RICE®

Web Development

COMP 431 / COMP 531

Redux

Scott E Pollack, PhD

September 27, 2016

Recap

- HTML and HTML5, Storage, Canvas
- JavaScript and Scope
- Forms, CSS, Events
- jQuery, AJAX, and fetch
- Modern JS
- MVC

~~Modern JS~~
~~MVC~~
~~React~~ & Redux
Testing

Homework Assignment 4
(JavaScript Game)
Due Thursday 10/6

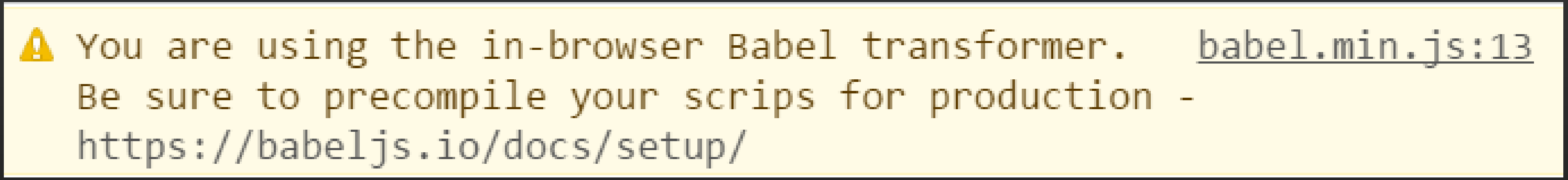
COMP 531
Draft Front-End Review
Due TONIGHT by 2 AM

Did you watch the videos?

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

Build Tools

- Babel in the browser is not production quality



```
⚠ You are using the in-browser Babel transformer.    babel.min.js:13
  Be sure to precompile your scrips for production -
  https://babeljs.io/docs/setup/
```

- Instead, transpile code and serve the resultant build artifact
- We'll use npm tools for this

Getting started

```
git clone https://github.com/skotep/starter.git
cd starter
npm install --verbose
```

What's in the box?

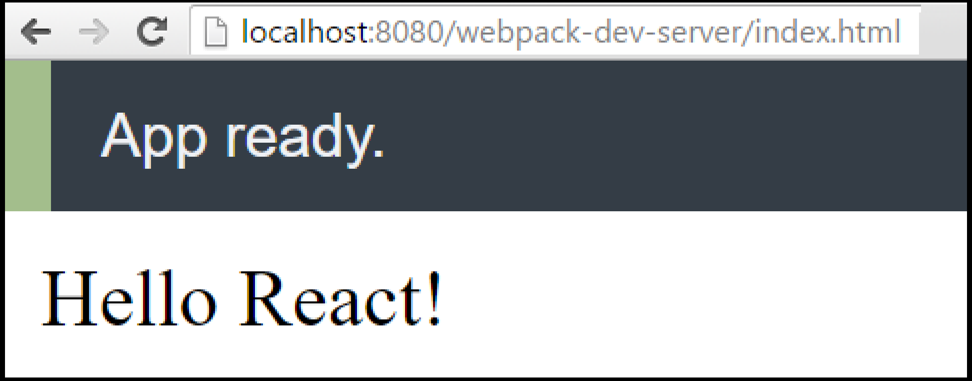
- Babel – transpilation tool
- Bluebird – promise library
- Chai – expectations
- Enzyme – React testing
- Eslint – linting
- Isomorphic-fetch – fetch wrapper
- Istanbul – coverage tool
- Jsdom – DOM mocking
- Karma – test runner
- Mocha – test framework
- Mockery – mocking framework
- Moment – time library
- React – view library
- Redux – state management
- Selenium – headless webdriver
- Sinon – spy library for testing
- Surge – frontend hosting
- Webpack – build tool
- Bootstrap – CSS framework
- font-awesome – good stuff

npm run

```
Lifecycle scripts included in starter:
  prestart
    npm run build
  start
    webpack-dev-server --content-base dist
  test
    npm run mocha

available via `npm run-script`:
  clean
    rimraf dist/bundle.js*
  lint
    eslint src --ext .js --ext .jsx --cache
  watch
    webpack --watch
  build
    webpack
  deploy
    webpack && surge -p dist
  mocha
    mocha --compilers js:babel-core/register --recursive src/**/*.spec.js
  karma
    karma start || exit 0
```

npm start

```
 npm start

> starter@1.0.0 prestart C:\cygwin\home\skotep\rw\demos\starter
> npm run build


> starter@1.0.0 build C:\cygwin\home\skotep\rw\demos\starter
> webpack

Hash: 74c039b07228b4e316fe
Version: webpack 1.13.2
Time: 3058ms
        Asset     Size  Chunks             Chunk Names
    bundle.js  1.57 kB       0  [emitted]  main
bundle.js.map  1.59 kB       0  [emitted]  main
    + 1 hidden modules

> starter@1.0.0 start C:\cygwin\home\skotep\rw\demos\starter
> webpack-dev-server --content-base dist

http://localhost:8080/webpack-dev-server/
webpack result is served from /
content is served from C:\cygwin\home\skotep\rw\demos\starter\dist
Hash: 74c039b07228b4e316fe
Version: webpack 1.13.2
Time: 3154ms
        Asset     Size  Chunks             Chunk Names
    bundle.js  1.57 kB       0  [emitted]  main
bundle.js.map  1.59 kB       0  [emitted]  main
chunk    {0} bundle.js, bundle.js.map (main) 53 bytes [rendered]
    [0] ./src/index.js 53 bytes {0} [built]
webpack: bundle is now VALID.
```

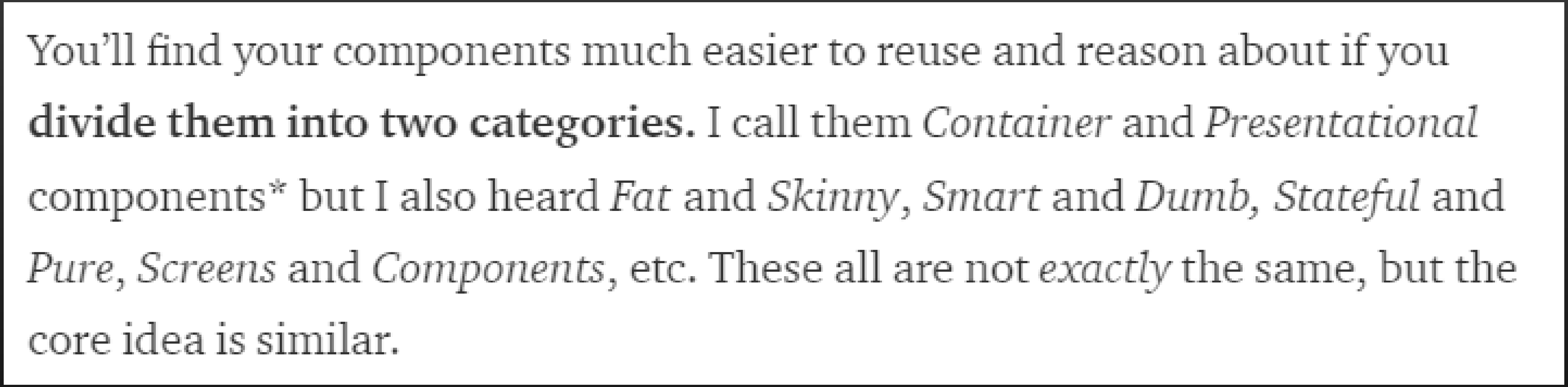


App ready.

Hello React!

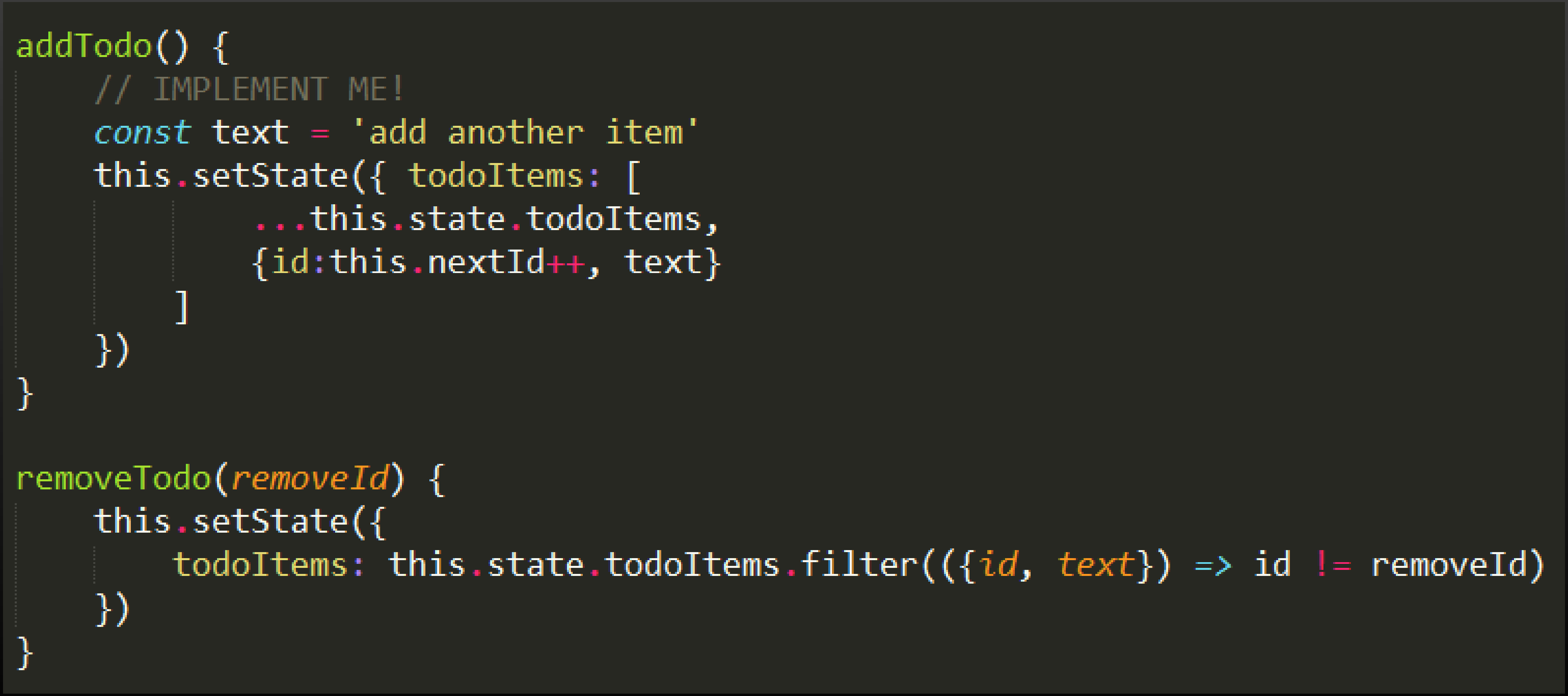
Demo: Webpack Live Reloading

Separation of Concerns


You'll find your components much easier to reuse and reason about if you **divide them into two categories.** I call them *Container* and *Presentational* components* but I also heard *Fat* and *Skinny*, *Smart* and *Dumb*, *Stateful* and *Pure*, *Screens* and *Components*, etc. These all are not *exactly* the same, but the core idea is similar.

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.z20p1v3je

The Fat Component



```
addTodo() {
    // IMPLEMENT ME!
    const text = 'add another item'
    this.setState({ todoItems: [
            ...this.state.todoItems,
            {id:this.nextId++, text}
        ]
    })
}

removeTodo(removeId) {
    this.setState({
        todoItems: this.state.todoItems.filter(({id, text}) => id != removeId)
    })
}
```

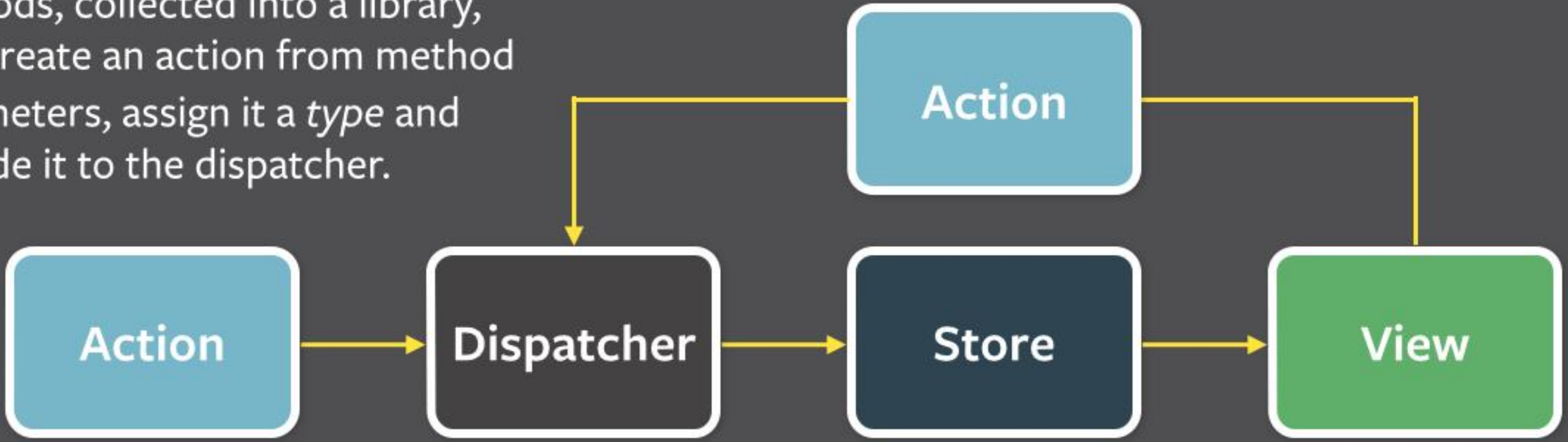
Idea

- Components are presentational
- Data comes in through props
- Components have little if any state
- Components generate actions to update “global” state
- “global” state trickles down as props to Components

FLUX
coffee
cold brew
10oz
FLUX
coffee
cold brew

Flux

Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.



Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

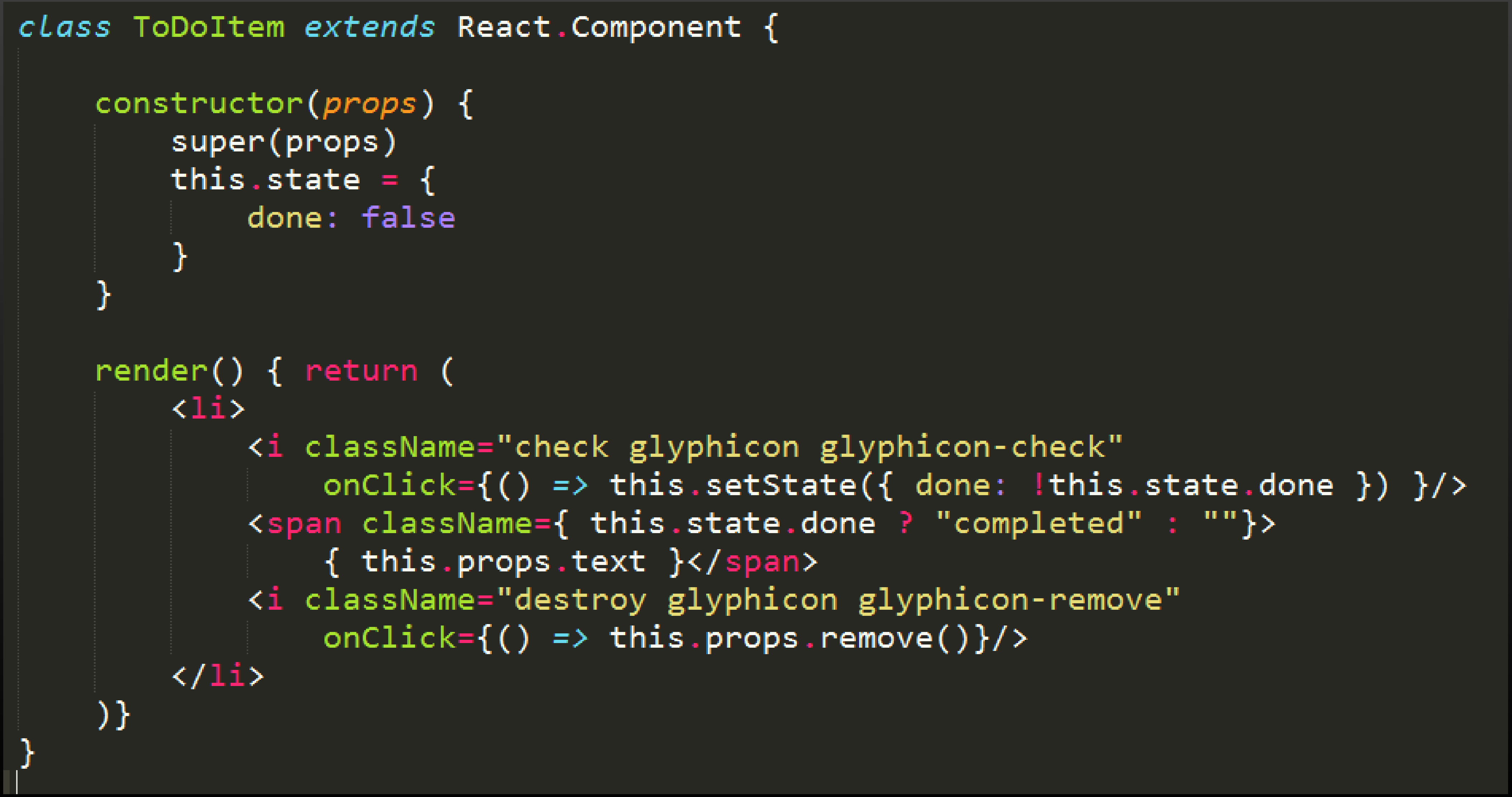
Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

https://facebook.github.io/flux/docs/overview.html

Design

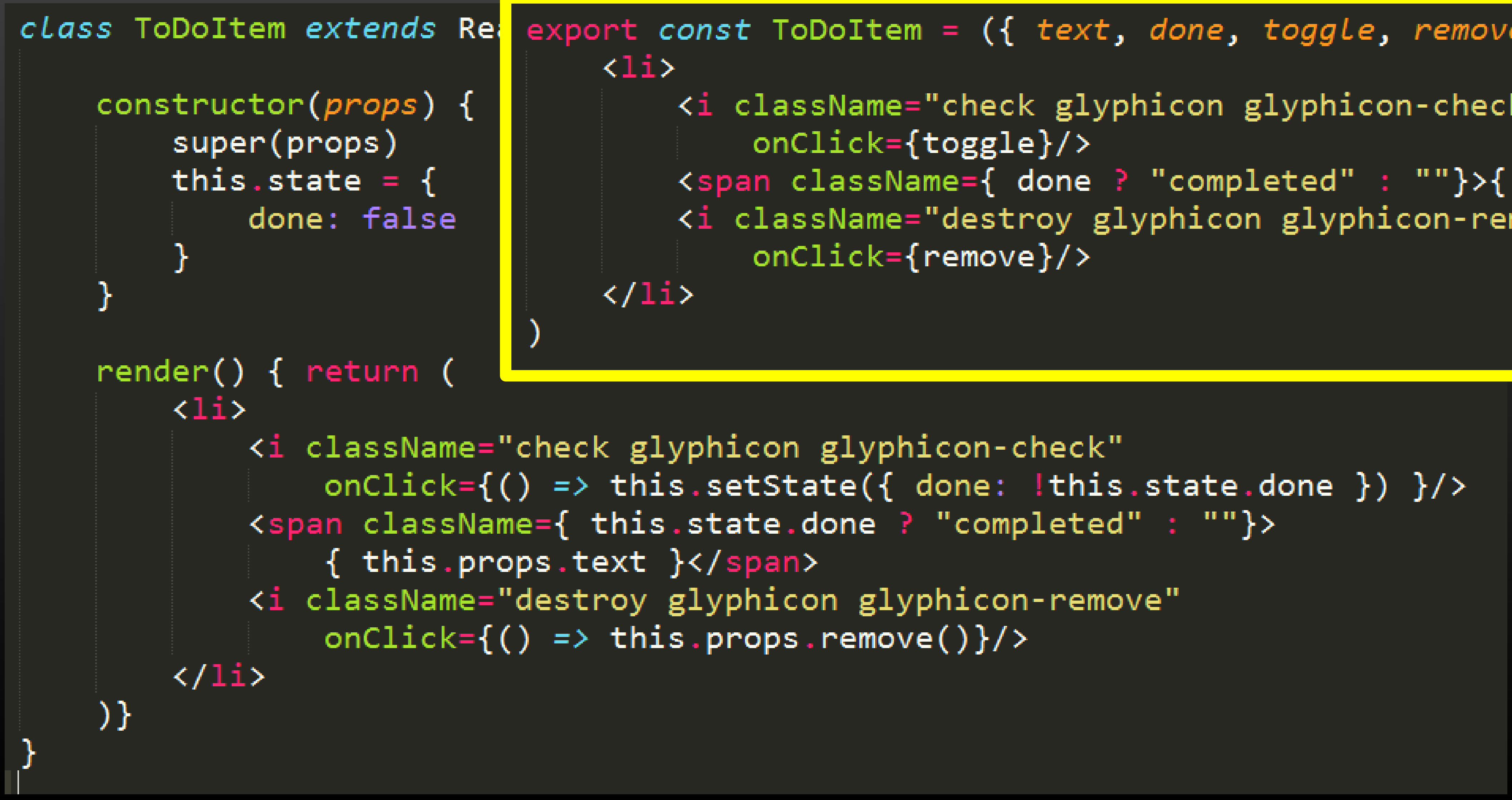
- How do we simplify our ToDo app?

Components => Simple and Primarily Presentation-only

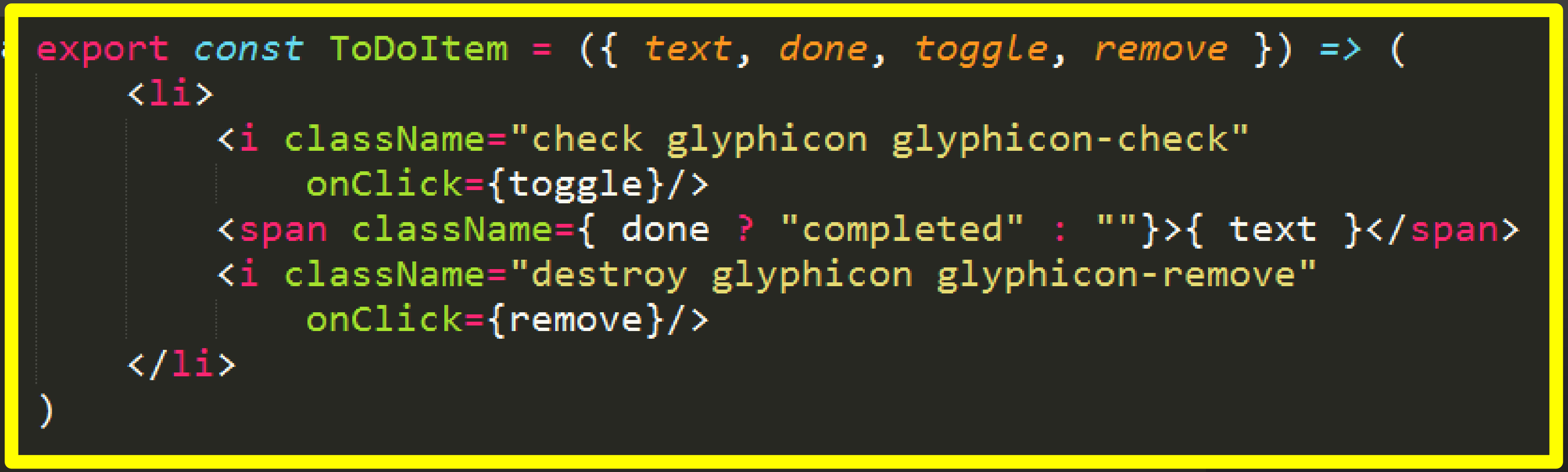


```
class ToDoItem extends React.Component {

    constructor(props) {
        super(props)
        this.state = {
            done: false
        }
    }

    render() { return (
        <li>
            <i className="check glyphicon glyphicon-check"
                onClick={() => this.setState({ done: !this.state.done }) }/>
            <span className={ this.state.done ? "completed" : ""}>
                { this.props.text }</span>
            <i className="destroy glyphicon glyphicon-remove"
                onClick={() => this.props.remove()}/>
        </li>
    )}
}
```

Components =>
Simple and Primarily Presentation-only



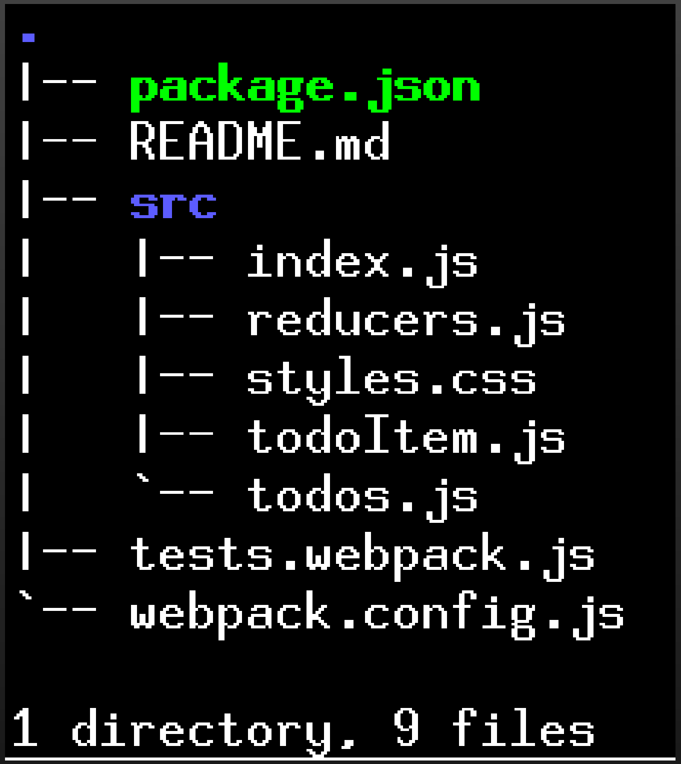
```
class ToDoItem extends Re

    constructor(props) {
        super(props)
        this.state = {
            done: false
        }
    }

    render() { return (
        <li>
            <i className="check glyphicon glyphicon-check"
                onClick={() => this.setState({ done: !this.state.done }) }/>
            <span className={ this.state.done ? "completed" : ""}>
                { this.props.text }</span>
            <i className="destroy glyphicon glyphicon-remove"
                onClick={() => this.props.remove()}/>
        </li>
    )}
}
```



```
export const ToDoItem = ({ text, done, toggle, remove }) => (
    <li>
        <i className="check glyphicon glyphicon-check"
            onClick={toggle}/>
        <span className={ done ? "completed" : ""}>{ text }</span>
        <i className="destroy glyphicon glyphicon-remove"
            onClick={remove}/>
    </li>
)
```

Refactoring

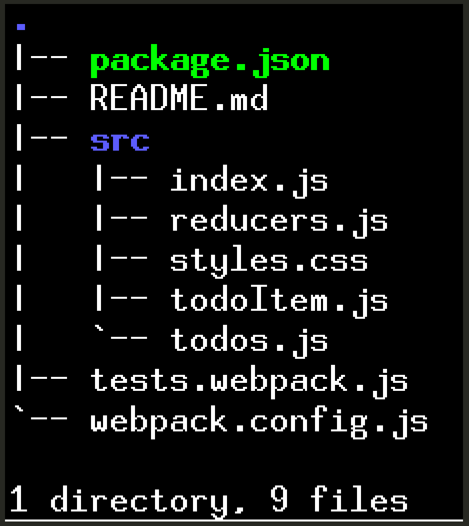
- index.js – entry point for application
- reducers.js – accepts actions, reduces state
- todoItem.js – a single ToDo item
- todos.js – application container



```
.
|-- package.json
|-- README.md
|-- src
|   |-- index.js
|   |-- reducers.js
|   |-- styles.css
|   |-- todoItem.js
|   `-- todos.js
|-- tests.webpack.js
`-- webpack.config.js

1 directory, 9 files
```

index.js

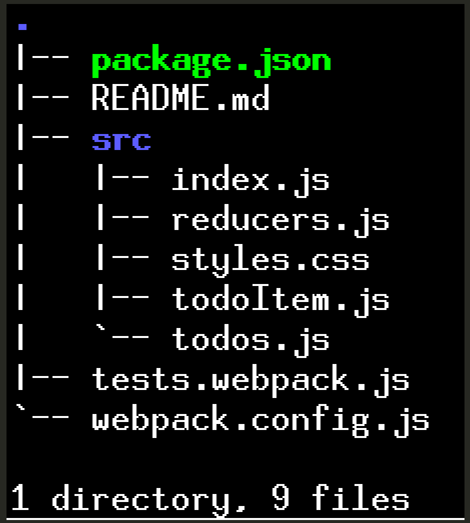
```javascript
require('expose?$!expose?jQuery!jquery')
require("bootstrap-webpack")
require('./styles.css')

import React from 'react'
import { render } from 'react-dom'

import { Provider } from 'react-redux'
import createLogger from 'redux-logger'
import { createStore, applyMiddleware } from 'redux'

import Reducer from './reducers'
import ToDos from './todos'

const logger = createLogger()
const store = createStore(Reducer, applyMiddleware(logger))

render(
    <Provider store={store}>
        <ToDos />
    </Provider>,
    document.getElementById('app')
)
```



```
.
|-- package.json
|-- README.md
|-- src
|   |-- index.js
|   |-- reducers.js
|   |-- styles.css
|   |-- todoItem.js
|   `-- todos.js
|-- tests.webpack.js
`-- webpack.config.js

1 directory, 9 files
```

```
import React, { Component, PropTypes } from 'react'
import { connect } from 'react-redux'

export const ToDoItem = ({ text, done, toggle, remove }) => (
    <li>
        <i className="check glyphicon glyphicon-check" onClick={toggle}/>
        <span className={ done ? "completed" : ""}>{ text }</span>
        <i className="destroy glyphicon glyphicon-remove" onClick={remove}/>
    </li>
)

ToDoItem.propTypes = {
    id: PropTypes.number.isRequired,
    text: PropTypes.string.isRequired,
    done: PropTypes.bool.isRequired,
    toggle: PropTypes.func.isRequired,
    remove: PropTypes.func.isRequired
}

export default connect(null, (dispatch, ownProps) => {
        return {
            remove: () => dispatch({ type: 'REMOVE_TODO', id: ownProps.id }),
            toggle: () => dispatch({ type: 'TOGGLE_TODO', id: ownProps.id })
        }
    })(ToDoItem)
```

todoItem.js



```
.
|-- package.json
|-- README.md
|-- src
|   |-- index.js
|   |-- reducers.js
|   |-- styles.css
|   |-- todoItem.js
|   `-- todos.js
|-- tests.webpack.js
`-- webpack.config.js

1 directory, 9 files
```

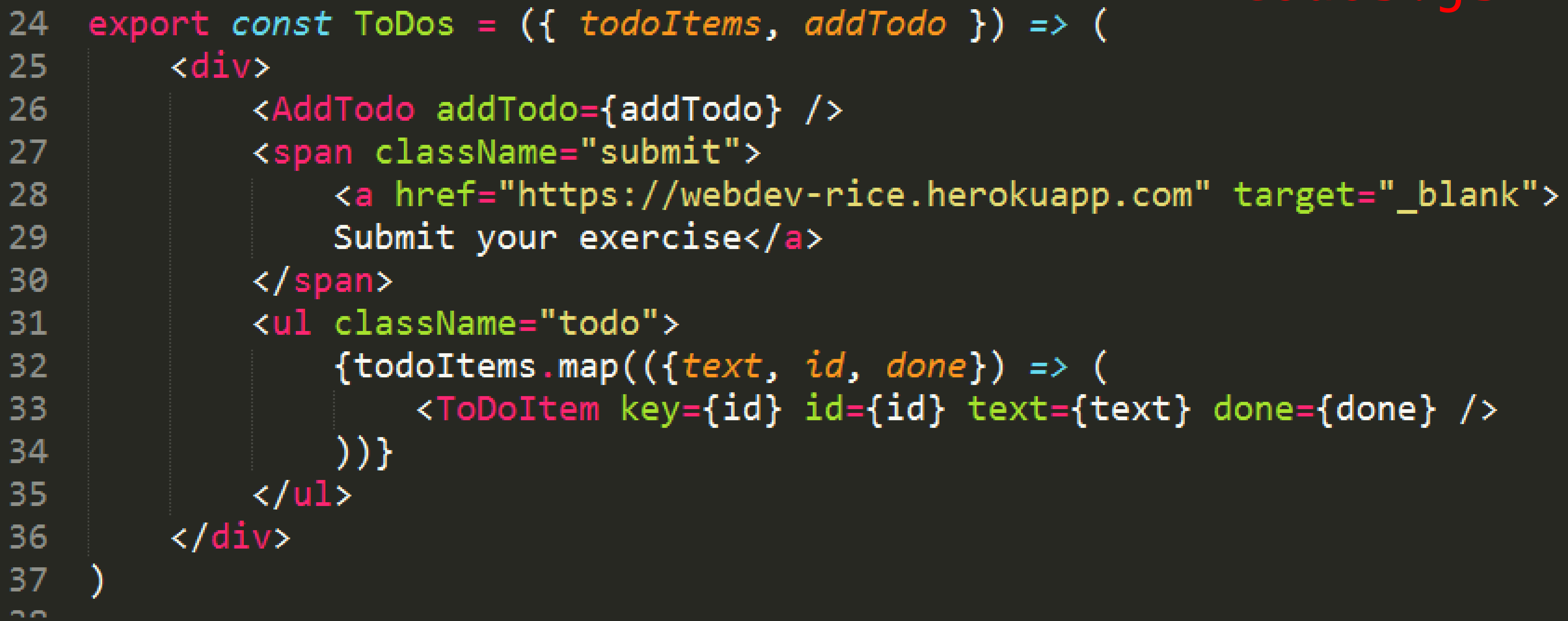
todos.js

```
import React, { Component, PropTypes } from 'react'
import { connect } from 'react-redux'

import ToDoItem from './todoItem'

export const AddTodo = ({ addTodo }) => {
    let newTODO;

    const _addTodo = () => {
        if (newTODO && newTODO.value) {
            addTodo(newTODO.value)
            newTODO.value = ''
        }
    }

    return (<span>
        <input type="text" placeholder="To Do" ref={ (node) => newTODO = node } />
        <button onClick={_addTodo}>Add Item</button>
    </span>)
}
```

```
.
|-- package.json
|-- README.md
|-- src
|   |-- index.js
|   |-- reducers.js
|   |-- styles.css
|   |-- todoItem.js
|   `-- todos.js
|-- tests.webpack.js
`-- webpack.config.js

1 directory, 9 files
```

I want access to the DOM node so I can clear the text when the button is clicked.

todos.js



```jsx
export const ToDos = ({ todoItems, addTodo }) => (
    <div>
        <AddTodo addTodo={addTodo} />
        <span className="submit">
            <a href="https://webdev-rice.herokuapp.com" target="_blank">
            Submit your exercise</a>
        </span>
        <ul className="todo">
            {todoItems.map(({text, id, done}) => (
                <ToDoItem key={id} id={id} text={text} done={done} />
            ))}
        </ul>
    </div>
)
```

todos.js

```
ToDos.propTypes = {
    todoItems: PropTypes.arrayOf(PropTypes.shape({
        ...ToDoItem.propTypes
    }).isRequired).isRequired,
    addTodo: PropTypes.func.isRequired
}

export default connect(
    (state) => {
        return {
            todoItems: state.todoItems
        }
    },
    (dispatch) => {
        return {
            addTodo: (text) => dispatch({ type: 'ADD_TODO', text })
        }
    }
)(ToDos)
```

reducers.js

```javascript
const Reducer = (state =  {
    nextId: 2,
    todoItems: [
        {id: 0, text: "This is an item", done: false},
        {id: 1, text: "Another item", done: false}
    ]
}, action) => {
    switch(action.type) {
        case 'ADD_TODO':
            // IMPLEMENT ME
        case 'REMOVE_TODO':
            // IMPLEMENT ME
        case 'TOGGLE_TODO':
            // IMPLEMENT ME
        default:
            return state
    }
}

export default Reducer
```

```
.
|-- package.json
|-- README.md
|-- src
|   |-- index.js
|   |-- reducers.js
|   |-- styles.css
|   |-- todoItem.js
|   `-- todos.js
|-- tests.webpack.js
`-- webpack.config.js

1 directory, 9 files
```

Read these

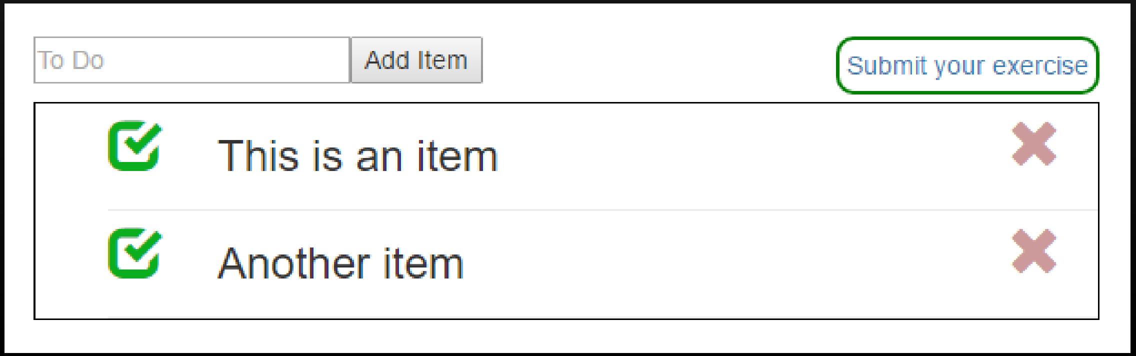
- https://medium.com/@learnreact/container-components-c0e67432e005#.h1vxrbvmq
- https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.z20p1v3je
- If you didn't this past weekend, watch this series ~3 hrs or so https://egghead.io/courses/getting-started-with-redux

In-Class Exercise: Hello Redux

https://webdev-rice.herokuapp.com

/inclass-11/...

- Download and unzip https://www.clear.rice.edu/comp431/sample/helloRedux.zip
- I have already reimplemented the TODO app with Redux, but the reducer is missing.
- Your task is to implement the reducer in ***reducers.js***
- When completed the page should load like the image below
  - The check box should be functional (strike through)
  - The X should be functional
  - "Add Item" adds new items.



This is an item

Another item