If the “failsafe” maven plugin is configured, you should get different results with:
$ mvn test
$ mvn package failsafe:integration-test

Explore

JUnit 5 cheat sheet.

Lab 3: Functional testing with web automation

Prepare

- Selenium and separation of concerns

Key Points

- Acceptance tests (or functional test) 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 or TestNG engines).
- The test script can easily get "messy" and hard to read. To improve the code (and its maintainability) we could apply the Page Objects patterns.
- Web browser automation is also very handy to implement “smoke tests”.

Lab

Selenium works with multiple browsers but, for sake of simplicity, the samples will be discussed with respect to Chrome/Chromium; you may adapt for Firefox.

Suggested setup:
Install Chrome/Chromium in your system (if needed), using the default installation paths.
Download the ChromeDriver and make sure it is available available in the PATH. (GeckoDriver for Firefox)
Install the “Selenium IDE” browser plugin. (or, alternatively, the Katalon Recorder).

In this lab:

1. Create a web automation with Selenium IDE recorder
2. Run the test as a Java project (JUnit 5 + Selenium)
3. Use the Web Page Object pattern

1/ Create a web automation with Selenium IDE recorder

Access the Redmine demo site and create a temporary account (http://demo.redmine.org)
Be sure to logout before recording the test.
Create (record) an automation macro with the Selenium IDE recorder tool to test login (a quick start for Katalon is available, which is similar to Selenium IDE):

a) Open http://demo.redmine.org
b) Sign-in with your credentials
c) Assert that you have successfully logged in (by verifying the presence of the username)
d) Logout

... and Stop recording. Test your macro (replay).

Add a new step, at the end, to confirm that, after logout, the home shows the “Sign in” option present. Enter this assertion “manually” (in the editor, but not recording).

2a/ Run the test as a Java project (JUnit 5, Selenium)

Prepare a (new) project to run JUnit tests and Selenium (sample POM.xml available; alternative site).

Take note of the information in this page under “quick reference”; then, in the section “Local browsers”, pick the example that suites your setup and run the test (as you usually do with JUnit). You will have to deploy the WebDriver implementation (binary) for you browser [→ download browser driver]. Be sure to include in the system PATH.

2b/ Export and run the test (Webdriver)

Export the test from Selenium IDE into a Java test class and include it in the previous project. Refactor the code that was generated to be compliant with JUnit 5 and the Selenium-Jupiter extension.

Run the test (programmatically).

3/ Use the Web Page Object pattern

Consider the example discussed here.
Implement the “Page object pattern” for a cleaner and more readable test, as suggested.

Notes:
Execute the suggested steps to interactively record the test case. Export it to Java and run the test with JUnit automation, refactoring for JUnit 5.
The text in the tutorial is somewhat old. You may **need to adapt to the current implementation** of the site under test (e.g.: IDs of page elements,...).
You should **stop** at “Increased readability” with Cucumber.

Explore

- Puppeteer - a Node library which provides a high-level API to control headless Chrome/Chromium.
- Another, more recent, Page Object Model example.

Lab 4: Behavior-driven development (Cucumber in Java)

Key Points

— The Cucumber framework enables the concept of “executable specifications”: with Cucumber we use concrete examples to specify what we want the software to do. **Scenarios are written before production code**.

— Cucumber executes features (test scenarios) written with the Gherkin language (readable by non-programmers too).