import \* as AuthActions from './auth.actions';

export interface State {

token: string;

authenticated: boolean;

}

const initialState: State = {

token: null,

authenticated: false

};

export function **authReducer**(state = initialState, action: AuthActions.AuthActions) {

switch (action.type) {

**case (AuthActions.SIGNUP):**

**case (AuthActions.SIGNIN):**

return {

...state,

authenticated: true

};

**case (AuthActions.LOGOUT):**

return {

...state,

token: null,

authenticated: false

};

case (AuthActions.SET\_TOKEN):

return {

...state,

token: action.payload

};

default:

return state;

}

}--------------------------

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

export const **SIGNUP** = 'SIGNUP';

export const **SIGNIN** = 'SIGNIN';

export const **LOGOUT** = 'LOGOUT';

export const **SET**\_**TOKEN** = 'SET\_TOKEN';

export class Signup implements **Action** {

readonly type = SIGNUP;

}

export class Signin implements Action {

readonly type = SIGNIN;

}

export class Logout implements Action {

readonly type = LOGOUT;

}

export **class SetToken implements Action** {

**readonly type = SET\_TOKEN;**

**constructor(public payload: string)** {}

}

export type AuthActions = Signup | Signin | Logout | SetToken;

<li><a style="cursor: pointer;"(click)="onLogout()"

\*ngIf**="(authState | async).authenticated"**>Logout</a></li><li class="dropdown" appDropdown \***ngIf="(authState | async).authenticated">**

<a style="cursor: pointer;" class="dropdown-toggle" role="button">Manage <span class="caret"></span></a>

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

import \* as **fromShoppingList** from '../shopping-list/store/shopping-list.reducers';

import \* as fromAuth from '../auth/store/auth.reducers';

export interface AppState {

shoppingList: **fromShoppingList**.State,

auth: **fromAuth**.State

}

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

shoppingList: **fromShoppingList**.**shoppingListReducer**,

auth: fromAuth.**authReducer**

};-------------------------------------

import { ShoppingListModule } from './shopping-list/shopping-list.module';

import { AuthModule } from './auth/auth.module'

import { CoreModule } from './core/core.module';

import { **reducers } from './store/app.reducers'**;

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,HttpClientModule,

AppRoutingModule, SharedModule,

ShoppingListModule, AuthModule,

CoreModule,

**StoreModule.forRoot(reducers)**

],

bootstrap: [**AppComponent**]

})

export class AppModule { }

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

import { DataStorageService } from '../../shared/data-storage.service';

import { AuthService } from '../../auth/auth.service';

import \* as fromApp from '../../store/app.reducers';

import \* as fromAuth from '../../auth/store/auth.reducers';

@Component({

selector: 'app-header',

templateUrl: './header.component.html'

})

export class **HeaderComponent** implements OnInit {

authState: Observable<**fromAuth**.**State**>;

constructor(private **dataStorageService: DataStorageService**,

private **authService: AuthService**,

private **store: Store**<fromApp.**AppState**>) {

}

**ngOnInit**() {

this.authState = this.**store**.**select**('auth');

}

**onSaveData**() {

this.**dataStorageService**.**storeRecipes**()

.subscribe(

(response) => {

console.log(response);} );}

**onFetchData**() {

this.dataStorageService.getRecipes();}

**onLogout**() {

this.authService.logout();}}

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

import \* as firebase from 'firebase';

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

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

import \* as fromApp from '../store/app.reducers';

import \* as AuthActions from './store/auth.actions';

@Injectable()

export class **AuthService** {

constructor(private router: Router, private store: Store<fromApp.AppState>) {}

signupUser(email: string, password: string) {

firebase.auth().createUserWithEmailAndPassword(email, password)

.then(

user => {

this.store.**dispatch**(new AuthActions.**Signup**());

firebase.auth().**currentUser**.**getToken**()

.then(

(token: string) => {

this.store.**dispatch**(new **AuthActions**.**SetToken**(token));

}

)

}

)

.catch(

error => console.log(error)

)

}

signinUser(email: string, password: string) {

firebase.auth().**signInWithEmailAndPassword**(email, password)

.then(

response => {

**this.store.dispatch(new AuthActions.Signin());**

this.router.navigate(['/']);

firebase.auth().currentUser.getToken()

.then(

(token: string) => {

this.**store**.**dispatch(new AuthActions.SetToken**(token));

}

)

}

)

.catch(

error => console.log(error)

);

}

logout() {

**firebase.auth().signOut();**

**this.store.dispatch(new AuthActions.Logout());**

}

}-----------------------------------------

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

import \* as fromApp from '../store/app.reducers';

import \* as fromAuth from './store/auth.reducers';

@Injectable()

export class **AuthGuard** implements **CanActivate** {

**constructor**(private **store: Store<fromApp.AppState**>) {}

**canActivate**(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {

return **this.store.select('auth').map((authState: fromAuth**.State) => {

return authState.authenticated;

});}}------------------------------

import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

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

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

import 'rxjs/add/operator/switchMap';

**import \* as fromApp from '../store/app.reducers';**

**import \* as fromAuth from '../auth/store/auth.reducers';**

@Injectable()

export class **AuthInterceptor** implements **HttpInterceptor** {

constructor(private store: **Store<fromApp.AppState>**) {}

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

console.log('Intercepted!', req);

return **this.store.select**('auth')

.**take**(1)

.**switchMap**((authState: fromAuth.State) => {

const copiedReq = **req.clone**({params: req.params.**set('auth', authState.token**)});

return next.handle(copiedReq);

})

}

}

//map function will wrap one observable and return 1 observable

//switchMap() function will wrap one observable and return a val

//I select a slice of the off state

//then we switch to map which map it to extract the token and

//return the request

//the issue is stores select sets up an ongoing subscription to our

//store and whenever we change to state, it will fire and

//then it will extract the token and send the request

//this is obvously not what we want and it's easy to fix

//take(1)

//only gets this value once