Angular 2 Testing for Hackers

Angular Connect 2016

Prerequisites

- Get hold of a TypeScript code editor (e.g. Visual Studio Code or WebStorm)
    - Install Visual Studio Code
- Install Node.js (maybe nvm too)
    - Install node 6.5.0 or later and npm 3.10.6 or later
    - Or use nvm
- Clone the repository
    - `git clone https://github.com/angular-workshops/angular2-testing.git`
    - `cd angular2-testing`
- Install the angular CLI tool globally (needs to be >= beta.15)
    - `npm install -g angular-cli`
- Install the karma CLI tool globally
    - `npm install -g karma-cli`
- Install the local dependencies for jasmine folder
    - `cd jasmine`
    - `npm install`
- Install the local dependencies for Tour of Heroes folder
    - `cd ../tour-of-heroes`
    - `npm install`
- The day before the workshop run “git pull” to update to the latest version of the repo since there may be changes

Browse the full Prerequisites document with more detailed information and instructions.

Overview

9:00am	Intro & Benefits of Testing
9:15am	Jasmine BDD Framework
10:15am	Unit Testing Strategies
10:30am	BREAK
11:00am	Unit Testing with Karma
11:30am	Unit Testing in Angular 2
12:30pm	LUNCH
1:30pm	Integration Testing in Angular 2

3:00pm	BREAK
3:30pm	End to End Testing with Protractor
5:00pm	FINISH

Introduction to Unit Testing and Jasmine

Step 1: Creating specs

Resources

- Jasmine BDD Framework
- Comparison of Testing Frameworks

Goals

- Create specs to test a `join(array, separator)` function in the `joiner.js` file

Demonstration

- Create a spec file `joiner.spec.js` to hold the Jasmine specs
- Add a describe block for the `Joiner` class
- Create a beforeEach block to show instantiating the `Joiner` class
- Add a `describe` block for the `join` function
- Add an `it` block to test the main use case:

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/jasmine/src/Joiner.spec.js

Tasks

- Add `it` blocks to test corner cases of the `joiner` function, such as:
  - `should return an empty string if array is empty`
  - `should join with a comma if no separator is provided`
  - `should work with an empty string separator`
  - `should error when not passed an array`
- Write the following specs for the Reverser.js file which reverses a number into a reverse string
  - `should return "0" for 0`
  - `should return "333" for 333`
  - `should return "123" for 321`
  - `should return "12.3" for 3.21`
  - `should return "5-" for -5`
- **Bonus:**
  - Find the bug in the bowling game algorithm in `BowlingGame.js`

Step 2: Creating custom Jasmine matchers

Resources

- DAMP & DRY tests
- Custom Matchers in Jasmine

Goals

- Create custom matchers to make our tests more readable

Demonstration

- Look at the test for `FivesArray` (`FivesArray.spec.js`) which has a bug so has a failing test
    - Add the Spec to the SpecRunner.html
- When it fails, the error message “`Expected 2 to be 0`” isn’t helpful. And testing the empty array is ok, but could be smoother.
- Create a custom matcher for `toBeEmptyArray`()
    - Don’t include the custom message yet
- Replace the verbose matcher with the custom matcher
- View the default error message
- Add better error messages for failed matches

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/jasmine/src/FivesArray.spec.js

Tasks

- Create your own custom matcher `toBeSquareRootOf()`

```
describe('toBeSquareRootOf', function() {
 it('should match that 3 is the square root of 9', function() {
  expect(3).toBeSquareRootOf(9);
 });
});
```

Step 3: Mocking out dependencies for tests

Resources

- Arrange - Act - Assert
- Unit Testing: Mocks, Stubs and Spies
- http://martinfowler.com/articles/mocksArentStubs.html
- Jasmine Spies - API reference

Goals

- Create spies to use to test a class that relies upon another class
- Use `spyOn & jasmine.createSpyObj`

Demonstration

- Fill in the `Player.spec.js` tests using `toBePlaying(song)` custom matcher and spies on Song.

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/jasmine/src/Player.spec.js

Tasks

- Spy on the customer object to complete the tests in `Order.spec.js`:
  - `should not discount unpreferred customers`
  - `should give preferred customers a 10% discount`

Unit Testing Strategies

Resources

- TDD: Is There Really Any Debate Any Longer?

Unit Testing with Karma

Step 4: Configuring and running Karma

Resources

- Karma Documentation

Goals

- Install and configure Karma
- Run our unit tests for the app via Karma

Demonstration

- `npm install -g karma-cli`
- `npm install --save-dev karma`
- `npm install --save-dev karma-chrome-launcher`
- `npm install --save-dev karma-jasmine jasmine-core`
- `cd jasmine`
- `karma init` *(answering all the questions; source & test files: `src/*.js`)*
- `karma start` *(trigger a test run)*

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/jasmine/karma.conf.js

Tasks

- Configure Karma to open up different browsers
  - You'll need to install another launcher. E.g. `karma-firefox-launcher`
  - And update the `karma.conf.js` file

Step 5: Testing projects with transpiled source

Resources

- Angular CLI Documentation

Goals

- Setup a new Angular CLI based app
- Walk through of Tour of Heroes app

Demonstration

- Create a new project
  - `ng new my-app` *(takes 2 mins 17 secs on my machine)*
- Demonstrate the development server
  - `cd my-app`
  - `ng serve`
- Browse to `localhost:4200`
- Demonstrate running tests
  - `ng lint`
  - `ng test`
  - `ng e2e`
- Demonstrate creating a production build
  - `ng build`
- Run the tour-of-heroes app - show the app running in the browser
  - `cd ../tour-of-heroes`
  - `ng serve`
- Walk through the application code in the IDE

Tasks

- Create your own empty app using the Angular CLI tool

Step 6: Debugging unit tests

Unit Testing in Angular

Step 7: Testing services, pipes and components in isolation

Resources

- Three Ways to Test Angular 2 Components

Goals

- Write isolated unit tests for services
- Write an isolated unit test for a pipe
- Write an isolated unit test for a component (no ATP)
- Add a custom matcher and associated type declaration

Demonstration

- Show a really simple spec for a service with no dependencies: `in-memory-data.service.spec.ts`

- Show an isolated spec that mocks its dependencies with Observable and Response objects in the mocks: `hero-search.service.isolated.spec.ts`

- Show an async pure isolated spec that mocks everything: `hero.service.pure-isolated.spec.ts`

- Show an isolated spec for a Component: `app.component.isolated.spec.ts`

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/shared/in-memory-data.service.spec.ts
- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/hero-search.service/hero-search.service.isolated.spec.ts
- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/hero.service/hero.service.pure-isolated.spec.ts
- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/app.component/app.component.isolated.spec.ts

Tasks

- Add an isolated spec for the `ExponentialStrengthPipe`
- Finish the hero.service.isolated.spec.ts tests for getHero, delete, create, update

Step 8: Understanding Asynchronous Tests

Resources

- What the hell is Zone.js and why is it in my Angular 2?
- Zone.js project
- Angular 2 - API docs: async()
- Angular 2 - API docs: fakeAsync()

Goals

- Create an async unit test for the `HeroDetail` component

Demonstration

- Show an isolated async spec for a Component: `hero-detail.component.isolated.spec.ts`

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/hero-detail.component/hero-detail.component.isolated.spec.ts

Tasks

- Create a `fakeAsync` spec for the `HeroDetailComponent.save()` method:

```
describe('save()', () => {
 it('should update the heroService and then goBack', fakeAsync(() => {
 }));
});
```

- **Bonus:** Using `window.history.back()` method is not so nice. Write tests and refactor the code to use `import {Location} from '@angular/common';`

Integration Testing with Angular Test Platform (ATP)

Step 9: Shallow Testing Components

Resources

- Angular 2 - NgModule Guide
- Angular Test Platform - Guide

Demonstration

- Write shallow integration tests for the `AppComponent`

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/app.component/app.component.shallow.spec.ts

Tasks

- Test that the navigation panel is shown in the `AppComponent` template

Step 10: Integration Testing Pipes

Goals

- Test a Pipe class from inside a template

Demonstration

- Write the `exponential-strength.pipe.shallow.spec.ts` tests

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/exponential-strength.pipe/exponential-strength.pipe.shallow.spec.ts

Tasks

- Refactor the `ExponentialStrengthPipe` integration test to allow the `value` and the `power` to be controlled by the properies on the component:
    - `{{ value | exponentialStrength: power }}`

Step 11: Integration Testing Services and Mock Http

Resources

- Testing stuff in Angular 2 API docs

Demonstration

- Create the `hero.service.shallow.spec.ts` file
- Setup the injector, via `TestBed`
- Test the `getHero()` method - subscribing to the `connection` and storing it for later
- Test the `getHero()` method - subscribing to the `connection` and responding inside the subscription
- Test the `getHero()` method - using the `MockBackend.connectionsArray` to get the connect

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/hero.service/hero.service.shallow.spec.ts

Tasks

- Test the case for `getHero(id)` where `id` is not valid
- Write a test for the back button click on the `HeroDetailComponent` using the `TestBed`.
- **Bonus**: Write integration tests for the `hero-search.service.ts` file

Step 12: Shallow Component Tests with Change Detection

Demonstration

- Configure shallow tests for `HeroesDetailComponent`
- Create spec for initial display of component (using `async` and `detectChanges`)
- Create spec for initial display of component (using `async` and `autoDetectChanges`)
- Create spec for initial display of component (using `fakeAsync` and `tick`)

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/hero-detail.component/hero-detail.component.shallow.spec.ts

Step 13: Shallow Component Tests with DOM interaction

Resources

- DebugElement API guide

Demonstration

- Continue shallow tests for `HeroesDetailComponent`
- Create spec for the **name input changing** (via nativeElement API)
- Create spec for the **name input changing** (via debugElement API)

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/hero-detail.component/hero-detail.component.shallow.spec.ts

Tasks

- Write tests for the `heroes.component.ts`. This will require:
  - Ignoring and/or mocking other components
  - Spying on services
  - Managing change detection
  - Interacting with the DOM

Step 14: Deep Testing Nested Components

Demonstration

- Create `heroes.component.deep.spec.ts` file
- Configure the TestBed to compile both `HeroesComponent` and `HeroComponent` and to do initial change detection due to the `HeroService` promises.
- Check that the list of HeroComponent elements is being rendered correctly and passed the correct input value.
- Check that the output events from each HeroComponent is being handled correctly.

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/src/app/heroes.component/heroes.component.deep.spec.ts

Tasks

- Consider moving the selection of heroes into the HeroComponent and outputting a new "select" event.

End-to-end Testing with Protractor

Step 15: Setting up Protractor

Resources

- Annotated Protractor Config Example

Goals

- Configure Protractor
  - Connect to a browser
  - Connect to all Angular 2 app roots
  - Load TypeScript specs

Demonstration

- Walk through the protractor.conf.js file

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/protractor.conf.js

Tasks

- Configure Protractor to also test against Firefox 47 (48 is not yet compatible)

Step 16: Writing a spec

Resources

- Protractor Locators
- Protractor Control Flow

Goals

- Create a spec file for Heroes page
- Import the necessary globals
- Create specs for the Heroes page

Demonstration

- Create `e2e/heroes.e2e-spec.ts` file
- Add a describe block
- Add `beforeEach` block to navigate to the page
- Create a set of `it` blocks to describe the page behaviour showing the various aspects of the Protractor API

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/e2e/heroes.e2e-spec.ts

Tasks

- Create `e2e/dashboard.e2e-spec.ts` file
- Add specs for the **dashboard** page of the app to the new file

Step 17: Using Page Objects

Resources

- Page Objects by Martin Fowler
- Protractor Page Objects

Goals

- Refactor the Heroes page specs to use a `HeroesPage` object

Demonstration

- Create `e2e/page-objects/heroes-page.ts`
- Add methods for each of the interactions with the browser found in `e2e/heroes.e2e-spec.ts`
- Import the `HeroesPage` into `e2e/heroes.e2e-spec.ts`
- Use the `HeroesPage` methods rather than interacting with the Protractor API directly

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/e2e/heroes-2.e2e-spec.ts
- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/e2e/page-objects/heroes-

page.ts

Tasks

- Create `e2e/page-objects/dashboard-page.ts` file
- Implement the `DashboardPage` with methods for the dashboard specs.
- Refactor the specs for the **dashboard** page to use `DashboardPage`

Step 18: Debugging strategies

Resources

- Debugging Protractor Tests
- NodeJS Debugger Docs

Goals

- Demonstrate pausing a protractor spec

Demonstration

- Add a `browser.pause()` statement in `e2e/page-objects/heroes-page.ts`
- Increase the default timeout in protractor.conf.js to give you time to debug:
- Run the tests and then see what the browser looks like when the spec pauses
- Show that you can open the browser developer console and run protractor commands:
    - Type `repl` to enter JavaScript code
    - Type `element(by.css('input')).getAttribute('value')` to show the value of the input element
    - Press `Ctrl-C` to exit the repl
    - Type `c` to continue to the next command

Code

- https://github.com/angular-workshops/angular2-testing/blob/solution/tour-of-heroes/e2e/page-objects/heroes-page.ts

Tasks

- Try debugging into other tests

Step 19: Taking screenshots

Resources

- Taking Screenshots with Protractor

Goals

- Add code to record a screenshot in the middle of a spec

Demonstration

- Walk through the `screenshot.ts` file
- Import the helper function into `e2e/page-objects/heroes-page.ts`
  ```
  import {saveScreenshot} from '../screenshot';
  ```
- Call the helper in `e2e/page-objects/heroes-page.ts`

```
addNewHero(name: string) {
  this.getAddHeroTextBox().sendKeys('New Hero');
  saveScreenshot('adding-a-hero');
  this.getAddHeroButton().click();
}
```

- Run the tests and check that the screenshot was written to disk like:
  `temp/screenshots/adding-a-hero-1474540605725.png`

Continuous Testing with Travis (*Optional*)

Step 20: Configure Travis to run our tests

Resources

- Travis CI Docs

Goals

- To configure Travis CI to run our linting, unit and e2e tests on every push to Github
- To configure Travis CI to use browsers hosted by SauceLabs

Demonstration

- Walk through Travis website and config file

Tasks

- Try pushing the project to your own GitHub repository and setting up Travis yourself.