

Angular 2 Unit Testing

Sang Shin

JPassion.com

“Code with Passion!”



Topics

- Why testing Angular apps?
- Introduction to Jasmine
- Introduction on Karma
- Unit-testing Angular app
 - > Testing components
 - > Testing services

Why Testing Angular Apps?

Angular is designed with testing in mind

- The built-in dependency injection makes unit testing easy – dependency objects can be mock objects during testing
- Separation of concerns (component class is separated from template) makes it easier to test

Introduction to Jasmine

What is and Why Jasmine?

- Behavior Driven Development (BDD) framework for testing JavaScript code
 - > Most popular choice for testing Angular applications
 - > Does not depend on any other framework (jQuery, Angular)
 - > Does not require DOM – no need to run it within a hosting HTML page
- Makes testings self-document
 - > Jasmine uses the term 'spec' to mean 'test'
 - > This emphasizes the BDD nature of Jasmine – we use Jasmine to specify expected behavior of the system

Writing Test Specs in Jasmine

- describe(), it(), expect()

// In Jasmine, use the describe function to group the tests together:

```
describe('sorting the list of users', function() {
```

```
  it('sorts in descending order by default', function() {
```

```
    // your test assertion goes here
```

```
    expect(sortThemInDescendingOrder()).toBe(sortedResult1);  
  });
```

```
  it('sorts in ascending order', function() {
```

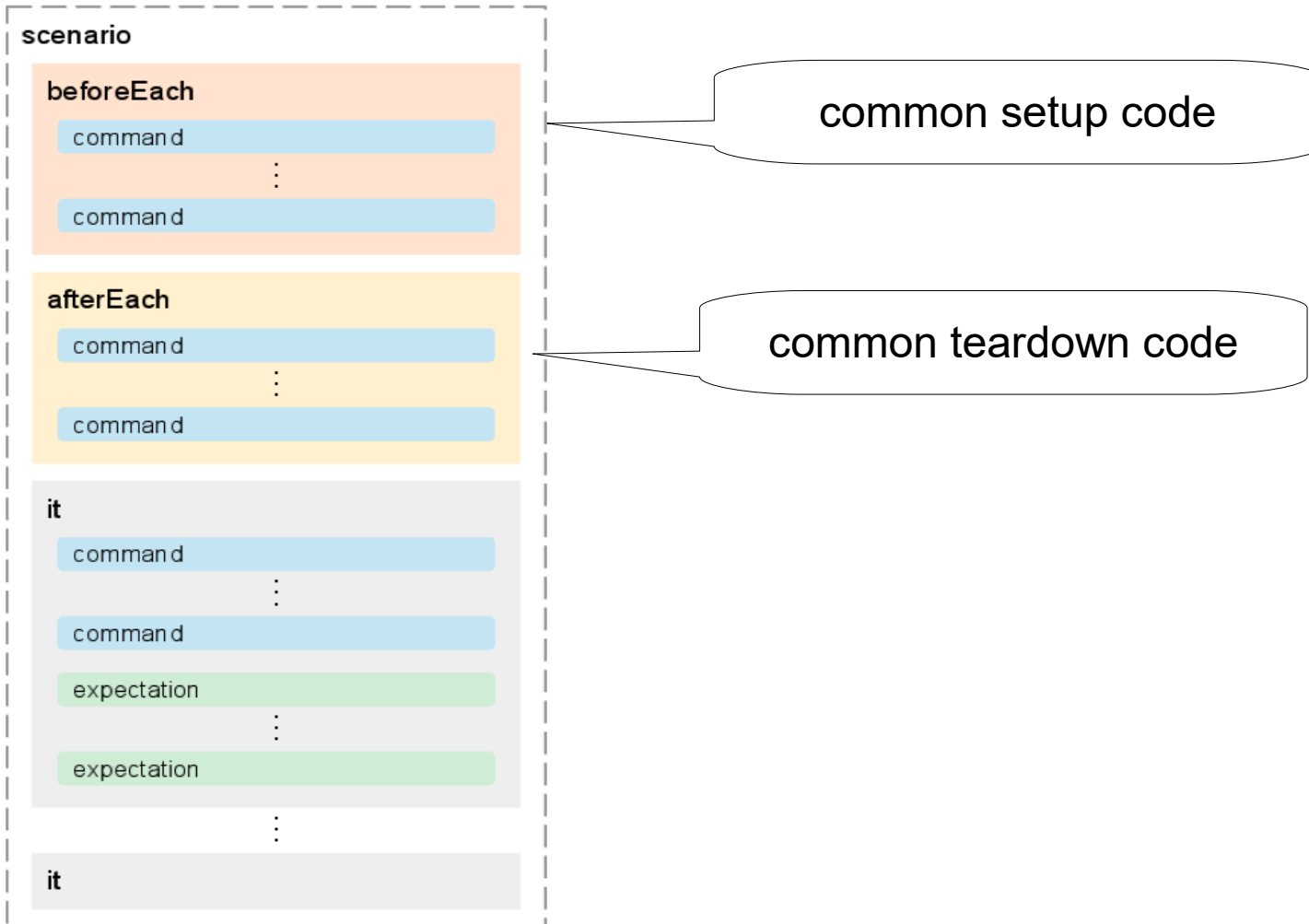
```
    // your test assertion goes here
```

```
    expect(sortThemInAscendingOrder()).toBe(sortedResult2);  
  });  
});
```

Jasmine Provided Matchers

- `toBe()`
- `toEqual()`, `toMatch()`
- `toBeDefined()`, `toBeUndefined()`, `toBeNull()`
- `toBeTruthy()`, `toBeFalsy()`
- `toContain()`
- `toBeLessThan()`, `toBeGreaterThan()`, `toBeCloseTo()`

Setting up and Tearing down



Introduction to Karma

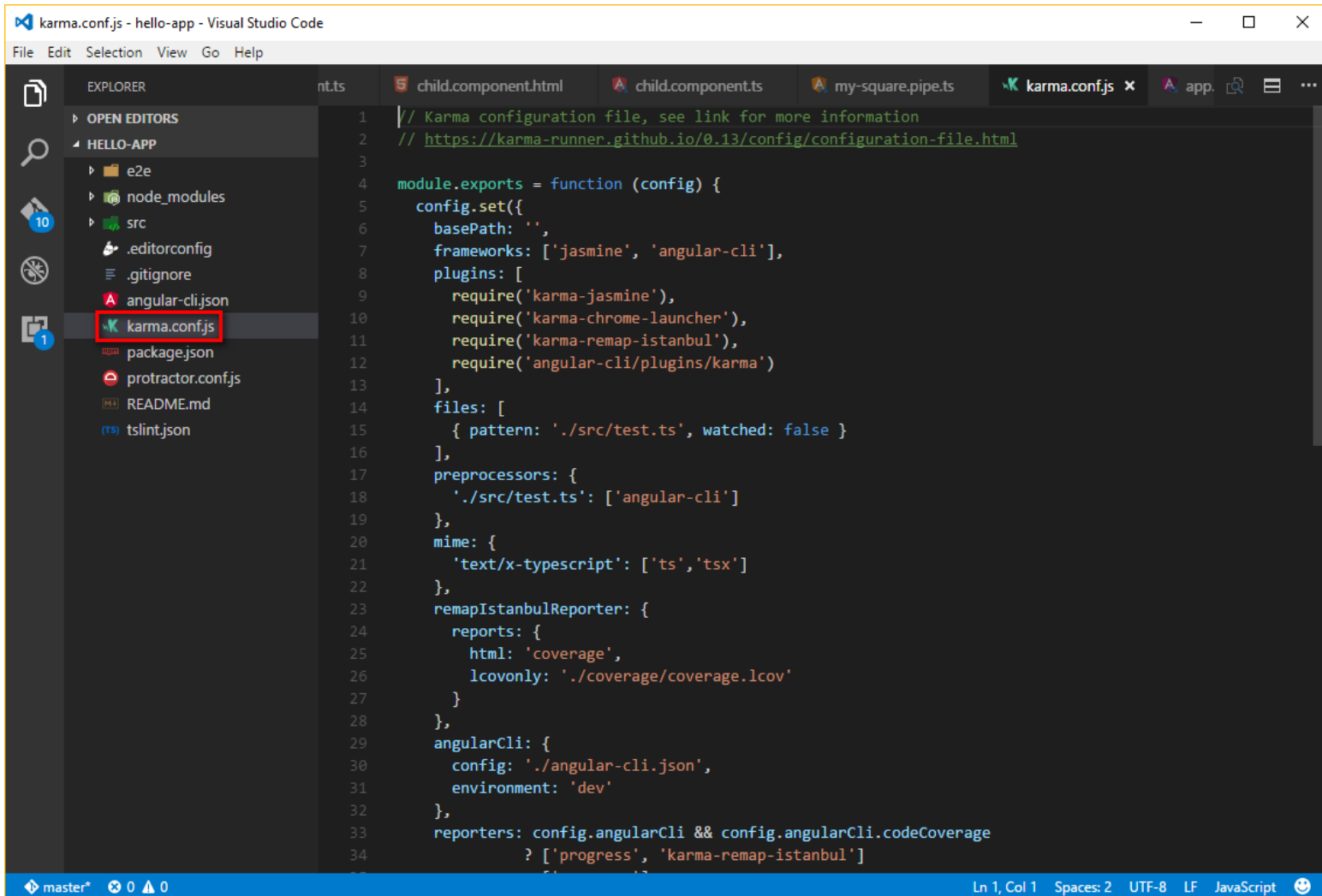
What is and Why Karma?

- JavaScript command line tool that can be used to spawn a web server which loads your application's source code and executes your tests
- Run against a number of browsers
 - > Chrome, Firefox, etc
- Display the results of your tests on the command line once they have run in the browser
 - > Or the result can be displayed within a tool (VSC/WebStorm)

Karma Installation & Configuration

- Karma is a NodeJS application, and should be installed through NodeJS package manager, *npm*
- Karma uses configuration file called *karma.conf.js*
- If you have created Angular 2 app using “angular-cli”, Karma gets installed and configured (with default configuration settings) automatically
- How to start unit testing using Karma for “angular-cli” generated application
 - > *ng test*

Karma Configuration File



The screenshot displays the Visual Studio Code interface with the 'karma.conf.js' file open in the editor. The Explorer sidebar on the left shows the project structure, with 'karma.conf.js' highlighted. The editor window shows the following code:

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/0.13/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', 'angular-cli'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-remap-istanbul'),
      require('angular-cli/plugins/karma')
    ],
    files: [
      { pattern: './src/test.ts', watched: false }
    ],
    preprocessors: {
      './src/test.ts': ['angular-cli']
    },
    mime: {
      'text/x-typescript': ['ts', 'tsx']
    },
    remapIstanbulReporter: {
      reports: {
        html: 'coverage',
        lcovonly: './coverage/coverage.lcov'
      }
    },
    angularCli: {
      config: './angular-cli.json',
      environment: 'dev'
    },
    reporters: config.angularCli && config.angularCli.codeCoverage
      ? ['progress', 'karma-remap-istanbul']
      : ['progress']
  });
};
```

The status bar at the bottom indicates the current file is 'karma.conf.js', located at line 1, column 1, using UTF-8 encoding with LF line endings.

Testing Components

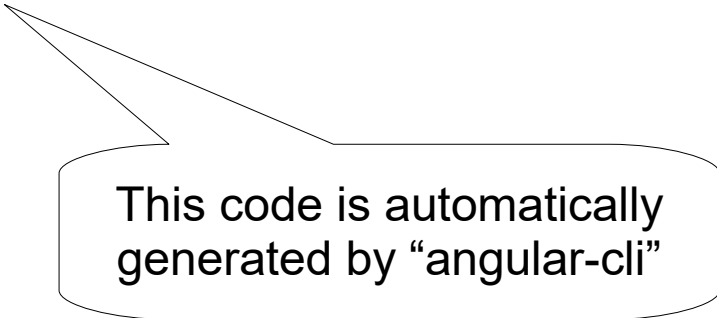
During Testing...

- The normal Angular bootstrap process does not occur
 - > Angular does not automatically load the module
 - > Angular does not automatically instantiate components either
- So we need to mimic the bootstrap process in our test code
 - > Load module
 - > Instantiate component (with their required dependencies)

Set things up before component testing

- Use *configureTestingModule* from *TestBed* to load application module

```
beforeEach(() => {  
  TestBed.configureTestingModule({  
    declarations: [  
      AppComponent  
    ],  
    providers: [ ],  
  });  
});
```



This code is automatically generated by “angular-cli”

Set things up before component testing

- Create test component

```
it('should have as title 'app works!', async() => {  
  let fixture = TestBed.createComponent(AppComponent);  
  let app = fixture.debugElement.componentInstance;  
  expect(app.title).toEqual('app works!');  
});
```

This code is automatically
generated by “angular-cli”

Write Test code

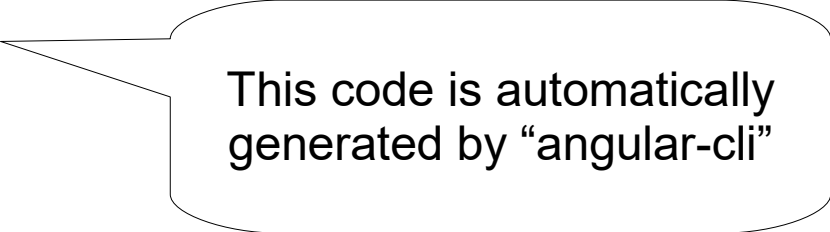
```
it('should have as title 'app works!', async() => {  
  let fixture = TestBed.createComponent(AppComponent);  
  let app = fixture.debugElement.componentInstance;  
  expect(app.title).toEqual('app works!');  
});
```

Testing Services

Set things up before Service testing

- Use *configureTestingModule* from *TestBed* to load application module with service provider

```
beforeEach(() => {  
  TestBed.configureTestingModule({  
    providers: [LogService]  
  });  
});
```



This code is automatically generated by “angular-cli”

```
it('should ...', inject([LogService], (service: LogService) => {  
  expect(service).toBeTruthy();  
}));
```

Code with Passion!
JPassion.com

