10 : Automated Testing Suites for Web Applications

IT6206 – Software Quality Assurance

Level III - Semester 6

Overview

- In this section, you will be introduced to the concepts of Automated Testing Suits for Web Applications.
- Automated testing is a software testing technique that uses tools and scripts to automate test cases, reducing the need for manual testing.
- Automated testing suites for web applications are designed to test the functionality, performance, and usability of web-based software.
- Some popular automated testing suites for web applications include Selenium, TestCafe, and Cypress with their own unique features and capabilities.

Intended Learning Outcomes

At the end of this lesson, you will be able to;

- Understand the definition, purpose,and the advantages of using automated testing suites for web applications.
- Understand the basic concepts of Selenium with its components, benefits and limitations.
- Identify different types of architectures and layers for automated testing suites for web applications.
- Learn the purpose of using recorders in automated testing suites.
- Understand about using Selenium API for automated testing.

List of sub topics

10.1 Introduction to Automated Testing Suites for Web Applications

- 10.1.1 Introduction to the topic
- 10.1.2 Types of automated testing for web applications
- 10.1.3 Introduction to Selenium
- 10.1.4 Components of Selenium Suite
- 10.1.5 Advantages of Selenium
- 10.1.6 Limitations of Selenium

10.2 Architectures of Automated Testing Suites for Web Applications

- 10.2.1 Types of architectures for automated testing suites for web applications
- 10.2.2 Layers in the automated testing suite architectures

List of sub topics

10.3 Recorders in Automated Testing Suites for Web Applications

- 10.3.1 Introduction to recorders
- 10.3.2 Selenium IDE
- 10.3.3 Katalon Recorder
- 10.3.4 Limitations of recorders
- 10.3.5 Best practices of using recorders

10.4 Working with Selenium API

- 10.4.1 Overview of Selenium API
- 10.4.2 Selenium API: Locating Elements
- 10.4.3 Selenium API: Interacting with Elements
- 10.4.4 Selenium API: Handling Windows and Frames
- 10.4.5 Selenium API: Executing JavaScript

List of sub topics

10.4 Working with Selenium API cont.

- 10.4.6 Selenium API: Working with Wait Commands
- 10.4.7 Selenium API: Handling Alerts and Pop-ups
- 10.4.8 Best Practices for Working with Selenium API

10.1 Introduction to Automated Testing Suites for Web Applications

10.1.1 Introduction

- Automated testing is the practice of using software tools to automatically run tests on software applications.

- It allows the developers and testers to test complex user interactions and business logic efficiently across multiple browsers and devices.

- Web applications are often complex systems with multiple pages, user inputs, and database interactions. Conducting manual testing on such complex applications could be time-consuming and error-prone. Automated testing helps to streamline the testing process by reducing the issues of manual testing.

- Automated testing also has the potential to cater to unique configurations of different types of browsers and devices.

10.1.1 Introduction cont.

- Automated testing suites have become a crucial part of modern software development, especially for web applications.

- There are many different automated testing frameworks and tools available for testing web applications, each with its own properties.

Image Source: https://testjet.ai/web-browser-automation-testing/

10.1.1 Introduction cont.

Below are some popularly used automated testing suites.

1. **Selenium**

It is one of the most popular and widely used open-source automated testing suite used in web applications. It supports multiple programming languages such as Java, Python, C#, JavaScript and Ruby. Selenium can be used in automated testing tasks such as functional testing, regression testing, and browser compatibility testing.

Within this topic, we will be mainly focusing on Selenium as an automated testing suite for web applications.

2. **Cypress**

A JavaScript based testing framework. It provides a complete end-to-end testing experience, with real-time reloading, automatic waiting features and powerful debugging tools.

10.1.1 Introduction cont.

3. TestCafe

A JavaScript-based testing framework which focuses on cross-browser testing and parallel test executions. It has the capability of handling complex user interactions and workflows.It also includes features such as automatic writing, smart assertions, and visual regression testing.

4. Puppeteer

Puppeteer is a Node.js based testing framework which is widely being used for testing complex user interactions and workflows that are difficult to be simulated with other testing frameworks. It also includes features such as headless testing, network traffic interception, and PDF generation.

10.1.2 Types of automated testing for web applications

There are many types of automated testing for web applications. Here we will be discussing about three most common and important types which are unit testing, integration testing, and end-to-end testing.

1. Unit Testing

- Unit testing is a type of testing which focuses on individual units or components of an application in isolation.
- With related to web applications, this involves the testing of individual functions or methods within a particular module or component.
- Unit tests are executed as a part of continuous integration or the build process to ensure that the changes to the code do not introduce bugs.

10.1.2 Types of automated testing for web applications

2. Integration Testing

- It focuses on testing how different components or modules of an application work together.
- With related to web applications, this involves testing how the front-end user interface interacts with the back-end server and database. It might also involve testing how different the microservices of the application interact with each other.
- Integration testing is executed as a part of continuous integration or deployment process to ensure that the application as a whole functions correctly.

10.1.2 Types of automated testing for web applications

3. End-to-end testing

- End-to-end testing focuses on testing the entire application workflow from the start to finish.
- With related to web applications, it involves simulating user interactions with the applications. Some examples of user interactions are clicking buttons, navigating through different pages, navigating through the screens, etc.
- These tests are executed manually or using automated testing suites to ensure that the application works as expected from the user's perspective.

10.1.3 Introduction to Selenium

- Selenium is a free and open-source test automation suite which is widely being used in web applications. It supports test automation across different browsers, platforms, and programming languages.

- Selenium allows users to simulate common activities performed by end-users such as entering text into fields, selecting drop-down values, checking boxes, and clicking links in documents.

- It also provides a common interface for all major browser technologies making it easier for the software developers and testers to work in different browser platforms.

10.1.4 Components of Selenium Suite

Selenium suite is comprised of four main components.

1. **Selenium WebDriver**

Selenium WebDriver can be used to create robust, browser-based regression automation suites and tests and to scale and distribute scripts across many environments. tIt is a collection of language specific bindings to drive a browser. It provides different drivers for different browsers such as Chrome, Firefox, Safari etc. while supporting multiple programming languages.

10.1.4 Components of Selenium Suite cont.

2. Selenium IDE

Selenium IDE is a Chrome, Firefox and Edge add-on that will do simple record-and-playback of interactions with the browser. The record and playback feature provided by the IDE makes the testing process easier even for the people with no or limited programming language experience. The IDE can be used to create quick bug reproduction scripts, and to create scripts to aid in automated testing.

10.1.4 Components of Selenium Suite cont.

3. Selenium Grid

Selenium Grid helps in the distributed running of Selenium tests in parallel across multiple remote machines. It facilitates scaling by distributing and running tests on several machines. It has the ability to manage multiple environments from a central point, making it easy to run the tests against a vast combination of browsers or Operating Systems.

Selenium Grid is comprised with a Hub and Nodes architecture. The nodes run the Selenium instances on which the test cases are executed. The central hub has multiple connecting nodes and acts as a server to control the whole test execution.

10.1.4 Components of Selenium Suite cont.

4. Selenium Remote Controller

This feature is currently not available as a component of Selenium.

It was used to inject the javascript code in the browser for automation and required an additional server for running the automation scripts.

10.1.5 Advantages of Selenium

Below are some key benefits and features of using Selenium for automated testing.

1. Cross-browser compatibility as Selenium supports testing across multiple browsers such as Chrome, Safari, Firefox, Edge, and Internet Explorer.
2. Selenium supports multiple programming languages and can be integrated with a wide range of testing frameworks and tools.
3. The record and playback feature allows the testers to record interactions with a web applications and play them back as automates tests.
4. Selenium can be integrated with other tools, such as test management systems, continuous integration servers etc.
5. Selenium has a large community of users and developers with documentations, tutorials and online support.

10.1.6 Limitations of Selenium

Below are some limitations of Selenium.

1. Selenium can only be used to test web applications in browsers that it supports.
2. Selenium does not provide the same level of support for mobile applications as it does for desktop browsers.
3. Creating automated tests with Selenium can be time-consuming as it requires knowledge of programming languages and web technologies.
4. Selenium is mainly being used for testing web applications. It may not be the best choice for testing desktop applications or other non-web software.
5. Selenium has limited capabilities in recognizing and interacting with images or non-HTML/CSS content on a page.

10.2 Architectures of automated testing suites for web applications.

10.2.1 Types of architectures for automated testing suites for web applications.

Automated testing is an essential part of modern web application development. Choosing the right architecture for an automated testing suite is crucial to ensure that testing is effective and efficient. There are several popular architectures used for automated testing suites, and each has its own strengths and weaknesses.

1. Record and Playback Architecture

- This is one of the most commonly used architectures. Here, the testing tool records the user's interactions with the web application and generates automated tests based on those interactions.
- The tests can then be played back to simulate the same interactions in subsequent testing runs. This architecture is simple and easy to use, but it can be brittle and may not work well with complex web applications.

10.2.1 Types of architectures for automated testing suites for web applications cont.

2. Keyword-driven Architecture

- This architecture uses a set of predefined keywords and commands to interact with the application and build test cases. The keywords are used to build test cases that can be executed against the application. This architecture is more flexible than record and playback, but it requires more upfront planning and design.

10.2.1 Types of architectures for automated testing suites for web applications cont.

3. Data-driven Architecture

- This architecture is useful for testing applications that require a large number of input combinations. In this architecture, tests are designed to use a set of input data to drive the testing process. The testing tool reads the input data from a file or database and uses it to execute a set of automated tests.

10.2.1 Types of architectures for automated testing suites for web applications cont.

4. Modular Architecture

- This architecture is useful for testing complex web applications that require a high degree of modularity and scalability. Tests are broken down into smaller modules or components, each of which can be tested independently. The testing tool can then combine these modules to create more complex tests that simulate real-world user interactions.

10.2.1 Types of architectures for automated testing suites for web applications cont.

5. Page Object Model Architecture

- This architecture is useful for testing web pages through a set of objects that represent different elements on the page. These objects are organized into a page object model, which makes it easy to write and maintain automated tests for complex web applications.

10.2.1 Types of architectures for automated testing suites for web applications cont.

- The choice of architecture for an automated testing suite will depend on the specific needs of the web application and the preferences of the testing team.

- By understanding the different architectures available and their pros and cons, testers can choose the one that is best suited for their needs and goals.

- A well-designed automated testing architecture can improve the quality and reliability of the web application, reduce testing time, and ultimately save time and resources for the development team.

10.2.2 Layers in the automated testing suite architectures

The test layer, the automation layer, and the framework layer are the three main levels that make up the usual layered design of automated testing suites for web applications.

1. Test Layer

- This is where the actual test cases are created.
- It contains test cases for end-to-end testing, unit testing, integration testing, and any other kinds of tests that the testing team chooses to use.
- Depending on the automation technology used, the tests are written in a scripting language or a programming language.

10.2.2 Layers in the automated testing suite architectures

2. Automation Layer

- Automation layer is responsible for executing the test cases written in the test layer.
- It engages with the web application and mimics user behaviors such as button clicks, form fills, and page scrolling. The web application's user interface and the database can both be accessed through the libraries and APIs in the automation layer.

3. Framework Layer

- Framework layer gives the automated tests a structure for planning and carrying out their operations.
- It has elements for test data management, test configuration, and test reporting. It also offers assistance with test planning, test ranking, and test automation administration.

10.3 Recorders in Automated Testing Suites for Web Applications

10.3.1 Introduction to recorders

- Recorders are essential tools in many automated testing suites which allows testers to record user interactions with the web application and automatically generate test scripts.

- Recorders help speeding up the testing process, reduce the likelihood of eros as well as ensure accurate test results.

- Recorders enable faster test case creation by automating the process of generating test scripts, which saves time and effort.

10.3.1 Introduction to recorders cont.

- More accurate test cases can be obtained through this process as the recordings accurately capture user behavior and can help ensure test cases are reflective of real user behavior.

- Reduction also supports in reducing errors and inconsistencies in the test scripts.

- Selenium IDE and Katalon recorder are some most commonly used types of recorders.

10.3.2 Selenium IDE

- Selenium IDE is a browser plugin that offers a simple interface for recording and playing back test scripts. It is an easy-to-use browser extension that records a user's actions in the browser using existing Selenium commands.

- This browser extension is available for Google Chrome, Mozilla Firefox, as well as Microsoft Edge. You can refer to the Selenium IDE documentation using the following link to learn more about the installation, launching, and recording test cases using Selenium IDE.

  https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started

10.3.3 Katalon Recorder

- Katalon recorder is the Selenium-IDE compatible record and playback tool for browser automation testing. It can be used to record, debug, execute and manage test cases. It also provided capabilities in exporting test suites to multiple programming languages such as C#, Java, Python, Ruby etc.

- Katalon Recorder's main User Interface contains four main sections, Main toolbar, Test Explorer, Test Case Detail View, and Log/Reference/Variable.Self-healing section.
- You can learn more about these sections by accessing the Katalon Recorder documentation through the following link.

  https://katalon.com/resources-center/blog/katalon-automation-recorder

10.3.4 Limitations of recorders

- Recordings may not capture all aspects of user behaviour, and manual testing or code-based testing may be necessary to ensure the accuracy of the testing process.Therefore, it is not a substitute for manual testing or code-based testing.

- Some user interactions may not be captured by recordings which could lead to incomplete or inaccurate test cases.

- Recorded scripts may not fit the specific needs of the web applications being tested and may require customization.

10.3.5 Best Practices for using recorders

- Recorders should be used as a part of a comprehensive testing approach that includes manual and code-based testing.

- Recorded scripts should be reviewed and updated regularly to ensure the accuracy and effectiveness.

- Recorded scripts may need to be customized to fit the specific needs of the web application being tested.

10.4 Working with Selenium API

10.4.1 Overview of Selenium API

- Selenium API is a suite of tools and libraries that developers and testers can use to automate web browsers.

- It provides a powerful and flexible framework for automating testing tasks, which can help reduce testing time and improve accuracy.

- Selenium WebDriver is the most widely used component of the Selenium API. Setting up the Selenium WebDriver includes installing drivers for the relevant web browser being used, configuring the Selenium environment, and importing necessary libraries.

10.4.2 Selenium API: Locating Elements

- Locators are used to find elements on a web page, such as buttons, form fields etc.

- Types of locators in Selenium includes ID, name, class name, tag name, link text, and partial link text.
  - By CSS ID: find_element_by_id
  - By CSS Class name: find_elements_by_class_name
  - By name attribute: find_element_by_name
  - By DOM structure or Xpath: find_element_by_xpath
  - By tagname: find_element_by_tag_name()
  - By link text: find_element_by_link_text
  - By partial link text: find_element_by partial_link_text
  - By HTML tag name: find_element_by_tag_name

10.4.2 Selenium API: Locating Elements cont.

- find_element_by_id() method of the webDriver class is being called to uniquely identify an element by its ID.

```
from selenium import webdriver

driver = webdriver.Chrome('./chromedriver')

driver.get("https://www.python.org")

# Returns first element with matching class

first_search_bar = driver.find_element_by_id("id-search-field")
```

- You can refer to the following link to learn more about the locators in Selenium API

https://www.browserstack.com/guide/locators-in-selenium

10.4.3 Selenium API: Interacting with Elements

- Actions are used to interact with elements on a web page, such as clicking buttons, filling out forms, and navigating to links

- Action class has the ability to handle keyboard and mouse events in Selenium WebDriver.

- Action class is defined and invoked using the following syntax.

```
Actions action = new Actions(driver);

action.moveToElement(element).click().perform();
```

Source: https://www.browserstack.com/guide/selenium-webelement-commands

10.4.3 Selenium API: Interacting with Elements

- Mouse actions in Selenium includes,
  - doubleClick() to perform double click on elements
  - clickAndHold() to perform long clicks on the mouse without releasing it.
  - dragAndDrop() to drag the elements from one point and drops to another
  - movetoElement() to shift the mouse pointer to the center of the element
  - contextClick() to perform right-click on the mouse

- Keyboard actions in Selenium includes,
  - sendKeys()
  - keyUp()
  - keyDown()

10.4.4 Selenium API:Handling Windows and Frames

- Web pages may contain multiple windows and frames, which can make automation challenging.Techniques for handling windows and frames in Selenium includes methods for switching between windows and frames, or for interacting with elements within frames.
- A window handle is a unique identifier which holds the address of all the windows. It is similar to the concept of a pointer to a window which returns a string value. The window handle functions helps to retrieve the handles of the windows.

10.4.5 Selenium API: Executing JavaScript

- Selenium provides the ability to execute JavaScript within the browser, which can be useful for interacting with elements or manipulating the page.
- JavaScriptExecutor is an interface that is used to execute JavaScript with Selenium. It enables the WebDriver to interact with HTML elements within the browser.
- executeScript method in JavaScriptExecutor executes the test script in the context of the currently selected window or the frame.

10.4.5 Selenium API: Executing JavaScript cont.

- executeAsyncScript method executes the asynchronous piece of JavaScript on the current window or frame.
- Getting started with the JavaScriptExecutor includes importing the relevant packages, creating a reference, and calling the relevant JavaScriptExecutor methods.

```
[java]//importing the package

Import org.openqa.selenium.JavascriptExecutor;

//creating a reference

JavascriptExecutor js = (JavascriptExecutor) driver;

//calling the method

js.executeScript(script, args);

[/java]
```

Source: https://www.browserstack.com/guide/javascriptexecutor-in-selenium

10.4.5 Selenium API: Executing JavaScript cont.

- JavaScriptExecutor to click a button

```
[java]

js.executeScript("document.getElementByID('element id ').click();");

[/java]
```

- JavaScriptExecutor to send text

```
[java]

js.executeScript("document.getElementByID('element id ').value = 'xyz';");

[/java]
```

- JavaScriptExecutor to interact with a checkbox.

```
[java]

js.executeScript("document.getElementByID('element id ').checked=false;");

[/java]
```

Source: https://www.browserstack.com/guide/javascriptexecutor-in-selenium

10.4.6 Selenium API: Working with Wait Commands

- Wait commands are used to ensure that elements have loaded properly before interacting with them.
- When a page loads on a browser, various web elements on it may be loaded at different times. Wait Commands direct a test script to pause for some time before the ElementNotVisibleException is throwed.
- The three main types of Waid Commands in Selenium are implicit waits, explicit waits, and fluent waits.
  - Implicit Wait : Here, the Selenium WebDriver will be directed to wait for a given time before throwing the exception.

```
import java.util.concurrent.TimeUnit;

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Source: https://www.browserstack.com/guide/wait-commands-in-selenium-webdriver

10.4.6 Selenium API: Working with Wait Commands

- explicit Wait : Here, the Selenium WebDriver will be directed to wait until a certain condition occurs before proceeding with the execution of the code.
- fluent wait: Here, a web element is being looked repeatedly at regular intervals until a timeout happens or until the desired object is found. It is capable of defining how frequently the webDriver will check if the web element appears before throwing the ElementNotVisibleException.

- To learn more about the wait commands, refer to the documentation given in the following link.

https://www.browserstack.com/guide/wait-commands-in-selenium-webdriver

10.4.7 Selenium API: Handling Alerts and Pop-ups

- Web pages may contain alerts or pop-ups that require user interaction.
- Techniques for handling alerts and pop-ups in Selenium includes methods for accepting or dismissing alerts, or for interacting with elements within pop-ups.
- Alert is a message or a notification box that notifies the user about some information or asks permission to do a certain action. The three main types of alerts in Selenium are:
  - Simple Alert
  - Prompt Alert
  - Confirmation Alert

10.4.7 Selenium API: Handling Alerts and Pop-ups

- Popup is a window that displays or pops up on the screen due to a certain activity.
- Selenium has designated methods for different types of alerts as well as handling popups.
- To learn more about these methods, refer to the documentation given in the following link.

  https://www.browserstack.com/guide/alerts-and-popups-in-selenium

10.4.8 Best Practices for working with Selenium API

- Always use a consistent coding style to make the code more readable and maintainable.

- Write modular code to make code more flexible and easier to maintain.

- Use debugging tools to identify and fix errors more efficiently.

Summary

- Automated testing suites for web applications are a powerful tool for ensuring software quality, reducing testing time and effort, and increasing test coverage.
- There are a variety of types of automated testing, including unit testing, integration testing, and end-to-end testing.
- Different automated testing tools offer various features and capabilities.
- Some popular automated testing suites for web applications include Selenium, TestCafe, and Cypress.

Summary

- It is important to carefully plan and execute automated testing to avoid common pitfalls, and to ensure that testing is effective and efficient.
- Automated testing is an essential component of modern software development, and can help to improve software quality and customer satisfaction.

References

[1] The Selenium Browser Automation Project Documentation, "https://www.selenium.dev/documentation/"

[2] Selenium Tutorial, "https://artoftesting.com/selenium-tutorial"

[3] About Katalon Platform, "https://docs.katalon.com/docs"

[4] Locators in Selenium, "https://www.browserstack.com/guide/locators-in-selenium"

[5] Selenium WebElement Commands, "https://www.browserstack.com/guide/selenium-webelement-commands"

[6] How to handle multiple windows in Selenium? "https://www.browserstack.com/guide/handle-multiple-windows-in-selenium"

[7] How to use JavascriptExecutor in Selenium, "https://www.browserstack.com/guide/javascriptexecutor-in-selenium"

References

[8] Selenium Wait Commands : Implicit, Explicit & Fluent Wait,

“https://www.browserstack.com/guide/wait-commands-in-selenium-webdriver”

[9] How to handle Alerts and Popups in Selenium,

“https://www.browserstack.com/guide/alerts-and-popups-in-selenium”