Introduction to Angular Framework

**o   Component Basics**

**o   Setting up the templates**

**o   Creating Components using CLI**

**o   Nesting Components**

**o   Data Binding - Property & Event Binding, String Interpolation, Style binding**

**o   Two-way data binding**

**o   Input Properties, Output Properties, Passing Event Data**

# Component Basics

<https://v2.angular.io/docs/ts/latest/api/core/index/Component-decorator.html>

<https://www.w3schools.com/bootstrap4/bootstrap_colors.asp>

Component is a building block, consist of following elements:

**Template**− This is used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.

**Class**− This contains properties and methods. This has the code which is used to support the view. It is defined in TypeScript.

**Metadata**− This has the extra data defined for the Angular class. It is defined with a decorator.

**Decorator** = We use the Component decorator provided by Angular to add metadata to the class. A class becomes an Angular component, when it is decorated with the Component decorator.

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'BookProj2020';}

Lab: Setup bootstrap

|  |  |  |
| --- | --- | --- |
| Step 1 | Install bootstrap | npm install --save bootstrap |
| Step 2 | Install jquery | npm install --save jquery |
| Step 3 | Install Popper | npm install --save popper |
| Step 4 | Update angular.json | "styles": [  "src/styles.css", "./node\_modules/bootstrap/dist/css/bootstrap.css" ], |
|  |  | "scripts": [ "./node\_modules/jquery/dist/jquery.min.js", "./node\_modules/bootstrap/dist/js/bootstrap.min.js"] |

# Setting up templates

A template is a HTML view where you can display data by binding controls to properties of an Angular component. You can store your component's template in one of two places. You can define it inline using the template property, or you can define the template in a separate HTML file and link to it in the component metadata using the @Component decorator's templateUrl property.

**Using inline template with template syntax,**

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

@Component ({

selector: 'my-app',

template: '

<div>

<h1>{{title}}</h1>

<div>Learn Angular</div>

</div>

'

})

export class AppComponent {

title: string = 'Hello World';

}

**Using separate template file such as app.component.html**

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

@Component ({

selector: 'my-app',

templateUrl: 'app/app.component.html'})

export class AppComponent {

title: string = 'Hello World';}

Lab: Create a NavBar

|  |  |  |
| --- | --- | --- |
| Step 1: | Create a links | <nav class="navbar navbar-expand-lg navbar-light bg-warning">  <a class="navbar-brand" href="#"><img class="img-responsive" src="../assets/logo.png" width="200" height="70" /></a>  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">  <span class="navbar-toggler-icon"></span>  </button>  <div class="collapse navbar-collapse" id="navbarNav" >  <ul class="navbar-nav">  <li class="nav-item active">  <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>  </li>  <li class="nav-item">  <a class="nav-link" href="#">Departments</a>  </li>  <li class="nav-item">  <a class="nav-link" href="#">Employees</a>  </li> </ul> </div>  <!-------------INSERT HERE-------------🡪  </nav> |
| Step 2: | Create Search Bar/Logout | <!-- Outside the div will make it Right aligned -->  <form class="navbar-form navbar-right" action="/action\_page.php">  <div class="input-group">  <input type="text" class="form-control" placeholder="Search">  <div class="input-group-btn">  <button class="btn btn-success" type="submit"> Submit </button>  </div> </div> </form>  <a href="logout"> <button type="button" class="btn btn-danger ml-4">Log Out</button></a></nav> |
| Step 3 | Set the outlet for routes | <router-outlet></router-outlet> |
| Step4 | Verify |  |

# Creating Components using CLI

|  |  |  |
| --- | --- | --- |
| Step 1 | Create dept component | ng g c dept |
| Step 2a | Create interface of dept in ts file | interface Dept{  did: Number,  dname: String} |
| Step 2b | Create depts.[] array | export class DeptComponent implements OnInit {  constructor() { }  ngOnInit(): void { }  depts:Dept[]= [  { did: 10,  dname: 'Sales'  },  { did: 20,  dname: 'Marketing'  },  { did: 30,  dname: 'Operations'  }  ] |
| Step 3 | Display table details in html | <div class="container">  <h1>Departments</h1>  <table class="table table-striped">  <thead>  <tr>  <th>ID</th>  <th>Name</th>  </tr>  </thead>  <tbody>  <tr \*ngFor="let dept of depts">  <td>{{ dept.did }}</td>  <td>{{ dept.dname }}</td>  </tr>  </tbody>  </table>  </div> |
| Step 4 | Make dept as part of app componenent –**app.component.html** | <h1 style="text-align:center"> {{title}} /h1>  <app-dept></app-dept>  <router-outlet></router-outlet> |
| Step 5 | Set style.css | h1{  color: #960  } |
| Step 6 | Test |  |

# Nesting Components

In above example title is specified in app.component.ts while is department component is display in app.component.html

<h1 style="text-align:center"> {{title}} </h1>

<app-dept></app-dept>

<router-outlet></router-outlet>

# Data Binding - Property & Event Binding, String Interpretation

Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data.

There are four forms of data binding(divided as 3 categories) which differ in the way the data is flowing.

|  |  |  |
| --- | --- | --- |
| I - **Component to the DOM** | | |
| **Interpolation** | Adds the value of a property from the component.  It is an easy way to convert to Property Binding. | <h1> {{title}} </h1> |
| **Property binding** | **[property]=”value”:** The value is passed from the component to the specified property or simple HTML attribute | <input type="email" [value]="user.email"> |
| II **-  DOM to the Component** | | |
| **Event Binding** | **(event)=”function”:** When a specific DOM event happens (eg.: click, change, keyup), call the specified method in the component | <button (click)="logout()"></button> |
| **III – DOM to component and Component to DOM** | | |
| **Two-way binding** | [(ngModel)]=”value”: Two-way data binding allows to have the data flow both ways. For example, in the below code snippet, both the email DOM input and component email property are in sync | <input type="email" [(ngModel)]="user.email"> |

Interpolation

<img class="img-responsive" src={{imagePath}} width="200" height="70" /></a>

export class AppComponent {

title = 'HRWorks';

imagePath: string = '../assets/logo.png';

}

Property Binding

<img class="img-responsive" [src]='imagePath' width="200" height="70" /></a>

|  |  |
| --- | --- |
| **Interpolation** | **Property Binding** |
| Must Use | |
| **UseCase:** To concatenate strings  Create a linkedin to connect to my profile by concatenate profile id in url in Department component. | **Use Case1:** To set an element property to a NON-String data value.      **UseCase 2:**  deptsubheader="List of all departments"  <b><span [innerHtml]='deptsubheader'></span></b> |
| <a href = "https://www.linkedin.com/in/{{mylinkedin}}"> Connect@ </a> | Bind the property value of **deptsubheader** with **innerHtml** attribute.  <b><span bind-innerHtml='deptsubheader'></span></b> |
| <h1>{{10+20}}</h1>  <h1>{{deptsubheader? deptsubheader : ‘No Header’ }} |  |

EventBinding

Event binding allows us to facilitates us to bind the events along with the methods. Whenever event is raised, we expect them to handle to perform certain.

**Types of Events Binding**

(focus)="focusCallback()"   
(blur)="blurCallback()"

(submit)="submitCallback()" (scroll)="scrollCallback()"

(cut)="cutCallback()"  
(copy)="copyCallback()"  
(paste)="pasteCallback()"

(keydown)="keydownCallback()"  
(keypress)="keypressCallback()"  
(keyup)="keyupCallback()"

(mouseenter)="mouseenterCallback()"  
(mousedown)="mousedownCallback()"  
(mouseup)="mouseupCallback()"

(click)="clickCallback()"  
(dblclick)="dblclickCallback()"

(drag)="dragCallback()"  
(dragover)="dragoverCallback()"  
(drop)="dropCallback()"

## **$event Payload**

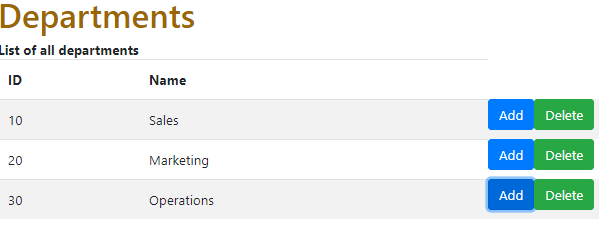
DOM Events carries the event payload. I.e the information about the event. We can access the event payload by using $event as an argument to the handler function.

[**https://www.tektutorialshub.com/angular/event-binding-in-angular/**](https://www.tektutorialshub.com/angular/event-binding-in-angular/)

**Note:** To access current changes by user, we use event object $event. It will be DOM event object if target event is a native DOM element event. It has properties as target and target.value.

Click Event Binding

###### Lab: Create a Add and Delete Button



|  |  |  |
| --- | --- | --- |
| Step 1: | Add buttons | <button class="btn btn-primary" (click)='addDepartment()'>Add</button><button class="btn btn-success" (click)='deleteDepartment()'>Delete</button> |
| Step2: | Handle the Events | addDepartment():void  { console.log('Add Clicked'); }  deleteDepartment():void  { console.log('Delete Clicked'); } |

<https://www.concretepage.com/angular-2/angular-2-event-binding-example>

Lab: Create a Login page

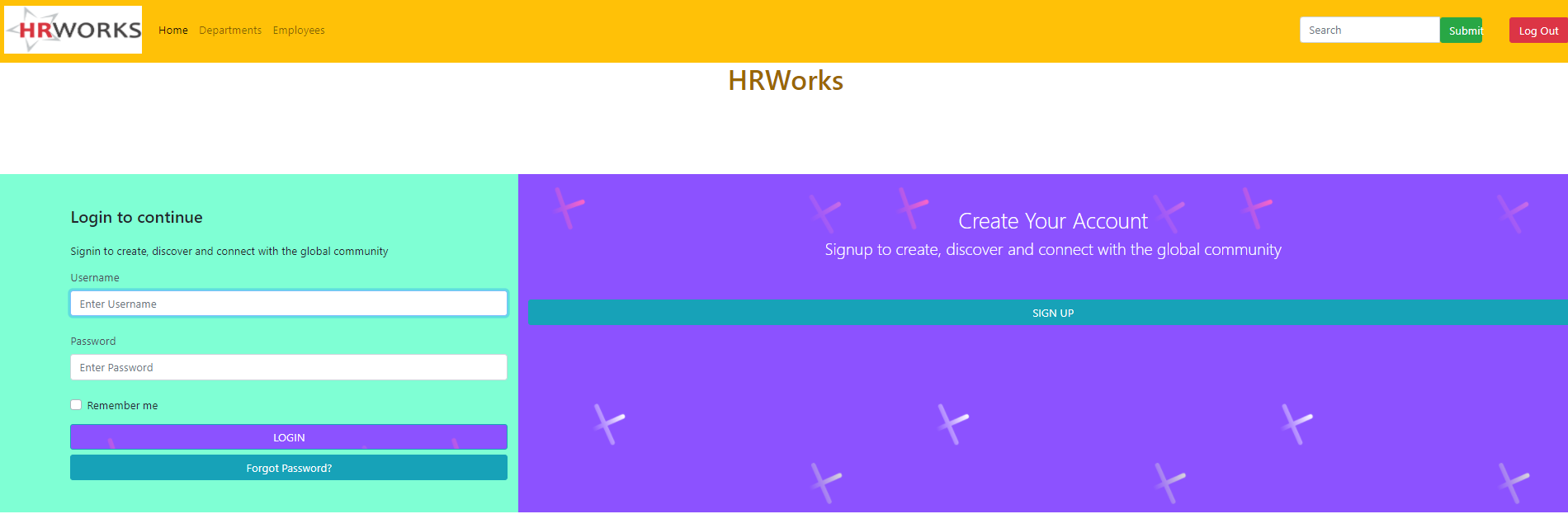
Reference : <https://bbbootstrap.com/snippets/login-form-background-image-15064663>

Step 1: Create login component.

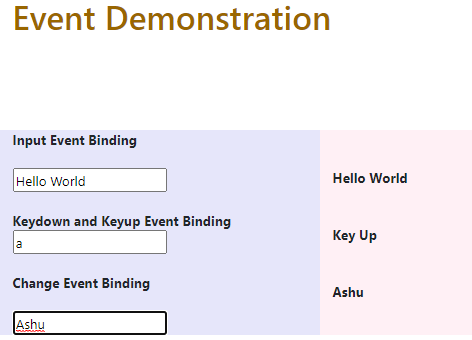
Step 2: With reference to <https://bbbootstrap.com/snippets/login-form-background-image-15064663>, Copy the html code to login.component.html.

Step 3: Paste the css file to src\style.css

Step4: Specify selector in app component.



Lab: Demonstrate the event binding in EventDemoComponent



|  |  |  |
| --- | --- | --- |
|  | **Create EventDemoComponent** | **Call it from app-component** |
|  | **EventDemo.html** | **EventDemo.ts** |
| **Input** | Input Event Binding </b><br/><br/> <input [value]="wish" (input)="wish=$event.target.value"><b>{{wish}}</b> | wish = 'Hello World'; |
| **KeyDonw/Up** | <input on-keydown="keywish='Key down'" (keyup)="keywish='Key Up'" >  <b> {{keywish}} </b> | keywish = '' |
| **OnChange** | <input on-change="changeText($event.target.value)">  <p [innerHTML] = "changewish"> </p> </b> | changewish=''  changeText(mytext:string)}this.changewish = mytext;  } |

# Style binding

The Style Binding uses the [] brackets.

<p

[style.color]="getColor()"

   [style.font-size.px]="'20'"

   [style.background-color]="status=='error' ? 'red': 'blue'">

   paragraph with multiple styles

</p>

# Two-way data binding

In two-way databinding, automatic synchronization of data happens between the Model and the View.

1. Sets a specific element property.
2. Listens for an element change event.

The [()] syntax combines the brackets of property binding, [], with the parentheses of event binding, ().

|  |  |  |
| --- | --- | --- |
| Step 1 | app.module.ts | import {FormsModule} from '@angular/forms';  @NgModule({  //declarations  imports: [ FormsModule ],  //providers  }) |
| Step 2 | eventdemo.component.html | <!-- 4. Two Way Binding -->  <div class="col-sm-10" style="background-color:lavenderblush;">  <b>Two way Binding</b>  <input [(ngModel)]="fullName" /> <br/><br/>  <p> {{fullName}} </p>  </div> |

Lab: Create Employee Component

|  |  |  |
| --- | --- | --- |
| Step 1: | Create an Employee component | Ng g c Listemployee |
| Step 2: | Create Employee Model class  model\employee.model.ts | export class Employee{  id : number;  name: string;  phoneNumber:number;  dateOfBirth:Date;  did:number;  photo?:sting;  isActive="true";  } |
| Step 3: | Create image repository in assets folder |  |
| Step 4: | Create an employees array in listemployee.ts | employees : Employee[] = [  {  id : 1,  name : 'Robert',  phoneNumber :1234567890,  dateOfJoining : new Date('11/20/1980'),  did :10,  photo: '../assets/images/Ron.jfif',  isActive : true  },  {  id : 2,  name : 'Ginni',  phoneNumber :1234567810,  dateOfJoining : new Date('1/2/1988'),  did :20,  photo: '../assets/images/Ginni.png',  isActive : true  },  {  id : 3,  name : 'Julie',  phoneNumber :12345678930,  dateOfJoining : new Date('3/3/1988'),  did :10,  photo: '../assets/images/Julie.jfif',  isActive : false  } |
| Step 5 | Make employee as child of department- app.department.ts | <app-list-employee></app-list-employee> |

# Input Properties, Output Properties, Passing Event Data

<https://www.tektutorialshub.com/angular/angular-passing-data-child-component/>

Input Properties are typically used to pass data from parent to child component.

To detect and react when an input property value changes

* Property Setter
* ngOnChanges Life Cycle Hook

Use Case: Department is parent Component and List Employee is child component. Display total number of departments in ListEmployee Component

|  |  |  |
| --- | --- | --- |
| Step 1: | Parent Component – Department.ts | getTotalDepartments():number  {return this.depts.length;} |
| Step 2: | Child Component – ListEmployee.ts | import { Component, OnInit,Input} from '@angular/core';  @Input() TotalDepartment: number; |
| Step 3: | Parent Component-Department.html | <app-list-employee [TotalDepartment]= "getTotalDepartments()"></app-list-employee> |

Use Case: Access name parameter from child component(Employee) in Parent Component (Department)

|  |  |  |
| --- | --- | --- |
| Step 1: | Child Component-ListEmployee | @Output() greetEvent=new EventEmitter();  name='Angular';  callGreetParent()  { this.greetEvent.emit(this.name); } |
| Step 2: | Child Component- | <button (click)="callGreetParent()"> Greet</button> |
| Step 3: | Create a method in Parent Component | greet(name:string)  { alert('Hello'+name); } |
| Step 4: | Bind the child method in parent component | <app-list-employee [TotalDepartment]= "getTotalDepartments()"  (greetEvent)=greet($event)></app-list-employee> |
| Step 5 | Verify |  |

Use Case : Display total number of employees(Child Component)in department component(Parent component).