**Redux Toolkit Query**

* RTK Query is a powerful data fetching and caching tool
* It is designed to simplify common cases for loading data in a web application, eliminating the need to hand-write data fetching & caching logic yourself.
* RTK Query is an optional addon included in the Redux Toolkit package, and its functionality is built on top of the other APIs in Redux Toolkit.
* Prevent unnecessary api request,
* Optimize performance issue

**Benefits of RTK Query**

* Easy setup
* Reduce boilerplate code
* Automatic caching
* Sophisticated caching options
* Automatic request retries
* Integrated error handling
* Built-in TypeScript support

**Create Api slice**

* API Slice will handle all api request and endpoints
* We should create our api slice into feature folder   
    
  export const apiSlice = createApi({  
   reducerPath : ‘api’,  
   baseQuery : fetchBaseQuery( { baseUrl : ‘api url’ } ),  
   tagTypes : [‘Post’,’Like’],  
   endpoints : builder => ({})  
  });

**Setup Store**

* Now setup our store   
    
  export const store = configureStore({   
   reducer : {  
   [apiSlice.reducerPath] : apiSlice.reducer   
   },   
   middleware : getDefaultMiddleware =>   
   getDefaultMiddleware.concat(apiSlice.middleware),  
   devTools : true   
  });

**Mutation & Query**

* Query handle get request endpoints
* Mutation handle others request like POST, PUT, PATCH, DELETE

**Endpoint Query**

* We can manage any get request by using query

endpoints : ( builder ) => ({  
 getPosts : builder.query({   
 query : () => ‘/posts’  
 })  
 })

* Now export custom manual hook   
  export const { useGetPostsQuery } = apiSlice ;
* Get query hook   
  const { data, isLoading, isError, isSuccess, error } = useGetPostsQuery();

**Endpoint Mutation**

* We can manage any get request by using query

endpoints : ( builder ) => ({  
 createPosts : builder.mutation({   
 query : (params) => ‘/posts’  
 })  
 })

* Now export custom manual hook   
  export const { useGetPostsMutation } = apiSlice ;
* Get query hook   
  const [ createPost, { data, isLoading, isError, isSuccess, error }] = useGetPostsMutation();

**Inject Endpoints**

* We can use our apis in our slice   
    
  export const postSlice = apiSlice.injectendpoints({  
   endpoints : builder => {  
     
   }  
  });

**Entity Adapter**

* Efficiently manages entity state with normalized data structures
* Provides a set of functions to manage entity state, such as addOne, addMany, updateOne, and removeOne
* Generates a set of reducers and selectors that can be used directly in a Redux store
* Can be customized with additional options, such as providing a custom ID selector function or a comparison function for sorting entities
* **Define a adapter**   
  const postAdapter = createEntityAdapter();

**Cashing Control**

* Caching invalidation is the process of conditional state updating
* To use cache invalidation we must provide Tags and first define some tags types with array   
  tagsType : [‘Posts’,’Post’,’Cat’]
* Then provided this tag to an query   
  providedTags : [‘Post’]
* Now Invalidation   
  invalidatesTags : [‘Post’]
* Set specific cache dynamic   
  invalidatesTags : (result, error, arg) => [“Posts” , { type : Post , id : arg.id }];