Webpack React Redux

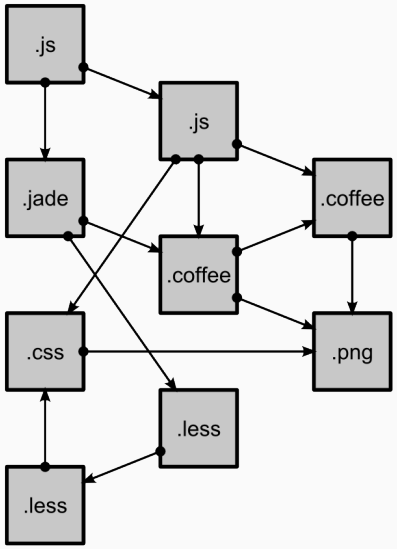
Alejandro Do Nascimento

2016

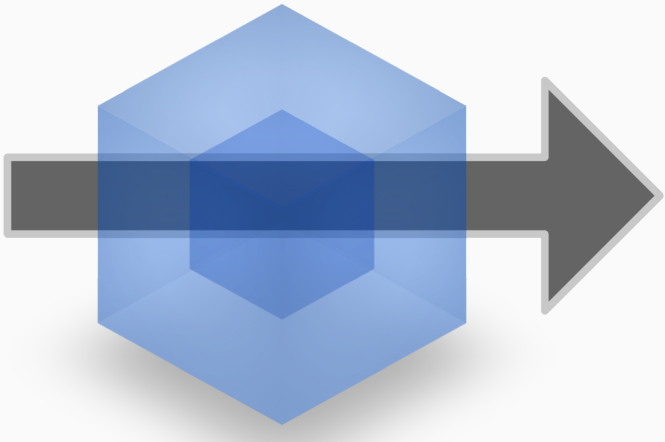
webpack

MODULE BUNDLER

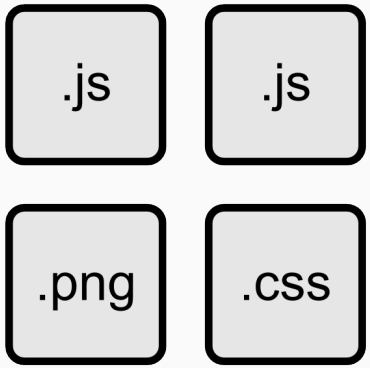
Webpack



modules
with dependencies



webpack
MODULE BUNDLER



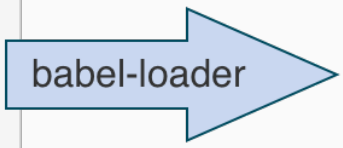
static
assets

Loaders

Webpack can only process JavaScript natively, but loaders are used to transform other resources into JavaScript. By doing so, every resource forms a module.

./App.js

```
import React from 'react'
import { Provider } from 'react-redux'
import AppRouter from './AppRouter'
export const App = ({ store }) => (
  <Provider store={store}>
    <AppRouter />
  </Provider>
)
```

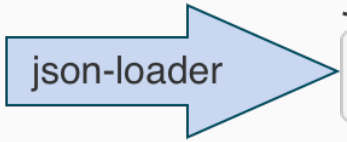


babel-loader!./App.js

```
'use strict';
Object.defineProperty(exports, "__esModule", {
  value: true
});
exports.App = undefined;
var _react = require('react');
var _react2 = _interopRequireDefault(_react);
var _reactRedux = require('react-redux');
var _AppRouter = require('./AppRouter');
var _AppRouter2 = _interopRequireDefault(_AppRouter);
function _interopRequireDefault(obj) { return obj && obj.__esModule
  ? obj : { default: obj }; }
var App = exports.App = function App(_ref) {
  var store = _ref.store;
  return _react2.default.createElement(
    _reactRedux.Provider,
    { store: store },
    _react2.default.createElement(_AppRouter2.default, null)
  );
};
```

./cats.json

```
[ "dave", "henry", "martha" ]
```



json-loader!./cats.json

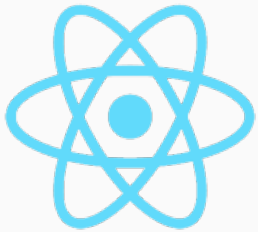
```
module.exports = [ "dave", "henry", "martha" ];
```

Config

```
module.export = {
  devtool: 'source-map',
  entry: {
    app: '/path/to/entry_file.js',
  },
  output: {
    path: '/path/to/destination/folder/',
    filename: '[name]-[hash].js'  // Creates a file app-e5fb0d5a9741647e0223.js
  },
  module: {
    loaders: [
      {
        test: /\.jsx?$/,  // For matching file names
        loaders: [ 'babel?cacheDirectory' ],
        include: [ 'scr/path/js/', 'scr/path/jsx' ]  // Path to search
      },
      {
        test: /\.scss$/,
        loaders: [ 'style', 'css', 'sass' ]
      }
    ]
  },
  plugins: [
    new webpack.DefinePlugin({
      'process.env': { 'NODE_ENV': JSON.stringify('production') }
    }),
  ]
}
```

Executing webpack

```
{
  "name": "crashui",
  "version": "1.0.0",
  "dependencies": {
    "babel-core": "^6.11.4",
    "react": "^15.2.1",
    "react-dom": "^15.2.1",
    "redux": "^3.5.2",
    ...
  },
  "devDependencies": {
    "babel-loader": "^6.2.4",
    "babel-preset-es2015": "^6.9.0",
    "babel-preset-stage-0": "^6.5.0",
    "clean-webpack-plugin": "^0.1.10",
    "standard": "^7.1.2",
    "standard-loader": "^4.0.0",
    "webpack": "^1.13.1",
    "webpack-dev-server": "^1.14.1",
    ...
  },
  "scripts": {  // Commands run in terminal. Ex: npm run build
    "build": "NODE_ENV=production webpack --config config/webpack.config.js",
    "lint": "standard \"frontend/**/*.js*\" --verbose | snazzy",
    "start": "NODE_ENV=development webpack-dev-server --config config/webpack.config.js",
    "test": "NODE_ENV=testing karma start"
  },
}
```



React

React

JavaScript library for building user interfaces

React lets you express how your app should look at any given point, and can automatically manage all UI updates when your underlying data changes.

Is declarative, which means that React conceptually hits the “refresh” button any time data changes, and knows to only update the changed parts

React

HTML

```
<body>
  <div id="app"> </div>
  <script src="./bundle.js"></script>
</body>
```

index.jsx

```
import './main.css'
import React from 'react'
import { render } from 'react-dom'
import App from './components/App.jsx'

render(<App name='Social Point' />, document.getElementById('app'))
```

App.jsx

```
import React from 'react'

export default class App extends React.Component {
  render () {
    return (
      <div> Hello {this.props.name} </div>
    )
  }
}
```

Components

```
<App name='Ainara' />
```

es5

```
var App = React.createClass({
  render: function () {
    return <div> Hello {this.props.name} </div>;
  }
});
```

es6 class

```
class App extends React.Component {
  render () {
    return (
      <div> Hello {this.props.name} </div>
    )
  }
}
```

Stateless

```
const App = props => {
    return <div> Hello {props.name} </div>
}
```

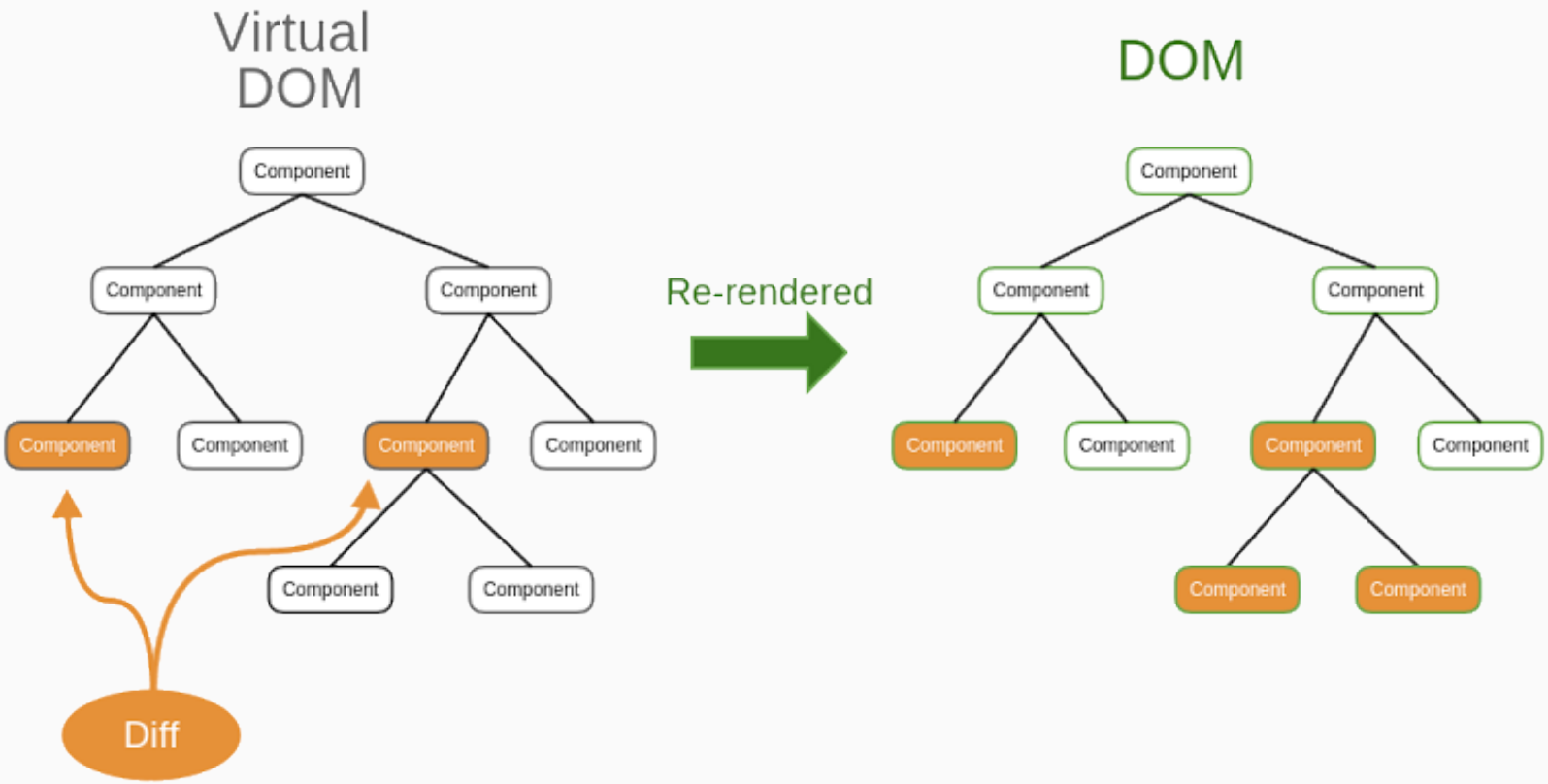
JSX

```
class App extends React.Component {
  render () {
    return (
      <div> Hello {this.props.name} </div>
    )
  }
}
```

Compile js

```
class App extends React.Component {
  render () {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    )
  }
}
```

Virtual Dom


Virtual DOM
Component
Component
Component
Component
Component
Component
Component
Component
Component
Diff
Re-rendered
DOM
Component
Component
Component
Component
Component
Component
Component
Component
Component

Anatomy of components

render()

It returns a single child element, either a virtual representation of a native DOM component or another composite component that you've defined yourself. This method is required.

props

```
class App extends React.Component {
  static propTypes = {
    name: React.PropTypes.string.isRequired
  }

  static defaultProps = {
    name: 'Stranger'
  }

  render () {
    return <div> Hello {this.props.name} </div>
  }
}

render(<App name='Alejandro' />, document.getElementById('app'))
```

Anatomy of components

State

```
class App extends React.Component {
  constructor (props) {
    super(props)
    // We have to manually bind class methods to maintain the 'this' reference
    this.increaseCounter = this.increaseCounter.bind(this)
    this.state = {
      counter: 0
    }
  }

  increaseCounter (event) {
    this.setState({ counter: this.state.counter++ })  // setState triggers a render()
  }

  render () {
    return (
      <div>
        {this.state.counter}
        <Button onClick={this.increaseCounter}>
          {this.props.name} click here and increase the counter
        </Button>)
      </div>
    )
  }
}
```

Lifecycle methods

componentWillMount()

Invoked before the initial render. Calling *setState* here will not trigger additional renders.

componentDidMount()

Invoked once after the initial render. Real DOM ref exist in this stage. Use for integration with other frameworks, timeouts, ajax call, etc.

componentWillReceiveProps(newProps)

Invoked when a component is receiving new props. Used to update state on prop changes.

```
componentWillRecieveProps (newProps) {
  this.setState({  // Does not trigger additional renders
    didIncrease: newProps.value > this.props.value
  })
}
```

Lifecycle methods

shouldComponentUpdate(nextProps, nextState)

Invoked before rendering before a new state or props changed. If false is returned the component will skip the render method.

componentWillUpdate()

Invoked immediately before rendering when new props or state are being received. You cannot use *this.setState()*.

componentDidUpdate()

Invoked immediately after the component's updates are flushed to the DOM.

componentWillUnmount()

Invoked immediately before a component is unmounted from the DOM. Used for cleanups.

Lifecycle methods

```
class Timer extends React.Component {
  constructor (props) {
    super(props)
    this.state = { secondsElapsed: 0 }
    // We only have to manually bind the tick method
    // the lifecycle methods are binded automatically by React.
    this.tick = this.tick.bind(this)
  }

  tick () {
    this.setState((prevState) => ({
      secondsElapsed: prevState.secondsElapsed + 1
    }))
  }

  componentDidMount () {
    this.interval = setInterval(() => this.tick(), 1000)
  }

  componentWillUnmount () {
    clearInterval(this.interval)
  }

  render () {
    return <div> Seconds Elapsed: {this.state.secondsElapsed} </div>
  }
}
```

Events

SyntheticEvent is a cross-browser wrapper around the browser's native event. They share the same API, except the wrapper work identically across all browsers.

Clipboard	Keyboard	Focus	Form	Mouse
onCopy	onKeyDown	onFocus	onChange	onClick
onCut	onKeyPress	onBlur	onInput	onContextMenu
onPaste	onKeyUp		onSubmit	...

```
...
increaseCounter (event) {
  this.setState({ counter: this.state.counter++ })
}
...
render () { return <Button onClick={this.increaseCounter}> Increase </Button> }
...
```

Composition

```
import MyComponent from './MyComponent.jsx'
...
  render () {
    const x = 42
    // key is a unique identifier that's required in arrays of components
    const componentsList = [
      <span key='1'> This will render </span>,
      <MyComponent key='2' text='asdf' />
    ]
    return (
      <div>
        <MyComponent prop1={x} {...this.props} />  // Renders MyComponent

        {componentsList}  // Renders all the components in the array

        {null}  // Renders a <noscript> tag
        {false}  // Renders a <noscript> tag
        {this.props.anwser === x && <p> Correct Anwser </p> }

        {[ 'Alejandro', 'Ivan', 'Yisus' ].map(name => {
          if (name === 'Yisus') {
            return null
          }
          let text = `Hello ${name}`
          return <p key=text> {text} </p>
        })}
      </div>
    )
  }
```

Controlled components

```
class MyForm extends React.Component {
  constructor (props) {
    super(props)
    this.state = { value: 'Hello!' }
    this.handleChange = this.handleChange.bind(this)
  }

  handleChange (event) {
    // Every time the user enters a new character we update the component
    // otherwise he will not see the changes
    this.setState({ value: event.target.value })
  }

  render () {
    return (
      <input
        type="text"
        value={this.state.value}
        onChange={this.handleChange}
      />
    )
  }
}
```

Redux

Redux

Redux is a predictable state container for JavaScript apps.

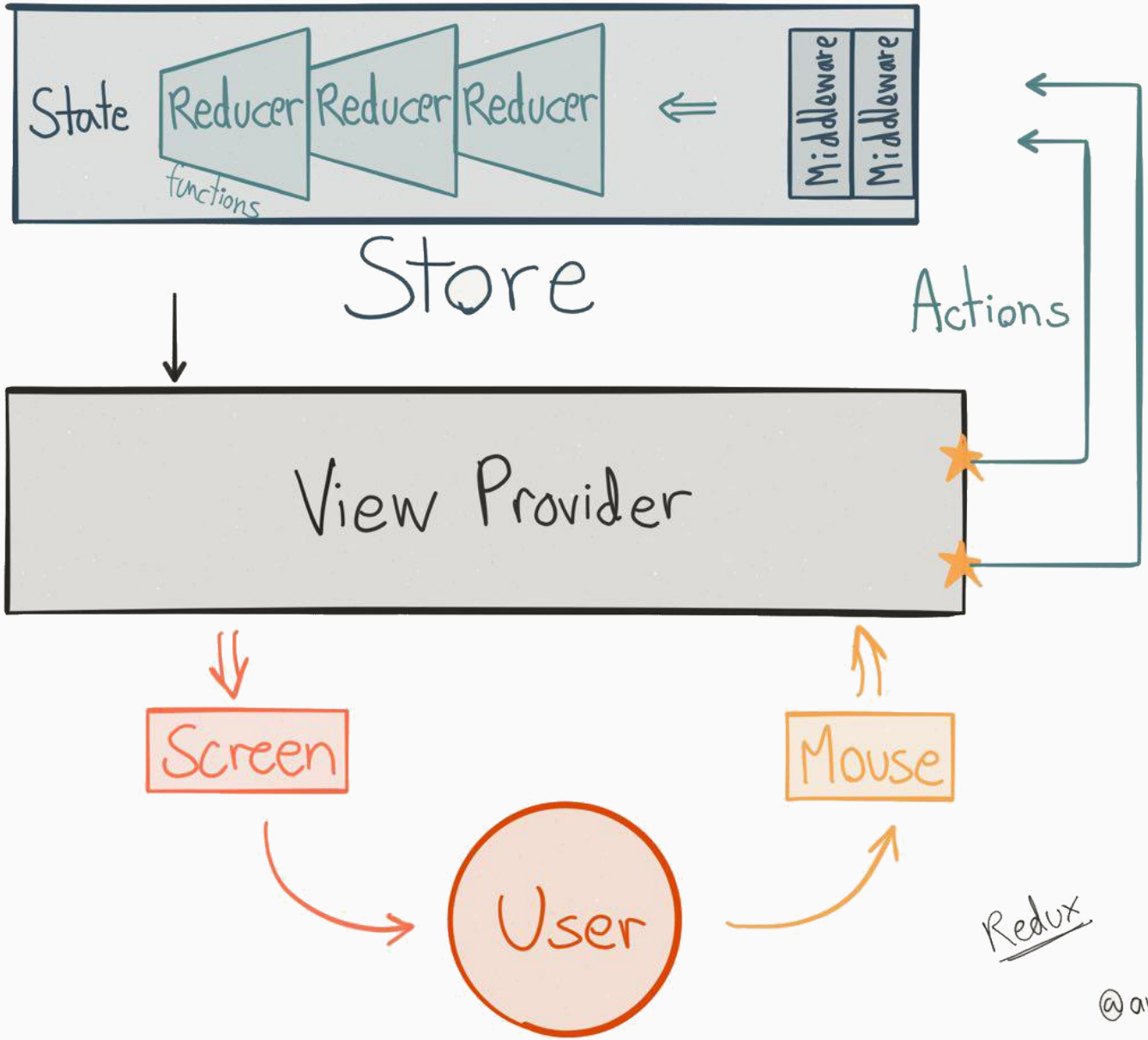
Three Principles

Single source of truth

State is read-only

Changes are made with pure functions

Architecture of a redux app



Actions

Payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using *store.dispatch()*.

```
const action = { type: 'TOGGLE_TODO', index: 6 }  // 'type' is the only thing required
```

Action creators are functions that return actions

```
function addTodo (text) {
  returns { type: 'ADD_TODO', text }
}

store.dispatch(addTodo('Pray to Yisus'))

const boundAddTodo = text => store.dispatch(addTodo(text))
boundAddTodo('Pray to Yisus')
```

Reducers

Specify how the application's state changes in response to actions.

State

```
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
```

Action

```
{
  type: 'SET_VISIBILITY_FILTER',
  filter: 'SHOW_FINISHED'
}
```

Reducers

The reducer is a pure function *(previousState, action) => newState*

```
function todoApp (state = initialState, action) {  // initializes the state
  switch (action.type) {  // looks the type of the action
    case SET_VISIBILITY_FILTER:
      return Object.assign({}, state, {  // returns the new state object
        visibilityFilter: action.filter  // with the changed visibility filter
      })
    case ADD_TODO:
      return Object.assign({}, state, {
        todos: [
          ...state.todos,
          {
            text: action.text,
            completed: false
          }
        ]
      })
    default:
      return state
  }
}
```

Reducers composition

One reducer for each part of the tree

```
function visibilityFilter(
    state = SHOW_ALL, action
  ) {
  switch (action.type) {
    case SET_VISIBILITY_FILTER:
      return action.filter
    default:
      return state
  }
}

function todos (state = [], action) {
  switch (action.type) {
    case ADD_TODO:
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ]
    default:
      return state
  }
}
```

State

```
{ visibilityFilter: 'SHOW_ALL',
  todos: [
    { text: 'Consider using Redux',
      completed: true, },
    { text: 'Keep all state in a single tree',
      completed: false } ] }
```

All the reducers are combined in one

```
function todoApp (state = {}, action) {
  return {
    visibilityFilter: visibilityFilter(
      state.visibilityFilter, action),
    todos: todos(state.todos, action)
  }
}

// Redux let's us do it with combineReducers

import { combineReducers } from 'redux'

const todoApp = combineReducers({
  visibilityFilter,
  todos
})
```

Store API

Implementation

```
const createStore = reducer => {
  let state
  let listener = []

  const getState = () => state

  const dispatch = action => {
    state = reducer(state, action)
    listeners.forEach(
      listener => listener()
    )
  }

  const subscribe = listener => {
    listeners.push(listener)
    return () => {
      listeners = listeners.filter(
        l => l !== listener
      )
    }
  }
}
```

Usage

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)

// Every time the state changes, log it
// Note that subscribe() returns a function
// for unregistering the listener
let unsubscribe = store.subscribe(() =>
  console.log(store.getState())
)

// Dispatch some actions
store.dispatch(addTodo('Learn about actions'))
store.dispatch(addTodo('Learn about reducers'))
store.dispatch(addTodo('Learn about store'))
store.dispatch(
  setVisibilityFilter(
    VisibilityFilters.SHOW_COMPLETED
  )
)

// Stop listening to state updates
unsubscribe()
```

React integration

```
import { setFilter } from 'actions'

export default class ChangeFilter extends Component {  // Usage <ChangeFilter store=store />
  componentDidMount () {
    this.unsubscribe =   // The return value of subscribe is an unsubscribe method
      this.props.store.subscribe(  // Subscribe to store changes
        () => this.forceUpdate())  // Update the component when something changes
  }

  componentWillUnmount () {
    this.unsubscribe()  // Unsubscribe from store change notifications
  }

  render () {
    const store = this.props.store
    const state = store.getState()  // Read the state of the store
    return (
      <div>
        Current filter {state.visibilityFilter}  // Read a value from the store
        <Button onClick={() => store.dispatch(  // Dispatch and action to update the store
          setFilter('SHOW_ALL')  // Action creator
        )} >
          SHOW ALL
        </Button>
      </div>
    )
  }
}
```

React integration

With **react-redux**

```
import { setFilter } from 'actions'
import { connect } from 'react-redux'

const ChangeFilter = props => {
  return (<div>
    Current filter {prop.visibilityFilter}
    <Button onClick={() => props.dispatchSetFilter('SHOW_ALL')} >
      SHOW ALL
    </Button>
  </div> ) } }
}

function mapStateToProp (state, props) {
  return {  // This will be passed as props
    visibilityFilter = state.visibilityFilter
  }
}

function mapStateToDispatch (dispatch, props) {
  return {  // This will be passed as props
    dispatchSetFilter: filter => dispatch(setFilter(filter))
  }
}

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

```
// main.jsx where App includes our ChangeFilter
render(
  <Provider store={store}> <App /> </Provider>,
  document.getElementById('root')
)
```

Links of interest

ES6

http://es6-features.org/

React

https://facebook.github.io/react/

http://calendar.perfplanet.com/2013/diff/

https://github.com/ReactTraining/react-router/tree/master/docs

Redux

http://redux.js.org/

https://egghead.io/courses/getting-started-with-redux

Normalizr

https://github.com/paularmstrong/normalizr

Enzyme

http://airbnb.io/enzyme/