Contents

***1. Core Java***

***2. WebServices and Why WS***

***3. REST APIs and how to Automate***

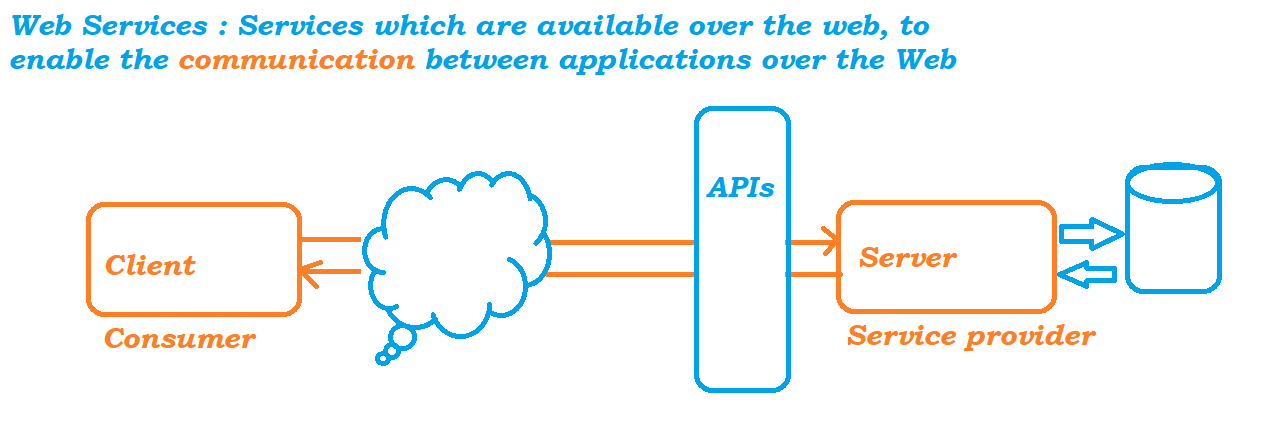
***4. Maven - Build Tool***

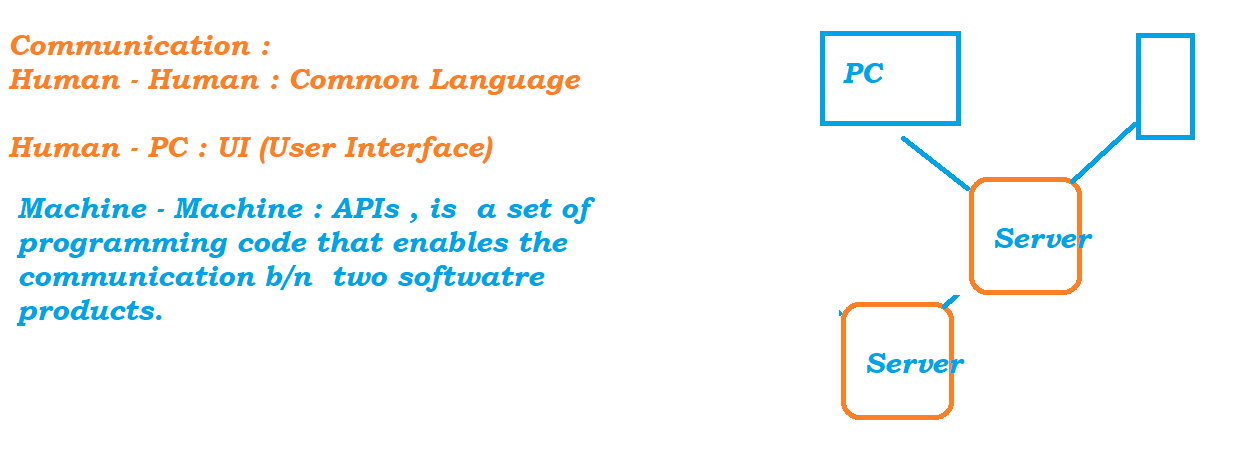
***5. TestNG - Framework***

***6. APIs - Github, Trello, Twitter***

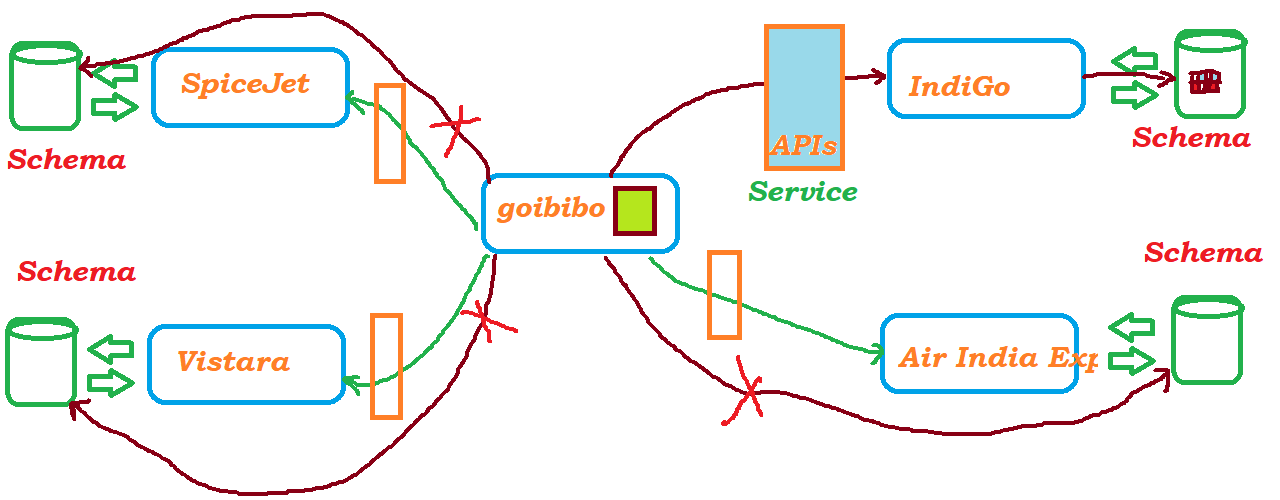
***7. POSTMAN***

***8. Rest Assured Framework***

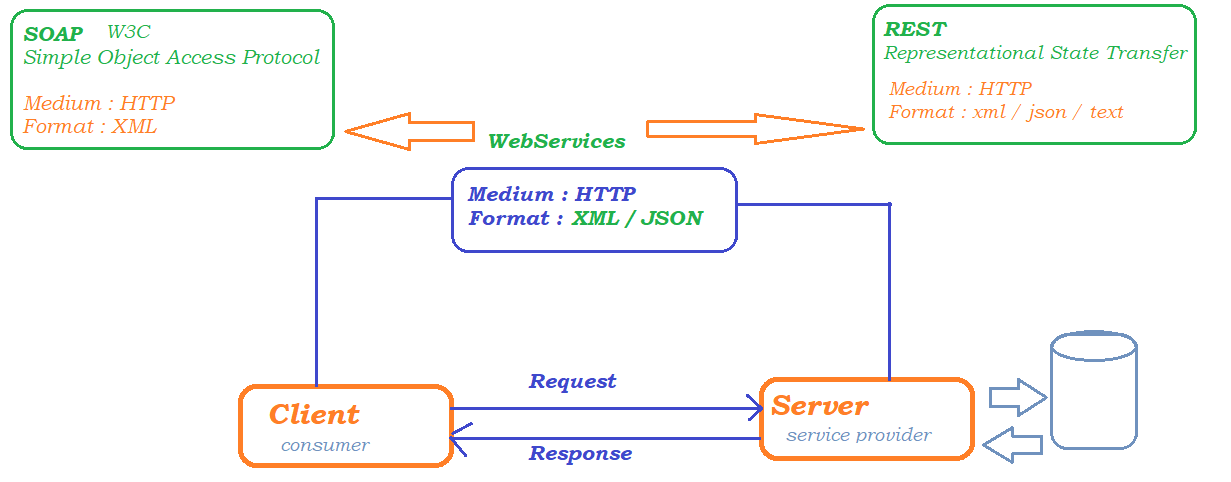
******

******

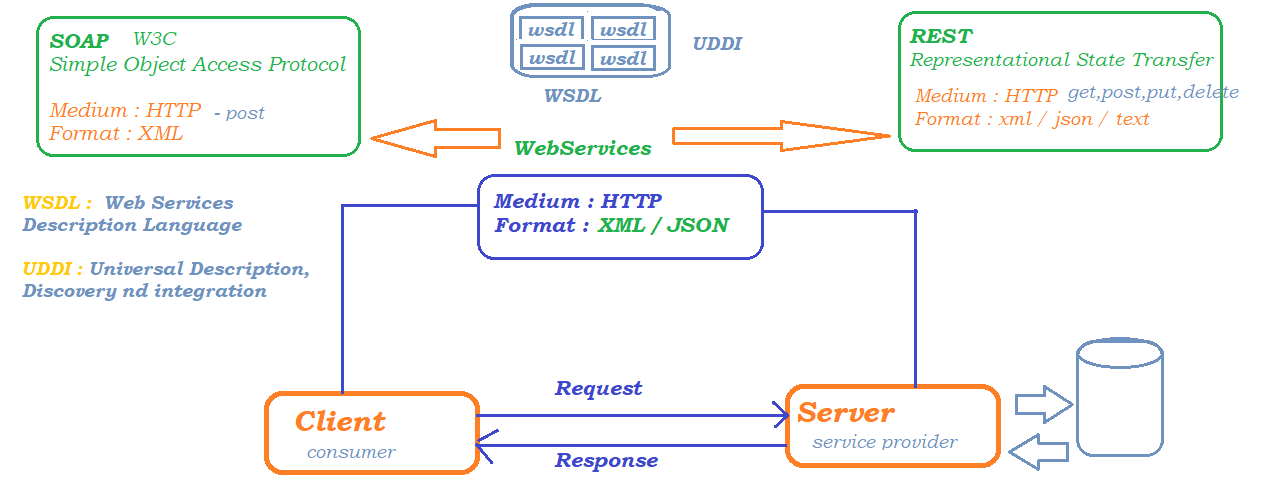
How WS Works ? Need of API in Software

******

Types of Web Services :

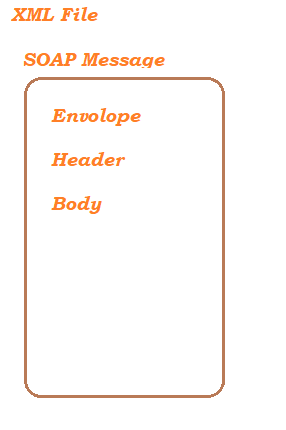


## Detailed Diagram on SOAP



## Sample SOAP Message :

<http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?op=CapitalCity>



RESTful Web Service

*REST : WebService that communicates or Exchange the messages b/n 2 Application using REST Architecture*

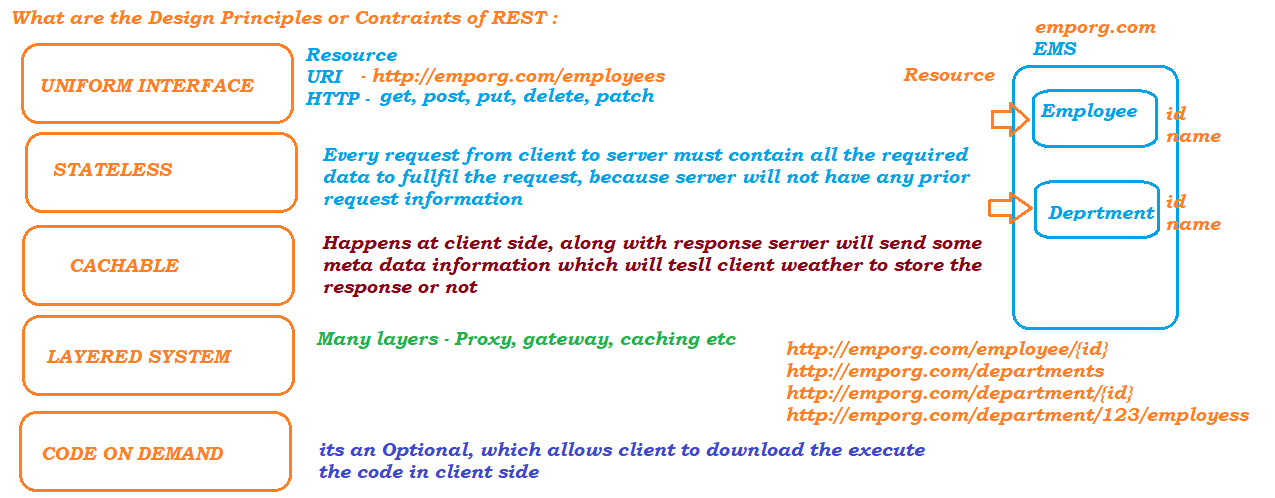
*OR Rest Principles then we call the Service as RESTful WebServies*

*- REST is nothing but Representational State Transfer its an Architecture style*

*- REST is not a protocol, there are no standard Rules, or no central body to define / control the rules*

*- REST is just a design principle*

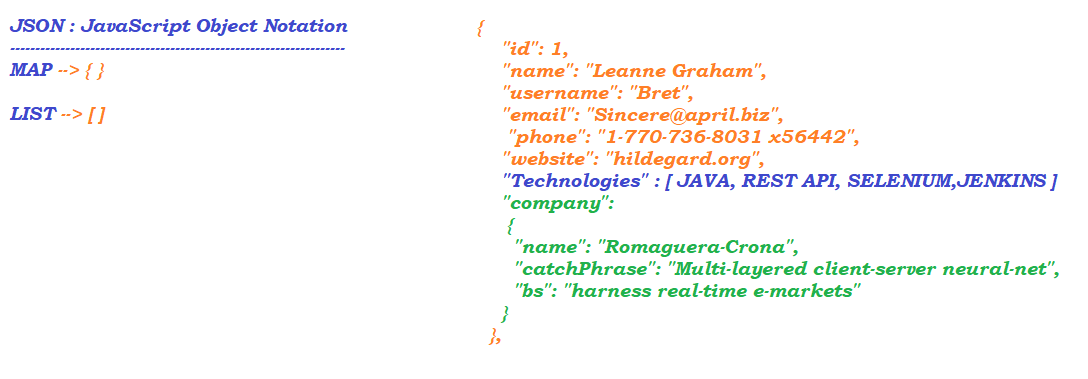
# What are the Design Principles or Constraints of REST :



Example of REST API :

<https://jsonplaceholder.typicode.com/>

JSON



PostMan

Its a third party software used to Test the REST Api’s. POSTMAN can be download and installed on a machine or it can be used as a browser plugin

STEPS to use PostMan

1. Download and install <https://www.postman.com/downloads/>

2. Double click on the postman icon

3. Once the application is launched you have to register with POSTMAN, you have to signup (you can use your gmail account to login)

# RestAPI Elements :

* **HTTP Request :**

Its a package of information requested by client to server

* + **Request Line -** specifies the method token ( GET, PUT, POST, DELETE, PATCH..)
  + **Request Header – *Optional*,** is used to store additional information while sending a request
    - **EX** : trying to access face book home page – **0 headers**
    - trying to perform some operation on facebook ,like send request or send message to friend – **additional headers**
  + **Request Body – Optional,** itis used to send additional information along with HTTP Request
  + **Authorization – Optional,** if API needs **Authorization** to fulfill the request then we have to pass the **token** or **key** provided by the Application developer
  + **Pre-Request Script, Optional,** Before sending a request if we want to execute some piece of code
* **HTTP Response :**

Its a package of information sent by server for the request made by client

* **Response Line - GET**
* **Response Headers –** contains meta data like, date, server who processed the request etc
* **Optional Body –** Optional, based on the API
* **Status Code –** APIs are validated through status code
* **Resource** – Any information stored in server and requested by client
* **Resource Identifier –** to identify the resource uniquely , its a complete URL in WebService we call it as URI
* **Representation –** Actual data returned by the server – HTML, XML, JSON
* **Representation Metadata –** Extra information sent by the server for every request made, date, server who processed the request etc
* **URI ::**
  + **scheme://domain/pathparameter**
  + **scheme://domain/queryparameter**

**scheme : http / https/ ftp ...**

**domain: name where the application is hosted or accessed**

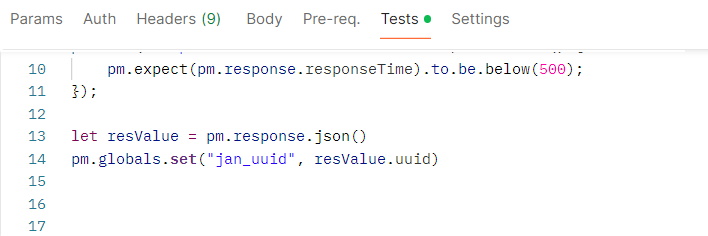
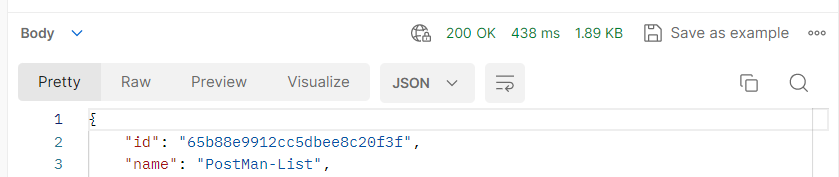
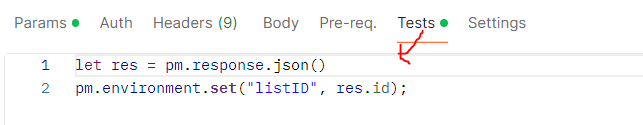
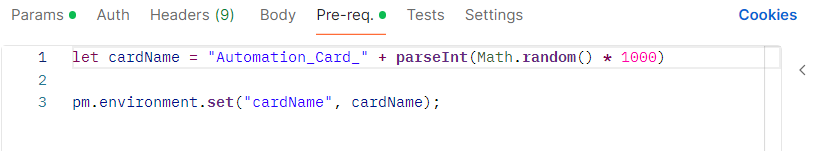
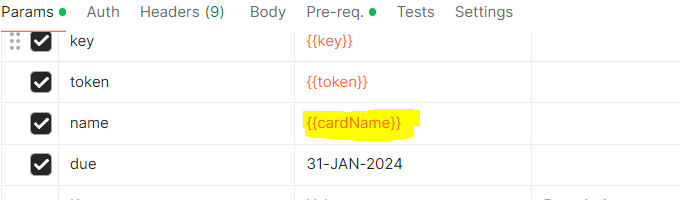
**pathparameter:** separated by / it is the actual location of the resource

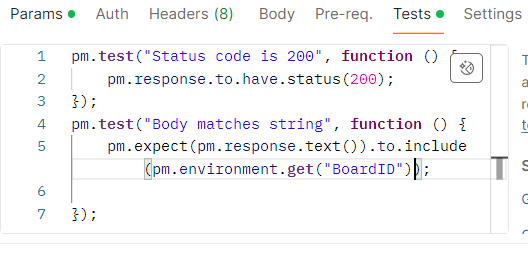
**queryparameter:** separated by ? it is the data sent to server

# VARIABLES IN POSTMAN

* **GLOBAL VARIABLES**

Variables which are common to all the projects are called global variables

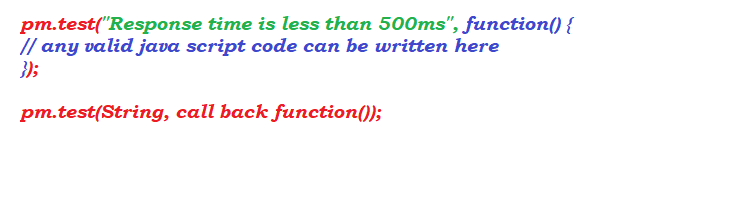
* **SAVING OUTPUT OF THE API TO GLOBAL VARIABLES**
  + pm.globals.set(“key”,”value”)
  + ****
  + ****
* **ENVIRONMENT VARIABLES**
  + Variables which are specific to project can be saved as Environment Variables
* **SAVING OUTPUT OF THE API TO ENVIRONMENT VARIABLES**
  + pm.environment.set(“key”,”value”)
  + ****
  + ****
* **PRE-REQUEST SCRIPT IN POSTMAN**
  + This is the block which is executed before sending the API request
  + ****
  + ****
* **WRING TESTS IN POSTMAN**

****

# Using output of one API in another API Call

In a parameter Value use the global variable value in a double flower bracket EX : {{Global Data}}

**Syntax of TEST in postMan ::**



HTTP Methods :

1. **GET** – is used to retrieve the data from the server, body is optional and we usually get 200 as a status code on successful execution of API and we may get an error code 401, 404
2. **POST** – is used to create OR insert the record in the application or DB,usually we will get 201 on successful execution of API and we may get an error code 401, 404,500 …
3. **PUT** – used to update the record , only one record will be updated at a time
4. **DELETE** – used to delete the record
5. **PATCH** - used to update the record, Only partial information is sufficient

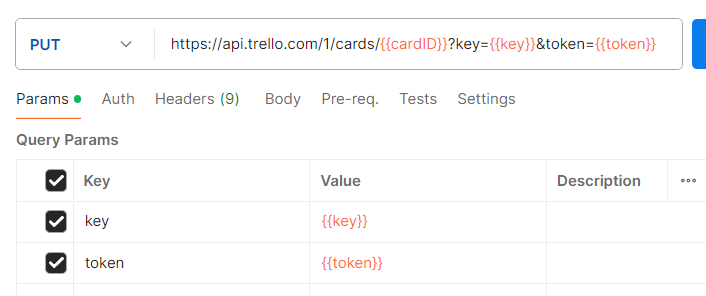
Trello API

* Register to Trello : <https://trello.com/>
* Create a sample Board -> List -> Card
* Go to API Documentation <https://developer.atlassian.com/cloud/trello/rest/api-group-actions/#api-group-actions>
* To use the APIs from trello we need to get authentication token from the Service provider
* Steps to get ID and Token
  + Click on Guides on top centre of <https://developer.atlassian.com/cloud/trello/>
  + Scroll down and click on trello REST APIs
  + click on API Introduction
  + click on new Pop-ups
  + Enter all the mandatory details
  + Click on API Key from the left panel,
  + Click on generate new API Key
  + Copy the newly created key and secrete key to notepad
  + On the same page click on token
  + Click on Allow and copy the token to notepad
* 

APIs:

* get Board
* create Board
  + saving the id of the Board
* create List
  + save the list id
* create Card
* create Card with dynamic data - Pre-request script
* delete Board

# Update Card:



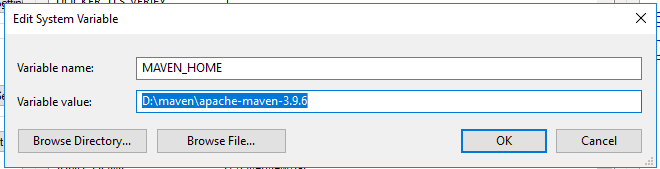
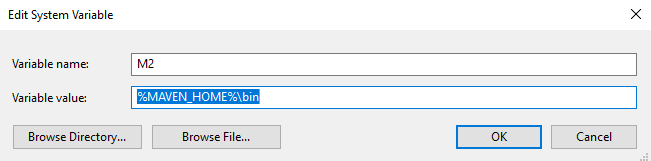
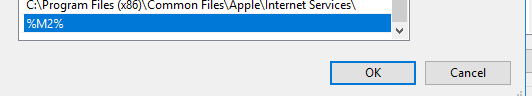
GIT-HUB

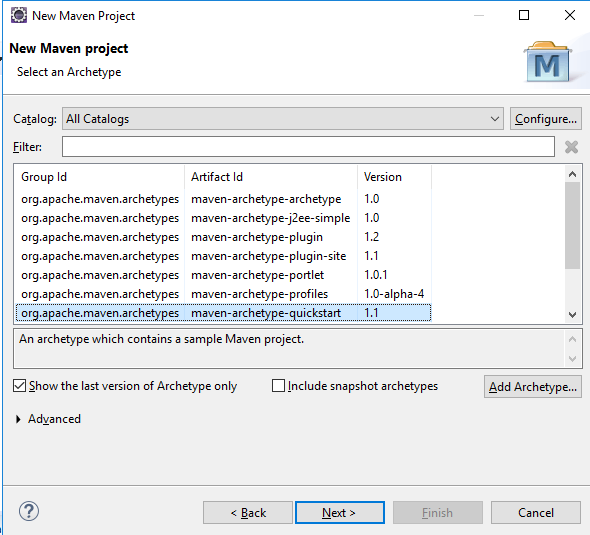
1. Create a github Account :: <https://github.com/signup>
2. go to google and search for github developer api
3. <https://docs.github.com/en/rest>
4. Get the Authentication token OR Bearer token
   1. Login to github.com with your credentials
   2. click on the user icon on top right corner
   3. click on settings from the list
   4. scroll down and click on developer settings from the left panel
   5. click on personal access token-> Tokens (classic)
   6. click on Generate new token and confirm with your password
   7. save the token generated

Rest API Automation Using Rest Assured

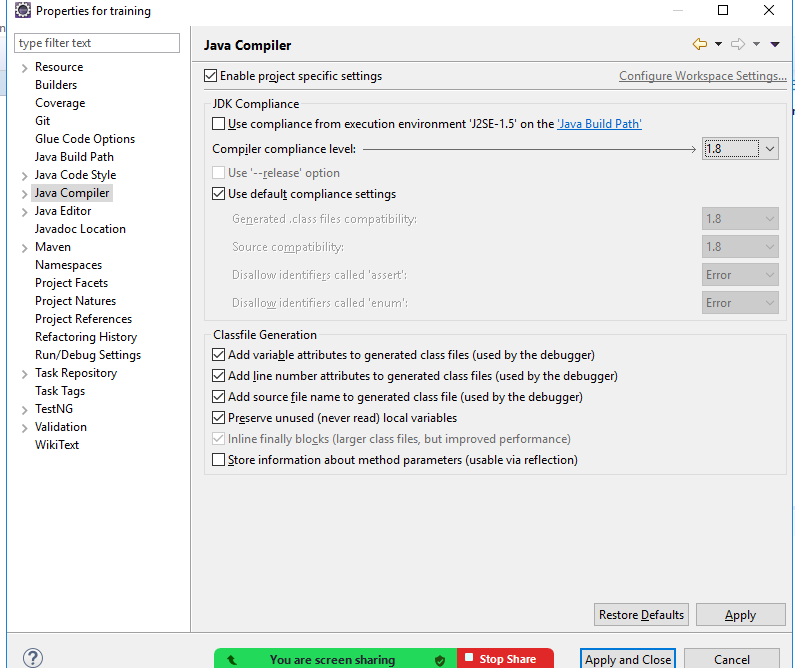
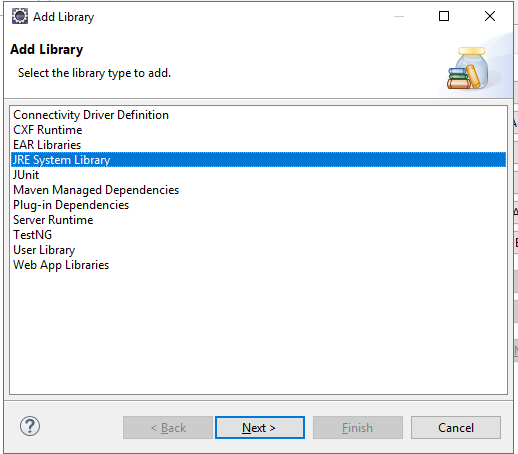
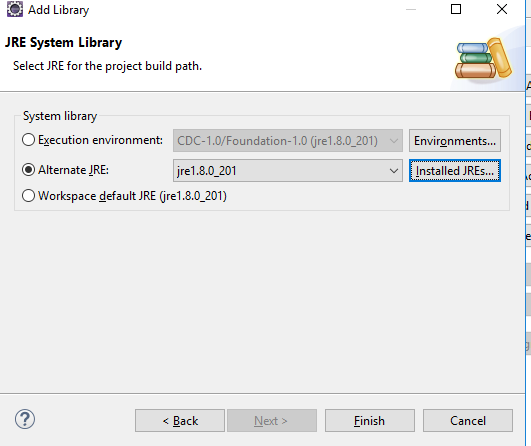
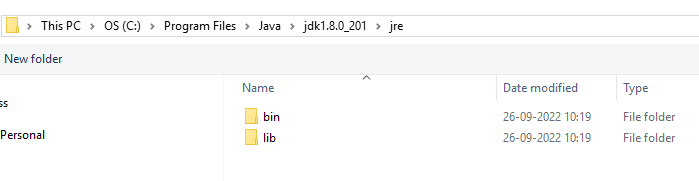
# Rest Assured is a java library built on top of HTTP Builder, all the HTTP Methods like, get, post, put, patch, delete are available inside REST Assured library. We cn easily automate the Request and Respose

Setup of Rest assured :

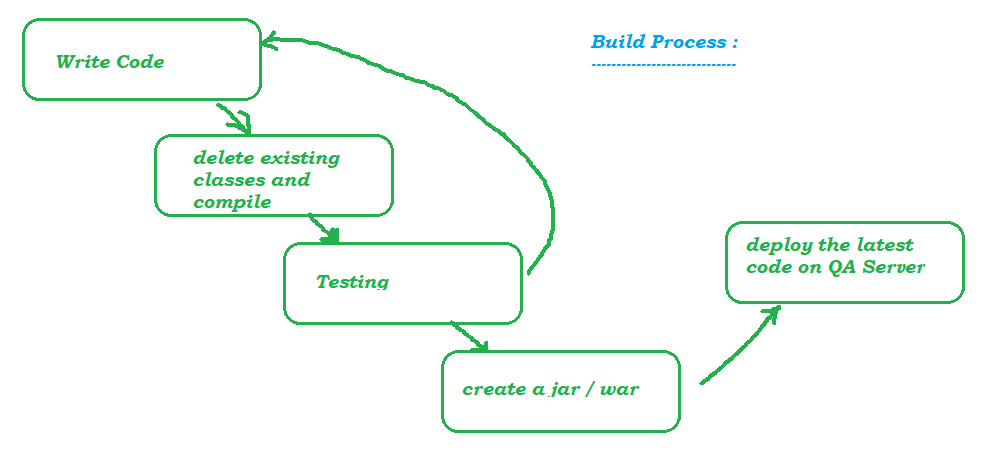
1. Java – JDK1.8 or above
2. Download Eclipse - <https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-12/R/eclipse-java-2023-12-R-win32-x86_64.zip>
3. Download Maven ( Build Automation Tool )- <https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.zip>
4. Configure Maven
   1. Right click on this PC
   2. go to properties
   3. click on Advanced System settings
   4. click on Environment Variables
   5. click on New Under system Variable
   6. 
   7. click on New Under system Variable
   8. 
   9. Select path variable and Click on edit and add %M2%
   10. ****
   11. Open command prompt and execute mvn­ –verison
5. Create a Project in eclipse and configure s REST Assured Project
   1. Open Eclipse
   2. Select workspace other than c driver
   3. Eclipse -> File -> New -> Maven project
   4. Select groupID and artifactID to be used and click on next

******

* 1. provide groupid and artifactid for your project
  2. click on Finish

1. Update JDK and JRE to the available versions in the machine
   1. Right click on your project and go to properties
   2. Click on java compiler from the left panel******
   3. click on java build path -> libraries
   4. select the wrong JRE present in your project and click on remove
   5. From the same window click on add library
   6. ******
   7. select Alternate JRE as shown below
   8. ******
   9. Select the jre from jdk location
   10. ****
   11. Click on Apply and close -> Apply -> Apply and close

Maven :

**

update pom.xml for rest dependencies

- <https://rest-assured.io/>

First Test on Rest Assured

a

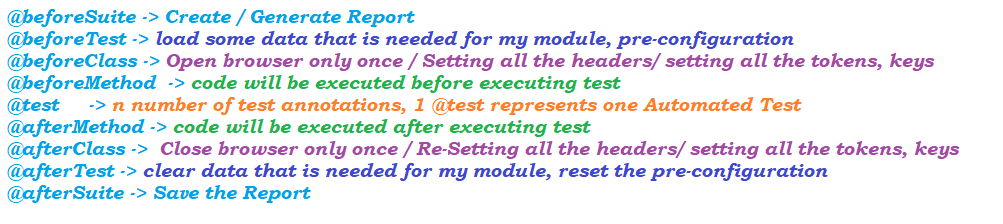
TestNG

* TestNG is a framework which can be used to execute Regression Tests
* To execute the tests we will be using TestNG Runtime Environment
* **Annotations** to identify the methods

Configuring TestNG Library

* In Eclipse Help -> Eclipse Market place -> Search for testing and install the plugin
* RC on the project -> properties -> Libraries -> Add library -> TestNG
* go to mvnrepository.com -> search for testing and update TestNG Dependencies
* Write a TestNG Class

annotations in testng

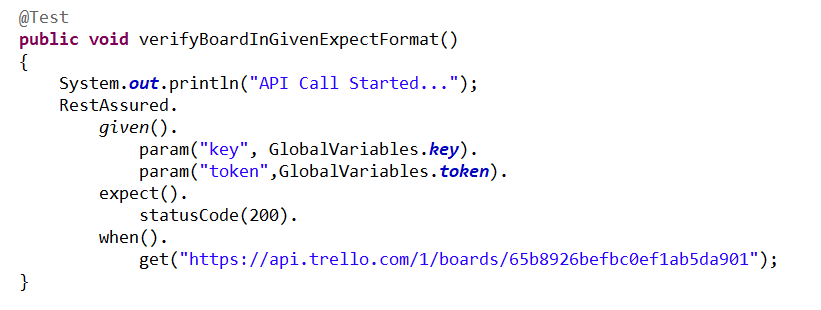
****

Rest Assured Test Format

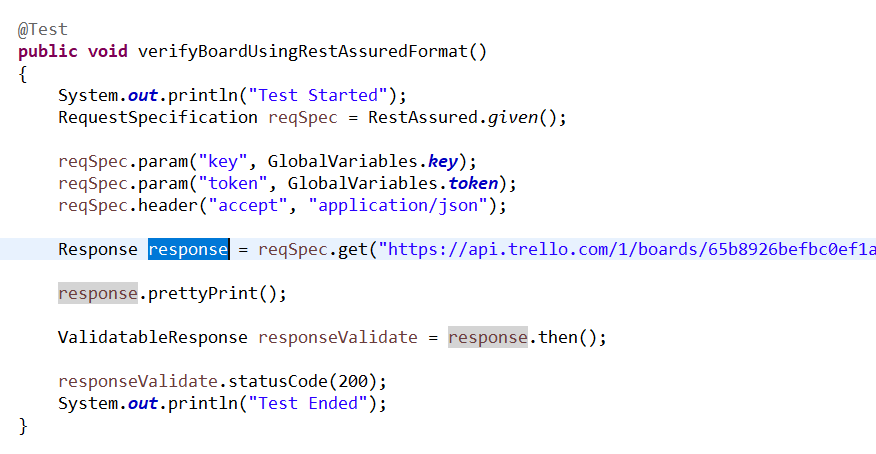
1. Given When Then (BDD Format) Format



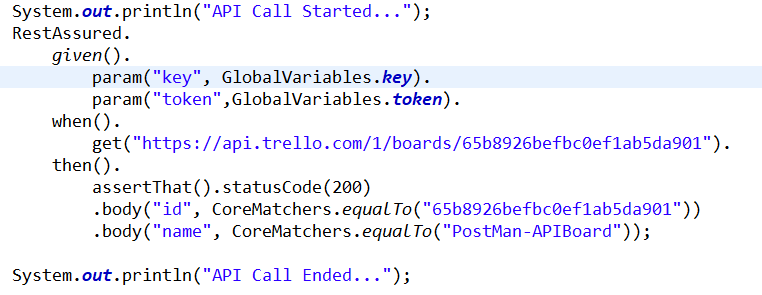
1. Given expect When Format



1. RestAssured Class Format

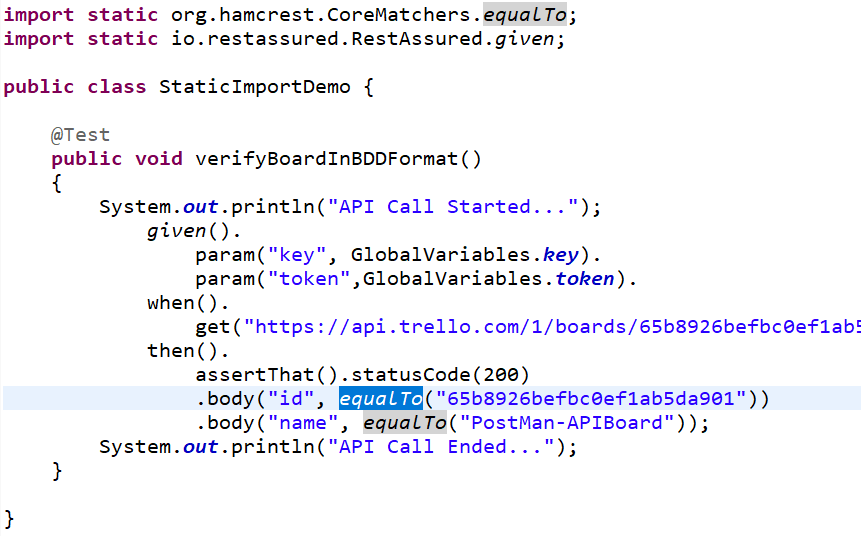


verification of test output using hamcrest packge :



static imports in java :

Using static imports we can import the static methods available in the class so that we can use them directly in the class



Global Varibles of rest OR reusability with rest assured constatnts



Automating post request

In case POST Request we can pass 3 types of parameter

1. path parameter

2. query parameter

3. body parameter

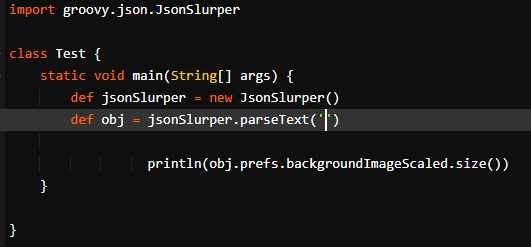
Passing Dynamic Data using FAKER-API

* search for java faker github
* <https://github.com/DiUS/java-faker>
* Update the pom.xml file with the faker dependency
* Refer the git hub documentation on how to use Faker API

Validations in REST

* Jayway Json Path
  + Java syntax
  + first update pom.xml with the jayway jsonpath dependency, <https://github.com/json-path/JsonPath>
  + Use json editor to understand the structure of Complex Json, <https://codebeautify.org/online-json-editor>
  + Use JsonPath evaluator to verify the expression - <https://jsonpath.com/>
* Sample Jayway Json Expressions:
  + Example json is getboard with Background

|  |  |
| --- | --- |
| * + id | * + $.id |
| * + print all urls in BGImageScaled | * + $.prefs.backgroundImageScaled[\*].url |
| * + Get first BGImge | * + $.prefs.backgroundImageScaled[0] |
| * + short URL | * + $.shortUrl |
| * + All URLs of JSON | * + $..url |
| * + Enabled Values | * + $.prefs.switcherViews[\*].enabled |
| * + width > 1000 | * + $.prefs.backgroundImageScaled.[?(@.width>1000)].width |
| * + url for width > 1000 | * + $.prefs.backgroundImageScaled.[?(@.width>1000)].url |
| * + names of false items | * + $.prefs.switcherViews[?(@.enabled==false)].viewType |

* JsonPath – (builtin)
  + Groovy (gpath expression we need to write)
  + <https://docs.groovy-lang.org/latest/html/groovy-jdk/java/util/Collection.html>
  + <https://www.tutorialspoint.com/execute_groovy_online.php>
  + 
  + NOTE : in the above image within the single quote paste the json output with out whitespaces

|  |  |
| --- | --- |
| * + id | * + id |
| * + print all urls in BGImageScaled | * + prefs.backgroundImageScaled[index].url |
| * + Get first BGImge | * + prefs.backgroundImageScaled[0] |
| * + short URL | * + shortUrl |
| * + All URLs of JSON | * + write 2 separate expressions |
| * + Enabled Values | * + prefs.switcherViews[inext].enabled |
| * + width > 1000 | * + prefs.backgroundImageScaled.findAll { it.width > 1000 } |
| * + url for width > 1000 | * + prefs.backgroundImageScaled.findAll { it.width > 1000 }.url |
| * + names of false items | * + $.prefs.switcherViews[?(@.enabled==false)].viewType |
| * + first element with width>1000 | * + prefs.backgroundImageScaled.find { it.width > 1000 } |

* Validate json response using VALIDATABLERESPONSE CLASS