Angular

**// To install angular on cmd use:**

npm install -g @angular/cli

**//To check angular version use::**

ng version

**// to create new project use::**

ng new projectName

**// To run the project use::**

ng serve

**// To generet a componenet use::**

ng g c yourcomponentName

“g” above means generate and “c” means component

**// To generate a module use::**

ng g m moduleName

“g” above means generate and “m” means Module

**// To generate a new component inside a specific module use::**

ng g c ModulefolderName\_Name/sNewcomponentName

=============================

**COMPONENTS**

To generate a componenet use::

ng g c yourcomponentName

To display any components in a browser append it "selector" in componentname.components.ts to app.component.html eg.

//selector inside the componetName.components.ts

Inside signupcomponent.ts

@Component({

    selector: 'app-signupcomponent', //embed this to display the signup component in appcomponent.html

  })

Inside app.component.html

  //append the selector in app.component.html

  <app-signupcomponent></app-signupcomponent>

  NOTE::: All component must be imported and declared IN appmodule.ts before use eg.

Inside appmodule.ts

  //import signupcomponent

  import { SignupComponent } from './signupcomponent/signupcomponent.component';

  //declared or register signup component as part of this module

  @NgModule({

    declarations: [SignupComponent]

  })

==============================================

**Module::**

They are files to include components in

To generate a module use::

ng g m moduleName

**To generate a component inside a specific module use**::

ng g c Modulefilepath/NewcomponentName

NOTE = You need to import all your modules and register it in app.module.ts before you can use it eg.

Inside app.module.ts

// "JusticemModule" is the class name of your module inside the file "./justicemodules/justice.module"

import { JusticemModule } from './justicemodules/justice.module';

//register module in "the imports" object

 imports: [JusticemModule],

NOTE=== component you create inside a child module needs to be imported, declared and export inside that child module

eg. inside justice.module.ts

// (1) import the component inside the child module “justice.module.ts”

import { SignupComponent } from './signupcomponent/signupcomponent.component';

//Then declare and export the component

@NgModule({

// (2) decare the component inside the module

  declarations: [SignupComponent],

// (3) export the component inside the module so that other component in app.module.ts can communicate with it

  exports: [SignupComponent]

})

==================================

Note== The meaning of "boostrap" selector in "@ngModule" decorator inside app.module.ts means "what is the parent component of the app". by default, is like this:

boostrap: [AppComponent]

so it actually asking you that, "what is the parent component?": [AppComponent]

So it actually tells angular the startup component in your app. Because there could be many component in your app. And without it how will angular know the parent component?

==================================

NOTE== In app.module.ts we::

register component in the "declare" object;

register module in "imports" object

====================

**To add a Template variable to html element inside it attribute use #variable\_Name**

Template variables helps you to assign html element tags like <p> to a variable and use that variable in different part of the same template. You can also access the variable in the component.

Example.

<input #box type="text" (click)="get(box.value)" />

To add an event to html element use (eventType)="functionName()" eg.

<input type="text" (click)="getName()" />

**Access a template variable in a component:** Option 1

// inside app.components.ts

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'justiceapp';

  /\* create a method that accept html <input> element and show it value

  if you want to pass in other html tags use:

  HTMLButtonElement = <button>

  HTMLParagraphElement = <p>

  HTMLAnchorElement <a>

  there is more for all the html tags you can search base on your tag

  \*/

  showValue(element:HTMLInputElement):void{

    // get the <input> element value and show it in alert

   var elementValue= element.value;

   alert(elementValue);

  }

}

<!-- inside app.component.html-->

<!--create a template variable and pass it to the component showValue() method-->

<input #firstName type="text" />

<!—below firstName represent the #firstName variable above-->

<button (click)="showValue(firstName)">Click</button>

**Access a template variable in a component:** Option 2 using @viewChild()

<!-- inside app.component.html-->

<!--create a template variable-->

<input #firstName type="text" />

<button (click)="showValue()">Click</button>

// inside app.components.ts

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

import { ViewChild, ElementRef } from '@angular/core';

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent{

  title = 'justiceapp';

  // pass an template vriable to the  @ViewChild("firstName")

// "firstName" below represent the #firstName variable in the template

  @ViewChild("firstName") firstNameRef!: ElementRef;

  showValue(){

    // below nativeElement returns the html tag <input>

  var inputEle = this.firstNameRef.nativeElement.value;

  alert(inputEle);

  }

}

====================

**=======CONDITIONAL STATEMENT ===**

==== if else statement

// "name" below is a property in app.components.ts class

<h1 \*ngIf="name == 'ben'; then ifBlock; else elseBlock">

  <p>peter</p>

</h1>

<ng-template #ifBlock>

  <p>justice</p>

</ng-template>

<ng-template #elseBlock>

  <p>Alex</p>

</ng-template>

===Switch statement

//colour is a property in app.components.ts classs

<div [ngSwitch]="colour">

  <p style="color:red;" \*ngSwitchCase="'red'">red color</p>

  <p style="color:yellow;" \*ngSwitchCase="'yellow'">red yellow</p>

  <p style="color:blue;" \*ngSwitchCas="'blue'">red blue</p>

  <p \*ngSwitchDefault>default</p>

</div>

======================

**===FOR LOOP**

FORMMAT TO LOOP THROUGH ELEMENT: \*ngFor="let #New\_variable\_Name of #Array\_Or\_Objectname"

<!--below is array of colours inside components -->

colour = ['yellow', 'blue', 'pink', 'red']; //loop through the above colours

<h3 \*ngFor="let item of colour">

    <p>{{ item }}</p>

</h3>

To loop through object inside arrays in components

colour = [ { name: 'justice', age: 21 }, { name: 'ben', age: 16 } ];

<!-- loop through above object -->

<h3 \*ngFor="let item of colour">

    <p>{{ item.name }}</p>

    <p>{{ item.age }}</p>

</h3>

==================

**CUSTOME DIRECTIVES**

Directives are a ways of assigning custom functionalities/functions to html element

For example. If you create a directive that will change the background-color of an element. You can assign the same directive to different element and their background color will change as well.

// To generate a directive use: ng g d customDirectiveName

Example::

ng g d mycustomdirective

//Then inside mycustomdirective.directive.ts

//First import the "ElementRef"

// "ElementRef" Represent the html element the Directive will be apply to

import { Directive,ElementRef } from '@angular/core';

@Directive({

  selector: '[appMycustomdirective]'

})

export class MycustomdirectiveDirective {

// Then pass the "ElementRef" as a type to a variable as a parameter in the contructor

// Now the parameter el represent the html element the directive will be assinged to, so apply what ever styel of functionality to it

  constructor(el:ElementRef) {

      el.nativeElement.style.color="red"; // when this directive is applied to html element, it color will be red

   }

}

// Then inside app.component.html add the selector tag of the above Directive to the html element you want the directives to be applied to

<h1 appMycustomdirective> Helloo World</h1>

========================================

**=========GET FORM VALUES (Template Driven Forms)====**

// first import Form modules and register it in app.modules.ts

//import

import { FormsModule } from '@angular/forms';

//register it

@NgModule({

  imports: [FormsModule],

})

// Then make a form in app.component.html

// #myform="ngForm" => initialize angular form to a variable “#myform”

// ngModel => the input fields value you want to get data from

// ngModel represent the html <input> tag “value” property

// (ngSubmit)="getvalues(myform.value) => onsubmit pass the valuas to a getValues() function

<form #myform="ngForm" (ngSubmit)="getvalues(myform.value)">

    <input type="text" name="username" ngModel placeholder="username" />

    <input type="text" name="email" ngModel placeholder="email" />

    <button>submit</button>

  </form>

//Get values function inside app.component.ts

  getvalues(val) {

    console.log(val);

  }

**=========================**

BINDING

In Angular there are two types of binding: one way and two-way binding

1) One way binding = data flow from the component class to html dom element only Or from the html dom element to the component class only.

Examples of one way binding:

"mypro" is a variable inside component class: mypro = true;

1. property binding:

Property binding is a way of binding a component class variable/property to a Dom-element

We use "[]" or "{{}}" eg

// "myimage" is a variable inside component class: myimage = "justice.png";

<img [src]="myimage" />

// This [] above, tells angular that "myimage" is property in the component class so he should assign it to the "src" html attribute.

 OR  <img src="{{ myimage }}" />

//this "{{}}" tells angular that "myimage" is a property so it should execute it. so it will be like <img src="justice.png" />

Note the concept here == Without the use of "{{}}" and "[]" how will angular know that "myimage" is a plain string or property in the component class?

B) style binding: we use [] ? eg

<p [style.color]="mypro? 'blue':'red' "> hellooo </p>

// Here am saying, if "mypro" is equal to true then <p> colour should be blue. else red

C) class Binding: we use []

<p [class]="mypro"> helloo </p>

// here am assigning a className to the <p> tag. You can also us boostrap class like <p [class.Active]="mypro"> helloo </p>

D)Event binding: data flow to the component class only. We use (event\_type)=methodName() eg

<button (click)=getdata()> click me </button>

// when the button is click, call getdata() method inside the component class

2) Two way Binding = Data flow between both the html dom and the component class.

We use "[(ngModel)]" directive to perform two way binding.

//first import "FormsModule" in app.module.ts and register it in the import selector.

import { FormsModule } from '@angular/forms';

  imports: [FormsModule]

//then create a property in app.component.ts and bind the dom element in app.component.html to each other.

mypro = "helloo"

 <input type="text" [(ngModel)]="mypro" />

// ngModel represent the html <input> tag “value” property

// The above "[]" will assign the "mypro" to the value of the input tag. So it be like this  <input type="text" value="mypro" /<

// And the above "()" will raise an event that will update the value of "mypro" with the "<input> tag value" as the user types

 <p> you have type: {{ mypro }}</p>

// i have bind the property and the <input> to each other with the [(ngModel)].

//So when the page first start, the value of the <input> will be "hello" But if the user types anything in the input field, the value of "mypro" will change to whatever the user types into the input field

===Split two-way binding==

It means what you want to do if the input value above change: eg:

we use [ngModel] directive and (ngModelChange) event to split two-way binding

[ngModel] = is a directive which will bind both the dom and the component class property together

(ngModelChange) = is an onchange event which will be watching whenever there is a change in the property bind together with the dom with [ngModel]. You just have to define a function in your component class and take some action whenever the value of the binded property change.

Example: Let say i want to show an alert whenever the binded property value is equal to "justice";

//make sure "FormsModule" is imported and registered in app.module.ts

// first define a property in app.component.ts

mypro = "helloo"

// Then bind the above property to a dom element with [ngModule] in app.component.ts then create (ngModelChange) event and assign function to it: the $event now will contain the value that the user will enter in the input field.

<input type="text" [ngModel]="mypro" (ngModelChange)="proMethod($event)"/>

 <p> you have type: {{ mypro }}</p>

//Then create the below function in app.component.ts and do what you want to do if the value change

mypro = "helloo"

  promethod(updatevalue): any {

    this.mypro = updatevalue;

    if (updatevalue == 'justice') {

      alert('the value is justice');

    }

  }

**==========================**

**// To Access a dom element from it component class with (@Viewchild)**

//first create a tag in app.component.ts and add a reference variable to it. below is #justice

<input type="text" #justice />

//Inside app.component.ts import...

import {ElementRef, ViewChild, AfterViewInit } from '@angular/core';

// then let your component class implement "AfterViewInit" interface

export class AppComponent implements AfterViewInit {

// create a property of type "EmentRef" and tell it which # reference in the dom you want it to be assigned to

  @ViewChild('justice') justiceref: ElementRef;

// overide the "ngAfterViewInit()" methode inside the "AfterViewInit" interface

  ngAfterViewInit() {

// what do you want the dom element to do? Here it should be focus.

    this.justiceref.nativeElement.focus();

//You can log it to the console and see some of it properties and methods you can use.

    console.log(this.justiceref);

  }

}

**==========================**

**@viewChild()**

It used to reference or access template Dom html elements and directives that belongs to the same component. It also used to access methods, properties and html Dom elements in different components.

#### Note: The initialization of the @viewChild() must always be done in the **ngAfterViewInit()** life cycle hook. It means your @viewChild() property will be ready after the ngAfterviewInit() hook is called.

**How to access Different components properties & methods**

// inside second.Component.ts

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

@Component({

  selector: 'app-second',

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

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

})

export class SecondComponent {

  constructor() { }

// create a method that will alert a messages

  showMsg(){

    alert("Inside second component");

  }

}

// inside appcomponent.ts

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

import { SecondComponent } from './second/second.component';

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent{

  // Declare/Create a reference object that you will use to

  // call metthods on the <SecondComponent>

  @ViewChild(SecondComponent) secondComponent!:SecondComponent;

  ngAfterViewInit(): void {

    // initialize it

    this.secondComponent = new SecondComponent();

  }

  // create a method that will be called on your templates

  show(){

   // call the showMsg() on the showMsg() method in <SecondComponent>

    this.secondComponent!.showMsg();

  }

}

<!--inside app.component.hmtl-->

<h1>Hello World?</h1>

<!--call the show() method in the component-->

<Button (click)="show()">show</Button>

<!-- Results: Inside second component -->

**How to access Child components properties & methods**

// inside second.Component.ts

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

@Component({

  selector: 'app-second-component',

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

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

})

export class SecondComponent {

  constructor() { }

  showMsg(){

    alert("inside Second Component");

  }

}

<!--inside app.component.hmtl-->

<h1>Hello World?</h1>

<!--call the show() method in the component-->

<Button (click)="show()">show</Button>

<!-- embed secondComponent template as child component-->

<!-- then create a reference variable #scndcmpont-dom-->

<app-second-component #scndcmpont\_dom></app-second-component>

// inside appcomponent.ts

import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';

import { SecondComponent } from './second/second.component';

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent{

  // Declare/Create a reference object

  // grab the reference variable that represent the <SecondComponent> itself embeded int the app.component.html template

  @ViewChild("scndcmpont\_dom") secondComponent!:SecondComponent;

  show(){

// call the showMsg() method in the <SecondComponent>

  this.secondComponent.showMsg();

  }

}

// result:  inside Second Component

**How to access a child component dom element using the @viewChild({read}) property**

The @viewChild has a “read” object property which indicate what exactly you want to grab on the reference variable.

For example, in School-room there is chairs, tables, students and teachers. You can reference the School-room and use the “read” property to specify what specific item you want in the room.

<!--Inside second.component.html-->

<p>Am second Components template</p>

// inside second.Component.ts

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

@Component({

  selector: 'app-second-component',

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

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

})

export class SecondComponent {

  constructor() { }

}

// inside appcomponent.ts

import { Component, ViewChild, OnInit, ElementRef } from '@angular/core';

import { SecondComponent } from './second/second.component';

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent{

  // Declare/Create a reference object

 // {read: ElementRef} = means read the element template of the SecondComponent

 // becuase the "scndcmpont\_dom" reference variable represnt the <SecondComponent.ts> itself not it template

  @ViewChild("scndcmpont\_dom", {read: ElementRef}) secondComponent!:ElementRef;

  show(){

   // grap the text <p>Am second Components template</p> in the secondcomponent template

  console.log(this.secondComponent.nativeElement.innerText);

  }

}

<!--inside app.component.hmtl-->

<h1>Hello World?</h1>

<!--call the show() method in the component-->

<Button (click)="show()">show</Button>

<!-- embed secondComponent template as child component-->

<!-- then create a reference variable #scndcmpont-dom-->

<app-second-component #scndcmpont\_dom></app-second-component>

<!--result:  Am second Components template-->

**How to use the @viewChild({static: true | false}) property**

**The** @viewChild({static: true | false}) helps you to specify when the initialization of the @viewChild() reference variable will be done.

* @viewChild({static: true}) 🡺 Means the initialization of the @viewChild() reference variable should be done after ngOnInit() life-cycle hook is called. Meaning the reference variable can be accessed only when & after ngOnInit() is called.
* @viewChild({static: false}) 🡺 Means the initialization of the @viewChild() reference variable should be done after ngAfterviewInit() life-cycle hook is called. Meaning the reference variable can be accessed only when & after ngAfterviewInit() is called.
* @viewChild() 🡺 This is the default and is equal to @viewChild({static: false}) which means the initialization of the @viewChild() reference variable should be done when and after ngAfterviewInit() life-cycle hook is called.

**Example 1:** @viewChild({static: false})

<!--inside app.component.hmtl-->

<h1>Hello World?</h1>

<!-- add a template reference variable  'myparagraph'-->

<p #myparagraph>This is a paragraph</p>

// inside appcomponent.ts

import { Component, ViewChild, OnInit, ElementRef, AfterViewInit } from '@angular/core';

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit, AfterViewInit{

 // Grap the "#myparagraph" in the template

 // {static: false} meanst the initiliation should be available after ngAfterViewInit() is called

  @ViewChild("myparagraph", {static: false}) parapgraph!: ElementRef;

  // the above is the same as: @ViewChild("myparagraph") parapgraph!: ElementRef;

ngOnInit(): void {

  console.log("inside ngOnInit");

  /\* This will be an error becuase the above @ViewChild("myparagraph") parapgraph!: ElementRef; has {static: false}

  Which means the initialation of the  @ViewChild("myparagraph") parapgraph!: ElementRef; should be availabe after ngAfterViewInit() is called

  And since ngOnInit() is called before it, It can't find the "myparaph" tag

  \*/

  console.log(this.parapgraph.nativeElement);

}

ngAfterViewInit(): void {

  console.log("inside ngAfterViewInit");

  /\* This is correct becuase @ViewChild("myparagraph") parapgraph!: ElementRef; has a  {static: false}

 Which means the initialation of the  @ViewChild("myparagraph") parapgraph!: ElementRef; should be availabe after ngAfterViewInit() is called

  \*/

  console.log(this.parapgraph.nativeElement);

}

}

/\*

RESULT:

inside ngOnInit

ERROR TypeError: this.parapgraph is undefined

inside ngAfterViewInit

<p #myparagraph>This is a paragraph</p>

\*/

**Example 2:** @viewChild({static: true})

    <!--inside app.component.hmtl-->

    <h1>Hello World?</h1>

    <!-- add a template reference variable  'myparagraph'-->

    <p #myparagraph>This is a paragraph</p>

  // inside appcomponent.ts

  import { Component, ViewChild, OnInit, ElementRef, AfterViewInit } from '@angular/core';

  @Component({

    selector: 'app-root',

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

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

  })

  export class AppComponent implements OnInit, AfterViewInit{

  // Grap the "#myparagraph" in the template

  // {static: true} means the initiliation should be available after ngOnInit() is called

    @ViewChild("myparagraph", {static: true}) parapgraph!: ElementRef;

  ngOnInit(): void {

    console.log("inside ngOnInit");

    /\* This is Correct.

    {static: true} means the initialiation should be done here

    \*/

    console.log(this.parapgraph.nativeElement);

  }

  ngAfterViewInit(): void {

    console.log("inside ngAfterViewInit");

    /\*

    This is Correct.

    {static: true} means the initiliation was done in ngOnInit(). And since ngOnInit() was called before ngAfterViewInit()

    the reference variable is availabe to be accessed here

    \*/

    console.log(this.parapgraph.nativeElement);

  }

  }

  /\*

  RESULT:

  inside ngOnInit

  <p #myparagraph>This is a paragraph</p>

  inside ngAfterViewInit

  <p #myparagraph>This is a paragraph</p>

  \*/

**Note: If you want to Access multiple template reference variables with the same name and perform the above operation on them all. Use @viewChildren() decorator.**

========================

**TO PASS DATA FROM PARENT TO CHILD COMPONENT**

We use the @Input() decorator. It's a public api for accepting values from parent component. Meaning any property that is attached to it becomes a public api container where the value will be passed by the parent component.

NOTE:: All Angular "decorators" are functions. eg @input(), @Component({}) @Output();

// First import the "Input", then create a property and attached the @input() decorated to it to make it a public api where it value will be passed by the parent component.

Inside child.component.ts….

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

export class ChildcomponentComponent implements OnInit {

//value of "wife" will be passed by the parent component

  @Input() wife;

}

//Then inside app.component.ts create a property and assign the value that you want to pass to the child component to it

export class AppComponent {

  wifeName = 'aunt Gladys';

}

Inside app.components.html

// Then in the child component html tag in appcomponent.html, assign the property value in parent component to the public api property in child component using property binding

//Below.. wife = @Input() wife in child component. wifeName is the wifeName = 'aunt Gladys' in parent component

  <app-childcomponent [wife]="wifeName"> </app-childcomponent>

Inside mychild.html

//Then finally display it in mychild.html file.

<h2>{{ wife }}</h2>

=========================

**TO PASS DATA FROM CHILD TO PARENT components**

The @output() decorator is used to pass data from child component to parent component.

// define a function with parameter in appcomponent.ts that the child component will pass a value to it

parentfunction(val){

    alert(val);

  }

// Then inside childcomponent.ts import the "Output" and "EventEmitter" class

import {Output, EventEmitter } from '@angular/core';

// Then create a property and assign new EventEmiiter() to it to make it an event like click event. Then add "@Output()" to the same property to tell your child component that, that property is going to pass value out of the component.

  @Output() myChildEvent = new EventEmitter();

// create a method and emit the value out to the parent function

pass\_value\_to\_parent\_funtion() {

   this.myChildEvent.emit("justice is my name");

  }

// Then inside child.component.html create a button to call the above function that will emit the value out to the parent function

  <button (click)="pass\_value\_to\_parent\_funtion()">call Emit</button>

// Then inside appcomponent.html Bind the child "mychildEvent" value to the parent function

 <app-childcomponent (myChildEvent)="parentfunction($event)"> </app-childcomponent>

====================

**ROUTING**

ROUTING is a way of providing a navigation link to different component in the app

// The navigation url and the component to open must be declared in app-routing.module.ts

eg.

// inside app-routing.module.ts

// import all the component you will direct the url path to

import { ChildcomponentComponent } from './childcomponent/childcomponent.component';

import { OxfordComponent } from './oxford/oxford.component';

import { PagenotfoundComponent } from './pagenotfound/pagenotfound.component';

// define a custom url path and the component to open if the url is clicked

const routes: Routes = [

  {

    path:"child",

    component:ChildcomponentComponent

  },

  {

    path:"oxford",

    component:OxfordComponent

  },

  {

    //page not found url, this path must be "\*\*" and should be the last url defined in the router

    path:"\*\*", component:PagenotfoundComponent

  }

  ];

// Then inside app.component.html

//add the url path to a link when clicked. Note "href" is replaced with "routerLink"

<a routerLink="oxford">oxford </a>

<br/>

<a routerLink="child">child </a>

<br/>

// here is where the component template will be displayed if the link is clicked

<router-outlet> </router-outlet>

============================

ROUTING MODULE

ROUTING MODULE is a way of providing navigation link between different component in a specific module.

To generate a module with routing.ts file Use: ng g m moduleName --routing

Example::

// create a module with routing

ng g m admin--routing

// then create two component inside the above module

ng g c admin/about

ng g c admin/contact

// Then inside app.module.ts import the “Admin” module and register it

import { AdminModule } from './admin/admin.module';

@NgModule({

  imports: [

    AdminModule

  ]

})

// Then inside admin.module.ts import the two compoent and register it.

import { ContactComponent } from './contact/contact.component';

import { AboutComponent } from './about/about.component';

@NgModule({

  declarations: [ContactComponent, AboutComponent]

});

// Then inside admin.routing.module.ts import the two component and provide the navigation url

import { ContactComponent } from './contact/contact.component';

import { AboutComponent } from './about/about.component';

const routes: Routes = [

  { path: 'contact', component: ContactComponent },

  { path: 'about', component: AboutComponent },

// "\*\*" represent Not found page. it should be the last path always, if not it will catch all urls.

  { path: '\*\*', component: NotfoundpageComponent }

];

// Then inside app.component.html add navigation link

<a routerLink="contact"> Contact</a>

<br />

<a routerLink="about"> About</a>

// "router-outlet" is where the component will be displayed if any of the above link is clicked

<router-outlet></router-outlet>

**//To add/pass a "parameter" to path/url use ":" eg.**

 { path: 'user/:id', component: ContactComponent },

 //then pass id of 1 to the url like this 127.0.0.1:8000/user/1 in app.component.html

 // To set the above dynamically to different users id link. use "routerLink" as property binding.

 // Then assign an array of [path, parameter] to it eg.

 <a [routerLink]="['user/', userid.id]"

 // "userid" is an array of object in component.ts

 userid:[

 {

 id:1,

 name:"justice"

 },

 {

 id:2,

 name:"ben"

 }

 ]

// TO RETRIEVE A PARAMETER FROM A URL

// import "ActivatedRoute" module

import { ActivatedRoute } from '@angular/router';

//”ActivatedRoute” represent the current url the user is on. So if the user is on http://localhost:4200/customers, then that’s the activated url.

// then create a variable with the type of "ActivatedRoute" in the constructor

constructor(private route:ActivatedRoute){}

// Then inside "ngOnInit" method use the above variable to access the parameter

// remember the subscribe() method takes in arrow function.

ngOnInit(){

 this.route.paramMap.subscribe(params=>{

// the "id" is what was defined in the router.module.ts { path: 'user/:id', component: ContactComponent },

    let id = params.get('id');

// "params" contain these methods which do different things get(), getAll(), has(), keys()

    console.log(id);

})

}

HOW TO NAVIGATE TO SOME CERTAIN URL/PATH IN YOUR COMPONENT

// To navigate to a certain path in your component use the “Router” class in your component and use it navigateByUrl(“path\_toNavigate”) method.

// First import the Router class

import { Router } from '@angular/router';

// Then create and object of it in the constructor

  constructor(private router: Router) {}

// Then use it navigateByUrl() method to navigate to the path you want to specified in your app.routing.module.ts

ngOnInit(): void {

      this.router.navigateByUrl('/signup');

}

===========================

**How to subscribe and detect whenever rout/url changes**

Inside app.component.html

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

  // First import the following

  import {

    Router,

    Event,

    NavigationStart,

    NavigationEnd,

    NavigationError,

  } from '@angular/router';

  @Component({

    selector: 'app-root',

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

    styleUrls: ['./app.component.sass'],

  })

  export class AppComponent {

    currentRoute: string;

    // inject the Router dependencies

    constructor(private router: Router) {

      this.currentRoute = '';

      this.router.events.subscribe((event: Event) => {

        // when a new navigation start

        if (event instanceof NavigationStart) {

          // Show progress loading indicator here in real world app

          console.log('Route change detected');

        }

        // when a navigation End

        if (event instanceof NavigationEnd) {

          // Hide  progress loading indicator here in real world app

          // The "event" contains object of the current-url & urlAfterRedirect

          console.log(event);

          // assign the current url the user is on to this var

          this.currentRoute = event.url;

        }

        // when a navigation fails due to an unexpected error

        if (event instanceof NavigationError) {

          // Hide  progress loading indicator here in real world app

          // Present error to user

          console.log(event.error);

        }

      });

    }

  }

===========================

**// SERVICE (basic service)**

It’s a Way of Using Data Among component. You store specific data in a service and have multiple components use it.

It can be data from rest api like php or python or your own data (usually in json Format).

// To generate a service use: ng g s serviceName

example::

// generate a service on cmd

ng g s myservice

// Then inside myservice.service.ts define a function in the export class that returns object of data. Will be data from http server in a real world

getservice(){

 return {

  name:"justce",

  age:21

}

};

// Then inside app.component.ts

// import the service class from the service

import { MyserviceService } from './myservice.service';

// define a property in the class to assign the name value in the service to it later on

  getname="";

// pass the imported service class as a type to an array in the constructor.. this is dependency injection

constructor(service:MyserviceService){

// Get the name value inside the service getservice method and assign the value to "getname"

this.getname = service.getservice().name;

}

// Then inside app.component.html display the name from the service

<p> {{getname}} </p>

// Then inside app.module.ts. import the service and register it in the providers selector.

import { MyserviceService } from './myservice.service';

@NgModule({

  providers: [MyserviceService],

})

===============

**Service with Http Connection**

**It’s** a way of making an http request to get data from a backend server like rest-api.

The concept is that, we don’t use component to make http connections to servers to get data. We do that with service, and then when the service get the data, we use any of our component to fetch the data from the service. So, service get the data from servers, and component fetch those data from the service. Eg

// First inside app.module.ts import “HttpClientModule” and register it in the import selector

import { HttpClientModule } from '@angular/common/http';

  imports: [  HttpClientModule]

// Then create a service called “AngularServiceService”

Ng g s AngularServiceService

// Now inside “angularservice.service.ts”

//import “HttpClient”, It a package that will give your service the capability to perform http get, post, delete and put request

import { HttpClient } from '@angular/common/http';

// Then inside the contructor, create a private variable of type “HttpClient” This will give the “http” property below the ability to access get, post, delete and post.

// It means this http property will depend on HttpClient to make http calls using get or post (form of dependency injection).

 constructor(private http: HttpClient) {}

/\*

  Then create a method of type “Observable”, make sure you import Observable too. And use the above “http” property to make an http get request to the backend server.

  The get() method takes in the url of the backend service and the format the data is in (text or json). You can ignore the second parameter because by default angular assumes the data is in json.

  But when it is in text, you need to tell the server in the second parameter of the get method.

  The “Observable” represent return type of the method. It means = this method below, when it finally get the data from the server and it returning it to our application,

  It should return it asynchroniously so that I can get access to the data without having to refresh the browser before (Is just like how Ajax works).

 you can also append .pipe(map(result=>{ //do something with the data before sending it to the subscribe component})) to the get<any>() method below.

It //allows you to do something with the data before sending it to the subscribe component class.

\*/

getservadata(): Observable<any> {

    return this.http.get<any>("https://justice.com/api/customersdetails.json", {  responseType: 'json'});

  }

  // That's it, the whole code explained above in “AngularServiceService.service.ts” is This

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class AngularserviceService {

  url = 'http://localhost/angularserver/api/getdata';

  constructor(private http: HttpClient) {}

  getservadata(): Observable<any> {

    return this.http.get<any>(this.url);

  }

}

/\*  Now the service has gotten the data from the server. So your component can access those data.

 Well, the concept is that, You simply inject the service to your component in the constructor, then subscribe to the data and just use it in your template.

 So in app.component.ts

 import the above service “AngularServiceService”

 \*/

import { AngularserviceService } from './angularservice.service';

//Then inject the service in the contructor parameter by creating a private property with a type of the service (dependency injection)

  constructor(private angservice: AngularserviceService ) {}

//The data from the server is a json array so create an array property that you will assign the data to it.

  employeedata = [];

//Now we need to subscribe to the service in ngOnInt method to get the data and assign it to “emploeedata” property in our component.

  ngOnInit(): void {

    this.angservice.getservadata().subscribe(

// it takes in an object

      {

        // next: represent the data coming from the service

        next: (data) => {

          this.employeedata = data

        },

        // when there is an error with the service

        error: (err) => {

          alert("There was an error of: "+ err);

        }

      }

      );

  }

/\*  Inside the “subscribe()” function. You just assign the data to the property value with a parameter arrow function.

 It’s like this== function(data){this.employeedat =data}

 Now the data is assigned “employeedata” as an array, so you need to loop through it in your template.html and display it

remember the "name", "age" and "country" append to the emdata below is the selector of the json object from the serve.

\*/

   <p \*ngFor="let emdata of employeedata">

      {{ emdata.name }} <span> {{ emdata.age }} </span>

      <span> {{ emdata.countr}} </span>

    </p>

**Http Error handling**

Note==It’s good practice to always check for http errors and notify the user of any bugs.

There is an interface in Angular called “**interceptor**” that is used for checking all https errors that may arise in the app globally.

Interceptors are middleware’s between the client and server Http communication. **Example below.**

(client\_make\_Http\_Request\_To\_server) ==> (Interceptor) ==> (Server)

And or

(client\_receiving\_data\_from\_Http\_server) <== (Interceptor) <== (Server\_Returns\_Http\_response\_To\_Client)

So create a new interceptor with: ng g interceptor interceptor\_Name\_Here

All you need to do is that, create an interceptor that will throw an error to all component that will subscribe to it, and register it in app.module.ts

**Example:**

// Inside my-http-interceptor.interceptor.ts create an interceptor that will throw an error to the component

//Note == below code will check each http request and response and notify the subscribe component of either the data from the server or an error. //So that the component can grab the data or error and display it in it component.

// Always copy below code to your interceptors simply

import {

  HttpEvent,

  HttpInterceptor,

  HttpHandler,

  HttpRequest,

  HttpErrorResponse

} from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

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

@Injectable()

export class HttpErrorInterceptor implements HttpInterceptor {

  intercept(request: HttpRequest<any>, next: HttpHandler ): Observable<HttpEvent<any>> {

    return next.handle(request).pipe(

      catchError((error: HttpErrorResponse) => {

        let errorMsg = '';

        //Check if error is a client side or server-side error

        //if is a client-side error?

        if (error.error instanceof ErrorEvent) {

          errorMsg = `Error: ${error.error.message}`;

        }

        //Server side error?

        else {

          errorMsg = `status Code: ${error.status},  Message: ${error.message}`;

        }

        //

        alert(errorMsg);

        //return this error to the subscribeed component so that it can dispaly it to the user

        return throwError(errorMsg);

      })

    );

  }

}

// Then register the above interceptor in app.module.ts Providers selector

  providers: [

    AngularserviceService,

Sinupservice,

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: HttpErrorInterceptor,

      multi: true

    }

  ],

//With that if am subscribing to any service in my component. I have to check for two things. Is either I will get the data I want or an error.

//Inside your component subscribe to a service and to get data and check and notify the user if there is an error too. So below if there is an error //I will simply display “Errormessage” in the template to the user.

Errormessage:string;

ngOnInit(): void {

    this.angserice.getservadata().subscribe(

      data => (this.employeedata = data),

      http\_error => (this.errormessage = http\_error)

    );

  }

// That’s a very good programming practice to check for all http errors. You can learn more about error checking with http intercept on :

// <https://medium.com/@satyapriyamishra111/angular-error-interceptor-4b102f938065>

===

**USE HTTERRORRESPONSE INSTEAD OF THE ABOVE LONG APPROACH if you want**

// first import the HttpErrorResponse module

import { HttpErrorResponse } from '@angular/common/http';

Then create an object of **HttperrorResponse** by passing it as a second argument to the subscribe() method. Then use it display error incase.

// ngOnnit method

ngOnInit(): void {

    this.angserice.getservadata().subscribe(

      data => {this.employeedata = data},

      // Create an object variable of httpErrorREsponse

      (err: HttpErrorResponse)=>{

      // use the above err object to show an alert incase error happens

        alert(err.message);

      }

);

  }

(Done with service)

**============================================**

**CORS ISSUE**

//Also if you entcounter cors issue solve it with this guide = <https://www.youtube.com/watch?v=OjmZPPKaj6A&t=92s>

**Cors** 🡺 means application running on another domain/server cannot reference application on difference domain/server.

Eg. <http://localhost:4200/> trying to fetch data from <http://localhost:8080> there will be cors error unless the back-end sever explicit tells it server to allow other domains to reference it.

It happens because your angular apps runs on it on localhost url server & it will try to fetch data from other different http server

**============================================**

**The “Environments” folder**

The “Environments” folder in angular is where you should save secret information’s like database configuration details. It’s just like the .env file in node.js

**Environment.pro.ts** = use during production mode, that’s when you deploy your project to live server.

**Environment.ts** = use during development mode, it what you should use during development on local host server.

The environment.ts contains an export “environment” object and you should add your secret details there and reference it in component or service with environment.object\_Name. Eg.

// Inside Environment/environment.ts declare an object “apiBaseUrl” and assign a key to it. The production: false, comes with angular

export const environment = {

    production: false,

  // declare object and assign key to it.

    apiBaseUrl: "http://localhost:8082"

  };

  // Then Inside app.component.ts import the environment and reference the above “apiBaseUrl”

  import { environment } from './../environments/environment';

  export class AppComponent {

    private apiServiceUrl=environment.apiBaseUrl;

  }

===========================

**DISABLED BUTTON**

The “disabled” attribute of the <button> tag originates from HTML and not angular. It’s makes the button clickable or not.

Mostly you will see it used in plain HTML in a way like…

 <button disabled>Click</button>

The absence of the disabled property indicate that the button is clickable. This is because you cannot set Boolean values like “true” and “false” to Boolean properties in HTML. For example, below code will all mark the button disabled even though the disabled property one is set to true and the other false…

    <button disabled="true">Click 1</button> // disabled button

    <button disabled="false">Click 2</button> // disabled button



**But in angular with the help of property-binding you can do it…**

The hack is that, when you bind the “disabled” property to a Boolean value, Angular will “show” the disabled property when set to true. And it will “hide” it when set to false.

  Inside app.components.html

<!--NON disabled button -->

    <button [disabled]="false">Click 1</button>

    <!--

    When you inspect it in the browser it will rendered as

    <button>Click</button>

    So you can see, Angular will just hide the "disabled" property when it set to false

    -->

<!-- disabled button -->

    <button [disabled]="true">Click 2</button>

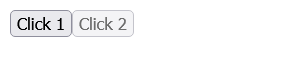
    <!--

    When you inspect it in the browser it will rendered as

    <button disabled>Click</button>

    So you can see, Angular will show the "disabled" property when it set to true

    -->

****

==============================

**====== REACTIVE FORM =====**

ANGULAR HAS TWO TYPE OF FORM IMPLEMENTATION:

1) Template driven form == form validation is done in app.component.html is basically use for small form project

2) Reactive Form == Form validation is done in app.component.ts It’s the recommended way to create form So always use reactive form.

// Inside app.module.ts import the ReactiveFromsModule and register it

import { ReactiveFormsModule } from '@angular/forms'

@NgModule({

imports:[ReactiveFormsModule]

})

// Then create a form in app.component.html

// "[formGroup]" = angular required that you give every form a name so that if there is more than one

//  form in the view it will know which one to call by using it name so [formGroup]="formName"

// "formControlName" = Angular uses it as the normal "name" html attribute for every input

// (ngSubmit)="getformvalues()" = Which function to call in app.component.ts when the submit button is clicked

<form [formGroup]="mynewform" (ngSubmit)="getformvalues()">

  <input type="email" formControlName="email" placeholder="email" />

  <br />

//am using this to check the above email input field errors 'get()' is a method of FormGroup class defined by angular

// it gives you the ability to check if the input field is touched, untouched, valid, invalid

  <p \*ngIf="mynewform.get('email').invalid && mynewform.get('email').touched">

    Email is required

  </p>

// you can also use it to access the validators given below as you give as

<p \*ngIf="mynewform.get('email').errors.required">

    Email is required

  </p>

  <br />

  <input formControlName="password" type="password" placeholder="password" />

  <br />

//disable the submit buttom if any of the form values is invalid

  <button [disabled]="mynewform.invalid">Submit</button>

</form>

//THEN inside app.component.ts import "FormGroup", "FormControl" and “Validators” class

import { FormGroup, FormControl, Validators } from '@angular/forms';

export class AppComponent {

// "mynewform" = below is the form name given to the form in app.component.html [formGroup]="mynewform"

// "email" and "password" = represent the input names in app.component.html formControlName="email", formControlName="password"

// "FormControl" takes in two parameters or 1: FormControl(Default\_InputValue, [array\_Of\_Validators]) or FormControl(Default\_InputValue)

mynewform = new FormGroup({

     email: new FormControl('', [Validators.required, Validators.email]),

    password: new FormControl('')

  });

// The function the submit button will call. Below will log all the form values

  getformvalues() {

    console.log(this.mynewform.value);

// Or use console.log(this.mynewform.getRawValue());

  }

}

// CUSTOM VALIDATION FORMMAT

// the “key” below is of type String. And it indicates that below function “classValidator()” will either return “Boolean” or “null”

function classValidator(control: AbstractControl) : {[key : string] : boolean} | null {

return null;

}

Control: represent the html <input> element

//passing parameters to it by wrapping it in factory function in custom validation

function ageVAlidator(min:number, max:number){

    return (control: AbstractControl): {[key:string]:boolearn} | null => {

    if(control.value !==null && (isNaN(control.value) || control.value < min || control.value > max )){

       return {"ageValidator":true}

    }

      return null;

    };

    }

==============

FormGroup = represent the whole html form container thats <form> </form>

formControlName = represents a single html input element <input> </input>

Formbuilder = is a service that you need to inject it in the constructor of appcomponent.ts it helps to make your code small

constructor(private fb: FormBuilder) {}

// "formGroup" bellow represents   <form [formGroup]="formGroups" (ngSubmit)="register()">

formGroups:FormBuilder;

//beloow should be in ngOninit method

 this.formGroups= fb.group({

// useName below represent  <input type="text" placeholder="username" formControlName="userName" />

userName:['',Validatiors],

userEmail:['',Validatiors.compose({[Validators.required, Validators.minLength(5)])],

});

// use getters to get each input tag and reference it in html

 get userName() {

    return this.newformGroup.get('userName');

  }

  get password() {

    return this.newformGroup.get('password');

  }

form methods used inside html

[formGroup]

(ngSubmit)

formControlName

form methods used inside components.ts

group()

Validators.compose([]);

===========================

**LAZY LOADING**

Is a ways of telling angular not to load certain modules until some link is clicked. We do this simply to increase performance of the app loading time. Because an app may contain many modules and because the root module app.module.ts is boostraped at the load of the app, all the modules imported in it will also be loaded. This increases the loading time of the app.

The whole concept is that, We don’t register and import modules we want to lazy load in app.module.ts. We only load such module until some link is clicked. So never register or import a module you want to lazy load in app.module.ts, if you do, the module will be loaded at run time of the app. Eg.

//Create a module called “success”

ng g m success –routing

//Create a component inside the “success” module

Ng g c success/justice

//Remember I don’t have to register or import the “success” module

//Inside app.component.html define a link that will be clicked to open the success component

<a routerLink="/success">Success</a>

  <router-outlet></router-outlet>

//Then inside app.routing.module.ts define the above path which when clicked will load the “success” module

//The import() methods contains the source path of the module. The then() method contains an arrow function with parameter “M’ which the Class of the //module is assigned to it. All what the two methods does is to import and register the “success” module when the “success” link path is clicked

const routes: Routes = [

  {

    path: 'success',

    loadChildren: () =>

      import('./success/success.module').then(m => m.SuccessModule)

  }

];

//Great now, your success module is imported & registered. But how will the “success” module know which of its component to load?

//So inside success.routing.module.ts define a default path and the component to load.

const routes: Routes = [{ path: '', component: SuccessComponent }];

//That is, it, the above will load the “successcomponent” inside the success module only when the “/succes” link path is clicked in app.component.tss

===========================

**SESSIONS**

WE USE LOCALSTORAGE IN ANGULAR FOR KEEPING USER DATAILS IN THE BROWSER AS A SESSION.

To set a sesion

// Use localStorage.setItem(“key”,”value\_user\_id”) to set a session on one component. Bellow am storing a user laravel api token to the browser

//Once the bellow local storage is set, you can retrieve the value with the key in any component.

  this.http.post(loginUrl, formData).subscribe(

      (result: any) => {

        //Store the user token in localstorage as a session

        localStorage.setItem('token', result.access\_token);

        //redirect the user to his profile page

        this.route.navigateByUrl('profile');

      },

To retrieve a session

//Use localStorage.getItem(“local\_storage\_Key”) to retrieve the above session in any component

  ngOnInit(): void {

     userToken = localStorage.getItem('token'); //get the above userToken from the broswer

    }

To delete a session

To delete a session and logout user user localStorage.removeItem(“Key”) this will delete the localStorage with such key from the browser

    localStorage.removeItem('token');

To show only the login, and signup button only when the user is not login use this.

// First set “islogin” property to false.

// Then inside ngOninit() method, check if a localStorage with usertoken or id is not null and assign it to “islogin”

// (it will return true if such key contains some value)

//PROPERTIES

  islogin = false

//ONINIT

  ngOnInit(): void {

    this. islogin = localStorage.getItem('token') !== null;

  }

/\* Then in the template, display the signup, and the login button only when the “islogin” is false

Then display the logout button when islogin is true. (don’t worry about the “logout” function, check bellow) \*/

  <a class="re" routerLink="signup" \*ngIf="!islogin">SignUp</a>

  <a class="re" routerLink="" \*ngIf="!islogin">Login</a>

  <a class="re" routerLink="profile" \*ngIf="islogin" (click)="logOut()">LogOut</a>

/\* Then create a logout() function in the component that will remove the localstorage key and redirect the user to the homepage if he clicks on the logout button. \*/

  //METHOD TO LOGOUT USERS

  logOut() {

    localStorage.removeItem('token');

  }

===========================

**--skip-import**

//Adding **--skip-import to any angular cli code will prevent such module or component from being imported and registered in app.module.ts**

**Ng g c oxford --skip-import**

dynamically.

===========================

===========================

**Angular quick tips**

\*) To generate a component that comes with both responsive navbar and sidebar.

// First, Type this in CLI by navigating to your app folder first “ng g @angular/material:material-nav --name=your\_Navbar\_Name\_Here”

// The Above command will create a component with navbar and sidebar in it html template. Copy the selector of this component and paste it in app.component.html to display it.

**Custom reactive forms validation**

// INSIDEDE HTML

    <form [formGroup]="bundleform" (ngSubmit)="onDone()">

    <input required type="text" name="amount" placeholder="ex: 100" formControlName="bundlenumber">

     <!--Validators-->

        <p style="color:red;" \*ngIf="bundleform.get('bundlenumber').invalid">

            Invalid character (only numbers are allowed)

        </p>

        <p style="color:red;" \*ngIf="bundleform.get('bundlenumber').errors?.['required']">

            Bundle number is required

        </p>

        <button id="done\_btn" mat-flat-button color="primary" [disabled]="bundleform.invalid">Done</button>

    </form>

   // INSIDEDE COMPONENT

     bundleform:FormGroup;

  constructor(

    private fb: FormBuilder

    ) {}

    ngOnInit(): void {

    this.bundleform = this.fb.group({

      bundlenumber: ['',Validators.compose([Validators.required, Validators.pattern("^[0-9]\*$")]) ]

      // use the below as it custom validator

      phoneNumber: ['', this.numeric]

    });

    }

        // Custom validadors

         numeric(control: AbstractControl) {

        let val = control.value;

        if (val === null || val === ''){

          return null;

        }

        if (!val.toString().match(/^[0-9]+(\.?[0-9]+)?$/)){

          return { 'invalidNumber': true };

        }

        return null;

      }