Redux 3.1_CW Exercises

setup

```
npx create-react-app my-app --template redux
```

But we'll be using codesandbox for the session.

codesandbox to begin: https://codesandbox.io/s/react-redux-ex01-h4ct6

read initial posts in UI

understanding

There are few core concepts to Redux and Redux Toolkit. In this exercise we will learn about:

1. Provider: Similar to the AuthProviders and others we have been using throughout the camp. It uses context to provide the store to the React App.

2. Store: This is where all your state resides.

    1. This is global. So keep things here which needs to be global.
    2. You can use local `useState()` in apps. And it is advisable to keep local states in that. For example, onChange of an input element shouldn't be put in the global state. https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

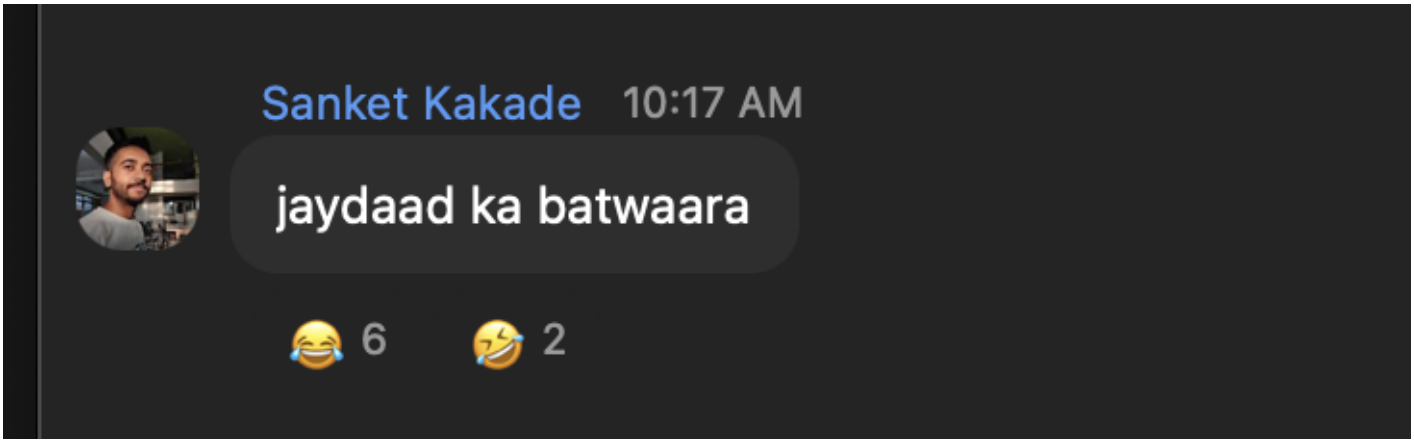
3. Slice: A piece of the bigger state. The convention is to take one feature and put it in one slice using `createSlice()`.

    For example, slices in a social media app would be:

    - users
    - posts
    - notifications
    - trending

    This also means that your store is nothing but a collection of slices.

Think of slices as a manager for a bigger object. It represents a small part and manages.



1. Selector: If you put everything in a global store it gets really big and then any update on it will run render function for the entire app, which will be a costly experience for large apps.

a note on performance - For smaller apps, this cost is negligible and that's why React experts, including me ;), warn against premature optimization. You should think about this when your app is visibly slow. - Redux toolkit is suited for large apps and thus when you're working in a startup, use it for main apps which would grow really fast. But if you're making side projects, or side apps, you can go with `useReducer + useContext` combo.

ex01: configure the redux store

challenge

Create a file named `store.js` and set up the Redux store using `configureStore` from Redux Toolkit.

solution

```
import { configureStore } from '@reduxjs/toolkit'
import { postSlice } from '../features/posts/postSlice'

export default configureStore({
  reducer: {
    posts: postSlice.reducer,
  },
})
```

COPY

understanding

This code sets up a storage area for your application's data using Redux. It links this storage area to the `postSlice` reducer, so any actions related to posts will be managed by this store. This store is

essential for keeping track of data that needs to be shared across various parts of your application.

ex02: creating a post slice

challenge

1. Create a file named `postSlice.js`. This file is where you define a "slice" of your Redux state, specifically for managing posts in your application.
2. Inside `postSlice.js`, import `createSlice` from Redux Toolkit. This function helps you create a slice with an initial state and reducer actions.
3. Use `createSlice` to define the `postSlice`. Give it a name ('posts') and provide an initial state, which contains an array of posts.
4. Define additional actions (reducers) inside the `reducers` object, but in this solution, it's left empty. You can add actions here to modify the state as needed.

solution

```
// postSlice.js
import { createSlice } from '@reduxjs/toolkit'

export const postSlice = createSlice({
  name: 'posts',
  initialState: {
    posts: [
      {
        postID: 'p1201',
        caption: 'learning redux',
        likes: 22,
        user: {
          userID: 'u1234',
          name: 'tanay',
        },
      },
      {
        postID: 'p1202',
        caption: "it's frustrating to begin",
        likes: 24,
        user: {
          userID: 'u1234',
          name: 'tanay',
        },
      },
    ],
  },
  reducers: {},
})

export default postSlice.reducer
```

COPY

understanding

This code sets up a slice of your application's state to manage posts. It starts with some initial posts in the state. You can later add actions to modify this state when needed, such as adding new posts or updating existing ones. This slice of state is ready to be integrated into your Redux store and used by your React components.

ex03: creating the `Posts` component

challenge

Create a React component named `Posts.js` that displays a list of posts. Use the `useSelector` hook from React-Redux to access the posts data from the Redux store.

Each post should display its caption and the number of likes.

solution

```
import React from 'react'
import { useSelector } from 'react-redux'

export default function Posts() {
  const posts = useSelector((state) => state.posts)

  return (
    <div>
      <div>
        {posts.posts.map((post) => (
          <article key={post.postID} className='post'>
            <div className='caption'> {post.caption} </div>
            <div className='likes'>{post.likes} ❤️ </div>
          </article>
        ))}
      </div>
    </div>
  )
}
```

COPY

ex04: setting up the provider

challenge

In your React application, set up the `Provider` from React-Redux to connect your Redux store to your application. Ensure that your Redux store is properly configured using `configureStore`. Use the provided `App` component as the root of your application.

solution

```
import { StrictMode } from 'react'
import ReactDOM from 'react-dom'

import { Provider } from 'react-redux'

import App from './App'
import store from './app/store'

const rootElement = document.getElementById('root')
ReactDOM.render(
  <StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </StrictMode>,
  rootElement,
)
```

COPY

entire solution

https://codesandbox.io/s/react-redux-ex01-sol-22434