Displaying Data using Templates

- Adding Properties to Components and Interpolation
- Creating a class for data (Model object)
- Template Expressions
- Working with Arrays and Build-in Directives
    - Repeating Directive (NgFor)
    - Conditional Directives (NgIf, NgSwitch)
- * vs <template>
- External HTML Template File

About Directives

Directives are the classes that can change the behaviour or appearance of the components by using CSS Classes, CSS Styles and events.

There are three kinds of directives in Angular:

1. **Components**
    1. Directives with a template.
    2. These are most commonly used Directive
2. **Structural directives**
    1. Change the DOM layout by adding and removing DOM elements.
    2. Change the structure of the view
    3. Example: ngFor and ngIf
3. **Attribute directives**
    1. Change the appearance or behavior of an element.
    2. Are used as attributes of elements
    3. Eg: ngStyle can change several element styles at the same time

Component Properties and Interpolation

The easiest way to display a component property is to bind the property name through interpolation.

Angular evaluates all expressions in double curly braces, converts the expression results to strings, and links them with neighboring literal strings. Finally, it assigns this composite interpolated result to an **element or directive property**

Edit File: \src\app\app.component.cs

```
@Component({
 selector: 'my-app',
 template: `<h1>{{CourseName}} Description</h1>
  <p>{{Description}}</p>
  <p>The sum of 1 + 1 is {{1 + 1 }}</p>`,
```

```
})
export class AppComponent
{
   CourseName: string = 'Angular';
   Description: string = "This is a demo of angular course"
}
```

Built-in Directives - *ngIF

Only ngIf:

```
ShowFaculty: boolean = false;
Faculty: string = "Sandeep Soni";
```

```
<span *ngIf="ShowFaculty">Faculty: {{Faculty}}</span>
```

ngIf else

If the condition is true it render the inner content else render the associated template.

```
<span *ngIf="ShowFaculty; else tmpAnyFaculty">Faculty: {{Faculty}}</span>
<ng-template #tmpAnyFaculty>
  Faculty: Any one can teach.
</ng-template>
```

ngIf then else

Ng if then else: If the condition is true it renders template1 else renders template2

File: **directives.html**

```
<span *ngIf="ShowFaculty; else tmpAnyFaculty">Faculty: {{Faculty}}</span>
<ng-template #tmpSpecificFaculty>
  Faculty: {{Faculty}}
</ng-template>
<ng-template #tmpAnyFaculty>
  Faculty: Any one can teach.
</ng-template>
```

Built-in Directives - *ngFor

- ***ngFor** is an angular built in directive that outputs an array element, **iterating through each element** inside it.
- Here **let** is a typescript syntax for declaring local variables and it can be referred anywhere in the specific template.
- **Let** also can create the index and keep it as a reference for the current item.

```
Subjects: string[] = ["MS.NET", "Java", "SharePoint"]
```

```
<p>Subjects: <span *ngFor="let s of Subjects;let i=index"><span *ngIf="i!=0">,</span>
{{s}}</span></p>
```

Built-in Directives - ngSwitch

ngSwitch works similarly the switch statement that we use in C# / java etc.,

Here for defining case we use ***ngSwitchCase** and for default case we use ***ngSwitchDefault**

```
Duration: number = 30;
```

```
<p>Duration:  <span [ngSwitch]="Duration">
                  <span *ngSwitchCase="1">One Day</span>
                  <span *ngSwitchCase="7">One Week</span>
                  <span *ngSwitchCase="30">One Month</span>
                  <span *ngSwitchDefault>Please check the value</span>
                </span>
```

Using Class for Model

Employee.ts: Add a class Employee as below

```
export class Employee
{
   id: number;
   name: string;
   salary: number;
   constructor(id: number, name: string, sal: number)
   {
      this.id = id;
      this.name = name;
      this.salary = sal;
   }
}
```

Component, now changes as below:

```
import { Employee } from './Employee';
@Component({
 selector: 'my-app',
 template: `<h2>Id = {{emp.id}}</h2>
   <p>Name: {{emp.name}}, Salary={{emp.salary}}</p>`
})
export class AppComponent
```

```
{
   emp: Employee = new Employee(1,"Emp1",10000)
}
```

Template Expressions

We put a template expression within the interpolation braces, we write expressions in a language that looks like JavaScript.

JavaScript expressions that have or promote side effects are prohibited, including:

- assignments (=, +=, -=, ...)
- new
- chaining expressions with ; or ,
- increment and decrement operators (++ and --)
- no support for the bitwise operators | and &

We have two new operators supported in Template expression | and ?.

The pipe operator (|)

The result of an expression might require some transformation before we're ready to use it in a binding. For example, we might want to display a number as a currency, force text to uppercase, or filter a list and sort it.

Pipes are simple functions that accept an input value and return a transformed value.

With | we can use: uppercase, lowercase, date and json

Eg: `Name: {{emp.name | uppercase}}`

To show the object state in template (mostly for debugging) we can use json:

Eg: `<p>emp: {{emp | json}}</p>`

We can also apply **parameters** to a pipe: date can be shown in long format.

<!-- pipe with configuration argument => "February 25, 1970" -->

Birthdate: {{currentHero?.birthdate | date:**'longDate'**}}

The safe navigation operator (?.):

This ideally used to avoid null reference error.

Eg:

If emp is null following expression will fail to evaluate and whole view would disapear

`<p>Id = {{emp.id}}</p>`

In such cases we should use ?. operator

`<p>Id = {{emp?.id}}</p>`

Result in such case would be just blank value for expression but the expression doesn't fail.

Note: It works perfectly with long property paths such as a?.b?.c?.d.

Working with Arrays

Three directives we are going to discuss in this example:

1. NgFor for repeating content.
2. NgIf for conditional content
3. NgSwitch for conditional content

Edit Employee.ts as below

```
export enum EmployeeType {
   Daily=0, Permanent, Contract, Retired
}
export class Employee
{
   id: number;
   name: string;
   salary: number;
   type: EmployeeType
   constructor(id: number, name: string, sal: number, type: EmployeeType)
   {
      this.id = id;
      this.name = name;
      this.salary = sal;
      this.type = type;
   }
}
```

Edit app.component.ts as below

```
import { Component } from '@angular/core';
import { EmployeeType } from "./employee"
import { Employee } from "./employee"

@Component({
 selector: 'my-app',
 template: `<h2>All Employees</h2>
```

```
  <table>
    <tr>
      <th>Selected</th>
      <th>Id</th>
      <th>Name</th>
      <th>Salary</th>
      <th>Type</th>
    </tr>
    <tr *ngFor="let emp of employees">
      <td><span *ngIf="emp == selectedEmployee">----></span></td>
      <td>{{emp.id}}</td>
      <td>{{emp.name}}</td>
      <td>{{emp.salary}}</td>
      <td>
        <span [ngSwitch]="emp.type">
          <span *ngSwitchCase="0">Daily</span>
          <span *ngSwitchCase="1">Permanent</span>
          <span *ngSwitchCase="2">Contract</span>
          <span *ngSwitchCase="3">Retired</span>
          <span *ngSwitchDefault>Other</span>
        </span>
      <td>
    </tr>
  </table>
`
})

export class AppComponent
{
  employees = [
    new Employee(1, "E1", 10000, EmployeeType.Daily),
    new Employee(2, "E2", 11000, EmployeeType.Contract),
    new Employee(3, "E3", 13000, EmployeeType.Permanent)
  ]
  selectedEmployee = this.employees[1];
}
```

Note: Angular not going to show and hide the message. It is adding and removing the element itself from the DOM. This improves performance, especially in larger projects when conditionally including or excluding big chunks of HTML with many data bindings.

* vs <template>

In this section we go under the hood and see how Angular strips away the * and expands the HTML into the <template> tags for us.

Expanding *ngIf:

```
<span *ngIf="emp == selectedEmployee">----></span>
```

To

```
<template [ngIf]="emp == selectedEmployee">
  <span>----></span>
</template>
```

Expanding *ngSwitch

```
<span *ngSwitchCase="'Abcd'">Abcd</span>
```

To

```
<template [ngSwitchCase]="'Abcd'">
   <span>Abcd</span>
</template>
```

Expanding *ngFor

```
<tr *ngFor="let emp of employees">
 . . .
</tr>
```

To

```
<template ngFor let-emp [ngForof]="employees">
  <tr>
   . . .
  </tr>
</template>
```

External HTML Template File

We can have an external template URL of the components view.

My-app.html

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
        <meta charset="utf-8" />
</head>
<body>
  This is demo of external HTML template<br/>
  Name={{personName}}, Age={{age}}
</body>
</html>
```

my-component.ts

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

@Component({
 selector: 'my-app',
 templateUrl: 'app/my-app.html'
})

export class AppComponent
{
  personName = "Abcd";
  age = 30
}
```