Lab 3: Redux in depth

> A continuation of day two's lab with more hands-on work with Redux, solidifying the Redux principles with actual practice.

Presentations

- Redux in Depth (/react-redux-fundamentals/pdf/RRFDay3.pdf)

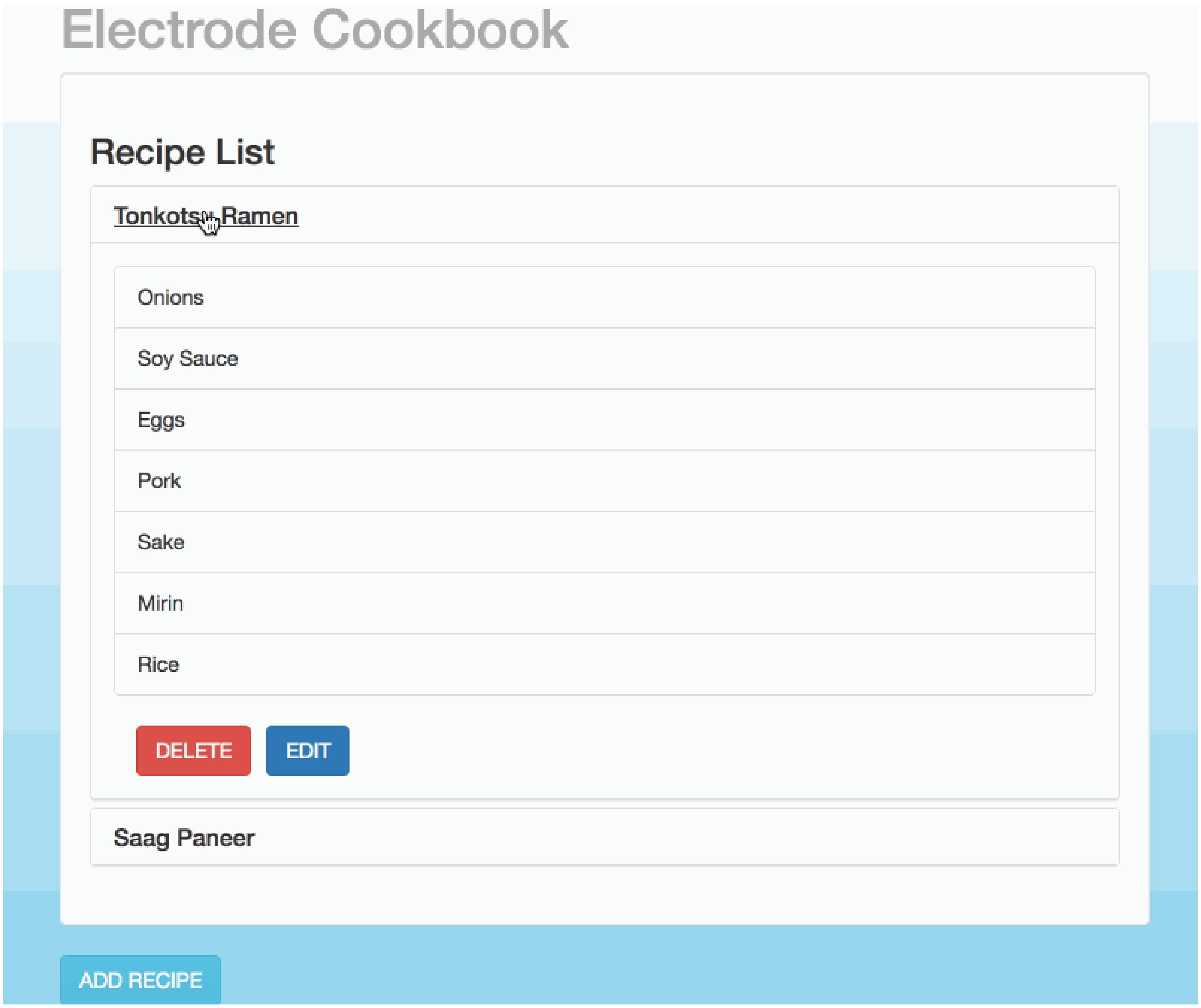
Lab Overview

In this lab, we will focus on finishing the lab that we started yesterday. The emphasis today is to help everyone get a firmer grasp on Redux concepts with some actual practice.

What you will learn:

- **Redux** - store, reducers, mapDispatchToProps, connect, dispatch, actions. We will often times refer to Redux - Quickstart (/react-redux-fundamentals/guides/redux-quickstart/).

Here's the same screencap from yesterday for easy reference:



Milestone 0: Getting Started

If you didn't finish yesterday's lab be sure to start from the reference implementation provided as today's lab will assume that yesterday's was finished.

Milestone 1: Redux - Connect Delete

Continuing where we left off in yesterday's lab, let's now use Redux actions and your `recipes` reducer to add more functionality to the application. The goal of this milestone is to connect the delete button in `RecipeDetail`; this should remove the currently displayed recipe from the dataset and update the UI. This will be your first taste of writing to the Redux store!

Doing this will take quite a few steps. Here is some guidance that will help you complete this milestone.

- First off, you will have to define the `mapDispatchToProps` function inside your `Cookbook.jsx` file. Inside `mapDispatchToProps` you will map `dispatch` - a Redux store method - to a function that will then be accessible inside your Cookbook.jsx component via `props`.
- `dispatch` will then send an `action` to the Redux store - you can give the action type the name "DELETE_RECIPE", and for the action payload you can pass an `id` which is used to identify the actual recipe that we want to delete. Read here for more details on `mapDispatchToProps`, and here for more information on actions.
- Note that inside the `Cookbook` component we don't know the `id` of the recipe which we want to delete, for that reason the id will effectively be passed from the `RecipeDetail` component when the delete button is clicked.
- You will still need the `id` as a parameter inside your `deleteRecipe` function and as part of the action payload.
- Be sure to pass `mapDispatchToProps` to the `connect` function as the second argument after `mapStateToProps`.
- Remember that `mapDispatchToProps` returns a function that is then accessible via `props` - so once you define it, it will be accessible via `this.props.deleteRecipe`, which you can then pass down the component hierarchy - down all the way to `RecipeList` which in turn handles the onClick for the delete button - so you will pass it from the `Cookbook` component down to the `RecipeList` first and then to the `RecipeDetail` component.
- You will have to associate an `id` with each recipe. In order to do this go inside `RecipeList` and add an `id` as a second argument to the `map` function. Read here for more details on `map`. Be sure to pass that `id` as a prop from `RecipeList` to `RecipeDetail`.
- use `console.log` as a sanity check to see if the function defined inside `Cookbook.jsx` is actually being called
- Finally implement the reducer inside your `reducers/index.jsx` file to actually "receive" and handle the `DELETE_RECIPE` action.
- The `reducer` is where you will actually delete the recipe from the Redux state. You can use the `filter` function for this. Read here for more information on reducers. Also use console.log inside your reducer file to make sure you are hitting it when your delete function is called.

> Remember that the `RecipeDetail` component should have the button but the action should be dispatched by a parent component. You will have to pass a function to the child component as a prop.

Milestone 2: Modal for Add / Edit

Let's start adding functionality for the add (in `Cookbook`) and edit (in `RecipeDetail`) buttons. The first step is to connect a modal popup dialog to both add and edit (as shown in the screencap). You can use the react-modal component for this. We will build out the functionality within the modal component in the next step.

Milestone 3: Redux - Connect Add / Edit

Both the add (in `Cookbook`) and edit (in `RecipeDetail`) buttons have very similar functionality, in that the fields are the same and they can be displayed in a similar fashion. Here are some guidelines to follow for the modal popup for both add and edit:

- There should be an editable field for the recipe's name.
- There should be an editable field for the ingredient list, too, but you can make things a bit easier by using a single field and a comma-delimited list. The app will need to create the list from the ingredient array when populating the field and parse it to an array when saving.

Milestone 3.5: Redux - Connect Save/Cancel

Inside the modal for add/edit, there should be a `Save` button which will save any new recipes or edits to existing recipes. You will also need to add a `Cancel` button.

Milestone 4: Persistence

Finally, persist the data between restarts. You can populate the storage initially with your mock data, but `Cookbook` should retrieve the data via redux and persist any changes made via the Edit, Add, and Delete functions. You can use simple local persistence via the file system, using a helper module such as redux-persist or redux-localstorage.

Resources

- Electrode Public Documentation
- Redux Basics
- ES6 Study Guide
- React Study Guide