React Redux
Design Lessons Learned

A Software Presentation From

Jeremy Zerr

Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr

Andrew Chumich

LinkedIn: https://www.linkedin.com/in/andrewchumich
Twitter: https://twitter.com/AndrewChumich

Introduction

- [ ] To gain more experience building with React + Redux, we are building a time tracking web app based on the TSheets API

- [ ] We have built other TSheets API apps in AngularJS, so this is a way for us to compare AngularJS to React + Redux for a similar app

- [ ] This project is on Github at react-redux-tsheets-app

Overview

- ☐ Brief intro to React and Redux
- ☐ Terminology
- ☐ App Design Process
  - ☐ Requirements
  - ☐ Redux State
  - ☐ Redux Actions/Reducers
  - ☐ React Components

What is React?

- [ ] Within the MVC design pattern, it is for building your View, but also to a lesser degree, your Controller
- [ ] Efficient DOM updating when data changes by using a Virtual DOM to compute differences before changing the actual DOM
- [ ] Components use one way data flow, data flow is explicit
    - [ ] Leads to using pure functions
    - [ ] Debugging is easier
    - [ ] Testing generally is easier on pure functions
- [ ] Typically see component markup built in JSX
- [ ] Sponsored by Facebook

What is Redux?

- [ ] Redux is a framework for managing the state for a web application, React components render that state
- [ ] A single data store contains the state for your app
- [ ] Your application emits an action, that defines something that just happened that will affect the state
- [ ] Reducers specify how to change the state when the action is received
- [ ] Hot reloading of code changes
- [ ] State changes can be tracked, and replayed

Why React + Redux?

- ☐ The props for React components come from the Redux store that tracks the state.
- ☐ React components react to user input and emit actions, either directly or indirectly.
- ☐ Redux handles the action by running the appropriate reducers which transform the current state into a new state.
- ☐ React components react to the new state and update the DOM.
- ☐ React components themselves are stateless (most of the time), all of the state is kept in the Redux store, one common place, for simplicity.

Terminology

What kinds of things do we work with?

React

- [ ] Component
- [ ] Props
- [ ] propTypes
- [ ] render
- [ ] JSX

```
export class Jobcode extends React.Component {
  static propTypes = {
    jobcodes: React.PropTypes.instanceOf(OrderedMap).isRequired,
    parentId: React.PropTypes.number.isRequired,
    onChange: React.PropTypes.func.isRequired,
    onChangeParent: React.PropTypes.func.isRequired,
    currentId: React.PropTypes.number.isRequired
  }

  render () {
    const {
      jobcodes,
      parentId,
      currentId,
      onChange,
      onChangeParent
    } = this.props
    return (
      <div className=''>
       <ManagedList
          list={jobcodes}
          currentId={currentId}
          onChange={onChange}
          onChangeParent={onChangeParent}
          parentId={parentId}
          isSelected={isSelected}

        />
      </div>
    )
  }
}
```

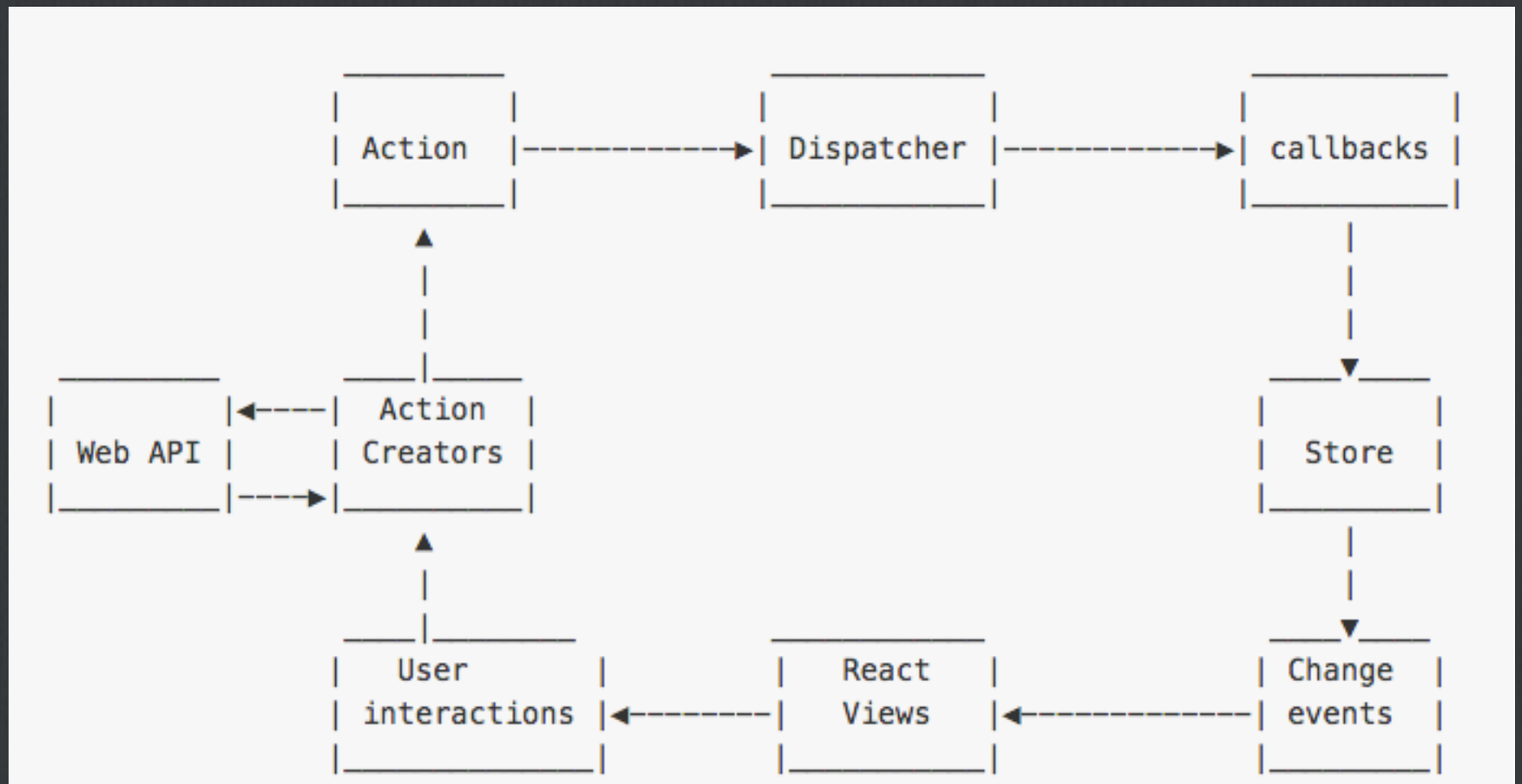
Redux

- [ ] Action
- [ ] Action Creator
- [ ] Dispatch
- [ ] Reducer
- [ ] Store

```
// ------------------------------------
// Constants
// ------------------------------------
export const GET_JOBCODES = 'GET_JOBCODES'
export const SET_JOBCODES = 'SET_JOBCODES'
export const UPDATE_PARENT_IDS = 'UPDATE_PARENT_IDS'

// ------------------------------------
// Actions
// ------------------------------------
export const updateParentIds = createAction(UPDATE_PARENT_IDS, (value = 0) => value)
export const setJobcodes = createAction(SET_JOBCODES, value => value)
export const getJobcodes = () => {…
}

export const actions = {
  setJobcodes,
  getJobcodes,
  updateParentIds
}

// ------------------------------------
// Reducer
// ------------------------------------
export default handleActions({
  SET_JOBCODES: (state, { payload }) => {
    return JobcodeUtils.set(state, payload)
  },
  UPDATE_PARENT_IDS: (state, { payload }) => {
    return JobcodeUtils.updateParentIds(state, payload)
  }
}, JobcodeUtils.getInitialJobcodes())
```

- [ ] from src/redux/modules/jobcodes.js

Redux


Action
Dispatcher
callbacks
Web API
Action Creators
Store
User interactions
React Views
Change events

React + Redux

- ☐ mapStateToProps
- ☐ mapDispatchToProps
- ☐ connect

```
const mapStateToProps = (state) => ({
  timesheet: state.timecard,
  jobcodes: state.jobcodes
})
const mapDispatchToProps = (dispatch) => {
  var allActions = Object.assign(
    {},
    timesheetActions,
    timecardActions,
    jobcodeActions
  )
  return bindActionCreators(allActions, dispatch)
}
export class TimecardView extends React.Component {
  static propTypes = {…
  }

  componentDidMount () {…
  }

  render () {…
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(TimecardView)
```

- ☐ from View component src/views/TimecardView.js

Design Process

How do we go about designing a React + Redux app?

App Design Process

1. What do you want your app to do? Mockups, UI

2. Design your state tree. What needs to be tracked with the state within the app?

3. How will the state change? Write actions and reducers.

4. Which UI components will call the actions? Call actions from container components

1) Our App Requirements

- [ ] A time tracking web application based off of the TSheets API
- [ ] Timecard page that allows clocking in and out of a Timesheet, selecting Jobcodes, and defining Notes, then saving via the API
- [ ] Timesheet list page that shows your timesheets
- [ ] Ability to Add/Edit individual Timesheets, similar to what you can do on a Timecard, except without the Clock In/Out.

1) App UI and Pages

- [ ] Three pages, a Timecard page, Timesheets List page, and a Timesheet Add/Edit page
- [ ] Timecard page:
  - [ ] For doing Clock In/Out, tracking your current activity
- [ ] Timesheets List page:
  - [ ] Shows your timesheets, past and present
- [ ] Timesheet Add/Edit page:
  - [ ] For adding new and editing existing Timesheets

Timecard Page

- [ ] Timecard page has:
  - [ ] Clock in/out button that changes depending on context
  - [ ] Jobcodes presented as a parent/child list, clicking on parent, shows all children
  - [ ] Notes as input field
  - [ ] When on the clock, your current time increments

2) Designing the Store

- ☐ Track the Timesheet that is on the clock, that will be the one shown on Timecard page
- ☐ We need to track all Timesheets over the last several days
- ☐ We want our Timesheet which is on the clock, the one used on the Timecard page, to live within the list of Timesheets to keep it simple to access all of the Timesheets.
    - ☐ So we chose just to save the id of the on the clock Timesheet in our state.
- ☐ We also need the entire list of Jobcodes, you need a Jobcode before clocking in
- ☐ There could be several places where a list of Jobcodes are used, both on the Timecard page, and Timesheet add/edit page. When a user picks a Jobcode, we want that to be remembered if they navigate away and come back. So we need to keep track of the currently selected Jobcode for each page separately.
    - ☐ We will save the selected jobcode id in the state, but have only one list of Jobcodes to avoid duplication.

Redux Store + Immutable.js

- ☐ React is designed for efficient DOM updating using a Virtual DOM
- ☐ Redux brings a common store used to track state
- ☐ If we turn the Redux store into an immutable object, we never change data, we only change references.
- ☐ In that case, we can avoid the Virtual DOM computation and diff, with a quick data reference check.
- ☐ The React components can then avoid doing the Virtual DOM creation and diff, because they know that if the reference is the same, no updates needed. See PureRenderMixin.
- ☐ We decided to use only Immutable.js data structures for our Redux store, and everywhere state is passed into React Components.

2) Our Redux Store

```
Map {
  OrderedMap timesheetList: {Number: {}, Number: {}, …},
  Map timecard: {
    _id: Number
  },
  Map jobcodes: {
    Map list: {Number: {},Number: {}, …},
    Map parent_ids: {
      timecard: Number,
      edit_timesheet: Number,
      add_timesheet: Number
    }
  }
}
```

Let's start coding!
Wait... never from scratch

- ☐ We used the React Redux Starter Kit
- ☐ Webpack so we can do hot reloading with the dev server
- ☐ Babel so we can use ES2015
- ☐ We had to add in Immutable.js ourselves
- ☐ Also already included Redux thunk for our async requests

3) Actions from the UI

- [ ] Action for updating fields on a timesheet within the timesheet list
- [ ] Actions for Clock in and Clock out that change a Timesheet
- [ ] Actions for loading Jobcodes from the API
    - [ ] Once Promise is complete, we fire another action to set the Jobcodes in the state
- [ ] Action for syncing a timesheet, basically clock in sends a POST, then clock out sends a PUT with an end date, which results in a completed Timesheet.
- [ ] Action for picking a jobcode id when navigating through the Jobcode selector

4) Components call Actions

- [ ] Sections of our app that we call Pages we create as a View Component, a.k.a Container component
- [ ] TimecardView container component
  - [ ] Timecard stateless component
    - [ ] Jobcode stateless component
      - [ ] ManagedList stateless component - can select button item and navigate within nested structure
    - [ ] Clock In/Out stateless component - calls clock in clock out

Components: Container (smart) + Stateless (dumb)

- Instead of one single component that takes state, calls actions, and renders DOM, we instead are choosing to design using a combination of two types of components
  - Container: passes state from Redux store as props to sub-components (container or stateless) and emits actions. Just renders sub-components. Has data concerns.
  - Stateless: no dependencies, only gets data and callbacks via props. Presentation-only. Can use other stateless sub-components, but no container sub-components
- Read more by Dan Abramov and a related Gist

TimecardView - container

- [ ] TimecardView is our container component
- [ ] Is mapped to a path via react-router in src/routes/index.js
- [ ] It initially populates the list of Jobcodes from the API
- [ ] Maps state and actions to the stateless components
- [ ] Notice callback functions for UI interactions passed down into Timecard sub-component

```
<Route path='/' component={CoreLayout}>
  <IndexRoute component={HomeView} />
  <Route path='/about' component={AboutView} />
  <Route path='/timecard' component={TimecardView} />
</Route>
```

```
<Timecard timesheet={timesheet.toJS()}
  onClockIn={() => clockIn()}
  onClockOut={() => clockOut()}
  jobcodes={jobcodes.get('list')}
  parentId={jobcodes.getIn(['parent_ids', 'timecard'])}
  onChangeJobcode={(id) => {
    updateTimecard({ jobcode_id: id })
    updateParentIds({ timecard: 0 })
  }}
  onChangeJobcodeParent={(id) => {
    updateParentIds({ timecard: parseInt(id, 10) })
  } }/>
```

Timecard - stateless

- [ ] Timecard is a stateless component

- [ ] Passes what is necessary to child components Jobcode and ClockIn

```
<Jobcode
  jobcodes={jobcodes}
  parentId={parentId}
  onChange={onChangeJobcode}
  onChangeParent={onChangeJobcodeParent}
  currentId={timesheet.jobcode_id}
/>
<ClockIn
  onTheClock={timesheet.on_the_clock}
  onClockIn={onClockIn}
  onClockOut={onClockOut}
  />
```

Jobcode - stateless

- [ ] Jobcode is a stateless component
- [ ] Passes what is necessary to ManagedList

```
<ManagedList
    list={jobcodes}
    currentId={currentId}
    onChange={onChange}
    onChangeParent={onChangeParent}
    parentId={parentId}
    isSelected={isSelected}

 />
```

ManagedList - stateless

- Designed to be generic, as there will be other data rendered in a component like this, such as another Timesheet field called Custom Fields
- Notice how we have been passing down the isSelected function
- That function determines when we select a child, that the parent also shows up as highlighted
- It was passed in via props from the Jobcode component

```
/**
 * @param  {Number} parentId - the parent jobcode id that you should go back to
 * @param  {Map} currentItem - the current jobcode
 * @param  {OrderedMap} list - list of jobcodes
 * @param  {String} style - the style to apply to the button
 * @param  {String} selectedStyle - the style to apply to the button if it is selected
 * @param  {Func} isSelected - a function that returns true if the item should be consi
 * @param  {Func} onClick - The function to call when a button is clicked
 * @return {String} - Markup to use as a list
 */
_renderList (parentId, currentItem, list, style, selectedStyle, isSelected, onClick) {
  return list.filter((v) => v.get('parent_id') === parentId).map((value, key) => {
    var s
    if (isSelected(value, currentItem, list)) {
      s = selectedStyle
    } else {
      s = style
    }
    var content = value.get('name')
    if (value.get('has_children')) {
      content += ' >'
    }
    return (
        <button
          key={key}
          style={s}
          onClick={() => onClick(value, parseInt(key, 10))}>{content}</button>
    )
  }).toArray()
}
```

Reusable Components

- [ ] Use propTypes and defaultProps

```
export class Jobcode extends React.Component {
  static propTypes = {
    jobcodes: React.PropTypes.instanceOf(OrderedMap).isRequired,
    parentId: React.PropTypes.number.isRequired,
    onChange: React.PropTypes.func.isRequired,
    onChangeParent: React.PropTypes.func.isRequired,
    currentId: React.PropTypes.number.isRequired
  }
```

- [ ] Since we are using Immutable.js, we can very explicitly check for instance of Immutable.js types, like OrderedMap and Map

What will we cover next?

- [ ] Async API operations using Redux thunk and actions/ action creators
- [ ] How to ensure that data is loaded that is needed for a React Component
- [ ] Using mapDispatchToProps and bindActionCreators to bring different action creators into React component scope
- [ ] Should you ever use React Component local state?

Thanks! Connect with Us!

A Software Presentation From

Jeremy Zerr

Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr

Andrew Chumich

LinkedIn: https://www.linkedin.com/in/andrewchumich
Twitter: https://twitter.com/AndrewChumich