Testing introduction

KARMA

Protractor
end to end testing for AngularJS

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

- Test small units (a class, a method)
- Don't use the DOM
- Run in browser
- Mock all dependencies

UI Test

- Test integrated application
- 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

- KARMA test runner for *unit* and *integration tests*
  - Use a *real* browser
  - Run tests from the command line
- Protractor test runner for *UI tests*
  - Interact with the browser as a human being would
  - More on this in a later chapter
- Jasmine 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

```
module.exports = config => { // npm module syntax
  config.set({                 // Call set on the karma config object

    frameworks: ['jasmine'],   // Frameworks karma should load

    files: [ 'src/**/*.js',    // Target which files should be loaded next
             'test/**/*.js'],

    reporters: ['progress'],   // Choose which reporters you want

    browsers: ['Chrome'],      // Choose which browsers to run

    singleRun: false           // If false, watch files and rerun on change
  });
}
```

Jasmine - Simple example

```
describe('SuperHero', () => {             // 'describe' a test suite
    let sut: SuperHero;                   // System under test

    beforeEach(() => {                    // 'beforeEach' hook,
        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!');
        });
    });
});
```

Lab: Test your pipe

```
> ng test

 10% building modules 1/1 modules 0 active(node:20612) DeprecationWarning: Tapable.plugi
29 10 2019 11:09:26.264:WARN [karma]: No captured browser, open http://localhost:9876/
29 10 2019 11:09:26.276:INFO [karma]: Karma v1.7.1 server started at http://0.0.0.0:9876
29 10 2019 11:09:26.276:INFO [launcher]: Launching browser Chrome with unlimited concurr
29 10 2019 11:09:26.282:INFO [launcher]: Starting browser Chrome
aptured browser, open http://localhost:9876/
29 10 2019 11:09:30.999:INFO [Chrome 77.0.3865 (Windows 10.0.0)]: Connected on socket 9E
Chrome 77.0.3865 (Windows 10.0.0): Executed 2 of 2 SUCCESS (0.016 secs / 0.009 secs)
```

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

```
let createHero = jasmine.createSpy('createHero');      // Create a spy

createHero.and.returnValue({ name: 'Daredevil'});      // Configure stub behavi

expect(createHero(123)).toEqual({ name: 'Daredevil'}); // Call the method

expect(createHero).toHaveBeenCalledWith(123);          // Verify behavior on th

jasmine.createSpyObj('SuperHero', [      // Creates an object with spy/stub met
    'useAbility', 'attack', 'canFly'
]);

spyOn(hero, 'useAbility')   // Create a spy method...
  .and.callThrough();       // but still will call the real method
                            // only works in `beforeEach` and `it`.
                            // Will automatically be cleaned-up after the test
```

To be continued

More on testing per chapter