e2e testing hybrid apps

By Michal Mikolajczyk • May 25, 2015 • 16 Comments

By definition, end-to-end testing is meant to assure that the whole integrated system operates correctly. For hybrid app developers, who run JavaScript apps in native-app context, using WebViews, that context should be a part of the tests.

The codebase: https://github.com/michalmikolajczyk/blog-article-ios-testing

My use case was an AngularJS app running inside a complex, native iOS app. I wanted to achieve full automation.

My goals were:

- allow to run the e2e tests in iOS, which includes navigation in the native context,
- allow to run the e2e tests in the browser too,
- allow to run single test suites,
- allow to run all e2e tests,
- use it with a single command.

An overview

To achieve the above, I used the WD driver for node.js, Mocha, Appium, and gulp. At first, I first wanted to go with Protractor, but it does not have support for switching contexts from native app to web view, and I ran into issues with the setup. From a perspective, it does seem to be possible to use Protractor, by utilising the library wd-bridge and Protractor's `onPrepare()` function, as described here https://github.com/angular/protractor/blob/master/docs/mobile-setup.md#setting-up-protractor-with-appium---iossafari. However, my setup has an advantage — due to not using Protractor, it is not limited to AngularJS, nor any other framework.

The initial test file

The simplest test file will include both the environment setup, and the specs.
The environment setup will be declared in a `before()` block. That means, that it will be run before all `it()` blocks, which are in the contained within the same scope. It is important to note that code outside of `it()` blocks will NOT wait for `before()` to finish.

```
'use strict';

var wd = require('wd');
var chai = require('chai');
var chaiAsPromised = require('chai-as-promised');

chai.use(chaiAsPromised);
chai.should();
chaiAsPromised.transferPromiseness = wd.transferPromiseness;

var appEnv;
var browser;
var desired;
var port;


// USING THE _IOSTEST VARIABLE TO SWITCH BETWEEN IOS AND DESKTOP
if (process && process.env && process.env._JUNE20IOSTEST && process.env._JUNE20IOSTEST !== '0')
```

```
 {
  // ENVIRONMENT CONFIGURATION VARIABLES FOR IOS TESTS
  appEnv = true;
  desired = {
    // "appium-version": "1.0",
    platformName: 'iOS',
    platformVersion: '8.3',
    deviceName: '=iPad Air',
    app: 'http://localhost:8080/path/to/your-app.zip',
    browserName: 'App Name'
  };
  port = 4723;
} else {
  // ENVIRONMENT CONFIGURATION VARIABLES FOR DESKTOP TESTS
  appEnv = false;
  desired = {
    browserName: 'chrome',
    version: '',
    platform: 'ANY'
  };
  port = 4444;
}


describe('Initiate tests', function () {

  // the timeout below is by official example.
  // setting up the environment for the iOS tests takes time.
  this.timeout(514229);
  browser = wd.promiseChainRemote('0.0.0.0', port);

  describe('Prepare environment', function () {

    after(function() {
      browser
        .quit();
    });

     if (!appEnv) {
        // SET UP THE ENVIRONMENT FOR DESKTOP TESTS
        before(function () {
          return browser
            .init(desired)
            .get('http://localhost:8080/app/');
        });
      } else {
      // SET UP THE ENVIRONMENT FOR IOS TESTS
      before(function () {
        return browser
          .init(desired)
          .contexts(function (err, items) {
            if (err) {
              throw err();
            }
            var webviewIndex = items.indexOf('WEBVIEW_1');
            return browser.context(webviewIndex);
          });
      });
```

```
    }

    // RUN THE ACTUAL FRONTEND MODULE TEST!
    describe('first test', function () {

      it('should print A title', function () {
        return browser
          .elementByCss('.page-title')
          .text().should.eventually.equal('A title');
      });

    });

  });

});
```

The decision which environment to setup — desktop or iOS — we base on an environment variable, in this case `_IOSTEST` . It is a standard approach, and allows the test to be initiated in multiple ways.

Run the test

To run the test, we need to install mocha and selenium. I use webdriver-manager app, which is bundled with protractor. We also need the libraries used directly in the test file: wd (a node.js client for webdriver), chai, and chai-as-promised

```
npm install -g mocha
npm install -g protractor
```

and:

```
npm install
```

if you use my example code, otherwise manually:

```
npm install wd
npm install chai
npm install chai-as-promised
```

Now let's start protractor in a terminal window, and let the process live.

```
webdriver-manager update
webdriver-manager start
```

Now, the actual test (in the browser).

```
mocha ./article-steps/initial-test-file.js
```

Ok, that works! So now let's test it on iOS!

testing on iOS

We need to install Appium first, when that's done, we also need to ensure there are no blockers for Appium, and grant access to the simulators. Blissfully, there are simple commands for both actions

```
npm install -g appium
appium-doctor –ios
```

```
sudo authorise_ios
```

IMPORTANT! Do not install appium with sudo, it will not work correctly.

Now that we have Appium, let's start it in the same way that we did with webdriver - that is in a new terminal, and let's let the process live.

```
appium
```

Ok. This is it! Let's run the test in iOS!
(you need to have your own app)

```
_IOSTEST=1 mocha test-e2e/main/main.js
```

cool.

Let's automate it!

Let's write the `gulpfile.js` first, so we can see what we need, and install the dependencies next.

```
'use strict';

var gulp = require('gulp');
var shell = require('gulp-shell');
var webdriverUpdate = require('gulp-protractor').webdriver_update;

function runMochaOnWebdriver () {
  shell.task('webdriver-manager start &', {quiet: true})();
  // allow webdriver to start, should not take more than 4181ms
  setTimeout(function () {
    return shell.task('_IOSTEST=0 mocha ./**/*.test.js')();
  }, 4181);
}

function runMochaOnAppium () {
  shell.task('appium', {quiet: true})();
  // allow appium to start, should not take more than 4181ms
  setTimeout(function () {
    return shell.task('_IOSTEST=1 mocha ./**/*.test.js')();
  }, 4181);
}


gulp.task('test-e2e', ['webdriver-update'], runMochaOnWebdriver);
gulp.task('test-ios', runMochaOnAppium);
gulp.task('webdriver-update', webdriverUpdate);
```

Now, install the dependencies.

```
npm install -g gulp
npm install
```

The tasks in gulpfile, `gulp test-e2e` and `gulp test-ios`, will start webdriver or appium, respectively, unless there already is a session running. The output of those commands is silent, to not pollute the stdout.
Ok, let's run it!

```
gulp test-ios
```

So now, we have a test which run against iOS, with a single command.

Separating concerns

The test code is not very DRY. The environment setup part should be separated. Still, we want to be able to execute our test suites in the same way.

We need to modify our test files, to separate module tests from environment setup. Keep in mind, that module tests are ran last. The module test I put in `/test-e2e/main/main.test.js` :

```
'use strict';

require('../initialize-environment.js')('api', function (browser) {

  describe('first test', function () {

    it('should print A title', function () {
      return browser
        .elementByCss('.page-title')
        .text().should.eventually.equal('A title');
    });

  });

});
```

We will add another file, which will be used by all test suites. We will place that file one level above all test suites, in `/test-e2e/initiate-environment.js`

```
'use strict';

module.exports = function (moduleName, callback) {

  var wd = require('wd');

  var chai = require('chai');
  var chaiAsPromised = require('chai-as-promised');

  chai.use(chaiAsPromised);
  chai.should();
  chaiAsPromised.transferPromiseness = wd.transferPromiseness;

  var appEnv;
  var browser;
  var desired;
  var port;


  // USING THE _IOSTEST VARIABLE TO SWITCH BETWEEN IOS AND DESKTOP
  if (process && process.env && process.env._JUNE20IOSTEST && process.env._JUNE20IOSTEST !== '0'
) {
    // ENVIRONMENT CONFIGURATION VARIABLES FOR IOS TESTS
    appEnv = true;
    desired = {
      // "appium-version": "1.0",
      platformName: 'iOS',
      platformVersion: '8.3',
      deviceName: '=iPad Air',
```

```
      app: 'http://localhost:8080/path/to/your-app.zip',
      browserName: 'App Name'
    };
    port = 4723;
  } else {
    // ENVIRONMENT CONFIGURATION VARIABLES FOR DESKTOP TESTS
    appEnv = false;
    desired = {
      browserName: 'chrome',
      version: '',
      platform: 'ANY'
    };
    port = 4444;
  }

  describe('Initiate tests', function () {

    // the timeout below is by official example.
    // setting up the environment for the iOS tests takes time.
    this.timeout(514229);
    browser = wd.promiseChainRemote('0.0.0.0', port);

    describe('Prepare environment', function () {

      after(function() {
        browser
          .quit();
      });

      if (!appEnv) {
        // SET UP THE ENVIRONMENT FOR DESKTOP TESTS
        before(function () {
          return browser
            .init(desired)
            .get('http://localhost:8080' + moduleName + '/');
        });
      } else {
        // SET UP THE ENVIRONMENT FOR IOS TESTS
        before(function () {
          return browser
            .init(desired)
            .contexts(function (err, items) {
              if (err) {
                throw err();
              }
              var webviewIndex = items.indexOf('WEBVIEW_1');
              return browser.context(webviewIndex);
            });
        });
      }

      // RUN THE ACTUAL FRONTEND MODULE TEST
      // USING SEPARATE MODULE TESTS!
      callback(browser);

    });

  });
```

```
};
```

Running test suites

But one test is not enough. We will want to have multiple tests. It would also be great to allow to run single tests, as well as all of them in one go. We need a form of test suites. My approach to that with protractor was using `suites`, but here, I rely globs and directory structure. I basically have it like this:

```
– root
    |
    – app
    – assets
    …
    – test-e2e
        |
        – main
            |
            – main.mocha.js
            – main-second.mocha.js
            ...
        |
        |
        – some-other-module
        – yet-another-module
        …
```

Let's make a copy of the test file, perhaps test another route or whatever, and put it into another suite. Now we modify the gulp file to allow for taking command line arguments:

```
'use strict';

var gulp = require('gulp');
var args = require('yargs').argv;
var shell = require('gulp-shell');
var webdriverUpdate = require('gulp-protractor').webdriver_update;

function runMochaOnWebdriver () {
  var suite = args.suite || '**';

  shell.task('webdriver-manager start &', {quiet: true})();
  // allow webdriver to start, should not take more than 4181ms
  setTimeout(function () {
    return shell.task('_IOSTEST=0 mocha test-e2e/' + suite + '/*.test.js')();
  }, 4181);
}

function runMochaOnAppium () {
  var suite = args.suite || '**';

  shell.task('appium', {quiet: true})();
  // allow appium to start, should not take more than 4181ms
  setTimeout(function () {
    return shell.task('_IOSTEST=1 mocha test-e2e/' + suite + '/*.test.js')();
  }, 4181);
}
```

```
gulp.task('test-e2e', ['webdriver-update'], runMochaOnWebdriver);
gulp.task('test-ios', runMochaOnAppium);
gulp.task('webdriver-update', webdriverUpdate);
```

Now we can run single test suites, or all tests, on desktop or on iOS.

Let's run a single suite on iOS:

```
gulp test-ios –suite=main
```

And let's run all the tests in the browser:

```
gulp test-e2e
```

The codebase is available here: https://github.com/michalmikolajczyk/blog-article-ios-testing

Done! :)
