Angular with Bootstrap, Data Binding, Directives

Presented by

Sagar
Java Consultant

Installing Bootstrap in Angular App

- **Installing Bootstrap**
  - Navigate to angular app in command line.
  - Use npm to install Bootstrap 4 as follows :

```
npm install bootstrap
```

- **Adding Bootstrap to Your Angular Project**
  - Open angular.json and find styles array
  - Add the path to Bootstrap's CSS file as follows:

```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```

Using Bootstrap in Angular?

- Use Bootstrap classes in html file.

```
<div class="container mt-5">
 <h1 class="text-center">Welcome to Angular with Bootstrap</h1>
  <div class="row">
   <div class="col-md-6">
```

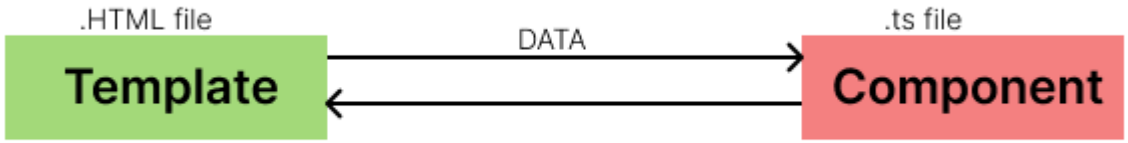
  - *Not adding <link> tag with CDN url*

- Navigate to *http://localhost:4200* in your browser, and you should see your Angular app styled with Bootstrap.

**Demo - Bootstrap with table data

What is Data Binding in Angular?

- Data binding is a mechanism that automatically synchronizes data between component's properties and the HTML template.
- In simpler terms, it's a way to bind the data (variables, objects, arrays, etc.) defined in your TypeScript code to your HTML template, and vice versa.



Types Data Binding in Angular

- **1. One-way binding :**
  - One-way data binding in Angular refers to the flow of data from the component to the view (or vice versa) in a single direction.
  - This approach ensures that changes in the component's data are reflected in the view, but not the other way around.
  - Types of One-Way Data Binding
    - Interpolation
    - Property Binding

Interpolation

- Interpolation :
  - Interpolation is used to display component data in the view.
  - It involves placing expressions within double curly braces ({{ }}) in the HTML template.

Interpolation …

- Component

Template

```ts
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  message = 'Hello, Angular!';
}
```

```html
<!-- app.component.html -->
<p>{{ message }}</p>
```

** Demo

Property Binding

- Property binding is one way from component to view.
- It lets to set a property of an element in the view[html] to property in the component[xxx.component.ts].
- This is done using square brackets ([ ]).
- Ex:

```
export class AppComponent {
  title = 'angular-oneway-property-binding';
  name = 'Aarush';
  isDisabled=true;
}
```

```
<h1 [innerText]="name">Name is replaced here</h1>
<p>Hello, {{ name }}!</p>
<button [disabled]="isDisabled">I am disabled</button>
```

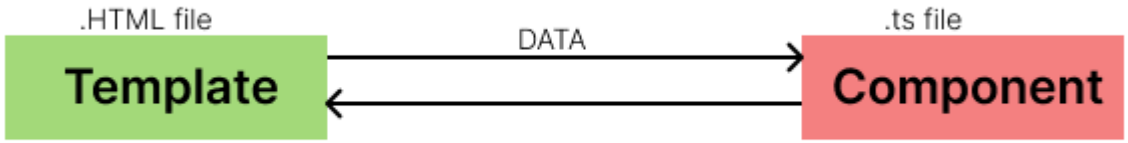
Property of <h1>

Property of <button>

** Demo

Two-way binding

- Two way binding allows to synchronize data between component's properties and the view in both directions.
- This means that any changes made to the component's property will be reflected in the view, and any changes made in the view will be updated in the component's property.
- It eliminates the need of manual updates



Two-way binding …

- The **[(ngModel)]** directive is used to establish two-way data binding between the name property of the component and the value attribute of the input element.
- Any changes made to the input field will be automatically reflected in the name property, and vice versa.
- Ex:
- app.component.ts

```
export class AppComponent {
  name = 'Arush';
}
```

app.component.html

```
<h1>Two-way Binding Example</h1>
<p>Enter User Name : <input [(ngModel)]="name" /> </p> <br/>
<p> Welcome, {{name}} </p>
```

** Demo

Event binding

- Event binding is a mechanism in Angular that allows you to bind component methods to HTML element events, i.e., click, keypress etc.
- When an event occurs on an HTML element, the specified component method is triggered, enabling you to perform actions based on the event.
- Method to trigger an event

```
clickCount=0;
  clickMe() {
    this.clickCount++;
  }
```

Event delegation

```
<button (click)="clickMe()">Click Me</button>
```

```
<span style="color: red">{{clickCount}} </span>
```

** Demo

Directives in Angular

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

1. **Structural Directives:** Change the DOM layout by adding or removing elements (e.g., ***ngIf, *ngFor**).

   These directives are prefixed with an asterisk (*) in the template.

   They are responsible for modifying the DOM structure.

   The most commonly used structural directives in Angular are ***ngIf, *ngFor, and *ngSwitch**

** Demo

Attribute Directives

Attribute Directives:

Angular provides several built-in attribute directives, such as **ngClass, ngStyle, ngModel** etc

These directives are used to apply classes, styles, and other attributes dynamically.

ngClass :

In Angular, the **ngClass** directive is used to dynamically add or remove CSS classes from an HTML element based on a condition.

** Demo

Attribute Directives . . .

ngStyle :

ngStyle directive allows to dynamically set inline styles on an HTML element based on the component's state or expressions

They can be useful for applying styles that change in response to user interactions or other conditions.

** Demo

Attribute Directives . . .

ngModel :

The **ngModel** directive in Angular is used for two-way data binding.

This means that it binds a form control (like an input, select, or textarea) to a property in the component class

This directive updates the property whenever the user changes the value in the UI.

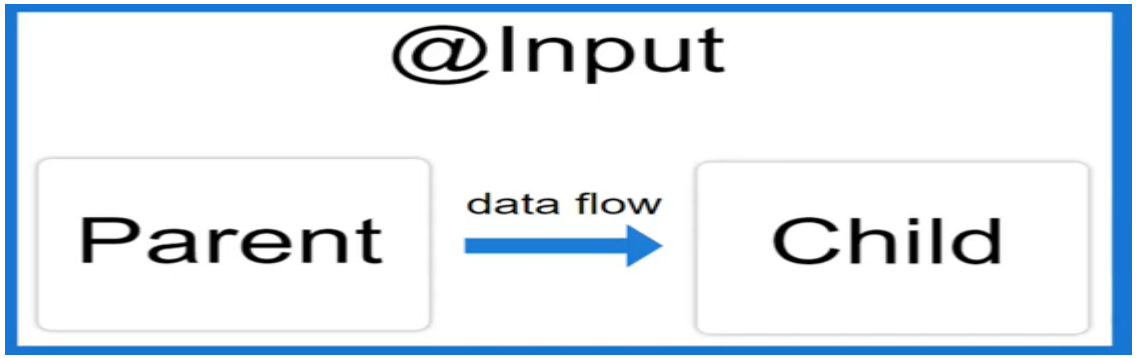
** Demo

Passing / Sharing data b/w components

- Data sharing between components in Angular refers to exchanging data between them.
- Components are usually required to communicate and exchange data to perform their functions.
- There are four Methods to Share Data between Angular Components:
  - Parent-to-Child: Sharing Data via **@Input**
  - Child-to-Parent: Sharing Data via **@Output** and **EventEmitter**

Parent to Child communication

- **1. Parent to Child Communication**
- Using **@Input** Decorator
- When you need to pass data from a parent component to a child component, you can use the **@Input** decorator.

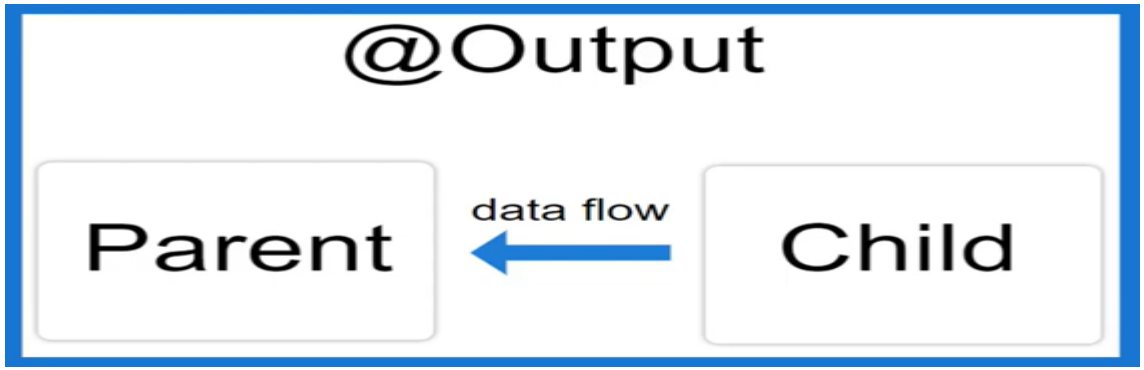


- C:\..\..>>**ng generate component child**

**Demo

Child to Parent Communication

- **2. Child to Parent Communication**
  - Using **@Output** Decorator and **EventEmitter**
- To send data from a child component back to the parent, use the **@Output** decorator and **EventEmitter.**



**Demo

Thank You