API Testing

# **Section 2**

Here in Postman we have Created:

1. Workspace: Automation Testing
2. Collections: Library
3. Requests: 1. Add Book

2. Get Book

3. Delete Book

# **Section 3**

Steps to Create Environments:

1. Create Environment: UAT
2. Create Variable: Base\_url
3. Create Initial Value : <https://rahulshettyacademy.com>
4. Create one more Environment: QA
5. Create Variable: Base\_url
6. Create Initial Value: https://216.10.245.166

**Summary: Here we have Created two types of environments – ( UAT and QA )**

Variables: They allow us to Store and Reuse Values in all requests and scripts.

**Creating Variables**

1. As the Base URL Variable name is same in both the environments therefore first we need to select the Environment then Proceed Further.
2. Choose UAT Environment
3. Go to Collection > Library: Add Book
4. Open add book Collection
5. In the url double click on Rahul Shetty
6. Option will come set as variable: Click at it
7. **E Base\_url** will be Mentioned click at it
8. **https://{{Base\_url}}.com/Library/Addbook.php**- ----- This Should Stay at end
9. **Remove ---- https:// and ------ .com ---- as this is already stored in variable value.**
10. **The variable value is wrapped in two curly braces**
11. Copy the Same Text in Get Book Collection
12. Same for Delete Book also.
13. Initial Value : <https://rahulshettyacademy.com>
14. Current Value : <https://rahulshettyacademy.com>
15. Both of Above values should match
16. Script will pick the Current value I have for Run. If Current value is Empty, then it can go and select Initial value.
17. Sent Request and Got Success 😊
18. Similarly Delete Request and Get Book Request will also work 😊

**Environment 2: QA Environment**

1. Now we have to run the Same Scripts in QA Server.
2. Now we will not change the Base url because it is pointing out to the UAT.
3. To change the Environment or To Run those same requests in QA environments Simply move to Environments Tab and Select QA
4. And run all the Scripts again.

**Variable Scopes:**

**Postman Supports the following variable Scopes:**

1. Global (We can Define Global Level Variables)
2. Collection (We can Define Collection Level Variables)
3. Environment (We can Define Environment Level Variables)
4. Data (We can Define Data Level Variables)
5. Local (We can Define Local Level Variables)

**So these are different places we can create variable and the Scope varies for each kind of variable.**

1. **Understand Global Variable:**

No Matter how many Environments or collections you create, there will be some variables which will be common in all locations. **Those variables we need to Create in Global.**

Like Currently we are in Automation Testing Workspace, so even if you create another COLLECTIONS, that GLOBAL variables can be accessed there too.

1. **Understanding Collection Level Variables:**

Like here we have Library Collection, click on it, You will find out a column with name VARIABLES.

It is Mentioned: These variables are specific to this collection and its requests.

1. **Understanding ENVIROMNENT Level Variables:**

We just saw that in the Case of UAT and QA environments where we defined variables.

1. **Understanding DATA Level Variables:**

We use it when we do Data Driven Testing, when we drive data from excel or csv File.

1. **Understanding LOCAL Level Variables:**

These are used with the Request, Add\_Book etc.

* **The Commonly Used variables are GLOBAL, COLLECTION, ENVIRONMENT**
* If same Variable is defined in Environment and in Collection so which one will be called?
* Ans: The value stored in the variable with narrowest scope will be used. For E.g. if a variable name: USERNAME is present in both Environment and Collection level, the Environment level variable will be used
* Hierarchy:

1. Global
2. Collection
3. Environment
4. Data
5. Local

**Section 3 – Lecture 9** Scripting in Postman

Postman contains a powerful runtime Enviroment based on Node.js that allows you to add dynamic behavior to requests and collections. This allows you to write test suites, build requests that can contain dynamic parameters, pass data between requests,

**If you** open up any request you will find 2 tabs one for: **Pre- request Script another for Tests.**

So these are the **Editors**, so we can start writing the JavaScript Code to Automate your Response and Use all the Validations.

Q. **When to Use Pre-Request Editor and when to use Test Editor?**

Ans. Pre- Request scripts are written in JavaScript, and **are run before the request is sent**

Tests scripts are written in JavaScript and **are run before the response is received.**

Diagram

Description automatically generated

1. **Pre-Request Editor**:

Here we will write the code related to Creating JSON Body Dynamically.

Right Now the Code is Hard Coded.

All the Scripting Related things we right over here.

1. **Tests Editor:**

After we will get Response back, Code written in Test Editor will be executed.

Here we will write: Assertions, Validations.

**Example of Assertion**

1. Let’s suppose I want to check the Status Code as: 200 OK. (This is the Basic Assertion).

We have to Write it in test window.

# **The PM Object:**

Before writing the script we should understand the Object: PM Stands for Postman.

All the Postman JavaScript functionality we can use using PM.

Object. Function

Object. Method

It provides access to request and response data and variables.

1. Let’s say I want to write one automation code which check the status Code as 200 OK.

* GO in the Test Window of AddBook
* Write pm.test() ---------- here pm is the object and test is a method / Property
* It takes 2 arguments.
* First argument is the Description: “Validate Status code is 200”.
* Second argument, it takes the actual function: function() where we write the code to make it pass or fail.
* Format: pm.test(“Validate Status Code as 200”, function()
* {
* Actual Code will be Written here
* };

FULL Code:

Library/ Add\_Book

Tests Area

Pm.test(“validate status code 200”, function())

{

pm.responce.to.have.status(200);

pm.responce.to.have.property(“Msg”);

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* Pm is object
* Response is a Property. Here is used coz we want to know the response
* Here If we want to access any request related thing we will use Request.
* SO if you want to put an assertion: like it is equal to 200.
* For that we will write: response.to.have.status
* This Response can hold multiple property like: it can hold property like MSG and and ID.
* **pm.test("validate status code is 200",Function())**
* **{**
* **pm.response.to.have.status(200);**
* **}**
* Send the Request and at bottom check Test Results Tab.

"Msg": "successfully added",

    "ID": "RockOn21227"

This Message comes at botton. Now to Validate the Msg Property we write:

**const ref=pm.response.json(); // at the Top of Code wherein const is a function, ref is the reference variable and pm.responce is object and .json that we need response back in json format**

**pm.expect(ref).have.property("Msg"); // and this statement inside curly braces.**

**Sceanrio**

We mainly evaluate the value inside the Property.

Now : To verify whether: successfully added message CONTENT is present or not we have codeline:

**pm.expect(ref.Msg).to.eql("successfully added");**

**Meaning:**

ref is reference variable which holded entire responce ,

Dot Msg we have used so that we only search in that property.

Dot equal is used for comparison: whether content is matching or not.

1. **Create Automation Test Which checks the Status Code?**

Ans. This should be done for a Request ( Like Add book , Delete Book).

So open up the **Request**, Click on **Test Tab** and Start **Writing Code**

|  |
| --- |
| 1. Pm.test(“validate Status code as OK”,function()) 2. { 3. Pm.responce.to.have.status(“OK”) 4. } |

// **Test** is Property takes 2 arguments.

// **Argument 1** can be **string** message

// **Argument 2** should be a **Function** wherein we will write the actual code

// **response**: so if we want to **access** anything related to **response**, we will use response.

// **Chai Assertion**

// Status: it is Status 200 which we can see on screen

1. **Create Automation Test Which checks the Property Msg is present or not?**

The Response comes in Json Directly so here we need to use Json

|  |
| --- |
| 1. Const jsondata = pm.responce.json(); 2. Pm.test(“Validate property Msg is present or not”,function() 3. { 4. Pm.expect(jsondata).have.property(“msg”); 5. } ); |

// **Const** is a function

// **jsondata** is reference variable where all Bottom response is Stored.

// **pm.responce.json**: pm Is object, response is response and .json means we want response in json format

// **expect**: It is a property

// **Property**: Since msg coming in response is a property that is why we used property.

// **Pm.expect(jsondata)** : Means we are expecting jsondata to have property msg.

1. **Create Automation Test to verify the content of the msg is matching or not.**

|  |
| --- |
| 1. Const jsondata =pm.responce.json(); 2. Pm.test(“validate Text of Property msg”,function() 3. { 4. Pm.expect(jsondata.msg).to.eql(“successfully added”); 5. }); |

// **jsondata.msg**: It will retrieve the data from Response coming under msg heading

// **to.eql**: It is used for Comparision.

// **successfully added** : It is the text from which we are comparing the retrieved text.

**Full Code**

|  |
| --- |
| const jsondata**=**pm.response.json();  pm.**test**("validate Content of msg",**function**()  {  pm.response.to.have("200");  pm.expect(jsondata).have.property("msg");  pm.expect(jsondata.msg).to.eql("Add Book operation failed, looks like the book already exists");  }); |

**Section 3- Lecture 10**

**How to Create Dynamic Value everytime we are creating Add Book.**

1. Open Pre Request Script
2. Select Enviroment > Global
3. Variable Tab : Company Code
4. Initial Value : RS
5. Note : Now to access above, open pre request
6. Code :

|  |
| --- |
| Console.log(Pm.globals.get(“CompanyCode”));  Const code=pm.globals.get(“CompanyCode”);  Const val=pm.variables.replaceIn(‘{{$randomInt}});  Pm.collectionvariables.set(“isbn”,code + val);  Make Changes in Body Section:  **"isbn": "*{{isbn}}*",** |

1. **Get: It is a method, used to get the name of the variable, passing company code inside it wil bring value of company code**
2. **Globals: By using this we will get access to all the global variables.**
3. **Console.log: Like we used SOPln in java for printing here we use Console.log to print in the console.**
4. **Const code : Storing value of Company code in variable – code**
5. **$randomInt = it is a predefined method used to create Random integer number**
6. **Const val = Storing value of integer in val.**
7. **Set: It is a method takes 2 arguments**

**Argument 1: Which variables wants to be setted ------ isbn**

**Argument 2: Which values needs to be set--------------- code + val**

**Now On Collection Level: Our Collection name is Library**

* **Create a Variable with name isbn**
* **Get : It is the Method used to Retreive the value from the editor**
* **Set : It is a method used to set back the variable value.**