ANGULAR TESTING WITH JASMINE, KARMA AND PROTRACTOR

accenture

MODULE OBJECTIVES

At the end of this module, you should be able to:

- Implement Unit Testing
- Write test cases with Jasmine
- Perform Isolated and Integrated Testing
- Use Angular Test Utilities
- Perform E2E Testing with Protractor

AGENDA

- Unit Testing
- Jasmine Overview
- Isolated and Integrated Testing
- Angular Test Utilities
- E2E Testing with Protractor

Routing in Angular

- Routing enables navigation from one view to the next as users perform application tasks.
- The Angular Router can interpret a browser URL as an instruction to navigate to a required view component.
- You can navigate imperatively when the user clicks a link/button, selects from a drop box etc
- We can pass optional parameters to the supporting view component that help it decide what specific content to present.
- Angular router logs activity in the browser's history so the back and forward buttons work as well.

Why Testing?

- Testing Guards against changes that break existing code
- It clarifies what the code does both when used as intended and when faced with deviant conditions
- It reveals mistakes in design and implementation

Unit Testing

Pause for Thought

Why are Unit Tests conducted?

Unit Test Definition

Tests the individual units of an entire solution

Ensures that the component is built and behaves per the specifications

Unit Test

Unit

Pass

Fail

Angular Testing Tools

Technology	Purpose
Jasmine	The Jasmine test framework provides everything needed to write basic tests
Angular testing utilities	Angular testing utilities create a test environment for the Angular application code under test. Use them to condition and control parts of the application as they interact within the Angular environment.
Karma	The karma test runner is ideal for writing and running unit tests while developing the application. It can be an integral part of the project's development and continuous integration processes.
Protractor	Use protractor to write and run end-to-end (e2e) tests. End-to-end tests explore the application as users experience it. In e2e testing, one process runs the real application and a second process runs protractor tests that simulate user behavior and assert that the application respond in the browser as expected.

Jasmine

- Jasmine is a behavior-driven development framework for testing JavaScript code
- It is an open source technology
- It is a simple API to test different components of JavaScript
- It does not require a DOM
- Jasmine is capable of testing synchronous and asynchronous JavaScript code
- Writing Test Cases using Jasmine is very easy as syntax is very simple
- Strong readability and helps logging errors
- Includes matchers and possible customer matchers
  - toBe, not.toBe, toEqual, toBeTruthy, toMatch, etc.

Suite & Spec in Jasmine

In Jasmine, there are two important terms

- suite
- spec

Suite Block: 'describe()' function

- A Jasmine suite is a group of test cases that can be used to test a specific behavior of the JavaScript code.
- Suite is the basic building block of Jasmine framework.
- It is a global Jasmine function 'describe' with two parameters: a string and a function.
- A test suite begins with a call to this describe with two parameters: a string and a function.
- The first parameter , a string is a name or title for a spec suite and the second parameter represents a function that implements the test suite.
- Nesting of describe function is also allowed in Jasmine.

Spec Block: 'it()' function

- A Jasmine spec represents a test case inside the test suite.
- Specs are defined by calling global jasmine function 'it’.
- This also takes a string and a function.
- The string is the title of the spec and the function is the spec, or testcase.
- A spec contains one or more expectations that test the state of the code.
- An expectation in Jasmine is an assertion that is either true or false.
- A spec with all true expectations is a passing spec.
- A spec with one or more false expectations is a failing spec.

Assertion: 'expect()' function

- Expectations are built with the function expect() which takes a ActualValue
- It is chained with a ‘Matcher’ function, which takes the ExpectedValue.

Matchers

- Each matcher implements a boolean comparison between the ActualValue and the ExpectedValue.
- It is the responsibility of Jasmine to report if the expectation is true or false Jasmine will then pass or fail the spec.

List of Matchers

Few of the frequently used Matchers in Jasmine are listed below:

MATCHER	PURPOSE
toBe()	passed if the actual value is of the same type and value as that of the expected value.
toEqual()	works for simple literals and variables; should work for objects too
toMatch()	to check whether a value matches a string or a regular expression
toBeNull()	to ensure that a property or a value is null.
toBeTruthy()	to ensure that a property or a value is true
ToBeFalsy()	to ensure that a property or a value is false
toContain()	to check whether a string or array contains a substring or an item.
toBeLessThan()	for mathematical comparisons of less than
toBeGreaterThan()	for mathematical comparisons of greater than
toBeCloseTo()	for precision math comparison

Note: For a complete list of Matchers and their description, please refer to the below link
https://github.com/JamieMason/Jasmine-Matchers#matchers

accenture

Setup and Teardown

Jasmine provides two global functions at suite level for SetUp and TearDown Purposes:

1. beforeEach()
2. afterEach().

beforeEach()

The beforeEach function is called once before each spec in the describe() in which it is called.

afterEach()

The afterEach function is called once after each spec.

Note:

- As a good practice, spec variables are defined at the top-level scope. i.e. the describe block.
- Initialization code is moved into a beforeEach function.

- The afterEach function is used to reset the variable before continuing.

accenture

Jasmine

```
// Test Suite
describe("A suite", function() {
  // Test Spec/Case
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
    //Expectations are built with the function expect which takes a value,
    // called the actual.
    // It is chained with a Matcher function, which takes the expected value.
  });
});
```

Karma

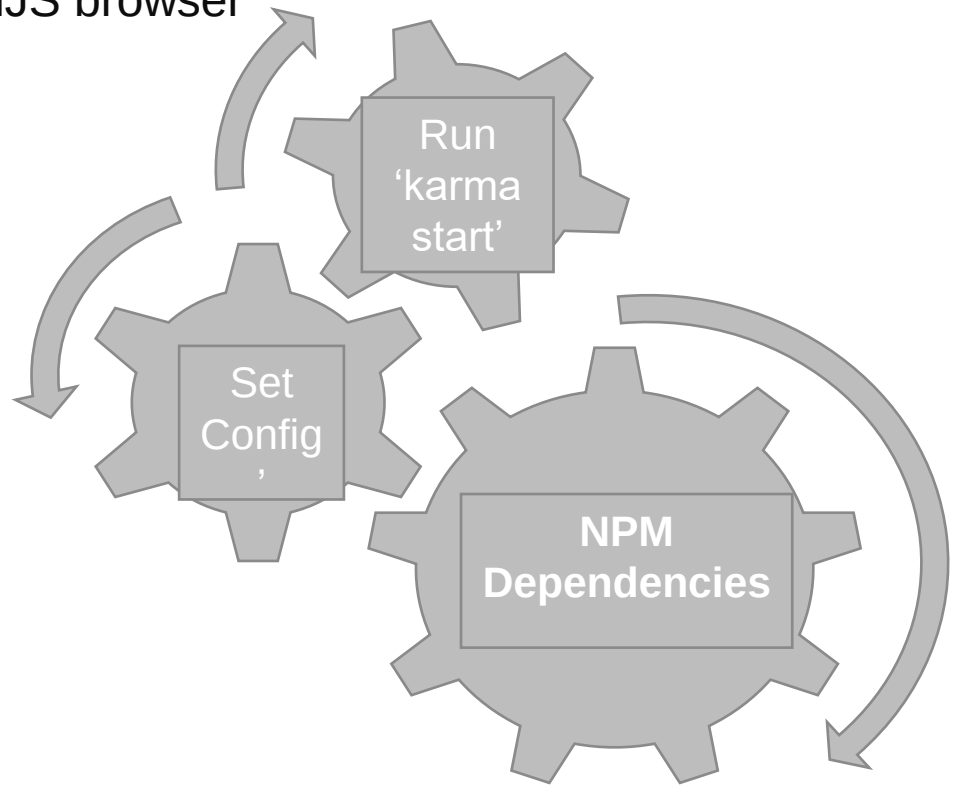
- Karma is test running framework
- It can run for any JavaScript testing framework (given that an adapter is created for that testing framework)
- Karma spawns an instance of the browser(s) specified
- Karma can watch for file changes, and run tests whenever those files change.
- All Karma configuration is placed in the file karma.conf.js

Setting Up and Configuring Karma

Required NPM modules:

- **karma**: Supplies the Karma Framework
- **karma-coverage**: Includes Karma code coverage
- **karma-jasmine**: Includes the Karma Jasmine adapter
- **karma-phantomjs-launcher**: Allows Karma to run in the PhantomJS browser

- A configuration file can be generated by running ***karma init***



Angular Testing

- **@angular/core/testing** provides wrappers and routines that Angular requires when testing components.
- Each unit test is put into its own separate file.

Isolated unit tests vs. Integrated Test

Isolated unit tests

- Examines an instance of a class only without any dependence on Angular or any injected values.
- The tester creates a test instance of the class with *new*, supplies test doubles for the constructor parameters as required and probes the test instance API.
- Eg: Test cases for Pipes and Services

Integrated Test

- Tests both the class and its template
- Constructed using angular **Angular Test Utilities**
- The Angular testing utilities include the TestBed class and several helper functions from @angular/core/testing
- Eg: Test cases for Components and Directives

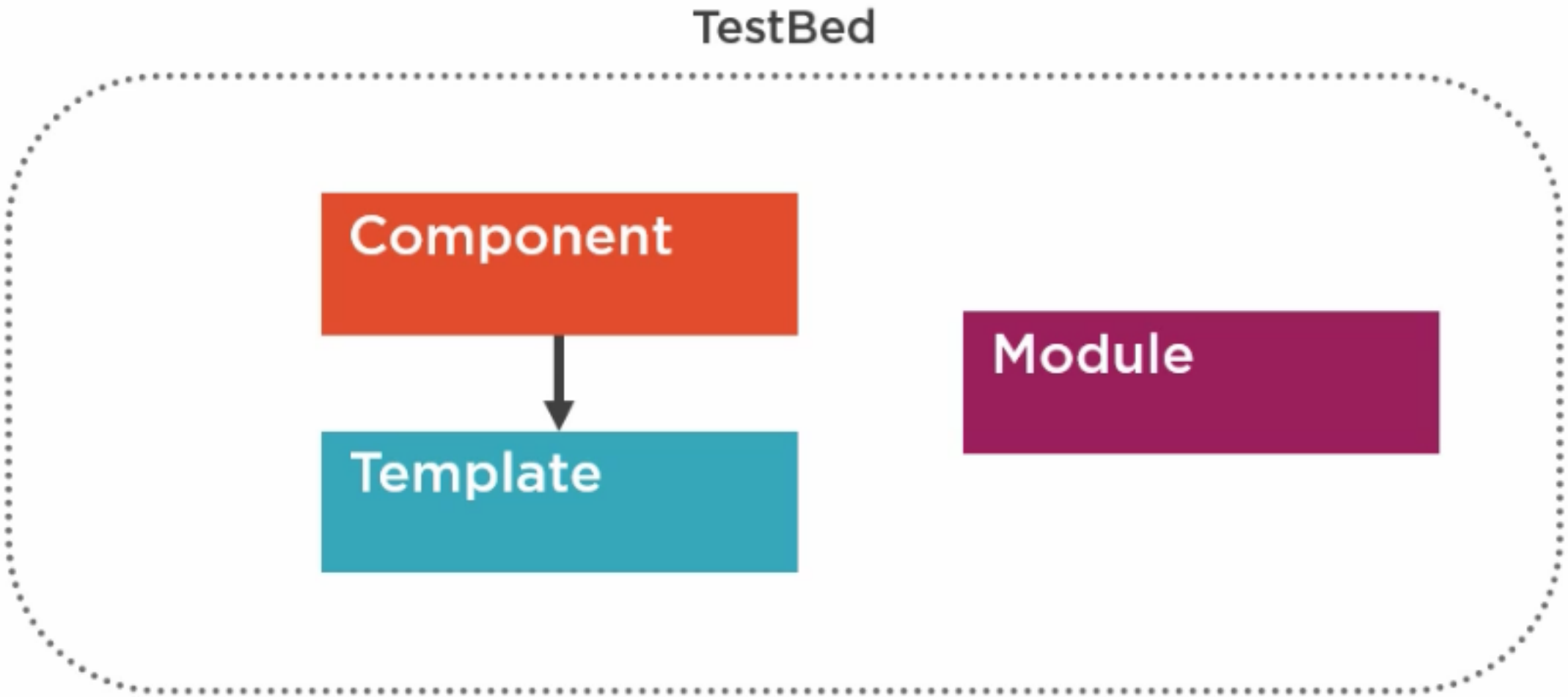
Isolated unit Test

```
describe('AppComponent', () => {
  beforeEach(function() {
    this.app = new AppComponent();
  });

  it('should have hello property', function() {
    expect(this.app.title).toBe('app works!');
  });
});
```

Integrated Tests

Integrated Tests



Unit Testing Demos

Unit Testing Demo – Testing a Service

Product.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class ProductService {

products=["IPhone","IPad","IPod","Moto X","Moto G","Moto
Z"];
getProductCount()
{
return this.products.length;
}
constructor() { }

}
```

Product.service.spec.ts

```
import { TestBed, inject } from '@angular/core/testing';
import { ProductService } from './product.service';

describe('ProductService', () => {
beforeEach(() => {
TestBed.configureTestingModule({
providers: [ProductService]
});
});

it('To get the Count of Products', inject([ProductService],
(service: ProductService) => {
expect(service.getProductCount()).toBe(6);
}));
});
```

Unit Testing Demo – Testing a Pipe

my-pipe.pipe.ts	my-pipe.pipe.spec.ts

import { Pipe, PipeTransform } from '@angular/core';
 	
@Pipe({
name: 'myPipe'
})
export class MyPipePipe implements PipeTransform
{
 	
transform(name: string): string {
return "Mr. "+name;
}
 	
}
	import { MyPipePipe } from './my-pipe.pipe';
	
	describe('MyPipePipe', () => {
	
	let pipe:MyPipePipe;
	beforeEach(() => {
	pipe = new MyPipePipe();
	});
	
	it("Pipe to Transform 'Thangaraj' to 'Mr. Thangaraj'",
	() => {
	const pipe = new MyPipePipe();
	expect(pipe.transform("Thangaraj")).toBe("Mr.
	Thangaraj");
	});
	});

Unit Testing Demo – Testing a Component

my-comp.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
selector: 'app-my-comp',
templateUrl: './my-comp.component.html',
styleUrls: ['./my-comp.component.css']
})
export class MyCompComponent implements OnInit {

constructor() { }

ngOnInit() {
}

}
```

Unit Testing Demo – Testing a Component

my-comp.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { MyCompComponent } from './my-comp.component';
describe('MyCompComponent', () => {
let component: MyCompComponent;
let fixture: ComponentFixture<MyCompComponent>;

beforeEach(async(() => {
TestBed.configureTestingModule({
declarations: [ MyCompComponent ]
})
.compileComponents();
}));
beforeEach(() => {
fixture = TestBed.createComponent(MyCompComponent);
component = fixture.componentInstance;
fixture.detectChanges();
});
it('MyCompComponent Object should be created', () => {
expect(component).toBeTruthy();
});});
```

Testing a Component with inline-template

inline-template.component.ts

```
import { Component, OnInit } from '@angular/core';
@Component({
selector: 'app-inline-template',
// templateUrl: './inline-template.component.html',
// styleUrls: ['./inline-template.component.css']
template:`
<div style="background-color:red">
<h1 align = "center">{{PageHeader}}</h1>
</div>
`,
})
export class InlineTemplateComponent implements OnInit {
PageHeader:string = "Unit Testing Inline Template in a Component Demo"
constructor() { }
ngOnInit() {}
}
```

accenture

Testing a Component with inline-template

inline-template.component.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { By } from '@angular/platform-browser';
import {DebugElement} from '@angular/core'
import { InlineTemplateComponent } from './inline-template.component';

describe('InlineTemplateComponent', () => {
let component: InlineTemplateComponent;
let fixture: ComponentFixture<InlineTemplateComponent>;
let debug: DebugElement;
let htmlElem:HTMLElement;

beforeEach(() => {
TestBed.configureTestingModule({ declarations: [ InlineTemplateComponent ]});
fixture = TestBed.createComponent(InlineTemplateComponent);
component = fixture.componentInstance;
debug = fixture.debugElement.query(By.css('h1'));
htmlElem = debug.nativeElement;
});
it('Message Displayed from Inline Template', () => {
fixture.detectChanges();
expect(htmlElem.textContent).toContain(component.PageHeader);});});
```

Testing a Component with external-template

external-template.component.ts	external-template.component.html

import { Component, OnInit } from '@angular/core';
 	
@Component({
selector: 'app-external-template',
templateUrl: './external-template.component.html',
styleUrls: ['./external-template.component.css']
})
export class ExternalTemplateComponent implements OnInit {
 	
title:string="I am from External Template Component";
constructor() { }
ngOnInit() {
}
}
	
	 {{title}}

Testing a Component with external-template

external-template.component.spec.ts

```
import { async, ComponentFixture, TestBed } from
'@angular/core/testing';
import { DebugElement } from '@angular/core';
import { By } from '@angular/platform-browser';
import { ExternalTemplateComponent } from
'./external-template.component';
describe('ExternalTemplateComponent', () => {
let component: ExternalTemplateComponent;
let fixture:
ComponentFixture<ExternalTemplateComponent>;
let debug: DebugElement;
let htmlElem: HTMLElement;
beforeEach(async(() => {
TestBed.configureTestingModule({declarations: [
ExternalTemplateComponent ]})
.compileComponents();
}));
```

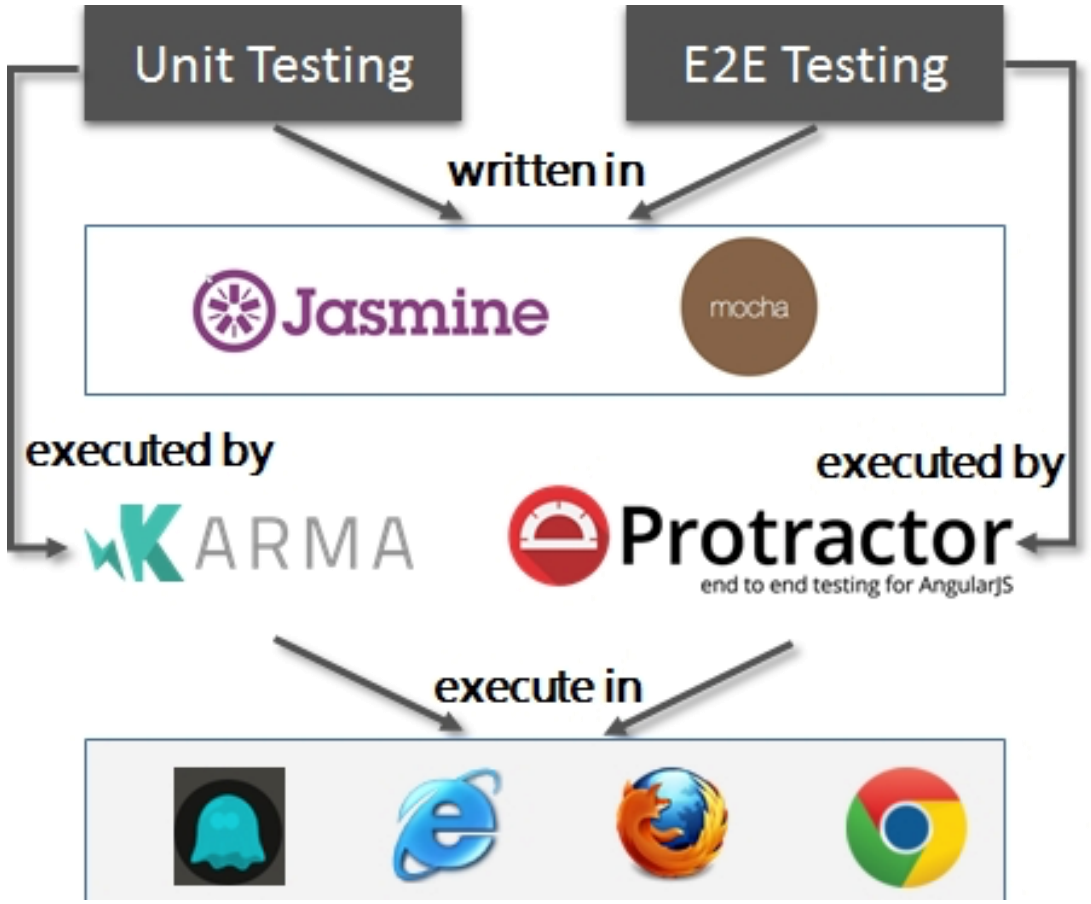
```
beforeEach(() => {
fixture =
TestBed.createComponent(ExternalTemplateComp
onent);
component = fixture.componentInstance;
debug = fixture.debugElement.query(By.css('h1'));
htmlElem = debug.nativeElement;
// fixture.detectChanges();
});
it('Message Displayed from External Template', ()
=> {
fixture.detectChanges();
expect(htmlElem.textContent).toEqual(component.ti
tle);});});
```

What is Protractor?

- Functional testing framework that run tests cases against the application running in a browser

- Builds on top of:
  Jasmine to specify the test cases
  Selenium to talk through the browser

- Protractor can also:
  Run multiple browsers at once
  Run the tests on a remote address



Introduction to E2E Testing with Protractor

- End to end testing (E2E) or also known as integration testing is an excellent way to test at a high level whether the applications function correctly
- It is done ensure that our components are working properly together
- E2E tests are not Unit tests
- E2E test is a high level test to check whether the components interact properly

accenture

Angular CLI and Protractor

- Protractor is a E2E test runner that can take scenario tests and run them in the browser.
- The test code is written using Jasmine.
- Protractor makes E2E and integration tests easy for Angular apps.
- Protractor knows when to run and check the DOM after Angular has done rendering the page.
- When we create an Angular Project we will have a folder named e2e. This is the folder where the e2e tests reside.

MODULE SUMMARY

Now, you should be able to:

- Implement Unit Testing
- Write test cases with Jasmine
- Perform Isolated and Integrated Testing
- Use Angular Test Utilities
- Perform E2E Testing with Protractor

QUESTIONS

THANK YOU