Congratulations! You’ve learned how to build and organize a React+Redux application with multiple slices of state.

By completing this lesson you now know:.

* The action.payload property is used to hold additional data that the reducer might need to carry out a given action. The name payload is simply a convention and its value can be anything!
* The spread syntax (...) and array methods such as .map(), .slice(), and .filter() can be used to immutably update the state of a complex app.
* *Reducer composition* is a design pattern for managing a Redux store with multiple slices.
* The *root reducer* delegates actions to *slice reducers* that are responsible for updating only their assigned slice of the store’s state. The root reducer then reassembles the slices into a new state object.
* combineReducers() is a method provided by the redux library that accepts a collection of reducer functions and returns a rootReducer that implements the reducer composition pattern.
* In a Redux application, slice reducers are often written in separate files. This pattern is known as [Redux Ducks](https://github.com/erikras/ducks-modular-redux).

In the Recipes application you completed in the final exercise, the store is passed from the entry point (**index.js**) through the main <App /> component as a prop. The <App /> component can then pass the slices of the store’s state to its sub-components.

This approach is called “prop drilling” or “prop threading” because the props are “threaded” through the top-level component in order to get them to the presentational components. This isn’t ideal considering that the top-level component doesn’t make use of those props. In the next lesson, you’ll learn how you can use the react-redux library to avoid “prop threading” and more tricks for building robust React+Redux applications!