**Karma Server Setup(Karma JavaScript test runner Setup)**

**References:**

[**http://monicalent.com/blog/2015/02/11/karma-tests-angular-js-require-j/**](http://monicalent.com/blog/2015/02/11/karma-tests-angular-js-require-j/)

[**http://thecodebarbarian.com/2015/06/12/testing-angularjs-directives**](http://thecodebarbarian.com/2015/06/12/testing-angularjs-directives)

[**https://salesforce.stackexchange.com/questions/34918/testing-angularjs-on-visualforce**](https://salesforce.stackexchange.com/questions/34918/testing-angularjs-on-visualforce)

[**https://developer.salesforce.com/blogs/developer-relations/2014/07/building-single-page-app-angularjs-salesforce-rest-api.html**](https://developer.salesforce.com/blogs/developer-relations/2014/07/building-single-page-app-angularjs-salesforce-rest-api.html)

**--global serve-https@2.x**

**Install Node.js**

<http://www.guru99.com/download-install-node-js.html>

**Install Karma**

npm install karma

**OR**

**Install Karma and Its and its variable components or javascript frameworks**

1. npm install karma karma-jasmine karma-phantomjs-launcher karma-re

quirejs --save-dev

2. npm install -g karma-cli

This tool executes the Karma commands for the project in the current directory.

**Create Karma config file (karma.conf.js)**

You can now navigate to **Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\test** and run:

karma init

**Now it will ask some questions:**

**1.** Which Testing Framework?? -> jasmine

**2.** Want to use Require.js?? -> yes

**3.** Want to capture any browser automatically?? -> It will automatically run the browser when you start karma using “**karma start**” -> You can specify multiple browsers from poosible options list. For Now -> chrome

**4.** Location of source and test files?? -> for eg.: C:\Users\Pooja\Desktop\Blackpurl\Angular-Testing\Sample\_Example

You can specify mutliple paths like: C:\Users\Pooja\Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\test

C:\Users\Pooja\Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\scripts

**5.** Want to exclude any of the files from the files included by pattern specified above?? -> no

**6.** Want to generate bootstrap file for RequireJs?? -> yes

This will create **test-main.js file** that configures the requirejs same as main.js config file and start the test. Here main.js is our app’s real Requirejs configuration, which we want to mirror as closely as possible in test-main.js, so that all our define statements that work in our app also work in our tests.

**7.** Want to watch all the files and run the tests on change?? -> Yes

If there will change in the test files than it will automatically detect them and run the test.?????????????????????????

Now the **test-main.js** and **karma.conf.js** files are created in **C:\Users\Pooja\app\test\** folder. **karma.conf.js** file is created according to the answers of above questions.

Open these files and you can change these files according to your requirement.

In karma.conf.js **autoWatch is set to** **true**, so that any changes to test-main.js are picked up automatically by karma test runner. But if you make changes to karma.conf.js, you will have to kill Karma and start the process again for it to consider these changes.

Here **C:\Users\Pooja is a default path taken by karma.**

So if you don’t specify this string in your complete file path then also it will be correct.

For Eg: C:\Users\Pooja\Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\scripts\\*\*\\*.js

**Can be wriiten as**

Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\scripts\\*\*\\*.js

**Start Karma Test runner**

1. Karma start

It will search for the karma.conf.js file in the **default**

**path(C:\Users\Pooja)**

OR

2. Karma start file\_Path

file\_Path : Eg:

(Desktop\Blackpurl\Angular-Testing\Sample

\_Example\app\test\karma.conf.js)

It will search for the karma.conf.js file in the path that you have Specified in **karma start** command.

Error: Cannot find module **'require.js'** during loading **"karma-requirejs"**

plugin

1. npm install --global --verbose requirejs

2. npm install requirejs@2.1.0 -> **it is required**

3. Then again run Karma start

Error: Cannot find module **'jasmine-core'**

1. npm install jasmine-core --save-dev

2. Then again run Karma start

**karma start Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\test\karma.conf.js**

It will open the browser window which you have answered in the above questions, if you don’t have specified any browser name then it will only start the server then you have to manually type this url <http://localhost:9876/?id=30455194> in any browser you want.

**basePath in karma.conf.js**

**FilePath for karma.conf.js** ->

(Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\test\karma.conf.js)

**basePath: ‘ ‘** -> It will take the path whcih is specifed for karma.conf.js during **karma start**

and then search all the files (specified in karma.conf.js) inside that

directory. (Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\test\)

**basePath: ‘/’** -> It will take the path **C:\** only and then search all the files (specified in

karma.conf.js) inside this directory.

**basePath: ‘../’** -> It will then search all the files (specified in karma.conf.js) outside the

current directory of karma.conf.js.

(Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\)

**basePath: ‘../..’ (Used in Our eg-app)** -> It will then search all the files (specified in

karma.conf.js) outside the current and parent of current directory of

karma.conf.js.

(Desktop\Blackpurl\Angular-Testing\Sample\_Example\)

**According to above basePath Changed in karma.conf.js file:**

// list of files / patterns to load in the browser

files: [

'test-main.js',

{pattern:

'C:\Users\Pooja\Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\scripts', included: false},

{pattern:

'C:\Users\Pooja\Desktop\Blackpurl\Angular-Testing\Sample\_Example\app\test', included: false}

],

**Is changed to**

// list of files / patterns to load in the browser

files: [

'app/test/test-main.js',

{pattern: 'app/scripts/\*\*/\*.js', included: false},

{pattern: 'app/test/spec/\*\*/\*.js', included: false},

{ pattern: 'app/vendor/\*\*/\*.js', included: false }

]

**{ pattern: 'app/vendor/\*\*/\*.js', included: false }**

Without this, when we setup test-main.js, we will get 404 WARN’s when trying to load Angular. As a rule, any file that you want to test, or that is a dependency of a file you want to test, must be picked up by Karma by matching an entry in “files”.

**Changes in test-main.js**

The default karma+requirejs setup is wrong. Delete or comment this line

var normalizedTestModule = file.replace(/^\/base\/|\.js$/g, '')

Becasue it appends the .js extension in a file which is not needed for require js; because Karma actually make an HTTP request for file (my\_filter\_spec) instead of file.js (my\_filter\_spec.js).

**basePath in karma.conf.js and baseUrl in test-main.js**

**basePath: ‘../..’** -> (Desktop\Blackpurl\Angular-Testing\Sample\_Example\)

**baseUrl: ‘/base’** -> It takes the exact same path which is defined by basePath.

(Desktop\Blackpurl\Angular-Testing\Sample\_Example\)

**baseUrl**: 'base/app/scripts': It will give error **“No timestamp”** for file. Adding **‘/’** at the start

of baseUrl will solve the problem (‘**/base/app/scripts’)**.

**baseUrl: ‘/base/app/scripts’ (Used in Our eg-app)** ->

(Desktop/Blackpurl/Angular-Testing/Sample\_Example/app/scripts)

All the file path in **define statement dependency** will be searched inside this baseUrl path.

But if you want to specify module id for each file dependency using **paths** in test-main.js then you have to specify the full path including base url (/base/app/scripts/subdirectory\_path/fileName) also and then use that module id in **define statement dependency.**

Eg:

define(['path/to/somewhere'], **function**() { });

It will load the file as [*http://localhost:9876/base/app/scripts/path/to/somewhere*](http://localhost:9876/base/app/scripts/path/to/somewhere)

**Testing Angular Files(Controllers/Directives/Filters)**

If you want to test something like filter, controller or directive, then each file (AMD module) must return the function you want to test, so we can inject it later.

**angular-mocks**

$ npm install --save angular  
$ npm install --save-dev angular-mocks

This will allow us to register our filter, controller or direcives etc. before we test them.

Error: cleanData is not a function at angular-mock.js

Go to the angular-mocks.js file and replace:

angular.element.cleanData(cleanUpNodes);

With

if (angular.element.cleanData) angular.element.cleanData(cleanUpNodes);

**Functions Used**

**describe()**

A **test suite** is created by calling describe() method with a string as its name and a function as abody to execute. A **test suite**, less commonly known as a 'validation suite', is a collection of [test cases](https://en.wikipedia.org/wiki/Test_case) that are intended to be used to test a software program to show that it has some specified set of behaviours. Occasionally, test suites are used to group similar test cases together.

**it()**

A **spec** is created by calling this method with the same parameter as describe and the name which you want to give for your test case. **spec** means unit test cases.

**expect(value\_or\_function\_to\_be\_checked)**

An expection is created by calling this method and with the value or function to be checked, then chained with different matcher functions for verifications.

**beforeEach()**

**afterEach()**

**beforeAll()**

**afterAll()**

These above 4 methods are used for our custom setup like if we want to set variable value or get any value before test case start then we can write that code inside any of these above functions.

**expect(value\_Or\_function\_to\_be\_checked).toBe(expectedResult)**

**expect(value\_Or\_function\_to\_be\_checked).toEqual(expectedResult)**

These above methods will return success if result returned by function is **same** as expected result otherwise it will throw error and return failure.

**expect(value\_Or\_function\_to\_be\_checked).not.toBe(expectedResult)**

**expect(value\_Or\_function\_to\_be\_checked).not.toEqual(expectedResult)**

These above methods will return success if result returned by function is **not same** as expected result otherwise it will throw error and return failure.

If expected result mismatches with actual result then it throws error.

**Benefits**

With these above functions we can test our functionality without using any DOM.

If we change the functionality of our filter, controller, directive or service in the future then we can reduce the risk of failing any functionality using these test cases.

Unit tests are distinct from end-to-end (E2E) tests. End-to-end testing involves using a browser automation tool like Selenium to interact with your application's interface and verify high level details like URLs, text, and HTML attributes. End-to-end tests are a useful insurance policy and sanity-check. If something is badly broken in your code, a failed end-to-end test will only alert you to the need for deep investigation.

Unit tests span a much broader range of functionality. If approached and written properly, unit tests can be a major enhancer of productivity. End-to-end tests (or worse, manually verifying new behavior) only tell you that something is wrong. Unit tests give you more direct information about the nature of failures. There's no need to adhere to a strict test-driven development (TDD) style. However, **keeping your test suite relatively in sync with new code is always a good idea.**

Tests are the best way to prevent software defects. When a small piece of code is updated and the developer manually opens their browser or Postman to verify that it still works. This approach (manual QA) is begging for a disaster.

The reason we test our code is to verify that it behaves as we expect it to. As a result of this process you'll find you have better feature documentation for yourself and other developers as well as a design aid for your APIs.

Karma is a direct product of the [AngularJS team](https://googletesting.blogspot.com/2012/11/testacular-spectacular-test-runner-for.html) from struggling to test their own framework features with existing tools.

This includes the option to test your code on various browsers and devices such as phones, tablets etc.

Jasmine is also dependency free and doesn’t require a DOM.