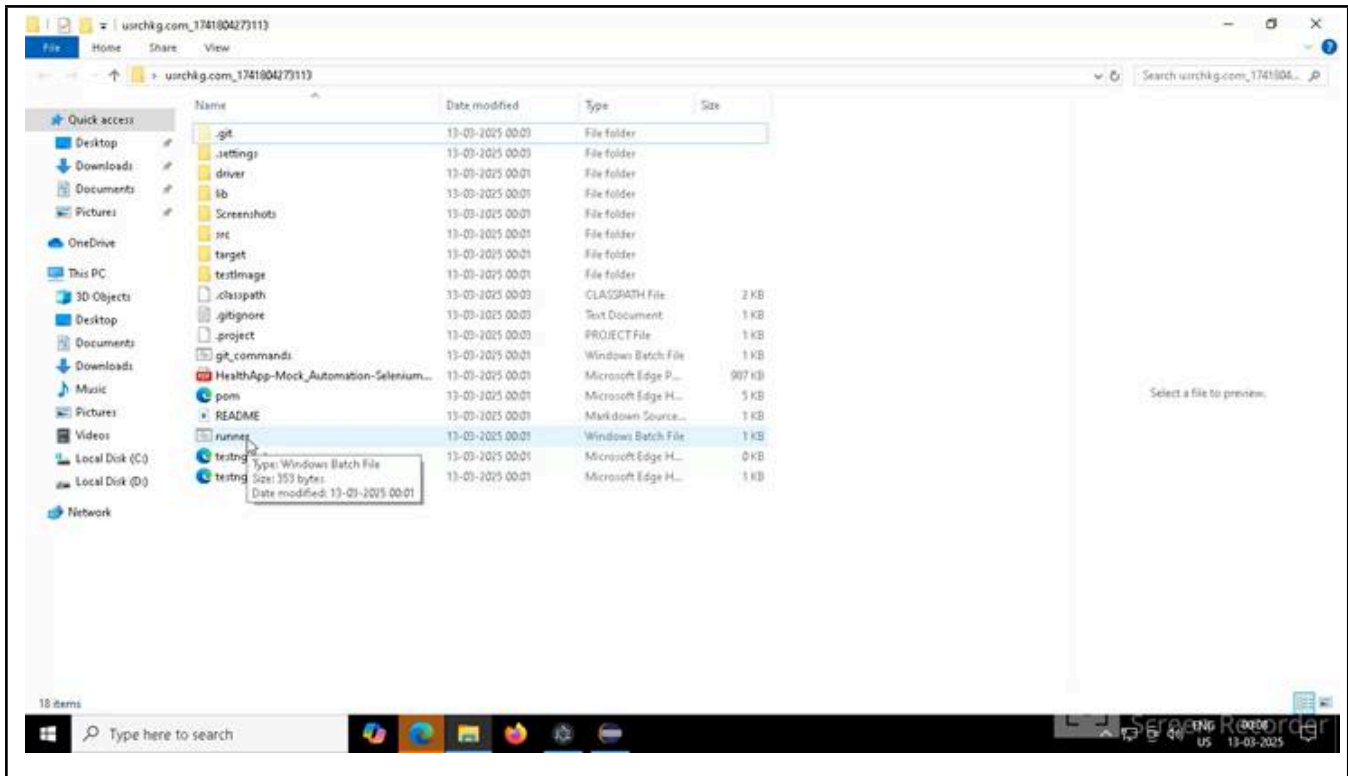# REST-ASSURED
(API - AUTOMATION PROJECT – PL1-6)

Pre-requisite:

Before you start working on your project, execute the runner file present in your project folder (Simply by double click). <mark>This is mandatory.</mark>


This will launch a command terminal for you where it will keep on pushing your updated code to GIT on regular intervals. Keep that command terminal open at the backend and you can continue working on your project.
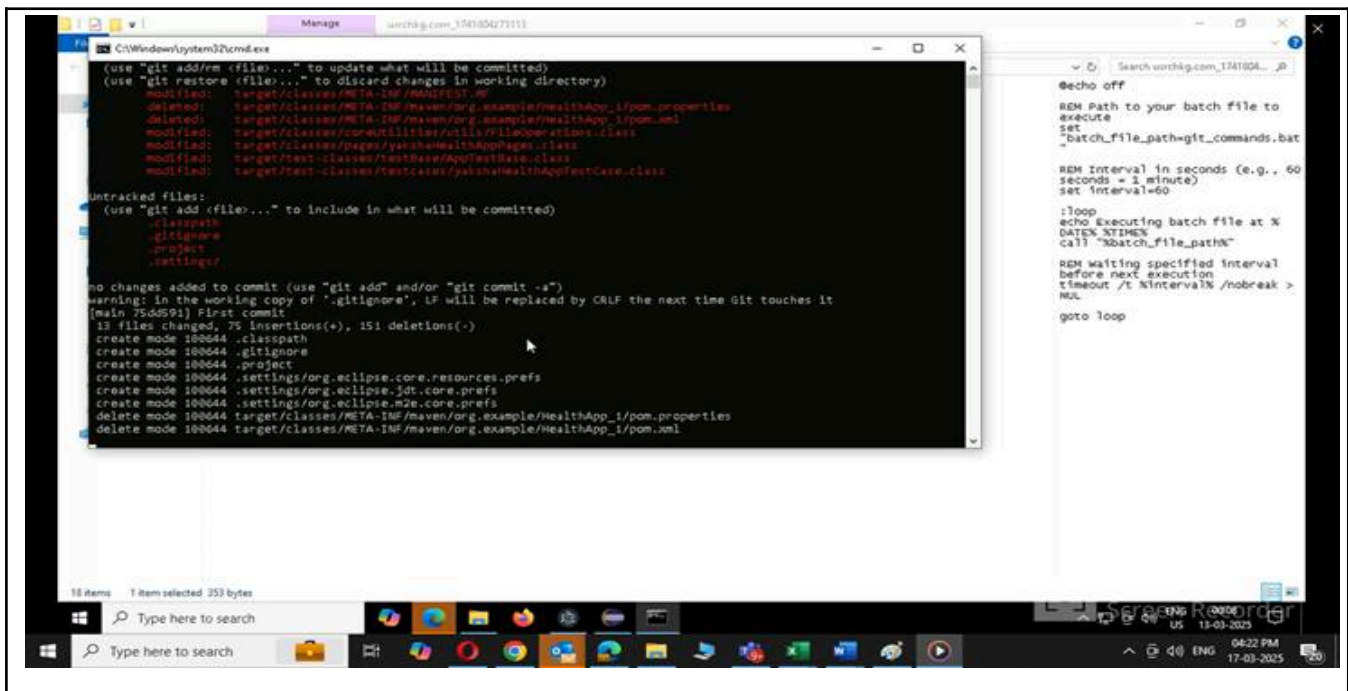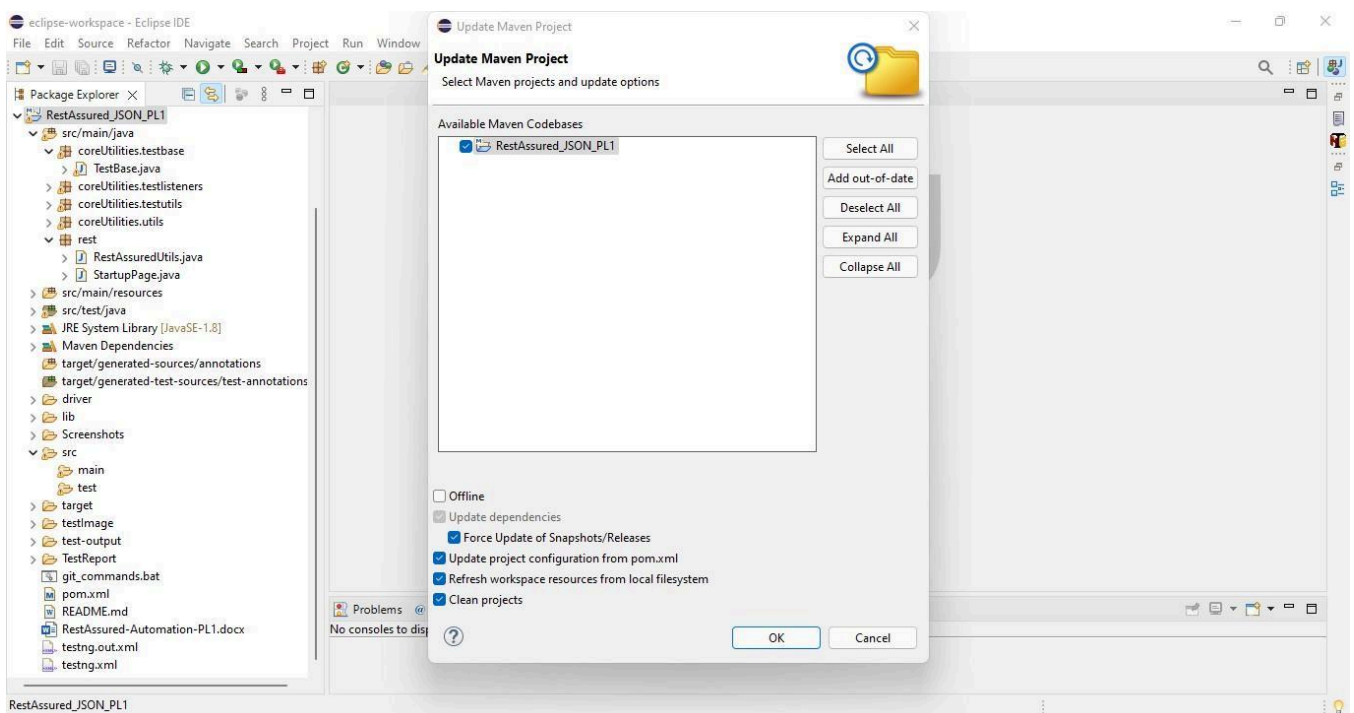
As soon as you import a project in eclipse, update the project using the maven update option as below. This is to resolve issue if any maven dependency not downloaded properly:

1. Right click on project : Go to "Maven" : Select "Update Project"
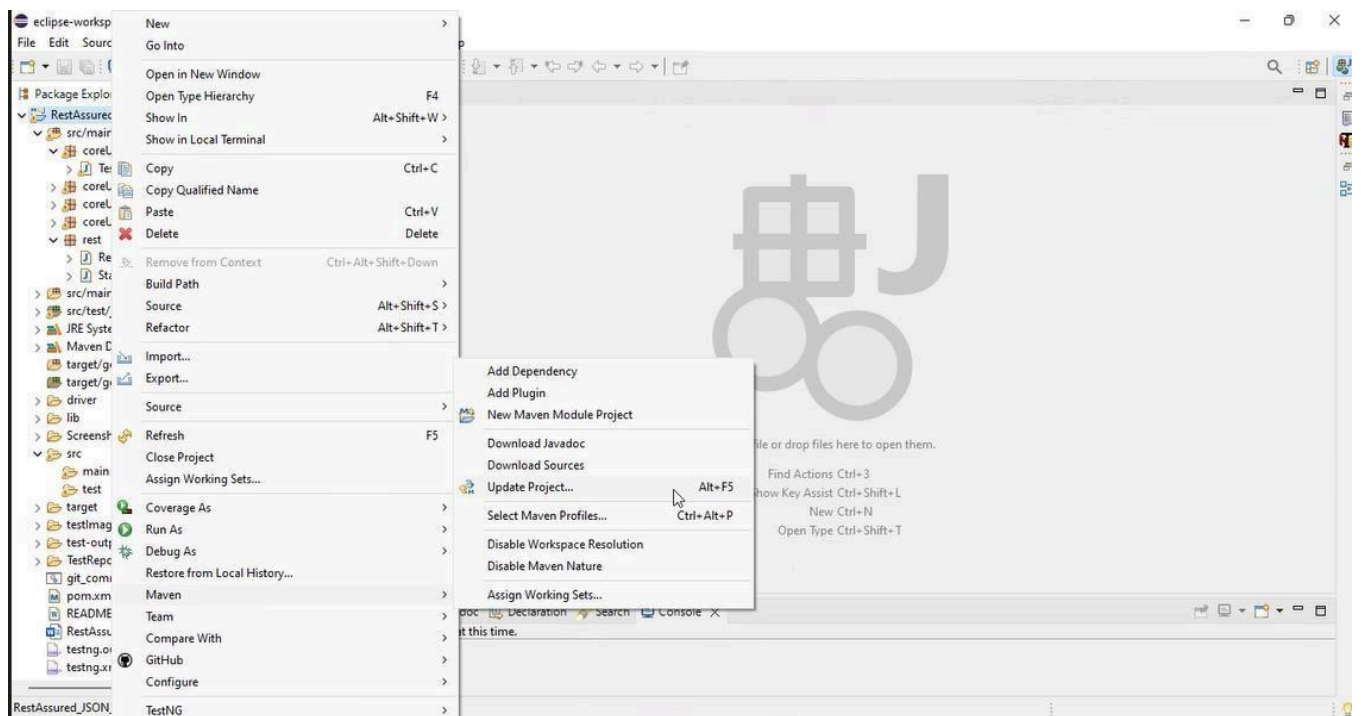
2. In Update Maven Project Box Select "Force Update of Snapshots/Releases" and click OK

Template Code Structure:

- Below are the packages and files you will be required to work upon.
- Other Files and packages you can ignore.
- In other Files and packages do not do any changes. It would affect your evaluation.
- You are not required to work in the "Test" Folder. Files there are noneditable. Editing those files and trying to save them will throw errors and would affect your evaluation.

Package	Class/File	Description
src/main/java/coreUtilities/utils/	FileOperations.java	1. It contains methods to read data from Excel files. 2. The method is already implemented.
/src/main/java/rest	ApiUtil.java	1. All core activities to be performed here. 2. The comments associated with each templated method here describe the expectation. 3. Declare any 4. variable/object you need to share data/status between different methods. 5. **Do not modify the signature of methods declared here. Signature means method name, method return type, number of arguments, any exception thrown (if present, if not then please don't add).** 6. Do not modify the signature of methods declared here. 7. You can create additional supportive common methods in 8. CommonEvents class.
/src/main/resources/	config.properties	1. Data present to be used in creating a new Post.
/src/main/resources/	TestData.xlsx	1. Data is present for creating requests.
/src/main/java/coreUtilities/utils	CommonEvents.java	1. Contains all common activities. 2. Certain templated common methods are declared here. 3. You implement them as per your needs. 4. You can add any additional method for common activity here
	Testng.xml	Execution needs to be kick-started from TestNG.xml

PROBLEM STATEMENT

Need to automate the following activities using RestAssured.

Key Activities to implement:

#	Summary	Action	Expected Result
1	Retrieve Localization Details with Valid Cookie Authentication using method: **getLocalizationValid(String endpoint, String cookieValue, Map<String, String> body)**	1. Construct the final URL by combining the BASE_URL and the provided endpoint parameter: a. https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/localization 2. Initialize a RequestSpecification using RestAssured: a. Add a cookie with name "orangehrm" and the value passed as cookieValue. b. Set the "Content-Type" header to "application/json". 3. Trigger a GET request to the above URL. 4. Parse the JSON response using **JsonPath**. From the `"data"` object: ● Extract `"language"` and save it into **language** variable ● Extract `"dateFormat"` and save it into **strDateFormat** variable 5. Wrap each value into its own **List<Object>** collection: ● Wrap **language** into **languages** list ● Wrap **strDateFormat** into **dateFormat** list 6. Create a CustomResponse object with: a. Full Response b. The extracted statusCode in variable statusCode and statusLine in variable status. c. languages list	1. Method should return a fully initialized CustomResponse object with: a. statusCode = 200 b. status = OK (or valid status line like "HTTP/1.1 200 OK") c. A non-null and non-empty languages list d. A non-null and non-empty dateFormat list e. The complete raw Response object

		d. dateFormat list 7. Return the CustomResponse object.	

2	Retrieve a list of active languages using session cookie authentication through method: **getActiveLanguages(String endpoint, String cookieValue, Map<String, String> body)**	1. Construct the complete URL by concatenating the BASE_URL and the provided endpoint:
		   a. https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/i18n/languages?limit=50&offset=0&sortOrder=ASC&activeOnly=true
		2. Create a RequestSpecification using RestAssured:
		   a. Add a session cookie named "orangehrm" with the provided cookieValue.
		   b. Add a header "Content-Type" with the value "application/json".
		3. If a request body is provided (not null), attach it to the request using .body().
		4. Send a GET request to the constructed URL.
		5. Parse the JSON response using **JsonPath**. From the `"data"` object:
		   ● Extract `"id"` and save it into **strlanguageId** variable
		   ● Extract `"name"` and save it into **strlanguageName** variable
		   ● Extract `"code"` and save it into **strlanguageCode** variable
		6. Wrap each value into its own **List<Object>** collection:
		   a. Wrap **strlanguageId** into **languageId** list
		   b. Wrap **strlanguageName** into **languageName** list
		   c. Wrap **strlanguageCode** into **languageCode** list
		7. Create and return a CustomResponse object initialized with:
		   a. The complete Response
		   b. The extracted statusCode in
			1. The method should return a CustomResponse object with:
			   a. statusCode = 200
			   b. status equal to a valid HTTP status line like "HTTP/1.1 200 OK"
			   c. Non-null and non-empty lists for:
			      i. languageId
			      ii. languageName
			      iii. languageCode
			   d. The complete raw Response object

		variable statusCode and statusLine in variable status. c. languageId, languageName, languageCode lists	

3	Retrieve dashboard shortcut access permissions using session-based authentication through the method:
	getDashboardShortcuts(String endpoint, String cookieValue, Map<String, String> body)
		1. Construct the final URL by combining the BASE_URL and the provided endpoint:
		   ● https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/dashboard/shortcuts
		2. Create a RequestSpecification using RestAssured:
		   ● Add a session cookie named "orangehrm" using the provided cookieValue.
		   ● Set the request header "Content-Type" to "application/json".
		3. If a request body (body) is provided (i.e., not null), attach it using .body() method.
		4. Parse the JSON response using **JsonPath**.
		5. Access each boolean field under the `"data"` object using bracket notation for dotted keys:
		   ● Extract `"leave.assign_leave"` and save it into **assignLeave** variable
		   ● Extract `"leave.leave_list"` and save it into **leaveList** variable
		   ● Extract `"leave.apply_leave"` and save it into **applyLeave** variable
		   ● Extract `"leave.my_leave"` and save it into into **myLeave** variable
		   ● Extract `"time.employee_timesheet"` and save it into **empTimesheet** variable
			1. The method should return a CustomResponse object containing:
			   ● statusCode = 200
			   ● A valid status line such as "HTTP/1.1 200 OK"
			   ● Non-null and non-empty lists for the following keys:
			   ➔ leaveAssignLeave
			   ➔ leaveLeaveList
			   ➔ leaveApplyLeave
			   ➔ leaveMyLeave
			   ➔ timeEmployeeTimesheet
			   ➔ timeMyTimesheet
			   ● The complete Response object from the API call

| | | ● Extract `"time.my_timesheet"` and save it into **myTimesheet** variable
6. Wrap each of these fields into separate List<Object> collections:
   ● Wrap **assignLeave** into **leaveAssignLeave** list
   ● Wrap **leaveList** into **leaveLeaveList** list
   ● Wrap **applyLeave** into **leaveApplyLeave** list
   ● Wrap **myLeave** into **leaveMyLeave** list
   ● Wrap **empTimesheet** into **timeEmployeeTimesheet** list
   ● Wrap **myTimesheet** into **timeMyTimesheet** list
7. Create a CustomResponse object using:
   ● The complete Response object
   ● The extracted statusCode in variable statusCode and statusLine in variable status.
   ● All above mentioned six field lists
8. Return the populated CustomResponse object. | |
| 4 | Retrieve a list of OpenID authentication providers using session-based authentication through method: **getAuthProviders(String endpoint, String cookieValue, Map<String, String> body)** | 1. Construct the final URL by concatenating BASE_URL and the given endpoint:

https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/auth/openid-providers?limit=50&offset=0

2. Create a RequestSpecification using RestAssured:

   ● Include the session cookie "orangehrm" using the provided cookieValue | ● Method must return a valid CustomResponse object containing:

   ● statusCode = 200

   ● status like "HTTP/1.1 200 OK"

   ● Non-null and non-empty lists for:

      ▪ providerIds

      ▪ providerNames

      ▪ providerUrls

      ▪ providerStatuses |

		● Add the header "Content-Type" with value "application/json"
		
		3. If a request body (body) is provided, attach it using .body().
		
		4. Trigger a GET request to the above URL.
		
		5. Parse the JSON response using **JsonPath**. From the `"data"` array:
		   ● Extract `"id"` fields into **providerIds** variable
		   ● Extract `"providerName"` fields into **providerNames** variable
		   ● Extract `"providerUrl"` fields into **providerUrls** variable
		   ● Extract `"status"` fields into **providerStatuses** variable
		   ● Extract `"clientId"` fields into **clientIds** variable
		6. Each of these should be stored as a **List<Object>**.
		
		7. Return a CustomResponse object populated with:
		
		   ● Full Response object
		
		   ● The extracted statusCode in variable statusCode and statusLine in variable status.
		   ● Extracted field lists: providerIds, providerNames, providerUrls, providerStatuses, clientIds
			▪ clientIds
			  o The full raw Response object

| 5 | Retrieve a list of registered OAuth clients in the admin panel using session-based cookie authentication via method: **getAdminOAuthList(String endpoint, String cookieValue, Map<String, String> body)** | 1. Construct the final URL by appending the endpoint to the base URL:

https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/oauth-clients?limit=50&offset=0

2. Use RestAssured to initialize a RequestSpecification:

    ● Add the session cookie "orangehrm" using the provided cookieValue

    ● Set the request header "Content-Type" to "application/json"

3. If a request body (body) is not null, include it in the request.

4. Trigger a GET request to the above URL.

5. Parse the JSON response using **JsonPath**. From the "data" array:

    ● Extract "id" fields into **ids** list
    ● Extract "name" fields into **names** list
    ● Extract "clientId" fields into **clientIds** list
    ● Extract "redirectUri" fields into **redirectUris** list
    ● Extract "enabled" flags into **enabledFlags** list
    ● Extract "confidential" flags into **confidentialFlags** list

6. All of these should be stored as **List<Object>**.

7. Create an instance of OAuthClientInfo and populate it with the extracted fields.

8. Return a CustomResponse object initialized with: | ● The method must return a CustomResponse containing:

    ● statusCode = 200

    ● A valid status line like "HTTP/1.1 200 OK"

    ● An OAuthClientInfo object with non-null and non-empty lists for:

        ▪ ids
        ▪ names
        ▪ clientIds
        ▪ redirectUris
        ▪ enabledFlags
        ▪ confidentialFlagss

● The full raw Response object |

		• The full Response • The extracted statusCode in variable statusCode and statusLine in variable status. • The **clientInfo** object containing all OAuth client details	

6	Retrieve a list of employees with detailed information using session-based authentication via method: **getEmployeeList(String endpoint, String cookieValue, Object body)**	1. Construct the complete URL by appending the endpoint to the base URL:
		
		https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/pim/employees?limit=50&offset=0&model=detailed&includeEmployees=onlyCurrent&sortField=employee.firstName&sortOrder=ASC
		
		2. Initialize a RequestSpecification using RestAssured:
		
		    • Add the session cookie "orangehrm" with the provided cookieValue.
		
		    • Set the header "Content-Type" to "application/json".
		
		3. If a request body (body) is provided (not null), include it in the request using .body().
		
		4. Trigger a GET request to the constructed URL and extract the Response.
		
		5. Parse the JSON response using **JsonPath**. From the "data" array:
		    • Extract "empNumber" fields into **empNumber** list
		    • Extract "lastName" fields into **lastName** list
		    • Extract "firstName" fields into **firstName** list
			• The method should return a CustomResponse object with:
			
			    • statusCode = 200
			
			    • A valid status string (e.g., "HTTP/1.1 200 OK")
			
			    • Non-null and non-empty lists for:
			
			        ▪ empNumber
			
			        ▪ firstName
			
			        ▪ lastName
			
			        ▪ employeeId
			
			• The complete raw Response object

		● Extract "employeeId" fields into **employeeId** list
		
		6. All of these should be stored as **List<Object>** collections.
		
		7. Create and return a CustomResponse object containing:
		
		   ● The full Response
		
		   ● The extracted statusCode in variable statusCode and statusLine in variable status.
		   ● Lists for empNumber, lastName, firstName, and employeeId	

7	Update an existing language package using session-based authentication through method:
	putLanguagePackage(String endpoint, String cookieValue, Object body)
		1. Retrieve a language package ID not present in the current language list using helper method: getMissingLanguageIdLessThan450().
		
		2. Construct the final URL using the base URL and the language package ID as a path variable:
		https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/i18n/languages/{lang_package_id}
		
		3. Initialize a RequestSpecification using RestAssured:
		
		   ● Add the "orangehrm" cookie with the given cookieValue.
		
		   ● Set the request header "Content-Type" to "application/json".
		
		4. If the body is not null, include it using .body().
		
		5. Trigger a PUT request to the endpoint and extract the Response.
			● The method should return a valid CustomResponse containing:
			
			   ● statusCode = 200
			
			   ● A valid HTTP status (e.g., "HTTP/1.1 200 OK")
			
			   ● Non-empty string values for:
			
			     ▪ adminLangId
			
			     ▪ adminLangName
			
			     ▪ adminLangCode
			
			   ● The complete raw Response object

		6. Parse the JSON response using **JsonPath**. From the `"data"` object:	
		- Extract `"id"` field and store it in **adminLangId** - Extract `"name"` field and store it in **adminLangName** - Extract `"code"` field and store it in **adminLangCode** 7. All of these should be stored as **String** variables. 8. Return a CustomResponse object initialized with: - The full Response - The extracted statusCode in variable statusCode and statusLine in variable status. - Extracted string values: adminLangId, adminLangName, adminLangCode	
8	Delete one or more language packages using session-based authentication via method: **deleteLangById(String endpoint, String cookieValue, String requestBody)**	1. Compose the full URL by appending the endpoint to the base URL: https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/i18n/languages 2. Initialize a RequestSpecification using RestAssured: - Add a session cookie named "orangehrm" with the provided cookieValue - Set the "Content-Type" header to "application/json" 3. Attach the constructed requestBody to the request.	- The method should return a CustomResponse containing: - statusCode = 200 - A valid HTTP status line such as "HTTP/1.1 200 OK" - A non-null and non-empty List<Object> containing the deletedIds - The full raw Response object

		4. Send a DELETE request to the full URL and extract the Response object.
		
		5. Parse the JSON response using **JsonPath**. From the `"data"` array:
		    ● Extract all IDs of deleted languages and store them in **deletedIds** (List<Object>)
		6. Return a CustomResponse object with:
		
		    ● The full Response object
		
		    ● The extracted statusCode in variable statusCode and statusLine in variable status.
		    ● The deletedIds list	

9	Update the enabled/disabled status of various system modules via a PUT request with session-based cookie authentication using method: **putModulesSettings(String endpoint, String cookieValue, String requestBody)**	1. Construct the complete URL by appending the endpoint to the base URL:
		
		https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/modules
		2. Initialize a RequestSpecification using RestAssured:
		
		    ● Add the cookie "orangehrm" with the value provided by cookieValue.
		
		    ● Set the header "Content-Type" to "application/json".
		
		    ● Attach the JSON request body using .body(requestBody).
		
		3. Trigger a PUT request to the above endpoint and extract the Response.
		
		4. Parse the response using JsonPath and retrieve the "data" object as a Map<String,
			● The method must return a CustomResponse object that includes:
			
			    ● statusCode = 200
			
			    ● A valid status line (e.g., "HTTP/1.1 200 OK")
			
			    ● A non-null Map<String, Object> containing updated moduleSettings
			
			    ● The complete raw Response object

		Object> which holds the current module status and save it into **moduleSettings.**
		
		5. Return a CustomResponse object initialized with:
		
		    ● The full Response object
		
		    ● The extracted statusCode in variable statusCode and statusLine in variable status.
		    ● Parsed moduleSettings map containing key–boolean pairs (e.g., "admin": true)	

10	Create a new OpenID provider entry using a POST request with session-based authentication via method: **postOpenIdProvide(String endpoint, String cookieValue, String requestBody)**	1. Build the full API endpoint URL:
		
		https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/auth/openid-providers
		
		2. Initialize a RequestSpecification using RestAssured:
		
		    ● Add the "orangehrm" session cookie
		
		    ● Set "Content-Type" header to "application/json"
		
		    ● Attach the constructed requestBody using .body()
		
		3. Trigger a POST request to the endpoint and extract the Response object.
		
		4. Parse the JSON response using **JsonPath**. From the "data" object:
		    ● Extract the **name** and save it in name
		    ● Extract the **url** and save it in url
		    ● Extract the **clientId** and save it in clientId
			● The method should return a valid CustomResponse object containing:
			
			    ● statusCode = 200
			
			    ● A valid status string like "HTTP/1.1 200 OK"
			
			    ● Non-null, correctly mapped values for:
			
			        ▪ name
			
			        ▪ url
			
			        ▪ clientId
			
			        ▪ clientSecret
			
			    ● The full Response object

| | | Extract the **clientSecret** and save it in `clientSecret` 5. All of these should be stored as **String** variables.

6. Return a CustomResponse object with: The full Response The extracted statusCode in variable statusCode and statusLine in variable status. Extracted fields: name, url, clientId, clientSecret | |

NOTE: "Please do not delete any file in the src folder. But you are free to add any other file".

Expectations:

1) **Learners should write automation scripts using Java and REST Assured to automate the API testing for all the provided methods (e.g., GET, POST, PUT, DELETE).** In other words, the automation script should perform all mentioned API interactions, including validation of responses.

2) **Learners should not use any pre-built libraries or tools to validate API responses (e.g., JSON schema validation tools).** They should manually validate the response content (e.g., status codes, response body, etc.) by writing their own logic for assertion.

IMPLEMENTATION/FUNCTIONAL REQUIREMENT

1.1   CODE QUALITY/OPTIMIZATIONS

1. Associates should have written clean code that is readable.
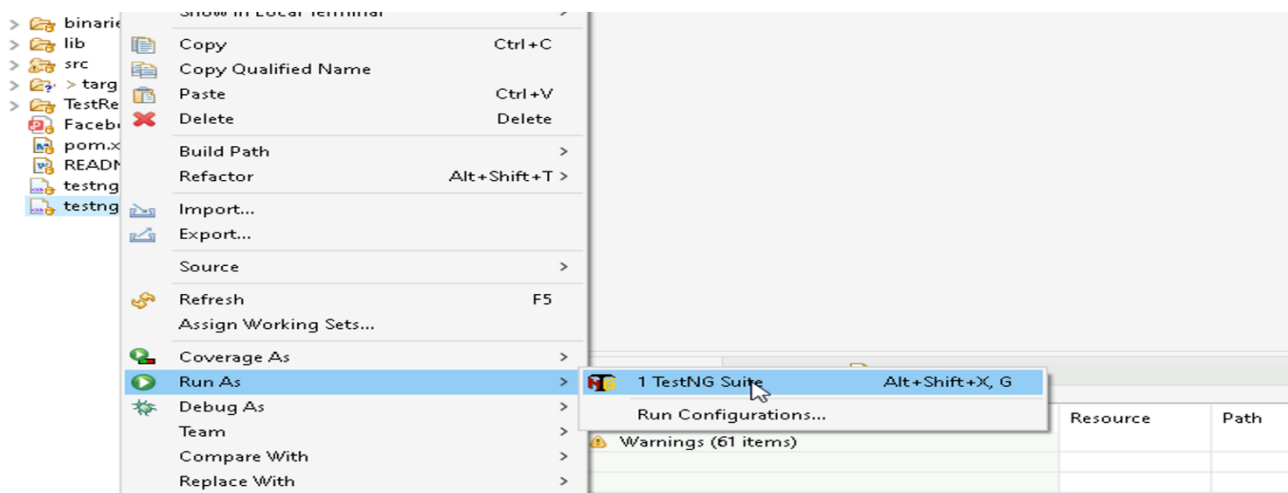2. Associates need to follow SOLID programming principles.

EXECUTION STEPS TO FOLLOW

---

1. **You are mandatory required to run test cases for applications before final submission. Without this project evaluation will not happen.**

2. **You can launch test cases any time as follows: Right-click on testng.xml and run TestNGSuite.**

NOTE:

- **When executing test cases via TestNG.xml, it launches the application UI using Selenium WebDriver. This is intentional behavior to facilitate cookie extraction for authentication.**

3. To do the final submission of the assessment :
   a. Press escape to come out of Fullscreen mode.
   b. Submit the assessment.
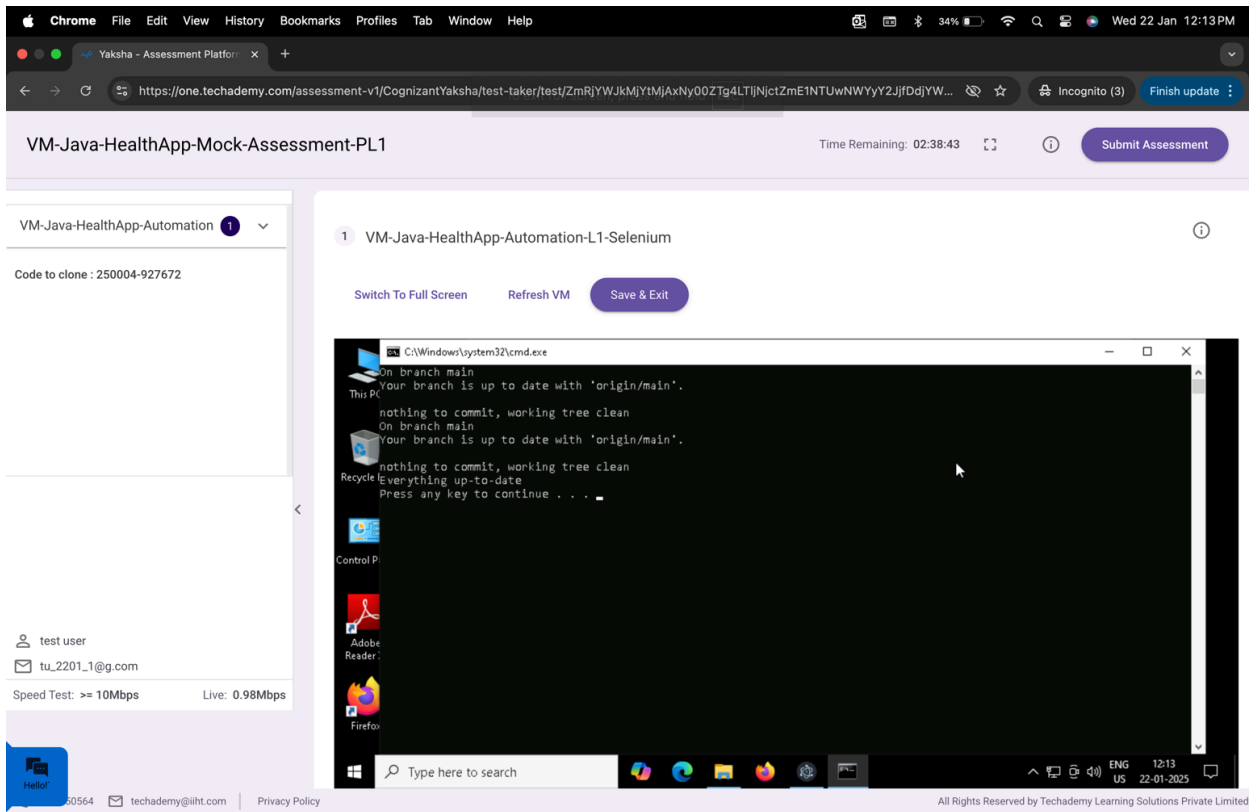
After the successful submission of the assessment, you will get a confirmation message displayed on your screen.

===

All the Best