Lesson 4 – Redux Toolkit

Topics

a. What is Redux Toolkit?
b. Installation
c. Purpose
d. Creating the Redux Store
e. Providing the Redux Store to React
f. Rules of Reducers
g. Creating Slice Reducers
h. Add Slice Reducers to the Store
i. Use Redux State and Actions in React Components
    a. useSelector hook
    b. useDispatch hook

Activity 4 - Basic CRUD Implementation using React and Redux Toolkit

The objectives of this activitiy are to:

a) Dispatch actions to modify the store.
b) Design and implement reducers.
c) Create and integrate action creators to handle CRUD operations.

1. In the **Components/Register.js**, import the useState hook. Create the state variable for each of the data in the form.

```
import { useState } from "react";
```

```
const Register = () => {
  //Retrieve the current value of the state and assign it to a variable.
  const userList = useSelector((state) => state.users.value);
  //Create the state variables
  const [name, setname] = useState("");
  const [email, setemail] = useState("");
  const [password, setpassword] = useState("");
  const [confirmPassword, setconfirmPassword] = useState("");
…
…
}
```

2. Write the event handler to assign the values of the input to the state variables for each of the inputs. Since, this input field is associated with the form state managed by react-hook-form, and its value is controlled using the name state variable and setname function. {...register("name", { value: name, onChange: (e) => setname(e.target.value) })}: This is using the register function from react-hook-form to associate the input with the form state.

Do this in all the inputs.

```
<input
  type="text"
  className="form-control"
  id="name"
  placeholder="Enter your name..."
  //register this input to the react-hook
  {...register("name", {
    onChange: (e) => setname(e.target.value),
  })}
/>
```

3. In the **src/Features/UserSlice.js**, write code to create the reducer **addUser**. The **addUser** reducer will update the value of the user state by pushing the new value to the state.
    a. In creating the reducer, follow the syntax:

```
reducers:{
   reducername:(state,action)=>{

   }
}
```

State is the current value of the state,
Action is triggered outside the reducer and provides a value as payload.
Payload is the value coming from the component that will be used to update the value of the state.

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

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

const initialState = { value: UsersData };

export const userSlice = createSlice({
  name: "users", //name of the state
  initialState,
  reducers: {
    addUser: (state, action) => {
      state.value.push(action.payload); //add the payload to the
state
    },
  },
});
export const {addUser} = userSlice.actions; //export the function

export default userSlice.reducer;
```

Don’t forget to export the **addUser** function.

4. In the Components/Register.js, import the **addUser** function.

```
import { addUser } from "../Features/UserSlice";
```

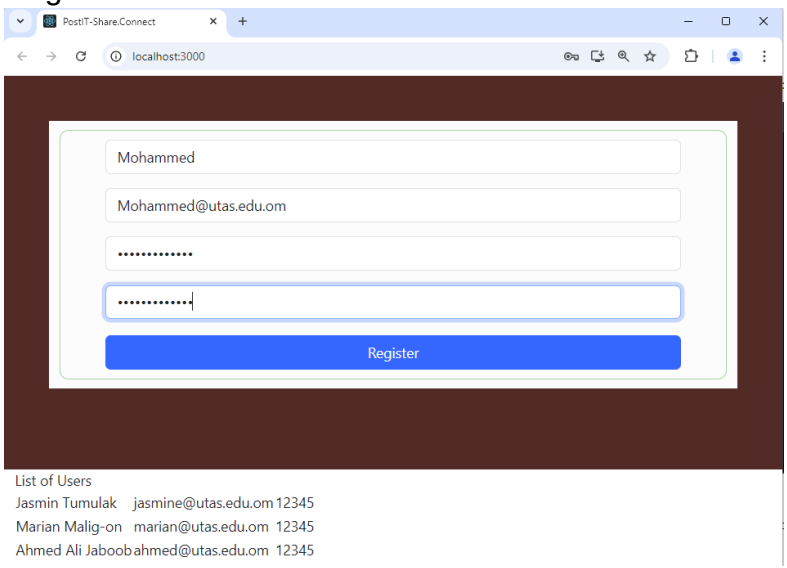
5. Call this function when the user clicks the button. However, this cannot be called directly because it must be registered as part of the store. To do this you need to use another hook which is the **useDispatch** hook. Import this in Register component also.

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

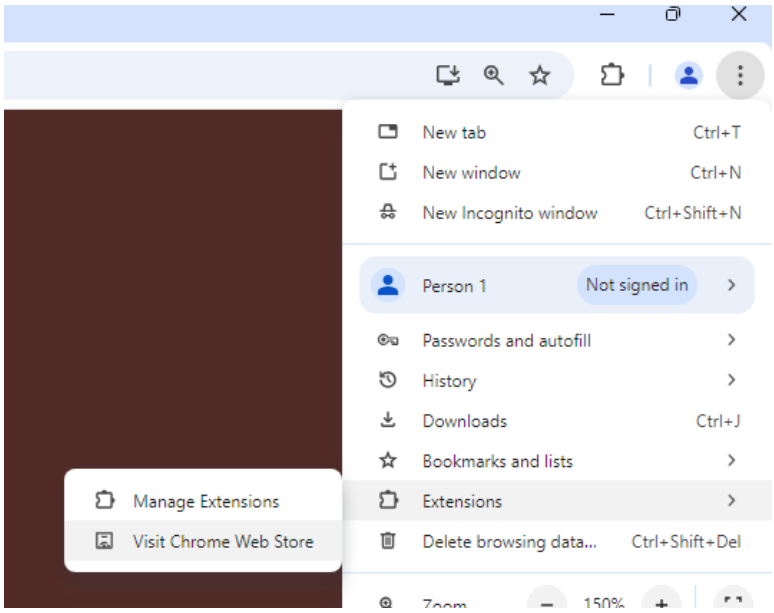
6. In the **Register** component, we have created a function **onSubmit()** in the previous activity to handle the form submission. Since we are using the react-hook-form, the data refers to the values that have been collected from the form inputs after the user submits the form. This data is usually an object that holds key-value pairs, where each key corresponds to the name of a form field, and the value is what the user has entered.

```
  // Handle form submission
  const onSubmit = (data) => {
    console.log("Form Data", data); // You can handle the form
submission here
    alert("Validation all good.");
  };
```

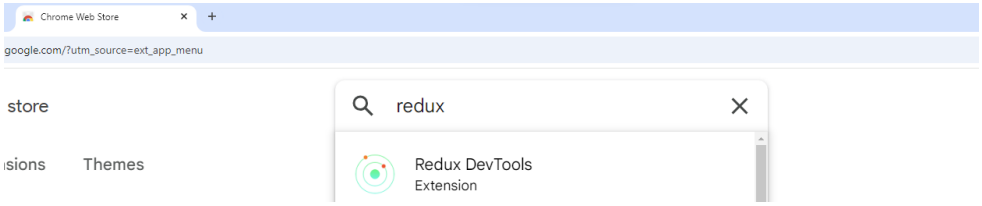
7. Let us update this function to handle the data submitted by user from the Register form and add the data to the Redux state.



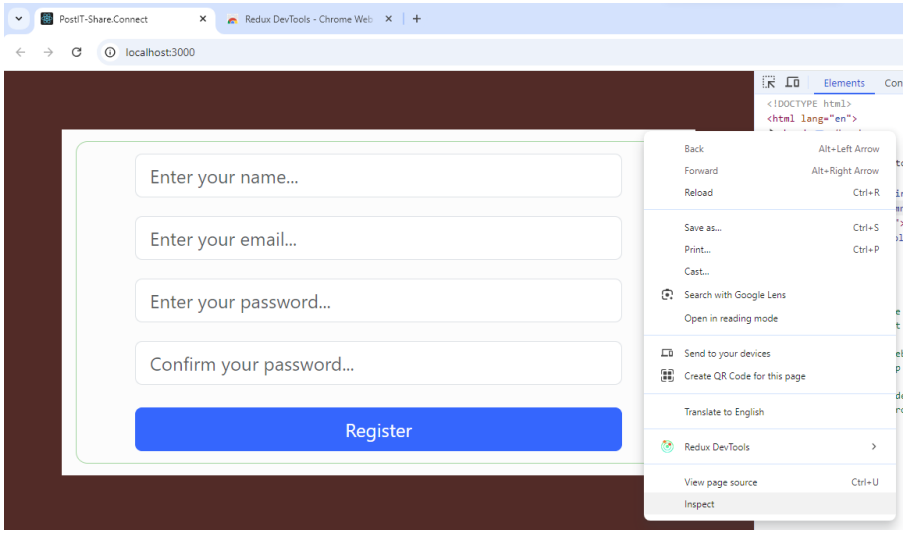
8. Let us check the current value of the Redux states. In Chrome, click on Extensions->Visit Chrome Web Store. We will use a Chrome extension that is useful in tracking the values of the Redux store and its states.



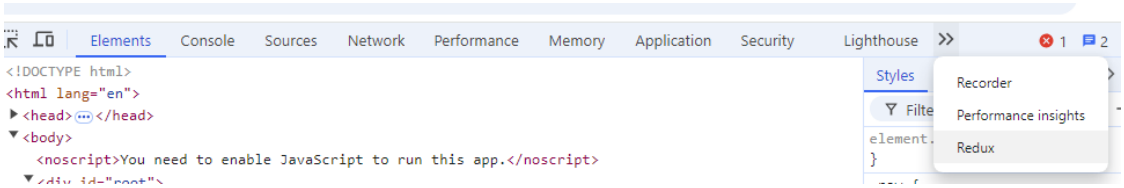
In Chrome Web Store, search for the Redux DevTools. Add this extension to the browser.



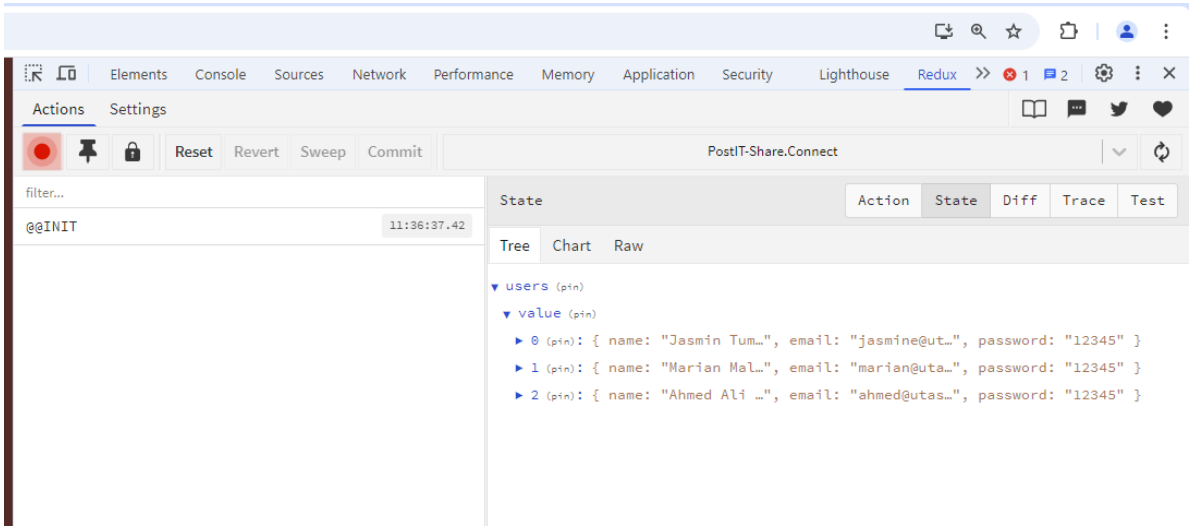
9. View the Redux Dev Tools. Right click on your page and select Inspect.



In the tab you will see the Redux tab.



Select the Redux tab. Select the State tab and here you will see the current value of the “users” state in the Redux store.

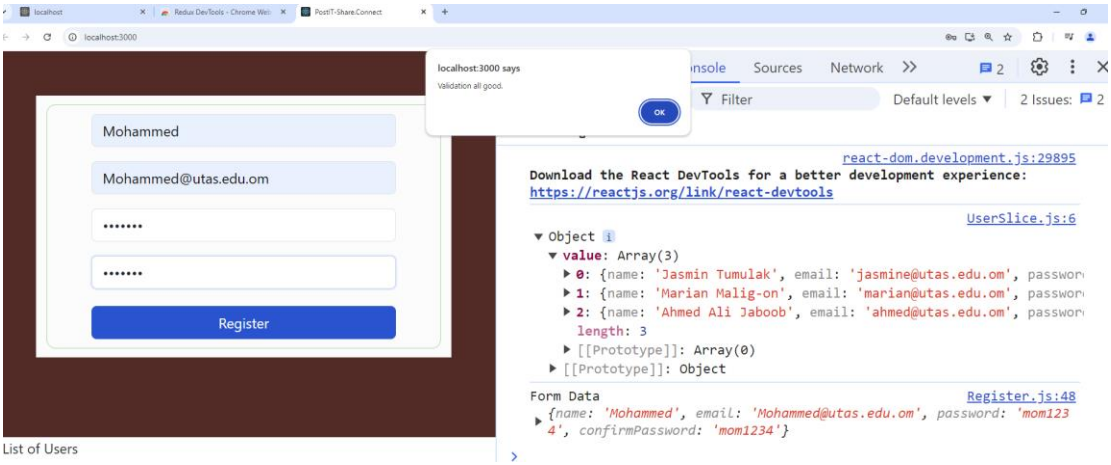


10. Let us continue to write the code to update the **onSubmit()** function. Write the try…catch block to have error trapping before processing the data.

```
  const onSubmit = (data) => {
    try {
      console.log("Form Data", data); // You can handle the
form submission here
      alert("Validation all good.");
    } catch (error) {
      console.log("Error.");
    }
  };
```

Note that we have a code: `console.log("Form Data", data);`

Let us check the output of this code. In your browser, write some information in the form. Click the submit button and inspect the page and go to the Console tab. Observe the value of the Form Data.



11. In the function onSubmit(), you can now create an object variable to store the data submitted from the form.

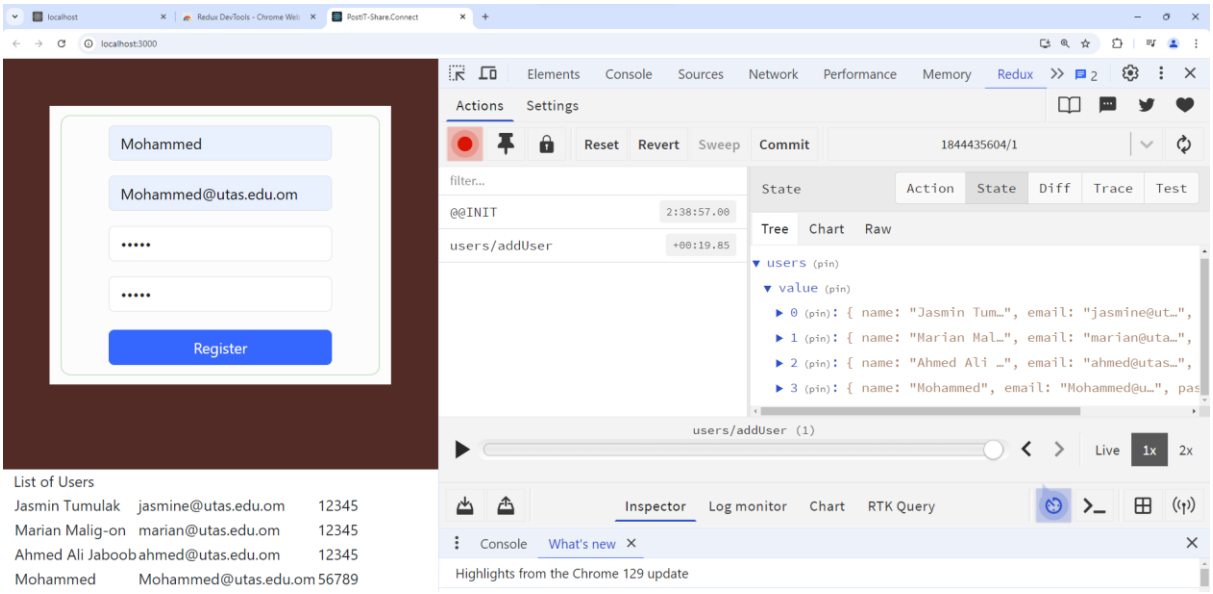
```
…..
const Register = () => {
  //Retrieve the current value of the state and assign it to a
variable.
….
  //For form validation using react-hook-form
…
  // Handle form submission
  const onSubmit = (data) => {
    try {
      // You can handle the form submission here
      const userData = {
        name: data.name,
        email: data.email,
        password: data.password,
      };
      console.log("Form Data", data);
      alert("Validation all good.");
    } catch (error) {
      console.log("Error.");
    }
  };
return (…..)
```

12. Create a variable for the useDispatch() hook. You may write it before the function onSubmit(). Use the useDispatch hook to dispatch an action, passing as parameter the userData.

```
  const dispatch = useDispatch();
```

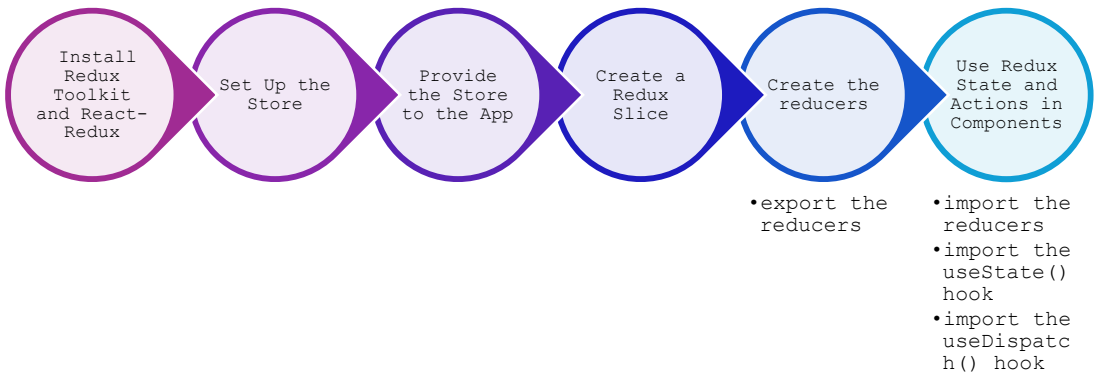
```
const onSubmit = (data) => {
    try {
      // You can handle the form submission here
      const userData = {
        name: data.name,
        email: data.email,
        password: data.password,
      };

      console.log("Form Data", data);
      alert("Validation all good.");
      dispatch(addUser(userData)); //use the useDispatch hook to
dispatch an action, passing as parameter the userData

    } catch (error) {
      console.log("Error.");
    }
  };
```

13. Let us check our code if it successfully added new data to the “users” state in the Redux store. Input some information and using Chrome Redux Dev Tools, you can check your app state. Right click on your browser->Inspect Element. Then to Redux tab.



You should see the actions invoked and the state with the values. Notice also the List of Users in the browser is updated.

14. Let us summarize the process that we have done so far in using the Redux Toolkit.



15. Then will try to implement the other CRUD functionalities. Let us start with the delete. In your reducer: src/Features/UserSlice.js, create another function, **deleteUser().**

```
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
console.log(initialState);
export const userSlice = createSlice({
  name: "users", //name of the state
  initialState, // initial value of the state
  reducers: {
    addUser: (state, action) => {
     //state is the current value of the state, action is triggered outside
the reducer and provides a value as payload
      state.value.push(action.payload); //the payload is the value
coming from the component, add the payload to the state
    },
    deleteUser: (state, action) => {
     //create a new array with the value that excludes the user with the
email value from the action payload, and assign the new array to the state.
      state.value=state.value.filter((user) => user.email !==
action.payload);
    },
  },
});

export const { addUser,deleteUser } = userSlice.actions; //export
the function

export default userSlice.reducer;
```

You have to export the deleteUser function.

16. In **src/Components/Register.js**, you need to create another function, handleDelete. The function will accept the parameter email that will be sent as payload to the deleteUser function in the reducer. You may write this function after the onSubmit function.

```
  const handleDelete = (email) => {
    dispatch(deleteUser(email));
  };
```

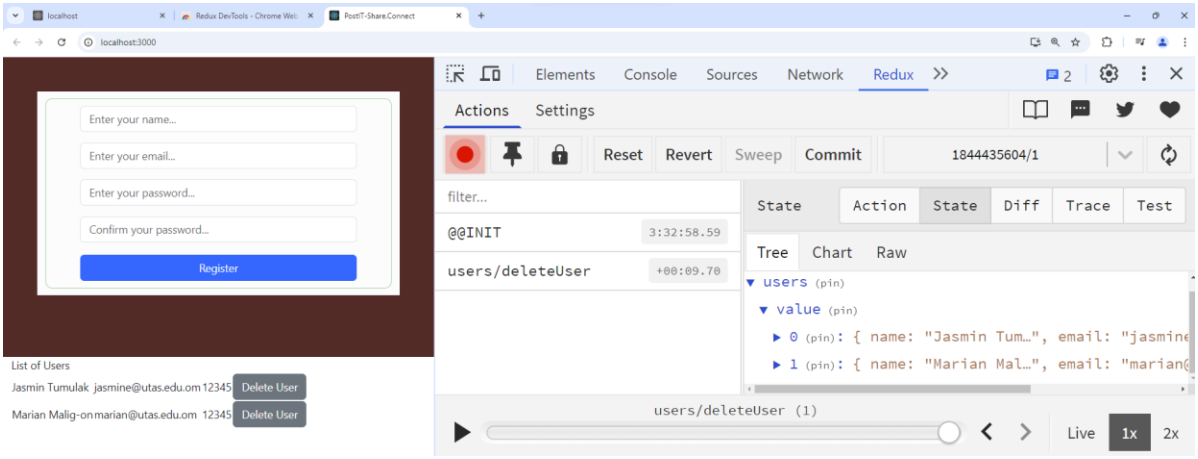
17. In src/Components/Register.js, import the deleteUser reducer.

```
import { addUser,deleteUser } from "../Features/UserSlice";
```

18. In the UI of src/Components/Register.js, you need to add the delete button. In that button we will call a function handle the delete user functionality. In the button create an event handler that will call the function handleDelete() and pass as parameter the email of the user.

```
          <table>
            <tbody>
              {userList.map((user) => (
                <tr key={user.email}>
                  <td>{user.name}</td>
                  <td>{user.email}</td>
                  <td>{user.password}</td>
                  <td>
                    <Button onClick={() =>
handleDelete(user.email)}>
                      Delete User
                    </Button>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
```

19. You can test your code again by checking in Chrome Dev Tools. Click the delete button and check your state if the value is deleted from the state.



20. Now we will implement the update functionality of the CRUD. In the src/Features/UserSlice.js. Create another function in the reducer, **updateUser().** For this example, we will update by using the email as the key to update the name and the password. Don't forget to export the new function.

```
….
export const userSlice = createSlice({
  name: "users", //name of the state
  initialState, // initial value of the state
  reducers: {
…..
    updateUser: (state, action) => {
      state.value.map((user) => {
        //iterate the  array and compare the email with the email from the
payload
        if (user.email === action.payload.email) {
          user.name = action.payload.name;
          user.password = action.payload.password;

        }
      });
    },
});
export const { addUser, deleteUser, updateUser } = userSlice.actions;
//export the function

export default userSlice.reducer;
```

21. In the src/Components/Register.js, import the updateUser() function.

```
import { addUser, deleteUser, updateUser } from "../Features/UserSlice";
```

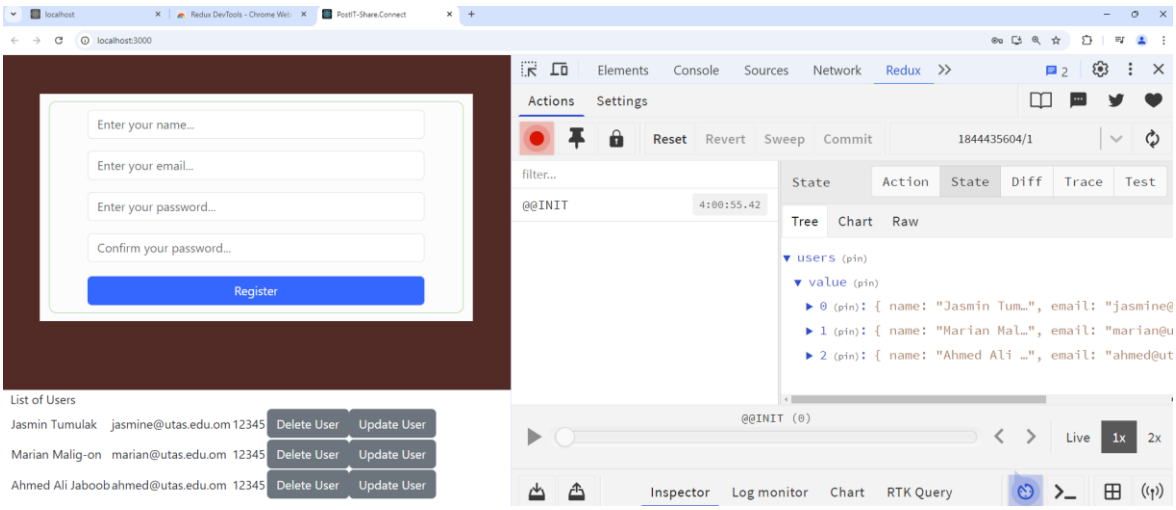
22. In the src/Components/Register.js, create a new function **handleUpdate** (). create an object with the values from the state variables. Then call the dispatch to invoke the updateUser function from the Users reducer and pass the userData as payload.

```
  const handleUpdate = (email) => {
    const userData = {
      name: name, //create an object with the values from the state variables
      email: email,
      password: password,
    };
    dispatch(updateUser(userData)); //use the useDispatch hook to
dispatch an action, passing as parameter the userData
  };
```

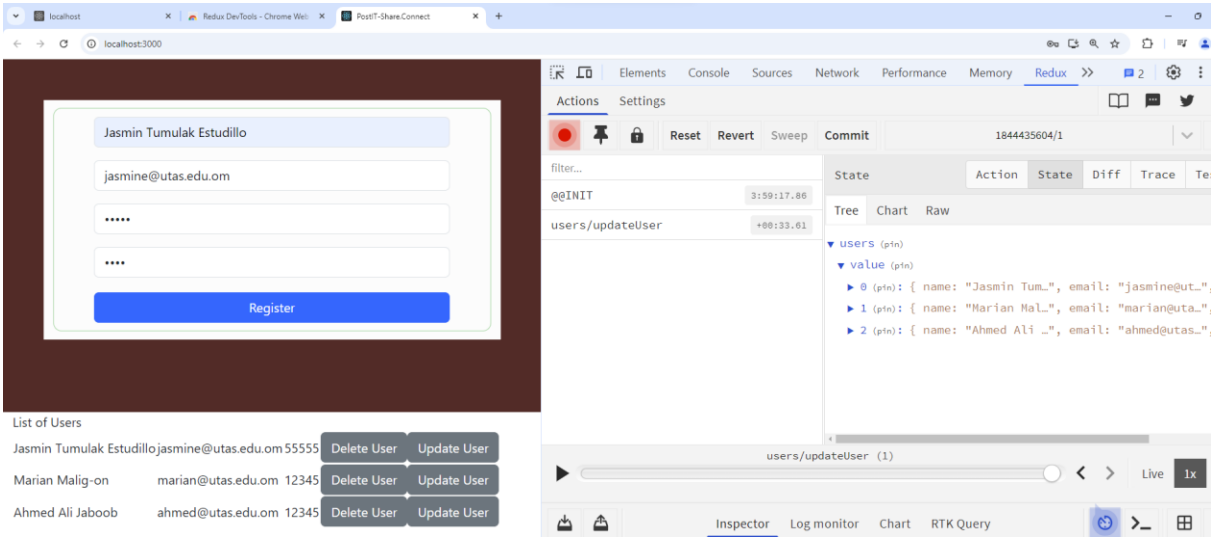
23. In Register.js, add another button for update user. Call the function handleUpdate() and pass the email as the argument.

```
<table>
      <tbody>
        {userList.map((user) => (
          <tr key={user.email}>
            <td>{user.name}</td>
            <td>{user.email}</td>
            <td>{user.password}</td>
            <td>
              <Button onClick={() => handleDelete(user.email)}>
                Delete User
              </Button>
              <Button onClick={() => handleUpdate(user.email)}>
                Update User
              </Button>

            </td>
          </tr>
        ))}
      </tbody>
    </table>
```

24. You can then test your code. Check the Redux Dev Tools to check the current values of the state.



Input new information in the form to be updated for the user. Click the Update User button. Check the Redux dev tools for the updated values of the state.



Note the update functionality does not have validation checking as this point. This is just to test our reducers.