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, let's 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);
})
```

Testing Services and HttpClient

1. Create a file named hero.service.spec.ts
2. Add the following code:

```
import { TestBed } from "@angular/core/testing";
import { HeroService } from "./hero.service";
import { MessageService } from "./message.service";
import { HttpClientTestingModule } from "@angular/common/http/testing";

describe('HeroService', () => {
  let mockMessageService;

  beforeEach(() => {
    mockMessageService = jasmine.createSpyObj(['add']);

    TestBed.configureTestingModule({
      imports: [ HttpClientTestingModule ],
      providers: [
        HeroService,
        {provide: MessageService, useValue: mockMessageService}
      ]
    })
  })
})
```

3. To get handle to the mock HttpClient service, so that we can adjust it and control it inside our tests. Add HttpTestingController as a special controller in the test.

```
import { HttpClientTestingModule, HttpTestingController } from
  "@angular/common/http/testing";

describe('HeroService', () => {
  let mockMessageService;
  let httpTestingController: HttpTestingController;

  beforeEach(() => {
    mockMessageService = jasmine.createSpyObj(['add']);

    TestBed.configureTestingModule({
      imports: [ HttpClientTestingModule ],
      providers: [
```

4. Inside beforeEach(), create the instance of HttpTestingController.

```
beforeEach(() => {
  mockMessageService = jasmine.createSpyObj(['add']);

  TestBed.configureTestingModule({
    imports: [ HttpClientTestingModule ],
    providers: [
      HeroService,
      {provide: MessageService, useValue: mockMessageService}
    ]
  });

  httpTestingController = TestBed.get(HttpTestingController);
})
```

5. Write the test case to check if calling is sent to the right URL. Here, the call will be sent to the actual backend code.

```
  httpTestingController = TestBed.get(HttpTestingController);
  service = TestBed.get(HeroService);
});

describe('getHero', () => {

  it('should call get with the correct URL',I() => {

    service.getHero(4).subscribe();


  });
})
```

6. Handling the HTTPClient Request to send back the required data to the service.

```
describe('getHero', () => {

  it('should call get with the correct URL', () => {

    service.getHero(4).subscribe();

    const req = httpTestingController.expectOne('api/heroes/4');
    req.flush({id: 4, name: 'SuperDude', strength: 100});
  });
})
```

7. Making multiple calls to HttpClient.

```
describe('getHero', () => {

  it('should call get with the correct URL', () => {

    service.getHero(4).subscribe();
    service.getHero(3).subscribe();

    const req = httpTestingController.expectOne('api/heroes/4');
    req.flush({id: 4, name: 'SuperDude', strength: 100});
    httpTestingController.verify();
  });
})
```

Mocking RouterLink

1. Create a fake router link in the **hero.component.integration.spec.ts**.

```
import { HeroComponent } from "../hero/hero.component";

@Directive({
  selector: '[routerLink]',

})
export class RouterLinkDirectiveStub {
  @Input('routerLink') linkParams: any;
  navigatedTo: any = null;

  onClick() {
    this.navigatedTo = this.linkParams;
  }
}

describe('HeroesComponent (deep tests)', () => {
```

2. Add a HostListener to listen the click event on parent component DOM node and wben anchor tag is clicked, fire my OnClick() function.

```
@Directive({
  selector: '[routerLink]',
  host: { '(click)': 'onClick()' }
})
export class RouterLinkDirectiveStub {
  @Input('routerLink') linkParams: any;
  navigatedTo: any = null;

  onClick() {
    this.navigatedTo = this.linkParams;
  }
}
```

3. Register the RouterLinkDirectiveStub component.

```
TestBed.configureTestingMod
  declarations: [
    HeroesComponent,
    HeroComponent,
    RouterLinkDirectiveStub
  ],
```

Testing the RouterLink

1. In **heroes.component.integration.spec.ts** file, write a deep integration test to check when the hero component was clicked, does it change the router URL.

```
  it('should have the correct route for the first hero', () => {
    mockHeroService.getHeroes.and.returnValue(of(HEROES));
    fixture.detectChanges();
    const heroComponents = fixture.debugElement.queryAll(By.directive(HeroComponent));

    let routerLink = heroComponents[0]
      .query(By.directive(RouterLinkDirectiveStub))
      .injector.get(RouterLinkDirectiveStub);

    heroComponents[0].query(By.css('a')).triggerEventHandler('click', null);

    expect(routerLink.navigatedTo).toBe('/detail/1');
  })

});
```

Testing ActivateRoute in hero-detail.component.ts

1. Create a new file in the hero-detail component. Name it as **hero-detail.component.ts**

2. Create three MockService, ActivateRoute and Location in the test.

   Write the following code:

```
import { TestBed } from "@angular/core/testing";
import { HeroDetailComponent } from "./hero-detail.component";

describe('HeroDetailComponent', () => {
  let mockActivatedRoute, mockHeroService, mockLocation;

  beforeEach(() => {
    mockActivatedRoute = {
      snapshot: { paramMap: { get: () => { return '3'; }}}
    }
    mockHeroService = jasmine.createSpyObj(['getHero', 'updateHero']);
    mockLocation = jasmine.createSpyObj(['back']);

    TestBed.configureTestingModule({
      declarations: [HeroDetailComponent],
      providers: [

      ]
    })
  })
})
```

3, Now, register the mocks in TestBed. Check from where the service will come when call will be sent to them.

```
import { TestBed } from "@angular/core/testing";
import { HeroDetailComponent } from "./hero-detail.component";
import { ActivatedRoute } from "@angular/router";
import { HeroService } from "../hero.service";
import { Location } from '@angular/common';
```

```
TestBed.configureTestingModule({
  declarations: [HeroDetailComponent],
  providers: [
    {provide: ActivatedRoute, useValue: mockActivatedRoute},
    {provide: HeroService, useValue: mockHeroService},
    {provide: Location, useValue: mockLocation},
  ]
})
```

3. Declare fixture and create the component instance.

```
    TestBed.configureTestingModule({
      declarations: [HeroDetailComponent],
      providers: [
        {provide: ActivatedRoute, useValue: mockActivatedRoute},
        {provide: HeroService, useValue: mockHeroService},
        {provide: Location, useValue: mockLocation},
      ]
    });
    fixture = TestBed.createComponent(HeroDetailComponent);
  })
```

4. Write the test to check if the h2 tag is populated with the name of hero. To do that mock the service object and run change detection to validate h2 tag is populated with hero.name property value.

```
  fixture = TestBed.createComponent(HeroDetailComponent);

  mockHeroService.getHero.and.returnValue(of({id: 3, name: 'SuperDude', strength: 100}));
});

it('should render hero name in a h2 tag', () => {
  fixture.detectChanges();

  expect(fixture.nativeElement.querySelector('h2').textContent).toContain('SUPERDUDE');
})
```

Oh!!!! Test fails..why??????

5. We are not importing FormsModule in our test.

```
TestBed.configureTestingModule({
  imports: [FormsModule],
  declarations: [HeroDetailComponent],
  providers: [
    {provide: ActivatedRoute, useValue: mockActivatedRoute},
    {provide: HeroService, useValue: mockHeroService},
    {provide: Location, useValue: mockLocation},
  ]
});
```

Testing Input Boxes.

2. Here, we can check the template of the heroes.component.html

```
3. div>
4.   <label>Hero name:
5.     <input #heroName />
6.   </label>
```

You can see there is <input> element which accepts value that will be added in the hero list.

4. Write a test inside the **heroes.component.integration.spec.ts** file to check if new value is added on button click and added in the list of <hero> elements in the parent component.

```
it('should add a new hero to the hero list when the add button is clicked', () => {
  mockHeroService.getHeroes.and.returnValue(of(HEROES));
  fixture.detectChanges();
  const name = "Mr. Ice";
  mockHeroService.addHero.and.returnValue(of({id: 5, name: name, strength: 4}));
  const inputElement = fixture.debugElement.query(By.css('input')).nativeElement;
  const addButton = fixture.debugElement.queryAll(By.css('button'))[0];

  inputElement.value = name;
  addButton.triggerEventHandler('click', null);
  fixture.detectChanges();

  const heroText = fixture.debugElement.query(By.css('ul')).nativeElement.textContent;
  expect(heroText).toContain(name);
})
```