Software Development Tools Needed

Visual Studio 2015 Pro or Enterprise

* Include Office/SharePoint Application option
* Extensions
  + Web Essentials
  + Chutzpah Test Adapter and Menu Context add-on

Node.js

* How to setup proxy server for node.js
  + For the 'proxy' key type 'npm config set proxy=http://xxxx.net:8080'
  + For the 'https-proxy' key type 'npm config set https-proxy=http://xxx.net:8443'
* Karma runs on [Node.js](http://nodejs.org/) and is available as an [NPM](https://npmjs.org/package/karma) package.
* Installing Node.js
* Npm –v
* Node -v
* On Mac or Linux we recommend using [NVM](https://github.com/creationix/nvm). On Windows, download Node.js from [the official site](https://nodejs.org/).
* **Note:**
* Karma currently works on Node.js **0.10**, **0.12.x**, **4.x**, and **5.x**. See [FAQ](http://karma-runner.github.io/0.13/intro/faq.html) for more info.
* Installing Karma and plugins
* The recommended approach is to install Karma (and all the plugins your project needs) locally in the project's directory.
* *# Install Karma:*
* $ npm install karma --save-dev
* *# Install plugins that your project needs:*
* $ npm install karma-jasmine karma-chrome-launcher --save-dev
* This will install karma, karma-jasmine and karma-chrome-launcher packages into node\_modules in your current working directory and also save these as devDependencies in package.json, so that any other developer working on the project will only have to do npm install in order to get all these dependencies installed.
* *# Run Karma:*
* $ ./node\_modules/karma/bin/karma start
* Commandline Interface
* Typing ./node\_modules/karma/bin/karma start sucks and so you might find it useful to install karma-cli globally. You will need to do this if you want to run Karma on Windows from the command line.
* $ npm install -g karma-cli
* Then, you can run Karma simply by karma from anywhere and it will always run the local version.

Npm Global Installs ( npm –g install <package> )

* Karma
* Karma-cli
* Karma-jasmine
* Jasmine
* Bower
* Eslint
* Gulp

If some packages are blocked from install, try on the wifi network instead.

# Unit Testing Angular Files using Visual Studio

1. Download and install Node.js
   1. <http://www.nodejs.org>
2. Download and install the Chutzpah Test Adapter for Visual Studio
   1. <https://visualstudiogallery.msdn.microsoft.com/f8741f04-bae4-4900-81c7-7c9bfb9ed1fe>
3. Download and install the Chutzpah Test Runner Context Menu for Visual Studio
   1. <https://visualstudiogallery.msdn.microsoft.com/71a4e9bd-f660-448f-bd92-f5a65d39b7f0>
4. Create a new Unit Test Project within the same solution as the source project.
5. Using the NuGet Package Manager:
   1. Install Jasmine, behavior-driven development framework for testing JavaScript.
6. Create an “ng-tests” folder.
7. Inside setup the same folder structure as it is in your source app.
8. Create test files for the source files you’d like to test.
   1. Be sure to add the suffix “.spec” to the end of the file name (i.e. main.controller.spec.js)
   2. And at this point only a few files are needed to complete setting up the test environment.
   3. Even a blank test file will do for now.
9. Now open the file and from the source project drag and drop the reference files at the top of the test file. It should create a “reference path” to the source projects file.
10. Now code your tests.
11. To run tests, simply right click on the “ng-tests” folder:
    1. Select “Run JS Test”. Results will be shown in the Test Explorer window.
    2. Select “Run Chutzpah With” and “Debugger” to launch the Jasmine static test result html page and Visual Studio in debug mode. Once the page is up, click on a test to re-run. Break-points within a test function will be triggered if related test was executed.

**Testing Design**

Contents

[Unit Testing Setup and Execution 3](#_Toc447550325)

[Unit Test Expectations 3](#_Toc447550326)

[Unit Test Work Flow 3](#_Toc447550327)

[Unit Test Examples 3](#_Toc447550328)

[Controller 3](#_Toc447550329)

[Service 5](#_Toc447550330)

[Directive 6](#_Toc447550331)

**CODING STANDARDS**

I’ve started reviewing some of the code and noticing “a lot” of things going on that needs correcting:

-          Values initiated outside the init() function

-          Too many functions, empty ones at that

o   Anonymous callback functions should be used resolve service calls

o   Then().Catch().Finally() should be used

-          Too many public properties and/or function (not used by the view)

o   Just extra values angular has to watch for during a digest cycle.

o   Makes it unclear on what to test for (if they are not in use)

-          Eslint errors for days

o   Get these fixed before check-in

-          No unit tests written for most files

o   Needs to be written for the upcoming refactoring of this code.

-          Code anti-patterns

o   Dead code

o   Hard coding string values

o   For each iterations over list for conditioning

  Use the underscore library, it’s much more efficient and makes it easier to accomplish functional querying

-          Use of the $injection service

o   Looks pretty to have less params

o   But you’ll never know what inside without going inside

o   Makes it difficult to test because you have to hunt for wired services

o   Just inject the services, factories, etc as arguments

# Unit Testing Setup and Execution

Be sure to setup and run your test environment according to the [Unit Testing Angular Files using Visual Studio](https://share.ey.net/sites/leases/_layouts/15/WopiFrame.aspx?sourcedoc=/sites/leases/Lists/Shared%20Documents/Development/Unit%20Testing%20Angular%20Files%20using%20Visual%20Studio.docx&action=default) document on the Leases SharePoint site.

# Unit Test Expectations

Quality tests over quantity is expected to build success software. All meaningful, feature specific function or methods should have well written tests to support healthy code.

All controllers, services or factories, and directives should have unit tests written against them.

Core features of each should have multiple behavioral driven tests that can support scenarios where variables or outcomes can change results.

# Unit Test Work Flow

The unit test pattern Behavior Driven Development (BDD) is the foundation of what the Jasmine library was built on. It make testing code human legible and easy to read and understand.

User stories and the Acceptance Criteria there within helps setup unit test initial so that the developer can write test in parallel with component development.

# Unit Test Examples

Below are example screen shot of unit from a controller, service and directive.

## Controller

describe('ContractsController', function () {

var service;

var vm;

var scope;

var mockContract;

//always init the app module before each call

beforeEach(module('app'));

//init any controllers or services using the inject function

beforeEach(inject(function (\_$controller\_, \_$rootScope\_, ContractServiceLocal) {

scope = \_$rootScope\_.$new();

service = ContractServiceLocal;

vm = \_$controller\_('ContractsController', { service: service });

//use Jasmines spyOn service to call, execute and await promise callbacks

spyOn(service, 'getContracts').and.callThrough();

//call a digest cycle afterwords to alert angular of promise called

scope.$root.$digest();

}));

it('should set the loading variable to false after initialization', function () {

expect(vm.loading).toBe(false);

});

it('should populate contracts in initialization phase', function () {

//which was called in the beforeEach method up above

expect(vm.contracts.length).toBeGreaterThan(0);

});

it('should setup for new contract entry', function () {

//vm represents the controller

vm.newContract();

expect(vm.contract.id).toBe(0);

expect(vm.showForm).toBe(true);

});

it('should save new contract entry and increment list by 1', function () {

spyOn(service, 'addContract').and.callThrough();

var initCount = vm.contracts.length;

vm.contract = {

id: 0,

companyName: 'BYRDSONG',

startDate: '1/1/2016',

endDate: '9/1/2016',

productName: 'BYRDSONG TOY1',

unitCost: 59.99,

quantity: 235

};

vm.saveContract();

scope.$root.$digest();

mockContract = \_.findWhere(vm.contracts, { 'companyName': 'BYRDSONG' });

expect(mockContract).toBeDefined();

expect(vm.contracts.length).toBeGreaterThan(initCount);

expect(vm.showForm).toBe(false);

});

it('should setup for modification of existing contract', function () {

var id = 1;

vm.editContract(id);

expect(vm.contract).toBeDefined();

expect(vm.showForm).toBe(true);

});

it('should save modified contract', function () {

spyOn(service, 'updateContract').and.callThrough();

var id = 1;

vm.editContract(id);

vm.contract.companyName = 'XX';

vm.saveContract();

scope.$root.$digest();

mockContract = \_.findWhere(vm.contracts, { 'companyName': 'xx' });

expect(mockContract).toBeDefined();

expect(vm.showForm).toBe(false);

});

it('should delete contract from list', function () {

spyOn(service, 'deleteContract').and.callThrough();

var initCount = vm.contracts.length;

var id = 1;

vm.deleteContract(id);

scope.$root.$digest();

mockContract = \_.findWhere(vm.contracts, { 'id': id });

expect(mockContract).not.toBeDefined();

expect(vm.contracts.length).toBeLessThan(initCount);

});

});

## Service

describe('contracts local database service', function () {

var service;

var scope;

var contracts = [];

var mockContract = {};

beforeEach(module('app'));

beforeEach(inject(function (\_$rootScope\_, ContractServiceLocal) {

//init new scope to help call a digest cycle for the service call

scope = \_$rootScope\_.$new();

service = ContractServiceLocal;

}));

it('should return contract list', function () {

spyOn(service, 'getContracts').and.callThrough();

service.getContracts()

.then(function (data) {

contracts = data;

});

scope.$root.$digest();

expect(contracts.length).toBe(3);

});

it('should return contract given the id', function () {

spyOn(service, 'getContract').and.callThrough();

var id = 1;

service.getContract(id)

.then(function (data) {

mockContract = data;

});

scope.$root.$digest();

expect(mockContract.id).toBe(1);

});

it('should add a new contract to the array', function () {

spyOn(service, 'addContract').and.callThrough();

mockContract = new Contract(4, 'xx', new Date('1/1/2016'), new Date('10/1/2016'), 'Widgets', 100, 2934);

service.addContract(mockContract)

.then(function (data) {

contracts = data;

});

scope.$root.$digest();

expect(contracts.length).toEqual(4);

});

it('should update existing contract (id:3) with $200 as unit cost', function () {

spyOn(service, 'getContract').and.callThrough();

var id = 3;

service.getContract(id)

.then(function (data) {

mockContract = data;

});

scope.$root.$digest();

mockContract.unitCost = 200;

spyOn(service, 'updateContract').and.callThrough();

service.updateContract(id, mockContract)

.then(function (data) {

contracts = data;

});

scope.$root.$digest();

var updatedContract = \_.findWhere(contracts, { 'id': id });

expect(updatedContract.unitCost).toBe(200);

});

it('should delete existing contract given contract id', function () {

spyOn(service, 'deleteContract').and.callThrough();

var id = 3;

service.deleteContract(id)

.then(function (data) {

contracts = data;

});

scope.$root.$digest();

mockContract = \_.findWhere(contracts, { 'id': id });

expect(mockContract).not.toBeDefined();

});

});

## 

## Directive

Example.Directive.js

(function (app) {

'use strict';

app.directive('exampleDir', ExampleDirective);

function ExampleDirective() {

return {

restrict: 'E',

replace: true,

template: '<span>{{ username }}</span>',

scope: {

username: '@'

}

};

}

})(angular.module('app'));

Example.Directive.spec.js

describe('Unit testing example-directive', function () {

var $compile,

$rootScope;

beforeEach(module('app'));

beforeEach(inject(function (\_$compile\_, \_$rootScope\_) {

$compile = \_$compile\_;

$rootScope = \_$rootScope\_;

}));

it('Sanity check', function () {

var element = $compile('<span>Apple</span>')($rootScope);

expect(element.html()).toContain('Apple');

});

it('Sets element with attribute "username" value', function () {

var element = $compile('<example-dir username="Apple"></example-dir>')($rootScope);

$rootScope.$digest();

expect(element.html()).toContain("Apple");

});

});

## Filter

Uppercase.filter.js

(function (app) {

'use strict';

app.filter('uppercase', UppercaseFilter);

function UppercaseFilter() {

return function (text) {

return ('' + (text || '')).toUpperCase();

}

}

})(angular.module('app'));

Uppercase.filter.spec.js

describe('Unit test for Uppercase filter', function () {

var $filter;

beforeEach(module('app'));

beforeEach(inject(function (\_$filter\_) {

$filter = \_$filter\_;

}));

it('should set value to all upper case', function () {

var uppercase = $filter('uppercase');

expect(uppercase('apple')).toEqual('APPLE');

});

});

<http://andyshora.com/unit-testing-best-practices-angularjs.html>

http://www.bradoncode.com/tutorials/angularjs-unit-testing/