**TASK 2 DOCUMENTATION**

**Automation Testing:**

Automation testing is a software testing technique that involves using specialized tools and scripts to automate the execution of test cases and compare the actual outcomes with expected results. It aims to increase the efficiency, effectiveness, and coverage of software testing by minimizing manual intervention. In automation testing, test scripts are created to simulate user interactions, validate software functionality, and identify defects or deviations from expected behavior. This approach is particularly useful for repetitive and time-consuming testing tasks.

**Importance of Automation Testing:**

* Efficiency: Automation allows tests to be executed faster and more consistently than manual testing, reducing the overall testing time and accelerating the development cycle.
* Repeatability: Automated tests can be repeated as often as needed without errors, ensuring consistent and reliable results, which is challenging in manual testing due to human errors and fatigue.
* Coverage: Automation enables broader test coverage by executing a large number of test cases in a short span of time, addressing a wide range of scenarios and reducing the risk of overlooking potential issues.
* Regression Testing: With frequent software updates, regression testing becomes crucial to ensure that new changes haven't broken existing functionality. Automation allows quick execution of regression tests, catching regressions early in the development process.
* Cost Savings: While there's an initial investment in creating and maintaining automated test scripts, the long-term benefits include reduced manual testing efforts, decreased chances of defects escaping into production, and overall cost savings.
* Accuracy: Automated tests execute predefined steps accurately and consistently, minimizing human error and increasing the accuracy of testing.
* Parallel Execution: Automation tools can run tests in parallel across various environments, configurations, and devices, allowing for comprehensive testing in a shorter time frame.
* Early Detection of Defects: Automated tests can be integrated into the continuous integration/continuous delivery (CI/CD) pipeline, catching defects early in the development lifecycle.

**Some Automation Testing Tools:**

* Selenium: A widely-used open-source tool for automating web applications across different browsers and platforms. It supports various programming languages like Java, Python, C#, etc.
* Cypress: A widely-used open-source tool for automating web applications across different browsers and platforms. It supports programming language JavaScript.
* Appium: An open-source tool for automating mobile applications on iOS and Android platforms, enabling cross-platform testing.
* JUnit and TestNG: Frameworks for writing and running Java-based test cases, providing features like test parameterization, test grouping, and reporting.

In my test script project I will create test script and execute it using cypress because I am familiar with JavaScript. Let us understand what is cypress

**Cypress:**

Cypress is an open-source end-to-end testing framework specifically designed for modern web applications. It aims to provide a fast, reliable, and developer-friendly environment for writing and executing tests.

**Key Features of Cypress:**

* Real-Time Reloads: Cypress provides real-time previews of your application as you write and modify test scripts.
* Time Travel: Cypress allows you to see what happens at each step of your test in the browser.
* Automatic Waiting: Cypress automatically waits for elements to appear, ensuring that you don't need to add explicit waits or sleep statements in your test scripts.
* Easy Debugging: Cypress provides an interactive browser-based console for debugging tests. Developers can use this console to inspect and interact with the application while tests are running.
* Synchronous Commands: Cypress uses a JavaScript-based scripting language for test writing, allowing you to chain commands synchronously, making the code more readable and intuitive.
* Fast Execution: Cypress is built to execute tests in the same run loop as the application. This architecture results in faster execution times and more reliable test results.
* Automatic Screenshots and Videos: Cypress captures screenshots and records videos of test execution by default, which aids in diagnosing issues.
* Network Traffic Control: Cypress lets you stub and intercept network requests, enabling you to test various scenarios without hitting real APIs or servers.
* Test Runner: Cypress provides a visual test runner that displays test results in real-time, making it easy to track progress and identify failures.

Cypress is particularly suitable for testing modern web applications that heavily rely on JavaScript and dynamic interactions. It's gaining popularity in the developer community due to its ease of use, powerful features, and efficient testing capabilities. Keep in mind that while Cypress is a great choice for web applications, it might not be suitable for testing other types of applications like mobile or desktop applications.

**Test Assertions using Cypress:**

Test assertions are used to verify that certain conditions or expectations are met during the execution of your tests. Cypress provides a rich set of built-in assertion functions that you can use to make assertions about the state of your application.

Here's how you can perform test assertions in Cypress:

**1. Select an Element:**

Before making assertions, you typically need to select an element on the page that you want to assert something about. You can use Cypress's powerful selection methods to target elements by their CSS selectors, attributes, or other properties.

**Syntax:**

cy.get('.button-class').should('be.visible'); // Selects a button element and asserts that it is visible

**2. Using Built-in Assertion Commands:**

Cypress provides a set of assertion commands that you can use to make various types of assertions. Here are a few examples:

Checking text content:

cy.get('.header').should('contain.text', 'Hi'); // Asserts that the element with class 'header' contains the text 'Hi'

Checking attribute values:

cy.get('input[name="username"]').should('have.attr', 'placeholder', 'Enter your username'); // Asserts that the input element's placeholder attribute is 'Enter your username'

Checking visibility:

cy.get('.modal').should('be.visible'); // Asserts that the element with class 'modal' is visible

Checking class names:

cy.get('.button').should('have.class', 'active'); // Asserts that the button element has the class 'active'

**3. Using Chai Assertion Library:**

Cypress uses the Chai assertion library behind the scenes to provide assertion capabilities. This means you can also use Chai's syntax for assertions:

**Syntax**

cy.get('.result').should(($result) => {

expect($result).to.have.text('Success'); // Using Chai assertion to check text content

});

**4. Handling Negations:**

You can use the .not keyword to perform negations in assertions. For example:

**Syntax:**

cy.get('.error-message').should('not.exist'); // Asserts that the element with class 'error-message' does not exist

5. Handling Time-Related Assertions:

Cypress has assertions for dealing with asynchronous operations, such as waiting for an element to appear:

**Syntax:**

cy.get('.loading-spinner').should('not.exist', { timeout: 10000 }); // Asserts that the loading spinner disappears within 10 seconds

These are just a few examples of how you can perform assertions in Cypress. The key is to use the .should() command along with various assertions to validate the expected behavior of your application. Cypress's rich set of assertion capabilities allows you to thoroughly test your application's functionality and user interactions.

**Cypress Login Test Scenarios for OrangeHRM**

Introduction

This documentation provides an overview of the Cypress test automation project that focuses on testing the login functionality of the OrangeHRM website. The project includes six distinct login test cases to validate various scenarios related to the login process.

**Project Structure**

The project follows a standard Cypress project structure, with test scripts located in the "cypress/integration" directory. The project directory structure is as follows:

├── fixtures/

├── e2e/

│ ├── login\_spec.js

├── plugins/

├── support/

└── ...

**Test Scenarios**

Test Case 1: Correct Username and Password

Description: This test case verifies that users can successfully log in with the correct username and password.

Steps:

Navigate to the OrangeHRM login page.

Enter a valid username and password.

Click the "Login" button.

Expected Outcome: The user should be successfully logged in and redirected to the dashboard.

Test Case 2: Incorrect Username and Password

Description: This test case validates the behavior when users provide incorrect credentials.

Steps:

Navigate to the OrangeHRM login page.

Enter an invalid username and password.

Click the "Login" button.

Expected Outcome: An error message should be displayed, and the user should remain on the login page.

Test Case 3: Correct Username with Incorrect Password

Description: This test case checks the scenario where users provide a correct username but an incorrect password.

Steps:

Navigate to the OrangeHRM login page.

Enter a valid username and an incorrect password.

Click the "Login" button.

Expected Outcome: An error message should be displayed, and the user should remain on the login page.

Test Case 4: Incorrect Username with Correct Password

Description: This test case tests the case where users provide an incorrect username but a correct password.

Steps:

Navigate to the OrangeHRM login page.

Enter an incorrect username and a valid password.

Click the "Login" button.

Expected Outcome: An error message should be displayed, and the user should remain on the login page.

Test Case 5: Blank Username and Password

Description: This test case checks the behavior when both the username and password fields are left blank.

Steps:

Navigate to the OrangeHRM login page.

Leave both the username and password fields blank.

Click the "Login" button.

Expected Outcome: An error message should be displayed, and the user should remain on the login page.

Test Case 6: Correct Username with Blank Password

Description: This test case validates the scenario where users enter a valid username but leave the password field blank.

Steps:

Navigate to the OrangeHRM login page.

Enter a valid username and leave the password field blank.

Click the "Login" button.

Expected Outcome: An error message should be displayed, and the user should remain on the login page.

**Running the Tests**

To run the tests, follow these steps:

* Make sure you have Cypress installed (npm install cypress).
* Open Cypress using the command npx cypress open.
* Click on the login\_spec.js file to execute the login test scenarios.

**Conclusion**

The Cypress test automation project demonstrates the comprehensive testing of the OrangeHRM website's login functionality. By covering various login scenarios, I got the benefit to understand cypress very well but the one limitation I got in this week was I want to work with Selenium but because of its Webdriver version differences I cant able to run it in my laptop and it wasted my 2 days, but I had an exciting week working with cypress and excited for the week 3.

**‘Thank You!’**