**Angular Course**

**Bootstrap process**

**Angular.json (**here it will be mentioned from which file we need to start/build)

**main.ts**(It says which module we need to bootstrap and same will be imported as well by default Appmodule)

**Appmodule.ts** file (It says which component we need to bootstrap by default AppComponent which needs to be imported as well)

**AppComponent** (using the selector tag we can add it in index.html file)

**Index.html** file (Since this file name is present in angular.json file it gets bootstrapped if required we can have our own html file created but it should be added in angular.json file)

**Routing Module:**

To create a separate routing module, we need to import **Routes** and **RouterModule** from @angular/router into the module.

* Create an array of constant variable and assign Routes as an interface with an array of paths we want to configure.
* Define an NgModule with imports and exports inside it where imports and exports are array.
* Create a class with some name, we need to export this class into our main module file to make use of this Routing Module
* To Make Routing work we need to have below base href tag in our index.html

<basehref="/">

**Below is the sample Basic Routing module.**

import{ Routes,RouterModule } from'@angular/router';

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

const routes : Routes = [

{path :'',redirectTo : 'Home',pathMatch : 'full'},

{path :"Home",component : HomeComponent},

]

@NgModule({

imports : [RouterModule.forRoot(routes)],

exports : [RouterModule]

})

exportclassAppRoutingModule {

}

To show our body of the components we need to mentioned **router-outlet** tag where ever we want to show

**<router-outlet></router-outlet>**

For tagging the paths we need to have the paths linked to the **routerLink** (directive)

<a class="navbar-brand" routerLink = '/'>Routing Concepts</a>

<li><arouterLink = 'Home'>Home</a></li>

// or we can do it with a property binding

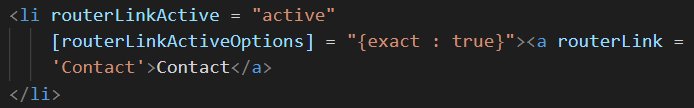
<a[routerLink]="['/about']"> About Us </a>

For complex paths



To make our navigation menu as active when we are in a certain components we have to add **routerLinkActive = “active”** (directive) and also **[routerLinkActiveOptions] = “{exact : true}”**

**Here “active” will be the class name, we can have our own class also which can be used in css.**



If we want to load a component apart from selecting it from the menu (i.e., from click event or from a method), we can use **navigate** method for which we need to import router class as an interface. Below is the basic example.

privateroute : Router

reload(){

this.route.navigate(['/Admin']);

}

Difference between navigate and routerLink is routerLink knows in which path we are currently in and navigate method doesn’t know the path we are in.

**Passing Query Params to url**

For passing query string parameters import Router and use navigate method on it like below.

Private route : Router

Example:

Localhost:4200/getTour?id:100

getTourDetails(tour){

    this.router.navigate(['/getTour'], { queryParams: { id: tour.\_id } });

  }

**We can even pass queryParams from HTML like below.**

Example : url….?allowEdit=false&isAdmin=true

[queryParams] = "{allowEdit : false,isAdmin : true}"

**Fetching query params from URL**

For getting the queries from the URL we need to subscribe by using ActivatedRoute

private activatedRoute: ActivatedRoute

this.activatedRoute.queryParams.subscribe(data => {

console.log(data);

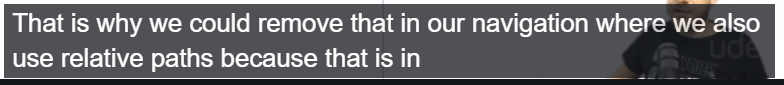
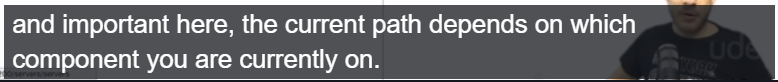
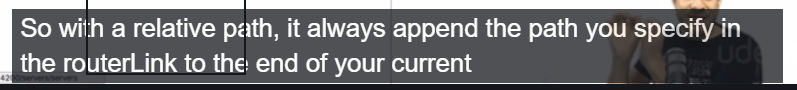
});

**If we want additional paths we can do below in html, so here URL would look like** <http://localhost:4200/Employee/1/Prudhvi>

[routerLink]="['/Employee',employee.id,employee.name]"

**About relative routing paths**

Relative Path:



**Children Routing:** As your application grows more complex, you may want to create routes that are relative to a component other than your root component. These types of nested routes are called child routes. This means you're adding a second <[router-outlet](https://angular.io/api/router/RouterOutlet)> to your app, because it is in addition

 routes: Routes = [

    {

      path: "first-component",

      component: FirstComponent,

      children: [

        { path: "child-a", component: ChildAComponent },

        { path: "child-b", component: ChildBComponent },

      ],

    },

  ];

So here path’s will be **first-component/child-a** or **first-component/child-b**

And since these (child-a and child-b) are children to parent (first-component) we need to add <router-outlet> at the appropriate place in parent component i.e., first-component.

This strategy will avoid writing multiple path in the routing module like below

const route : Routes = [

  {path : 'Employees',component : EmployeessComponent},

  {path : 'Employee/:id/:name',component : EmployeeComponent}

]

**Services:**

Services help us not to repeat the same code in multiple components.

To create a service, we need to import **Injectable** form @angular/core

**ng g service serviceFileName --module=app.module**

**Basic Structure of Service.**

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

@Injectable({

providedIn: 'root'

})

exportclassBasicService {

constructor() { }

name :basicInterface[] = [

{

id :25,

name :'Prudhvi Karanam',

Profession :'Senior Software Engineer',

image :'../../assets/1014426\_607990045937868\_1880709065\_n.jpg'

},

{

id :32,

name :'Brad Traversy',

Profession :'Professional Online Tutor'

}

]

}

**HTTP calls using Services:**

To make HTTP calls we need to import HTTPClientModule from @angular/http/core (may vary depending on the version) in main module file.

Along with the above we also need to import HTTPCLient into the service file which we created.

Now in the service file we need to create a method in which we need to return/call a get, post, delete etc methods.

In the component file we need to call this service using subscribe method.

**Basic Service file with get method:**

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

import {basicsInterface} from'./basics.interface';

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

@Injectable({

providedIn: 'root'

})

exportclassBasicsService {

constructor(private http : HttpClient) { }

getprofiles(){

returnthis.http.get('http://localhost:3000/Users');

}

}

Component file in which we are calling service

profiles;

constructor(privatebasicsService : BasicsService) { }

ngOnInit() {

this.basicsService.getprofiles().subscribe((values) => {

returnthis.profiles = values;

})

}

**Reactive Forms**

To work with reactive forms, we need to import **ReactiveFormsModule** into main module.

ReactiveFormsModule imports us few classes and interfaces (Form Control, Form Group, Form Builder, Form Validator).

Key Points to remember:

Using this reactive approach data in html and ts file will always be sync. i.e., like [(ngModel)],

**but [(ngModel)] is not supported by Reactive approach**.

Below are directives for Reactive form approach

* Ng-pristine: new control (input, text-area, radio button etc).
* Ng-dirty: data modified.
* Ng-touched: control touched; it is active control.
* Ng-untouched: control untouched; it is inactive control.
* Ng-valid: control valid.
* Ng-invalid: control invalid.

Basic Reactive form

customerForm : FormGroup;

  constructor(private fb:FormBuilder) {

    this.customerForm = fb.group({

      firstName : [""]

    })

   }

<p>

  reactive works!

</p>

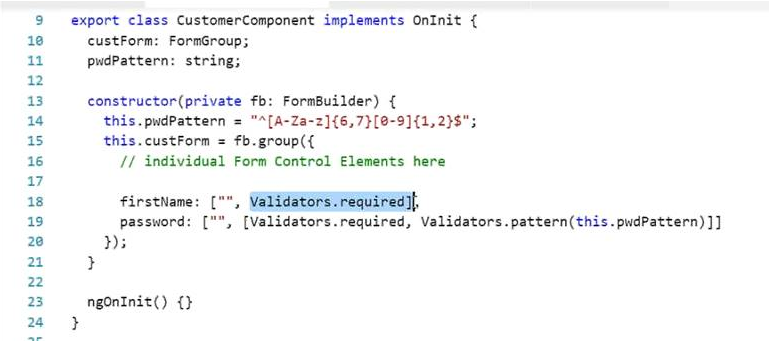
<form [formGroup] = "customerForm">

  <input type="text" id='email' formControlName = 'firstName'>

</form>

With Validators:

**Component File**

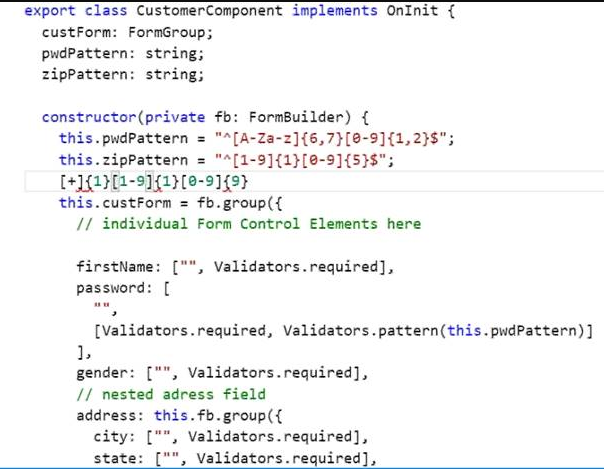


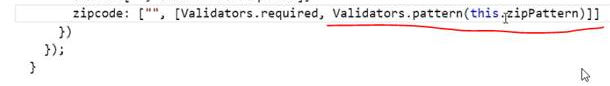
**HTML file**

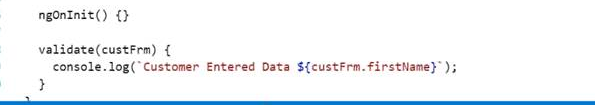


**Nested Form**

Component file







For Nested forms we need to mentioned attribute as formGroupName (formGroupNam e should always be added inside [**formGroup]** , else nested form won’t be working).

**Nested form HTML:**



**Pipes**

Pipes takes some value and transforms/formats that into another value.

Angular comes with several built-in pipes and in addition we can create own custom pipe.