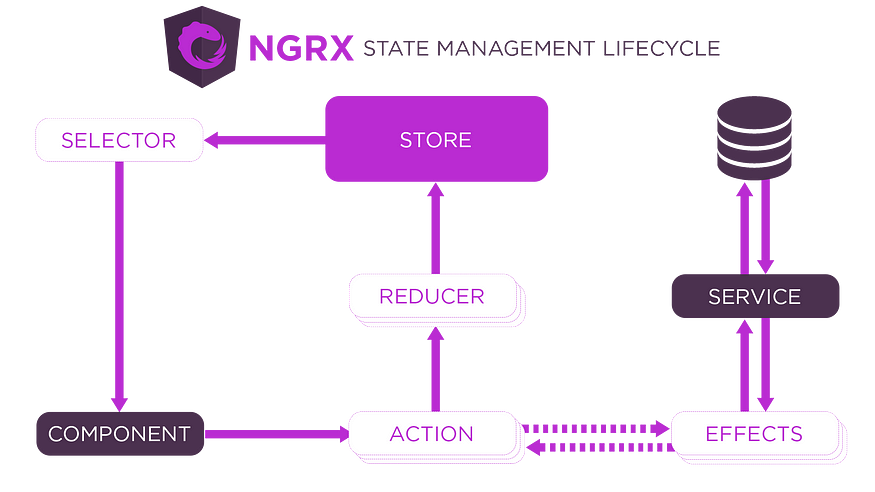
**What`s NgRx?**

As mentioned in the introduction, NgRx is a library for state management inspired by Redux. It implements the Redux principles for Angular, providing a predictable and structured way of managing the application state. NgRx uses concepts such as actions, reducers, effects, and selectors to manipulate the state in a controlled manner.



**Store**

“Store” is the part of the state manager that is implemented with a structure of immutable data. In other words, the data contained therein cannot be changed directly. Therefore, all state changes must be done through actions. These actions define what will be changed in the state through a mechanism called the ‘reducer,’ which we will talk about in the course of the article. This function processes communication with the state.

**Reducer**

The responsibility of the “reducer” is to process all necessary actions that will change the store's state. It receives as input the current state and the action and returns the new state after the change.

**Action**

The “actions” are simply objects that represent a change in the state. They are sent to the store with the information that will change the store’s state.

**Selectors**

The “selectors” are functions used to access and get specific information of state stored. Therefore they allow the components to request only the data necessary, instead of getting directly the complete state of the store, this makes the code more modular, reusable, and easy to maintain. Furthermore, they can make calc and transformation in the state before delivering to the component that requested it.

**Effects**

The “effects” refer to functionalities that deal with asynchronous tasks or side effects, such as network requests, database access, calls to external APIs, or any operation that is not purely synchronous. However, we will not use them in this article. I prefer to write an article specifically dedicated to this topic soon.

**NgRx in Practice**

if you want to clone it. However, you can also apply the concepts in an application from scratch, in my application, I will use state manager to favorite the products listed. Our application will have some components that will request the favorite products from our store. You will see more details in the following steps. Just a note: the code implementation of this project was done using Angular version 16 and NgRx version 7.8.0.

**Installing NgRx in Angular**

To install the NgRx in you project, run the following command in the root folder:

npm install @ngrx/store --save

**Structuring NgRx in our Project**

After installing the NgRx in our project, now is the hour to structure it, in the following step we will create our actions, reducers, selectors, and the initial state.

First, we will create a folder with the name “states”, inside of the folder app from our application, inside this folder that just create, we will create another folder called “favorite-product”. Right away we will create more three folders inside from “favorite-product” folder, are they: action, reducer, and selector. The structure will be that:

app ->  
 states ->  
 favorite-product ->  
 action  
 reducer  
 selector

Now we will create a file called “app.state.ts” inside of the “favorite-product” folder, this file will be the model from our initial state:

import { FavoriteProduct } from "src/app/models/favorite-product.model";  
  
export interface AppState {  
 products:FavoriteProduct[];  
}

Notice that within our “AppState” interface, we have a property called “products” that will represent the type of our initial state, which is an array of “FavoriteProduct”. Therefore, we will need to create this interface in our project. In my application, I created it in the following path: “src/app/models/favorite-product.model”. However, you can place it wherever you want; the important thing is to import it correctly in our “app.state.ts” file.

export interface FavoriteProduct {  
 id: number;  
 name: string;  
 price: number;  
 image: string;  
 isFavorite?:boolean;  
}

**Creating the action**

In this step, we will create our action inside the “src\app\states\favorite-product\action” folder. Our file will be named “app.action.ts”. In this file, we will have functions that represent all the actions that will alter the state of the data we are manipulating. See the following code:

import { createAction, props } from '@ngrx/store';  
import { FavoriteProduct } from "src/app/models/favorite-product.model";  
  
export const add = createAction('[FavoriteProduct] Add', props<{ product: FavoriteProduct }>());  
export const remove = createAction('[FavoriteProduct] Remove', props<{ product: FavoriteProduct }>());  
export const updateAllState = createAction('[FavoriteProduct] Update all state of favorites products',   
 props<{ products: FavoriteProduct[] }>());  
export const clear = createAction('[FavoriteProduct] Clear');

It is noticed that the “createAction” function from the “@ngrx/store” module takes an action description as the first parameter and the data to be processed as the second parameter. Our “favorite-product” state will have the following actions: add, remove, updateAllState, and clear.

**Creating the reducer**

We need to create our “app.reducer.ts” file in the “src\app\states\favorite-product\reducer” folder. In this file, we will initialize the initial state of the favorite product list and implement the business logic for each action before updating the state. Please see the following code for more details:

import { createReducer, on } from '@ngrx/store';  
import { add, remove, clear, updateAllState } from '../action/app.action';  
import { AppState } from '../app.state';  
  
export const initialState: AppState = {  
 products:[],  
};  
  
export const favoriteReducer = createReducer(  
 initialState,  
 on(add, (state, {product}) => (  
 {  
 ...state,  
 products: [...state.products, product]  
 }  
 )  
 ),  
 on(remove, (state, {product}) => ({  
 ...state,  
 products: state.products.filter((p)=> product.id != p.id)  
 })),  
 on(updateAllState, (state, {products}) => (  
 {  
 ...state,  
 products  
 }  
 )  
 ),  
 on(clear, state => initialState)  
);

First, notice that the initial state is being initialized with an empty array of favorite products. Then, take note that the “createReducer” function from the “@ngrx/store” module is receiving the initial state and several “ons” functions that associate actions with state changes.

* The “add” action simply adds a new product to the current state.
* Similarly, the “remove” action removes a specific product from the current state.
* The “updateAllState” action updates the state with a list of products received as a parameter.
* Lastly, but not less important, the “clear” action resets the state by setting it to the initial state, effectively clearing all favorite products from the storage.

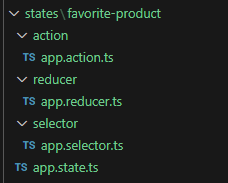
**Creating the selector**

Now, to retrieve data from the storage, we need to create our selector. For this, we must create the file ‘app.selector.ts’ in the folder ‘src\app\states\favorite-product\selector’. Please see the following code for more details:

import { createFeatureSelector, createSelector } from '@ngrx/store';  
import { AppState } from '../app.state';  
import { FavoriteProduct } from "src/app/models/favorite-product.model";  
  
// Get complete state of the favorites products in application  
export const selectAppState = createFeatureSelector<AppState>('favorite');  
  
// get All favorites products  
export const selectProducts = createSelector(  
 selectAppState,  
 (state: AppState) => state.products  
);  
  
// get One favorite product by ID  
export const selectProductById = createSelector(  
 selectProducts,  
 (products: FavoriteProduct[], props: { productId: number }) =>  
 products.find(product => product.id === props.productId)  
);

Notice that in the first function, we are obtaining the entire state from the ‘favorite’ storage. In the second function, we are using this complete state to retrieve only the favorite products. Finally, the last function utilizes the state of favorite products to obtain a specific product by its ID. We can use all these functions in our application to retrieve the necessary data.

Our folder structure is organized as follows:



Before using our state management, we need to import the StoreModule into our module and include our reducer within it:

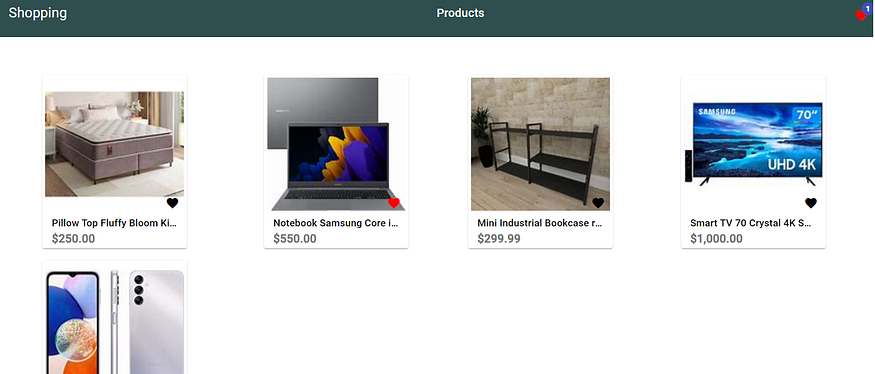
import { NgModule } from '@angular/core';  
import { AppComponent } from './app.component';  
import { StoreModule } from '@ngrx/store';  
import { favoriteReducer } from './states/favorite-product/reducer/app.reducer';  
  
@NgModule({  
 declarations: [  
  
 ],  
 imports: [  
 StoreModule.forRoot({favorite:favoriteReducer})...  
 ],  
 providers: [],  
 bootstrap: [AppComponent]  
})  
export class AppModule { }

**Using an action**

To use an action from our state, we first need to import “Store” from the “@ngrx/store” module and include it in the constructor. Here’s an example:

import { Store } from '@ngrx/store';  
  
@Component({  
 selector: 'app-home-product',  
 templateUrl: './home-product.component.html',  
 styleUrls: ['./home-product.component.scss']  
})  
export class HomeProductComponent{  
  
constructor(private readonly storageService:StorageService<FavoriteProduct[]>){}  
  
}

After importing the ‘store’, we can now use an action. In our application, a product is added to the favorites whenever we click on the heart icon for a specific product. In the image below, you can see that the product ‘Notebook’ has been added to the favorites list. Also, notice the heart icon on the top right corner, which indicates the number of items added to the favorites — in this case, it shows ‘1’.



Therefore, with each click on the favorite icon, we trigger the ‘dispatch’ function, which takes the desired action as a parameter. In this case, the action would be ‘add’. See the following code:

<mat-icon [style.color]="product.isFavorite?'red':'black'" class="mat-icon-lg icon"   
(click)="addProductToFavorites(product)" >favorite</mat-icon>

import { Store } from '@ngrx/store';  
  
@Component({  
 selector: 'app-home-product',  
 templateUrl: './home-product.component.html',  
 styleUrls: ['./home-product.component.scss']  
})  
export class HomeProductComponent{  
  
constructor(private readonly storageService:StorageService<FavoriteProduct[]>){}  
  
 addProductToFavorites(product:FavoriteProduct){  
 const favoriteProduct:FavoriteProduct = {  
 ...product,  
 isFavorite: true  
 }  
 this.store.dispatch(add({product}));  
 }  
  
}

So, whenever we trigger the ‘dispatch’ function, the clicked product is sent to the ‘reducer,’ which will be responsible for processing the data before storing it in the state.

**Using an selector**

To use a selector, we also need to import the ‘Store’ from the ‘@ngrx/store’ module. However, the function used to retrieve data using a selector is the ‘pipe’. See more details in the following code:

import { Component } from '@angular/core';  
import { Observable } from 'rxjs';  
import { FavoriteProduct } from 'src/app/models/favorite-product.model';  
import { Store, select } from '@ngrx/store';  
import { AppState } from 'src/app/states/favorite-product/app.state';  
import { selectProducts } from 'src/app/states/favorite-product/selector/app.selector';  
  
@Component({  
 selector: 'app-header',  
 templateUrl: './header.component.html',  
 styleUrls: ['./header.component.scss']  
})  
export class HeaderComponent {  
  
 favoritesProducts:Observable<Array<FavoriteProduct>> = this.store.pipe(select(selectProducts))  
  
 constructor(private router: Router, private store:Store<AppState>) { }  
  
}

Notice that inside the ‘pipe’, we use the ‘select’ function from the ‘@ngrx/store’ module, which takes the selector as a parameter. In this case, the selector is ‘selectProducts’, which returns an ‘Observable’ of an array of ‘FavoriteProduct’. This list is stored in a property and is used in the template to display the quantity of favorite products. See more details in the template code below:

<mat-icon [style.color]="'red'"   
matBadge="{{(favoritesProducts | async)?.length}}"   
matBadgePosition="above after">favorite</mat-icon>