Testing mobile apps in the cloud (Sauce Labs) using Webdriver and Mocha in Javascript

Overview

In this tutorial, we will create a test in Javascript+Mocha in order to validate a simple application interaction using Sauce Labs for cloud mobile testing.

Please note

Within this tutorial, only one Test Execution will be used; it will contain one Test Run with all the results for the different used browsers. Thus, the overall test run status will be affected by the results made for all the browsers.

Instead of this approach, a different one could be creating a Test Execution per each browser; this would require some adaptions in order to produce a XML report per each used browser. This approach would give the ability to take advantage of Test Environments (more info in Working with Test Environments).

Requirements

- Install NodeJS
- Install all dependencies using "npm"

Description

This tutorial is based on Sauce Labs's own tutorial for NodeJS.

You may start by cloning the repository https://github.com/saucelabs-sample-test-frameworks/JS-Mocha-WebdriverIO-Appium-Android

```
git clone https://github.com/saucelabs-sample-test-frameworks/JS-Mocha-WebdriverIO-Appium-Android
```

We need to cadd the dependency on `wdio-junit-reporter` to generate JUnit XML reports (you have to use it for JUnit, since you cannot rely on mocha-junit-reporter since it does not handle the events from `webdriverio` library).

package.json

```
{
  "name": "mocha-webdriverio",
  "version": "0.0.1",
  "description": "Parallel tests with Mocha and Webdriverio  =============",
  "main": "",
  "repository": {
    "type": "git",
    "url": "git@github.com:saucelabs-sample-test-frameworks/JS-Mocha-WebdriverIO.git"
  },
  "scripts": {
    "test": "./node_modules/.bin/wdio wdio.conf.js"
  },
  "dependencies": {
    "webdriverio": "*",
    "parallel-mocha": "*",
    "mocha": "*",
    "chai": "*",
    "wdio-mocha-framework": "^0.3.10",
    "wdio-sauce-service": "~0.1"
  },
  "devDependencies": {
    "bitcoder/wdio-junit-reporter": "*",
}
```

Please note

The standard "wdio-junit-reporter" npm package can produce one or more JUnit XML reports. We'll generate the multi report file generation capability.

However, the produced JUnit XML contains testcase elements with a non-suitable classname attribute. Since "wdio-junit-reporter" is unable to customize this attribute, we need to make a change so it generates a proper attribute value. This is the reason we're using a forked version from the upstream project.

We need to configure webdriverio with the capabilities (devices/browsers) we want, along with the reporter/JUnit related configurations.

wdio.conf.js

```
exports.config = {

    //
    // ==================
    // Service Providers
    // ==================
    // WebdriverIO supports Sauce Labs, Browserstack, and Testing Bot (other cloud providers
    // should work too though). These services define specific user and key (or access key)
    // values you need to put in here in order to connect to these services.
    //
    user: process.env.SAUCE_USERNAME,
    key: process.env.SAUCE_ACCESS_KEY,


    //
    // ==================
    // Specify Test Files
    // ==================
    // Define which test specs should run. The pattern is relative to the directory
    // from which `wdio` was called. Notice that, if you are calling `wdio` from an
    // NPM script (see https://docs.npmjs.com/cli/run-script) then the current working
    // directory is where your package.json resides, so `wdio` will be called from there.
    //
    specs: [
        './tests/*.js'
    ],
    // Patterns to exclude.
    exclude: [
        // 'path/to/excluded/files'
    ],
    //
    // ============
    // Capabilities
    // ============
    // Define your capabilities here. WebdriverIO can run multiple capabilities at the same
    // time. Depending on the number of capabilities, WebdriverIO launches several test
    // sessions. Within your capabilities you can overwrite the spec and exclude options in
    // order to group specific specs to a specific capability.
    //
    // First, you can define how many instances should be started at the same time. Let's
    // say you have 3 different capabilities (Chrome, Firefox, and Safari) and you have
    // set maxInstances to 1; wdio will spawn 3 processes. Therefore, if you have 10 spec
    // files and you set maxInstances to 10, all spec files will get tested at the same time
    // and 30 processes will get spawned. The property handles how many capabilities
    // from the same test should run tests.
    //
    maxInstances: 40,
    //
    // If you have trouble getting all important capabilities together, check out the
    // Sauce Labs platform configurator - a great tool to configure your capabilities:
    // https://docs.saucelabs.com/reference/platforms-configurator
```

```
    //
    capabilities: [
        // maxInstances can get overwritten per capability. So if you have an in-house Selenium
        // grid with only 5 firefox instance available you can make sure that not more than
        // 5 instance gets started at a time.
        //maxInstances: 5,
        //
        {
            browserName: '',
            appiumVersion: '1.4.16',
            deviceName: 'Samsung Galaxy S4 Emulator',
            deviceOrientation: 'portrait',
            platformVersion: '4.4',
            platformName: 'Android',
            app: 'https://github.com/appium/sample-code/blob/master/sample-code/apps/ApiDemos/bin/ApiDemos-
debug.apk?raw=true',
            waitforTimeout: 300,
            commandTimeout: 300
        }, {
            browserName: '',
            appiumVersion: '1.4.16',
            deviceName: 'Android Emulator',
            deviceOrientation: 'portrait',
            platformVersion: '5.1',
            platformName: 'Android',
            app: 'https://github.com/appium/sample-code/blob/master/sample-code/apps/ApiDemos/bin/ApiDemos-
debug.apk?raw=true',
            waitforTimeout: 300,
            commandTimeout: 300
        }
    ],
    //
    // ====================
    // Test Configurations
    // ====================
    // Define all options that are relevant for the WebdriverIO instance here
    //
    // By default WebdriverIO commands are executed in a synchronous way using
    // the wdio-sync package. If you still want to run your tests in an async way
    // e.g. using promises you can set the sync option to false.
    sync: true,
    //
    // Level of logging verbosity: silent | verbose | command | data | result | error
    logLevel: 'error',
    //
    // Enables colors for log output.
    coloredLogs: true,
    //
    // Saves a screenshot to a given path if a command fails.
    screenshotPath: './errorShots/',
    //
    // Set a base URL in order to shorten url command calls. If your url parameter starts
    // with "/", then the base url gets prepended.
    baseUrl: 'http://saucelabs.github.io',
    //
    // Default timeout for all waitFor* commands.
    waitforTimeout: 10000,
    //
    // Default timeout in milliseconds for request
    // if Selenium Grid doesn't send response
    connectionRetryTimeout: 90000,
    //
    // Default request retries count
    connectionRetryCount: 3,
    //
    // Initialize the browser instance with a WebdriverIO plugin. The object should have the
    // plugin name as key and the desired plugin options as properties. Make sure you have
    // the plugin installed before running any tests. The following plugins are currently
    // available:
    // WebdriverCSS: https://github.com/webdriverio/webdrivercss
    // WebdriverRTC: https://github.com/webdriverio/webdriverrtc
```

```
// Browserevent: https://github.com/webdriverio/browserevent
// plugins: {
//     webdrivercss: {
//         screenshotRoot: 'my-shots',
//         failedComparisonsRoot: 'diffs',
//         misMatchTolerance: 0.05,
//         screenWidth: [320,480,640,1024]
//     },
//     webdriverrtc: {},
//     browserevent: {}
// },
//
// Test runner services
// Services take over a specific job you don't want to take care of. They enhance
// your test setup with almost no effort. Unlike plugins, they don't add new
// commands. Instead, they hook themselves up into the test process.
services: ['sauce'],
// Framework you want to run your specs with.
// The following are supported: Mocha, Jasmine, and Cucumber
// see also: http://webdriver.io/guide/testrunner/frameworks.html
//
// Make sure you have the wdio adapter package for the specific framework installed
// before running any tests.
framework: 'mocha',
//
// Test reporter for stdout.
// The only one supported by default is 'dot'
// see also: http://webdriver.io/guide/testrunner/reporters.html
// reporters: ['dot'],
reporters: ['dot','junit'],
reporterOptions: {
    outputDir: './',
        junit: {
        outputDir: './',
                outputFileFormat: {
                  multi: function (opts) {
                            return `${opts.capabilities}.xml`
                      }
                }
    }
},

// Options to be passed to Mocha.
// See the full list at http://mochajs.org/
//

mochaOpts: {
    ui: 'bdd'
},
// =====
// Hooks
// =====
// WebdriverIO provides several hooks you can use to interfere with the test process in order to enhance
// it and to build services around it. You can either apply a single function or an array of
// methods to it. If one of them returns with a promise, WebdriverIO will wait until that promise got
// resolved to continue.
//
// Gets executed once before all workers get launched.
// onPrepare: function (config, capabilities) {
// },
//
// Gets executed before test execution begins. At this point you can access all global
// variables, such as `browser`. It is the perfect place to define custom commands.
// before: function (capabilities, specs) {
// },
//
// Hook that gets executed before the suite starts
// beforeSuite: function (suite) {
// },
//
// Hook that gets executed _before_ a hook within the suite starts (e.g. runs before calling
```

```
    // beforeEach in Mocha)
    // beforeHook: function () {
    // },
    //
    // Hook that gets executed _after_ a hook within the suite starts (e.g. runs after calling
    // afterEach in Mocha)
    // afterHook: function () {
    // },
    //
    // Function to be executed before a test (in Mocha/Jasmine) or a step (in Cucumber) starts.
    // beforeTest: function (test) {
    // },
    //
    // Runs before a WebdriverIO command gets executed.
    // beforeCommand: function (commandName, args) {
    // },
    //
    // Runs after a WebdriverIO command gets executed
    // afterCommand: function (commandName, args, result, error) {
    // },
    //
    // Function to be executed after a test (in Mocha/Jasmine) or a step (in Cucumber) starts.
    //afterTest: function (test) {
    //},
    //
    // Hook that gets executed after the suite has ended
    // afterSuite: function (suite) {
    // },
    //
    // Gets executed after all tests are done. You still have access to all global variables from
    // the test.
    // after: function (result, capabilities, specs) {
    // },
    //
    // Gets executed after all workers got shut down and the process is about to exit. It is not
    // possible to defer the end of the process using a promise.
    // onComplete: function(exitCode) {
    // }
}
```

The test use the Page Objects pattern, implemented in several classes such as the following one.

pages/home.page.js

```
/**
 * Created by titusfortner on 11/23/16.
 */

var Page = require('./page');

var HomePage = Object.create(Page, {

    graphicsTab: {
        get: function () {
            return browser.element(`~Graphics`);
        }
    },

    click: {
        value: function (tabName) {
            if (tabName === "Graphics") {
                this.graphicsTab.click();
            } else {
                throwError("Not implemented");
            }
        }
    }

});


module.exports = HomePage;
```

Test: Verify the existence of a meny entry

tests/menu-test.js

```
var expect = require('chai').expect;

var HomePage = require('../pages/home.page'),
    MenuPage = require('../pages/menu.page');

describe('Mocha Spec Sync example', function() {
    it("verify Arcs entry in menu", function() {
        HomePage.click("Graphics");
        expect(MenuPage.arcsEntry.isVisible()).to.equal(true);
    });
});
```

Before running the test(s), you need to export some environment variables with your Sauce Lab's username along with the respective access key, which you can obtain from within the User Settings section in your Sauce Lab's profile page.

```
export SAUCE_USERNAME=<your Sauce Labs username>
export SAUCE_ACCESS_KEY=<your Sauce Labs access key>
```

Test(s) then can be run in parallel using NPM "test" task.

```
npm test
```

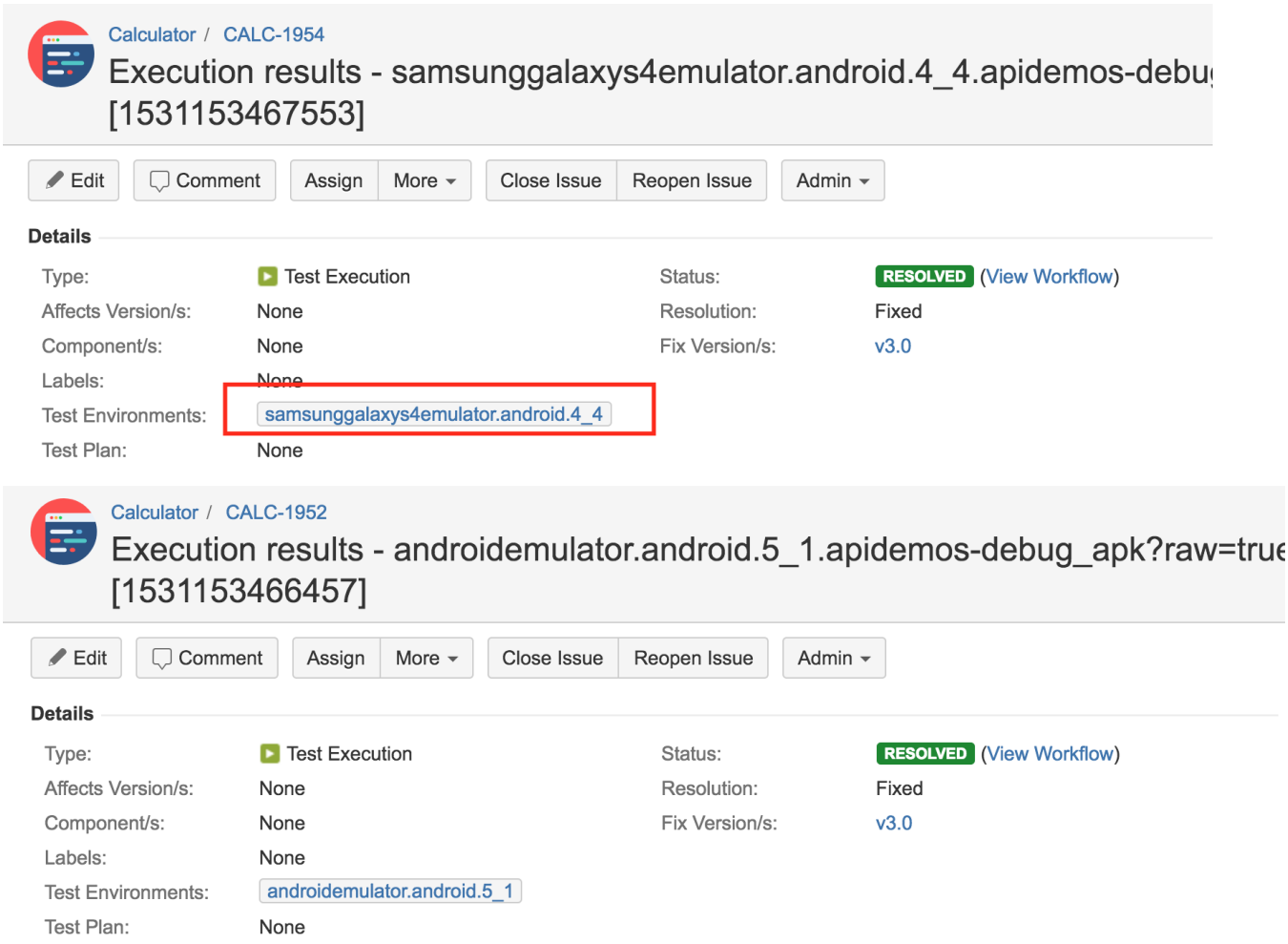
After successfully running the tests and generating the JUnit XML reports (e.g. androidemulator.android.5_1.apidemos-debug_apk?raw=true.xml, samsung galaxys4emulator.android.4_4.apidemos-debug_apk?raw=true.xml), it can be imported to Xray (either by the REST API or through the **Import Execution Results** action within the Test Execution).

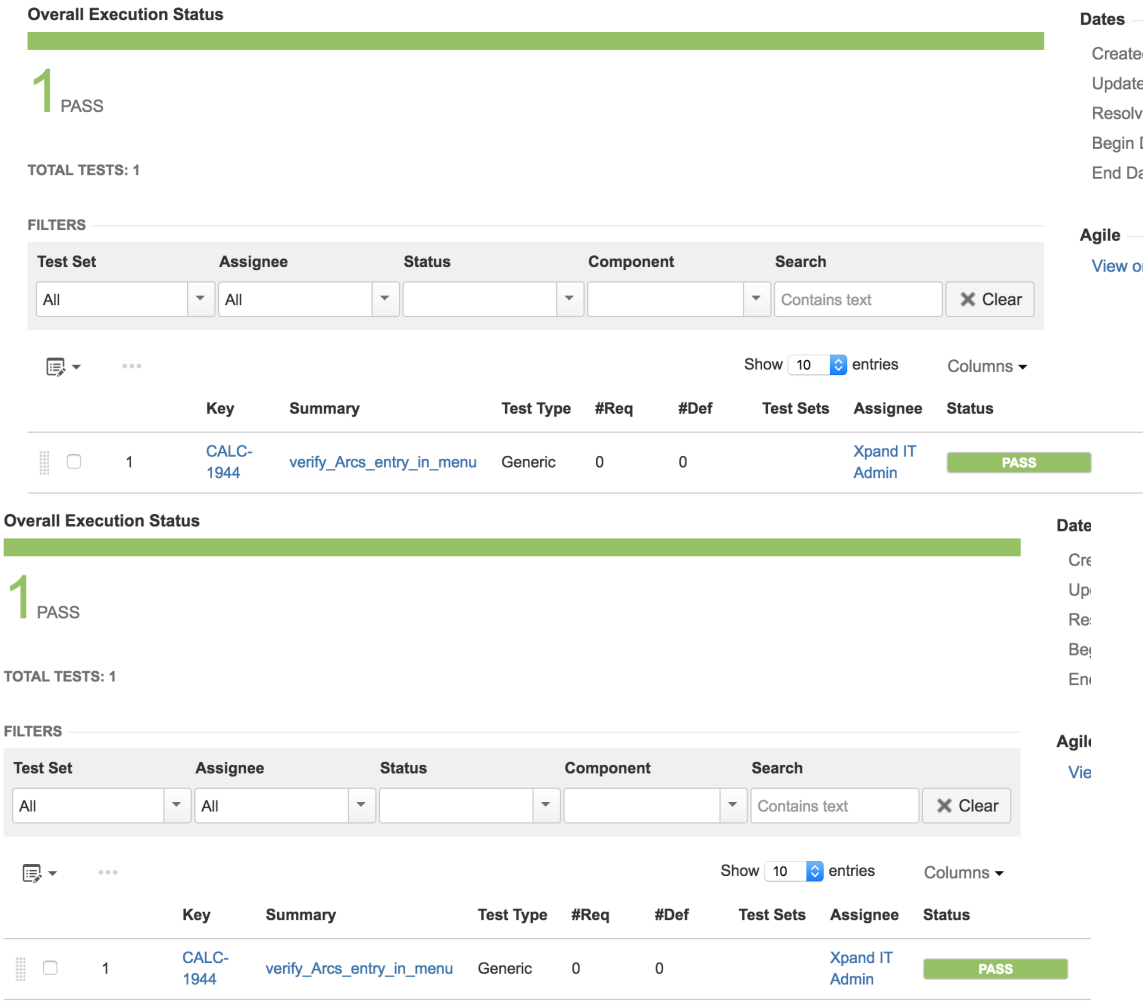
We'll use some shell-script sugar to do that for us and at the same time populate the Test Environment field on the Test Execution issues that will be created.

```
PROJECT=CALC
JIRASERVER=https://yourjiraserver
USERNAME=user
PASSWORD=pass
FIXVERSION=v3.0

for FILE in `ls *.xml`; do
 TESTENV=$(echo $FILE | cut -d "." -f 1-3)
 curl -H "Content-Type: multipart/form-data" -u $USERNAME:$PASSWORD -F "file=@$FILE"  "$JIRASERVER/rest/raven/1.
0/import/execution/junit?projectKey=$PROJECT&testEnvironments=$TESTENV&fixVersion=$FIXVERSION"
done
```

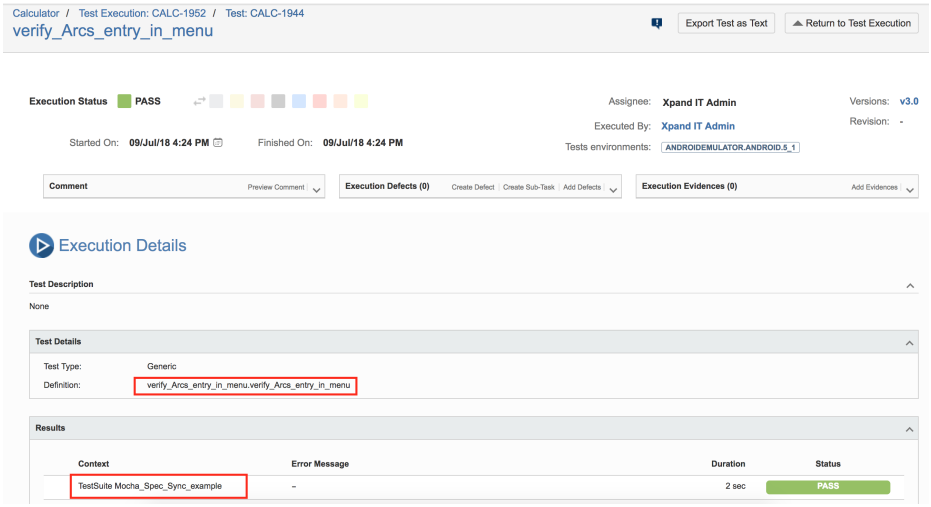
In our case, two Test Executions will be created: one per each mobile device. Each one contains the same Test case.



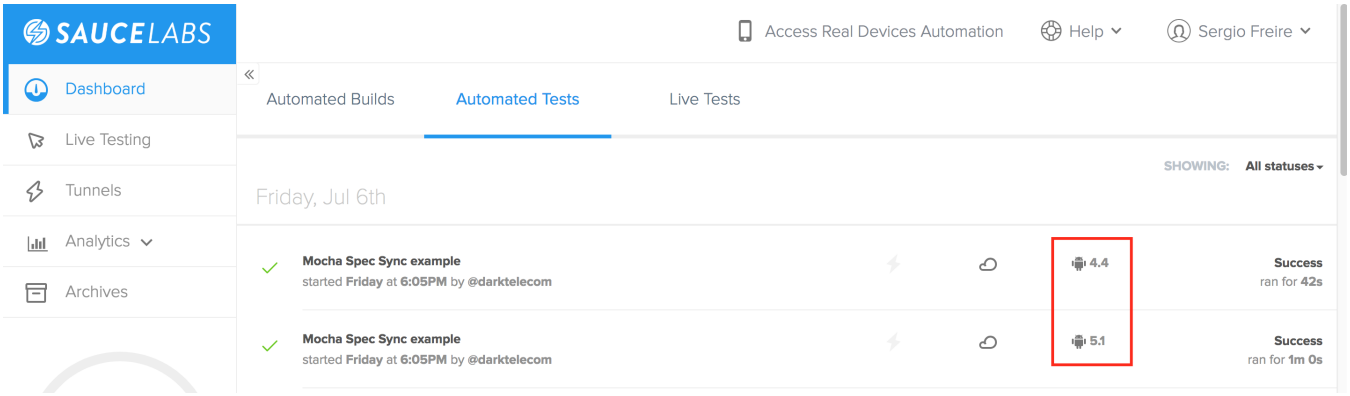


Mocha tests are mapped to Generic Tests in Jira, and the **Generic Test Definition** field contains the namespace, the name of the class, and the method name that implements the Test case.

The execution screen details will provide information on the overall test run result.



In Sauce Labs you can see some info about it.



References

- https://github.com/saucelabs-sample-test-frameworks/JS-Mocha-WebdriverIO-Appium-Android
- https://wiki.saucelabs.com/display/DOCS/Mobile+Application+Testing
- http://webdriver.io/guide/reporters/junit.html