Assignment 3 – The Article

Paper

Title: Selenium-Jupiter: A Junit 5 extension for Selenium WebDriver

Authors: Boni García, Carlos Delgado Kloos, Carlos Alario-Hoyos, Mario Munoz-Organero

Abstract (Copied from the paper)

Selenium WebDriver is a library that allows controlling web browsers (e.g., Chrome, Firefox, etc.) programmatically. It provides a cross-browser programming interface in several languages used primarily to implement end-to-end tests for web applications. Junit is a popular unit testing framework for Java. Its latest version (i.e., JUnit5) provides a programming and extension model called Jupiter. This paper presents Selenium-Jupiter, an open-source JUnit 5 extension for Selenium WebDriver. Selenium-Jupiter aims to ease the development of Selenium WebDriver tests thanks to an automated driver management process implemented in conjunction with the Jupiter parameter resolution mechanism. Moreover, Selenium-Jupiter provides seamless integration with Docker, allowing the use of different web browsers in Docker containers out of the box. This feature enables cross-browser testing, load testing, and troubleshooting (e.g., configurable session recordings). This paper presents an example case in which Selenium-Jupiter is used to evaluate the performance of video conferencing systems based on WebRTC. This example case shows that Selenium-Jupiter can build and maintain the required infrastructure for complex tests effortlessly.

©2022 Elsevier Inc. All rights reserved.

- Why? What problem are the author(s) trying to solve?

  They want to provide a comprehensive programming model for developing end-to-end tests for web applications using Selenium WebDriver on top of Junit 5 since as they say this contribution is completely missing in the literature as far as they know. With this they aim to solve current (2022) challenges in end-to-end testing. It's especially the maintainability and flakiness that seem to be the most relevant challenges in end-to-end testing.

  They are also trying to solve the difficulty in creating load tests. Apache JMeter that is commonly used for load tests is not suitable when actual browsers, implementing the WebRTC stack, are required to generate the load.

  The driver management is also an issue due to the relatively fast updates of modern browsers due to automated updates which, eventually, leads to incompatibilities between the browsers and the driver versions used in manual tests.

  Finally they are also trying to solve the problem with troubleshooting (aka failure analysis) for the Selenium WebDriver.

- How? What method has been used to answer the question?

They are using Docker containers set up to run Selenium WebDriver to drive browsers used in the test. Selenium WebDriver is a collection of tools that are designed to automate web application testing. It allows you to write code that can perform actions in a browser in the same way a human does.

To manage the different drivers used in the Docker containers for the browsers they used WebDriverManager a Java library for automated driver management.

Apache JMeter has been commonly used to evaluate the performance of web applications but it's not suitable for this kind of tests where actual browsers are required to generate the load. The solution they proposed is to integrate Docker in the Jupiter programming model.

In the discussed example case, the following research questions where:

- RQ1. How can Selenium WebDriver be used to evaluate the end-to-end performance of WebRTC videoconference applications?
- RQ2. What are the benefits and limitations of using Selenium-Jupiter to implement these kinds of tests?

- What is the result from the method?

  In regards to RQ1 they conclude that the WebRTC stats are meaningful mechanisms to evaluate different performance metrics such as Bit rate, Jitter delay, Packet loss etc. They can be gathered using Selenium WebDriver and webrtc-internals a tool available in Chrome.

  For RQ2 the main benefit of Selenium-Jupiter is the use of Docker, in the example case. This makes it possible to chose the type and number of required browsers. They regard scalability as the main drawback of this approach. Since Selenium-Jupiter uses a single Docker engine to handle browsers in containers it is constrained by the physical resources of the hardware used, i.e., RAM and CPU.

- What is the validity of the result according to the author?

- Does the research solve the problem/answer the question?

- What are to be done next? future work? and why?

- And lastly something from you. What do you think and why? If you make claims support those with evidence or logic.