NGRX Introduction

**Notes: -**

**1-**NgRx is a framework for building reactive applications in Angular. NgRx provides libraries for:

* Managing global and local state.
* Isolation of side effects to promote a cleaner component architecture.
* Entity collection management.
* Integration with the Angular Router.
* Developer tooling that enhances developer experience when building many different types of applications.

Packages

NgRx packages are divided into a few main categories

State

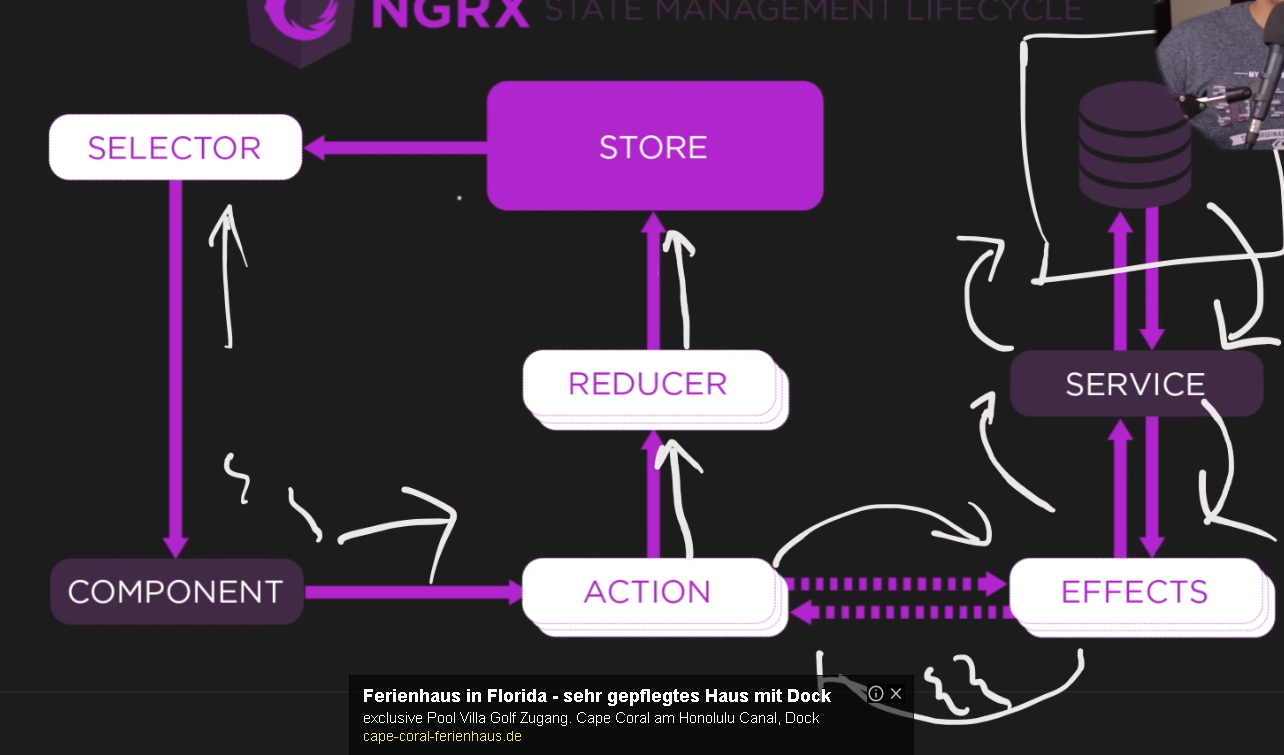
* [Store](https://ngrx.io/guide/store) - RxJS powered global state management for Angular apps, inspired by Redux.
* [Effects](https://ngrx.io/guide/effects) - Side effect model for @ngrx/store.
* [Router Store](https://ngrx.io/guide/router-store) - Bindings to connect the Angular Router to @ngrx/store.
* [Entity](https://ngrx.io/guide/entity) - Entity State adapter for managing record collections.
* [ComponentStore](https://ngrx.io/guide/component-store) - Standalone library for managing local/component state.

Data

* [Data](https://ngrx.io/guide/data) - Extension for simplified entity data management.

View

* [Component](https://ngrx.io/guide/component) - Extension for fully reactive Angular applications.



**Components: -**

**1-action: which is JavaScript object used to pass action type and payload (data) to the reducer and communicate with the effect that is responsible to call services**

**2-effects: which are used to call services and return response to the action**

**3-Reducer: is JavaScript function which is responsible to take action and detect its type and its payload and detect its type and make action to call effect to return response from service and then place the new result in the old state in the store**

**4-store: whish is the global storage that used for all other resources such as component and services**

**5-Selector: which is used to get data from the store and return to the resources caller**

**(The reducer is immutable which means that if you get all movies and then insert another movie you will not see all movie on the received state)**

**Steps: -**

**1-we install the following packages**

**//An in-memory web API for Angular demos and tests that emulates CRUD operations**

npm i angular-in-memory-web-api

npm install --save @angular/material @angular/cdk

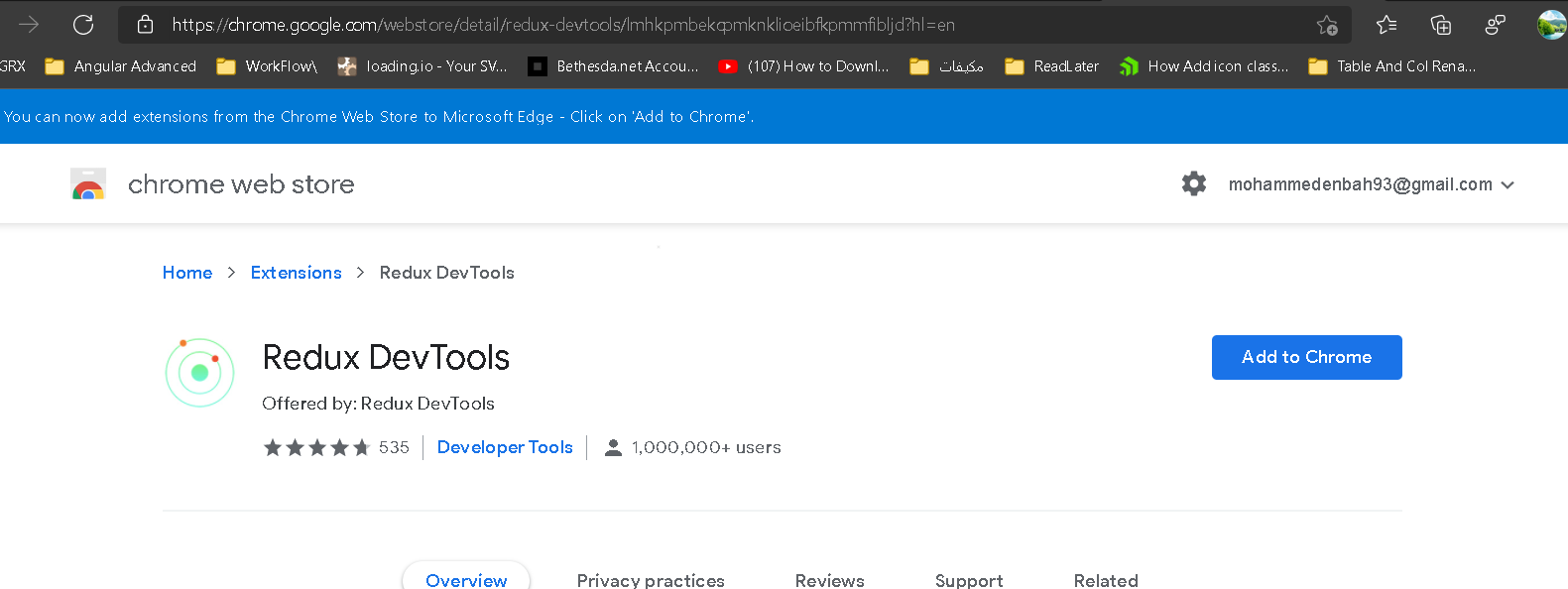
**//the below packages are required for the NGRX , and display NGRX state in dev tools**

**npm i @ngrx/store@11.1.1**

**//install Redux DevTools extension in order to see the redux store on your broser and show on Redux tab on developer tools**

**npm i @ngrx/store-devtools**

**or install extension called Redux DevTools**



2-on the app.module.ts we set the Redux import as below

**import { NgModule } from '@angular/core';**

**import { BrowserModule } from '@angular/platform-browser';**

**import { HttpClientModule } from '@angular/common/http';**

**import { AppRoutingModule } from './app-routing.module';**

**import { AppComponent } from './app.component';**

**import { MovieListComponent } from './movie-list/movie-list.component';**

**import { BrowserAnimationsModule } from '@angular/platform-browser/animations';**

**import { MaterialModules } from './app.material.module';**

**import { HttpClientInMemoryWebApiModule } from 'angular-in-memory-web-api';**

**import { InMemoryService } from './Service/in-memory.service';**

**import { FormsModule } from '@angular/forms';**

**import { StoreModule } from '@ngrx/store';**

**import { environment } from '../environments/environment';**

**import { movieReducer } from './Store/Reducers/movie.reducers';**

**import { StoreDevtoolsModule } from '@ngrx/store-devtools';**

**// import { reducers, metaReducers } from './reducers';**

**@NgModule({**

**declarations: [AppComponent, MovieListComponent],**

**imports: [**

**BrowserModule,**

**AppRoutingModule,**

**BrowserAnimationsModule,**

**FormsModule,**

**MaterialModules,**

**HttpClientModule,**

**HttpClientInMemoryWebApiModule.forRoot(InMemoryService),**

**//dont set store dev tools module before the Store module (the order is important)**

**//it will store reducer on your module**

**StoreModule.forRoot({ movies: movieReducer }),**

**StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: environment.production }),],**

**providers: [],**

**bootstrap: [AppComponent],})**

**export class AppModule {}**

**On the Store > Actions**

**import { createAction, props } from '@ngrx/store';**

**import { Movie } from '../../Models/movie';**

**//we create three actions as below with specify the header name, and parameter we want to pass export const getMovies = createAction('[Movie] Get movie')**

**export const addMovies = createAction('[Movie] Add movie',**

**//the below syntax is to pass the class instace instead of make anonomous object**

**(movie: Movie) => movie**

**// props<{ movie: Movie }>());**

**export const addMoviesSuccess = createAction('[Movie] Add movie success',**

**props<{ movie: Movie }>());**

**On the Store > Reducers**

**import { createReducer, on } from '@ngrx/store';**

**import { Movie } from 'src/app/Models/movie';**

**import { addMovies, getMovies } from '../Actions/movie.action';**

**//to prevent to add to array only to read or replace new array with the old array**

**export interface MovieState {movies: ReadonlyArray<Movie>;}**

**const initialState: ReadonlyArray<Movie> = [];**

**//it will contains two parameters : initial state , action**

**//the reducer will map the action with the function handling**

**export const movieReducer = createReducer(**

**initialState,**

**on(getMovies, (state) => [...mockMovies()]),**

**//cannot apply state.push(movie) its not allowed in NGRX**

**//in the below syntax [...state,movie] it equal to**

**on(addMovies, (state,movie) => [...state,movie]));**

**function mockMovies(): Movie[] {**

**const movie = new Movie('Avengers: Endgame',2798,new Date('26 Apr, 2019 05:30:00'));**

**movie.id = 1;**

**const movie1 = new Movie('Avengers: Infinity War',2048,new Date('27 Apr, 2018 05:30:00'));**

**movie1.id = 2;**

**const movie2 = new Movie('Age of Ultron',1403,new Date('01 May, 2015 05:30:00'));**

**movie2.id = 3;**

**const movies = [movie, movie1, movie2];**

**return movies;}**

on the app.component.ts

**<div style="padding: 20px;">**

**<div style="padding: 20px;">**

**<h2>Create movie</h2>**

**<form class="example-form">**

**<mat-form-field class="example-full-width">**

**<mat-label>Movie name</mat-label>**

**<input name='name' [(ngModel)]="newMovie.name" matInput>**

**</mat-form-field>**

**<mat-form-field class="example-full-width">**

**<mat-label>Total earning</mat-label>**

**<input name='earning' [(ngModel)]="newMovie.earning" type='number' matInput placeholder="In million">**

**</mat-form-field>**

**<mat-form-field appearance="fill">**

**<mat-label>Release date</mat-label>**

**<input name='releaseDate' [(ngModel)]="newMovie.releaseDate"**

**matInput [matDatepicker]="picker">**

**<mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>**

**<mat-datepicker #picker></mat-datepicker>**

**</mat-form-field>**

**<button (click)="addNewMovies()" style="margin-left: 33%;" mat-raised-button color="primary">Add Movie</button>**

**</form>**

**</div>**

**<mat-divider></mat-divider>**

**<div style="padding: 20px;">**

**<app-movie-list [movies]='movies' ></app-movie-list>**

**</div></div>**

**//on code behind**

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

**import { Store } from '@ngrx/store';**

**import { Movie } from './Models/movie';**

**import { DataService } from './Service/data.service';**

**import { addMovies, getMovies } from './Store/Actions/movie.action';**

**@Component({**

**selector: 'app-root',**

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

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

**export class AppComponent implements OnInit {**

**movies: Movie[] = [];**

**newMovie: Movie = new Movie();**

**title = 'movieApp';**

**constructor(private dataService: DataService, private store: Store) {}**

**ngOnInit(): void {}**

**addNewMovies(): void {**

**//we have to call dispatch to call reducer to execute action with pass parameter**

**this.store.dispatch(addMovies(this.newMovie));}}**

on the movie list. component .ts

**<div class="movies">**

**<section \*ngFor="let movie of movies$ | async">**

**<mat-card>**

**<h1> {{movie.name}} </h1>**

**<h2> {{ movie.releaseDate | date }} </h2>**

**<h3> ${{ movie.earning }} million </h3>**

**</mat-card>**

**</section>**

**</div>**

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

**import { Store } from '@ngrx/store';**

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

**import { Movie } from '../Models/movie';**

**import { DataService } from '../Service/data.service';**

**import { getMovies } from '../Store/Actions/movie.action';**

**import { MovieState } from '../Store/Reducers/movie.reducers';**

**@Component({**

**selector: 'app-movie-list',**

**templateUrl: './movie-list.component.html',**

**styleUrls: ['./movie-list.component.css'],})**

**export class MovieListComponent implements OnInit {**

**@Input() movies: Movie[] = [];**

**movies$:Observable<Movie[]>;**

**//this observable readonly array the key passed must match the key in the registerd app.module**

**//such as StoreModule.forRoot({ movies: movieReducer })**

**constructor(private dataService: DataService,private store:Store<MovieState>) {}**

**ngOnInit(): void {this.getAllMovies();}**

**getAllMovies(): void {.**

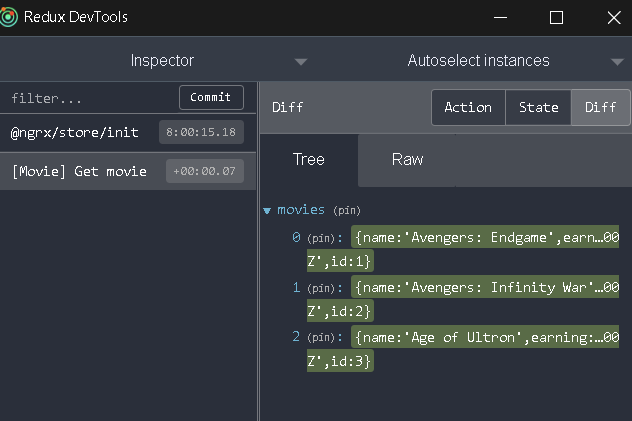
**//we have to apply dispatch to call getMovies that call reducer to call getMovies() and then store //the result on the store managment**

**this.store.dispatch(getMovies());**

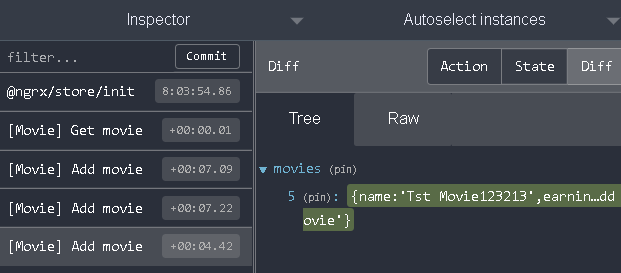
**//we get the state value from the store managment**

**this.movies$ = this.store.select('movies');}}**

We will see that on the left part you see the action name and in the right part you see payload



You will see that you can track your state



**Understanding Effects**

**Notes:-**

**1-Effects is used to isolate between services and the actions NGRX**

**2-we have to install @ngrx/effects as below**

npm install @ngrx/effects

**3-on the app.module.ts we inejct the MovieEffects as below**

**imports: [**

**//dont set store dev tools module before the Store module (the order is important)**

**//it will store reducer on your module**

**StoreModule.forRoot({ movies: movieReducer }),**

**StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: environment.production }),**

**//we inject effects**

**EffectsModule.forRoot([MovieEffects]),],**

4-on actions we declare 2 action one for input and another for handle response data

**import { createAction, props } from '@ngrx/store';**

**import { Movie } from '../../Models/movie';**

**//we have 2 actions ,getMoviesSuccess as input action on the effect and the response return we call getMovicesSuccess Movie[]**

**export const getMovies = createAction('[Movie] Get movie');**

**export const getMoviesSuccess = createAction(**

**'[Movie] Get movie success',**

**// (movies: ReadonlyArray<Movie>) => ({ movies })**

**props<{ movies: ReadonlyArray<Movie> }>());**

**//we have 2 addMovies ,addMoviesSuccess as input action on the effect and the response return we call getMovicesSuccess Movie[]**

**export const addMovies = createAction(**

**'[Movie] Add movie',**

**//the below syntax is to pass the class instace instead of make anonomous object**

**(movie: Movie) => movie**

**// props<{ movie: Movie }>());**

**export const addMoviesSuccess = createAction(**

**'[Movie] Add movie success',**

**// (movie: Movie) => movie**

**props<{ movie: Movie }>());**

5-we modify the reducer to handle the two actions as below

**import { createReducer, on } from '@ngrx/store';**

**import { Movie } from 'src/app/Models/movie';**

**import { addMoviesSuccess, getMoviesSuccess } from '../Actions/movie.action';**

**//to prevent to add to array only to read or replace new array with the old array**

**export interface MovieState {movies: Array<Movie>;}**

**const initialState: ReadonlyArray<Movie> = [];**

**//it will contains two parameters : initial state , actions**

**//the reducer will map the action with the function handling**

**//we assign mulitple actions on the reducer**

**export const movieReducer = createReducer(**

**initialState,**

**on(getMoviesSuccess, (state, { movies }) => [...movies]),**

**//cannot apply state.push(movie) its not allowed in NGRX**

**//in the below syntax [...state,movie] it equal to**

**on(addMoviesSuccess, (state, { movie }) => [...state, movie]));**

**6-on effect we define two effects that take action as parameter and then call data service and the resposne returned it will call another action as below**

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

**import { createEffect, Actions, ofType } from '@ngrx/effects';**

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

**import { catchError, concatMap, exhaustMap, map, tap } from 'rxjs/operators';**

**import { Movie } from 'src/app/Models/movie';**

**import { DataService } from 'src/app/Service/data.service';**

**import {getMovies,getMoviesSuccess,addMovies,addMoviesSuccess,}**

**from '../Actions/movie.action';**

**@Injectable()**

**export class MovieEffects {**

**//in effect we pass the action as input then call the data servcie to return respone and then execute another action**

**//we will create effect in order to call services to return response from it**

**loadMovie$ = createEffect(() =>**

**this.action$.pipe(**

**ofType(getMovies),**

**//source items are ignored while the previous Observable is not completed**

**exhaustMap(() =>**

**this.dataService.getMovies().pipe(**

**map((movies) => getMoviesSuccess({movies})),**

**// catchError(() => EmptyError)))));**

**//in effect we pass the action as input then call the data servcie to return respone and then execute another action**

**//we will create effect in order to call services to return response from it**

**addMovie$ = createEffect(() =>**

**this.action$.pipe(**

**ofType(addMovies),**

**tap((movie) => console.log(movie)),**

**//waits for the previous Observable to complete before creating the next one**

**concatMap((movie:Movie) =>**

**this.dataService.addMovies(movie).pipe(**

**map((newMovie) => addMoviesSuccess({movie: newMovie})),**

**// catchError(() => EmptyError)))));**

**constructor(private action$: Actions, private dataService: DataService) {}}**

**Selector in NGRX**

**Notes:-**

**1-selector is pure function which grahp of slice of data of the store**

**(can grab single slice of data or combine data)**

**2-all the resources grab data from the store through the selector only**

**(we use the selectors to apply client filteration that filter on data store only not going again to the database)**

**3-each state has its own reducer , selector , actions , effect ( interact with server)**

**Steps:-**

**1-**

**import { createSelector } from '@ngrx/store';**

**import { Movie } from 'src/app/Models/movie';**

**import { MovieState } from '../Reducers/movie.reducers';**

**// Selector Configuration //**

**//we define the selector which each one get single state (movies)**

**export const movieSelector = createSelector(**

**(state: MovieState) => state.movies,**

**(movies: ReadonlyArray<Movie>) => movies);**

**//we define the selector which each one get single state (movies)**

**export const userSelctor = createSelector(**

**(state: MovieState) => state.user,**

**(user: string) => user);**

**//Selector Filter (used by the component)//**

**//in this filter we pass the selector we want and we make filter on it**

**export const findByUserName = (username: string) =>**

**createSelector(movieSelector, (movies) => {**

**return movies.filter((movie: Movie) => movie.userName === username);});**

**export const greater = (amount: number) =>**

**createSelector(movieSelector, (movies) => {**

**return movies.filter((movie: Movie) => movie.earning >= amount);});**

**//on user selector**

**export const getName = () =>**

**createSelector(userSelctor, (user) => {**

**return user;});**

**2-on the app.component.ts we register all the data inside state through resolvers**

**getAllMovies(): void {**

**//it will call the reducer to call action that call effect that return response to action then to reducer**

**this.store.dispatch(getMovies());**

**this.store.dispatch(assignUser('Subrat'));}**

**3-on the movie-list.component.ts**

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

**import { select, Store } from '@ngrx/store';**

**import { assignUser, getMovies } from '../Store/Actions/movie.action';**

**import { MovieState } from '../Store/Reducers/movie.reducers';**

**import { findByUserName, getName, greater } from '../Store/selectors/movieSelector';**

**@Component({**

**selector: 'app-movie-list',**

**templateUrl: './movie-list.component.html',**

**styleUrls: ['./movie-list.component.css'],**

**})**

**export class MovieListComponent implements OnInit {**

**//we get data state from the data store**

**// movies$ = this.store.select('movies');**

**user$ = this.store.pipe(select(getName()));**

**movies$ = this.store.pipe(select(findByUserName('Sanjit')));**

**// movies$ = this.store.pipe(select(movieUserSelector));**

**// movies$ = this.store.pipe(select(greater(1000)));**

**constructor(private store: Store<MovieState>) {}**

**ngOnInit(): void {**

**setTimeout(() => {this.movies$ = this.store.pipe(select(greater(2000)));}, 5000);}}**

**4-on the movie-list.component.html we call the two observable variables with async**

**<h1>{{user$ | async}}</h1>**

**<div class="movies">**

**<section \*ngFor="let movie of movies$ | async">**

**<mat-card>**

**<h1> {{movie.name}} </h1>**

**<h2> {{ movie.releaseDate | date }} </h2>**

**<h3> ${{ movie.earning }} million </h3>**

**</mat-card>**

**</section>**

**</div>**

**Note:-**

**In case you don’t want to pass parameter such as username and you want to take it on the selector as below**

**export const movieUserSelector = createSelector(**

**(state: MovieState) => state.movies,**

**(state: MovieState) => state.user,**

**(movies: ReadonlyArray<Movie>, user: Readonly<string>) => {**

**return movies.filter((movie: Movie) => movie.userName === user);});**

**Use of Meta-Reducer in Angular**

**Notes: -**

**1-to apply best way to config your reducers and make tracking for each state and action happened we can use Meta-reducer**

**Advantages: -**

**1-configure all reducers in best approach**

**2-keep tracking for each action and state**

**3-can clean Redux reducers from state in case of apply logout as below**

**Steps: -**

**1-we create class called config.reducer.ts and set the following code as below**

**import {ActionReducer,ActionReducerMap,INIT,MetaReducer,} from '@ngrx/store';**

**import { environment } from 'src/environments/environment';**

**import { logout } from '../../Actions/movie.action';**

**import { movieReducer, MovieState, userReducer } from '../movie.reducers';**

**//we collect all reducer in this const variable and used on the app.module**

**export const reducers: ActionReducerMap<MovieState> = {**

**movies: movieReducer,**

**user: userReducer,};**

**//we can create multiple meta data that used to keep track on each state , action passed from the //component to the reducer as below for development environment and production state**

**const debugMeta = (reducer: ActionReducer<any>): ActionReducer<any> => {**

**return (state, action) => {**

**console.log('state', state);**

**console.log('action', action);**

**return reducer(state, action);};};**

**const logoutMeta = (reducer: ActionReducer<any>): ActionReducer<any> => {**

**return (state, action) => {**

**//if the action type is logout reset the reducer as below**

**if (action?.type === logout.type) {**

**return reducer(undefined, { type: INIT })}**

**return reducer(state, action);};};**

**//this const used on the app.module to switch between meta data based on environment state**

**export const metaReducers: MetaReducer<MovieState>[] = environment.production**

**? [logoutMeta]**

**: [debugMeta, logoutMeta];**

**2-on the actions > we create new action called logout as below**

**export const logout = createAction('[User] logout');**

**3-on the app.component.ts we pass the action as below**

**//we pass the logout in order to clean the reducers through the meta data as below**

**logout(): void {this.store.dispatch(logout());}**

**4-on the app.module.ts we pass the two reducers as below**

**imports: [**

**//we pass the two constants on the app.module as below**

**StoreModule.forRoot(reducers, { metaReducers }),**

**//StoreModule.forRoot({ movies: movieReducer, user: userReducer }),**

**StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: environment.production }),**

**//we inject effects**

**EffectsModule.forRoot([MovieEffects]),]**