# **Section 4 Bootstrap Fundamentals**

Installing Boostrap = https://getbootstrap.com/docs/4.6/getting-started/introduction/

Bootstrap Container = The boostrap requires the containing element to wrap site content. Container class provides a responsive fixed width container, container-fluid class provides a full width container spanning the entire wdith of the viewport.

Boostrap Grid System = 12 column grid (THIS MEANS that in a row we can create only 12 columns, the 13rd will be on the next row), build with css flex box, fully responsive

<div class="row">

<div class="col-md-1 bg-light border">Col - 1</div>

<div class="col-md-1 bg-light border">Col - 2</div>

<div class="col-md-1 bg-light border">Col - 3</div>

<div class="col-md-1 bg-light border">Col - 4</div>

<div class="col-md-1 bg-light border">Col - 5</div>

<div class="col-md-1 bg-light border">Col - 6</div>

<div class="col-md-1 bg-light border">Col - 7</div>

<div class="col-md-1 bg-light border">Col - 8</div>

<div class="col-md-1 bg-light border">Col - 9</div>

<div class="col-md-1 bg-light border">Col - 10</div>

<div class="col-md-1 bg-light border">Col - 11</div>

<div class="col-md-1 bg-light border">Col - 12</div>

<div class="col-md-1 bg-light border">Col - 13</div>

</div>

Bootstrap Columns = different size columns

<div class="container mt-5">

<div class="row">

<div class="col-md-12 bg-light p-5 border">col-md-12</div>

</div>

<div class="row">

<div class="col-md-6 bg-light p-5 border">col-md-6</div>

<div class="col-md-6 bg-light p-5 border">col-md-6</div>

</div>

<div class="row">

<div class="col-md-2 bg-light p-5 border">col-md-2</div>

<div class="col-md-8 bg-light p-5 border">col-md-8</div>

<div class="col-md-2 bg-light p-5 border">col-md-2</div>

</div>

<div class="row">

<div class="col-md-2 bg-light p-5 border">col-md-2</div>

<div class="col-md-4 bg-light p-5 border">col-md-4</div>

<div class="col-md-5 bg-light p-5 border">col-md-5</div>

<div class="col-md-1 bg-light p-5 border">col-md-1</div>

</div>

</div>

Bootstrap Responsive Columns = with col-sm-6 we are saying to the browser how much space the column is going to occupy in the small screens

<div class="container mt-5">

<div class="row">

<div class="col-md-12 col-sm-12 bg-light p-5 border">col-md-12</div>

</div>

<div class="row">

<div class="col-md-6 col-sm-6 bg-light p-5 border">col-md-6</div>

<div class="col-md-6 col-sm-6 bg-light p-5 border">col-md-6</div>

</div>

<div class="row">

<div class="col-md-2 col-sm-2 bg-light p-5 border">col-md-2</div>

<div class="col-md-8 col-sm-8 bg-light p-5 border">col-md-8</div>

<div class="col-md-2 col-sm-2 bg-light p-5 border">col-md-2</div>

</div>

<div class="row">

<div class="col-md-2 col-sm-2 bg-light p-5 border">col-md-2</div>

<div class="col-md-4 col-sm-4 bg-light p-5 border">col-md-4</div>

<div class="col-md-5 col-sm-5 bg-light p-5 border">col-md-5</div>

<div class="col-md-1 col-sm-1 bg-light p-5 border">col-md-1</div>

</div>

</div>

Boostrap Typography = uses a default font-size=16px, line-height=105, font-family="Helvetica",Helvetica, Arial, sans-serif, all <p> elements have margin-top= 0 and margin-bottom=1rem. Display headings are used to stand out more than normal headings, they have larger font-size and lighter font-weight. <div class="display-1">Display 1</div>

Bootstrap Contextual Colors = https://getbootstrap.com/docs/4.0/utilities/colors/

Bootstrap Tables = https://getbootstrap.com/docs/4.0/content/tables/

Bootstrap Images = We have 3 different types of images: 1. Rounded Corner Images, 2. Circle Images, 3. Thumbnail Images. https://getbootstrap.com/docs/4.0/content/images/

Bootstrap Buttons = https://getbootstrap.com/docs/4.0/components/buttons/

<a href="#" class="btn btn-primary">Link Button</a>

Bootstrap Spinner = https://getbootstrap.com/docs/4.0/components/progress/#how-it-works

https://getbootstrap.com/docs/4.3/components/spinners/

Bootstrap Cards = https://getbootstrap.com/docs/4.3/components/card/

Bootstrap Inputs = https://getbootstrap.com/docs/4.3/components/forms/, <https://getbootstrap.com/docs/4.3/components/alerts/=>

# **Section 5 Get Start with Angular & Angular CLI**

SPA (Single Page Application) = A Single Pahe Application is an app that doesn't need to reload the page during its use and works within a Web Browser. The files that contain the code for the site are being dowloaded in the Web Browser that's why the page isn't being loaded again.

With the use of Angular CLI it is easy to create and develop an Angular project. In order to download the Angular CLI we have to install first the node package manager (Node.js). In order to install the Angular CLI we have to run the followin command "npm install -g @angular/cli" and then in order to create a new project we have to run the following command

* ng new <Project-name>

# **Section 6 Angular Components**

Angular is a component based framework. A component is a combination of data html template and logics. The component represent an area of a view that shows inside the browser. We can reuse components (nav-bar etc.) For every area we can create a specific component.

We can use the selectors of every component into the html of every component. So we can call different components inside of the html of another component.

In order to generate an Angular component using the Angular CLI we have to use the following command "ng g c <component-name>". The files component.spec.ts are responsible for unit testing.

NgOnInit = the NgOnInit is going to run when the current component is fully loaded in the browser. OnInit is a template.

Data binding = {{title}}

Share data between components =

* Parent to child component via @input decorator
  + Parent: <app-post [fromParent]=’parentMessage’></app-post>
  + Child: @Input() fromParent:string;
* Child to parent component via @output decorator or @output decorator and event emitter
  + Parent: @ViewChild(PostComponent) childComp; ngAfterViewInit() { … }
* Child to parent when there is Event using the @output decorator and event emitter
  + Child: <button (click)=’sendMessage()’> Click</button>, @Outpu() messageEvent=new EventEmitter<string>(); sendMessage(){ this.messageEvent.emit(this.outputChildMessage); }
  + Parent: <app-post (messageEvent)=’receiveMessage($event)’, receiveMessage($event) { this.fromChildOutput=$event; }, <p>{{ fromChildOutput }}

Share data from the child components to parent component -> @ViewChild, @Output or Emitter.

# **Section 7 Data Binding & Events Handling**

String interpolation = in order to show information of a variable, or anything else we must use the string interpolation. An example of it is the following, {{ message }}

Property binding = <img [src]="imgURL" alt="googleImages"> or <img src="{{ imgURL }}" alt="googleImages">

Class binding = <p [class.text-red] = "boolFlag">{{messageFromChild}}</p>

Style binding = <h2 [style.color] = "boolFlag ? 'red' : 'blue'"> Style Binding </h2>, in order to bind and pass some styling to different elements we have to check the DOM style object (https://www.w3schools.com/jsref/dom\_obj\_style.asp)

Event binding = <button (click)="buttonClick()"> Click me Again! </button>

Event filtering = <input type="text" (keyup)= "onKeyup($event)">

Template variable = <input type="text" (keyup)= "showTemplateVariable($event)">

Two way data binding = app.module.ts -> import FormsModule, app.component.html -> [(ngModel)]

Two way vs One way data binding = With one way data binding I can send data from the component to the View but with the two way binding I can send data from the component to the view and vice versa

# **Section 8 Angular Directives**

Directives are components without a view. Angular components are directives with a view. We have different types of directives:

- Component directive = which is an angular directive with a template view

- Structural directive = which can change the DOM layout by adding and removing DOM elements

- Attribute directive = which can change the appearance or behavior of an element, component, or another directive

- Custom directive = which can create our custom directive from scratch

Generally directives are classes that add additional behavior to elements in any Angular applications. USe Angular's built-in directives to manage forms, lists, styles and what users see.

\*ngFor Directive = We use it to render an array inside the view, it is a structural directive, we can manipulate the DOM, <li \*ngFor="let str of stringArray"> {{str}} </li>

Fetching object array using NgFor = <li \*ngFor="let obj of objectArray"> {{obj.id}}, {{obj.postName}} </li>

Angular change detection = created 2 functions addNew() , onDelete(post)

ngFor directive index = we used an alternative on how we can send the index to the js function

ngIF directive = I created an example in the code

ngTemplate = we use the ngTemplate to render html in a template, in our view

(Structural directive) ngSwitchCase = created an example in the code

(Attribute directive) ngStyle = created an example in the code

(Attribute directive) ngClass = with ngClass we can pass classed to the elements of our html files

Structural vs Attribute directive = with structural directives we can manipulate the objects from the DOM, we can insert them etc. With Attribute directives we have change the appearance of the DOM objects.

# **Section 9 Angular Pipes**

Angular Pipes = they are used to transform the data into a particular format when we only need that data transformed 'in' a template or the HTML view. Some types of Pipes are the following:

- Uppercase Pipe

- Lowercase Pipe

- Decimal / Number Pipe

- Currency Pipe

- Date Pipe

- JSON Pipe

- Precent Pipe

- Slice Pipe

Uppercase , Lowercase pipe = {{ title | uppercase }} , {{ title | lowercase }}

Number pipe = {{ count | number }}, {{ count | number: '1.2-2' }}

Currency pipe = {{ price | currency: "JPY" : false : 2.1-1}}, iban.com/currency-codes

Date pipe = angular.io/api/common/DatePipe

JSON pipe = {{ postObj | json }}

Percent pipe = {{ 0.567 | percent : '1.1-1' }}

Slice pipe = {{ postArray | slice 2:4` }}

Customer pipe = new folders under app folder, for example append.pipe.ts, we have also to import the custom pipe into the file app.module.ts

Custom pipes using angular cli = ng g pipe Pipes/appendCLI

# **Section 10 Angular Service**

What is Angular service = we use angular services to share data and common methods among components whether there is relationship between components or not

Create angular service manually = create a folder named Services, add a new service post.service.ts,

Dependency Injection =

import { Engine } from './Engine';

class Car {

private engine: Engine;

public constructor (engine: Engine) {

this.engine = engine;

}

public startEngine(): void {

this.engine.fireCylinders();

}

}

Generate angular service using Angular CLI = ng g s services/user

Usage of angular services = we use services to do some actions and the other classes call the services functions and they perform some changes to the data through the services

Angular interface = ng g i models/post

# **Section 11 Angular Forms I (Template Driven Form)**

Angular Form types = Template - Driven Form (we get the full control of the form and we can validate the input data as we want) and Reactive Form

Creating a Bootstrap form = we create the bootstrap form (simple form)

NgForm directive = in order to create a form Template-Driven, we have to use special directive which is called ngForm. We also have to import the library FormsModule to the app.module.ts (for more information check AngularForms project)

NgForm Explained & FormGroup Class = In Angular to handle forms we have 2 classes formGroup (we use them with Form Tags) and formControl (we use them with Input fields)

Ng Model & FormControl class = in order to get the values that the user has typed we have to use the directive ngModel in every input element and then we have to use and assign a name to the specific input

Form Validation = check the app.component.html file

Styling invalid inputs = check the app.component.html file

Form validation types = check the app.component.html file

Validating email input field = check the app.component.html file

Validating text area = check the app.component.html file

Quick fix required not found error = check the app.component.html file

Form submission = we can get the submitted values by using the f parameter from the function onSubmit that is in the file app.component.ts, ngSubmit -> Enables binding AngularJS expressions to onsubmit events.

Disabling the submit button when form is invalid = check the app.component.html file

# **Section 12 Angular Forms II (Reactive Forms)**

Reactive form setup = we have to import in the file app.module.ts the package ReactiveFormsModule and also we have to included in the imports area

Creating the reactive form controls programmatically = see the app.component.html and app.component.ts files

Reactive form basic validation = see the app.component.html and app.component.ts files

Reactive form adding multiple validations = see the app.component.html and app.component.ts files

Complete all Validations = see the app.component.html and app.component.ts files

Reactive form submit and getting form values = see the app.component.html and app.component.ts files

Nested Form Groups = code example

Reactive Form Array = code example, with #skill in an input we can get the inputs of the input

Reactive Form Builder = code example

Adding Custom Validations I = code example

Adding Custom Validations II = creating a new folder under app folder called validators, creating a new file called nospace.validators.ts,

import { AbstractControl, ValidationErrors } from "@angular/forms";

export class noSpace {

static noSpaceValidations(control: AbstractControl): ValidationErrors | null{

let controlValue = control.value as string;

if(controlValue.indexOf(' ') >= 0){

return { noSpaceValidator: true }

} else {

return null

}

}

}

# **Section 13 Angular Routing & Navigation**

ng new ang-routing

ng g c post-list

In order to create a route we have to go to the file app.module.ts and we have to import the library RouterModule. Then we have to go at the imports and we have to insert the RouterModule.forRoot([ { path: 'posts', component: PostListComponent } ]). When there is a router and we want to display the contents of the component that we are about to use we have to insert the tag router-outlet

With routerLink the page is not being refreshed, we utilize the files that Angular already downloaded. With href the site reloads and we reload all the files that are needed so the concept of SPA is being disappeared.

Angulaer RouterLinkActive = with routerLinkActive we can apply a class after we press the button <button routerLink="/posts" routerLinkActive="active">Post List</button>

Code example for Router Parameter Variables <button [routerLink]="['/post', index]">View Content</button>

Get Router Parameters = paramMap gives all the router parameter values. Code example

What is Observable = rxjs, a sequence of data that is emitted asynchronously or synchronously overtime of period

Observable Next = with .next function we can return values from the observer. Code example

Observable vs Functions = with Observables we can return many things many times (As the definition says) but with a function we can return only one result at a time.

Observable Subscribe & Unsubscribe = we can close the subscription using the unsubscribe function

Multiple Router Parameters = Code example

Query Parameters = Code example

Separate Module for Angular Routing = ng g module app-routing --module app --flat, code example

Navigate Programmatically = code example

Wild Card Routers = we will add a page not found now, ng g c fournotfour, then we define the wild card router, it must always the last route

# **Section 14 Design The Blog Front End I - Preparing**

Generate components =

ng g c layouts/header

ng g c layouts/category-navbar

ng g c layouts/footer

ng g c pages/home

ng g c pages/single-category

ng g c pages/single-post

ng g c pages/terms-and-conditions

ng g c pages/contact-us

ng g c subscription-form

ng g c comments/comment-form

ng g c comments/comment-list

ng g c pages/about-us

Static routing and navigation =

# **Section 15 Design The Blog Front End II - Main Common Components**

# **Section 16 Design The Blog Front End III - All Pages Design**

# **Section 17 Get Start With Admin Area & Firebase Firestore**

# **Section 18 Admin area – CRUD Operation I (Category CRUD)**

# **Section 19 Admin area – CRUD Operation II (POST CRUD)**

# **Section 20 Authentication & Authorization**

# **Section 21 Design The Blog Front End I - Preparing**