**Section16 NGRX Introduction**

**Notes: -**

**1-NGRX provide state management in bigger app applications**

**(When you have multi component that some of these components going to store data and another fetch these data)**

**(We have to talk with backend services in order to loading the data)**

Persistent State on backend

Backend

Store and fetch data

Training Service

Application state

Lost when application refresh

New Training Component

Current Training Component

Welcome Component

**2-examples of state**

**A-Auth Service > isloading to show or hide loading spinner**

**B-shopping Service > ingredients**

**C-Recipe Service > recipes**

**When the application becomes bigger and more complex you need NGRX**

SignInComponent

Authenticate User

Display spinner without asynx

AuthService

Application state

Lost when application refresh

Training Service

Current Training Component

New Training Component

Welcome Component

**With Rxjs we apply Subject which apply emit on part emit data and subscribe part**

State Changing Event

User Event in UI / APP

Observable

Operators

Update UI

Listener

**(this way works but we apply in tricky way , so when the application grown up , we have to apply NGRX with state management)**

**//on the service part we apply emit data**

**recipesChanged = new Subject<Recipe[]>();**

**addRecipe(recipe: Recipe) {**

**this.recipes.push(recipe);**

**this.recipesChanged.next(this.recipes.slice());}**

**//on the listener part**

**ngOnInit(): void {**

**this.recipeSub = this.recipeService.recipesChanged.subscribe((result: Recipe[]) => {**

**this.recipes = result;});**

**this.recipes = this.recipeService.getRecipes();}**

**Lesson01 What is NGRX**

**Notes: -**

**1-issues with RXJS approach**

**A-state can be updated anywhere**

**B-state is possibly mutable, it will reflect the change on all parts and notify them**

**C-handling side effect , http calls is unclear**

**D-No specific pattern is applied**

**A-Redux Structure**

**1-Redux is state management pattern, is library that help you to apply this pattern on any application**

**2-Components: -**

**A-Store: - its large central store (application state), all the other parts like services, components receive state from the store**

**B-Actions: - it’s used to send new payload / data into the reducer**

**(Its JavaScript object which use identifier (type of action), and payload (data to send))**

**(It will send the data to the Reducer layer)**

**C-reducer: - it’s just function that take the state from the store and take the event and detect the type of it and take the payload and then perform action based on the types**

**(Then it will apply the changes on the store)**

**3-with NGRX it handles send http request through the structure whereas the Redux structure cannot do that**

**Redux structure**

Store

Save reduced state directly

Reducer Combines state

Reducer

Sent to

Receive State

Actions

Dispatch

Receive State

Component

Services

**NGRX structure**

Store

(Application Store)

Different to normal redex

A-Deeply integrated into Angular

Use RXJS

Use TypeScript

Save reduced state directly

Reducer Combines state

Reducer

Sent to

Listen

Side Effect

Do something http

Actions

Receive State

Receive State

Dispatch

Services

Component

**Lesson02 Getting started with reducer**

**Notes: -**

**1-we will install the following package**

**npm install --save @ngrx/store**