WebdriverIO

An Automated Browser Testing Framework

Prepared By

Ottor Mills	ID#: 180917	Email: 180917@gist.edu.cn
David Thomas	ID#: 180912	Email: 180912@gist.edu.cn
Nicoy Smith	ID#: 180902	Email: 180902@gist.edu.cn
Kenneth Anglin	ID#: 180907	Email: 180907@gist.edu.cn

Course Instructor: Thomas Canhao Xu
Course: SWEN3010
Date: April 30, 2019

Table of Contents

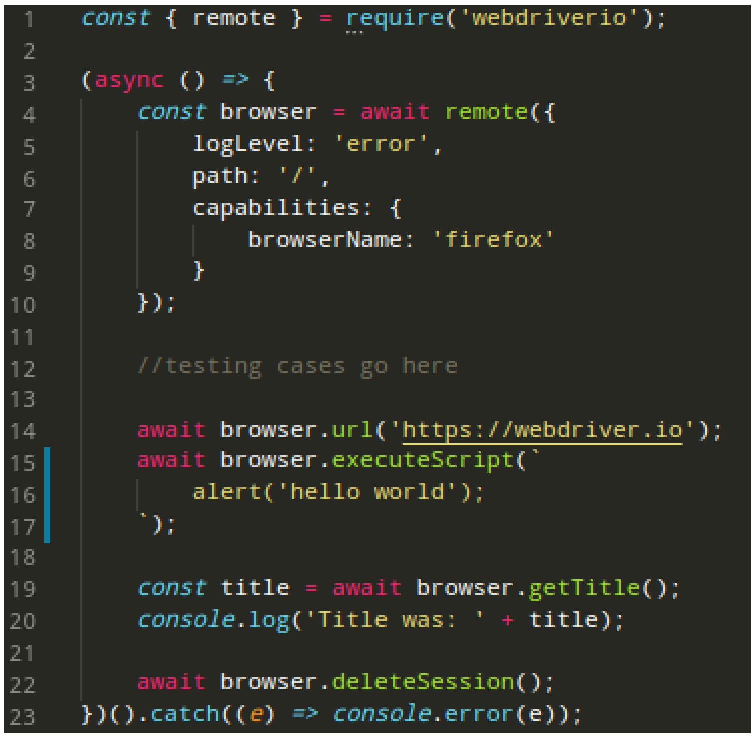
What is WebdriverIO

WebdriverIO is an an automated web testing framework that implements bindings of Webdriver/Selenium 2.0 in NodeJS. A key advantage of WebdriverIO is it uses Javascript which createas a consistent programming experience in terms of the testing language, whether it be manipulating the components of the webpage using client-side javascript or creating test cases using the WebdriverIO library.

Another advantage of WebdriverIO is how relatively easy it is to set up. WebdriverIO's setup consists of downloading the webdriver module using the node package manager (npm) and running a setup. Once setup is complete test scripts can be written, and a browser of choice can be selected by referencing its driver in the configuration file. Once the browser driver has been started WebdriverIO will connect to it as utilize it to conduct test cases.

Writing a Simple Test

Below is a simple test to open the WebdriverIO homepage in a firefox window.



```
1   const { remote } = require('webdriverio');
2
3   (async () => {
4       const browser = await remote({
5           logLevel: 'error',
6           path: '/',
7           capabilities: {
8               browserName: 'firefox'
9           }
10      });
11
12      //testing cases go here
13
14      await browser.url('https://webdriver.io');
15      await browser.executeScript(`
16          alert('hello world');
17      `);
18
19      const title = await browser.getTitle();
20      console.log('Title was: ' + title);
21
22      await browser.deleteSession();
23  })().catch((e) => console.error(e));
```

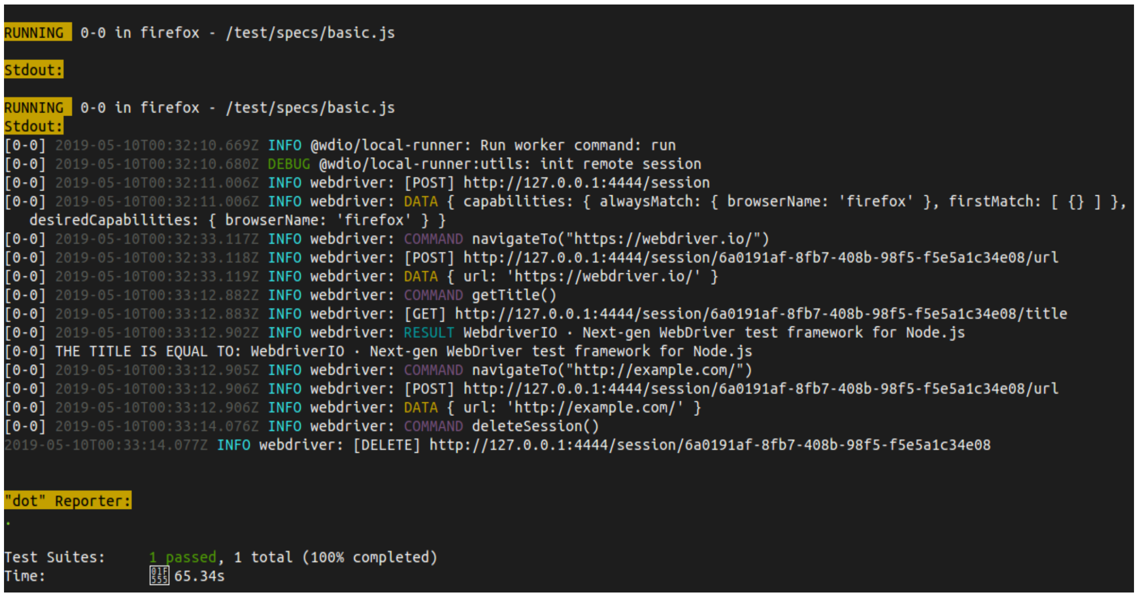
Code Explanation:

Step	Description
Line 1	The WebdriverIO module is being imported
Line 3	An await function is created and ran on definition (see in line 23). Since all functions made to the browser window are asynchronous the await function will enable these functions to be written in a traditional programming fashion instead of promise chaining.
Line 4	The browser object is being defined using the remove function that is provided in the WebdriverIO module. This function takes a single json object

	with the keys as shown. After this is completed it a new browser window is opened
Line 14	The browser is instructed to open the WebdriverIO homepage
Line 15	The browser executes some javascript code. It will display an alert message
Line 19	The browser gets the title of the current tab and it is assigned to the title constant
Line 20	The result (title) is displayed in the console
Line 23	The function is called and any errors / exceptions encountered are logged to the console

This sample above is only a quick approach to run a simple test. As shown by the code, a detailed outline of which steps passed and which steps did not are not shown. In other words it lacks verbosity. In order for test to provide a detailed outline the testrunner.

Testing with Testrunner



```
RUNNING  0-0 in firefox - /test/specs/basic.js

Stdout:

RUNNING  0-0 in firefox - /test/specs/basic.js
Stdout:
[0-0] 2019-05-10T00:32:10.669Z INFO @wdio/local-runner: Run worker command: run
[0-0] 2019-05-10T00:32:10.680Z DEBUG @wdio/local-runner:utils: init remote session
[0-0] 2019-05-10T00:32:11.006Z INFO webdriver: [POST] http://127.0.0.1:4444/session
[0-0] 2019-05-10T00:32:11.006Z INFO webdriver: DATA { capabilities: { alwaysMatch: { browserName: 'firefox' }, firstMatch: [ {} ] },
   desiredCapabilities: { browserName: 'firefox' } }
[0-0] 2019-05-10T00:32:33.117Z INFO webdriver: COMMAND navigateTo("https://webdriver.io/")
[0-0] 2019-05-10T00:32:33.118Z INFO webdriver: [POST] http://127.0.0.1:4444/session/6a0191af-8fb7-408b-98f5-f5e5a1c34e08/url
[0-0] 2019-05-10T00:32:33.119Z INFO webdriver: DATA { url: 'https://webdriver.io/' }
[0-0] 2019-05-10T00:33:12.882Z INFO webdriver: COMMAND getTitle()
[0-0] 2019-05-10T00:33:12.883Z INFO webdriver: [GET] http://127.0.0.1:4444/session/6a0191af-8fb7-408b-98f5-f5e5a1c34e08/title
[0-0] 2019-05-10T00:33:12.902Z INFO webdriver: RESULT WebdriverIO · Next-gen WebDriver test framework for Node.js
[0-0] THE TITLE IS EQUAL TO: WebdriverIO · Next-gen WebDriver test framework for Node.js
[0-0] 2019-05-10T00:33:12.905Z INFO webdriver: COMMAND navigateTo("http://example.com/")
[0-0] 2019-05-10T00:33:12.906Z INFO webdriver: [POST] http://127.0.0.1:4444/session/6a0191af-8fb7-408b-98f5-f5e5a1c34e08/url
[0-0] 2019-05-10T00:33:12.906Z INFO webdriver: DATA { url: 'http://example.com/' }
[0-0] 2019-05-10T00:33:14.076Z INFO webdriver: COMMAND deleteSession()
2019-05-10T00:33:14.077Z INFO webdriver: [DELETE] http://127.0.0.1:4444/session/6a0191af-8fb7-408b-98f5-f5e5a1c34e08


"dot" Reporter:
.

Test Suites:      1 passed, 1 total (100% completed)
Time:             65.34s
```

Testrunner is function that is included in the WebdriverIO module. Testrunner makes the testing experience much easier by providing a plethora of useful features. Shown above is the same test mentioned previously but executed using the testrunner. As shown there is a high level of verbosity. This may come in handy when debugging errors. Multiple test cases may be tested asynchronously using multiple threads.

Firefox web browser also has a headless mode which uses a minimal amount of system resources. This results in a testing environment where hundreds or even thousands of test cases can be completed relatively quickly. This is useful especially during vulnerability testing as many possible inputs may be entered into a text input field.

During setup, Testrunner provides the user with options such as the use of a wide range of wel known 3rd party testing add-ons such as Mocha, Cucumber or Jasmine as a testing framework and xunit, dot or spec as the reporter.

Interacting with Webpage Elements

In order to test web components the tester has two options. They can either use the *executeScript* method the WebdriverIO module and execute client-side javascript in the browser which can be used to fill forms, click buttons among other things. The Jquery framework can be used to greatly enhance the testing experience by simplifying native javascript functions. There also exists the option of simulating keystrokes using the built in functions of the WebdriverIO module such as *pressKeyCode* which mimics keystrokes on the keyboard to enter data in order to circumvent javascript blocked data entry used on some websites to provide some level of protection against bots.

Testing SQL Injection Vulnerability

A server may be tested for Sql Injection vulnerabilities by submitting malicious text via web forms or parameters in the browser's address bar. Sqlmap is "an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers". As such if the primary aim of a particular set of test cases is to expose any vulnerabilities through Sql Injection then Sqlmap is a very useful tool. It saves time by preventing the need to write additional test cases. Sqlmap is also compatible with a large array of relational databases.

Although the stock sqlmap supports only GET requests it can be modified to support other http request methods. Additional information about SqlMap can be found at the following resource: http://sqlmap.org

Cloud Testing Service Integration

WebdriverIO is compatible with a few cloud based testing services such as Sauce Labs, Browserstack and TestingBot. The implementation of these services is made simple through the use of a user and key provided by each of the services. Once WebdriverIO has been configured the test cases will be send to the respective cloud service where they will be executed in an enterprise setting, subjecting software to a faster and more thorough testing. Cloud testing services usually subject software to intense or real world traffic.

DDOS attacks are a common type of malicious attack that cost organizations millions of dollars annually. Executing a DDOS attack required an intense amount of resources which is in most cases unavailable with regular, small scale software testing. As such a more through test is done and a broader spectrum of test cases are carried out.

Screenshots

WebdriverIO allows for the saving of screenshots. This is a very useful feature in software testing because it allows for graphical samples of the user interface after each test case have been run and can be be referenced at any time. Screenshots of the user interface can also be taken in headless mode. Once the tests are complete the debuggers (or image processing software) can analyze the screenshots and identify any anomalies to fix or improve.

Distribution of Tasks

ID	Contribution (%)
180917	30
180912	23.3
180902	23.3
180907	23.3