**Jasmine test framework** : Default unit testing framework in Angular

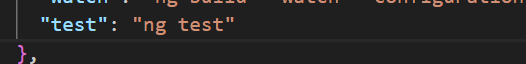
It checks .spec.ts files

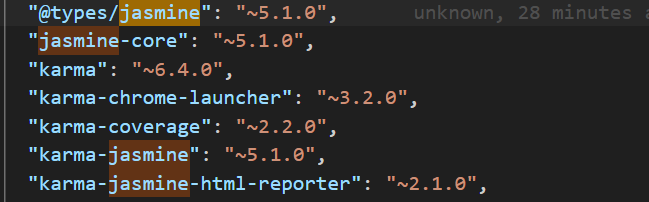
<https://angular.dev/guide/testing>

<https://angular.dev/cli/test>

<https://angular.dev/guide/testing/components-scenarios#testing-with-a-spy>

In package.json





**Key Concepts**:

**TestBed**: Angular's testing utility that helps create a test module (like **NgModule** but for testing).

**ComponentFixture**<T>: A wrapper around the component that allows access to its instance and the DOM.

**describe**(): A Jasmine function used to define a **test suite** (a group of related tests).

An Angular unit test is based on a describe container, which has several different blocks such as it, beforeEach etc.

**describe(string, function)** – takes a title and function containing one or multiple specs. It is also called a test suite. It as declare two variables, component, a reference to the component to be tested and fixture, a wrapper for a component, its template and output.

describe(FirstComponentComponent, () => {

**let** component: FirstComponentComponent;

**let** fixture: ComponentFixture< FirstComponentComponent >;

}

**FirstComponentComponent**: A label for this test suite, typically the name of the component being tested.

**component**: A reference to the actual component instance.

**fixture**: A testing wrapper that helps control and inspect the component.

**it()** is a **Jasmine function** for defining a test case.

**Test Description**: 'should create the component' (explains what the test does).

expect(component).toBeTruthy();:

* expect() is a **Jasmine assertion function**.
* .toBeTruthy() checks if the component is successfully created (true if it exists)

**Matchers** are functions in Jasmine that compare the actual test result with an expected value. Types:

Truthy & Falsy Matchers :toBeTruthy(), toBeFalsy()

Equality Matchers: toBe(), toEqual()

Comparision Matchers:toBeGreaterThan()

String & Array Matchers:toContain(x)

Spy Matchers: toHaveBeenCalled()

**Optional Lifecycle Hooks:**

[**https://angular.dev/guide/testing/components-scenarios#the-async-beforeeach**](https://angular.dev/guide/testing/components-scenarios#the-async-beforeeach)

**beforeEach()**: Runs before each test (it() block).

**afterEach(function)** – the beforeEach function tells the testing framework to run the function passed to it after each test. This is often useful if you want to clean up some global setup state that is shared across tests.

afterEach(() => {

*// Ensure that there are no outstanding requests to be made*

httpTestingController.verify();

});

**beforeAll**: Runs **once** before any test in the describe block.

**afterAll**: Runs **once** after all tests in the describe block.

**TestBed.configureTestingModule**({...}):

* Creates a test module that declares FirstComponentComponent.
* Mocks dependencies and sets up the testing environment.

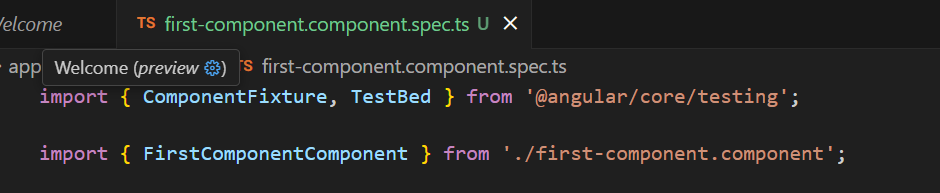
.compileComponents(): Ensures the component's template and styles are compiled **before running the test**.

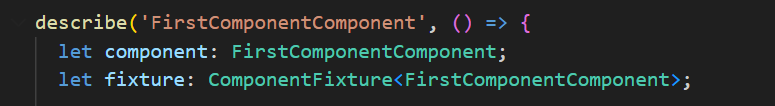
**mock** is a fake version of a real object (service, API, or dependency) used in testing. Instead of calling actual implementations, we provide **dummy data** to control test behavior.

**spy** is a Jasmine feature that tracks function calls (how many times they were called, with what arguments, etc.).  
Instead of replacing the whole service like a mock, a **spy just observes** a method.

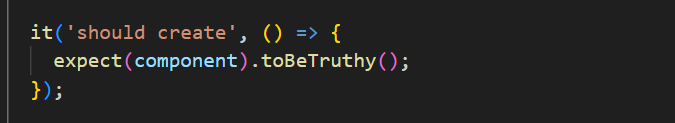
spy object is considered a "**partial mock**," it wraps around a real object and allows you to monitor specific interactions with it,

**spyOn()** allows us to **track and mock method calls** on services, components, or objects.

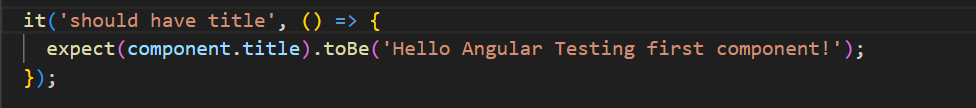




First test case:



Can add more test case

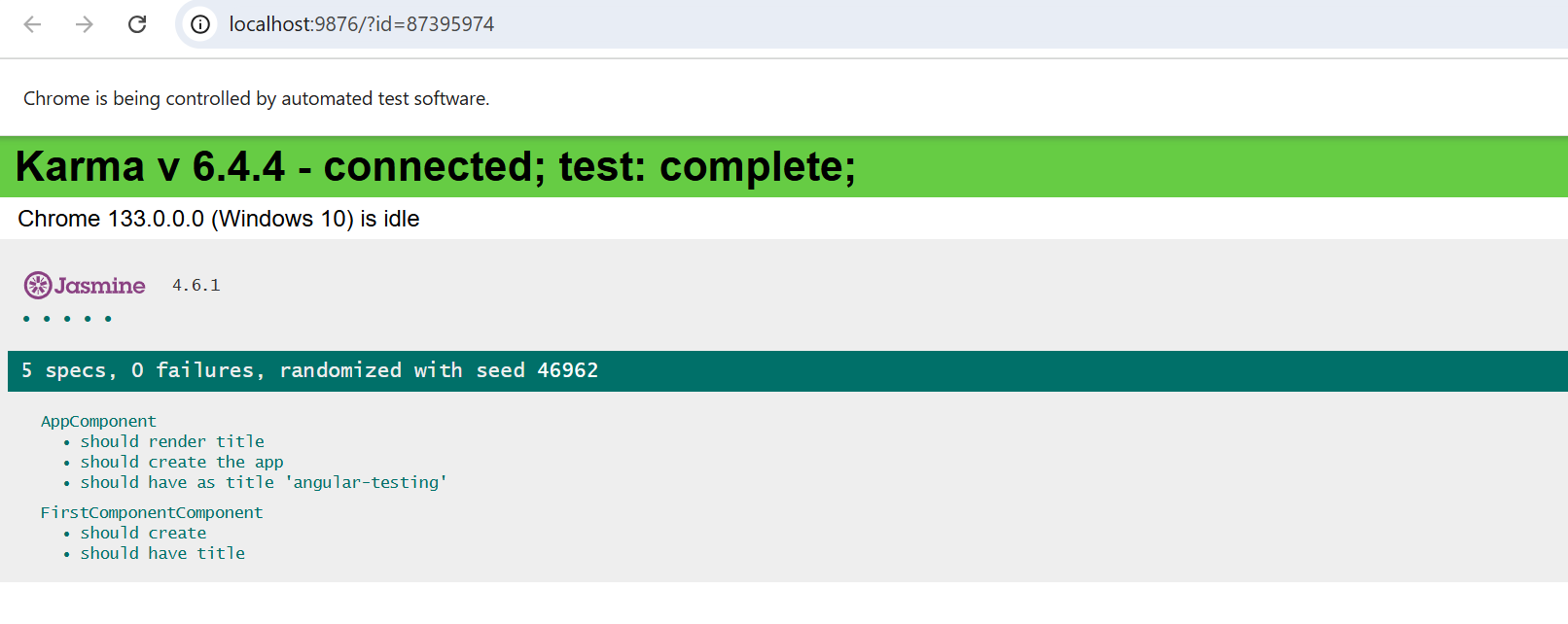


Run the test:

ng test

This will:

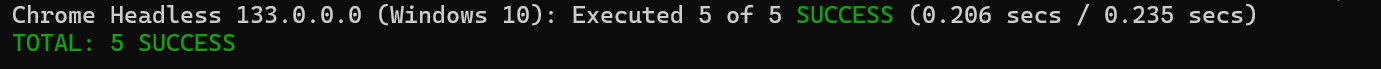
* Start **Karma**, which opens a browser.
* Run all tests inside \*.spec.ts files.
* Display test results in the console.



**(Optional) Run Tests in Headless Mode**

If you don’t want the browser to open, use:

ng test --watch=false --browsers=ChromeHeadless



Generate a **test coverage report**

ng test --no-watch --code-coverage

**ng test**

* Runs the unit tests in your Angular application.

**--no-watch**

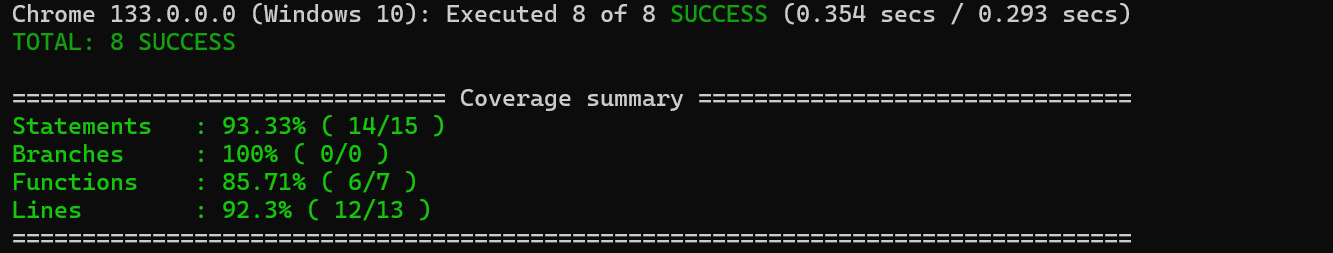
* Normally, ng test **watches for file changes** and re-runs tests automatically.
* Adding --no-watch **disables the watch mode**, meaning:
  + The tests **run once** and then **exit**.

**--code-coverage**

* Generates a **test coverage report**.
* Shows **how much of your code is covered** by tests.
* The report is stored in:

coverage/index.html

* You can open this file in a browser to see:
  + **Statements** covered
  + **Branches** tested
  + **Functions** tested
  + **Lines** covered



Analysis:

**Coverage Breakdown**

1. **Statements: 93.33% (14/30)**
   * Out of 30 total statements (lines of code that perform actions), 25 have been executed by your tests.
   * 6.67% of statements are not covered (5 statements were never executed).
2. **Branches: 100% (0/0)**
   * This refers to **conditional branches** (e.g., if-else or switch cases).
   * A value of 0/0 means your code doesn't contain any conditional branches, so there's nothing to measure.
3. **Functions: 85.71% (6/7)**
   * Out of 7 functions/methods, only 6 were executed.
   * 1 function were **never tested or executed** in your test suite.
4. **Lines: 92.3% (12/13)**
   * 12 out of 13 lines of code were executed.
   * 12 lines of code were **not reached** during testing.

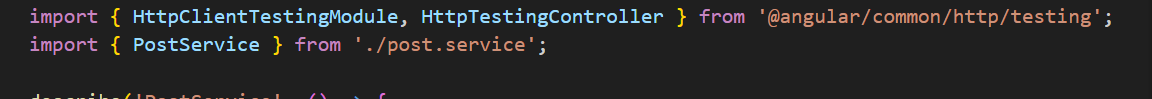
**Using --include to Run a Single Component's Tests**

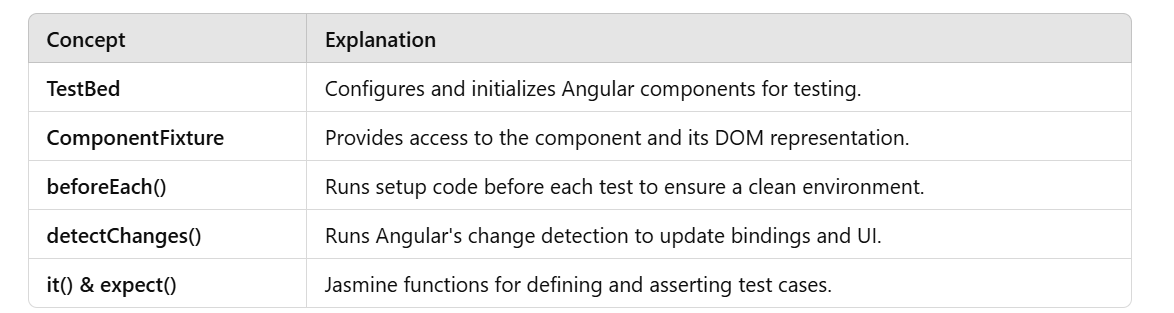
ng test --include=src/app/first-component/first-component.component.spec.ts

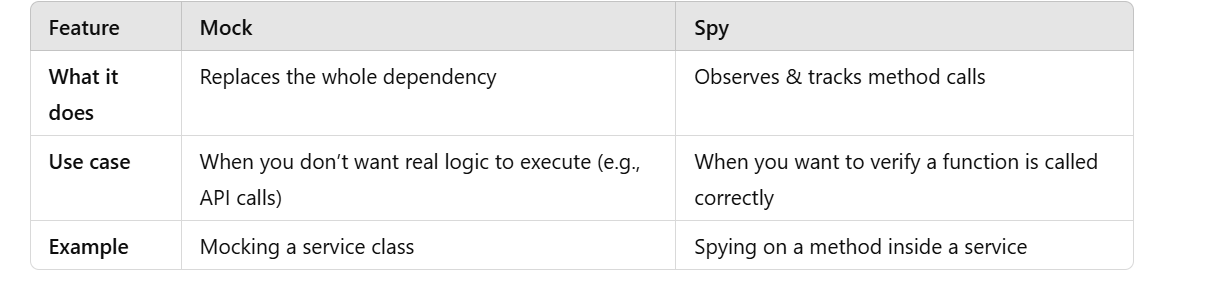
ng test --include=src/app/post.service.spec.ts

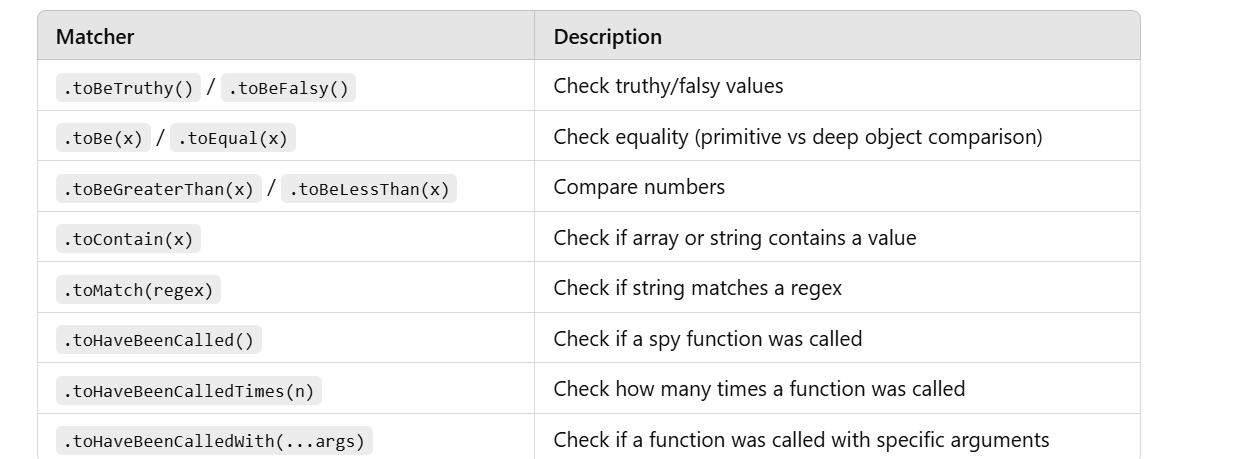
**Testing in Service:**

Use HttpTestingController to **mock API responses in tests**.









Snapshot testing:

**Karma/Jasmine** does not natively support **snapshot testing** like Jest does. Snapshot testing is typically associated with Jest, which serializes the rendered output of a component and compares it to a stored version.

However, you can achieve **snapshot-like testing in Karma/Jasmine** using additional tools:

However, you can achieve **snapshot-like testing in Karma/Jasmine** using additional tools:

**1. Using Karma Snapshot Plugin**

There is a third-party package called [karma-snapshot](https://www.npmjs.com/package/karma-snapshot), which enables snapshot testing in Karma.

npm install karma-snapshot karma-snapshots-preprocessor --save-dev

* Use Jest for snapshot testing (toMatchSnapshot).
* Use Karma for browser-based testing.