**Infosys**

**AngularJS Unit Testing Using Mocha Test Framework**

**January, 2015**

**INFOSYS LIMITED**

**Bangalore**

|  |  |  |  |
| --- | --- | --- | --- |
| Document No. |  | Ver. Rev. : | 1.0 |
| Authorized by: |  | Signature/:  Date: |  |

**COPYRIGHT NOTICE**

©2015 Infosys Limited, Bangalore, India. All rights reserved. Infosys believes the information in this document is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. Except as expressly permitted, neither this document nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior permission of Infosys Limited and/or any named intellectual property rights holders under this document.

Infosys Limited

Hosur Road

Electronic City, 3rd Cross

Bangalore 560 100

India.

Telephone: (91) (80)28520 261-270

Fax: (91) (80) 8520 362

Website: <http://www.infosys.com>

**Author: Daya Shankar Pathak**

**Date written: 29th January 2015**

**Project Details**

         Project involved: TMOADMP2

         H/W Platform:

         S/W Environment: Node.js

         Application Type: Web Application

         Project Type: N/A

**Target Readers**

Developers who are dealing with single page application using angularJs framework with

node.js server-side programming environment.

**Assumptions:**

Reader is convenient with the node.js and angularJs framework.

**Keywords**

AngularJS, Controllers, Directive, Mocha, Chai, Sinon, Karma, Grunt, Dependency Injection, Yeoman, Unit Test Cases

**Table of Contents**

[Purpose of Unit Testing 5](#_Toc410144237)

[AngularJs & Unit testing in AngularJs : 5](#_Toc410144238)

[Mocha Test Framework 5](#_Toc410144239)

[Chai - Assertion Library: 6](#_Toc410144240)

[SINON: 6](#_Toc410144241)

[KARMA : The Spectacular Test Runner 7](#_Toc410144242)

[Testing of Controllers 9](#_Toc410144243)

[Testing of Directives: 12](#_Toc410144244)

[Unit Test Code Coverage: 13](#_Toc410144245)

[Creating an AngularJs Application using Yeoman and setup for using mocha test framework: 15](#_Toc410144246)

[Reference(s) 18](#_Toc410144247)

#### Purpose of Unit Testing:

Basically unit testing is about testing the individual units of code. By ‘Units of Code’ we mean the function or the logic which have been written to build the application.

Building an application is a gradual process of writing pieces of code. We often add/modify the existing code in accordance with new requirements/enhancements in the application. So , in that case also these test cases results will tell us whether the new code is impacting the old one or not. And in this way it saves our some debugging efforts also.

**AngularJs & Unit testing in AngularJs** :

AngularJs is a JavaScript framework. It is a single page application framework developed at Google.

It has a great advantage of outstanding ability to be tested.

There are many javascript test libraries like QUNIT, JASMINE and MOCHA.

Mocha test framework has been built specifically for testing NodeJs modules.

Since we are dealing with a Node JS application, so here we will use mocha test framework in this document.

#### Mocha Test Framework :

Mocha is a javascript testing framework. It runs on node.js and browser. It provides a setup for testing (through different hooks) and it describes the structure of test suits and test cases. These (hooks, test suits, test cases) are described below.

It uses a declarative syntax approach to nest expectations into cases and suites.

***‘Hooks’ in Mocha :***

Hooks are functions provided by mocha to define some processes like loading up some module/ preconditions etc. which are required to make the successful execution of test cases. For example:

* before() , after() : These may be put before specific test case as per the requirement of that test case.
* beforeEach() , afterEach() : If all the test cases of file require certain preconditions/setup in common, we can use these(beforeEach() , afterEach()) functions, these will execute before each and every test cases in the file instead of the specific one.

Eg:

|  |
| --- |
| beforeEach(function(){  // runs before each test in this block  module('moduleName');  });  //this is the first test suite which covered the test cases inside it  describe("functionality/description", function() {  //the test case  it(description for testing', function () {  });  }); |

Now we need additional libraries with this test framework to check our test cases. Here assertion and spy libraries come into picture.

Mocha allows us to use different assertion libraries (Eg: should.js, expect.js, chai.js etc).

Here we will discuss the chai assertion library and sinon spy framework.

#### Chai - Assertion Library:

Chai is the assertion library which provide various ways of checking things in the test cases**.**

There are various assertion styles and BDD styles ( expect(),should(),assert() etc. ) which performed the basic checking activities and mark the test cases as failed or passed.

Please refer to http://[chaijs](http://chaijs.com/).com for various assertion styles.

For eg :

|  |
| --- |
| var content = ‘hello’;  //Test case using ‘expect’ BBD style.  it(content variable should be a string , **function** () {  expect(content).to.be.a('string');  }); |

#### SINON:

As there is no assertion framework built in to Mocha , so we used the **chai** assertion library. In the same way there is no spy framework built in to Mocha, so we are using **sinon.js.**

Sinon describes itself as “standalone test spies, stubs and mocks for JavaScript.” Suppose we have to check whether some event is fired or not on invoking of some function (the one inside which the event is supposed to fire). Then in that case we can put spy on the $emit/$broadcast, whichever we have used to fire the event.

Eg :

|  |
| --- |
| it('should listen XYZ function and emit the goToNext event', function () {  var spy = sinon.spy($rootScope, '$emit');  //invoking the function in which the event is supposed to be fired  // Object : is the reference(scope) of controller/directive/service in which XYZ() is registered.  Object.XYZ();  expect($rootScope.$emit.called).to.equal(true);  }); |

Thus we have seen the flexibility of Mocha framework i.e we can choose different framework (i.e for assertion,spy etc) as per our need which offers a complete toolset to produce quality code that is easy to maintain, bug-free and well documented.

So with MochaJS as test framework we are using  [ChaiJS](http://chaijs.com/) as assertion library and [SinonJS](http://sinonjs.org/) as spyes/stubs/mocks utility library.

***Now we need a test runner* :** **KARMA**

#### KARMA : The Spectacular Test Runner

Karma is a test runner for JavaScript that runs on [Node.js](http://nodejs.org/). It is very well suited to testing [AngularJS](http://angularjs.org/) or any other JavaScript projects.

It is configurable through a JSON file and also with the use of various plugins.

***Installation :***

Installing Karma is done via the Node Package Manager (NPM). From any command prompt, enter the command: npm install –g karma.

**Configuration file** : *karma.conf.js*

Before running Karma we must configure it. This is the most important step in setting up Karma.

Here is the sample karma config file :

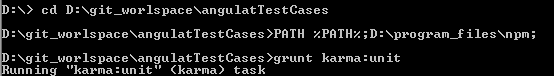
|  |
| --- |
| module.exports = **function** (config) {  config.set({  // base path, that will be used to resolve files and exclude  basePath: '..',    // frameworks to use  frameworks: ['mocha','chai','sinon'],  // list of files / patterns to load in the browser  files: [  'bower\_components/angular/angular.js',  'bower\_components/angular-mocks/angular-mocks.js',  'src/\*.js',  'test/\*.mocha.js'  ],  // list of files to exclude  exclude: ['test/karma.conf.js'],  // preprocess matching files before serving them to the browser  preprocessors: {  'src/\*.js': ['coverage']  },  // test results reporter to use  // report on progress, collect coverage and junit reports  reporters: ['progress', 'coverage'],  // web server port  port: process.env['KARMA\_PORT'] || 8080,  // enable / disable colors in the output (reporters and logs)  colors: process.env['KARMA\_COLORS'] || **true**,  // level of logging  // possible values: LOG\_DISABLE || LOG\_ERROR || LOG\_WARN || LOG\_INFO || LOG\_DEBUG  logLevel: config.LOG\_INFO,  // enable / disable watching file and executing tests whenever any file changes  autoWatch: **true**,  // Start these browsers, currently available:  // - Chrome  // - ChromeCanary  // - Firefox  // - Opera  // - Safari  // - PhantomJS  browsers: ['Chrome'],    // Which plugins to enable  plugins: [  'karma-phantomjs-launcher',  'karma-mocha',  ],  // Continuous Integration mode  // if true, it capture browsers, run tests and exit  singleRun: **false**  });  }; |

**Running Karma :**

***To run karma***

* Go to your project directory
* Provide the path of npm
* Give the command grunt karma:unit (grunt : is the task runner used)

**Eg :**



#### Testing of Controllers :

Controllers deal with most of our application logic which also include the participation of services (like service for persisting the data, for invoking mock services etc.)

Eg:

* ***Controller - ‘testingCtrl’***

|  |
| --- |
| angular.module('testingApp')  .controller('testingCtrl',  function($scope, $location) {  $scope.btnClick = false;  $scope.redirectUrl = function(){  $scope.btnClick = true;  $location.path('/newPath');  };  }); |

In the above demo controller, as we can see there are some services (angular default services) i.e $location and $scope are injected as dependency for the controller.

So, we need to inject these dependencies in the unit testing of this controller as well.

* Unit test for ***‘testingCtrl’***

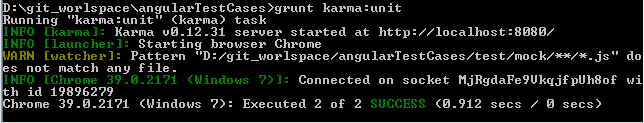
|  |
| --- |
| describe('test cases for controller', function() {    var $rootScope = null,  $controller = null,  testingCtrlScope = null,  $location = null;  beforeEach(function(){  **module**('testingApp');  });    // load scope and controller before run test cases  beforeEach(**inject**(function (\_$rootScope\_, \_$controller\_,\_$location\_) {  $rootScope = \_$rootScope\_;  testingCtrlScope = $rootScope.$new();  $controller = \_$controller\_;  $location = \_$location\_;  $controller('testingCtrl', {'$scope': testingCtrlScope });  }));    describe("testingCtrl", function() {  //checking scope is defined  it('should have scope defined', function () {  angular.isDefined(testingCtrlScope);  });    //to check the scope value and path  it('checking the scope value after click of redirect function', function(){  testingCtrlScope.redirectUrl();  expect($location.url()).to.equal('/newPath');  expect(testingCtrlScope.btnClick).to.equal(true);  });  });    }); |

NOTE : The [module()](https://docs.angularjs.org/api/ngMock/function/angular.mock.module) and [inject()](https://docs.angularjs.org/api/ngMock/function/angular.mock.inject) functions help to retrieve modules and dependencies during tests.

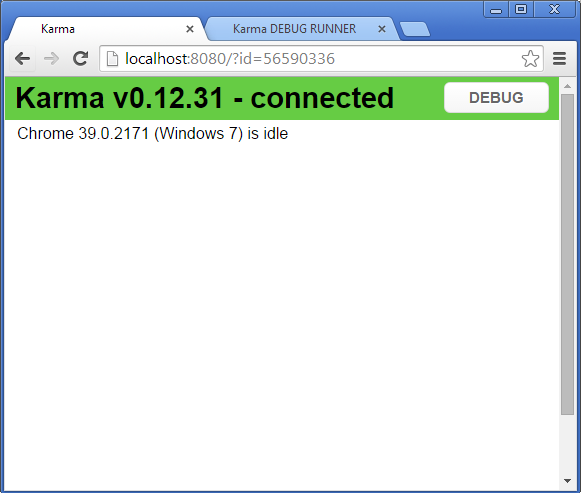
By convention underscore notation is used when adding dependencies in inject(), which is not considered while matching the corresponding service references. i.e for '\_$location\_' , $injector will look for '$location' service, underscores will not be considered.

Executing the test cases for controller : ***‘testingCtrl’***

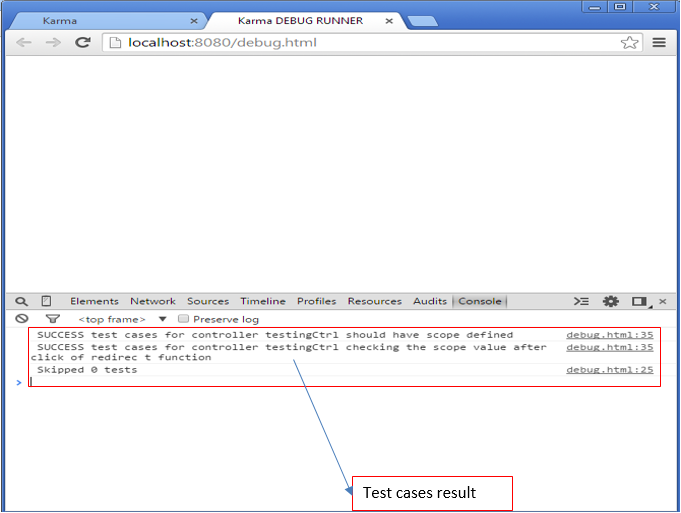
* ***Run the command : grunt karma:unit***

The command prompt would appear something like the below one:

* As we are using chrome browser, after successful execution it would open the new chrome window for debugging purpose on the server provided in the karma.conf.js.



* If you want to debug your test cases you can click on ‘DEBUG’ (shown in above chrome window), which will open a new tab (as shown below) in which you can see your test case results and debug the same.



***Note: This is how we can execute our test cases in browser and can debug.***

#### Testing of Directives:

Testing a directive that uses inline templates (html that is embedded within the directive's code) is very straight forward. Let’s choose the directive which uses external templates in our understanding process, which give us more learning.

Let the directive to be tested be :

* ***Directive : ‘'testDir'’***

|  |
| --- |
| angular.module('testingAppApp').directive('testDir', function() {  return {  restrict: 'A',  templateUrl: /app/views/test.html'  };  }); |

* ***The template test.html:***

|  |
| --- |
| <div >  <button>this is the test button for directive testing</button>  </div> |

* ***Unit testing of directive : “testDir” requires :***

1. As the directive testing also requires the inclusion of html files, so first of all we need to convert those html into angularJs template.
2. For this we have to add a devDependency in package.json which will converting HTML files into AngularJS templates – “ng-html2js”.
3. Run the command : npm install karma-ng-html2js-preprocessor --save-dev
4. And the same is needed to register in ‘pre-processor’ in karma.conf.js.

|  |
| --- |
| preprocessors: {  'app/scripts/\*\*/\*.html': ['ng-html2js']  },  ngHtml2JsPreprocessor: {  moduleName: 'templates'  },  // Which plugins to enable  plugins: [  'karma-ng-html2js-preprocessor',  ] |

1. Now this template is ready to be referenced in the unit testing of directive.

* ***Test cases for directive : “testDir”***

|  |
| --- |
| describe('tesingApp', function() {    var $rootScope = null,  $compile = null,  $scope = null;  beforeEach(function(){  module('testingAppApp');  module('templates');  });  // load scope and controller before run test cases  beforeEach(inject(function (\_$compile\_, \_$rootScope\_) {    $compile = \_$compile\_;  $rootScope = \_$rootScope\_;  $scope = $rootScope.$new();  }));    describe("declineModal", function() {    it('declineModal check content', function () {  var element = $compile("<div test-dir></div>")($scope);  $scope.$digest();  var totalBtn = element.find('button').length;  expect(totalBtn).to.equal(1);  });  });  }); |

As we know testing of directive requires the DOM , and which demands the use of services like $compile in our unit testing for the same.

This service renders the directive i.e compile it to the template and link it to the scope. And thus we can resolve our testing curiosities for the same.

Till now we have learned the basic of angularJs testing and also have gone through the examples of controller and directive. In the same way we can proceed with other testable like service, factory and filter.

For more on anguarJs unit testing code examples, please refer to the documentation on angularJs testing :

<https://docs.angularjs.org/guide/unit-testing>

#### Unit Test Code Coverage:

After writing the unit test cases, the obvious question one might have is to know how much percentage of code is covered in unit testing.

To view the code coverage follow the below steps:

* Add karma-coverage in the devDependencies in package.json :

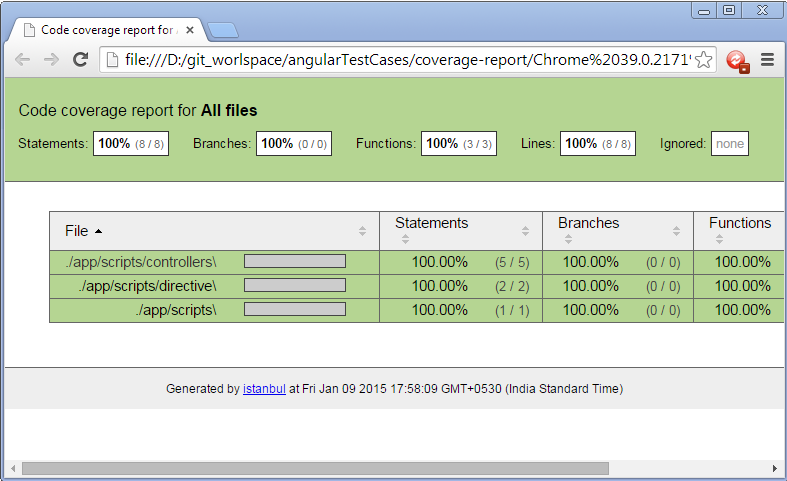
i.e. execute the command: npm install karma-coverage --save-dev

* Do some modification in the karma.conf.js

|  |
| --- |
| //pre-processor for collecting code coverage  preprocessors: {  'app/scripts/\*\*/\*.js': 'coverage',  }  // report on progress, collect coverage reports  reporters: ['progress', 'coverage']  // optionally, configure the reporter  coverageReporter: {  type : 'html',  dir : 'coverage-report/'  }  // Which plugins to enable  plugins: [  'karma-coverage',  ] |

* After the above changes, run the test cases which generate a ‘coverage-report’ folder that contains the index.html for showing the coverage report :

The coverage report will looks like the below one:



#### 

#### Creating an AngularJs Application using Yeoman and setup for using mocha test framework:

Note : The Node should be installed on the system as everything else in the below steps relies on it .

* Install yeoman : npm install -g yo

Yeoman : It is a Web Scaffolding tool for modern web applications .

* Install generator-angular : npm install -g generator-angular
* Create a directory and cd into the same : mkdir project-name && cd project-name

Note: project-name🡪 should be the name of your project for eg : angularTestCases

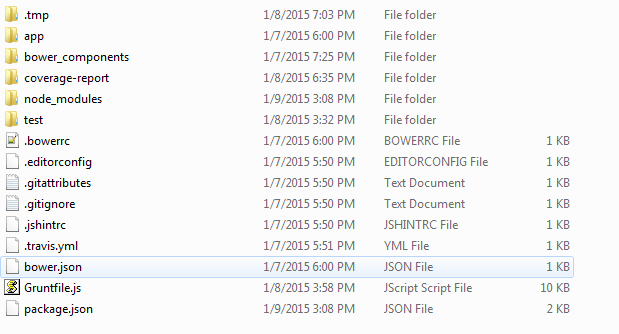
* Create an angular app : yo angular [app-name]

app-name 🡪 Should be the name of your angular application eg :testingApp

After this command you will be asked certain question regarding the inclusion of various angular modules.

* Now application is ready and the app structure would be looking like the below one :

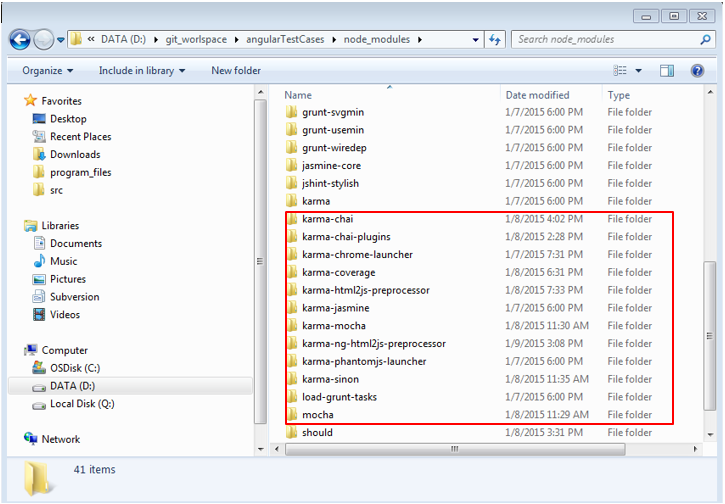
(Note: karma.conf.js file and the test cases files reside in the ‘test’ folder of the project directory)



* Now to get ready with mocha-chai-sinon testing framework, add the following devDependencies in the package.json :
* Install mocha : npm install mocha --save-dev
* To use mocha testing framework with karma : npm install karma-mocha --save-dev
* Installing chai assertion library : npm install karma-chai --save-dev
* Installing sinon spy library : npm install karma-sinon --save-dev
* To use ‘chrome’ as test launcher : npm install karma-chrome-launcher --save-dev
* To get the test coverage report : npm install karma-coverage --save-dev
* Also add these plugins in the ‘plugins’ array of karma.conf.js
* After running all the above commands the devDependencies will be added in the package.Json, and the package.Json file would be looking like (dependencies added by the above command are highlighted):

|  |
| --- |
| {  "name": "testingapp",  "version": "0.0.0",  "dependencies": {},  "devDependencies": {  "chai": "^1.10.0",  "grunt": "^0.4.5",  "grunt-autoprefixer": "^0.7.3",  "grunt-concurrent": "^0.5.0",  "grunt-contrib-clean": "^0.5.0",  "grunt-contrib-concat": "^0.4.0",  "grunt-contrib-connect": "^0.7.1",  "grunt-contrib-copy": "^0.5.0",  "grunt-contrib-cssmin": "^0.9.0",  "grunt-contrib-htmlmin": "^0.3.0",  "grunt-contrib-imagemin": "^0.8.1",  "grunt-contrib-jshint": "^0.10.0",  "grunt-contrib-uglify": "^0.4.0",  "grunt-contrib-watch": "^0.6.1",  "grunt-filerev": "^0.2.1",  "grunt-google-cdn": "^0.4.0",  "grunt-karma": "^0.9.0",  "grunt-newer": "^0.7.0",  "grunt-ng-annotate": "^0.4.0",  "grunt-svgmin": "^0.4.0",  "grunt-usemin": "^2.1.1",  "grunt-wiredep": "^1.7.0",  "jasmine-core": "^2.1.3",  "jshint-stylish": "^0.2.0",  "karma": "^0.12.31",  "karma-chai": "^0.1.0",  "karma-chrome-launcher": "^0.1.7",  "karma-coverage": "^0.2.7",  "karma-html2js-preprocessor": "^0.1.0",  "karma-jasmine": "^0.3.4",  "karma-mocha": "^0.1.10",  "karma-ng-html2js-preprocessor": "^0.1.2",  "karma-phantomjs-launcher": "^0.1.4",  "karma-sinon": "^1.0.4",  "load-grunt-tasks": "^0.4.0",  "mocha": "^2.1.0",  "should": "^4.4.2",  "sinon": "^1.12.2",  "time-grunt": "^0.3.1"  },  "engines": {  "node": ">=0.10.0"  },  "scripts": {  "test": "grunt test"  }  } |

* At the same time corresponding packages are also added in the node\_modules folder of project directory. The node\_modules folder would be looking like :



As of now we are prepared with the basic environment setup and the application is ready for writing the test cases.

# **Reference(s)**

1. <https://docs.angularjs.org/>

2. <http://mochajs.org/>

3. <http://chaijs.com/>