{c}odeBetter
Rise above the rest
401, Shekhar Central, Palasia Square, Indore, MP - 452001
www.codebetter.in
+91 88230 75444
+91 9993928766

Automation Testing

Section 1 – Selenium Automation

- **Introduction, Setup WebDriver, Launching Browser, Locating elements by id, name, classname**
- **Locating elements by XPathQueries, Attributes**
- **Handling TextFields, RadioButton and CheckBoxes**
- **Handling DropDowns and List Of Web Elements**
- **Mouse and Key Events, Actions, Alerts and IFrames**
- **Navigation between pages**

*1. Create a test script to automate the submission of a form on a web page. Use WebDriver to launch the browser, locate the form elements by id, name, and class name, and enter appropriate data into text fields. Validate the form submission by checking for success messages or error notifications.
*2. Write a test script to automate the selection of checkboxes and radio buttons on a web page. Use WebDriver to locate the elements by id or xpath queries and interact with them to select/deselect options. Verify that the selected options are reflected correctly on the page.
*3. Develop a test script to automate the selection of options from a dropdown menu and handle a list of web elements. Use WebDriver to locate the dropdown element, select specific options, and verify that the selected option is displayed correctly. Also, handle a list of web elements by iterating through them and performing actions on each element.

#1. Build a test script to automate mouse and keyboard events on a web page. Use WebDriver's Actions class to simulate mouse actions like hovering, right-clicking, and dragging. Also, simulate keyboard events like typing, pressing keys, and performing key combinations. Verify that the expected actions are performed on the page.
#2. Create a test script to automate the handling of alerts and iframes on a web page. Use WebDriver to handle alerts by accepting, dismissing, or entering text into prompt alerts. Also, interact with elements inside iframes by switching to the iframe context, locating elements, and performing actions. Validate that the expected actions are executed successfully.
#3. Write a test script to automate navigation between multiple pages on a website. Use WebDriver to click on links, buttons, or navigation elements to navigate to different pages. Verify that the correct page is loaded by checking for specific elements or page titles.

Section 2 – TestNG Setup, Annotations and Assertions

- **Page Automation Model, Flow Testing**
- **Data Driven Testing, Read/Write Excel Sheet**
- **Database Testing - MySQL**

*1. Implement a page automation model for a web application using the Page Object Model (POM) design pattern. Create separate classes for each page of the application, with methods representing the actions and interactions on that page. Use WebDriver to locate elements and perform actions in the respective page classes. Develop test script to execute end-to-end flows on the application using the POM.

*2. Write test scripts to automate the testing of specific flows in a web application. Identify critical user flows such as login, registration, checkout, etc. and develop test scripts to simulate and validate these flows using the automation framework. Utilize the page automation model to navigate through different pages and perform actions required for each flow.

#1. Implement data-driven testing by reading test data from an external source such as an Excel sheet. Create test Scripts that can read data from Excel sheets and use the data to drive the test execution. Parameterize test cases with different data sets, execute them, and validate the expected results. Implement data-driven tests for different scenarios, such as login with multiple user credentials.

#2. Develop functions to read and write data to an Excel sheet using a library like Apache POI. Create reusable methods to read test data from Excel sheets and use it in test scripts. Similarly, create methods to write test results or other relevant data back to the Excel sheet. Implement error handling and exception reporting for smooth data processing.

#3. Create test scripts to automate the testing of database operations using MYSQL as the database. Use JDBC (Java Database Connectivity) to establish a connection with the database and perform CRUD (Create, Read, Update, and Delete) operations on tables. Implement test scripts to validate data insertion, retrieval, updating, and deletion from the database.

Section 3 – Appium

*1. Set up the Appium framework and configure the necessary dependencies and drivers to automate mobile app testing. Install the required SDKs and tools for Android and IOS devices. Set up the desired capabilities for the target devices and platforms.

*2. Write test scripts to automate the installation and launch of mobile apps on target devices. Use Appium's capabilities to install the app package and launch the app on emulators or real devices. Verify that the app launches successfully and is ready for interaction.

*3. Use Appium's locator strategies to locate and interact with elements in mobile apps. Demonstrate the ability to find elements by ID, XPath, class name, and other attributes. Write test scripts to perform actions on text fields, buttons, checkboxes, radio buttons, and other interactive elements.

*4. Develop test scripts to simulate user interactions with mobile apps. Implement actions such as tapping, swiping, scrolling, and pinching to zoom. Validate that the app responds correctly to user inputs and that the expected behavior is observed.

#1. Write test scripts to handle alerts, pop-ups, and permission requests that may appear during app usage. Implement logic to accept or dismiss alerts and handle permissions, such as camera access or location permissions, as required.

#2. Execute test scripts on multiple devices concurrently to ensure the compatibility and responsiveness of the mobile app across various screen sizes, resolutions, and device capabilities. Implement test scripts that can be executed in parallel on different devices connected to the Appium server.
#3. Implement data-driven testing for mobile apps using external data sources such as Excel sheets or CVS files. Read test data from the external sources and use it to drive the test execution for different scenarios. Parameterize test cases with different data sets and validate the expected results.
#4. Set up and configure Appium Grid to distribute test execution across multiple devices or emulators. Write test scripts that can be executed in parallel on different devices connected to the Appium Grid. Validate the scalability and efficiency of the mobile app across multiple devices simultaneously.

Section 4 – Cucumber

*1. Set up Cucumber framework and configure the necessary dependencies. Create feature files to define the behavior and scenarios for the application under test. Write feature scenarios using Gherkin syntax to describe the expected behavior in a human-readable format.
*2. Implement steps definitions for the scenarios defined in the feature files. Map the step definitions to the corresponding Gherkin steps in the feature files. Execute the Cucumber tests and verify that the steps are executed correctly and produce the expected outcomes.
*3. Implement data-driven testing using Cucumber's scenario outline feature. Define example tables in the feature files and provide test data for different scenarios. Create steps definitions that can handle dynamic test data and validate the expected results for each data set.
*4. Utilize Cucumber's hooks feature to set up pre- and post- conditions for scenario. Implement scenario hooks to execute common setup and teardown actions. Use the background section in feature files to define steps that are common to all scenarios within a feature.

#1. Add tags to scenarios and features to categorize and control the execution of tests. Use tags to run specific subsets of scenarios or exclude certain scenarios from execution. Configure the test runner to execute tests based on the specific tags.
#2. Integrate reporting plugins or frameworks with Cucumber to generate comprehensive test reports. Customize the reports to include relevant information such as test execution status, step details, and overall test coverage. Analyze the test results to identify failure and track test progress.
#3. Integrate Cucumber with other test frameworks and tools such as Junit or TestNG. Use the capabilities of these frameworks to enhance test management, parallel execution, and test configuration. Explore the integration with build tools like Maven or Gradle for seamless test execution.
#4. Combine Cucumber with selenium or Appium for automated web or mobile app testing. Write step definitions that interact with web elements or mobile app elements using Selenium or Appium APIs. Perform actions such as clicking, inputting data, and verifying expected outcomes.