Core Redux concepts like \*\*state management\*\*, \*\*actions\*\*, and \*\*reducers\*\* using the modern `@reduxjs/toolkit` approach.

---

## 🔧 Project Purpose

To demonstrate \*\*Redux\*\* in React through a simple \*\*ToDo application\*\*, where a user can:

- Add a task

- Toggle task completion

- Delete a task

---

## 🏗️ Folder & File Overview

```

redux-todo-app/

├── src/

│ ├── app/

│ │ └── store.js ✅ Sets up Redux store

│ ├── features/

│ │ └── todos/

│ │ ├── TodoSlice.js ✅ Contains state logic (slice)

│ │ └── TodoList.js ✅ React component using the slice

│ ├── App.js ✅ Main app entry

│ └── index.js ✅ Connects Redux store to React

```

---

## 🧠 Step-by-Step Explanation

---

### 🔹 1. `store.js` – Configures Redux Store

```js

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

import todoReducer from '../features/todos/TodoSlice';

export const store = configureStore({

reducer: {

todos: todoReducer,

},

});

```

✅ \*\*Purpose\*\*:

- Creates the central \*\*Redux Store\*\*

- Uses `configureStore` from Redux Toolkit (simpler than legacy `createStore`)

- Registers `todos` reducer

---

### 🔹 2. `TodoSlice.js` – Contains State + Actions

```js

import { createSlice } from '@reduxjs/toolkit';

let nextId = 1;

const todoSlice = createSlice({

name: 'todos',

initialState: [],

reducers: {

addTodo: (state, action) => {

state.push({ id: nextId++, text: action.payload, completed: false });

},

toggleTodo: (state, action) => {

const todo = state.find(t => t.id === action.payload);

if (todo) todo.completed = !todo.completed;

},

deleteTodo: (state, action) => {

return state.filter(t => t.id !== action.payload);

},

},

});

export const { addTodo, toggleTodo, deleteTodo } = todoSlice.actions;

export default todoSlice.reducer;

```

✅ \*\*Purpose\*\*:

- This is the \*\*slice\*\*: it contains the state (`initialState`) and reducers.

- `createSlice` automatically creates action types and creators.

- Three reducer functions:

- `addTodo`: Adds a new todo object to the array.

- `toggleTodo`: Finds todo by ID and toggles its `completed` status.

- `deleteTodo`: Removes the todo by filtering it out of the array.

---

### 🔹 3. `TodoList.js` – UI Component Using Redux State

```js

import React, { useState } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { addTodo, toggleTodo, deleteTodo } from './TodoSlice';

const TodoList = () => {

const [text, setText] = useState('');

const todos = useSelector(state => state.todos); // Get todos from Redux store

const dispatch = useDispatch(); // Used to dispatch actions

const handleAdd = () => {

if (text.trim()) {

dispatch(addTodo(text));

setText('');

}

};

return (

<div>

<h2>Todo List</h2>

<input

value={text}

onChange={e => setText(e.target.value)}

placeholder="Enter todo"

/>

<button onClick={handleAdd}>Add</button>

<ul>

{todos.map(todo => (

<li key={todo.id}>

<span

style={{

textDecoration: todo.completed ? 'line-through' : 'none',

cursor: 'pointer',

}}

onClick={() => dispatch(toggleTodo(todo.id))}

>

{todo.text}

</span>

<button onClick={() => dispatch(deleteTodo(todo.id))}>❌</button>

</li>

))}

</ul>

</div>

);

};

export default TodoList;

```

✅ \*\*Purpose\*\*:

- Uses `useSelector()` to \*\*read todos from Redux store\*\*

- Uses `useDispatch()` to \*\*send actions to the store\*\*

- Provides buttons and input to interact with the todo state:

- Add a todo

- Toggle complete by clicking on it

- Delete using ❌ button

---

### 🔹 4. `App.js` – Loads the Main Todo Component

```js

import React from 'react';

import TodoList from './features/todos/TodoList';

function App() {

return (

<div className="App">

<h1>Redux Todo App</h1>

<TodoList />

</div>

);

}

export default App;

```

✅ \*\*Purpose\*\*:

- Simple wrapper for the `TodoList` component

- Basic layout

---

### 🔹 5. `index.js` – Connects Redux Store to React

```js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import { Provider } from 'react-redux';

import { store } from './app/store';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<Provider store={store}>

<App />

</Provider>

);

```

✅ \*\*Purpose\*\*:

- Wraps the whole app in `<Provider>` to connect Redux

- Passes the store so components can access Redux state

---

## 🔄 Redux Flow Summary

1. \*\*User interacts with UI\*\* (e.g., enters a task and clicks "Add").

2. \*\*Component dispatches an action\*\* like `addTodo("Learn Redux")`.

3. \*\*Redux reducer updates the state\*\* in `TodoSlice`.

4. \*\*React re-renders\*\* with updated state from `useSelector`.

---

## 🧪 Key Redux Concepts Used

| Concept | Example Used |

|---------------------|-----------------------------------------------|

| Store | `store.js` with `configureStore()` |

| Reducers | `addTodo`, `toggleTodo`, `deleteTodo` |

| Actions | Created by `createSlice` |

| Slice | `todoSlice` in `TodoSlice.js` |

| State Selector | `useSelector(state => state.todos)` |

| Dispatching Actions | `dispatch(addTodo(text))` |

| Middleware (optional) | Redux Toolkit supports built-in middleware |

---