Master/Detail Components

At the moment, the `HeroesComponent` displays both the list of heroes and the selected hero's details.

Keeping all features in one component as the application grows will not be maintainable. You'll want to split up large components into smaller sub-components, each focused on a specific task or workflow.

In this page, you'll take the first step in that direction by moving the hero details into a separate, reusable `HeroDetailComponent`.

The `HeroesComponent` will only present the list of heroes. The `HeroDetailComponent` will present details of a selected hero.

Make the `HeroDetailComponent`

Use the Angular CLI to generate a new component named `hero-detail`.

```
ng generate component hero-detail
```

The command scaffolds the following:

- Creates a directory `src/app/hero-detail`.

Inside that directory four files are generated:

- A CSS file for the component styles.
- An HTML file for the component template.
- A TypeScript file with a component class named `HeroDetailComponent`.
- A test file for the `HeroDetailComponent` class.

The command also adds the `HeroDetailComponent` as a declaration in the `@NgModule` decorator of the `src/app/app.module.ts` file.

Write the template

Cut the HTML for the hero detail from the bottom of the `HeroesComponent` template and paste it over the generated boilerplate in the `HeroDetailComponent` template.

The pasted HTML refers to a `selectedHero`. The new `HeroDetailComponent` can present *any* hero, not just a selected hero. So replace 'selectedHero' with 'hero' everywhere in the template.

When you're done, the `HeroDetailComponent` template should look like this:

src/app/hero-detail/hero-detail.component.html

```
<div *ngIf="hero">

  <h2>{{hero.name | uppercase}} Details</h2>
  <div><span>id: </span>{{hero.id}}</div>
  <div>
    <label>name:
      <input [(ngModel)]="hero.name" placeholder="name"/>
    </label>
  </div>

</div>
```

Add the `@Input()` hero property

The `HeroDetailComponent` template binds to the component's `hero` property which is of type `Hero`.

Open the `HeroDetailComponent` class file and import the `Hero` symbol.

src/app/hero-detail/hero-detail.component.ts (import Hero)

```
import { Hero } from '../hero';
```

The `hero` property must be an *Input* property, annotated with the `@Input()` decorator, because the *external* `HeroesComponent` will bind to it like this.

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

Amend the `@angular/core` import statement to include the `Input` symbol.

src/app/hero-detail/hero-detail.component.ts (import Input)

```
import { Component, OnInit, Input } from '@angular/core';
```

Add a `hero` property, preceded by the `@Input()` decorator.

src/app/hero-detail/hero-detail.component.ts

```
@Input() hero: Hero;
```

That's the only change you should make to the `HeroDetailComponent` class. There are no more properties. There's no presentation logic. This component simply receives a hero object through its `hero` property and displays it.

Show the `HeroDetailComponent`

The `HeroesComponent` is still a master/detail view.

It used to display the hero details on its own, before you cut that portion of the template. Now it will delegate to the `HeroDetailComponent`.

The two components will have a parent/child relationship. The parent `HeroesComponent` will control the child `HeroDetailComponent` by sending it a new hero to display whenever the user selects a hero from the list.

You won't change the `HeroesComponent` *class* but you will change its *template*.

Update the `HeroesComponent` template

The `HeroDetailComponent` selector is `'app-hero-detail'`. Add an `<app-hero-detail>` element near the bottom of the `HeroesComponent` template, where the hero detail view used to be.

Bind the `HeroesComponent.selectedHero` to the element's `hero` property like this.

heroes.component.html (HeroDetail binding)

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

`[hero]="selectedHero"` is an Angular property binding.

It's a *one way* data binding from the `selectedHero` property of the `HeroesComponent` to the `hero` property of the target element, which maps to the `hero` property of the `HeroDetailComponent`.

Now when the user clicks a hero in the list, the `selectedHero` changes. When the `selectedHero` changes, the *property binding* updates `hero` and the `HeroDetailComponent` displays the new hero.

The revised `HeroesComponent` template should look like this:

heroes.component.html

```
<h2>My Heroes</h2>

<ul class="heroes">
  <li *ngFor="let hero of heroes"
    [class.selected]="hero === selectedHero"
    (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

The browser refreshes and the app starts working again as it did before.

What changed?

As before, whenever a user clicks on a hero name, the hero detail appears below the hero list. Now the `HeroDetailComponent` is presenting those details instead of the `HeroesComponent`.

Refactoring the original `HeroesComponent` into two components yields benefits, both now and in the future:

1. You simplified the `HeroesComponent` by reducing its responsibilities.
2. You can evolve the `HeroDetailComponent` into a rich hero editor without touching the parent `HeroesComponent`.
3. You can evolve the `HeroesComponent` without touching the hero detail view.
4. You can re-use the `HeroDetailComponent` in the template of some future component.

Final code review

Here are the code files discussed on this page and your app should look like this live example / download example.

src/app/hero-detail/hero-detail.component.ts | src/app/hero-detail/hero-detail.component.html | src

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-hero-detail',
  templateUrl: './hero-detail.component.html',
  styleUrls: ['./hero-detail.component.css']
})
export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;

  constructor() { }

  ngOnInit() {
  }

}
```

Summary

- You created a separate, reusable `HeroDetailComponent`.
- You used a property binding to give the parent `HeroesComponent` control over the child `HeroDetailComponent`.
- You used the `@Input` decorator to make the `hero` property available for binding by the external `HeroesComponent`.