Lesson 3 – Redux Fundamentals

Topics

a. What is a State in ReactJS?
b. What is State Management?
c. What is Redux?
d. Why and When Should to Use Redux?
e. Redux Libraries and Tools
f. Redux Terms and Concepts
    a. Immutability
    b. Terminologies
g. Redux Application Data Flow

Activity 3 - Redux Toolkit Setup

The objectives of this activitiy are to:

a) Understand the basic concepts of Redux Toolkit
b) Set up and configure a Redux store.
c) Create slices.

This is a basic example for implementing application state management using React and Redux Toolkit.

1. **Install the dependencies in your client.**
    a. npm install @reduxjs/toolkit react-redux
2. **Create a Redux Store**

    In the folder client/src, create a new folder: **Store**. Inside this folder, create a file named **store.js**.

    Import the **configureStore** API from Redux Toolkit.

The configureStore function from @reduxjs/toolkit is used to create a Redux store.

You'll start by creating an empty Redux store, and exporting it:

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

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

3. **Import the store in index.js**

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import { store } from "./Store/store";

const root =
ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App/>
  </React.StrictMode>
);
```

4. **Provide the Redux Store to the React Application**
    a. Once the store is created, you can make it available to our React components by putting a React-Redux <Provider> around the application in **src/index.js**.
    b. Import the <Provider> component.

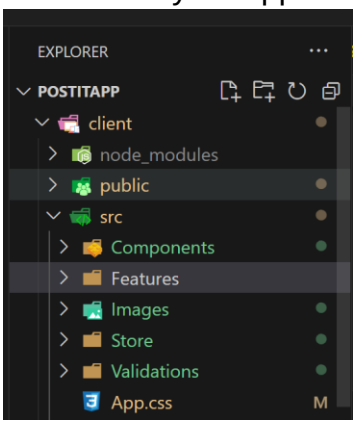
```
import {Provider} from "react-redux";
```

The Provider component from react-redux is used to **connect the Redux store to your React application**. Its primary function is to make the Redux store available to all components in your app, so they can access and interact with the store using hooks like useSelector and useDispatch.

    c. Put the<Provider> component around your <App>, and pass the store as a prop:

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


const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}> {/* put a <Provider> around your
<App>, and pass the store as a prop:*/}
  <React.StrictMode>
    <App />
  </React.StrictMode>
  </Provider>
);
```

5. **Create a Redux State Slice:**
   In client/src, create a new folder:**Features**. The Features folder will contain the reducers of your application.



   Create a new file in src/Features/ named: **UserSlice.js**. In that file, import the createSlice API from Redux Toolkit.

   Creating a slice requires the following:
   a. a string name to identify the slice,
   b. an initial state value,
   c. and one or more reducer functions to define how the state can be updated.

Once a slice is created, you can export the generated Redux action creators and the reducer function for the whole slice.

Note: Every aspect of the application that you want manage state, you can create a reducer for each.

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = { value: [] }; //list of user is an object
with empty array as initial value

export const userSlice = createSlice({
  name: "users", //name of the state
  initialState, // initial value of the state
  reducers: { },
});

export default userSlice.reducer;
```

Export the all the reducers so it can be accessed outside the file.

6. **Add Slice Reducers to the Store**. In src/Store/store.js, import the usersReducer from the features/UserSlice.js file.

```
import { configureStore } from "@reduxjs/toolkit";
import usersReducer from "../Features/UserSlice.js"; //import the
reducer

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

users: This is the **key** or **slice name** that defines a part of your global Redux state. You can use any name to represent the state being managed by that reducer.

7. For this example, you will provide the first example data. However, in real scenarios data may come from a different such as a database. Create a new file in src/Exampledata.js. Create a variable which contains an array of objects.

```
export const UsersData = [
  {
    id: 1,
    name: "Jasmin Tumulak",
    email: "jasmine@utas.edu.om",
    password: "12345",
  },
  {
    id: 2,
    name: "Marian Malig-on",
    email: "marian@utas.edu.om",
    password: "12345",
  },
  {
    id: 3,
    name: "Ahmed Ali Jaboob",
    email: "ahmed@utas.edu.om",
    password: "12345",
  },
];
```

8. In Features/UserSlice.js, change the initial value of the slice to the value from the Exampledata.js file. You may just comment the first assignment statement of the initialState variable.

```
import { createSlice } from "@reduxjs/toolkit";
import { UsersData } from "../Exampledata";

//const initialState = { value: [] }; //list of user is an object
with empty array as initial value
const initialState = { value: UsersData }; //Assign the data from
the exampleData
…..
…..
```

9. In src/index.js, temporarily render the <Register> component instead of the App component so we can view the component.

```
…..
…..
root.render(
  <Provider store={store}> {/* put a <Provider> around your
<App>, and pass the store as a prop:*/}
  <React.StrictMode>
    <Register /> //Render the Register component instead of the App
  </React.StrictMode>
  </Provider>


);
```

10. In Register.js file, import the useSelector hook.

```
import { useSelector } from "react-redux";
```

The use of the useSelector hook is accessing Redux State and to extract data from the Redux store state.

11. Retrieve the current value of the state and assign it to a variable. This code will allow you to have access to the current value of the **users** state.

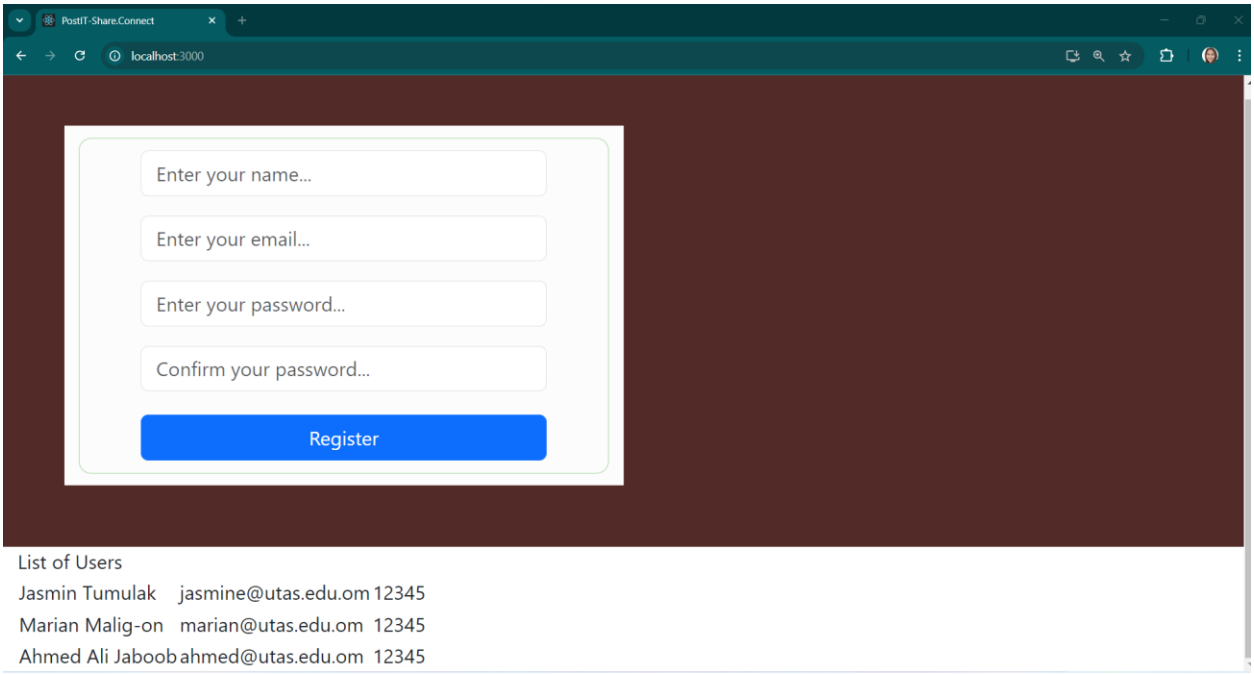
```
import { userSchemaValidation } from
"../Validations/UserValidations";
import * as yup from "yup";
import { useForm } from "react-hook-form";
…..
import { yupResolver } from "@hookform/resolvers/yup";
import { useSelector } from "react-redux";

const Register = () => {
 //Retrieve the current value of the state and assign it to a
variable.
  const userList = useSelector((state) => state.users.value);
…..
}
```

12. Then you can write the code to iterate over the array of list of users by displaying it in a table using the array **map()** method. You can create another row for this in Register.js. Below is the complete code for the UsersTest.js.

```
……
import { useSelector } from "react-redux";

const Register = () => {
  //Retrieve the current value of the state and assign it to a
variable.
  const userList = useSelector((state) => state.users.value);

…..
  };
  return (
    <Container fluid>
      <Row className="formrow">
….
      </Row>
      <Row>
        <Col md={6}>
          List of Users
          <table>
            <tbody>
              {userList.map((user) => (
                <tr key={user.email}>
                  <td>{user.name}</td>
                  <td>{user.email}</td>
```

```
                    <td>{user.password}</td>
                  </tr>
                ))}
              </tbody>
            </table>
          </Col>
        </Row>
      </Container>
    );
};

export default Register;
```

This is the expected output.



Technical Terms:

Redux is a library used to manage the state of an application, especially in JavaScript-based apps like those built with React. Imagine your app is like a giant warehouse, and the state is all the information the app needs to keep track of, like user preferences, data from the server, or UI status. Redux helps you organize and control all this information efficiently.

State: The data or information your app needs to function.

Store: Redux creates a central "store," which is like a big box where you keep all the state in one place.

Actions: When something happens (e.g., a user clicks a button), an "action" is created. This action tells Redux what you want to change in the store.

Reducers: A reducer is a function that takes the current state and an action, and returns the new state. It’s like a blueprint for how the store should change based on the action.

Dispatch: When you want to trigger an action, you "dispatch" it, which sends it to the reducer to update the store.