A Basic Introduction to Unit Testing in Angular Apps

Why Unit Tests?

- Unit tests **answer** the basic **question** of **"does my component/pipe/service/etc work as intended?"**
  - Same with concerns over **input** and **injection**
- **Pros** of unit tests
  - **Guard** against breaking changes
  - **Analyze** code behavior (expected and unexpected)
  - **Reveal** design mistakes

Analyzing the Testing Setup (as created by the CLI)

- **Angular** naturally **generates** unit **test files** → End in **xxx.spec.ts**
- Every **block** starting with the **it** function is an **individual test**
  - **All blocks** are **executed totally independently** of the **it** blocks before them
- The **beforeEach** function **executes code** that is run **before** each **test**

Running Tests (with the CLI)

- We **start** the **testing environment** by entering **ng test** into the **Angular CLI**
- This opens a **new browser window** with **test results**
- We also see test **results** in the **CLI**

Adding a Component and Some Fitting Tests

- **Generating** a **component** with the **CLI** creates a **XXX.spec.ts** file
- Within the **describe** function, specify your **component** name and **beforeEach** function
  - In **beforeEach**'s callback function, add the **TestBed.configureTestingModule** method

    - Declare the component's name in the **declarations** array of the previous method's argument
- Still within **describe**, add **it** functions that represent **individual tests**
    - You need a **fixture**, and **app**, and an **expectation**

Testing Dependencies: Components and Services

- We test for **service injection** within a component test by calling **fixture.debugElement.injector.get(<service-name>)**
- We must call **fixture.detectChanges()** after **injecting** the **service** to **await changes**
- We can directly modify component values from the test

Simulating Async Tasks

- The **spyOn(XXX)** method listens to executions, and **returns** not what is executed, but its **own value**
- We **wrap** the **whole callback function** within **it** inside of an **async** function
- We must place our **expectations** within a **fixture.whenStable().then(XXX) callback**

Using "fakeAsync" and "tick"

- **fakeAsync** allows us to omit **whenStable(XXX)** within our tests
- **tick() forces** all **asynchronous tasks** to **finish** before progressing

Isolated vs Non-Isolated Tests

- **Pipe** tests don't need a **beforeEach** function → Much **slimmer**