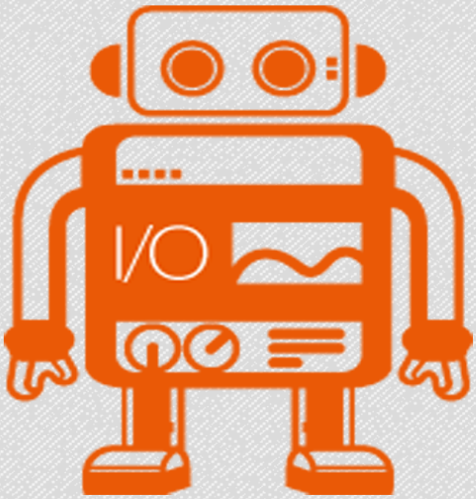

Get Started with Visual Regression Testing and WebdriverIO

Micah Godbolt
Kevin Lamping

Get Started with Visual Regression Testing and WebdriverIO

Kevin Lamping and Micah Godbolt

This book is for sale at http://leanpub.com/visual-regression-testing-and-webdriverio-guide

This version was published on 2017-02-21

Contents

Preface

The content for this book is from learn.visualregressiontesting.com[1]. It is a 6-day email course available for anyone interested in Visual Regression Testing.

Throughout this book, you'll notice mentions of the current day in the course (e.g. "It's Day 1!"), or references to previous/later days. This is due to the nature of the original content.

Each chapter in this book matches a day in the course. So six chapters is equal to six days.

With that said, let's get started!

[1] http://learn.visualregressiontesting.com

What to know about Visual Regression Testing

Hey Folks, Micah and Kevin here from visualregressiontesting.com.

Over the next week, you're going to learn the essentials Visual Regression Testing.

You may be surprised by how much fun writing these tests will be. After all, you're teaching a computer how to use your website! That's pretty neat in our book.

Some Terminology

Let's kick it off by expanding our vocabulary. We'll be using these terms throughout the course:

Regression Testing

Change hurts. Especially when it's unwanted.

"Regressions" are changes, bad ones, to the functionality of your site. To say you've found a "regression" means you found code that used to work but no longer does.

"Regression Testing" is a type of testing that checks for these bugs in existing functionality after an update.

While you definitely want to put new code through the wringer, it's also important to look at existing features to ensure they weren't adversely affected by the updates. That's what regression testing does.

Automated Testing

Automation comes in many forms, whether through robots or driverless cars. For us, it's writing code that "automates" actions on a website, similar to how AI is written to "automate" driving a car.

While the goal of website automation is to test without a human clicking links and entering text, in no way can it take the place of hard-working people.

First of all, someone has to write the automation and know how to keep it up to date. That can easily be a full-time job on a large enough application.

But there is also a lot of nuance to websites. Checking to see if an animation worked as designed is very difficult with automated tests. Automation simply aims to handle the boring, repetitive tasks, leaving you time to test the hard stuff.

Functional Testing

There are many types of software testing out there; Wikipedia has an extensive list of the various definitions[2].

We're focusing on "functional" testing, which checks the UI functionality of a website and ensures that it works properly. This is not the same as testing JavaScript functions (that would be unit testing).

Functional testing also goes by "system testing", "end-to-end tests" and many other monikers, but for our course we'll use the term "functional testing".

Visual Testing

There's one last term to cover, and that's the concept of "visual testing." In this course, we'll talk about both standard testing (e.g. make sure certain text is on the page) and visual testing (e.g. make sure the page looks the same as before).

Visual Regression Testing has a unique set of benefits. Just because an element is on a page does not mean it's in the right location (or the right size or the right color).

With visual testing, we take screenshots of various parts of the page to serve as the "baseline" of how the site should work. Then in subsequent test runs, we take new screenshots and compare to those baselines. Any differences are flagged for review.

The Benefits of Functional Testing

Why take the time to learn all of this? Is it really worth it to write code that only tests the code you really should be writing?

Here's the thing: you're already doing this. Every time you refresh your page to see a change you've made, you're running a manual regression test. Throughout the lifetime of a website, hundreds of hours are spent regression testing it.

But the truth is, we're bad at this. We either get lazy and skip a few tests, or we unknowingly miss a defect or two (we're especially bad at detecting small visual changes[3]).

With automation, regression testing the site becomes less of a repetitious chore, so we're more likely to do it on a consistent basis. You can even set up hooks to automatically test your code after every change.

And computers are much better at spotting pixel changes. It's all numbers to the computer, and numbers are easy to compare.

[2] https://en.wikipedia.org/wiki/Software_testing#Testing_types
[3] https://en.wikipedia.org/wiki/Change_blindness

Some Disclaimers

We really believe in the benefits of automated testing, so we should probably throw out some disclaimers before we get you too excited:

- Functional testing can be finicky, because websites are complicated.
- You can't test everything because, again, websites are complicated.
- Automation doesn't work in all browsers, because some browsers just don't support it. You're best bets are going to be Firefox, Chrome and Internet Explorer.

Basically, it's not magic (although it feels like it at times). There are certain things you just won't be able to automate testing for, so it's best not to fight that fact.

The Technology

We'll go in to installation of the tools tomorrow, but here's what we'll be using:

- Selenium[4] - A browser automation tool
- WebdriverIO[5] - A Node.js library to talk to Selenium
- WebdriverCSS[6] - An add-on to WebdriverIO to help with Visual Regression Testing

Course Requirements (aka today's homework)

There are a few requirements you'll need to have going before getting started:

- A working NodeJS environment[7] (Learn about Node and NPM if you are new to it[8])
- A light understanding of the command line[9]
- An understanding of JavaScript fundamentals[10]

What's Next?

Phew, you've made it through the entire first day without any code being shown. Don't worry, we've saved all that for tomorrow, where we'll take a look at writing your very first automated test (we're so excited for you!)

[4] https://en.wikipedia.org/wiki/Selenium_(software)
[5] http://webdriver.io/
[6] https://github.com/visualregressiontesting/webdrivercss
[7] https://github.com/creationix/nvm#node-version-manager-
[8] https://docs.npmjs.com/getting-started/what-is-npm
[9] https://www.codecademy.com/learn/learn-the-command-line
[10] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Basics

Today you're going to write your first regression test

Day 2 is here; we hope you're ready to dig in. Let's start with some installs!

First, create a folder to store all of your code. You can name it whatever you want.

Then, open a command line and navigate to that folder. From there, you'll create a new `package.json` file by running the command:

```
npm init -y
```

If you're new to NodeJS or unfamiliar with NPM, have a look at their about page for a quick intro video[11].

This file just stores information about your project and the software you need. It's useful to have on hand if you need to set the project up again later.

Now to run the actual installation:

```
npm install --save webdriverio visualregressiontesting/webdrivercss chromedriver
```

WebdriverIO

WebdriverIO is a JavaScript functional test library. It lets you write test instructions in JavaScript, then passes them along to Selenium, which tells the browser what actions to take.

For us, writing tests in JavaScript is awesome. We don't have to learn Java for test automation!

WebdriverCSS

WebdriverCSS is a Visual Regression Testing plugin for WebdriverIO. It's a pretty basic tool that boils down to two basic functions:

1. Capture images of the specified portion of the website.
2. Compare those images to previous ones, creating a "diff" version if differences are found.

You can do a fair amount of configuration with the tool, but for now, those two concepts are all you need to know.

[11] https://docs.npmjs.com/getting-started/what-is-npm

ChromeDriver

We're testing in real browsers folks, which is really quite awesome. But it does require some set up. While getting Selenium running is easier than ever, it's too complex for a single email.

As a worthy alternative, we're going to use ChromeDriver. ChromeDriver is a Selenium-like tool that allows us to run tests on a real Chrome browser.

To get it going, just open a new command line window (making sure you're in your project folder), and run:

```
./node_modules/.bin/chromedriver --url-base=/wd/hub --port=4444
```

This will be a constantly-running service. You can stop the it by pressing `Ctrl-C`.

- If you're interested in why we reference `./node_modules/bin/` and a way around that, read up on npm scripts[12].

Let's make a test!

Now that we have ChromeDriver running, we can use it to run some tests. In your project folder, create a new file called `tests.js`. Open it up in your favorite code editor.

The first thing we'll do is load WebdriverIO in to the file. To do that, we'll use a require statement:

```
var wdio = require("webdriverio");
```

> **Disclaimer** For the rest of this course we're going to leave out repetitive bits of the code in our examples. To see the full code, check out the code samples for each day[13].

With that loaded, our next step is to set up a browser instance.

[12] http://firstdoit.com/npm-scripts/

[13] http://learn.visualregressiontesting.com/code-samples.zip

```
var browser = wdio.remote({
        desiredCapabilities: {
                browserName: "chrome"
        }
}).init();
```

What just happened here? Well, we told WebdriverIO to start a new Chrome browser. If you want to read more about it, check out the documentation[14].

Get to our page

Now that we have a browser to play around with, let's do some damage. We're going to request a website by using the `url` browser command.

```
browser.url("http://learn.visualregressiontesting.com");
```

Check the title

Now that we have a page loaded, we can do stuff with it. Let's validate it's actually the right place.

How about checking the title of the page and logging it through `console.log`? Here's what the syntax looks like:

```
browser.url("http://learn.visualregressiontesting.com")
        .getTitle().then(function(title) {
            console.log("Title is: " + title);
        });
```

Here we asked our browser to get the title of the page (the aptly named `getTitle` command), and then logged that value out using the generic `then` command.

Note: If you're not familiar with JavaScript Promises, this syntax may look a little strange. For the sake of brevity, we're not going to get into the details of it in this email. Fortunately there are a lot of resources on JavaScript Promises[15] already out there for you to find.

End the test

Finally, we need to tell WebdriverIO to shut things down. That's as simple as adding `.end()` at the bottom of our commands:

[14] http://webdriver.io/guide/getstarted/configuration.html
[15] http://lmgtfy.com/?q=javascript+promise+tutorials

```
browser.url("http://learn.visualregressiontesting.com")
        ... tests are here ...
        .end();
```

Just to recap, here's what your `tests.js` file should look like:

```
var wdio = require("webdriverio");
var browser = wdio.remote({
        desiredCapabilities: {
                browserName: "chrome"
        }
}).init();

browser.url("http://learn.visualregressiontesting.com")
        .getTitle().then(function(title) {
            console.log("Title is: " + title);
        })
        .end();
```

Now it's time to run the test! You can do so by asking Node (via the command line) to ever so kindly execute your code:

```
node tests.js
```

If all went well, you should have seen the following message in your command line:

```
Title is: Learn Visual Regression Testing
```

Tomorrow's outlook

So, how does it feel to have programmatically taken control of a browser? A little awesome, right?

Okay, if you're underwhelmed, that's because we didn't really test much. We only peeked at the page title then closed everything down before we got ourselves in to real trouble.

We promise that over the next couple of days we'll get our hands extra dirty with mouse clicks, keyboard taps, HTML checks and very real visual tests.

If you can't wait until then, take a look at the WebdriverIO API page[16] to see a full list of commands available for use.

[16] http://webdriver.io/api.html