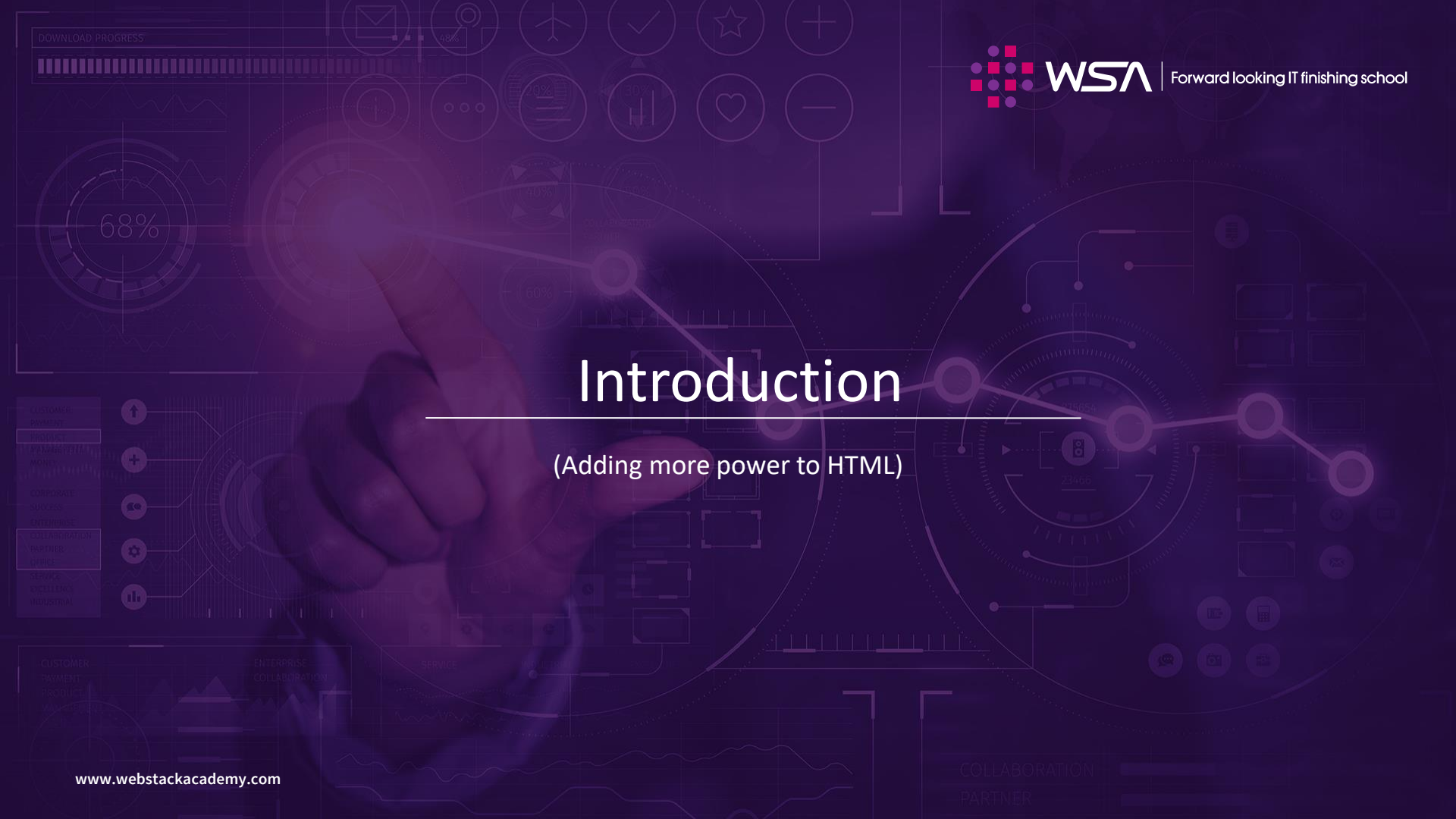
WSA | Forward looking IT finishing school

Directives

Angular


WSA
Forward looking IT finishing school
Introduction
(Adding more power to HTML)
www.webstackacademy.com

Introduction to Directives

- A directive is a custom HTML element (provided by Angular) that is used to extend the power of HTML. It helps to manipulate the DOM in a better manner.

- Angular templates are dynamic. When Angular renders them, it transforms the DOM according to the instructions given by directives.

- A directive is a class with a @Directive decorator.

- A component is technically a directive - but components are so distinctive and central to Angular applications that Angular defines the @Component decorator, which extends the @Directive decorator with template-oriented features.

Type of Directives

- At high level directives can be categorized into two areas:
  - Structural directives
  - Attribute directives

- **Structural directives:** Structural directives alter layout by adding, removing, and replacing elements in DOM (ex: ngIf)

- **Attribute directives:** Alter the appearance or behavior of an existing element. In templates they look like regular HTML attributes, hence the name (ex: ngModule)

- Apart from that the user has the power to create **Custom directives** (ex: Changing all the user input to lower-case) by adding additional functionality to the HTML DOM


WSA
Forward looking IT finishing school
Structural Directives
(Altering the structure / layout of the DOM)
www.webstackacademy.com

ngIf – Conditional Directive

- The `ngIf` directive is used when you want to display or remove an element based on a condition. It works similar to the conditional statement in any programming language
- If the condition is false the element the directive is attached to will be removed from the DOM.
- The condition is defined by passing an expression to the directive which is evaluated in the context of it's host component.
- The difference between HTML `[hidden]` and ngIf is that the first method simply hides the element. The second method with ngIf removes the element completely from the DOM.
- In case of the DOM becoming bigger, the [hidden] will create performance issues as it will take more time for it to load

ngIf – Usage example

```
<div *ngIf="coursesList.length > 0; then listCourses else
errCourses"></div>

<ng-template #listCourses>
        List of courses: {{coursesList}}
</ng-template>

<ng-template #errCourses>
        No courses now. Try back later!!
</ng-template>
```

In the above given example, depending on the component variable `coursesList.length` appropriate section will be loaded in DOM. The `<ng-template>` is provided by Angular for this purpose.

ngSwitch – Handling multiple conditionals

- The **`ngIf`** is helpful when the number of conditions are smaller. When it becomes more it becomes quite challenging to maintain (readability / code length etc..) the code.

- Like in most of the programming languages (ex: JavaScript) there is a directive in Angular called **`ngSwitch.`**

- This directive allows us to render different elements depending on a given condition, in fact the NgSwitch directive is actually a number of directives working in conjunction.

- In ngSwitch we bind an expression to the ngSwitch directive and matching expression will render the element it's attached to.

- If no conditions are met in the switch statement it will check to see if there is an **`ngSwitchDefault`** directive, where default action is performed (Typically error handling or some basic handling)

ngSwitch – Usage example

```
<ul [ngSwitch]="courseNumber">
    <li *ngSwitchCase="'1'">Hello Fullstack Student</li>
    <li *ngSwitchCase="'2'">Hello Frontend Student</li>
    <li *ngSwitchCase="'3'">Hello Masters Student</li>
    <li *ngSwitchCase="'4'">Hello Backend Student</li>
    <li *ngSwitchDefault>You are not WSA student</li>
</ul>
```

In the above given example, depending on the component variable **courseNumber** appropriate conditional will be loaded in DOM.
The **ngSwitch, ngSwitchCase, ngSwitchDefault** are provided by Angular for this purpose.

ngFor – Loop manipulation of DOM

The ngFor directory provides many exported values which can be used for efficient DOM manipulation.

Value	Description
`index : number`	The index of the current item in the iterable.
`first : boolean`	True when the item is the first item in the iterable.
`last : boolean`	True when the item is the last item in the iterable.
`even : boolean`	True when the item has an even index in the iterable.
`odd : boolean`	True when the item has an odd index in the iterable

ngFor – Loop manipulation of DOM

The ngFor directory provides many exported values which can be used for efficient DOM manipulation.

Value	Description
`index : number`	The index of the current item in the iterable.
`first : boolean`	True when the item is the first item in the iterable.
`last : boolean`	True when the item is the last item in the iterable.
`even : boolean`	True when the item has an even index in the iterable.
`odd : boolean`	True when the item has an odd index in the iterable

ngFor – Change propagation

- When the contents of the iterator changes, ngFor makes the corresponding changes to the DOM:
  - When an item is added, a new instance of the template is added to the DOM.
  - When an item is removed, its template instance is removed from the DOM.
  - When items are reordered, their respective templates are reordered in the DOM.
  - Otherwise, the DOM element for that item will remain the same.
- Angular uses object identity to track insertions and deletions within the iterator and reproduce those changes in the DOM.
- It is possible for the identities of elements in the iterator to change while the data does not change.
- Even if the data hasn't changed, Angular will tear down the entire DOM and rebuild it, which will create performance issues in case of large set of data

ngFor – trackBy option

- To customize the default tracking algorithm, ngFor supports **`trackBy`** option.
- The trackBy is nothing but a function which has two arguments: index and item.
- If trackBy is given, Angular tracks changes by the return value of the function.

```
<ul *ngFor="let course of newCoursesList2; trackBy: trackCourse">
    <li>{{ course.id }} -- {{ course.name }}</li>
</ul>
```


WSA
Forward looking IT finishing school
Attribute Directives
(Changing DOM attributes using directives)
www.webstackacademy.com

ngClass Directive

The NgClass directive allows you to set the CSS class dynamically for a DOM element.

```
<p [ngClass]="'one two'">
    Using NgClass with String.
</p>
<p [ngClass]="['three', 'four']">
    Using NgClass with Array.
</p>
<p [ngClass]="{'one': true, 'two': true, 'three': false }">
    Using NgClass with Object.
</p>
```

ngStyle Directive

- The NgStyle directive lets you set a given DOM elements style properties
- One way to set styles is by using the NgStyle directive and assigning it an object literal

```
<button (click)="changeButtonColor()" [ngStyle] = "{
backgroundColor: 'red' }"> Click here to change color </button>
```


WSA
Forward looking IT finishing school
Creating a Custom Directive - CLI
(Implementing your own custom directive)
www.webstackacademy.com

Creating a Directive – CLI mode

- **Step-1:** **Execute the following command in CLI to generate a directive. It will add a TS file into your project folder.**

```
wsa@wsa-VirtualBox:~/Angular/Directives$ ng g d test
CREATE src/app/test.directive.spec.ts (216 bytes)
CREATE src/app/test.directive.ts (137 bytes)
UPDATE src/app/app.module.ts (658 bytes)
wsa@wsa-VirtualBox:~/Angular/Directives$
```

Creating a Directive – CLI mode

- **Step-2:** **Adding your code into the directive**

```
import { Directive , HostListener, Host, ElementRef } from …

@Directive({
    selector: '[appCaseformat]'     // Directive Decorator
})

export class CaseformatDirective {

    constructor(private el: ElementRef) { } // Access underlying DOM

    @HostListener('focus') onFocus()  // Takes event as an argument
    {                                 calls the method upon event
        console.log ("Focus called");
    }

}
```

Creating a Directive – HostListener

- This is a function decorator that accepts an event name as an argument.
- When that event gets fired on the host element it calls the associated function.

```
@HostListener('mouseover') onHover() {
    console.log("Mouse hover...");
}
```

Creating a Directive – CLI mode

- **Step-3:** **Making use of your directive**

```
<h2>Creating a custom directive</h2>
<input type="text" appCaseformat>
```

Exercise

- Create a custom directive to implement **Title Casing**. Here are the rules for title casing:
    - Capitalize the first and the last word.
    - Capitalize nouns, pronouns, adjectives, verbs, adverbs, and subordinate conjunctions.
    - Lowercase articles (a, an, the) and prepositions (under, between, over, on, at).
- Assume there will only be three articles and three prepositions in the sentence. Inputs given in any case combination should return the output in Title case
- Some examples:
    - How Are You Doing Today?
    - Our Office **between** Metro and Barton Centre
    - What Is **the** Use?

WSA | Forward looking IT finishing school

WebStack Academy

#83, Farah Towers,
1st Floor, MG Road,
Bangalore – 560001

M: +91-809 555 7332
E: training@webstackacademy.com

WSA in Social Media:

www.webstackacademy.com