# ANGULAR UNIT TESTING

# Angular CLI, Jasmine, Karma, Protractor & Cucumber

## • Understanding Testing Concepts in Angular

ARC Tutorials

- **Unit Testing**
  - Testing smaller granular independent modules, services or pipes is called as unit testing
  - Focus is on "task" level testing
  - Provides input and verifies output for smaller sample
  - Ideal for testing individual pieces of the big picture
  - Frameworks used are Jasmine and Karma

- **End to End Testing**
  - Automating the entire end to end functional flows
  - Helps in building automated test suites when the application or product sizes outgrows
  - Protractor framework is used for End to End Testing
  - BDD frameworks like Cucumber are also used for testing the functional flows

## • Testing Utilities Provided by Angular

ARC Tutorials

- Auto generation of code
- Angular natively supports unit tests using Jasmine and Karma
- Angular CLI has built-in commands to run the unit tests
- Angular CLI has built-in commands to run the End to End tests

- Using Angular CLI – we generate the test files (spec) files
- We can also use Angular In-Memory service to mock the services

- Angular provides easy way to Mock Services and use Classes which will helps in testing
- Support for extending and using other frameworks like BDD (cucumber) etc

- **Frameworks and Tooling we will learn:**
  - Angular CLI
  - Jasmine
  - Karma
  - Protractor
  - Cucumber

- **Jasmine Framework**

ARC Tutorials

- Jasmine is an open source testing framework for JavaScript
- Jasmine is Behavior Driven Development testing framework
- In BDD – the tests are written in non-technical language which makes it easy for Business Analysts too to write test specs
- It does not require any JavaScript framework
- Easy to ready syntax, easy to maintain
- Jasmine has native support for Async testing
- Jasmine supports Spy objects using which we can target an element we want to test

- **Basic Syntax End to End Testing**

```
describe('appComponent', () => {
    It('it should navigate', () => {
        console.log("navigated to page");
    })
})
```

Subscribe and Ask your doubts in comments section

## • Karma Framework

ARC Tutorials

- Karma is an open source testing framework for JavaScript
- Karma is created and maintained primarily by the core Angular team
- The framework is natively integrated into the Angular apps by default
- Since it's integrated, we can easily run our tests from the CLI console
- One of the striking features is the ability to run on different browsers resolutions like Tablet, Mobile etc

- Karma is used for running and executing the test scripts.
- Hence the framework plays well with other JavaScript frameworks like Jasmine, Mocha etc
- The framework can be integrated right into the build pipelines which means it will run in our build and deployment pipelines

  - Automating the entire end to end functional flows
  - Helps in building automated test suites when the application or product sizes outgrows
  - Protractor framework is used for End to End Testing
  - BDD frameworks like Cucumber are also used for testing the functional flows

Ads in

## • Protractor Framework

ARC Tutorials

- Protractor is an open source E2E testing framework for Angular applications

- It is built by a team in Google on the top of WebDriver.

- It also works as a solution integrator that combines powerful technologies such as NodeJS, Selenium, Jasmine, WebDriver, Cucumber, Mocha etc.

- Along with testing of Angular application, we can also use Protractor for automated regression tests for any web applications.

- It allows us to test our application just like a real user because it runs the test using an actual browser

## • Cucumber Framework

ARC Tutorials

- Cucumber is a tool that supports Behaviour-Driven Development(BDD)

- Cucumber is Open Source software

- Cucumber reads executable specifications written in plain text and validates that the software does what those specifications say

- The specifications consists of multiple examples, or scenarios

- Gherkin is a set of grammar rules that makes plain text structured enough for Cucumber to understand

- **How to Run Unit Tests?**

  - Unit testing is a type of testing where the focus of the test is to test a particular small piece of the application

ARC Tutorials.

  - The test can be for components, services, pipes, directives etc

  - Angular has support for testing Unit Tests

  - Understanding unit tests in Angular

  - To test run the below command
    - **ng test**

```
Unit tests
1. Each of the unit test file will having ending with spec.ts

2. ng test

3. First compile your application
    - open browser to run the tests
    - if you are running your tests in build pipeline
        - headless execution

4. The tests written in Jasmine and are executed using Karma runner
```

```
5. Jasmine report
    - red x mark means test case failed
    - green dot means the test has passed

6. Karma.conf.js -> which is used for configuring parameters in testing
```

- **Commonly used Options**
  - --browsers= browser
  - --code-coverage
  - --karmaConfig=*karmaConfig*
  - *--main=main*
  - --configuration=*configuration*
  - -progress=true|false

- **How to Run End to End Tests?**

ARC Tutorials

  - End to End test means automating the application's workflow for a functionality end to end
  - Angular has support for testing End to End tests
  - Angular uses Protractor for running the end to end tests
  - We can also configure the default parameters that the protractor takes
  - Understanding End to End tests in Angular
  - **Running tests using Angular CLI**
    - ng e2e

**Subscribe and Ask your doubts in comments section**

## • Commonly used Options

- --browsers= browser
- --baseUrl = baseUrl
- --specs
- --code-coverage
- --host=host
- --port=port
- --prod = true/false
- --protractorConfig=*protractorConfig*
- --suite=*suite*
- --*main=main*
- --configuration=*configuration*

Su

## • Why Code Coverage is Important?

ARC Tutorials

- Angular has support for generating code coverage test reports
- Most, if not all, clients will ask you to submit the test code coverage report
- If the code coverage is more than 85% - your code is considered to be good
- **We can generate code coverage for:**
  - For Unit Tests

## • Ways to generate code coverage

1. **For Unit Tests:**
   1. **By adding "codeCoverage" in tests inside angular.json**
   2. **By adding the CLI ng test option –code-coverage**

## • Why Skip Tests?

ARC Tutorials

- Angular has support for skipping testing End to End tests
- Its very useful and helpful sometimes to suppress a failing test case than fix it
- Specially during deployments or code promotions if some "non core" test case is failing – we can skip it for various reasons
- **Ways to skip tests in Angular**

1. For **New** applications
2. For **Existing** application
3. For **existing** tests

**Subscribe and Ask your doubts in comments section**

## • Ways to Skip Tests

1. **Using CLI – For New Applications**

   ng new –skipTests

2. **By adding "skipTests" to true in schematics in Angular.json**

   SkipTests: true

   **Not encouraged option**

3. **By Adding "x" in front of the tests**
   - Xdescribe
   - Xit

```
Angular CLI tooling w.r.t testing
    ng test  // unit tests
    ng e2e  // end to end tests
    ng test --code-coverage  // getting code coverage
    ng new --skip-tests  // skipping tests
```

# Angular : Basic Concepts in Jasmine

- **Understanding basic concepts**

ARC Tutorials

- **Describe** - A test suite begins with a call to the global Jasmine function describe with two parameters: a string and a function
- **Specs** - Specs are defined by calling the global Jasmine function it, which, like describe takes a string and a function
- **Expect** - Expectations are built with the function expect which takes a value, called the actual
- **Matcher** - It is chained with a Matcher function, which takes the expected value.

```
For e.g
    describe("I want to test Login Component", function{
        it("Check login tax", func){
            var tax = 15;
            expect(tax).toBe(10); // asserting the value
        } // fail

        it("Check login input element", func){

        }

        it("Check login heading", func){

        }
    })
```

# Angular : Understand Angular Default Test Spec

**Subscribe and Ask your doubts in comments section**

ARC Tutorials

- **Understanding default test spec file**
  - **TestBed, Async** – Import from built-in modules
  - **Importing Required Modules -** import all required modules
  - **Importing Required Components** – import all required components
  - **BeforeEach Methods** – You will see 2 beforeEach methods implemented
    - - Covered in separate dedicated episode in next tutorial
  - **TestBed.configureTestingModule({**
    - *this is same as configuring a module*
  - **})**
  - **A fixture is a wrapper for the component and it's template**

```
2. Import Components required for testing
  - for which we are writing our tests

3. describe('reviewComponent', ()=> {

})

4. Inside describe we will have it functions

  describe('reviewComponent', ()=> {

    it('test case #1', ()=> {

    })

  })
```

```
5. We create a fixture for component and template
  -what is a fixture?
    - its wrapper around the component and the template
    - using fixture - we can get properties of component class and template

6. BeforeEach -> method ->

  before running test scripts - we need some ground work
    - setting up component
    - setting up data
    - importing services etc etc etc
```


```
6. BeforeEach -> method ->

  before running test scripts - we need some ground work
    - setting up component
    - setting up data
    - importing services etc etc etc
```


```
7. BeforeEach - Next episodes
  - why are 2 beforeEach?
  - what does each do?
  - what we can do in beforeEach

8. TestBed - main utility to define our module, components etc

*/
```

# Angular : What is AAA Pattern?

- **Understanding basic concepts**

ARC Tutorials

  - **Arrange** – “arrange” everything like setup ground work for working with tests for execution.
    - For e.g create a object of Component etc
  - **Act** – Act on your unit test case, calling methods, processing data etc
  - **Assert** – Verifying the actual data of test result and expected data

# Angular : BeforeEach Method

Subscribe and Ask your doubts in comments section

- **Understanding BeforeEach Method**

ARC Tutorials

- **We use an async before each.** The purpose of the async is to let all the possible asynchronous code to finish before continuing
- Before running any test in angular you need to configure an angular testbed.
- This allows you to create an angular environment for the component being tested.
- Any module, component or service that your tested component needs have to be included in the testbed. Finally, after setting the configuration, you call the compile components function

# Angular : Angular TestBed (ATB)

Subscribe and Ask your doubts in comments section

- **Understanding Angular TestBed (ATB)**

ARC Tutorials

- The **TestBed** is the first and largest of the Angular testing utilities.
- It creates an Angular testing module — a **@NgModule** class — that you configure with the **configureTestingModule** method to produce the module environment for the class you want to test.
- Configures and initializes environment for unit testing and provides methods for creating components and services in unit tests.
- It creates a dependency injection (DI) context and allows us to override providers, services, and whole modules.
- It compiles, instantiates, and renders to HTML our components attaching them to the fixture instance. Any module, component or service that your tested component needs have to be included in the testbed.
- Finally, after setting the configuration, you call the compile components function

```
// Arrange

beforeEach(async(() => {
  TestBed.configureTestingModule({
    imports: [
      RouterTestingModule
    ],
    declarations: [
      AppComponent
    ],
    providers: [],
  }).compileComponents();
}));
```