React – Redux

In this lab, you will modify an existing React app to use Redux.

# Objectives

In this lab, you will

* Modify an existing React app
* Create a Container for an existing Status Component
* Modify the StatusContainer to use Redux
* Change references to <StatusContainer> from <Status>
* Run the app

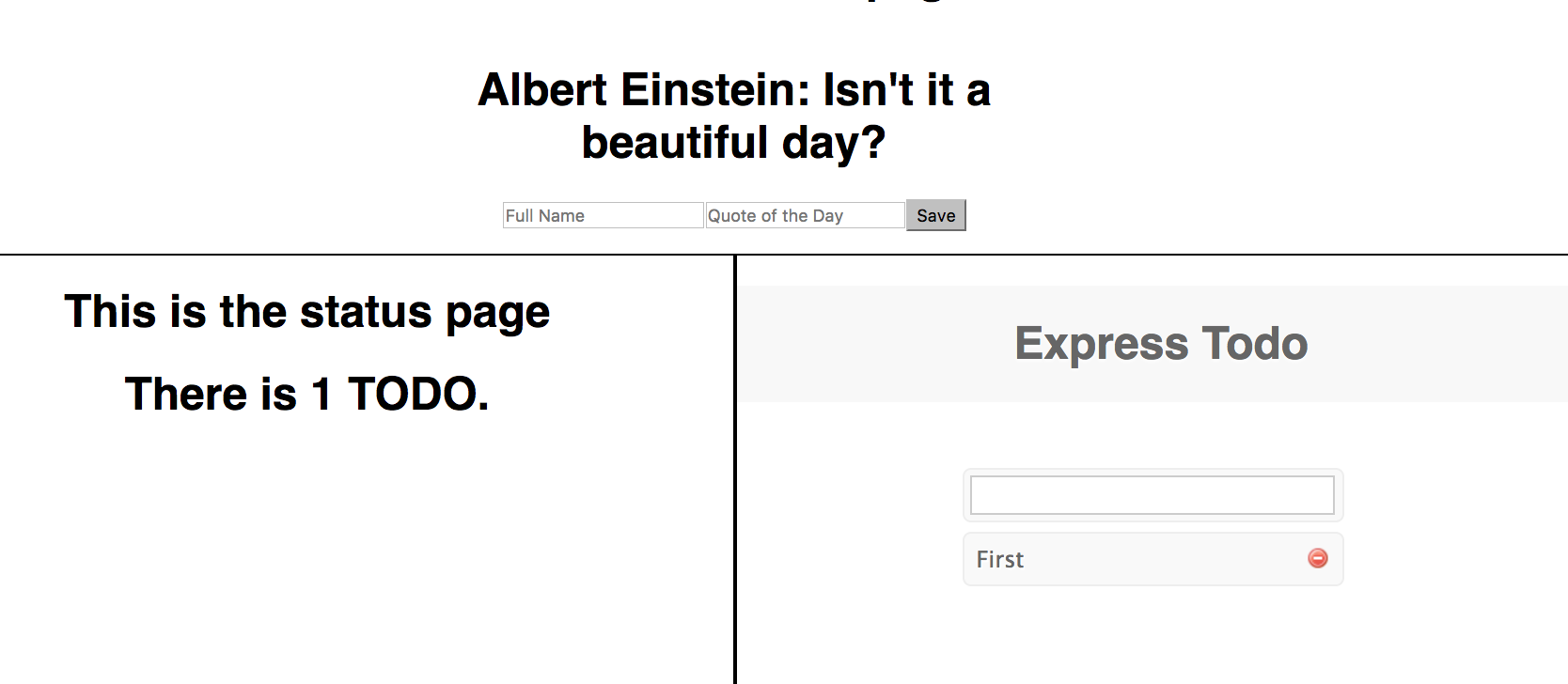
# Run the Solution app to verify correctness

1. Go to the solution/react-app folder and start the application. It should display an app with several components. The <Status> component displays the number of TODO items while the <Todos> component displays the actual TODO application. In addition, the <Quote> app displays the name and quote of the day. Start the app and view the page at <http://localhost:3000>

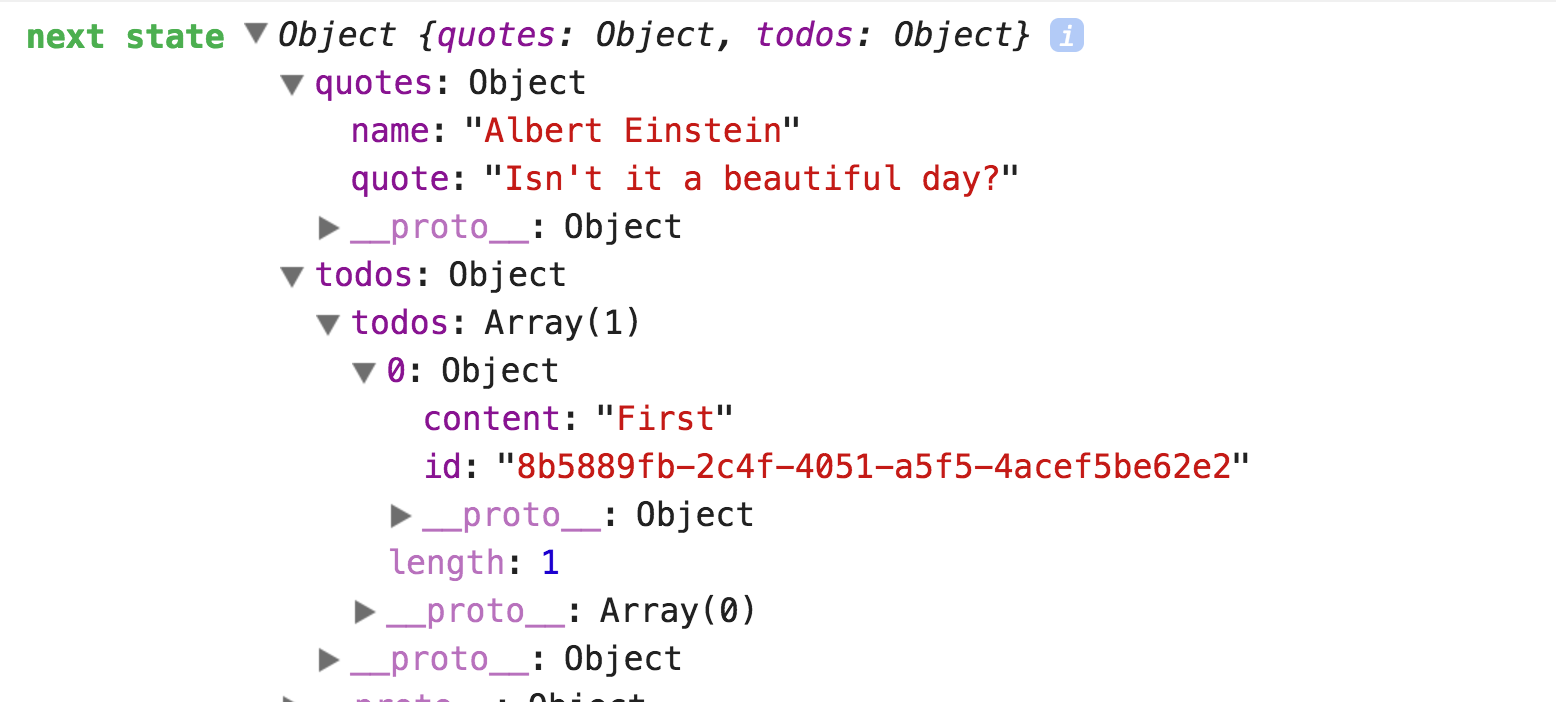
yarn install

yarn start

1. You should see the following:



1. The above shows three Container Components. The <Quote> component allows you to enter a name and quote. The <Status> displays the number of TODO items. The <Todo> component allows you to enter and remove the TODO items.
2. Because we use Redux, each of these components are independent of each other and could be placed on any page with no changes. Way cool!!!
3. The only connection is the Redux store, containing the STATE of the application. The image below comes from the Redux console display.



1. In the above, the STATE contains two different, independent namespaces, quotes and todos. The quotes namespace contains the current name and quote. The todos contains the current array of TODO items.
2. Our Redux Container Components manage this state (called the Redux store). We use Redux to map the state properties into the components attributes and map the controller logic (called Actions) into callback attributes.

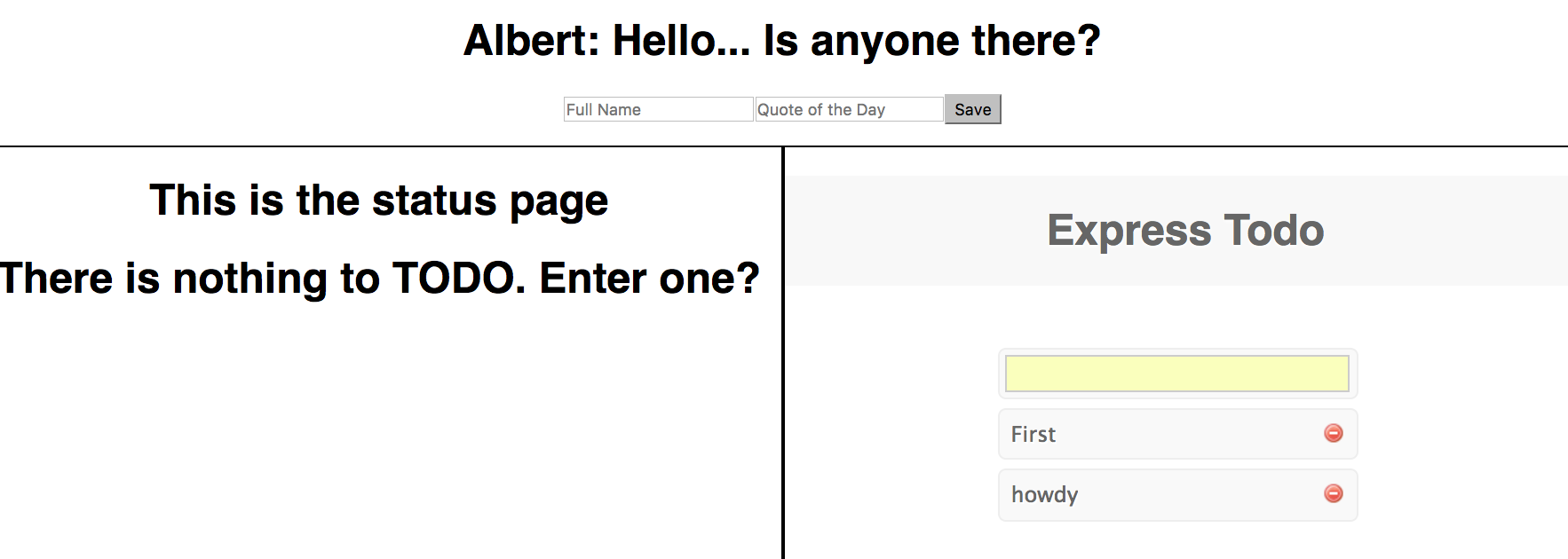
# Start the Lab Exercise

1. Stop the solution.
2. Go to the lab/react-app folder and start the lab with:

yarn install

yarn start

1. You should see the following when you open <http://localhost:3000> Actually, the string for “Albert: Hello… Is anyone there?” only shows up when you enter the name and quote.



1. Notice the bug!!! The status page does NOT display the number of TODO items. It is not yet hooked up to the Redux system.

# Redux Background

1. Redux requires several configuration parts.
2. The Action contains the interface between the UI Components, the Business Logic, and the state. The UI calls the Action. The Action calls the Business Logic and creates the state changes. The Action calls the Reducer with a command object containing the updated state data.
3. The Reducer manages the global state for the application. It accepts state changes from the Action and updates the global state. It is a pure function with no asynchronous parts.
4. The Container maps the global state data (called the Redux store) into UI Attributes and maps the UI callback attributes to the Actions.
5. The Redux Store component combines the Reducers into a single store of state data.
6. The Redux <Provider> Component makes the store available to its child components.
7. Most of the above parts are boiler plate code. Each namespace (like the TODO namespace) requires a single Action and Reducer which defines the interface TO the business logic and puts the results into the global STATE.
8. Each UI component that wants to USE the Redux store must have a Redux Container wrapped around it.
9. After creating the Redux store, the work is easy. Just create Redux Component wrappers around our UI Components.

# The <Quote> Component

1. The <Quote> Component, src/quotes/Quote.js, gets the name and quote from the user and uses the saveQuote prop to return the data to the Container as shown below:



1. In the above, lines 26-30 define the required attributes.
2. In the above, line 17 displays the current name and quote values passed into the component by the Container.
3. The Redux Container, <QuoteContainer>, must map the two inputs and the callback to attributes on the <Quote> View Component as shown below:



# The TODO Container

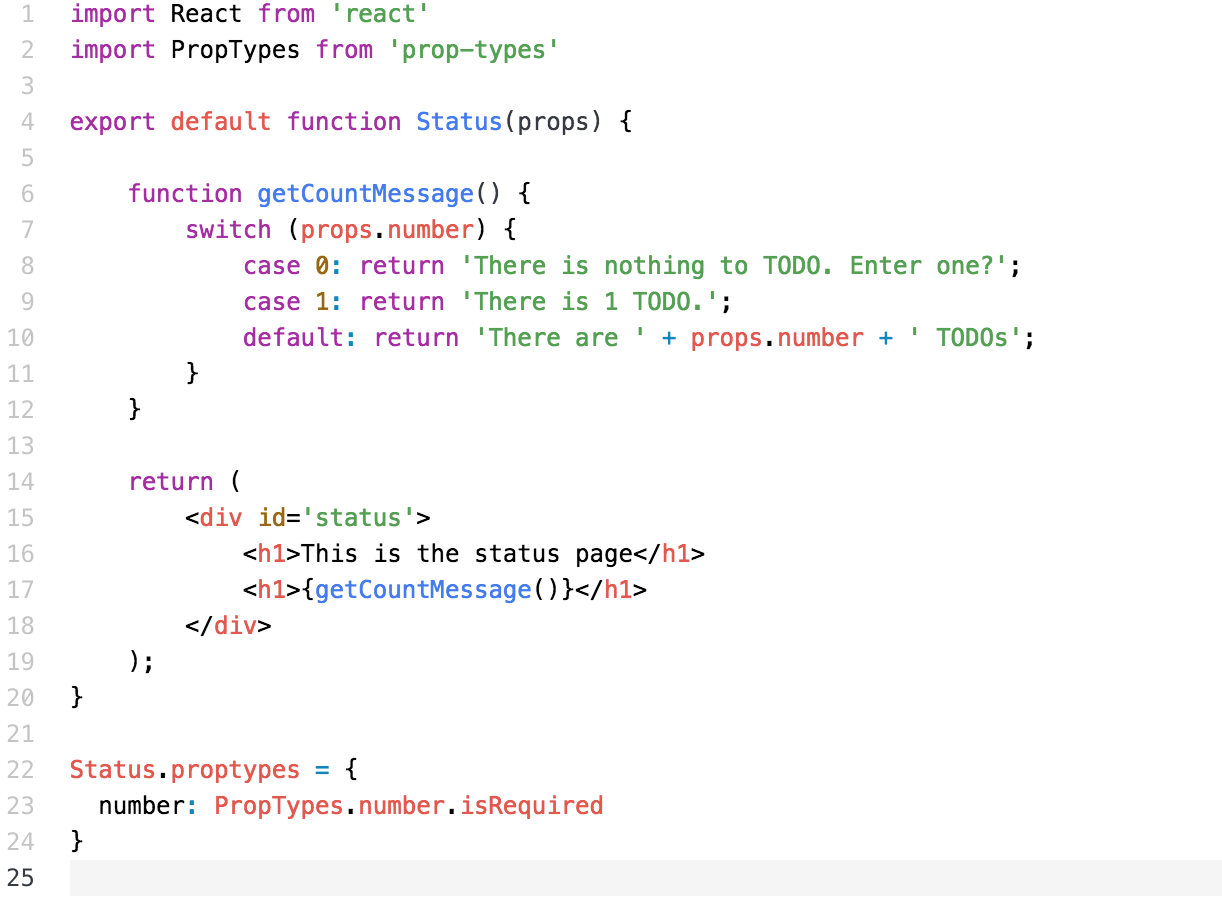
1. The <TodoContainer> does a similar mapping as shown below:



1. In the above, in lines 32-35, we wrap the <TodoList> component with the Redux store.
2. In lines 16-29, we map the callback attributes, addTodo, update, and remove, to the TodoAction functions.
3. In lines 7-11, we map the todos array to the input attribute, todos.

# The <Status> Component

1. Out task in this lab is to wrap the <Status> View Component with the Redux store.
2. The current component, src/components/Status.js, is a normal Presentation Component as shown below:



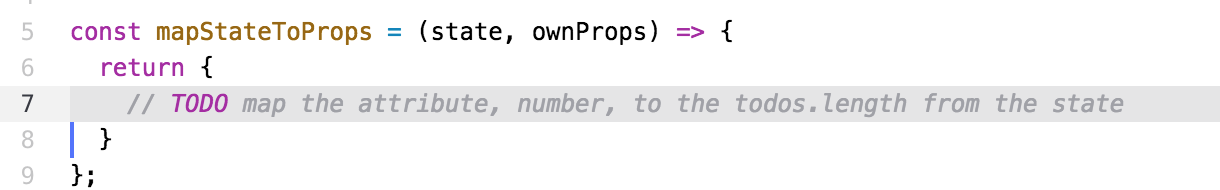
1. In the above, line 23 defines the one input property, number. The problem is that nothing sets this value whenever the todo list changes.

# Edit the <StatusContainer> Component

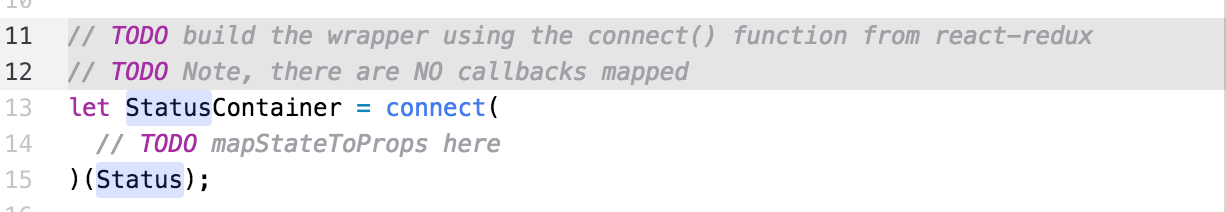
1. The <StatusContainer> wraps the <Status> component, connecting it to the Redux store. Add the pieces below:



1. In the above, import the component from the same folder.



1. In the above, map the number attribute to the todos length. Remember to reference the ‘todos namespace in the state.



1. In the above, connect the mappers to the Status component.
2. QUESTION: If there were callbacks, how would we map them?

# Change References to <Status>

1. In the src/components/Home.js change the references to <Status> to the new <StatusContainer>
2. Restart the app and add some TODO items.

# Extra Credit

1. Someone wants to display the quote data in a different, independent Component. We must now inject the quote data from the state into another Component.
2. Create a new component, QuoteOfTheDay, in the file, src/quotes/QuoteOfTheDay.js.
3. The Container Component uses Redux to inject the current quote data into the Presentation component.
4. The output should look like the entry below:



1. Since this is so simple, create the Presentation Component and the Redux mapping in the same file.
2. Modify the <Layout> component to display the new component.

Congratulations. You have completed this lab.