

- property binding
- attribute binding
- event binding
- directives,
code blocks
- pipe



Agenda

- Recap last lecture topics
- UI Libraries
- Directives
- Life cycle Methods
- Types & Interfaces



UI Libraries



Angular Material

<https://material.angular.io/>

Ng Bootstrap

<https://ng-bootstrap.github.io/#/getting-started>

PrimeNG

<https://primeng.org/installation>

Directives



Directives

Directives are classes that add additional behavior to elements in your Angular applications.

<https://angular.dev/guide/directives>

Directives are kind of instructions to the DOM , Directives are components without a view. They are components without a template. Or to put it another way, components are directives with a view.

There's different types of directives :

- Component Directives
- Structural Directives
- Attribute Directives



Directives

Directives are kind of instructions to the DOM.

There's different types of directives :

- Component Directives - directives with a template.
- Structural Directives - change the DOM layout by adding and removing DOM elements.
- Attribute Directives - change the appearance or behavior of an element, component, or another directive.



Directives

Structural directives:

Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements.

- NgIf
- NgFor
- NgSwitch [Self-Study]



Directives

NgFor (Old versions before v17)

If using standalone components you need to import commonModule in the imports array of component.

```
<li *ngFor="let person of people; let i = index"> (1)
    {{ i + 1 }} - {{ person.name }} (2)
</li>
```



Directives

With built in @for

The @for block renders its content in response to changes in a collection.

Collections can be any JavaScript iterable, You can optionally include an @empty section immediately after the @for block content. The content of the @empty block displays when there are no items.

```
@for (user of users; track user.id) {  
  {{ user.name }}  
} @empty {  
  Empty list of users  
}
```



Directives

Track key

- When Angular renders a list of elements with @for, those items can later change or move. Angular needs to track each element through any reordering, usually by treating a property of the item as a unique identifier or key.
- This ensures any updates to the list are reflected correctly in the UI and tracked properly within Angular
- Loops over immutable data without trackBy as one of the most common causes for performance issues across Angular applications. Because of the potential for poor performance, the track expression is required for the @for loops. When in doubt, using track \$index is a good default.



Directives

Nglf (Old versions before v17)

The Nglf directive is used when you want to display or remove an element based on a condition.

If the condition is false the element the directive is attached to will be removed from the DOM.

```
<div *ngIf="loggedIn; else anonymousUser">
    The user is logged in
</div>
<ng-template #anonymousUser>
    The user is not logged in
</ng-template>
```



Directives

With built in @if

```
@if (loggedIn) {  
    The user is logged in  
} @else {  
    The user is not logged in  
}
```



*ngIf then and else (legacy before v17)

```
<div *ngIf="condition; then thenBlock else elseBlock"></div>
```

```
<ng-template #thenBlock>Content to render when condition is  
true.</ng-template>
```

```
<ng-template #elseBlock>Content to render when condition is  
false.</ng-template>
```

More info for *ngIf

<https://angular.io/api/common/NgIf>



Directives

Attribute directives

You should add **CommonModule** to imports array or import **NgClass** and **NgStyle** separated in imports.

- **ngClass** : It changes the class attribute that is bound to the component or element it's attached to.

```
<div [ngClass]="isSpecial ? 'special' : "">This div is special</div>
```

- **ngStyle** : It's used to modify or change the element's style attribute. This attribute directive is quite similar to using style metadata in the component class.

```
<div [ngStyle]="{{'background-color': isSpecial? 'green' : 'red' }}>This div is  
special</div>
```

Component lifecycle



Component lifecycle

- A component instance has a lifecycle that starts when Angular instantiates the component class and renders the component view along with its child views.
- You don't have to implement all (or any) of the lifecycle hooks, just the ones you need.
- After your application instantiates a component or directive by calling its constructor, Angular calls the hook methods you have implemented at the appropriate point in the lifecycle of that instance.



Component lifecycle

ngOnInit()

Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component input properties.

ngOnDestroy()

Cleanup just before Angular destroys the directive or component. Unsubscribe Observables and detach event handlers to avoid memory leaks.

<https://angular.dev/guide/components/lifecycle>

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy



Thank you



Lap

Users list

- Using the provided users list to render the users cards with the following:
 - Profile picture
 - Username
 - Email
 - Phone number
 - Birthdate
 - Role chip
- Based on the user role show a chip with different color. If admin show chip with red, if user show with green, if moderator show chip with yellow

Users

Search



admin

Username
Email
Phone
Birthdate



User

Username
Email
Phone
Birthdate



Moderator

Username
Email
Phone
Birthdate



User

Username
Email
Phone
Birthdate

Users list

- **[Bonus]** Also you can search and in the users list by Email and After user search, reset button will appear when click will reset the fields and show all users again.
- **[Bonus]** How to read data from JSON file

Users

The screenshot shows a user list interface titled "Users". At the top, there is a search bar containing the text "email" and a blue "Search" button. Below the search bar, there are four user profiles, each represented by a light blue rounded rectangle with a circular placeholder image at the top. The first profile is labeled "admin" in a red box, the second is "User" in a green box, the third is "Moderator" in a yellow box, and the fourth is "User" in a green box. Each profile contains four data fields: Username, Email, Phone, and Birthdate.

User Type	Username	Email	Phone	Birthdate
admin				
User				
Moderator				
User				



To-do App

Create a to-do app to create self-notes with the following features:

- User can add new task
- User can delete Task
- User can mark as completed, and when marked as completed, it will be marked with a linethrough. **[Bonus]**

The screenshot shows a simple web-based to-do application. At the top right, the text "To-Do App!" is displayed in a large white font. Below it, a smaller button labeled "Add New To-Do" is visible. A central input field contains the placeholder text "Enter new task". To the right of this input field is a blue "Add" button. The main body of the app is a large blue area containing the text "Let's get some work done!" followed by a horizontal line. At the bottom right, there is a small logo consisting of a blue hexagon with internal lines forming a cube-like structure, next to the text "Proudly powered by Cosmic JS".