Project structure:

```
todo-app/
├── public/
│   └── index.html
├── src/
│   ├── App.jsx
│   ├── App.css
│   ├── main.jsx
│   ├── redux/
│   │   ├── store.js
│   │   └── todosSlice.js
│   └── index.css
├── package.json
└── vite.config.js
```

Let's convert the plain HTML/JS from Lab 6 to a modern React 18 app with Redux using `configureStore` and React Hooks.

Step 1: Install Redux and React-Redux in the project:

```
npm install @reduxjs/toolkit react-redux
```

Step 2: Set Up Redux Store

1. **Create a Redux slice** for managing the to-do list. In your project, create a new folder `src/redux` and inside it create a file `todosSlice.js`:

A Redux slice:

In Redux, a slice refers to a portion of the Redux state along with the actions that can modify that part of the state. For example, a "todos" slice might handle tasks like adding, removing, or toggling tasks in a to-do list.

Simplify Redux setup with `createSlice`

Instead of manually defining action types, action creators, and reducers, `createSlice` bundles them into one step.

`createSlice` gives you:

- **Action creators**: These are functions that automatically generate action objects for you, like `addTodo()` or `removeTodo()`.
- **Action types**: The action types are automatically derived from the slice name and the reducer function names. For example, the action type for `addTodo` would be `todos/addTodo`.
- **Reducer function**: A reducer function is created automatically that handles the state changes based on the dispatched actions.

```
// src/redux/todosSlice.js
import { createSlice } from '@reduxjs/toolkit';

const todosSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    addTodo: (state, action) => {
      state.push({
        id: Date.now(),
        text: action.payload,
      });
    },
    removeTodo: (state, action) => {
      return state.filter((todo) => todo.id !==
action.payload);
    },
  },
});

export const { addTodo, removeTodo } = todosSlice.actions;
export default todosSlice.reducer;
```

2. **Create the Redux Store**: In the `src/redux` folder, create `store.js`:

```
// src/redux/store.js

import { configureStore } from '@reduxjs/toolkit';
import todosReducer from './todosSlice';

export const store = configureStore({
  reducer: {
    todos: todosReducer,
```

```
  },
});
```

Wrap Your App with Redux Provider: In `src/main.jsx`, wrap your `App` component with `Provider` to give Redux access to the entire app.

```
// src/main.jsx

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 './redux/store';

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

Step 3: Set Up the App Component

Now, let's move the logic into the `App.jsx` file using React hooks and Redux.

1. **App Component**: In `src/App.jsx`, set up the component with Redux state management and React hooks.

```
    // src/App.jsx

import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { addTodo, removeTodo } from './redux/todosSlice';
import './App.css';
```

```
const App = () => {
  const [input, setInput] = useState('');
  const dispatch = useDispatch();
  const todos = useSelector((state) => state.todos);

  const handleAddTodo = () => {
    if (input.trim()) {
      dispatch(addTodo(input.trim()));
      setInput('');
    }
  };

  const handleRemoveTodo = (id) => {
    dispatch(removeTodo(id));
  };

  return (
    <div className="container">
      <h1>To-Do List</h1>
      <div className="input-container">
        <input
          type="text"
          id="todo-input"
          value={input}
          onChange={(e) => setInput(e.target.value)}
          placeholder="Add a new task..."
        />
        <button id="add-btn" onClick={handleAddTodo}>
          Add
        </button>
      </div>
      <ul id="todo-list">
        {todos.map((todo) => (
          <li key={todo.id} className="todo-item">
            <span>{todo.text}</span>
            <button onClick={() =>
handleRemoveTodo(todo.id)}>Delete</button>
          </li>
        ))}
      </ul>
```

```
    </div>
  );
};

export default App;
```

Step 4: Add the CSS Code

You can use the same `css` file and add it to the project.

CSS File: In the `src` folder, `App.css` looks like:

```
/* src/App.css */

body {
  font-family: Arial, sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
  background-color: #f0f0f0;
}

.container {
  background-color: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  width: 300px;
  text-align: center;
}

.input-container {
  display: flex;
  margin-bottom: 20px;
}

#todo-input {
  flex: 1;
```

```
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px 0 0 4px;
}

#add-btn {
  padding: 10px;
  border: none;
  background-color: #28a745;
  color: white;
  cursor: pointer;
  border-radius: 0 4px 4px 0;
}

#add-btn:hover {
  background-color: #218838;
}

#todo-list {
  list-style: none;
  padding: 0;
}

.todo-item {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  margin-bottom: 10px;
}

.todo-item button {
  background-color: #dc3545;
  color: white;
  border: none;
  padding: 5px 10px;
  cursor: pointer;
  border-radius: 4px;
```

```
}

.todo-item button:hover {
  background-color: #c82333;
}
```

Step 5: Run the Application

1. **Start the Vite Development Server**: In the terminal, run the following command to start the development server:

   ```
   npm run dev
   ```

2. **Test the Application**:
   - Open your browser and go to the URL `http://localhost:5173`.
   - It should work as before: You should see your to-do list app.