Jest Framework has **Snapshot testing** support.

**Faster execution** (runs outside a browser).

**Key concepts:**

**Test Structure**

describe('Math operations', () => {

it('should add two numbers correctly', () => {

expect(2 + 2).toBe(4);

});

});

**Test Suites (describe())**

* Groups related tests together.
* Helps organize tests in a structured way.

**Individual Tests (it() or test())**

* Defines a single test case.
* it() and test() are **interchangeable**.

it('should return true', () => {

expect(true).toBe(true);

});

test('should return false', () => {

expect(false).toBe(false);

});

**Matchers (expect())**

**Hooks (beforeEach, afterEach, etc.)**

* Used for **setup and cleanup** before/after tests.

beforeEach(() => {

console.log('Runs before each test');

});

afterEach(() => {

console.log('Runs after each test');

});

beforeAll(() => {

console.log('Runs once before all tests');

});

afterAll(() => {

console.log('Runs once after all tests');

});

**Mocking & Spies (jest.spyOn(), jest.fn())**

Used to mock dependencies like APIs or services.

**Mocking a Function**

const mockFn = jest.fn();

mockFn();

expect(mockFn).toHaveBeenCalled();

**Spying on a Method**

const obj = { getUser: () => 'John' };

jest.spyOn(obj, 'getUser').mockReturnValue('Mocked User'); expect(obj.getUser()).toBe('Mocked User');

**Asynchronous Testing (async/await)**

Jest supports testing **async code** using async/await.

it('should fetch user data', async () => {

const fetchData = async () => 'User Data';

await expect(fetchData()).resolves.toBe('User Data');

});

**Running a Single Test (only & skip)**

* **Run only one test:**

it.only('should run this test only', () => {

expect(true).toBe(true);

});

**Skip a test:**

it.skip('should not run this test', () => {

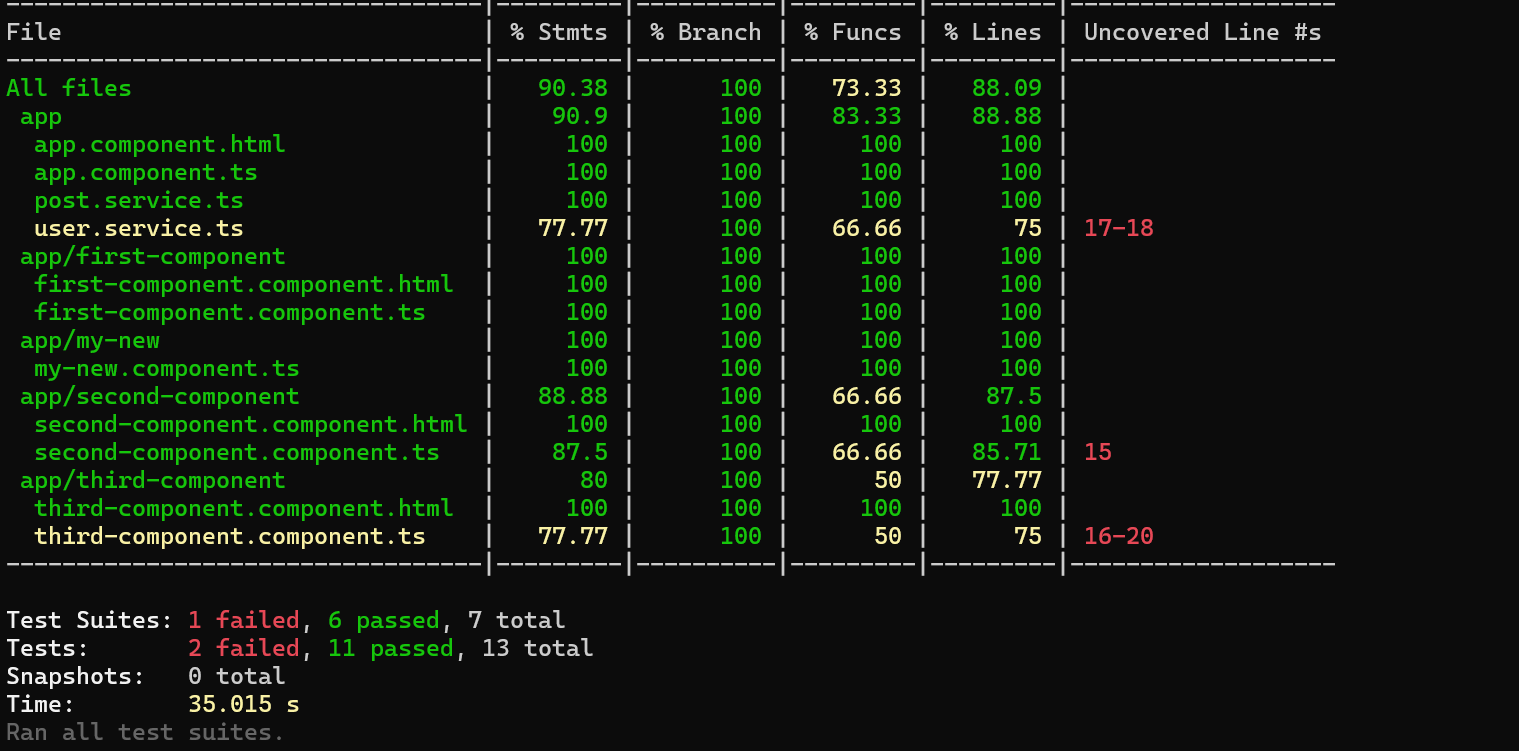
expect(true).toBe(true);

});

**Code Coverage (--coverage)**

Check which parts of your code are tested.

npm test -- --coverage

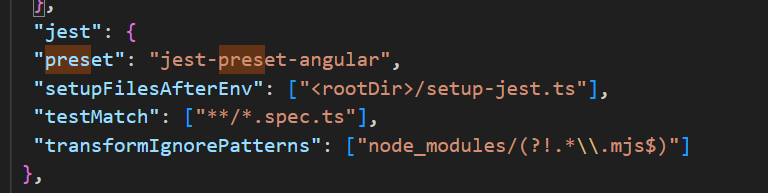


**Jest setup:**

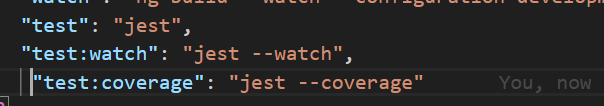
First, install Jest, jest-preset-angular, and other required dependencies:

npm install --save-dev jest jest-preset-angular @testing-library/angular @types/jest ts-jest

Modify package.json to use Jest instead of Karma:



Modify the test script to use Jest instead of Karma:



Now, you can run your tests with:

npm test

For **watch mode** (auto-runs tests on file changes):

npm run test:watch

For **code coverage report**:

npm run test:coverage

To test only one component

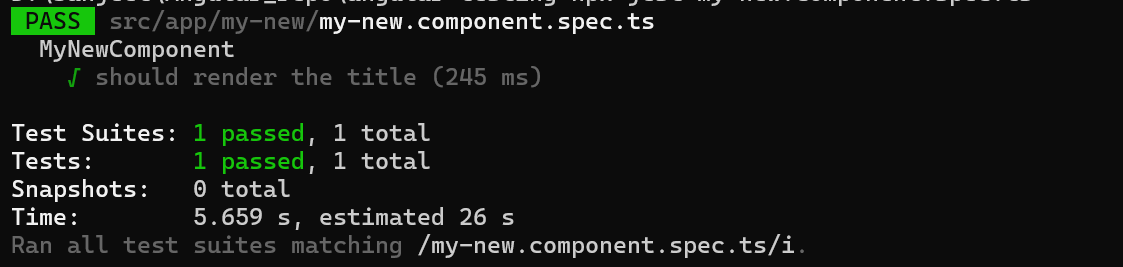
Run **only one test file** using:

npx jest my-new.component.spec.ts

npx jest my-button.component.spec.ts

or using npm

npm test -- my-new.component.spec.ts



**Jasmine test cases need some modifications to work with Jest** because:

* **Jasmine uses spyOn(object, 'method')**, while **Jest uses jest.spyOn(object, 'method')**.
* **Jasmine has beforeEach, afterEach, and matchers like toHaveBeenCalled()**, which are also in Jest, but some assertions work differently.
* **Karma auto-detects changes**, while **Jest runs tests in isolation**.

**Converting Jasmine to Jest**

Here’s what you need to change in your Angular test files:

spyOn(service, 'getUser').and.returnValue(of(mockUser));

change to Jest:

jest.spyOn(service, 'getUser').mockReturnValue(of(mockUser));

**Replace createSpyObj**

**Jasmine:**

const httpClientSpy = jasmine.createSpyObj('HttpClient', ['get']);

change to Jest:

const httpClientSpy = jest.spyOn(global, 'HttpClient');

**Convert done Callbacks for Async Tests**

**Jasmine:**

it('should fetch data', (done) => {

service.getData().subscribe((data) => {

expect(data).toEqual(mockData);

done();

}); });

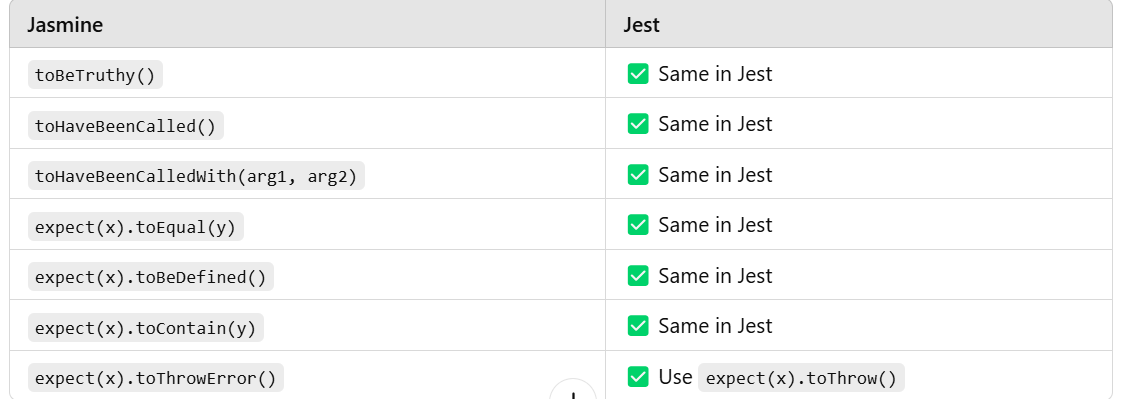
Change to Jest with async/await:

it('should fetch data', async () => {

await expect(service.getData().toPromise()).resolves.toEqual(mockData);

});

Most Jasmine matchers work in Jest (toBeTruthy(), toHaveBeenCalled(), etc.), but some need tweaks:



**Snapshot Testing in Jest**

Snapshot testing in Jest helps detect unexpected changes in UI components by comparing the rendered output to a saved "snapshot."

describe('MyButtonComponent', () => {

it('should match snapshot', async () => {

const { container } = await render(MyButtonComponent); expect(container).toMatchSnapshot(); });

});

**container.toMatchSnapshot()**: Compares the rendered HTML with the stored snapshot.

**Running Snapshot Tests**

Run the test using:

npm test

The first time the test runs, Jest will create a snapshot file in \_\_snapshots\_\_:

**Updating Snapshots**

If the UI changes intentionally, update snapshots using:

npm test -- -u

**Snapshot Testing with Components Having Inputs**

If a component has @Input(), you can pass props in the test.

If toMatchSnapshot() still doesn't work, try:

npx jest --clearCache

npm test