University of Victoria
Engineering & Computer Science Co-op
Work Term Report
Spring 2021

Testing Frameworks for the Cloudscheduler Web Interface

University of Victoria
High-Energy Physics Research Center
Victoria, BC, Canada

Elisabeth Klassen
V00942394
Work Term 2
Software Engineering
elisabethklassen@uvic.ca
April 20, 2021

In partial fulfillment of the academic requirements of this co-op term

Supervisor's Approval: To be completed by Co-op Employer

This report will be handled by UVic Co-op staff and will be read by one assigned report marker who may be a co-op staff member within the Engineering and Computer Science Co-operative Education Program, or a UVic faculty member or teaching assistant. The report will be retained and available to the student or, subject to the student's right to appeal a grade, held for one year after which it will be deleted.

I approve the release of this report to the University of Victoria for evaluation purposes only.

Signature: [illegible] Position: Research Assistant Date: April 20th 2021

Name (print): Colson Driemel E-Mail: colsond@uvic.ca

For (Company Name) University of Victoria

Letter of Transmittal

Elisabeth Klassen
1815 Teakwood Road
Victoria, BC
V3G 2Y5

February 6, 2021

Dr. Ash Senini
3800 Finnerty Road
Victoria, BC
V8P 5C2

To whom it may concern,

This is the coop report "Testing Frameworks for the Cloudscheduler Web Interface", written as part of Elisabeth Klassen's second work term (term 2A, Software). This coop was undertaken under the supervision of Randall Sobie, at the High Energy Physics Research Center (HEPRC) at the University of Victoria.

The Cloudscheduler v2 web interface, one of the projects run by this group, required a set of tests for the web framework. This report will look at three possible testing frameworks for the website – Selenium and Unittest, Selenium and Behave, and Testcafe – and analyze which of the three would be best, in terms of code maintainability and readability, ease of setting up and running the tests, and ease of creating and editing the tests. It will analyze the strengths and weaknesses of each framework and present a final decision on which one would be best for this project.

The team at HEPRC was extremely helpful, in both the decision-making process and the development of these tests. Special thanks to Colson Driemel for his assistance with coding errors and locating bugs in the web frontend.

Elisabeth Klassen
HEPRC

Attachment: Testing Frameworks for the Cloudscheduler Web Interface

Table of Contents

Summary

The Cloudscheduler web interface requires a set of unit tests to ensure its proper functionality. Three frameworks - Selenium and Unittest, Selenium and Behave, and Testscafe - were evaluated for their efficiency in creating these tests.

The criteria established were readability (the ease of a user writing and running the tests to understand what is occurring), maintainability (ease of setup and creation of tests), and scope (the amount of features the framework could test). Each framework was evaluated using an analysis of these features and a short sample test suite.

It was concluded that Selenium and Unittest would make the best framework. Going forward, a suite of these tests should be created to test the Cloudscheduler web interface.

Glossary

Virtual machine (vm)	A file that behaves as if it were a computer [1]
Selenium	A program that interacts with various webdrivers to allow remote operation of a browser [2]
Unittest	A testing framework built into python [3]
Behave	A gherkin-language testing framework for python [4]
TestCafe	A JavaScript web testing framework that allows remote operation of a browser [5]
Webdriver	A program that remotely runs one or more different web browsers [6]
Geckodriver	A webdriver for Mozilla Firefox [7]
Gherkin	A human-readable plaintext language for writing tests [8]

Introduction

The Cloudscheduler v2 program (from now on referred to as “Cloudscheduler”) is a program to boot and organize virtual machines for scientific projects around the globe [9]. It organizes computing resources for many important projects and schedules jobs on virtual machines for many of said projects.

Cloudscheduler is written primarily in Python3. It uses a set of pollers to transfer information on the states of clouds and jobs into a database [9]. Using the data from this database, it boots virtual machines appropriately for the jobs that are waiting to be run, based on a series of internal criteria [9]. It has been in use since early 2019 [9].

Cloudscheduler has both a command-line interface and a web interface. [9] The command line interface has a set of unit tests to ensure its proper functionality, while the web interface does not. This means that the web interface requires manual testing to find bugs, which can often be unreliable or miss certain problems. Additionally, it takes a lot of time.

This report will discuss three possible testing frameworks for creating the web tests – one using Selenium and Unittest, one using Selenium and Behave, and one using TestCafe.

Objective

The Cloudscheduler web interface currently lacks a set of tests to ensure its proper functionality when new features are developed. Thus, the goal is to choose a proper framework for these web tests to be implemented.

The tests should be easily understandable, both by the creators, any additional contributors, and by people who need to run them. They also should be easy to set up, run and maintain. Additionally, the tests should have a wide scope of usable browsers and operating systems, so the tests can be ensured to be reliable.

The test framework chosen must have the wide variety of features necessary to test the Cloudscheduler interface. The framework must have a license that works with Cloudscheduler as an open-source project. As well, the framework chosen must be such that a developer assigned to it can create the test framework within a period of several months.

This report will look at these frameworks in terms of the goals set out above – comprehensibility, maintainability, and scope. However, the scope of the report will be limited to the three frameworks investigated and will not cover any additional concerns that could arise outside of the three goals previously stated.

Discussion

This section of the report will investigate the three frameworks – Selenium and Unittest, Selenium and Behave, and TestCafe – in detail. It will discuss the advantages and disadvantages of each, with the goals of comprehensibility, maintainability, and scope in mind.

Selenium and Unittest

The first framework under consideration is a combination of Selenium and Unittest. Selenium is a webdriver interface designed to be usable with most modern browsers and a variety of languages [2]. Unittest is a built-in Python testing framework using classes to provide the tests [3]. This section will discuss the advantages and disadvantages of this framework for the Cloudscheduler testing framework.

Advantages

The Selenium and Unittest framework has a variety of advantages. It is easy to understand for anyone who programs in Python, the setup and maintenance is simple, and it has a wide scope.

Selenium and Unittest tests are fairly easy to understand, especially for the people who would be programming Cloudscheduler. They are written in Python [3], like the majority of the codebase, meaning that programmers who work with the rest of the codebase will understand them. The tests are also written in a traditional Python style, which makes it easy to read through the tests and understand how variables are being passed and setup is being done. Additionally, for those who are running the tests, the output they produce is simple and readable, especially when run to be more verbose [3]. It is easy to tell when tests are failing, and when run with the verbose argument, the tests and their containing classes are named as they are run, making it simple to track down the errors [10].

The setup and maintenance for these tests is also simple. One needs to use pip, the Python package manager, to install the Selenium bindings – the code that allows Selenium to communicate with Python [11]. Then, in order to make Selenium work, one must also install the drivers for the browsers one wants to test on [12]. Unittest, however, comes preinstalled with Python [3], meaning that no import is necessary – the programmer may simply import the module into their code and use it. The tests are also easy to maintain – they are written in Python, like the rest of the codebase, and use traditional Python styles for writing.

Additionally, the scope of testable features is large using Selenium and Unittest. Selenium is usable on almost any modern browser because it interacts with drivers created for each individual browser [12]. It can also perform a wide variety of interactions, including clicks, drag-and-drop actions, and typing [13]. In the event that the Selenium bindings do not provide support for an action, they also provide a method to allow the programmer to execute JavaScript (ie, to use code that directly interacts with the webpage) to perform an action [14].

Disadvantages

On the other hand, the Selenium and Unittest framework has a variety of disadvantages as well. The tests can be prohibitive to understand for some users, the Selenium setup has significant drawbacks, and the scope is limited by individual drivers and the features they implement.

While Selenium and Unittest tests are, overall, readable for anyone who works in Python, this breaks down when the person reading the tests is not a programmer. Python, overall, is a human-readable language, but the plain text descriptions of the tests are often short and rarely descriptive unless the creator of the tests devotes significant time to rectifying this [3]. For those who need to run the tests without fully understanding code, there are idiosyncrasies in the way Selenium interacts with the browser that can make the actions of the tests oftentimes confusing.

The Selenium setup also has some major disadvantages, the main one being the use of drivers. Each browser or family of browsers Selenium interacts with requires a different driver [12]. Each driver is created by a different developer – typically the developer of the browser – and each driver must be installed independently [12]. If one wants to run tests across multiple browsers, as is necessary for the testing of the Cloudscheduler interface, this makes the setup increasingly difficult, as each driver's installation procedure must be added to the setup.

The scope of Selenium is also limited by the drivers themselves – and, sometimes, by features within Selenium that have not been implemented. For example, Selenium has no way to fill out popups with multiple blanks [15], and, as one of these is used for the Cloudscheduler login, it requires a workaround. Additionally, each driver is implemented by a different group of people [12]. While Selenium has a standard for which features should be available in a driver, features will vary by driver and driver creator [16].

Selenium and Behave

The second framework to consider is a combination of Selenium and Behave. Selenium, as explained above, is a cross-browser, cross-language webdriver interface [2]. Behave is a gherkin language testing framework that uses Python as its underlying implementation. This section will discuss the advantages and disadvantages of this framework.

Advantages

The Selenium and Behave framework has several advantages. They use readable test scripts, they have a great amount of reusability, and they have a large feature scope.

Test scripts written with Selenium and Behave are easily human-readable. The test files themselves are written in Gherkin language, which is a stylized form of plain English that allows tests to be run, making them understandable even to someone who doesn't code [4]. The implementations, while less readable, are written in Python, like the majority of the codebase, and are directly tied to the plaintext steps within the code, so someone reading them can always see what the step is intended to do [4]. Additionally, the outputs for the tests are readable. Each Gherkin step is listed, as is its status (whether it succeeded, failed, or was skipped) and the time it took to execute [10]. This means the tests iterate through each action they take live in front of the user.

These scripts also have a fair amount of reusability. Because each step is implemented in a different place from where it is written [4], the steps can be reused across multiple tests without having to modify any part of the test scripts. This can make the tests easier to maintain and add to, because new tests can build upon steps used in the previous ones, and code does not need to be written out multiple times. The setup is also simple – both Selenium [11] and Behave [17] can be installed via pip. Selenium drivers can then be installed separately [12].

Selenium and Behave have a wide range of testable features, as well. Like in the Unittest framework, above, the webdriver used is Selenium, which supports a wide variety of frameworks and actions [2]. Selenium can be used in almost any browser, because of its use of browser-specific webdrivers and an interface that implements them all [12]. It also can test a majority of the features a web site possesses [13], and can use JavaScript commands when its own interface fails [14].

Disadvantages

Selenium and Behave also has some significant disadvantages. The code can be very difficult to read from a programmer's perspective, test steps are hard to find and write, and the Selenium setup has several implementation drawbacks.

The Selenium and Behave framework has its own difficulties in terms of readability. While the test scripts themselves are easy to understand from a reading perspective, finding what they do in terms of code is much more difficult. The implementation steps are stored in different files from their plaintext counterparts [4], meaning that, in order to find what is actually happening in a single test, one must look up each step in the plaintext file and locate it in the Python file. This is complicated by the fact that neither file needs to indicate where a particular step is from - Behave draws all the functions together and implements them as necessary [4]. In a large testing framework, this can make it difficult to find how any step was written. Additionally, the Behave framework has its own rules for how test scripts must be written, matching the Gherkin language framework, which, while fairly intuitive, has its own idiosyncrasies that need to be learned [4].

Additionally, the Selenium and Behave framework has significant maintainability problems. In addition to making it difficult to find the test steps, as above, the added level of complexity makes creating and using the tests more difficult. The steps are individually called, directly from the test files, meaning that all variables passed must be in Behave's context variable [4]. This makes passing many types of data, such as flags, quite difficult, and means that a prior step must know what data is to be passed to each future step. It also, especially when combined with the problems of reading the test steps, makes it difficult to track how variables were assigned and fixtures were set up, as everything was assigned in an arbitrary setup step that is not easy to find. This makes maintaining and learning the codebase difficult. Additionally, the setup problems requiring multiple Selenium drivers also apply [12].

The scope of Selenium itself also has significant limitations, as discussed above. It is dependent on the browser drivers themselves to implement the necessary features. [16] It also has several idiosyncrasies, including particular objects and types that are unsupported, that make it more difficult to properly test in a web browser [15], although these often have workarounds.

TestCafe

The third framework to consider is called TestCafe. TestCafe is a combination webdriver and testing framework, created as an alternative to Selenium and written in JavaScript. This section will discuss the advantages and disadvantages of TestCafe.

Advantages

There are several advantages to using TestCafe for the codebase. The test outputs are simple and easy to understand, the code has many maintainability advantages, and the scope of testable features is quite wide.

TestCafe's test output is simple and understandable. It lists the browser and operating system it is operating under, then goes through the tests one at a time, putting a check mark or an x when they are complete [10]. A single time estimate is printed at the end. While minimalistic, this makes it easy to see at a glance what is happening with the tests and where they are failing. The test names are also written as plaintext descriptions, making it simple to see at a glance, even for a non-programmer, which tests are working incorrectly.

TestCafe has many maintainability advantages over the other two frameworks. The primary one is that it is entirely within one framework. There is no concern when the programmers update a part of the system as to whether TestCafe will get out of sync with itself, because TestCafe is entirely one module, and so will always be compatible with itself. Additionally, the TestCafe install is simple. It requires a package called npm (node package manager), which may or may not be installed on the test machine, and if not can be installed through the default package manager [5]. However, from within npm, it only requires a single installation command to set up.

TestCafe also has scope advantages the other two frameworks do not possess. As well as being able to test a large variety of browsers, TestCafe comes with support for testing under various operating systems, so code can be tested across Windows, MacOS, Ubuntu, and more. TestCafe is also written in JavaScript, meaning that it can interact directly in JavaScript. While Selenium can do this [14], TestCafe is designed in this language.

Disadvantages

TestCafe also comes with its own set of disadvantages. It is difficult to read, has some issues with maintainability, and has its own set of scope challenges.

One of the biggest disadvantages of TestCafe is that it is quite difficult to read. TestCafe is written in JavaScript, a language that sees little use in the codebase [5]. This means that anyone attempting to learn the tests would need to learn JavaScript on top of the code required to understand the codebase. Additionally, JavaScript syntax is significantly different from most of the code used in the codebase, meaning that it would not be intuitive to anyone who is used to the usual code on Cloudscheduler. This makes it prohibitive to teach to new employees, and even more so for current employees to add to their repertoire. While there would be some employees who know and use JavaScript, the incongruity with the rest of the codebase makes the tests prohibitive to learn.

TestCafe also has several maintainability issues. The largest difficulty with the maintainability of TestCafe is that it is fundamentally different from most of the codebase. This means it runs on its own update schedule and its own set of features, meaning that upgrades are likely to be inconsistent with upgrades to the codebase. Additionally, TestCafe is impossible to integrate with the current testing framework, aside from a script that could run both test setups. This means that the testing framework now becomes two testing frameworks, increasing the difficulty of maintaining it. Additionally, TestCafe's setup relies on a separate package manager [5], meaning the complexity of the setup is increased.

There are also a few scope issues with TestCafe. While TestCafe supports a large number of browsers, operating systems, and features, there are a few missing functionalities. For example, TestCafe has no way to select by XPath, a common language syntax used to find elements on a webpage [18].

Conclusions

In conclusion, the Unittest and Selenium framework is the best framework for the demands of the Cloudscheduler interface.

Unittest and Selenium is much simpler to write in and understand than either of the other frameworks. Behave and Selenium uses a plaintext mapping that makes it difficult to find steps and follow the passing of variables. TestCafe is written in JavaScript, a scripting language rarely used elsewhere in the code with its own steep learning curve.

Unittest and Selenium is also fairly maintainable compared to the other two options. Unittest, being a built-in framework, needs no outside updates or configuration. Neither Behave nor TestCafe has this feature. Selenium does require more maintenance, but the main issue is caused by the browser drivers, which can be independently added and removed.

Additionally, Unittest and Selenium is a test framework with a fairly wide scope. Unittest has a significant list of useful features, and Selenium can replicate the vast majority of browser features, in the majority of browsers. Behave and Unittest and TestCafe are also fairly solid in this regard, but Unittest and Selenium handles this very well.

Overall, with regards to the three goals as a whole, Selenium and Unittest does the best job of meeting the requirements and is the best framework to use for the job.

Recommendations

With this conclusion in mind, the best path forward is to create a set of browser-based tests in Unittest and Selenium.

These tests should leverage the full abilities of this framework to create cross-browser tests that test all of the many features of the Cloudscheduler interface and the various pages used to implement it. With reference to the earlier goals, the tests should be written and documented to be as clear as possible, including the use of clear variable and function names, descriptive comments, and useful documentation. They should, additionally, use the minimum amount of additional configuration and setup steps necessary for their successful operation, and should ideally have options to use or not use setups that require extra configuration.

References

[1] Microsoft, "What is a virtual machine?," Microsoft Azure, 2021. [Online]. Available: https://azure.microsoft.com/en-ca/overview/what-is-a-virtual-machine/. [Accessed 31 March 2021].

[2] Selenium, "The Selenium Project and Tools," 20 February 2021. [Online]. Available: https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/. [Accessed 20 February 2021].

[3] Python Software Foundation, "unittest — Unit testing framework," Python Software Foundation, 20 February 2021. [Online]. Available: https://docs.python.org/3/library/unittest.html. [Accessed 20 February 2021].

[4] B. Rice, R. Jones and J. Engel, "Tutorial," ReadTheDocs, 2017. [Online]. Available: https://behave.readthedocs.io/en/stable/tutorial.html. [Accessed 20 February 2021].

[5] Developer Express Inc, "TestCafe," Developer Express Inc, 2021. [Online]. Available: https://devexpress.github.io/testcafe/. [Accessed 20 February 2021].

[6] Selenium, "Webdriver," The Selenium Project, 20 February 2021. [Online]. Available: https://www.selenium.dev/documentation/en/webdriver/. [Accessed 20 February 2021].

[7] Mozilla, "README," GitHub, 3 March 2020. [Online]. Available: https://github.com/mozilla/geckodriver/blob/master/README.md. [Accessed 20 February 2021].

[8] Cucumber, "Gherkin Reference," SmartBear, 2019. [Online]. Available: https://cucumber.io/docs/gherkin/reference/. [Accessed 31 March 2021].

[9] F. Berghaus, K. Casteels, C. Driemel, M. Ebert, F. F. Galindo, C. Leavett-Brown, D. MacDonell, M. Paterson, R. Seuster, R. J. Sobie, S. Tolkamp and J. Weldon, "High-Throughput Cloud Computing with the Cloudscheduler VM Provisioning Service," *Computing and Software for Big Science,* no. 4, 2020.

[10] E. Klassen, "Additional Notes," 19 January 2021. [Online]. Available: https://github.com/hep-gc/cloudscheduler/blob/web-testing/web_tests/additional_notes.md. [Accessed 27 February 2021].

[11] The Selenium Project, "Installing Selenium Libraries," 27 February 2021. [Online]. Available: https://www.selenium.dev/documentation/en/selenium_installation/installing_selenium_libraries/. [Accessed 27 February 2021].

[12] The Selenium Project, "Driver Requirements," The Selenium Project, 27 February 2021. [Online]. Available: https://www.selenium.dev/documentation/en/webdriver/driver_requirements/. [Accessed 27 February 2021].

[13] B. Muthukadan, "Navigating," ReadTheDocs.io, 2018. [Online]. Available: https://selenium-python.readthedocs.io/navigating.html. [Accessed 27 February 2021].

[14] D. Bhattacharjee, "How to click on a button with Javascript executor in Selenium with python?," TutorialsPoint, 28 July 2020. [Online]. Available: https://www.tutorialspoint.com/how-to-click-on-a-button-with-javascript-executor-in-selenium-with-python. [Accessed 27 February 2021].

[15] A. Tolfsen, "Missing support for HTTP authentication prompts," GitHub, 4 October 2016. [Online]. Available: https://github.com/w3c/webdriver/issues/385. [Accessed 27 February 2021].

[16] The Selenium Project, "Driver specific capabilities," Selenium, 27 February 2021. [Online]. Available: https://www.selenium.dev/documentation/en/driver_idiosyncrasies/driver_specific_capabilities/. [Accessed 27 February 2021].

[17] B. Rice, R. Jones and J. Engel, "Installation," Behave, 2017. [Online]. Available: https://behave.readthedocs.io/en/stable/install.html. [Accessed 5 April 2021].

[18] TestCafe, "Support XPath Based Selectors," TestCafe, 26 January 2017. [Online]. Available: https://github.com/DevExpress/testcafe/issues/1178. [Accessed 27 March 2021].