## **Intro to Testing**

In a perfect world you would be able to write code, deploy it and never have to touch it again because it functions flawlessly. The reality of course is that things can, and often do go wrong. Whether its your own mistakes that have caused a bug or some unforeseen circumstances, you need a way to ensure code integrity as well as confidence in your apps.

The first common solution is manual-testing, but as applications grow in size and complexity it becomes unrealistic to rely on just manual testing to verify new features, catch bugs and notice regressions.

Best practices dictate that a combination of unit testing and end-to-end testing (commonly referred to as E2E testing), should be implemented. This guide will cover unit testing specifically but be sure to read our [E2E testing guide](http://localhost:3000/docs/angularjs/end-to-end-testing) for full test coverage.

## **What is Unit Testing**

The goal of unit testing is to take the smallest unit of functionality (a method or a function), isolate it from the rest of the code and ensure that it behaves as expected.

## **Intro to ngMock**

Luckily Angular was built with testing in mind. Dependency injection makes it very easy to mock services, controllers and even modules to isolate functions and methods of code thus enabling unit testing. The Angular team has created a module specifically for unit testing core Angular services called [ngMock](https://docs.angularjs.org/api/ngMock) that does just that.

## **Learning Unit Testing**

The following resources will get you up to speed on how to write your own Unit tests.

* [Pluralsight's ngMock course by Bradley Braithwaite](https://app.pluralsight.com/library/courses/angularjs-ngmock-unit-testing/table-of-contents) - a great hands-on introductory course to unit testing with ngMock
* [Jasmine Testing Framework](http://jasmine.github.io/2.4/introduction.html) - We use this framework in conjunction with ngMock to write our Unit Tests. You may have noticed the Pluralsight course covers it briefly but we recommend checking out their documentation for a deeper insight into their syntax and additional assertions that you can make.
* Examples in [Components](https://github.com/ordercloud-api/angular-components) and [Seed](https://github.com/ordercloud-api/angular-seed/tree/master/src/app) - Each of our components is unit tested to ensure proper functionality. Within each component directory there is a /tests subdirectory. The spec.js file within that directory will include our test scripts. You can take a look at how we write our unit tests to help you write your own.

## **Automated Testing**

Our seed comes with ngMock already configured and ready to run and we've leveraged the automation power of Gulp to make running your tests easier than ever. The following terminal command will run all of your unit tests: gulp test:unit

.

Best practices dictate that unit tests file structure should mirror that of your code. In each component you'll notice there is a test folder that includes a spec.js file with the tests for that specific component. When creating a new unit test you will simply need to make sure the file ends in spec.js and is in the /src directory so that it is picked up by our intelligent build system.