Exercise (Instructions): Unit Testing Angular Applications

Objectives and Outcomes

In this exercise, you will learn about unit testing in your Angular applications. You will learn about Angular mocks, Karma, and Jasmine and learn how to use them to carry out unit testing and e2e testing. At the end of this exercise, you should be able to:

- Configure a Karma configuration file
- Set up unit tests using Jasmine and carry out the unit test automatically

Setting up the Unit Test Environment

- First, set up Jasmine core to be available for use within your project:

```
npm install jasmine-core --save-dev
```

- Then, set up the Karma command line tools globally as follows:

```
npm install karma-cli -g
```

Remember to use sudo if you are in OSX or Linux environments.

- Then set up karma-jasmine plugin to make use of Jasmine with Karma:

```
npm install karma-jasmine --save-dev
```

- In order to set up browser environments to carry out the tests, set up PhantomJS, and Karma launchers for PhantomJS and Chrome as follows:

```
npm install phantomjs karma-phantomjs-launcher karma-chrome-launcher --save-dev
```

You can also set up for Firefox, IE and Safari if you use these browsers.

Setting up Angular Mocks

- You should also install the ngMock module as follows:

```
bower install angular-mocks -S
```

Unit Testing of MenuController

- Next, we will configure Karma to conduct the unit test. First, create a folder in *conFusion* folder, named *test*.
- Move to the *test* folder, and create a file named *karma.conf.js* there. This file will contain the configuration information for the Karma tests. Add the following configuration to the file:

```
// Karma configuration

module.exports = function(config) {
  config.set({

    // base path that will be used to resolve all patterns (eg. files, exclude)
    basePath: '../',

    // frameworks to use
    // available frameworks: https://npmjs.org/browse/keyword/karma-adapter
    frameworks: ['jasmine'],

  })
}
```

- Next, add the list of files to be included, and those to be excluded into the config as follows:

```
    // list of files / patterns to load in the browser
    files: [
      'bower_components/angular/angular.js',
      'bower_components/angular-resource/angular-resource.js',
      'bower_components/angular-ui-router/release/angular-ui-router.js',
      'bower_components/angular-mocks/angular-mocks.js',
      'app/scripts/*.js',
      'test/unit/**/*.js'
    ],

    // list of files to exclude
    exclude: [
    'test/protractor.conf.js', 'test/e2e/*.js'
    ],
```

- The add some configuration information as follows. See the comments for details:

```
    // preprocess matching files before serving them to the browser
    // available preprocessors: https://npmjs.org/browse/keyword/karma-preprocessor
    preprocessors: {
    },

    // test results reporter to use
    // possible values: 'dots', 'progress'
    // available reporters:  https://npmjs.org/browse/keyword/karma-reporter
    reporters: ['progress'],

    // web server port
    port: 9876,

    // enable / disable colors in the output (reporters and logs)
    colors: true,

    // level of logging
    // possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || conf
ig.LOG_INFO || config.LOG_DEBUG
    logLevel: config.LOG_INFO,

    // enable / disable watching file and executing tests whenever any file changes
    autoWatch: true,
```

- Next, let us configure the browsers to use for testing as follows:

```
    // start these browsers
    // available browser launchers: https://npmjs.org/browse/keyword/karma-launcher
     browsers: ['Chrome','PhantomJS', 'PhantomJS_custom'],

    // you can define custom flags
    customLaunchers: {
      'PhantomJS_custom': {
        base: 'PhantomJS',
        options: {
          windowName: 'my-window',
```

```
          settings: {
            webSecurityEnabled: false
          },
        },
        flags: ['--load-images=true'],
        debug: true
      }
    },

    phantomjsLauncher: {
      // Have phantomjs exit if a ResourceError is encountered (useful if karma exits wit
hout killing phantom)
      exitOnResourceError: true
    },

    // Continuous Integration mode
    // if true, Karma captures browsers, runs the tests and exits
    singleRun: false,

    // Concurrency level
    // how many browser should be started simultaneous
    concurrency: Infinity
```

- Save the changes. Now the Karma configuration is ready for launching the test. Next we create a test for MenuController.
- Create a folder named *unit* in the *test* folder. Move to the *unit* folder and then create a folder there named*controllers*. Then move to the *controllers* folder. We will store the tests for the controllers here.
- Create a file named *menucontroller.js* and start the configuration of the test as follows:

```
describe('Controller: MenuController', function () {

  // load the controller's module
  beforeEach(module('confusionApp'));

  var MenuController, scope, $httpBackend;
```

```
});
```

- Now we will inject the mock dependencies as follows:

```
  // Initialize the controller and a mock scope
  beforeEach(inject(function ($controller, _$httpBackend_,  $rootScope, menuFactory) {

          // place here mocked dependencies
      $httpBackend = _$httpBackend_;

      $httpBackend.expectGET("http://localhost:3000/dishes").respond([
        {
      "id": 0,
      "name": "Uthapizza",
      "image": "images/uthapizza.png",
      "category": "mains",
      "label": "Hot",
      "price": "4.99",
      "description": "A",
      "comments":[{}]
      },
      {
      "id": 1,
      "name": "Zucchipakoda",
      "image": "images/zucchipakoda.png",
      "category": "mains",
      "label": "New",
      "price": "4.99",
      "description": "A",
      "comments":[{}]
      }
      ]);

    scope = $rootScope.$new();
    MenuController = $controller('MenuController', {
      $scope: scope, menuFactory: menuFactory
    });
```

```
            $httpBackend.flush();

  }));
```

- Finally, set up all the tests in the file as follows:

```
  it('should have showDetails as false', function () {

    expect(scope.showDetails).toBeFalsy();

  });

  it('should create "dishes" with 2 dishes fetched from xhr', function(){

      expect(scope.showMenu).toBeTruthy();
      expect(scope.dishes).toBeDefined();
      expect(scope.dishes.length).toBe(2);

  });

  it('should have the correct data order in the dishes', function() {

      expect(scope.dishes[0].name).toBe("Uthapizza");
      expect(scope.dishes[1].label).toBe("New");

  });

  it('should change the tab selected based on tab clicked', function(){

      expect(scope.tab).toEqual(1);

      scope.select(3);

      expect(scope.tab).toEqual(3);
      expect(scope.filtText).toEqual('mains');

  });
```

- Save the changes. Now the test is ready to be executed.
- Move back to the *test* folder and then type the following at the prompt to execute the test:

```
    karma start karma.conf.js
```

- All the tests should successfully complete. You can edit some of the test values to cause some of the tests to fail.

Conclusions

In this exercise, you learnt about using Karma and Jasmine to carry out unit tests on Angular application components.