

Automation Test Engineer

REST ASSURED - NON Functional QA

EDYODA

Phase-End Project 3

Non-Functional Testing Using Postman, REST Assured, and JMeter

Project Agenda: To automate functionalities using <https://petstore.swagger.io/> REST API services

Scenario:

You are working as a Test Engineer in XYZ Corp. Your company has decided to automate a few functionalities for one of the **Pet Store** companies.

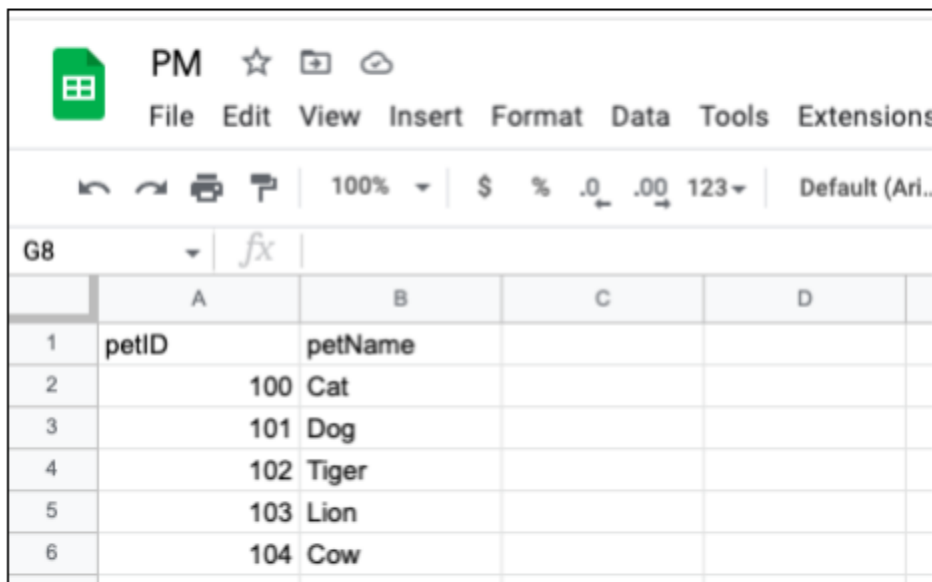
You have been asked to design an end-to-end functionality to automate three REST API services using Postman and Rest Assured.

Tools Required:

- Postman latest version
- Java 1.8+
- Maven latest version
- Rest Assured Maven dependency version 4.5.1
- TestNG Maven dependency version 7.1.0
- Hamcrest Maven dependency
- Newman for Postman latest version
- JMeter

Expected Deliverables:**Postman Assignment 001:****Create petID and PetName :**

Create a CSV file with two columns, petID and petNme, having 20 rows' details as below example



The screenshot shows a Google Sheet titled 'PM' with a menu bar (File, Edit, View, Insert, Format, Data, Tools, Extensions) and a toolbar. The active cell is G8. The table has 6 columns (A-F) and 7 rows (1-7). The first row (1) has headers 'petID' and 'petName' in columns A and B. The subsequent rows (2-6) contain data: (100, Cat), (101, Dog), (102, Tiger), (103, Lion), and (104, Cow).

	A	B	C	D	E	F
1	petID	petName				
2	100	Cat				
3	101	Dog				
4	102	Tiger				
5	103	Lion				
6	104	Cow				

Open Postman

Create a new collection called **Pet_ID_Testing** and arrange three end-to-end services as shownbelow:

In the collection, add a new POST service request with following details:

URL: <https://petstore.swagger.io/v2/pet>

Service Type: POST

JSON Body:

```
{
  "id": 344,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"}
```

In the JSON body, **id** and **name**, which are highlighted in **YELLOW** color, should be parameterized in Postman and runtime data should be driven through the CSV file.

In the Postman Tests section, validate for this POST call. Response status code should be 200.

In the Postman Tests section validate for this POST call. Response body should contain text as '**available**'.

Validate petID:

In the collection, add a new GET service request with the following details:

URL: <https://petstore.swagger.io/v2/pet/2003>

In the service URL pet **id**, which is highlighted in **YELLOW** color, should be parameterized in Postman and runtime data should be driven through the CSV file.

In the Postman Tests section validate for this GET call. Response status code should be 200.

Delete petID:

In the collection, add a new DELETE service request with the following details:

URL: <https://petstore.swagger.io/v2/pet/2003>

In the service URL pet **id**, which is highlighted in **YELLOW** color should be parameterized in Postman and runtime data should be driven through the CSV file.

In the Postman Tests section validate for this GET call. Response status code should be 200.

Steps to Run:

Run collection

Select the CSV file to run these End-to-End scenarios 20 times

Check the checkbox for Save Response

Open Postman Console and Run the collection

Validate that all parameterized data [PetID/PetName] should be filled with CSV runtime data

All status codes should pass

Export the collection as a JSON file

Run this JSON collection using the Postman Newman command from cmd/terminal.

Postman Assignment_002:

Hit a **PUT** call having service URL = <https://petstore.swagger.io/v2/pet>

Create a global variable of this URL where **testURL** is the variable name and its value is <https://petstore.swagger.io>. Use this variable while hitting the URL in Postman.

PUT Call request JSON body:

```
{
  "id": 9223372016900013000, "category": {
    "id": 20021,
    "name": "string" },
    "name": "doggie", "photoUrls": [
      "string"
    ], "tags": [
      {
        "id": 0,
        "name": "string"
      }
    ],
    "status": "available_QA"
  }
```

Create 3 test Environments as DEV, QA, PROD. The PUT call JSON body **status** field should be parameterized and its value should change as per environment:

When Environment is DEV then "status": "available_DEV"

When Environment is QA then "status": "available_QA"

When Environment is PROD then "status": "available_PROD"

Validate id = 20021 in response

Validate response = 200

Validate **status** value is changing as per environment in Json Response.

Postman Assignment_003:

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/user/Username001>

Use Username001 as global parameter

Validate response as 200 in postman

Validate username = Username001 in Json response

Validate email = Positive@Attitude.com in Json response

Validate userStatus = 1 in Json response

Postman Assignment_004:

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/pet/findByStatus>

Use Postman params as **status**

When **status=available** and if after hitting the URL , response status = 200 then validate for all pet details it's response status = **available**.

When **status=pending** and if after hitting the URL , response status = 200 then validate for all pet details it's response status = **pending**.

When **status=sold** and if after hitting the URL , response status = 200 then validate for all pet details it's response status = **sold**.

Postman Assignment_005:

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/user/login>

Use Postman basic Authentication with username=Username001 and

Password = @tt!tude

Validate response as 200 in postman

Validate code = 200 in response

REST Assured Assignment:

Open Eclipse/IntelliJ

Create a Maven project

Add Maven dependency for TestNG and REST Assured

Create a TestNG test with below REST API execution details -

POST CALL

URL: <https://petstore.swagger.io/v2/pet>

Service Type: POST

JSON Body:

```
{
  "id": 344,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "Doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

PetID is parameterized

Once POST call is successful - validate response code

Validate that PetID from response code should be same as request

PetID

GET CALL

URL: <https://petstore.swagger.io/v2/pet/2003>

Service Type: GET

Service URL pet **id**, which is highlighted in **YELLOW** color should be parameterized and be the same as POST call PetID

Validate response code as 200

Validate that response JSON body contains keys **status** and **id**

Validate **status** value is **available**

DELETE CALL

URL: <https://petstore.swagger.io/v2/pet/2003>

Service Type: DELETE

Service URL pet **id**, which is highlighted in **YELLOW** color, should be parameterized and the same as POST call PetID

Validate response code is 200

Validate that response JSON body contains keys **code** and **message**

Validate **message** value is **PetID**

POST→GET→DELETE services are interrelated. If the POST call fails, then the GET and DELETE call will also fail.

Note: This TestNG test should run from testNG.xml file

You can use either TestNG assertion or Hamcrest assertion

Rest Assured Assignment 002:

Hit a **PUT** call having service URL = <https://petstore.swagger.io/v2/pet>

PUT Call request JSON body:

```
{
  "id": 9223372016900013000, "category": {
    "id": 20021,

    "name": "string" },
    "name": "doggie", "photoUrls": [

    "string"
  ], "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available_QA"
}
```

Create 3 test Environments as DEV , QA , PROD. Where PUT call JSON body **status** field should be parameterized and its value should change as per environment as below.

- When Environment is DEV then "status": "available_DEV"
- When Environment is QA then "status": "available_QA"
- When Environment is PROD then "status": "available_PROD"

All assertion/validation should be done using hamcrest

Validate response as 200 in response

Validate id = 20021 in response

Steps to Run this :

Create a testNG test with parameters as environment. As an example putCallTesting(String Env).

As per environment name the value should be read from HasMap and the same value should populate in Json response.

As an Example **putCallTesting("Dev")** method should pick its corresponding value from Hashmap where **Dev** is the **KEY** and its value should be used in the JSON request body **status** field.

Rest Assured Assignment 003:

Create a testNG test and implement below scenario

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/user/Username001>

All assertion/validation should be done using hamcrest

Validate response as 200 in response

Validate username = Username001 in Json response

Validate email = Positive@Attitude.com in Json response

Validate userStatus = 1 in Json response

Rest Assured Assignment 004:

Create a testNG test and implement below scenario

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/user/login>

Rest Assured Assignment 005:

Create a testNG test and implement below scenario

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/pet/findByStatus>

Use Rest Assured params as **status**

All assertion/validation should be done using hamcrest

When **status=available** and if after hitting the URL , response status = 200 then validate for all pet details it's response status = **available**.

When **status=pending** and if after hitting the URL , response status = 200 then validate for all pet details it's response status = **pending**.

When **status=sold** and if after hitting the URL , response status = 200 then validate for all pet details it's response status = **sold**.

Rest Assured Assignment 006:

Create a testNG test and implement below scenario

Hit a **GET** call with URL : <https://petstore.swagger.io/v2/user/logout>

All assertion/validation should be done using hamcrest

Validate response as 200 in response

Validate code = 200 in response

Validate message = OK in response