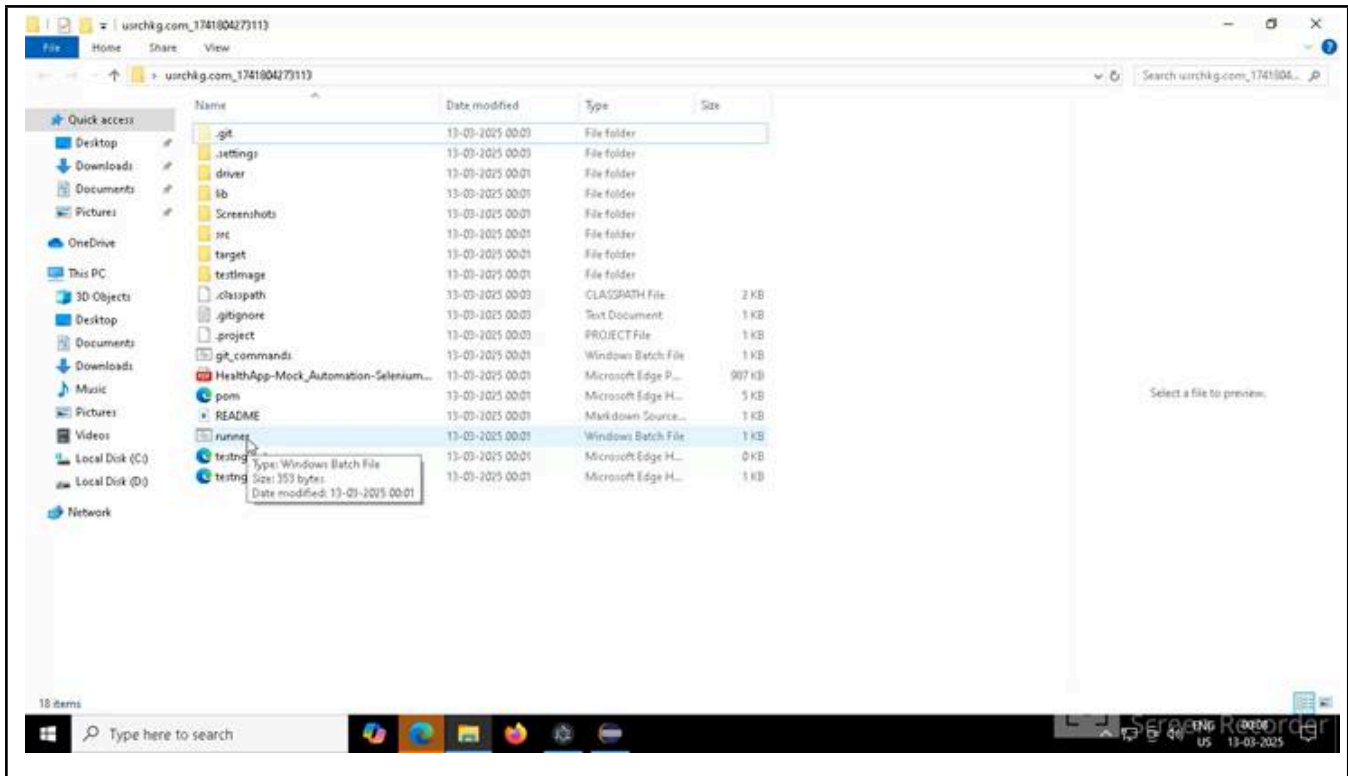# REST-ASSURED
(API - AUTOMATION PROJECT – PL1-4)

Pre-requisite:

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


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.
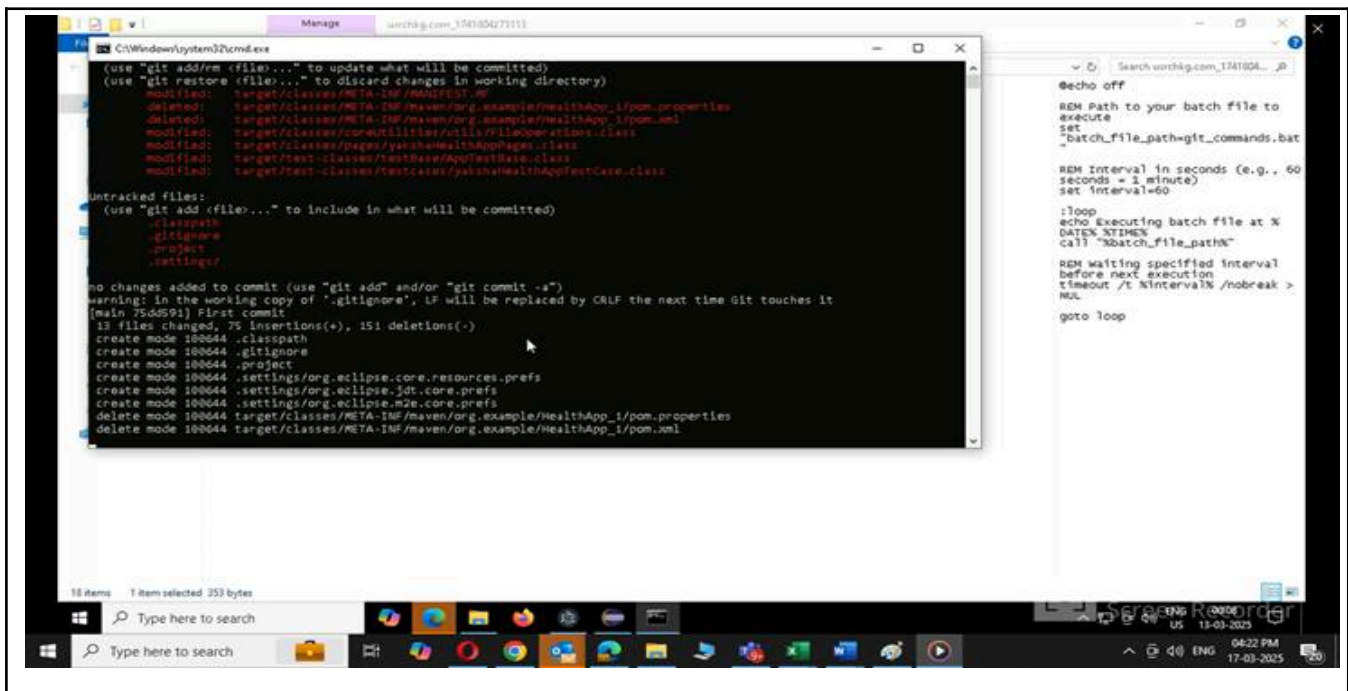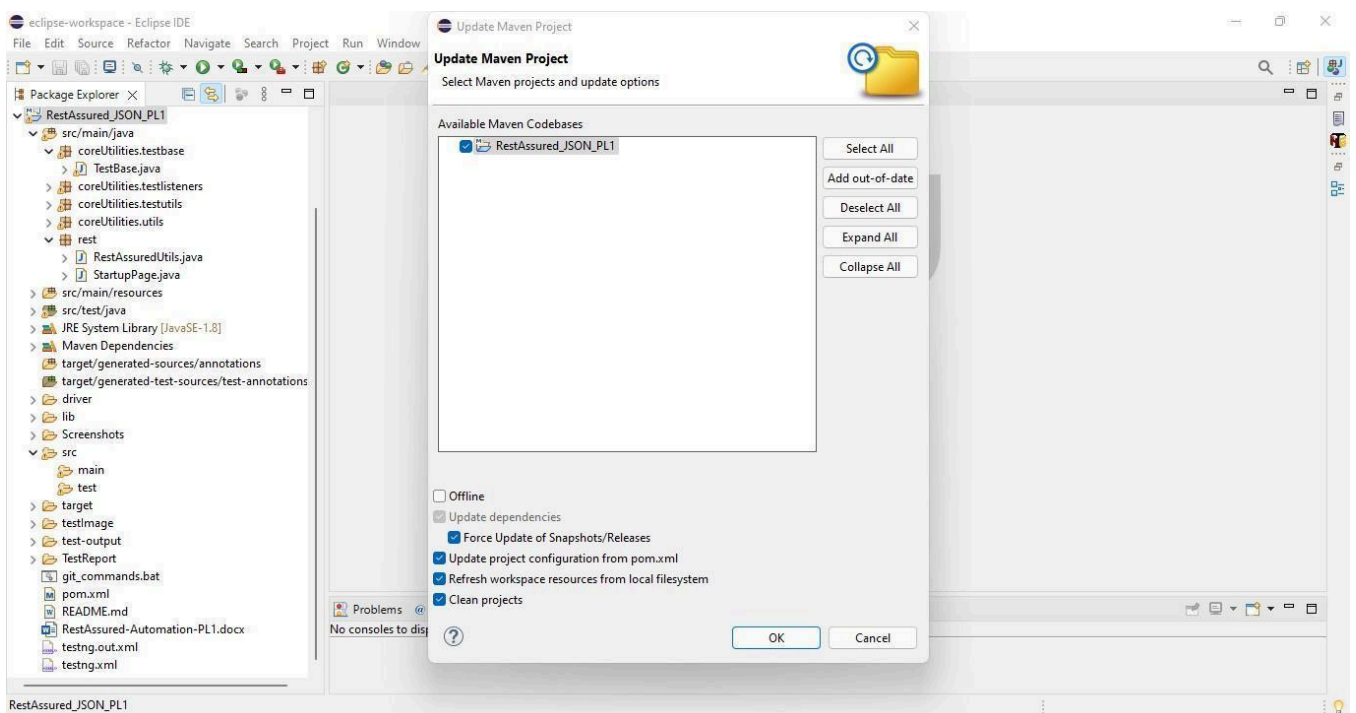
As soon as you import a project in eclipse, update the project using the maven update option as below. This is to resolve issue if any maven dependency not downloaded properly:

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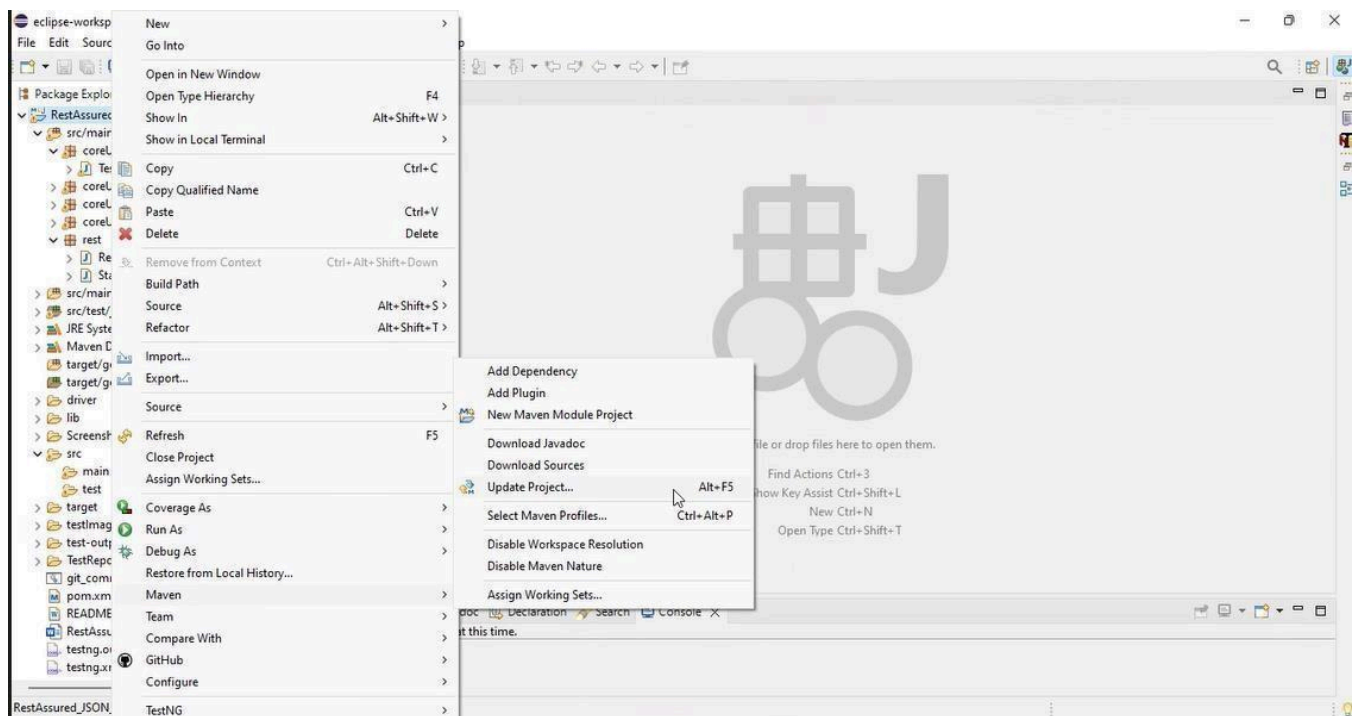
2. In Update Maven Project Box Select "Force Update of Snapshots/Releases" and click OK

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. You can create additional supportive common methods in 7. 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	Perform login with Cookie Authentication using method: **GetLogin(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/auth/login 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. Create a CustomResponse object with: a. Full Response b. Extracted statusCode c. Extracted statusLine 5. Return the CustomResponse object.	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. The complete raw Response object
2	Retrieve action summary using session cookie authentication through method: **GetEmpActionSummary(String endpoint, String cookieValue, Map<String, String> body)**	1. Construct the complete URL by concatenating the BASE_URL and the provided endpoint: a. /web/index.php/api/v2/dashboard/employees/action-summary 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.	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. The complete raw Response object

		5. Create and return a CustomResponse object initialized with: 　a. The complete Response 　b. Extracted statusCode 　c. Extracted statusLine	
3	Retrieve dashboard shortcut access permissions using session-based authentication through the method: **GetDashboardShortcut(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. Extract the following fields and store them in corresponding variables as a Boolean: 　● `leave.assign_leave` to be extracted and saved in **leaveAssignLeave** variable. 　● `leave.leave_list` to be extracted and saved in **leaveLeaveList** variable. 　● `leave.apply_leave` to be extracted and saved in **leaveApplyLeave** variable. 　● `leave.my_leave` to be extracted and saved in **leaveMyLeave** variable. 　● `time.employee_timesheet` to be extracted and saved in **timeEmployeeTimesheet** variable. 　● `time.my_timesheet` to be extracted and saved in **timeMyTimesheet** 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 boolean for the following keys: 　　➔ leaveAssignLeave 　　➔ leavLeaveList 　　➔ leaveApplyLeave 　　➔ leaveMyLeave 　　➔ timeEmployeeTimesheet 　　➔ timeMyTimesheet 　● The complete Response object from the API call

		5. Create a CustomResponse object using: - The complete Response object - Extracted statusCode and statusLine - All six Boolean variables: → leaveAssignLeave → leavLeaveList → leaveApplyLeave → leaveMyLeave → timeEmployeeTimesheet → timeMyTimesheet 6. Return the populated CustomResponse object.	
4	Retrieve employee leaves information using session-based authentication through method: **GetEmpLeaveInfo(String endpoint, String cookieValue, Map<String, String> body)**	1. Construct the final URL by combining the BASE_URL and the provided endpoint: o /web/index.php/api/v2/dashboard/employees/leaves?date={currentDate} 2. Create a RequestSpecification using RestAssured: o Add a session cookie named "orangehrm" using the provided cookieValue. o 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. Create a CustomResponse object using: ● The complete Response object ● Extracted statusCode and statusLine 5. Return the populated CustomResponse object.	● Method must return a valid CustomResponse object containing: ● statusCode = 200 ● status like "HTTP/1.1 200 OK" ● Non-null and non-empty lists of leaves o The full raw Response object

| 5 | Retrieve employee subunit using session-based cookie authentication via method: **GetEmpSubunit(String endpoint, String cookieValue, Map<String, String> body)** | 1. Construct the final URL by combining the BASE_URL and the provided endpoint:
  o /web/index.php/api/v2/dashboard/employees/subunit
2. Create a RequestSpecification using RestAssured:
  o Add a session cookie named "orangehrm" using the provided cookieValue.
  o 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 response using `JsonPath` and retrieve the `data` object as a `List<Map<String, Object>>`.
5. From this list, access the **first element** and then extract the nested `subunit` object (as a `Map<String, Object>`).
6. From this `subunit` object and the first element, extract and store the following values:
  • `id` (from subunit) to be extracted and saved in **subUnitId** variable.
  • `name` (from subunit) to be extracted and saved in **subUnitName** variable.
  • `count` (from the first element of data list) to be extracted and saved in **subUnitCount** variable.
7. Create a CustomResponse object using:
  • The complete Response object
  • Extracted statusCode and statusLine | • The method must return a `CustomResponse` containing:
  • `statusCode = 200`
  • A valid `status` line like "`HTTP/1.1 200 OK`"
  • An object with non-null and non-empty for:
    ▪ subUnitId
    ▪ subUnitName
    ▪ subUnitCount
• The full raw `Response` object |

		● Extracted subUnitId, subUnitName, and subUnitCount values 8. Return the populated CustomResponse object.	

6	Update an employee name using session-based authentication via method: **PutEmpName(String endpoint, String cookieValue, Object body)**	1. Construct the complete URL by appending the endpoint to the base URL:
		
		`https://opensource-demo.ora ngehrmlive.com/web/index.ph p/api/v2/admin/pay-grades/{ id}`
		
		2. Initialize a **RequestSpecification** using **R estAssured**:
		
		   ● Add the session cookie `"orangehrm"` with the provided `cookieValue`.
		
		   ● Set the header `"Content-Type"` t o `"application/json"`.
		
		3. Include body in the request using `.body()`.
		
		4. Trigger a **PUT** request to the constructed URL and extract the **Response**.
		
		5. Parse the response using `JsonPath` and retrieve the `data` object as a `Map<String, Object>`.
		
		6. From this map, extract and wrap values into lists as follows:
		
		   ● `id` to be extracted and saved in **idList** (as a `List<Integer>` containing one element).
		
		   ● `name` to be extracted and saved in **nameList** (as a `List<String>` containing one element).
			● 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:
			
			      ▪ `idList`
			
			      ▪ `nameList`
			
			      ▪ `currencyDetails`
			
			   ● The complete raw `Response` object

		7. From the `currencies` list inside the `data` object:
		
		    ● For each currency, extract `name` and `id`, and save them in **currencyDetails** (as a `List<String>`) as "Currency Name: " + currencyName + ", Currency ID: " + currencyId.
		
		8. Create and return a **CustomResponse** object containing:
		
		    ● The full `Response`
		
		    ● The extracted `statusCode` and `statusLine`
		
		    ● The extracted `idList, nameList, and currencyDetails`	

7	Create employee status using session-based authentication through method: **PostEmpStatus(String endpoint, String cookieValue, String body)**	1. Construct the complete URL by appending the endpoint to the base URL:
		`https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/employment-statuses`
		
		2. Initialize a **RequestSpecification** using **RestAssured**:
		    ● Add the session cookie `"orangehrm"` with the provided `cookieValue`.
		    ● Set the header `"Content-Type"` to `"application/json"`.
		
		3. Include body in the request using `.body()`.
		4. Trigger a **POST** request to the constructed URL and extract the **Response**.
		
		5. Parse the response using `JsonPath` and retrieve the `data`
			● 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:
			
			        ▪ `idList`
			
			        ▪ `nameList`
			
			● The complete raw `Response` object

		object as a `Map<String, Object>`.	
		6. From this map, extract and wrap values into lists as follows:	
		● `id` to be extracted and saved in **idList** (as a `List<Integer>` containing one element).	
		● `name` to be extracted and saved in **nameList** (as a `List<String>` containing one element).	
		7. Create and return a **CustomResponse** object containing:	
		● The full `Response`	
		● The extracted `statusCode` and `statusLine`	
		● The extracted `idList` and `nameList`	

8	Update employee status using session-based authentication through method: **PutEmpStatus(String endpoint, String cookieValue, Object body)**	1. Compose the full URL by appending the endpoint to the base URL:
		
		`https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/employment-statuses/{id}`
		
		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 valid `CustomResponse` containing:
			
			    ● `statusCode = 200`
			
			    ● A valid HTTP `status` (e.g., `"HTTP/1.1 200 OK"`)
			
			    ● Non-empty string values for:
			
			        ▪ `idList`
			
			        ▪ `nameList`
			
			● The complete raw `Response` object

		4. Send a **PUT** request to the full URL and extract the **Response** object.	
		5. Parse the response using `JsonPath` and retrieve the `data` object as a `Map<String, Object>`.	
		6. From this map, extract and wrap values into lists as follows:	
		• `id` to be extracted and saved in **idList** (as a `List<Integer>` containing one element).	
		• `name` to be extracted and saved in **nameList** (as a `List<String>` containing one element).	
		7. Return a **CustomResponse** object with:	
		• The full `Response` object	
		• Extracted `statusCode` and `statusLine`	
		• The extracted `idList` and `nameList`	

9	Delete employee status using session-based authentication through method: **DeleteEmp(String endpoint, String cookieValue, String body)**	1. Compose the full URL by appending the endpoint to the base URL: `https://opensource-demo.orangehrmlive.com/web/index.php/api/v2/admin/employment-statuses/{id}`
		
		2. Initialize a **RequestSpecification** using **RestAssured**:
		    • Add a session cookie named `"orangehrm"` with the provided `cookieValue`
		    • Set the `"Content-Type"` header to `"application/json"`
			• The method should return a valid `CustomResponse` containing:
			    • `statusCode = 200`
			    • A valid HTTP `status` (e.g., `"HTTP/1.1 200 OK"`)
			    • The complete raw `Response` object

		3. Attach the constructed `requestBody` to the request.
		4. Send a **DELETE** request to the full URL and extract the **Response** object.
		5. Return a **CustomResponse** object with:
		   ● The full `Response` object
		
		   ● Extracted `statusCode` and `statusLine`	

10	Create a new OpenID provider entry using a POST request with session-based authentication via method:
	PostEmpName(String endpoint, String cookieValue, String body)
		1. Add the request body while executing the api to get the correct response.
		
		2. Build the full API endpoint URL:
		https://opensource-demo.ora
		ngehrmlive.com/web/index.ph
		p/api/v2/admin/job-categori
		es
		
		3. Initialize a **RequestSpecification** using **RestAssured**:
		   ● Add the "`orangehrm`" session cookie
		
		   ● Set "`Content-Type`" header to "`application/json`"
		
		   ● Attach the constructed `requestBody` using `.body()`
		
		4. Trigger a **POST** request to the endpoint and extract the **Response** object.
		
		5. Parse the response using `JsonPath` and retrieve the `data` object as a `Map<String, Object>`.
		
		6. From this map, extract and wrap values into lists as follows:
		   ● `id` to be extracted and saved in **idList** (as a
			● 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:
			     ▪ `nameList`
			
			     ▪ `idList`
			
			   ● The full `Response` object

| | | | List<Integer> containing one element).

 • name to be extracted and saved in **nameList** (as a List<String> containing one element).
 7. Return a **CustomResponse** object with:

 • The full Response

 • statusCode, statusLine

 • The extracted idList and nameList | |

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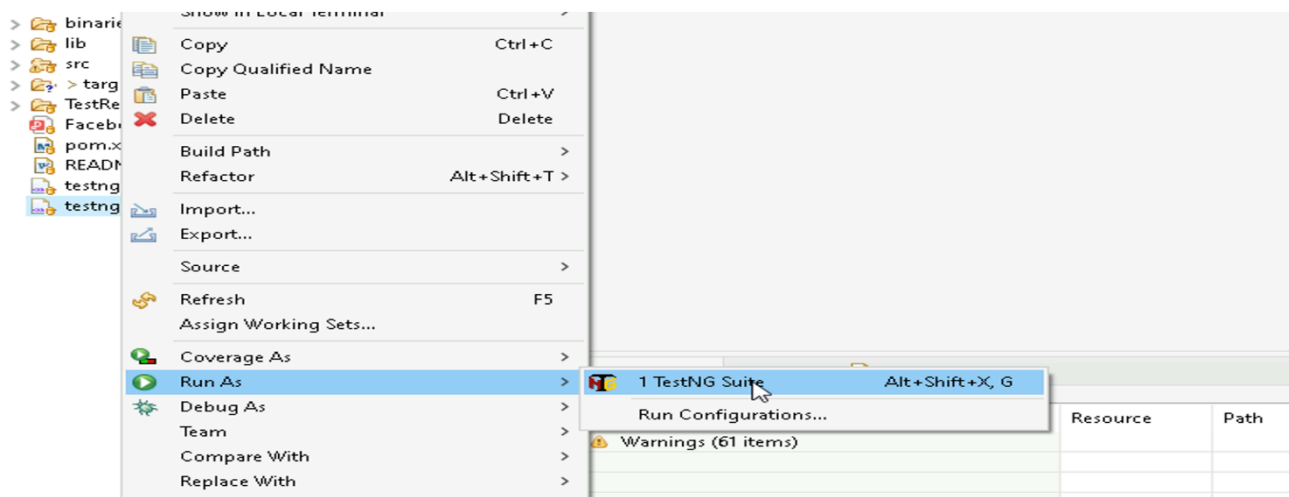
NOTE:
- **When executing test cases via TestNG.xml, it launches the application UI using Selenium WebDriver. This is intentional behavior to facilitate cookie extraction for authentication.**

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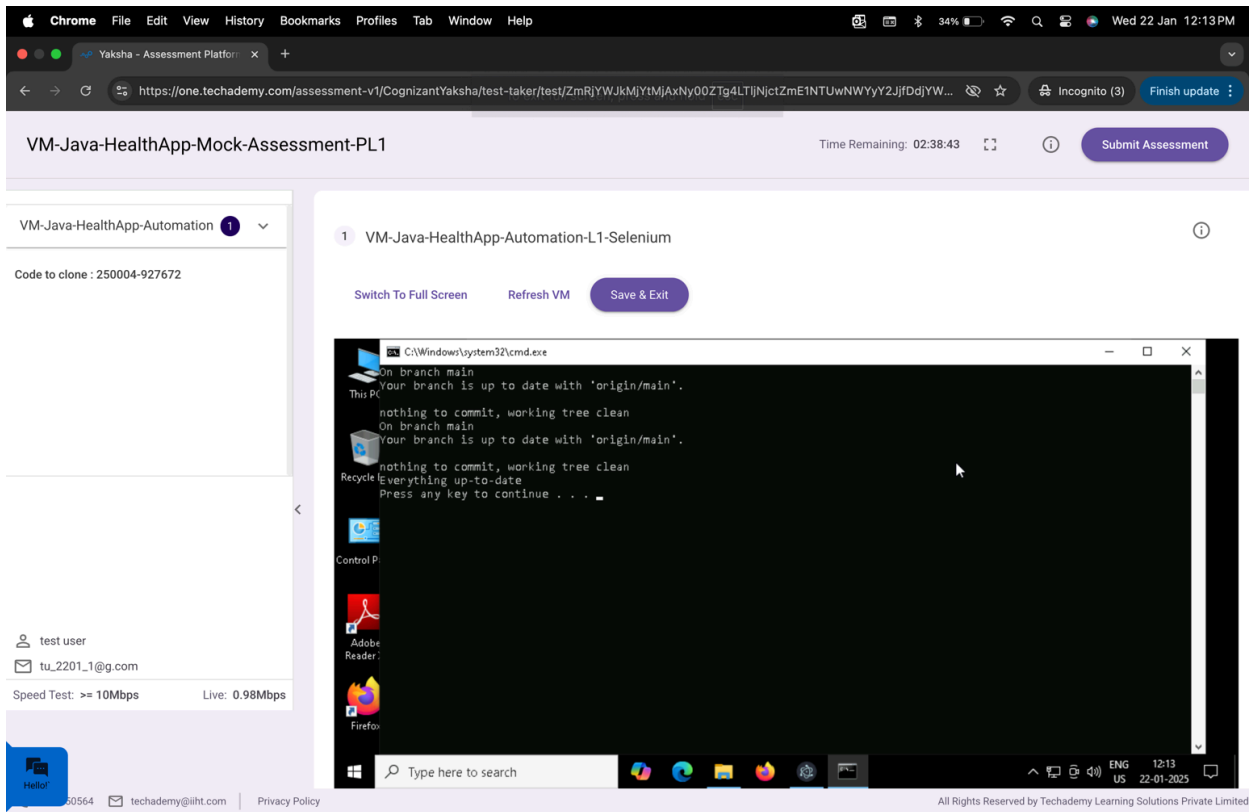
After the successful submission of the assessment, you will get a confirmation message displayed on your screen.

==

All the Best