Example : using Redux Toolkit with React to create a basic counter app

1. First, create react app and install the required packages

   ```
   npx create-react-app myreduxapp
   npm install @reduxjs/toolkit react-redux
   ```

2. In Redux Toolkit, a "slice" represents a part of your state and includes the reducer logic. Create a slice for the counter.

   src/redux/counterSlice.js

   ```
   import { createSlice } from '@reduxjs/toolkit';
   const initialState = { value: 0 };
   const counterSlice = createSlice({
                                        name: 'counter',
                                        initialState,
                                        reducers: {
                                          increment: (state) => {
                                            state.value += 1;
                                          },
                                          decrement: (state) => {
                                            state.value -= 1;
                                          },
                                          reset: (state) => {
                                            state.value = 0;
                                          }
                                        }
                                    });
   export const { increment, decrement, reset } = counterSlice.actions;
   export default counterSlice.reducer;
   ```

3. Now, set up the Redux store and include the counter slice reducer.

   src/redux/store.js

   ```
   import { configureStore } from '@reduxjs/toolkit';
   import counterReducer from './counterSlice';
   export const store = configureStore({
                                          reducer: {
                                            counter: counterReducer
    }});
   ```

4. Next, create a simple React component to display the counter and buttons to increment, decrement, and reset.
   src/App.js

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement, reset } from './redux/counterSlice';

function App() {
  const count = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div style={{ textAlign: 'center' }}>
      <h1>Counter: {count}</h1>
      <button onClick={() => dispatch(increment())}>Increment</button>
      <button onClick={() => dispatch(decrement())}>Decrement</button>
      <button onClick={() => dispatch(reset())}>Reset</button>
    </div>
  );
}
export default App;
```

5. Wrap your root component with the Provider component from react-redux to make the Redux store available to your React components.
   src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './redux/store';
ReactDOM.render(
  <Provider store={store}>
                                <App />
  </Provider>,
  document.getElementById('root')
);
```

State changes using Redux DevTools

- The good news is that @reduxjs/toolkit automatically includes the setup for Redux DevTools in development mode, so you don't need to do much. It will automatically connect the Redux store to the DevTools.
- In src/redux/store.js, no additional configuration is needed, because configureStore already includes support for the DevTools. Only you need to install the Redux DevTools extension in your browser.
- This makes debugging and visualizing your state changes super easy!

6. Now, you can run your app by using:

   npm start

Example: Simple to-do list app using Redux Toolkit with React

1. First, create react app and install the required packages

   npx create-react-app myreduxapp

   npm install @reduxjs/toolkit react-redux

2. Create a slice to manage the to-do list state, actions, and reducer logic.

   src/redux/todoSlice.js

```
import { createSlice } from '@reduxjs/toolkit';
const initialState = [];
const todoSlice = createSlice({
        name: 'todos',
        initialState,
        reducers: {
                addTodo: (state, action) => {
                                const newTodo = {
                                                id: Date.now(),
                                                text: action.payload,
                                                completed: false,
                                };
                                state.push(newTodo);
                },
                toggleTodo: (state, action) => {
                  const todo = state.find((todo) => todo.id ===
              action.payload);
                   if (todo) {
                                        todo.completed = !todo.completed;
                   }
                },
                deleteTodo: (state, action) => {
                        return state.filter((todo) => todo.id !==
              action.payload);
                },
  },
});
export const { addTodo, toggleTodo, deleteTodo } = todoSlice.actions;
export default todoSlice.reducer;
```

3. Set up the Redux store and include the todoSlice reducer.
   src/redux/store.js

```
import { configureStore } from '@reduxjs/toolkit';
import todoReducer from './todoSlice';

export const store = configureStore({
                                        reducer: {
                                          todos: todoReducer,
                                        },
});
```

4. Create TodoList Component to display the list of to-dos and buttons to toggle and delete them.
   src/components/TodoList.jsx

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { toggleTodo, deleteTodo } from '../redux/todoSlice';

function TodoList() {
  const todos = useSelector((state) => state.todos);
  const dispatch = useDispatch();

  return (
    <div style={{ display: 'flex', justifyContent: 'center' }}>
      <ul style={{ listStyleType: 'none', padding: 0 }}>
        {todos.map((todo) => (
          <li
            key={todo.id}
            style={{
              textDecoration: todo.completed ? 'line-through' : 'none',
              display: 'flex',
              justifyContent: 'space-between',
              alignItems: 'center',
              width: '300px',
              padding: '10px',
              border: '1px solid #ddd',
```

```
            marginBottom: '10px',
            borderRadius: '5px',
          }}
        >
          <span>{todo.text}</span>
          <div>
            <button onClick={() => dispatch(toggleTodo(todo.id))}>
              {todo.completed ? 'Undo' : 'Complete'}
            </button>
            <button onClick={() =>
dispatch(deleteTodo(todo.id))}>Delete</button>
          </div>
        </li>
      ))}
    </ul>
  </div>
 );
}

export default TodoList;
```

5. create AddTodo Component to handle adding new to-dos to the list.
   src/components/AddTodo.jsx

```
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';
import { addTodo } from '../redux/todoSlice';

function AddTodo() {
  const [input, setInput] = useState('');
  const dispatch = useDispatch();

  const handleSubmit = (e) => {
    e.preventDefault();
    if (input.trim()) {
      dispatch(addTodo(input));
      setInput('');
    }
```

```
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={input}
        onChange={(e) => setInput(e.target.value)}
        placeholder="Add new todo"
      />
      <button type="submit">Add Todo</button>
    </form>
  );
}

export default AddTodo;
```

6. Create the main App component that renders the AddTodo and TodoList components.

   src/App.js

```
import React from 'react';
import AddTodo from './components/AddTodo';
import TodoList from './components/TodoList';

function App() {
  return (
    <div style={{ textAlign: 'center' }}>
      <h1>Todo List</h1>
      <AddTodo />
      <TodoList />
    </div>
  );
}

export default App;
```

7. Wrap your app with the Redux Provider to make the store available to your components.
   src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './redux/store';

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

8. Now, run your app using:
   npm start

Example: Fetch data from an API using Redux Toolkit

1. First, create react app and install the required packages

   npx create-react-app myreduxapp
   npm install @reduxjs/toolkit react-redux axios

2. Now use Redux Toolkit's createAsyncThunk to handle async API calls.
   src/redux/userSlice.js

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

// Create async thunk to fetch users from an API
export const fetchUsers = createAsyncThunk('users/fetchUsers', async ()
=> {
  const response = await
axios.get('https://jsonplaceholder.typicode.com/users');
  return response.data;
});

// Create a slice for user data
const userSlice = createSlice({
  name: 'users',
  initialState: {
                users: [],
                status: 'idle', // idle | loading | succeeded | failed
                error: null,
  },
  reducers: {},
  extraReducers: (builder) => {
              builder
                .addCase(fetchUsers.pending, (state) => {
                  state.status = 'loading';
                })
                .addCase(fetchUsers.fulfilled, (state, action) => {
                  state.status = 'succeeded';
                  state.users = action.payload;
                })
```

```
      .addCase(fetchUsers.rejected, (state, action) => {
        state.status = 'failed';
        state.error = action.error.message;
      });
  },
});

export default userSlice.reducer;
```

3. Set up the Redux store and include the user slice reducer.
   src/redux/store.js

```
import { configureStore } from '@reduxjs/toolkit';
import userReducer from './userSlice';

export const store = configureStore({
                        reducer: {
                          users: userReducer,
                        },
});
```

4. Create UserList component to dispatch the fetchUsers action to retrieve data when the component mounts, and then display the list of users.
   src/components/UserList.jsx

```
import React, { useEffect } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { fetchUsers } from '../redux/userSlice';

function UserList() {
  const dispatch = useDispatch();
  const { users, status, error } = useSelector((state) => state.users);

  // Fetch users when the component is first rendered
  useEffect(() => {
    if (status === 'idle') {
      dispatch(fetchUsers());
    }
  }, [status, dispatch]);
```

```
  let content;
  if (status === 'loading') {
    content = <p>Loading...</p>;
  } else if (status === 'succeeded') {
    content = (
      <ul>
       {users.map((user) => (
         <li key={user.id}> {user.name} - {user.email}  </li>
       ))}
      </ul>
    );
  } else if (status === 'failed') {
    content = <p>{error}</p>;
  }
  return (
    <div>
      <h1>User List</h1>
      {content}
    </div>
  );
}
export default UserList;
```

5. Wrap everything in the App component, which renders the UserList component.
   src/App.js

```
import React from 'react';
import UserList from './components/UserList';

function App() {
  return (
    <div style={{ textAlign: 'center' }}>
      <UserList />
    </div>
  );
}

export default App;
```

6. Now, wrap the app with the Redux Provider to make the store available to your components.

   src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './redux/store';

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

7. Run your app by using:

```
npm start
```

Example : using Redux Toolkit to fetch a list of posts from an API, displays them, and includes error handling and loading states.

1. First, create react app and install the required packages

   npx create-react-app myreduxapp

   npm install @reduxjs/toolkit react-redux axios

2. use createAsyncThunk to handle the asynchronous operation of fetching posts from the API.

   src/redux/postsSlice.js

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

// Async thunk to fetch posts from the API
export const fetchPosts = createAsyncThunk('posts/fetchPosts', async () => {
  const response = await axios.get('https://jsonplaceholder.typicode.com/posts');
  return response.data;
});
const postsSlice = createSlice({
  name: 'posts',
  initialState: {
    posts: [],
    status: 'idle', // idle | loading | succeeded | failed
    error: null,
  },
  reducers: {},
  extraReducers: (builder) => {
    builder.addCase(fetchPosts.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(fetchPosts.fulfilled, (state, action) => {
        state.status = 'succeeded';
        // Add fetched posts to the array
        state.posts = action.payload;
      })
```

```
      .addCase(fetchPosts.rejected, (state, action) => {
        state.status = 'failed';
        state.error = action.error.message;
      });
  },
});
export default postsSlice.reducer;
```

3. Set Up Redux Store to include the postsSlice reducer to manage posts data.
   src/redux/store.js

```
import { configureStore } from '@reduxjs/toolkit';
import postsReducer from './postsSlice';
export const store = configureStore({
  reducer: {
    posts: postsReducer,
  },
});
```

4. Create PostsList component dispatch the fetchPosts action to retrieve data and display it. It will handle the loading state, success, and failure cases.
   src/components/PostsList.jsx

```
import React, { useEffect } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { fetchPosts } from '../redux/postsSlice';

function PostsList() {
  const dispatch = useDispatch();
  const { posts, status, error } = useSelector((state) => state.posts);

  // Fetch posts when the component is first rendered
  useEffect(() => {
    if (status === 'idle') {
      dispatch(fetchPosts());
    }
  }, [status, dispatch]);
  let content;
```

```
  if (status === 'loading') {
    content = <p>Loading...</p>;
  } else if (status === 'succeeded') {
    content = (
      <ul style={{ listStylePosition: 'inside', textAlign: 'center', padding: 0 }}>
       {posts.map((post) => (
         <li key={post.id} style={{ marginBottom: '10px' }}>
           <strong>{post.title}</strong>
           <p>{post.body}</p>
         </li>
       ))}
      </ul>
    );
  } else if (status === 'failed') {
    content = <p>{error}</p>;
  }
  return (
    <div>
      <h1 style={{ textAlign: 'center' }}>Posts</h1>
      {content}
    </div>
  );
}
export default PostsList;
```

5. Set Up the App Component to wrap everything and render the PostsList component.

   src/App.js

```
import React from 'react';
import PostsList from './components/PostsList';
function App() {
  return (
    <div>
      <PostsList />
    </div>
  );}
export default App;
```

6. Integrate Redux with React by wrapping the App component with the Provider to make the Redux store available to all components.
   src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './redux/store';

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

7. Now, run the app using:

```
npm start
```