**Section08 Observables in Angular**

**Notes: -**

**1-observable is actually an object that we import from third party package RXJS and working through various data sources such as ,**

**User Input Events**

**Http Requests**

**Triggered in code**



**2-RXJS is implement the observable pattern which contains Observable and observer**

**(which contains whenever the button click it will generate event in the data package is emitted automatically or as the angular http server does it)**

**(and whenever the response returns the response is emitted as data package)**

**Observer is using the subscribe function**

**3-types of data packages / hooks cycle**

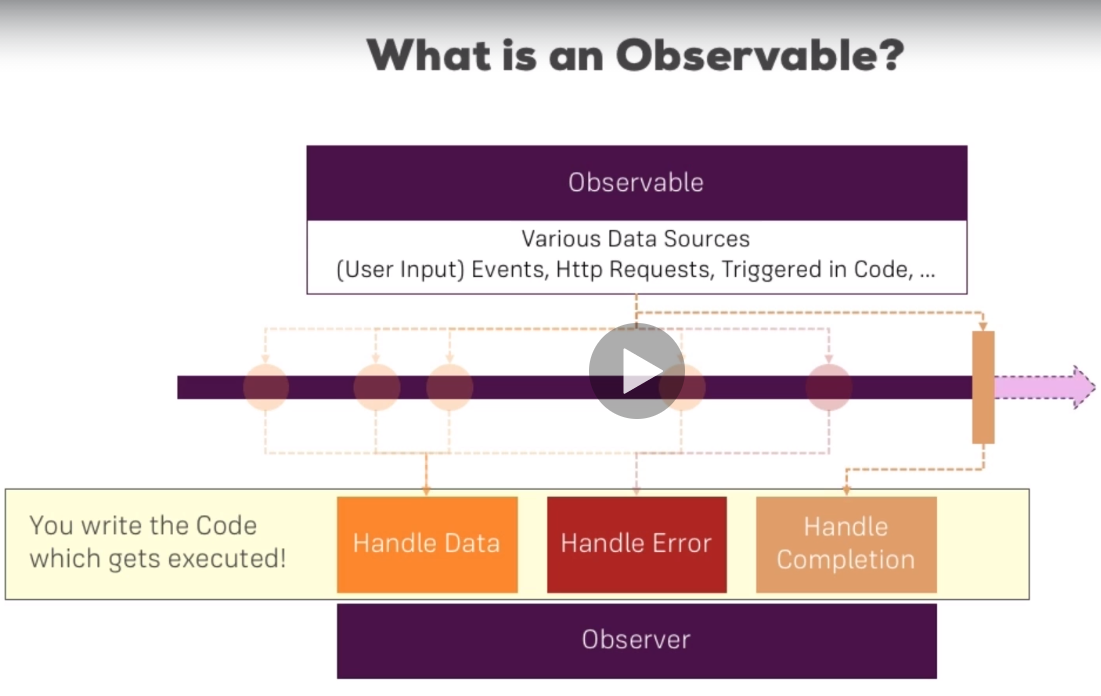
**A-Data data package**

**B-Error data package**

**C-Complete data package**

**(on each data package response, you can determine the operation you want on receive response and when error occur and when complete operation)**

**4-we can handle callback functions and retry operation on the RXJS observables**



**Lesson02 Closing look at observables**

**Notes: -**

**1-to create separate routing file on the target module we type the following command**

**ng g module app-routing --flat --module=app**

**ngOnInit() {**

***//the params called observers which is stream of data that listen to any data / event coming into***

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

**this.id = +params.id;});}**

**2-observable is feature provided by RXJS which provide listen to event coming into through observable and observer ways**

**3-observables does not need to close when navigate away from the component its initialized, you have to unsubscribe with on Destroy event**

**4-when go and return to the same component it will create another observable which execute its own code on another side (and it will cause memory leak)**

**(so subscription is built from observable with send and subscribe events)**

**//you have to register Subscription on the subscribe and on Destroy you have to unsubscribe**

**ngOnInit(): void {**

**interval(1000).subscribe((count) => {console.log(count);})}**

**//fix the issue**

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

**import { interval, Subscription } from 'rxjs';**

**@Component({**

**selector: 'app-home',**

**templateUrl: './home.component.html',**

**styleUrls: ['./home.component.css']})**

**export class HomeComponent implements OnInit , OnDestroy {**

**private firstSub:Subscription;**

**constructor() { }**

**ngOnInit(): void {**

**this.firstSub = interval(1000).subscribe((count) => {**

**console.log(count);})}**

**//it will unsubscribe the subscription so it will kill the subscription once we go out the page**

**ngOnDestroy(): void {**

**this.firstSub.unsubscribe();}}**

**Lesson03 Create Custom Observable**

**Notes: -**

**1-we can apply custom subscription by implement create observable and on each 1 second notify the observable itself that new value received and assign subscribe event to show the new value received as below**

**//there are another commands such as**

**// observer.error();**

**// observer.complete();**

**ngOnInit(): void {**

***// //this is called observable that send***

**const customIntervalObservables = Observable.create(observer => {**

**let count = 0;**

**setInterval(() => {**

***//we notify the observer that there is new data there sending***

**observer.next(count);**

**count++;}, 1000)});**

***//we apply observer that listen to each hit coming into***

**customIntervalObservables.subscribe(data => {**

**console.log(data);});}**

**Lesson04 Errors and Completion**

**Notes: -**

**1-when the error handling hit it will cancel the subscription without execute the complete handling, whereas the complete handling it will end the subscription**

**(an error handling cancel the subscription and let it dies , whereas the complete execute is also finish the subscription also)**

**2-RXJS is merge all the two handlers that assign to the custom observable together**

**3-with Angular observable it applies the event source which apply observable pattern**

**export class HomeComponent implements OnInit, OnDestroy {**

**private firstSub: Subscription;**

**constructor() { }**

**ngOnInit(): void {**

***//this is called observable that send***

**const customIntervalObservables = Observable.create(observer => {**

**let count = 0;**

**setInterval(() => {**

***//we notify the observer that there is new data there sending***

**observer.next(count);**

***//after the complete happen no longer emit values passed through it***

**if (count > 5) {observer.complete();}**

**if (count > 3) {**

***//when hit the error method the observable dies and not fire another hit***

***//you dont need to unsubscribe***

**observer.error(new Error('Count is greater 3!'));}**

**count++;}, 1000)});**

***//we apply observer that listen to each hit coming into***

**this.firstSub = customIntervalObservables.subscribe(data => {**

**console.log(data);},**

***//it will handle the error hit on the observer part***

**error => {**

**console.log(error);**

**alert(error.message);},**

**() => {console.log('Completed!');});}**

**ngOnDestroy(): void {**

**this.firstSub.unsubscribe();}}**

**Lesson05 Operators in Angular**

**Notes:-**

**1-operators is an awesome feature by RXJS which you can filter your data inside the subscription**

**(Operators handle with the data inside the subscription before using it by use pipe)**

Data

Data

Data

Observable

Operators

Subscription

Observer

**2-with using pipe we can use multiple operators like map , filter , before we handle the data where on filter if the condition is true it will move to the map operator which map the data coming into and go to the subscribe handle section otherwise it will block this coming request and looking to the next request**

**Example:-**

**ngOnInit(): void {**

**//this is called observable that send**

**const customIntervalObservables = Observable.create(observer => {**

**let count = 0;**

**setInterval(() => {**

**//we notify the observer that there is new data there sending**

**observer.next(count);**

**//after the complete happen no longer emit values passed through it**

**if (count > 5) {**

**observer.complete();}**

**if (count > 3) {**

**//when hit the error method the observable dies and not fire another hit**

**//you dont need to unsubscribe**

**observer.error(new Error('Count is greater 3!'));}**

**count++;**

**// observer.error();**

**// observer.complete();**

**}, 1000)});**

**//we apply observer that listen to each hit coming into**

**this.firstSub = customIntervalObservables.pipe(**

**filter((data:number) => {return data > 0}),**

**map((data:number) => {**

**return 'Round '+ (data+1);**

**})).subscribe(data => {console.log((data+1));},**

**//it will handle the error hit on the observer part**

**error => {**

**console.log(error);**

**alert(error.message);},**

**() => {console.log('Completed!');});}**

**Lesson06 Subject in RXJS**

**Notes:-**

**1-subject is special type of observable where on inside the observable you call next() inside it , whereas the subject you can call next() outside**

Observable

Subject

Subscription

Subscription

Observer

Observer

**2-with subject its more flexibility than using Event Emitter with more high performance**

**3-you should unsubscribe the subject on the component level**

**4-you should use the subject in case there is no @output parameters, use Event Emitter**

**Example:-**

**If we want when click in user component to show / hide element in app component, we use Event emitter on the service level as below**

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

**@Injectable({providedIn: 'root'})**

**export class UserService {**

**constructor() { }**

**activatedEmitter = new EventEmitter<boolean>();}**

**//app.component.html**

**<p \*ngIf="userActivated">Activated!</p>**

**//on the code behind**

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

**import { Subscription } from 'rxjs';**

**import { UserService } from './services/user.service';**

**@Component({**

**selector: 'app-root',**

**templateUrl: './app.component.html',**

**styleUrls: ['./app.component.css']})**

**export class AppComponent implements OnInit,OnDestroy {**

**userActivated:boolean = false;**

**private activatedSub:Subscription;**

**constructor(private userService:UserService) {}**

**ngOnInit() {**

**this.activatedSub = this.userService.activatedEmitter.subscribe(didActivate => {**

**this.userActivated = didActivate;});}**

**ngOnDestroy(): void {**

**this.activatedSub.unsubscribe();}}**

**//user component.html**

**<p>User with <strong>ID {{ id }}</strong> was loaded</p>**

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

**//on the code behind**

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

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

**import { UserService } from 'src/app/services/user.service';**

**@Component({**

**selector: 'app-user',**

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

**styleUrls: ['./user.component.css']})**

**export class UserComponent implements OnInit {**

**id: number;**

**constructor(private route: ActivatedRoute,private userService:UserService) {}**

**ngOnInit() {**

**//the params called observers which is stream of data that listen to any data / event coming into**

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

**this.id = +params.id;});}**

**onActivate(){this.userService.activatedEmitter.emit(true);}}**

**With using Subject**

**//on the user component code behind you have to change emit to next()**

**onActivate(){**

**this.userService.activatedEmitter.next(true);}**

**//change Event Emitter to Subject**

**export class UserService {**

**constructor() { }**

**activatedEmitter = new Subject<boolean>();}**

**//subscribe as usual**

**ngOnInit() {**

**this.activatedSub = this.userService.activatedEmitter.subscribe(didActivate => {**

**this.userActivated = didActivate;});}**