Candidate Name:MD ATIQUL ISLAM.

Email- atik.cmttiu1001@gmail.com

Qustions of the Assessment:

2nd Assessment:

#Web Automation:

Application URL: https://hishabee.business/

#API testing:

1. go to (https://reqres.in/) and take (CRUD) 5 APIs and automate this API by Karate/Cypress/Robot Framework.

2. go to (https://petstore.swagger.io/#/) and take (CRUD) APIs and validate this API with Postman, write 3 tests by using dynamic Assertion. Generate Newman report.

1.Answer: This is the examples of how you can automate CRUD (Create, Read, Update, Delete) operations for 5 sample APIs from "https://reqres.in/" using the Karate, Cypress, and Robot Frameworks for API testing. Please note that you need to install the respective tools and set up your project before running these scripts.

Using Karate for API Testing:

feature

Feature: CRUD Operations on reqres.in APIs

```
  Background:
    * url 'https://reqres.in/api'

  Scenario: Create a new user
    Given path 'users'
    And request { "name": "John", "job": "Engineer" }
```

```
    When method POST
    Then status 201

Scenario: Read user details
    Given path 'users/2'
    When method GET
    Then status 200
    And match response.name == 'Janet'

Scenario: Update user information
    Given path 'users/2'
    And request { "name": "Janet Doe" }
    When method PUT
    Then status 200
    And match response.name == 'Janet Doe'

Scenario: Delete a user
    Given path 'users/2'
    When method DELETE
    Then status 204

Scenario: Attempt to read a deleted user
    Given path 'users/2'
    When method GET
    Then status 404
```

Using Cypress for API Testing:

javascript

```
describe('CRUD Operations on reqres.in APIs', () => {
    it('Create a new user', () => {
        cy.request({
            method: 'POST',
            url: 'https://reqres.in/api/users',
            body: { name: 'John', job: 'Engineer' },
        }).should((response) => {
            expect(response.status).to.eq(201);
        });
    });

    it('Read user details', () => {
        cy.request('GET', 'https://reqres.in/api/users/2').should((response) => {
            expect(response.status).to.eq(200);
            expect(response.body.data.name).to.eq('Janet');
        });
    });

    it('Update user information', () => {
        cy.request({
            method: 'PUT',
            url: 'https://reqres.in/api/users/2',
```

```
        body: { name: 'Janet Doe' },
      }).should((response) => {
        expect(response.status).to.eq(200);
        expect(response.body.name).to.eq('Janet Doe');
      });
  });

  it('Delete a user', () => {
    cy.request('DELETE', 'https://reqres.in/api/users/2').should((response) => {
      expect(response.status).to.eq(204);
    });
  });

  it('Attempt to read a deleted user', () => {
    cy.request('GET', 'https://reqres.in/api/users/2').should((response) => {
      expect(response.status).to.eq(404);
    });
  });
});
```

Using Robot Framework for API Testing (with Requests Library):

```
robotframework
*** Settings ***
Library         RequestsLibrary
```

*** Test Cases ***

Create a new user

    ${headers}   Create Dictionary   Content-Type=application/json

    ${data}   Create Dictionary   name=John   job=Engineer

    ${response}   Post Request   https://reqres.in/api/users   data=${data} headers=${headers}

    Should Be Equal As Integers ${response.status_code}   201

Read user details

    ${response}   Get Request   https://reqres.in/api/users/2

    Should Be Equal As Integers ${response.status_code}   200

    ${name}   Set Variable ${response.json().data.name}

    Should Be Equal ${name}   Janet

Update user information

    ${headers}   Create Dictionary   Content-Type=application/json

    ${data}   Create Dictionary   name=Janet Doe

    ${response}   Put Request   https://reqres.in/api/users/2   data=${data} headers=${headers}

    Should Be Equal As Integers ${response.status_code}   200

    ${name}   Set Variable ${response.json().name}

    Should Be Equal ${name}   Janet Doe

Delete a user

    ${response}   Delete Request   https://reqres.in/api/users/2

    Should Be Equal As Integers ${response.status_code}   204

Attempt to read a deleted user

${response} Get Request https://reqres.in/api/users/2

Should Be Equal As Integers ${response.status_code} 404

2.Answer: This is a general outline of how to perform API testing on the Swagger Petstore API using Postman, including writing three tests with dynamic assertions and generating a Newman report. Please note that you'll need to adapt these steps according to the specific APIs and test scenarios you want to validate.

Step 1: Set Up Postman Environment

1. Open Postman and create a new collection named "Petstore API CRUD Tests."

2. Inside the collection, create a new environment (e.g., "Petstore Environment") and add variables for the base URL of the Petstore API and any other variables you might need.

Step 2: Create API Requests

In the collection, create API requests for CRUD operations (Create, Read, Update, Delete) on the Petstore API. Here's an example of how to create a "Create a pet" request:

Request 1: Create a pet

- Request Type: POST

- URL: `{{petstore_base_url}}/pet`

- Body:

```
json
{
    "id": 1,
    "category": {
        "id": 0,
        "name": "string"
    },
    "name": "doggie",
    "photoUrls": [
        "string"
    ],
    "tags": [
        {
            "id": 0,
            "name": "string"
        }
    ],
    "status": "available"
}
```

Step 3: Write Dynamic Assertions

For each API request, write dynamic assertions in the "Tests" tab to validate the response. Here are three examples:

Test 1: Verify Status Code

javascript

```
pm.test("Status code is 200 OK", function () {
    pm.response.to.have.status(200);
});
```

Test 2: Verify Response Contains Correct Pet Name

javascript

```
var jsonData = pm.response.json();
pm.test("Response contains the correct pet name", function () {
    pm.expect(jsonData.name).to.eql("doggie");
});
```

Test 3: Verify Response Contains ID

javascript

```
var jsonData = pm.response.json();
pm.test("Response contains an ID", function () {
    pm.expect(jsonData.id).to.be.a('number');
});
```

Step 4: Run Collection in Postman

- Click the "Runner" button in Postman.
- Select your collection and environment.
- Click "Start Run" to execute the requests.

Step 5: Generate Newman Report

1. Open your command prompt or terminal.

2. Install Newman globally if you haven't already:

```
bash
npm install -g newman
```

3. Run Newman with the following command:

```
bash
newman run "Petstore API CRUD Tests.postman_collection.json" -e "Petstore Environment.postman_environment.json" -r html
```

Replace `"Petstore API CRUD Tests.postman_collection.json"` and `"Petstore Environment.postman_environment.json"` with the actual filenames of your collection and environment files.

4. Newman will execute the collection, and an HTML report will be generated.

You can open the HTML report in your browser to view the test results, including the dynamic assertions.