Test Components with Templates and Child Components

---

1. We need to test component is instantiated along with its template. Once instantiated, writing a basic test to check if integration testing is working correctly.

```
import { TestBed, ComponentFixture } from "@angular/core/testing";
import { HeroComponent } from "./hero.component";
import { NO_ERRORS_SCHEMA } from "@angular/core";

describe('HeroComponent (shallow tests)', () => {
  let fixture: ComponentFixture<HeroComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [HeroComponent],
      schemas: [NO_ERRORS_SCHEMA]
    });
    fixture = TestBed.createComponent(HeroComponent);
  });

  it('should have the correct hero', () => {
    fixture.componentInstance.hero = { id: 1, name: 'SuperDude', strength: 3};

    expect(fixture.componentInstance.hero.name).toEqual('SuperDude');
  })
})
```

2. We need to test the template of the component is displayed with hero id & hero name.

```
it('should render the hero name in an anchor tag', () => {
  fixture.componentInstance.hero = { id: 1, name: 'SuperDude', strength: 3};
  fixture.detectChanges();

  expect(fixture.nativeElement.querySelector('a').textContent).toContain('SuperDude');
})
```

3. Testing if component is instantiated. For this test, you can get instance of component as **component = fixture.componentInstance;**

```
it('should create the component', () => {

    expect(component).toBeTruthy();

    console.log(component);

});
```

4. Using DeugElement on fixture.

```
  it('should have the correct hero', () => {
    fixture.componentInstance.hero = { id: 1, name: 'SuperDude', strength: 3};

    expect(fixture.componentInstance.hero.name).toEqual('SuperDude');
  });

  it('should render the hero name in an anchor tag', () => {
    fixture.componentInstance.hero = { id: 1, name: 'SuperDude', strength: 3};
    fixture.detectChanges();

    let deA = fixture.debugElement.query(By.css('a'));
    expect(deA.nativeElement.textContent).toContain('SuperDude');

    // expect(fixture.nativeElement.querySelector('a').textContent).toContain('SuperDude');
  })
})
```

Integration Testing for HeroesComponent.

1. Create a separate file for Integration test as **heroes.component.integration.spec.ts**

```
describe('HeroesComponent (shallow tests)', () => {
  let fixture: ComponentFixture<HeroesComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [HeroesComponent]
    })
    fixture = TestBed.createComponent(HeroesComponent);
  })
})
```

2. Just check if everything is initialized correctly and loaded for testing.

```
describe('HeroesComponent (shallow tests)', () => {
  let fixture: ComponentFixture<HeroesComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [HeroesComponent],
      providers: []
    })
    fixture = TestBed.createComponent(HeroesComponent);
  });

  it('should do nothing', () => {
    expect(true).toBe(true);
  })
})
```

You get error for app-hero component as child component is not loaded for testing.

3. Solution – add NO_ERRORS_SCHEMA to ignore errors.

```
describe('HeroesComponent (shallow tests)', () => {
  let fixture: ComponentFixture<HeroesComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [HeroesComponent],
      providers: [],
      schemas: [NO_ERRORS_SCHEMA]
    })
    fixture = TestBed.createComponent(HeroesComponent);
  });

  it('should do nothing', () => {
    expect(true).toBe(true);
  })
})
```

4. Again, you get error for Injector Service.
   To resolve, add providers to TestingModule.

```
describe('HeroesComponent (shallow tests)', () => {
  let fixture: ComponentFixture<HeroesComponent>;
  let mockHeroService;

  beforeEach(() => {
    mockHeroService = jasmine.createSpyObj(['getHeroes', 'addHero', 'deleteHero']);

    TestBed.configureTestingModule({
      declarations: [HeroesComponent],
      providers: [
        { provide: HeroService, useValue: mockHeroService }
      ],
      schemas: [NO_ERRORS_SCHEMA]
    })
    fixture = TestBed.createComponent(HeroesComponent);
  });
```

5. Now, lets write the test to check if getHeroes is fetching data correctly from mocked service.

```
it('should set heroes correctly from the service', () => {
  mockHeroService.getHeroes.and.returnValue(of())
  expect(true).toBe(true);
})
```

This returned value should be Observable with dummy data.

```
let HEROES;

beforeEach(() => {
  HEROES = [
    {id:1, name: 'SpiderDude', strength: 8},
    {id:2, name: 'Wonderful Woman', strength: 24},
    {id:3, name: 'SuperDude', strength: 55}
  ]
```

Now, complete the test with HEROES.

```
it('should set heroes correctly from the service', () => {
  mockHeroService.getHeroes.and.returnValue(of(HEROES))
  fixture.detectChanges();

  expect(fixture.componentInstance.heroes.length).toBe(3);
})
```

6. To test Child Component, we can mock it and then test. Hence, remove NO_ERROR_SCHEMA to test child component.

Test Components with Templates and Child Components

```
describe('HeroesComponent (shallow tests)', () => {
  let fixture: ComponentFixture<HeroesComponent>;
  let mockHeroService;
  let HEROES;

  @Component({
    selector: 'app-hero',
    template: '<div></div>',
  })
  class FakeHeroComponent {
    @Input() hero: Hero;
    // @Output() delete = new EventEmitter();
  }

  beforeEach(() => {
```

Now, declare the FakeHeroComponent inside the TestingModule.

```
  TestBed.configureTestingModule({
    declarations: [
      HeroesComponent,
      FakeHeroComponent
    ],
    providers: [
      { provide: HeroService, useValue: mockHeroService }
    ],
    // schemas: [NO_ERRORS_SCHEMA]
  })
  fixture = TestBed.createComponent(HeroesComponent);
});
```

7. Write test to check the <li> element is generated for each hero component.

```
it('should create one li for each hero', () => {
  mockHeroService.getHeroes.and.returnValue(of(HEROES))
  fixture.detectChanges();

  expect(fixture.debugElement.queryAll(By.css('li')).length).toBe(3);
})
```