Mocha JS & Chai

More for functional testing (for navigation/interaction of UI, try CasperJS).

*Upload to Github*

Create a repository

cd to directory where you want your tests to be

git clone git@github.com:project.git

git add –A

git commit –m “Message”

git push

*Initialize project*

cd to project folder > npm init > when filling fields, enter test command: mocha

type package.json to view (cat package.json in Unix)

npm install mocha –g (installs mocha globally)

npm install chai –g (also: --save-dev)

npm install blanket --save-dev (for test coverage)

e.g.

“scripts”: {  
 "test": "make test"  
}

*Assertion libraries*

var chai = require('chai').expect; //required files are put into variables

Also, check out should.js (shouldjs.github.io)

node has a built-in assert (e.g. return assert.equal(5, plus(2, 3))

*Running tests*

npm test

mocha nameOfTest

mocha (runs all tests)

*Hooks*

Executing something or after tests (before, beforeEach, after, afterEach)

*Selectively run tests*

it.only( . . .

it.skip(. . .

*Reporters*

e.g.

mocha testName.js --reporter dot – minimizes clutter

doc - outputs html

doc > output.html > open output.html

*Blanket (and screenshotting)*

*Spy*

A spy is a function that replaces a particular function where you want to control its behavior in a testand record how that function is used during the execution of that test.

For example, you can test JavaScript alerts with a spy as follows:

spyyn(window, ‘alert’);

. . .

expect(window.alert).toHaveBeenCalledWith(‘a message’);

*mocha.opts*

e.g.

--reporter nyan

--recursive

--no-colors

--timout 5000 (mocha --timeout 5000 does the same)

*Promises*

Complete an async expectation by passing the error as a parameter in the done() callback. For example:

it(“should pass”, function(done) {

promise.then(function(value){

expect(value).equals(true);

done();

}).done(done, done); // <-- on fail

});

*Sinon.JS*

Stubs

Use a stub when you want to control a method’s behavior from a test to force the code down a specific path (e.g. forcing a method to throw an error to test error handling). Also used when you want to prevent a specific method from being called directly. In the following example, a fake version of a method called isValid will be called instead of the original one:

spyOn(user, ‘isValid’).andReturns(true);

*Sinon Fake Server*

Set up fake responses to AJAX requests made for certain URLs. This can be really handy for a few reasons. First, it allows you to test your code that makes AJAX calls regardless of which AJAX library you are using. Second, you may want to test a function that makes an AJAX call and does some preprocessing on the response before the promise resolves. Third, maybe there are several responses that can be returned based on if the request succeeds or fails such as a successful credit card charge, an invalid credit card number, an expired card, an invalid CVC, etc.

it('should return a collection object containing all users', function(done) {

var server = sinon.fakeServer.create();

server.respondWith("GET", "/users", [ //URL is /users

200, //200 means request has succeeded

{ "Content-Type": "application/json" },

'[{ "id": 1, "name": "Gwen" }, { "id": 2, "name": "John" }]'

]);

Users.all().done(function(collection) {

expect(collection.toJSON()).to.eql([

{ id: 1, name: "Gwen" },

{ id: 2, name: "John" }

]);

done();

});

server.respond();

server.restore();

});

*Selenium WebDriver*

npm i selenium-webdriver chai mocha --save-dev

Start up the selenium server

Download the Chrome driver jar file and move it to the folder where your .exe files are (e.g. /usr/bin/chromedriver)

cd to the folder containing the server.js file and use the command node server.js

*Using a node server to host*

Create a server.js file with the following code:

var http = require('http'),

fs = require('fs');

fs.readFile('./index.html', function(err, html) {

if(err) {

throw err;

}

http.createServer(function(request, response) {

response.writeHeader(200, {

"Content-Type" : "text/html"

});

response.write(html);

response.end();

}).listen(8000);

});

console.log('Server running at localhost:8000');

To close the server, use Ctrl + C.

**Push AVL from device to JM – Troubleshooting**

For a journey created for a 1580 leader using vehicle GEOTRACQABT5800:

- avl messages should flow from device through parser-services to avl-push service and into vehicle.avl NSB endpoint

- these avl messages should be forwarded to journey.avl.filter endpoint

- messages received within the journey's buffered scheduled timespan (buffer: company setting, default 3 minutes) should update journey.avl.monitor endpoint

- messages with lat-long outside a checkpoint's departure zone or inside a checkpoint's arrival zone should arm trigger

- subsequent message back inside checkpoint zone or outside a checkpoint's arrival zone should disarm trigger

- absent any disarming, a trigger should fire once trigger duration buffer expired (company setting, default 3 minutes) or, for ignition off inside arrival zone, should fire immediately

In User Profile, the “My PageSize” option currently doesn’t work

No pagination in Journey page

Trying to change password for a JourneyManager or JourneyUser leads to an error page but the password change works

JM mobile login screen has non-working “hamburger” (menu icon that only appears when screen is small)

No way to log out of JM when I reach the “Insufficient Permissions” screen (need to reset cache & cookies)

Review “Next Tile” journey post-MVP

Vehicle still says [unassigned] in Journey Changed email

Can still postpone a Journey after having already departed

Fix Add\_Asset\_Certificate, Edit\_AssetCertificate

**Phantom JS**

npm install phantomjs

Update phantomjs references to phantomjs-prebuilt

Go to your Environment Variables and the phantom/bin location to your PATH variable

**Casper JS**

C:\Users\JCrisologo\node\_modules\phantomjs\lib\phantom\bin\phantomjs.exe