Observales

It is part of our data source .It is a object we import from the rxjs.It follows some pattern. So we have the observable and we have the observer.

An observable can emit event on some action like click event.

Observer : This is your code you can say. It can do 3 different activities :

1. Can handle the data
2. Can handle the error.
3. We can handle the completion of the observable.

In above 3 hooks , our code gets executed.

Note : Http requests are the asynchronous activities. We do not know how much time they will take, or how they will perform.

We can not wait for these tasks or we can say the http events. Because these will block our tasks. We need some ways to handle these tasks. So to handle this situation we may use callback , promises . Now the new approach is observable. Angular in itself uses the observables a lot.

What is the main advantage of using the observables ;  
It is it’s operator that makes it special.

Note :  **We have the observables and the observer with 3 hooks .**

**Handle data , Handle error , Handle completion**

**Creating first simple observable :**

import ‘rxjs/Rx’;

ngOnInit(){

const myNumbers = Observable.interval(1000);

myNumbers.subscribe(

(number : number)=>{

console.log(number)

},

(error)=>console.log(‘The error is ‘+error);

)

**}**

Building a custom observable :

i

ngOnInit() {

const myNumbers = Observable.interval(1000);

/\* myNumbers.subscribe(

(number : number)=>console.log(number),

(error)=>console.log(error)

);\*/

const myObserver = Observable.create(

(observer : Observer<string>)=>{

setTimeout(()=> {

observer.next('First Point');

},1000);

setTimeout(()=>{

observer.next('Second Point');

},2000);

setTimeout(()=>{

observer.error('The error is here')

},3000);

setTimeout(()=>{

observer.complete();

},4000);

setTimeout(

()=>{

observer.next('Third point')

},5000);

});

myObserver.subscribe(

(data : string)=>console.log(data),

(error : string)=>console.log(error),

()=>'completed'

)

**Unsubscribe :**

We can see the the documentation of the observer at : <http://reactivex.io/rxjs/class/es6/MiscJSDoc.js~ObserverDoc.html>

Just type : <http://reactivex.io/rxjs/>

**Subject in observer :**

Operators in Observable :

Go to **reactivex.io/rxjs. Click on the observable .**

What the map operator do. Map simply stores the data from the observable to any observable with the data transformation of our choice.

const myNumbers = Observable.interval(1000).map(

(data : number) => {

return data\*2;

})

**Routing :**

Why we need router ?

We like to have :

localhost:4200/users , localhost:4200/user/2 etc

When we put /users , so we need to load the users component.

Better to write the route configuarion inside the app.module.ts

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

const appRoute : Routes = [

{path : '<blank>',component : HomeComponent},

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

{path : 'servers',component : ServersComponent}

];

Inside the imports :

imports: [

RouterModule.forRoot(appRoutes)

]

Once gets registered :

Inside the app.component.html

<div class="col-lg-12">

<ul class="nav nav-tabs">

<li role="presentation" class="active"><a href="/">Home</a></li>

<li role="presentation"><a href="/users">Users</a></li>

<li role="presentation"><a href="/servers">Servers</a></li>

</ul>

</div>

<div class="col-lg-12">

<router-outlet></router-outlet>

</div>

Now our these urls works :

localhost:4200/users

localhost:4200/servers

localhost:4200/home

**Note : Page is reloading for each url so we need to use something new .**

**routerLink**

**<div class="col-lg-12">**

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

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

**<li role="presentation"><a routerLink="/users">Users</a></li>**

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

**</ul>**

**</div>**

**Styling the link :**

Angular gives a special directive for that :

routerLinkActive=”active”

To create the effect on thr first link

[routerLinkActiveOptions]= “{exact : true}”

**Navigating programmatically**

Load the url on the button click

**For example in the home page we make a button Load server.**

<p>

<button (click)="loadServer()">Load Server</button>

</p>

Import {Router} ‘@angular/router’;

constructor(private router : Router){

//passing the router as an dependency injection

}

loadServer(){

//we need the reference of the route in this function so that we can route our url

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

}

**Relative path in programmatic routing :**

**Passing parameter tko routes :**

In app.component.ts they are we need to check for the appRoutes

{path : 'users/:id',UserComponent}

**Fetching route params :**

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

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

@Component({

selector: 'app-user',

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

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

})

export class UserComponent implements OnInit {

constructor(private route : ActivatedRoute) { }

ngOnInit() {

var id = this.route.snapshot.params['id'];

console.log('The id is '+id)

} }

Thus we got the above id :

ngOnInit() {

var id = this.route.snapshot.params['id'];

this.user = {id : this.route.snapshot.params['id'], name : this.route.snapshot.params['name'] }

}

Name is not displayed as we have not taken the name as data member there.

export class UserComponent implements OnInit {

constructor(private route : ActivatedRoute) { }

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

ngOnInit() {

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

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']

}

);

}

}

Above we are subscribing to the parameters changes :

**Important note about the Route observable :**