4   Web layer test automation

v2025-03-05

Learning objectives

— Write web-layer tests using Selenium API and automate their execution in JUnit.

— Understand best practices for elements location and possible causes of flaky tests

— Apply the Page Object Pattern

Key Points

— Acceptance tests exercise the user interface of the system as if a real user was using the application. The system is treated as a black box.

— Browser automation (control the browser interaction from a script) is an essential step to implement acceptance tests on web applications. There are several frameworks for browser automation (e.g.: Puppeteer); for Java, the most used framework is the WebDriver API, provided by Selenium (that can be used with JUnit engine).

— The test script can easily get "messy" and hard to read. To improve the code (and its maintainability) we could apply the Page Objects Pattern.

— Web browser automation is also helpful to implement "smoke tests".

Useful resources

- B. Garcia's Book "Hands-On Selenium WebDriver with Java"

- Criticism on the Page Object Pattern for modern web apps (and alternatives).

4.1   WebDriver "hello world"

Selenium WebDriver offers a programming interface (i.e., API) to drive/control a web browser, as if a real user was operating the browser.

You are required to install the driver implementation (binding) for the specific browser you want to use. This step can be simplified with the help of a Web Driver management framework (such as included in the Selenium-Jupiter extension).
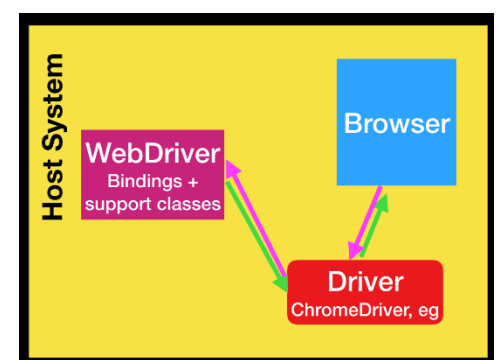
a) Implement the example discussed in the ["hello world" section](#) of B. Garcia's book (search "Example 2-1").

   Adapt to a custom scenario (choose another web site,...).

   Main [dependencies for POM.xml](#):

   - `Selenium 4: org.seleniumhq.selenium:`**`selenium-java`**
   - `JUnit extension: io.github.bonigarcia:`**`selenium-jupiter`**
   - `JUnit 5: org.junit.jupiter:`**`junit-jupiter`**

b) Enrich your test script to consider some other actions upon the "browser", such a sillustrated in the [getting started example](#).

c) Refactor the previous example to use the approach in "Example 2-2", based on [Selenium-Jupiter extension](#) (which uses dependency injection to get the WebDriver instance). Note the changes: (1) @ExtendWith(SeleniumJupiter.class); (2) use a dependency injection to get a "browser" instance; (3) no explicit need for "quit".

4.2 Interactive test recording

Often you can record your tests interactively to explore the "locators" (e.g.: id for a given web element) speeding up the test preparation.

Record the test interactively

a) Install the [Selenium IDE](#) plug-in/add-on for your browser or the alternative [Katalon Recorder](#) plugin.

   Using the https://blazedemo.com/ dummy travel agency web app, record a test in which you select a and buy a trip. Be sure to add relevant "asserts" or verifications to your test.

   Replay the test and confirm the success. Also experiment to break the test (e.g.: by explicitly editing the parameters of some test step in the test script).

b) Add a new step, at the end, to assert that the confirmation page contains the title "BlazeDemo Confirmation". Enter this assertion "manually" (in the editor, **not** recording).

c) Be sure to save you Selenium IDE test project (it creates a *.side file, to be included in your git).

Export and run the test using the WebDriver API

d) Export the test from Selenium IDE into a Java test class and include it in the previous project.

   You need to refactor the generated code to be compliant with JUnit 5! Select only the relevant parts from the generated ouput.

   Run the test programmatically (as a JUnit 5 test).

Refactor to use Selenium extensions for JUnit 5

Be sure to use the [Selenium-Jupiter extension](#). Note that this library will ensure several tasks:

- enable dependency injection with respect to the WebDriver implementation (i.e., hides the use of WebDriverManager to resolve the specific browser implementation). You do not need to pre-install the WebDriver binaries; they are retrieved on demand.
- if using dependency injection, it will also ensure that the WebDriver is initialized and closed.
- You may check this related example (consider only for the parts related to the WebDriver...).

4.3 Selecting elements with locators

Consider the test case of searching for a book in this (fictitious) online store.

a) Implement a simple test that, for a search for "Harry Potter", finds a result that corresponds to the book "Harry Potter and the Sorcerer's Stone" by J. K. Rowlling.

b) Check the locators being used. Are there any instances of "xpath"? What about identifier-based locators? Which locator strategies are more robust?

c) Create a new version of the previous test, considering a refactoring that (1) favors well defined selectors[1], and (2) increases robustness by introducing explicit waits that tolerate search response latency.

By.cssSelector("[data-testid=book-search-item]"

4.4 Page Object pattern

Consider the example discussed here (or, for a more in depth discussion, here).

Note: the target web site implementation may have changed from the time the article was written and the example may require some adaptations to run (i.e., to pass the tests). However, it is not mandatory to have the example running.

"Page Object model is an **object design pattern** in Selenium, where webpages are represented as classes, and the various elements [of interest] on the page are defined as variables on the class. All possible user interactions [on a page] can then be implemented as methods on the class."

a) Implement the "Page object" design pattern for a cleaner and more readable test using the same application problem from exercise **Error! Reference source not found.**, in a new project. Suggestion: use the Page Factory approach.

4.5 Browser variations

Consider using a browser that is not installed in your system. You may resort to a Docker image easily (see 3.4 Docker browsers section).

Note that, in this case, the WebDriver will connect to a remote browser (no longer direct communication) and you should Docker installed in your system.

---

[1] You may need the inspect the page elements/HTML to find identifiers.