



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David



Web development Workshop

June-July 2022, London, UK



Week-4

Web development Workshop

June-July 2022, London, UK

Angular Fundamentals

- Interpolation
- Property binding
- Event binding
- Two way binding
- Directives
- Parent to child data
- Child to parent event
- Services
- Dependency Injection

Understanding Angular

- Components
- Templates
- Directives
- Services



Components

Components are the main building block for Angular applications.

Each component consists of:

- An HTML template that declares what renders on the page
- A TypeScript class that defines behavior
- A CSS selector that defines how the component is used in a template
- Optionally, CSS styles applied to the template

Templates

- The integral part of Angular component is Template.
- It is used to generate the HTML content.
- Templates are plain HTML with additional functionality.

Interpolation

Interpolation refers to embedding expressions into marked up text. By default, interpolation uses the double curly braces `{{` and `}}` as delimiters.

- * It wont work with boolean value

Property binding

Use property binding to do things such as toggle button functionality, set paths programmatically, and share values between components

```
<img [src]="src" [alt]="name">
```

Property binding

- *Update or change property of input element
- *It work with boolean value

```
<input type="text" value={{name}} disabled={{disable}}>  
<br /> <br />  
<input type="text" [value]=name [disabled]=disable>
```

Event binding

Listen for and respond to user actions such as keystrokes, mouse movements, clicks & touches.

```
<button (click)="onSave()">Save</button>
```

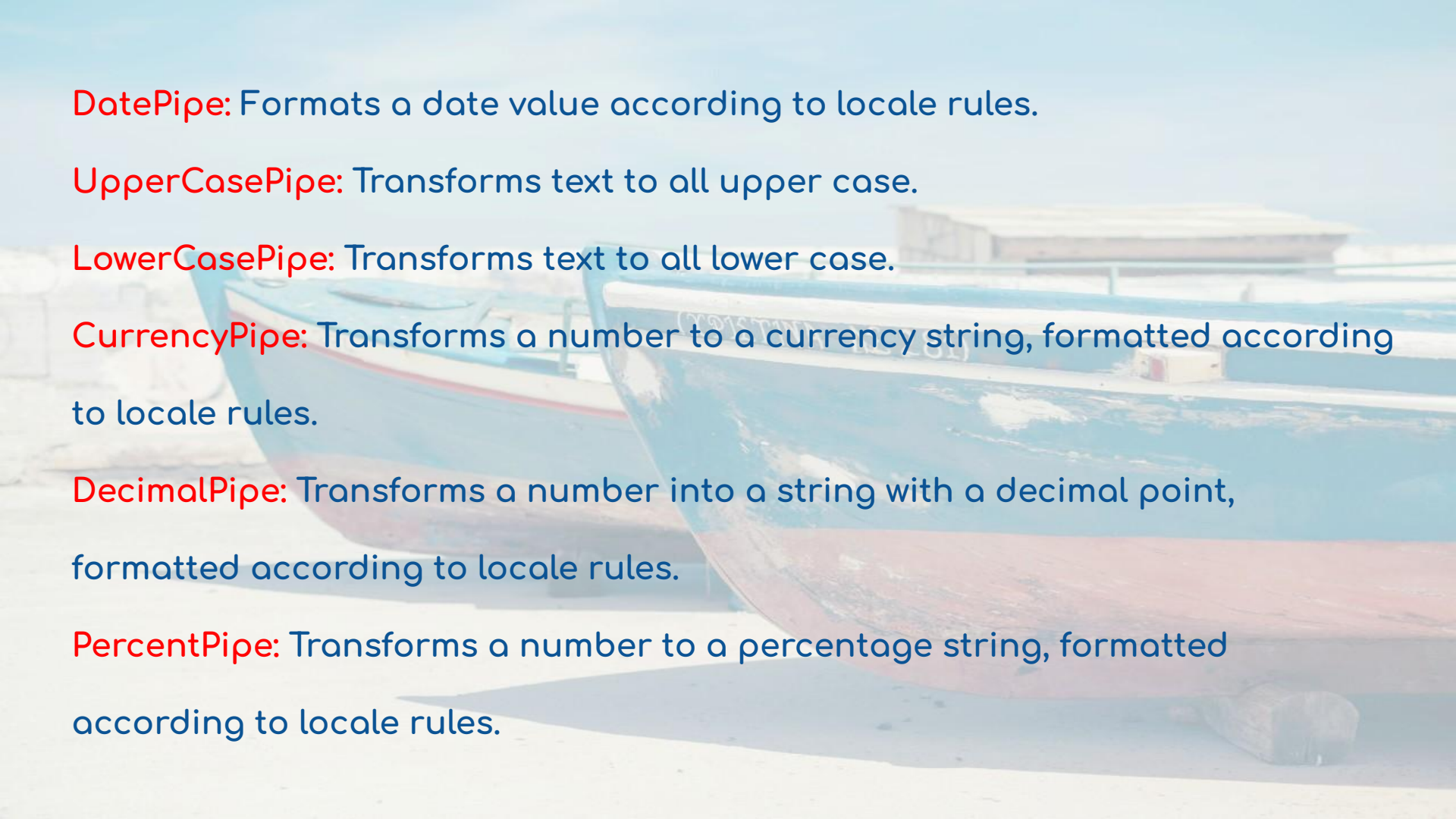
Click - target event name

Onsave - template statement

Pipe & custom pipe

Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. Pipes are useful because you can use them throughout your application, while only declaring each pipe once.

For example, you would use a pipe to show a date as April 15, 1988 rather than the raw string format.



DatePipe: Formats a date value according to locale rules.

UpperCasePipe: Transforms text to all upper case.

LowerCasePipe: Transforms text to all lower case.

CurrencyPipe: Transforms a number to a currency string, formatted according to locale rules.

DecimalPipe: Transforms a number into a string with a decimal point, formatted according to locale rules.

PercentPipe: Transforms a number to a percentage string, formatted according to locale rules.



<h1>{{ title | uppercase}}</h1>

<h1>{{ title | slice :3:8}}</h1>

<h1>{{user | json | uppercase}}</h1>

<h1>{{003.3400 | number:'1.2-3'}}</h1>

<h1>{{100 | currency : 'GBP'}}</h1>

Custom pipe

Create custom pipes to encapsulate transformations that are not provided with the built-in pipes. Then, use your custom pipe in template expressions, the same way you use built-in pipes—to transform input values to output values for display.

```
ng g p custom-pipes/usdGbp
```

Two way binding

Updating & displaying same time

app.component.ts

```
title = 'Two Way Binding';  
data:any;
```

app.component.html

```
<input type="text" [(ngModel)]="data">  
<h3>{{data}}</h3>
```

add FormsModule in app.module.ts

```
import {FormsModule} from '@angular/forms'  
imports: [BrowserModule,FormsModule],
```

Directives

Directives are DOM elements to interact with your application

Provides Additional features

ng g directive <directive-name>

If / else

Simple form with shorthand syntax:

```
<div *ngIf="condition">Content to render when condition is  
true.</div>
```

Simple form with expanded syntax:

```
<ng-template [ngIf]="condition"><div>Content to render  
when condition is  
true.</div></ng-template>
```

If / else

Form with an "else" block:

```
<div *ngIf="condition; else elseBlock">Content to render when condition is true.</div>  
<ng-template #elseBlock>Content to render when condition is false.</ng-template>
```

Shorthand form with "then" and "else" blocks:

```
<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>
```

SwitchCase

```
<div [ngSwitch]="color">
  <h1 *ngSwitchCase="'red'" style="color: red;" >Red
Color</h1>
  <h1 *ngSwitchCase="'green'" style="color: green;"
>Green Color</h1>
  <h1 *ngSwitchCase="'blue'" style="color: blue;" >Blue
Color</h1>
  <h1 *ngSwitchCase="'orange'" style="color: orange;"
>Orange Color</h1>
  <h1 *ngSwitchDefault >Unknown Color</h1>
</div>
```

Counter

app/app.component.html

```
<h1>{{ title }}</h1>
<button (click)="counter('minus') ">--</button>
<p>Count :{{count}} </p>
<button (click)="counter('add') ">++</button>
```

app/app.component.ts

```
export class AppComponent {
  title = 'Counter With Angular 14';
  count=0;
  counter(type:string) {
    type==='add'?this.count++:this.count--
  }
}
```

Parent-to-child-data

Use data from parent component to child component

`app/app.component.ts`

```
export class AppComponent {  
  title = 'Reusable Component';  
  userDetails=[  
    {name:'user1',email:'user1@test.com'},  
    {name:'user2',email:'user2@test.com'},  
  ]  
}
```

Parent-to-child-data

`app/app.module.ts`

```
import { UserDetailsComponent } from './user-details/user-details.component';

declarations: [
  AppComponent,
  UserDetailsComponent,
],
```

`app/user-details/user-details.component.ts`

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

inside export class add this:

@Input() item: {name:string, email:string} = {name: '', email: ''};
```

Parent-to-child-data

`user-details/user-details.component.html`

```
<li>{{item.name}}</li>  
<li>{{item.email}}</li>
```

`app/app.component.html`

```
<ul *ngFor="let user of userDetails">  
  <app-user-details [item]='user' ></app-user-details>  
</ul>
```

Child-to-Parent-data

Use data from child component to parent component.

Use `import { Output }`

Module

Angular Module is basically a collection of related features / functionality. Angular Module groups multiple components and services under a single context.

```
ng g module <module-name>
```

Services

Service is a broad category encompassing any value, function, or feature that an application needs. Angular distinguishes components from services to increase modularity and reusability.

```
ng g service <service-name>
```

Angular Routing

Change what the user sees by showing or hiding portions of the display that correspond to particular components, rather than going out to the server to get a new page

```
ng new routing-app --routing --defaults
```

Angular Forms

Template-driven forms

They are useful for adding a simple form to an app, such as an email list signup form

Reactive forms:

provide a model-driven approach to handling form inputs whose values change over time

Angular Forms

Login

Register

Contact



References:

<https://www.freecodecamp.org/news/learn-typescript-beginners-guide/>

<https://angular.io/guide/understanding-angular-overview>

<https://www.typescriptlang.org/docs/handbook/intro.html>