Let’s Redux

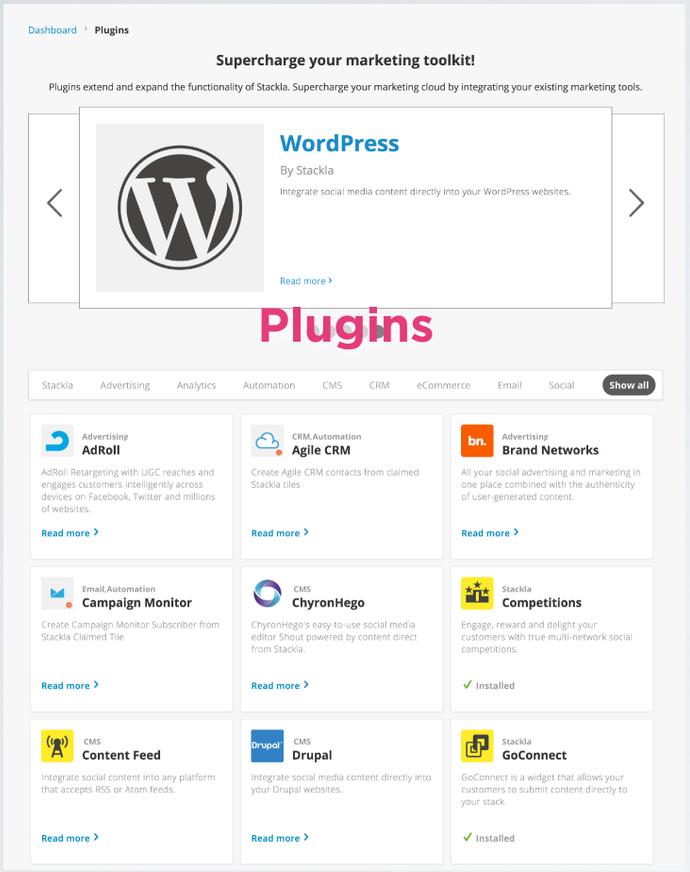
for people who use React but haven’t tried Redux

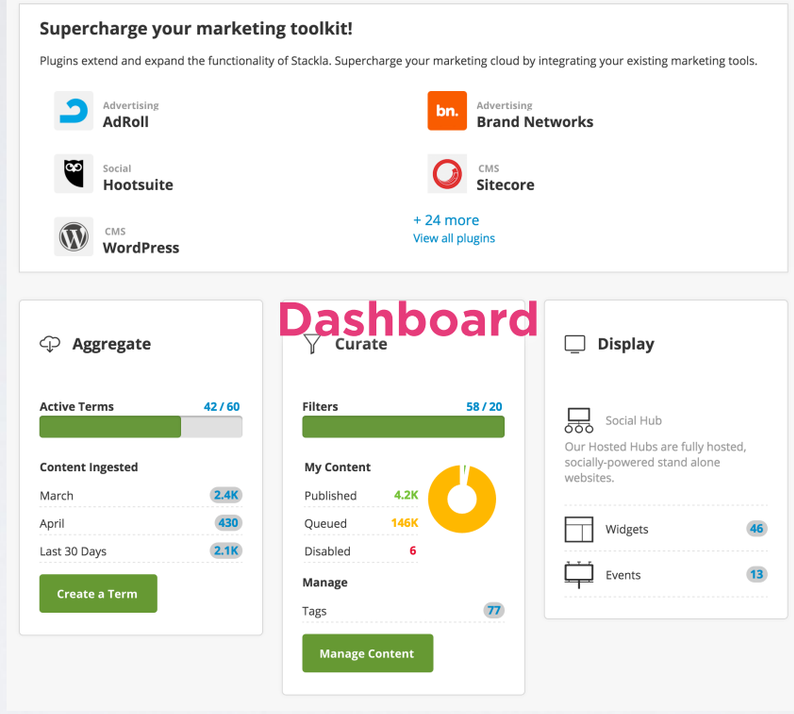
@josephj6802

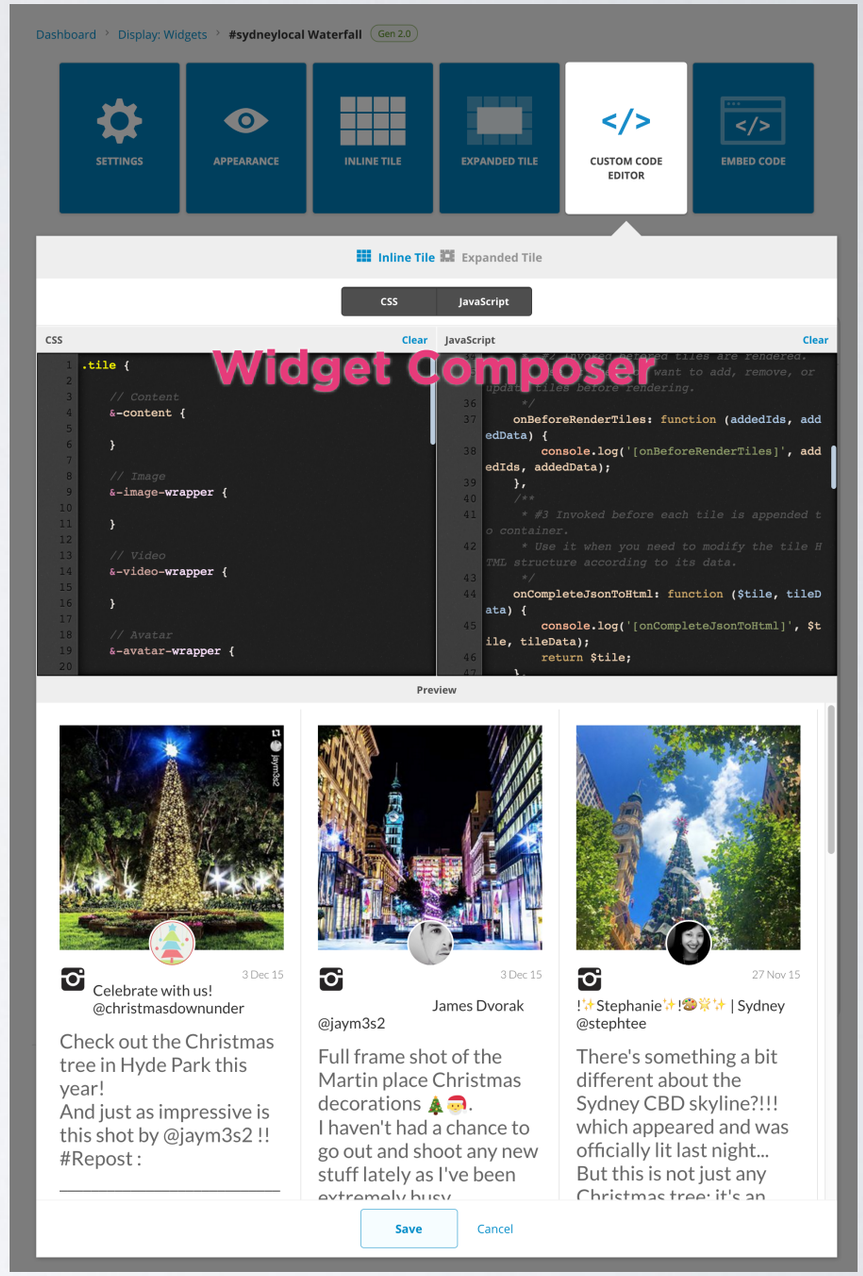
Stackla

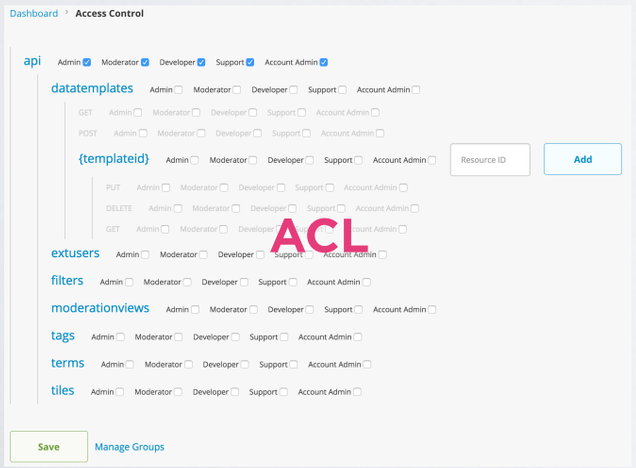
React.js is Awesome

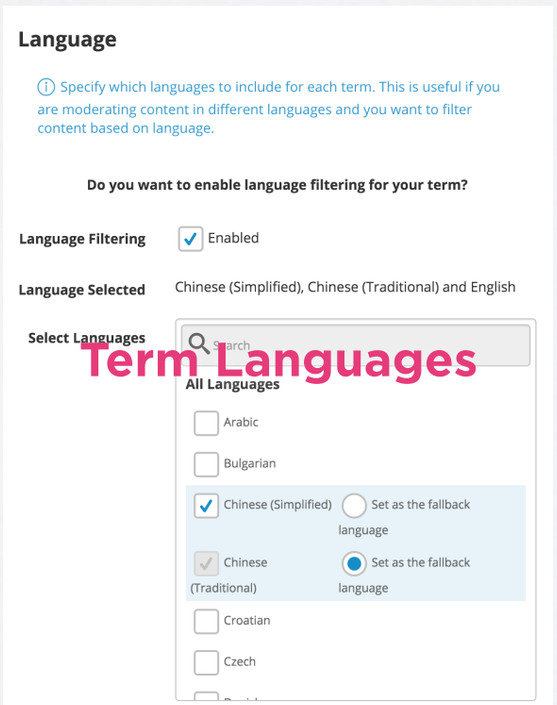
Building a complicate UI was never so easy











Stackla

React.js is Awesome

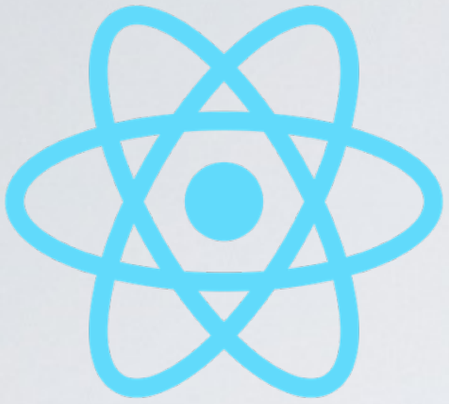
Building a complicate UI was never so easy

Advantages

- Easy to learn
- Manage State instead of DOM
- One-way Data Flow Model
- UI = fn(State)

Absolutely a correct technical decision

Stackla



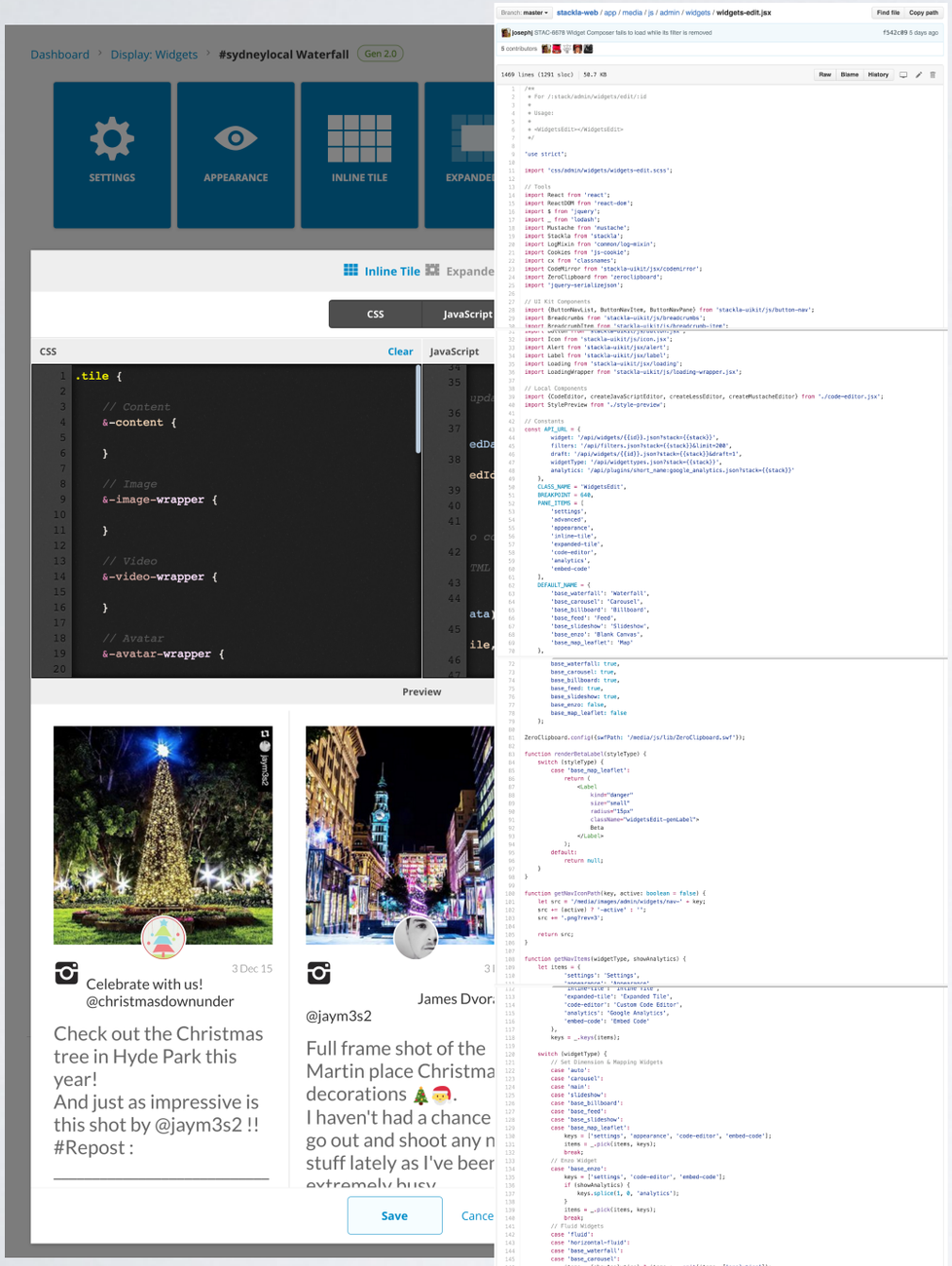
“It worked well with simple components, but **as our interfaces become more complicate** we soon found…”

PROBLEMS!!

~~React doesn’t~~ **We** don’t have a **front-end application architecture**.

Code Smells

<WidgetsEdit/> Top-level Component



- **Delegation is difficult**
  - Top-down props
- **Mixed concerns**
  - View logic
  - Data fetching
  - Data decoration
- **Poor state management**
  - Component-specific
  - Mutable
  - Not maintainable

1469 lines in total!!

Stackla

React.js = “Good” Parent

who **takes care of everything** for childs

```
class WidgetsEdit extends Component {
    constructor(props) {
        super(props);
    }
    componentWillMount() {
        $.ajax(...).then((...) => {
            // ... decorate data ...
            this.setState(...)
        });
    }
    render() {
        let manyStates = that.state;
        // ... view logic ...
        return (
            <ButtonNavList {...manyStates}>
                <ButtonNavItem {...manyStates}/>
            </ButtonNavList>
            <StylePreview {...manyStates}/>
            <CodeEditor {...manyStates}/>
        )
    }
}
```

Preparing data, passing props, handling events for childs

Stackla

Handling Children Events

```
<CodeEditor
    data={that.state.data}
    stack={that.state.stack}
    count={that.state.count}
    scaffolds={that.state.scaffolds}
    draftId={that.state.draftId}
    editorGroups={that.state.editorGroups}
    onTemplateCreate={that.handleTemplateCreate}
    onSave={that.handleSave.bind(that, true)}
    onCancel={that.handleNavDismiss}
    onClone={that.handleEditorClone}
    onDelete={that.handleEditorDelete}
    onChange={that.handleEditorChange}
    onInfoChange={that.handleEditorInfoChange}>
    {loadingNode}
</CodeEditor>
```

Child passes all events to its parent

because only the top-level component should access state

22 Event Handlers

```
1105    // Event Handlers
1106    //================
1107 ▶  handleActionsDismiss () { ...
1110    },
1111 ▶  handleBeforeUnload () { ...
1122    },
1123 ▶  handleEnableChange (e) { ...
1129    },
1130 ▶  handleEditorSave (e) { ...
1134    },
1135 ▶  handleDocumentKeydown (e) { ...
1148    },
1149 ▶  handleDiscard (e) { ...
1163    },
1164 ▶  handleNameChange (e) { ...
1170    },
1171 ▶  handleSave (real: boolean = true, e) { ...
1229    },
1230 ▶  handleFilterChange (e) { ...
1239    },
1240 ▶  handleNavClick (name: string, e) { ...
1253    },
1254 ▶  handleTemplateCreate (newKey) { ...
1268    },
1269 ▶  handleNavDismiss () { ...
1278    },
1279 ▶  handlePreviewChange (mode) { ...
1285    },
1286 ▶  handleEditorChange (changes) { ...
1296    },
1297 ▶  handleEditorInfoChange (storeKey, groupKey, data) { ...
1318    },
1319 ▶  handleEditorClone (storeKey, groupKey) { ...
1347    },
1348 ▶  handleEditorDelete (storeKey, groupKey) { ...
1370    },
1371 ▶  handleFieldChange (e) { ...
1380    },
1381 ▶  handleAjaxError (xhr, status, msg) { ...
1420    },
1421 ▶  handleSaveSuccess (real: boolean = true, data: {} = {}, status, e) { ...
1458    },
1459 ▶  handleSavedDismiss () { ...
1464    },
1465 ▶  handleToggleAdvanced () { ...
```

Stackla

Delegation or Not?

```
that.log('renderNav() is executed');

nodes = _.map(items, (item, key) => {
    let isActive = (that.state.activeNavKey === key), ...

    if (state.loading) { ...
    }

    if (key === 'code-editor') {
        isError = (_.size(that.state.error) > 0);
    } else { ...
    }

    if (key === 'code-editor' && !that.canAccessEditor) {
        isDisabled = true;
    }

    if (key === 'analytics' && (!that.state.showAnalytics || ['base_waterfall', '
        yle.style) === -1)) {
        return;
    }

    return (
        <ButtonNavItem
            key={key}
            name={key}
            disabled={isDisabled}
            error={isError}
            active={isActive}
            onSelect={that.handleNavClick}>
            <img src={getNaviconPath(key, isActive)} width="78" height="59"/>
            <span className="st-small">{item}</span>
        </ButtonNavItem>
    );
});

return (
    <ButtonNavList
        active={(that.state.activeNavKey !== null)}
        onDismiss={that.handleNavDismiss}>
        {nodes}
    </ButtonNavList>
```

Should <ButtonNavItem/> appear in top-level as illustrated?

If your think NO, plz put inside…

- Pass all <ButtonNavItem/> required props to <ButtonNavList/>
- Handle its events in <ButtonNavList/>
- Handle its events in <WidgetsEdit/>

Not Convenient

If your think YES, plz keep there…

Handle child logic in top-level component.

Dirty!

Both ways are not good…

Stackla

Fetching Data for Childs

```
236     componentWillMount () {
237         var that = this,
238             url = {},
239             attrs = {},
240             globalVariables = Stackla.globalVariables;
241
242         that.log('componentWillMount() is executed');
243
244         that.canAccessEditor = (globalVariables.developerFeatureEnabled && globalVariable
245
246         attrs = {
247             stack: that.props.params.stack,
248             id: that.props.params.id
249         };
250
251         url.filters = Mustache.render(API_URL.filters, attrs);
252         url.widget  = Mustache.render(API_URL.widget, attrs);
253         url.draft = Mustache.render(API_URL.draft, attrs);
254         url.widgetType = Mustache.render(API_URL.widgetType, attrs);
255         url.analytics = Mustache.render(API_URL.analytics, attrs);
256
257         that.loadDeferreds = [
258             $.ajax(url.filters),
259             $.ajax(url.widget),
260             $.ajax(url.widgetType),
261             $.ajax(url.analytics),
262             $.ajax(url.draft, {type: 'PUT', data: '{}'}) // FIXME - data is really odd
263         ];
264
265         that.loadInitData();
266     },
```

No Separation of Concerns

Stackla

Why Redux?

Not have children?

https://github.com/reactjs/redux/issues/151#issuecomment-173749456

Why Redux?

Solve all the mentioned issues

- **Delegation is ~~difficult~~ easier**
  - Top-down props ~~including events~~ is not required
- **~~Mixed~~ Separation of concerns**
  - View logic
  - Data fetching goes to **"Action"**
  - Data decoration goes to "**Reducer**"
- **~~Poor~~ Better state management**
  - ~~Component~~ **App**-specific
  - ~~Mutable~~ **Immutable**
  - ~~Not~~ **Maintainable**

Basic Concept

1. Action

2. Reducer

3. State

https://github.com/reactjs/redux/issues/151#issuecomment-134713785

Stackla

1. Action

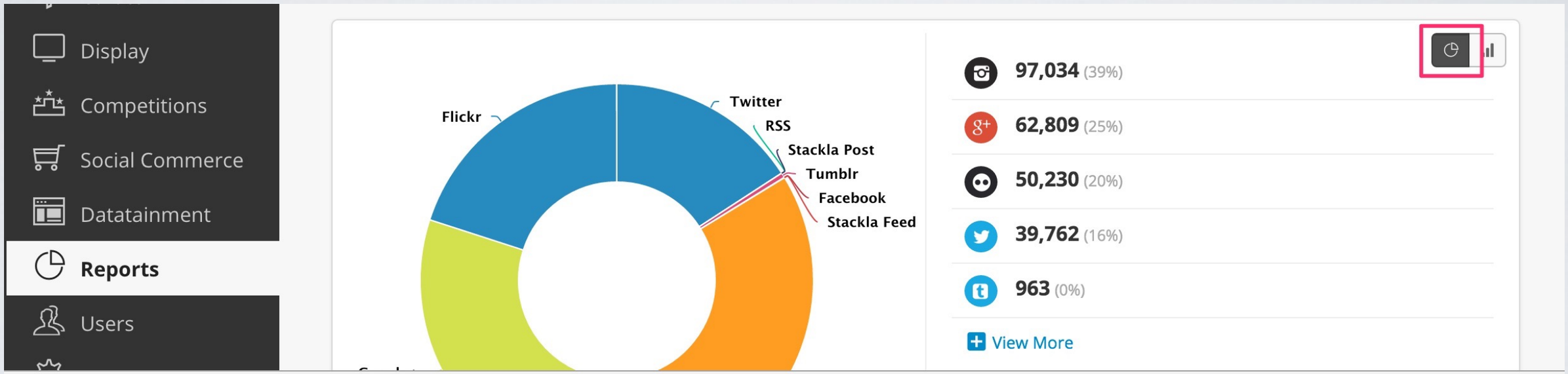
Pure **Object**

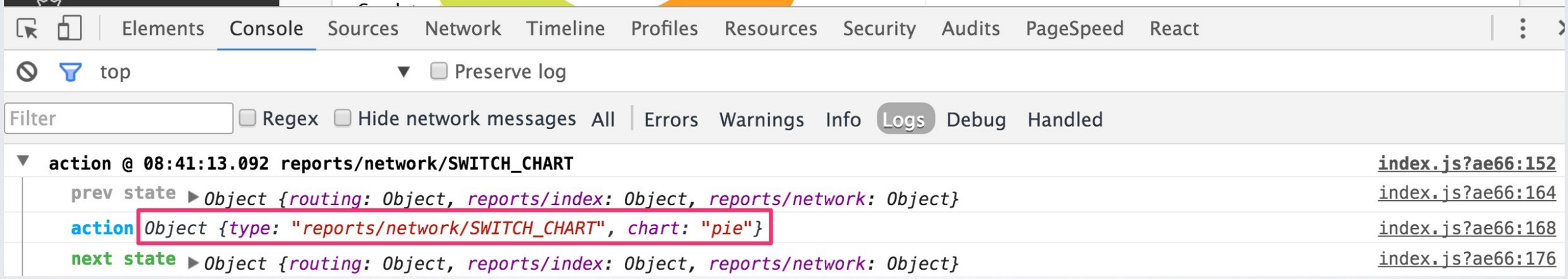
Minimum data which describes the change

```
{
    type: 'SORT_PLUGINS',
    by: 'id',
    direction: 'desc'
}
```

Only the “**type**” property is **required**

1. Action





Stackla

1-1. Action Creator

Pure **function** which **creates an action**

Make it **easier** to create an action

```
function sortPlugins(by, direction = 'desc') {
    return {
        type: 'SORT_PLUGINS',
        by,
        direction
    };
}
```

Reusable, **Portable**, and **Easy to Test**

(Return promise for asynchronous action)

2. Reducer 

Pure **function** which returns the **next state**

reducer(previousState, action) => state

```
function reducer(state = {}, action) {
    switch (action.type) {
        case 'SORT_PLUGINS':
            return {
                ...state,
                plugins: {
                    orderBy: action.by,
                    orderByDirection: action.direction,
                    data: _.sortByOrder(state.data, action.by,
                          action.direction)
                }
            };
        default:
            return state;
    };
}
```

Initial State

Always return current state

2. Reducer

```
▼ action @ 08:41:13.092 reports/network/SWITCH_CHART
    prev state ▼ Object {routing: Object, reports/index: Object, reports/network: Object}
                 ▶ reports/index: Object
                 ▼ reports/network: Object
                     chart: "column"
                   ▶ data: Array[11]
                     didInvalidate: false
                     filters: ""
                     isFetching: false
                     lastUpdated: 1460673632245
                     listAll: false
                     total: 251028
                   ▶ __proto__: Object
                 ▶ routing: Object
                 ▶ __proto__: Object
    action  Object {type: "reports/network/SWITCH_CHART", chart: "pie"}
    next state ▼ Object {routing: Object, reports/index: Object, reports/network: Object}
                 ▶ reports/index: Object
                 ▼ reports/network: Object
                     chart: "pie"
                   ▶ data: Array[11]
                     didInvalidate: false
                     filters: ""
                     isFetching: false
                     lastUpdated: 1460673632245
                     listAll: false
                     total: 251028
                   ▶ __proto__: Object
                 ▶ routing: Object
                 ▶ __proto__: Object
```

reducer(prevState, action)

=> nextState

Stackla

3. Store 📜

A plain **object** which holds **application state**

Store is **the only one state** for the whole app

```
{
    plugins: {
        data: [
            {id: 1, name: 'AdRoll'},
            {id: 2, name: 'Agile CRM'},
            {id: 3, name: 'Brand Networks'}
        ],
        orderBy: 'id',
        orderByDirection: 'desc'
    }
}
```

Dispatching actions is the **only way** to **update** store

3. Store 📜

Stackla

3-1. Store Creation

with Reducer (the spec of your store)

store = createStore(reducer);

```
import {createStore} from 'redux';

let store = createStore(reducer);
```

The store provides several useful **API methods**

3-2. Store APIs

store#dispatch

```
store.dispatch(sortPlugins('name', 'desc'));
```

store#subscribe store#getState

```
let currentValue;
store.subscribe(() => {
    let previousValue = currentValue;
    currentValue = store.getState();
    if (previousValue === currentValue) {
        return;
    }
    // DO SOMETHING...
});
```

Redux Data Flow

App

store.**subscribe**(eventHandlerFn);

Action | Action | Action

store.**dispatch**(actionCreatorFn);

Store

store = **createStore**(reducerFn);

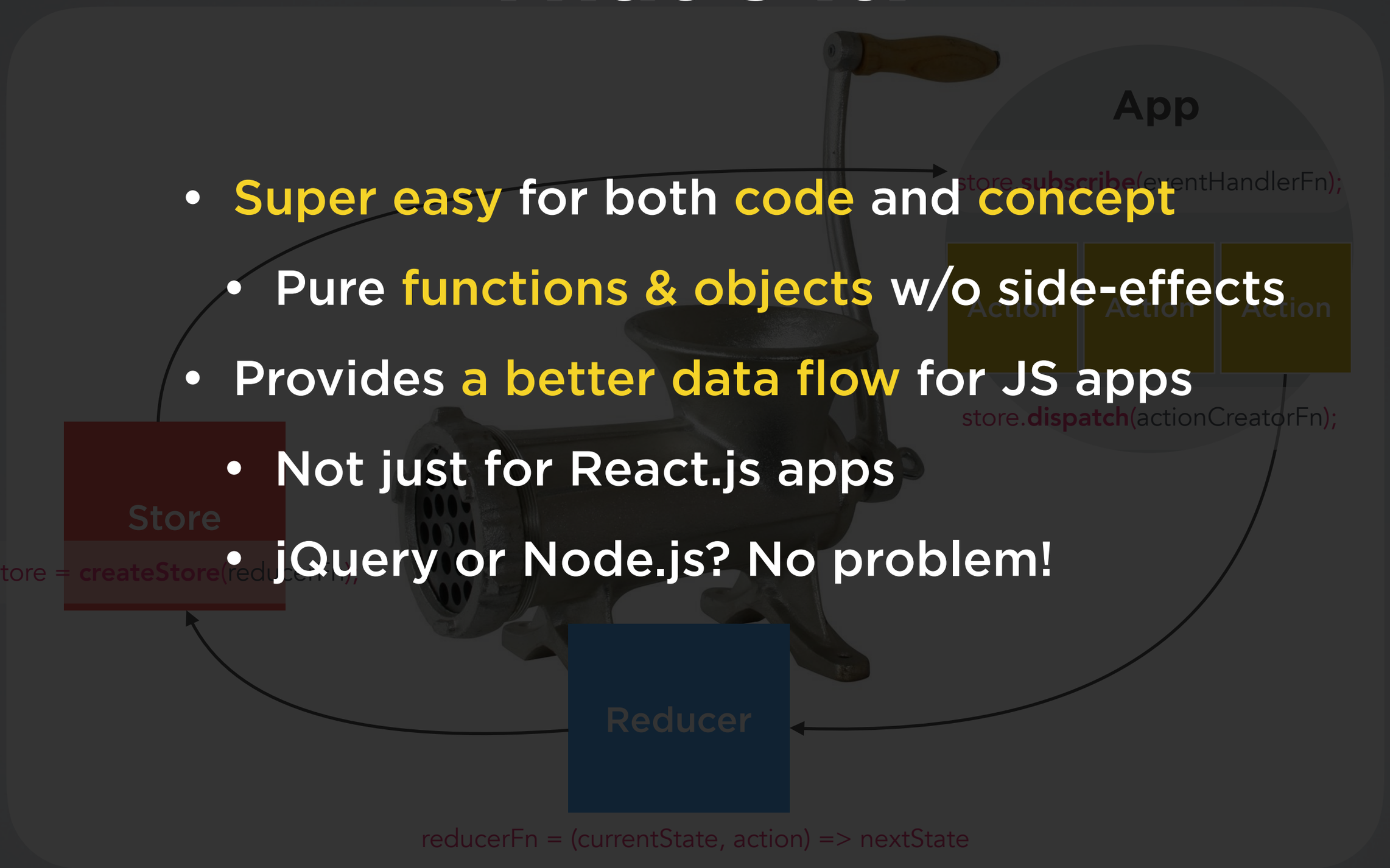
Reducer

reducerFn = (currentState, action) => nextState

Redux Data Flow

That's it!

- Super easy for both code and concept
- Pure functions & objects w/o side-effects
- Provides a better data flow for JS apps
- Not just for React.js apps
- jQuery or Node.js? No problem!



Redux and React?

First, let's check the list again

- **Delegation is easier**
  - Top-down props is not required 🤔
- **Separation of concerns**
  - View logic 👍
  - Data fetching goes to **"Action"** 👍
  - Data decoration goes to "**Reducer**" 👍
- **Better state management**
  - **App**-specific 👍
  - **Immutable** 👍
  - **Maintainable** 👍

React.js = “Good” Parent

who **takes care of everything** of its child components

```
class WidgetsEdit extends Component {
    constructor(props) {
        super(props);
    }
    componentWillMount() {
        $.ajax(...).then((...) => {
            // ... decorate data ...
            this.setState(...)
        });
    }
    render() {
        let manyStates = that.state;
        // ... decorate state ...
        return (
            <ButtonNavList {...manyStates}>
                <ButtonNavItem {...manyStates}/>
            </ButtonNavList>
            <StylePreview {...manyStates}/>
            <CodeEditor {...manyStates}/>
        )
    }
}
```

Preparing data & **handling events** for child components

Stackla

React + Redux = Bad Parent

who only cares about himself

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import {fetchData} from ‘./actions';

class WidgetsEdit extends Component {
    constructor(props) {
        super(props);
    }
    componentWillMount() {
        this.props.dispatch(fetchData());
    }
    render() {
        return (
            <ButtonNavList>
                <ButtonNavItem/>
            <ButtonNavList>
            <StylePreview/>
            <CodeEditor/>
        )
    }
}
WidgetsEdit = connect(mapStateToProps)(WidgetsEdit);
```

Delegation = Preparing data and dispatching **for itself**

Stackla

React.js w/ Redux

Child component **connects to store** by itself

```
import React from 'react';
import {connect} from 'react-redux';
import {modeSwitch} from './actions';

let StylePreview({url, dispatch} = props) => {
    return (
        <a onClick={() => dispatch(modeSwitch('mobile'))}>/>
            Mobile View
        </a>
        <iframe src={url}/>
    );
}

StylePreview = connect((state) => { // map state to props
    return {
        url: state.widget.url,
    }
})(StylePreview);
```

… And **dispatches actions** by itself

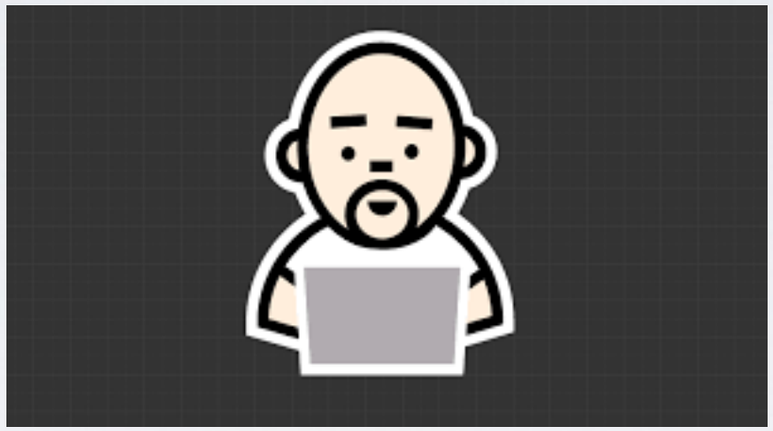
Stackla

Summary

- Good start to learn **functional programming**
- **Add complexity** but also **add maintenance** = worthy
- Single Store
  - **Time Travel** (e.g. **Undo**) is possible
- No need to access React state API anymore
- Not only for React.js

Further Reading

egghead.io



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

Very good material for learning Redux!

Discussion - Store Structure

One Store - Could it possibly fit our application?

```
{
    "app": {
        data: {
            filters: [],
            terms: [],
        },
        isFetching: false,
    },
    "reports/network": {
        data: {
            networks: [],
        },
        isFetching: false,
        didInvalidate: false,
        lastUpdated: 15372124,
        orderBy: 'created_at',
        orderByDirection: 'asc',
    },
    "widgets/index": {
        data: {
            widgets: []
        }
        isFetching: false,
        didInvalidate: false,
        lastUpdated: 15372124,
        orderBy: 'created_at',
        orderByDirection: 'asc',
    }
}
```

THANK YOU!

Let's Redux!!

Stackla