# NG – Unit Testing

|  |
| --- |
| Testing |
| **Unit Testing**   * This is sometimes also called Isolated testing. * It’s the practice of testing small isolated pieces of code. * If your test uses some external resource, like the network or a database, it’s not a unit test.   **Functional Testing**   * This is defined as the testing of the complete functionality of an application. * In practice with web apps, this means interacting with your application as it’s running in a browser just like a user would interact with it in real life, i.e. via clicks on a page. * This is also called End To End or E2E testing.   we are only going to cover how to perform unit testing Jasmine:  * Jasmine is a javascript testing framework that supports a software development practice called Behaviour Driven Development, or BDD for short. It’s a specific flavour of Test Driven Development (TDD). * Jasmine, and BDD in general, attempts to describe tests in a human readable format so that non-technical people can understand what is being tested. However even if you are technical reading tests in BDD format makes it a lot easier to understand what’s going on.   For example if we wanted to test this function:  function helloWorld() {  return 'Hello world!';  }  We would write a jasmine test spec:  **describe**('Hello world', () => {  let expected = "";    **beforeEach**(() => {  expected = "Hello World";  });    **afterEach**(() => {  expected = "";  });    **it**('says hello', () => {  **expect**(**helloWorld**())  .**toEqual**(expected);  });  });   * The describe(string, function) function defines what we call a Test Suite, a collection of individual Test Specs. * The it(string, function) function defines an individual Test Spec, this contains one or more Test Expectations. * The expect(actual) expression is what we call an Expectation. In conjunction with a Matcher it describes an expected piece of behaviour in the application. * The matcher(expected) expression is what we call a Matcher. It does a boolean comparison with the expected value passed in vs. the actual value passed to the expect function, if they are false the spec fails.     **Running Jasmine tests:**  <link *rel*="stylesheet" *href*="jasmine.css">  <script *src*="jasmine.js"></script>  <script *src*="jasmine-html.js"></script>  <script *src*="boot.js"></script>  <script *src*="main.js"></script>  <script *src*="test.js"></script>     * hello world function was in main.js * Our Jasmine Unit Test are in test.js * The order of script tags is important.  Karma:  * Karma is a tool which lets us spawn browsers and run jasmine tests inside of them all from the command line. * The results of the tests are also displayed on the command line. * Karma can also watch your development files for changes and re-run the tests automatically. * Karma lets us run jasmine tests as part of a development tool chain which requires tests to be runnable and results inspectable via the command line.   Disabled and focused tests   |  | | --- | | */\**  *You can disable tests without commenting them our by just pre-pending x to the describe or it functions,*  *\*/*  *// These tests will not be run.*  **xdescribe**('Hello world', () => {  **it**('says hello', () => {  **expect**(**helloWorld**())  .**toEqual**('Hello world!');  });  });  **describe**('Hello world', () => {  *// These tests will not be run.*  **xit**('says hello', () => {  **expect**(**helloWorld**())  .**toEqual**('Hello world!');  });  });  */\**  *Conversely you can also focus on specific tests by pre-pending with f, like so:*  *\*/*  **fdescribe**('Hello world', () => {  **it**('says hello', () => {  **expect**(**helloWorld**())  .**toEqual**('Hello world!');  });  });  **describe**('Hello world', () => {  *// Out of all the tests in all the tests suites and tests specs,*  *// these are the only ones that will be run.*  **fit**('says hello', () => {  **expect**(**helloWorld**())  .**toEqual**('Hello world!');  });  ...  }); |   **Summary**   * Jasmine is a testing framework that supports Behavior Driven Development. We write tests in Test Suites which are composed of one or more Test Specs which themselves are composed of one or more Test Expectations. * We can run Jasmine tests in a browser ourselves by setting up and loading a HTML file, but more commonly we use a command line tool called Karma. Karma handles the process of creating HTML files, opening browsers and running tests and returning the results of those tests to the command line. * If you use the Angular CLI to manage projects it automatically creates stub jasmine spec files for you when generating code. It also handles the Karama configuration, transpilation and bundling of your files so all you need to do in order to run your tests is type the command ng test.   … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |

|  |
| --- |
|  |
| … |