To create new component: ng g c [name]

To create bootstrap by cli: npm install –save bootstrap

To create new directive ng g d name

Understanding DataBinding: It is Communication between TypeScriptCode and Template. Implementation: From TypeScript to HTML : String Interpolation({{data}}) Property Binding ([property] = “data”) include it inside square bracket, indidate to angular when you use property binding, ex:

[[disable]] = “abcb”

From HTML To TypeScript: Event Binding ((event) = “expression”).

When you click button, you trigger something to TypeScript code.

But we have two-way-binding ([( ngModel)] = “data”)

String Interpolation: I don’t want to hardcode to HTML file

Property Binding: [proterty] = “abc..xyz”

Event binding: (click)=””

YOUR\_ELEMENT properties  or YOUR\_ELEMENT events

$event is super important

(<HTMLInputElement>event.target).value

Get value of inpur with event: Event

Data two way Binding: combine property with event binding, we have special directive ngModel

ngModel: To using ngModel we need add formsModule form @angular/forms to your import in App Module(Form Module should be there default by CLI project).

Very simple , binding both , reacting binding both

Important: 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.

Summary: different forms of Databinding:

String interpolation, Property Binding, Event Binding, 2 way binding

Directives are instructions in the DOM, Component are also instructions in the DOM

With structure directive: we must use \*before it \*ngif

To Use else of if: we use ;else nameOfElse and combine with ng-template

Ex \*ngif=”abcd.xys ; else ki”

<ng-template #ki>…</ng-template>

Structure directive: ngif can add or remove element

Attribute Directive: ngStyle don’t add or remove element

Constructor: run when component created

ngStyle: need some configuration to do something, must use property binding on the directives. Super important to understand the web package here (ngoac vuong) is not a part of directive style, it used to bind some property in directive. Property binding is not same as directive. We are binding two property in directive. ngStyle contain array object have key and value

ngStyle allow dynamic update (big advantage) style ex:

[ngStyle]=”{backgroundColor: getColor()}” that is reason to use property binding to configure because getColor dynamic update

let i = index to get Index in ngFor

Add style or bootstrap in .angular-cli.json. We add boostrap and style to global style so it effect to whole of project. Ex: ../nodes\_module/dnmasmnd why we must use .. at level 1 because we add style here to index.html but index.html inside src folder so we must up 1 level

.className and Tab, help you generate div with className in bootstrap

import { HeaderComponent } from './header/header.component';

import { HeaderComponent } from './header/header.component';

HeaderComponent

HeaderComponent

Create component don’t have spec file

Ng c g abc –spec false

**Bootstrap:**

list-group-item clearfix

**to use for list**

pull-left

**float-left**

list-group-item-heading

**this is heading of list**

list-group-item-text

**this is content of list**

pull-right

**float: right**

img-responsive

**to bootstrap resize the image**

**Typescript offer a shortcut to create properties in class**

**Constructor(public name:string, public amount:number) {}**

**Angular augury also used to debug**

**To solve problem in Angular:**

**1: Read the error first**

**2: use sourcemap to debug**

**3: use augury**

**Split login in AppComponent to couple small component: can’t put a lot of login in a component**

**But issue are we must pass data between components**

Following introduce to Binding to Custom Properties and Binding to Custom Events

Custom Properties: from outside want to bind to local

Use @Input() to exposing a property from a component to outside for binding property

Ex: AppComponent: appcomponent.html

<app-server-element \*ngFor=”let serverElement of serverElements” [propertyNameAtExposing]=”serverElement”></app-server-element>

We want to binding properties(element property) of app-server-element to bind value from serverElement to element properties. But It will error when you are at outside component html file and binding to server-element.ts(regularly must be from server-element.html)

So that to solve this problem we use @Input(‘propertyNameAtExposing’) for element property

This above way called Alias(bi danh) if you don’t want you can use default: @Input()

Custom Events: from local want to bind to outside(inform to parent component, app component contain cockpit component)

Angular make css file only for component that it belong to

Angular make css for tag in every component are encapsulation css with same tag but different attribute:

p at component1: p \_ngcontent-ejo-1

p at component2: p \_ngcontent-ejo-2

**This is default behavior of view encapsulation in angular: emulates(key word)**

every component emulates the shadow dom, it not supported by all browser, each element has its kind of own shadow dom behind it where you could assign styles to each element.

These are 3 types of EncapsulationView:

1. EncapsulationView.None: remove shadow dom technology , ex: only have p tag.
2. EncapsulationView.Emulate: it use shadow dom technology, it it only effect for own component: ex contain p \_ngcontent-ejo-1
3. EncapsulationView.Native: it find use define css only in css file of this component, not effect to boostrap

**Local Reference in Templates:**

Remove two-way binding by local reference on this template: #serverNameInput(can appy for any html element not only input), we can use it everywhere in our template, not in typescript code(only in template). Local Reference it will bring whole the element of html

When we use it, we must passing it to method or related thing to can use at TypeScript4

**Ps: Getting access to the Template and Dom with localReference**

**Getting access to the template and dom with @ViewChild**

@ViewChild(‘argument’): View Child need to pass argument here, this argument is actually as the selector of element. How we want to selector element, not like selector in css

Argument: name of local reference: (**ElementRef**)

Conclusion: **2 types of binding Local Reference : using method or use @ViewChild**

**Inportant Thing: Don’t change the value of local reference in TypeScript ex:**

**This.serverContent.nativeElement.value = ‘ABC’; You can not change value Dom from TypeScript by this way**

**Projecting Content into components with ng-content:**

Sometimes you have complex block of code, you want to pass into a component from outside, you don’t want this block of code in inside. This block of code was moved to outside between the opening and closing element of your component at outside ex: <app-server-component></app-server-component>

Everything you place between the opening and closing tag of your own component is lost by default, it removed from dom and angular don’t take care about it. You must use special directive at where you move the code from components to outside

Ex:

<div>

<ng-content></ng-content>

</div>

<app-server-component>

<div>

Kjdsjkadjksajk,asdkjsakjdkjas,dsakdklsalk

</div>

</app-server-component>

**Kjdsjkadjksajk,asdkjsakjdkjas,dsakdklsalk is projected in your component**

Which more complex HTML Code property binding really isn’t best solution(custom property binding

Ex: \*ngFor=”let serverElement of serverElements” [srvElement]=”serverElement”)

Disposal = removal

**Component Life Cycle**

ngOnChanges: Called after a bound input property changes

ex: @Input apply it from outside and its value changed

It’s executed at the start when a new component is created but after it also always called whenever one of our bound input properties changes And with I mean properties decorate with ADD input. So whenever these properties received new values

We can’t use the element as an object because objects are references types so we passed the object via input (@Input(‘srvElement’) element: {}) therefore both properties in the server element component and the object in the array of server Elements here in the component they both point to same place and memory And therefore if we changed the name there it will update in the child component in the server element but it will not trigger NG changes because technique the property were binding here with an input that didn’t change the value of that didn’t change. **It still same object memory. It is difference between reference and primitive type.**

**To effect ngChange we will bind a primitive with values changed which just a string therefore ngOnChanges should get fired again**

ngOnInit: Called once the component is initialized

This method gets executed once the component has beean initialized.( The object was created). It would be run after constructor.

ngDoCheck: this method will be executed a lot because this will run whenever change detection runs

A lot of trigger for ngDoCheck. So that is not something where you want to run amazingly powerful code because that would cost you a lot of **performance**. Any changes such as click button, do something else do check will trigger.

ngAfterContentInit: Called after content has been projected into view.

ngAfterContentChecked: Called every time the projected content has bean checked

ngAfterViewInit: Called after the component’sview has been initialized

ngAfterViewChecked: Called everytime the view has been checked.

ngOnDestroy: Called once the component is about to be destroyed

**@ViewChild: if we want access it in lifecycle hook, we must access it at pharse ngAfterViewInit()**

**@ContentChild: We apply local reference for (ngcontent tags) but at component.ts we can use @ViewChild we must use @ContentChild. Why because it not part of the view, it part of content.**

**ngFor and ngIf Recap**

**We can’t have more than one structural directive on the same element**

<li

class="list-group-item"

\*ngFor="let number of numbers"

\*ngIf="number % 2 == 0">

{{number}}

</li>

Above code you can see, we have \*ngFor and \*ngIf are structural directive for one element

**ngClass**

[ngClass]="{odd: odd % 2 !== 0}"

[ngClass]="{even: even % 2 === 0}"

Square brackets here indicate that we are binding to some property on our ngClass directive

**ngStyle**

ngStyle allows us to also pass an object to some property which is also named ngStyle on the same

**Create Basic Attribute Directive**

@Directive({

selector: '[appBasicHighLight]',

})

export class BasicHighLightDirective implements OnInit {

constructor(private elementRef: ElementRef) {

elementRef.nativeElement

}

ngOnInit() {

this.elementRef.nativeElement.style.backgroundColor = 'green';

}

}

But this above way isn’t best practice. Not good way to directly access your element.

Access Elements directly like this is not a good practice

**Create Attribute Directive By Renderer:**

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

@Directive({

selector: '[appBetterHighlight]'

})

export class BetterHighlightDirective implements OnInit {

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

ngOnInit() {

this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'blue');

}

}

This way is better approach to create Attribute Directive.

Angular is not limited to running in the browser, it also works with service workers And These are environments where you might not have access to the dorm. So If you try to change the dorm as you did here in basic highlight by directive accessing the native element and style of this element you might get an error in some circumstances(hoan canh)

<https://angular.io/api/core/Renderer2>

**HostEvent**

To make interactive and I only want to style as blue or give as blue background if I have a over it and If I move my mouse i should go back to transparent

To do that we add decorator, it is host listener decorator which

@HostListener('mouseenter') mouseover(eventData: Event) {

this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'blue');

}

@HostListener('mouseleave') mouseleave(eventData: Event) {

this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'transparent');

}

**Using HostBinding To Bind Host Properties**

Using the renderer is not wrong though the decorator I’m refreshing to is called at host binding

@HostBinding('style.backgroundColor') backgroundColor: string;

@HostListener('mouseenter') mouseover(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'blue');

this.backgroundColor = 'blue';

}

@HostListener('mouseleave') mouseleave(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'transparent');

this.backgroundColor = 'transparent';

}

**HostListener and HostBinding is a great way for working with the element inside of a directive and with HostBinding you can point to any property of the element you are sitting on**

**Binding to Directive Properties:**

Step1:

@Input() defaultColor: string = 'transparent';

@Input() highlightColor: string = 'blue';

@HostListener('mouseenter') mouseover(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'blue');

this.backgroundColor = this.highlightColor;

}

@HostListener('mouseleave') mouseleave(eventData: Event) {

// this.renderer.setStyle(this.elRef.nativeElement, 'backgroundColor', 'transparent');

this.backgroundColor = this.defaultColor;

}

Create two @Input and change defaultColor and highlightColor to this.backgroundColor

Step2:

<p appBetterHighlight [defaultColor]="'yellow'" [highlightColor]="'red'">Style me with better directive!</p>

If we passing down a spring we can remove square bracket and single quotation

<p appBetterHighlight defaultColor="yellow" [highlightColor]="'red'">Style me with better directive!</p>

This step remain have a bug, when the screen first load, background-color is transparent.

Step3:

ngOnInit() {

this.backgroundColor = this.defaultColor;

}

Mus do above code ,set background color = default color that binding to Input

**Behind the scene structural directive:**

don’t have \* because in angular syntax don’t have this one:

<div \*ngIf="!onlyOdd">

<li

class="list-group-item"

[ngClass]="{even: even % 2 === 0}"

[ngStyle]="{backgroundColor: even % 2 !== 0 ? 'yellow' : 'transparent'}"

\*ngFor="let even of evenNumbers">

{{ even }}

</li>

</div>

**And This is Behind the scence of this above code**

<ng-template [ngIf]="!onlyOdd">

<li

class="list-group-item"

[ngClass]="{even: even % 2 === 0}"

[ngStyle]="{backgroundColor: even % 2 !== 0 ? 'yellow' : 'transparent'}"

\*ngFor="let even of evenNumbers">

{{ even }}

</li>

</ng-template>

**Building a structural Directives**

You can write own structural directive

So Whenever some input parameter here changes I want to execute a method and therefore I can implement a set of a set keyword

<div \*appUnless="onlyOdd">

<li

class="list-group-item"

[ngClass]="{even: even % 2 === 0}"

[ngStyle]="{backgroundColor: even % 2 !== 0 ? 'yellow' : 'transparent'}"

\*ngFor="let even of evenNumbers">

{{ even }}

</li>

</div>

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

if (!condition) {

this.vcRef.createEmbeddedView(this.templateRef);

} else {

this.vcRef.clear();

}

} // why because in the end we will use property binding with square brackets.

//turn styles into a method

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

}

Explain above code:

* Condition is value from \*appUnless
* This.viewcontainerRef.create... To create ng template it is embedded template
* This,viewcontainerRef.clear to clear it it condition is not true
* Why we use @Input here because behind the scene of \*ng structural directive is property binding

**Familiar with ngSwitch**

<div [ngSwitch]="value">

<p \*ngSwitchCase="10">10<p>

<p \*ngSwitchCase="5">5<p>

<p \*ngSwitchCase="3">3<p>

<p \*ngSwitchDefault>This is default<p>

</div>

value = 10;

**Building and Using Dropdown Directive**

@HostBinding('class.open') isOpen: boolean = false;

constructor() {}

@HostListener('click') toggleOpen(eventData:Event) {

this.isOpen = !this.isOpen;

}

**To toggle class .open only set true or false for this one**

**Create A Service**

To get access to our service don’t create new service and use code in service:

const service = new LoggingService();

service.logStatusChange(accountStatus);

**Angular Offer some greater tool which will give up access to our service. It is angular Dependency Injector**

**What is dependency injector?**

A component depend on another service

**Inject A Service to A Component**

To Inject A service to component, we must remember add provider for this service

Have a code here

ngOnInit() {

this.accounts = this.accountsService.accounts;

}

**Accounts in accountsService is reference Type so when assign this.accounts = …., it means we references current accounts to accounts in accountsService**

**One instance of Service, remove service in provider of Child Component**

**But also have a problems a service can be inject to some component and angular will create the instance of the service. But it wrong if create new instance of service in every component which it injected. So we must assume that the service injected to components have only one instance to assume that data is make sense.**

**Many instance of Service, provide Service in provider in every child Component**

**Hierarchical Injector**

The angular Dependency Injector actually is a hierarchical injector that means if we provide a service in some place of our app, let’s say on one component the angular framework knows how to create the instance of that service for this component and import all its child component.

And Actually this component and all its child components and the child components of the child components will receive the same instance of service

There are others places where we provide a service to the highest possible level

APP MODULE: Same Instance of Service is available Application-wide, the same instance of the class of the service is available in our whole app in all components in all

APPCOMPONENT: same instance for all its child component

ANY OTHER COMPONENT: If we provide a service here at this component , it will have its own instance of the service and this instance will only be available for this component and this will actually even override if we were provide the same service on a higher level And What exactly we want to doing.

**With service, we can provide some event which we can trigger in one component and listened to in another**

statusUpdated = new EventEmitter<String>();

this.accountsService.statusUpdated.emit(status);

**Use @Injectable at service that inside it you can to inject another service**

@Injectable()

export class AccountsService {

constructor(private loggingService: LoggingService) {}

**Cross component communitation**

Account Component:

this.accountsService.statusUpdated.emit(status);

New Account Component:

constructor(private loggingService: LoggingService,

private accountsService: AccountsService) {

this.accountsService.statusUpdated.subscribe(

(status: string) => alert('New status: ' + status)

);

}

**So now we are communicating between components for a service which really can save a lot of time and with that I hope you saw a lot reasons why services can be very helpful and how you can use them**

**Recipe Services(own projects)**

Recipe services manage our recipe list

export class RecipeService {

private recipes: Recipe[] = [

new Recipe('A Test Recipe 1', 'This is simply a test 1',

'http://maxpixel.freegreatpicture.com/static/photo/1x/Food-Kitchen-Meals-Home-Made-Dishes-Recipe-Bio-1175493.jpg'),

new Recipe('A Test Recipe 2', 'This is simply a test 2',

'https://cdn.pixabay.com/photo/2016/06/15/19/09/food-1459693\_960\_720.jpg')

];

getRecipes() {

return this.recipes.slice();

}

}

Make recipes as private so can’t access this recipes from outside and use method getRecipes() to get it.But inside this method use recipes.slice() to copy this recipes to new array.

**When we use service to emit data from children from parent, we also must subscribe that event.**

**Have one problem with services:**

onAddItem() {

const newIngredient = new Ingredient(this.nameInputRef.nativeElement.value,

this.amountInputRef.nativeElement.value);

this.shoppingListService.addIngredient(newIngredient);

}

**Services have**

addIngredient(ingredient: Ingredient) {

this.ingredients.push(ingredient);

}

getIngredients() {

return this.ingredients.slice();

}

**And in Shopping-list component**

ngOnInit() {

this.ingredients = this.shoppingListService.getIngredients();

}

Make a reference to ingredients array

**But have a problem when add new ingredient to arrays it not effect to the ingredients because you can see above code we have this.ingredients.slice(). So when we add , the new ingredient added to original and not added to copy**

We have a couple of ways to solve this problem: the easiest way is remove slice() but it is bad way.

We will simply add a new event emitter in service, so when ingredients change you can emit it

ingredientChanged = new EventEmitter<Ingredient[]>();

addIngredient(ingredient: Ingredient) {

this.ingredients.push(ingredient);

this.ingredientChanged.emit(this.ingredients.slide()); //Important to emit new change to shopping-list component, but we continue copy to new ingredients

}

In Shopping List Component:

ngOnInit() {

this.ingredients = this.shoppingListService.getIngredients();

this.shoppingListService.ingredientChanged.subscribe(

(ingredients : Ingredient[]) => {this.ingredients = ingredients}

);

}

**New feature of ES6 to push array to an array**

this.ingredients.push(...ingredients);

**Angular Router(Navigation but not refresh per every navigate)**

In our app, we got three sections:

* Home
* Servers
  + - View and Edit Servers
    - A Service is used to load and update Servers
* Users
  + - View Users

This app will be improved by adding routing but definitely feel free to play around with it - besides routing, everything should be working fine.

<https://angular.io/guide/router>

We must declare Router in App Module to apply for whole application

const appRoutes : Routes = [

{path: '', component: HomeComponent},

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

{path: 'servers', component: ServersComponent}

];

And then register for this Router

imports: [

BrowserModule,

FormsModule,

HttpModule,

RouterModule.forRoot(appRoutes)

],

And then use directive <router-outlet></router-outlet> at the place where we want to use router

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

<router-outlet></router-outlet>

</div>

* Reloading problem in router:
* <li role="presentation" class="active"><a href="/">Home</a></li>
* <li role="presentation"><a href="/servers">Servers</a></li>
* <li role="presentation"><a href="/users">Users</a></li>

Try to link to every component in router. But when we click it will reload in url. We don’t want to do that, don’t want to refresh the app when links clicked. But it is natural behavior because with every link we click and you request gets sent to the server and it returns a new page

**That however is not the best behavior because it means it restarts our application on every navigation.**

**Solve problem about refresh every navigation By Using Router Link(Navigation with Router Link)**

Instead of using href attribute we use routerLink. Now this will tell to angular that this element on which router link is placed here

* <li role="presentation"><a routerLink="/servers">Servers</a></li>

And another way to use routerLink are property Binding:

* <li role="presentation"><a [routerLink]="’/servers’">Servers</a></li>
* <li role="presentation"><a [routerLink]="[‘/servers’]">Servers</a></li>

RouterLink catches the click event on the element to prevent the default event(click event) which would be to send the request and instead of it analyzes what we passes to router link directive.

It give us better experience because don’t restart the application therefore it keep application state and it’s much faster than reloading the the page all the time

**Understand Navigation path**

<https://www.udemy.com/the-complete-guide-to-angular-2/learn/v4/t/lecture/6656286?start=0>

<a routerLink="./servers">reloaded page</a>

Current Path append servers

**<a routerLink="servers">reloaded page</a>**

Current Path append servers

<a routerLink="../servers">reloaded page</a>

Back one level and append servers

**Using routerLinkActive**

Use this one to css for navigation

<li role="presentation" routerLinkActive="active"><a routerLink="/">Home</a></li>

<li role="presentation" routerLinkActive="active"><a routerLink="/servers">Servers</a></li>

<li role="presentation" routerLinkActive="active"><a [routerLink]="['/users']">Users</a></li>

But have a problem with **routerLinkActive**="active" because routerLinkActive analysis the path of routerLink but at every router link always have “/”, that is reason to make for routerLinkActive at Home always active. It didn’t work correctly

To solve above problem, we can add some configutation to this router link active directive it is:

routerLinkActiveOptions

this directive need some propertybinding because we don’t just pass a string there, we pass a javascript object and that would not work without having our square brackets. So with square brackets we can pass anything which will be resolved dynamically like this javascript

routerLinkActiveOptions="{exact: true}"

>

<a routerLink="/">Home</a>

</li>

So everything is just slash ex: localhost:4200/ and not if it is only part of the path.

**Navigation Programmatically**

You can use route in your code to navigate.

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

constructor(private router: Router) { }

onLoadServers() {

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

}

**Relative path in Programmatic Navigation**

Unlike the router link the navigate method doesn’t know on which route you are currently on

onReload() {

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

}

Above code the navigate method don’t know which current router, so when you are in servers and you click button , it wouldn’t throw exception because it isn’t **localhost:4200/servers/servers**

To get current Router, we must use Activated Router

constructor(private serversService: ServersService,

private router: Router,

private route: ActivatedRoute) { }

onReload() {

this.router.navigate(['/servers'], {relativeTo: this.route});

}

**Pass Parameters to Routes**

{path: 'users/:id', component: UsersComponent},

Id is parameter

**Fetching Router Parameter**

ngOnInit() {

this.user = {

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

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

};

}

To fetching Router parameter id and name

**Fetching Router Parameters Reactively**

For Example:

At user component, we add

<a [routerLink]="['/users',10, 'Max']">ReLoad User (10)</a>

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

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

ngOnInit() {

this.user = {

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

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

};

}

But above code when we change the parameters it will load again component and update new information base on parameters which we put on URL

But when we add a routerLink with same paremeters and click it doesn’t effect. It is not a bug. This is default behavior and What is happening here?. We load our data by using this snapshot object on the route. Now if we load a new round what happens angular has a look at our app modules finds the route. Hopefully loads the component initializes the component and gives us the data by accessing the snapshot. **Now what is happens if we have been this component before.** We are clicking the link which is on user component, url still change, but we have already on the component which should get loaded. So angular doesn’t really instantiate this component again. That’s right.

Why we must render again component. We already are on or you might say because the data we want to load changed( the paremeters) but angular doesn’t known and you can say it’s good that by default I would create whole component and destroy old one if we already are on that component

The changes at here is subsequence changes we need a difference approach, we use params as a observable which allow you to easily work with asynchronous task and **This is an asynchronous task because the paremeters of your currently loaded route might change at some point in the future if user clicks in this link. But you don’t know when you didn’t know if and you don’t know how long it will take. So therefore you can’t block your code and wait for this to happen here because it might never happen.**

**Observable is an easy way to subscribe to some event which might happen in the future to then execute some code when it happens without having to wait for it now**

ngOnInit() {

this.user = {

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

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

};

this.route.params.subscribe(

(params: Params) => {

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

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

}

);

}

This above block of code doesn’t execute when ngOnInit runs, it only execute in the future when having any subsequence change paremeters