**Inclass 3**

* Due October 7, 6PM
* Submit your answers in this document to the inclass 3 drop box.
* 20% deducted each day this is late.
* Worth 8% of your final mark.

Contents

[E2E Testing 2](#_Toc462980813)

[Protractor 2](#_Toc462980814)

[Jasmine 2](#_Toc462980815)

[*describe* Blocks 3](#_Toc462980816)

[*it* Blocks (specs) 3](#_Toc462980817)

[Document Locator Methods 3](#_Toc462980818)

[Web Element Functions 3](#_Toc462980819)

[Browser Functions 4](#_Toc462980820)

[Testing Set-Up 4](#_Toc462980821)

[Installing and Running Protractor on Windows 4](#_Toc462980822)

[Installing and Running Protractor on a Mac 8](#_Toc462980823)

[XPath 15](#_Toc462980824)

[UnitTesting 16](#_Toc462980825)

[Karma 16](#_Toc462980826)

[Jasmine 16](#_Toc462980827)

[Unit Testing Set-Up 16](#_Toc462980828)

[Install Jasmine 17](#_Toc462980829)

[Configuring Karma 17](#_Toc462980830)

[Run the Test 19](#_Toc462980831)

[Testing Simple Controllers 20](#_Toc462980832)

[Testing Simple Services 23](#_Toc462980833)

# E2E Testing

As applications grow, manual testing can become very difficult. End to end tests can help test the interface.

## Protractor

Protractor is a Node.js program that the AngularJS team built to perform end-to-end tests. Protractor uses WebDriver to run most main stream browsers and simulate user actions in the browser. Protractor also uses Jasmine syntax to perform test assertions.

* Commands tell Protractor to do something with the application such as navigate to a page or click on a button.
* Expectations tell Protractor to assert something.

When testing with Protractor, a test file is comprised of one or more suites of tests which contain assertions that use Jasmine syntax.

## Jasmine

Jasmine is a test driven development framework for JavaScript. Jasmine offers simple but powerful syntax to set up and manage tests.

#### Assertions

Jasmine helps you to organize your tests and perform assertions. Assertions are simple Boolean expressions that evaluation to true or false.

##### Expectations and Matchers

Jasmine enables assertions with **expectation functions** and **matchers**. The expectation function receives the actual value. The expectation is chained with a matcher that performs a Boolean comparison between the actual value and an expected value. Among many other functions, matchers may include:

to(Not)Be(null/true/false)

to(Not)Equal(value)

to(Not)Match(regex/string)

toBeDefined()

toBeUndefined()

toBeNull()

toBeTruthy()

toBeFalsy()

|  |
| --- |
| expect(nameService.getName()).not.toBeUndefined();  expect(nameService.getName()).toBe('Mary'); |

### *describe* Blocks

*describe* blocks group a **suite** of tests together while also stating the purpose of the tests.

|  |
| --- |
| describe('This is a test for my AppCtrl controller: ', function () {  }); |

### *it* Blocks (specs)

*it* blocks define each test - the it block is known as a **spec**. The test may contain one or more assertions.

|  |
| --- |
| describe('This test inspects the name model: ', function () {  ...  it('Should show Mary.', function () {  expect(nameService.getName()).toBe('Mary');  });  }); |

### Document Locator Methods

To find objects in a document, these locator methods (in addition to many others) can be applied to locate items:

by.binding('{{status}}')

by.className('redBtn')

by.css('.redBtn')

by.id('loginButton')

by.input("user")

by.linkText('Go Home')

by.partialLinktext('Home')

by.model('message')

by.name('email')

by.repeater("cat in pets")

by.repeater("cat in pets").row(1).column("{{cat.name}}"))

by.select("user")

by.selectedOption("red")

by.tagName('h2')

by.xpath('')

## Web Element Functions

To assist with testing, several additional functions can be used to extract data from elements and to perform actions on elements.

|  |  |
| --- | --- |
| **clear()** | If this element is a text entry element, this will clear the value. |
| **click()** | Click this element. |
| **getAttribute(name)** | Get the value of a given attribute of the element. |
| **getCssValue(propertyName)** | Get the value of a given CSS property. |
| **getLocation()** | Where on the page is the top left-hand corner of the rendered element? |
| **getSize()** | What is the width and height of the rendered element? |
| **getTagName()** | Get the tag name of this element. |
| **getText()** | Get the visible (i.e. not hidden by CSS) innerText of this element, including sub-elements, without any leading or trailing whitespace.  getText() does not work with the input element. Instead use the following:  var firstNameElement = element(by.model('firstName'));  var inputContents = firstNameElement.getAttribute('value'); |
| **isDisplayed()** | Is this element displayed or not? This method avoids the problem of having to parse an element's "style" attribute. |
| **isEnabled()** | Is the element currently enabled or not? This will generally return true for everything but disabled input elements. |
| **isSelected()** | Determine whether or not this element is selected or not. |
| **sendKeys(keysToSend)** | Use this method to simulate typing into an element, which may set its value. |

## Browser Functions

To run a test in a browser, the following functions can be used to direct a browser to a site or get information about the page title.

|  |  |
| --- | --- |
| **get()** - Gets url. | browser.get('http://ssdprogram.ca/protractor/index.html'); |
| **getTitle()** - Gets page title. | browser.getTitle() |

## Testing Set-Up

There are several steps needed to prepare for testing AngularJS applications with protractor.

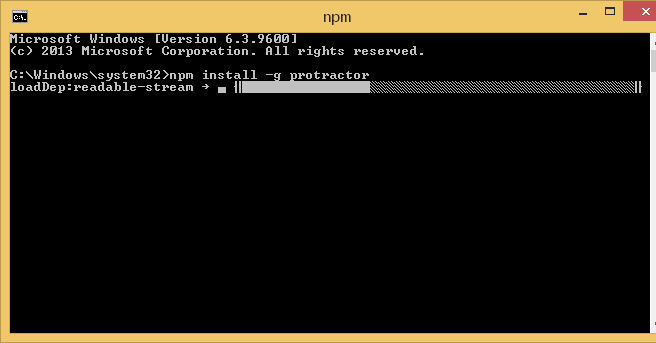
### Installing and Running Protractor on Windows

1. Download and install the appropriate Nodejs installer from

https://nodejs.org/en/

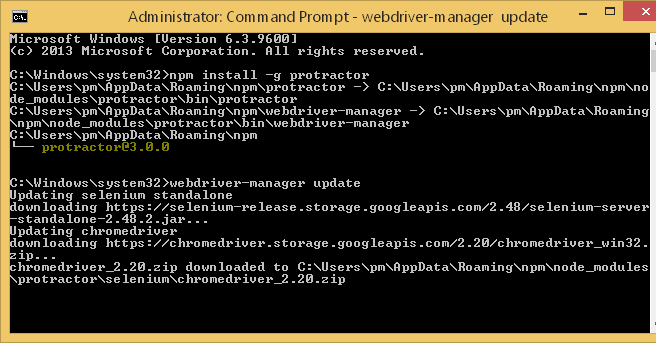
2. Then while running the command prompt as administrator enter the following command and press return:

**npm install –g protractor**



3. Then run the following command to update webdriver-manager:

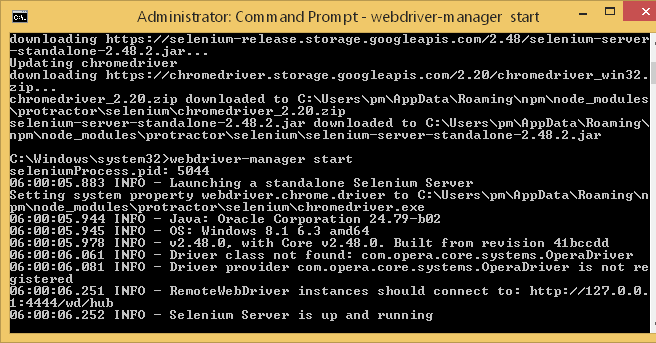
**webdriver-manager update**



4. Next enter the command to start webdriver-manager which is a server needed to run selenium which enables the interface tests:

**webdriver-manager start**

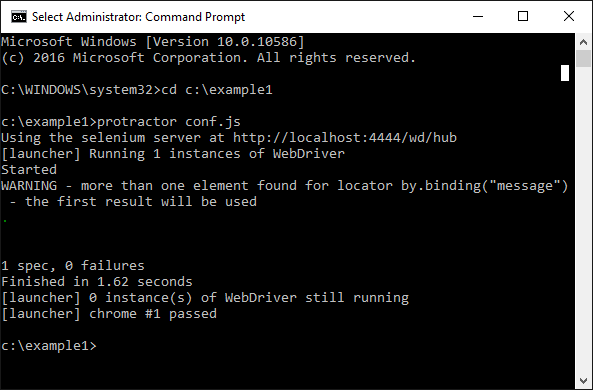
|  |
| --- |
| **If you get this error**    Download and install the latest JRE.  [**http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html**](http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html)  Next make sure your environment variable to your java.exe file is set and reboot. |



5. Download and extract the files from example1 for today’s notes and then place them inside a folder such as C:\angularTest

6. While webdriver-manager is running from step 4, in a separate command prompt that you open as administrator run the command at the example directory:

**protractor conf.js**



If 0 failures are reported your install was a success.

### Installing and Running Protractor on a Mac

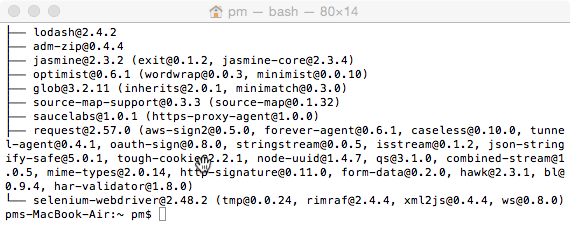
1. Download and install the appropriate NodeJs installer from

<https://nodejs.org/en/> (Choose all defaults)

2. Then in terminal run the command to install Protractor:

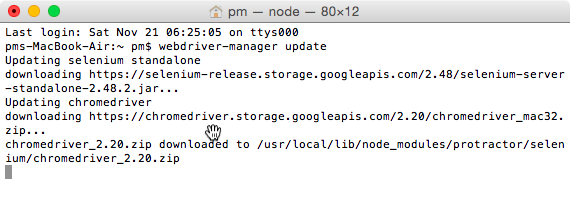
**npm install –g protractor**

Hopefully you should see a bunch of output that looks like the following:



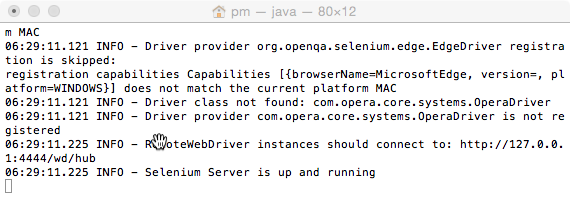
3. Then enter the command to update webdriver-manager to get the latest version:

**webdriver-manager update**



4. Next enter the command to start webdriver-manager. This launches a server that enables the web interface tests:

**webdriver-manager start**



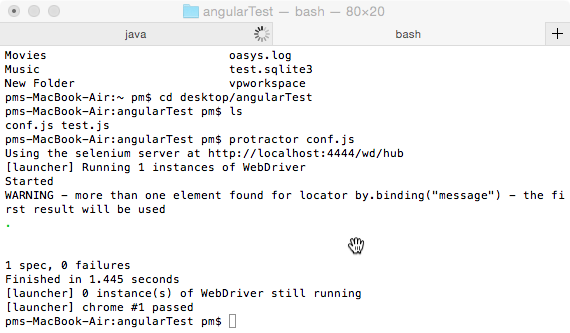
5. Download and extract the files from example1 for today’s notes and then place them inside a folder such as Users/username/desktop/angularTest

6. While webdriver-manager is running from step 4, in a second terminal tab enter the command:

**protractor conf.js**

****

When you press enter,if you see the test ran and there were 0 failures your install was a success.

****

Example : Testing the Protractor Install

🞑 Now, this is the AngularJS application that was tested during the install. To test Angular applications with Protractor, you need to serve your HTML file from a web server like Apache or you can use my AngularJS application which is hosted at http://ssdprogam.ca/protractor/index.html. Here is the code for it.

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- The ng-app directive -->  <body ng-app='' ng-init="message='Hello world!'">  Message: <input type="text" ng-model="message">  <p ng-bind="message"></p>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  </body>  </html> |

Here is the test for the angular application. This is the test that was run during the installation. You will need to adjust this link highlighted in yellow if you are hosting your application locally.

**test.js**

|  |
| --- |
| describe('angularjs homepage todo list', function () {  it('should add a todo', function () {  // Add in your test url.  browser.get('http://ssdprogram.ca/protractor/index.html');  expect(browser.getTitle()).toEqual("Starting Angular");  var inputElement = element(by.model('message'));  // getText() does not work with input elements.  expect(inputElement.getAttribute('value')).toEqual('Hello world!');  expect(element(by.binding('message')).getText())  .toEqual('Hello world!');  });  }); |

The conf.js file tells protractor where the webdriver-manager is running and where the test.js file is. Here is the copy of the conf.js file that was used during the installation. The default selenium address and port of 4444 are also listed.

**conf.js**

|  |
| --- |
| exports.config = {  seleniumAddress: 'http://localhost:4444/wd/hub',  specs: ['test.js']  }; |

Exercise

🖍 What AngularJS specific references are made to locate items in the index.html file during the end to end test in Example 1?

|  |
| --- |
| by.model(‘message’)  by.binding(‘message’) |

Exercise

🖍 What JavaScript specific references are used to locate items in index.html in ?

|  |
| --- |
| getTitle()  getAttribute()  get() |

Exercise

🖍 The following script is located at **http://ssdprogram.ca/protractor/index2.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Check boxes</title>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  <script>  var formSample = angular.module('formSample', []);  formSample.controller('ExampleController', ['$scope', function ($scope) {  $scope.chk1 = true;  $scope.chk2 = 'Hello';  $scope.update = function () {  alert('hi');  }  }]);  </script>  </head>  <body ng-app="formSample">  <form name="myForm" ng-controller="ExampleController">  Value1: <input type="checkbox" ng-model="chk1" ng-change="update()"> <br />  Value2: <input type="checkbox" ng-model="chk2"  ng-true-value="'Hello'" ng-false-value="'Good-bye'"> <br />  <p>chk1 = <label ng-bind="chk1"></label> </p>  <p>chk2 = <label ng-bind="chk2"></label></p>  </form>  </body>  </html> |

Using protractor with jasmine syntax, write a test that:

* verifies the page title.
* verifies that the first checkbox is checked.
* verifies that the label which is bound to chk2 has a value of ‘Hello’.

**Big Hint:**

**Find the element by model and then use this test expect(myElement.checked);**

In your test, remember to modify the location of the server to your local server where the page is hosted.

Show your test here:

|  |
| --- |
| describe('exercise one test', function () {  it('should check the title page', function () {  // Add in your test url.  browser.get('http://ssdprogram.ca/protractor/index2.html');  expect(browser.getTitle()).toEqual("Check boxes");  });  it('should check the first checkbox is checked', function () {  var checkbox = element(by.model('chk1'));  expect(checkbox.checked);  });  it('should check the second checkbox label is Hello', function () {  var checkbox = element(by.binding('chk2'));  expect(checkbox.getText()).toEqual('Hello');  });  }); |

Show a screenshot of your second command/terminal window which shows the test results:

|  |
| --- |
|  |

Example : Testing Sending Text, Clearing Inputs, and Button Clicks

🞑 Here is a more advanced example that shows how to test other controls on a web page. Note this test references a index3.html file which I have hosted on my website but you may wish to test it on a local http server.

**test.js**

|  |
| --- |
| describe('Protractor tutorial.', function () {  // Added this helper function. getText() does not work with <input/> element.  var myGetText = function (element) {  var inputText = element.getAttribute('value');  return inputText;  }  it('Checking page items.', function () {  var firstNameElement = element(by.model('firstName'));  var nameInputText;  // Must get address from live server.  browser.get('http://ssdprogram.ca/protractor/index3.html');  // Check Title  expect(browser.getTitle()).toEqual('Starting Angular');  // Check input box contents.  nameInputText = myGetText(firstNameElement);  expect(nameInputText).toEqual('Josie');  // Check input box.  element(by.model('firstName')).sendKeys(' Louis');  nameInputText = myGetText(firstNameElement);  expect(nameInputText).toEqual('Josie Louis');  // Check submitted input in paragraph tag.  var btnName = element(by.id('btnName'));  btnName.click();  var elemGreeting = element(by.binding('submittedName'));  expect(elemGreeting.getText()).toEqual('Greetings: Josie Louis');  // Check cleared input box.  element(by.model('firstName')).clear();  nameInputText = myGetText(firstNameElement);  expect(nameInputText).toEqual('');  });  }); |

**js/app.js**

|  |
| --- |
| // Declare module that references our controllers.  var myApp = angular.module('myApp', ['myControllers']);  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  myControllers.controller('AppCtrl', ['$scope', function ($scope) {  $scope.title = "Angular JS!";  $scope.firstName = "Josie";  $scope.error = false;  $scope.submittedName = '';  $scope.numbers = ["1", "2", "3", "4", "5", "6", "7", "8", "9"];  $scope.add = function (data) {  $scope.submittedName = 'Greetings: ' + data;  }  }]);  return myControllers;  }()); |

This file is hosted at http://ssdprogram.ca/protractor/index3.html. However, you may wish to host it locally.

**index3.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- Notice here that ng-app references our 'myApp' module. -->  <body ng-app='myApp' ng-controller="AppCtrl">  <h1 ng-bind="title"></h1>  <!-- Name input. -->  <input type="text" id="nameInput" ng-model="firstName" />  <p id="greeting" ng-bind="submittedName"></p>  <button id="btnName" class="nameInput" ng-click="add(firstName)">Upddate</button>  <!-- Reference the application level controller for the title model. -->  <ul ng-repeat="number in numbers">  <li ng-bind="number"></li>  </ul>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  <script src="./js/app.js"></script>  </body>  </html> |

Exercise

🖍 See if you can verify if the number count in is 9. You can do this by testing the repeater. Start with syntax like the following ;

|  |
| --- |
| element.all(by.repeater('number in numbers')).count().then(function(count)  {  console.log(count);  }); |

Show your revised test script here:

|  |
| --- |
| describe('exercise 4 test', function () {  it('count of elements should be 9', function () {  // Add in your test url.  browser.get('http://ssdprogram.ca/protractor/index3.html');    element.all(by.repeater('number in numbers')).count().then(function(count)  {  expect(count).toEqual(9);  });  });  }); |

## XPath

I am not going to cover XPath since this is not an AngularJS topic. Basically though XPath allows you to query an HTML or XML document. Sometimes auto-generated pages generate elements without id’s so you may want to use XPath to navigate from one element with an id to a neighboring or nested element which does not have an id.

Example

🞑 While this example is far from complete, it has been simplified to show how XPath might be used to navigate through the DOM. This example shows how you could implement a test to detect if a validation message is present in an ASP.NET MVC application. First, the test checks for an element with a specific id. Then, an XPath query is made to obtain the sibling element. Then, the value of the attribute *data-valmsg-for* is tested to ensure it matches the id value. Lastly a test is made to determine if the validation message is displayed.

**test.js**

|  |
| --- |
| describe('angularjs homepage todo list', function () {  it('should add a todo', function () {  // Add in your test url where your application is hosted.  browser.get('http://ssdprogram.ca/protractor/index4.html');  var firstNameElem // Get input element by id.  = element(by.id("FirstName"));  var firstNameMsg // Get neighbouring element.  = firstNameElem.element(by.xpath('following-sibling::span'));  // Check attribute value of neighbouring element.  expect(firstNameMsg.getAttribute('data-valmsg-for')).toEqual("FirstName");  // Determine if text in sibling element is displayed.  expect(firstNameMsg.isDisplayed()).toBe(true);  });  }); |

For good measure, here is the HTML needed to display the validation message.

index4.html

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <body ng-app=''>  <input data-val="true" data-val-required="A first name is required."  id="FirstName" name="FirstName" type="text" value="">  <!-- Sibling of input element. -->  <span class="text-danger field-validation-error" data-valmsg-for="FirstName"  data-valmsg-replace="true">  <span for="FirstName" class="">A first name is required.</span>  </span>  <script src="https://code.angularjs.org/1.5.8/angular.js">  </script>  </body>  </html> |

# UnitTesting

Now we will discuss how to perform basic unit tests. Unit tests allow you to validate individual sections of code. The AngularJS team recommends using Karma and Jasmine for testing.

## Karma

Karma is a command line module which runs via NodeJS. Karma spawns a web browser of your choice to ensure your application works in the browser you need.

## Jasmine

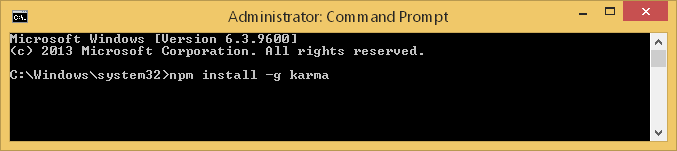
Similar to protractor for end-to-end testing, karma uses jasmine syntax for performing assertions and for grouping test suites of specs. We also can use **beforeEach** and **afterEach** functions for set up and tear down before each ***it*** block. The **beforeEach** function is called once before each spec is run. The **afterEach** function is called once after a test.

|  |
| --- |
| describe("A spec with setup and tear-down", function () {  var total;  beforeEach(function () {  total = 1;  total += 1;  });  afterEach(function () {  total = 0;  });  it("is just a function, so it can contain any code", function () {  expect(total).toEqual(1);  });  it("can have more than one expectation", function () {  expect(total).toEqual(1);  expect(true).toEqual(true);  });  }); |

## Unit Testing Set-Up

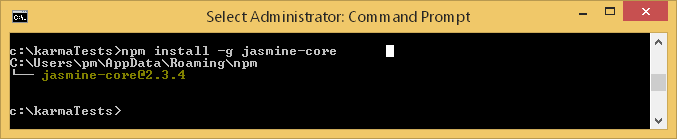
There are several steps needed to prepare for unit testing with AngularJS. After installing Nodejs, open the command prompt as Administrator and then run the command:

**npm install -g karma**



### Install Jasmine

To install Jasmine, type “npm install -g jasmine-core”



### Configuring Karma

To create a configuration file for Karma:

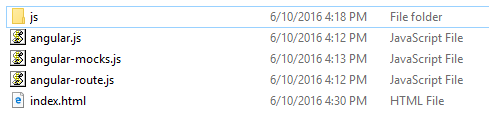
* Create a folder and place your test files in it. For this case, you could take the files for from today’s notes and place them in the folder.
* Also, place the following files in it:

[https://code.angularjs.org/1.5.8/angular.js](https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js)

[https://code.angularjs.org/1.5.8/angular-route.js](https://code.angularjs.org/1.5.6/angular-route.js)

[https://code.angularjs.org/1.5.8/angular-mocks.js](https://code.angularjs.org/1.5.6/angular-mocks.js)

Once finished your folder structure will look like the following:



**Note:** You do not actually need the index.html file since this is not being tested.

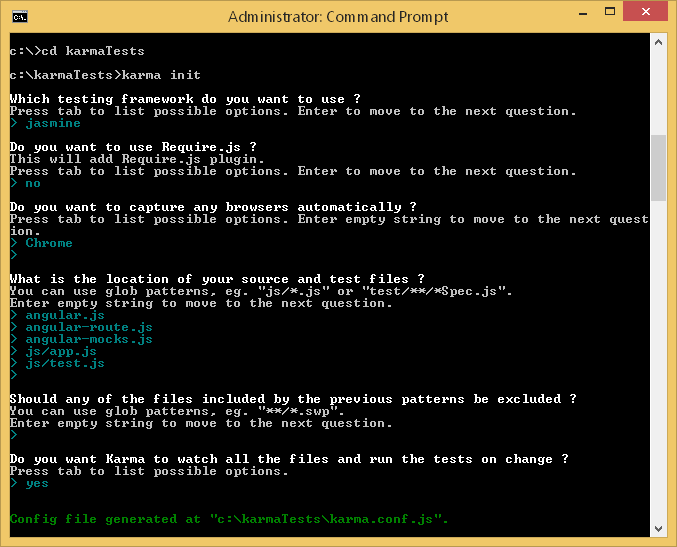
* While running the command prompt as Administrator, navigate to the folder using the “cd” command to change the directory.
* Once your command prompt is pointing to the folder where your AngularJS and application resides type:

**karma init**

This instruction will initiate a series of prompts to help you set up your test file. The first question asks which framework you would like to use. Here, Jasmine is selected by default but you can tab to others. For this demonstration though we are going to use Jasmine. When prompted for browsers you may select one or several different browser types. Click tab to choose another browser.

When prompted to select files, you might find it easier to place all javascript files you reference in the C:\Program Files\nodejs\node\_modules\karma\bin directory. To prepare, download and store the AngularJS libraries to your bin folder:

Note: You can find the content for js/app.js and js/test.js in .



You can choose to watch all files so they are tested as soon as changes are made to your project. I like this option because you don’t have to wait for karma to launch to run the tests as you write your code. If you prefer to run the test only when you choose select “no” when prompted.

After finishing the configuration questions, a karma.conf.js file is output to the project directory. Below is a sample file which contains references to the app.js and test.js files in my project directory.

**karma.conf.js**

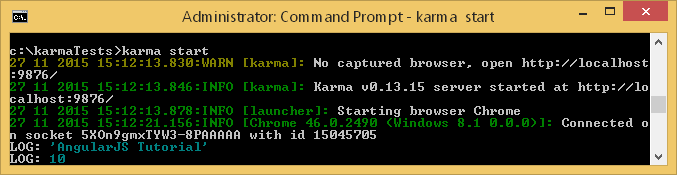


To reconfigure Jasmine, you can adjust the values in the karma.conf.js file with a text editor if you prefer. Or, you can delete the existing file and let the wizard guide you through the list of questions another time.

### Run the Test

Run the test from the Command Prompt while running it as Administrator while pointing to your project directory by typing:

**Karma start**



## Testing Simple Controllers

This next section shows how to perform simple tests on a controller.

Example : Testing a Controller

🞑 This example shows how to set up testing for an AngularJS controller. Details about how the test file is structured are described in the comments of the js/test.js file.

**js/app.js**

|  |
| --- |
| // Declare module that references our controllers.  var myApp = angular.module('myApp', ['cardAppControllers']);  var cardAppControllers = (function () {  var cardAppControllers = angular.module('cardAppControllers', []);  // Declare the application controller and inject the scope reference.  cardAppControllers.controller('AppCtrl', ['$scope', function ($scope) {  // Define the title model.  $scope.title = "AngularJS Tutorial";  $scope.StartNum = 10;  $scope.StartString = "Hello!";  }]);  return cardAppControllers;  }()); |

**js/test.js**

|  |
| --- |
| // This is the start of the test suite.  describe('Unit: AppCtrl', function () {  // Load the module with AppCtrl  beforeEach(module('myApp'));  var ctrl, scope;  // inject the $controller and $rootScope services  // in the beforeEach block  beforeEach(inject(function($controller, $rootScope) {  // Create a reference to the $rootScope  scope = $rootScope.$new();  // Create the controller  ctrl = $controller('AppCtrl', {  $scope: scope  });  }));  // This is the start of the spec.  it('AppCtrl should have expected values for title, StartNum, StartString',  function() {  expect(scope.title).toEqual("AngularJS Tutorial");  console.log(scope.title);  expect(scope.StartNum).not.toBeUndefined();  expect(scope.StartNum).toEqual(10);  console.log(scope.StartNum);  expect(scope.StartString).toEqual("Hello!");  });  }) |

**index.html**

|  |
| --- |
| <!DOCTYPE html>  <html ng-app="myApp">  <head>  <title></title>  </head>  <body ng-controller="AppCtrl">  <h1 ng-bind="title"></h1>  <ng-view></ng-view>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  <script src="js/app.js"></script>  </body>  </html> |

Exercise

🖍 Fill in the blanks with the following terms. Use each term only once:

it, actual, expected, beforeEach, describe

a) \_\_\_\_\_\_\_\_\_describe\_\_\_\_\_\_\_\_\_ blocks group tests together.

b) Expected functions receive the \_\_\_\_\_\_\_actual\_\_\_\_\_\_\_\_ value.

c) \_\_\_\_\_\_\_\_\_it\_\_\_\_\_\_\_\_\_ blocks group assertions together.

d) Matchers receive the \_\_\_\_\_\_expected\_\_\_\_\_\_ value.

e) \_\_\_\_\_\_\_\_\_beforeEach\_\_\_\_\_\_\_\_\_ blocks perform a series of instructions before each test.

In Jasmine we use the describe function to group our tests together:

Exercise

🖍 What does a *describe* statement do? (1 mark)

|  |
| --- |
| Describe group up a suite of tests, it is also the test initializer. |

Exercise

🖍 What does an *it* statement do? (1 mark)

|  |
| --- |
| The it statement define each test or spec which can contains one or more assertions. |

Exercise

🖍 When testing the *AppCtrl* controller in , indicate if the following tests pass or fail. (2 marks)

a) \_\_\_\_\_\_PASS\_\_\_\_\_\_ expect(scope.JasmineTest).toBeUndefined();

b)\_\_\_\_\_\_PASS\_\_\_\_\_\_ expect(scope.StartNum).toBeGreaterThan(0);

Exercise

🖍 Writing tests can be really challenging at times so you need techniques to debug your tests. What debugging technique is used inside test.js?

|  |
| --- |
| The technique used is the console.log to log some values during the test. |

Exercise

🖍 Add the following declaration to the AppCtrl controller in : (2 marks)

$scope.Started = true;

List five different tests to verify prove *$scope.Started* is *true*. You may need to look up Jasmine syntax on Google to answer this. Test your answers to verify they are correct.

|  |
| --- |
| expect(scope.Started).toBeDefined();  expect(scope.Started).not.toBeUndefined();  expect(scope.Started).not.toBeNull();  expect(scope.Started).not.toBe(false);  expect(scope.Started).toBe(true); |

Exercise

🖍 Unit tests should always be written in files that are separate from your project code. Why? (1 mark)

|  |
| --- |
| Because the unit test must work without the interference of the project and project code. |

Exercise

🖍 Given the following code;

var myApp = angular.module('myApp', ['myControllers']);

// Controller - dispatches inputs and outputs.

var myControllers = (function () {

var myControllers = angular.module('myControllers', []);

// Controllers are defined by the controller function.

myApp.controller('ExampleController', ['$scope', function ($scope) {

$scope.update = function (input) {

$scope.mailService = input;

}

}]);

return myControllers;

}());

Write a unit test to check to ensure the *mailService* model was assigned a value after calling the *update()* function. Show your entire test.js file needed to create this test: (4 marks)

|  |
| --- |
| // This is the start of the test suite.  describe('Unit: AppCtrl', function () {  // Load the module with AppCtrl  beforeEach(module('myApp'));  var ctrl, scope;  // inject the $controller and $rootScope services  // in the beforeEach block  beforeEach(inject(function($controller, $rootScope) {  // Create a reference to the $rootScope  scope = $rootScope.$new();  // Create the controller  ctrl = $controller('AppCtrl', {  $scope: scope  });  }));  // This is the start of the spec.  it('mailService should have the input value used in the update function after it is called',  function() {  let input = "test";  scope.update(input);  expect(scope.mailService).toEqual(input);  });  }) |

## Testing Simple Services

AngularJS helps to enforce separation of logic and presentation by restricting how you implement AngularJS solutions. Dependency injection is built in to AngularJS so you can test each unit separately without worrying about external services. During unit tests, you must mock service dependencies when testing a specific unit. For example, unit tests will not allow for external dependencies involving XHR or JSONP requests to a real server. However, we can mock these services in order to focus on a specific unit which uses these services.

Here is a sample that mocks (fakes) a service called *nameService*. Notice how the inject block receives the *nameService* and creates a local instance of it. The underscore before and after the nameService parameter (\_nameService\_), is optional but allows us to create a local instance with the same name as our original service.

|  |
| --- |
| describe('nameService tests: ', function () {  var nameService;  beforeEach(function () { // Execute before each spec is run.  module('myApp'); // Load module.  // Inject service for testing.  inject(function (\_nameService\_) { // The \_underscores\_ are for convenience  nameService = \_nameService\_; // so local service can have same name.  });  });  it('Should show Mary.', function () { // Test service method.  var result = nameService.getName();  expect(result.First).toBe('Mary');  });  }); |

Example : Testing a Service

🞑 This example demonstrates how to test a simple service.

The AngularJS, controller uses the service to obtain a first name.

**js/app.js**

|  |
| --- |
| // Module.  var myApp = angular.module('myApp', ['myControllers']);  // Controllers.  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  // Inject scope and service reference.  myControllers.controller('AppCtrl', ['$scope', 'nameService',  function ($scope, nameService) {  $scope.personName = nameService.getName(); }]);  return myControllers;  }());  // Service  myApp.factory("nameService", function () {  return {  getName: function () {  return { "First": "Mary", "Last": "Jane" };  }  }  }); |

The test is set up with an injection of the nameService in the *beforeEach()* block. The service is stored in a local reference which is then used to call the *getName()* function of the service.

**js/test.js**

|  |
| --- |
| describe('nameService tests: ', function () {  var nameService;  beforeEach(function () { // Execute before each "it" test is run.  module('myApp'); // Load module.  // Inject service for testing.  inject(function (\_nameService\_) {// The \_underscores\_ are for convenience  nameService = \_nameService\_; // so local service can have same name.  });  });  it('Should show Mary.', function () { // Test service method.  var result = nameService.getName();  expect(result.First).toBe('Mary');  });  }); |

The first name is displayed when running the application.

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head><title>Starting Angular</title></head>  <body ng-app='myApp' ng-controller="AppCtrl">  <h1 ng-bind="personName.First"></h1>  <script src="https://code.angularjs.org/1.5.8/angular.js"></script>  <script src="js/app.js"></script>  </body>  </html> |

Remember, when running this test in karma you have to reference the app.js and test.js file.

Exercise

🖍 Modify the test in by adding a separate *it* block which verifies that the last name is ‘Jane’.

|  |
| --- |
| describe('nameService tests: ', function () {  var nameService;  beforeEach(function () { // Execute before each "it" test is run.  module('myApp'); // Load module.  // Inject service for testing.  inject(function (\_nameService\_) {// The \_underscores\_ are for convenience  nameService = \_nameService\_; // so local service can have same name.  });  });  it('Should show Mary.', function () { // Test service method.  var result = nameService.getName();  expect(result.First).toBe('Mary');  });    it('Last name should show Jane.', function () { // Test service method.  var result = nameService.getName();  expect(result.Last).toBe('Jane');  });  }); |