academeez

Redux - @ngrx/*

7. @ngrx/router-store

@ngrx/router-store

academeez

- Our state is strongly effected by routing
  - navigating to a different route sometimes need to effect the data in our state
  - For example moving to a different route should fetch data from server
- @ngrx/router-store will dispatch actions based on navigation

@ngrx/router-store actions

academeez

- The actions dispatched are of type **RouterAction**
- There are different actions with different timing you can set

Order of actions

Success case:

- `ROUTER_REQUEST`
- `ROUTER_NAVIGATION`
- `ROUTER_NAVIGATED`

Error / Cancel case (with early Navigation Action Timing):

- `ROUTER_REQUEST`
- `ROUTER_NAVIGATION`
- `ROUTER_CANCEL` / `ROUTER_ERROR`

Error / Cancel case (with late Navigation Action Timing):

- `ROUTER_REQUEST`
- `ROUTER_CANCEL` / `ROUTER_ERROR`

RouterRequestAction

- Dispatched at the start of each navigation

```
export declare type RouterRequestAction<T extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    type: typeof ROUTER_REQUEST;
    payload: RouterRequestPayload<T>;
};
```

```
export declare type RouterRequestPayload<T extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    routerState: T;
    event: NavigationStart;
};
```

RouterRequestAction

- During navigation before guards or resolvers

```
export declare type RouterNavigationAction<T extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    type: typeof ROUTER_NAVIGATION;
    payload: RouterNavigationPayload<T>;
};
```

```
export declare type RouterNavigationPayload<T extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    routerState: T;
    event: RoutesRecognized;
};
```

RouterNavigated

academeez

- After successful navigation

```
export declare type RouterNavigatedAction<T extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    type: typeof ROUTER_NAVIGATED;
    payload: RouterNavigatedPayload<T>;
};
```

```
export declare type RouterNavigatedPayload<T extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    routerState: T;
    event: NavigationEnd;
};
```

RouterCancelAction

academeez

- When a navigation is canceled for example due to a guard

```
export declare type RouterCancelAction<T, V extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    type: typeof ROUTER_CANCEL;
    payload: RouterCancelPayload<T, V>;
};
```

```
export declare type RouterCancelPayload<T, V extends BaseRouterStoreState = SerializedRouterStateSnapshot> = {
    routerState: V;
    storeState: T;
    event: NavigationCancel;
};
```

RouterErrorAction

academeez

- When there is an error during navigation

```
export declare type RouterErrorAction<T, V extends BaseRouterStoreState =
SerializedRouterStateSnapshot> = {
    type: typeof ROUTER_ERROR;
    payload: RouterErrorPayload<T, V>;
};
```

```
export declare type RouterErrorPayload<T, V extends BaseRouterStoreState = SerializedRouterStateSnapshot> = {
    routerState: V;
    storeState: T;
    event: NavigationError;
};
```

EX

academeez

- Create a routing with a home page and a todo page
- on the todo page we fetch the list from a server
- We will use Effect that will listen for a route action and use @ngrx/data to fetch the todo list from the server upon route transition

Summary

- @ngrx/data is a shortcut library for managing entities in our state that are in sync with a server REST api
- @ngrx/data will save us creating a lot of repeating code like HttpClient services, reducers, actions, effects.
- @ngrx/data will provide us an easy to use **EntityCollectionService** which will provide us methods to query the server and read the data from a managed cache in the store

academeez

Thank You

You are now an @ngrx/* expert!