Angular

Day 2 Overview

- Angular CLI
  - Angular CLI Capabilities
  - Project Generation
  - To Serve a Project
  - Component Generation
  - Service Generation
  - Class Generation
  - Lab: Contact Management App (Part 1)
  - Lab: Contact Management App (Part 2)
- Services
  - Lab: Refactor Contact List to Use Service
- Routing
  - Lab: Add Routing to Contact Management App

Day 2 Overview

- Testing with Jasmine
  - Benefits of Automated Testing
  - Jasmine Overview
  - Jasmine Syntax
  - Lab: Number Adder
- Jasmine + Angular
  - Lab: Slug Service Test

Angular CLI

Angular CLI Capabilities

- Saves you from writing repetitive code
- Generate a new project (`ng new <project-name>`)
- Serve a project (`ng serve`)
- Run a test suite (`ng test`)
- Generate a new component, directive, service, class, or module (`ng generate …`)

Project Generation

```
$ ng new <project-name>
```

To Serve a Project

```
$ ng serve
```

Spins up a server at http://localhost:4200

Component Generation

```
$ ng generate component <component-name>
```

Service Generation

```
$ ng generate service <component-name>
```

Class Generation

```
$ ng generate class <class-name>
```

Lab: Contact Management App (Part 1)

- Use Angular CLI to create a new project called ContactMe
- `ng new contact-me —prefix contact-me —routing`
- Use Angular CLI to create a `contact-list` component
- The home page should show a list of contacts, each having a name and phone number

Lab: Contact Management App (Part 2)

- Add a “new contact” form with name and phone fields
- New contact should show up on the page when form is saved, just like yesterday’s lab
- Don’t create a new component, just use the same one

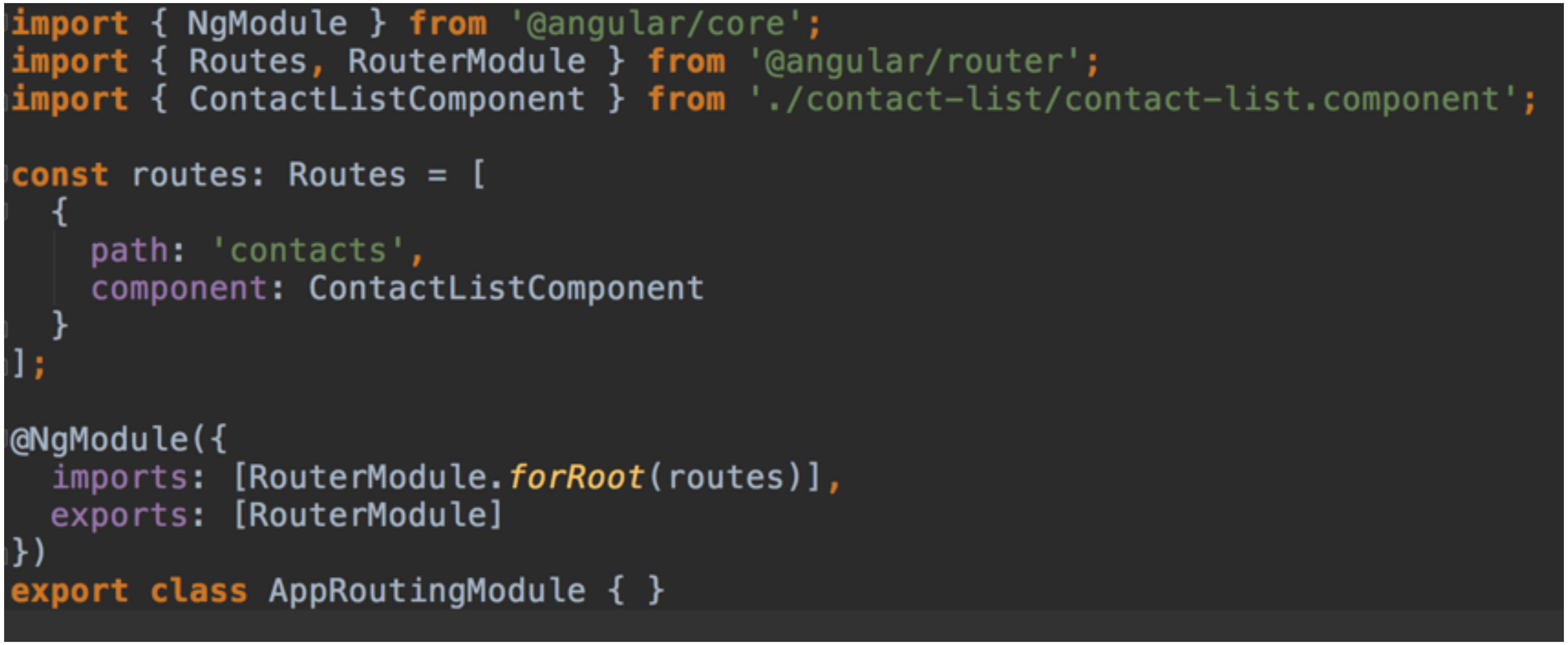
Services

Lab: Refactor Contact List to Use Service

- Generate a `ContactService` using Angular CLI (`ng generate service contact`)
- Get `ContactService` injected into `ContactListComponent`
- Move the contact list into a function in `ContactService` called `getList()`
- Move contact saving into a function in `ContactService` called `save()`
- Make `ContactService` use localStorage

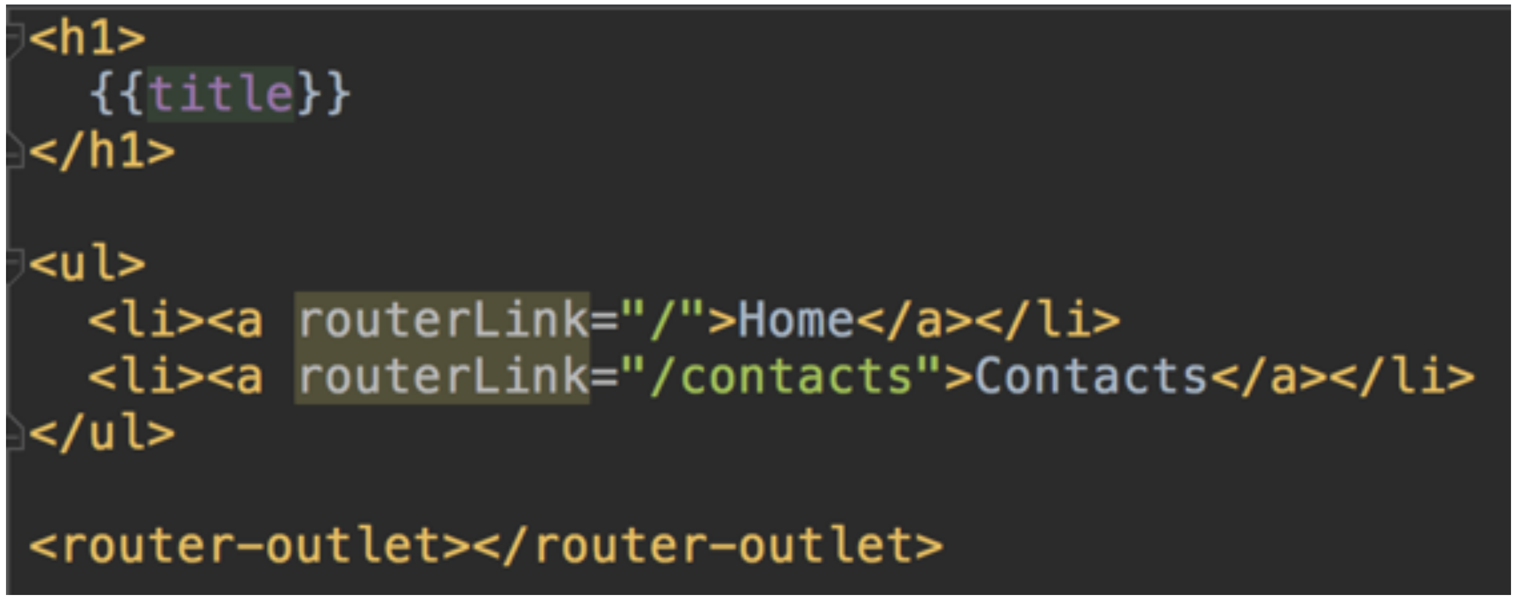
Routing

Routing Example



```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ContactListComponent } from './contact-list/contact-list.component';

const routes: Routes = [
  {
    path: 'contacts',
    component: ContactListComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Link Example



```
<h1>
  {{title}}
</h1>

<ul>
  <li><a routerLink="/">Home</a></li>
  <li><a routerLink="/contacts">Contacts</a></li>
</ul>

<router-outlet></router-outlet>
```

Router Outlet

- `<router-outlet></router-outlet>` is the place where the component specified by the router goes

Lab: Add Routing to Contact Management App

- Add two links to home page: Contacts and Home

Testing with Jasmine

Benefits of Automated Testing

- Automates the manual work of testing
- Protects against regressions
- Helps you think through the features (including abuse cases)
- Helps reveal mistakes and poor designs
- Helps clarify what the code does
- Can be part of a build/deployment process

Jasmine Testing Overview

- Jasmine Overview
- Basic Jasmine syntax
- Jasmine Lab

Jasmine Overview

- Jasmine is a JavaScript testing framework
- Heavily inspired by RSpec (Ruby)
- Relatively new (initial release was 2010)
- The Jasmine testing framework is commonly used with the Karma test runner

Jasmine Syntax: describe

- Put the name of the function or a description of the general area of code inside a `describe`
- First argument is a description, second argument is an anonymous function containing one or more tests

-
  ```
  describe(‘someFunction, function() {
     /* tests go here */
  });
  ```

Jasmine Syntax: it

- An `it` is nested inside a `describe`
- The describe describes the general thing you're testing; the it describes the specific test case
- First argument is test case, second argument is anonymous function containing test
- Best practice: one test per it

-
  ```
  it('returns the sum of two numbers', function() {
    /* test goes here */
  });
  ```

Jasmine Matchers

- `toEqual`
- `toBe`
- `toBeNull`
- `toBeUndefined`
- `toBeTruthy`
- Full list here: https://github.com/jasmine/jasmine/tree/master/src/core/matchers

Jasmine Syntax: beforeEach

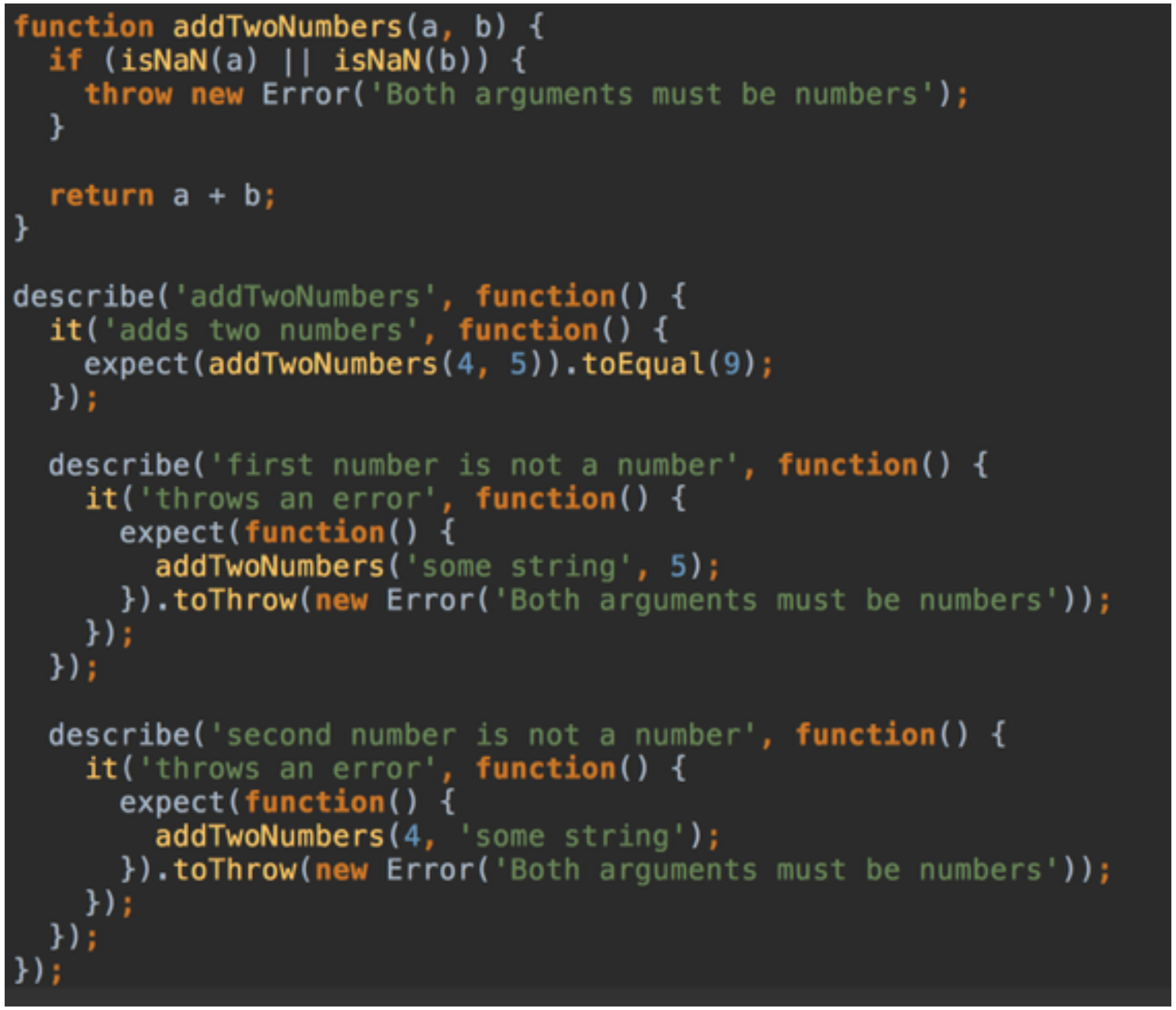
- If you want something to run before each test, put it inside beforeEach

```
describe(‘foo’, function() {
  beforeEach(function() {
    /* whatever you want */
  });
}
```

Lab: Number Adder

- Write a function that can parse the following string inputs:
  - ‘1 + 1’
  - ‘5 + 2 + 8’
  - ‘3+4’
  - ‘7 + -2’
  - ‘7+-2’
  - ‘-5’
  - ‘1 plus 2’
- Write a test for every success case as well as every error case you can think of
- Rule: no using `eval()`
- `$ jasmine spec/addNumbers.spec.js`

Lab: Number Adder

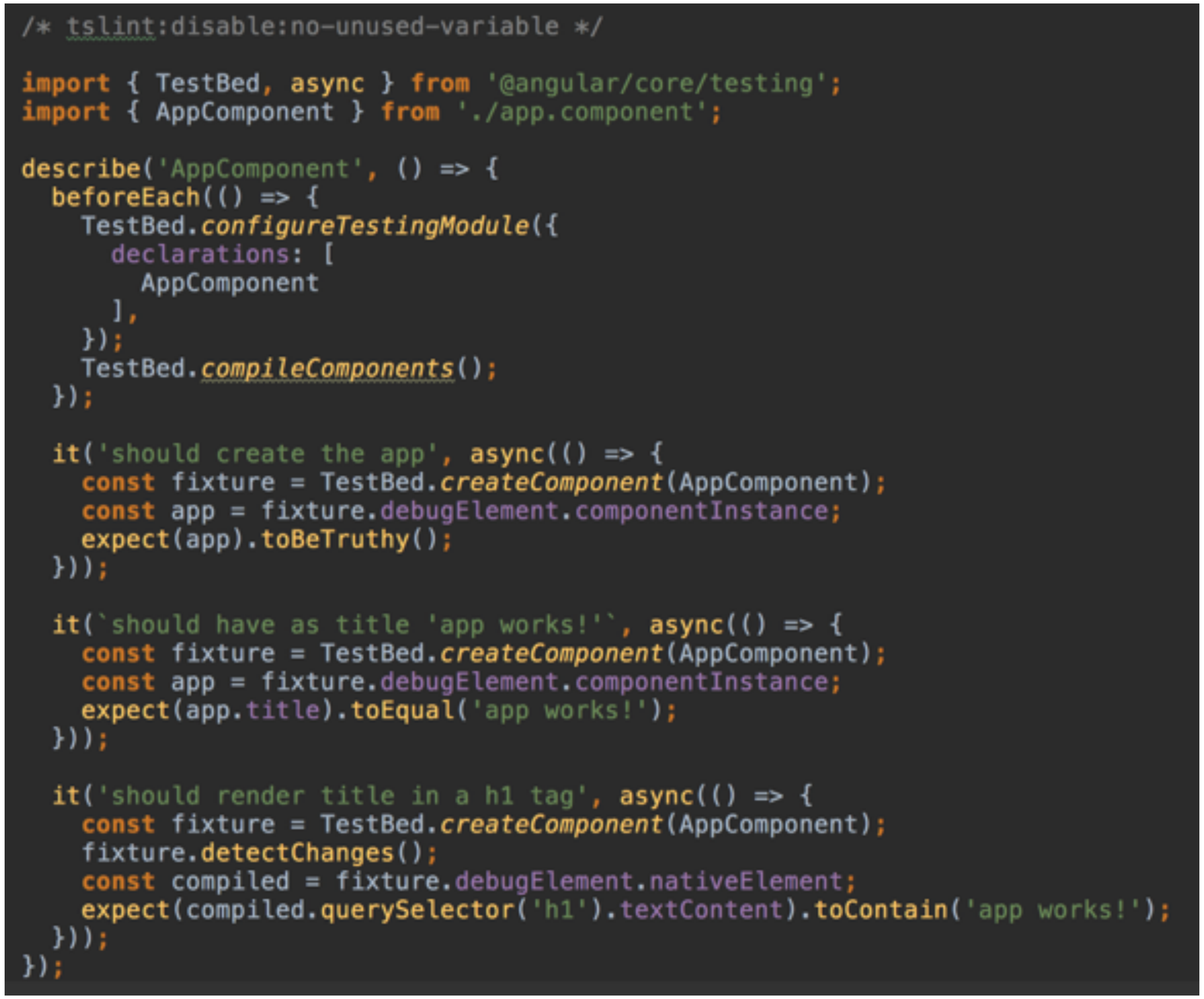


```
function addTwoNumbers(a, b) {
  if (isNaN(a) || isNaN(b)) {
    throw new Error('Both arguments must be numbers');
  }

  return a + b;
}

describe('addTwoNumbers', function() {
  it('adds two numbers', function() {
    expect(addTwoNumbers(4, 5)).toEqual(9);
  });

  describe('first number is not a number', function() {
    it('throws an error', function() {
      expect(function() {
        addTwoNumbers('some string', 5);
      }).toThrow(new Error('Both arguments must be numbers'));
    });
  });

  describe('second number is not a number', function() {
    it('throws an error', function() {
      expect(function() {
        addTwoNumbers(4, 'some string');
      }).toThrow(new Error('Both arguments must be numbers'));
    });
  });
});
```

Jasmine + Angular

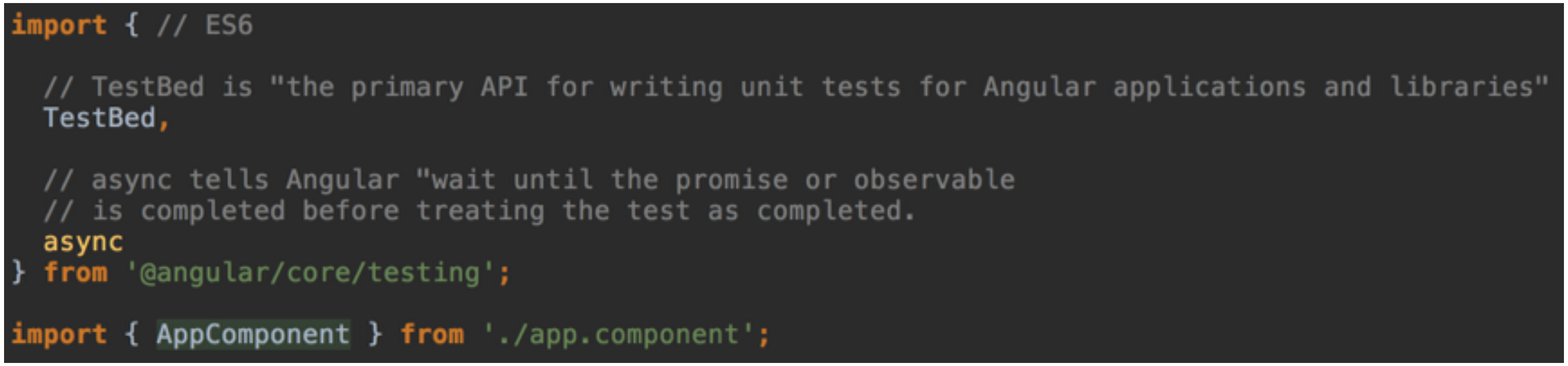
- Angular uses Jasmine for testing
- Angular runs tests using the Karma test runner
- End-to-end (e2e) tests can be written using Protractor (although we won’t be touching much on Protractor today due to time)
- Angular provides other tools that make testing easier (e.g. TestBed)
- https://angular.io/docs/ts/latest/guide/testing.html

Jasmine + Angular



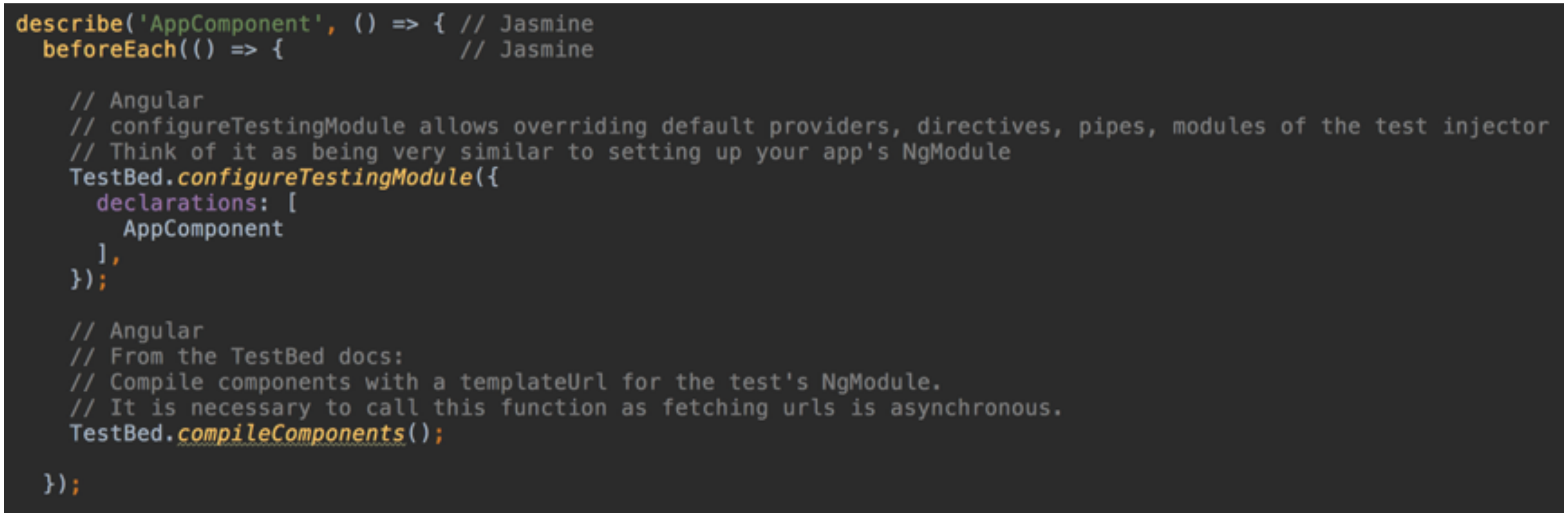
```
/* tslint:disable:no-unused-variable */

import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    });
    TestBed.compileComponents();
  });

  it('should create the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));

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

  it('should render title in a h1 tag', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('app works!');
  }));
});
```

Test Breakdown: import



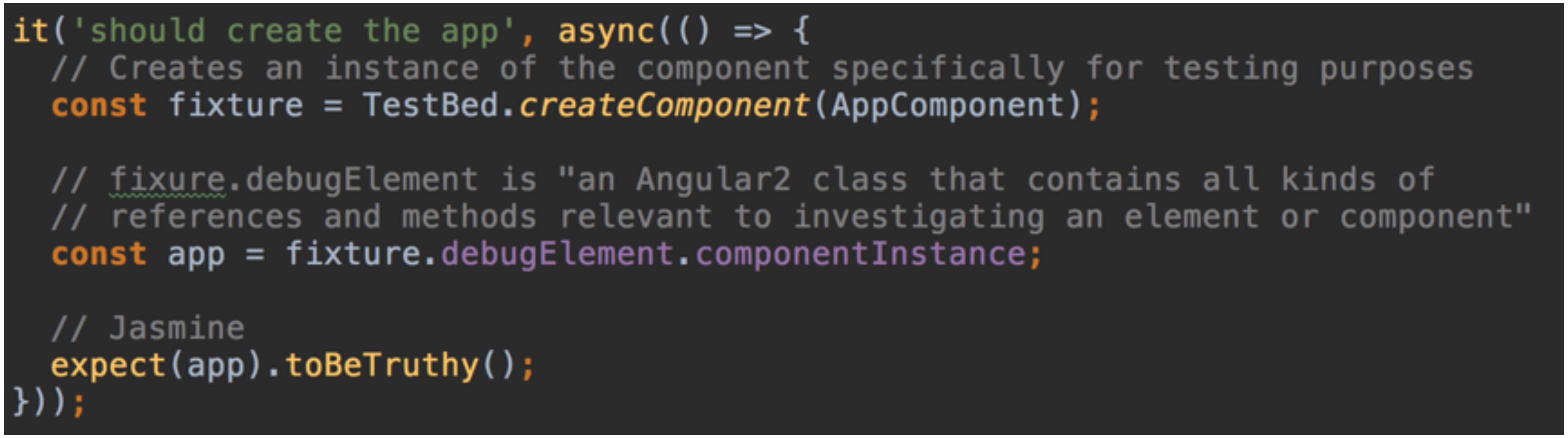
```
import { // ES6

  // TestBed is "the primary API for writing unit tests for Angular applications and libraries"
  TestBed,

  // async tells Angular "wait until the promise or observable
  // is completed before treating the test as completed.
  async
} from '@angular/core/testing';

import { AppComponent } from './app.component';
```

Test Breakdown: beforeEach



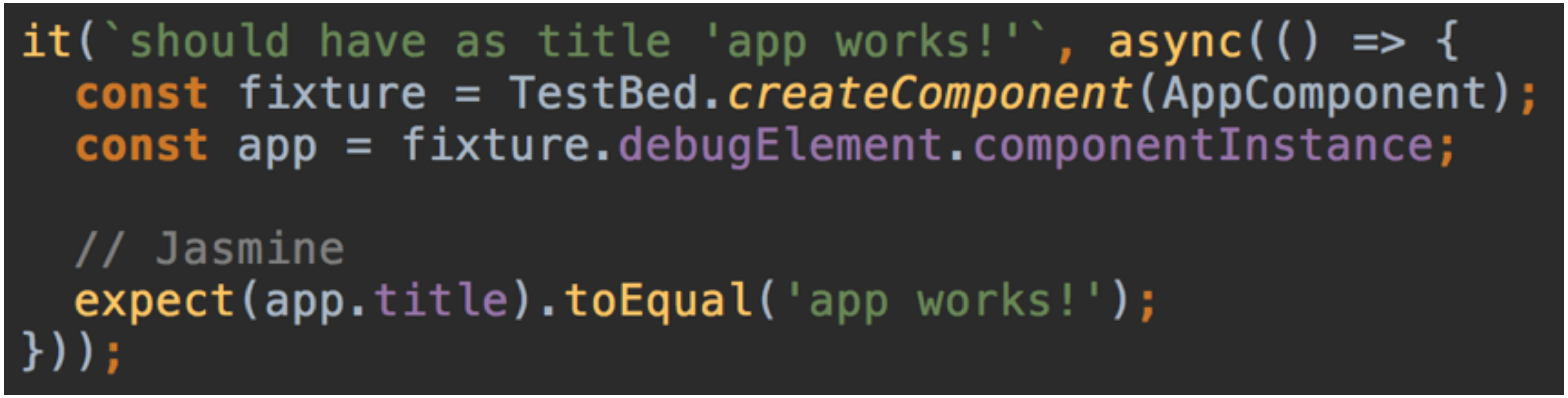
```
describe('AppComponent', () => { // Jasmine
  beforeEach(() => {             // Jasmine

    // Angular
    // configureTestingModule allows overriding default providers, directives, pipes, modules of the test injector
    // Think of it as being very similar to setting up your app's NgModule
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    });

    // Angular
    // From the TestBed docs:
    // Compile components with a templateUrl for the test's NgModule.
    // It is necessary to call this function as fetching urls is asynchronous.
    TestBed.compileComponents();

  });
```

Test Breakdown: AppComponent Creation



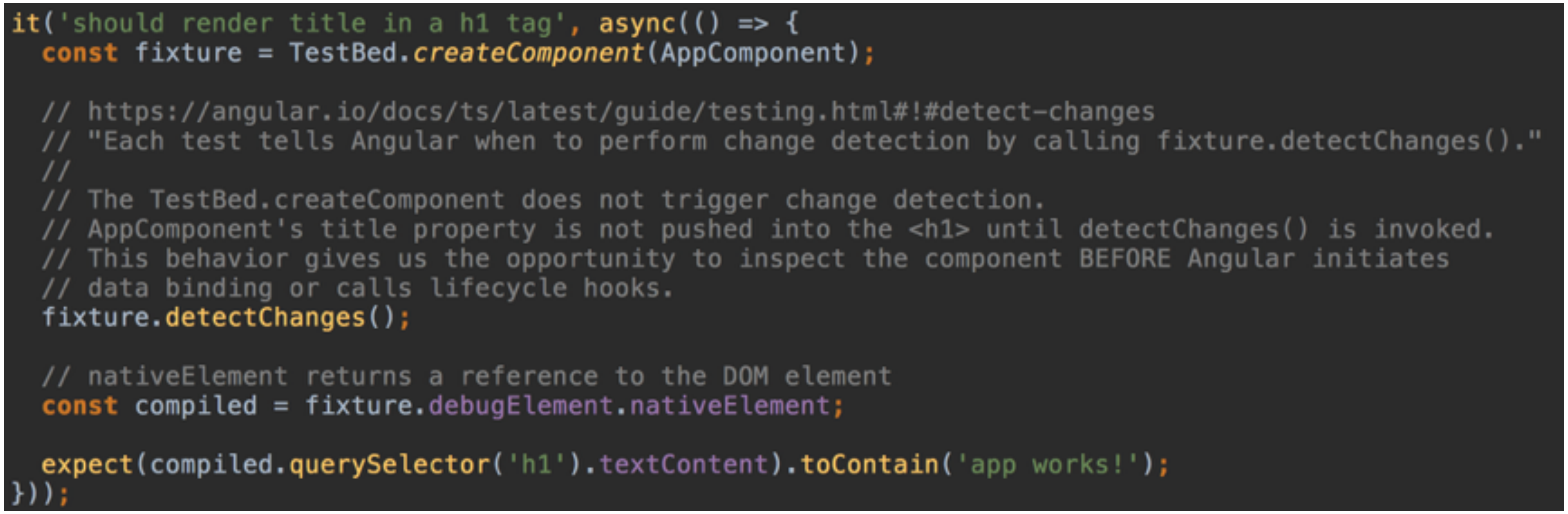
```
it('should create the app', async(() => {
  // Creates an instance of the component specifically for testing purposes
  const fixture = TestBed.createComponent(AppComponent);

  // fixure.debugElement is "an Angular2 class that contains all kinds of
  // references and methods relevant to investigating an element or component"
  const app = fixture.debugElement.componentInstance;

  // Jasmine
  expect(app).toBeTruthy();
}));
```

Test Breakdown: Title



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

  // Jasmine
  expect(app.title).toEqual('app works!');
}));
```

Test Breakdown: Data Binding



```
it('should render title in a h1 tag', async(() => {
  const fixture = TestBed.createComponent(AppComponent);

  // https://angular.io/docs/ts/latest/guide/testing.html#!#detect-changes
  // "Each test tells Angular when to perform change detection by calling fixture.detectChanges()."
  //
  // The TestBed.createComponent does not trigger change detection.
  // AppComponent's title property is not pushed into the <h1> until detectChanges() is invoked.
  // This behavior gives us the opportunity to inspect the component BEFORE Angular initiates
  // data binding or calls lifecycle hooks.
  fixture.detectChanges();

  // nativeElement returns a reference to the DOM element
  const compiled = fixture.debugElement.nativeElement;

  expect(compiled.querySelector('h1').textContent).toContain('app works!');
}));
```

Lab: Slug Service Test

- Use Angular CLI to create a new version of ContactMe (or any app name you want)

- Use Angular CLI to create a `SlugService`

- Write tests for a `slugify()` function that will convert a string like “`Paul McCartney`” to a string like “`paul-mccartney`”

Lab: Contact Details Page

- Go back to original ContactMe app
- When a contact saves, give it a slug in addition to a name and phone number
- Add a link to a working detail page route that matches a pattern like `/contacts/paul-mccartney`

Day 2 Review

- Angular CLI
    - Lab: Contact Management App (Part 1)
    - Lab: Contact Management App (Part 2)
- Services
    - Lab: Refactor Contact List to Use Service
- Routing
    - Lab: Add Routing to Contact Management App

Day 2 Review

- Testing with Jasmine
  - Benefits of Automated Testing
  - Jasmine Overview
  - Jasmine Syntax
  - Lab: Number Adder
- Jasmine + Angular
  - Lab: Slug Service Test

Evaluation Time

https://goo.gl/gEZ9sd