**Let’s explore what is the problem with the redux toolkit?**

Consider **Module 6 Video Website** , when home page is load, then it call two network api to fetch videos and tags. When we go single video then networks will be called to fetch the single video related network. Again when we go in home page, again network calls, again when we go the same video again it call some networks. But this is unnecessary network calls as first time we have already called networks to fetch videos and tags data. This is too much unnecessary networks calls. This is the problem.

**Solution:** if we can cache the videos and tags when first time networks call. Similarly for indivisual video.

RTK Query gives us caching benefits.

**RTK Query:**

RTK Query is **data fetching & caching tool**. It gives us caching mechanism.

**Features:**

* Tracking loading, error, success state
* Avoid duplicate request for same data
* Optimistic updates to make the UI feels faster
* Managing cache lifetimes as the user interacts with the UI
* RTK query sync client with the server smoothly

In RTK Query we will create an apiSlice. Which we configure. Then this slice we will add into store.

Here is simple apiSlice configuration:

apiSlice using **createAPI** method which will take an object. Inside object we configure this slice.

**reducerPath, baseQuery** using fetchBaseQuery. fetchBaseQuery method take an object. Which configure the baseURL. endPoints, which is a builder function which take an object. Inside this builder we add our api method using builder.query(). This take and object. Inside query property we provide endPoints by using a callback. Bulder.query means get request.

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";

export const apiSlice = createApi({

  reducerPath: "api",

  baseQuery: fetchBaseQuery({

    baseUrl: "http://localhost:9000",

  }),

  endpoints: (builder) => ({

    getVideos: builder.query({

      query: () => "/videos",

    }),

    getVideo: builder.query({

      query: (videoId) => `/videos/${videoId}`,

    }),

    getRelatedVideos: builder.query({

      query: ({ id, title }) => {

        const tags = title.split(" ");

        let queryString = tags.map((tag) => `title\_like=${tag}`).join("&");

        queryString = `/videos?${queryString}&id\_ne=${id}&\_limit=4`;

        return queryString;

      },

    }),

  }),

});

// export all the data fetching functions

export const { useGetVideosQuery, useGetVideoQuery, useGetRelatedVideosQuery } =

  apiSlice;

store:

import { configureStore } from "@reduxjs/toolkit";

import { apiSlice } from "../features/api/apiSlice";

export const store = configureStore({

  reducer: {

    [apiSlice.reducerPath]: apiSlice.reducer,

  },

  middleware: (getDefaultMiddlewares) =>

    getDefaultMiddlewares().concat(apiSlice.middleware),

});

Note: RTK cache fetch value till 1 minute. When 1 minute will over, then it automatically refetch. 1 minute is default time. We can customized it.

keepUnusedDataFor: property takes times as second.

    getVideos: builder.query({

      query: () => "/videos",

      // by default 60s.

      keepUnusedDataFor: 5,

    }),

**Configuration:**

Second parameter we can pass an object as configuration parameter.

**refetchOnMountOrArgChange:**

**pollingInterval:** millisecond time interval. Refetch at a certain interval.

  const {

    data: videos,

    isError,

    isLoading,

  } = useGetVideosQuery(undefined, {

    refetchOnMountOrArgChange: true,

  });

**Skip:** we can pass true, or false. Or decimal value. To stop the api call.

Skip: true; means it can not call api. When this value is true then it only call api. Now we can handle api call.

We have more features:

  const { data: videos, isError, isLoading, refetch, isFetching, isSuccess } = useGetVideosQuery();

refetch, isSuccess, isFetching.

Above was all about **query.** Means GET request only. Mutation is below.

**POST Request:**

  const [addVideo, { data: video, isError, isLoading }] = useAddVideoMutation();

mutation will not called automatically when component is render. It just return **addVideo()** function, and an object. Need to call **addVideo()** function by passing data.

**Note:** when we add new video. It will not show in home page immediate. Because home page shows cache value only. When interval is over then it automatically refetch. But if we want to refetch immediately when add new video is added. We need to do that.

RTK Query has memory where it cache data. We can control this cache. We can set tags for each cache. Then we can invalidate any cache memory using tags.

1st add some tags to configure. Then uses these tags into any query or mutation.

**tagTypes add:** some whitelisted tags

export const apiSlice = createApi({

  reducerPath: "api",

  baseQuery: fetchBaseQuery({

    baseUrl: "http://localhost:9000",

  }),

  tagTypes: ["videos"],

**providesTags:** so that we can identify from others place to invalidate this cache memory.

    getVideos: builder.query({

      query: () => "/videos",

      providesTags: ["videos"],

      // by default 60s.

    }),

**invalidatesTags:** invalidate tags when this addVideo() mutation is called.

    addVideo: builder.mutation({

      query: (data) => ({

        url: "/videos",

        method: "POST",

        body: data,

      }),

      invalidatesTags: ["videos"],

    }),

**Invalidate specific cache:**

**tagsType:**

export const apiSlice = createApi({

  reducerPath: "api",

  baseQuery: fetchBaseQuery({

    baseUrl: "http://localhost:9000",

  }),

  tagTypes: ["videos", "video"],

**providesTags:**

    getVideo: builder.query({

      query: (videoId) => `/videos/${videoId}`,

      providesTags: (result, error, arg) => [

        {

          type: "video",

          id: arg.id,

        },

      ],

    }),

Here provides tags cache dynamically with id.

**InvalidatesTags:**

    editVideo: builder.mutation({

      query: ({ id, data }) => ({

        url: `/videos/${id}`,

        method: "PUT",

        body: data,

      }),

      invalidatesTags: (result, error, arg) => [

        "videos",

        {

          type: "video",

          id: arg.id,

        },

      ],

    }),

Here dynamically we have invalidated cache based on id.

Here result means after editVideois done. Arg means argument passed through the query.

**Full CRUD Operations:**

**Store.js**

import { configureStore } from "@reduxjs/toolkit";

import { apiSlice } from "../features/api/apiSlice";

export const store = configureStore({

  reducer: {

    [apiSlice.reducerPath]: apiSlice.reducer,

  },

  middleware: (getDefaultMiddlewares) =>

    getDefaultMiddlewares().concat(apiSlice.middleware),

});

**apiSlice.js**

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";

export const apiSlice = createApi({

  reducerPath: "api",

  baseQuery: fetchBaseQuery({

    baseUrl: "http://localhost:9000",

  }),

  tagTypes: ["videos", "video", "relatedVideos"],

  endpoints: (builder) => ({

    getVideos: builder.query({

      query: () => "/videos",

      providesTags: ["videos"],

      // by default 60s.

    }),

    getVideo: builder.query({

      query: (videoId) => `/videos/${videoId}`,

      providesTags: (result, error, arg) => [

        { type: "video", id: arg.videoId },

      ],

    }),

    getRelatedVideos: builder.query({

      query: ({ id, title }) => {

        const tags = title.split(" ");

        let queryString = tags.map((tag) => `title\_like=${tag}`).join("&");

        queryString = `/videos?${queryString}&id\_ne=${id}&\_limit=4`;

        return queryString;

      },

      providesTags: (result, error, arg) => [

        { type: "relatedVideos", id: arg.id },

      ],

    }),

    addVideo: builder.mutation({

      query: (data) => ({

        url: "/videos",

        method: "POST",

        body: data,

      }),

      invalidatesTags: ["videos"],

    }),

    editVideo: builder.mutation({

      query: ({ id, data }) => ({

        url: `/videos/${id}`,

        method: "PUT",

        body: data,

      }),

      invalidatesTags: (result, error, arg) => [

        "videos",

        { type: "video", id: arg.id },

        { type: "relatedVideos", id: arg.id },

      ],

    }),

    deleteVideo: builder.mutation({

      query: (id) => ({

        url: `/videos/${id}`,

        method: "DELETE",

      }),

      invalidatesTags: (result, error, arg) => ["videos"],

    }),

  }),

});

// export all the data fetching functions

export const {

  useGetVideosQuery,

  useGetVideoQuery,

  useGetRelatedVideosQuery,

  useAddVideoMutation,

  useEditVideoMutation,

  useDeleteVideoMutation,

} = apiSlice;