cd C:\Users\{your\_username}\sandbox

git clone [https://github.com/genma123/simple-react-example.git](https://github.com/genma123/nodeJSBootcampReactWorkshop.git)

cd simple-react-example

npm install

npm run start

Verify that it is running on port 3000. Its functionality should be self-explanatory. Then kill the server.

Now install Redux and Immutable:

npm install redux --save

npm install react-redux --save

npm install immutable --save

~~Modify to use Redux~~. Use this example (which was used in the PowerPoint presentation) as an example:

<https://github.com/genma123/tic-tac-toe>

Another example to look at is here:

<https://www.sitepoint.com/how-to-build-a-todo-app-using-react-redux-and-immutable-js/>

Good Luck!

UPDATE: **This (modifying from scratch to use Redux) has turned out to be too much to expect**, therefore a solution has been posted to GitHub under the “solution” branch:

<https://github.com/genma123/simple-react-example/tree/solution>

Here are the steps you would need to perform:

1. Create actions.js. Use the “Tic Tac Toe” example identified above as guidance. To make this easier, be aware that here are two actions to handle:
2. Clicking on the Add a Square button, the payload must be:
3. text — String
4. color — String
5. key — number
6. Clicking on the X on a square to remove it, the payload is just the index (ordinal position from 0 through N).
7. Try to implement actions.js on your own before looking at the solution
8. Then add these from the solutions branch (copy from GitHub):
9. containers.js (Note that both state and actions are mapped to props):

import { connect } from 'react-redux';

import { addSquare, removeSquare } from './actions';

import App from './App';

export const WrappedApp = connect(

function mapStateToProps(state) {

return { appState: state };

},

function mapDispatchToProps(dispatch) {

return {

addSquare: (text, color, key) => dispatch(addSquare(text, color, key)),

removeSquare: id => dispatch(removeSquare(id))

}

}

)(App);

1. reducer.js:

import Immutable from 'immutable';

import \_ from 'lodash';

export default function(appState, action) {

switch(action.type) {

case 'ADD\_SQUARE':

const square = { text: action.payload.text, color: action.payload.color, key: action.payload.key };

var squares = appState.get('squares');

const statex = appState.set('squares', squares.push(square));

return statex.set('sequence', statex.get('sequence') + 1);

case 'REMOVE\_SQUARE':

const statey = appState.set('squares',

Immutable.fromJS(\_.filter(appState.get('squares').toJS(), function(t) { return t.key !== action.payload.id; })));

return statey.set('sequence', statey.get('sequence') - 1);

default:

return appState;

}

}

1. Modify the following as shown:
2. index.js:

import React from 'react';

import ReactDOM from 'react-dom';

import { createStore } from 'redux';

import { Provider } from 'react-redux';

import Immutable from 'immutable';

import reducer from './reducer';

import{ WrappedApp } from './containers';

import registerServiceWorker from './registerServiceWorker';

import './index.css';

const initAppState = Immutable.fromJS(

{ squares: [], // NOTE: becomes an Immutable List!

sequence: 0

});

const store = createStore(reducer, initAppState);

ReactDOM.render(

<Provider store={store}>

<WrappedApp />

</Provider>,

document.getElementById('root'));

registerServiceWorker();

1. App.js (Again note that both state and actions are mapped to props):

import React, { Component } from 'react';

import Status from './Status';

import Grid from './Grid';

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.addSquare = this.addSquare.bind(this);

this.removeSquare = this.removeSquare.bind(this);

}

addSquare() {

this.props.addSquare(this.\_text.value, this.\_select.value.toLowerCase(), this.props.appState.get('sequence'));

}

determineStatus() {

return this.props.appState.get('sequence') + " Squares Currently Displayed";

}

removeSquare(id) {

this.props.removeSquare(id);

}

render() {

return (

<div className="App">

<label htmlFor="text">Enter text: </label>

<span><input id="text" type="text" ref={(el) => this.\_text = el}></input></span>

<span><button onClick={this.addSquare}>

Add a Square

</button></span><span><select ref={(el) => this.\_select = el}><option>Red</option>

<option>Blue</option>

<option>Green</option>

<option>Yellow</option>

</select></span>

<Status message={this.determineStatus()}/>

{/\* NOTE 'squares' is an Immutable List, must be converted to raw JavaScript!! \*/}

<Grid squares={this.props.appState.get('squares').toJS()} deleteSquare={this.removeSquare} />

</div>

);

}

}

export default App;

1. Now the hands-on assignment. I implemented the above solution by using an Immutable.js data structure to represent the state. I did so because it is the generally accepted “best practice” when working with Redux. However, Redux does not by any means require Immutable.js.

To get some hands-on experience with this example, modify the implementation to NOT use Redux, just “plain-vanilla” JSON objects. The result should be simpler. Please try to avoid any logic that would actually mutate a JSON object “in place” as that would potentially “break” the Redux functionality.