Here we will test the heroes component.

1. Create a file named **heroes.component.spec.ts** under heroes folder.
2. Write code to setup test.

```
import { HeroesComponent } from "./heroes.component";

describe("HeroesComponent Test Suite", () => {
  // declare the variable of type HeroesComponent
  let compoenent: HeroesComponent;

  // The HeroesComponent has  heroes: Hero[]; property
  // For testing we need to initialize the heroes with dummy data
  let HEROES;

  beforeEach(() => {
    HEROES = [
      { id: 100, name: "Spider Man", strength: 8 },
      { id: 200, name: "Wonder Woman", strength: 43 },
      { id: 300, name: "Bat Man", strength: 55 },
    ];


  });
});
```

3. Now, we need to instantiate the component object, which has a dependency on HeroService object.

```
constructor(private heroService: HeroService) { }
```

So, we need to provide the service object to the component during testing. Since, we don't want to test two units at the same time. We want to test only one, that is, HeroesComponent.

Therefore, we need to fake **HeroesService**.

To mock the **HelloService**, use **jasmine.createspyObject().**

```
import { HeroService } from "../hero.service";
import { HeroesComponent } from "./heroes.component";

describe("HeroesComponent Test Suite", () => {
  // declare the variable of type HeroesComponent
  let component: HeroesComponent;
  let mockHeroService;

  // The HeroesComponent has  heroes: Hero[]; property
  // For testing we need to initialize the heroes with dummy data
  let HEROES;

  beforeEach(() => {
    HEROES = [
      { id: 100, name: "Spider Man", strength: 8 },
      { id: 200, name: "Wonder Woman", strength: 43 },
      { id: 300, name: "Bat Man", strength: 55 },
    ];

    mockHeroService = jasmine.createSpyObj([
      "getHeroes",
      "addHero",
      "deleteHero",
    ]);

    component = new HeroesComponent(mockHeroService);
  });
});
```

4. Next, we need to write testcase to test delete() method of HeroesComponent.

```
describe("delete", () => {
    it("should remove the indicated hero from the heroes list", () => {
      // Arrange
      component.heroes = HEROES;

      // Act
      component.delete(HEROES[2]);
```

```
    // Assert
    expect(component.heroes.length).toBe(2)
  });
```

5. We need to mock object to return an Observable when deleteHero is called. Since, we are just subscribing to that result and not doing anything on the data. We can mock it to return some value.

```
describe("delete", () => {
    it("should remove the indicated hero from the heroes list", () =>
 {
      mockHeroService.deleteHero.and.returnValue(of(true));

      // Arrange
      component.heroes = HEROES;

      // Act
      component.delete(HEROES[2]);

      // Assert
      expect(component.heroes.length).toBe(2);
    });
```

Testing Interaction

In our previous test case, as we mocked the deleteHero() method of service and just returned true on its invocation, so we did not check whether method was called or not.

```
delete(hero: Hero): void {
   this.heroes = this.heroes.filter(h => h !== hero);
   this.heroService.deleteHero(hero).subscribe();
 }
```

Check deleteHero() was called with right parameters.

```
it('should call deleteHero', () => {
  mockHeroService.deleteHero.and.returnValue(of(true))
  component.heroes = HEROES;

  component.delete(HEROES[2]);

  expect(mockHeroService.deleteHero).toHaveBeenCalled();
})
```