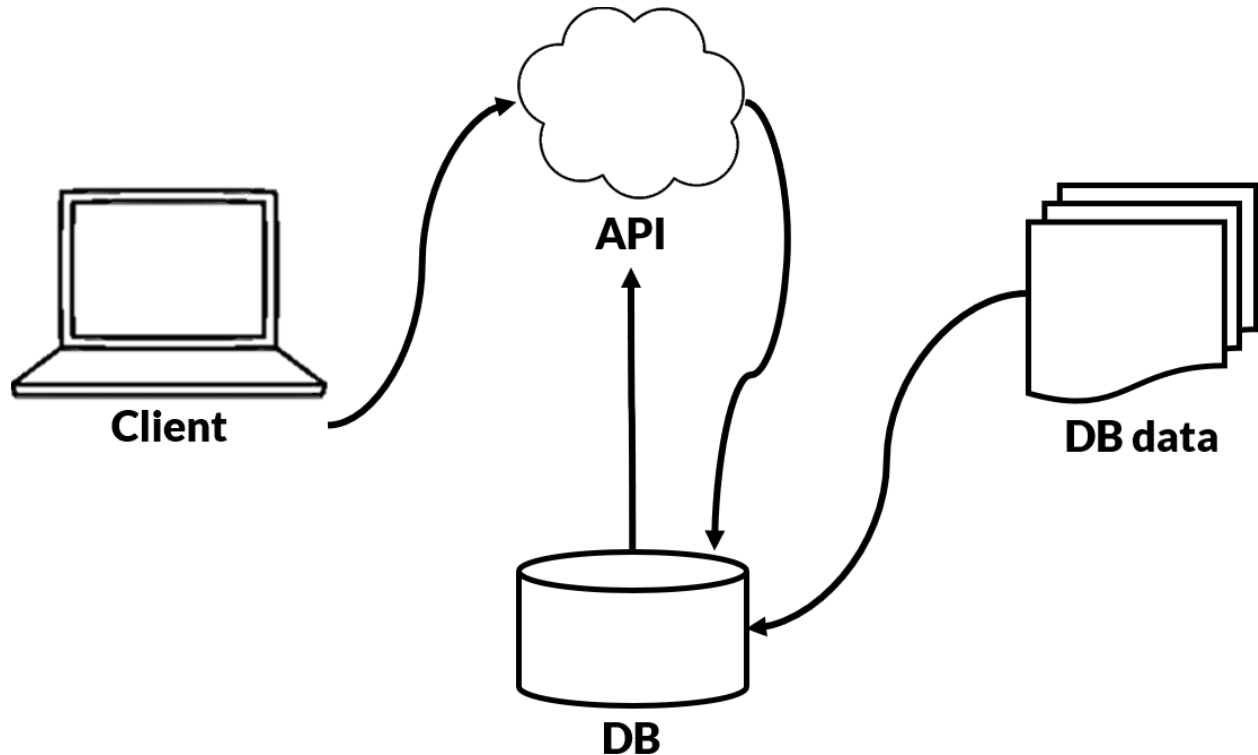


What is an API?

An API, or Application Programming Interface, is a set of definitions and protocols that allow one application to communicate with another application.

APIs can be exposed through local files (such as a JAR file in a Java program, .H file in C/C++ programs, etc.) to allow two local applications to communicate with each other. This doesn't require a network as the two applications are communicating within a single device.



What is a Web Service?

A Web service is a way for two machines to communicate with each other **over a network**.

All Web services are APIs but all APIs are not Web services!

A web server running on a computer listens for requests from other computers. When a request from another computer is received, over a network, the Web service returns the requested resources. This resource could be JSON, XML, an HTML file, Images, Audio Files, etc.

What is REST API?

REST stands for *REpresentational State Transfer*.

It means when a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource via HTTP protocol.

E.g. when a developer calls Instagram API to fetch a specific user (the resource), the API will return the state of that user, including their name, the number of posts that user posted on Instagram so far, how many followers they have, and more.

The representation of the state can be in a JSON format, and probably for most APIs this is indeed the case. It can also be in XML or HTML format.

E.g. Google Map embedded in 3rd party websites – play search API

Key terms:

1. **Client** — the client is the person or software who uses the API. It can be a developer, for example you, as a developer, can use Twitter API to read and write data from Twitter, create a new tweet and do more actions in a program that you write. Your program will call Twitter's API. The client can also be a web browser. When you go to the Twitter website, your browser is the client who calls Twitter API and uses the returned data to render information on the screen.
2. **Resource** — a resource can be any object the API can provide information about. In Instagram's API, for example, a resource can be a user, a photo, a hashtag. Each resource has a unique identifier. The identifier can be a name or a number.

The six parameters are:

1. API endpoint*
2. Method*
3. Rest Endpoint*
4. Headers*
5. Params
6. Data (Payload)

**required*

What is Endpoint?

What the server does when you, the client, call one of its APIs depends on 2 things that you need to provide to the server:

1. An identifier for the resource you are interested in. This is the URL for the resource, also known as the **endpoint**. In fact, *URL stands for Uniform Resource Locator*.
2. The operation you want the server to perform on that resource, in the form of an HTTP method, or verb. The common HTTP methods are **GET, POST, PUT, and DELETE**.

E.g.

Endpoint = Unique Phone number;

Where **baseURL** = Country code, **resource** = STD Code, **parameter** = Phone Number

Endpoint Syntax: *baseURL/resource?parameters*

Reference:

https://rapidapi.com/blog/api-vs-web-service/?utm_source=google&utm_medium=cpc&utm_campaign=DSA&utm_content=1t1&gclid=Cj0KCQiArdLvBRCrARIsAGhB_syT2KIEbUWu89BWC3SnsDFpopfsnHz_gTZOFMi2urbtq2jluoInsAMaAlbsEALw_wcB

<https://blog.yellowant.com/rest-api-calls-made-easy-7e620e4d3e82>

HTTP Methods

CRUD

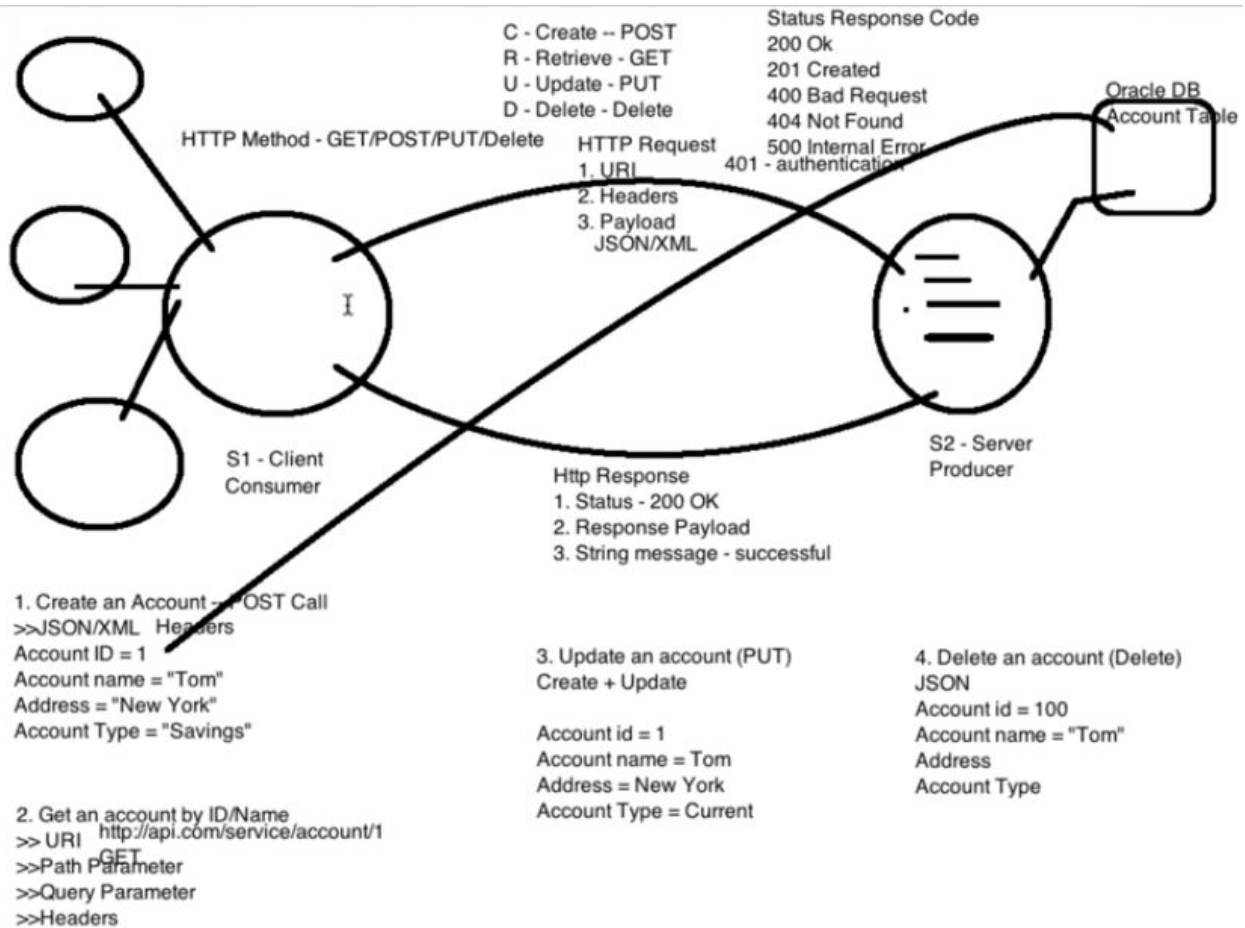
C - Create -> POST (create)

R - Retrieve -> GET

U - Update -> PUT (create + update)

D - Delete -> Delete

Naveen Automation Labs



Download POSTMAN (Rest Client) from Chrome Apps as a plugin or as a desktop client from Postman website.

URI = URL (EndPoint URL) + Service URL

https://youtu.be/Vnx2uBtn_bQ?list=PLFG0YjJG_fqp891lruz5fCRPWsDtEXJky&t=2202

UDEMY TRAINING

Running the API jar file

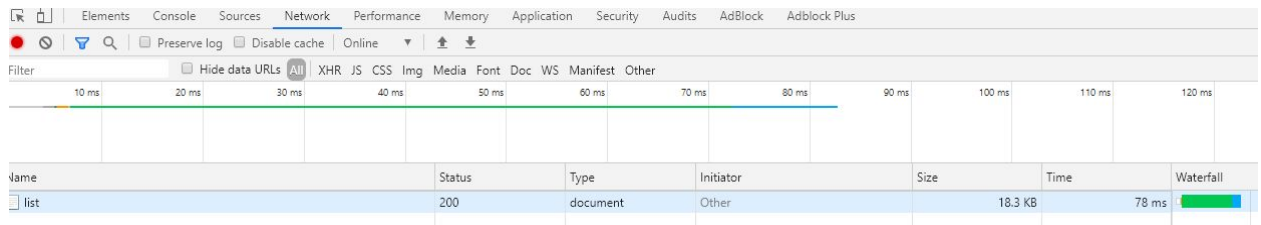
We will use a locally deployed API to learn this course. Run the studentApp.jar file from the command prompt.

By default it runs on port 8080. As 8080 is already taken by Jenkins, change the port number to 8085.


1. Go to the jar location (use cd).
2. Enter `java -jar studentApp.jar --server.port=8085`

Inspect in Chrome

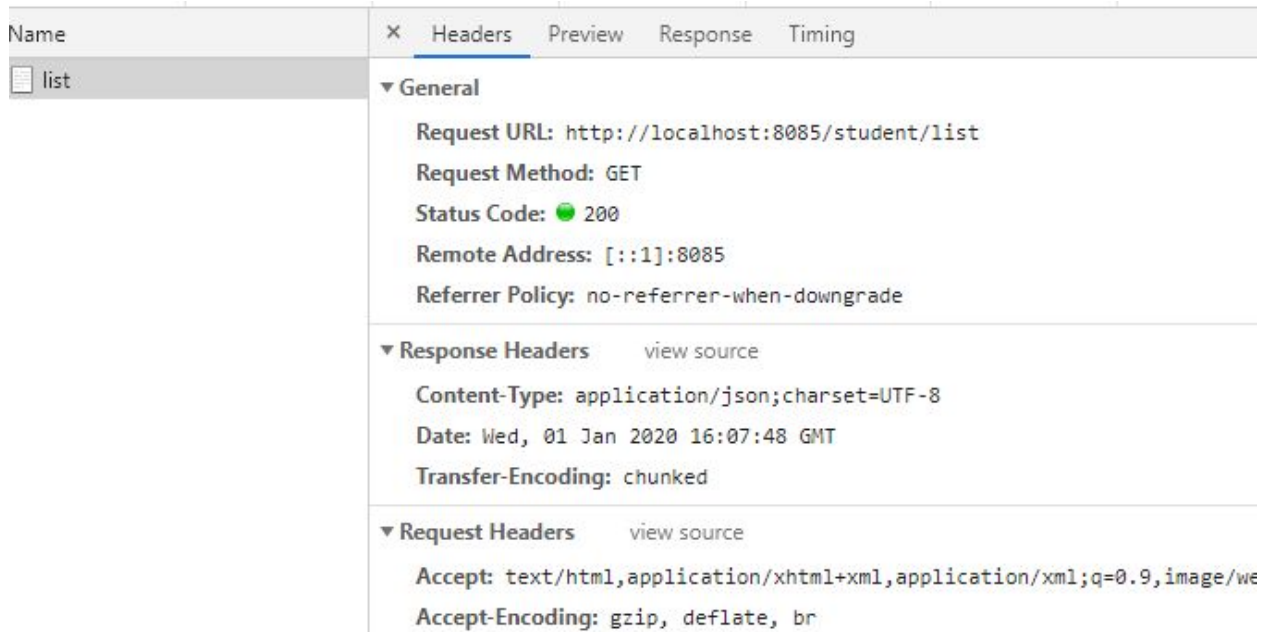
1. Open Chrome. Open Developer tools.
2. Enter `localhost:8085/student/list`
3. Go to Networks. Check status=200 (OK)



The screenshot shows the Chrome DevTools Network tab. The 'list' resource is selected, showing a status of 200, type of document, and a size of 18.3 KB. The waterfall view shows the request taking 78 ms.

Name	Status	Type	Initiator	Size	Time	Waterfall
list	200	document	Other	18.3 KB	78 ms	

4. Click the `<list>`.

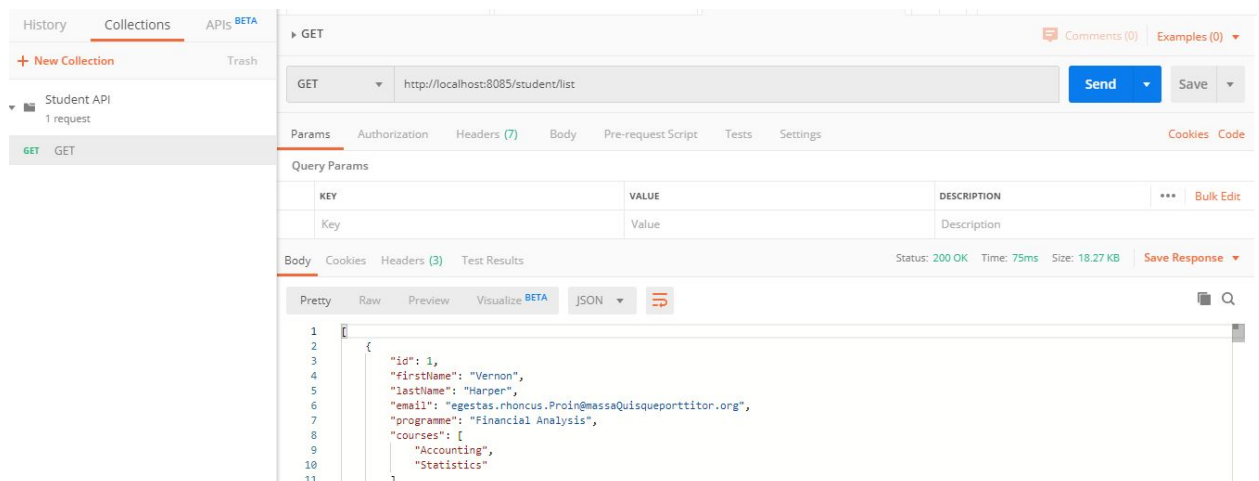


Postman

1. Install Postman desktop application
2. To save all requests create a collection. Add a folder e.g. GET for storing all GET requests.

GET Method

Send a GET request. Enter <http://localhost:8085/student/list> & check if you're getting a response back.



Property

To save the url, “<http://localhost:8085>” as a property, click gear icon->Add->Add Environment to create a new environment.

MANAGE ENVIRONMENTS

Add Environment

Local Host Environment

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	url	http://localhost:8085	http://localhost:8085			
	Add a new variable					

To use this property in GET, change to this new environment, & type

{{url}}/student/list and hit [Send].

GET by student id (status = 200 OK)

{{url}}/student/1 -> student with id as 1 is pulled up.

Query Parameter (where clause)

WHERE the response by *programme*

{{url}}/student/list?programme=Computer Science

Online JSON Viewer (to pretty format the response JSON)

<http://jsonviewer.stack.hu/>

Limit

Limit to only 2 students. Use ‘&’ to include additional parameters.

{{url}}/student/list?programme=Computer Science&limit=2

Bad Request (status = 400)

A negative limit. {{url}}/student/list?limit=-2

GET ▼ `{{url}}/student/list?limit=-2` Send ▼

Params ● Authorization Headers (7) Body Pre-request Script Tests Settings Co

Query Params

	KEY	VALUE	DESCRIPTION	...
<input checked="" type="checkbox"/>	limit	-2		
	Key	Value	Description	

Body Cookies Headers (4) Test Results Status: 400 Bad Request Time: 21ms Size: 187 B Save R

Pretty Raw Preview Visualize BETA JSON ▼ ≡

```

1  {
2    "error": "Limit should >= 0"
3  }

```

Using params feature

GET -> `http://localhost:8085/student/list`

Under Params tab, enter KEY as programme and VALUE as Computer Science. Notice Postman will automatically update the request url. You can also enter params using Bulk Edit (if you have stored it already).

```

programme:Computer Science
limit:2

```

GET ▼ `http://localhost:8085/student/list?programme=Computer Science&limit=2` Send ▼ Save ▼

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

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	programme	Computer Science			
<input checked="" type="checkbox"/>	limit	2			

Save all requests

History-> select all requests->Add to Collection

POST Method

Post is adding new data to the DB.

To post/add new student data, select POST from dropdown, enter URI {{url}}/student

Enter the new student information in the *Body* section. Choose 'raw'->Change text to **JSON**

Existing Data

```
{
  "id": 1,
  "firstName": "Vernon",
  "lastName": "Harper",
  "email": "egestas.rhoncus.Proin@massaQuisqueporttitor.org",
  "programme": "Financial Analysis",
  "courses": [
    "Accounting",
    "Statistics"
  ]
}
```

Modify existing data to fit new student info. Note: 'id' is auto-generated by DB, so you should not include it.

New Data

```
{
  "firstName": "Deesha",
  "lastName": "Dipin",
  "email": "monkey@monkey.com",
  "programme": "Circus",
  "courses": [
    "Nursery"
  ]
}
```

POST `{{url}}/student` Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL BETA ☒ JSON

```
1 {
2   "firstName": "Deesha",
3   "lastName": "Dipin",
4   "email": "monkey@monkey.com",
5   "programme": "Circus",
6   "courses": [
7     "Nursery"
8   ]
9 }
```

Body Cookies Headers (4) Test Results Status: 201 Created Time: 172ms Size: 203 B Sav

Pretty Raw Preview Visualize BETA JSON ≡

```
1 {
2   "msg": "Student added"
```

Verify if the new data was successfully added by sending a GET request

GET `{{url}}/student/list?programme=Circus` Send

Body Cookies Headers (3) Test Results Status: 200 OK Time: 24ms Size: 253 B Sav

Pretty Raw Preview Visualize BETA JSON ≡

```
1 [
2   {
3     "id": 101,
4     "firstName": "Deesha",
5     "lastName": "Dipin",
6     "email": "monkey@monkey.com",
7     "programme": "Circus",
8     "courses": [
9       "Nursery"
10    ]
11  }
12 ]
```

Note: The new data added stays good (in DB) as long as the student jar is running. If you restart the jar, new data is lost (only the default 100 student data comes up every time you run the jar).

PUT Method

PUT request is updating an existing data, here, a student data. E.g. Change programme from 'Circus' to 'school'.

Select PUT. Query by student ID. Enter URI `{{url}}/student/101`. Choose Body, set text to JSON.

Existing Data

```
{
  "id": 101,
  "firstName": "Deesha",
  "lastName": "Dipin",
  "email": "monkey@monkey.com",
  "programme": "Circus",
  "courses": [
    "Nursery"
  ]
}
```

→ Change Circus to School

Update data

```
{
  "firstName": "Deesha",
  "lastName": "Dipin",
  "email": "monkey@monkey.com",
  "programme": "School",
  "courses": [
    "Nursery"
  ]
}
```

PUT `{{url}}/student/101` Send

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL BETA JSON ▼

```
1 {
2   "firstName": "Deesha",
3   "lastName": "Dipin",
4   "email": "monkey@monkey.com",
5   "programme": "School",
6   "courses": [
7     "Nursery"
8   ]
9 }
```


Body Cookies Headers (3) Test Results Status: 200 OK Time: 32ms Size: 155 B

Pretty Raw Preview Visualize BETA JSON ▼ 

```
1 {
2   "msg": "Student Updated"
3 }
```

Verify update by sending a GET request

GET `{{url}}/student/101`

Pretty Raw Preview Visualize BETA JSON ▼ 

```
1 {
2   "id": 101,
3   "firstName": "Deesha",
4   "lastName": "Dipin",
5   "email": "monkey@monkey.com",
6   "programme": "School",
7   "courses": [
8     "Nursery"
9   ]
10 }
```

PATCH Method

If you want to **update only one field** use make a PATCH request

E.g. update the email ID from `'monkey@monkey.com'` to `'cutipie@peach.com'`

URI: `{{url}}/student/101`

Body:

```
{
  "firstName": "Deesha",
```

```
    "lastName": "Dipin",
    "email": "cutipie@peach.com",
    "programme": "School",
    "courses": [
        "Nursery"
    ]
}
```

The screenshot shows a REST client interface with a PATCH request to `{{url}}/student/101`. The request body is a JSON object with fields: `firstName` (Deesha), `lastName` (Dipin), `email` (cutipie@peach.com), `programme` (School), and `courses` (an array containing Nursery). The response status is 200 OK with a time of 38ms. The response body is a JSON object with `msg` (Updated).

PATCH `{{url}}/student/101`

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON

```
1 {
2   "firstName": "Deesha",
3   "lastName": "Dipin",
4   "email": "cutipie@peach.com",
5   "programme": "School",
6   "courses": [
7     "Nursery"
8   ]
9 }
```

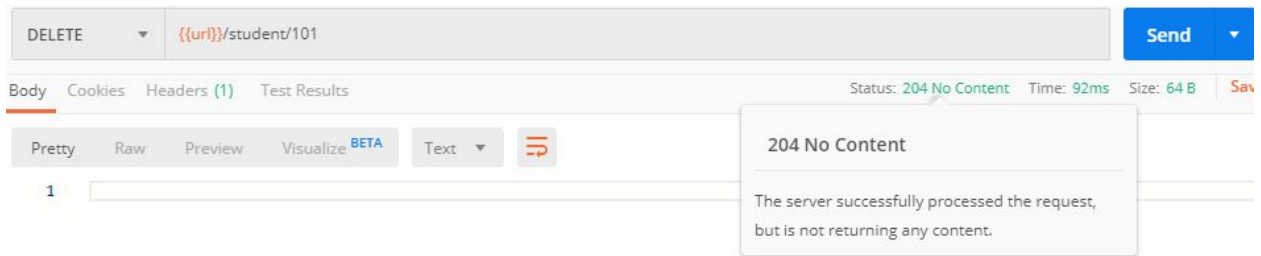
Body Cookies Headers (3) Test Results Status: 200 OK Time: 38ms

Pretty Raw Preview Visualize BETA JSON

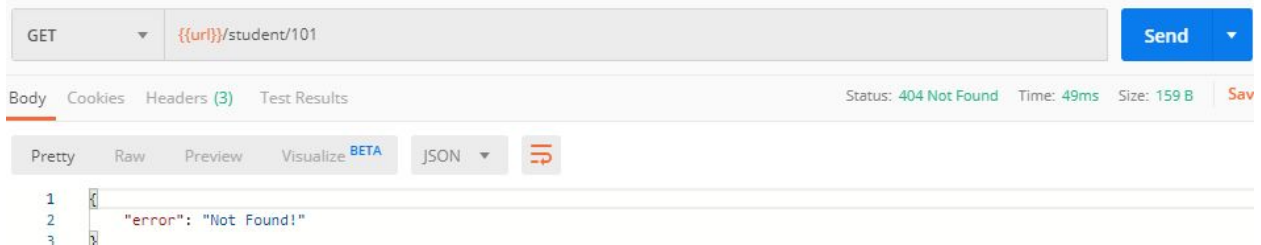
```
1 {
2   "msg": "Updated"
3 }
```

DELETE Method

It deletes a particular student data from the server. E.g. Delete the student with id as 101. Set to DELETE. Enter URI: `{{url}}/student/101`. Hit [Send]



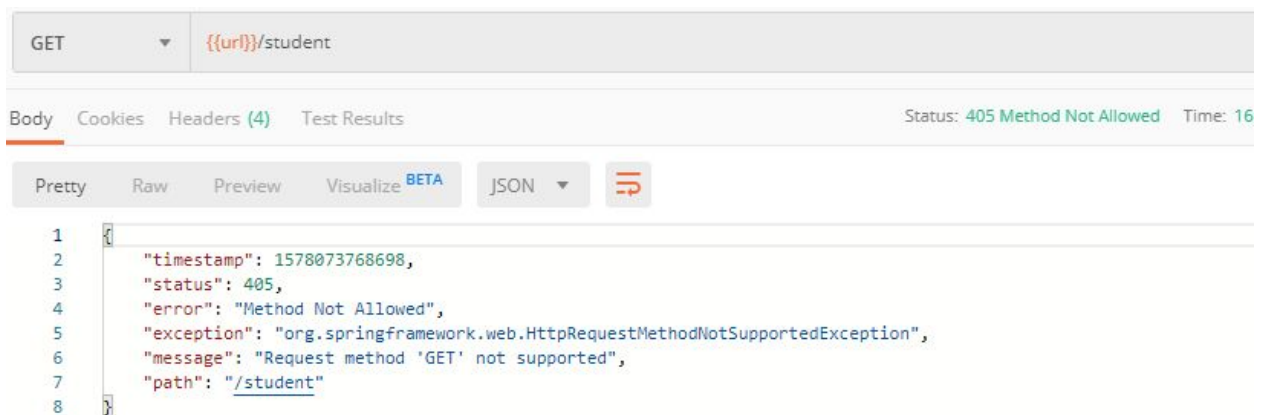
Verify using GET on ID 101 - no record found!



OPTIONS Method

OPTIONS request is like HELP section. E.g. if you want to know what methods are allowed on a resource URI, you use OPTIONS.

E.g. To add new student data, in POST request, the URI used was `{{url}}/student`. If you use this URI on a GET request, you'll get an error.



OPTIONS request to the rescue!

If you want to find out the methods allowed for a resource URI,

Create new tab in POSTMAN, select OPTIONS, enter URI `{{url}}/student` & hit [Send]

Click Headers tab. The methods allowed on this URI & other info are listed here.

OPTIONS	<code>{{url}}/student</code>	Send
Body	Cookies	Headers (3)
		Status: 200 OK Time: 29ms Size: 88 B
KEY	VALUE	
Allow	POST	
Content-Length	0	
Date	Fri, 03 Jan 2020 17:53:21 GMT	

Try using OPTIONS on URI `{{url}}/student/101`

OPTIONS	<code>{{url}}/student/101</code>	Send
Body	Cookies	Headers (3)
		Status: 200 OK Time: 13ms Size: 109 B
KEY	VALUE	
Allow	DELETE,PUT,GET,HEAD,PATCH	
Content-Length	0	
Date	Fri, 03 Jan 2020 17:57:15 GMT	

HEAD Method

HEAD request is similar to a GET request. It's used for ping a response.

A GET request will return a Body and a Header response whereas a HEAD will only return the Header response.

URI: `{{url}}/student/101`

Body

GET

{{url}}/student/101

Send

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 20ms Size: 251 B

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "id": 101,
3   "firstName": "Deesha",
4   "lastName": "Dipin",
5   "email": "cutipie@peach.com",
6   "programme": "School",
7   "courses": [
8     "Nursery"
9   ]
10 }
```

Header

GET

{{url}}/student/101

Send

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 20ms Size: 251 B

KEY	VALUE
Content-Type	application/json; charset=UTF-8
Transfer-Encoding	chunked
Date	Fri, 03 Jan 2020 18:08:19 GMT

Now try HEAD request on the same URI

No body is returned. Status=200 OK

HEAD

{{url}}/student/101

Send

Body Cookies Headers (3) Test Results

Status: 200 OK Time: 20ms Size: 123 B

Pretty Raw Preview Visualize BETA JSON

```
1
```

Headers

HEAD	{{url}}/student/101	Send
Body	Cookies	Headers (3)
		Status: 200 OK Time: 20ms Size: 123 B
KEY	VALUE	
Content-Type	application/json; charset=UTF-8	
Content-Length	121	
Date	Fri, 03 Jan 2020 18:09:56 GMT	

Uses:

HEAD request helps to test the health of a method. E.g. to check the content type, or content-length, to test if status is 200 OK, etc.

Rest Assured

Setup

1. Create a simple maven project.
2. Add dependencies:
 1. Rest-assured
 2. JUnit
3. Run the studentAPI jar file before you proceed

Static Imports

4. Import static imports

In order to use REST assured effectively it's recommended to statically import methods from the following classes

```
io.restassured.RestAssured.*
```

```
io.restassured.matcher.RestAssuredMatchers.*
```

```
org.hamcrest.Matchers.*
```

Refer: <https://github.com/rest-assured/rest-assured/wiki/Usage>

init() method

Base URI = http://localhost

Port = 8085

Base Path = /student ----> (here, student is the resource)

```
@BeforeClass
    public static void init() {
        RestAssured.baseURI = "http://localhost";
        RestAssured.port = 8085;
        RestAssured.basePath = "/student";
    }
```

Understanding given, when, then methods in Rest Assured

- given()-> Set cookies, add auth, add params, set header info
- when() -> Call GET, POST, PUT, DELETE, etc methods
- then() -> Validate status code, extract response, response-body, headers, cookies

GET using get()

URI: http://localhost:8085/student/list

Status Code: 200 OK

```
public void getAllStudentList() {

    when()
        .get("/list")
        .body()
        .prettyPrint();

}
```

Status Code validation

```
{
    given()
    .when()
    .get("/list")
    .then()
```

```

        .statusCode(200); // Negative scenario: 400 - Expected status
code <400> doesn't match actual status code <200>
    }

```

POST using post()

URI: `http://localhost:8085/student`

Status Code: 201 Created

1. Pass the body in JSON to the `body()` before the `post()` method.

This is the body in JSON

```

{
    "firstName": "Deesha",
    "lastName": "Dipin",
    "email": "monkey@monkey.com",
    "programme": "Circus",
    "courses": [
        "Nursery"
    ]
}

```

- 1.1 Create a Student java object & send it as the body (in JSON) to the POST method

- a. Create a Student class with private class variables String firstName, String lastName, String email, String programme & List<String> courses, under `src/main/java`.
- b. Write *getters & setters methods* for each variable (select all variables in eclipse IDE->Source->Generate getters & setters).
- c. Create a student object

// Create a Java Student object

```

Student student = new Student();
student.setFirstName("Deesha");
student.setLastName("Dipin");
student.setEmail("monkey@monkey.com");
student.setProgramme("Circus");
student.setCourses(Arrays.asList("Nursery",
"Gymnastics"));

```

- d. Add the *Jackson Databind* dependency to send this object as a JSON to the `body()` method. Set the **Content-type as JSON** under the `given()` method.
To post a body in JSON: REST Assured supports mapping Java objects to and from JSON. For JSON you need to have either Jackson, Jackson2, Gson or Johnzon in the classpath.

Courtesy: <https://github.com/rest-assured/rest-assured/wiki/Usage>

```

given()
  .contentType (ContentType.JSON) // Set Content-type as JSON
  .when()
  .body(student) // pass the student object in body()
  .post() // Student will be assigned a new id
  .then()
  .statusCode(201); // status code is 201 here

```

PUT using put()

URI: <http://localhost:8085/student/101> → (101 is the id of the student whose data is going to be modified)
 Status Code: 200 OK

Update the student of id 101

```

{
  "firstName": "Deesha",
  "lastName": "Dipin",
  "email": "cutiepie@peaches.com",
  "programme": "School",
  "courses": [
    "Playing",
    "Eating Fruits",
    "Ladder Climbing"
  ]
}

```

It's the same as the post() method with a few changes.

- You create a student object with the *updated* details.
- You have to pass the id of the student as a resource in put() method.
- Status code is 200.

```

given()
  .contentType (ContentType.JSON)
  .when()
  .body(student)
  .put("/101") // enter the 'id' here
  .then()
  .statusCode(200);

```

PATCH using patch()

patch() is very similar to the put() method except in patch() only one data is modified.

```
Student student = new Student();
student.setFirstName("Deesha");
student.setLastName("Dipin");
student.setEmail("monkey@monkey.com");
student.setProgramme("Pre-KG"); // only this parameter is changed
student.setCourses(Arrays.asList("Nursery", "Gymnastics"));
```

```
given()
.contentType(ContentType.JSON)
.when()
.body(student) // pass the student object in body()
.patch("/101")
.then()
.statusCode(200);
```

DELETE using delete()

URI: <http://localhost:8085/student/101> → (101 is the id of the student whose data is going to be deleted)
Status Code: 204

```
public void deleteStudentRecord() {
    when()
    .delete("/101")
    .then()
    .statusCode(204); // status code is 204 for delete()
}
```

TestBase class - Making init() method common across all methods

Because init() method is called for every request @BeforeClass, you may create a new TestBase class under *src/main/java* & add this method inside it. After that, just extend this class to your Get/Post/Put/Delete test classes.

```
public class TestBase {
    // Set init() method
    @BeforeClass
    public static void init() {
        RestAssured.baseURI = "http://localhost";
    }
}
```

```

        RestAssured.port = 8085;
        RestAssured.basePath = "/student";
    }
}

```

Logging

Logging Request Information

Printing Request logs

To print request logs, call the `log()` method and the 'what you want to print' (header, parameters, body,...) after the `given()` method.

1. Print only the Request Headers

```

// Get student list
given()
    .log()
    .headers() // Logs only the request headers
    .when()
    .get("/list")
    .then()
    .statusCode(200);

```

Output:

```

Headers:      Accept= */*

```

2. Print only the Request Parameters

```

// Input params
given()
    .param("programme", "Financial Analysis")
    .param("limit", "1")
    .log()
    .params() // Logs only the parameters
    .when()
    .get("/list")
    .then()
    .statusCode(200);

```

Output:

Params

Authorization

Headers (7)

Body

Pre-request Script

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	programme	'Financial Analysis'
<input checked="" type="checkbox"/>	limit	1

```
Request params: programme=Financial Analysis
                limit=1
```

Query params: <none>

```
Form params:      <none>
```

```
Path params:      <none>
```

```
Multiparts:      <none>
```

3. Print only the Request body

```
// Add new student
Student student = new Student();
student.setFirstName("Dipin");
student.setLastName("Dipin");
student.setEmail("test@test.com");
student.setProgramme("School");
student.setCourses(Arrays.asList("Rhymes", "Dance"));
given()
.contentType(ContentType.JSON)
.log()
.body() // Logs only the body (pretty-printed by default)
.when()
.body(student)
.post()
.then()
.statusCode(201);
```

Output:



```
{
  "firstName": "Dipin",
  "lastName": "Dipin",
  "email": "monkey123@monkey.com",
  "programme": "Phew",
  "courses": [
    "Nursery"
  ]
}
```

```
{
  "firstName": "Dipin",
  "lastName": "Dipin",
  "email": "monkey123@monkey.com",
  "programme": "Phew",
  "courses": [
    "Nursery"
  ]
}
```

4. Print all e.g. headers, cookies, body

```
// Add new student
Student student = new Student();
student.setFirstName("Deesha");
student.setLastName("Dipin");
student.setEmail("test@test.com");
student.setProgramme("School");
student.setCourses(Arrays.asList("Rhymes", "Dance"));
given()
.contentType(ContentType.JSON)
.log()
.all() // Logs everything including e.g. headers, cookies, body
.when()
.body(student)
.post()
.then()
.statusCode(201);
```


Output:

```
Request method:      POST
Request URI: http://localhost:8085/student
Proxy:               <none>
Request params:      <none>
Query params: <none>
Form params:  <none>
Path params:  <none>
Headers:        Accept=*//*
                Content-Type=application/json; charset=UTF-8
Cookies:        <none>
Multiparts:     <none>
Body:
{
  "firstName": "Sindiyal",
  "lastName": "Dipin",
  "email": "siln@seraph.com",
  "programme": "Work",
  "courses": [
    "Testing"
  ]
}
```

5. Print all e.g. headers, cookies, body only if validation fails

```
// Add the same student again to make the test fail
Student student = new Student();
student.setFirstName("Deesha");
student.setLastName("Dipin");
student.setEmail("test@test.com");
student.setProgramme("School");
student.setCourses(Arrays.asList("Rhymes", "Dance"));
given()
.contentType(ContentType.JSON)
.log()
.ifValidationFails() // Logs everything if a test validation fails
.when()
.body(student)
.post()
.then()
.statusCode(201);
```

Logging Response Information

Printing Request logs

To print request logs, call the `log()` method and the 'what you want to print' (header, parameters, body,...) after the `then()` method.

1. Print Response Headers

```
// GET student of programme="Financial Analysis" & limit search to '1'
given()
.param("programme", "Financial Analysis")
.param("limit", "1")
.when()
.get("/list")
.then()
.log()
.headers()
.statusCode(200);
```

Output:

Body Cookies Headers (3) Test Results		Status: 200 OK
KEY	VALUE	
Content-Type ⓘ	application/json; charset=UTF-8	
Transfer-Encoding ⓘ	chunked	
Date ⓘ	Wed, 08 Jan 2020 01:00:14 GMT	

Content-Type: application/json; charset=UTF-8

Transfer-Encoding: chunked

Date: Wed, 08 Jan 2020 01:01:22 GMT

2. Print Response Status

```
// GET student of programme="Financial Analysis" & limit search to '1'
given()
.param("programme", "Financial Analysis")
.param("limit", "1")
.when()
.get("/list")
.then()
.log()
```

```
.status()  
.statusCode(200);
```

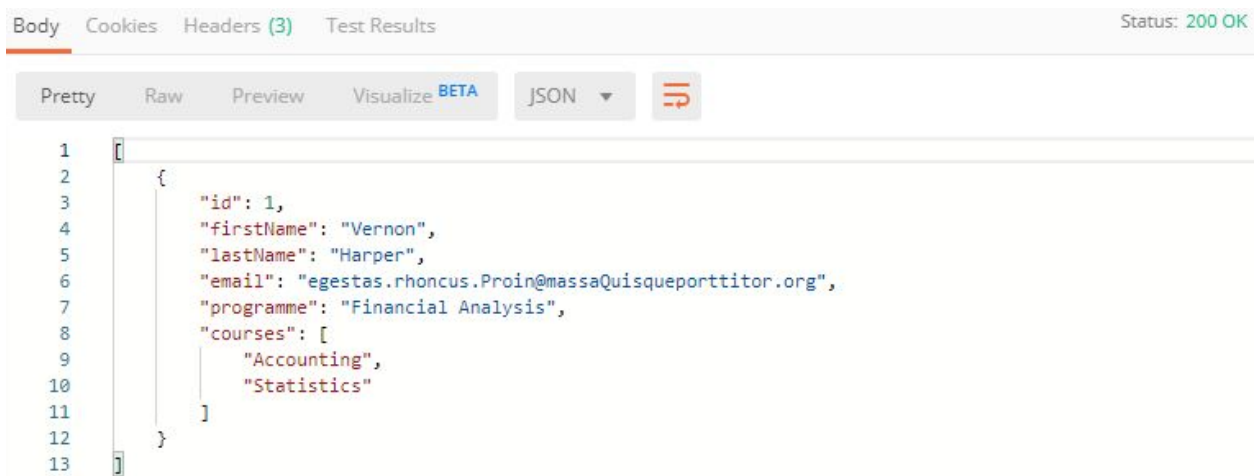
Output:

```
HTTP/1.1 200
```

3. Print Response Body

```
// GET student of programme="Financial Analysis" & limit search to  
'1'  
given()  
.param("programme", "Financial Analysis")  
.param("limit", "1")  
.when()  
.get("/list")  
.then()  
.log()  
.body()  
.statusCode(200);
```

Output:



```
[  
  {  
    "id": 1,  
    "firstName": "Vernon",  
    "lastName": "Harper",  
    "email": "egestas.rhonus.Proin@massaQuisqueporttitor.org",  
    "programme": "Financial Analysis",
```

```

        "courses": [
            "Accounting",
            "Statistics"
        ]
    }
]

```

4. Print all responses only if validation fails

```

// GET student of programme="Financial Analysis" & limit search to
'1'
given()
.param("programme", "Financial Analysis")
.param("limit", "-1") // To fail, pass limit as -1
.when()
.get("/list")
.then()
.log()
.ifError();

```

Output:

```

HTTP/1.1 400
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 08 Jan 2020 01:06:34 GMT
Connection: close

```

```

{
    "error": "Limit should >= 0"
}

```

Note: Conversely, if limit is passed as '1', it should not print any logs.

Extracting data (specific key-values) from JSON response

Walmart Search API Walkthrough

Scenario:

Go to Walmart Open API. This lets us query through Walmart's extensive catalogue. Search for any item, say, ipod. Extract the data from JSON response using Rest Assured.

1. Go to <https://developer.walmartlabs.com/>

- Click **[Developer Console]** tab. Enter the API key `75e3u4sgb2khg673cbv2gjjup`

The screenshot shows the 'Developer Console' tab of the 'Open API' interface. At the top, there are navigation links for 'Home', 'Developer Console', 'Documentation', and 'Blog'. A search bar is on the right. Below the navigation, the title 'I/O Docs: Interactive API Tool' is displayed, followed by a note: 'Test our API services using I/O Docs. You can also view our written documentation. Note that you will need your key to make the calls.' A dropdown menu is set to 'Walmart Open API'. Below this, the 'API Key' is entered as '75e3u4sgb2khg673cbv2gjjup'. There are two toggle buttons: 'Toggle All Endpoints' and 'Toggle All Methods'. The 'Lookup API' section is active, showing a list of endpoints: 'GET Items /v1/items/id', 'GET Reviews /v1/reviews/id', and 'GET Search /v1/search'. Each endpoint has links for 'List Methods' and 'Expand Methods'.

- Scroll down to Search API. Click the 'Search' next to GET. By default the *query* is set to ipod and format to JSON.

The screenshot shows the 'Search API' endpoint details. The title 'Search API' is at the top, with links for 'List Methods' and 'Expand Methods'. Below the title, there is a 'GET Search /v1/search' button. A description states: 'Allows text search on the Walmart.com catalogue and returns matching items available for sale online'. A table lists the parameters for the search:

Parameter	Value	Type	Description
query	ipod	string	Search text - whitespace separated sequence of keywords to search for
format	json	string	Format of the output in either JSON or XML
categoryId		int	Category id of the category for search within a category. This should match the id field from Taxonomy API
facet		string	Parameter to enable facets.
facet.filter		string	Filter to apply on the facet attribute values to narrow down the search.
facet.range		string	Range filter for facets which take range values, like price.

At the bottom of the table, there is a 'Try it!' button.

- Click **[Try it]**.

Try it! [Clear Results](#)

Request URI

```
http://api.walmartlabs.com/v1/search?query=ipod&format=json&apiKey=75e3u4sgb2khg673cbv2gjup
```

Request Headers [Select content](#)

```
X-Originating-IP: 49.206.127.7
```

Response Status [Select content](#)

```
200 OK
```

Response Headers [Select content](#)

```
Accept-Ranges: bytes
Content-Type: application/json; charset=utf-8
Date: Wed, 08 Jan 2020 01:25:40 GMT
Server: Mashery Proxy
X-Mashery-Responder: prod-j-worker-us-west-1c-15.mashery.com
X-Tb: 0
Content-Length: 73945
Connection: Keep-alive
```

Response Body [Select content](#)

```
{
  "query": "ipod",
  "sort": "relevance",
  "responseGroup": "base",
  "totalResults": 286,
  "start": 1,
  "numItems": 10,
  "items": [
    {
      "itemId": 514672965,
      "parentItemId": 514672965,
      "name": "Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A",

```

5. Click 'Select Content'.

Response Body [Select content](#)

```
{
  "query": "ipod",
  "sort": "relevance",
  "responseGroup": "base",
  "totalResults": 286,
  "start": 1,
  "numItems": 10,
  "items": [
    {
      "itemId": 514672965,
      "parentItemId": 514672965,
      "name": "Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A",
      "msrp": 99.99,
      "salePrice": 54.99,
      "upc": "788619271508",
      "categoryPath": "Home Page/Seasonal/Last Minute Gifts",
      "shortDescription": "Refurbished - This is a Refurbished item. While this item has been tested to be in working condition, it will show signs of use and cosmetic blemishes which do not affect the functionality of the item. Examples of which are: scratches, scuffs, dents, hairline cracks or dings The 16GB iPod touch (Space Gray) (5th Generation) from Apple has now reached its 5th generation, offering an incredible iPod touch as a music player, video camera, pocket computer and portable gaming device all in a single sleek anodized aluminum finish with exceptional battery life. The iPod touch offers a powerful A5 processor. The iPod touch also features the stunning Retina display with a Multi-Touch interface found on the iPhone 5. This is a refurbished iPod.",
      "longDescription": "Color: Space Gray, 5th Generation, Supports iOS 9, 16 GB flash memory4" Widescreen Multi-Touch IPS Retina 1136x640 Display, 800:1 contrast ratio, 500 cd/m&sup2; max brightness, Fingerprint-resistant oleophobic coatingFaceTime HD camera (1.2 MP photos, 720p HD video), Maps location-based service, Built-in speaker and microphoneRefurbishedPackage Includes:Apple iPod touch 16GBEarbud headphonesLightning to USB cableAC USB power adapterComes in plain white box (non-retail packaging)",
      "thumbnailImage": "https://i5.walmartimages.com/asr/7dfb497d-8deb-45f8-bb80-6a654511bbb6_1.15948cd973ec20332e7a6e10938049ef.jpeg?odnHeight=100&odnWidth=100&odnBg=FFFFFF",
      "mediumImage": "https://i5.walmartimages.com/asr/7dfb497d-8deb-45f8-bb80-6a654511bbb6_1.15948cd973ec20332e7a6e10938049ef.jpeg?odnHeight=180&odnWidth=180&odnBg=FFFFFF",
      "largeImage": "https://i5.walmartimages.com/asr/7dfb497d-8deb-45f8-bb80-6a654511bbb6_1.15948cd973ec20332e7a6e10938049ef.jpeg?"

```

6. To pretty format this JSON response, open a new tab, go to <http://jsonviewer.stack.hu/> And paste the contents under **Text** tab.

Request URI

http://api.walmartlabs.com/v1/search?query=ipod&format=json&apiKey=75e3u4sgb2khg673cbv2gjjup

Request Headers [Select content](#)

X-Originating-IP: 49.206.127.7

Response Status [Select content](#)

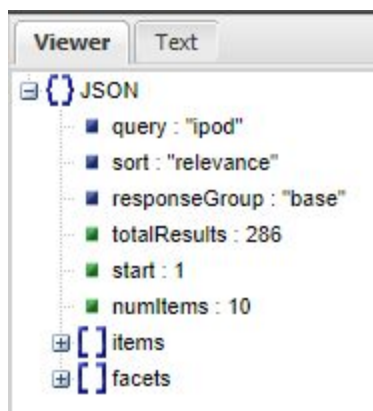
200 OK

Response Headers [Select content](#)

Accept-Ranges: bytes
Content-Type: application/json; charset=utf-8
Date: Wed, 08 Jan 2020 01:25:40 GMT
Server: Mashery Proxy
X-Mashery-Responder: prod-j-worker-us-west-1c-15.mashery.com
X-Tb: 0
Content-Length: 73945
Connection: keep-alive

Extract JSON Path

Extract 'numItems'



1. Setup Prerequisite:

```
static final String APIKEY = "75e3u4sgb2khg673cbv2gjjup";

@BeforeClass
public static void init() {
```



```
RestAssured.baseURI = "http://api.walmartlabs.com";
RestAssured.basePath = "/v1";
}
```

Extraction

Extracting key-value pairs

1. Extract 'numItems'

```
int numItems = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
    .when()
    .get("/search")
    .then()
    .extract()
    .path("numItems");
```

Output: 10

2. Extract 'query' (another example)

A String value is returned.

```
String query = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
    .when()
    .get("/search")
    .then()
    .extract()
    .path("query");
```

Output: ipod

Extracting a key-value from a hash map inside an array

3. Extract first name by providing list index value

Extract 'name' under [*items* -> {} o-> name

A String value is returned.

[*items* -> array

`{}` `0` -> hash map

name: "Refurbished..."

-> key-value pair



```
String name = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
    .when()
    .get("/search")
    .then()
    .extract()
    .path("items[0].name");
```

Output: Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4
Digital Music Video Player MGG82LL/A

Extracting a hashmap inside a hashmap inside an array (Inception? ;-))

4. Get the gift options for the first product

Extract 'giftOptions' under [items-> {}-> {}giftOptions

Note: This returns a *HashMap<String, String>*. The output is enclosed by square brackets `{}` indicating a *HashMap* is returned.

[] items -> array

{ } 0 -> hash map

{ } giftOptions -> hash map

```
HashMap<String, String> giftOptions = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
```

```

.when()
.get("/search")
.then()
.extract()
.path("items[0].giftOptions");

```

Output: {allowGiftReceipt=false, allowGiftWrap=true}

Note: Currently it does not have any value.

Print the size the array

5. Print the total size of items[]

Get size of[] items

```

List<String> names= given()
.queryParam("query","ipod")
.queryParam("apiKey",APIKEY)
.queryParam("format","json")
.when()
.get("/search")
.then()
.extract()
.path("items.size");

```

Extracting all the 'names' inside the array

6. Get all the 'names' from[] items i.e items[0].name, items[1].name...

```

List<String> names= given()
.queryParam("query","ipod")
.queryParam("apiKey",APIKEY)
.queryParam("format","json")
.when()
.get("/search")
.then()
.extract()
.path("items.name");

```

Output: [Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A, Apple iPod touch 7th Generation 32GB - Space Gray (New Model), Refurbished Apple iPod Touch 16GB MGG82LLA - Space Gray (5th generation), Apple iPod touch 7th Generation 128GB - Gold (New Model), Apple iPod Touch 6th Generation 16GB Refurbished, Refurbished Apple iPod Touch 6th Generation 16GB,

Refurbished Apple iPod Touch 5th Generation 16GB Pink MGFY2LL/A, AGPTEK 8GB MP3 Player with FM Radio, Voice Recorder, Music Player 70 Hours Playback & Supports up to 128GB, Rose Gold A02, Apple iPod touch 32GB - Blue (Previous Model), Insten iPod Touch 5th Generation / 6th Generation Screen Protector Tempered Glass LCD Film Cover Clear for Apple iPod Touch 5th Gen/6th Gen]

7. Get the all the values for Name==Apple iPod touch 32GB

Print all values where name='Apple iPod touch 32GB - Blue (Previous Model)' from []items

```
SELECT * from []items WHERE name='Apple iPod touch 32GB - Blue (Previous Model)'
```

```
List<String> names = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
    .when()
    .get("/search")
    .then()
    .extract()
    .path("items.findAll{it.name=='Apple iPod touch 32GB - Blue (Previous Model)'}");
```

Output: Too long to write here

Notes:

findAll() method

- **JsonPath** is an alternative to using **XPath** for easily getting values from an Object document. It follows the Groovy GPath syntax when getting an object from the document. You can regard it as an alternative to XPath for JSON.
- **GPath** similar to XPath expressions and you can use it not only with XML but also with POJO classes.

- Package: *java.util*

Method: **findAll(Closure closure)** --> *Closure is an independent block of code*

Return Type: Collection

Description: Finds all values matching the closure condition.

Reference:

1. <https://www.javadoc.io/doc/io.rest-assured/json-path/3.2.0/io/restassured/path/json/JsonPath.html>
2. http://docs.groovy-lang.org/latest/html/documentation/#_gpath
3. <http://docs.groovy-lang.org/latest/html/groovy-jdk/java/util/Collection.html>

Additional Filters

8. Get the sale price for Name==Apple iPod touch 32GB

```
SELECT salePrice FROM []items WHERE name = 'Apple iPod touch 32GB - Blue (Previous Model)'
```

Note: This returns a *List<Float>*. The output is enclosed by square brackets [] indicating a *List* is returned.



```
List<String> names = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
    .when()
    .get("/search")
    .then()
    .extract()
    .path("items.findAll{it.name=='Apple iPod touch 32GB'}.salePrice");
```

Output: [199.99]

9. Get the Names which have salePrice less than 150

```
SELECT name FROM []items WHERE salePrice < 150
```

Note: This returns a *List<String>*. The output is enclosed by square brackets [] indicating a *List* is returned.

```
List<String> names = given()
```

```

.queryParam("query","ipod")
.queryParam("apiKey",APIKEY)
.queryParam("format","json")
.when()
.get("/search")
.then()
.extract()
.path("items.findAll{it.salePrice<150}.name");

```

Output: [Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A, Refurbished Apple iPod Touch 16GB MGG82LLA - Space Gray (5th generation), Apple iPod Touch 6th Generation 16GB Refurbished, Refurbished Apple iPod Touch 6th Generation 16GB, Refurbished Apple iPod Touch 5th Generation 16GB Pink MGFY2LL/A, AGPTEK 8GB MP3 Player with FM Radio, Voice Recorder, Music Player 70 Hours Playback & Supports up to 128GB, Rose Gold A02, Refurbished Apple iPod touch 32GB 6th gen 4" MP3 MP4 Music Video Player - Gray - MKJ02LL/A]

10. Get the msrp of items that Start with name =Ref

```
SELECT msrp FROM []items WHERE `name` LIKE `Ref%
```

Note: This returns a *List<String>*. The output is enclosed by square brackets [] indicating a *List* is returned.

```

List<String> names = given()
.queryParam("query","ipod")
.queryParam("apiKey",APIKEY)
.queryParam("format","json")
.when()
.get("/search")
.then()
.extract()
.path("items.findAll{it.name==~/Ref.*/.msrp}");

```

Output: [99.99, 199.0, 199.0, null, 199.0]

11. Get the saleprice of items that End with name =ed

```
SELECT msrp FROM []items WHERE `name` LIKE `Ref%
```

Note: This returns a *List<String>*. The output is enclosed by square brackets [] indicating a *List* is returned.

```

List<String> salePrice = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "json")
    .when()
    .get("/search")
    .then()
    .extract()
    .path("items.findAll{it.name==~/.*ed/}.salePrice");

```

Output: [129.95]

Extracting data from XML response

- a. Get XML response from Walmart Open API by changing the format to xml.

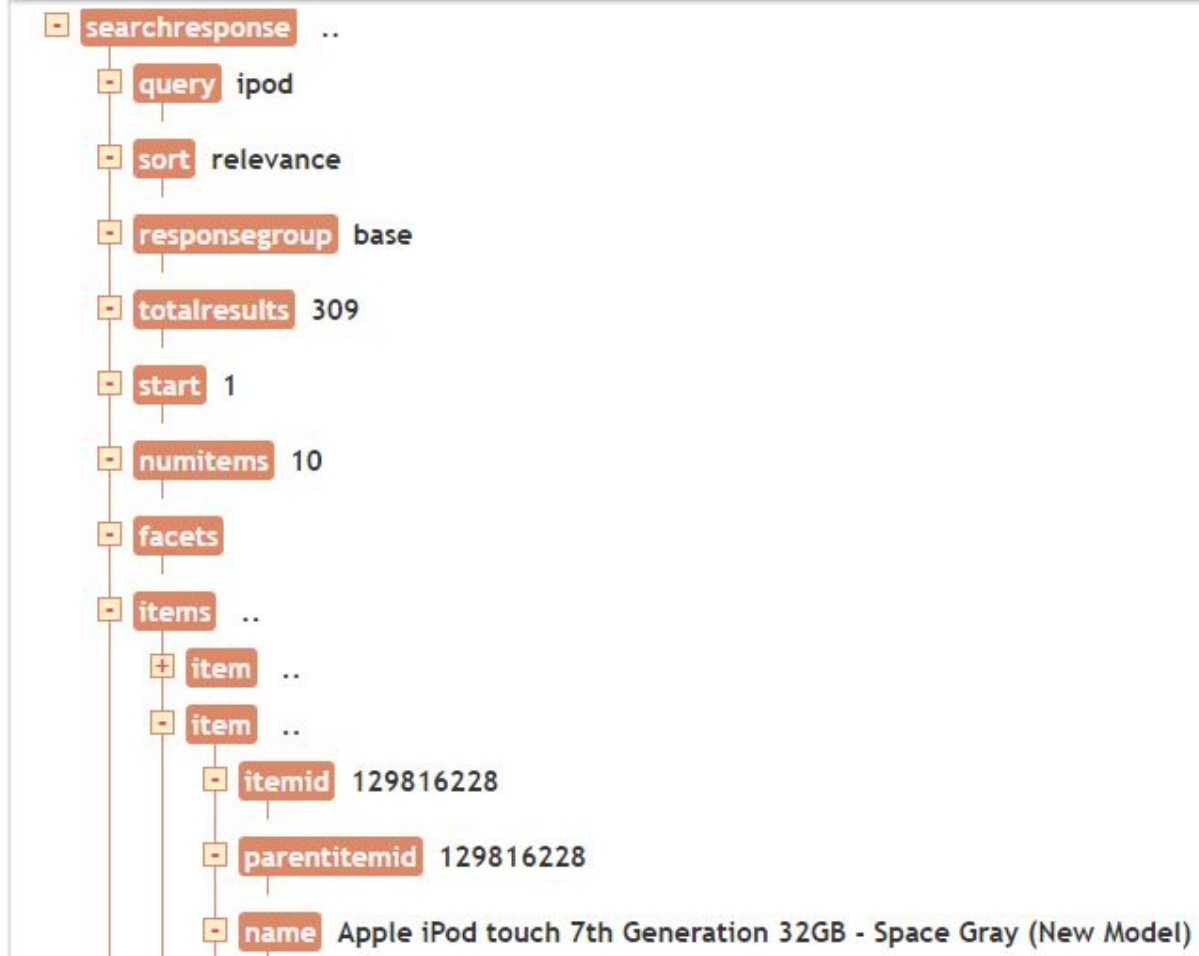
Search API

GET Search /v1/search

Allows text search on the Walmart.com catalogue and returns matching items available for sale

Parameter	Value	Type	Description
query	ipod	string	Search text - whitespace separated sequence of words
format	xml	string	Format of the output in either JSON or XML
categoryId		int	Category id of the category for search within from Taxonomy API

- b. To see XML in tree view, go to <https://codebeautify.org/xmlviewer#> & paste the code and click [Tree View].



1. Extract 'numItems'

```
String numItems = given()
    .queryParams("query","ipod")
    .queryParams("apiKey",APIKEY)
    .queryParams("format","xml") // change format to xml
    .when()
    .get("/search")
    .then()
    .extract()
    //.path("numItems"); // JSON
    .path("searchresponse.numItems");
```


Output: 10

2. Extract first name by providing list index value

searchresponse ->

items ->

item ->

name: "Refurbished..."



```
String productName= given()
    .queryParams("query","ipod")
    .queryParams("apiKey",APIKEY)
    .queryParams("format","xml")
    .when()
    .get("/search")
    .then()
    .extract()
// .path("items[0].name"); // JSON
    .path("searchresponse.items.item[0].name");
```

Output: Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4
Digital Music Video Player MGG82LL/A

3. Get the gift options for the first product

```
// STEP 1: Static Import:
import static io.restassured.path.xml.XmlPath.*;

// STEP 2: Get the whole xml response as a String value
String xml = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "xml")
    .when()
    .get("/search")
    .asString();

// TO print as as STRING -> getString
// STEP 3: Use with() (& its getString()) methods to get only the
values of the giftOptions

String giftOptions =
with(xml).getString("searchresponse.items.item[0].giftOptions");
```

Output: falsefalsefalse

Note: Currently it does not have any value.

4. Print the total size of items[]

Get size of[] items

```
// NOTE: Use NodeChildrenImpl object's size() method to get size.

NodeChildrenImpl val= given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "xml")
    .when()
    .get("/search")
    .then()
    .extract()
    // .path("items.size()"); // JSON
    .path("searchresponse.items.item"); // path() returns a NodeChildrenImpl
object

System.out.println("The size of the items is: "+ val.size());
```

5. Get all the 'names'

// Use the *with().getList()* method

```
String xml = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "xml")
    .when()
    .get("/search")
    .asString();
// .then()
// .extract()
// .path("items.name"); // JSON

List<String> names =
with(xml).getList("searchresponse.items.item.name");
```

Output: [Refurbished Apple iPod Touch 5th gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A, Apple iPod touch 7th Generation 32GB - Space Gray (New Model), Refurbished Apple iPod Touch 16GB MGG82LLA - Space Gray (5th generation), Apple iPod touch 7th Generation 128GB - Gold (New Model), Apple iPod Touch 6th Generation 16GB Refurbished, Refurbished Apple iPod Touch 6th Generation 16GB, Refurbished Apple iPod Touch 5th Generation 16GB Pink MGFY2LL/A, AGPTEK 8GB MP3 Player with FM Radio, Voice Recorder, Music Player 70 Hours Playback & Supports up to 128GB, Rose Gold A02, Apple iPod touch 32GB - Blue (Previous Model), Insten iPod Touch 5th Generation / 6th Generation Screen Protector Tempered Glass LCD Film Cover Clear for Apple iPod Touch 5th Gen/6th Gen]

6. Get the sale price for Name==Apple iPod Touch 6th Generation 16GB Refurbished

```
SELECT salePrice FROM []items WHERE name = 'Apple iPod Touch 6th Generation 16GB Refurbished'
```

```
String xml = given()
    .queryParams("query", "ipod")
    .queryParams("apiKey", APIKEY)
    .queryParams("format", "xml")
    .when()
```

```
.get("/search")  
.asString();
```

```
List<String> salePrice =  
with(xml).getList("searchresponse.items.item.findAll{it.name=='Apple  
iPod Touch 6th Generation 16GB Refurbished'}.salePrice");
```

Output: [161.82]

7. Deep search in XML path

```
String xml = given()  
    .queryParams("query", "ipod")  
    .queryParams("apiKey", APIKEY)  
    .queryParams("format", "xml")  
    .when()  
    .get("/search")  
    .asString();
```

```
List<String> salePrice =  
with(xml).getList("**.findAll{it.name=='Apple iPod Touch 6th  
Generation 16GB Refurbished'}.salePrice");
```

Output: [161.82]

Assertion

Validate numItems in the JSON response

1. Using JUnit Assertions

```
// Perform static import  
import static org.junit.Assert.*;  
  
//Extract numItems  
int numItems = given()  
    .queryParams("query", "ipod")  
    .queryParams("apiKey", APIKEY)  
    .queryParams("format", "json")  
    .when()  
    .get("/search")  
    .then()
```

```

.extract()
.path("numItems");
// Using JUnit assertion
// The total number of items are: 10
// import static org.junit.Assert.*;

assertEquals(10, numItems); // pass

```

2. Using Hamcrest

Without extracting an item, we can assert using hamcrest library.

// Add dependency

```

<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest-all -->
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-all</artifactId>
    <version>1.3</version>
    <scope>test</scope>
</dependency>

```

// Perform static import

```
import static org.hamcrest.Matchers.*;
```

My first Hamcrest test

We'll start by writing a very simple JUnit 5 test, but instead of using JUnit's `assertEquals` method, we'll use Hamcrest's `assertThat` construct and the standard set of matchers.

```

import org.junit.jupiter.api.Test;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;

public class BiscuitTest {
    @Test
    public void testEquals() {
        Biscuit theBiscuit = new Biscuit("Ginger");
        Biscuit myBiscuit = new Biscuit("Ginger");
        assertThat(theBiscuit, equalTo(myBiscuit));
    }
}

```

Refer:

<http://hamcrest.org/JavaHamcrest/>

<http://hamcrest.org/JavaHamcrest/tutorial>

Hamcrest Assertion Examples:

Validate that the JSON or XML response body conforms to one or more Hamcrest matchers.

1. Verify numItems



```
// Using Hamcrest assertion
given()
  .queryParams("query", "ipod")
  .queryParams("apiKey", APIKEY)
  .queryParams("format", "json")
  .when()
  .get("/search")
  .then()
  .body("numItems", equalTo(10)); // 11 will fail the test case
```

2. Verify query



```
given()
  .queryParams("query", "ipod")
  .queryParams("apiKey", APIKEY)
  .queryParams("format", "json")
  .when()
  .get("/search")
  .then()
  .body("query", equalTo("ipod"));
```

3. Check single name in ArrayList



```
given()
  .queryParams("query", "ipod")
  .queryParams("apiKey", APIKEY)
  .queryParams("format", "json")
  .when()
  .get("/search")
  .then()
  .body("items.name", hasItem("Refurbished Apple iPod Touch 5th
gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A"));
```

4. Check multiple names in ArrayList

```
given()
  .queryParams("query", "ipod")
  .queryParams("apiKey", APIKEY)
  .queryParams("format", "json")
  .when()
  .get("/search")
  .then()
  .body("items.name", hasItems("Refurbished Apple iPod Touch 5th
gen 16GB WiFi MP3 MP4 Digital Music Video Player MGG82LL/A",
"Apple iPod touch 7th Generation 32GB - Space Gray (New
Model)"));
```


Project Source Code

<https://github.com/dipindashoff/rest-assured-api-testing/tree/master/students-application-rest>

TUTORIALS

<https://www.journaldev.com/21501/rest-assured-tutorial>

<https://www.baeldung.com/rest-assured-tutorial>

<https://www.toolsqa.com/rest-assured-tutorial/>

<https://techbeacon.com/app-dev-testing/how-perform-api-testing-rest-assured>

REST with Java (JAX-RS) using Jersey - Tutorial

<https://www.vogella.com/tutorials/REST/article.html>

Tricks

JUnit

1. Running a single JUnit test in Eclipse

Select the method name under the @Test that you want to run. Right click->Run as->JUnit Test. This will run only this method.

Eclipse

- Press Ctrl+Shift+ ↑ (moves cursor to current method declaration)
- Press Alt+Shift+x (or d for debug) then press t (hotkey for "Run JUnit Test")
- press Alt+← (to get back to the line of code you were before)
- Or press Ctrl+Q (to go to last edit location)
- Ctrl+Shift+B (to add a breakpoint)
- Ctrl+O (to show all the methods under the class)
- Alt+Back (go back to where your cursor)

Troubleshooting

1. JUnit Exception: java.lang.SecurityException: class "org.hamcrest.Matchers"'s signer information does not match signer information of other classes in the same package

Fix: Replace the hamcrest-core jar file in eclipse/plugins with that of Maven

Replace the \$ECLIPSE_HOME\plugins\org.hamcrest.core_1.3.0.v201303031735.jar with Maven hamcrest-core-1.3.jar in .m2 repository (obviously renaming it to same name as eclipse jar)

Refer: <https://stackoverflow.com/questions/9651784/hamcrest-tests-always-fail>

2. java.lang.IllegalStateException: Cannot serialize object because no JSON serializer found in classpath. Please put either Jackson or Gson in the classpath

Fix: When you try to post a body in JSON using a Java object, you have to add Jackson Databind dependency to map this java object to JSON.

REST Assured supports mapping Java objects to and from JSON and XML. For JSON you need to have either Jackson, Jackson2, Gson or Johnzon in the classpath

Refer: <https://github.com/rest-assured/rest-assured/wiki/Usage>

3. When I try to import the 3 static import members in **src/main/java** -> **TestBase.java**, I get eclipse compile-time error: the imports cannot be resolved

```
import static io.restassured.RestAssured.*;

import static io.restassured.matcher.RestAssuredMatchers.*;

import static org.hamcrest.Matchers.*;
```

Fix: The reason is the <scope>test</scope> under each of the 3 dependencies. Remove it & it will work just fine. Refer: <https://github.com/rest-assured/rest-assured/issues/891>