NgRx.

A screenshot of a computer

Description automatically generated

A diagram of a company

Description automatically generated

We have two projects , standalone and regular angular app

AddNgRx by using a command

Ng add @ngrx/store

In regularapp

Above command Adds this in app.module.ts

StoreModule.forRoot({})

In standaldone app

In main .ts file

providers: [provideStore({})]

Now we need Reducer as this changes the data in Store even for intial data.

Now create a folder called store in root and add ngrx related files. Create coutner.reducer.ts.

const initialState = 0;

export const counterReducer = createReducer(

initialState,

on(increment , (state, action) => state + action.value)

);

Now to use the reducer in the application we add it in StoreMdoule in app module ts file in key value pairs like below

],

imports: [BrowserModule, StoreModule.forRoot({

counter:counterReducer

})],

And if your company project is using standaldone , then follow below steps

In main.ts file

providers: [provideStore({counter: counterReducer})]

Ch 375:

We deckarea variableas coutn$ with $ at end , this is ngrx convention

count$:Observable<number>

constructor(private store:Store<{counter:number}>){

this.count$ = store.select('counter');

}

We can subsrcribe to observable or use coutn$ with async pipe in html to listen to observable changes

<p class="counter">{{ count$ | async }}</p>

Ch376:Dispatching Actions

Create coutner.actions.ts file in store folder

Actions belonging to feature is put in [] brackets and make it unique

export const increment = createAction(

'[Counter] Increment'

);

export const decrement = createAction(

'[Counter] Decrement'

);

And you listen to actions in reducers like below using on function, here state means old state , below e.g. add action.value

const initialState = 0;

export const counterReducer = createReducer(

initialState,

on(increment , (state) => state + 1)

);

Now remove counter service in counter-controls component tsvand dispatch action from component. Here execute increment by using increment(). WWhen you dispatch and action from component , you listen to in reducer and update state

constructor(private store: Store) {}

increment() {

this.store.dispatch(increment());

//this.store.dispatch(new IncrementAction(2));

}

You can add data to Actions using props function var:type manner

export const increment = createAction(

'[Counter] Increment',

props<{value : number}>()

);

export const decrement = createAction(

'[Counter] Decrement',

props<{value : number}>()

);

TO extract data you use action in reducer like below (yu can use any name like action in here or var , or etc)

export const counterReducer = createReducer(

initialState,

on(increment , (state, action) => state + action.value)

);

And update the call in conutner contorls componetns ts file

increment() {

this.store.dispatch(increment({value: 2}));

//this.store.dispatch(new IncrementAction(2));

}

If your project is sousing older syntax or you don’t like create reducer use below in counter .reducer.ts

//below is same as above but logic added

export function counterReducer(state = initialState , action:CoutnerActions | Action){//Add Action as reducer is triggered for everyaction or any aciton

if(action.type === INCREMENT){

return state + (action as IncrementAction).value;//add brackets to avoid typsecript coimplaints as we know value is in action

}

return state;

}

With below old syntax logic in counter.action .ts

export const INCREMENT = '[Counter] Increment';

export class IncrementAction implements Action{

readonly type = INCREMENT;

constructor(public value: number){// or use payload:{value:number}

}

}

export type CoutnerActions = IncrementAction;

Instead of selecting whole store using below in this case counter

constructor(private store:Store<{counter:number}>

//private counterService: CounterService

) {

this.count$ = store.select('counter');

}

You get whole above total store slice . If You may want to transform state or data in store before you select it. TO demostratate it lets ceraet counter.selector.ts file in store folder

export const selectCount = (state: {counter:number}) => state.counter ;//selector funciton,

//state means overall store .you reutrn state.counter

And in counter-output component ts you can pass selector function instead of passin gocunter key

constructor(private store:Store<{counter:number}>) {

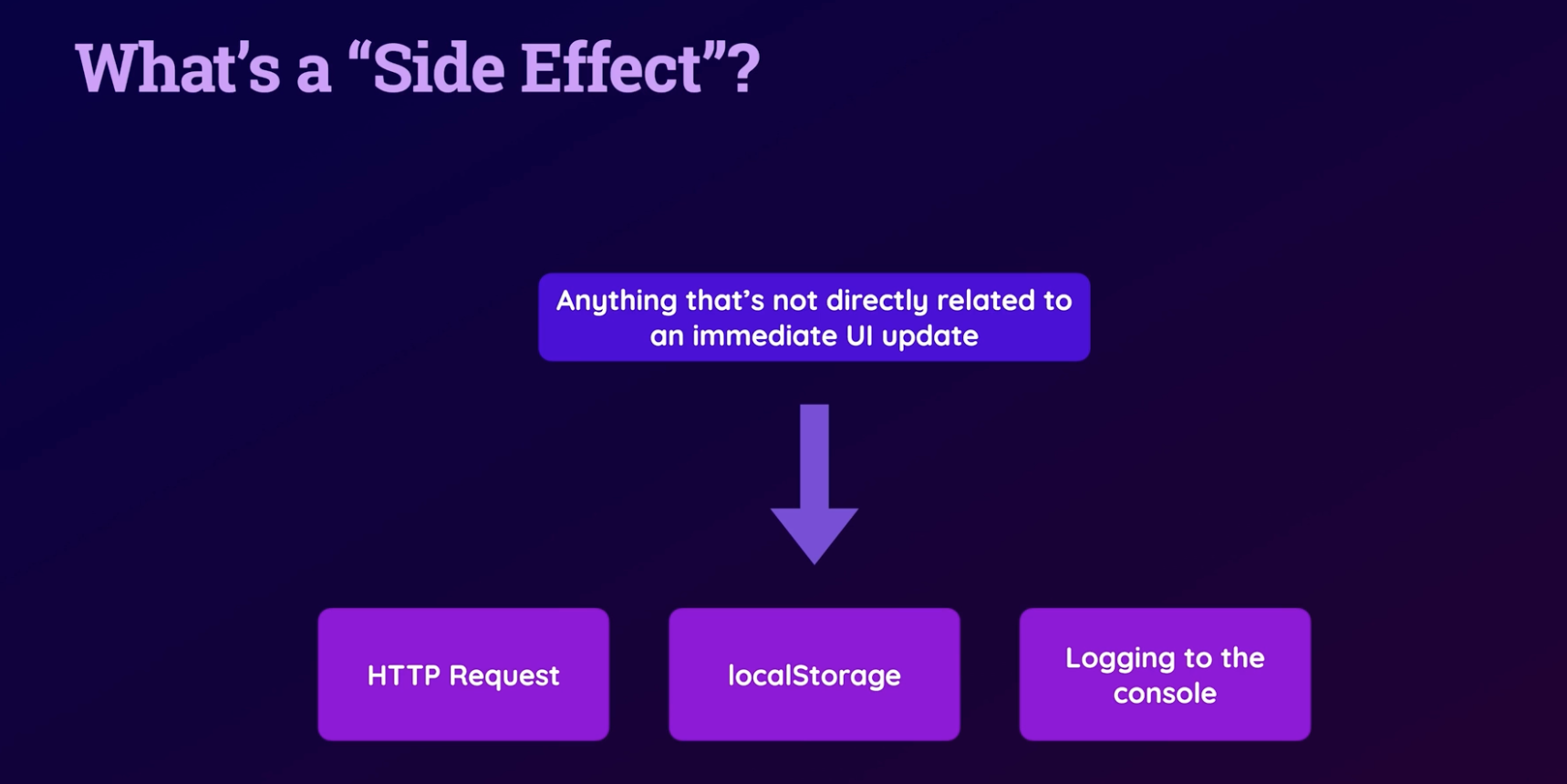
//this.count$ = store.select('counter');

this.count$ = store.select(selectCount);//ch 382

}

It’s a good practice to defining Selectors in one place]

Effects:



You must avoid side effects and should be asynchronies functions in reducer, clean and simple .Sometimes you may want to log to console , or local storage etc. To handle side effects add ngrxpackage using command

Ng add @ngrx/effects

Above commands add in standalone main.ts file

bootstrapApplication(AppComponent, {

providers: [provideStore({ counter: counterReducer }), provideEffects()]

});

In reugulra app it adds effects

imports: [BrowserModule, StoreModule.forRoot({

counter:counterReducer

}), EffectsModule.forRoot([])],

Create counter.effects.ts(good practice to keep all separate)) in store directory

@Injectable()

export class CounterEffects{

saveCount = createEffect(

()=> this.actions$.pipe(

ofType(increment, decrement),//lets us define for which aciton we need to execute code here

tap((action) => {

console.log(action);

localStorage.setItem('count', action.value.toSting());

}) //execute code some action you want to trigger to backend

// or perform side efcct without any trigger not dispacth hereafter

//.Here we use tap operator to register a fun that doesnt yield new observable once its done

), {dispatch:false} );

constructor(private actions$:Actions){//since we need to injet soemthing(Actions not Action) here in this class, $ indicates its an obersvable

}

}

Older syntax which is not supported in newer ngrx anymore

//older version of ngrx syntax

@Effect({dispatch: false})

saveCount = this.actions$.pipe(

ofType(increment, decrement),//lets us define for which aciton we need to execute code here

tap((action) => {

console.log(action);

localStorage.setItem('count', action.value.toString());

}) //execute code some action you want to trigger to backend

// or perform side efcct without any trigger not dispacth hereafter

//.Here we use tap operator to register a fun that doesnt yield new observable once its done

) ;

In standalone provide class name in array in main .ts file

bootstrapApplication(AppComponent, {

providers: [provideStore({ counter: counterReducer }), provideEffects([CounterEffects])]

});

And in regular angular proect in app module ts

imports: [BrowserModule, StoreModule.forRoot({

counter:counterReducer

}), EffectsModule.forRoot([CounterEffects])],

Here we are saving data value only saved first time in local storeage , To get latest data from store e use withLatestFrom() like below

@Injectable()

export class CounterEffects{

saveCount = createEffect(

()=> this.actions$.pipe(

ofType(increment, decrement),//lets us define for which aciton we need to execute code here

withLatestFrom(this.store.select(selectCount)),

tap(([action, counter]) => {//from store we get aciton and counter so we use array

console.log(action);

localStorage.setItem('count', counter.toString());

}) //execute code some action you want to trigger to backend

// or perform side efcct without any trigger not dispacth hereafter

//.Here we use tap operator to register a fun that doesnt yield new observable once its done

), {dispatch:false} );

//older version of ngrx syntax

// @Effect({dispatch: false})

// saveCount = this.actions$.pipe(

// ofType(increment, decrement),//lets us define for which aciton we need to execute code here

// tap((action) => {

// console.log(action);

// localStorage.setItem('count', action.value.toString());

// }) //execute code some action you want to trigger to backend

// // or perform side efcct without any trigger not dispacth hereafter

// //.Here we use tap operator to register a fun that doesnt yield new observable once its done

// ) ;

constructor(private actions$:Actions , private store:Store<{counter:number}>){//since we need to injet soemthing(Actions not Action) here in this class, $ indicates its an obersvable

}

}

Adding second effect , load data from localstorage into store so we can continue with counter. When we reoload app. For that we need two new action so in counter.action.ts file we add them

Init is clean , Set needs value to set a value

export const init = createAction(

'[Counter] Init'

);

export const set = createAction(

'[Counter] Set',

props<{value : number}>()

);

Load storage data and dispatch set once its done

//take coutner and store it into ngrx store so when we

//reload the app we conitnue from that value in counter

loadCount = createEffect(() => this.actions$.pipe(

ofType(init),

switchMap(()=>{ //lets you to new a observable chain

const storedCounter = localStorage.getItem('count');

if(storedCounter){

return of(set({value: +storedCounter}));//swithcMap needs observable. convert to a number so use +

}

return of(set({value:0}));//of fn return observable by qrapping the value

})

))

Best place to use ithis initalixing action is in app component ts file

export class AppComponent implements OnInit {

ngOnInit(): void {

this.store.dispatch(init());

}

constructor(private store:Store){}

}