Class 28 - A Basic Introduction to Unit Testing in Angular Apps

---

Class 28 Course Content

Lesson Outline

Today we will learn:

1. **Why Unit Tests are important**
2. **How to run unit tests using the CLI**
3. **How to test components, directives, services, and pipes**
4. **How to simulate asynchronous tasks in Angular tests**
5. **Some extra utilities the Angular 12+ package offers**

Lesson Notes

- **Unit Testing:** *Unit Testing* is a software testing methodology that focuses on making sure individual pieces of the source code work as planned.

Unit Test Project Steps

STEP 1: New Project Creation && Analyzing the Testing Setup

Terminal:

- Create a new project.

- Open in VSCode.

```
ng new app-for-testing

cd app-for-testing

code .
```

app.component.spect.ts file:

- Walk through the main function and explain the basic idea of what each function is testing for.

- *Note*: All testing block are independent and don't effect eachother.

- *Note-Continued*: We must create a fresh initialization of the component for every "it" block and run tests on this "fixture/app" simulation. We can "expect" our component to "be something".

Terminal:

- Run the tests.
- Change the title in the `app.component.ts` file to something new. Save and inspect the error log.

```
ng test
```

STEP 2: Testing Components && Services

Terminal:

- Create a new "user" component.

```
ng g c user
```

user/user.component.html:

- Add two "divs" that conditionally display if a variable "isLoggedIn" is true or false.

```
<div *ngIf="isLoggedIn">
  <h2>User is Logged In.</h2>
  <p>Welcome: {{ user.name }}!</p>
</div>

<div *ngIf="!isLoggedIn">
  <h2>User is Logged Out.</h2>
  <p>Please Log In!</p>
</div>
```

user/user.component.ts:

- Create a `user!: { name: string }` variable and a boolean `isLoggedIn = false` variable.

user/user.service.ts file:

- Create a new file titled "user.service.ts" file inside the "user" folder.
- Create a simple user with a name. Inject this service in the root of our application.

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

@Injectable({ providedIn: "root" })
export class UserService {
  user = {
    name: "Will",
    };
  }
```

user/user.component.ts file:

- Inject the "UserService" in the constructor and set the "user" variable equal to the "UserService" user on "ngOnInit()"

```
constructor(private userService: UserService) {}

ngOnInit(): void {
  this.user = this.userService.user;
}
```

user/user.component.spec.ts file:

- Write a test that checks that our "user.name" is indeed coming from the service.
- Write a test that checks if our "isLoggedIn" variable properly shows and hides their respective divs/content.
- Write another test that checks that no name is diplayed if the "user" isn't logged in.

```
it("should pull the user name from the user service", () => {
  let userService = fixture.debugElement.injector.get(UserService);
  expect(component.user.name).toEqual(userService.user.name);
});

it("should display the user name if the user is logged in", () => {
  component.isLoggedIn = true;
  fixture.detectChanges();
  const compiled = fixture.debugElement.nativeElement;
  expect(compiled.querySelector("p")?.textContent).toContain(
    component.user.name
  );
});

it("shouldn't display the user name if the user is logged out", () => {
  const compiled = fixture.debugElement.nativeElement;
  expect(compiled.querySelector("p")?.textContent).not.toContain(
    component.user.name
  );
});
```

STEP 4: Simulating Asynchronous Tasks

shared/data.service.ts file:

- Create the "shared" folder and a "data.service.ts" file inside it.
- Create a `getDetails()` method in this service that returns a resolved promise after 1500ms (1.5 seconds).

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

@Injectable({ providedIn: "root" })
export class DataService {
  getDetails() {
    const resultPromise: Promise<string> = new Promise((resolve, reject)
=> {
      setTimeout(() => {
        resolve("Data");
      }, 1500);
    });
    return resultPromise;
  }
}
```

user/user.component.ts file:

- Inject the new "DataService" in the constructor.
- Create a component variable `data!: string` without initializing a value.
- inside the "ngOnInit()" function, create a promise that sets the `dataService.getDetails()` result to our local data varaible.

```
export class UserComponent implements OnInit {
  user!: { name: string };
  isLoggedIn = false;
  data!: string;

  constructor(
    private userService: UserService,
    private dataService: DataService
  ) {}

  ngOnInit(): void {
    this.user = this.userService.user;
    this.dataService.getDetails().then((data: string) => (this.data =
data));
  }
}
```

user/user.component.spec.ts file:

- Create a test to check that we are only fetching our data asynchronously.

- Create a test to check that we are receiving the correct data using the asynchronous call.
- *Note*: Make sure you import { async } from "@angular/core/testing".

```
it("should fail fetch data if called synchronously", () => {
  let dataService = fixture.debugElement.injector.get(DataService);
  fixture.detectChanges();
  expect(component.data).toBe(undefined!);
});

it("should fetch data successfully if called asynchronously", async(() =>
{
  let dataService = fixture.debugElement.injector.get(DataService);
  fixture.detectChanges();
  fixture.whenStable().then(() => {
    expect(component.data).toBe("Data");
  });
}));
```

STEP 5: Creating Isolated Tests

shared/reverse.pipe.ts file:

- Create the `reverse.pipe.ts` file inside the "shared" folder.
- Add logic to the pipe so it returns the reverse of whatever string is passed to the function.
- Add the ReversePipe declaration to the `app.module.ts` file

```
import { Pipe } from "@angular/core";

@Pipe({ name: "reverse" })
export class ReversePipe {
  transform(value: string) {
    return value.split("").reverse().join("");
  }
}
```

shared/reverse.pipe.spec.ts file:

- Create a new file `reverse.pipe.spec.ts` file inside the "shared" folder.
- Write a tests that sends a string to the `reverse.pipe.ts` file function and "expects" that string to be the opposite value.

```
import { ReversePipe } from "./reverse.pipe";

describe("Pipe: ReversePipe", () => {
  it("should reverse the string", () => {
    let reversePipe = new ReversePipe();
    expect(reversePipe.transform("hello")).toEqual("olleh");
  });
});
```

Additional Notes

Unit Testing Info

- Unit Testing is important because it can help gaurd against breaking changes, it can help analyze your code behavior, and reveal design mistakes before it's too late.
- Writing proper tests takes a lot of time to master, and there are many different ways to perform tests depending on a variety of intended outcomes.
- Isolated Tests: You can test pipes that transform data in an isolated enviornment. You don't need the Angular Testing Package to test these.

Resources

- Angular Docs - Guide to Testing
- Angular Blog - Testing Components in Angular 2 w/ Jasmine