HAZZA INSTITUTE OF TECHNOLOGY

iqracity

HAZZA INSTITUTE OF TECHNOLOGY

MERN Stack (ES6 + REACT) Course

iqracity

Lab 16

Total Time:
3 hours

Pre-Lab Activities:

- No Pre-Lab Activity

Learning Outcomes:

- Getting Started with React Redux

Lab Tasks:

  - Getting a text Editor
  - Installing Node in the local machine
  - Installing Express and React
  - Installing our database locally (MongoDB)

Student Activities:

  - To exploretext Editor
  - To exploreNode in the local machine
  - To exploreExpress and React
  - To exploreMongoDB

Pre-Reqs of the Lab

Before you begin the Lab, you must have gone through the video lectures, lecture slides shared.

Contents of the Lab

After completing the above exercises go to https://react-redux.js.org/introduction/getting-started in order to install and use React Redux

Learning outcome of the Lab

After the lab you will be equipped with contents provided in the lab manual.

Good Luck!

Lab Solution

Getting Started with React Redux

React Redux is the official React UI bindings layer for Redux. It lets your React components read data from a Redux store, and dispatch actions to the store to update state.

Installation

React Redux 8.x requires **React 16.8.3 or later** / **React Native 0.59 or later**, in order to make use of React Hooks.

Using Create React App

The recommended way to start new apps with React and Redux is by using the official Redux+JS template or Redux+TS template for Create React App, which takes advantage of Redux Toolkit and React Redux's integration with React components.

```
# Redux + Plain JS template
npx create-react-app my-app --template redux

# Redux + TypeScript template
npx create-react-app my-app --template redux-typescript
```

An Existing React App

To use React Redux with your React app, install it as a dependency:

```
# If you use npm:
npm install react-redux

# Or if you use Yarn:
yarn add react-redux
```

You'll also need to install Redux and set up a Redux store in your app.

React-Redux v8 is written in TypeScript, so all types are automatically included.

API Overview

Provider

React Redux includes a <Provider /> component, which makes the Redux store available to the rest of your app:

```
import React from 'react'
import ReactDOM from 'react-dom/client'

import { Provider } from 'react-redux'
import store from './store'

import App from './App'

// As of React 18
const root = ReactDOM.createRoot(document.getElementById('root'))
root.render(
<Provider store={store}>
<App />
</Provider>
)
```

Hooks

React Redux provides a pair of custom React hooks that allow your React components to interact with the Redux store.

useSelector reads a value from the store state and subscribes to updates,
while useDispatch returns the store's dispatch method to let you dispatch actions.

```
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import {
  decrement,
  increment,
  incrementByAmount,
  incrementAsync,
  selectCount,
} from './counterSlice'
import styles from './Counter.module.css'
```

```
export function Counter() {
  const count = useSelector(selectCount)
  const dispatch = useDispatch()

  return (
<div>
<div className={styles.row}>
<button
        className={styles.button}
        aria-label="Increment value"
        onClick={() => dispatch(increment())}
>
        +
</button>
<span className={styles.value}>{count}</span>
<button
        className={styles.button}
        aria-label="Decrement value"
        onClick={() => dispatch(decrement())}
>
        -
</button>
</div>
     {/* omit additional rendering output here */}
</div>
  )
}
```

Learning React Redux

Learn Modern Redux Livestream

Redux maintainer Mark Erikson appeared on the "Learn with Jason" show to explain how we recommend using Redux today. The show includes a live-coded example app that shows how to use Redux Toolkit and React-Redux hooks with Typescript, as well as the new RTK Query data fetching APIs.

See the "Learn Modern Redux" show notes page for a transcript and links to the example app source.

Help and Discussion

The #redux channel of the Reactiflux Discord community is our official resource for all questions related to learning and using Redux. Reactiflux is a great place to hang out, ask questions, and learn - come join us!

You can also ask questions on Stack Overflow using the #redux tag.

Install Redux Toolkit and React Redux

Add the Redux Toolkit and React Redux packages to your project:

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

Create a Redux Store

Create a file named src/app/store.js. Import the configureStore API from Redux Toolkit. We'll start by creating an empty Redux store, and exporting it:

```
app/store.js

import { configureStore } from '@reduxjs/toolkit'

export default configureStore({
  reducer: {},
})
```

This creates a Redux store, and also automatically configure the Redux DevTools extension so that you can inspect the store while developing.

Provide the Redux Store to React

Once the store is created, we can make it available to our React components by putting a React Redux <Provider> around our application in src/index.js. Import the Redux store we just created, put a <Provider> around your <App>, and pass the store as a prop:

```
index.js

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

// As of React 18
const root = ReactDOM.createRoot(document.getElementById('root'))

root.render(
<Provider store={store}>
<App />
</Provider>
)
```

Create a Redux State Slice

Add a new file named src/features/counter/counterSlice.js. In that file, import the createSlice API from Redux Toolkit.

Creating a slice requires a string name to identify the slice, an initial state value, and one or more reducer functions to define how the state can be updated. Once a slice is created, we can export the generated Redux action creators and the reducer function for the whole slice.

Redux requires that we write all state updates immutably, by making copies of data and updating the copies. However, Redux Toolkit's createSlice and createReducer APIs use Immer inside to allow us to write "mutating" update logic that becomes correct immutable updates.

```
features/counter/counterSlice.js

import { createSlice } from '@reduxjs/toolkit'

export const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0,
  },
  reducers: {
    increment: (state) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

Add Slice Reducers to the Store

Next, we need to import the reducer function from the counter slice and add it to our store. By defining a field inside the reducers parameter, we tell the store to use this slice reducer function to handle all updates to that state.

```
app/store.js

import { configureStore } from '@reduxjs/toolkit'
import counterReducer from '../features/counter/counterSlice'

export default configureStore({
 reducer: {
  counter: counterReducer,
 },
})
```

Use Redux State and Actions in React Components

Now we can use the React Redux hooks to let React components interact with the Redux store. We can read data from the store with useSelector, and dispatch actions using useDispatch. Create a src/features/counter/Counter.js file with a <Counter> component inside, then import that component into App.js and render it inside of <App>.

```
features/counter/Counter.js
```

```
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { decrement, increment } from './counterSlice'
import styles from './Counter.module.css'

export function Counter() {
  const count = useSelector((state) => state.counter.value)
  const dispatch = useDispatch()

  return (
<div>
<div>
<button
        aria-label="Increment value"
        onClick={() => dispatch(increment())}
>
        Increment
</button>
<span>{count}</span>
<button
        aria-label="Decrement value"
        onClick={() => dispatch(decrement())}
>
        Decrement
</button>
</div>
</div>
  )
}
```

Now, any time you click the "Increment" and "Decrement buttons:

- The corresponding Redux action will be dispatched to the store
- The counter slice reducer will see the actions and update its state
- The <Counter> component will see the new state value from the store and re-render itself with the new data

What You've Learned

That was a brief overview of how to set up and use Redux Toolkit with React. Recapping the details:

SUMMARY

- Create a Redux store with configureStore
- configureStore accepts a reducer function as a named argument
- configureStore automatically sets up the store with good default settings
- Provide the Redux store to the React application components
- Put a React Redux <Provider> component around your <App />
- Pass the Redux store as <Provider store={store}>
- Create a Redux "slice" reducer with createSlice
- Call createSlice with a string name, an initial state, and named reducer functions
- Reducer functions may "mutate" the state using Immer
- Export the generated slice reducer and action creators
- Use the React Redux useSelector/useDispatch hooks in React components
- Read data from the store with the useSelector hook
- Get the dispatch function with the useDispatch hook, and dispatch actions as needed