# **Start Process of Angular Project**

$ Ng server

->

**/src/main.ts**

platformBrowserDynamic().bootstrapModule(AppModule);

->

**/src/app/app.module.ts**

@NgModule({

…

Bootstrap:[AppComponent]

})

->

**/src/app/app.component.ts- >**This is the root component!

# **Component**

(In this example, component name is server)

**/src/app/server/server.component.ts**

Import {Component} from ‘@angular/core’;

@Component({

Selector:’app-server’ <- Will used like <app-server></app-server>

Or Selector:’[app-server]’ <- Will used like <div app-server></div>

Or Selector:’.app-server’ <- Will used like <div class=”app-server”></div>

templateUrl: ‘./server.component.html’

or template: `

<other-app></other-app>

<h1>whatever</h1>`

styleUrls:[‘./server.component.css’, ‘blah.css’]

or styles: [`

h3{ color:blue;}

`]

})

export class ServerComponent {

}

**src/app/server/server.component.html**

write html tag

**/src/app/app.module.ts**

Import {ServerComponent} from ‘./server/server.component’;

@NgModule({

Declarations: [

..,

ServerComponent]

…

})

This process is possible by **$ ng g c server**

**$ ng g c server –spec false -> exclude test file such as …wwspecs.ts**

# Data Binding

**Logic -> Template (Ouput Data)**

String Interpolation {{data}}

Property Binding [property]=”data”

**Template to Logic (React to Event)**

Event Binding (event)=”expression”

**Two-Way-Binding – Combine above two**

[(ngModel)] = “data”

**String Interpolation {{data}} Example**

**./src/app/server/server.component.ts**

export class ServerComponent {  
 serverID: number = 10;

serverStatus: string = ‘offline’;

getServerStatus(){

return this.serverStatus;

}

}

**./src/app/server/server.component.html**

<p>{{ ‘Server’ }}with ID {{ serverID }} is {{ getServerStatus() }} </p>

* Server with ID 10 is offline

**Property Binding [property]=”data” Example**

**./src/app/servers/servers.component.ts**

export class ServersComponent impoements OnInit {

allowNewServer = false;

constructor(){

setTimeout(()=>{

this.allowNewServer = true; },2000);

}

}

**./src/app/servers/servers.component.html**

<button class=”btn btn-primary” [disabled]=”!allowNewServer”></button>

<p [innerText]=”allowNewServer”></p>

**Event Binding (event)=”expression” Example**

**./src/app/servers/servers.component.ts**

export class ServersComponent impoements OnInit {

serverCreationStatus = “No server was created”;

onCreateServer(){

this.serverCreationStatus =”Server was created”;

}

}

**./src/app/servers/servers.component.html**

<button

class=”btn btn-primary”

(click)=”onCreateServer()” ></button>

<p>{{ serverCreationStatus }}<p>

**One way Binding Without ngModel -> Input tag is not effected by logic**

**./src/app/servers/servers.component.ts**

export class ServersComponent impoements OnInit {

serverName=’’;

onUpdateServerName(event: Event){

this.severName = (<HTMLInputElement>event.target).value;

}

}

**./src/app/servers/servers.component.html**

<input

type=”text”  
 class=”form-control”

(input)=”onUpdateServerName($event)”>

<p>{{ serverName }}<p>

**Two Way Binding using [(ngModel)] = “data” – input and, p tag change concurrently.**

For Two-Way-Binding to work, you need to enable the ngModel  directive. This is done by adding the FormsModule  to the imports[]  array in the AppModule.

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

**./src/app/servers/servers.component.ts**

export class ServersComponent impoements OnInit {

serverName=’’;

onUpdateServerName(event: Event){

this.severName = (<HTMLInputElement>event.target).value;

}

}

**./src/app/servers/servers.component.html**

<input

type=”text”  
 class=”form-control”

[(ngModel)]=”serverName”>

<p>{{ serverName }}<p>

# ngIF Structural Directives

**Structural Directives -> Add or Remove elements**

**./src/app/servers/servers.component.ts**

export class ServersComponent impoements OnInit {

serverName=’’;

serverCreated=false;

onCreateServer(){

this.serverCreated=true;

this.serverCreationStatus =”Server was created”;

}

onUpdateServerName(event: Event){

this.severName = (<HTMLInputElement>event.target).value;

}

}

**./src/app/servers/servers.component.html**

<input  
 type=”text”  
 class=”form-control”  
 [(ngModel)]=”serverName”>

<button

class=”btn btn-primary”   
 (click)=”onCreateServer()” ></button>

<p **\*ngIf**=”serverCreated; else noServer”>Server was created, server name is {{ serverName }}<p>

<ng-template #noServer>

<p>No server was created!</p>

</ng-template>

# ngStyle ngClass Attribute Directives

**Attribute Directives -> Change the element**

**./src/app/server/server.component.ts**

@Component({

selector:’app-server’,

templateUrl:’.server.component.html’,

styles:[‘

.online{

color:white;

}’]

})

export class ServerComponent {  
 serverID: number = 10;

serverStatus: string = ‘offline’;

constructor(){

this.serverStatus = Math.random()>.5?’online’:’offline’

}

getColor(){

return this.serverStatus === ‘online’ ? ‘green’ : ‘red’;

}

getServerStatus(){

return this.serverStatus;

}

}

**./src/app/server/server.component.html**

* **This is property binding of directives.**

<p

[ngStyle]=”{backgroundColor:getColor()}”  
[ngClass]”{online:serverStatus===’online’}”>

{{ ‘Server’ }}with ID {{ serverID }} is {{ getServerStatus() }} </p>

# ngFor Structural Directives

**./src/app/servers/servers.component.ts**

export class ServersComponent impoements OnInit {

serverName=’’;

serverCreated=false;

servers = [‘Testserver’,’Testserver 2’];

onCreateServer(){

this.serverCreated=true;

this.servers.push(this.serverName);

this.serverCreationStatus =”Server was created”;

}

onUpdateServerName(event: Event){

this.severName = (<HTMLInputElement>event.target).value;

}

}

**./src/app/servers/servers.component.html**

<input  
 type=”text”  
 class=”form-control”  
 [(ngModel)]=”serverName”>

<button

class=”btn btn-primary”   
 (click)=”onCreateServer()” ></button>

<p **\*ngIf**=”serverCreated”>Server was created, server name is {{ serverName }}<p>

<app-server \*ngFor=”let server of servers”></app-server>

# How to add bootstrap

$ npm install –save bootstrap

**./.angular-cli.json**

“apps”:[{

“styles”:[

“../node\_modeuls/bootstrap/dist/css/bootstrap.min.css” -> depends on installed directory

“styles.css”],

}]

# Binding Custom Properties (Parent -> (property) -> Child)

**./app.component.ts**

export class AppComponent{

serverElements = [{type:’server’,name:’Testserver’,content:’Just a test!’}];

}

**./app.comoponent.html**

<app-server-element

\*ngFor=”let serverElement of serverElements”

[element]=”serverElement”>

**or** [srvElement]=”serverElement”> -> for set Alias

</app-server-element>

**./server-element/server-element-component.ts**

import {Component,Input} from ‘@angular/core’;

@Component({ selector:’app-server-element’ …})

export class ServerElementComponent {

@Input() element:{type: string, name: string, content: string});

**or** Input(‘srvElement’) element:{type: string, name: string, content: string}); -> if Alias is set outside the component

}

**./server-element/server-element-component.html**

{{element.name}}

<strong \*ngIf=”element.type ===’server’” style=”color:red”>{{element.content})</strong>

<em \*ngIf=”element.type ===’blueprint’”>{{element.content}}</em>

# Binding Custom Event (Child -> (Event) -> Parent)

**./app.component.ts**

export class AppComponent{

serverElements = [}type:’server’,name:’Testserver’,content:’Just a test!’}];

onServerAdded(serverData:{serverName:string, serverContent:string}){

this.serverElements.push({

type:’server’,

name:serverData.serverName,

content: serverData.serverContent});

}

onBlueprintAdded(blueprintData:{serverName:string, serverContent:string}){

this.serverElements.push({

type:blueprint,

name: blueprintData.serverName,

content: blueprintData.serverContent});

}

}

**./app.comoponent.html**

<app-cockpit

(serverCreated)=”onServerAdded($event)”

(bluePrintCreated)=”onBlueprintAdded($event)”

**or** (bpCreated)=”onBlueprintAdded($event)” -> If alias is set in the child component

></app-cockpit>

<app-server-element

\*ngFor=”let serverElement of serverElements”

[element]=”serverElement”>

**or** [srvElement]=”serverElement”> -> for set Alias

</app-server-element>

**./cockpit/cockpit.component.ts**

@import {Component,EvenEmitter, Output} from ‘@angular/core’

export class CockpitComponent {

@Output() serverCreated = new EventEmitter<{serverName:string, serverContent:string}>();

@Output() blueprintCreated= new EventEmitter<{serverName:string, serverContent:string}>();

**or** @Output(‘bpCreated’) blueprintCreated= new EventEmitter<{serverName:string, serverContent:string}>(); -> set Alias

newServerName=’’;

newServercontent

onAddServer(){

this.serverCreated.emit({

serverName:this.newServerName,serverContent:this.newServerContent});

}

onAddBlueprint(){

this.blueprintCreated.emit({

serverName:this.newServerName,serverContent:this.newServerContent});

}

**./cockpit/cockpit.component.html**

<input type=”text” class=”form-control” [(ngModel)]=”newServerName”>

<input type=”text” class=”form-control” [(ngModel)]=”newServerContent”>

<button

class=”btn btn-primary”

(click)=”onAddServer()”>Add Server</Button>

<button

class=”btn btn-primary”

(click)=”onAddBlueprint()”>Add Server</Button>

# View Encapsulation

By default, each css is only applied to its own view. In rendered view, all tag will have its distinguished attribute such as “\_ngcontent-ejo-1”.

import {..ViewEncapsulation} from ‘@angular/core’

@Component({

…

encapsulation: ViewEncapsulation.None -> Html tag won’t have any distinguished attribute. It’s css will be applied globally.

encapsulation: ViewEncapsulation.Naitve -> Default Mode + Ignore global CSS on this component

encapsulation: ViewEncapsulation.Emulated -> Default Mode

})

# Local Reference and View Child in Template

**./cockpit/cockpit.component.html**

<input type=”text” class=”form-control” #serverNameInput>

<input type=”text” class=”form-control” #serverContentInput >

<button

class=”btn btn-primary”

(click)=”onAddServer(serverNameInput)”>Add Server</Button>

<button

class=”btn btn-primary”

(click)=”onAddBlueprint(serverNameInput)”>Add Server</Button>

**./cockpit/cockpit.component.ts**

@import {Component,EvenEmitter, Output, ViewChild} from ‘@angular/core’

export class CockpitComponent {

@Output() serverCreated = new EventEmitter<{serverName:string, serverContent:string}>();

@Output(‘bpCreated’) blueprintCreated= new EventEmitter<{serverName:string, serverContent:string}>();

@ViewChild(‘serverContentInput’) serverContentInput : ElementRef

onAddServer(nameInput: HTMLInputElement){

this.serverCreated.emit({

serverName:nameInput.value, -> Local Reference Style, passed from event

serverContent:this.serverContentInput.nativeElement.value }); -> ViewChild Style, directly access to the template dom

}

onAddBlueprint(nameInput: HTMLInputElement){

this.blueprintCreated.emit({

serverName:nameInput.value,

serverContent: this.serverContentInput.nativeElement.value });

}

# Contents inside Ng Component tag

By default, if <app-root><p>something</p></app-root>, <app-root> is ng component, <p>something</p> will be ignored.

To avoid this,

inside the template, for example, app.component.html,

Put <ng-content></ng-content> where the added content should be inserted.

# Content Child

**./app.component.ts**

export class AppComponent{

serverElements = [{type:’server’,name:’Testserver’,content:’Just a test!’}];

}

**./app.component.html**

<app-server-element

\*ngFor = “let serverElement of serverElement”>

<p #contentParagraph>

{{serverElement.name}} {{serverElement.content}}

</p>

</app-server-element>

**./server-element/server-element-component.html**

@import {…ContentChild} from ‘@angular/core’

…  
export class ServerElementComponent implements

@ContentChild(‘contentParagraph’) paragraph: ElementRef;

ngAfterContentInit(){

console.log(‘Text Content of paragraph: ‘ + this.paragraph.nativeElement.textContent)

* will show ‘Test Server Just a test’

}

# Component Life Cycle

ngOnChanges: Called after a bound input property changes

ngOnInit: Called once the component is initialized

ngDoCheck: Called during every change detection run

ngAfterContentInit: Called after content (ng-content) has been projected into view

ngAfterContentChecked: Called every time the projected content has been checked

ngAfterViewInit: Called after the component’s view (and child views) has been initialized.

ngAfterViewChecked: Called every time the view (and child views) have been checked.

ngOnDestroy: Called once the component is about to be destroyed

These hooks should be imported and implemented on class.

# Creating Directive

**./basic-highlight/basic-highlight.directive.ts**

import {Directive} from ‘@angular/core’;

@Directive({

selector:’[appBasicHighlight]’

})

export class BasicHighlightDirective implements OnInit{

constructor(private elementRef: ElementRef) { <- private make it getter setter

}

ngOnInit(){

this.elementRef.nativeElement.style.backgroundColor = ‘green’;

}

}

**./app.module.ts**

..

@NgModule({

declarations: [

AppComponent,

BasicHighlightDirective

],….})

**./app.component.html**

<p appBasicHighlight>Style me</p>

# Renderer for a better Directive

**Use this rather than basic directive, since basic directive will not work in server-side. Detail methods are here, https://angular.io/api/core/Renderer2**

$ng g d better-highlight

delete spec file

create new folder

move directive file to the folder, and change path in app.module.ts

**./better-highlight.directive.ts**

import { Directive, OnInit, ElementRef, Renderer2 } from ‘@angular/core’;

@Directive({

selector: ‘[appBetterHighlight]’

})

export class BetterHighlightDirective implements OnInit{

constructor(private elRef: ElementRef, private renderer: Render2){ }

ngOnInit(){

this.renderer.setStyle(this.elRef.nativeElement, ‘background-color’, ‘blue’)

}

}

**./app.component.html**

<p appBetterHighlight>Style</p>

# Host Listener

**./better-highlight.directive.ts**

import { Directive, OnInit, ElementRef, Renderer2, HostListener } from ‘@angular/core’;

@Directive({

selector: ‘[appBetterHighlight]’

})

export class BetterHighlightDirective implements OnInit{

constructor(private elRef: ElementRef, private renderer: Render2){ }

ngOnInit(){}

@HostListener(‘mouseenter’) mouseover(eventData: Event){

this.renderer.setStyle(this.elRef.nativeElement, ‘background-color’, ‘blue’, false, false)

}

@HostListener(‘mouseleave’) mouseleave(eventData: Event){

this.renderer.setStyle(this.elRef.nativeElement, ‘background-color’, ‘transparent’, false, false)

}

}

**./app.component.html**

<p appBetterHighlight>Style</p>

# Host Binding

**./better-highlight.directive.ts**

import { Directive, OnInit, ElementRef, Renderer2, HostListener, Hostbinding } from ‘@angular/core’;

@Directive({

selector: ‘[appBetterHighlight]’

})

export class BetterHighlightDirective implements OnInit{

@HostBinding(‘style.backgroundColor’) backgroundColor: string = ‘transparent’;

constructor(private elRef: ElementRef, private renderer: Render2){ }

ngOnInit(){}

@HostListener(‘mouseenter’) mouseover(eventData: Event){

this.backgroundColor = ‘blue’

}

@HostListener(‘mouseleave’) mouseleave(eventData: Event){

this.backgroundColor = ‘transparent’

}

}

**./app.component.html**

<p appBetterHighlight>Style</p>

# Binding to Directive Properties

**./better-highlight.directive.ts**

import { Directive, OnInit, ElementRef, Renderer2, HostListener, Hostbinding, input } from ‘@angular/core’;

@Directive({

selector: ‘[appBetterHighlight]’

})

export class BetterHighlightDirective implements OnInit{

@Input() defaultColor: string = ‘transparent’;

@Input() highlightcolor: string = ‘blue’;

@HostBinding(‘style.backgroundColor’) backgroundColor: string;

constructor(private elRef: ElementRef, private renderer: Render2){ }

ngOnInit(){

this.backgroundColor = this.defaultColor;

}

@HostListener(‘mouseenter’) mouseover(eventData: Event){

this.backgroundColor = this.highlightColor;

}

@HostListener(‘mouseleave’) mouseleave(eventData: Event){

this.backgroundColor = this.defaultColor;

}

}

**./app.component.html**

<p appBetterHighlight defaultColor=”yellow” [highlightColor]=”’red’”>Style</p> <- Differnent binding style is applied (with bracket, single quotation mark or not)

# Star Syntax transforms to

<div \*ngIf=”!onlyOdd”>

<li> something..</li>

</div>

is actually converted into,

<ng-template [ngIf]=”!onlyOdd”>

<div \*ngIf=”!onlyOdd”>

<li> something..</li>

</div>

</ng-template>

# Custom Structural Directive

$ ng g d unless

remove spec file

**./unless.directive.ts**

import { Directive, Input, TemplateRef, ViewContainerRef } from ‘@angular/core’;

@Directive({

selector: ‘[appUnless]’

})

export class UnlessDirective {

@Input() set appUnless(condition: boolean){

if (!condition) {

this.vcRef.createEmbeddedView(this.templateRef);

} else {

this.vcRef .clear();

}

}

constructor(private templateRef: TemplateRef<any>, private vcRef: ViewContainerRef) { }

}

**./app.component.html**

<div \*appUnless=”!onlyOdd”>

<li> something </li>

</div>

# Structural Directive ngSwitch

**./app.component.ts**

*…*

*export class AppComponent {*

value *= 10;*

*}*

**./app.component.html**

*<div [*ngSwitch*]=”value”>*

*<p \**ngSwitchCase*=”5”> value is 5</p>*

*<p \**ngSwitchDefault*> value is Default</p>*

*</div>*

# Creating a Service for utility

**./logging.service.ts**

*export class LoggingService {*

*logStatusChange(status:string){*

*console.log(‘Status is ‘+status);*

*}*

*}*

./app.component.ts

**./new-account/new-account.component.ts**

*..*

*import {*LoggingService*} from ‘../logging.service’;*

*..*

*@Component({*

*…*

providers*: [*LoggingService*]*

*})*

*export class NewAccountComponent {*

*…*

constructor*(private* loggingService*:* LoggingService*) { }*

* *above line is same with below instanciate process.*

*const service = new LoggingService();*

*service.logStatusChange(accountStatus);*

*this.*loggingService*.logStatusChange(accountStatus);*

*}*

# Service for Data Storage

**./accounts.service.ts**

*export class AccountsService {*

*accounts = [*

*{*

*name: ‘Master Account’,*

*status: ‘active’*

*},*

*{*

*name: ‘Testaccount’,*

*status: ‘inactive’*

*}*

*]*

*addAccount(name:string, status: string){*

*this.accounts.push({name:name, status:status});*

*}*

*updateStatus(id: number, status:string){*

*this.accounts[id].status = status;*

*}*

*}*

**./app.component.ts**

*import { Component } from ‘@angular/core’;*

*import { AccountsService } from ‘./accounts.service’;*

*@Component({*

*…*

*selector: ‘app-root’,*

*providers: [AccountsService]*

*})*

*export class AppComponent {*

*accounts: {name: string, status: string}[] = [];*

*constructor(private accoutsService: AccountsService) {}*

*ngOnInit() {*

*this.accounts = this.accountsService.accounts;*

*}*

*}*

**./new-account/new-account.component.ts**

*import {*LoggingService*} from ‘../logging.service’;*

import { AccountsService } from ‘./accounts.service’;

*@Component({*

*…*

providers*: [*LoggingService*] -> Since, app.component.ts already initiate ‘AccountsService’ instance, it will be provided into its all child components. Injection Hiarachy!. So this component don’t need to say about providers:[AccountService], if it is, it will be overwritten and different service. But still we need import that class.*

*Or, this provider setting can be declared in app.module.ts, like @NgModule({ … providers:[AccountsService]”}). In this case, whole components under this module will receive the this same service.*

*})*

*export class NewAccountComponent {*

*…*

constructor*(*private loggingService: LoggingService, private accountsService: AccountsService*) { }*

*onCreateAccount(accountName: string, accountStatus: string) {*

*this.*accountsService.addAccount(accountName, accountStatus); *this.*loggingService*.logStatusChange(accountStatus);*

*}*

*}*

**./account/account.component.ts**

*import {*LoggingService*} from ‘../logging.service’;*

import { AccountsService } from ‘./accounts.service’;

*@Component({*

*…*

providers*: [*LoggingService*]*

*})*

*export class AccountComponent {*

*@Input() account: {name: string, status: string};*

*@Input() id: number;*

constructor*(*private loggingService: LoggingService, private accountsService: AccountsService*) { }*

*onSetTo(status: string) {*

*this.*accountsService.updateAccount(this.id, status); *this.*loggingService*.logStatusChange(accountStatus);*

*}*

*}*

# Injecting Service into Service

**./accounts.service.ts**

import { Injectable } from ‘@angular/core’;

import {LoggingService } from ‘./logging.service’;

@Injectable

export class AccountService {

…

constructor (private loggingService: LoggingService) {}

addAccount (name:string, status:string) {

…

this.loggingService.logStatusChange(status);

}}

# Service for cross component communication

**./accounts.service.ts**

*import { EventEmitter, Injectable } from ‘@angular/core’;*

*…*

*@Injectable()*

*export class AccountsService {*

*…*

*accounts = [ … ];*

statusUpdated = new EventEmitter<string>();

updateStatus(id: number, status: string) {

this.accounts[id].status = status;

this.loggingService.logStatusChange(status);

}

*}*

**./account/account.component.ts -> from here through account service**

*import { AccountsService } from ‘../accounts.service’;*

*..*

*export class AcccountComponent {*

*counstructor(private accountsService: AccountsService) {}*

*onSetTo(status: string) {*

*this.accountsService.updateStatus(this.id, status);*

*this.accountsService.*statusUpdated.emit(status);

*}*

*}*

**./new-account/new-account.component.ts -> receive emit at here**

*…*

*export class NewAccountComponent {*

*} constructor(private accountsService: AccountsService) {*

*this.accountsService.statusUpdated.*subscribe*(*

*(status: string) => alert(‘New Status: ‘ + status)*

*)*

*}*

# Setting up and loading Routes

*./app.module.ts*

*…*

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

const appRoutes: Routes = [

{ path: '', component: HomeComponent },

{ path: 'users', component: UserComponent },

{ path: 'servers', component: ServersComponent },

];

*@NgModule({*

*…*

*imports: [*

*…*

RouterModule.forRoot(appRoutes)

*],*

*export class AppModule { }*

**./app.component.html**

*…*

*<div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">*

*<ul class="nav nav-tabs">*

*<li role="presentation*" routerLinkActive="active" [routerLinkActiveOptions]="{exact:true}"*>* **-> routerLinkActive=”{classname}”: when router is activated which class will be injected. "[routerLinkActiveOptions]="{exact:true}" prevent misunderstood with “/” from “/server” or “/users”**

*<*a routerLink="/"*>Home</a>* ***-> routerLink prevent reloading***

*</li>*

*<li role="presentation"* routerLinkActive="active"*>*

*<*a routerLink="/servers"*>Servers</a>*

*</li>*

*<li role="presentation*" routerLinkActive="active"*>*

*<a* [routerLink]="['/users']"*>Users</a>*

*</li> </ul>*

*</div>*

*<div class=”row”>*

<router-outlet></router-outlet>***-> This position will be the place where routed component will be inserted.***

*</div>*

**Router Path Style**

routerLink = “/server” -> root/server

routerLink = “server” -> {currentPage}/server

routerLink = “../server” -> {parentPage}/server

# Programmatically Navigate Router

There should be router setting in app.module.ts

**./home/home.component.html**

*<button class="btn btn-primary*" (click)="onLoadServers()*">Load Servers</button>*

**./home/home.component.ts**

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

*@Component({*

*…*

*export class HomeComponent implements OnInit {*

*constructor(*private router: Router*) { }*

*onLoadServers() {*

this.router.navigate(['/servers']);

*}*

*}*

# Programmatically Navigate Router (Relative Path)

There should be router setting in app.module.ts

**./servers/servers.component.html**

*<button class="btn btn-primary" (click)="onReload()">ReLoad Page</button>*

**./servers/servers.coponent.ts**

*import { Router,* ActivatedRoute *} from '@angular/router';*

*export class ServersComponent implements OnInit {*

*constructor(private serversService: ServersService,*

*private router: Router,*

*private* route*:* ActivatedRoute*) { }*

*onReload() {*

*this.router.navigate(['/servers']); -> localhost/servers*

*this.router.navigate(['servers']); -> localhost/servers (Different action with routerlink function)*

*this.router.navigate(['servers'*], { relativeTo: this.route }); *-> localhost/servers/server*

*}*

*}*

# Parameter in Router

**./app.module.ts**

*const appRoutes: Routes = [*

*{ path: '', component: HomeComponent },*

*{ path: 'users', component: UsersComponent },*

*{ path:* 'users/:id/:name'*, component: UserComponent },* ***-> It will make available localhost/users/{id}/{name}***

*{ path: 'servers', component: ServersComponent },*

*];*

**./users/user/user.component.html**

*<p>User with ID {{user.id}}loaded.</p>*

*<p>User name is {{user.name}}</p>*

*<hr>*

*<a [routerLink]="['/users',10,'Anna']">Load Anna (10)</a>* **-> On Same component, Parameters should be subscribed in ts**

**./users/user/user.component.ts**

*import {* ActivatedRoute, Params *} from '@angular/router';*

*…*

*export class UserComponent implements OnInit {*

*user: { id: number, name: string };*

*constructor(private route:* ActivatedRoute*) { }*

*ngOnInit() {*

*this.user = {*

*id:* this.route.snapshot.params['id'],

*name:* this.route.snapshot.params['name'],

*};*

this.route.params.subscribe( **-> This part is for route call in same component such as Anna/10 in next html code (Route Observable)**

(params: Params) => {

*this.user.id = params['id'];*

*this.user.name = params['name'];*

*}*

*);*

*}*

*}*

* This is same with below code. But Angular do this instead of us.

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

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

import { Subscription } from 'rxjs/Subscription';

import { OnDestroy } from '@angular/core/src/metadata/lifecycle\_hooks';

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit, OnDestroy {

user: { id: number, name: string };

paramsSubscription: Subscription;

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.user = {

id: this.route.snapshot.params['id'],

name: this.route.snapshot.params['name'],

};

this.paramsSubscription = this.route.params.subscribe(

(params: Params) => {

this.user.id = params['id'];

this.user.name = params['name'];

}

);

}

ngOnDestroy() {

this.paramsSubscription.unsubscribe();

}

}

# Query String

./servers/servers.comoponet.html

*<a*

[routerLink]="['/servers',5,'edit']"***-> /servers/5/edit***

[queryParams]="{allowEdit:'1'}"***-> /servers/5/edit?allowEdit=1***

fragment="loading"***-> /servers/5/edit?allowEdit=1#loading***

*href="#"*

*class="list-group-item"*

*\*ngFor="let server of servers">*

*{{ server.name }}*

*</a>*

# Query String Programmatically

**./home/home.componet.html**

*<button class="btn btn-primary" (click)="onLoadServer(1)">Load Servers</button>*

**./home/home.component.ts**

*…*

*onLoadServer(id: number) {*

this.router.navigate(['/servers', id, 'edit'], { queryParams: { allowEdit: '1' }, fragment: 'ladong' });

*} ->* ***/servers/1/edit?allowEdit=1#ladong***

**/servers/edit-server/edit-server.componet.ts**

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

*export class EditServerComponent implements OnInit {*

*constructor(private serversService: ServersService,*

*private route: ActivatedRoute) { }*

*ngOnInit() {*

console.log(this.route.snapshot.queryParams); -> /servers/1/edit?**allowEdit=1**#loading console.log(this.route.snapshot.fragment); -> -> /servers/1/edit?allowEdit=1#**loading**

*this.server = this.serversService.getServer(1);*

*this.serverName = this.server.name;*

*this.serverStatus = this.server.status;*

*}*

# Get Query String in component

**./app.module.ts**

*..*

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

*const appRoutes: Routes = [*

*{ path: '', component: HomeComponent },*

*{ path: 'users', component: UsersComponent },*

*{ path: 'users/:id/:name', component: UserComponent },*

*{ path: 'servers', component: ServersComponent },*

{ path: 'servers/:id', component: ServerComponent },

*{ path: 'servers/:id/edit', component: EditServerComponent },*

*];*

*..*

**./servers/servers.component.html**

*<a*

[routerLink]="['/servers',server.id]"

*[queryParams]="{allowEdit:'1'}"*

*fragment="loading"*

*href="#"*

*class="list-group-item"*

*\*ngFor="let server of servers">*

*{{ server.name }}*

*</a>*

**./servers/server/server.component.ts**

*import { ActivatedRoute, Params } from '@angular/router';*

*…*

*export class ServerComponent implements OnInit {*

*server: { id: number, name: string, status: string };*

*constructor(private serversService: ServersService,*

private route: ActivatedRoute*) { }*

*ngOnInit() {*

*const id =* +this.route.snapshot.params['id']; -**> + sign converts to number type, since all query string is string type**

*this.server = this.serversService.getServer(id);*

this.route.params.subscribe((params: Params) => {

this.server = this.serversService.getServer(+params['id']);

});

*}*

# Nested Router

**./app.module.ts**

*const appRoutes: Routes = [*

*{ path: '', component: HomeComponent },*

*{ path: 'users', component: UsersComponent },*

*{ path: 'users/:id/:name', component: UserComponent },*

*{ path: 'servers', component: ServersComponent },*

*{ path: 'servers/:id', component: ServerComponent },*

*{ path: 'servers/:id/edit', component: EditServerComponent },*

*];*

* Changed to

*const appRoutes: Routes = [*

*{ path: '', component: HomeComponent },*

*{*

*path: 'users', component: UsersComponent,* children*: [*

*{ path: '*:id/:name'*, component: UserComponent },*

*]*

*},*

*{*

*path: 'servers', component: ServersComponent,* children*: [*

*{ path: '*:id'*, component: ServerComponent },*

*{ path: '*:id/edit'*, component: EditServerComponent },*

*]*

*},*

*];*

**./app.component.htm**

*<router-outlet></router-outlet>* **-> Parent Router position (/server or /users)**

**./users/users.componet.html**

**./servers/servers.component.html**

*<router-outlet></router-outlet>* **-> Children Router position (/server/{id} or /users/{id}/{name})**

# Router Parameter Practice

**./servers/server/server.component.html**

*<button class="btn btn-primary" (click)="*onEdit*()"></button>*

**./servers/server/server.component.ts**

*import { ActivatedRoute, Params, Router } from '@angular/router';*

*…*

*onEdit() {*

this.router.navigate(['edit'], { relativeTo: this.route }); *->* **this is better than below, since it is in same component, but this case router will lost parameter**

*this.router.navigate(['/servers', this.server.id, 'edit']); -> verbose*

this.router.navigate(['edit'], { relativeTo: this.route, queryParamsHandling: 'preserve' }); **-> This will preserve parameter when we move to the next link.**

*}*

**./servers/servers.component.html**

<a [routerLink]="['/servers',server.id]" [queryParams]="{allowEdit: server.id===3?'1':'0'}" fragment="loading" href="#" class="list-group-item"

\*ngFor="let server of servers">

{{ server.name }}

</a>

**./servers/edit-server/edit-server.component.ts**

*…*

*export class EditServerComponent implements OnInit {*

*allowEdit = false;*

*…*

*ngOnInit() {*

*console.log(this.route.snapshot.queryParams);*

*console.log(this.route.snapshot.fragment);*

this.route.queryParams.subscribe(

(queryParams: Params) => {

this.allowEdit = queryParams['allowEdit'] === '1'?true:false;

}

*);*

*this.route.fragment.subscribe();*

*this.server = this.serversService.getServer(1);*

*this.serverName = this.server.name;*

*this.serverStatus = this.server.status;*

*}*

**./servers/edit-server/edit-server.component.html**

*<h4* \*ngIf="!allowEdit">*You're not allowed to edit!</h4>*

*<div* \*ngIf="allowEdit">

*Something for Edit*

*</div>*