# Fundamentals

## Tour of Heroes App: Display a selection list

In this page, you'll expand the Tour of Heroes app to display a list of heroes, and allow users to select a hero and display the hero's details.

## Create mock heroes

You'll need some heroes to display.

Eventually you'll get them from a remote data server. For now, you'll create some mock heroes and pretend they came from the server.

Create a file called mock-heroes.ts in the src/app/ folder. Define a HEROES constant as an array of ten heroes and export it. The file should look like this.

src/app/mock-heroes.ts

content\_copyimport { Hero } from './hero';

export const HEROES: Hero[] = [

{ id: 11, name: 'Dr Nice' },

{ id: 12, name: 'Narco' },

{ id: 13, name: 'Bombasto' },

{ id: 14, name: 'Celeritas' },

{ id: 15, name: 'Magneta' },

{ id: 16, name: 'RubberMan' },

{ id: 17, name: 'Dynama' },

{ id: 18, name: 'Dr IQ' },

{ id: 19, name: 'Magma' },

{ id: 20, name: 'Tornado' }

];

## Displaying heroes

Open the HeroesComponent class file and import the mock HEROES.

src/app/heroes/heroes.component.ts (import HEROES)

content\_copyimport { HEROES } from '../mock-heroes';

In the same file (HeroesComponent class), define a component property called heroes to expose the HEROES array for binding.

src/app/heroes/heroes.component.ts

content\_copyexport class HeroesComponent implements [OnInit](https://angular.io/api/core/OnInit) {

heroes = HEROES;

}

### List heroes with \*[ngFor](https://angular.io/api/common/NgForOf)

Open the HeroesComponent template file and make the following changes:

* Add an <h2> at the top,
* Below it add an HTML unordered list (<ul>)
* Insert an <li> within the <ul> that displays properties of a hero.
* Sprinkle some CSS classes for styling (you'll add the CSS styles shortly).

Make it look like this:

heroes.component.html (heroes template)

content\_copy<h2>My Heroes</h2>

<ul class="heroes">

<li>

<span class="badge">{{hero.id}}</span> {{hero.name}}

</li>

</ul>

That shows one hero. To list them all, add an \*[ngFor](https://angular.io/api/common/NgForOf) to the <li> to iterate through the list of heroes:

content\_copy<li \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes">

The [\*ngFor](https://angular.io/guide/template-syntax#ngFor) is Angular's repeater directive. It repeats the host element for each element in a list.

The syntax in this example is as follows:

* <li> is the host element.
* heroes holds the mock heroes list from the HeroesComponent class, the mock heroes list.
* hero holds the current hero object for each iteration through the list.

Don't forget the asterisk (\*) in front of [ngFor](https://angular.io/api/common/NgForOf). It's a critical part of the syntax.

After the browser refreshes, the list of heroes appears.

### Style the heroes

The heroes list should be attractive and should respond visually when users hover over and select a hero from the list.

In the [first tutorial](https://angular.io/tutorial/toh-pt0#app-wide-styles), you set the basic styles for the entire application in styles.css. That stylesheet didn't include styles for this list of heroes.

You could add more styles to styles.css and keep growing that stylesheet as you add components.

You may prefer instead to define private styles for a specific component and keep everything a component needs— the code, the HTML, and the CSS —together in one place.

This approach makes it easier to re-use the component somewhere else and deliver the component's intended appearance even if the global styles are different.

You define private styles either inline in the @[Component.styles](https://angular.io/api/core/Component#styles) array or as stylesheet file(s) identified in the @[Component.styleUrls](https://angular.io/api/core/Component#styleUrls) array.

When the CLI generated the HeroesComponent, it created an empty heroes.component.css stylesheet for the HeroesComponent and pointed to it in @[Component.styleUrls](https://angular.io/api/core/Component#styleUrls) like this.

src/app/heroes/heroes.component.ts (@Component)

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-heroes',

[templateUrl](https://angular.io/api/core/Component#templateUrl): './heroes.component.html',

[styleUrls](https://angular.io/api/core/Component#styleUrls): ['./heroes.component.css']

})

Open the heroes.component.css file and paste in the private CSS styles for the HeroesComponent. You'll find them in the [final code review](https://angular.io/tutorial/toh-pt2#final-code-review) at the bottom of this guide.

Styles and stylesheets identified in @[Component](https://angular.io/api/core/Component) metadata are scoped to that specific component. The heroes.component.css styles apply only to the HeroesComponent and don't affect the outer HTML or the HTML in any other component.

## Master/Detail

When the user clicks a hero in the **master** list, the component should display the selected hero's **details** at the bottom of the page.

In this section, you'll listen for the hero item click event and update the hero detail.

### Add a click event binding

Add a click event binding to the <li> like this:

heroes.component.html (template excerpt)

content\_copy<li \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes" (click)="onSelect(hero)">

This is an example of Angular's [event binding](https://angular.io/guide/template-syntax#event-binding) syntax.

The parentheses around click tell Angular to listen for the <li> element's click event. When the user clicks in the <li>, Angular executes the onSelect(hero) expression.

In the next section, define an onSelect() method in HeroesComponent to display the hero that was defined in the \*[ngFor](https://angular.io/api/common/NgForOf) expression.

### Add the click event handler

Rename the component's hero property to selectedHero but don't assign it. There is no selected hero when the application starts.

Add the following onSelect() method, which assigns the clicked hero from the template to the component's selectedHero.

src/app/heroes/heroes.component.ts (onSelect)

content\_copyselectedHero: Hero;

onSelect(hero: Hero): void {

this.selectedHero = hero;

}

### Add a details section

Currently, you have a list in the component template. To click on a hero on the list and reveal details about that hero, you need a section for the details to render in the template. Add the following to heroes.component.html beneath the list section:

heroes.component.html (selected hero details)

content\_copy<h2>{{selectedHero.name | [uppercase](https://angular.io/api/common/UpperCasePipe)}} Details</h2>

<div><span>id: </span>{{selectedHero.id}}</div>

<div>

<label>name:

<input [([ngModel](https://angular.io/api/forms/NgModel))]="selectedHero.name" placeholder="name"/>

</label>

</div>

After the browser refreshes, the application is broken.

Open the browser developer tools and look in the console for an error message like this:

content\_copyHeroesComponent.html:3 ERROR TypeError: Cannot read property 'name' of undefined

#### What happened?

When the app starts, the selectedHero is undefined by design.

Binding expressions in the template that refer to properties of selectedHero—expressions like {{selectedHero.name}}—must fail because there is no selected hero.

#### The fix - hide empty details with \*ngIf

The component should only display the selected hero details if the selectedHero exists.

Wrap the hero detail HTML in a <div>. Add Angular's \*[ngIf](https://angular.io/api/common/NgIf) directive to the <div> and set it to selectedHero.

Don't forget the asterisk (\*) in front of [ngIf](https://angular.io/api/common/NgIf). It's a critical part of the syntax.

src/app/heroes/heroes.component.html (\*ngIf)

content\_copy<div \*[ngIf](https://angular.io/api/common/NgIf)="selectedHero">

<h2>{{selectedHero.name | [uppercase](https://angular.io/api/common/UpperCasePipe)}} Details</h2>

<div><span>id: </span>{{selectedHero.id}}</div>

<div>

<label>name:

<input [([ngModel](https://angular.io/api/forms/NgModel))]="selectedHero.name" placeholder="name"/>

</label>

</div>

</div>

After the browser refreshes, the list of names reappears. The details area is blank. Click a hero in the list of heroes and its details appear. The app seems to be working again. The heroes appear in a list and details about the clicked hero appear at the bottom of the page.

#### Why it works

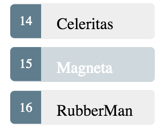
When selectedHero is undefined, the [ngIf](https://angular.io/api/common/NgIf) removes the hero detail from the DOM. There are no selectedHero bindings to consider.

When the user picks a hero, selectedHero has a value and [ngIf](https://angular.io/api/common/NgIf) puts the hero detail into the DOM.

### Style the selected hero

It's difficult to identify the selected hero in the list when all <li> elements look alike.

If the user clicks "Magneta", that hero should render with a distinctive but subtle background color like this:



That selected hero coloring is the work of the .selected CSS class in the [styles you added earlier](https://angular.io/tutorial/toh-pt2#styles). You just have to apply the .selected class to the <li> when the user clicks it.

The Angular [class binding](https://angular.io/guide/template-syntax#class-binding) makes it easy to add and remove a CSS class conditionally. Just add [class.some-css-class]="some-condition" to the element you want to style.

Add the following [class.selected] binding to the <li> in the HeroesComponent template:

heroes.component.html (toggle the 'selected' CSS class)

content\_copy[class.selected]="hero === selectedHero"

When the current row hero is the same as the selectedHero, Angular adds the selected CSS class. When the two heroes are different, Angular removes the class.

The finished <li> looks like this:

heroes.component.html (list item hero)

content\_copy<li \*[ngFor](https://angular.io/api/common/NgForOf)="let hero of heroes"

[class.selected]="hero === selectedHero"

(click)="onSelect(hero)">

<span class="badge">{{hero.id}}</span> {{hero.name}}

</li>

## Final code review

Your app should look like this [live example](https://angular.io/generated/live-examples/toh-pt2/stackblitz.html) / [download example](https://angular.io/generated/zips/toh-pt2/toh-pt2.zip).

Here are the code files discussed on this page, including the HeroesComponent styles.

src/app/mock-heroes.ts

src/app/heroes/heroes.component.ts

src/app/heroes/heroes.component.html

src/app/heroes/heroes.component.css

content\_copyimport { Hero } from './hero';

export const HEROES: Hero[] = [

{ id: 11, name: 'Dr Nice' },

{ id: 12, name: 'Narco' },

{ id: 13, name: 'Bombasto' },

{ id: 14, name: 'Celeritas' },

{ id: 15, name: 'Magneta' },

{ id: 16, name: 'RubberMan' },

{ id: 17, name: 'Dynama' },

{ id: 18, name: 'Dr IQ' },

{ id: 19, name: 'Magma' },

{ id: 20, name: 'Tornado' }

];

## Summary

* The Tour of Heroes app displays a list of heroes in a Master/Detail view.
* The user can select a hero and see that hero's details.
* You used \*[ngFor](https://angular.io/api/common/NgForOf) to display a list.
* You used \*[ngIf](https://angular.io/api/common/NgIf) to conditionally include or exclude a block of HTML.
* You can toggle a CSS style class with a class binding.