NGRX

ФРЕЙМВОРК ДЛЯ СОЗДАНИЯ РЕАКТИВНЫХ ПРИЛОЖЕНИЙ НА ANGULAR (STORE,EFFECTS)

Из чего состоит:

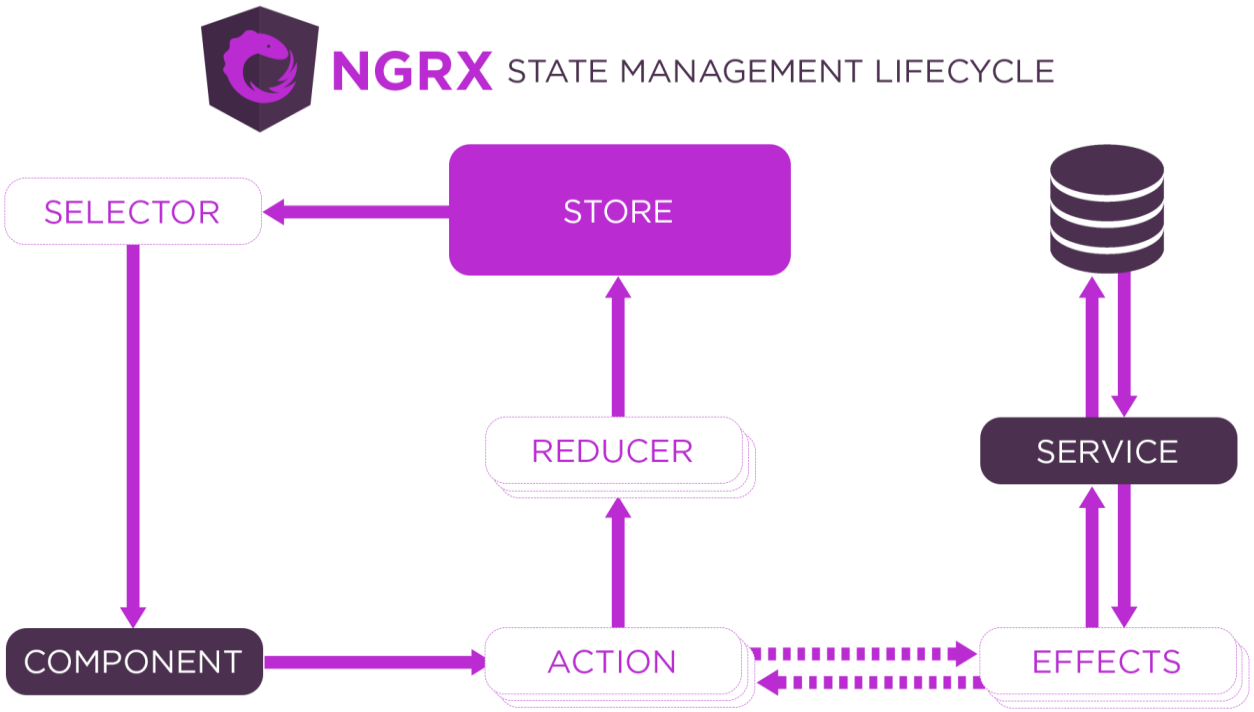
1.State
Store - глобальный стэйт менеджмент
Effects - побочные эффекты
Router Store - стэйт навигации
Entity - стэйт для работы с коллекциями (адаптер для CRUD операций)
ComponentStore - библиотека локального стора для компонентов

2.Data
Data - расширение для управления сущностями (это абстракция над Store, Effects и Entity - сокращает объем кода)

3.View
Component - расширение для контроля рендеринга (хелперы ngrxPush pipe, ngrxLet directive).

4.Developer Tooling
Store Devtools - инструмент для отладки
Schematics – ngrx cli

Принцип работы



Преимущества

- Единственный источник правды

- Состояние read-only (только для чтения)

- Изменения в стор вносятся с простыми функциями

- Использование шаблона redux дает нам много интересных функций (dev tools), облегчающих отладку

- Тестирование приложений становится проще, поскольку мы вводим чистые функции для обработки изменений состояния

КАК ТОЛЬКО ВЫ ПОЧУВСТВУЕТЕ СЕБЯ КОМФОРТНО ПРИ ИСПОЛЬЗОВАНИИ NGRX, ПОНИМАНИЕ ПОТОКА ДАННЫХ В ВАШИХ ПРИЛОЖЕНИЯХ СТАНЕТ НЕВЕРОЯТНО ПРОСТЫМ И ПРЕДСКАЗУЕМЫМ

Недостатки

- Кривая обучения

- Отдельная библиотека которая не входит в Angular

- Множество заготовок кода — heavy boilerplate

Подключение

```
@NgModule({
  imports: [
    BrowserModule.withServerTransition({ appId: 'npmrun' }),
    BrowserAnimationsModule,
    FormsModule,
    AppRoutingModule,
    HttpClientModule,
    HeaderModule,
    FooterModule,
    StoreModule.forRoot(reducers,{}),
    EffectsModule.forRoot([AuthEffects]),
    StoreDevtoolsModule.instrument({
      maxAge: 25, // Retains last 25 states
      logOnly: environment.production, // Restrict extension to log-only mode
    }),

  ],
  declarations: [
    AppComponent,

  ],
  providers: [...APP_SERVICES_PROVIDERS,...APP_SERVICES_RESOLVERS,AuthGuard],
  bootstrap: [ AppComponent ],

})
export class AppModule {
  constructor(
    @Inject(PLATFORM_ID) private platformId: object,
    @Inject(APP_ID) private appId: string) {
    const platform = isPlatformBrowser(platformId) ?
      'in the browser' : 'on the server';
    console.log(`Running ${platform} with appId=${appId}`);
  }
}
```

```
@NgModule({
  declarations: [AdminpanelComponent],
  imports: [
    CommonModule,
    AdminpanelRoutingModule,
    AdminDetailModule,
    StoreModule.forFeature(fromAdmin.featureName, fromAdmin.adminReducer),
    EffectsModule.forFeature([AdminEffects]),
  ]
})
export class AdminpanelModule {
  constructor(){
    console.log("LoginModule")
  }

 }
```

Создание общего State

```
export interface State {
        auth: fromAuth.State;
        router: fromRouter.RouterReducerState<any>;
        ui: fromUI.State;
        users: fromUsers.State
}

export const reducers: ActionReducerMap<State> = {
        auth: fromAuth.authReducer,
        router: fromRouter.routerReducer,
        ui: fromUI.UIReducer,
        users: fromUsers.usersReducer
};

export const selectAuthState = createFeatureSelector<fromAuth.State>('auth');
export const selectUIState = createFeatureSelector<fromUI.State>('ui');
export const selectUsersState = createFeatureSelector<fromUsers.State>('users');
```

Создание State

```
export interface State {
        isLoggedIn: boolean;
        expireData: IExpiresData;
        authUser: IAuthUser;
        lang: string;
        // checkEnabledFunc:{ [key: string]: boolean}
}
export const initialState: State = {
        isLoggedIn: false,// need false
        expireData: null,
        authUser: null,
        lang: 'uk',


};
```

Actions


```typescript
import { createAction, props, ActionCreator } from '@ngrx/store';
import { IAuthUser } from 'src/app/interfaces/iauth.user';
import { IExpiresData } from 'src/app/interfaces/iexpires.data';

import { IRightsBodyTable, IRoleItem } from 'src/app/interfaces/itables';

export const Login = createAction('[Auth] Login');

export const LoginConfirmed = createAction('[Auth] Login Confirmed', props<IExpiresData>());

export const LoginSuccess = createAction('[Auth] Login Success', props<IAuthUser>());

export const RefreshExpiresData = createAction('[Auth] Refresh Expires Data');

export const ToLoginPage = createAction('[Auth] To Login Page');

export const SetUserLanguageByDefault = createAction('[Auth] SetUserLanguageByDefault Success');

export const ChangeUserLanguage = createAction('[Auth] ChangeUserLanguage Success', props<{payload: string}>());

export const Logout = createAction('[Auth] Logout');

export const LogoutSuccess = createAction('[Auth] Logout Success');

export const UpdateAuthUser = createAction('[Users] Update Auth User', props<IAuthUser>());

export const LoginError = createAction('[Auth] Login Error', props<IAuthUser>());
```

Reducers

```
const reducer = createReducer(
        initialState,
        on(AuthActions.Login, (state: State) => ({ ...state,isLoggedIn: false })),
        on(AuthActions.LoginConfirmed, (state: State, expireData: IExpiresData) =>  ({ ...state, isLoggedIn: true, expireData })),
        on(AuthActions.LoginSuccess, (state: State, authUser:IAuthUser) =>  ({ ...state, authUser })),
        on(AuthActions.LogoutSuccess, (state: State) =>  ({ ...state,  isLoggedIn: false, expireData: null, authUser: null  })),

        on(AuthActions.RefreshExpiresData, (state: State) =>  ({ ...state })),

        on(AuthActions.SetUserLanguageByDefault, (state: State) =>  ({ ...state, lang: state.authUser['lang']})),
        on(AuthActions.ChangeUserLanguage, (state: State, currentLanguage: {payload:string}) =>  ({ ...state, lang:currentLanguage.payload


        on(AuthActions.UpdateAuthUser,  (state: State, authUser: IAuthUser) =>  ({ ...state, authUser })),
        on(AuthActions.UpdateAuthUserFuncs,  (state: State, inputDataFormUI:{role:IRoleItem,rowFunc:IRightsBodyTable}) =>  ({ ...state, au

        on(AuthActions.LoginError, (state: State) =>  ({ ...state,  isLoggedIn: false, expireData: null, authUser: null  })),
        // on(AuthActions.LogoutError, (state: State) =>  ({ ...state,  isLoggedIn: false, expireData: null, authUser: null  })),
);
```

Selectors

```
export const selectIsLoggedIn = (state: State) => state.isLoggedIn;
export const selectAuthUser = (state: State) => state.authUser;
export const selectAuthUserLanguage = (state: State) => state.lang;
export const selectAuthExpiresData = (state: State) => state.expireData;

export const selectAuthUserRoles = createSelector(
        selectAuthUser,
        (state: IAuthUser) => state?.roles
);
export const selectAuthUserFuncs = createSelector(
        selectAuthUser,
        (state: IAuthUser) => state?.funcs
);
```

Effects

```
@Injectable()
export class AuthEffects {
  @Effect()
  checkToken$ = this.actions$.pipe(ofType(fromAuth.CheckToken),
                switchMap(() => this.authService.tryAuth()
                  .pipe(map((response: IResponse) => {
                    if(!response.status) {
                      return fromAuth.LogoutSuccess();
                    } else {
                      return fromAuth.CheckTokenSuccess(response.response)
                    }
                  }),
                  catchError((err) => {
                    console.log(err);
                    return of(false);
                  })

            )
          )
  );
```

```
@Effect()
forceLogout$ = this.actions$.pipe(ofType(fromAuth.ForceLogout),
                  switchMap((payload:{mode:number}) => {

                    if(payload.mode===2) {
                      return [ fromUI.ClosePopup(),fromAuth.LogoutSuccess(),fromAuth.ForceLogoutSuccess()]
                    } else {
                      return [fromAuth.LogoutSuccess(),fromAuth.ForceLogoutSuccess()]
                    }

                  })

);
```

```
  @Effect({ dispatch: false })
  loginError$ = this.actions$.pipe(ofType(fromAuth.LoginError),
              tap(()=>this.authService.clearLocalStorage() ) ,
              tap(()=>this.router.navigate(['login']) )

  );
```

Использование в компоненте

1.

```
export class RightsComponent implements OnInit, OnDestroy, AfterViewInit {

  public rightsTableData$:Observable<{headerTable,bodyTable}> = this.store.select(selectUsersRightsTable)

  public selectAuthUserFuncs$ = this.store.select(selectAuthUserFuncs);
```

2.

```
constructor(private store: Store,
```

3.

```
<div class="body-table" #bodyTable>

    <ng-container *ngFor="let user of selectUsersListDataList$ | async | filterUsersMain: SS.FILTER_USERS " >
        <div class="table-row" [attr.user_id]="user.userID" >
            <div class="cell" [title]="user.firstName" >
                <span class="text"> {{ user.firstName }}  </span>
            </div>
            <div class="cell" [title]="user.lastName">
                <span class="text"> {{ user.lastName }} </span>
            </div>
```

4.

```
  public toEditUser(user: IAuthUser): void {
    this.store.dispatch(EditUser( user as IAuthUser));
    this.store.dispatch(OpenPopup({payload: 'user-edit'}));
  }

  public openUserInvitePopup(): void {
    this.store.dispatch(OpenPopup({payload: 'user-invite'}));
  }
```

5.

```
public ngOnInit(): void {

  this.rightsTableData$$ = this.rightsTableData$

    .subscribe( (data: {bodyTable:RightsBodyTable[],headerTable:HeaderTable[]}) => {

      this.bodyTable = data?.bodyTable;
      this.headerTable  = data?.headerTable;
      this.cdr.detectChanges();
    });

    this.selectAuthUserFuncs$$ = this.selectAuthUserFuncs$
      .subscribe( (data: string[]) => {

        if (data) {
          this.canEditRole = this.HS.checkIsEnabledFunction('upsertRole', data);
          this.cdr.detectChanges();
        }
      });

  this.translate$$ = this.translate.onLangChange.subscribe( (item:any) => {});
  this.SS.usersPages$$.subscribe( (usersPages: IPageItem[])=>{
    usersPages.forEach( (page: IPageItem) => {
      page.name ==='rights' && !page.isShow ? this.router.navigate(['controllers']) : null;
    });
  })
}
```

= ( ) ?

: