Testing Angular Application

Unit testing

- UNIT TESTING is a level of software testing where individual units/ components of a software are tested.
- The purpose is to validate that each unit of the software performs as designed.
- A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.
- In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class.
- In case of JavaScript/TypeScript in Angular we have to test our functions.

Benefits of Unit Testing

- Improve the design of implementations.
  Start coding a feature without giving it a lot of thought to the design is a very common mistake among developers. Using unit testing is going to enforce to think and re-think the design.

- Allows refactoring
  Since you already have tests that ensure you that everything is working as expected, you can easily add changes to that code with the certainty that you are not adding any bugs.

- Add new features without breaking anything.
  When you are adding a new feature you can run the tests to ensure that you ain't breaking any other part of the application.

When we create project using Angular CLI there are some files which gets created for testing purpose.

• jasmine-core

Jasmine is an open source testing framework for JavaScript.
It aims to run on any JavaScript-enabled platform.
It has a bunch of functionalities to allow us the write different kinds of tests.

• karma

Karma is a task runner for our tests. It uses a configuration file in order to set the startup file, the reporters, the testing framework, the browser among other things.

Consider below example which uses concept of Unit testing

For example if we wanted to test this function:

```
function fun()
{
  return 'Marvellous Infosystems!';
}
```

We would write a jasmine test spec like so:

```
describe('fun', () => {                          // 1
  it('says hello', () => {                       // 2
    expect(fun())                                // 3
        .toEqual('Marvellous Infosystems!');     // 4
  });
});
```

1. describe(string, function)
This function defines what we call a Test Suite, a collection of individual Test Specs.

2. it(string, function)
This function defines an individual Test Spec, this contains one or more Test Expectations.

3. expect(actual)
This expression is what we call an Expectation. In conjunction with a Matcher it describes an expected piece of behaviour in the application.

4. matcher(expected)
This expression is what we call a Matcher. It does a boolean comparison with the expected value passed in vs. the actual value passed to the expect function, if they are false the spec fails.

Jasmine comes with a few pre-built matchers like :

```
expect(array).toContain(member);
expect(fn).toThrow(string);
expect(fn).toThrowError(string);
expect(instance).toBe(instance);
expect(mixed).toBeDefined();
expect(mixed).toBeFalsy();
expect(mixed).toBeNull();
expect(mixed).toBeTruthy();
expect(mixed).toBeUndefined();
expect(mixed).toEqual(mixed);
expect(mixed).toMatch(pattern);
expect(number).toBeCloseTo(number, decimalPlaces);
expect(number).toBeGreaterThan(number);
expect(number).toBeLessThan(number);
expect(number).toBeNaN();
expect(spy).toHaveBeenCalled();
expect(spy).toHaveBeenCalledTimes(number);
expect(spy).toHaveBeenCalledWith(...arguments);
```

Jasmine has a few functions we can use:

beforeAll
This function is called once, before all the specs in describe test suite are run.

afterAll
This function is called once after all the specs in a test suite are finished.

beforeEach
This function is called before each test specification, it function, has been run.

afterEach
This function is called after each test specification has been run.

This are the contents of files which are created by Angular CLI and which required for testing

Karma.conf.js file

```
module.exports = function (config)
{
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular-devkit/build-angular/plugins/karma')
            ],
    client:
    {
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    coverageIstanbulReporter:
    {
      dir: require('path').join(__dirname, '../coverage'),
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  });
};
```

This file contains some important things as

frameworks
This is where jasmine gets set as a testing framework. If you want to use another framework this is the place to do it.

reporters
This is where you set the reporters. You can change them or add new ones.

autoWatch
If this is set to true, the tests run in watch mode. If you change any test and save the file the tests are re-build and re-run.

browsers
This is where you set the browser where the test should run. By default is google but you can install and use other browsers launchers.

test.ts file

This file is required by karma.conf.js and loads recursively all the .spec and framework files

```
import 'zone.js/dist/zone-testing';
import { getTestBed } from '@angular/core/testing';
import {BrowserDynamicTestingModule, platformBrowserDynamicTesting} from '@angular/platform-browser-dynamic/testing';

declare const require: any;

// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);

// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().map(context);
```

While testing there is no need to change anything in this file, but there are some important things as

An environment to run angular tests is being created using all the imports at the begging of the file.
TestBed is a powerful unit testing tool provided by angular, and it is initialized in this file.
Finally, karma loads all the tests files of the application matching their names against a regular expression.
All files inside our app folder that has "spec.ts" on its name are considered a test.

app.component.spec.ts file

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

// run before every test case
describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  // test 1
  it('should create the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));

    // test 2
  it(`should have as title 'app'`, async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app');
  }));

    // test 3
  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('Welcome1 to app!');
  }));
});
```

To start Unist testing we can use below CLI command as
ng test
The ng test command builds the app in watch mode, and launches the karma test runner.
We can update app.component.spec.ts file for unit testing purpose.