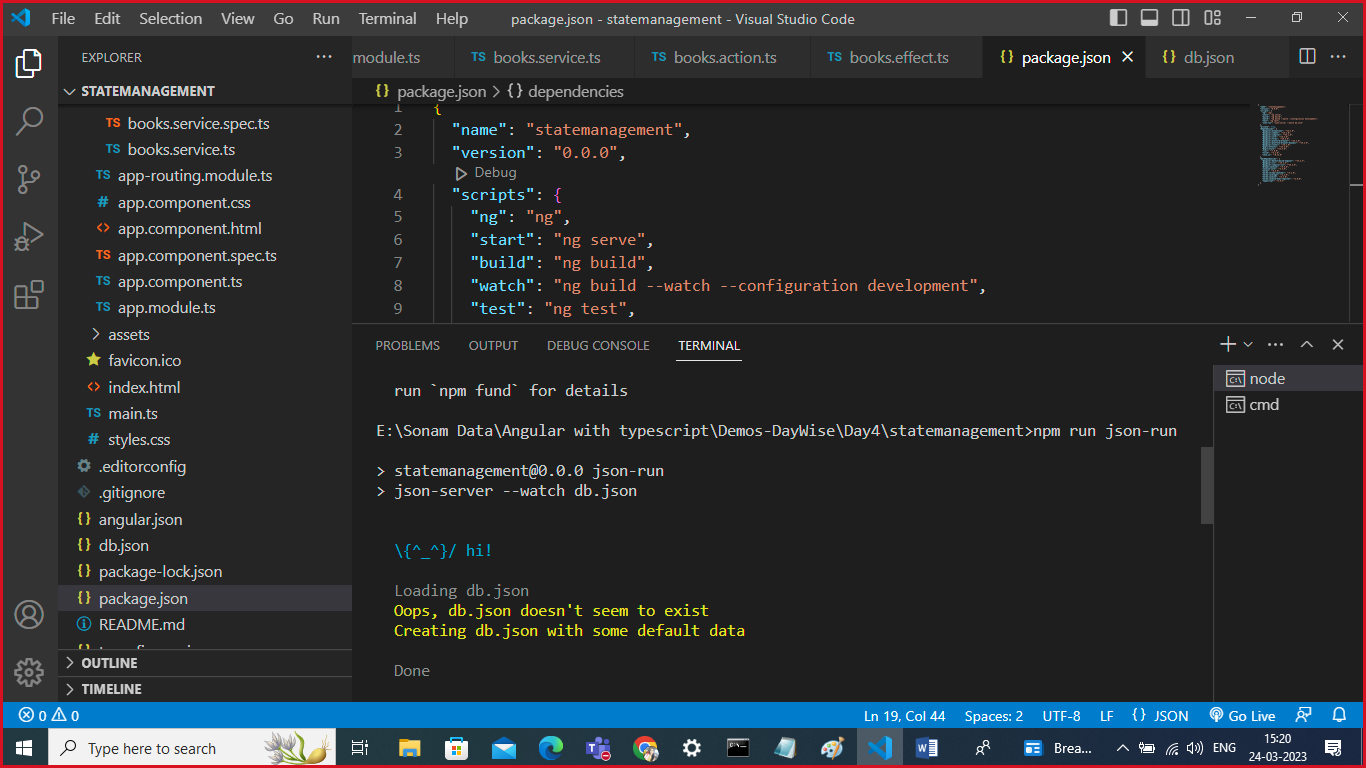
NgRx: CRUD with API

Set up a JSON Server: npm install –g json-server

Edit your package.json file by including below script.

"json-run": "json-server --watch db.json"

Execute :



Edit db.json file

{

  "books": [

    {

      "id": 1,

      "name": "Advanced JAVA",

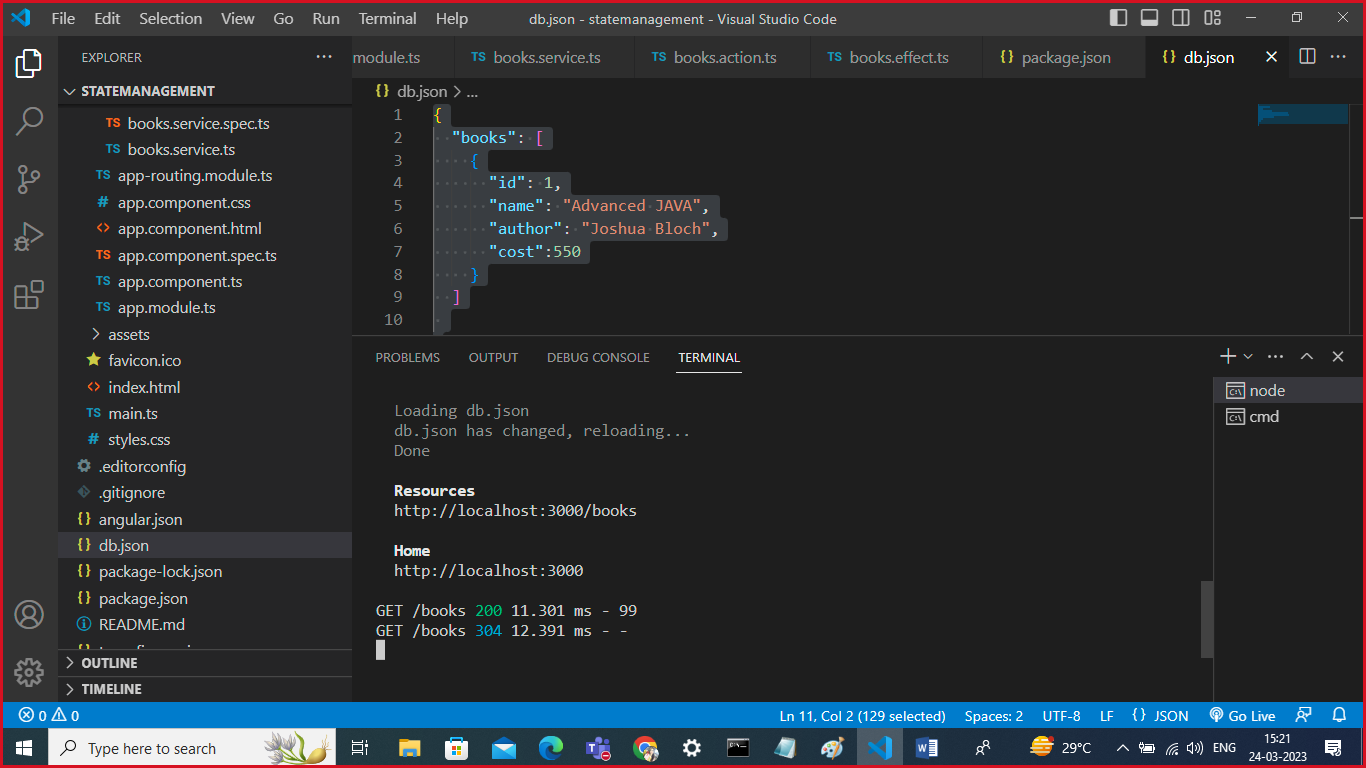
      "author": "Joshua Bloch",

      "cost":550

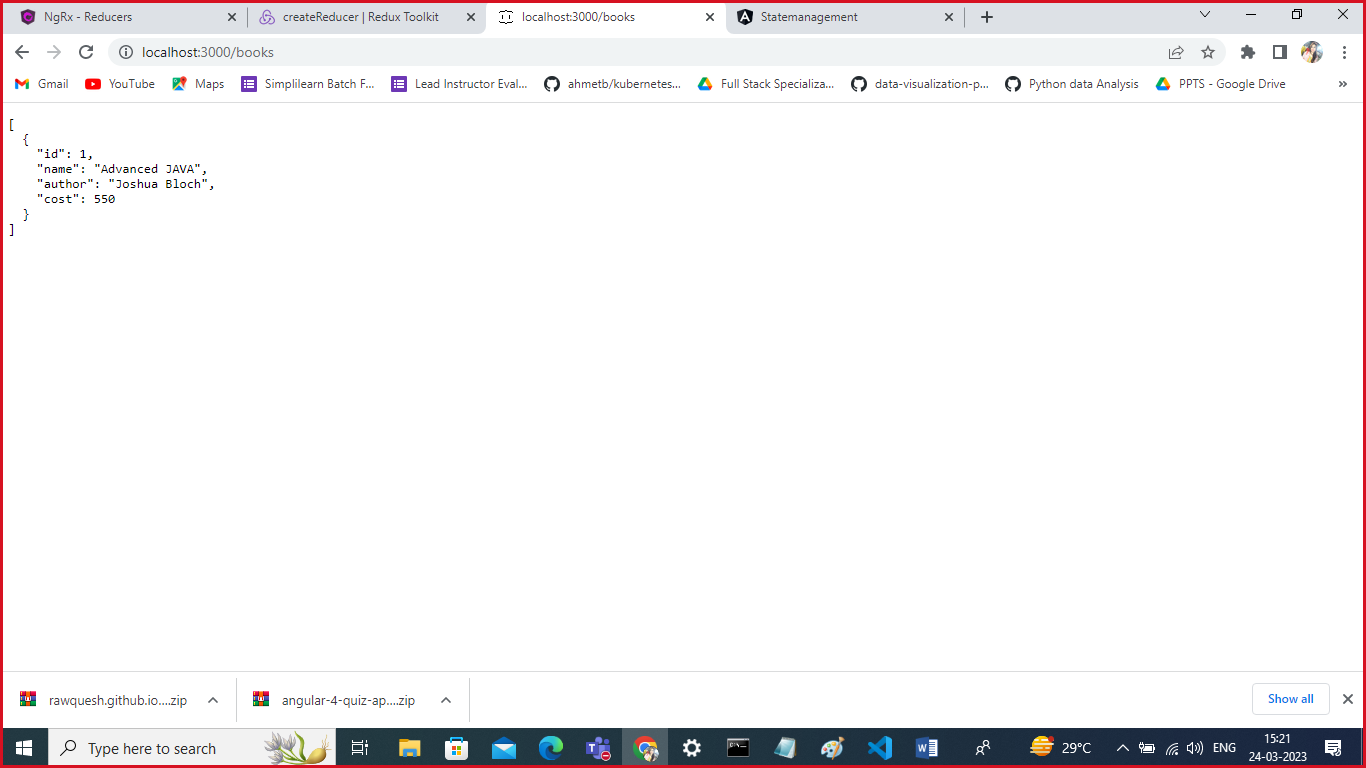
    }

  ]

}



Check get API



Create app: ng new

Then create module: ng g m books –routing

Created component: ng g c home –skip-tests

Edit app.routing.module.ts with module lazy loading

const routes: Routes = [

  {path:'books',loadChildren:()=>

import('./books/books.module').then((x)=>x.BooksModule)}

];

Create route in books-routing.module.ts

const routes: Routes = [

  {path:'',component:HomeComponent,pathMatch:'full'}

];

Create model: ng g interface books/store/books

export interface Books {

    id:number;

    name:string;

    author:string;

    cost:number

}

Create reducer

Src/app/books/store/books.reducer.ts

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

import { Books } from "./books";

import { booksFetchAPISuccess } from "./books.action";

//empty array assigned for initial data store

export const initialState: ReadonlyArray<Books>=[];

//create reducer by sending initial value as input

export const BooksReducer= createReducer(initialState,

    on(booksFetchAPISuccess,(state,{allBooks})=>{

        return allBooks;

    }));

Create selectors:

src/app/books/store/books.selector.ts

import { createFeatureSelector } from "@ngrx/store";

import { Books } from "./books";

export const selectBooks=

createFeatureSelector<Books[]>('mybooks');

//selector created to fetch th data from feature module

//this mybooks selector must be register the reduce to get the data

Update books module.ts for registering selector.

imports: [

    CommonModule,

    BooksRoutingModule,

    StoreModule.forRoot({}),

    EffectsModule.forRoot({}),

    StoreModule.forFeature('mybooks',BooksReducer),

    EffectsModule.forFeature([BooksEffect])

  ],

Ng g s books –skip-tests

Books.service.ts

import { HttpClient } from '@angular/common/http';

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

import { Books } from './store/books';

@Injectable({

  providedIn: 'root'

})

export class BooksService {

  constructor(private http: HttpClient) { }

  get(){

    return this.http.get<Books[]>("http://localhost:3000/books");

  }

}

ng g class books/store/books.action

books.action.ts

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

import { Books } from "./books";

//action invoke the API

export const invokeBooksAPI= createAction(

    '[Books API] Invoke Books Fetch API'

);

//action invoke after successfull API Call

export const booksFetchAPISuccess= createAction(

    '[Books API] Fetch API Success',

    props<{allBooks:Books[]}>()

)

ng g class books/store/books.effect

books.effect.ts file

import { Injectable } from "@angular/core";

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

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

import { EMPTY, mergeMap, withLatestFrom,map } from "rxjs";

import { BooksService } from "../books.service";

import { booksFetchAPISuccess, invokeBooksAPI } from "./books.action";

import { selectBooks } from "./books.selector";

@Injectable()

export class BooksEffect {

    constructor(

        private action$:Actions,

        private service:BooksService,

        private store:Store

    ){}

    loadAllBooks$= createEffect(()=>

        this.action$.pipe(

            ofType(invokeBooksAPI),

            withLatestFrom(this.store.pipe(select(selectBooks))),

            mergeMap(([,bookfromStore])=>{

                if(bookfromStore.length>0){

                    return EMPTY;

                }

                return this.service.get()

                .pipe(map((data)=>booksFetchAPISuccess({allBooks:data})))

            }))

        )

}

[Home.component.ts](http://Home.component.ts)

import { Component } from '@angular/core';

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

import { invokeBooksAPI } from '../store/books.action';

import { selectBooks } from '../store/books.selector';

@Component({

  selector: 'app-home',

  templateUrl: './home.component.html',

  styleUrls: ['./home.component.css']

})

export class HomeComponent {

  constructor(private store:Store){}

  books$= this.store.pipe(select(selectBooks));

  ngOnInit(){

    this.store.dispatch(invokeBooksAPI());

    //invoke ngrx effect which will call api from service

  }

}

[Home.component.html](http://Home.component.html)

<h1>Books Data</h1>

<div class="container">

    <div class="row">

        <table>

            <thead>

                <tr>

                    <th>Id</th><th>Name</th><th>Author</th><th>Cost</th>

                </tr>

            </thead>

            <tbody>

                <tr \*ngFor="let b of books$ | async">

                    <td>{{b.id}}</td>

                    <td>{{b.name}}</td>

                    <td>{{b.author}}</td>

                    <td>{{b.cost}}</td>

                </tr>

            </tbody>

        </table>

    </div>

</div>

Make sure you import books module in app.module.ts

 imports: [

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    BooksModule

  ],

Create Global State Management

Create Entity: ng g interface shared/store/appstate

Appstate.ts

export interface Appstate {

    apiStatus:string;

    apiResponseMessage:string

}

App.action.ts

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

import { Appstate } from "./appstate";

export const setAPIStatus= createAction(

    '[API] success or failure status',

    props<{apiStatus:Appstate}>()

)

App.reducer.ts

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

import { setAPIStatus } from "./app.action";

import { Appstate } from "./appstate";

export const initialState: Readonly<Appstate>={

    apiResponseMessage:'',

    apiStatus:''

}

export const appReducer=createReducer(initialState,

    on(setAPIStatus,(state,{apiStatus})=>{

        return{

            ...state,

            ...apiStatus

        }

    }))

App.selector.ts

import { createFeatureSelector } from "@ngrx/store";

import { Appstate } from "./appstate";

export const selectAppState= createFeatureSelector<Appstate>('appState')

registered the reducer in app.module.ts

imports: [

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    BooksModule,

    StoreModule.forRoot({appState:appReducer})

  ],

Create add component and adjust the routing

Update action

export const invokeSaveNewBookApi=createAction(

    '[Books API] Invoke save new book API',

    props<{newBook:Books}>()

)

export const saveNewBookAPISuccess=createAction(

    '[Books API] save new api success',

    props<{newBook:Books}>()

)

Update effect

import { Injectable } from "@angular/core";

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

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

import { EMPTY, mergeMap, withLatestFrom,map, switchMap } from "rxjs";

import { setAPIStatus } from "src/app/shared/store/app.action";

import { Appstate } from "src/app/shared/store/appstate";

import { BooksService } from "../books.service";

import { booksFetchAPISuccess, invokeBooksAPI, invokeSaveNewBookApi, saveNewBookAPISuccess } from "./books.action";

import { selectBooks } from "./books.selector";

@Injectable()

export class BooksEffect {

    constructor(

        private action$:Actions,

        private service:BooksService,

        private store:Store,

        private appStore:Store<Appstate>

    ){}

    saveNewBooks$=createEffect(()=>{

        return this.action$.pipe(

            ofType(invokeSaveNewBookApi),

            switchMap((action)=>{

                this.appStore.dispatch(

                    setAPIStatus({apiStatus:{apiResponseMessage:'',apiStatus:''}})

                );

                return this.service.create(action.newBook).pipe(

                    map((data)=>{

                        this.appStore.dispatch(

                            setAPIStatus({

                                apiStatus:{apiResponseMessage:'',apiStatus:'success'}

                            })

                        )

                        return saveNewBookAPISuccess({newBook:data})

                    })

                );

            })

        )

    })

    loadAllBooks$= createEffect(()=>

        this.action$.pipe(

            ofType(invokeBooksAPI),

            withLatestFrom(this.store.pipe(select(selectBooks))),

            mergeMap(([,bookfromStore])=>{

                if(bookfromStore.length>0){

                    return EMPTY;

                }

                return this.service.get()

                .pipe(map((data)=>booksFetchAPISuccess({allBooks:data})))

            }))

        )

}

Update reducer

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

import { Books } from "./books";

import { booksFetchAPISuccess, saveNewBookAPISuccess } from "./books.action";

//empty array assigned for initial data store

export const initialState: ReadonlyArray<Books>=[];

//create reducer by sending initial value as input

export const BooksReducer= createReducer(initialState,

    on(booksFetchAPISuccess,(state,{allBooks})=>{

        return allBooks;

    }),

    on(saveNewBookAPISuccess,(state,{newBook})=>{

        let newState=[...state];

        newState.unshift(newBook);

        return newState;

    }),

    );

Add.component.ts

import { Component } from '@angular/core';

import { Router } from '@angular/router';

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

import { selectAppState } from 'src/app/shared/store/app.selector';

import { Appstate } from 'src/app/shared/store/appstate';

import { Books } from '../store/books';

import { invokeSaveNewBookApi } from '../store/books.action';

@Component({

  selector: 'app-add',

  templateUrl: './add.component.html',

  styleUrls: ['./add.component.css']

})

export class AddComponent {

  constructor(

    private store:Store,

    private appStore:Store<Appstate>,

    private router:Router

  ){}

  bookForm:Books={id:0,author:'',name:'',cost:0}

  save(){

    this.store.dispatch(invokeSaveNewBookApi({newBook: this.bookForm}));

    let apiStatus$= this.appStore.pipe(select(selectAppState))

    apiStatus$.subscribe((apstate)=>{

      if(apstate.apiStatus=='success'){

        this.appStore.dispatch({apiStatus:{apiResponseMessage:'',apiStatus:''}});

      }

      this.router.navigate(['/']);

    })

  }

}

Add.component.html