**QA( Manual+Automation) Test - Nimap task**

**Automation tester-**

**Explanation:**

* **Parameterization:** Use TestNG's @DataProvider or external configuration (like a properties file) to supply different login credentials.
* **Steps:**
  1. Open the browser and navigate to [TestFFC](https://testffc.nimapinfotech.com/).
  2. Locate the username and password fields.
  3. Input the credentials from your data provider.
  4. Click the login button.
  5. Validate successful login (e.g., by checking that a certain element or URL is present).

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class LoginTest {

WebDriver driver;

@BeforeClass

public void setUp() {

// Set your WebDriver executable path if needed

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\Kanchan\\OneDrive\\Documents\\chromeUpdate\\chromedriver-win64 (1)\\chromedriver-win64\\chromedriver.exe");

WebDriver driver=**new** ChromeDriver();

driver.manage().window().maximize();

}

@DataProvider(name = "loginData")

public Object[][] loginData() {

return new Object[][] {

{"username1", "password1"},

{"username2", "password2"}

// Add more test data as needed

};

}

@Test(dataProvider = "loginData")

public void testLogin(String username, String password) {

driver.get("https://testffc.nimapinfotech.com/");

// Locate and fill in the login fields

driver.findElement(By.id("usernameField")).sendKeys(“kanchan”);

driver.findElement(By.id("passwordField")).sendKeys(“kanchan@1234”);

driver.findElement(By.id("loginButton")).click();

// Validate login by asserting an element's text, URL, or presence of a logout button

String expectedUrl = "https://testffc.nimapinfotech.com/home";

Assert.assertEquals(driver.getCurrentUrl(), expectedUrl, "Login failed for user: " + username);

}

@AfterClass

public void tearDown() {

driver.quit();

}

}

**2)Verify the Toast/Popup Message after PunchIn**

**Explanation:**

* **Toast/Popup Verification:** After performing the "PunchIn" action, a toast or popup message appears. Use explicit waits (e.g., WebDriverWait) to capture the message.
* **Steps:**
  1. Trigger the PunchIn action (click on the PunchIn button).
  2. Wait for the toast/popup element to become visible.
  3. Validate that the text matches the expected message.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.Assert;

import org.testng.annotations.Test;

public class PunchInTest extends LoginTest {

@Test(dependsOnMethods = "testLogin")

public void testPunchInToastMessage() {

// Assume user is already logged in from previous test

// Click on the PunchIn button

driver.findElement(By.id("punchInButton")).click();

// Wait for the toast/pop-up message to appear

WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement toastElement = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("toastMessage")));

String actualMessage = toastElement.getText();

String expectedMessage = "PunchIn Successful";

Assert.assertEquals(actualMessage, expectedMessage, "Toast message did not match!");

}

}

**3)Add Customer with Parameterization**

**Explanation:**

* **Customer Addition:** Use parameterized data (again via TestNG's @DataProvider) for customer details.
* **Steps:**
  1. Navigate to the "Add Customer" page/form.
  2. Fill in the customer details from your data provider.
  3. Submit the form.
  4. Validate that the customer has been added (e.g., by checking for a confirmation message or the presence of the customer in a list).

public class AddCustomerTest extends LoginTest {

@DataProvider(name = "customerData")

public Object[][] customerData() {

return new Object[][] {

{"John", "Doe", "john.doe@example.com", "1234567890"},

{"Jane", "Smith", "jane.smith@example.com", "0987654321"}

// More customer data can be added here

};

}

@Test(dataProvider = "customerData", dependsOnMethods = "testLogin")

public void testAddCustomer(String firstName, String lastName, String email, String phone) {

// Navigate to the Add Customer page

driver.findElement(By.id("addCustomerNav")).click();

// Fill in the customer form

driver.findElement(By.id("firstName")).sendKeys(“John”);

driver.findElement(By.id("lastName")).sendKeys(“Doe");

driver.findElement(By.id("email")).sendKeys("john.doe@example.com"”);

driver.findElement(By.id("phone")).sendKeys(“1234567890

”);

// Submit the form

driver.findElement(By.id("submitCustomer")).click();

// Validate the addition of the customer (e.g., check for a confirmation message)

WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement confirmationMessage = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("confirmationMsg")));

String expectedConfirmation = "Customer added successfully";

Assert.assertEquals(confirmationMessage.getText(), expectedConfirmation, "Customer was not added as expected!");

}

}

**(Optional) Implementing with Cucumber**

Feature: Automation Tester Task for TestFFC

Scenario Outline: Login and PunchIn

Given I navigate to "https://testffc.nimapinfotech.com/"

When I enter username "<username>" and password "<password>"

And I click the login button

Then I should be logged in successfully

When I click the PunchIn button

Then I should see a toast message "PunchIn Successful"

Examples:

| username | password |

| user1 | pass1 |

| user2 | pass2 |

Scenario Outline: Add Customer

Given I am logged in

When I navigate to the Add Customer page

And I enter customer details: "<firstName>", "<lastName>", "<email>", "<phone>"

And I submit the customer form

Then I should see a confirmation message "Customer added successfully"

Examples:

| firstName | lastName | email | phone |

| John | Doe | john.doe@example.com | 1234567890 |

| Jane | Smith | jane.smith@example.com | 0987654321 |

b. Implement Step Definitions (e.g., AutomationSteps.java):

import io.cucumber.java.en.\*;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.Assert;

public class AutomationSteps {

WebDriver driver;

WebDriverWait wait;

@Given("I navigate to {string}")

public void i\_navigate\_to(String url) {

System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

driver = new ChromeDriver();

driver.manage().window().maximize();

driver.get(url);

wait = new WebDriverWait(driver, 10);

}

@When("I enter username {string} and password {string}")

public void i\_enter\_username\_and\_password(String username, String password) {

driver.findElement(By.id("usernameField")).sendKeys(username);

driver.findElement(By.id("passwordField")).sendKeys(password);

}

@When("I click the login button")

public void i\_click\_the\_login\_button() {

driver.findElement(By.id("loginButton")).click();

}

@Then("I should be logged in successfully")

public void i\_should\_be\_logged\_in\_successfully() {

// Validate successful login, e.g., by checking the URL or a specific element

String expectedUrl = "https://testffc.nimapinfotech.com/home";

Assert.assertEquals(driver.getCurrentUrl(), expectedUrl);

}

@When("I click the PunchIn button")

public void i\_click\_the\_PunchIn\_button() {

driver.findElement(By.id("punchInButton")).click();

}

@Then("I should see a toast message {string}")

public void i\_should\_see\_a\_toast\_message(String expectedMessage) {

String actualMessage = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("toastMessage"))).getText();

Assert.assertEquals(actualMessage, expectedMessage);

}

@Given("I am logged in")

public void i\_am\_logged\_in() {

// Assuming that the login steps have been executed, or implement login steps here

}

@When("I navigate to the Add Customer page")

public void i\_navigate\_to\_the\_Add\_Customer\_page() {

driver.findElement(By.id("addCustomerNav")).click();

}

@When("I enter customer details: {string}, {string}, {string}, {string}")

public void i\_enter\_customer\_details(String firstName, String lastName, String email, String phone) {

driver.findElement(By.id("firstName")).sendKeys(firstName);

driver.findElement(By.id("lastName")).sendKeys(lastName);

driver.findElement(By.id("email")).sendKeys(email);

driver.findElement(By.id("phone")).sendKeys(phone);

}

@When("I submit the customer form")

public void i\_submit\_the\_customer\_form() {

driver.findElement(By.id("submitCustomer")).click();

}

@Then("I should see a confirmation message {string}")

public void i\_should\_see\_a\_confirmation\_message(String expectedConfirmation) {

String actualConfirmation = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("confirmationMsg"))).getText();

Assert.assertEquals(actualConfirmation, expectedConfirmation);

driver.quit();

}

}

c. Runner Class (Using TestNG with Cucuimport io.cucumber.testng.AbstractTestNGCucumberTests;

import io.cucumber.testng.CucumberOptions;

@CucumberOptions(

features = "src/test/resources/features", // location of feature files

glue = {"stepDefinitions"}, // package containing step definitions

plugin = {"pretty", "html:target/cucumber-reports.html"}

)

public class TestRunner extends AbstractTestNGCucumberTests {

}

**Create a Request for Login (Valid Credentials):**

1. **Method:** POST
2. **URL:** {{baseUrl}}/api/login
3. **Headers:**
   * Content-Type: application/json

{

"username": "{{username}}",

"password": "{{password}}"

}

pm.test("Status code is 200", function () {

pm.response.to.have.status(200);

});

// Capture auth token from the response if available

let jsonData = pm.response.json();

if(jsonData.token){

pm.environment.set("authToken", jsonData.token);

}

**Create a Request for Login (Invalid Credentials):**

1. **Method:** POST
2. **URL:** {{baseUrl}}/api/login
3. **Headers:**
   * Content-Type: application/json

{

"username": "invalid@example.com",

"password": "wrongpassword"

}

**Tests Script (Optional):** Validate error response:

pm.test("Status code is 401 or appropriate error code", function () {

pm.response.to.have.status(401);

});

 **Method:** POST

 **URL:** {{baseUrl}}/api/customers

 **Headers:**

* Content-Type: application/json
* Authorization: Bearer {{authToken}} *(if your API uses Bearer token for authentication)*

 **Body (raw JSON):**

{

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com",

"phone": "1234567890"

}

**Tests Script (Optional):** Validate that the customer was added successfully:

pm.test("Status code is 201", function () {

pm.response.to.have.status(201);

});

let responseData = pm.response.json();

pm.test("Customer added successfully", function () {

pm.expect(responseData.message).to.eql("Customer added successfully");

});

**1. Sign Up Module**

**Test Cases**

**TC\_SU\_01: Verify Mandatory Fields Validation**

* **Description:** Ensure that all required fields are marked and cannot be left blank.
* **Preconditions:** User is on the Sign Up page.
* **Test Steps:**
  1. Navigate to the Sign Up page.
  2. Leave all mandatory fields empty.
  3. Click the “Sign Up” button.
* **Expected Result:**
  1. Error messages are displayed for each mandatory field (e.g., “Email is required”, “Password is required”).
* **Field Validations:**
  1. **Email:** Must be in proper format (e.g., user@example.com).
  2. **Password:** Minimum length (e.g., 8 characters), must include at least one uppercase letter, one lowercase letter, one digit, and one special character.
  3. **Phone Number:** Only digits; length validation (e.g., 10 digits).
  4. **Username/Full Name:** Must not be empty; can include alphabet characters.
* **Potential Bugs:**
  1. Missing validation messages.
  2. Accepting invalid email formats.
  3. Weak password accepted without proper strength validation.

**TC\_SU\_02: Verify Sign Up with Valid Data**

* **Description:** Validate successful account creation with valid inputs.
* **Preconditions:** User is on the Sign Up page.
* **Test Steps:**
  1. Fill in valid details for all mandatory fields.
  2. Click the “Sign Up” button.
* **Expected Result:**
  1. User is successfully registered and navigated to the appropriate confirmation/next step page.
* **Potential Bugs:**
  1. Registration failure even with valid data.
  2. No confirmation message or redirection.

**TC\_SU\_03: Verify Error on Duplicate Email/Username**

* **Description:** Ensure that the system does not allow registration using an already existing email or username.
* **Preconditions:** An account already exists with the test email/username.
* **Test Steps:**
  1. Enter an existing email/username along with valid data for other fields.
  2. Click “Sign Up”.
* **Expected Result:**
  1. Error message like “Email already registered” or “Username already taken” is displayed.
* **Potential Bugs:**
  1. Duplicate registrations allowed.
  2. Unclear or no error messaging.

**2. Forgot Password Module**

**Test Cases**

**TC\_FP\_01: Verify Forgot Password Link Navigation**

* **Description:** Ensure that clicking on the “Forgot Password” link navigates to the appropriate reset page.
* **Preconditions:** User is on the Login page.
* **Test Steps:**
  1. Click on the “Forgot Password” link.
* **Expected Result:**
  1. User is redirected to the Forgot Password page.
* **Potential Bugs:**
  1. Incorrect navigation or broken link.

**TC\_FP\_02: Validate Mandatory Field (Email) on Forgot Password Page**

* **Description:** Ensure that the email field cannot be left empty.
* **Preconditions:** User is on the Forgot Password page.
* **Test Steps:**
  1. Leave the email field blank.
  2. Click the “Submit” button.
* **Expected Result:**
  1. Display error message like “Email is required”.
* **Field Validations:**
  1. **Email:** Must be in a valid format.
* **Potential Bugs:**
  1. No error message when email is blank.
  2. Invalid email formats accepted.

**TC\_FP\_03: Verify Forgot Password with Valid Email**

* **Description:** Ensure that the system sends a password reset email when a valid email is submitted.
* **Preconditions:** The email provided is registered.
* **Test Steps:**
  1. Enter a valid, registered email address.
  2. Click the “Submit” button.
* **Expected Result:**
  1. A message confirms that a reset link has been sent.
* **Potential Bugs:**
  1. No confirmation message.
  2. Password reset email not received.

**3. Sign with OTP Module**

**Test Cases**

**TC\_OTP\_01: Verify OTP Request**

* **Description:** Validate that the user can request an OTP.
* **Preconditions:** User has initiated a sign-in or sign-up process where OTP is required.
* **Test Steps:**
  1. Enter the required details (e.g., mobile number or email).
  2. Click the “Send OTP” button.
* **Expected Result:**
  1. An OTP is sent, and a message is displayed (e.g., “OTP sent successfully”).
* **Potential Bugs:**
  1. OTP not sent or incorrect messaging.
  2. Delay in receiving OTP beyond acceptable time.

**TC\_OTP\_02: Validate OTP Input Field**

* **Description:** Check the OTP field for valid input.
* **Preconditions:** OTP has been requested.
* **Test Steps:**
  1. Enter a valid OTP.
  2. Enter an invalid or incomplete OTP.
* **Expected Result:**
  1. For valid OTP, the process proceeds to the next step.
  2. For invalid/incomplete OTP, an error message (e.g., “Invalid OTP”) is displayed.
* **Field Validations:**
  1. **OTP:** Numeric only; fixed length (e.g., 4-6 digits depending on implementation).
* **Potential Bugs:**
  1. Accepting invalid or incomplete OTP.
  2. No error message for incorrect OTP.

**4. Login Module**

**Test Cases**

**TC\_LG\_01: Verify Login with Valid Credentials**

* **Description:** Validate that a user can successfully log in with valid credentials.
* **Preconditions:** User is on the Login page; registered account exists.
* **Test Steps:**
  1. Enter valid email/username.
  2. Enter the correct password.
  3. Click the “Login” button.
* **Expected Result:**
  1. User is successfully logged in and navigated to the dashboard/home page.
* **Potential Bugs:**
  1. Incorrect redirection after login.
  2. Failure message despite valid credentials.

**TC\_LG\_02: Verify Login with Invalid Credentials**

* **Description:** Validate that the system displays appropriate error messages for invalid credentials.
* **Preconditions:** User is on the Login page.
* **Test Steps:**
  1. Enter an invalid or unregistered email/username.
  2. Enter an incorrect password.
  3. Click “Login.”
* **Expected Result:**
  1. Display error message like “Invalid username or password.”
* **Potential Bugs:**
  1. System allows login with invalid credentials.
  2. Vague or no error messaging provided.

**TC\_LG\_03: Validate Field-Level Input on Login**

* **Description:** Ensure proper field validations (e.g., email format) on the Login page.
* **Preconditions:** User is on the Login page.
* **Test Steps:**
  1. Enter an invalid email format (e.g., “user@”).
  2. Click “Login.”
* **Expected Result:**
  1. Error message indicates invalid email format.
* **Field Validations:**
  1. **Email/Username:** Should be in a valid email format if an email is required.
  2. **Password:** Should not be empty.
* **Potential Bugs:**
  1. Accepting an invalid email format.
  2. No validation message for empty fields.