Stage 1 – API Testing using postman

Stage 2 – API Automation (Robot Framework + Request Library)

1. API – Application Programming interface

* A Mechanism that enables two software component to communicate to each other.

Easily understand the api architecture with client and server.

* Server – the database that holds the stock details and its price
* Client – the mobile app

1. Ways of API works

SOAP – Simple Object Access Protocol

* It is very less flexible that was widely used in the past. Client and server exchange the messages using xml.

REST API – Representation State transfer

* The clients send the request to the server as data.
* The server uses the client input to start internal function and return the data
* REST defines a set of functions such as GET, POST, PUT, DELETE etc.,
* Client and server exchange the data using http.
* Main feature of REST is statelessness. Server do not store client data between the requests.

Web API / Web Services

* An application processing the interface between a web server and web browser.
* All web services are API but not all API are web services.

GraphQL

* GraphQL is a query language that was developed specifically for APIs.
* It is designed to the give the specific data requested by the client and not more.
* GraphQL gives front end developers the ability to query multiple databases, microservices, and API with a single graphql endpoint.

1. Installation
2. Install SOAP (Not required now, just listen)
3. Install Postman (RestAPI – development, testing, management )
4. SOAP API Testing - WSDL (Web Service Description Language)
5. Create SOAP Project
6. Provide the WSDL details
7. It shows the available methods. Try running the methods by providing the required data in the xml (request body)
8. REST API – Web Archive Description Language (HTTP) – Things to note
9. Method – Select/Update/Delete/Create (GET, PUT, DELETE, POST, PATCH, HEAD, OPTIONS)
10. Url 🡪 base url/resource
11. Request Header & body – depends on the request
12. Response (Server Status Code)
    1. 200 🡪 Success
    2. 201 🡪 Created
    3. 404 🡪 resource not found
    4. 401 🡪 Auth error
    5. 403 🡪 Forbidden auth error

HTTP Methods:

Get 🡪 select (Path parameter & query parameter)

Post 🡪 create

Delete 🡪 delete

PUT/PATCH 🡪 update

TestCase for Login UI in facebook

1. Valid Username and valid password
2. Valid username and invalid password
3. Invalid username and valid password
4. Valid Mobile number and password (all combinations)
5. Empty username and empty password (all combinations)

API – assume checking product in amazon using product id

1. Without authorization
2. With restricted authorization
3. With full authorization
4. Without parameter
5. With invalid parameters
6. With proper parameters
7. …..
8. Work on Petshop API – Get pet by id (Get Method with Parameter)

Base url - <https://petstore.swagger.io/v2>

Resource - /pet/{petid}

Any one tool is enough:

1. Get pet by id 🡪 Test using SoupUI
   1. Create a rest project
   2. Add the baseurl+resource
      1. Baseurl (endpoint) +resource+pathparameter
      2. <https://petstore.swagger.io/v2/pet/5>
   3. Run the request and check the response. Status code is 200
   4. We can add test case to check. Make sure minimum one assertion is added.
      1. Valid pet id 🡪 200 status code
      2. Invalid pet id (-90) 🡪 400 status code
      3. Not available pet id 🡪 404 status code
2. Structure of the postman to create request

Workspace

Collections 🡪 folder

Create All Request 🡪 HTTP Request

Sub Collections 🡪 Folder

Create Any Request 🡪 HTTP Request

1. Get pet by id 🡪 Test using postman
   1. <https://petstore.swagger.io/#/>
2. Can create variable at different scopes
   1. Global
   2. Collection
   3. Environment
   4. Data
   5. Local
3. Activity
   1. Create a collection “PetStore Orders API”
   2. Get pet inventories by status
4. API Security
   1. API Key
   2. Authentication
5. Oauth 2.0
   1. Auth url
   2. Access token url
   3. Client id
   4. Secret id
   5. Scope
   6. Re-direct url – app owner/ developer

Oauth 2.0

Level 1 - get authorization code

endpoint --> authorization url?client\_id=&scope=&redirect\_uri= ---> Auth code

Level 2 - Use authorization code and get access token

endpoint --> access\_token\_url?auth\_code, client\_secret ---> access token

Calendar API –

[https://developers.google.com/calendar/api/v3/reference/](https://developers.google.com/calendar/api/v3/reference/calendarList/list)

Authorization:

<https://developers.google.com/oauthplayground/>

* As of now, we are using access token (bearer) while doing the request.
* Next class, we will use complete oauth 2.0 configuration.

1. Get Calendar List api
   1. No Auth
   2. Full authorization

<https://developers.google.com/calendar/api/v3/reference/calendarList/list>

1. Create calendar with name as “Fun day 2023”

<https://developers.google.com/calendar/api/v3/reference/calendars/insert>

1. Collection configuration
   1. Authorization can be configured in collection level itself instead of adding the authorization on each request under the collection.
   2. If you configure, pre-request and post response at collection level then it will run before and after each request. Make sure the authorization is selected as “Inherit auth from parent”
   3. Variable can be created at collection level
2. Oauth 2.0 configuration in postman
   1. For google, callback url for postman,

Browser - <https://oauth.pstmn.io/v1/browser-callback>

Desktop - <https://oauth.pstmn.io/v1/callback>

1. How to write javascript in post response and create a dynamic variable?
2. Data Driven using postman
   1. Csv
   2. Json
3. Git API
   1. Rest api 🡪 <https://docs.github.com/en/rest/repos?apiVersion=2022-11-28>
   2. Authentication
      1. Basic auth (username & password)
      2. Bearer token
      3. Oauth 2.0

<https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/authorizing-oauth-apps>

* 1. [List public repositories](https://docs.github.com/en/rest/repos/repos#list-public-repositories)
  2. [List repositories for the authenticated user](https://docs.github.com/en/rest/repos/repos#list-repositories-for-the-authenticated-user)
  3. [Create a repository for the authenticated user](https://docs.github.com/en/rest/repos/repos#create-a-repository-for-the-authenticated-user)
  4. [Update a repository](https://docs.github.com/en/rest/repos/repos#update-a-repository)
  5. [List repositories for a user](https://docs.github.com/en/rest/repos/repos#list-repositories-for-a-user)
  6. [Delete a repository](https://docs.github.com/en/rest/repos/repos#delete-a-repository)

API Automation

1. Programming knowledge is required
   1. Java / C# / Python
      1. Selenium – Web Automation
      2. Appium – Mobile Automation
      3. Restassured.io (Java), RestSharp (C#), RequestLibrary (Python) – API Automation
2. Robot Framework (Keyword Driven Framework – extensively designed using Python)
3. Installation for Robot Framework
   1. Install python

Add python and scripts path to environment Path variable.

C:\Program Files\Python311\

C:\Program Files\Python311\Scripts

* 1. Install pycharm IDE (Community)
  2. Install Robot Framework

**pip install robotframework**

1. Basics of python
2. Basic of Robot Framework
   1. Standard Libraries – Builtin, DateTime, String, Collections etc
   2. External Libraries
      1. Selenium library (Web Automation)
      2. Appium library (android/ios automation)
      3. Request library (API Automation)
3. Create a project and map the python interpreter.
4. Install robot framework language server plugin in pycharm
5. Sections in robot framework
   1. Settings
   2. Test Cases
   3. Variable
   4. Keywords
6. Variable in robot framework
   1. Scalar $
   2. List @
   3. Dictionary &

Robot framework- Web Automation - Selenium Library

1. Selenium Library - <https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html>
2. Launch browser
3. Architecture of webdriver

Source code (Robot framework+selenium) 🡪 Browser

1. Click, type, Select
2. Inspect 🡪 tagname, attribute, text or not
3. Basic locators
   1. Id
   2. Name
   3. Classname
   4. Tagname
   5. Link text
   6. Partial link text

When there are duplicate locators then find\_element picks the first element.

1. Advance locators
   1. Xpath
   2. Css
2. To inspect 🡪 f12 or ctrl+shift+c
3. Find\_element 🡪 takes only 500ms to check the presence
4. Synchronization
   1. Unconditional wait (not recommended ) -- Sleep 8s
   2. Conditional wait
      1. Implicit wait
         1. Default implicit wait – 0s
         2. Applicable for all find\_element and find\_elements methods
         3. Example: Implicit wait – 30s
            1. If element not present, it will check for 30s and then throw error
            2. If element is present, it will do the operation the immediately
      2. Explicit wait
5. Dropdown
   1. With Select tag
   2. Without select tag
      1. Click Element
6. Mutliple tabs/windows, frame, alert
7. Multiple tabs/windows
   1. Using title
8. Close window vs Close browser
9. Frame – embedding one html into another html file
   1. Even though the locator is correct, we get element not found error.
   2. Check for tagname – frame or iframe
   3. Use Select frame keyword to switch to the frame.
10. Alert
11. Mouse/Keyboard activities
12. Upload the file
13. Javascript
    1. Click on hidden elements
    2. Type on readonly text box

Robot Framework – API Automation – Request Library

1. Install the request library
2. Keyword doc – <https://marketsquare.github.io/robotframework-requests/doc/RequestsLibrary.html>

Postman

1. Monitor – scheduler
2. Newman 🡪 helps to run postman collections in command line
   1. Install <https://nodejs.org/en>
   2. Install newman through command line

npm install -g newman

* 1. To run collections

C:\Mine\Company\Swift Support>newman run "PetShop API.postman\_collection.json"

* 1. To run collection with environment

C:\Mine\Company\Swift Support>newman run "PetShop API.postman\_collection.json" -e QA.postman\_environment.json

Customize framework for api automation

1. Create project for each application
2. Create a test suites and suite file (Robot file)
3. Create a first case and then connect to test templates if needed to run with multiple set of test data.
4. Test Template – To run one Test case with multiple set of test data
   1. Create a keyword with proper arguments
   2. Register the template at setting level or test level
5. Connect Test Template with DataDriven Library that helps to achieve data driven using csv or excel

pip install --upgrade robotframework-datadriver

pip install --upgrade robotframework-datadriver==0.2.7

Assignments:

1. Complete the patch update in calendar api -<https://developers.google.com/calendar/api/v3/reference/calendars/patch>
2. Complete at least 4 api calls - <https://reqres.in/>

Locating strategy

|  |  |  |
| --- | --- | --- |
| **Strategy** | **Match based on** | **Example** |
| id | Element id. | id:example |
| name | name attribute. | name:example |
| identifier | Either id or name. | identifier:example |
| class | Element class. | class:example |
| tag | Tag name. | tag:div |
| xpath | XPath expression. | xpath://div[@id="example"] |
| css | CSS selector. | css:div#example |
| dom | DOM expression. | dom:document.images[5] |
| link | Exact text a link has. | link:The example |
| partial link | Partial link text. | partial link:he ex |
| sizzle | Sizzle selector deprecated. | sizzle:div.example |
| data | Element data-\* attribute | data:id:my\_id |
| jquery | jQuery expression. | jquery:div.example |
| default | Keyword specific default behavior. | default:example |