easy peasy → for state management

08 January 2024
13:07

Basic javascript

a) In main → Store Provider & wrap main & store = {store}

from easy-peasy (npm i)

b) store → create Store and actions

```
export const store = create Store ({
  // state
  counter : 0,
  // actions
  add : action ( state, payload) => {
      state.counter += payload.
  },
  remove : ....
})
```

c) In use — use Store State

```
let val = useStoreState (state => state.counter)
```

[illegible]

models // having the interfaces

↳ custom interface (used by state)

↳ custom model interface (uses as state data type)

↳ used by state model interface

↳ has all the model interfaces

```
interface DataModel{
    counter: number
}

export interface CounterModel{
    data: DataModel,
    add: Action<CounterModel, number>, //second parameter is the payload received
    subtract: Action<CounterModel, number>,
    customOperation: Thunk<CounterModel, number>
}

export interface PokemonModel {
    name: string,
    url: string
}
```

```
export interface PokemonListModel{
    pokemons: PokemonModel[],
    setPokemons: Action<PokemonListModel, PokemonModel[]>,
    getPokemons: Thunk<PokemonListModel, PokemonModel[] | undefined>,
    unsetPokemons: Action<PokemonListModel, PokemonModel[] | undefined>
}

export interface StoreModel{
    counter: CounterModel,
    pokemon: PokemonListModel
```

[illegible] payload [illegible]

payload [illegible] parameter

use action | undefined to make payload param as optional

Store.ts

```
import { createStore, persist } from "easy-peasy";
import { StoreModel } from "./models";
import { counterModel } from "./counter";
import { pokemonModel } from "./pokemon";

// export const store = createStore<StoreModel>({
//   counter: counterModel,
//   pokemon: pokemonModel,
// });
```

creating the store model

file defining the model (like kanda)

↳ we create the model here

easy-peasy in typescript !!!

```
import axios from "axios";
import { action, thunk } from "easy-peasy";
import { url } from "../api";
import { PokemonListModel } from "./models";

export const pokemonModel: PokemonListModel = {
    pokemons: [],
    setPokemons: action((state, payload) => {
        state.pokemons = payload;
    }),
    getPokemons: thunk(async(actions) => {
        let data = await axios.get(url)
        actions.setPokemons(data.data.results)
    }),
    unsetPokemons: action((state) => {
        state.pokemons = [] // simple this is enough
    })
```

(dont forget to add this)

action is what that modifies the state

thunk calls the actions, so modifies indirectly based on api calls

app.tsx (using)

```
// import { useStoreState } from 'easy-peasy'
import React from 'react'
import { useStoreActions, useStoreState } from './store/hooks'
import { PokemonModel } from './store/models';
import { useStoreRehydrated } from 'easy-peasy';

const App = () => {
  // let {counter} = useStoreState<CounterModel>(state => state.data) // to not k
n, we use hooks
  // let {counter} = useStoreState(state => state.counter.data) // we dont have t
his case
  // let {add, subtract, customOperation} = useStoreActions(actions => actions.co
  let {pokemons} = useStoreState(state => state.pokemon)
  let {getPokemons, unsetPokemons} = useStoreActions(actions => actions.pokemon)
```

→ to stop the repeating <---->

use the hooks — import from hooks

→ useStore,

to get state values

to get the functions and actions

hooks.ts

```
import { createTypedHooks } from 'easy-peasy';
import { StoreModel } from './models';


const typedHooks = createTypedHooks<StoreModel>();

export const useStoreActions = typedHooks.useStoreActions;
export const useStoreDispatch = typedHooks.useStoreDispatch;
export const useStoreState = typedHooks.useStoreState;
```

→ used the base store model here (etc.)

persist in easy-peasy :-

```
// persist will basically store the state in the session storage
export const store = createStore<StoreModel>(persist({
    counter: counterModel,
    pokemon: pokemonModel,
  }));
```

wrapping inside

will create persist

```
  let isRehydrated = useStoreRehydrated() //
rehydrated will tell us if store is persisted (retrieved from session storage)
  return (
    <div>
    {/* <div>{counter}</div>
    <button onClick={() => customOperation(1)}>INC</button>
    <button onClick={() => subtract(1)}>DEC</button> */}
    <button onClick={() => getPokemons()}>getPOKEMONS</button>

    {isRehydrated
    ?
    <ul>
      {pokemons.map((pokemon: PokemonModel, idx) => (<li key={idx}>{pokemon.name}</li>))}
    </ul>
    :
    <h1>Loading...</h1>
    }
    <button onClick={() => unsetPokemons([])}>removePOKEMONS</button>
    </div>
  )
)
```

basically reset the state to [], so persist (session storage) will also [illegible] updated to []

→ so, no again calls data fetched or reload

note that because terminally or session exe

→ basically taking data from session storage if it exists there (no when state can be achieved)