National University of Computer and Emerging Sciences

Laboratory Manual

for

Web Engineering (SL3003)

Lab Instructors	Sana Ejaz, Hassan Raza
Section	6C 1,2
Semester	Spring 2025

Department of Software Engineering

FAST-NU, Lahore, Pakistan

React Redux

Objectives

- Learn to create and configure a Redux store manually.
- Use Redux actions, reducers, and `connect` or `useSelector/useDispatch` hooks to manage and access global state.
- Move the hardcoded `conceptsData` to the Redux store.
- Add interactivity to add/remove/update concepts (optional but recommended).

Scenario:

You've already created a documentation website with React Router v6. Now it's time to extend the app by integrating **React Redux** for global state management. The goal is to store and manage the documentation concepts in a central Redux store and practice dispatching actions and selecting state across components.

Task 1: Redux Setup and Store Configuration

1. **Install Required Packages**:
    - Install `redux` and `react-redux`.

    ```
    npm install redux react-redux
    ```

2. **Create Redux Folder Structure**:
    - Make a `redux/` folder with `store.js`, `reducers/conceptReducer.js`, and `actions/conceptActions.js`.
3. **Move conceptsData to Redux**:
    - Define `conceptsData` in your reducer's initial state.
    - Create a reducer that returns this data by default.
4. **Create an Action**:
    - In `conceptActions.js`, define a `LOAD_CONCEPTS` action type and action creator (for practice, even if not used yet).
5. **Configure the Store**:

    - Create the store using createStore().
    - Combine reducers (even if only one for now).
6. **Wrap the App with Redux Provider**:
    - In index.js, wrap your App with <Provider store={store}>.

```
import { Provider } from 'react-redux';

import store from './redux/store';

<Provider store={store}>

        <BrowserRouter>

            <App />

        </BrowserRouter>

</Provider>
```

Task 2: Display Concepts from Redux Store

Use Redux in ConceptsOverviewPage:

- Replace the hardcoded list by accessing `concepts` from Redux using `useSelector`.
- Display the names of all concepts with links (as done before).

Task 3: Dynamic Concept Detail with Redux

Fetch Concept by ID:

- In `ConceptDetailPage`, use `useParams` to get the concept ID.
- Use `useSelector` to find that concept from the Redux store.
- Display its `name` and `description`.

Task 4: Nested Examples Page using Redux

1. **Show Examples**:
    - In `ExamplesPage`, get the `conceptId` using `useParams`.
    - Find the concept using `useSelector` and display its `examples` array.
2. **Back Button**:
    - Add a button that navigates back to the concept detail page using `useNavigate`.

Task 5: (Optional Challenge): Add/Delete Concepts

1. **Create New Redux Actions**:
    - Add `ADD_CONCEPT` and `REMOVE_CONCEPT` action types and creators.
2. **Update Reducer**:
    - Handle the add and remove actions to update the state.
3. **Create Basic UI**:
    - Add buttons to add a dummy concept or delete one by ID.

Homework

Task: Introduction to React Redux Toolkit

Instructions:

1. **Set Up Redux Toolkit**:
    - Install `@reduxjs/toolkit` and `react-redux`.
2. **Create a Redux Slice**:
    - Inside a `features/counter/` folder, create a file named `counterSlice.js`.
    - Use `createSlice()` to define:
        - `initialState` as `{ value: 0 }`
        - Reducers for:
            - `increment`
            - `decrement`
            - `reset`
3. **Create the Store**:
    - In `app/store.js`, use `configureStore()` to set up the store with the `counter` reducer.

- Wrap your `<App />` component with `<Provider store={store}>` in `index.js`.

4. **Use the Redux State in a Component**:
    - Create a `Counter.js` component.
    - Use `useSelector` to read the counter value.
    - Use `useDispatch` to trigger actions from the slice.
5. **UI Requirements**:
    - Display the counter value.
    - Buttons for:
        - "+" → increments the counter
        - "-" → decrements the counter
        - "Reset" → resets the counter to 0