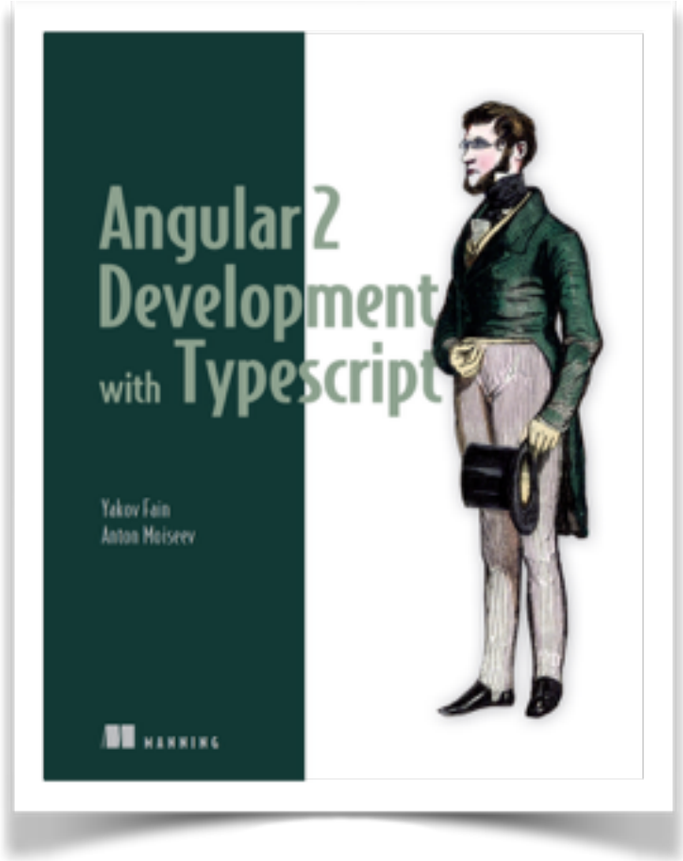
# Angular 2 Tooling

Yakov Fain, Farata Systems

@yfain

# About myself

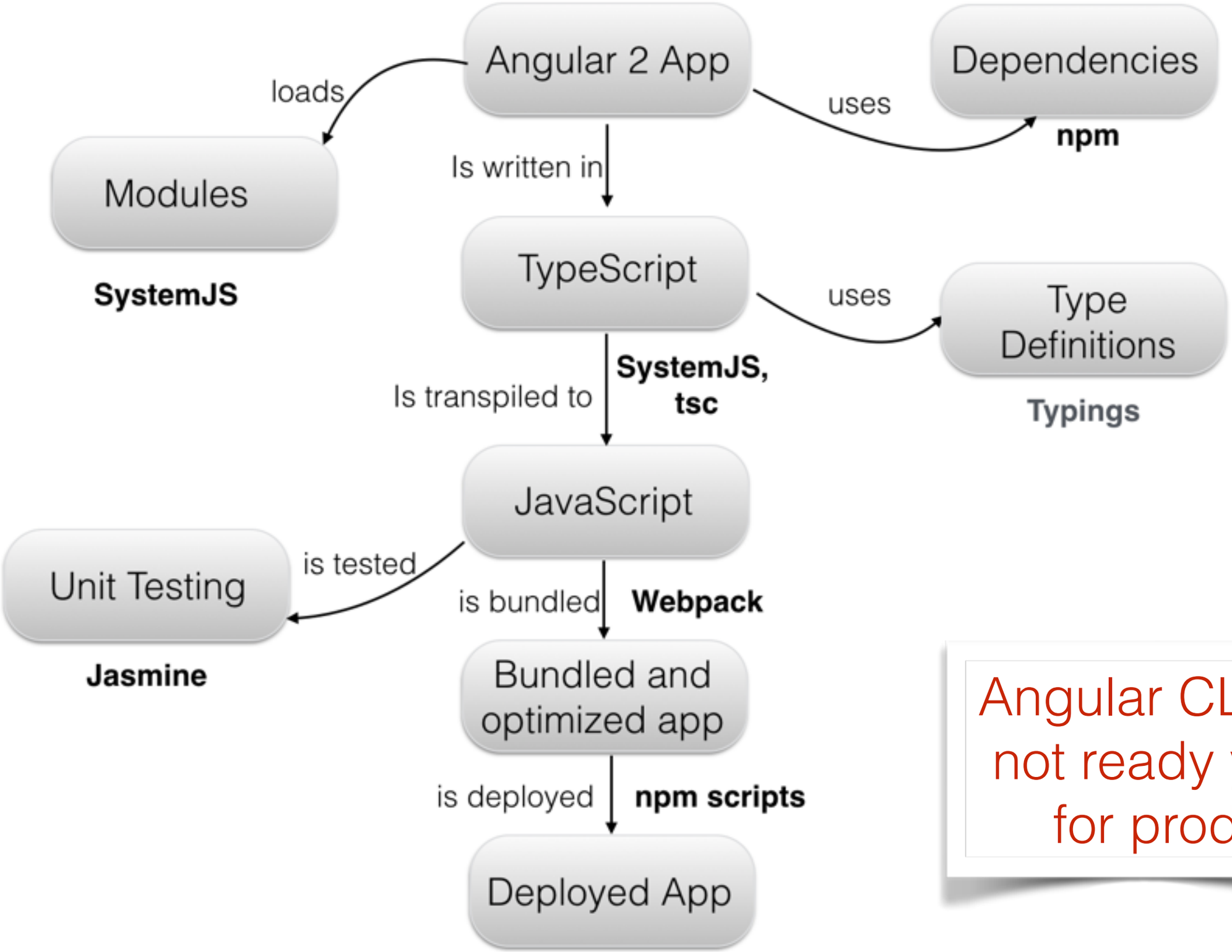
- Co-founder of two companies:
  - Farata Systems
  - SuranceBay
- Java Champion
- Recently co-authored the book Angular Development with TypeScript



# The Agenda

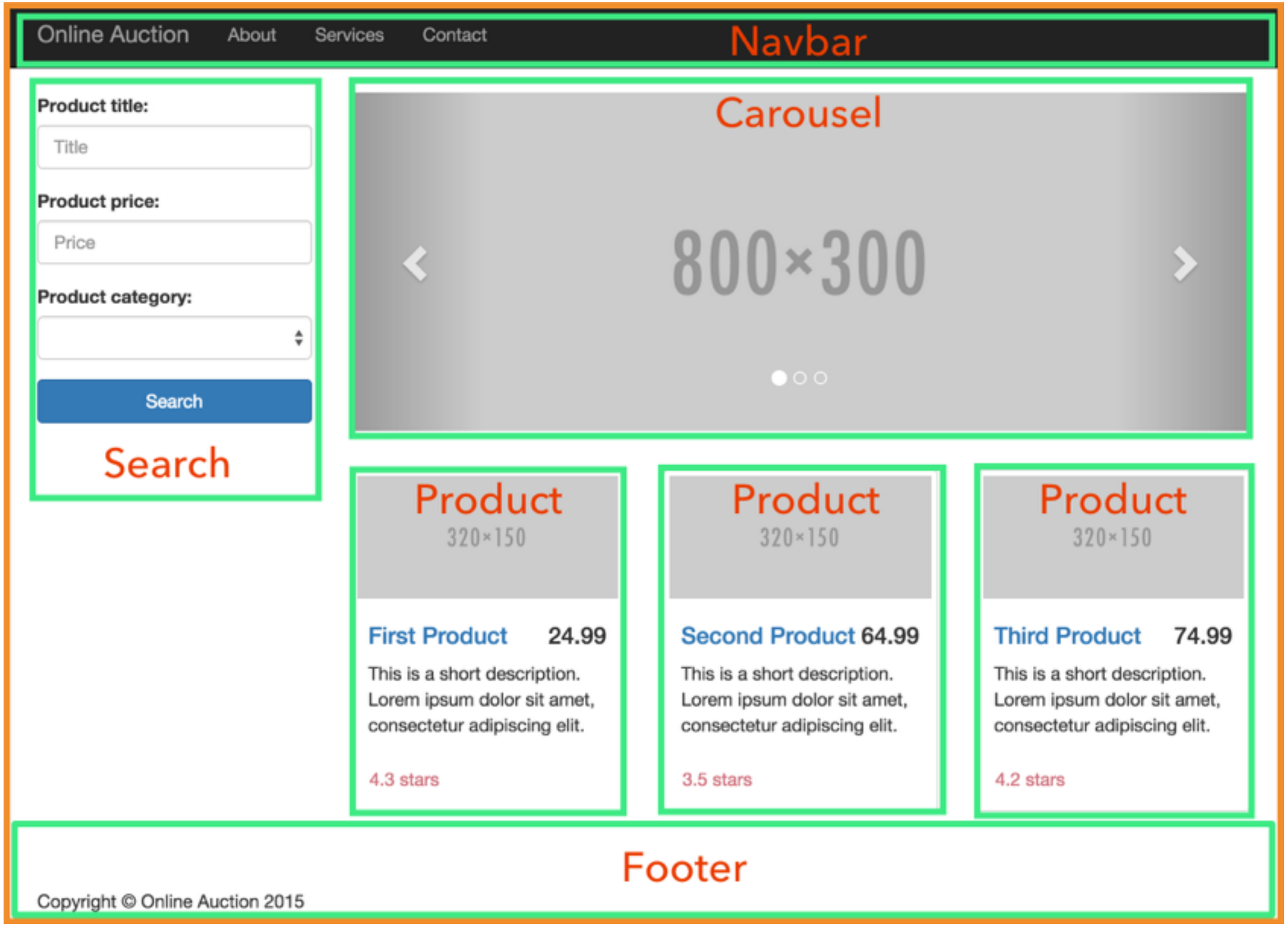
- Configuring an Angular 2 project
- Module loader SystemJS
- Type definition files and Typings manager
- Unit Testing
- Build automation with Webpack

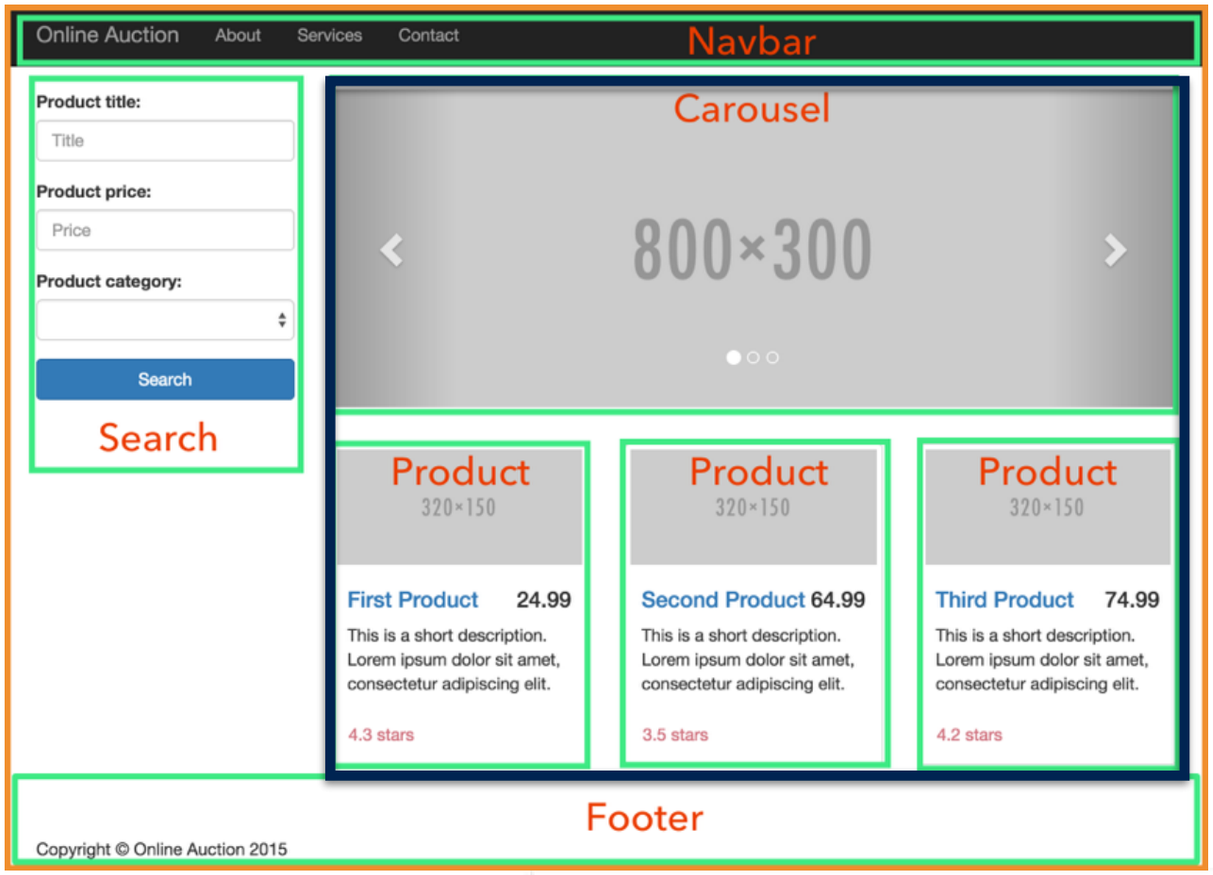
# The tools that can be used today




Angular CLI is not ready yet for prod

# An app is a tree of components

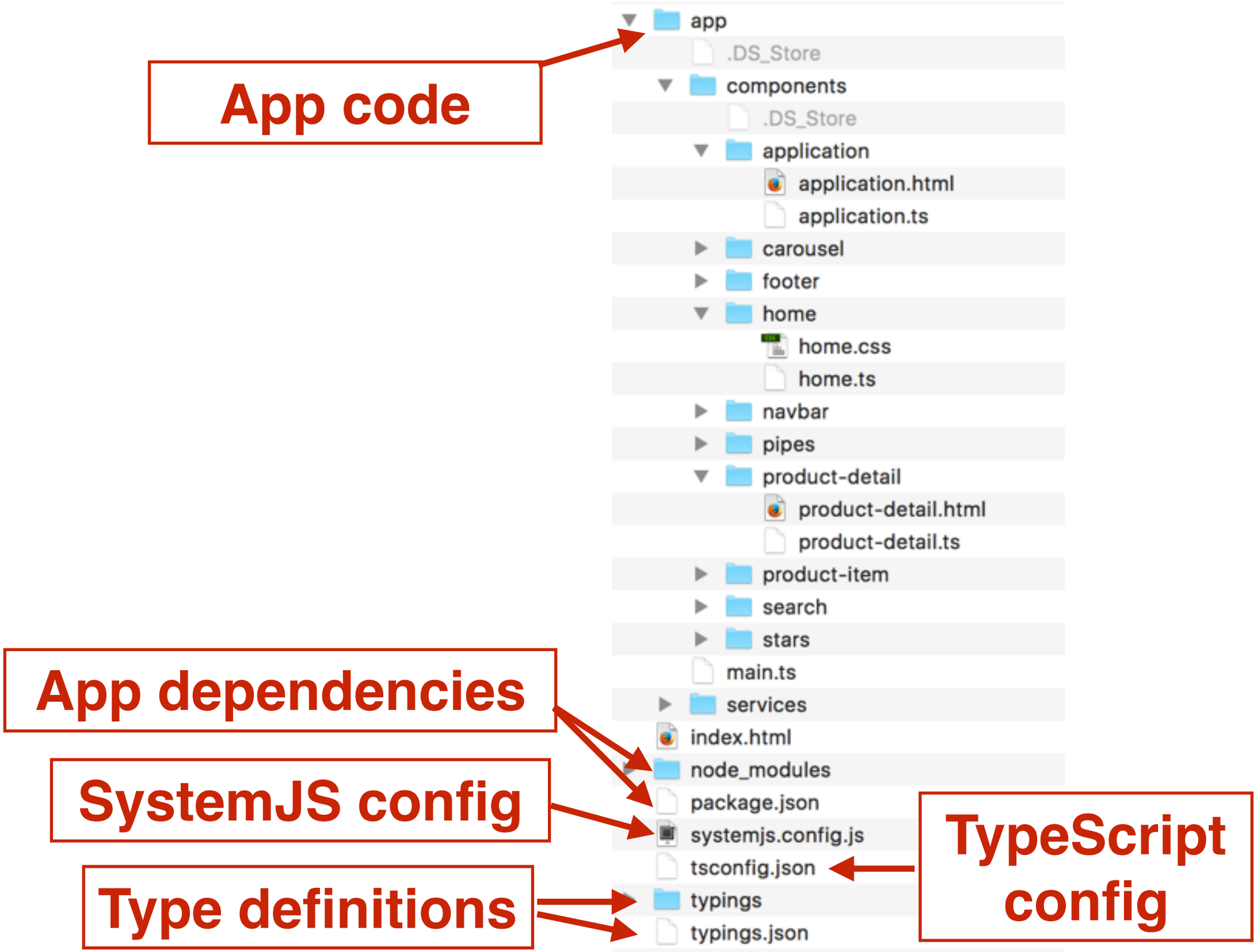





SystemJS loads components

```
import {Component} from '@angular/core';
import { Routes, ROUTER_DIRECTIVES} from '@angular/router';

import HomeComponent from '../home/home';
import NavbarComponent from '../navbar/navbar';
import FooterComponent from '../footer/footer';
import SearchComponent from '../search/search';
import ProductDetailComponent from "../product-detail/product-detail";

@Component({…})
@Routes([…])
export default class ApplicationComponent {…}
```

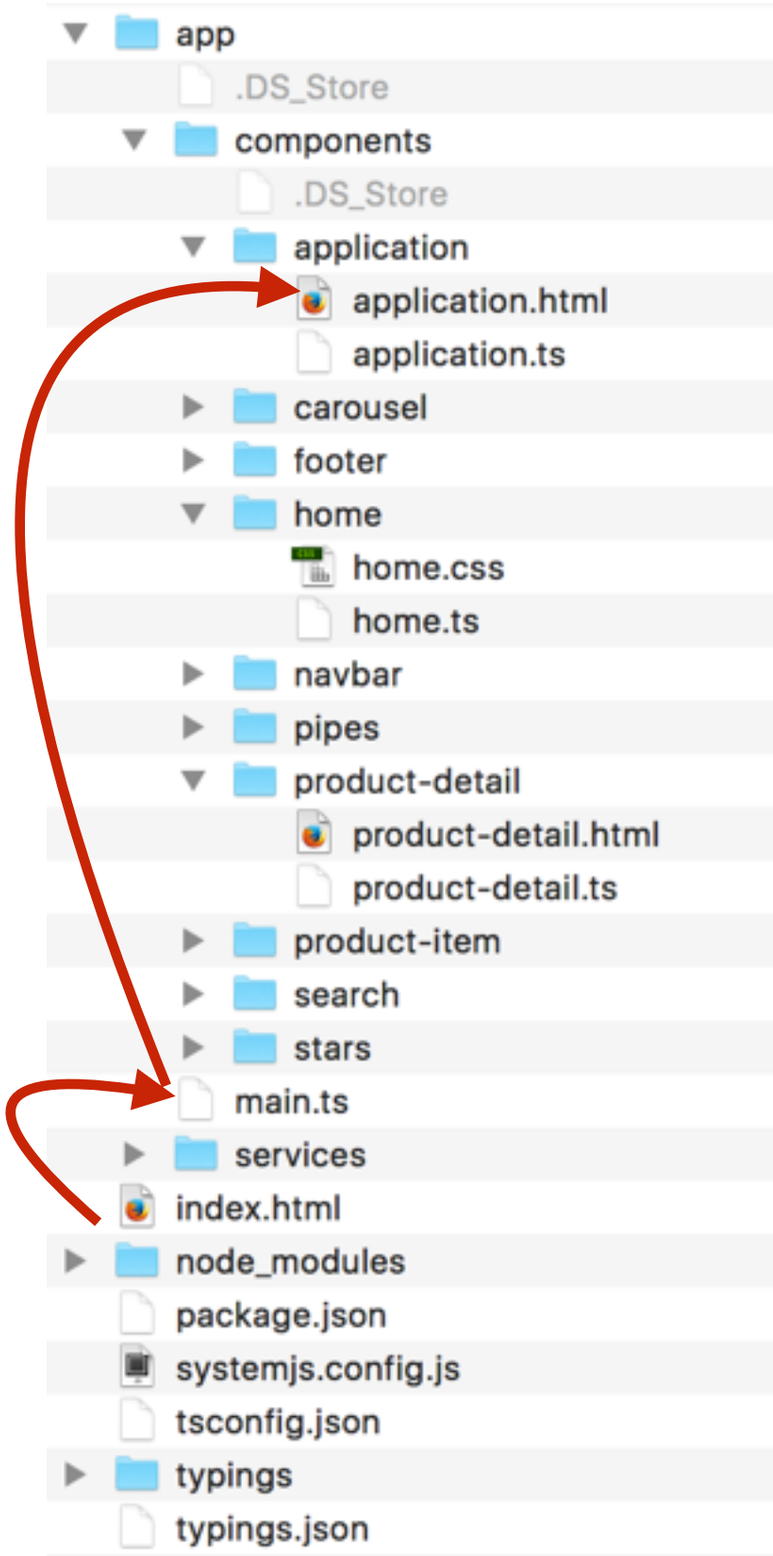
# Sample Project Structure



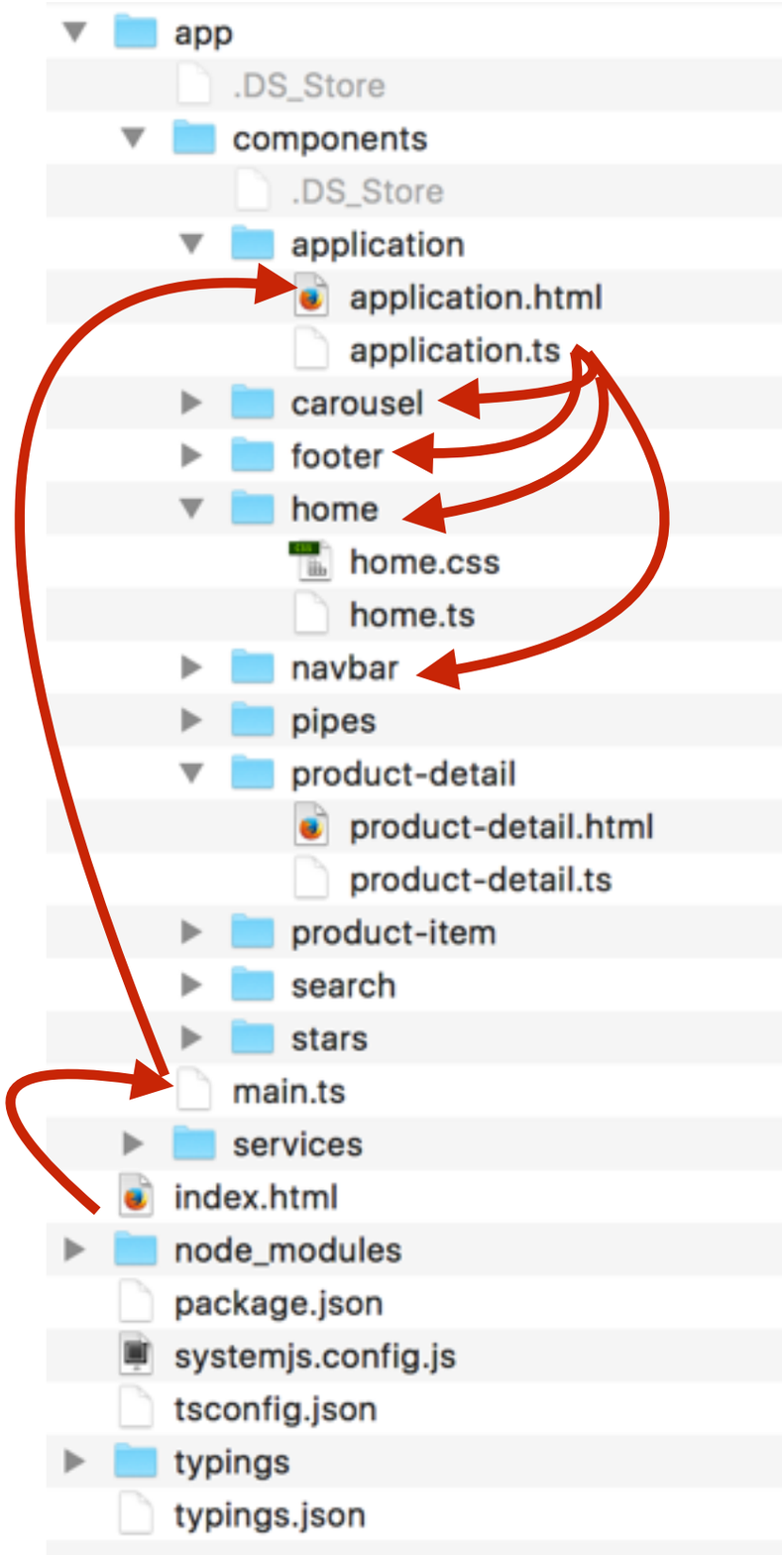
# Sample Project Structure

**SystemJS transpiles TS and loads JS modules**

```
bootstrap(ApplicationComponent)
```



# Sample Project Structure




```
bootstrap(ApplicationComponent)
```

# The npm package manager

- Package managers: npm, jspm, bower…
- npm comes with Node.js
- npmjs.org is a repository of 250K+ packages

# Installing npm packages

- To install a package xyz  globally:

  npm install xyz -g

- App dependencies are configured in the file **package.json**

- Local install of xyz in your project's dir **node_modules**:

  npm install xyz

# package.json (cont.)

- package.json is for reproduceable builds
- Install **xyz** and add it to **dependencies** section:
  npm install xyz —save
- Install **xyz** with adding it to **devDependencies** section:
  npm install xyz —save-dev

# package.json

npm docs:
https://docs.npmjs.com

To run from cmd window

App dependencies

Dev dependencies

```
{
  "name": "test_samples",
  "description": "A sample Weather app",
  "private": true,
  "scripts": {
    "start": "live-server",
    "test": "karma start karma.conf.js"
  },
  "dependencies": {
    "@angular/common": "2.0.0-rc.1",
    "@angular/compiler": "2.0.0-rc.1",
    "@angular/core": "2.0.0-rc.1",
    "@angular/http": "2.0.0-rc.1",
    "@angular/platform-browser": "2.0.0-rc.1",
    "@angular/platform-browser-dynamic": "2.0.0-rc.1",
    "@angular/router": "2.0.0-rc.1",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.6",
    "systemjs": "^0.19.27",
    "zone.js": "^0.6.12"
  },
  "devDependencies": {
    "jasmine-core": "^2.4.1",
    "karma": "^0.13.22",
    "karma-chrome-launcher": "^0.2.3",
    "karma-firefox-launcher": "^0.1.7",
    "karma-jasmine": "^0.3.8",
    "live-server": "0.8.2",
    "typescript": "^1.8.10"
  }
}
```

# Starting a new Angular project with npm

1. Generate *package.json* for your project:
   npm init -y

2. Add dependencies to *package.json*

3. Download dependencies into the dir *node_modules*:
   npm install

4. Install live-server locally:
   npm install live-server —save-dev

# index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Basic Routing Samples</title>
  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/typescript/lib/typescript.js"></script>
  <script src="node_modules/reflect-metadata/Reflect.js"></script>
  <script src“node_modules/systemjs/dist/system.src.js"></script>

  <script src“systemjs.config.js"></script>

  <script>
    System.import(‘app')
          .catch(function (err) {console.error(err);});
  </script>
</head>
<body>
  <my-app></my-app>
</body>
</html>
```

# systemjs.config.js

```
System.config({
    transpiler: 'typescript',
    typescriptOptions: {emitDecoratorMetadata: true},
    map: {
        'app' : 'app',
        'rxjs': 'node_modules/rxjs',
        '@angular'                    : 'node_modules/@angular',

     packages: {
        'app'                            : {main: 'main.ts', defaultExtension: 'ts'},
        'rxjs'                           : {main: 'index.js'},
        '@angular/core'                  : {main: 'index.js'},
        '@angular/common'                : {main: 'index.js'},
        '@angular/compiler'              : {main: 'index.js'},
        '@angular/router'                : {main: 'index.js'},
        '@angular/platform-browser'      : {main: 'index.js'},
        '@angular/platform-browser-dynamic': {main: 'index.js'},
        '@angular/http'                  : {main: 'index.js'}

      }
});
```

# Demo

CH9: test_weather
npm start

https://github.com/Farata/angular2typescript

# Type Definitions

# Type definitions files

- Type definition files (***.d.ts**) declare types for JavaScript or TypeScript libraries and frameworks
- ***.d.ts** files help IDE with type-ahead help
- TypeScript Static Analyzer uses ***.d.ts** files to report errors

# Type Definitions (cont.)

- **ambient** types are defined with the keyword `declare`. Thry're not linked it to the actual JS code.
- http://definitelytyped.org has the largest collection of type definitions
- Type definition manager installs ***.d.ts** files in the dir **typings**

# Type Definitions (end)

- **You could** refer a d.ts file in your TypeScript files:

```
/// <reference path="typings/jquery.d.ts" />
```

- **But** tsc searches for all d.ts files in subdirs starting from the root dir specified as `rootDir` in tsconfig.json

- An IDE shows a type in red, if it can't find the definition for this type. Install the required d.ts file.

# Type Definition Managers

- **tsd** is deprecated
  Installs tyoe definitions files only from definitelytyped.org

- **Typings**
  Installs definitions from various repositories

- To install Typings globally:

```
npm install -g typings
```

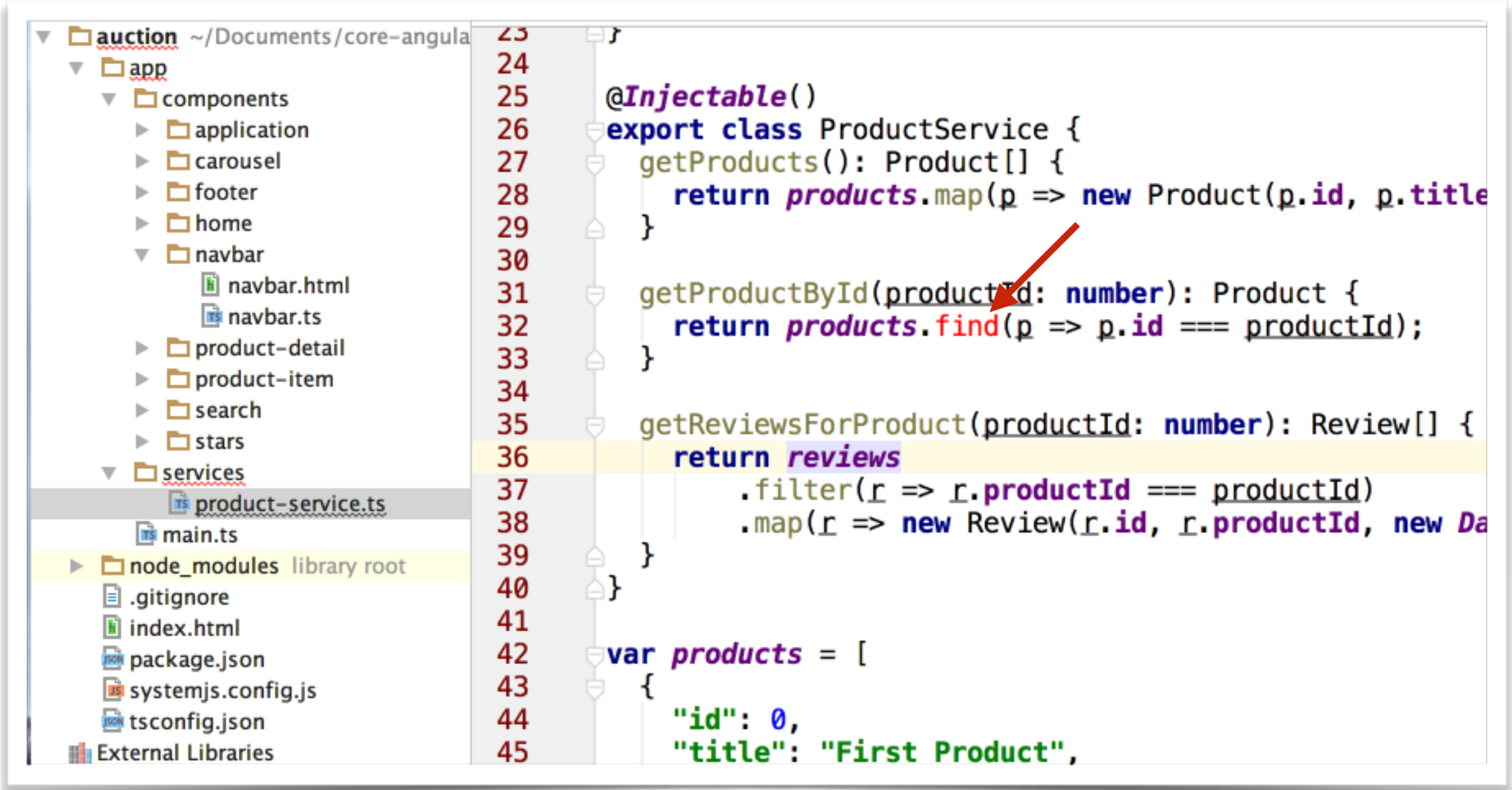
# How to install definitions for a JS lib

- To install jQuery type definitions and save config in typings.json:

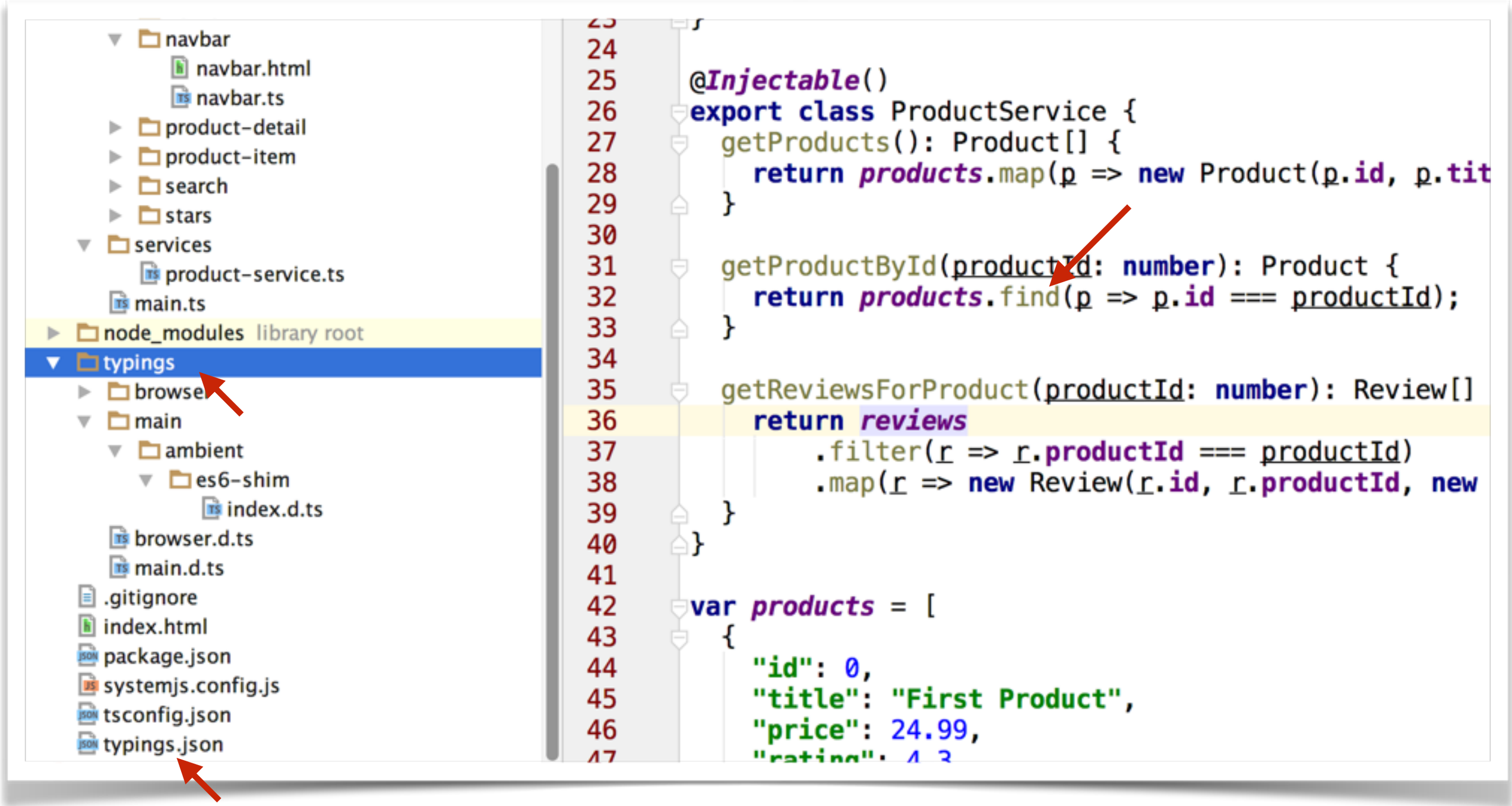
```
typings install jquery --ambient —save
```

- Without —ambient Typings will try to find *.d.ts in its own registry https://github.com/typings/registry

# tsc target ES5, no type definitions for ES6



# typings install es6-shim --ambient --save



# How tsc finds node modules

We set **"module": "commonjs"** in tsconfig.json and tsc applies the Node strategy:

- Look for ambient declarations (`import {Component} from 'angular2/core'`)

- Check if the path is relative (`import {Product} from './product.service'`)

  - Look for **`product.service.ts`** file

  - Look for **`product.service.d.ts`** file

  - If not found, look in the dir **`node_modules`**

# Unit Testing

# Types of testing for Angular apps

- **Unit testing** asserts that a small unit of code accepts the expected data and returns the expected result.

  Jasmine

- **End-to-end** testing asserts that the entire application works as expected.

  Protractor

# Installing Jasmine and type definitions

- npm install jasmine-core --save-dev
- typings install jasmine --ambient —save-dev
- typings install es6-shim --ambient --save-dev

# Jasmine spec files

- TypeScript test files have name extensions **.spec.ts**
- A test (a.k.a. spec) is written in the function `it()`
- A test suite is written in `describe() that` **includes one or more** `it()`

```
import ApplicationComponent from './application';

describe('ApplicationComponent', () => {

  it('is successfully instantiated', () => {

    const app = new ApplicationComponent();

    expect(app instanceof ApplicationComponent).toEqual(true);
  });
});
```

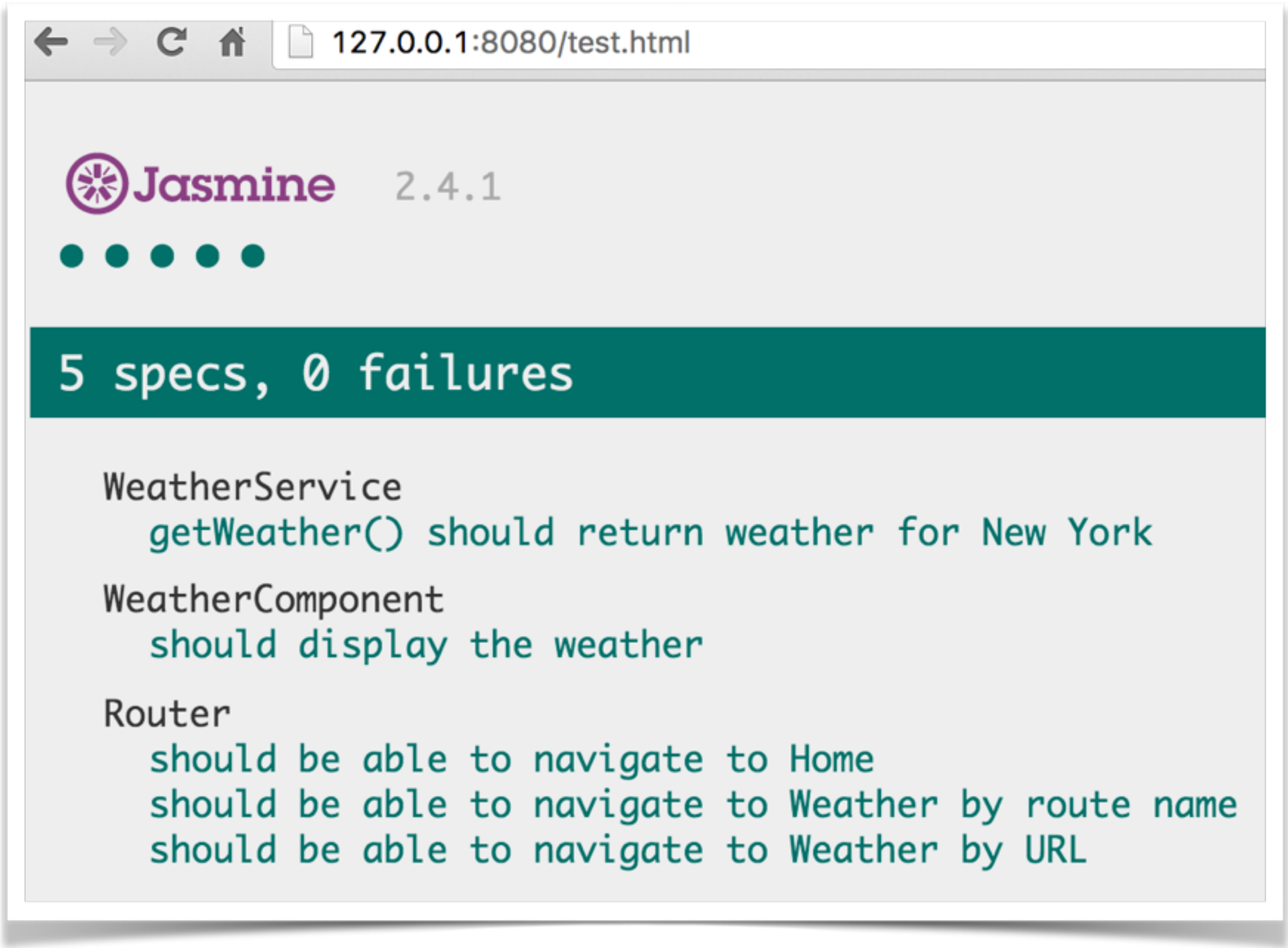
Matcher

# A fragment from HTML runner

```
<body>
<script>
  var SPEC_MODULES = [
    'app/components/app.spec',
    'app/components/weather.spec',
    'app/services/weather.service.spec'
  ];

  Promise.all([
    System.import('@angular/core/testing'),
    System.import('@angular/platform-browser-dynamic/testing')
  ])
  .then(function (modules) {
    var testing = modules[0];
    var browser = modules[1];

    testing.setBaseTestProviders(
      browser.TEST_BROWSER_DYNAMIC_PLATFORM_PROVIDERS,
      browser.TEST_BROWSER_DYNAMIC_APPLICATION_PROVIDERS);

    // Load all the spec files.
    return Promise.all(SPEC_MODULES.map(function (module) {
      return System.import(module);
    }));
  })
  .then(window.onload)
  .catch(console.error.bind(console));
</script>
</body>
```

App specs

# The output of Jasmine HTML Runner



# Setup and teardown phases

- **Setup**. Write the code to execute before each test or suite in `beforeAll()` and `beforeEach()`.
- **Teardown.** Write the code to execute after each test or suite in `afterAll()` and `afterEach()`.

# Angular testing lib

- Includes wrappers for Jasmine's functions `describe(), it()` et al.
- Adds new functions: `beforeEachProviders(), inject(), async(), fakeAsync()` and more.
- Adds new matchers: `toBePromise(), toBeAnInstanceOf()` et al.

Jasmine matchers: https://github.com/JamieMason/Jasmine-Matchers

# Demo

CH9: test_weather

1. npm start
2. localhost:8080/test.html

# inject()

```
beforeEachProviders(() => [
  MockBackend,
  BaseRequestOptions,
  provide(Http, {
    useFactory: (backend, options) => new Http(backend, options),

    deps: [MockBackend, BaseRequestOptions]
  }),
  WeatherService]);


beforeEach(inject([WeatherService], (service) => {
 // the setup code goes here
}));
```

# async()

async() runs in the zone and will complete only after its code is done executing

```
it(' does something', async(inject([AClass], object => {

  myPromise.then(() => {
                    expect(true).toEqual(true); });

}), 3000));
```

# fakeAsync()

Speeds up the testing of asynchronous services by simulation the passage of time

```
it('should be able to navigate to weather using commands API',

  fakeAsync(inject([Router, Location], (router: Router, location: Location) => {

    router.navigate(['/weather']);

    tick(1000);

    expect(location.path()).toBe('/weather');

  })
));
```

# Testing Services

1. Define providers

2. Inject services

```
beforeEachProviders(() => [
  MockBackend,
  BaseRequestOptions,
  WeatherService
  { provide: Http,
    useFactory: (backend, options) => new Http(backend, options),
    deps: [MockBackend, BaseRequestOptions]
  }
]);

beforeEach(inject([MockBackend, WeatherService], (_mockBackend, _service) => {
  mockBackend = _mockBackend;
  service = _service;
}));
```

# Testing Services (cont.)

3. Write the it() block

```
it('getWeather() should return weather for New York', async(() => {
    let mockResponseData = {
      cod: '200',
      list: [{
        name: 'New York',
        main: {
          temp: 57,
          humidity: 44
        }
      }]
    };

    mockBackend.connections.subscribe((connection: MockConnection) => {
      let responseOpts = new ResponseOptions({body:
                                        JSON.stringify(mockResponseData)});
      connection.mockRespond(new Response(responseOpts));
    });

    service.getWeather('New York').subscribe(weather => {
      expect(weather.place).toBe('New York');
      expect(weather.humidity).toBe(44);
      expect(weather.temperature).toBe(57);
    });
  }));
});
```

# Testing Components

- `TestComponentBuilder` has methods that returns `ComponentFixture`

- `ComponentFixture` has references to both the component and the native HTML element.

- Via the fixture you acccess component's properties and find specific HTML elements within the component's template.

```
let testComponentBuilder: TestComponentBuilder;

beforeEachProviders(() => [TestComponentBuilder]);

beforeEach(inject([TestComponentBuilder], (tcb) => {
  testComponentBuilder = tcb;
}));
```

# Testing Components

- `TestComponentBuilder` has methods that returns `ComponentFixture`
- `ComponentFixture` has references to both the component and the native HTML element.
- Via the fixture you acccess component's properties and find specific HTML elements within the component's template.

# Testing Components (cont.)

1. Define providers including TestComponentBuilder

2. Inject TestComponentBuilder and services

```
let component: WeatherComponent;
let testComponentBuilder: TestComponentBuilder;

beforeEachProviders(() => [
  TestComponentBuilder,
  WeatherComponent,
  {provide: WeatherService, useClass: MockWeatherService}
]);

beforeEach(inject([TestComponentBuilder, WeatherComponent], (tcb, cmp) => {
  testComponentBuilder = tcb;
  component = cmp;
}));

// the it() blocks go here
```

# Testing Components (cont.)

3. Write the it() block and invoke `detectChanges()` on the fixture.

4. Run `expect()` to check the rendered values

```
it('should display the weather ', fakeAsync(() => {
  let fixture = testComponentBuilder.createFakeAsync(WeatherComponent);
  let element = fixture.nativeElement;
  let component = fixture.componentInstance;
  component.weather = {place: 'New York', humidity: 44, temperature: 57};

  fixture.detectChanges();

  expect(element.querySelector('h3').innerHTML).toBe('Current weather in New York:');
  expect(element.querySelector('li:nth-of-type(1)').innerHTML).toBe('Temperature: 57F');
  expect(element.querySelector('li:nth-of-type(2)').innerHTML).toBe('Humidity: 44%');
}));
```

# Testing Component Router

- beforeEachProviders(() => [ROUTER_FAKE_PROVIDERS]);
- Use router's methods `navigate()` or `navigateByURL()`
- In `navigate()` specify the path of the configured route
- In `navigateByURL()` specify the URL segment

# Testing Component Router (cont.)

```
describe('Router', () => {

  beforeEachProviders(() => [ROUTER_FAKE_PROVIDERS]);

  it('should be able to navigate to home using commands API',
    fakeAsync(inject([Router, Location], (router: Router, location: Location) => {
      router.navigate(['/']);
      tick();
      expect(location.path()).toBe('');
    }))
  ));

  it('should be able to navigate to weather using commands API',
    fakeAsync(inject([Router, Location], (router: Router, location: Location) => {
      router.navigate(['/weather']);
      tick();
      expect(location.path()).toBe('/weather');
    }))
  ));
));
```

# Karma test runner

- Karma runs from the console
- Karma can run tests against multiple browsers

To install Karma:

```
npm install karma karma-jasmine karma-chrome-launcher karma-firefox-launcher --save-dev
```

# Karma config file: karma.conf.js

```
module.exports = function (config) {
  config.set({
    browsers: ['Chrome', 'Firefox'],
    frameworks: ['jasmine'],
    reporters: ['dots'],
    singleRun: true,

    files: [
      'node_modules/typescript/lib/typescript.js',
      'node_modules/reflect-metadata/Reflect.js',
      'node_modules/systemjs/dist/system.src.js',
      'node_modules/zone.js/dist/zone.js',
      'node_modules/zone.js/dist/async-test.js',
      'node_modules/zone.js/dist/fake-async-test.js',

      {pattern: 'node_modules/rxjs/**/*.js',         included: false, watched: false},
      {pattern: 'node_modules/rxjs/**/*.js.map',     included: false, watched: false},
      {pattern: 'node_modules/@angular/**/*.js',     included: false, watched: false},
      {pattern: 'node_modules/@angular/**/*.js.map', included: false, watched: false},

      {pattern: 'karma-systemjs.config.js',          included: true,  watched: false},
      {pattern: 'karma-test-runner.js',              included: true,  watched: false},

      // Application
      {pattern: 'app/**/*.ts', included: false, watched: true}
    ],
    proxies: {
      '/app/': '/base/app/'
    },
    plugins: [
      'karma-jasmine',
      'karma-chrome-launcher',
      'karma-firefox-launcher'
    ]
  })
};
```

# Karma script to run tests: karma-test-runner.js

```
Error.stackTraceLimit = Infinity;

jasmine.DEFAULT_TIMEOUT_INTERVAL = 1000;

__karma__.loaded = function () {};

function resolveTestFiles() {
  return Object.keys(window.__karma__.files)
      .filter(function (path) { return /\.spec\.ts$/.test(path); })
      .map(function (moduleName) { return System.import(moduleName); });
}

Promise.
  all([
    System.import('@angular/core/testing'),
    System.import('@angular/platform-browser-dynamic/testing')
  ]).
  then(function (modules) {
    var testing = modules[0];
    var browser = modules[1];

    testing.setBaseTestProviders(
        browser.TEST_BROWSER_DYNAMIC_PLATFORM_PROVIDERS,
        browser.TEST_BROWSER_DYNAMIC_APPLICATION_PROVIDERS);
  }).
  then(function () { return Promise.all(resolveTestFiles()); }).
  then(function () { __karma__.start(); },
       function (error) { __karma__.error(error.stack || error); });
```

# Demo

## Chapter 9, test_weather app

npm test

# Building and Deploying Apps with Webpack

# Objectives

- A SystemJS-based project makes too many requests and downloads megabytes

We want:

- Minimize the number of requests
- Reduce the download size
- Automate the build in dev and prod

# Module Loaders

1. Traverse the tree of JS modules starting from an endpoint
2. Work with multiple module formats (ES6, CommonJS, AMD, UMD, global)
3. Can pre-process loaded resources (transpile, minimize, etc.)
4. Can work with different resources
5. Can load modules in the browser, can prepare bundles “offline”
6. Can perform “dead code elimination”

# Webpack bundler

1. Gained popularity after it was adopted by Instagram
2. It’s a powerful and production-ready tool
3. The config file is compact
4. Has its own loader (doesn’t use SystemJS)

# Webpack Hello World

main.js

```
document.write('Hello World!');
```

Create a bundle:

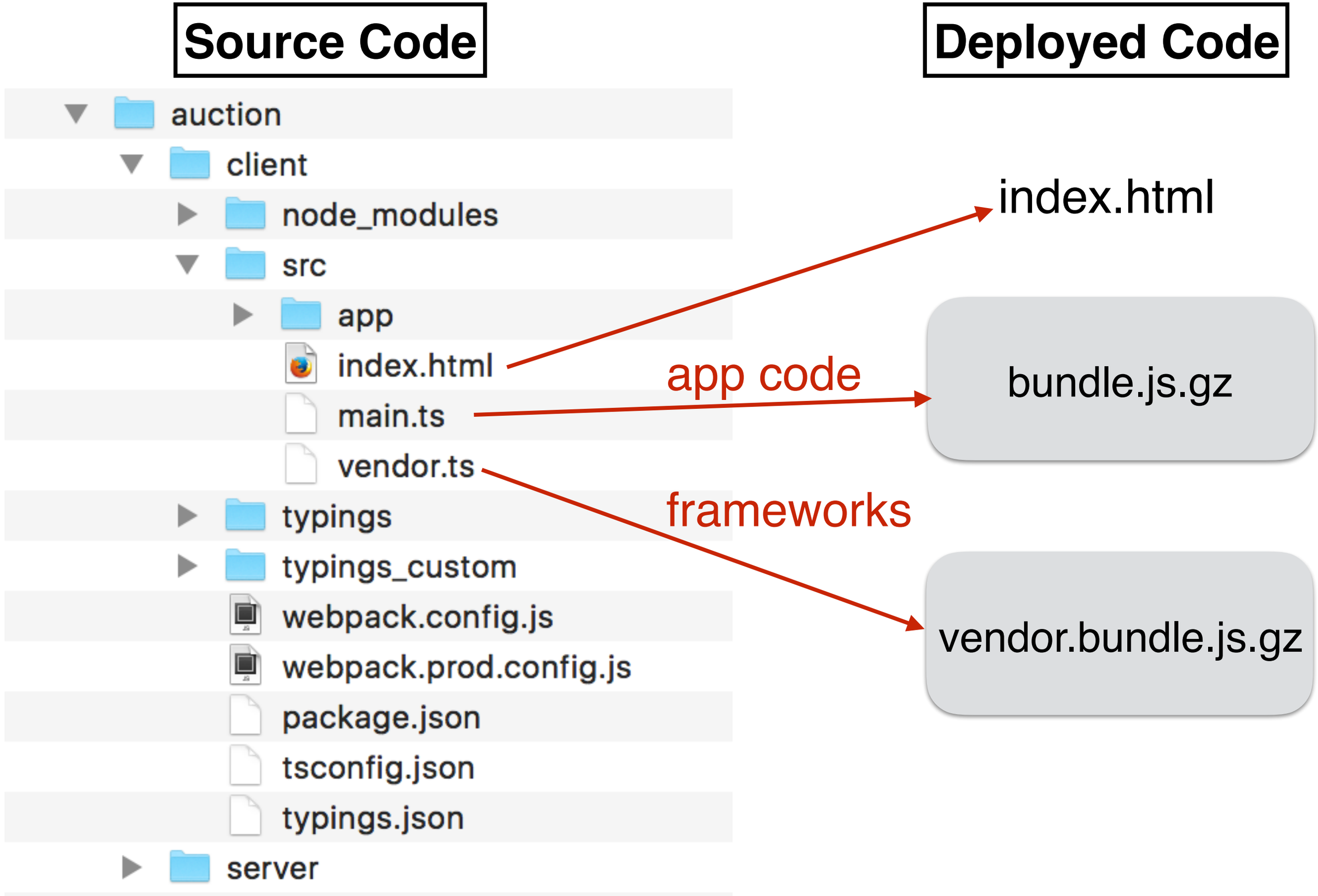
```
webpack main.js bundle.js
```

webpack.config.js

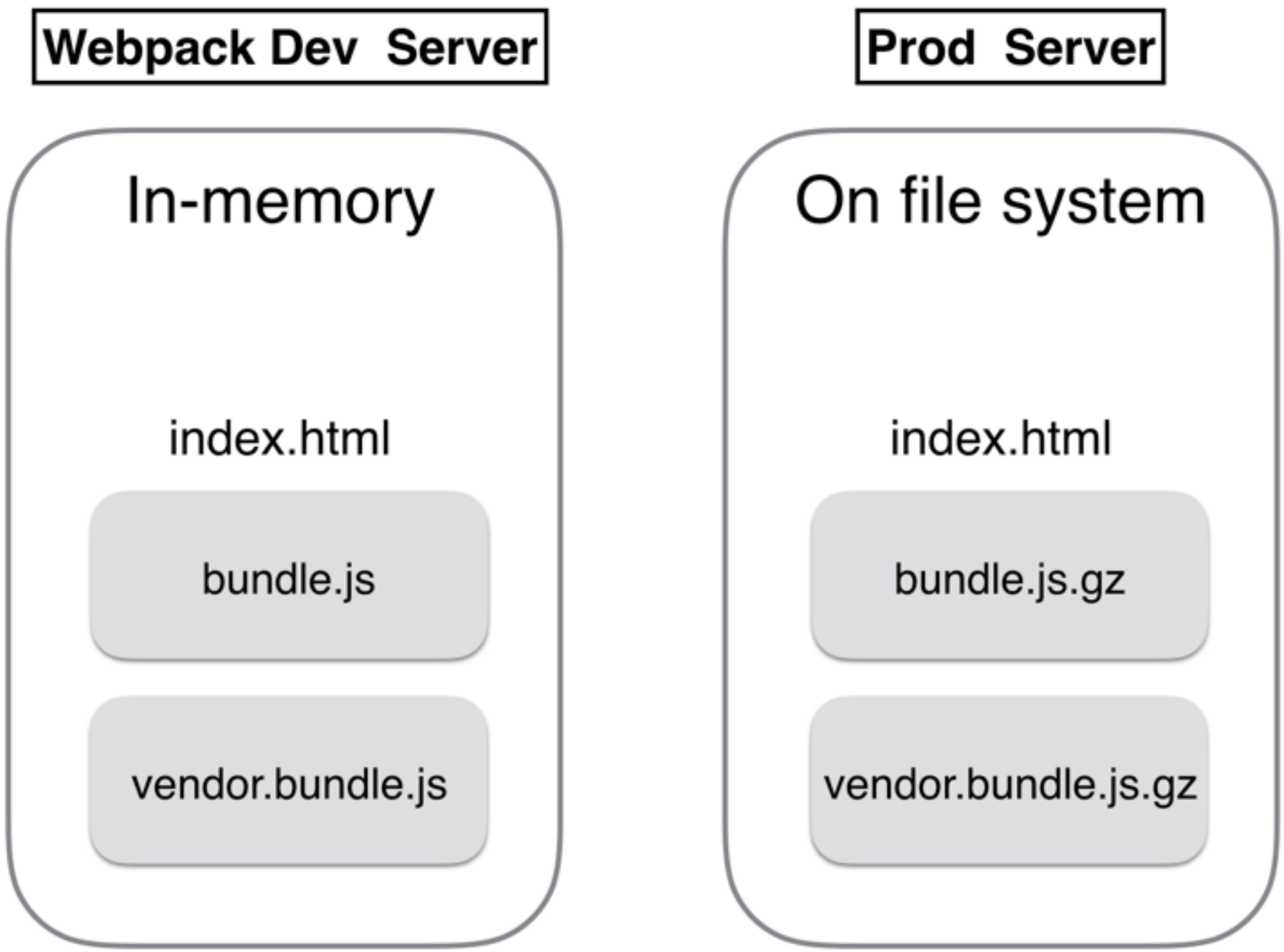
```
module.exports = {
  entry: './main',
  output: {
    path: './dist',
    filename: 'bundle.js'
  },
  watch: true,
  devServer: {
    contentBase: '.'
  }
};
```

index.html

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <script src="/dist/bundle.js"></script>
</body>
</html>
```


Source Code
Deployed Code
auction
client
node_modules
src
app
index.html
main.ts
vendor.ts
typings
typings_custom
webpack.config.js
webpack.prod.config.js
package.json
tsconfig.json
typings.json
server
index.html
app code
bundle.js.gz
frameworks
vendor.bundle.js.gz

# Webpack Dev Server



# webpack.prod.config

```
const path = require('path');

const CommonsChunkPlugin   = require('webpack/lib/optimize/CommonsChunkPlugin');
const CompressionPlugin    = require('compression-webpack-plugin');
const CopyWebpackPlugin    = require('copy-webpack-plugin');
const DedupePlugin         = require('webpack/lib/optimize/DedupePlugin');
const DefinePlugin          = require('webpack/lib/DefinePlugin');
const OccurenceOrderPlugin = require('webpack/lib/optimize/OccurenceOrderPlugin');
const UglifyJsPlugin       = require('webpack/lib/optimize/UglifyJsPlugin');

const ENV = process.env.NODE_ENV = 'production';
const metadata = {
  env: ENV
};

module.exports = {
  debug: false,
  devtool: 'source-map',
  entry: {
    'main'  : './src/main.ts',
    'vendor': './src/vendor.ts'
  },
  metadata: metadata,
  module: {
    loaders: [
      {test: /\.css$/,  loader: 'to-string!css', exclude: /node_modules/},
      {test: /\.css$/,  loader: 'style!css', exclude: /src/},
      {test: /\.html$/, loader: 'raw'},
      {test: /\.ts$/,   loader: 'ts', query: {compilerOptions: {noEmit: false}}}
    ]
  },
  output: {
    path    : './dist',
    filename: 'bundle.js'
  },
  plugins: [
    new CommonsChunkPlugin({name: 'vendor', filename: 'vendor.bundle.js', minChunks: Infinity}),
    new CompressionPlugin({regExp: /\.css$|\.html$|\.js$|\.map$/}),
    new CopyWebpackPlugin([{from: './src/index.html', to: 'index.html'}]),
    new DedupePlugin(),
    new DefinePlugin({'webpack': {'ENV': JSON.stringify(metadata.env)}}),
    new OccurenceOrderPlugin(true),
    new UglifyJsPlugin({
      compress: {screw_ie8 : true},
      mangle:  {screw_ie8 : true}
    })
  ],
  resolve: {extensions: ['', '.ts', '.js']}
};
```

# index.html after Webpack build

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset=UTF-8>
  <title>Angular Webpack Starter</title>
  <base href="/">
</head>
<body>
  <my-app>Loading...</my-app>
  <script src="vendor.bundle.js"></script>
  <script src="bundle.js"></script>
</body>
</html>
```

# Demo

Chapter 10, angular2-webpack-starter

https://github.com/Farata/angular2typescript/tree/master/chapter10/angular2-webpack-starter

For more comprehensive Angular/Webpack starter see
https://github.com/AngularClass/angular2-webpack-starter

# Thank you!

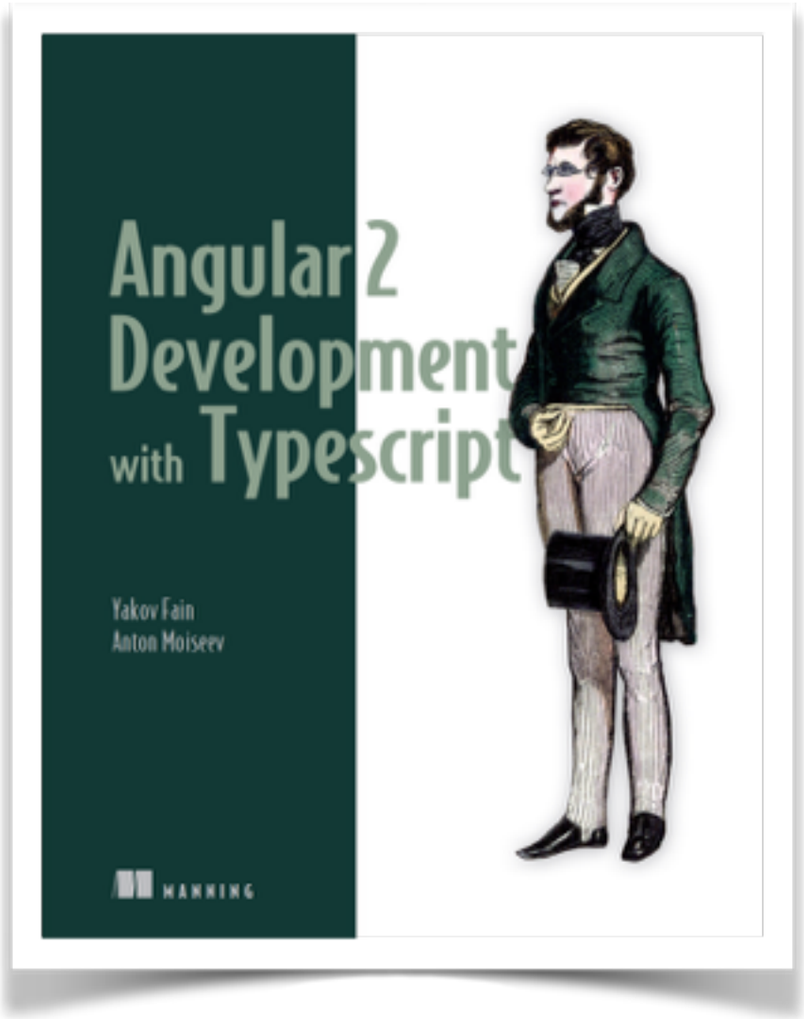
- Book code samples:

  https://github.com/Farata/angular2typescript

- Our company: faratasystems.com

- Blog: yakovfain.com

- Twitter: @yfain




discount code: faindz