API Testing Course

What all will be covered?

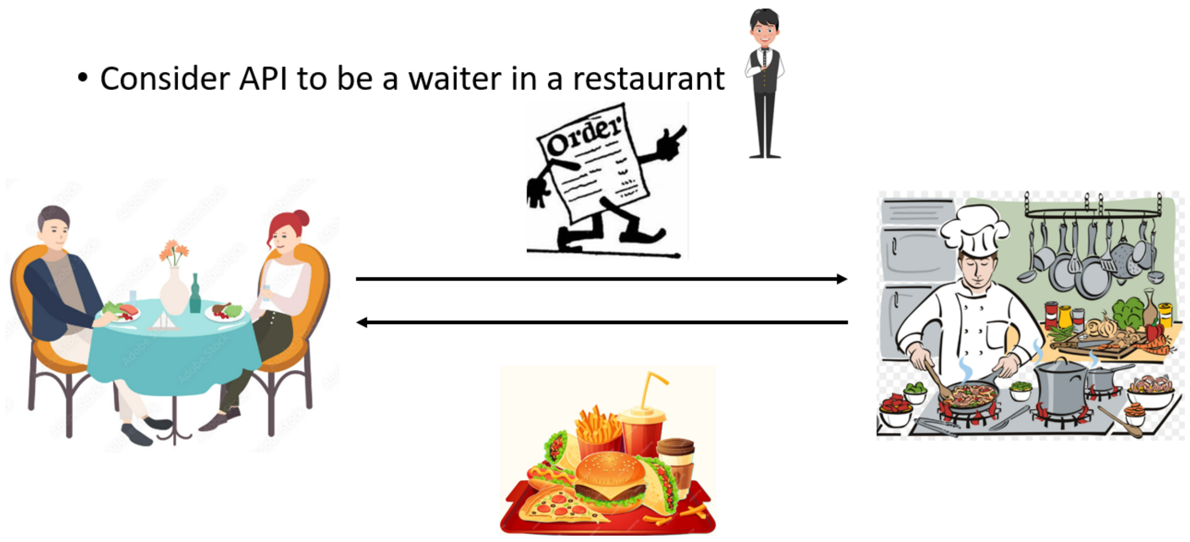
- All about API's & API Testing
- Postman
- REST Assured
- TestNG
- Maven
- Jenkins
- Git
- JMeter

API

- Application Programming Interface
- Means by which two computers/programs/applications interact with each other
- Basis of how data transfer takes place between two applications

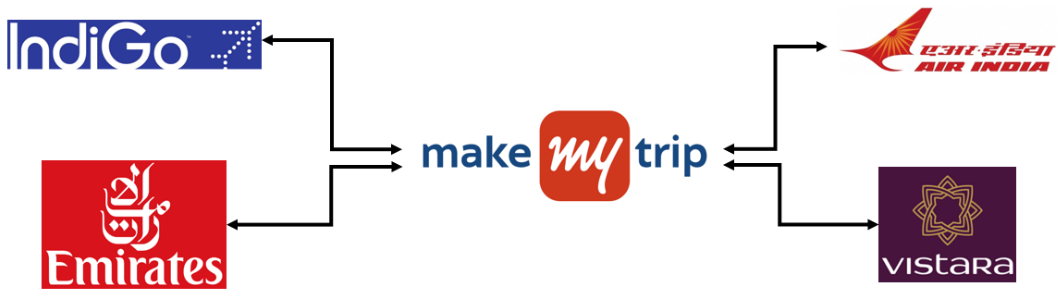
API - Personified Example

- Consider API to be a waiter in a restaurant



API – Real Time Example

- We will find this PayPal Checkout option in most of the ecommerce sites
- Clicking on it redirects to PayPal where we can complete the transaction
- Card details remains with PayPal
- Request will have the Payment amount etc and response will have the payment success message along with Transaction ID

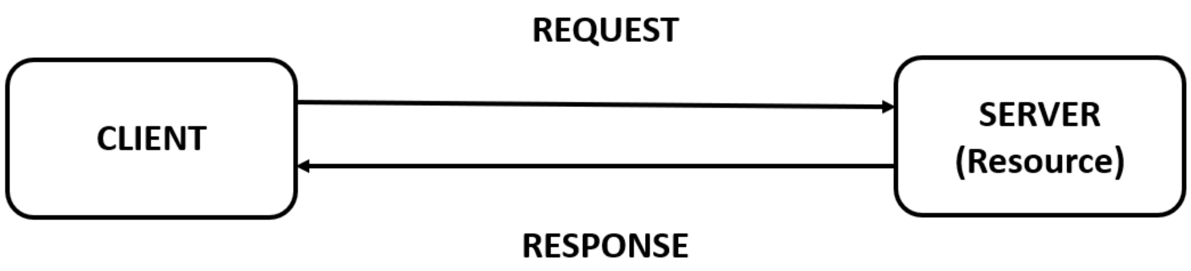


Importance of API

- It's how data moves between applications
- Allows capabilities of one computer/application/program to be used by the other
- Saves tremendous developer time. Ex:- Integrating Google Maps, Google pay etc.
- No need for the developer to understand the underlying code
- Improves CX-Customer Experience which is crucial for business success
- In 2015, Fortune magazine wrote:

"How a business wins or loses is increasingly dependent on how well they connect to external third-party apps, devices and services."

Components of a API Flow



Components of a API Flow

- **Client:-**

Entity that uses an API to access or modify information (Resource) present in another Computer/Program/Application

- **Resource:-**

Information which is being accessed or modified by the Client

- **Server:-**

Computer/Program/Application hosting the resource

Components of a API Flow – Request

- **Endpoint:-**

The API Request is directed to this URL. Consider this to be the address of the resource hosted in the Server

- **Parameters/Params:-**

Variables that are passed along with the Endpoint URL

- **HTTP Methods:-**

Operation that the Client would like to perform on the resource in the server

- **Request Headers:-**

Key-Value pairs that provide additional info about the request. Ex:- Content-Type provides info on the format of data in the Request Body

- **Request Body:-**

Contains actual data required to create, update or delete a resource in the Server

- **Authentication/Authorization**

Authentication	Authorization
Process of verifying who the user is (to identify who you are)	Process of verifying what the user has access to (to identify what you have access to)
Usually done before authorization	Usually done after successful authentication
Example: Employees in a company are required to authenticate themselves before entering the company premises, ODC and logging in to their systems	Example: After an employee successfully authenticates, the system determines what information the employees are allowed to access

Components of a API Flow – Request - Authorization Supported by Postman

Authorization	Meaning
No Auth	Request which does not require any Authorization
API Key	Key-Value pair passed either through Request Header or Query Params
Basic Auth	Username and Password – Will be passed in the Request Header as a Base64 encoded string

API vs Web Service

API	Web Service
Allows applications to connect over a standardized manner **without need for a Network**	Type of API which must be **accessed over a Network**
Supports XML & JSON	Supports only XML

All Web Services are APIs but all API's are not Web Services

Components of a API Flow – Request – HTTP Methods

1. GET – Retrieve/Get data from the Server
2. POST – Create a new resource in the Server
3. PUT – **Replace** an existing resource in a Server
4. PATCH – **Update** existing resource in a Server
5. DELETE – Delete/Remove a resource from the Server

Components of a API Flow – Request – HTTP Methods – Examples

Table Name – Students List

Rows – Records
Columns – Fields

Serial Number	Roll Number	Name	Phone Number
1	001	Arjun	1234567890
2	002	Ramesh	0987654321
3	003	Alex	7447657657

1. GET – Retrieve/Get details of student - Arjun
2. POST – Create a new student - Alex
3. PUT – Replace Arjun's record with Rajesh's details
4. PATCH – Update Alex's Phone Number
5. DELETE – Delete/Remove Alex from the table

Components of a API Flow – Response

- **Response Headers:**

Similar Key-Value pairs that provide additional info about the response

- **Response Body:**

Contains data returned by the Server

- **HTTP Status Codes:**

Indicates whether a specific HTTP request has been successfully completed

Components of a API Flow – Response – HTTP Status Codes

HTTP Status Codes	Meaning
1XX – (100 to 199)	Informational Responses
2XX – (200 to 299)	Successful Responses
3XX – (300 to 399)	Re-directional Messages
4XX – (400 to 499)	Client Side Errors
5XX – (500 to 599)	Server Side Errors

HTTP Status Codes	Meaning
200	Success/OK
201	Created
202	Accepted
301	Permanent Redirect
302	Temporary Redirect
400	Bad Request
404	Not Found
500	Internal Server Error

Types of API's – Based on Availability, Declaration and Invocation

- **Open APIs**

Publicly available APIs that anyone can use to access a company's data

- **Internal APIs**

APIs used within a company/organization to communicate information between internal apps

- **Partner APIs**

APIs designed specifically for third-party developers/partners, and are more limited in access

Types of API's – Based on Protocols

	SOAP	REST
Stands For	Simple Object Access Protocol	REpresentational State Transfer
Design	Exposes Operation	Exposes Data
Transport Protocol	Independent. Can work on any Transport Protocol – HTTP, SMTP etc.	Only with HTTP
Data Format	Only XML	XML, JSON, Plain Text, HTML
Performance	Messages are larger, makes communication slower	Smaller messages and caching support. So faster
Scalability	Difficult to scale – The server **maintains state** by storing all previous messages exchanged with a client	Easier to scale - It's **stateless**, so every message is processed independently of previous messages
Use Case	Useful in legacy applications and private APIs	Useful in modern applications and public APIs

XML

- Stands for – Extensible Markup Language
- Designed to Store and Transmit data
- Makes use of Tags which aren't pre-defined

```
<books>
        <book>
                <title>One Up Wall Street</title>
                <author>Peter Lynch</author>
        </book>
        <book>
                <title> Intelligent Investor</title>
                <author> Benjamin Graham</author>
        </book>
</books>
```

JSON

- Stands for – JavaScript Object Notation
- Lightweight format to store and transmit data
- Uses Key-Value pairs separated by commas
- Data Structures:
    - Object – Set of Key-Value pairs
    - Array – List of values
- Data Types:
    - String - {"name":“Arjun"}
    - Number - {"age":28}
    - Object
    - Array
    - Boolean – {“student”:true}
    - Null

JSON – Objects

- Can hold multiple Key-Value pairs
- Key – Will be a String & Value – Can be of any previously mentioned type
- Keys and Values are separated by Colons
- Each Key-Value pair is separated by a Comma
- Uses curly braces – **{}**

```
{
   “student": {
     "name":      “Arjun",
     “roll no":    001,
   }
}
```

JSON – Arrays

- Can hold multiple Objects/Strings/Numbers etc
- Uses square brackets – **[]**

    - **Array of Strings** – [“Arjun”,”Arun”,”Amit”]
    - **Array of Numbers** – [01,02,03]
    - **Array of Booleans** – [true,false,false]
    - **Array of Objects** –

```
{
"employees":[
              {"name":"Ram", "email":"ram@gmail.com", "age":23},
              {"name":"Shyam", "email":"shyam23@gmail.com", "age":28},
              {"name":"John", "email":"john@gmail.com", "age":33},
              {"name":"Bob", "email":"bob32@gmail.com", "age":41}
            ]
}
```

JSON vs XML

JSON	XML
JavaScript Object Notation	Extensible Markup Language
JSON supports strings, numbers, Booleans, null, array, objects	XML data is in a string format
JSON can use arrays to represent the data	XML does not contain the concept of arrays
JSON has no tags	XML data is represented in tags, i.e., start tag and end tag
File size is smaller as compared to XML	File size is larger
It is less secure than XML	It is more secure than JSON

Message in JSON vs XML

JSON

```
{
"employees":[
{"name":"Ram", "email":"ram@gmail.com"},
{"name":"Bob", "email":"bob32@gmail.com"}
            ]
}
```

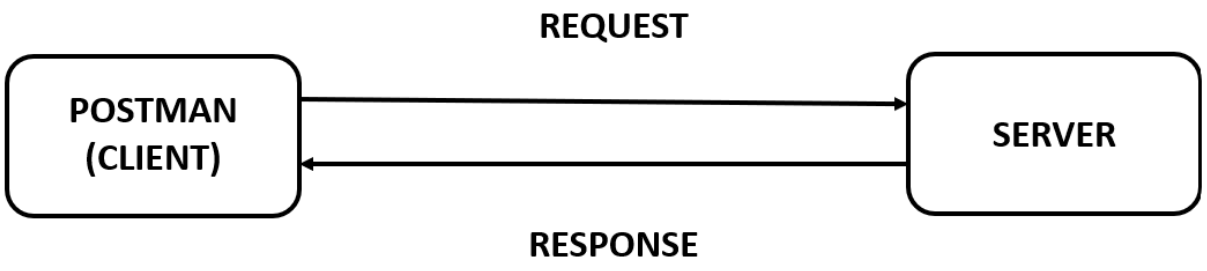
XML

```
<employees>
        <employee>
                <name>Ram</name>
                <email>ram@gmail.com</email>
        </employee>
        <employee>
                <name>Bob</name>
                <email>bob32@gmail.com</email>
        </employee>
</employees>
```

How to decide between SOAP and REST

- Totally based on the technical needs
- Important deciding factor for SOAP:
    - **Stateful operations:** Performing repetitive, chained tasks such as the financial industry requires means that you may need to retain certain client data within the server for future use. By default, REST is stateless. RESTful apps will not save any previous transactions, but SOAP supports stateful operations.

POSTMAN



POSTMAN – Installation

- https://www.postman.com/downloads/
- Install Windows-64 bit
- Use your personal Mail ID while creating an account with Postman

Postman – Workspace

- Helps to organize our API Work
- Helps to collaborate effectively across the team and organization
- Hierarchy of storing requests:
    - Workspace
        - Collection
            - Folder
                - Requests

Postman - Workspace

- Create
- Specify who can access
- Workspace Settings

Postman – Collections

- *Postman Collections* are a group of saved requests
    - Auth
    - Tests/Pre-Request Scripts
    - Variables
    - Runs & Run
    - Share
    - Delete

Postman – Folders

- Folders are used to further organize the requests
    - Auth
    - Tests
    - Pre-Request Scripts
    - Run
    - Delete

Postman – Explore Various Requests

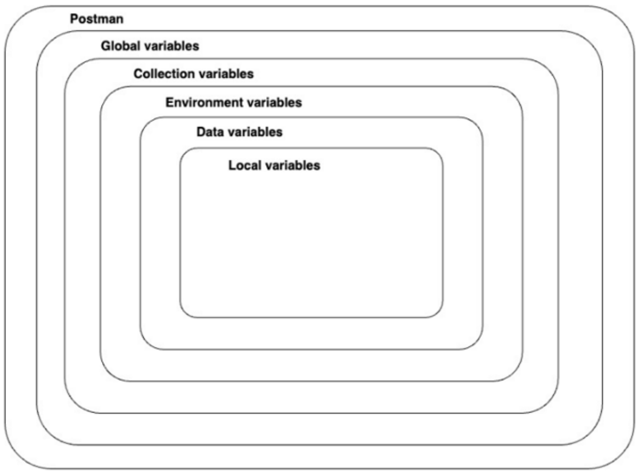
- GET - https://www.google.com/search?q=lion&tbm=isch
- GET, POST, PUT, DELETE API's from reqres.in
- Unsuccessful Response
- Authorization – Request & Folder Levels:
    - Bearer Token example:
    - GET: https://gorest.co.in/public/v2/users
    - Bearer Token: 2fb2718ea99979c5e9ed0f93e50e92adefc2cffe886c1d80cc2485bc11170dfe

Postman – Variables

- Enables you to store and reuse values
- Helps us to work efficiently and set up dynamic workflows

Variables - Scopes: Postman supports variables at various scopes to be used as per our needs.

- **Global Variables** – Broadest and available throughout the Workspace
- **Collection Variables** – Available throughout the requests in the Collection and also is Environment independent
- **Environment Variables** – Enables you to scope your work to different environments. Ex – Dev, Testing, Production
- **Data Variables** – Comes from external CSV and JSON files. To define data sets while doing data driven testing through Collection Runner or Newman
- **Local Variables** – Temporary and are scoped to a single request or collection and are no longer available once the run is complete

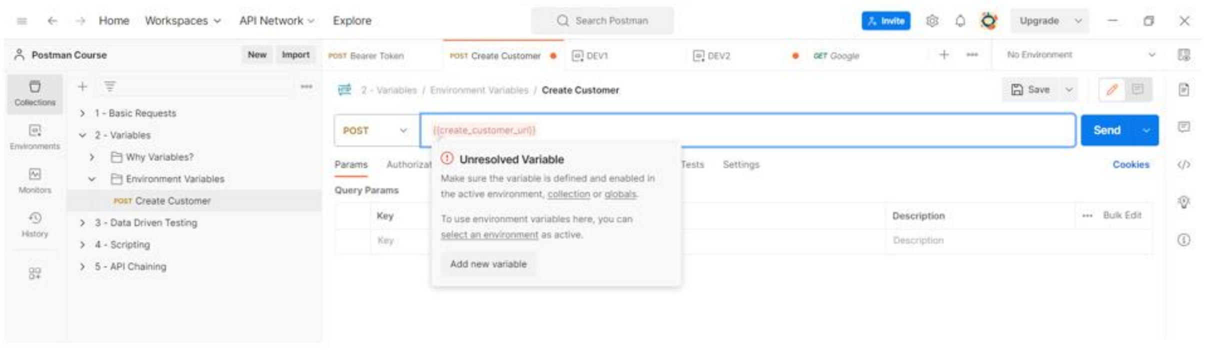


- If a variable with the same name is declared in two different scopes, the value stored in the variable with narrowest scope will be used
- For example, if there is a global variable named **username** and a local variable named **username**, the local value will be used when the request runs

Environment Variables

Assume, there's an API to create a customer and to test the API through different Environments, we usually will only have to change the Endpoint URL. In this case, Environment variable comes handy

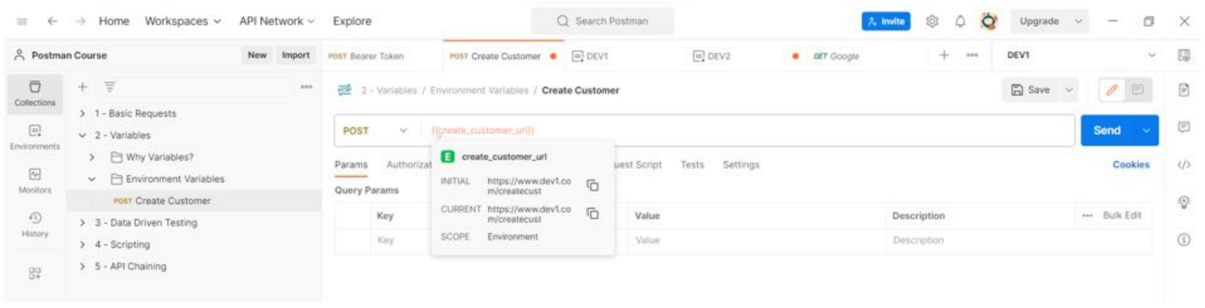
Postman – Unresolved Variable


POST {{create_customer_url}}

Unresolved Variable

Make sure the variable is defined and enabled in the active environment, collection or globals.

To use environment variables here, you can select an environment as active.


POST {{create_customer_url}}

create_customer_url

INITIAL https://www.dev1.com/createcust

CURRENT https://www.dev1.com/createcust

SCOPE Environment

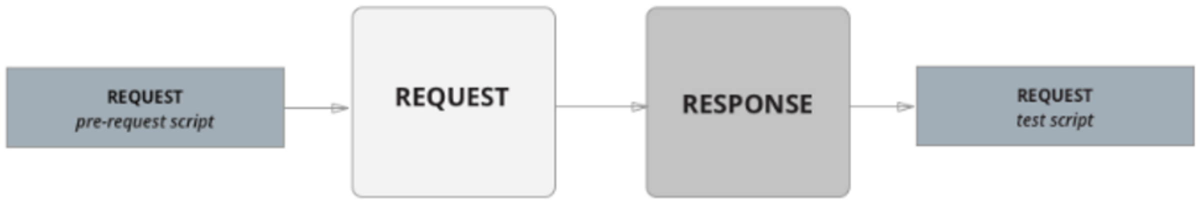
Postman – Data Driven Testing

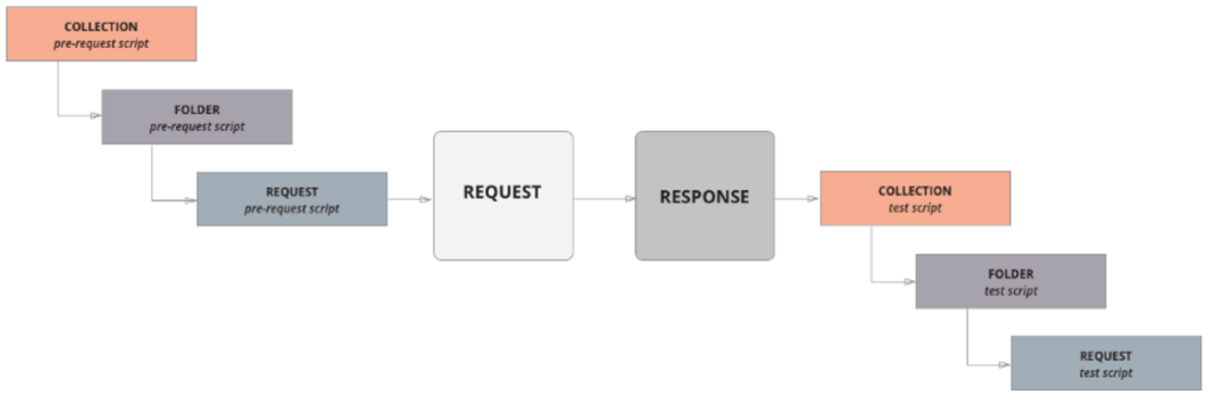
- When to use? – To run the same test with multiple sets of data
- Demonstrate using Create User API from reqres.in

Postman – Scripting

- Written in JavaScript
- Can be done at 2 places:
    1. Pre-request Scripts – Executes Before
    2. Tests – Executes After
- Can be done at 3 levels:
    1. Collection
    2. Folder
    3. Request

Postman – Script Execution Order





Postman – Validating Responses

- **pm.test** – Function – Returns Boolean indicating whether the test passed or failed – We should provide a name for the test
- **pm.response** – Object used to validate the data returned by the response
- **pm.expect** – Assertion to test the response detail

Defining Variables through Scripts

Method	Use-case	Example
pm.globals	Use to define a global variable.	pm.globals.set("variable_key", "variable_value");
pm.collectionVariables	Use to define a collection variable.	pm.collectionVariables.set("variable_key", "variable_value");
pm.environment	Use to define an environment variable in the currently selected environment.	pm.environment.set("variable_key", "variable_value");
pm.variables	Use to define a local variable.	pm.variables.set("variable_key", "variable_value");
unset	You can use unset to remove a variable.	pm.environment.unset("variable_key");

Postman – Collections - Schedule Runs

- Periodically run collection at a specified time on the Postman Cloud
- Your collection will be automatically run on the Postman Cloud at the configured frequency
- High Run frequency helps catch issues quicker but increases resource usage
- You can add up to 5 team members for Email notifications

Postman – Collections – Performance

- To simulate real-world traffic from our local machine and observe the performance of our APIs
- Virtual Users – Each user runs the collection parallelly & repeatedly for the Test Duration
- Test Duration
- Load Profile – Used to change the number of Virtual Users the run
- Data File

Postman – Monitors

- A monitor lets you run a collection periodically to check for its performance and response
- Main difference for now between Monitor and Schedule Runs is that, in Schedule Runs, you can deselect a Request under a Collection. But, in Monitors, you will have to run the Collection as a whole

Postman – Cookie

- **Cookie?** - a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number
- The servlet sends cookies to the browser by using the HttpServletResponse.addCookie(javax.servlet.http.Cookie) method, which adds fields to HTTP response headers to send cookies to the browser, one at a time
- The browser returns cookies to the servlet by adding fields to HTTP request headers

REST Assured

Java JDK Setup

- Search for “Java JDK download” in google
- Download the version which is mentioned as “long-term support release of Java SE Platform”
- Install Java by using the downloaded file
- Whenever an App requires Java, it refers to the Environment Variables to find the path to Java in the system
- Setup your “System Environment Variables”
    - Under System Variables, add JAVA_HOME=C:\Program Files\Java\jdk-21
    - Then, add the variable, path=%JAVA_HOME%\bin
- After setting up, try → cmd → java -version

Eclipse IDE Installation

- IDE – Integrated Development Environment
- Used to write and manage the code
- Famous Java IDE’s:
    - Eclipse – Most widely used – This is what we will use in this course
    - IntelliJ
- Download & Install “Eclipse IDE for Java Developers”

Maven

- Project Management Tool based on POM – Project Object Model
    - Helps in providing a defined project structure
    - Helps in managing all the dependencies

    - Helps in version control
- Downloads the dependencies from mvn repository – "https://mvnrepository.com/"
- M2 folder

Maven Setup

- From recent times, Maven is coming in as a part of Eclipse IDE itself
- Create a Maven project
- Add the below dependencies to the pom.xml file:
    - Rest-Assured – Now the project will have knowledge of Rest Assured
    - TestNG
- After adding the dependencies to the pom.xml file, make sure to update the project

REST Assured

- REST Assured is a Java library
- Used to test Webservices based on REST
- Similar to Postman tool – Used to trigger and validate a REST API
- Can be integrated with the Testing Frameworks - TestNG or Junit
- Uses BDD style – Behavior Driven Development

TestNG – Testing Framework

- A testing framework is a set of guidelines or rules used for creating and designing test cases
- A framework is comprised of a combination of practices and tools that are designed to help QA professionals test more efficiently

- TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier
- Makes use of Annotations & Attributes to manage the execution of Test Cases

TestNG - @Test Annotation

- A test method is a Java method annotated by @Test
- Acts as a Java Compiler and compiles the Method that's annotated with @Test
- Every method that's annotated with @Test will be run irrespective of whether it's a main method or not

BDD - Behavior Driven Development

- BDD is a way for software teams to work that closes the gap between business people and technical people
- Cucumber Library
- Uses the below keywords for defining the steps:-
    - Given
    - When
    - Then
    - And
    - But

Below is a simple example:

Scenario: Buy products from Amazon website

Given The home page of Amazon has loaded successfully

When The products are added to the cart

And Payment has been done successfully

Then The order details along with the reference number should be displayed to the customer

REST Assured – Installation – Setup

- What's needed?
    - Java → cmd → java –version
    - IDE → Eclipse → Eclipse IDE for Java Developers
    - Maven → No need to install separately → Will be included in Eclipse → Recent development
    - TestNG & RestAssured → Will be added as a dependency in Maven

Java – Object

- An entity that has state and behavior is known as an object
- Chair, bike, marker, pen, table, car, etc.
    - **State:** Represents the data (value) of an object
    - **Behavior:** Represents the behavior (functionality) of an object such as deposit, withdraw, etc.

Java – Class

- A class is a group of objects which have common properties
- It is a template or blueprint from which objects are created
- Cycle is a Class and Mountain, Race, Gearless etc. are Objects or instance of the Class

Java – Method

- A Method is a block of code which runs only when it's called

- Methods at times are used to perform specific actions and they are also called as functions

Java – Accessing Static and Non-Static Methods

- Show an example of how Static and Non-Static methods can be accessed

Java – Access Modifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

RestAssured – Basic GET Request

- Show an example of how to send a GET request

RestAssured – Hamcrest Matchers

- Hamcrest is a framework for writing matcher objects allowing ‘match’ rules to be defined declaratively
- Basically, it’s used for performing assertions in our test cases

RestAssured – Hamcrest Matchers

Number related assertions:

- equalTo
- greaterThan
- greaterThanOrEqualTo
- lessThan
- lessThanOrEqualTo

String related assertions:

- equalTo
- equalToIgnoringCase
- equalToIgnoringWhiteSpace
- containsString
- startsWith
- endsWith

Not Assertion: Not, inverts the meaning of other assertions

Ways to create a Request body

- Using HashMap
- Using JSON Library
- Using external JSON File
- POJO

Creating a Request body – HashMap-1

- Is a Java Collection where the data is stored as Key-Value pairs
- Import - **import** java.util.HashMap;
- Create an Object and start declaring the Key-Value pairs

```
HashMap<String,Object> requestbody=new
HashMap<String,Object>();

requestbody.put("name", "Arjun");
```

- Why not used widely?
  - Time consuming
  - Need for use of Java Collections
  - Complexity involved

Creating a Request body – JSON Library

- Various JSON Libraries available: JSON Simple, GSON, Jackson, JSON in Java(org.json)
- Add the dependency in pom.xml file
- We cannot directly pass the requestbody that we created using JSONObject
  - body(requestbody1.toString())
- Import the Library:
  - **import** org.json.simple.JSONObject;
  - **import** org.json.*;

HashMap and JSON Library

When we use HashMap, we can directly pass the request that we created, but when we use a JSON Library, we should use the toString() or toJSONString() methods

Creating a Request body – External JSON File

- Create a .json file inside the Project and store the JSON Request in it
- We the use the **File Class** in Java
  - .\\ → Represents current project location
  - Or we can paste the complete path as well
- Then we use the **FileReader Class** in Java which is used to read data from the file
  - Import the correct package from java.io – For File Class, we always use packages from java.io
  - **import** java.io.File & **import** java.io.FileReader

- Then we use the **JSONTokener Class** which takes a source string and extracts characters and tokens from it
- Finally we use the **JSONObject Class**
  - For JSONTokener, JSONObject → Import from org.json
  - **import** org.json.JSONTokener & **import** org.json.JSONObject

Validating a JSON Schema

- Get the JSON Schema from the creator/developer of the API
- In our case, create a JSON Schema using the Response
- Add the generated JSON Schema to the Classpath of our project
- Add the JSON Schema Validator dependency in the pom.xml file
- Assert the response against the stored Schema and validate the result

Schema Preparation

- Search for "JSON to JSON Schema Converter" - https://www.liquid-technologies.com/online-json-to-schema-converter
- Paste the expected JSON Response and Generate the Schema
- Save the generated Schema in – Target →Properties → Show in System Explorer → Under Classes create a txt file → Save the Schema in .json format
- Make sure that the file is saved as JSON (File Type should be JSON)

Adding the dependency in pom.xml file

- Navigate to - https://mvnrepository.com/
- Search for “json schema validator” and use the one from io.rest-assured and paste the Dependency in the pom.xml file

```
<dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>json-schema-validator</artifactId>
        <version>5.4.0</version>
</dependency>
```

- Verify whether the dependency got added to the Maven Dependencies folder

Assert the Response

- Create a Java Class
- Add the usual imports and to the make sure to add the below one as well:

```
import static io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchemaInClasspath;
```

- Assert the response body against the stored JSON Schema Document

```
assertThat().body(matchesJsonSchemaInClasspath("singleuserschema.json"))
```

Assert fail

- Demonstrate a assertion failure by changing the data type in the Schema

Path Parameters

- Variable parts of a URL path
- Used to point to a specific resource - Address – Usually specifies where we should go
- Ex: Below URL says that we should be going to the search folder in the Google server
- https://www.google.com/search

Query Parameters

- Placed at the end of the Request URL
- Format: name=value pairs
- Starts after '?' symbol
- Separated by '&' symbol
- Ex: Below URL takes us to the Search folder in Google server and there, it searches/queries for 'tiger' as an 'isch'(image search)
- https://www.google.com/search?q=tiger&tbm=isch

Declaring/Parameterizing Path and Query Parameters

- We use **.pathParam** and **.queryParam** methods in the given() portion
- For Path Param, we will have to give a name to identify the same. But for Query Param, there will already be a name
- While mentioning the URL in the when() portion, the Path Param has to be declared within **{Path Param Name}** braces
- But, Query Param will be passed automatically at the end of the URL

Cookies

- As we saw earlier, Cookies will be sent by the Server as a part of the response
- We will want to capture the Cookies information which is sent back by the Server
- Cookie Validations:
    - **.cookie(“cookie_name_here”)** – To validate that a cookie exists in the response – Irrespective of value
    - **.cookie(" cookie_name_here ", “expected_value_here")** – To extract and validate a Cookie value
    - We may not be able to validate a Cookie value as it might change every time

Cookies – Value Extraction

- Store the Response in a Response Object and then get the cookie using **.getCookie** method
- **.getCookies** method - The response cookies as simple name/value pair – Type will be Map<String, String>
- **.getDetailedCookie** method – Retrieves cookie including all attributes associated with the given name
- **.getDetailedCookies** method - The response cookies with all the attributes

Headers

- Response Header values may or may not change
- Response Header Validations:
  - **.header("header_name_here", "expected_value_here") method** – To validate whether a particular header name along with the corresponding value is present in the response
- Response Header Extraction:
  - **.getHeader** method – Get a single header value associated with the given name
  - **.getHeaders** method – Get all the response headers

Log methods

- If we just want to print the cookies and headers, we can simply use the **.log** method
- .log().all() –
- .log().body() –
- .log().headers() –
- .log().cookies() –

Parsing the Response

- Explain the JSON Response body structure using the below example:

  JSON Response Body Example.txt

- Explain the use of JSON Path Finder: https://jsonpathfinder.com/

- At times, the response might be complex and hence it’s suggested to use path finder tools wherever needed
- To do more meaningful validations, it’s always preferred to capture the response in a variable and then do the validations
- When we do the validations within the then() section, it might not be possible to many operations like looping etc.
- Then use TestNG to do the assertions
- Here, we validate using jsonPath

- If the objects within the JSON Array “data” changes their order, then the below jsonPath may not fetch Byron, as it’s based on the index

Assert.*assertEquals*(res.jsonPath().get("data[3].first_name"), "Byron");

- So to overcome this, we will have to capture all the first_name’s from the Response and then do the validation
- We have a Class called **JSON Object Class** for parsing through the JSON Response
- And it requires org.json and json-path dependencies in the pom.xml file
- We have to convert the Response from Response Format to JSON Object Format and then parse through the JSON Object
- Show an example using the “validatingResponseBody3” class

Basic Authentication

- Will be specified in the given() section
- Will use **.auth** and **.basic** methods to declare the Authentication components
- Basic Auth example:
  - GET: https://postman-echo.com/basic-auth
  - Username: postman
  - Password: password

Digest Authentication

- Similar to Basic Authentication – Only the backstage execution or the mechanism will be different
- Will be specified in the given() section
- Will use **.auth** and **.digest** methods to declare the Authentication components
- Digest Auth example:
  - GET: https://postman-echo.com/digest-auth
  - Username: postman
  - Password: password

Pre-Emptive Authentication

- Similar to Basic Authentication – Only the backstage execution or the mechanism will be different
- Will be specified in the given() section
- Will use **.auth**, **.preemptive** and **.basic** methods to declare the Authentication components
- Pre-Emptive Auth example:

    - GET: https://postman-echo.com/basic-auth
    - Username: postman
    - Password: password

Bearer Token Authentication

- Will be specified in the given() section as a header component
- Will not use **.auth** method, rather will be declared as a Header component
- Show an example of Bearer Token being passed in the Header using the Postman tool
- Header Name: Authorization
- Header Value: Bearer<space>[Bearer Token Here]
- Bearer Token example:
    - GET: https://gorest.co.in/public/v2/users
    - Bearer Token: 2fb2718ea99979c5e9ed0f93e50e92adefc2cffe886c1d80cc2485bc11170dfe

OAuth or OAuth1 Authentication

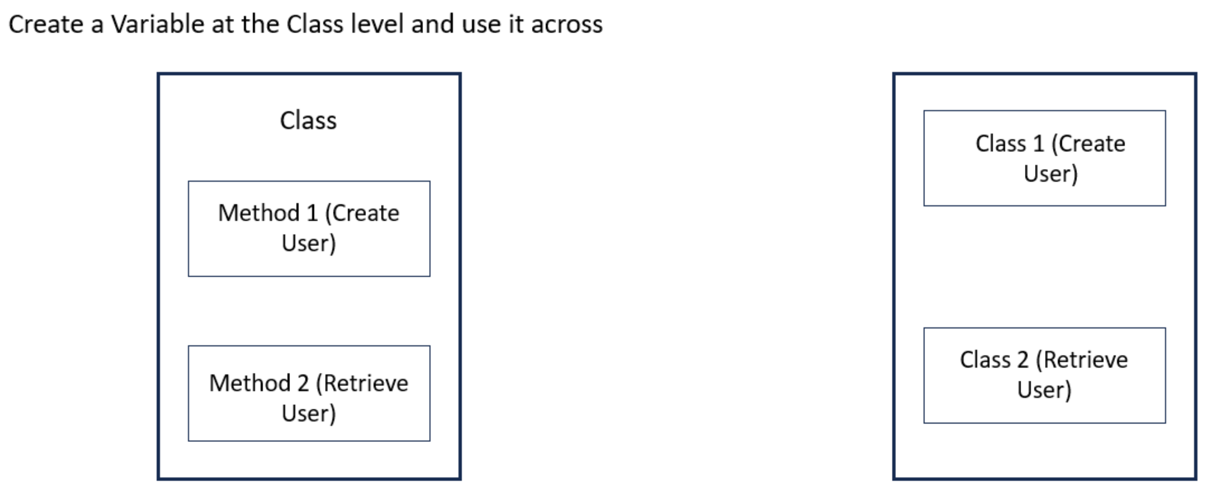
- Can be called as OAuth or OAuth1 Authentication
- Will be specified in the given() section
- Will use **.auth** and **.oauth** methods to declare the Authentication components
- OAuth Syntax:

auth().oauth("consumerKey", "consumerSecret", "accessToken", "secretToken")

GitHub Faker Library

- Can be used to generate fake data
- Need "com.github.javafaker" dependency and also will have to import "**import** com.github.javafaker.Faker;"

API Chaining in REST Assured



- We make use of the TestNG XML File and a TestNG Interface
- We do not have the concept of variables in REST Assured as we saw in Postman and hence we use TestNG feature
  - Step 1: Get the details of user at index 3
  - Step 2: Use the first name of the fetched data to create a new user

- We make use of the '**ITestContext**' interface from TestNG
- Declare a variable for the ITestContext interface and using this, we will make the desired value available for other Test Cases
- Using .setAttribute(String Name, Object Value) and .getAttribute(Attribute Name), we set and get the variable
- Consider this to be similar to Global Variable

- Create a TestNG XML File
- Package → TestNG → Convert to TestNG → Browse → Select the package location → Finish
- Order the Classes in the TestNG XML File as per their required execution order
- Execute it from the Test NG XML File

TestNG

- TestNG is a testing framework inspired from JUnit and Nunit
- Introduces some new functionalities that make it more powerful and easier to use
- Designed to cover all categories of tests: unit, functional, end-to-end, integration, etc.

TestNG XML File – Hierarchy

- **Suite** – Defined by the <suite> tag and is represented by one xml file. It can contain one or more tests
- **Test** – Defined by the <test> tag and can contain one or more TestNG Classes
- **TestNG Class** - A Java class that contains at least one TestNG annotation. It is represented by the <class> tag and can contain one or more test methods
- **Test Method** - A Java method annotated by @Test in your source
    - Suite
        - Test
            - Package
                - Class
                    - Method
- Classes will get executed in the order of their placement and Methods within the classes will be run in alphabetical order

TestNG XML File – Sample

```
<suite name="Suite1">
        <test name="Test1" >
                <classes>
                        <class name="Class1" />
                </classes>
        </test>
        <test name="Test2">
                <classes>
                        <class name="Class2"/>
                        <class name="Class3"/>
                </classes>
        </test>
</suite>
```

Show an example by running using a testing.xml file

TestNG – Attributes of @Test Annotation

- **enabled** – True/False
- **invocationCount** - Number of times this method should be invoked
- **priority** – The priority for this test method. Lower priorities will be scheduled first
- **timeOut** - The maximum number of milliseconds this test should take. Will be in Milli Seconds. Demo using Thread.sleep

- **dependsOnMethods** - The list of methods this method depends on

TestNG – Priority – Negative Values

- I prefer not to use negative values as it looks confusing to me
- Again, the methods will be executed from level of lowest priority number to the highest
- Show an example

TestNG – Groups

- Allows to perform grouping of different test methods
- Ex: We can group list of Test Cases/Methods to be run for Regression Testing

TestNG – Include/Exclude

- Used to include or exclude a particular method during the execution

TestNG – Before & After

- @BeforeSuite: The annotated method will be run before all tests in this suite have run
- @AfterSuite: The annotated method will be run after all tests in this suite have run
- @BeforeTest: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run
- @AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run
- @BeforeClass: The annotated method will be run before the first test method in the current class is invoked
- @AfterClass: The annotated method will be run after all the test methods in the current class have been run

- @BeforeMethod: The annotated method will be run before each test method
- @AfterMethod: The annotated method will be run after each test method

TestNG – Parameters

- Can be scoped under:
  - Suite Level
  - Test Level
  - Classes Level
  - Class Level

TestNG – Data Provider

- Used to run the same tests with multiple sets of values
- Explain the difference in use case between Parameters and Data Provider
- Uses @DataProvider annotation
- Show an example

TestNG – Parallel Execution

- Runs many tests concurrently or in parallel in distinct **thread** processes
- Allows you to run multiple tests concurrently on different browsers, devices, or environments instead of running them sequentially
- Advantages:
  - Reduced execution time
  - Increased test coverage

- Improved efficiency
- Early bug detection

- Disadvantages:
    - Test Dependencies
    - Increased Infrastructure Requirements

TestNG – Parallel Execution – Levels

```
<suite name="My suite" parallel="tests" thread-count="5">
<!-- contents omitted for brevity -->
</suite>
```

```
<suite name="My suite" parallel="classes" thread-count="5">
<!-- contents omitted for brevity -->
</suite>
```

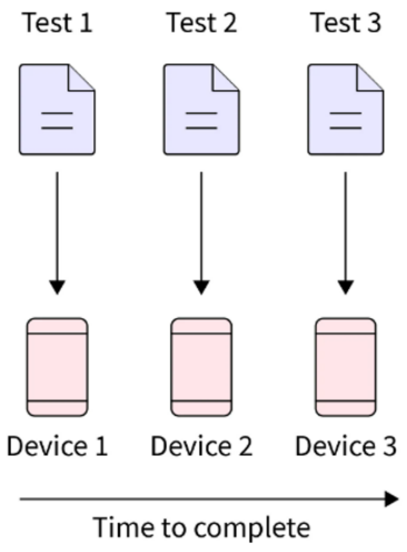
```
<suite name="My suite" parallel="methods" thread-count="5">
<!-- contents omitted for brevity -->
</suite>
```

Parallel Execution – Components

Parallel Execution Levels:

- Test
- Class
- Method
- Instance

Thread: A thread is a sequential execution flow within the process, with its own individual stack and program counter



TestNG – Linking testng.xml File with pom.xml File

- Running the tests through pom.xml file itself
- Need: Surefire Plugin and Maven Compiler Plugin
- To add the plugin: Right click on the project → Maven → Add Plugin → Enter the Group ID, Artifact ID and the Version number (get these from mvnrepository.com) → Click OK → Plugins will get added to the pom.xml file under <build> & <plugins> tags
- Now, within the Surefire Plugin, use the <configuration>, <suiteXmlFiles> & <suiteXmlFile> tags to specify the testing.xml file
- After specifying the plugins and testing.xml file in the pom.xml file, right click on the project → Maven → Update Project → Tick force update → Click OK
- Show an example

Surefire and Maven Compiler Plugins.txt

Running Maven Project through Command Prompt

- Eclipse already has in built Maven plugin and so we were able to create a Maven project
- We will now have to install and setup Maven in our OS to run the project from outside of Eclipse
- Google → Maven Download → Download the zip file – (Binary zip archive apache-maven-3.9.6-bin.zip) → Extract
- Set Environment Variables → path → Add the path of bin folder

C:\Users\91956\apache-maven-3.9.6-bin\apache-maven-3.9.6\bin

- cmd → mvn –version
- From pom.xml folder location in cmd, run → mvn test

<u>**The need to run a project from cmd**</u>

- Not having to open the IDE every time
- For CI-CD Integration – Jenkins in this case

Git vs GitHub

Git	GitHub
Git is a version control system that allows developers to track changes in their code	GitHub is a web-based hosting service for git repositories
Local	Remote
Is a Software	Is a Service
CLI Tool	GUI Tool
Installed locally on the system	Hosted on the web
Focused on version control and code sharing	Focused on centralized source code hosting

Git Installation

- Download – "Standalone Installer" - From https://www.git-scm.com/download/win
- Run and Install the application
- Right click in any folder → You will see an option "Open Git Bash Here"

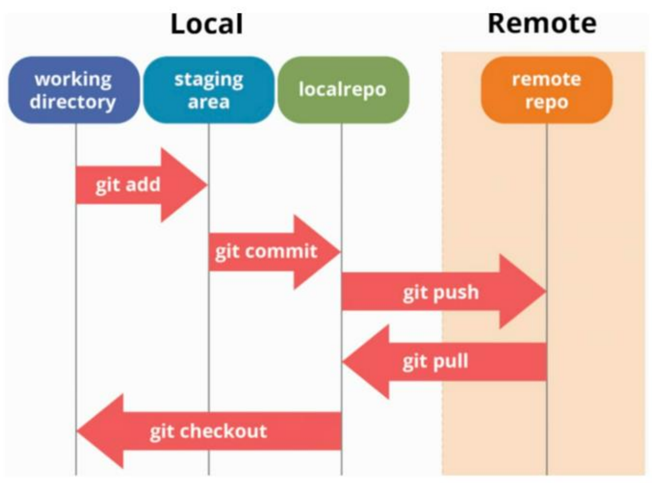
GitHub Account

Sign up – https://github.com/

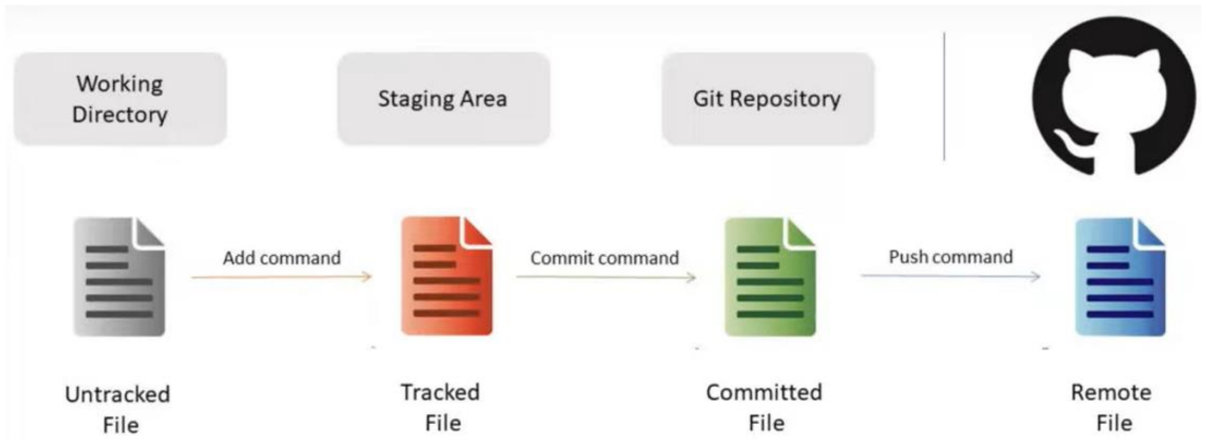
Git & GitHub Flow

- WD – Workspace - Eclipse
- SA – Virtual – Not Visible
- LR
- RR – Becomes global – From

here Jenkins will pull



Git & GitHub Flow – File names at various stages



Git Commands

- **git init** → To create a empty local repository → We won't be able to see the .git file, will be hidden
- Use below commands only during the first time:
  - **git config --global user.name "arjun"** //any username
  - **git config --global user.email "arjunvaradharajan@gmail.com"**
- **git status** → Shows the list of files and a few other details - Can be used to check the status in all the stages
- **git add –A** → Will add all the files to staging
- **git add filename.java** → To add only a specific file
- **git add *.java** → All the files having .java will be added to staging
- **git commit –m "Type any message here"** → To commit the files to Local/Git Repository

GitHub – Remote Repository

- GitHub homepage → New → Repository Name → Private/Public → Jenkins won't be able to fetch if it's a private repo, so choose public → Create Repository

- Repository will be created and you will be able to see the Repo URL
- **git remote add origin “GitHub Repo URL here”** – To establish connection between Git and GitHub - Will have to be run only one time to establish the connection for the first time
- **git push –u origin master** – Code will be pushed from local to remote repository

Note:

- **rm -fr .git** – This command can be used to delete any local repository that was created using git init command

REST Assured – Hybird Framework

Framework Characteristics

- **Characteristics of a good framework:**
    - Ease of Use
    - Scalability
    - Maintainability
    - Reusability
    - Compatibility
    - Minimal Manual Intervention

Framework Phases

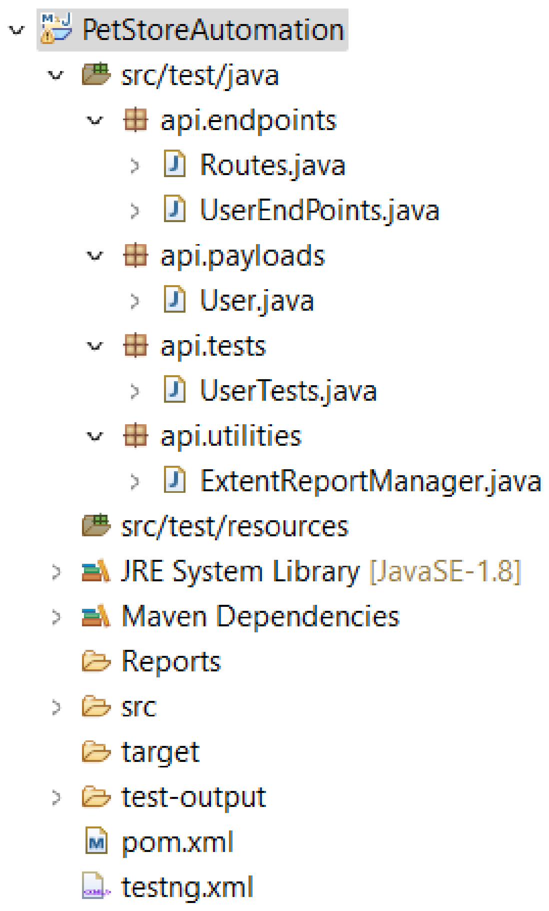
- Understanding the requirement
- Choose a proper automation tools/libraries like Selenium, REST Assured etc.
- Design the structure
- Development
- Execution
- CI – Continuous Integration

Swagger

- Swagger is an Open Source set of rules, specifications and tools for developing and describing RESTful APIs
- Swagger allows you to describe the structure of your APIs so that machines can read them
- The Swagger framework allows developers to create interactive, machine and human-readable API documentation
- https://petstore.swagger.io/

- Test Flow: Create User → Retrieve User(Using unsername) → Update User(Update email) → Delete User (Using unsername)

Framework High-level Design



```
PetStoreAutomation
  src/test/java
    api.endpoints
      Routes.java
      UserEndPoints.java
    api.payloads
      User.java
    api.tests
      UserTests.java
    api.utilities
      ExtentReportManager.java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  Reports
  src
  target
  test-output
  pom.xml
  testng.xml
```

Framework Dev

- Create a Maven Project in Eclipse → Select – Create a simple project (skip archetype selection) → Give Group and Artifact ID → Finish
- We get 4 packages as default: src/main/java; src/main/resources; src/test/java; src/test/resources
- src/main/java – This is for development code (From a Dev point of view)
- src/test/java – This is for Unit test cases (From a Dev point of view)
- resources – Will be used to maintain any required resources

- We only use test packages and so we can delete the main packages

Framework Dev – pom.xml

- Add all the required dependencies to the pom.xml file
- Also add the required plugins to the file – We will do this later when needed
- After adding, always update your project. Right click on the project → Maven → Update Project → Tick force update → Click OK

Framework Dev – Creating Structure-1

- Under src/test/java, create the below packages:
    - api.endpoints
    - api.payloads
    - api.tests
    - api.utilities

Framework Dev – Endpoints Package

- Create a Class named '**Routes**'
- This Class will be used to maintain all the URL's that will be required
- Base URL - https://petstore.swagger.io/v2
- Create User – POST - https://petstore.swagger.io/v2/user
- Retrieve User – GET - https://petstore.swagger.io/v2/user/{username} (username is a Path Parameter)
- Update User – PUT - https://petstore.swagger.io/v2/user/{username}

- Delete User – DELETE - https://petstore.swagger.io/v2/user/{username}

Framework Dev – Endpoints Package

- Create a Class named '**UserEndPoints**'
- Maintains CRUD method implementations
- Create Methods for each action and we will be calling these methods from the Tests and will be returning the responses to the Tests
- We will also get the Parameters such as payload, username etc. from the Tests

Framework Dev – Payloads Package

- Create a Class named '**User**'
- It's a POJO Class – Plain Old Java Object
  - Declare the variables based on fields in the payload
  - For each variable, initiate a getter (for retrieving) and setter (for assigning) method
  - For this, Select the variables → Go to Source → Select Generate Getters and Setters → Select All → Click Generate
  - After this, we will have to import this POJO class in the 'UserEndPoints' class
    - In UserEndPoints Class → **import** api.payloads.User;

Framework Dev – Tests Package

- Create a Class named '**UserTests**'
- This will store all the tests/test cases
- We will prepare the data here using the 'Faker' library and will pass it to the POJO class (User.java)

- Also, will write the tests/test cases here

Framework Dev – testing.xml & pom.xml Files

- After writing the test cases under UserTests.java, create a testing.xml file for the same at project level
- Add the Sure Fire and Maven Compiler Plugins to the pom.xml file and specify the testing.xml file's name within the Sure Fire plugin
- From the folder location of the pom.xml file, through the Command Prompt, run the command 'mvn test' and execute the project from the pom.xml file

Framework Dev – Git & GitHub

- Then, push the code to Git & GitHub repositories

Reports in TestNG for REST Assured

- index.html & emailable-report.html
- Show an example

Jenkins

- Open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software
- Use case: Can be used to schedule jobs which are recurring. For Ex: Daily Regression Task

Jenkins – Installation & Walk Through

- Pre-Requisites: Java JDK
- Download "Jenkins.msi" file from https://www.jenkins.io/
- For reference on installation, please watch: https://www.youtube.com/watch?v=Zdxko2bPAAw
- Jenkins URL: http://localhost:8080/
- Show an example

Jenkins – Scheduling a Job

- To run a Job at specified intervals of time
- Format:
- Values:

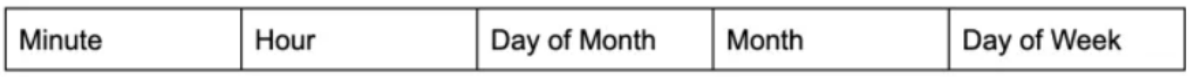
  MINUTES - (0-59)

  HOURS - (0-23)

  Day of the month - (1-31)

  Month of the year - (1-12)

  Day of the week - (0-7) where 0 and 7 are Sundays
- Show an example



Minute	Hour	Day of Month	Month	Day of Week