state\utils\entity\entity.action.ts

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

export const ENTITY\_REMOVE\_ACTION = '[ENTITY] REMOVE';

export const ENTITY\_REMOVE\_ALL\_ACTION = '[ENTITY] REMOVE ALL';

export interface EntityMeta {

entityType: string;

entityId: string | string[];

entityRemove?: boolean;

}

export function entityMeta(type: string, id: string | string[]): EntityMeta {

return {

entityType: type,

entityId: id,

};

}

export function entityRemoveMeta(

type: string,

id: string | string[]

): EntityMeta {

return {

entityId: id,

entityType: type,

entityRemove: true,

};

}

export function entityRemoveAllMeta(type: string): EntityMeta {

return {

entityId: null,

entityType: type,

entityRemove: true,

};

}

export interface EntityAction extends Action {

readonly payload?: any;

readonly meta?: EntityMeta;

}

export class EntityRemoveAction implements EntityAction {

type = ENTITY\_REMOVE\_ACTION;

readonly meta: EntityMeta;

constructor(entityType: string, id: string | string[]) {

this.meta = entityRemoveMeta(entityType, id);

}

}

export class EntityRemoveAllAction implements EntityAction {

type = ENTITY\_REMOVE\_ALL\_ACTION;

readonly meta: EntityMeta;

constructor(entityType: string) {

this.meta = entityRemoveAllMeta(entityType);

}

}

\state\utils\entity\entity-state.ts

export interface EntityState<T> {

entities: {

[id: string]: T;

};

}

state\utils\entity\entity.reducer.ts

export const initialEntityState: EntityState<any> = { entities: {} };

/\*\*

\* Higher order reducer for reusing reducer logic for multiple entities

\*

\* Utilizes entityId meta field to target entity by id in actions

\*/

export function entityReducer<T>(

entityType: string,

reducer: (state: T, action: Action) => T

) {

return (

state: EntityState<T> = initialEntityState,

action: EntityAction

): EntityState<T> => {

let ids: string[];

let partitionPayload = false;

if (

action.meta &&

action.meta.entityType === entityType &&

action.meta.entityId !== undefined

) {

ids = [].concat(action.meta.entityId);

// remove selected entities

if (action.meta.entityRemove) {

if (action.meta.entityId === null) {

return initialEntityState;

} else {

let removed = false;

const newEntities = Object.keys(state.entities).reduce((acc, cur) => {

if (ids.includes(cur)) {

removed = true;

} else {

acc[cur] = state.entities[cur];

}

return acc;

}, {});

return removed ? { entities: newEntities } : state;

}

}

partitionPayload =

Array.isArray(action.meta.entityId) && Array.isArray(action.payload);

} else {

ids = Object.keys(state.entities);

}

const entityUpdates: { [id: string]: T } = {};

for (let i = 0; i < ids.length; i++) {

const id = ids[i];

const subAction = partitionPayload

? { ...action, payload: action.payload[i] }

: action;

const newState = reducer(state.entities[id], subAction);

if (newState) {

entityUpdates[id] = newState;

}

}

if (Object.keys(entityUpdates).length > 0) {

return {

...state,

entities: { ...state.entities, ...entityUpdates },

};

}

return state;

};

}

state\utils\entity\entity.selectors.ts

import { EntityState } from './entity-state';

export function entitySelector<T>(state: EntityState<T>, id: string): T {

return state.entities[id] || undefined;

}

import \* as StateEntitySelectors from './entity.selectors';

export { StateEntitySelectors };

state\utils\loader\loader.action.ts

export const LOADER\_LOAD\_ACTION = '[LOADER] LOAD';

export const LOADER\_FAIL\_ACTION = '[LOADER] FAIL';

export const LOADER\_SUCCESS\_ACTION = '[LOADER] SUCCESS';

export const LOADER\_RESET\_ACTION = '[LOADER] RESET';

export interface LoaderMeta {

entityType: string;

loader: {

load?: boolean;

error?: any;

success?: boolean;

};

}

export interface LoaderAction extends Action {

readonly payload?: any;

readonly meta?: LoaderMeta;

}

export function loadMeta(entityType: string): LoaderMeta {

return {

entityType: entityType,

loader: {

load: true,

},

};

}

export function failMeta(entityType: string, error?: any): LoaderMeta {

return {

entityType: entityType,

loader: {

error: error ? error : true,

},

};

}

export function successMeta(entityType: string): LoaderMeta {

return {

entityType: entityType,

loader: {

success: true,

},

};

}

export function resetMeta(entityType: string): LoaderMeta {

return {

entityType: entityType,

loader: {},

};

}

export class LoaderLoadAction implements LoaderAction {

type = LOADER\_LOAD\_ACTION;

readonly meta: LoaderMeta;

constructor(entityType: string) {

this.meta = loadMeta(entityType);

}

}

export class LoaderFailAction implements LoaderAction {

type = LOADER\_FAIL\_ACTION;

readonly meta: LoaderMeta;

constructor(entityType: string, error?: any) {

this.meta = failMeta(entityType, error);

}

}

export class LoaderSuccessAction implements LoaderAction {

type = LOADER\_SUCCESS\_ACTION;

readonly meta: LoaderMeta;

constructor(entityType: string) {

this.meta = successMeta(entityType);

}

}

export class LoaderResetAction implements LoaderAction {

type = LOADER\_RESET\_ACTION;

readonly meta: LoaderMeta;

constructor(entityType: string) {

this.meta = resetMeta(entityType);

}

}

state\utils\loader\loader-state.ts

export interface LoaderState<T> {

loading?: boolean;

error?: boolean;

success?: boolean;

value?: T;

}

state\utils\loader\loader.reducer.ts

export const initialLoaderState: LoaderState<any> = {

loading: false,

error: false,

success: false,

value: undefined,

};

/\*\*

\* Higher order reducer that adds generic loading flag to chunk of the state

\*

\* Utilizes "loader" meta field of actions to set specific flags for specific

\* action (LOAD, SUCCESS, FAIL, RESET)

\*/

export function loaderReducer<T>(

entityType: string,

reducer?: (state: T, action: Action) => T

): (state: LoaderState<T>, action: LoaderAction) => LoaderState<T> {

return (

state: LoaderState<T> = initialLoaderState,

action: LoaderAction

): LoaderState<T> => {

if (

action.meta &&

action.meta.loader &&

action.meta.entityType === entityType

) {

const entity = action.meta.loader;

if (entity.load) {

return {

...state,

loading: true,

value: reducer ? reducer(state.value, action) : state.value,

};

} else if (entity.error) {

return {

...state,

loading: false,

error: true,

success: false,

value: reducer ? reducer(state.value, action) : undefined,

};

} else if (entity.success) {

return {

...state,

value: reducer ? reducer(state.value, action) : action.payload,

loading: false,

error: false,

success: true,

};

} else {

// reset state action

return {

...initialLoaderState,

value: reducer

? reducer(initialLoaderState.value, action)

: initialLoaderState.value,

};

}

}

if (reducer) {

const newValue = reducer(state.value, action);

if (newValue !== state.value) {

return { ...state, value: newValue };

}

}

return state;

};

}

state\utils\loader\loader.selectors.ts

export function loaderValueSelector<T>(state: LoaderState<T>): T {

return state.value;

}

export function loaderLoadingSelector<T>(state: LoaderState<T>): boolean {

return state.loading;

}

export function loaderErrorSelector<T>(state: LoaderState<T>): boolean {

return state.error;

}

export function loaderSuccessSelector<T>(state: LoaderState<T>): boolean {

return state.success;

}

state\utils\loader\loader.helpers.ts

import { EntityLoadAction } from '../entity-loader/entity-loader.action';

import { MonoTypeOperatorFunction } from 'rxjs';

export function ofLoaderLoad(

entityType: string

): MonoTypeOperatorFunction<EntityLoadAction> {

return filter(

(action: EntityLoadAction) =>

action.meta &&

action.meta.loader &&

action.meta.entityType === entityType &&

action.meta.loader.load

);

}

export function ofLoaderFail(

entityType: string

): MonoTypeOperatorFunction<EntityLoadAction> {

return filter(

(action: EntityLoadAction) =>

action.meta &&

action.meta.loader &&

action.meta.entityType === entityType &&

action.meta.loader.error

);

}

export function ofLoaderSuccess(

entityType: string

): MonoTypeOperatorFunction<EntityLoadAction> {

return filter(

(action: EntityLoadAction) =>

action.meta &&

action.meta.loader &&

action.meta.entityType === entityType &&

!action.meta.loader.load &&

!action.meta.loader.error

);

}