# **Unit Testing in Angular using Jasmine and Karma**

## Official angular testing Doc: [**Angular - Testing**](https://angular.io/guide/testing)

## Unit testing in angular help in:

* Early Detection of Bugs
* Code Quality & Documentation
* Refactoring & Regression Testing
* Continuous Integration and Deployment (CI/CD)

In Angular, Jasmine and Karma are commonly used for writing and executing unit tests. Angular's built-in dependency injection system and modularity make it easier to write unit tests for individual components, services, and modules, allowing developers to validate their code's correctness in a controlled environment before integrating it into the larger application.

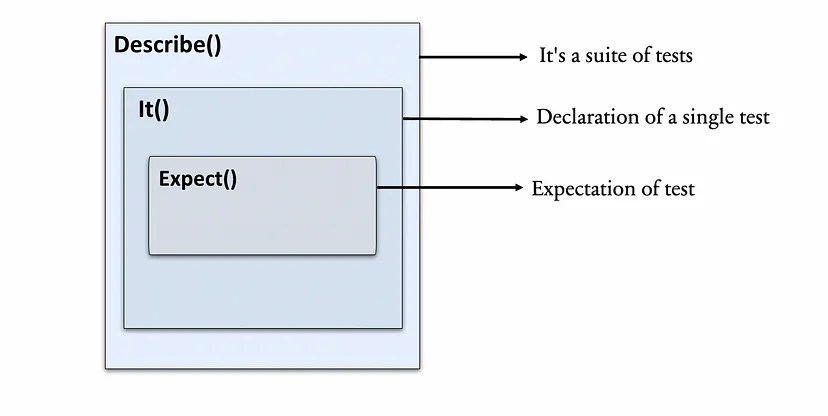
**Jasmine:** Behavior-driven development (BDD) testing framework for JavaScript code. It provides functions to help with structuring your tests and also making assertions in a human readable way. Jasmine is not specific to Angular, but it is widely used for testing Angular applications because it is easy to use and provides a good set of features for testing JavaScript code. [[Official Jasmine Doc]](https://jasmine.github.io/index.html)

**npm install --save-dev jasmine**

**Karma:** Karma is a test runner that helps to automate the process of running Jasmine tests. It can launch real browsers and run tests within them, which is important for testing Angular applications because they are browser-based. Karma can also watch development files for changes and re-run the tests automatically, which saves time when developing and testing applications. A **karma.conf.js** file contains configuration for karma. This file will specify the browsers that you want to run your tests in, the files that you want to include in your tests, and the reporters that you want to use to output the results of your tests. [[Official Karma Doc]](https://karma-runner.github.io/6.4/index.html)

**npm install karma**

**How to write a unit test**   
Generally, Angular generates a test case file on its own and provides a basic skeleton of the test case. It will follow this structure to write test cases.



**describe():** It is used to group related test cases. It accepts two arguments: a string that describes the group of specs(test cases), and a function that contains the specs(test cases) or nested describe statements. It is known as a test group or test suite.

**it():** It is used to define a single test spec. It also takes two arguments: a test spec description and a function that contains the expectation or assertions for the spec.

**expect():** It is used to create an expectation or assertion in a test spec. It takes a single argument, which is the value that you want to test, and then a matcher function that tests the value against the expected value.

**How to run unit tests****:** To run all the unit tests in the application: **ng test** / **npm run test**

To run a particular unit test from a particular spec file, add **‘f’** in front of the describe() and it() methods to tell jasmine to focus on a particular set of unit tests. Use **fdescribe()** and **fit()** to run specific unit tests.

# ***Example:***

Add function in angular component  


1. **Create a test group or test suite by writing** **describe() a method for the app component.**
2. **Create a test case that checks that the app component is initialized.**
3. **Follow standardized *AAA(Arrange-Act-Assert)* [**[AAA pattern](https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022#write-your-tests)**] pattern.Use *toBeTruthy()* matcher.**
4. **Write a test case for** **add() method.**



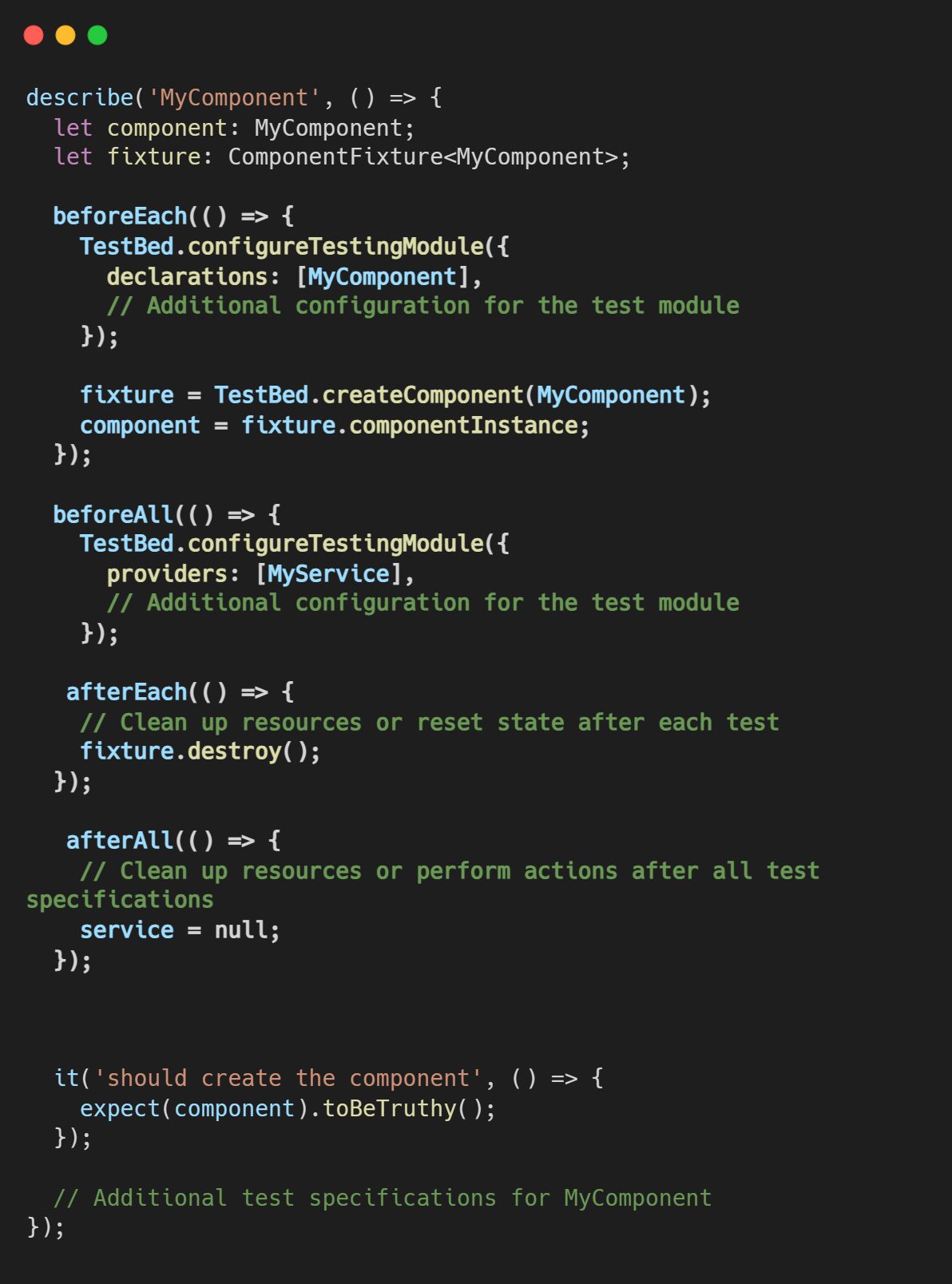
Avoiding initialization code every time using methods like beforeEach(), beforeAll(), afterEach(), afterAll()

**beforeEach():** Called before each test specification

**beforeAll():** Called once before all the specification

**afterEach():** Called after each test specification

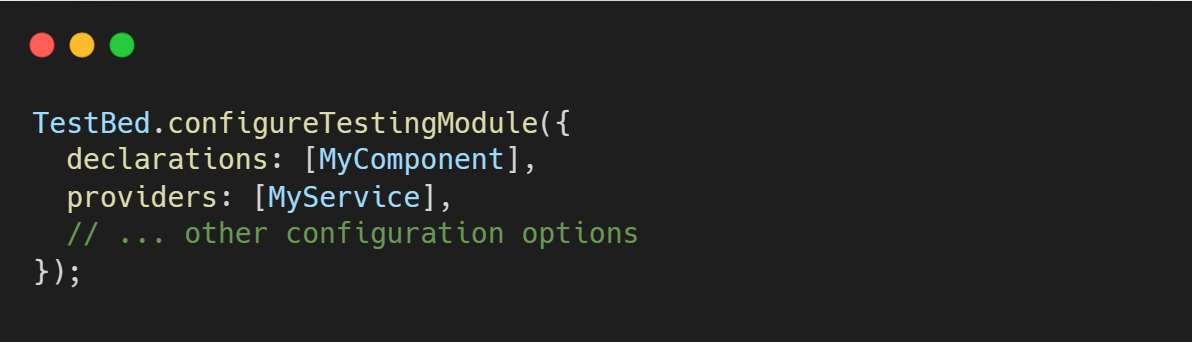
**afterAll():** Called once after all the specifications.



**Testbed (**[Official TestBed Doc](https://angular.io/api/core/testing/TestBed)**):**   
TestBed is a configured environment used for testing applications. It includes all necessary components and conditions for testing the system. Tests are performed by inputting data and evaluating output to ensure correct functionality. Used to create instances of Angular testing modules. The TestBed class is part of the “@angular/core/testing package” and is used to set up the environment for testing Angular components, services, and other constructs.

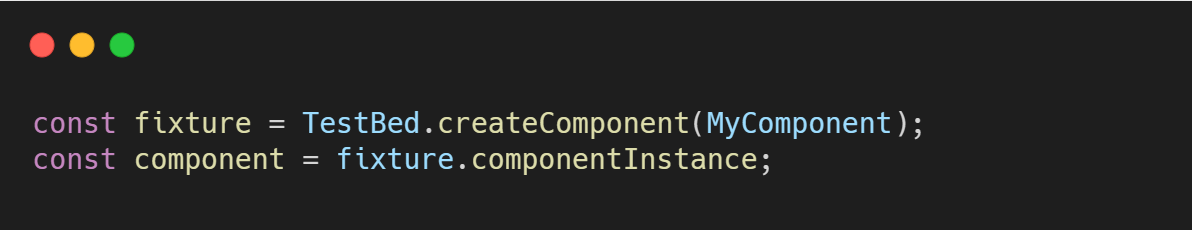
Some key aspects of Testbed :

**TestBed.configureTestingModule** : Configuring Testing Modules



**TestBed.createComponent** : Creating Component Fixtures

// A fixture is a wrapper for a component and its template.



**TestBed.inject** : Accessing Injected Services



**TestBed.overrideModule** : Setting Up Module-Level Configurations



**TestBed.resetTestingModule** : Resetting the Testing Environment



**Mocking:**

In Angular testing, mocking is the practice of replacing real implementations of certain dependencies, such as services or components, with simulated or controlled implementations. This is done to isolate the unit of code being tested and create a controlled environment for testing. Mocking is particularly useful when you want to test a specific part of your application in isolation without relying on the actual behavior of its dependencies.

Mocking a component:

Mocking a component involves creating a mock version of the component that can be used in place of the actual component during testing. This is particularly useful when you want to isolate a specific component from its dependencies or when you need to simulate certain behaviors for testing purposes.



Mocking a Service using SPY:

In Angular testing, a "spy" is a Jasmine feature that allows you to observe and control the behavior of functions, methods, or properties during a test. Spies can be used to track calls to functions, change their implementation, and provide controlled responses. In the context of Angular testing, spies are commonly used to mock or monitor the behavior of services, methods, and events.

Method Spies: Use method spies to replace methods of services, such as HTTP requests or other asynchronous operations, to control their behavior during testing.



Property Spies: Property spies are useful in scenarios where you want to monitor changes to a property, such as tracking changes to an input property in an Angular component.

