

Testing introduction





Testing

- Why do we test?
- Different kinds of test
- · Unit/integration testing setup
- Our first test
- Dealing with asynchronousity
- Integration test a component



Why do we test?

- Prevent regression
- Create confidence for refactoring code
- Documented intentions
- Improved design
- Fewer bugs into production

Different kinds of tests

Unit test

UI Test

- Test small units (a class, a method) Test integrated application
- Don't use the DOM
- Run in browser
- · Mock all dependencies
- · Only test behavior via the DOM
- Run on target platform (phone, tablet or PC)
- Mock the backend or nothing at all (end-to-end test)



New kid on the block: Integration Test

- Test (parts of) our applications
- Can be shallow or deep
- Direct DOM interaction
- Don't test the UI itself
- · Mock away backend

Meet the players

- ✓ ✓ A R M A test runner for *unit* and *integration tests*
 - Use a *real* browser
 - Run tests from the command line
- Protractor test runner for *UI tests*end to end testing for Angular/s
 - Interact with the browser as a human being would
 - More on this in a later chapter
- System is our test framework
 - Doubles as our assertion and mocking framework
 - Behavior Driven Development (BDD)
 - Write specifications



Karma

- Run test: karma start
- Configure in karma.conf.js file

Jasmine - Simple example

```
describe('SuperHero', () => {
    let sut: SuperHero;
                                              // 'describe' a test suite
// System under test
                                              // 'beforeEach' hook,
     beforeEach(() => {
         sut = new SuperHero('Spiderman'); // runs before each test
     describe('when ability', () => {
                                            // You can (and should)
         let result: string;
                                             // nest test suites
         beforeEach(() => {
             result = sut.ability();
         it('should climb walls', () => { // 'it' specifies a test
             expect(result)
                                              // 'expect' creates a matcher
             .toBe('climb walls'));
         });
});
```



Our first unit test

```
// greet.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ /*...*/ })
export class GreetPipe implements PipeTransform {
    transform(value: any, args: any[]): any {
        return `Hey ${value}!`;
    }
}
```

Testing the pipe

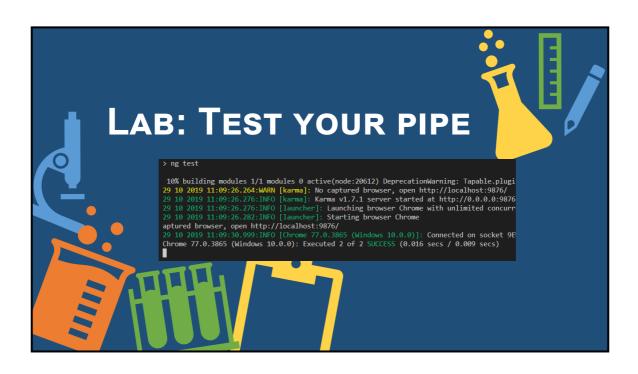
```
// greet.pipe.spec.ts
import { GreetPipe } from './greet.pipe.ts';

describe('GreetPipe', () => {
    let sut: GreetPipe;

    beforeEach(() => sut = new GreetPipe());

    describe('transform', () => {
        it('should return "hey Frank"! when called with "Frank"', () => {
            const actual = sut.transform('Frank');
            expect(actual).toBe('Hey Frank!');
        });
    });
});
```





Isolating unit tests

- Only test the unit and not its dependents or dependencies
- Use Test Doubles to isolate dependencies
 - **Dummies**: are passed around but never actually used.
 - Spies: record information about calls
 - Stubs: provide canned answers to calls made during the test
 - Mocks: pre-configured with details of the calls they expect

Jasmine uses a combination of spies and stubs



Jasmine - Test double example

To be continued

More on testing per chapter