MARTIAL ARTS
HAS BRUCE LEE.

AUTOMATED MOBILE
TESTING HAS APPIUM.

Maybe you can't do a one-fingered push-up, but you can master automated testing for mobile applications with Appium running on Sauce Labs. Optimized for continuous integration workflows, our scalable, reliable, and secure platform enables you to run your builds in parallel, so you can get your native or hybrid iOS and Android apps to market faster.

Download Your Free Appium Bootcamp Guide.

See why these companies trust Sauce Labs.

Dropbox salesforce YAHOO! PayPal intuit VISA

sauce LABS

Mobile Web Application Testing

By Shauvik Roy Choudhary

MOBILE VALIDATION

GENERAL VALIDATION

Mobile web applications are built with the same foundational web technologies as desktop web applications. Therefore, any validation for web applications also applies to the mobile web. The World Wide Web Consortium (W3C) has several validators available at:

http://www.w3.org/QA/Tools/

These validators include: 1) the HTML Validator, to check the markup of webpages, 2) the Link Checker, to find and report broken links, and 3) the CSS Validator, to check the style sheets and their conformance to the W3C standards.

MOBILEOK CHECKER

The W3C has standards and best practices in place for mobile webpages, which are in sync with the mobile browser specifications for rendering webpages. To check whether a mobile webpage conforms to the W3C best practices, use the mobileOK checker tool at: http://validator.w3.org/mobile/.

Figure 1: *The W3C mobileOK Checker*

The tool not only reports failures for the checks performed, but also reports details, linking back to the W3C recommendation. This can be a good starting point to understand the issue and develop a fix for it.

MANUAL TESTING IN MOBILE BROWSERS

SELECTING MOBILE BROWSERS FOR TESTING

It is essential for mobile webpages to be tested on multiple mobile browsers, especially those browsers that the target audience would be using. Selecting which browsers to test your web application in can be a tough decision. One way to help you make this decision is by viewing analytics of your existing sites or browser market share statistics using a tool such as the one provided by:

http://gs.statcounter.com

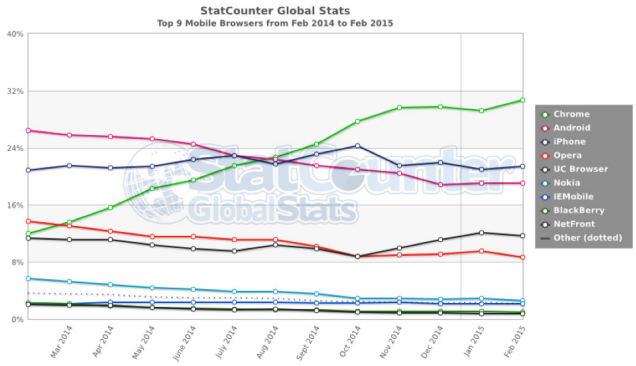

Figure 2: *Top Mobile Browsers from Feb 2014 to Feb 2015*

The statistics can be further narrowed down to show statistics from a particular geographic region or time frame. As of February 2015, the two most popular mobile browsers globally are Chrome on Android and Safari on iPhone. The Android browser was recently replaced by

Chrome in the Android OS but still has a significant market share, ranking as #3 as shown in the chart.

TESTING ON AN EMULATED MOBILE BROWSER

CHROME DEVELOPER TOOLS

The Google Chrome browser provides mobile web development tools out of the box. The Chrome Dev Tools provide support for understanding, debugging and optimization of the web application and browser components.

To open Developer Tools on Chrome, select the hamburger menu ☰ on the right and then select the **More tools > Developer Tools** menu option.

This will bring up the Chrome Dev Tools pane on the side of the browser window. To access the mobile emulation and debugging features, turn on the "device mode" by clicking on the small mobile button in the top left corner of the Dev Tools panel as shown in the figure below:

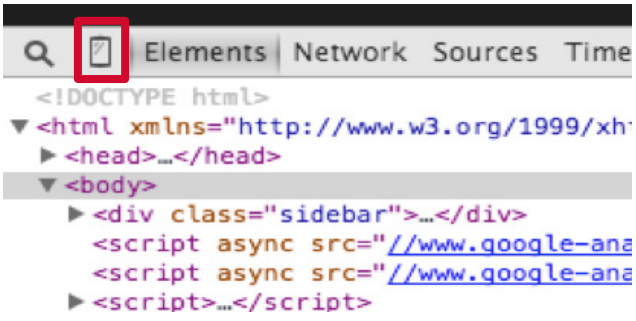

Figure 3: *Chrome Developer Tools Panel*

When "device mode" is enabled, the browser viewport transforms to a device emulator, where the user is allowed to select from a set of devices with specific screen resolutions. Users can switch between different devices and see how the application would appear and behave in those browsers.

TESTING ORIENTATIONS

One of the most important things to test while using this view is to inspect the application in portrait and landscape mode. Clicking on the "Swap Dimensions" button can toggle this setting.

TESTING NETWORK BEHAVIOR

A mobile web app might not perform equally across different network settings. Being on a slow network can cause connection timeouts or timing issues in the application, making it fail or appear differently. Use the Network option to make the browser throttle the web app by simulating the different network conditions, from GPRS to Wi-Fi. You can also test the offline capabilities of your application through this feature.

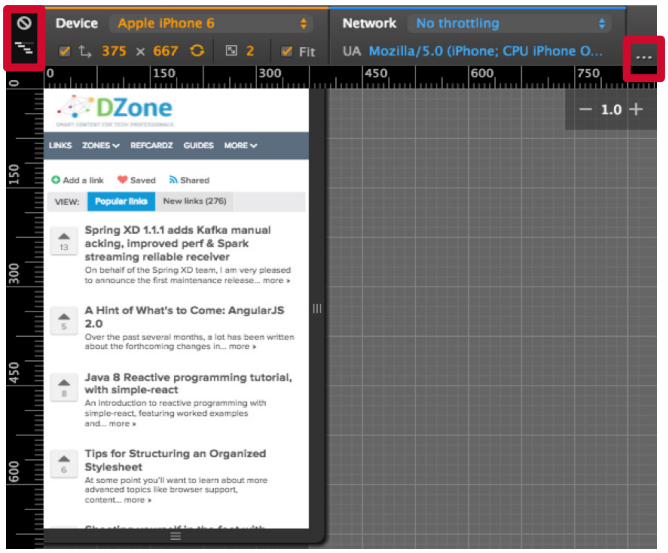

Figure 4: *Chrome Developer Tools Device Simulation*

TESTING MEDIA QUERIES

Media queries were introduced in CSS3 to allow the rendering of the application to be adaptive to different screen widths. Knowing the boundaries at which these CSS styles change in the application can be very helpful for testing how the webpage looks at these boundary values. You can choose devices that are across each boundary or manually slide the handlebar at the end of the mobile viewport to inspect the responsiveness of the app as the resolution changes. Queries targeting a maximum width are represented in blue, those with a range are shown in green, and those targeting a minimum width are represented in orange.

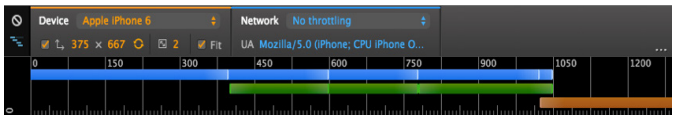

Figure 5: *Chrome Developer Tools Screen-Width Boundaries*

TESTING CUSTOM RESOLUTIONS

The Chrome Dev Tools have the configurations for most popular phones, but not all phones. If you need to test for a custom resolution, you can create a new model by clicking "More overrides" ⋯ on the top right and then selecting **Emulation > Device** in the secondary pane as shown below:

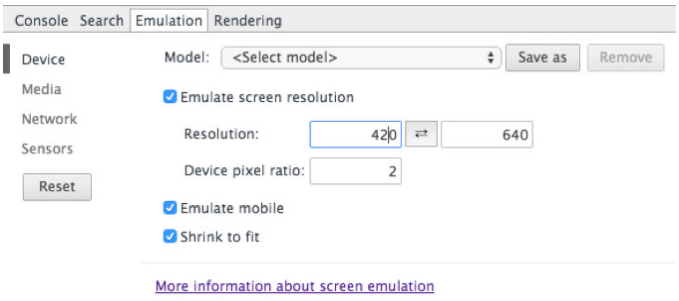

Figure 6: *Chrome Developer Tools Device Emulation Pane*

TESTING TOUCH, GPS, AND ACCELEROMETER

If your webpage is interactive, and allows users to use touch gestures, you can select the "Emulate touch screen" option under the Sensors pane of the Emulation settings to enable this feature. You can perform the touch gestures by using a multi-touch track pad or your computer touch screen.

For testing map components or other features that rely on the phone's location, you can emulate the geolocation coordinates by providing the latitude and longitude of the target location. The test coordinates can be found from a mapping application, like Google Maps. These are entered as numbers by using a negative sign for coordinates with degrees South or degrees West.

To test applications that make use of the phone's accelerometer (such as games), you can enable the "Accelerometer" setting under the Sensors pane. This binds a 3D model of the phone, which is displayed in the emulated accelerometer settings to the right. Turning this model controls the virtual orientation and can be used effectively to test such applications.

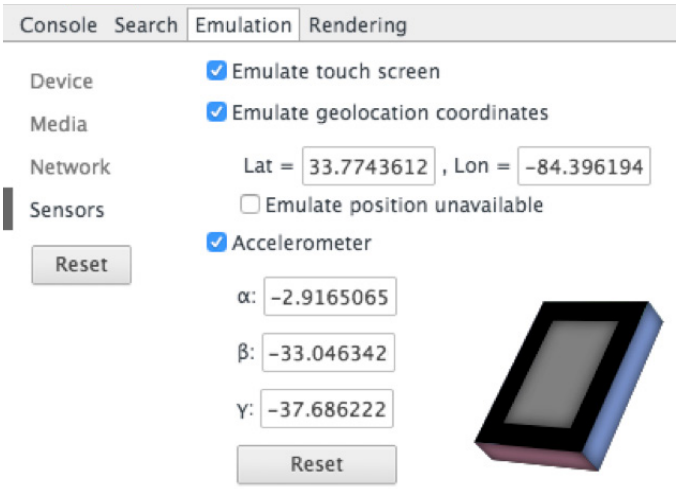

Figure 7: *Chrome Developer Tools Accelerometer Simulator*

TESTING ON A REAL MOBILE BROWSER WITH REMOTE DEBUGGING

If you have a real device to test your webpage, then it might be challenging to control and inspect the page that is loaded on the mobile browser. To help in this scenario, Chrome Dev Tools provides Remote Debugging capabilities that connect the Desktop Chrome browser to any mobile browser that implements the remote debugging protocol. This setting can be loaded by selecting the hamburger menu ☰, then **More tools > Inspect Devices**; or by typing the URL chrome://inspect and then navigating to the Devices pane.

You also need to connect your phone to the computer and select the USB debugging option. On an Android phone, this is accessed by going to **Settings > Developer Options** and selecting the "USB debugging" option. If you can't find the developer options setting, you will need to enable it on your phone.

Once USB debugging is enabled, and the phone is connected to the computer, Chrome will list your device, as well as any pages loaded in the mobile browser of that device. In the Devices pane, you can remotely open a new tab with a URL, or you can interact with already loaded pages.

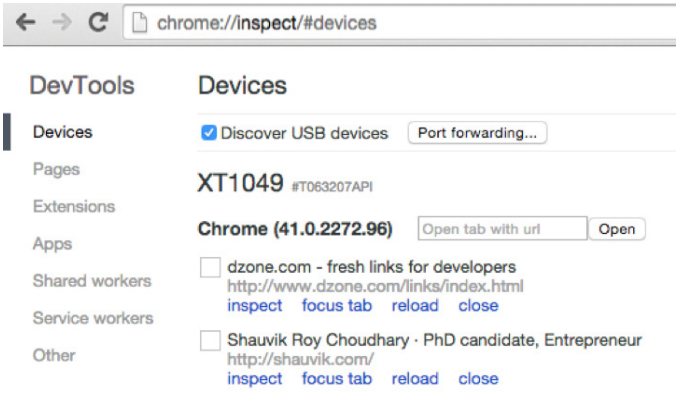

Figure 8: *Chrome Developer Tools Real Device Inspector*

INSPECTING WEBPAGES LOADED ON THE MOBILE DEVICE

You can click on the "Inspect" option under a webpage listed on the Inspect devices page to load the page into the Chrome Dev Tools. As you interact with the pages in the Dev Tools, you can select to see a screencast of the webpage on the computer by selecting the Screencast option on the top right.

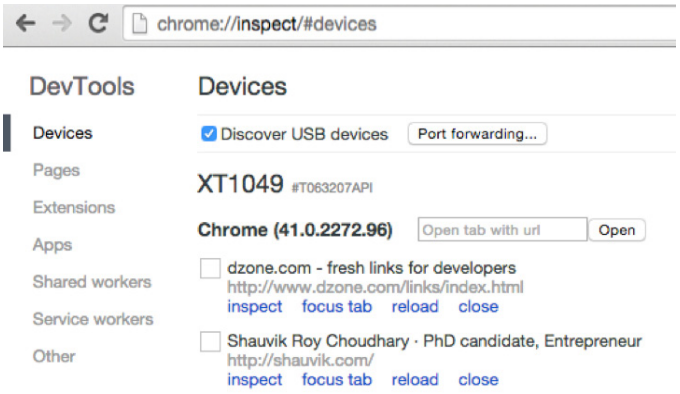

Figure 9: *Chrome Developer Tools Inspection Pane*

TESTING LOCAL WEBSITES

To test webpages hosted locally on your computer and not accessible by a public IP or URL, you can select the port forwarding option to map the computer's web server port to the mobile device. After this setting has been enabled, you can browse the webpage on your mobile device just as if it was hosted on the phone, using the URL http://localhost:[port]

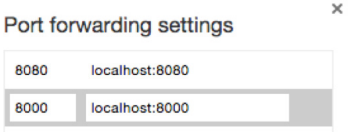

Figure 10: *Chrome Port Forwarding Settings*

AUTOMATED TESTING IN MOBILE BROWSERS

SELENIUM PRIMER (IN PYTHON)

Selenium (also known as WebDriver) is a framework that allows writing automated scripts for web applications. The Selenium project provides libraries for multiple languages, including Java, Ruby, and Python. In this tutorial, we will be using Selenium's Python bindings specifically to test mobile web apps.

INSTALLING SELENIUM

The Selenium library for Python can be installed using the pip command:

```
$ pip install selenium
```

If you don't have pip available on your system, install it by following the directions at https://pip.pypa.io/en/latest/installing.html.

The Selenium library provides functions for performing various actions for testing. These range from actions to select a browser and load the web app in the browser, to actions for interacting with the webpage. We will cover these in the rest of this section.

LOADING THE WEB APPLICATION INTO A BROWSER SESSION

To start a browser session and load a page into the session, you can initialize a particular WebDriver class and use it to load the web application using the driver.get() function:

```
driver = selenium.webdriver.Chrome()
driver.get("http://www.google.com")
```

FINDING ELEMENTS

In order to perform actions or assertions on elements on a webpage, you need to obtain a reference to such elements. You can locate webpage elements by using their different HTML properties. The corresponding WebDriver functions are listed in the table below.

find_element_by_id	Unique id
find_element_by_name	Name attribute
find_element_by_xpath	Xpath locator
find_element_by_link_text	Text inside the link
find_element_by_partial_link_text	Partial link text
find_element_by_tag_name	HTML tag name
find_element_by_class_name	HTML class attribute
find_element_by_css_selector	CSS selector

Example:

```
<input type="text" name="txtName" id="txtId" />
```

To find this text input element, one could use any of the following locators:

```
element = driver.find_element_by_id("txtId")
element = driver.find_element_by_name("txtName")
element = driver.find_element_by_xpath("//input[@id=
'txtId']")
```

By default, the above functions return the first element that is found. If you would like to work with later elements identified by your locator, you would need to use the `find_elements_by_<locator>` version of the function (which finds multiple elements) and access the later elements in the array returned.

PERFORMING ACTIONS

Once you have a reference to an element, you can interact with it through the following functions:

element.click()	Perform a click operation
element.send_keys("<text>")	Type "text" into the element
element.submit()	Submit the form element associated with element

There are also several other advanced functionalities, such as operations on HTML select elements or alert dialogs, and on performing drag-drop operations. Information on these can be found in the documentation for the Selenium-Python package.

ASSERTIONS

For performing assertions, you can either use the assert directive in Python or write your tests using the unittest package.

INSTALLING CHROMEDRIVER

ChromeDriver provides control of the Google Chrome browser though the WebDriver interface. You can download the latest version of the ChromeDriver from:

http://chromedriver.storage.googleapis.com/index.html

Place the ChromeDriver binary in suitable path. For this tutorial, I will place ChromeDriver inside /usr/local/bin.

TESTING ON AN EMULATED MOBILE BROWSER

An easy way to run automated mobile web tests is to emulate a mobile browser on a desktop computer, thereby avoiding the need to set up a real device. ChromeDriver provides an experimental option to emulate a supported mobile browser.

To switch which mobile browser you are emulating, you can change the deviceName to the name of another device that you would like Chrome to emulate for the test.

SAMPLE TEST SCRIPT:

```
import os, time
from selenium import webdriver

# Set up chromedriver path
chromedriver = "/usr/local/bin/chromedriver"
os.environ["webdriver.chrome.driver"] = chromedriver

# Configure mobile emulation in Desktop Chrome
device = { "deviceName": "Google Nexus 5" }
options = webdriver.ChromeOptions()
options.add_experimental_option("mobileEmulation", device)
driver = webdriver.Chrome(executable_path=chromedriver,
chrome_options=options)

# Test steps
driver.get("http://www.google.com");
search_box = driver.find_element_by_name("q")
search_box.send_keys("DZone")
search_box.submit()
time.sleep(5) # Sleep for 5 seconds
assert "www.dzone.com" in driver.page_source

# Quit chromedriver to close the browser window
driver.quit()
```

This test loads up http://www.google.com and searches for "DZone" and checks if its domain www.dzone.com is present in the search results. There is a sleep function added to allow the page to load the search result.

When this test runs, either it will fail and display an error message, or it will not report anything. For managing larger sets of tests, and for better test reporting, you might consider using the Python unittest module.

TESTING WITH THE PYTHON UNITTEST MODULE

By using the unittest module, you will avoid repeating the setup and teardown operations for each test and get a better pass-fail report from running your test case.

```
import unittest, os, time
from selenium import webdriver

# Set up chromedriver path
chromedriver = "/usr/local/bin/chromedriver"
os.environ["webdriver.chrome.driver"] = chromedriver

# Configure mobile emulation in Desktop Chrome
device = { "deviceName": "Apple iPhone 5" }
options = webdriver.ChromeOptions()
options.add_experimental_option("mobileEmulation", device)

class GoogleSearch(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome(executable_
path=chromedriver, chrome_options=options)

    def test_title(self):
        driver = self.driver
        driver.get("http://www.google.com")
        self.assertIn("Google", driver.title)

    def test_dzone(self):
        driver = self.driver
        driver.get("http://www.google.com")
        search_box = driver.find_element_by_name("q")
        search_box.send_keys("DZone")
        search_box.submit()
        time.sleep(5) # sleep for 5 seconds
        assert "www.dzone.com" in driver.page_source

    def tearDown(self):
        self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

This test code, written using the unittest module, contains a setUp function to perform the initialization before each test, and a tearDown function to clean up the driver instance after each test. This test code contains two tests: test_title() and test_dzone(), as shown above. Upon running this test, we get the following output:

```
$ python google_test.py
..
----------------------------------------------------------------
Ran 2 tests in 16.636s
 OK
[Finished in 17.08s]
```

MOBILE-WEB TESTING ON A REAL OR VIRTUAL DEVICE

Some situations cannot be tested on an emulated mobile browser, and the test might need the full functionality of a physical mobile device or the capabilities of a virtual device (e.g. the iPhone Simulator or the Android Emulator). In such scenarios, ChromeDriver can be used for testing Android apps, but it is often easier to use the open-source Appium tool, which makes it possible to run tests on Android, as well as on iOS devices by using ChromeDriver and iOS Auto underneath.

INSTALLING AND RUNNING APPIUM

Appium can be downloaded and installed using the node package manager. If you don't have npm available on your command line, install Node.js from its website.

```
$ npm install -g appium
```

Then to run Appium, you would run the following command:

```
$ appium &
```

By default, Appium shows all messages in its log, which is nice for debugging purposes. You can configure the Appium tool using its command-line options.

For testing your web application in either a real or virtual Android device, you also need to download and install the Android SDK from the Android developer website. Similarly, for testing on the iPhone Simulator or a real iOS device, install the Apple iOS developer tool chain. To easily check if you have the required dependencies, you can run the appium-doctor tool:

```
$ appium-doctor
```

TEST MODIFICATION FOR RUNNING APP USING APPIUM

For using Appium for mobile web testing on a real or virtual mobile device, we will use a Remote WebDriver instead of

the ChromeDriver and initialize it with a set of capabilities to describe the environment on which to test.

```
capabilities = {
  "platformName": "Android",
  "platformVersion": "4.4",
  "deviceName": "Android Phone",
  "browserName": "Chrome"
}
driver = webdriver.Remote("http://localhost:4723/wd/hub",
capabilities)
```

This code essentially replaces the statements inside the **setUp** function in the previous section to run the same test on a real Android device. Appium chooses a connected real device or a running virtual device on the computer. In the case of iOS, you can change the configuration to the following to target the Safari browser in the iPhone Simulator.

```
capabilities = {
  "platformName": "iOS",
  "platformVersion": "7.1",
  "browserName": "Safari",
  "deviceName": "iPhone Simulator"
}
```

Running mobile web tests on a real iPhone is not as straight forward as other cases shown in this document. It requires setting up ios-webkit-debug-proxy, which is used by Appium to communicate with Mobile Safari. More information on this can be found in the Appium documentation.

REFERENCES:

ChromeDriver documentation:
https://sites.google.com/a/chromium.org/chromedriver/

Appium Documentation.
http://appium.io/slate/en/master

Selenium-Python API Docs:
http://selenium-python.readthedocs.org/

Android Developer Website:
http://developer.android.com/sdk/index.html

ABOUT THE AUTHOR

Dr. Shauvik Roy Choudhary received his PhD in Computer Science from Georgia Tech, where he worked on techniques for cross-platform testing and maintenance for web and mobile applications. Currently, he is building a startup to bring new mobile testing and automation tools to developers, while also teaching at Georgia Tech. Prior to his PhD, Shauvik received his MS in Information Security from Georgia Tech and his BE in Computer Engineering from Mumbai University. Over the years, he has held several development, testing and research positions at companies including, Google, Fujitsu Labs of America, Yahoo! Inc., IBM Research, Goldman Sachs and HSBC Software. For more information about Shauvik, visit his website at http://shauvik.com.

RECOMMENDED BOOK

Learning Selenium Testing Tools with Python further discusses ways you can utilize Selenium to test your web applications. Like the examples in this Refcard, it uses Python to demonstrate automated testing, and gives a more comprehensive look at Python's unites module. A great resource to continue learning about Python Selenium testing, this book will help with all kinds of web application testing use cases.

BUY NOW

CREDITS:

Editor: **G. Ryan Spain** | Designer: **Farhin Dorothi** | Production: **Chris Smith** | Sponsor Relations: **Brandon Rosser** | Marketing: **Chelsea Bosworth**

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

VERSION 1.0 $7.95