Lab 2: React + Redux

> A recipe list project that helps gently introduce Redux, and how it ties in with React. Create components from scratch, and implement a set of requirements.

Presentations

- Components and Redux Intro (/react-redux-fundamentals/pdf/RRFDay2.pdf)

Lab Overview

In this lab, we will build an Electrode `Cookbook` app. The app should display a list of recipe names, any one of which can be expanded to display the recipe's ingredients. Then, we will integrate Redux so that our recipe data is stored inside Redux. All of the milestones in this lab are required - it should be feasible to get through them in the time allotted. If you are stuck at any point don't hesitate to ask for assistance. This lab will be challenging as we will have to wire up much of the app ourselves - so we will not be able to copy and paste everything.

Also, the expectation is not to have a fully functional application as shown in the screencap below. Instead, we will break down this lab into two days - today and tomorrow so that you can get something close to what's shown in the screencap.

Required concepts before starting:

- Javascript - ES6, node, npm, chrome debugger
- React - components, classes, props, JSX

> If any of these concepts above need a refresher, please follow the links before getting started with the lab.

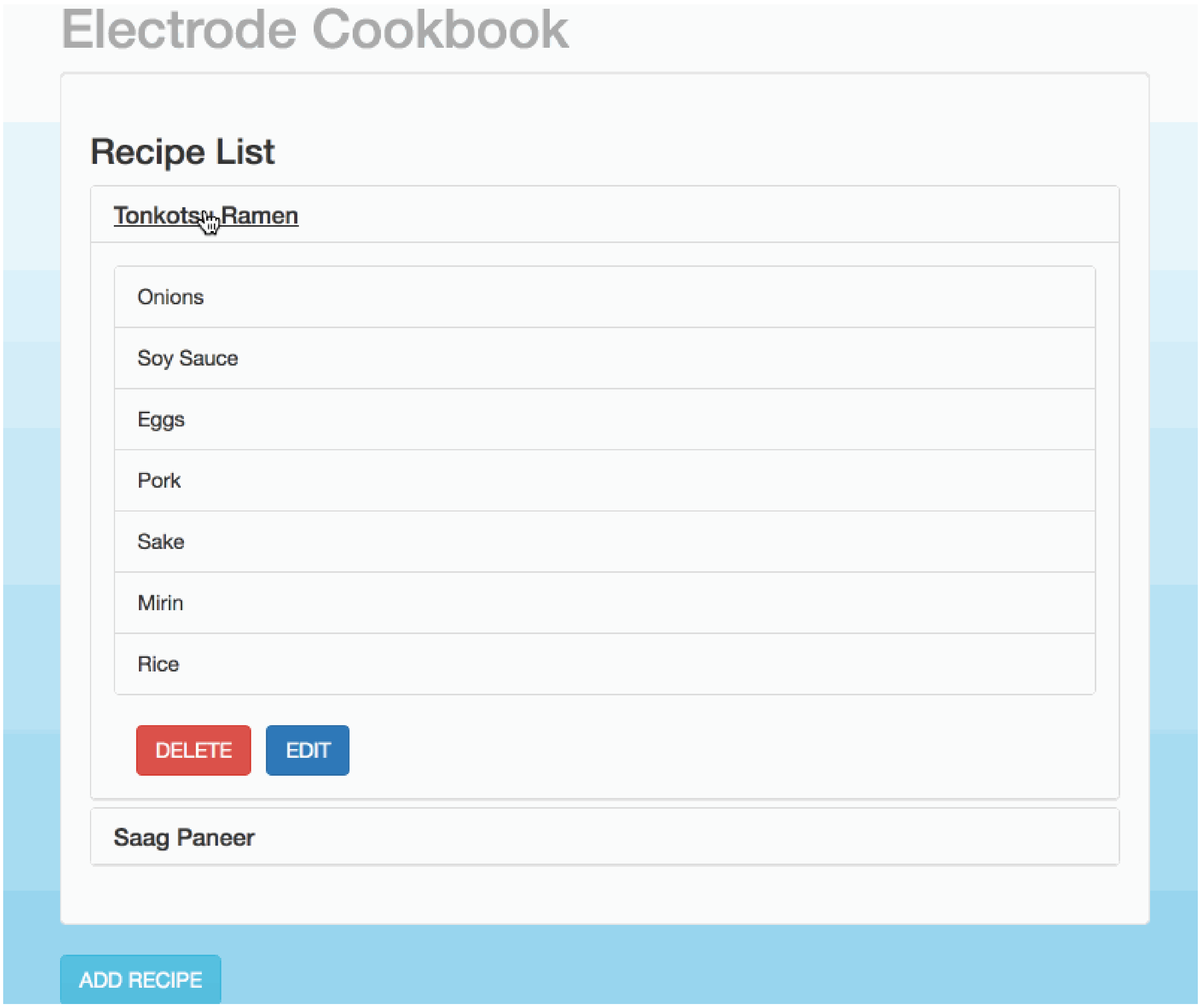
What you will learn:

- Electrode - starting an app from scratch
- **React** - passing props, using map, further understanding of components.
- **Redux** - store, reducers, mapStateToProps, connect. We will refer to Redux - Quickstart (/react-redux-fundamentals/guides/redux-quickstart/).

Why we are doing this lab:

Maintaining state on the front-end is a common problem for many apps. Redux is a powerful new paradigm for managing state in a performant, functional, and maintainable way. The learning curve can be steep because it has it own vocabulary and design patterns. React and Redux are now essential parts of the front-end toolbelt for tech companies around the world. Here is a list of some companies using them. Notable ones being Facebook, Airbnb, Reddit, Netflix, and of course Walmart!

Here's a screencap of what the finished product might look like:



> A note about styling: You'll want to add at least *some* CSS styling as you go along, but don't spend too much time on this at all. You want just enough styling to make the controls appear cleanly, but not so much that you get bogged down. It's not important that your app look exactly like the above screencap. The focus should really be on functionality and the concepts within React and Redux.

Milestone 1: Create `Cookbook` App

First, let's scaffold a new Electrode app using ignite. We'll use the `electrode` generator to make a new app:

```
$ ignite generate-app
```

The generator will prompt you for input values. You can accept the defaults in most cases, but we suggest using the following values for three of the prompts:

- Application Name: **`cookbook`**
- Which framework for the server: **`HapiJS`**
- Would you like to make a Progressive Web App: **`No`**

When the scaffolding is complete, make sure you can run the app:

```
$ cd cookbook
$ clap dev
```

The name of the parent directory above is the same as the application name we entered (ex. cookbook).

When you see the `Hapi.js server running` message, open a browser to see the app: http://localhost:3000

> Before we move on, let's open the `cookbook` folder in our editor and take a look at the folder structure. In the `src` folder we can see `client` and `server`. In `client` we find `components` and `routes.jsx` which are core to React. We can also see `actions` and `reducers` which are core to Redux. On the `server` side we see `views/index-view.jsx` which we will use for our initialState.

Milestone 2: Create Component Stubs

Next we'll create stubs for our primary components. These are just empty components for now but will help us get the app laid out and define the functional areas.

Create the `RecipeDetail` component which will display the recipe data as `client/components/recipe-detail.jsx`:

```
import React, {Component} from "react";

class RecipeDetail extends Component {
  constructor(props) {
    super(props);
  }

  deleteRecipe() {
    this.props._deleteRecipe(this.props.id);
  }

  editRecipe() {
    this.props._editRecipe(this.props.id);
  }

  render() {
    return (
      <div>
        <div>This is the RecipeDetail component</div>
        <div>
          <button type="submit" onClick={this.deleteRecipe.bind(this)}>DELETE</button>
          <button type="submit" onClick={this.editRecipe.bind(this)}>EDIT</button>
        </div>
      </div>
    )
  }
};

export default RecipeDetail;
```

Next, create the `RecipeList` component, which will use `RecipeDetail` to display all the recipes, as `client/components/recipe-list.jsx`:

```
import React, {Component} from "react";
import RecipeDetail from "./recipe-detail";

class RecipeList extends Component{
  constructor() {
    super();
  }

  deleteRecipe(id) {
    this.props._removeRecipe(id);
  }

  editRecipe(id) {
    this.props._modifyRecipe(id);
  }

  render(){
    return (
      <div>
        <h3>Recipe List</h3>
        <RecipeDetail />
      </div>
    );
  }
};

export default RecipeList;
```

> The _ (underscore) is Walmart style convention for an event handler.

We'll need two form components, one each for adding and editing a recipe.

Create the edit component at `client/components/edit-recipe-form.jsx`

```
import React, {Component} from 'react';

class EditRecipeForm extends Component {

  render() {
    return (
      <div>EditRecipeForm</div>
    );
  }
};

export default EditRecipeForm;
```

Create the add component at `client/components/add-recipe-form.jsx`

```
import React, {Component} from 'react';

class AddRecipeForm extends Component {

  render() {
    return (
      <div>AddRecipeForm</div>
    );
  }
};

export default AddRecipeForm;
```

Finally, let's create the `Cookbook` component, which will give us our main view at `client/components/cookbook.jsx` :

```
import React, {Component} from "react";
import RecipeList from "./recipe-list";
import AddRecipeForm from './add-recipe-form';
import EditRecipeForm from './edit-recipe-form';

class Cookbook extends Component {
  constructor(props) {
    super(props);
  }

  addRecipe(id) {
    this.props._addRecipe();
  }

  render(){
    return (
      <div>
        <RecipeList/>
        <button onClick={this.addRecipe.bind(this)}>Add Recipe</button>
      </div>
    )
  }
};

export default Cookbook;
```

Now let’s add a new route to test `Cookbook` . Make the following changes the contents of `src/client/routes.jsx` .

Import the `Cookbook` component at the top of the file

```
import Cookbook from "./components/cookbook";
```

Locate the `routes` array and the `/cookbook` route after the route for the `Home` component.

```
const routes = [
  {
    path: "/",
    component: withRouter(Root),
    init: "./init-top",
    routes: [
      {
        path: "/",
        component: Home,
        exact: true
      },
      {
        path: "/cookbook",
        component: Cookbook
      }
    ]
  }
];
```

> Notice how we `import` components. This is ES6 syntax and doesn't require that we specify the filetype (ex. jsx). Also notice how the components all have a `render` method. This is a required lifecycle method for every React class-based component.

At this point, you should be able to run `clap dev` and see the new components at http://localhost:3000/cookbook. We'll leave the demo code for the `home.jsx` component as-is for now so we can refer to it later.

You should see all the components except the two forms on the screen. However none of the components are connected yet to display data and the buttons won't do anything. You'll have to take it from here as we won't be providing the code for accomplishing the remaining milestones. Do the best you can, work together, and ask questions when you get stuck.

Milestone 3: Add Recipe Data

You should pass data between components using `props`. Per Facebook's "Thinking in React" documentation, `props` are best explained as "a way of passing data from parent to child." That's it. In essence, props are just a communication channel between components, always moving from top (parent) to bottom (child). React Study Guide - Props

At this point, you need to add some mock recipe data to start working with. Use a small dataset, declared as an array in `cookbook.jsx` (parent) so you can pass it to the nested component `recipe-list.jsx` (child).

Each recipe object should minimally have a name and a list of ingredients, and you'll need at least two to work with. Add this to `cookbook.jsx`:

```
const recipes = [
  {
    title: 'Tonkotsu Ramen',
    ingredients: ['Onions', 'Soy Sauce', 'Eggs', 'Pork', 'Sake', 'Mirin', 'Rice']
  },
  {
    title: 'Saag Paneer',
    ingredients: ['Spinach', 'Onion', 'Ginger', 'Garlic', 'Vegetable Oil', 'Milk', 'Lemon Jui
ce', 'Garam Masala']
  }
];
```

You can add the `recipes` declaration outside your class declaration and it will still be accessible inside the class. Now that you have `recipes`, pass the array to the child component `recipe-list.jsx` using `props`. Within your RecipeList, you should now be able to access that array using `this.props` - try rendering some part of the `recipes` array from RecipeList as a sanity check.

Milestone 4: Redux - Initialize Data

Now that you've got a feel for how data should flow in React through props, let's refactor a bit to bring Redux into the picture. Instead of initializing your data within `cookbook.jsx`, you should now initialize the data using your Redux store.

- You can go to `src/client/app.jsx` to see how the Redux store is initialized in the boilerplate code that comes with the existing scaffolding. The pattern being used here enables server-side rendering (SSR) and hot module reloading (HMR).
- Notice that the `createStore` uses the `rootReducer` and an `initialState` to construct the store. In this case `initialState` is used for SSR and is used by electrode. The first argument to `createStore`, is the important one. It is how the shape of the redux store is being defined.
- In order to add our recipes to the redux state we need to write a reducer for that branch of the state tree. Let's use the existing reducers in `src/client/reducers/index.jsx` to help us write our own. Notice that reducers take the current state an anction and return a changed state, the current state or a default state.

Now we will namespace our the data with a `reducer` and use `combinerReducers` to add it to the top level `rootReducer` being used by `createStore`. Our reducer will take care of supplying our inital cookbook state–the default recipes.

- Store and namespace the data with a `reducer` and `combinerReducers`. Go to this link Redux - Initializing a Reducer (/react-redux-fundamentals/guides/redux-quickstart) to learn how to initialize a Reducer. Keep in mind the examples are for a cart and not a recipe list.
- For this milestone, your `recipelist` reducer only needs to handle the `default` case. Don't worry about defining any action types or writing an if/else or switch statement. Just follow the pattern and return the

current state or initial state. The reducer isn't doing anything special just yet, but we do need to create it since Redux uses the registered reducers to construct the state tree.
- We'll get into `actions` and further building out the reducer tomorrow.
- Connect the data to your components with `connect` and `mapStateToProps`. Check out corresponding sections in the React Quickstart guide.

You should see the same content in your browser as the last milestone; however, the key distinction is that the data is now being stored in Redux. Redux will allow us to easily interact / modify the data, persist, debug, and many other things.

Milestone 5: Display Recipe Data

`RecipeList` should display just the recipe names in a vertical list. When the user clicks on one of the names, it should expand to show the ingredients. You can use an accordion-style react component for this; there are many to choose from, but try to use something that is implemented in React–try to avoid using jQuery or a similar Javascript library if you can. We recommend react-sanfona.

> Notice in `react-sanfona` how they use `.map` to render multiple items from a list. This is a common pattern in React and JSX. You can also read more here: .map to render lists in React

Use the same `.map` pattern to render all the ingredients in the details component. Be sure to also use the `RecipeDetail` component inside your Accordion to render the ingredients for each recipe. **You should now be able to see all your recipes and ingredients displayed on the screen in an accordion.**

To implement the accordion, you'll want one overall `<Accordion>` element, and one `<AccordionItem>` for each recipe.

Resources

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