Karma Jasmine

We want to Test our angular components.

Do you remember unit testing in java?

How many files are created in a component?

4.

1) html file

2) css file

3) ts file

4) spec.ts file

What is spec.ts file?

This is the testing file

What is Karma?

Karma is a test runner.

What is Jasmine?

Jasmine is one of the testing framework for angular.

1. Create an angular project – aug5
2. cd aug5
3. ng test

check that all 3 test specs are passed.

Remove the title code and check

Java var

<https://www.geeksforgeeks.org/var-keyword-in-java/>

Jasmine notes:

Test Driven Developement

to test whether app works fine or not

to identify all possible issues

to find bugs and errors

to fix bugs

testing is a process to evaluate the functionality of a software application

testing is used identify whether product is meeting requirement or not

testing helps product security

to check whether app is working as per business requirement

they will have all user test cases which needs to be tested and success

UML Diagrams

use case diagram

test cases (expected output/ actual output) FAIL / SUCCESS

waterfall approach

requirements

designing

coding

testing

deployment

maintenance

agile model

write tests

and aim to fail

write the code

and aim to succeed

and refactor the code

jasmine testing framework

karma test runner

will run our test cases in multiple browsers automatically when a change happens in code

to configure karma.conf.js file,

karma init

to run tests

ng test

Task:

1) create a new angular project

2) create couple of components

3) take a backup of karma.conf.js file contents

4) use karma init (in cmd) to configure the karma test runner

5) use ng test (in cmd) to run tests

karma init gives error showing "command not found"

solution:

(to install karma, use below command)

npm install -g karma-cli

before

withdraw()

class Employee

{

public Employee(Address address)

{

}

}

Address x=new Address();

Employee rama=new Employee(x);

TasK:

create a function that returns "success" message. success message is stored in a class variable.

create a test implementation for this function

debugging:

identify error (logical error)

describe:

fdescribe:

xdescribe:

there can be many describe in a spec.ts file

all describe will be executed (usually)

when there is fdescribe f for first

then only fdescribe functions will execute

when there is xdescribe

those xdescribes are excluded in the test

difference between

describe (default. will execute if there are no fdescribe specs)

fdescribe (execute only fdescribe specs)

xdescribe (except the xdescribe specs)

Scenario 1 - describe only

describe('test1', ..)

describe('test2', ..)

describe('test3', ..)

// Specs executed:

// test1

// test2

// test3

Scenario 2 - single fdescribe

fdescribe('test1', ..)

describe('test2', ..)

describe('test3', ..)

// Specs executed:

// test1

Scenario 3 - multiple fdescribe

fdescribe('test1', ..)

fdescribe('test2', ..)

describe('test3', ..)

// Specs executed:

// test1

// test2

Scenario 4 - single xdescribe

xdescribe('test1', ..)

describe('test2', ..)

describe('test3', ..)

// Specs executed:

// test2

// test3

Scenario 4 - multiple xdescribe

xdescribe('test1', ..)

xdescribe('test2', ..)

describe('test3', ..)

// Specs executed:

// test3

Scenario 5 - fdescribe and xdescribe exists

fdescribe('test1', ..)

xdescribe('test2', ..)

describe('test3', ..)

// Specs executed:

// test1

---------------------------------------------------------------------

in spec.ts file,

we use component.result is

component is the object of LoginComponent

component.result is correct because (result is an instance variable) this.result

lets see if a function returns a value.

we need to test the returned value

how to access

private variables expect(component['marks']).toEqual(20);

private functions expect(component['fnGetMarks']()).toEqual(20);

static variables expect(LoginComponent.score).toEqual(100);

from our spec.ts ?

---------------------------

Pipes

{{ "hello world" | uppercase }}

output: HELLO WORLD

But the effect of pre defined pipes can be tested when

"hello world" is stored in a variable. (variable can be testing)

but

when we create custom pipes, we can test the custom pipes independantly (without applying the pipe)

we can directly test the pipe in out spec.ts