# Angular Unit Testing

* Testing our applications play an important role in reducing bugs and saving time by writing stable applications
* There are multiple ways to test an application - manual tests, automated integration tests, automated unit tests, performance tests and end-to-end(e2e) tests. Each of these techniques test different aspects of the application.
* Unit test is the technique that tests a piece of code in isolation
* Unit tests play an important role in software development. They help in catching the bugs early and ensure quality of the code.

## **Why unit testing is important?**

* Unit testing Analyse the code behaviour (expected and unexpected changes).
* It behaves as a safeguard against breaking changes. One of the best ways to keep your project bug free is through a test suite.
* It also reveals the design mistakes.

## Unit Testing an Angular app with Jasmine and Karma

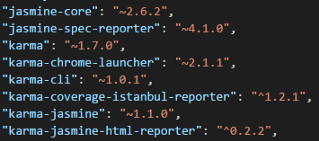
* Angular is built with testing in mind. The modular architecture and the way [Dependency Injection works](https://www.dotnetcurry.com/angularjs/1446/dependency-injection-di-angular), makes any of the Angular code blocks easier to test.
* There are many testing frameworks for Angular App but the most common ones are Mocha and Jasmine.

## Jasmine

* Jasmine comes as the default testing framework with the Angular CLI.
* Jasmine is a popular Javascript testing framework.
* Jasmine is as well as assertion library and mocking library. It provides all the functionality we need to write tests.
* Jasmine is a JavaScript testing framework that supports a software development practice called [Behaviour-Driven Development](https://en.wikipedia.org/wiki/Behavior-driven_development), or BDD for short. It’s a specific flavour of [Test-Driven Development](https://en.wikipedia.org/wiki/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.
* The testing framework functionality gives us functions like describe, it, beforeEach, and afterEach.

## Karma

* Karma is a test runner. It will automatically create a browser instance, run our tests, then gives us the results. The big advantage is that it allows us to test our code in different browsers without any manual change in our part.
* with [Karma](https://karma-runner.github.io/2.0/index.html) we can:
* Test on real platforms (mobile, desktop, tablet)
* Debug code from the browser or console
* Integrates with continuous integration tools such as [Jenkins](https://jenkins.io/)
* Use it with any testing framework not just [Jasmine](https://jasmine.github.io/)
* Karma uses a config file in the root directory of the app called karma.config.ts to define its testing behaviour.
* Which specifies where the test files, which frameworks and plugins to use, and how to configure those plugins.
* When you create the project all the dependencies get installed among them everything you are going to need to create the tests.



For example if we wanted to test this function:

TypeScript

function helloWorld() {

return 'Hello world!';

}

We would write a Jasmine test spec like so:

TypeScript

describe('Hello world', () => { (1)

it('says hello', () => { (2)

expect(helloWorld()) (3)

.toEqual('Hello world!'); (4)

});

});

|  |  |
| --- | --- |
|  | 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. |

## [Built-In Matchers](https://codecraft.tv/courses/angular/unit-testing/jasmine-and-karma/#_built_in_matchers)

Jasmine comes with a few pre-built matchers like so:

TypeScript

expect(array).toContain(member);

expect(fn).toThrow(string);

expect(fn).toThrowError(string);

expect(instance).toBe(instance);

expect(mixed).toBeDefined();

expect(mixed).toBeFalsy();

expect(mixed).toBeNull();

expect(mixed).toBeTruthy();

expect(mixed).toBeUndefined();

expect(mixed).toEqual(mixed);

expect(mixed).toMatch(pattern);

expect(number).toBeCloseTo(number, decimalPlaces);

expect(number).toBeGreaterThan(number);

expect(number).toBeLessThan(number);

expect(number).toBeNaN();

expect(spy).toHaveBeenCalled();

expect(spy).toHaveBeenCalledTimes(number);

expect(spy).toHaveBeenCalledWith(...arguments);

## [Setup and Teardown](https://codecraft.tv/courses/angular/unit-testing/jasmine-and-karma/#_setup_and_teardown)

Sometimes in order to test a feature we need to perform some setup, perhaps it’s creating some test objects. Also we may need to perform some cleanup activities after we have finished testing, perhaps we need to delete some files from the hard drive.

These activities are called *setup* and *teardown* (for cleaning up) and Jasmine has a few functions we can use to make this easier:

**beforeAll**

This function is called **once**, before all the specs in a test suite (describe function) are run.

**afterAll**

This function is called **once** after all the specs in a test suite are finished.

**beforeEach**

This function is called before **each** test specification (it function) is run.

**afterEach**

This function is called after **each** test specification is run.

We might use these functions like so:

TypeScript

describe('Hello world', () => {

let expected = "";

beforeEach(() => {

expected = "Hello World";

});

afterEach(() => {

expected = "";

});

it('says hello', () => {

expect(helloWorld())

.toEqual(expected);

});

});

## Running the tests

* In a default Angular CLI application, tests are run by using the **ng test** command, this will start from the root and go though and test every file with an extension of .spec.ts
* Tests will execute after a build is executed via Karma, and it will automatically watch your files for changes.
* You can run tests a single time via --watch=false.
* To run all the tests in our application we simply type ng test in our project root.
* This runs all the tests in our project in Jasmine via Karma.
* It watches for changes to our development files, bundles all the developer files together and re-runs the tests automatically.