**Automation testing**

**1.What are the challenges faced in the automation testing**

**Environment Issues**: Automated tests need a stable and consistent environment to execute successfully. Issues such as server downtime, configuration drift, or differences in environments (development, testing, production) can cause test failures. So due to env slowness many automation scripts are tends to fail hence I need to run it again. This was very challenging

**Flaky Tests and Reliability Issues:** Sometimes, automated tests can fail intermittently due to factors like timing issues or environmental instability. To handle this, I use best practices like adding waits, using mock data, and ensuring the test environment is stable and mirrors production as closely as possible.

Dynamic webelement handled by using contains, start with,end with attributes and following, preceding nodes ans also using multiple attributes in a xpath

Synchronization issues handled by explicit wait by giving certain conditions

flaky tests that fail intermittently due to environmental factors is handled by proper exception handling,

**2. What are the types of waits supported by WebDriver?**

**Implicit wait** - Implicit wait commands Selenium to wait for a certain amount of time before throwing an exception.

**General syntax**: driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);

**Example** : driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

**Explicit wait** - Explicit wait is used to tell the Web Driver to wait for certain conditions before throwing an "ElementNotVisibleException" exception.

In order to declare explicit wait, one has to use **ExpectedConditions.** The following [Expected Conditions](https://www.browserstack.com/guide/expectedconditions-in-selenium) can be used in Explicit Wait.  
1. alertIsPresent()  
2.elementSelectionStateToBe()  
3.elementToBeClickable()  
4. elementToBeSelected()  
5.frameToBeAvaliableAndSwitchToIt()  
6.invisibilityOfTheElementLocated()  
7.invisibilityOfElementWithText()  
8.presenceOfAllElementsLocatedBy()  
9.presenceOfElementLocated()  
10.textToBePresentInElement()  
11.textToBePresentInElementLocated()  
12.textToBePresentInElementValue()  
13.titleIs()  
14.titleContains()  
15.visibilityOf()  
16.visibilityOfAllElements()  
17.visibilityOfAllElementsLocatedBy()  
18.visibilityOfElementLocated()

**General syntax**: WebDriverWait wait = new WebDriverWait(WebDriver Reference, TimeOut);

**Example:**

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementId")));

**Fluent wait** - It is used to tell the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception.

**General syntax**: Wait wait = new FluentWait(WebDriver reference).withTimeout(timeout, SECONDS).pollingEvery(timeout, SECONDS).ignoring(Exception.class);

**3. Mention the types of navigation commands**

driver.navigate().to("<https://www.ebay.in/>"); - Navigates to the provided URL

driver.navigate().refresh(); - This method refreshes the current page

driver.navigate().forward(); - This method does the same operation as clicking on the Forward Button of any browser. It neither accepts nor returns anything.

driver.navigate().back(); - This method does the same operation as clicking on the Back Button of any browser. It neither accepts nor returns anything.

**4. What is the major difference between driver.close() and driver.quit()?**

driver.close()

This command closes the browser’s current window. If multiple windows are open, the current window of focus will be closed.

driver.quit()

 When quit() is called on the driver instance and there are one or more browser windows open, it closes all the open browser windows.

**6. What is an alternative option to driver.get() method to open an URL in Selenium Web Driver?**

An alternative option to the driver.get() method to open an URL in Selenium Web Driver is to use the driver.navigate().to() method.

**7. Mention different ways of locating an element in Selenium?**

The various ways of locating an element in Selenium are: by ID, by name, by classname, by tagname, by link text, by partial link text, by CSS selector, and by XPath.

**8. How to scroll down a page using JavaScript?**

scrollBy() method is used to scroll down the webpage

**General syntax:**

executeScript("window.scrollBy(x-pixels,y-pixels)");

First, create a JavaScript object

   JavascriptExecutor js = (JavascriptExecutor) driver;

Launch the desired application

   driver.get(“[https://www.amazon.com](https://www.amazon.com/)”);

Scroll down to the desired location

   js.executeScript("window.scrollBy(0,1000)");

The window is not scrolled vertically by 1000 pixels

**9. How to assert the title of a webpage?**

Get the title of the webpage and store in a variable

    String actualTitle = driver.getTitle();

Type in the expected title

   String expectedTitle = “abcdefgh"

Verify if both of them are equal

   if(actualTitle.equalsIgnoreCase(expectedTitle))

   System.out.println("Title Matched");

  else

  System.out.println("Title didn't match");

Alternatively,

   Assert.assertEquals(actualTitle, expectedTitle);

**10. How to mouse hover over a web element?**

Actions class utility is used to hover over a web element in Selenium WebDriver

Instantiate Actions class.

    Actions action = new Actions(driver);

In this scenario, we hover over search box of a website

  actions.moveToElement(driver.findElement(By.id("id of the searchbox"))).perform();

**11.What is POM (Page Object Model)?**

Every webpage of the application has a corresponding page class that is responsible for locating the web elements and performing actions on them. Page Object Model is a design pattern that helps create object repositories for the web elements. POM improves code reusability and readability. Multiple test cases can be run on the object repository.

**Example**:BDD framework that follows POM design pattern

 **Define Page Objects**: Create classes for each page of the application. Each class should include:

* Locators for the web elements on the page.
* Methods for interacting with these elements (e.g., click a button, enter text).

 **Create Feature Files**: Write feature files in a natural language that describe the behavior of the application. Feature files use the Gherkin syntax which includes:

* **Feature**: A high-level description of the functionality.
* **Scenario**: Specific situations or use cases to test.
* **Given, When, Then**: Steps that define the preconditions, actions, and expected outcomes.

 **Implement Step Definitions**: Create step definition files that map the steps defined in your feature files to the methods in your page objects. These files translate the Gherkin steps into actions performed using the Page Objects.

**Example Setup**

Here’s a simple example using Cucumber and Java with Selenium WebDriver:

**Page Object Class**

|  |
| --- |
| // LoginPage.java  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  public class LoginPage {  private WebDriver driver;    // Locators  private By usernameField = By.id("username");  private By passwordField = By.id("password");  private By loginButton = By.id("loginButton");    // Constructor  public LoginPage(WebDriver driver) {  this.driver = driver;  }    // Methods  public void enterUsername(String username) {  driver.findElement(usernameField).sendKeys(username);  }    public void enterPassword(String password) {  driver.findElement(passwordField).sendKeys(password);  }    public void clickLogin() {  driver.findElement(loginButton).click();  }  } |

**Feature File:**

|  |
| --- |
| # login.feature  Feature: Login functionality  Scenario: User logs in with valid credentials  Given the user is on the login page  When the user enters username "user" and password "password"  And the user clicks the login button  Then the user should be redirected to the home page |

**Step Definitions:**

|  |
| --- |
| // LoginSteps.java  import io.cucumber.java.en.\*;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.chrome.ChromeDriver;  public class LoginSteps {  private WebDriver driver;  private LoginPage loginPage;  @Given("the user is on the login page")  public void the\_user\_is\_on\_the\_login\_page() {  driver = new ChromeDriver();  driver.get("https://example.com/login");  loginPage = new LoginPage(driver);  }  @When("the user enters username {string} and password {string}")  public void the\_user\_enters\_username\_and\_password(String username, String password) {  loginPage.enterUsername(username);  loginPage.enterPassword(password);  }  @And("the user clicks the login button")  public void the\_user\_clicks\_the\_login\_button() {  loginPage.clickLogin();  }  @Then("the user should be redirected to the home page")  public void the\_user\_should\_be\_redirected\_to\_the\_home\_page() {  // Add assertions or validations here  driver.quit();  }  } |

**12. How to take screenshots in WebDriver?**

TakeScreenshot interface can be used to take screenshots in WebDriver.

getScreenshotAs() method can be used to save the screenshot

File scrFile = ((TakeScreenshot)driver).getScreenshotAs(outputType.FILE);

File destFile = new File(“define path to save the screenshot”);

FileUtils.copyFile(scrFile,destFile);

**13.What is the** **difference between getwindowhandles() and getwindowhandle()?**

getwindowhandle() method retrieves the unique identifier of the current browser window and its return type is String, while getwindowhandles() method returns a set of unique identifiers of all the browser windows opened by WebDriver and its return type is set of string.

**14.Is there a way to type in a textbox without using sendKeys()?**

Yes! Text can be entered into a textbox using JavaScriptExecutor

JavascriptExecutor jse = (JavascriptExecutor) driver;

//jse.executeScript("document.getElementById(‘email').value=“[abc.efg@xyz.com](mailto:abc.efg@xyz.com)”);

// Locate an element by XPath  
       WebElement inputElementByXPath = driver.findElement(By.xpath("//input[@id='inputId']"));  
  
       // Send keys using JavaScript  
jse.executeScript("arguments[0].value=’Text for XPath’;”, inputElementByXPath);

**15. How to select a value from a dropdown in Selenium WebDriver?**

Select class in WebDriver is used for selecting and deselecting options in a dropdown.

The objects of Select type can be initialized by passing the dropdown webElement as a parameter to its constructor.

WebElement testDrop = driver.findElement(By.id("testingDropdown"));

Select dropdown = new Select(testDrop);

WebDriver offers three ways to select from a dropdown:

selectByIndex: Selection based on index starting from 0

   dropdown.selectByIndex(5);

selectByValue: Selection based on value

   dropdown.selectByValue(“Books”);

selectByVisibleText: Selection of option that displays text matching the given argument

   dropdown.selectByVisibleText(“The Alchemist”);

**16.How to set browser window size in Selenium?**

The window size can be maximized, set or resized

To maximize the window

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

To set the window size

  Dimension d = new Dimension(400,600);

  driver.manage().window().setSize(d);

Alternatively,

The window size can be reset using JavaScriptExecutor

 ((JavascriptExecutor)driver).executeScript("window.resizeTo(1024, 768)");

**17. When do we use findElement() and findElements()?**

findElement() is used to access any single element on the web page. It returns the object of the first matching element of the specified locator. Return type of find element is webElement.

**General syntax:**  WebElement element = driver.findElement(By.id(example));

findElements() is used to find all the elements in the current web page matching the specified locator value. All the matching elements would be fetched and stored in the list of Web elements.Return type of find elements is a list.

**General syntax:** List <WebElement> elementList = driver.findElements(By.id(example));

**18.Explicit waits are generally considered better than implicit waits in many automation scenarios for several reasons:**

**• Explicit Wait**: Allows you to specify a certain condition to be met before proceeding. You have full control over what condition to wait for (such as an element being visible, clickable, present, etc.) and for how long to wait.

• **Explicit Wait:** Waits only for specific elements or conditions, reducing the overall waiting time by avoiding a blanket wait period.

**• Implicit Wait:** Applies a global wait time for all elements, which can lead to either unnecessary delays (if elements load faster) or failures (if the required condition is not covered by the implicit wait).

**• Implicit Wait:** Causes the script to wait for every element by the set time, whether it needs that long or not, leading to inefficiencies.

The main disadvantage of implicit wait is that it can slow down your tests. This is because, by default, the implicit wait time is set to zero. As such, if an element is not found immediately, your test will keep trying to find it for the duration of the implicit wait time. This can add a significant amount of time to your test suite. Another disadvantage of implicit wait is that it can cause your tests to fail if the element you are waiting for takes longer to appear than the implicit wait time. Finally, implicit wait can make your tests less reliable because they can introduce flakiness.

**19.diff between get and navigate To ()**

**get(url):**

• Used when you simply want to navigate to a URL.

• Suitable for straightforward page loading scenarios.

driver.get(“url”);

**navigate().to(url):**

• Used when you want more flexibility and control over the browser’s navigation, especially when combined with methods like .back(), .forward(), or .refresh().

• More appropriate when dealing with complex scenarios involving multiple navigations.

driver.navigate().to(“url”);

driver.navigate().back(); / driver.navigate().forward(); / driver.navigate().refresh();

Creating an automation framework can present several challenges, including:  
  
1. Choosing the Right Tools and Technologies  
  
• Challenge: Selecting appropriate tools that align with the project requirements, team skills, and future scalability.  
• Solution: Conduct thorough research and consider factors such as compatibility, community support, and maintenance.  
  
2. Designing a Scalable Architecture  
  
• Challenge: Building a framework that can easily accommodate changes, new features, and additional test cases.  
• Solution: Use design patterns (like Page Object Model) and modular architecture to separate concerns and promote reusability.  
  
3. Managing Dependencies  
  
• Challenge: Handling external dependencies like libraries, frameworks, and browsers can complicate the setup and execution of tests.  
• Solution: Use dependency management tools (e.g., Maven, Gradle) to simplify versioning and updates.  
  
4. Ensuring Test Reliability  
  
• Challenge: Dealing with flaky tests that fail intermittently due to timing issues, synchronization problems, or environmental factors.  
• Solution: Implement explicit waits, proper exception handling, and retry mechanisms to enhance test stability.  
  
5. Integrating with CI/CD Pipelines  
  
• Challenge: Integrating the framework with continuous integration/continuous deployment (CI/CD) systems can be complex.  
• Solution: Create scripts and configuration files for smooth integration with tools like Jenkins, GitLab CI, or CircleCI.  
  
6. Handling Dynamic Content  
  
• Challenge: Interacting with web applications that use dynamic content (AJAX, single-page applications) can be difficult.  
• Solution: Use appropriate waiting strategies and frameworks designed for dynamic content handling.  
  
7. Reporting and Logging  
  
• Challenge: Providing clear and actionable reporting can be challenging, especially when dealing with large test suites.  
• Solution: Implement logging frameworks and customizable reporting tools to capture relevant data effectively.  
  
8. Maintaining the Framework  
  
• Challenge: Keeping the framework up to date with changes in application features, UI, and dependencies.  
• Solution: Establish a maintenance plan and regular reviews to adapt the framework to evolving requirements.  
  
9. Training and Knowledge Sharing  
  
• Challenge: Ensuring that team members are adequately trained on the framework and its best practices.  
• Solution: Create comprehensive documentation, conduct training sessions, and foster a culture of knowledge sharing.  
  
10. Cost and Resource Allocation  
  
• Challenge: Balancing the costs of automation with its benefits, especially in resource-constrained environments.  
• Solution: Start small with high-impact areas and gradually expand the automation coverage based on return on investment.  
  
Conclusion  
  
Addressing these challenges requires careful planning, collaboration, and ongoing refinement of the automation framework. By anticipating potential issues and implementing strategic solutions, teams can build effective and sustainable automation solutions.has context menu