Create the employee component

Using the Angular CLI, generate a new component named employee.

ng generate component employee

The CLI creates a new folder, src/app/employee/ and generates the three files of the EmployeeComponent.

The EmployeeComponent class file is as follows:

app/employee/employee.component.ts (initial version)

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

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrls: ['./employee.component.css']

})

export class EmployeeComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

You always import the Component symbol from the Angular core library and annotate the component class with @Component.

@Component is a decorator function that specifies the Angular metadata for the component.

The CLI generated three metadata properties:

1. selector— the component's CSS element selector
2. templateUrl— the location of the component's template file.
3. styleUrls— the location of the component's private CSS styles.

The CSS element selector, 'app-employee', matches the name of the HTML element that identifies this component within a parent component's template.

The ngOnInit is a lifecycle hook Angular calls ngOnInit shortly after creating a component. It's a good place to put initialization logic.

Always export the component class so you can import it elsewhere ... like in the AppModule.

Add a *employee* property

Add a employee property to the EmployeeComponent for a employee named "Windstorm."

employee.component.ts (employee property)

employee = 'Windstorm';

Show the employee

Open the employee.component.html template file. Delete the default text generated by the Angular CLI and replace it with a data binding to the new employee property.

employee.component.html

{{employee}}

Show the *EmployeeComponent* view

To display the EmployeeComponent, you must add it to the template of the shell AppComponent.

Remember that app-employee is the element selector for the EmployeeComponent. So add an <app-employee> element to the AppComponenttemplate file, just below the title.

src/app/app.component.html

<h1>{{title}}</h1>

<app-employee></app-employee>

Assuming that the CLI ng serve command is still running, the browser should refresh and display both the application title and the employee name.

Create a Employee class

A real employee is more than a name.

Create a Employee class in its own file in the src/app folder. Give it id and name properties.

src/app/employee.ts

export class Employee {

id: number;

name: string;

}

Return to the EmployeeComponent class and import the Employee class.

Refactor the component's employee property to be of type Employee. Initialize it with an id of 1 and the name Windstorm.

The revised EmployeeComponent class file should look like this:

src/app/employee/employee.component.ts

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

import { Employee } from '../employee';

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrls: ['./employee.component.css']

})

export class EmployeeComponent implements OnInit {

employee: Employee = {

id: 1,

name: 'Windstorm'

};

constructor() { }

ngOnInit() {

}

}

The page no longer displays properly because you changed the employee from a string to an object.

Show the employee object

Update the binding in the template to announce the employee's name and show both id and name in a details layout like this:

employee.component.html (EmployeeComponent's template)

<h2>{{ employee.name }} Details</h2>

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

<div><span>name: </span>{{employee.name}}</div>

The browser refreshes and display's the employee's information.

Format with the *UppercasePipe*

Modify the employee.name binding like this.

<h2>{{ employee.name | uppercase }} Details</h2>

The browser refreshes and now the employee's name is displayed in capital letters.

The word uppercase in the interpolation binding, right after the pipe operator ( | ), activates the built-in UppercasePipe.

Pipes are a good way to format strings, currency amounts, dates and other display data. Angular ships with several built-in pipes and you can create your own.

Edit the employee

Users should be able to edit the employee name in an <input> textbox.

The textbox should both *display* the employee's name property and *update* that property as the user types. That means data flow from the component class *out to the screen* and from the screen *back to the class*.

To automate that data flow, setup a two-way data binding between the <input> form element and the employee.name property.

Two-way binding

Refactor the details area in the EmployeeComponent template so it looks like this:

src/app/employee/employee.component.html (EmployeeComponent's template)

<div>

<label>name:

<input [(ngModel)]="employee.name" placeholder="name">

</label>

</div>

[(ngModel)] is Angular's two-way data binding syntax.

Here it binds the employee.name property to the HTML textbox so that data can flow *in both directions:* from the employee.name property to the textbox, and from the textbox back to the employee.name.

The missing *FormsModule*

Notice that the app stopped working when you added [(ngModel)].

To see the error, open the browser development tools and look in the console for a message like

Template parse errors:

Can't bind to 'ngModel' since it isn't a known property of 'input'.

Although ngModel is a valid Angular directive, it isn't available by default.

It belongs to the optional FormsModule and you must *opt-in* to using it.

*AppModule*

Angular needs to know how the pieces of your application fit together and what other files and libraries the app requires. This information is called *metadata*

Some of the metadata is in the @Component decorators that you added to your component classes. Other critical metadata is in @NgModule decorators.

The most important @NgModuledecorator annotates the top-level AppModule class.

The Angular CLI generated an AppModule class in src/app/app.module.ts when it created the project. This is where you *opt-in* to the FormsModule.

Import *FormsModule*

Open AppModule (app.module.ts) and import the FormsModule symbol from the @angular/forms library.

app.module.ts (FormsModule symbol import)

import { FormsModule } from '@angular/forms'; // <-- NgModel lives here

Then add FormsModule to the @NgModule metadata's imports array, which contains a list of external modules that the app needs.

app.module.ts ( @NgModule imports)

imports: [

BrowserModule,

FormsModule

],

When the browser refreshes, the app should work again. You can edit the employee's name and see the changes reflected immediately in the <h2> above the textbox.

Declare *EmployeeComponent*

Every component must be declared in *exactly one* NgModule.

*You* didn't declare the EmployeeComponent. So why did the application work?

It worked because the Angular CLI declared EmployeeComponent in the AppModule when it generated that component.

Open src/app/app.module.ts and find EmployeeComponent imported near the top.

import { EmployeeComponent } from './employee/employee.component';

The EmployeeComponent is declared in the @NgModule.declarations array.

declarations: [

AppComponent,

EmployeeComponent

],

Note that AppModule declares both application components, AppComponent and EmployeeComponent.

Final code review

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

src/app/employee/employee.component.ts

src/app/employee/employee.component.html

src/app/app.module.ts

src/app/app.component.ts

src/app/app.component.html

src/app/employee.ts

1. import { Component, OnInit } from '@angular/core';
2. import { Employee } from '../employee';
4. @Component({
5. selector: 'app-employee',
6. templateUrl: './employee.component.html',
7. styleUrls: ['./employee.component.css']
8. })
9. export class EmployeeComponent implements OnInit {
10. employee: Employee = {
11. id: 1,
12. name: 'Windstorm'
13. };
15. constructor() { }
17. ngOnInit() {
18. }
20. }