Angular Component Testing

Aitrich

Introduction to TDD in Angular

- TDD is a software development approach where tests are written before the actual code.
- In Angular, TDD ensures robust and maintainable applications.
- TDD cycle: Red - Green - Refactor.

TDD Steps in Angular

❑ **RED: Write a Failing Test**

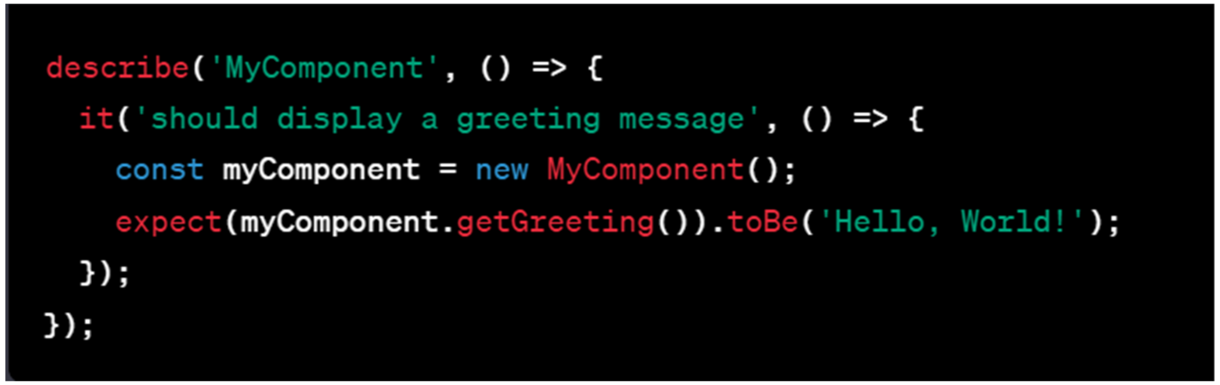
- Define the behavior you want (e.g., a component function).
- Write a test that describes this behavior and make sure it fails.
- Use testing frameworks like Jasmine or Jest.

❑ **GREEN: Write Minimal Code**

- Write the minimum code required to make the failing test pass.
- Don't worry about code quality at this stage; focus on making the test pass.

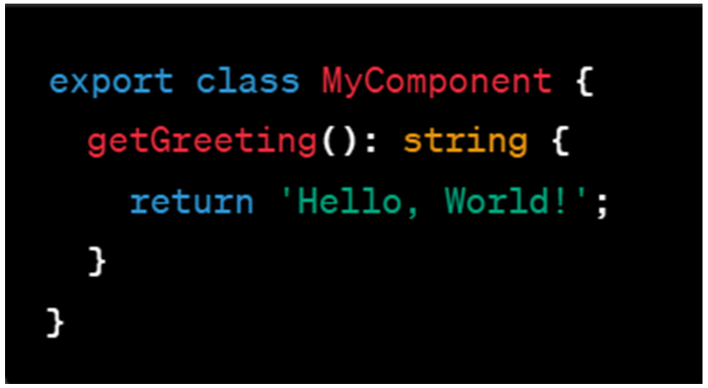
❑ **REFACTOR: Improve Code Quality**

- Refactor your code to improve readability, maintainability, and performance.
- Ensure all tests still pass after refactoring.



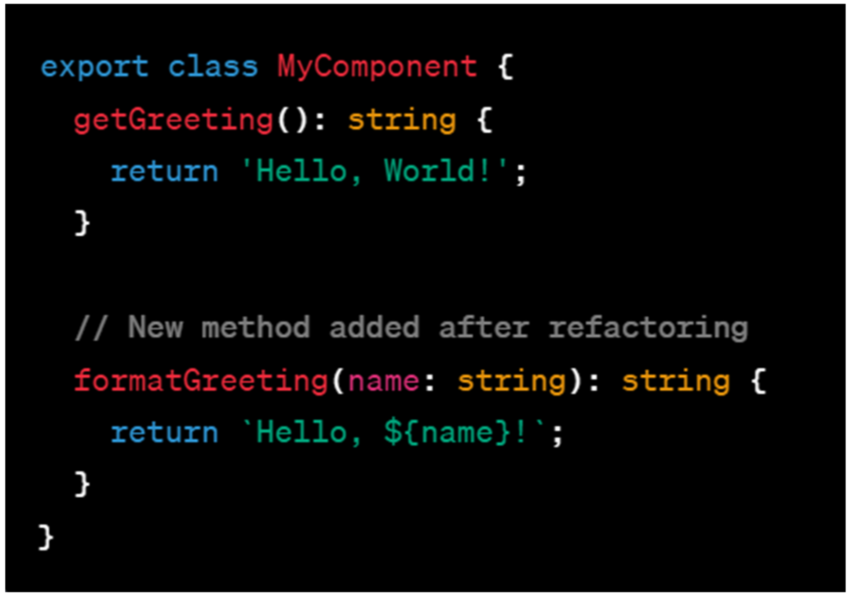
```
describe('MyComponent', () => {
  it('should display a greeting message', () => {
    const myComponent = new MyComponent();
    expect(myComponent.getGreeting()).toBe('Hello, World!');
  });
});
```

Red



```
export class MyComponent {
  getGreeting(): string {
    return 'Hello, World!';
  }
}
```

Green



```
export class MyComponent {
  getGreeting(): string {
    return 'Hello, World!';
  }

  // New method added after refactoring
  formatGreeting(name: string): string {
    return `Hello, ${name}!`;
  }
}
```

Refactor

Aitrich

Benefits of TDD in Angular

Early Detection of Bugs: TDD catches issues before they become major problems.

Documentation: Tests serve as living documentation of your application.

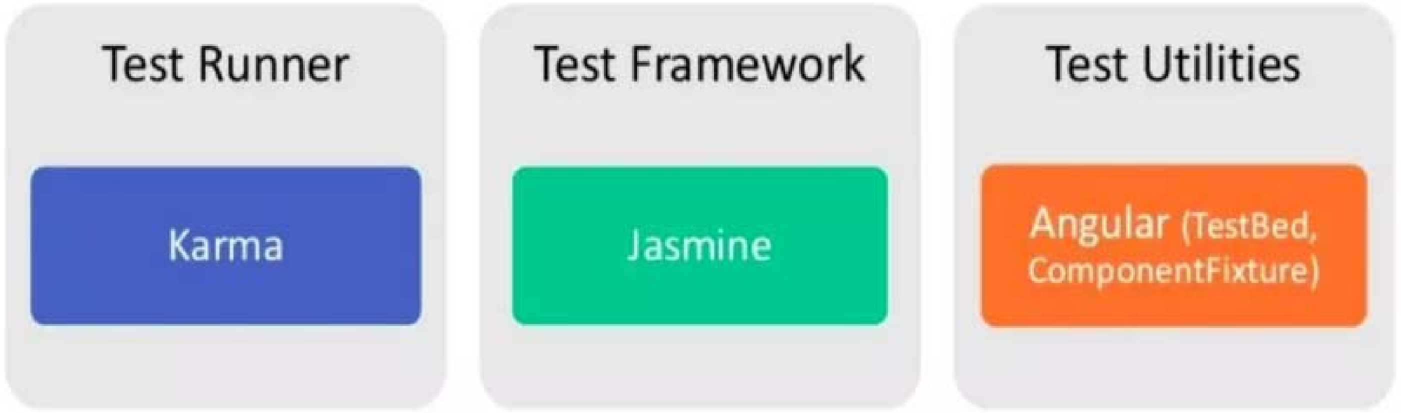
Faster Development: Catching bugs early reduces development time in the long run.

Improved Code Quality: Encourages clean, modular, and well-structured code.

Confidence: Developers have confidence that changes won't break existing functionality.

Aitrich

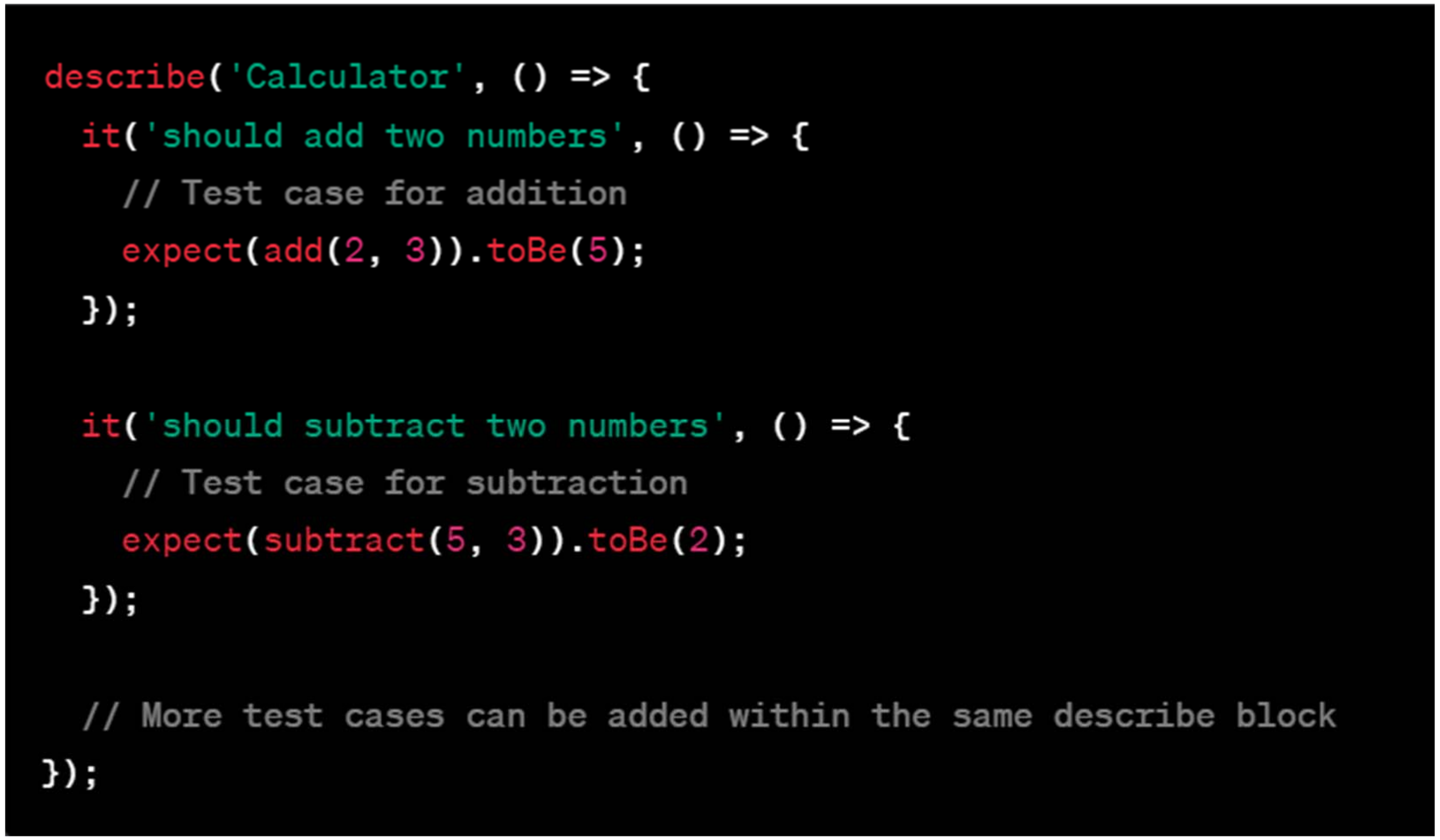
Angular Testing Tools



TEST SUITE

- In Jasmine, a test suite is a way to group related test cases or specifications. It provides a structure for organizing your tests and makes it easier to manage and run them. Test suites are created using the **describe** function in Jasmine.
- The **describe** function is used to define a test suite named "Calculator." The first argument to **describe** is a string that describes the suite, and the second argument is a function that contains the test cases.

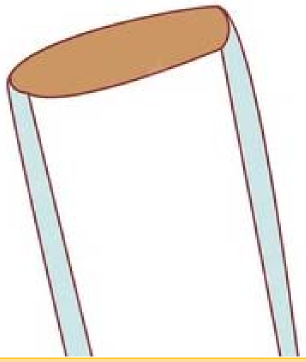
EXAMPLE



```
describe('Calculator', () => {
  it('should add two numbers', () => {
    // Test case for addition
    expect(add(2, 3)).toBe(5);
  });

  it('should subtract two numbers', () => {
    // Test case for subtraction
    expect(subtract(5, 3)).toBe(2);
  });

  // More test cases can be added within the same describe block
});
```



Jasmine

- Jasmine is a popular behavior-driven development (BDD) framework for writing unit tests in Angular.
- It provides a clean and expressive syntax for defining test cases and expectations.
- Jasmine supports various testing features like test suites, specs, matchers, and spies.

Karma

- Karma is a test runner that works well with Jasmine and other testing frameworks.
- It allows you to execute tests in multiple browsers and provides real-time feedback during development.
- Karma is often used for running unit tests in Angular applications.

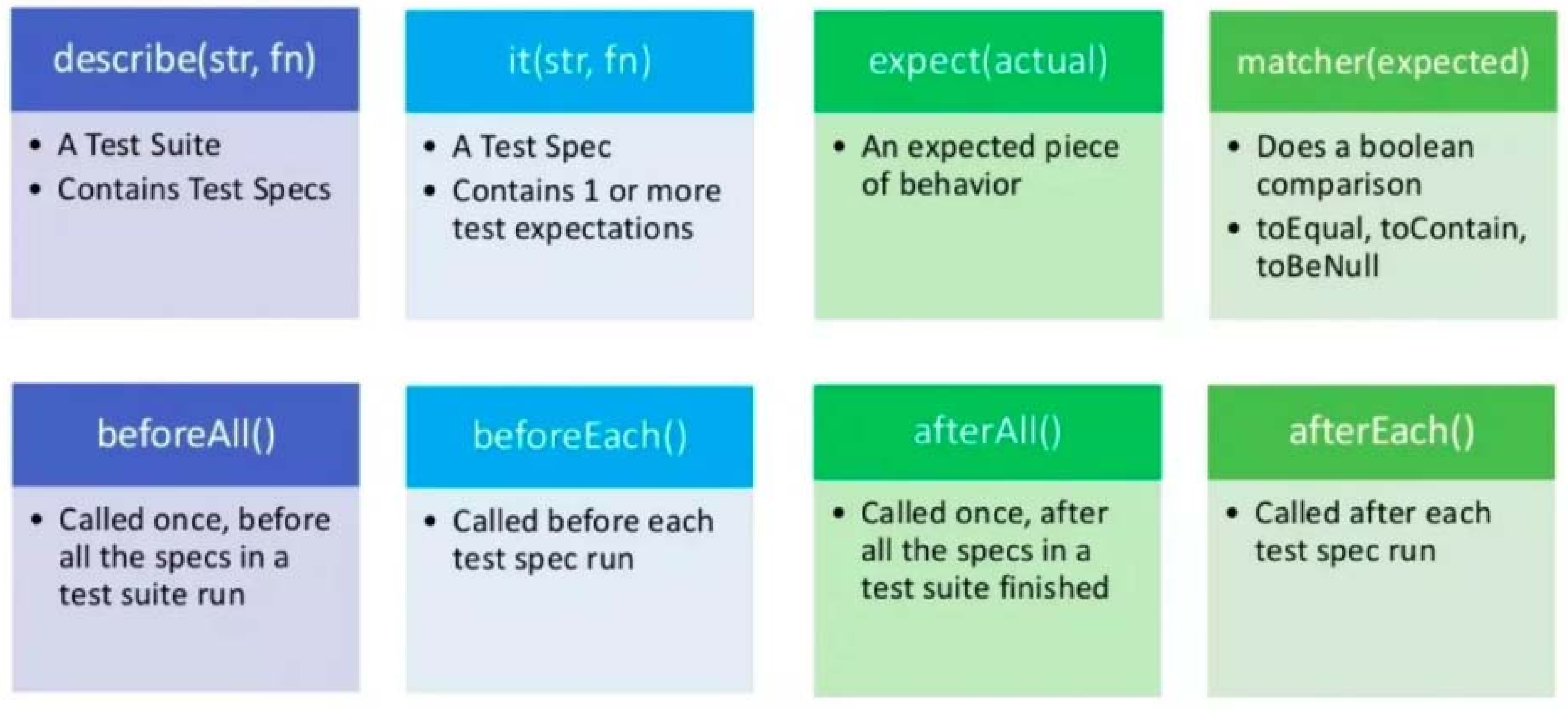
Protractor

- ❑ Protractor is an end-to-end (E2E)
- ❑ testing framework designed for
- ❑ Angular applications.
- ❑ It simulates user interactions with the application and checks its behavior.
- ❑ Protractor is particularly useful for testing user flows and ensuring that Angular components work together seamlessly.

Test Bed

- ❑ Angular's Test Bed is a utility for configuring and creating instances of Angular components for testing purposes.
- ❑ It provides a controlled environment for testing components, services, and modules.
- ❑ Test Bed is commonly used in unit testing Angular components.

Jasmine Test Specifications in Angular



describe(str, fn)
- A Test Suite
- Contains Test Specs

it(str, fn)
- A Test Spec
- Contains 1 or more test expectations

expect(actual)
- An expected piece of behavior

matcher(expected)
- Does a boolean comparison
- toEqual, toContain, toBeNull

beforeAll()
- Called once, before all the specs in a test suite run

beforeEach()
- Called before each test spec run

afterAll()
- Called once, after all the specs in a test suite finished

afterEach()
- Called after each test spec run

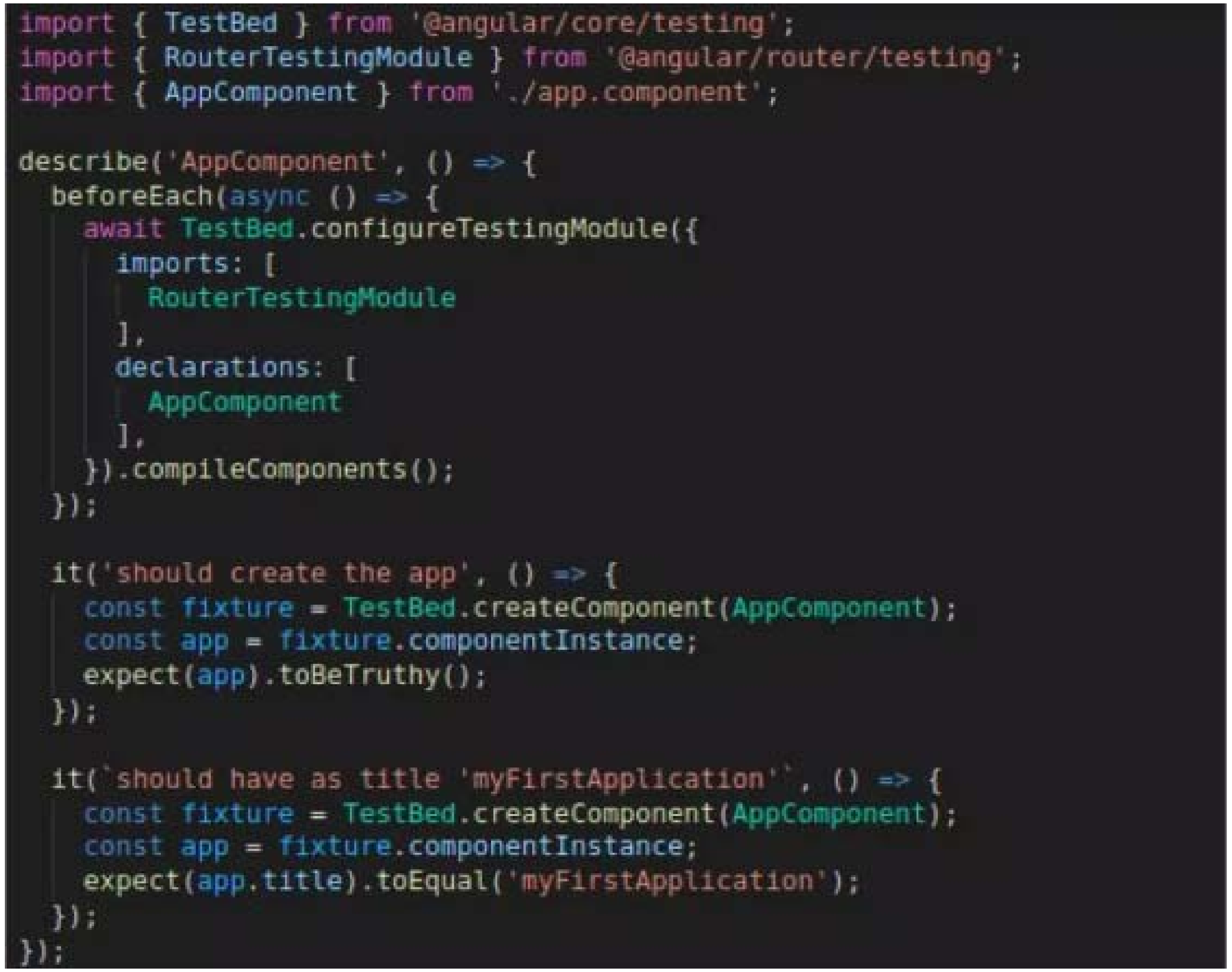
Aitrich

AAA Pattern

- ❑ Arrange-” arrange” everything like setup a groundwork for working with tests for execution.
- ❑ Act: Act on your Unit test case , Calling methods , processing data etc.
- ❑ Assert-Verifying the actual data of test result and expected data

Aitrich

Angular Default spec file



```
import { TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have as title 'myFirstApplication'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('myFirstApplication');
  });
});
```

Angular Default spec file

- ❑ Test Bed , Async --Import from built in modules
- ❑ Importing required modules
- ❑ Import required components
- ❑ Before Each Methods
- ❑ Test Bed.configureTestingModule
- ❑ A fixture is a wrapper for the component

Before each Method

- We use an async before each. The purpose of the async is to let all the possible asynchronous code finish before continuing.
- Before running any test in angular you need to configure an angular testbed
- This allows you to create an angular environment for the component being tested.
- Any module, component, or service that your tested component needs has to be included in the testbed.

Code Coverage

- The CLI can run unit tests and create code coverage reports. Code coverage reports show you any parts of your code base that might not be properly tested by your unit tests.
- To generate a coverage report run the following command in the root of your project.

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

- When the tests are complete, the command creates a new /coverage folder in the project. Open the index.html file to see a report with your source code and code coverage values.
- If you want to create code-coverage reports every time you test, set the following option in the CLI configuration file, angular. Json:

```
“test”: {
  “options”:{
    “codeCoverage”:true
      }
    }
```

Aitrich