Automated Web Tests with WebDriver/Selenium 2

Lab Exercises

Table of Contents

Introduction

This workbook is designed to accompany Wakaleo Consulting's ATDD and Automated Web Testing training course. It consists of a sequence of lab exercises which will introduce the concepts of Autaomted Acceptance Testing and Automated Web Testing for Java developers. These lab exercises should be completed in sequence in the presence of an instructor. The instructor will distribute the software and code necessary to complete these exercises. If, at any time during the lab exercise, you encounter problems with your software installation or if you do not understand any of the instructions, please ask your instructor for help.

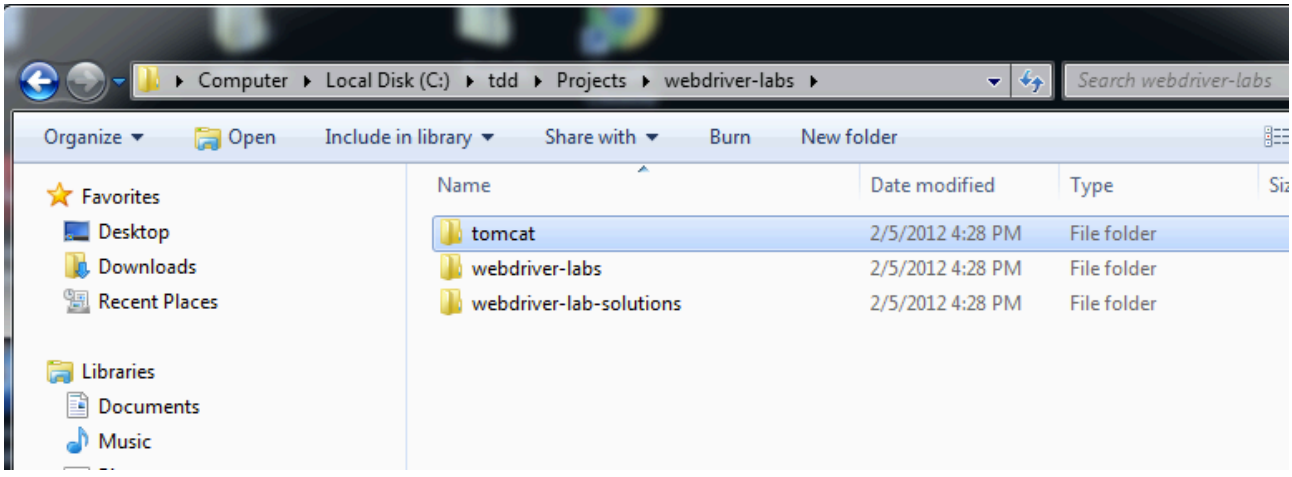
This lab manual and training materials assume that you have been supplied with the necessary software prerequisites. Your instructor will distribute installation media or provide instructions for downloading this material from the Internet. The supporting software required for this course are:

- Java Development Kit version 1.6.0_15 (JDK)
- IntelliJ or Eclipse 3.5 or greater
- Apache Maven 2.2.0 or greater
- Firefox

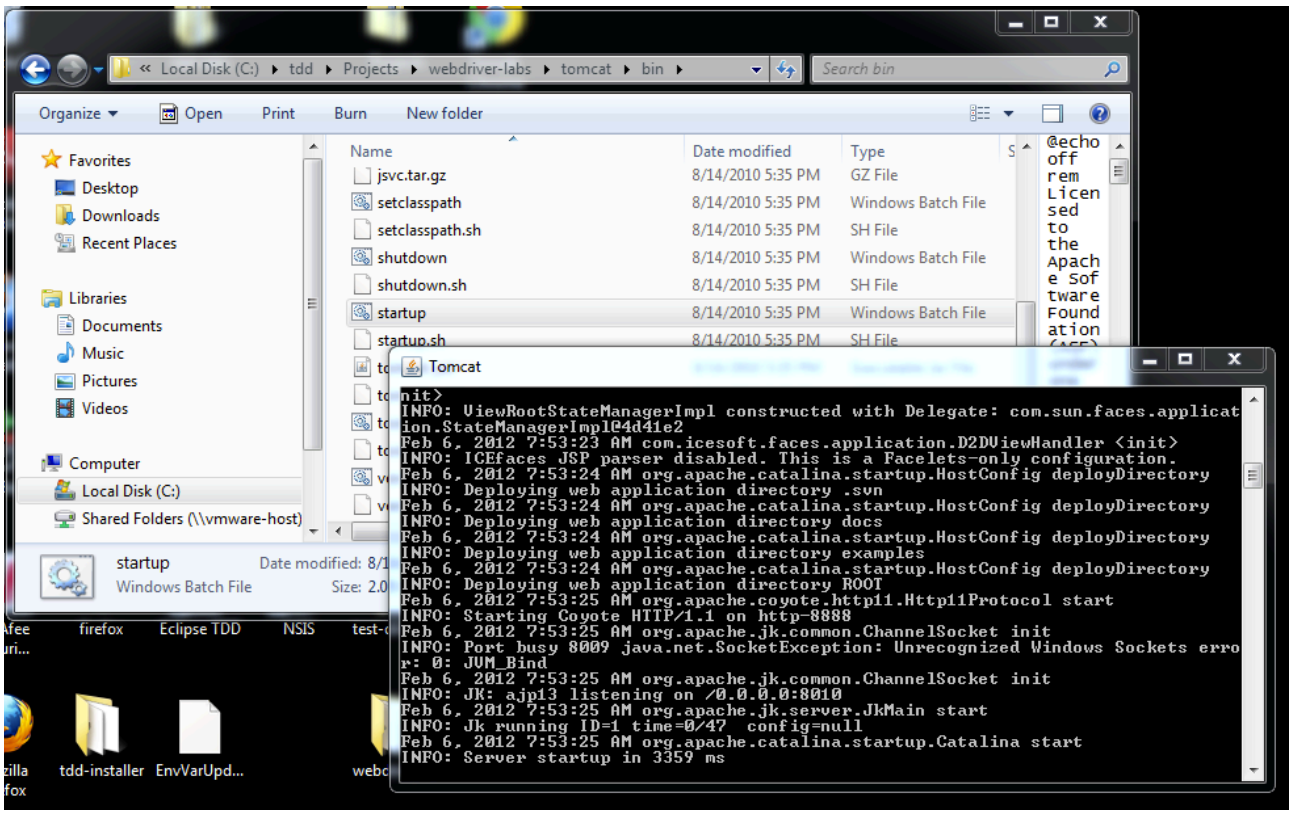
The labs assume that a recent version of Java (JDK 1.5.0 or higher), Maven and Firefox have been installed on the workstations.

Running the web application to be tested

We will be testing a simple AJAX application called 'memorygame', provided in the tomcat directory which will have been installed into the `C:\tdd\projects\webdriver-labs` directory (see below).



You can start the tomcat server by clicking on the `startup.bat` file in the `tomcat\bin` directory (see below).



When you start up tomcat, the application will be running on port 8888 (http://localhost:8888/memorygame).

Lab 1 - First steps with WebDriver

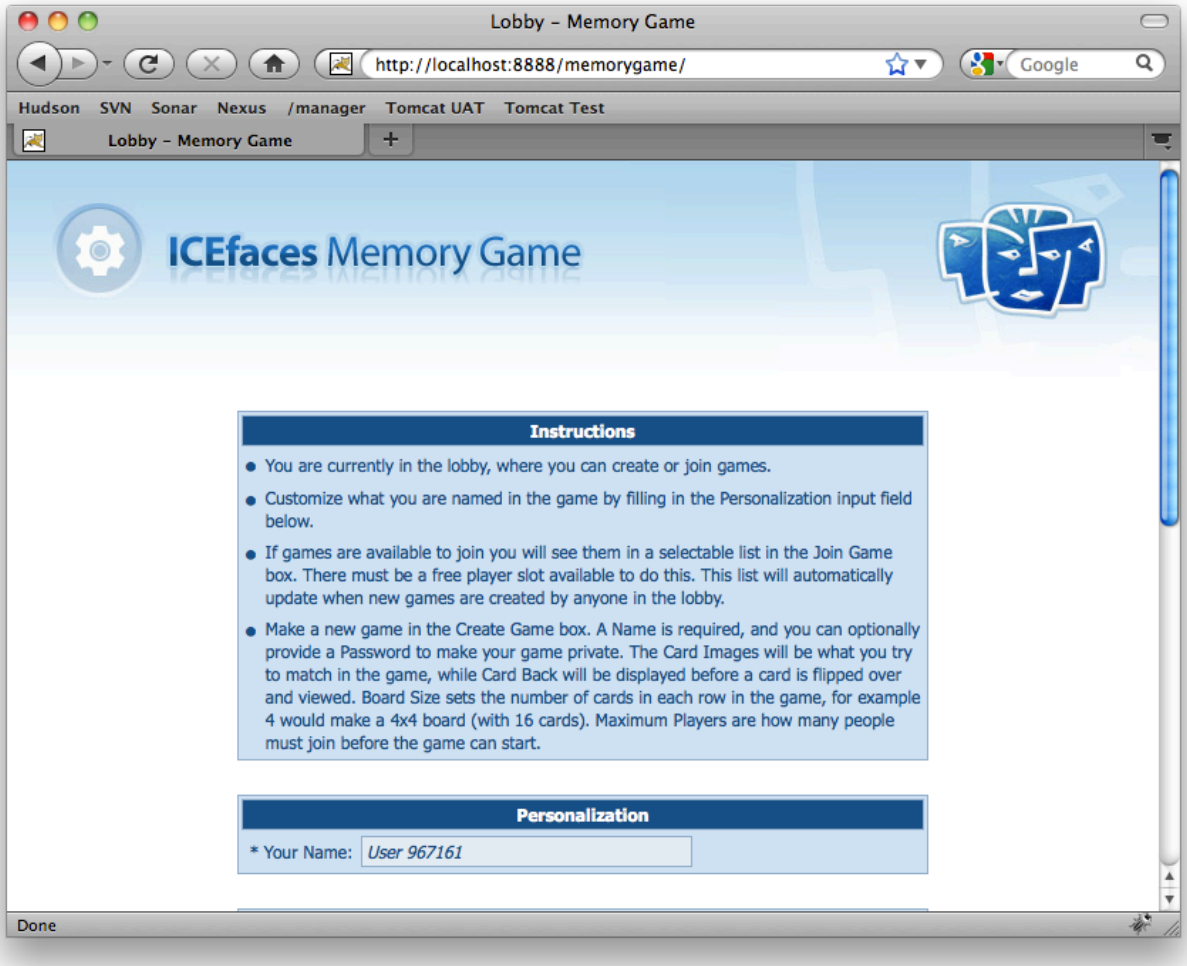
Goal

The aim of this lab exercise is to write a simple WebDriver test to open a web application and verify the title of the home page.

Lab Exercises

Step 1: Ensure that the test application is run successfully

Start up the Tomcat instance and open a browser to http://localhost:8888/memorygame to make sure the application is running:



Step 2: Write a test that opens this page and verifies the title

Open the webdriver-labs project in Eclipse and create a new test class called `WhenAUserOpensTheHomePage`. Add a test to check the title of the page, e.g.

```
@Test
public void theHomePageShouldDisplayMemoryGameInTheTitle() {
      WebDriver driver = new FirefoxDriver();
      driver.get("http://localhost:8888/memorygame");
```

```
        assertThat(driver.getTitle(), containsString("Memory Game"));
        driver.close();
    }
```

Run this test in Eclipse to ensure that it works.

Step 2: Write a test that opens this page and verifies the exact title

Now write another test that verifies that the title starts with the word “Lobby”.

Step 3: Refactor

Refactor your tests to eliminate any duplication.

Lab 2 - Identifying page elements

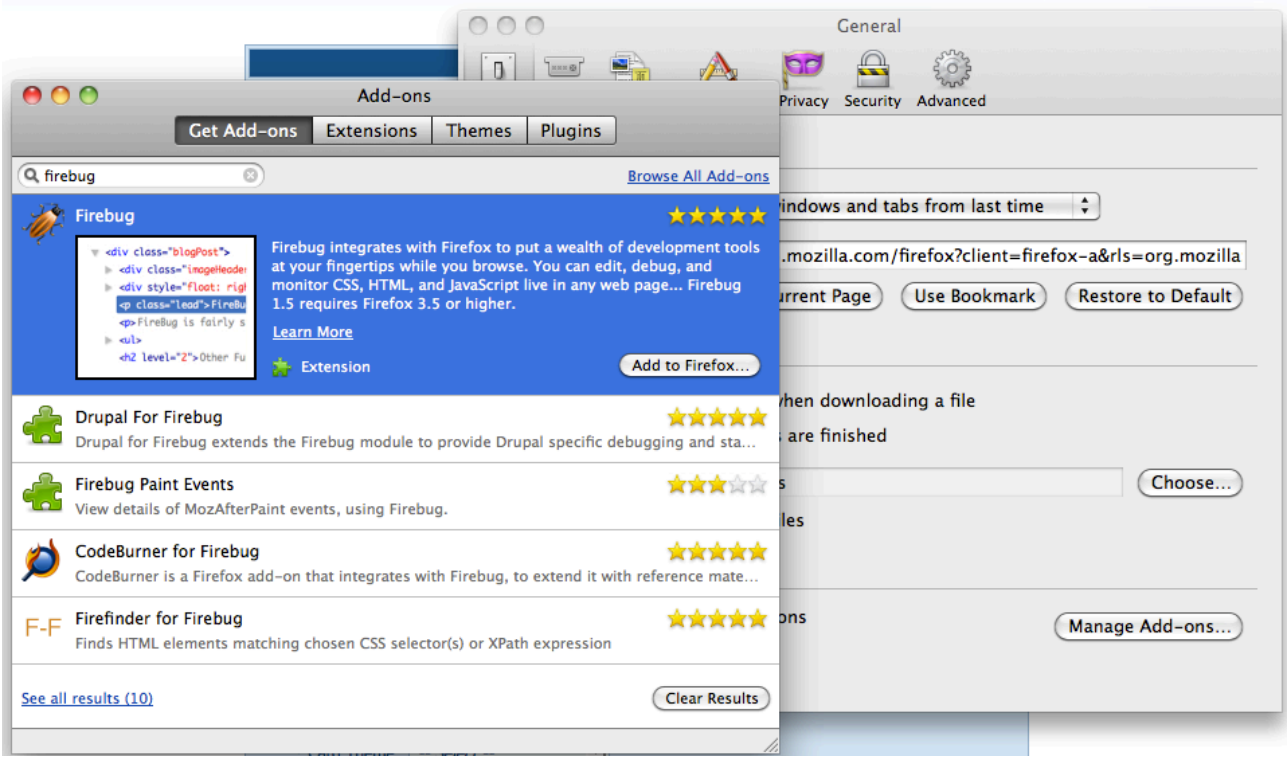
Goal

The aim of this lab exercise is to learn to identify web elements.

Lab Exercises

Step 1: Installing Firebug and Firefinder

From this lab on, we will be using the Firebug and Firefinder plugins to help verify XPath and CSS Selector expressions. Install the Firebug and Firefinder plugins for Firefox:



Once you have installed the plugins, you will need to restart Firefox. In all of the following steps, you can use the Firebug and Firefinder extensions to help you identify elements on the HTML page and to test your XPath and CSS selector expressions.

Step 2: Identifying a link

Write a test to find the “Advanced Options” link on the page. You may wish to use the By.linkText method, e.g.

```
@Test
public void theHomePageShouldContainAnAdvancedOptionsLink() {
      WebElement userNameField
            = driver.findElement(By.linkText("Advanced Options"));
}
```

Step 3: Verifying the text displayed in the submit button

Write a test to ensure that the text displayed in the submit button is “Create Game”. You might do this using the CSS class used in the submit button (“iceCmdButton”), by using either the By.className() method or the by.cssSelector() method.

Step 4: Finding a field
Use an XPath expression to find the “Your Name” field. Hint: use FireFinder to identify the id. The id is a generated value, but it always ends in the value ‘userName’.

Step 5: Finding a table title
Ensure that the page has a table entitled ‘Instructions’. Use the XPath with the contains() function to do this.

Step 6: Nested XPath selectors
Ensure that the table with the heading of “Instructions” contains the words “You are currently in the lobby”. Again, use the XPath with the contains().

Lab 3 - Working with forms

Goal

The aim of this lab exercise is to learn to work with forms in WebDriver.

Lab Exercises

Step 1: Submitting a form

Create a new test class called `WhenAUserCreatesANewGame`, and create a test called `theUserCanCreateANewGameOnTheHomePage()`. In this class, ensure that a user can enter values into the mandatory text and dropdown fields, and then submit the form. To check that the form has been submitted correctly, check that the page title contains the value that you entered in the 'Game Name' field.

Step 2: Error handling

Add a test to check that if no values in the dropdown lists are selected, when the form is submitted two error messages will be displayed.

Lab 4 - Page Objects

Goal

The aim of this lab exercise is to learn to work with Page Objects.

Lab Exercises

Step 1: Create a Page Object for the New Game page.

Create a new page object class called `MemoryGameHomePage`. This first page object will only contain three fields: gameName, gamePassword and errorMessage. Write a test using this page object to check that an error message is displayed when you forget to enter the name field.

Hint: your first test might look like this:

```
public class WhenAUserEntersAnEmptyName {

	private MemoryGameHomePage page;

	@Before
	public void openTheBrowser() {
		page = PageFactory.initElements(new FirefoxDriver(),
                                        MemoryGameHomePage.class);
		page.open("http://localhost:8888/memorygame/");
	}

	@After
	public void closeTheBrowser() {
		page.close();
	}

	@Test
	public void whenTheUserEntersNoValuesAnErrorMessageShouldAppear() {
		page.setGameName("");
		page.createGame();
		assertThat(page.getErrorMessage(), is("Value is required."));
	}
```

Step 2: Check the dynamic error messages

Error messages should appear as soon as the user presses RETURN. Check that, if the user enters an empty string and presses return, an error message is displayed.

Step 3: Refactoring

Refactor this class so that it uses an AbstractBasePage to define the open() and close() methods, and any other methods that you think should be common.

Lab 5 - Using FindBy annotations

Goal

The aim of this lab exercise is to add more complex features to our Page Object using the FindBy annotations.

Lab Exercises

Step 1: Integrating the dropdown lists

Add support for the two drop-down lists. Add a test to check that a new game can be created if the name and mandatory drop-down lists are filled in correctly. Since this is an AJAX application, you will have to add manual pauses at certain points.

Step 2: Multiple error messages

Add support for multiple error messages. Add a test that checks that several error messages are displayed if the fields are not filled in correctly. Can you use the FindBy annotation in this case?

Lab 6 - AJAX Support

Goal

The aim of this lab exercise is to add AJAX support to our Page Object.

Lab Exercises

Step 1: Integrate the `AjaxElementLocatorFactory` class.

Refactor the Page Object to use the `AjaxElementLocatorFactory` class. How does this enable us to simplify our Page Object?