Access State with useSelector

Start Assignment

Due No Due Date **Points** 1 **Submitting** a website url

FORK (https://github.com/learn-co-curriculum/react-hooks-redux-use-selector-lab/fork)
(https://github.com/learn-co-curriculum/react-hooks-redux-use-selector-lab)
(https://github.com/learn-co-curriculum/react-hooks-redux-use-selector-lab/issues/new)

Learning Goals

- Use the **React Redux** library to connect the store to the **React** application
- Utilize the `<Provider />` component and the `useSelector` hook to access **Redux** store content

Overview

In this lesson, we want to explore how `useSelector` is used to connect regular React components with the **Redux** store. This is also a good opportunity to review the steps for using the `redux` and `react-redux` packages in your app.

Instructions

Some files are provided, including `UserInput` and the reducer in the `usersSlice.js` file, but the **Redux** store isn't fully hooked up yet.

Connecting to Redux

In `src/index.js`, use the `createStore` method from `redux`, passing in the provided reducer, `usersReducer`, to create a `store`. Use `Provider` from `react-redux` to wrap `<App />`, passing `store` as a prop to the `Provider`. This will give your components access to the store.

Test by Dispatching an Action

Run `npm start` and open up your browser's dev console. If everything is connected correctly in `index.js`, a form should appear in the browser. Submitting something using the form will cause a `console.log` to fire in our reducer, indicating that the values have been added to our store.

In `UserInput.js`, we can see the code that fires when we press the submit button:

```
// ...

function handleOnSubmit(event) {
  event.preventDefault();
```

```
    dispatch({ type: "users/add", payload: formData });
  }

  return <form onSubmit={handleOnSubmit}>{/* ... */}</form>;
```

We can see that, *on submit*, `handleOnSubmit()` is called. `event.preventDefault()` is called to stop the page from refreshing, then `dispatch()` is called with a custom action, `{type: 'users/add', payload: formData}`.

The `dispatch` function is provided by calling the `useDispatch` hook from **React Redux** in our component

Using the useSelector Hook

Now that we've got a working store, we want to get access to it and display the contents of our store's state.

1. Import the `useSelector` hook in `Users.js`
2. Call `useSelector` inside your component, passing in a callback function that accepts one argument, `state`, the current version of your store's state. Use `state` to access the array of `users` and return that from the callback function. Save the return value of calling `useSelector` to a variable so that you can use the `users` array in your component.

The Users component should display the username of a user submitted to the store. To pass the final test, it should also display a total count of current users. Try to use `useSelector` to solve both. You can call `useSelector` multiple times to return different values: one for `users` and one for the `userCount`.

Conclusion

With all tests passing, you should have a working form that adds and successfully displays usernames, as well as a total count of those users. While these are small bits of data, we've got a fully integrated React/Redux application, ready to be expanded upon!