SYNTAX
TECHNOLOGIES

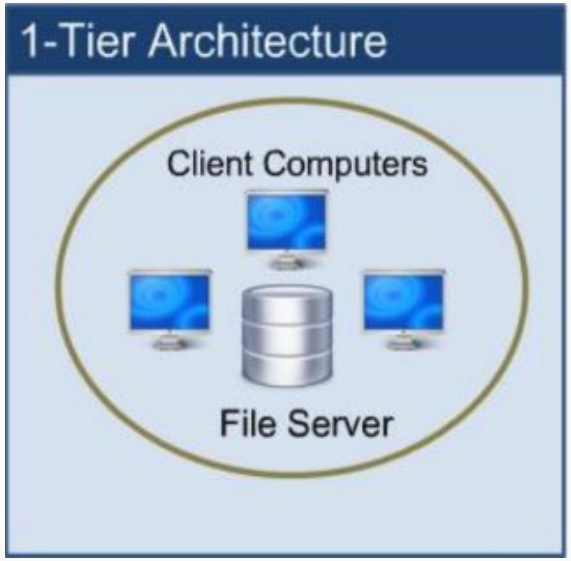
API/Web Services

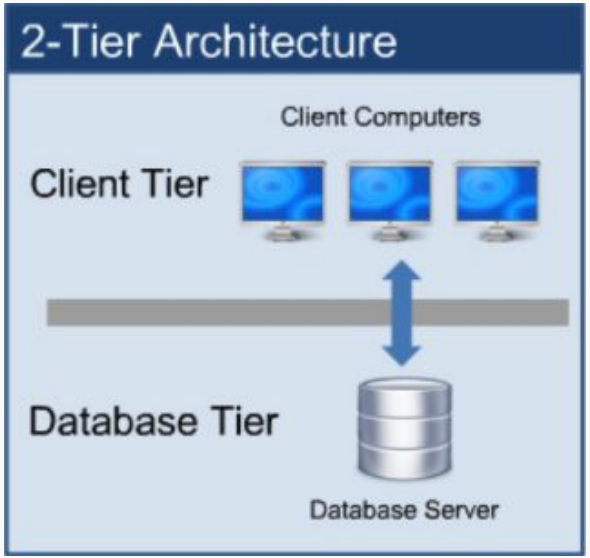
Class 1

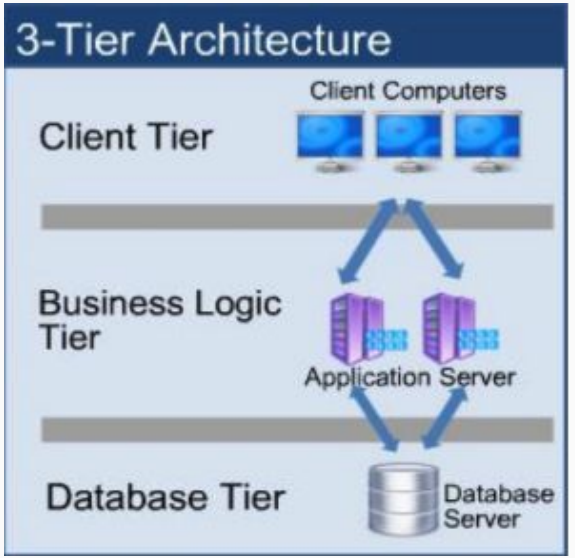
Agenda

Client/ Server Architecture
What is an API?
What is a Web Service?
Type of Web Services
Restful Web Services
Manual Testing with Postman

Client/Server Architecture

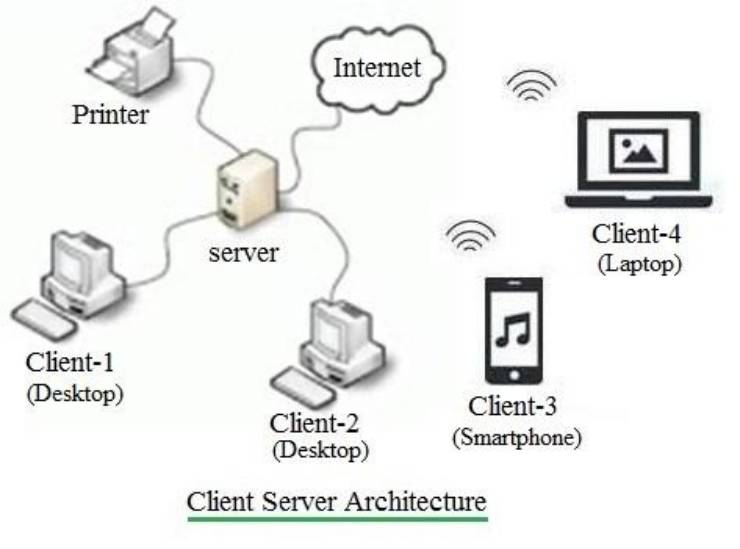






Presentation Layer → UI
Application Layer → Business logic is written in this tier. It is also called Business Tier. It connects front end and back end of application. (API)
Data Layer → DataBase (Oracle, SQL Server, MySQL, DB2, MongoDB)

Client/Server Architecture

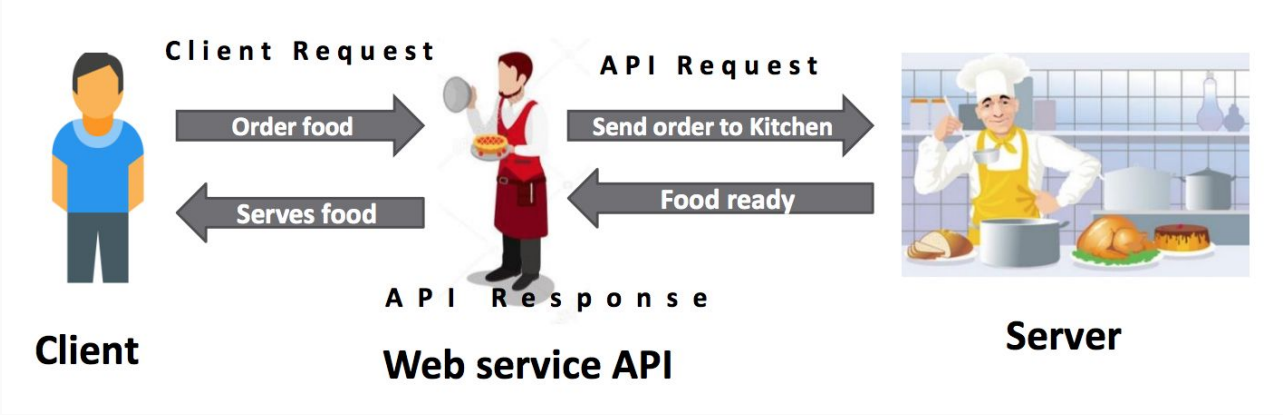


Client Server Architecture

A client is a computer hardware device or software that accesses a service made available by a server. The server is often (but not always) located on a separate physical computer.

A server is a physical computer dedicated to run services to serve the needs of other computers. Depending on the service that is running, it could be a file server, database server, or web server.

What are API's?



APIs, or Application Programming Interfaces, are the connecting tissue between different systems or layers of an application.

They are simply a method of communication between two systems

What are API's?

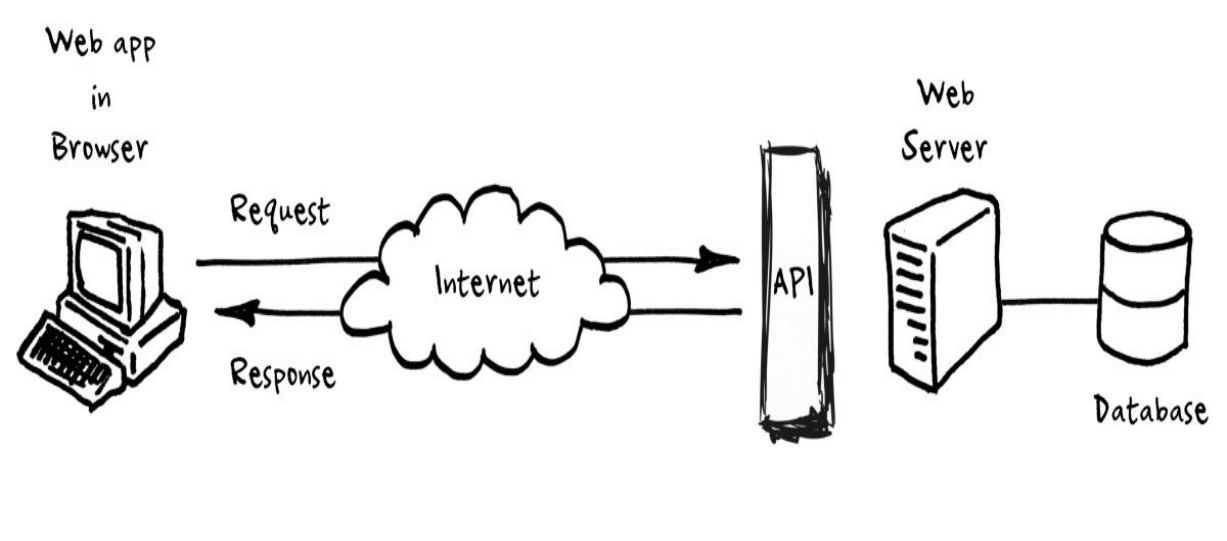
- **API** stands for **A**pplication **P**rogramming **I**nterface.
- **API** is a piece of software that plugs one application directly into the data and services of another by granting it access to specific parts of a server.
- **API's** enable **communication and data exchange between two separate software systems**.
- **API** is a set of codes which allows two or more than two applications to communicate each other internally or externally and provide a result to end users or to another API.
- **API** performs what user has requested without exposing internal logic or working of task done.

Why do we need API's?

We as Humans need a UI to interact with a system. We need to visually see what it is we need or want to do. We need to know what buttons to click, where to type a text, etc.. Without a UI we simply won't be able to communicate with any system. The UI is the method of communication between a human and a system.

Why do we need API's?

But systems don't need a UI to interact with other systems, instead they use other methods to communicate. One of those methods are through API's.

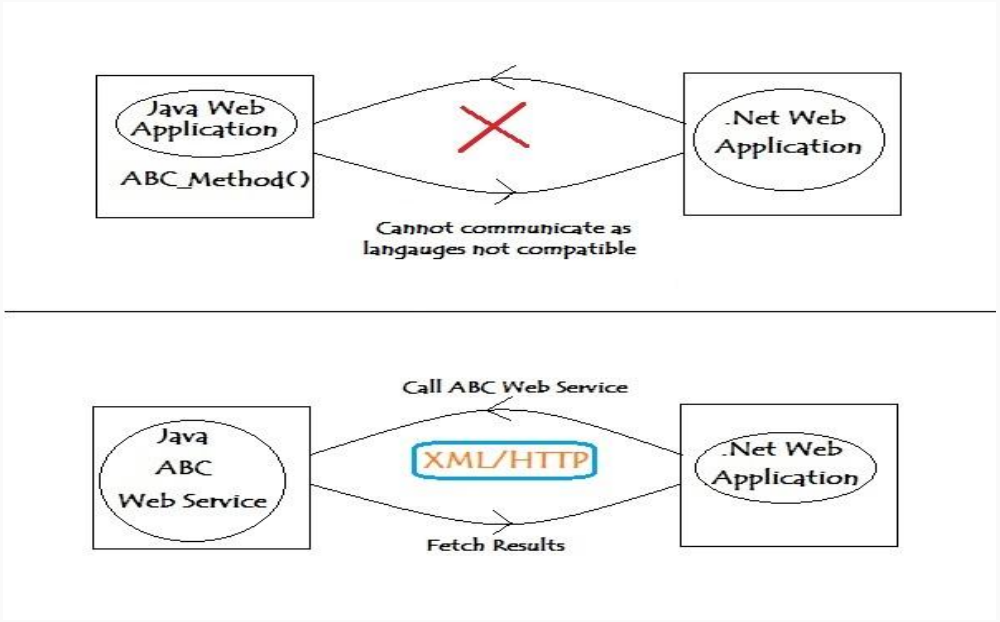


Why do we need API's?

- Systems developed in different technologies- there must be a common language for communication
- Companies want to protect their business logic, if Amazon wants to let you purchase an item on their website using your bank account, your bank account company will not allow Amazon to have direct access to their business logic/code/data therefore they must use API's

What are Web Services?

A Web service is a web application that can communicate with other web based applications over web. Web service implementation allows two web applications developed in different languages to interact with each other using a standardized medium like XML, JSON, HTTP etc.



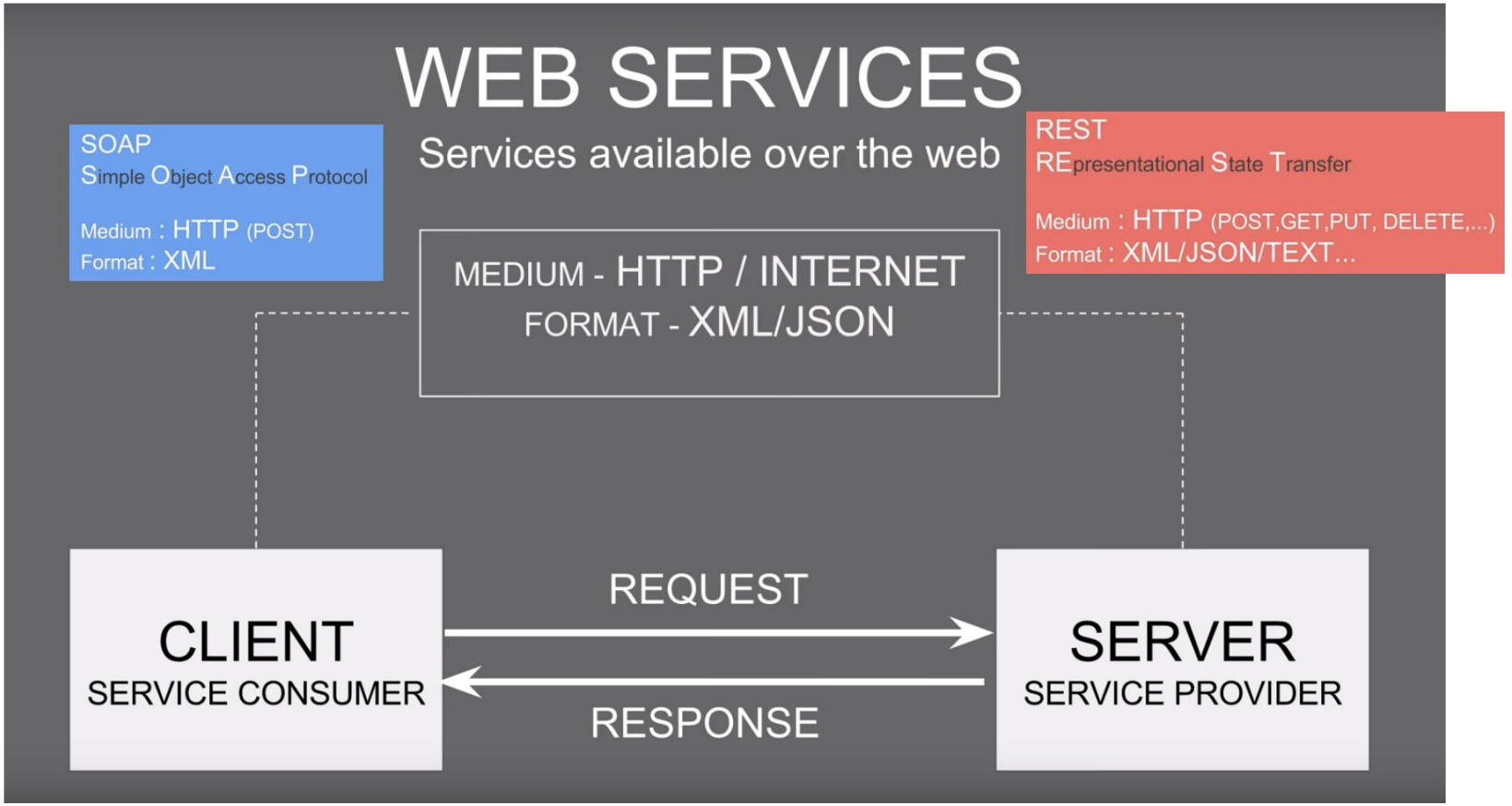
What are Web Services?

Web Services is the mechanism of communication through which two applications / machines will exchange the data irrespective of their underlying architecture and the technology.

Features of Web Services:

- As web services are based on open standards like XML, HTTP so these are **operating system independent.**
- Likewise web services are **programming language independent**, a java application can consume a PHP web service
- Web services can be **published over internet** to be consumed by other web applications

Types of Web Services?



What is SOAP?

Simple Object Access Protocol (SOAP) is a protocol that was designed to ensure that programs built on different platforms and programming languages could exchange data in an easy manner.

SOAP uses XML for communication and supports only POST and GET calls

What is REST?

Representational State Transfer (REST) is a style of architecture based on a set of principles that describe how networked resources are defined and addressed.

REST can also be referred to as **RESTful APIs** or **RESTful web services.**

A RESTful API is an application program interface (API) that uses HTTP requests to **GET, PUT, PATCH, POST and DELETE data.**

A Rest API Works the same way normal Client Server Application Works.

SOAP vs REST

REST Web Service	SOAP Web Service
REST is an architectural style, any web service following REST architecture is called RESTFul Web Service.	SOAP is a protocol, a set of rules which must be followed while creating a web service
Supports multiple data formats - JSON, XML, CSV etc.	Supports XML message format only
REST resources are exposed by the service URI and HTTP Verbs- GET, PUT, POST and DELETE	SOAP exposes its named methods to be consumed by a client e.g. an open SOAP web service "WeatherWS" exposes its various operations as methods- GetWeatherInformationResponse, GetCityForecastByZIP etc
REST is considered light weight and faster(no XML parsing is required)	Slower than REST because of the use of XML format but considered more secure (uses WS-security)

Why API Testing?

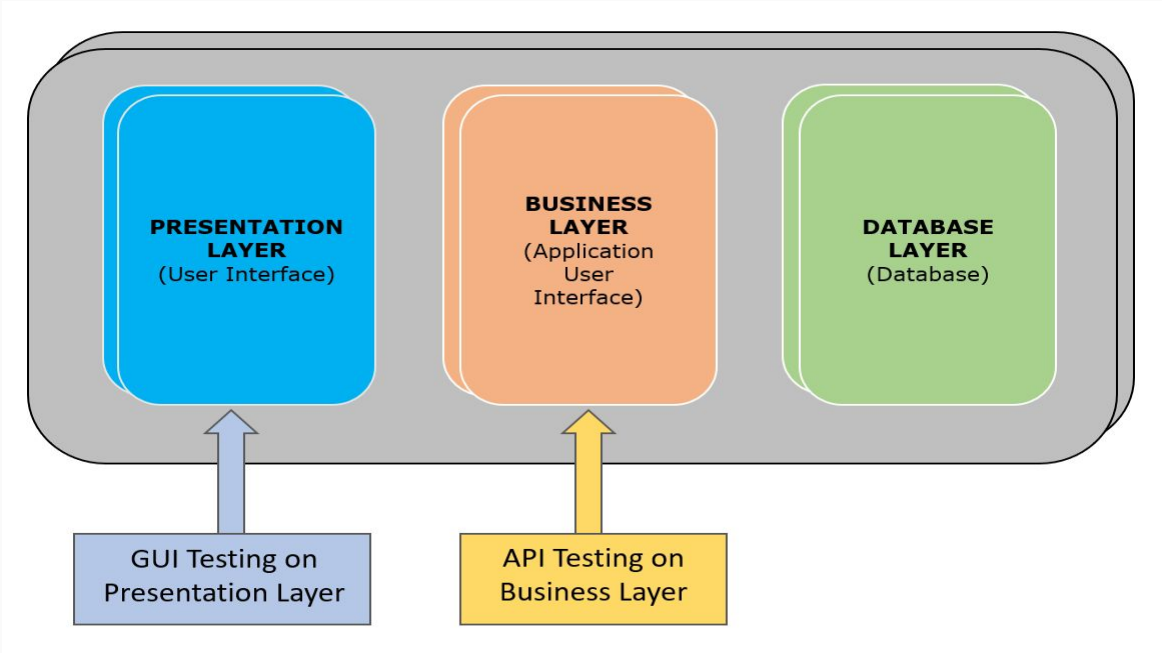
In the agile development world, requirements are changing during short release cycles frequently and GUI tests are more difficult to maintain according to those changes.
Thus, API testing becomes critical to test application logic.

In GUI testing we send inputs via keyboard texts, button clicks, drop-down boxes, etc.,

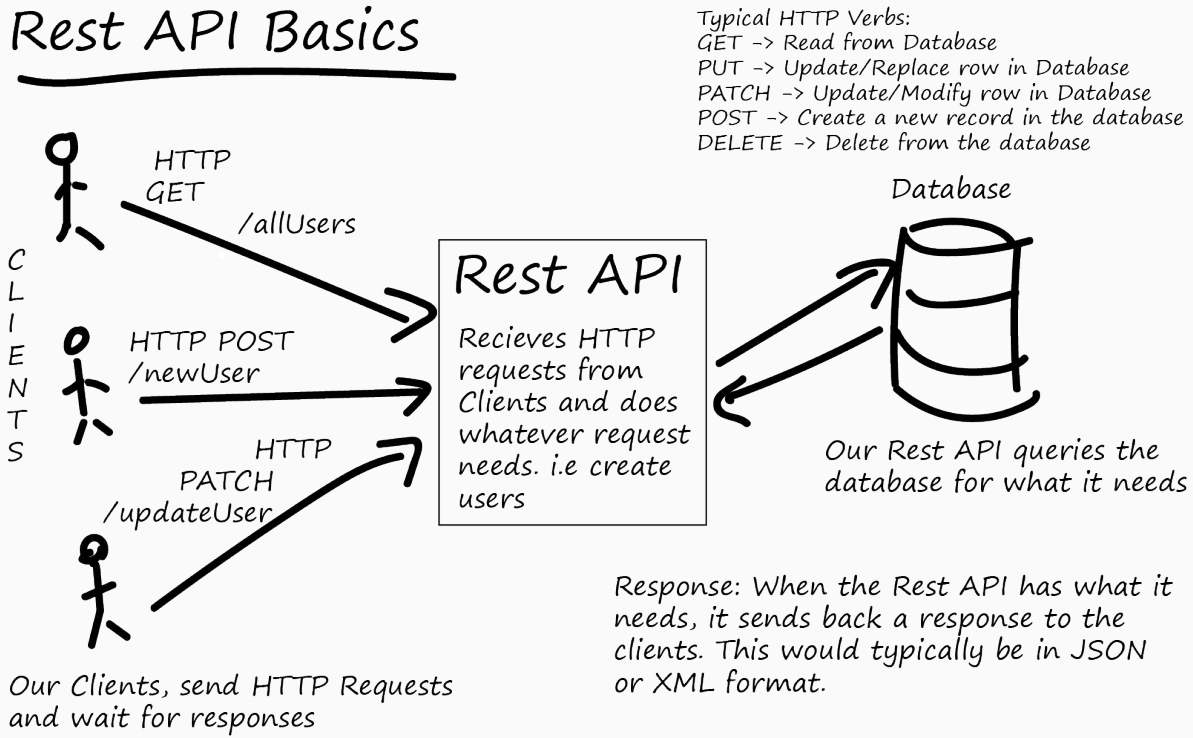
In API testing we send requests (method calls) to the API and get output (responses).

What is API Testing?

API testing tests business logic tier directly and checks expected functionality, reliability, performance, and security.
API testing won't concentrate on the look and feel of an application.



REST API



A service represents a resource that can be accessed from the web

Every resource is uniquely addressable using a uniform and minimal set of commands (typically using HTTP commands of GET, POST, PUT, or DELETE over the Internet)

REST API

An API is developed to serve a purpose/functionality which may fall in one of these categories:

Creating data , Retrieving data, Updating data or Deleting data.

Those are basic operations performed by APIs. These functions are called as CRUD(Create, Retrieve, Update and Delete) as an acronym.

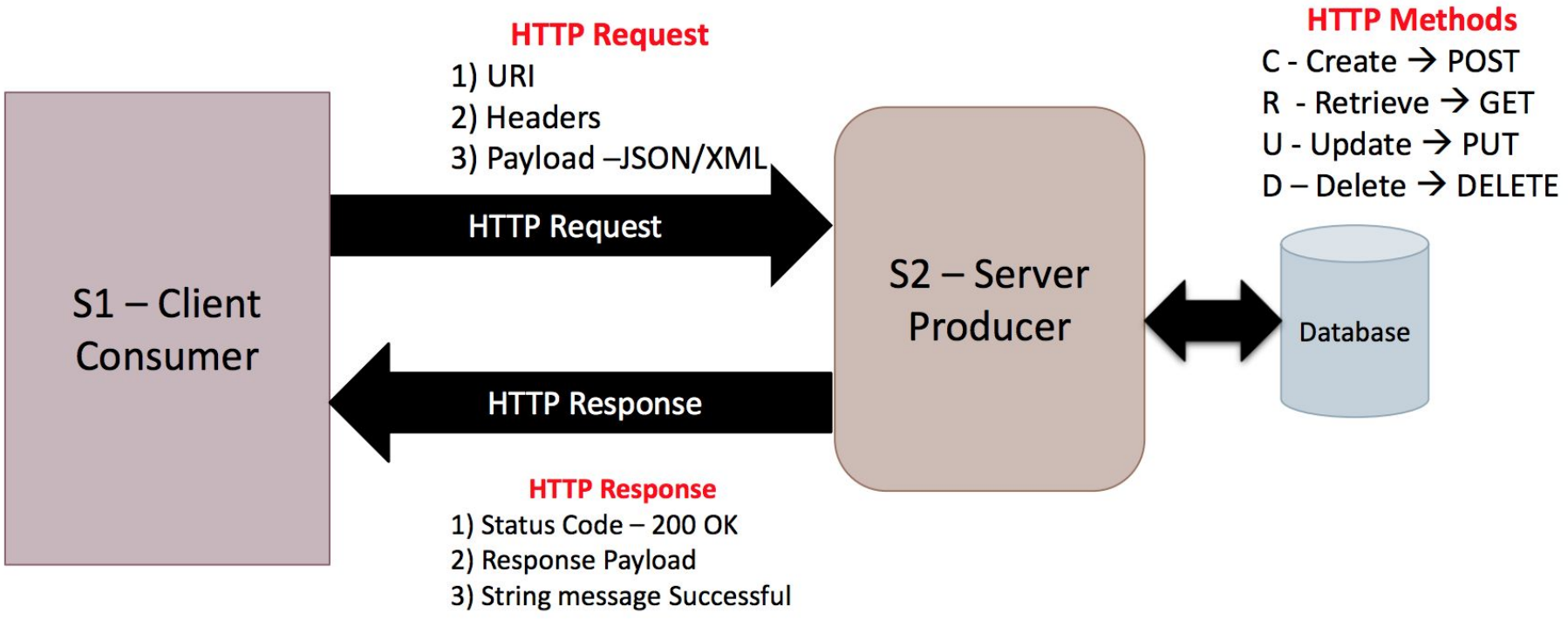
In REST, these operations are called as HTTP methods. There are more HTTP methods other than these. Like GET, POST, PUT, DELETE, PATCH etc.

REST API

REST commonly makes use of different HTTP request methods to implement CRUD functions:

POST – Create new resource
GET – Retrieve resource
PUT – Update resource
DELETE – Delete resource

REST API Flow



SYNTAX
TECHNOLOGIES

API

Class 2

Agenda

Manual Testing API’s/Web services with Postman

Tools to Test APIs/Web services

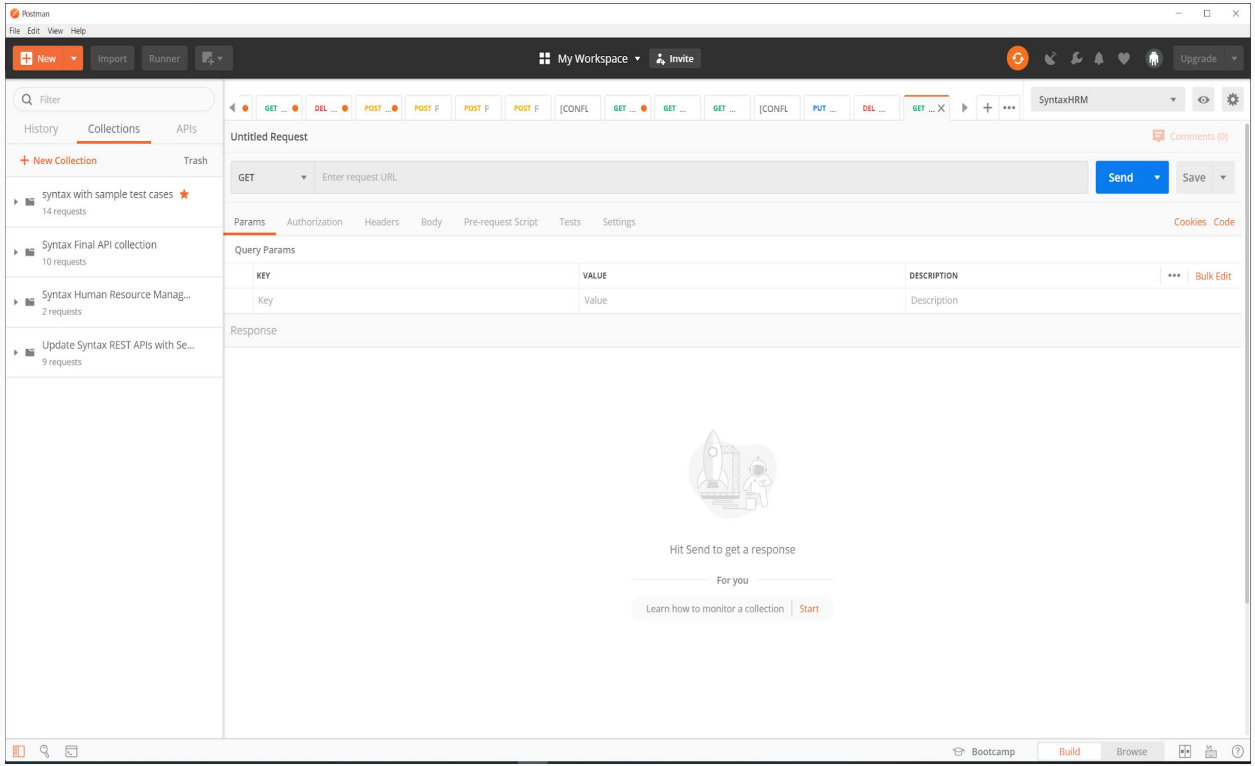
- Postman
- SoapUI
- SoapUI Pro
- Ready! API
- HTTP Client
- REST Assured

Postman

- We will be using Postman as it is the most popular manual testing tool in the market today
- Postman offers us the flexibility to organize our “Calls” as needed
- We will learn how to make “calls” to our API’s or “hit” our endpoints
- As the course goes on you will familiarize yourself with API terminology

Postman

- Download Postman, then open Postman and you should see a similar screen to the one below



Rules

- To make a call to an API we must use HTTP methods such as POST, GET, PUT, DELETE, etc.
- Let’s recap on what these calls do
- Using API terminology
  POST - we are creating data
  GET - we are retrieving data
  UPDATE/PUT - we are updating data
  PATCH - updating one or a couple specific sets of key pairs instead of passing a full payload
  DELETE - we are deleting data

Rules

To make a call to an endpoint/API we need the following:

First thing we need is a URI

URI - Uniform Resource Identifier

What is a URI?

URI → Domain URL/Base URL + API/web Service URL/resource/endpoint

URL - this would be your Domain URL/Base URL - amazon.com, google.com, facebook.com, etc.

API Endpoint

An endpoint is one end of a communication channel.
When an API interacts with another system, the touchpoints of this communication are considered endpoints.
An endpoint is simply a unique URL that represents an object or collection of objects.
Each endpoint is the location from which APIs can access the resources they need to carry out their function.

Rules

In this example our domain/base URL will be: http://www.xyx.com
Service URL is what you will “attach” or append to your Domain/Base URL

In this example our service/endpoint/resource URL will be: /createUser

So our URI would be: http://www.xyz.com/createUser

Rules

What is the difference between URI, URL, and API service URL/resource?

URI = Domain/Base URL + ServiceURL

- You would need a URL and API service URL/resource to create a URI

Rules

- We will also need headers to be sent with our request
- With our headers we will specify what type of request we are going to send
- If we are sending a request in JSON format then we need to specify that
- Specifying we are sending JSON request: Content-Type = application/json
- We will also need to pass our token/key as a header(if one is required)
- Authorization = “Bearer(space) your JSON web token”

Bearer

- What is Bearer?
  - Bearer is a keyword for authorization, bearer is an identifier, whenever you have to authorize any specific key you need to pass the key with ‘Bearer’ keyword
- When we talk about the security domain, there are a lot of protocols we need to follow, one of them is to authenticate yourself using ‘Bearer’ or else you won’t be given authority
- You can learn more about ‘Bearer’ by visiting:
- https://oauth.net/2/bearer-tokens/

POST Request

With POST you will need:

- Headers
  - BODY/Payload (your data in JSON, XML, String, etc)

JSON example:

{
“Key” : “Value”,
“Key” : “Value”,
“Key” : “Value”,
“Key” : “Value” - with last pair you don’t use a comma (,)
}

POST Request

HTTP POST requests are used to create new resources. The basic idea is to pull data out of the HTTP request body and use it to create a new row in the database.

Request that the resource at the URI do something with the provided entity.

Often POST is used to create a new entity, but it can also be used to update an entity.

A HTTP POST method is used to create a new resource in collection of resources with a request body passed as a JSON/XML.

PUT Request

PUT requests will be used to make updates to existing resources. It's important to note that PUT is used to replace the entire resource – it doesn't do partial updates.
PUT can create a new entity or update an existing one.
PUT is not a safe method as it performs data creation and modifications

A PUT request is idempotent. Idempotency is the main difference between the expectations of PUT versus a POST request.

Example: **PUT /api/updateStudentProfile**

GET Request

With GET you will need:

- URI
- Headers

No BODY/Payload is required since GET you are only retrieving data from a server and not creating
If you need to send data with a GET call to narrow down your search then you can send your data in form of JQuery Parameters or Path Parameters

GET Request

Retrieves information.

GET requests must be safe and idempotent, meaning regardless of how many times it repeats with the same parameters, the results are the same.

Example1: **Get/api/getAllStudentProfiles**
View a profile of available syntax student

Example2: **Get/api/getStudentProfile/{studentId}**
View a profile of specific syntax student

GET Request

This HTTP method is used to **read/retrieve** resource representation only

Status Codes:
200 (OK) –> If GET API finds the requested resource.
404(Not Found) –> If GET API does not find the requested resource.
400 (Bad Request) –> If GET request is not formed properly.

Parameters

Parameters are options you can pass with the endpoint.

There are two types of parameters:

- **path parameters**
- **query string parameters**

Not all endpoints contain each type of parameter.

JQuery Parameter

These Parameters are sent with the URI
Using the URI we created above:

http://www.xyz.com/createUser

but for this example let's change our endpoint URL to /getUser because with a GET call we want to retrieve data or information

So our new URI is now:
http://www.xyz.com/getUser

JQuery Parameter

Now lets send parameters. We know that /getUser will GET us a user but lets narrow down our search to find “John Doe”
So our URI with parameters will be:
http://www.xyz.com/getUser?firstName=John&lastName=Doe

JQuery Parameters will start after the question mark ‘?’
AND if we are sending more than one parameter then we separate with ‘&’ operator

Path Parameter

What is a PATH parameter?
So looking at the URI we created above:

http://www.xyz.com/getUser?firstName=John&lastName=Doe

Lets remove the JQuery parameters for this example so our new URI will now be
http://www.xyz.com/getUser

Path Parameter

Now if we want to add a PATH parameter to narrow down our search then our new URI will look like:

http://www.xyz.com/getUser/firstNames

So as you see, we added /firstNames as a path parameter. This is NOT a JQuery parameter as we mentioned above. The whole point of PATH and JQuery parameters is really just to narrow down our search. So by adding this PATH parameter we really narrowed our search to get the first names of users

PUT Request

With a PUT(update) call you will need:

- URI
- Headers

AND a body/payload

Note: If you are attempting to update information that does not exist in given server then PUT will behave as a POST call and create the information UNLESS developers have restricted that from happening

DELETE Request

With a DELETE call you will need:

- URI
- Headers

AND you may/may not need a payload

If a payload is not required then you will send data in form as JQuery parameter or PATH parameter

DELETE Request

It is used to **delete** a resource identified by a URI. In other words, a 204 status with no body, or the JSEND-style response and HTTP status 200 are the recommended responses.
Request that a resource be removed; however, the resource does not have to be removed immediately. It could be an asynchronous or long-running request.

Example: DELETE
/api/deleteStudentProfile/{studentId}

- delete available syntax student profile

DELETE Request

It returns 200 (OK) ,204 (No Content) status code.

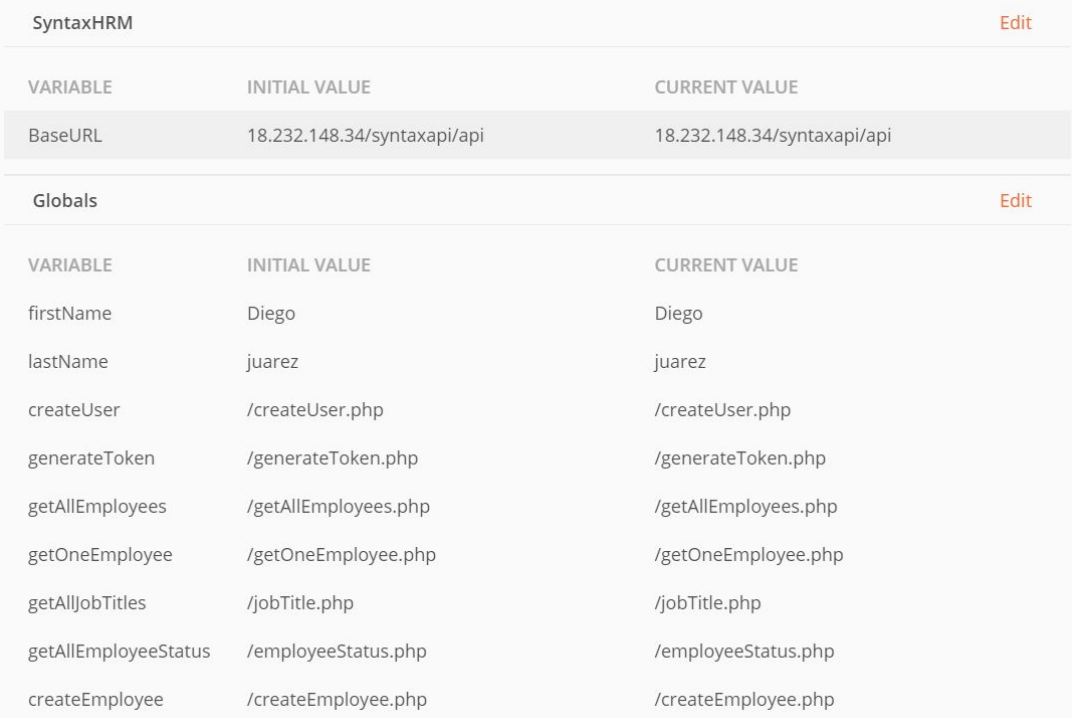
It may return as 202 (Accepted) status code if request is queued.

It is not a safe method as it performs on modification of data.

If we hit the same request again after first hit, it will give you 404 (Not Found) .
So DELETE request are idempotent after second call onward.

Postman

- Making calls
- Setting environment variables
- Setting global variables(Variables we can use globally in any environment)



SyntaxHRM — Edit

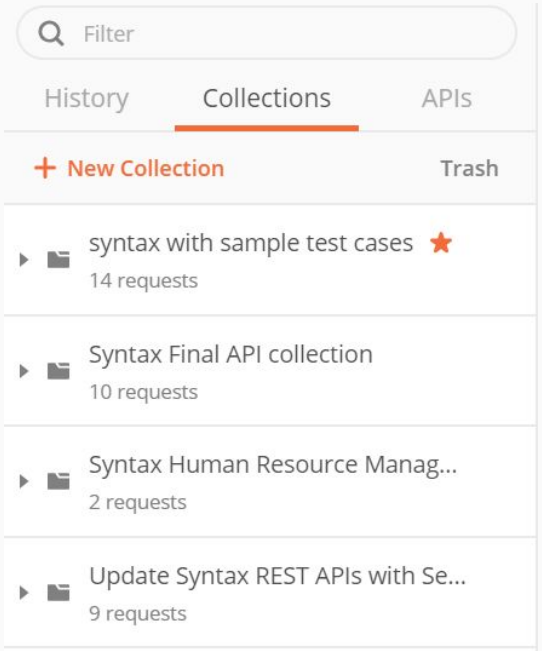
VARIABLE	INITIAL VALUE	CURRENT VALUE
BaseURL	18.232.148.34/syntaxapi/api	18.232.148.34/syntaxapi/api

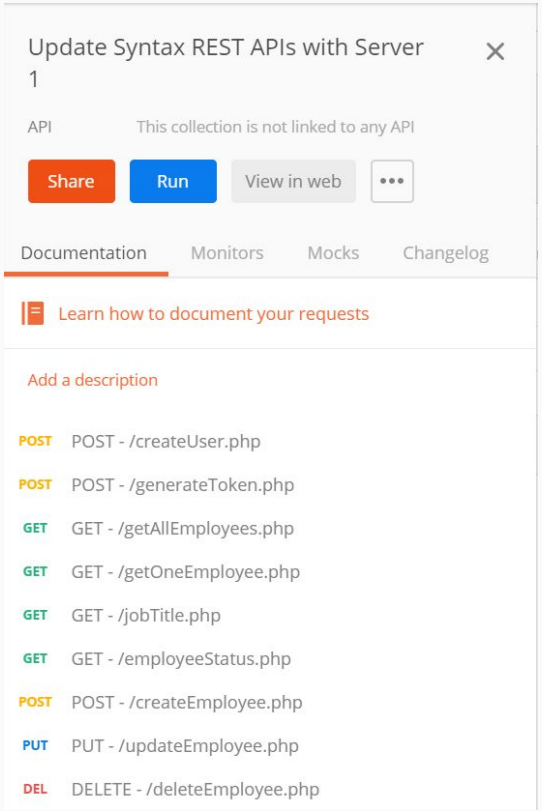
Globals — Edit

VARIABLE	INITIAL VALUE	CURRENT VALUE
firstName	Diego	Diego
lastName	juarez	juarez
createUser	/createUser.php	/createUser.php
generateToken	/generateToken.php	/generateToken.php
getAllEmployees	/getAllEmployees.php	/getAllEmployees.php
getOneEmployee	/getOneEmployee.php	/getOneEmployee.php
getAllJobTitles	/jobTitle.php	/jobTitle.php
getAllEmployeeStatus	/employeeStatus.php	/employeeStatus.php
createEmployee	/createEmployee.php	/createEmployee.php

Postman

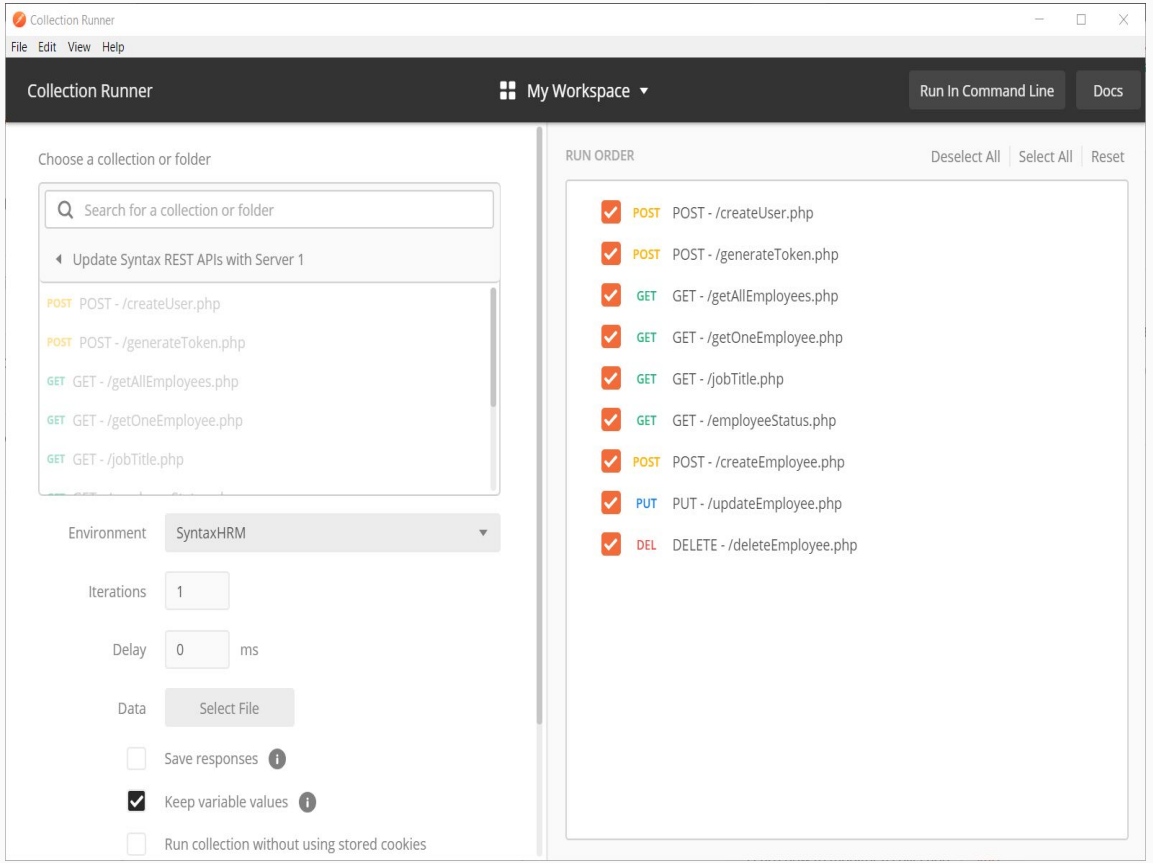
- Collections and Collection Runner





Postman

- Collections and Collection Runner



SYNTAX
TECHNOLOGIES

API

Class 3

Agenda

What is RestAssured?
Types of Parameters

RestAssured

REST Assured is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs.

REST Assured supports BDD syntax

REST Assured can be used easily in combination with existing unit testing frameworks, such as JUnit and TestNG.

RestAssured

REST Assured BDD Syntax:

given() → prepare request
when() → send the request
then() → validate request

```
given().
      when().
      get("https://got-quotes.herokuapp.com/quotes?
      char=tyrion").prettyPrint();
```

Parameters

```
@Test
public void pathParametersTest() {

    RestAssured.baseURI = "http://pure-ravine-92491.herokuapp.com/syntax";

    given().
        pathParam("studentId", 80).
    when().
        get("/api/getStudentProfile/{studentId}").
        prettyPrint();
}

@Test
public void queryParametersTest() {

    RestAssured.baseURI = "https://got-quotes.herokuapp.com";
    given().
        queryParam("char", "tyrion").
    when().
        get("/quotes").
        prettyPrint();
}
```

Hamcrest Matcher

Hamcrest is a library we use to perform assertions.

Hamcrest allows checking for conditions in our code using existing matchers classes.

To use Hamcrest matchers in JUnit we use the **assertThat** statement followed by one or several matchers.

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

public class Assertions {

    @Test
    public void getWithAssertion1() {

        RestAssured.baseURI="http://pure-ravine-92491.herokuapp.com/syntax";

        given().
            pathParam("studentId", "81").
        when().
            get("/api/getStudentProfile/{studentId}").
        then().assertThat().
            body("firstName", equalTo("Sandesh"));
    }
```

Hamcrest Matcher

`allOf` - matches if all matchers match (short circuits)

`anyOf` - matches if any matchers match (short circuits)

`not` - matches if the wrapped matcher doesn't match and vice

`equalTo` - test object equality using the equals method

`is` - decorator for equalTo to improve readability

`hasToString` - test Object.toString

`instanceOf`, `isCompatibleType` - test type

`notNullValue`, `nullValue` - test for null

`sameInstance` - test object identity

`hasEntry`, `hasKey`, `hasValue` - test a map contains an entry, key or value

`hasItem`, `hasItems` - test a collection contains elements

`hasItemInArray` - test an array contains an element

`closeTo` - test floating point values are close to a given value

`greaterThan`, `greaterThanOrEqualTo`, `lessThan`, `lessThanOrEqualTo`

`equalToIgnoringCase` - test string equality ignoring case

`equalToIgnoringWhiteSpace` - test string equality ignoring differences in runs of whitespace

`containsString`, `endsWith`, `startsWith` - test string matching

HTTP Response Codes

200 OK: - Code indicates that the request was successful.
201 Created:- Code indicates that request was successful and a resource was created. It is used to confirm success of a PUT or POST request.
400 Bad Request:- It happens especially with POST and PUT requests, when the data does not pass validation, or is in the wrong format.
404 Not Found:- response indicates that the required resource could not be found.
401 Unauthorized:- error indicates that you need to perform authentication before accessing the resource.
405 Method Not Allowed:- HTTP method used is not supported for this resource.
409 Conflict:- Conflict request to create the same resource twice.
500 Internal Server Error:- Occurs due to some error on Server side.

SYNTAX
TECHNOLOGIES

API

Class 4

Agenda

JSON Path
Headers
Serialization and DeSerialization

JSON Path

- JsonPath is an alternative to using XPath for easily getting values from a Object document.
- JsonPath is Another way to validate response body

```
@Test
public void jsonPath() {
    RestAssured.baseURI = "http://pure-ravine-92491.herokuapp.com/syntax";

    Response rsp = given().when().get("/api/getAllStudentProfiles");

    JsonPath jsPath = rsp.jsonPath();

    //to retrieve second name from the list of names from JSON
    String singleName = jsPath.get("firstName[1]");
    System.out.println(singleName);

    //to retrieve all first names from the list of names from JSON
    List<String> fName = jsPath.get("firstName");

    for (String string : fName) {
        System.out.println(string);
    }
}
```

Headers

HTTP Headers are an important part of the API request and response as they represent set of metadata associated with the API request and response.

Most of these headers are for management of **connections between client, server and proxies and do not require explicit validation** through testing.

Headers are mostly classified as **request headers and response headers**, know the major request and response headers.

Headers

Headers are a name, followed by :, followed by the value of the header.

Example:

- **Content-Type header** tells the server that the content of this message is JSON.

- **Accept header** tells the server that the client (application sending the message) will only accept response payloads represented in JSON.

For some headers we may need to set values or set assertions to ensure that they convey the right information and everything works fine in the API.

POST

```
@Test
public void postRequest() {

    Map<String, Object> map = new LinkedHashMap<String, Object>();

    map.put("id", 199.0);
    map.put("firstName", "Misha");
    map.put("lastName", "Galustyan");
    map.put("age", 27.0);
    map.put("batch", 5.0);
    map.put("batchStartDate", "2019-03-03");
    map.put("batchEndDate", "2020-03-01");
    map.put("course", "Selenium");
    map.put("attendanceNature", "Online");
    map.put("streetAddress", "123 Test");
    map.put("city", "Dream");
    map.put("state", "VA");
    map.put("zipCode", 12345);

    RestAssured.baseURI="http://pure-ravine-92491.herokuapp.com/syntax";

    given().
        header("Content-Type", "application/json").
        body(map).
        accept(ContentType.JSON).
    when().
        post("/api/createStudentProfile").
    then().
        assertThat().statusCode(201);
}
```

Put

```
@Test
public void putRequestTest() {

    Map<String, Object> map = new LinkedHashMap<String, Object>();

    map.put("id", 37);
    map.put("firstName", "Manya");
    map.put("lastName", "Galustyan");
    map.put("age", 27.0);
    map.put("batch", 4);
    map.put("batchStartDate", "2019-03-03");
    map.put("batchEndDate", "2020-03-01");
    map.put("course", "Selenium");
    map.put("attendanceNature", "Online");
    map.put("streetAddress", "123 Test");
    map.put("city", "Dream");
    map.put("state", "VA");
    map.put("zipCode", 12345);

    RestAssured.baseURI="http://pure-ravine-92491.herokuapp.com/syntax";

    given().
        header("Content-Type", "application/json").
        body(map).
        accept(ContentType.JSON).
    when().
        put("/api/updateStudentProfile").
    then().
        assertThat().statusCode(200);
}
```

Delete

```
@Test
public void deleteStudent() {

    RestAssured.baseURI="http://pure-ravine-92491.herokuapp.com/syntax";
    Response rsp=
    given().
        pathParam("studentId", "34").
    when().
        delete("/api/deleteStudentProfile/{studentId}");
    rsp.then().assertThat().statusCode(200);
}
```

Payload

A Payload is the body of the HTTP request or response.

Typically when working with an HTTP API we will send and receive JSON or XML payloads.

When we send request we have request payload

When we get response we have response payload