**in Redux Approach**

we have **one central store** in your application is one central place where you manage your **application state**

keep in mind with **multiple services** so you go back **to one central place** to really ensure that there is only one place where your state lifts this single **source of truth** your components and services

these two pieces still have access to this store but also to each other.

so they received a state from the store but of course **you can still communicate between components and services, this is not interrupted**

we just change the way we store our state

however, there is another important thing we have a **way to accessing** our store and state but how we dispatched so called actions for dead actions are clearly defined in this approach

you could have action which is save post, this action would next reach a so call reducer.

**Reducers** are functions you write yourselves which **take action and a potential payload as an input and then do somethin**g with it **to manipulate the original state**.

and they do this in the **immutable** way which means **they don't edit the old state** they **overwrite** it with a new state by simply taking the old one.

**editing something** but in **a new javascript object** and then **saving it does** SD read **X set up actions and reducers**

we need to **call reducers to edit the state** remember we need this to store

a reducer function **will be trigger** whenever **an action is dispatched**.

so therefore we get two things

we get the state this is the current state of the application and keep in mind this is passed in automatically

by ngrx and **we get an action the action actually is of type action** which you **can import from at your X store**

and again I will show you how to dispatch action soon

import {Action } from '@ngrx/store';

export function **shoppingListReducer**(**state, action:Action**){}

Now the state is the current state of the application **the first tim**e this runs though there will be **no current state**.

Which is why we can define some initial state or name it initial state here and for my ingredients this will be actually be an object where if we have a look at the shopping service I want to take this array

import \* as **ShoppingListActions** from './shopping-list.actions';

import { **Ingredient** } from '../../shared/ingredient.model';

const **initialState** = {

ingredients: [

new Ingredient('Apples', 5),

new Ingredient('Tomatoes', 10),

]

};

export function **shoppingListReducer(state = initialState,**

**action: ShoppingListActions.ShoppingListActions)** {

**switch (action.type)** {

case ShoppingListActions.ADD\_INGREDIENT:

return {

...state,

ingredients: [...state.ingredients, action.payload]

};

**case ShoppingListActions.ADD\_INGREDIENTS**:

return {

...state,

ingredients: [...**state.ingredients, ...action.payload]**

};

case ShoppingListActions.UPDATE\_INGREDIENT:

const ingredient = state.ingredients[action.payload.index];

const updatedIngredient = {

...ingredient,

...action.payload.ingredient

};

const ingredients = [...state.ingredients];

ingredients[action.payload.index] = updatedIngredient;

return {

...state,

ingredients: ingredients

};

case ShoppingListActions.DELETE\_INGREDIENT:

const oldIngredients = [...state.ingredients];

oldIngredients.splice(action.payload, 1);

return {

...state,

ingredients: oldIngredients

};

default:

return state;

}

}

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

import { Ingredient } from '../../shared/ingredient.model';

export const ADD\_INGREDIENT = 'ADD\_INGREDIENT';

export const ADD\_INGREDIENTS = 'ADD\_INGREDIENTS';

export const UPDATE\_INGREDIENT = 'UPDATE\_INGREDIENT';

export const DELETE\_INGREDIENT = 'DELETE\_INGREDIENT';

export **class AddIngredient implements Action** {

readonly type = ADD\_INGREDIENT;

constructor(public payload: Ingredient) {}

}

export class **AddIngredients implements Action** {

**readonly type = ADD\_INGREDIENTS;**

constructor(public payload: Ingredient[]) {}

}

export class **UpdateIngredient implements Action** {

readonly type = UPDATE\_INGREDIENT;

constructor(public payload: {index: number, ingredient: Ingredient}) {}

}

export class **DeleteIngredient** **implements Action** {

readonly type = DELETE\_INGREDIENT;

constructor(public payload: number) {}

}

export type **ShoppingListActions** =

**AddIngredient |**

**AddIngredients |**

**UpdateIngredient |**

**DeleteIngredient**;

npm install -save @ngrx/store

**import { shoppingListReducer } from './shopping-list/store/shopping-list.reducers';**

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

HttpClientModule,

AppRoutingModule,

SharedModule,

ShoppingListModule,

AuthModule,

CoreModule,

**StoreModule.forRoot({shoppingList: shoppingListReducer})//eagerly load**

],

bootstrap: [AppComponent]

})

export class AppModule { }

@Component({

selector: 'app-shopping-list',

templateUrl: './shopping-list.component.html',

styleUrls: ['./shopping-list.component.css']

})

export class **ShoppingListComponent** implements **OnInit** {

shoppingListState: **Observable<{ingredients: Ingredient[]}>;**

constructor(private **slService: ShoppingListService, private store**: **Store**<{shoppingList: {ingredients: Ingredient[]}}>) { }

ngOnInit() {

**this.shoppingListState = this.store.select('shoppingList');**

}

onEditItem(index: number) {

**this.slService.startedEditing.next(index);**

}

}--------------shopping-list--------------------

<div class="row">

<div class="col-xs-10">

<app-shopping-edit></app-shopping-edit>

<hr>

<ul class="list-group">

<a

class="list-group-item"

style="cursor: pointer"

\*ngFor="let ingredient of (shoppingListState | async).ingredients; let i = index" (click)="onEditItem(i)">

{{ ingredient.name }} ({{ ingredient.amount }})

</a>

</ul>

</div>

</div>-----------------

import {Component,OnInit, OnDestroy,ViewChild

} from '@angular/core';

import { NgForm } from '@angular/forms';

import { Subscription } from 'rxjs/Subscription';

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

import { Ingredient } from '../../shared/ingredient.model';

import { ShoppingListService } from '../shopping-list.service';

import \* as ShoppingListActions from '../store/shopping-list.actions';

@Component({

selector: 'app-shopping-edit',

templateUrl: './shopping-edit.component.html',

styleUrls: ['./shopping-edit.component.css']

})

export class ShoppingEditComponent implements OnInit, OnDestroy {

@ViewChild('f') slForm: NgForm;

subscription: Subscription;

editMode = false;

editedItemIndex: number;

editedItem: Ingredient;

constructor(private slService: ShoppingListService, private store: Store<{shoppingList: {ingredients: Ingredient[]}}>) { }

ngOnInit() {

this.subscription = this.slService.startedEditing

.subscribe(

(index: number) => {

this.editedItemIndex = index;

this.editMode = true;

this.editedItem = this.slService.getIngredient(index);

this.slForm.setValue({

name: this.editedItem.name,

amount: this.editedItem.amount

})

}

);

}

onSubmit(form: NgForm) {

const value = form.value;

const newIngredient = new Ingredient(value.name, value.amount);

if (this.editMode) {

this.store.dispatch(new ShoppingListActions.UpdateIngredient({index: this.editedItemIndex, ingredient: newIngredient}))

} else {

this.store.dispatch(new ShoppingListActions.AddIngredient(newIngredient))

}

this.editMode = false;

form.reset();

}

onClear() {

this.slForm.reset();

this.editMode = false;

}

onDelete() {

this.store.dispatch(new ShoppingListActions.DeleteIngredient(this.editedItemIndex));

this.onClear();

}

**ngOnDestroy**() {

this.**subscription.unsubscribe**();

}

}