**Why we need redux toolkit?**

There are some problems while using react-redux. Large scale app making very problem in react-redux.

**Redux requires too much boilerplate code**

* Actions types
* Action creators
* Switch statements in reducer
* Too much modular file structure

**Other packages need to be installed**

* Redux-thunk
* Redux-devtools

In large scale application, it requires many modular file that is too irrating for developer experience

**Biggest problem:** immutable state update

But we can solved this problem using immer package

All these problem will be solved using **Redux Toolkit**

Less modular file, less boilerplate, opinionated, batteries-included toolset for efficient redux development. Mainly for large application redux toolkit gives us less code write experiences

**Raw Redux-toolkit**

Think as features. Consider whole app as pizza. Then each feature will be a slice. We will work for each slice.

Require createSlice method from toolkit. Then create a counter slice using createSlice method by passing an object. Which object takes name, initialState, and reducers.

**Note:** reducers object takes all the reducer that before we do using switch case. Now think each case as a reducer function. Which take state, and action.

Two benefits:

Each function works as a reducer function. And no need to return anything. Second benefit is, we can mute state. no need to check mutable or immutable. Behind the scene it uses immer to handle mute state.

Then export the counter slice as default reducer. Since createSlice return a single reducer even we write many reducers inside counter slice.

Below is code

Note: mainly by creating counterSlice. We do the two task. One is for reducer and another is for actions

**Create reducer and actions:**

|  |
| --- |
| // import createSlice from redux toolkit  const { createSlice } = require("@reduxjs/toolkit");  // initial state  const initialState = {    count: 0,  };  // create a slice  //note: createSlice return a reducer.  const counterSlice = createSlice({    name: "counter",    initialState,    reducers: {      increment: (state, action) => {        state.count++;      },      decrement: (state, action) => {        state.count--;      },    },  });  // exports reducer and actions  // note: createSlice return an actions intelligently. here export actions as named exports  module.exports = counterSlice.reducer;  module.exports.counterActions = counterSlice.actions; |

**Configure Store:**

Now configure the store using configureStore method. This method take an object then pass a reducer object. This object contains all the reducer

|  |
| --- |
| // import configureStore method  const { configureStore } = require("@reduxjs/toolkit");  // import all reducers here  const counterReducer = require("../features/counter/counterSlice");  // configure store  const store = configureStore({    reducer: {      counter: counterReducer,    },  });  // export store  module.exports = store; |

**Add middleware:**

|  |
| --- |
| // configure store  const store = configureStore({    reducer: {      counter: counterReducer,      dynamicCounter: dynamicCounter,    },    middleware: (defaultMiddlewares) => {      return defaultMiddlewares().concat(createLogger());    },  }); |

**We can pass payload data:**

store.dispatch(dynamicCounterActions.increment(3));

store.dispatch(dynamicCounterActions.increment(7));

**we can receive the passed data:**

// create a slice

//note: createSlice return a reducer.

const dynamicCounterSlice = createSlice({

  name: "dynamicCounter",

  initialState,

  reducers: {

    increment: (state, action) => {

      state.count += action.payload;

    },

    decrement: (state, action) => {

      state.count -= action.payload;

    },

  },

});

**Note:** interesting things is that. When we dispatch an action. All the reducer get this actions. Only that action is perform is matching with the action type. In react-redux we can just add a case inside switch case.

But in redux toolkit we need to use extrareducers property inside create slice.

**Redux-toolkit and thunk**

We have ‘createSlice’ method, similarly we have ‘createAsyncThunk’ method. It takes first parameter as an action name and then pass an async function.

const { createSlice, createAsyncThunk } = require("@reduxjs/toolkit");

// intial state

const initialState = {

  loading: false,

  posts: [],

  error: "",

};

// create thunk slice

const fetchPosts = createAsyncThunk("posts/fetchPosts", async () => {

  const response = await fetch("https://jsonplaceholder.typicode.com/posts");

  const posts = await response.json();

  return posts; // it return a promise.

  // this promise we can catch inside extrareducers as events

  // manually no need to handle error here. handle error inside extrareducers

  // promise can have three case:

  //1. pending 2. fullfill 3. rejected

});

// posts slice

const postsSlice = createSlice({

  name: "posts",

  initialState,

  extraReducers: (builder) => {

    builder.addCase(fetchPosts.pending, (state, action) => {

      state.loading = true;

      state.error = "";

    });

    builder.addCase(fetchPosts.fulfilled, (state, action) => {

      state.loading = false;

      state.error = "";

      state.posts = action.payload;

    });

    builder.addCase(fetchPosts.rejected, (state, action) => {

      state.loading = false;

      state.error = action.error;

      state.posts = [];

    });

  },

});

module.exports = postsSlice.reducer;

module.exports.fetchPosts = fetchPosts;