# Due March 27, 6PM

20% deducted each day this is late.

Worth 10% of your mark.

Please complete the exercises in this document and submit it to the drop box when you are finished.

Contents

[Due March 27, 6PM 1](#_Toc414636472)

[Testing AngularJS 2](#_Toc414636473)

[Karma 2](#_Toc414636474)

[Jasmine 2](#_Toc414636475)

[*describe* Blocks 2](#_Toc414636476)

[*it* Blocks (specs) 2](#_Toc414636477)

[Setup and Teardown 3](#_Toc414636478)

[Unit Testing Set-Up 3](#_Toc414636479)

[Installing Git 3](#_Toc414636480)

[Installing Node.js 4](#_Toc414636481)

[Installing the Karma Module 4](#_Toc414636482)

[Configure Jasmine 5](#_Toc414636483)

[Testing 7](#_Toc414636484)

[Testing Services 11](#_Toc414636485)

[Mocking Services 13](#_Toc414636486)

[Testing a Controller with Dependencies 17](#_Toc414636487)

[Testing Directives 21](#_Toc414636488)

[Testing a Directive with a Template URL 23](#_Toc414636489)

# Testing AngularJS

Today 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

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 just Boolean expressions that evaluation to true or false.

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()

|  |
| --- |
| expect(nameService.getName()).toBe('Mary'); |

### *describe* Blocks

*describe* blocks group tests together while also stating the purpose of the tests. This block of tests is known as a **spec**.

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

### *it* Blocks (specs)

*it* blocks define each test. 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');  });  }); |

### Setup and Teardown

To help with setup and resetting values before and after each it block Jasmine provides the global **beforeEach** and **afterEach** functions. 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);  });  }); |

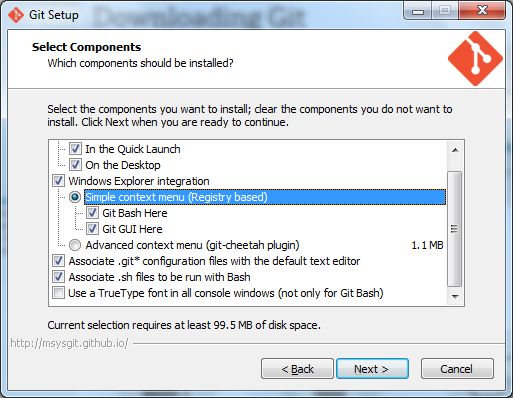
## Unit Testing Set-Up

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

### Installing Git

If Git isn’t on the machine already, download and install it:

<http://git-scm.com/download/win>

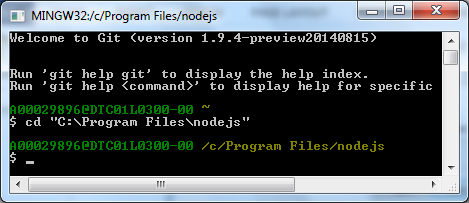


### Installing Node.js

After getting Git, download and install the appropriate Node.js installer from <http://nodejs.org/download/>

### Installing the Karma Module

Installation of the karma module is odd on Windows. I have been able to make it work only after installing it twice. From the GitBash command line while running it as Administrator, navigate to C:\Program Files\nodejs with the cd command to change directories. Note, if a space exists in the file path you will need to place the path in quotes.



Then run the command:

**npm install -g karma**

After, run the command:

**npm install karma**

After Karma is installed, you should see the directory “C:\Program Files\nodejs\node\_modules\karma” within the Node.js path.

### Configure Jasmine

To create a configuration file for Jasmine, at the “C:\Program Files\nodejs\node\_modules\karma\bin” folder type the command:

**karma init**

This will initiate a series of questions. 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:

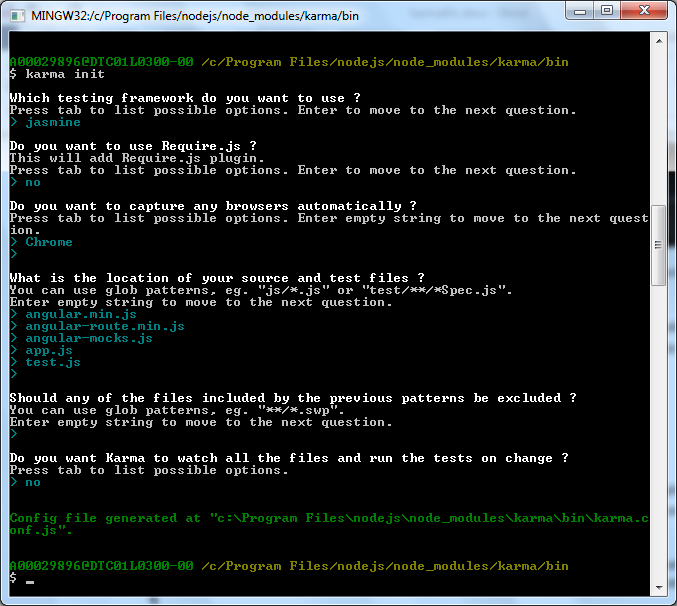
[https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js](https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular.min.js)

[https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.min.js](https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular-route.min.js)

[https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-mocks.js](https://ajax.googleapis.com/ajax/libs/angularjs/1.3.3/angular-mocks.js)

If you have trouble identifying the correct file path to your project JavaScript files, just place them in the C:\Program Files\nodejs\node\_modules\karma\bin folder too. Or, you can reference them from their project directory but be sure to use forward slashes.

In the screenshot below I have referenced all JavaScript files in their load order and they are currently in the C:\Program Files\nodejs\node\_modules\karma\bin directory.



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 so it saves time. 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 C:\Program Files (x86)\nodejs\node\_modules\karma\bin 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.

## Testing

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

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', ['ngRoute', 'cardAppControllers'])  .config(function ($routeProvider) {  $routeProvider.when("/home", {  //templateUrl: 'views/main.html',  //controller: 'AppCtrl'  })  // '/home/number/:numberID/suit/:suitID'  .when("/aLink/num/:numID/str/:strID", {  templateUrl: 'views/aView.html',  controller: 'ACtrl'  })  // If no route is selected then use the 'home' route.  .otherwise({ redirectTo: '/home' });  });  var cardAppControllers = (function () {  var cardAppControllers = angular.module('cardAppControllers', []);  // Declare the application controller and inject the scope reference.  cardAppControllers.controller('AppCtrl', ['$scope', function ($scope) {  // Define the title model.  $scope.title = "AngularJS Tutorial";  $scope.StartNum = 10;  $scope.StartString = "Hello!";  }]);  // Declare controller that populates 'list.html' with data and inject the scope.  cardAppControllers.controller('ACtrl', ['$scope', '$routeParams',  function ($scope, $routeParams) {  // Define the cards model.  $scope.subtitleA = "Detail A";  $scope.myNum = $routeParams.numID;  $scope.myString = $routeParams.strID;  }]);  return cardAppControllers;  }()); |

**js/test.js**

|  |
| --- |
| 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 new scope that's a child of the $rootScope  scope = $rootScope.$new();  // Create the controller  ctrl = $controller('AppCtrl', {  $scope: scope  });  }));  it('AppCtrl should have expected values for title, StartNum, StartString',  function() {  // expect(scope.greeting).toBeUndefined();  // scope.sayHello();  expect(scope.title).toEqual("AngularJS Tutorial");  console.log(scope.title);  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://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.min.js"></script>  <script src="js/app.js"></script>  </body>  </html> |

To run karma enter at the gitbash prompt at the C:\Program Files (x86)\nodejs\node\_modules\karma\bin directory enter the following prompt:

**karma start karma.conf init**

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 blocks group tests together while also stating the purpose of the tests. |

Exercise

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

|  |
| --- |
| it blocks define each test. The test may contain 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?

|  |
| --- |
| Comment out statements in order to isolate units and not test multiple components at the same time. |

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 to answer this. Test your answers to verify they are correct.

|  |
| --- |
| expect(scope.Started).toBe(**true**);  expect(scope.Started).not.toBe(**false**);  expect(scope.Started).toEqual(**true**);  expect(scope.Started).not.toEqual(**false**);  expect(scope.Started).toBeTruthy(); |

Exercise

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

|  |
| --- |
| There are several reasons.   1. Search for a string in project/production source is easier. 2. Specify which files to include for code coverage 3. To simplify build scripts 4. If not, might confuse developers reading the code, and run the risk of shipping test code by accident. |

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)

|  |
| --- |
| describe('Unit: myControllers', **function** () {  beforeEach(module('myApp'));   **var** ctrl, scope;   beforeEach(inject(**function** ($controller, $rootScope) {  scope = $rootScope.$new();  ctrl = $controller('ExampleController', {  $scope: scope  });  }));   it('ExampleController should have expected values for mailService',  **function** () {  scope.update("The Mail Service");  expect(scope.mailService).toBeDefined();  expect(scope.mailService).toEqual("The Mail Service");  }); }) |

## Testing 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 can mock service dependencies to focus on a specific unit. For example, unit tests generally 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 "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');  });  }); |

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://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.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 app.test 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('Should show Jane.', **function** () {  **var** result = nameService.getName();  expect(result.Last).toBe('Jane');  }); }); |

## Mocking Services

To unit test controllers or specific services, you often will need to mock their dependencies. Remember, the goal of a unit test is to isolate units of code and not to test multiple components at once. For example, unit testing is not inherently designed to wait for XML or JSON responses so we can create mock responses with AngularJS *$httpBackend* service.

In the sample below, notice that the $httpBackend service is injected along with the myService reference. Once we have a local reference to both of these services, we can then call the *getNumbers()* function of the service that we are testing.

When $httpBackend.flush() is called, the http requests are sent and then the mock response is read.

|  |
| --- |
| describe('Testing a service with $httpBackend.', function () {  var myService, $httpBackend; // Local instances.  beforeEach(function () {  module('myApp'); // Load module.  inject(function (\_myService\_, \_$httpBackend\_) { // Inject services.  $httpBackend = \_$httpBackend\_;  myService = \_myService\_;  });  });  it("Test 'myService' with mock $http service.", function () {  // For this request, fake (mock) success response.  $httpBackend.expectGET('http://ssdprogram.ca/testJson.php')  .respond(200, [1, 2]);  var myData;  var response = myService.getNumbers();  response.then( function (data) { myData = data; });  $httpBackend.flush(); // Invoke mock http request.  expect(myData[1]).toEqual(2);  console.log("$httpBackEnd test results: " + myData);  });  }); |

Example : Testing with httpBackend

🞑 Here is a full example which demonstrates a test for an asynchronous call to an http service. The service to be tested contains a reference to the $http service of the AngularJS framework.

**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', ['$scope', 'myService',  function ($scope, myService) {  $scope.title = "Angular JS!";  $scope.error = false;  var promise = myService.getNumbers();  promise.then(  function (numbers) {  $scope.numbers = numbers;  },  function (errorReason) {  $scope.error = true;  $scope.msg = "An error occurred while fetching data.";  });  }]);  return myControllers;  }());  myApp.factory('myService', ["$http", "$q", function ($http, $q) {  var myService = {  getNumbers: function() {  var deferred = $q.defer();    $http({  method: 'GET',  url: 'http://ssdprogram.ca/testJson.php'  }).success(function (data) {  console.log("Success: " + data);  deferred.resolve(data);  }).error(function (data) {  console.log("Fail: Rejected!");  deferred.reject("Rejected!"); // Return rejection.  });  return deferred.promise; // Promise to return return  } // resource when available.  };  return myService;  }]); |

Notice that the mock response HTTP code is 200 which indicates that the response is successful. When this test runs, the mock data is retrieved in the *it* block and tests can be made to ensure it conforms to the expected data.

**js/test.js**

|  |
| --- |
| describe('Testing a service with $httpBackend.', function () {  var myService, $httpBackend; // Local instances.  beforeEach(function () {  module('myApp'); // Load module.  inject(function (\_myService\_, \_$httpBackend\_) { // Inject services.  $httpBackend = \_$httpBackend\_;  myService = \_myService\_;  });  });  it("Test 'myService' with mock $http service.", function () {  // For this request, fake (mock) success response for any  // request to http://ssdprogram.ca/testJson.php.  $httpBackend.expectGET('http://ssdprogram.ca/testJson.php')  .respond(200, [1, 2]);  var myData;  var response = myService.getNumbers();  response.then(  function (data) {  myData = data;  },  function (errorReason) {  myData = errorReason;  });  $httpBackend.flush(); // Invoke mock http request.  expect(myData[1]).toEqual(2);  console.log("$httpBackEnd test results: " + myData);  });  }); |

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <!-- Notice here that ng-app references our 'myApp' module -->  <body ng-app='myApp'>  <!-- Reference the application level controller for the title model. -->  <div ng-controller="AppCtrl">  <h1 ng-bind="title"></h1>  <p ng-show="**{{**msg != ''**}}**" ng-bind="msg"></p>  <ul ng-repeat="number in numbers">  <li ng-bind="number"></li>  </ul>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  <script src="js/app.js"></script>  </body>  </html> |

Exercise

🖍 Run this test in Jasmine with Karma when the *autoWatch* setting is true in the *karma.conf.js* file and the *singleRun* property is false. These are the default settings that are generated in the *karma.conf.js* file. Leave the browser set to either Chrome or Firefox so you can view the logging activity. Where in the application are the log values generated? In what order do the log values appear? Does this prove that the service is called?

|  |
| --- |
| The log values in this example are generated when the http call successfully receives response or when fails to receive the data. Also, log is generated when unit test is done.  The log values appear in the following order:  LOG: 'Success: 1,2'  LOG: '$httpBackEnd test results: 1,2'  No. It’s just a mock service behavior. |

Exercise

🖍 Modify test.js from Example 3 so the expected result and matcher instruction reads;

expect(myData[1]).not.toEqual(2);

As well, change the expected HTTP response from 200 to 400 which means bad request.

a) Then, run the test again. Which branch of code executes in the actual service to log data?

|  |
| --- |
| }).error(**function** (data) {  console.log("Fail: Rejected!");  deferred.reject("Rejected!"); // Return rejection. }); |

b) Does the test pass or fail?

|  |
| --- |
| Pass |

c) If we move the instruction;

$httpBackend.flush(); // Invoke mock http request.

to the back of the *it* block, why do we receive an error?

|  |
| --- |
| Only after $httpBackend.flush() is called, the http requests are sent and then the mock response is read. Otherwise, the response is undefined. |

## Testing a Controller with Dependencies

Testing a controller with dependencies presents some challenges. Generally, we need to inject all dependencies used by our controller test suite. In the sample below these dependencies include:

* A reference to $controller which is the service needed to create controllers.
* A reference to any custom service used.
* A reference to $rootScope which allows you to initialize the controller scope.
* A reference to the $httpBackend service if a call to a service which makes an HTTP request needs to be mocked.

The following snippet shows a declaration for a unit test with the AppCtrl controller which makes a call to a service that performs an HTTP request.

|  |
| --- |
| describe('Testing AppCtrl: ', function() {  "use strict";  var myService, $httpBackend, ctrl, scope;  beforeEach(function() {  module('myApp');  inject(function (\_myService\_, $controller, $rootScope,  \_$httpBackend\_) {  // Pretend we are loading controller for 1st time  // so we pass it a child of the $rootScope.  scope = $rootScope.$new();  // Assign the controller.  ctrl = $controller('AppCtrl', {  $scope: scope // Assign child of rootScope.  });  myService = \_myService\_;  $httpBackend = \_$httpBackend\_;  $httpBackend .expectGET('http://ssdprogram.ca/testJson.php')  .respond(200, ["21", "22"]);  });  }  );  }); |

Example : Testing a Controller that Calls a Custom Service which makes an HTTP call

🞑 Here is a complete example which performs a test on the AppCtrl controller.

**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', ['$scope', 'myService',  function ($scope, myService) {  $scope.title = "Angular JS!";  $scope.error = false;  $scope.numbers = [];  $scope.getData = function() {  var promise = myService.getNumbers();  promise.then(  function(numbers) {  $scope.numbers = numbers;  console.log('AppCtrl: ' + numbers);  },  function(errorReason) {  $scope.error = true;  $scope.msg = "Error while getting data.";  console.log('AppCtrl: ' + 'error!');  });  }  }]);  return myControllers;  }());  myApp.factory('myService', ["$http", "$q", function ($http, $q) {  var myService = {  getNumbers: function () {  var deferred = $q.defer();  // Could also be written as  // $http.get("http://ssdprogram.ca/testJson.php")  $http({  method: 'GET',  url: 'http://ssdprogram.ca/testJson.php'  }).success(function (data) {  console.log("myService success: " + data);  deferred.resolve(data);  }).error(function () {  console.log("myService fail: Rejected!");  deferred.reject("Rejected!"); // Return rejection.  });  return deferred.promise; // Promise to return return  // resource when available.  }  };  return myService;  }]); |

**js/test.js**

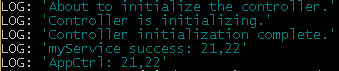
|  |
| --- |
| describe('Testing AppCtrl: ', function() {  "use strict";  var myService, $httpBackend, ctrl, scope;  beforeEach(function() {  module('myApp');  inject(function (\_myService\_, $controller, $rootScope,  \_$httpBackend\_) {  // Pretend we are loading controller for 1st time  // so we pass it a child of the $rootScope.  scope = $rootScope.$new();  // Assign the controller.  ctrl = $controller('AppCtrl', {  $scope: scope // Assign child of rootScope.  });  myService = \_myService\_;  $httpBackend = \_$httpBackend\_;  });  }  );  it('AppCtrl properly invokes myService', function () {  // Send request and get mock response.  $httpBackend .expectGET('http://ssdprogram.ca/testJson.php')  .respond(200, ["21", "22"]);  expect(scope).toBeDefined();  expect(scope.getData).toBeDefined();  scope.getData();  $httpBackend.flush();  expect(scope.numbers[0]).toBe("21");  expect(scope.error).toBeFalsy();  });  }); |

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <body ng-app='myApp'>  <div ng-controller="AppCtrl">  <h1 ng-bind="title"></h1>  <p ng-show="**{{**msg != ''**}}**" ng-bind="msg"></p>  <ul ng-repeat="number in numbers">  <li ng-bind="number"></li>  </ul>  <button ng-click="getData()">Get Data</button>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">  </script>  <script src="js/app.js"></script>  </body>  </html> |

Exercise

🖍 Using Example 4, insert log statements into your *beforeEach()* block and controller so the output becomes the following:



a) Where in the test.js file does this instruction need to be placed?

console.log('About to initialize the controller.');

|  |
| --- |
| Before the block of code below.  ctrl = $controller('AppCtrl', {  $scope: scope // Assign child of rootScope. }); |

b) Where in the test.js file does this instruction need to be placed?

console.log('Controller initialization complete.');

|  |
| --- |
| After the block of code below.  ctrl = $controller('AppCtrl', {  $scope: scope // Assign child of rootScope. }); |

c) Where in the app.js file does this instruction need to be placed?

console.log('Controller is initializing.');

|  |
| --- |
| Before  $scope.title = "Angular JS!"; |

Exercise

🖍 Add an assertion to ensure that the title within the controller is assigned the value “Angular JS!” Show your assertion here and ensure it works.

|  |
| --- |
| Put the following line of code after $httpBackend.flush()  expect(scope.title).toBe("Angular JS!"); |

## Testing Directives

When testing directives, the test suite needs access to the $compile service which can generate HTML from the directive as well as the scope.

|  |
| --- |
| describe('Directive unit test: ', function () {  var $compile, $scope;  beforeEach(module('myApp')); // Load the module.  // Inject and store references to $rootScope and $compile.  beforeEach(inject(function (\_$compile\_, \_$rootScope\_) {  $compile = \_$compile\_;  $scope = \_$rootScope\_.$new();  }));  it('Replaces the element with the appropriate content', function () {  var element = $compile("<my-header></my-header>")($scope); // Generate HTML.  expect(element.html()).toContain("Google - California!"); // Check HTML.  });  }); |

This test would be useful for the following directive declaration:

|  |
| --- |
| myDirectives.directive('myHeader', function () {  return {  restrict: 'E',  replace: true,  template: '<h1>Google - California!</h1>'// Inline tag declarations.  }}); |

Example : Testing Inline Directives

🞑 This example shows how to test a directive which declares the tags as a string which is assigned to the template option of the directive. This is the easiest type of directive to test since it is does not need to make a trip to the server to retrieve content from a separate view file.

**js/app.js**

|  |
| --- |
| // Application module that injects routing, controller, and directive dependencies.  var myApp = angular.module('myApp', ['myControllers', 'myDirectives']);  // Controller - dispatches inputs and outputs.  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', function ($scope) {  $scope.title = "AngularJS Tutorial";  }]);  return myControllers;  }());  // Directives  var myDirectives = (function () {  var myDirectives = angular.module('myDirectives', []);  myDirectives.directive('myHeader', function () {  return {  restrict: 'E',  replace: true,  // Template allows inline tag declarations.  template: '<h1>Google - California!</h1>'  }  });  return myDirectives;  }()); |

**js/test.js**

|  |
| --- |
| describe('Directive unit test: ', function () {  var $compile, $scope;  // Load the myApp module.  beforeEach(module('myApp'));  // Inject and store references to $rootScope and $compile.  beforeEach(inject(function (\_$compile\_, \_$rootScope\_) {  $compile = \_$compile\_;  $scope = \_$rootScope\_.$new();  }));  it('Replaces the element with the appropriate content', function () {  // Compile HTML containing the directive.  var element = $compile("<my-header></my-header>")($scope);  // Ensure the compiled element contains the templated content.  expect(element.html()).toContain("Google - California!");  });  }); |

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  </head>  <body ng-app='myApp'>  <div ng-controller="AppCtrl" ng-model="title">  <h1>**{{**title**}}**</h1>  <a href="#home">Home</a><br /><br />  <my-header></my-header>  </div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">  </script>  <script src="js/app.js"></script>  </body>  </html> |

## Testing a Directive with a Template URL

To test directives with a TemplateURL you must use the $httpBackend service since AngularJS searches the server for a partial view. A template is provided as a mock response. In this sample note how the *expectGET()* function returns a template.

|  |
| --- |
| describe('Unit testing directive using templateURL', function () {  var $compile, scope, $httpBackend, template;  beforeEach(module('myApp'));  beforeEach(inject(function (\_$compile\_, \_$rootScope\_, \_$httpBackend\_) {  $compile = \_$compile\_; // Need to create HTML drom directive.  scope = \_$rootScope\_.$new(); // Create child of $rootScope.  $httpBackend = \_$httpBackend\_; // Needed to mock templateURL response.  template = '<ul class="rating">' // Create a mock template.  + '<li ng-repeat="star in stars" class="filled">&#9733;</li>'  + '</ul>';  // Return mock template whenever 'views/stars.html' is called.  $httpBackend.expectGET('views/stars.html').respond(template);  }));  it('Replaces the element with appropriate content', function () {  var starTag = '<div my-stars rating="3"></div>'; // Create tag with directive.  var element = $compile(starTag)(scope); // Compile to HTML.  $httpBackend.flush(); // Get mock response.  var bullet = '<li ng-repeat="star in stars" class="filled ng-scope">★</li>';  expect(element.html()).toContain(bullet); // Check for bullet tag.  var totalStars = element.children('li').length; // Check star count.  expect(totalStars).toEqual(3);  });  }); |

Example : Testing Directives with a TemplateURL

🞑 This example demonstrates how to mock the templateURL with a mock $httpBackend response. Here is the initial directive which references the HTML with a templateUrl.

**js/app.js**

|  |
| --- |
| // Application module that injects routing, controller, and directive dependencies.  var myApp = angular.module('myApp', ['myControllers', 'myDirectives']);  // Directive - Modifies HTML behaviour.  var myDirectives = (function () {  var myDirectives = angular.module('myDirectives', []);  // directive() is a factory method to create directives.  myDirectives.directive('myStars', function () {  return {  restrict: 'A',  replace: true,  scope: { rating: '=' },  link: function ($scope, elem, attrs, ctrl) {  $scope.stars = [];  for (var i = 0; i < $scope.rating; i++) {  $scope.stars.push({}); // push empty objects onto an array  }  },  templateUrl: function (element, attr) { return 'views/stars.html' },  }  });  return myDirectives;  }());  // Controller - dispatches inputs and outputs.  var myControllers = (function () {  var myControllers = angular.module('myControllers', []);  // Controllers are defined by the controller function.  myControllers.controller('AppCtrl', ['$scope', function ($scope) {  $scope.title = "AngularJS Tutorial";  }]);  return myControllers;  }()); |

To test a directive that uses a templateUrl, a reference to the $compile service is needed to generate the HTML from the mock response and a reference to $httpBackend is also required. The directive is located in the application module scope.

**js/test.js**

|  |
| --- |
| describe('Unit testing directive using templateURL', function () {  var $compile, scope, $httpBackend, template;  beforeEach(module('myApp'));  beforeEach(inject(function (\_$compile\_, \_$rootScope\_, \_$httpBackend\_) {  $compile = \_$compile\_; // Need to create HTML drom directive.  scope = \_$rootScope\_.$new(); // Create child of $rootScope.  $httpBackend = \_$httpBackend\_; // Needed to mock templateURL response.  template = '<ul class="rating">' // Create a mock template.  + '<li ng-repeat="star in stars" class="filled">&#9733;</li>'  + '</ul>';  // Return mock template whenever 'views/stars.html' is called.  $httpBackend.expectGET('views/stars.html').respond(template);  }));  it('Replaces the element with appropriate content', function () {  var starTag = '<div my-stars rating="3"></div>'; // Create tag with directive.  var element = $compile(starTag)(scope); // Compile to HTML.  $httpBackend.flush(); // Get mock response.  var bullet = '<li ng-repeat="star in stars" class="filled ng-scope">★</li>';  expect(element.html()).toContain(bullet); // Check for bullet tag.  var totalStars = element.children('li').length; // Check star count.  expect(totalStars).toEqual(3);  });  }); |

**views/stars.html**

|  |
| --- |
| <ul class="rating">  <li ng-repeat="star in stars" class="filled">  &#9733;  </li>  </ul> |

**rating.css**

|  |
| --- |
| .rating{  margin: 0;  padding: 0;  color:#a9a9a9;  }  ul.rating {  display: inline-block;  }  .rating li {  list-style-type: none;  display: inline-block;  padding: 1px;  text-align: center;  font-weight: bold;  }  .filled {  color: #00ff21;  } |

**index.html**

|  |
| --- |
| <!doctype html>  <html>  <head>  <title>Starting Angular</title>  <link href="rating.css" rel="stylesheet" />  </head>  <body ng-app='myApp' ng-controller="AppCtrl">  <h3 ng-bind="title"></h3>  <div my-stars rating="5"></div>  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">  </script>  <script src="js/app.js"></script>  </body>  </html> |

Exercise

🖍 Using the code from Example 6, modify the directive in app.js to log the star count as each one is added. Your log data should resemble the following:



Show your modified directive here:

|  |
| --- |
| // Directive - Modifies HTML behaviour. **var** myDirectives = (**function** () {  **var** myDirectives = angular.module('myDirectives', []);   // directive() is a factory method to create directives.  myDirectives.directive('myStars', **function** () {  **return** {  restrict: 'A',  replace: **true**,  scope: { rating: '=' },  link: **function** ($scope, elem, attrs, ctrl) {  $scope.stars = [];  **for** (**var** i = 0; i < $scope.rating; i++) {  $scope.stars.push({}); // push empty objects onto an array  console.log('adding star : ' + i);  }  },  templateUrl: **function** (element, attr) { **return** 'views/stars.html' },  }  });  **return** myDirectives; }()); |

Exercise

🖍 Using the code from Example 6, test a new directive which has no stars. Verify that no stars appear when the directive is referenced. Show your new test.js file here:

|  |
| --- |
| describe('Unit testing directive using templateURL', **function** () {  **var** $compile, scope, $httpBackend, template;  beforeEach(module('myApp'));   beforeEach(inject(**function** (\_$compile\_, \_$rootScope\_, \_$httpBackend\_) {  $compile = \_$compile\_; // Need to create HTML drom directive.  scope = \_$rootScope\_.$new(); // Create child of $rootScope.  $httpBackend = \_$httpBackend\_; // Needed to mock templateURL response.  template = '<ul class="rating">' // Create a mock template.  + '<li ng-repeat="star in stars" class="filled">&#9733;</li>'  + '</ul>';   // Return mock template whenever 'views/stars.html' is called.  $httpBackend.expectGET('views/stars.html').respond(template);  }));   it('Replaces the element with appropriate content', **function** () {  **var** starTag = '<div my-stars rating="0"></div>'; // Create tag with directive.  **var** element = $compile(starTag)(scope); // Compile to HTML.  $httpBackend.flush(); // Get mock response.  **var** bullet = '<li ng-repeat="star in stars" class="filled ng-scope">★</li>';  expect(element.html()).not.toContain(bullet); // Check for bullet tag.  **var** totalStars = element.children('li').length; // Check star count.  expect(totalStars).toEqual(0);  }); }); |

Exercise

🖍 Create a new directive which uses a templateURL to display a radio button list that looks like the following with the ‘Do Not Mail.’ option selected.



Test your option list using an $httpBackend mock response to ensure that ‘Do Not Mail’ is checked.

Show your app.js file here:

|  |
| --- |
| **var** app = angular.module('myApp', []);  app.controller('MainCtrl', **function**($scope) {   $scope.mailOptions = { label: "Mail Options", required:**true**, valueList: [{ text: "Do Not Mail", value: "No" },{text:"Mail", value:"yes"}] }; });  app.directive("csRadioField", **function** () {   **return** {  scope: { options: '=', ngModel: '=' },  required: ['ngModel', '^form'],  restrict: 'E',  templateUrl: **function** (element, attr) { **return** 'views/mailOption.html' },  }; }); |

Show your test.js file here:

|  |
| --- |
| describe('Unit testing directive using templateURL', **function** () {  **var** $compile, scope, $httpBackend, template;  beforeEach(module('myApp'));   beforeEach(inject(**function** (\_$compile\_, \_$rootScope\_, \_$httpBackend\_) {  $compile = \_$compile\_; // Need to create HTML drom directive.  scope = \_$rootScope\_.$new(); // Create child of $rootScope.  $httpBackend = \_$httpBackend\_; // Needed to mock templateURL response.   **var** template = **function** () {  **return** '<div ng-form="myform">' +  '<div class="control-group" class="{{options.class}}">' +  '<div class="control-label">{{options.label || "Radio"}} {{ options.required ? "\*" : ""}} </div>' +  '<div class="controls">' +  '<div class="radio" ng-repeat="(key, option) in options.valueList">' +  '<label> <input type="radio" name="myfield" ng-value="option.value" ng-model="$parent.ngModel" ng-required="options.required" />{{option.text}} </label>' +  '</div>' +  '<div class="field-validation-error" data-ng-show="myform.myfield.$invalid && myform.myfield.$dirty"> ' +  '<div data-ng-show="myform.myfield.$error.required">{{options.label}} is required!!!</div>' +  '</div>' +  '</div>' +  '</div>' +  '</div>';  };   // Return mock template whenever 'views/stars.html' is called.  $httpBackend.expectGET('views/mailOption.html').respond(template);  }));   it('Replaces the element with appropriate content', **function** () {  **var** mailTag = '<cs-radio-field options="mailOptions" ng-model="mailValue"></cs-radio-field>';  **var** element = $compile(mailTag)(scope); // Compile to HTML.  $httpBackend.flush(); // Get mock response.   **var** mailStatus  expect(mailStatus).toEqual("Do Not Mail");  }); }); |