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

**An Important Note about Route Observables**

Angular cleans up the subscription you set up here whenever this component is destroyed because if it wouldn’t do it as what are you doing here you’re subscribing to parameter changes and let’s say you then leave this component and later you come back**. Well once you left this component will be destroyed and when you come back a new one will be created but this subscription here will always live in the memory. You must do it by manual**

export class UserComponent implements OnInit, OnDestroy {

paramsSubscription: Subscription;

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

(params: Params) => {

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

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

}

);

ngOnDestroy() {

this.paramsSubscription.unsubscribe();

}

It is very important to understand problem behind the scences

**Passing query parameters and fragments**

For example: localhost:4200/users/1/Max?abc=123&bcd=456

The question is how you can pass it by routerLink? Yes we can do it by routing and programatically approach

* You can pass query parameters and fragment in routerLink:
* <a
* [routerLink]="['/servers', 5, 'edit']"
* [queryParams]="{allowEdit: '1'}"
* [fragment]="'loading'"
* href="#"
* class="list-group-item"
* \*ngFor="let server of servers">
* {{ server.name }}
* </a>

Use [queryParams] and [fragment]

Fragment: to jump to sections in html file

* You can do it programmatically
* <button class="btn btn-primary" (click)="onLoadServers(1)">Load Servers(1)</button>

onLoadServers(id: number) {

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

}

**Retrieve query params and fragment**

Same to fetching param from router parameter. You must subscribe query params and fragment to allow you to react to changed the paremeters

this.route.queryParams.subscribe(

(queryParams) => {console.log(queryParams)}

);

this.route.fragment.subscribe(

(fragment) => {console.log(fragment)}

);

To make sure parameter is spring convert to number we do:

const id = +this.route.snapshot.params['id'];

with icon ‘+’ at beginning of string parameter

**Setting up nested Routes**

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

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

]},

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

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

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

]}

And then at children component we don’t use name of these component such as:

<!-- <app-user></app-user> -->

<router-outlet></router-outlet>

<router-outlet></router-outlet>

<!-- <button class="btn btn-primary" (click)="onReload()">Reload Page</button>

<app-edit-server></app-edit-server>

<hr> -->

<!-- <app-server></app-server> -->

**Instead of writing by this way**

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

**You can write by this way**

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

and let remember this.route is ActivatedRoute

**Use queryParamsHandling:**

At navigate, we get the queryParamsHandling property. This property take a string as a value and this could be merge to merge our old query params with any new params

this.router.navigate(['edit'], {relativeTo: this.route, queryParamsHandling : 'merge'}); //preserve to override old one to new one

🡺 <http://localhost:4200/servers/3/edit?allowEdit=1>

It get allowEdit is old queryParam to merge to new queryParam

**Use RedirectTo Attribute in app Router**

Use redirectTo to redirect to already path

{path: 'not-found', component: PageNotFoundComponent},

{path: 'something', redirectTo: 'not-found'}

If you don’t want to duplicate PageNotFoundComponent, you can redirect path something to not-found

**A more convenient way to catch all routes which are not covered by our application is to use double asterisk, it used to catch all path you don’t know and the order is super important here. Make sure this is very generic route is the last one in your area of routes because your routes read from top to bottom here. Wildcard Routes is \*\***

Important: Redirection Path Matching

Section 11, Lecture 131

In our example, we didn't encounter any issues when we tried to redirect the user. But that's not always the case when adding redirections.

By default, Angular matches paths by prefix. That means, that the following route will match both /recipes  and just /

{ path: '', redirectTo: '/somewhere-else' }

Actually, Angular will give you an error here, because that's a common gotcha: This route will now **ALWAYS** redirect you! Why?

Since the default matching strategy is "prefix" , Angular checks if the path you entered in the URL does **start with the path** specified in the route. Of course every path starts with ''  (Important: That's no whitespace, it's simply "nothing").

To fix this behavior, you need to change the matching strategy to"full" :

{ path: '', redirectTo: '/somewhere-else', pathMatch: 'full' }

Now, you only get redirected, if the full path is ''  (so only if you got NO other content in your path in this example).

For example: we only redirect when we have something

**Localhost:8080/abcd => redirect to localhost:8080/bcde**

**If we have path: ‘’ => localhost:8080 => redirect to abcd => abcd => wrong**

**Outsourcing the Route Configuration**

You can config your application with multiple module. For example: You create app-routing.module.ts with name AppRoutingModule, but it is only outsource module. We need add it again to the main module.

To do that, in the AppRoutingModule we use exports attribute in @NgModule, exports attribute

If you want to add this module to the imports of another module what should be accessible to this module which imports this module And one thing we want to make accessible is our route module

exports: [RouterModule]

**An Introduce to Guard and protecting Routes with can Activate**

Guard this a thing to protect Route

export class AuthGuard implements CanActivate {

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean

| Observable<boolean> | Promise<boolean> {

throw new Error("Method not implemented.");

}

}

It can run both asynchronously returning an observable or a promise or synchoronously, because you might have some guards which execute some code which runs completely on the client. Therefore it runs synchronously or you might have some code which takes a couple of seconds finish because you use a timeout in there or you reach out to a server

Create new Service => AuthService and then a real application just might reach out to a server and allow us to log in or log out and check our current authentication state

**Guard make a route can be access Arbitrary, it help make a protection to Route**

**For Example:**

**Step 1: Create Authenticate Service**

export class AuthService {

loggedIn = false;

isAuthenticated() {

const promise = new Promise(

(resolve, reject) => {

setTimeout(()=> {

resolve(this.loggedIn);

}, 800);

}

);

return promise;

}

login() {

this.loggedIn = true;

}

logout() {

this.loggedIn = false;

}

}

This service have isAuthenticated() method to authenticate LoggedIn Or Not. Inside this method , you create a promise and set Time out for resolve is 800 miliseconds. It means when you click on the Route, It would be promise about 800 miliseconds

**Step 2: Create AuthGuard Service**

@Injectable()

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService,

private router: Router) {

}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean

| Observable<boolean> | Promise<boolean> {

return this.authService.isAuthenticated().then(

(authenticate: boolean) => {

if (authenticate) {

return true;

} else {

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

}

}

);

}

}

This service have a main method is canActivate, this method use authService to authenticate, if true then route work as normally but if false it would be navigate to home page

**Step 3:**

Add canActivate Attribute to Router which we want

{path: 'servers', canActivate : [AuthGuard], component: ServersComponent, children: [

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

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

]},

**Step 4:**

providers: [ServersService, AuthService, AuthGuard],

Add services to whole application

Beside that we also can protect:

**Protecting Child Routes with canActivateChild**

**Step1: Implement Interface CanActivateChild and override method CanActivateChild**

canActivateChild(childRoute: ActivatedRouteSnapshot,

state: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean> {

return this.canActivate(childRoute, state);

}

Call again this method to canActivate

**Step2:**

{path: 'servers',

// canActivate : [AuthGuard],

canActivateChild: [AuthGuard],

component: ServersComponent,

children: [

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

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

]},

Use canActivateChild instead of canActivate to only apply for childrens

**Controlling Navigation with canDeActivate**

We have already study about navigation with router, now I want to focus on to control of where you are allowed to leave router or not

**Let Imagine you are on editing mode of a form, but you want navigate to another router, but not easily to navigate, you must verify that Do you want to router or click out or refresh when you are in editing mode?, you must ask user this one => So this convenience method of keeping the user from accidentally navigating away and How can we implement it?**

**Now let’s make sure that whenever the user tries to accidentally navigate away that we prevent them**

**However a guard always needs to be a service because we need to provide it just like we provided you off guard**

onUpdateServer() {

this.serversService.updateServer(this.server.id, {name: this.serverName, status: this.serverStatus});

this.changesSaved = true;

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

}

Above code is you are editing a serverName or serverStatus and then you click button update to update new Information , you can see we have this.changesSaved variable this is variable to inform you updated or not.

export interface CanComponentDeactivate {

canDeactivate: () => Observable<boolean> | Promise<boolean> | boolean;

}

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

// Try To connect this component CanComponentDeactivate with Guard

}

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

canDeactivate()

}

This guard need to implement canDeactivate() method which called by router Once we try to leave a router

export class CanDeactivateGuard implements CanDeactivate<any> {

canDeactivate(component: any, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState?: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean> {

}

// canDeactivate(component: CanComponentDeactivate,

// currentRoute: ActivatedRouteSnapshot,

// currentState: RouterStateSnapshot,

// nextState?: RouterStateSnapshot) :

// Observable<boolean> | Promise<boolean> | boolean {

// return component.canDeactivate();

// }

}

Let analyze this above code. We have a DeactivateGuard with name: CanDeactivateGuard, it implement CanDeactivate<T> with T is a generic we can put any type to that, maybe “any”. But let see, if you put any to that, you will implement a method:

canDeactivate(component: any, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState?: RouterStateSnapshot): boolean | Observable<boolean> | Promise<boolean> {

}

But if you put **CanComponentDeactivate is:**

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {

canDeactivate(component: CanComponentDeactivate,

currentRoute: ActivatedRouteSnapshot,

currentState: RouterStateSnapshot,

nextState?: RouterStateSnapshot) :

Observable<boolean> | Promise<boolean> | boolean {

return component.canDeactivate();

}

}

**Why we do that, because CanComponentDeactive have method canDeactivate(), when you navigate the router will check method canDeactivate of CanDeactivateGuard, but inside CanDeactivateGuard you implement by method canDeactivate of CanComponentDeactivate but this method only is pattern in interface. So It is good when you implement detail logic checking this method at components which you want to apply Deactivated where we allow leave or not**

export class EditServerComponent implements OnInit, CanComponentDeactivate

Implement check logic

canDeactivate() : boolean | Observable<boolean> | Promise<boolean> {

// provide detail check logic when change router to router check

if (!this.allowEdit) {

return true;

}

if ((this.serverName !== this.server.name ||

this.serverStatus !== this.server.status) && !this.changesSaved) {

return confirm("Do you want to discard the changes?");

} else {

return true;

}

}

**Passing Static Data To a Route**

How to get some static data once a route is loaded

Now you might correctly say : Hey we learned how to get data how to get a parameter out of URL then use that But that’s not what I mean.

I mean that some of your routes depend on data they receive either statically each time they are loaded or they will resolve dynamically. May be I show some examples to make this clearer. Let’s start with static data

**Step 1: Create Error Component and change content of html file**

<h4>{{errorMessage}}</h4>

**Step 2 : create errorMessage variable inside ErrorPageComponent**

**Step 3 : declare it inside app-routing module**

// {path: 'not-found', component: PageNotFoundComponent},

{path: 'not-found', component: ErrorPageComponent, data: {message: 'Page not Found!'}},

{path: '\*\*', redirectTo: 'not-found'}

But why we have data message at here. Now we might say If I define it statically here and we will soon fetch it in error page. Why don’t you define it there to begin with error

Example:

errorMessage: string = "Not found page";

But you only use

errorMessage: string;

**Step 4: Fetching data from component By snapshot**

errorMessage: string;

constructor(private route: ActivatedRoute) { }

ngOnInit() {

this.errorMessage = this.route.snapshot.data['message'];

}

**Step 5: If you are in this page and error message change => fetching data by subscribe**

this.route.data.subscribe(

(data: Data) => {this.errorMessage = data['message']}

);

**Summary: You have static data and you want to pass to route**

**Resolve Dynamic Data with the resolve Guard**

The last lecture we learned how to pass static data, now let’s say we have some dynamic data we have to fetch before a route can be displayed or can be rendered

server-resolver.service.ts ServerResolver

We use Resolver This is also a service just like canActivate and canDeActivate which will allow us to run some code before a route render(It mean it would be load data successfully before router render). It will fetch some data you need later

**Step 1 : Create Server Resolver Service file and code some thing**

interface Server {

id: number;

name: string;

status: string;

}

@Injectable()

export class ServerResolver implements Resolve<Server> {

constructor(private serversService: ServersService) {

}

resolve(route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Server | Observable<Server> | Promise<Server> {

// resolve will fetch data at here

// Route Snapshot is enough because ServerComponent it will execute every time

return this.serversService.getServer(+route.params['id']);

}

}

At this above code we inject server Service to ServerResolver because ServerResolver when serverService help to getServer base on the serverId;

**Step 2: Add ServerResolver to app.module.ts**

Because ServerResolver is also a service that a reason we must add it to providers on App.module.ts

**Step 3: Add ServerResolver to app-routing.module.ts**

{path: ':id', component: ServerComponent, resolve: {server: ServerResolver}},

When we router to localhost:8080/servers/id router will use resolve attribute to run some code before routing render, I will run the code in ServerResolver with method

resolve(route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Server | Observable<Server> | Promise<Server> {

// resolve will fetch data at here

// Route Snapshot is enough because ServerComponent it will execute every time

return this.serversService.getServer(+route.params['id']);

}

To load data of server base on serverId

After that it assign to server data, so it have data with key is server and data is Server was got by serverId

**Step 4: Add Resolve to router when render**

ngOnInit() {

this.route.data.subscribe(

(data: Data) => {

console.log(data['server']);

this.server = data['server'];

}

);

resolve: {server: ServerResolver}}, this code is get by data

**Understanding Location Strategies**

Now if we look at our application we get a coupe of routes in these /servers ; /users

And much more, Now here’s one important thing

It works fine here in your local setup but actually this is not something you should take for granted. If you have a route like this your domain slash servers and you are hosting this one on a real server or somewhere in the web this might not work out-of-the-box because their routes URL is always parsed handled by server first. **For example you send a request to a web application built by angular in the front-end and Spring in the back-end. But If we do as the way we are doing, it’s wrong because every request is process by controller, that is a reason it can’t understand router. Your real server will return index.html file and contain your angular application app. Why?**

Because as I mentioned all your URL are parsed by server first So not by angular by the server. Now if you have /servers here . I will look for a /servers on your server on the real server hosting your web app. So you don’t have a router here because you only have one file here it is index.html contain your angular and you want angular take over to parse this route. Server host it don’t know Router. Here’s your 404 error page

**We must use hash sign in our router. And Let’s see how we can enable it**

@NgModule({

// don't need to declaration at here

imports: [

RouterModule.forRoot(appRoutes, {useHash: true})

],

exports: [RouterModule]

})

use useHash, default is false, we convert it to true

<http://localhost:4200/#/servers(hash> mode routing)

You can see the hash is between hosting and router. What this hashtag will do is it informs your web server?

**Hey I only care about the part before hashtag. So all the part after will be ignored by your web server**

Therefore this will run even on your server. By default the part after hashtag can now be parsed by your angular application

**Apply Router to Recipe and shopping list project**

{

path: '', redirectTo: '/recipes', pathMatch: 'full'

},

Repeat again about pathMatch: full

<li routerLinkActive="active"><a routerLink="/recipes">Recipes</a></li>

<li routerLinkActive="active"><a routerLink="/shopping-list">Shopping List</a></li>

Repeat again bout routerLinkActive: it is css change with every router selected

Parts you want to router, all of them you should put in Router Module, and in html file it will replace by <router-outlet> tag

{

path: 'recipes', component: RecipesComponent,

children: [

{path: '', component: RecipeStartComponent},

{path: ':id', component: RecipeDetailComponent}

]

},

Recipes have two path children, it means in recipes.component.html file if it use <router-outlet> => in this outlet will include 2 component RecipeStartComponent and RecipeDetailComponent, but RecipeStartComponent is loaded **when localhost:8080/recipes** and RecipeDetailComponent is loaded **When localhost:8080/recipes/id**

Recipe Details Item :

// const id = +this.route.snapshot.params['id'];

// this.recipe = this.recipeService.getRecipe(id);

this.route.params.subscribe(

(params: Params) => {

this.recipe = this.recipeService.getRecipe(+params['id']);

}

);

Why we don’t need the code above which commented becuase the first time when component ngOnInit we don’t need it show, only show it when we click on item

**A good point with following code:**

{path: ':id', component: RecipeDetailComponent},

{path: 'new', component: RecipeEditComponent},

We create new component and we call localhost:8080/recipes/new. It will show exception. Because As you know I find the path from top to bottom so It map new to **path: ':id',** so It try to load RecipeDetail Component with new => Can’t load => recipe.ingredients => NPE

* To solve that We move new above id =>

{path: 'new', component: RecipeEditComponent},

{path: ':id', component: RecipeDetailComponent},

**Understanding Observables**

You can handle data, handle error or handle completion because there are the three types of data package you can receive. So you can determine what should happen if I receive a new data package what should happen if I receive an error what should happen when the observable eventually complete side note and observable doesn’t have to complete. **Of course you use it to handle asynchronous tasks because all these data sources here, user event are triggered in your code or http request are asynchronous tasks you don’t know when they will happen** **and you don’t know how long that will take.**

**So if you execute your normal application code you don’t want to wait for these events or you don’t want to wait for the completion of HTTP request because that would block your program, block your logic.**

**🡺** therefore we need methods of handling such asynchronous tasks and historically you might have used callbacks or promises And It’s not necessarily bad to use them, observable is just a different approach of handling , different alternative and angular embraces(bao quanh) observables and angular use them a lot and actually observables have one major advantage

this.route.params

.subscribe(

(params: Params) => {

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

}

);

We also could implement an anonymous function which gets executed in the case of an error and one which gets executed if the observable complete:

1. Handle Data
2. Case of error
3. Gets executed if the observables completes
4. ngOnInit() {
5. this.route.params
6. .subscribe(
7. (params: Params) => {
8. this.id = +params['id'];
9. }, () => {
10. }, () => {
12. }
13. );
14. }

**Let start Observable by dive into a simple project**

this.route.params

.subscribe(

(params: Params) => {

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

}

);

Above code Angular doesn’t really render the whole userComponent just because the parameter changed. Therefore it uses observables to still give us a chance to react to update ID and this is what we do here with observable angular offers us here now.

**(params: Params)** this one called callback. Keep in mind this is just an argument passed to the subscribe method here this is anonymous function where we handle the parameters and then we simply extract the ID in this case

**Build and using a first simple observable**

A tons of different ways of creating an observable

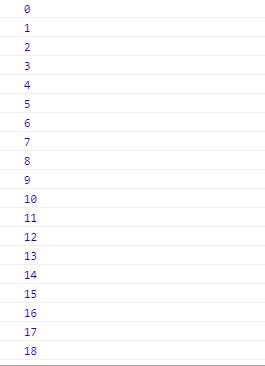
1. Interval is very simple way of creating a new observable here we simply pass a number and that will be the milliseconds it should wait. But when emitting data automatically so it will do the data emitting automatically for us

import { Observable } from "rxjs/Observable";

import 'rxjs/Rx';

const myNumbers = Observable.interval(1000);

I will pass 1000 thousand here to emit a new piece of data every second, need to import ‘rxjs/Rx’



const myNumbers = Observable.interval(1000);

myNumbers.subscribe(

(number: number) => {

console.log(number);

}

);

**Building and Using a Custom Observable from Scratch**

I want to create an observable which will fire off for two seconds and after four seconds and which also fail after five second and alternatively then I want to show how it would work if it completes

const myObservable = Observable.create(

create method take a function as an argument and this function should hold your asynchronous code, and this function actually receives an argument, it is an observer of type observer.

const myObservable = Observable.create(

(observer: Observer<string>) => {

The observer we pass here will be our final observer but it putted to this function which will make up our observable.

But if you build one from scratch you actually need to built this bridge from observable to observe

const myObservable = Observable.create(

(observer: Observer<string>) => {

setTimeout(

() => {

observer.next('first package');

}

,2000);

setTimeout(

() => {

observer.next('second package');

}

,4000);

setTimeout(

() => {

observer.error('this does not work well');

}

,5000);

}

);

**observer.next:** next emits a normal data package

observer.error: show error

observer.complete: to complete observable and therefore it’s done it’s dead, We no longer use it

**UnSubScribe**

We subscribe in this component and then we navigate away to other components but even though your component has been destroyed but subscription remain continue, This can be a problem because of your observable still has an active subscription. Then you’re creating a memory leak because you still have a subscription which you probably not interested in anymore and you still receive the data and handle that data.

* Let make sure you unsubscribe if you leave an area where handle subscribe, Angular don’t do that for you, you must process it by manually
* myNumbersSubscription: Subscription;
* ngOnDestroy(): void {
* this.myNumbersSubscription.unsubscribe();
* }

Make a habit to unsubscribe because it doesn’t clean up automatically

**Where we can learn more about observable**

[**http://reactivex.io/rxjs/**](http://reactivex.io/rxjs/)

**Using Subject to Pass and Listen to Data**

Subject basically is like an observable but it allows you to conveniently push it to emit new data during your code which might a bit more what you’re looking for

A Subject is observable and observer at the same time. This is why I can conveniently call next year and I can pass a value and hearable to simply send

onActivate() {

this.usersService.userActivated.next(this.id);

}

So with that I’m kind of pushing a new data package which contains this ID

It kind of is comparable to the event emitter which ships with angular and actually that event emitter is built on such a subject. It is a good practice to use such a subject instead of event emitter

ngOnInit() {

this.usersService.userActivated.subscribe(

(id: number) => {

if (id === 1) {

this.user1Activated = !this.user1Activated;

} else {

this.user2Activated = !this.user2Activated;

}

}

)

So next time if you want to implement cross component communication consider using a subject instead of event emitter as you saw

**Let me explain by my self:**

**Step1: Create Service with variable create new Subject**

**Step2: In your code, use subject with next method**

You use next method, it is method to push data to data package. If you push data to data package , it also run code inside subscribe by this subject

**Step3: create ngOnInit method and use above subject to subscribe**

When component execute, of course the block of code inside subscribe mustn’t execute. It is execute whenever you push or change data package. In Step 2 when you click button to push id to data package 🡺 that time, code inside subscribe method of subject would be execute. Because It subscribe every action of you, and if you change anything, it will execute.

**In Subject you also use the functions like error, complete and react to them by implementing**

**Understanding Observable Operator**

Map operator: maps simply maps the data you get back into a new observable with any transformation of your choice, map also take a function as an argument, in this function we will get the data and should return the transformed data

const myNumbers = Observable.interval(1000)

.map(

(data: number) => {

return data \* 2;

}

);

Because Observable operator simple return new Observable so that you can of course also chain these operator!