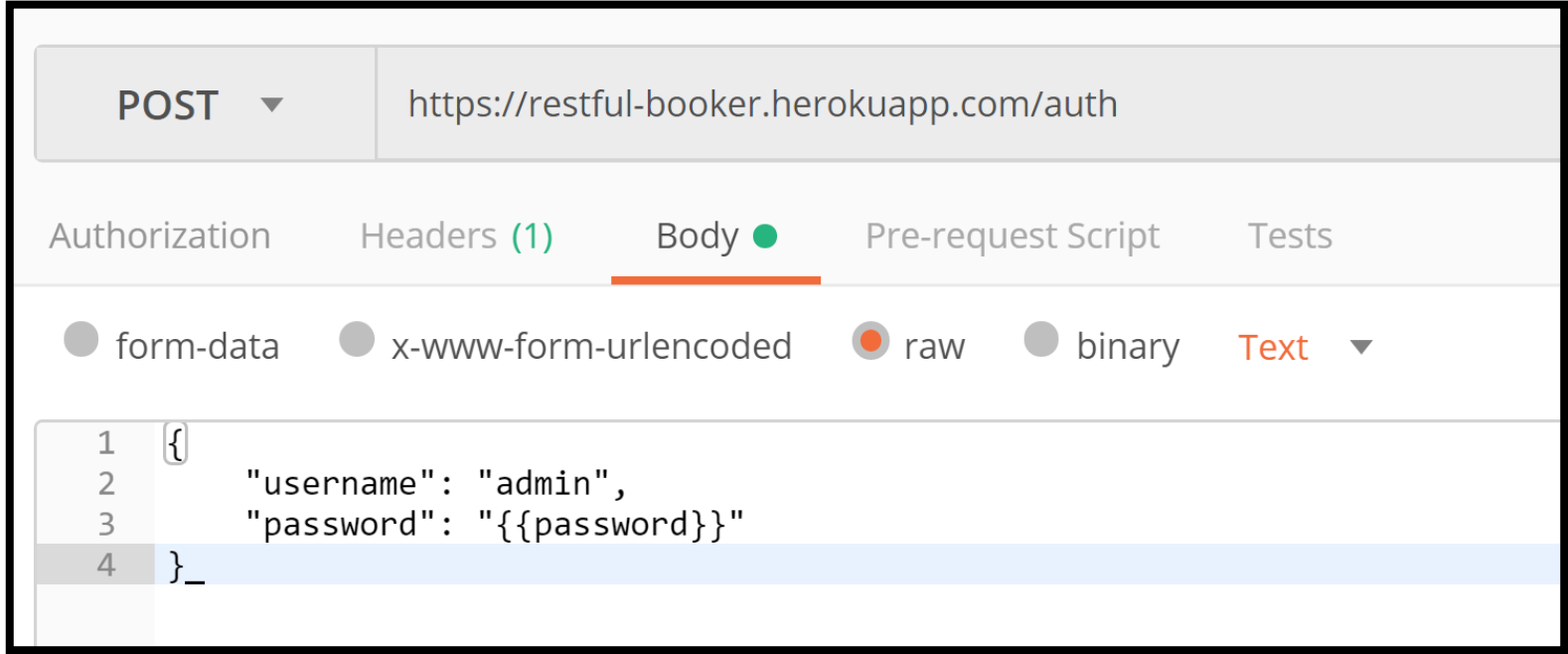
OpenWeather Production

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	base_url	https://api.openweat...	https://api.openweathermap.org			
<input checked="" type="checkbox"/>	api_version	2.5	2.5			
<input checked="" type="checkbox"/>	city	Sydney	Sydney			
<input checked="" type="checkbox"/>	api_key	<your key>	<your key>			

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.

Form Usage Example



The image shows a screenshot of the Postman application interface. At the top, the method is set to **POST** and the URL is `https://restful-booker.herokuapp.com/auth`. Below the URL bar, there are five tabs: **Authorization**, **Headers (1)**, **Body** (which is selected and highlighted with an orange underline), **Pre-request Script**, and **Tests**. Under the **Body** tab, there are five radio button options: **form-data**, **x-www-form-urlencoded**, **raw** (which is selected), **binary**, and **Text** (with a dropdown arrow). The body content is displayed in a text area with line numbers 1 through 4 on the left. The JSON body is:

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

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}}"  
}
```

EXERCISE

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

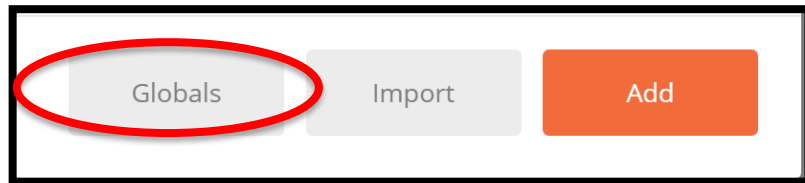
Benefits of Using Variables



- `{{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}}`

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 <small>i</small>	CURRENT VALUE <small>i</small>	...	
<input checked="" type="checkbox"/>	userId	1	1	Persist All	Reset All

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)

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

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

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**

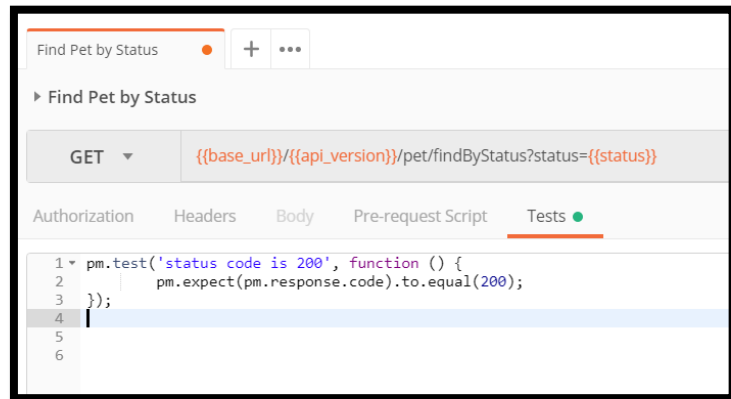
- Verify that the API is up and available for consumers

4. **Security**

- Ensure that API does not allow unauthorised access and usage
- For example, check for permissions of different sets of users

Postman 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



Writing a Basic API Test in JavaScript

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);  
});
```

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

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

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.

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

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");
```

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 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

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.

Module 6

Collaborating Through Postman Workspaces

Module Objectives

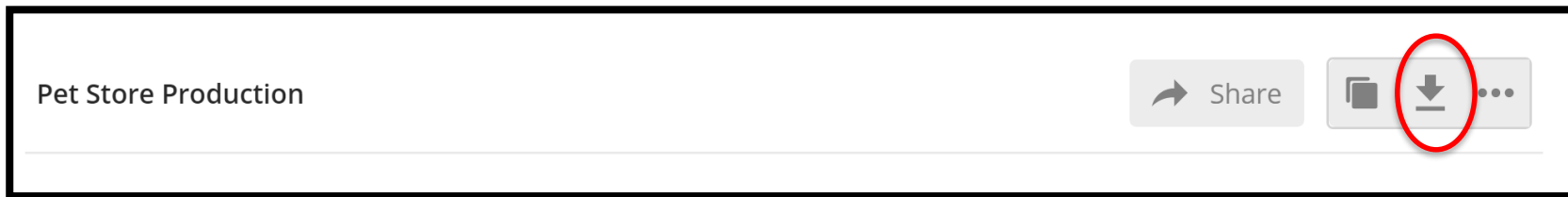
After completing this module, you will be able to:

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

Postman Workspaces

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 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



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

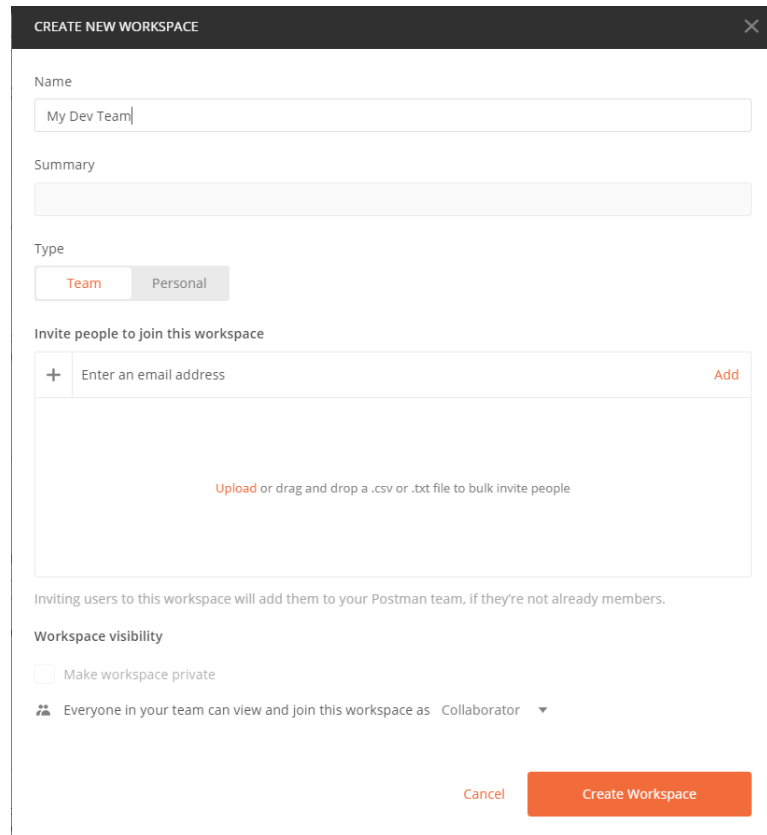
Why use Workspaces?

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

Creating Workspaces

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.



The screenshot shows the 'CREATE NEW WORKSPACE' dialog box. It has a dark header with the title and a close button. The form includes a 'Name' field with the text 'My Dev Team', a 'Summary' text area, and a 'Type' section with 'Team' and 'Personal' buttons. Below this is an 'Invite people to join this workspace' section with a table for adding email addresses and a large area for uploading CSV or TXT files. At the bottom, there is a 'Workspace visibility' section with a checkbox for 'Make workspace private' and a dropdown menu for 'Everyone in your team can view and join this workspace as' set to 'Collaborator'. The dialog ends with 'Cancel' and 'Create Workspace' buttons.

CREATE NEW WORKSPACE

Name

My Dev Team

Summary

Type

Team Personal

Invite people to join this workspace

+	Enter an email address

Add

Upload or drag and drop a .csv or .txt file to bulk invite people

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

Workspace visibility

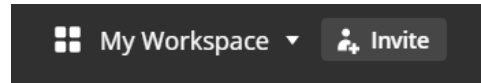
☐ Make workspace private

Everyone in your team can view and join this workspace as Collaborator

Cancel Create Workspace

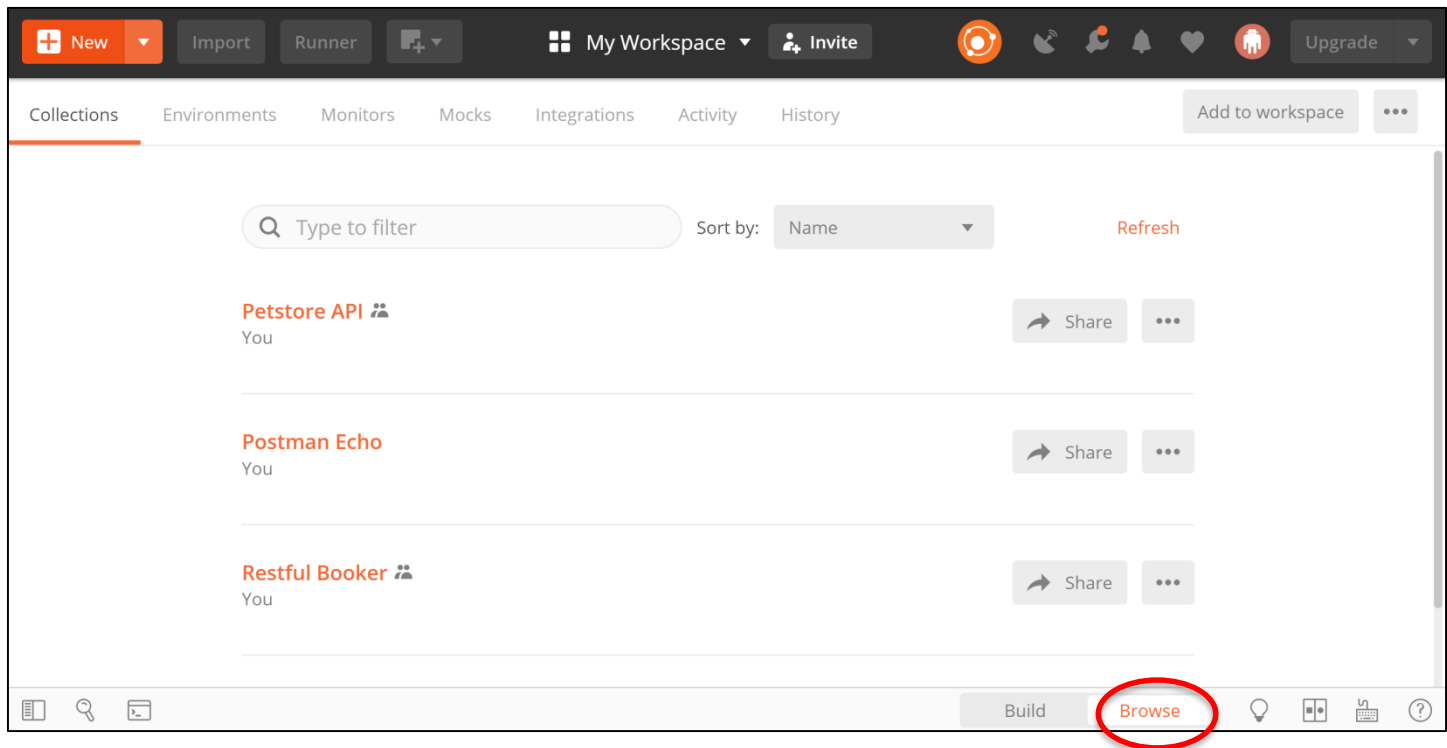
EX6.1 – Create a Workspace

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



Browsing Workspaces

Browse Workspaces



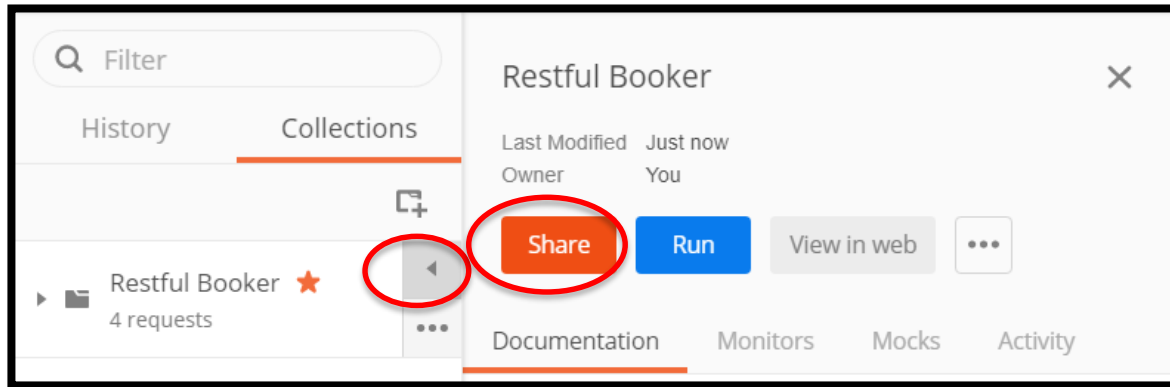
DEMO

Browse Workspaces

Using Workspaces to Share & Organize

Sharing Collections

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

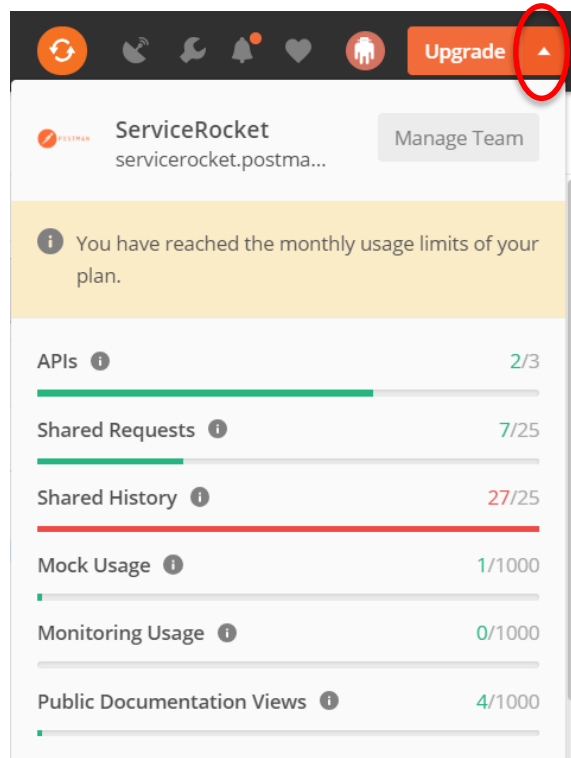


EX6.2 – Share a Collection

1. Return to your default workspace called **My Workspace**.
2. Open the **Restful Booker** Collection and click “Share Collection”.
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

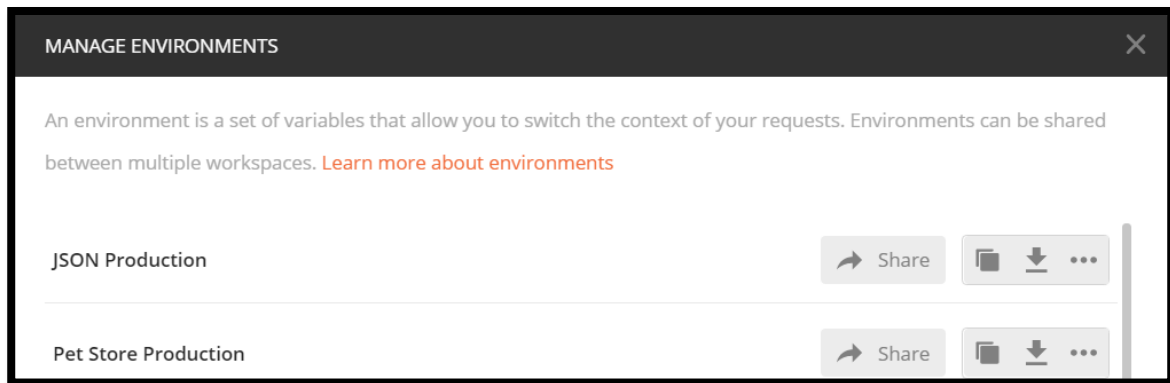
Shared Usage Limits

- 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.



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

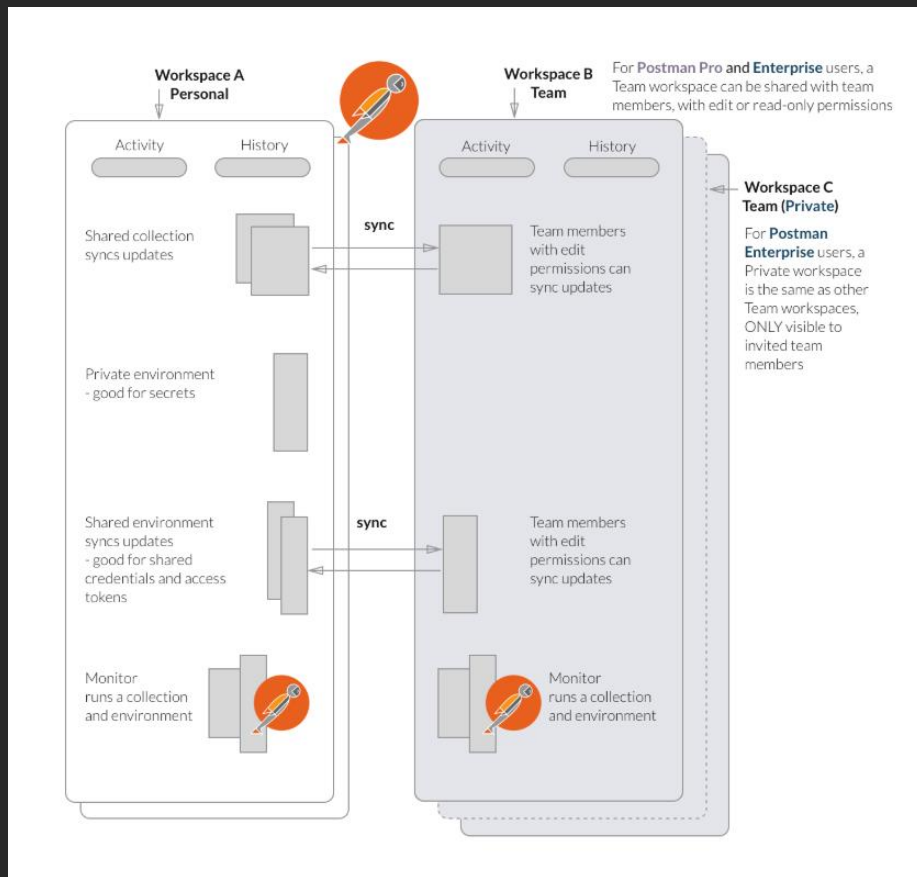


EX6.3 – Share an Environment

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



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**

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)

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>



**You have reached the end of the
Postman Fundamentals course!**

