React Native Coding Standard

Nhan Cao

R&D,Dept. of BeeSight Soft, VietNam

nhan.cao@beesightsoft.com

***Abstract*—This document discusses about React Native coding standard using for BeeSight Soft. This document is internal purpose.**

***Keywords*—coding standard, convention**

I. Introduction

BEESIGHT SOFT using beesight cli tool [1] to generate React native (rct) [2] template. Project generated by beesight will be support some main of initial features are: Ignite [3] structure compatible, integrated with Reactotron [4] debugger tool, integrated with React native debugger tool [5], Nativebase [6] design, support Fonts [7], support I18n [8] for multiple language, support multi Environment configuration [9], project unique version [10], splash screen [11], app icon generate [12] with style guide of Airbnb [13].

II. Delivery Package

A. Platform

iOS: from iOS 8.0
Android: from Android 4.1, min api level is 16, min 1GB Ram

B. System Build Requirements

OS: macOS High Sierra Version 10.13.5
xcodebuild: Xcode 9.4.1 Build version 9F2000
git: git version 2.15.2 (Apple Git-101.1)
brew: Homebrew 1.6.9
pod: 1.5.0
adb: Android Debug Bridge version 1.0.40
node: v9.10.1
watchman: 4.9.0
react-native-cli: 2.0.1
react-native: 0.54.2
app-icon: 0.6.2
beesight: v1.3.9

C. IDE

Visual studio code
Android
Xcode

D. Debugger

React native debugger
Reactotron

III. Setup Application

A. For New Project

Step 1: generate rct project using command beesight rct

Step 2: cd project
Step 3: Install dependencies of the application with npm i

B. For Exist Project

Step 1: git clone this repo
Step 2: cd to the cloned repo
Step 3: Install dependencies of the application with npm i

C. Run Application

For iOS
react-native run-ios
For Android
Start Android emulator
react-native run-android

IV. STRUCTURE

A. Default Structure

android/ – This is the directory where all of the native Android code lives. If you dive in there you'll find .gradle files, .java files, and .xml files. This is the directory you would open with Android Studio. You'll rarely have to work in this directory.

ios/ – Like the android directory this is where all of your native iOS code lives. You'll find your xcode project in there, .plist files, .h files, .m files, etc. The ios template using CocoaPods [14], if you want to open your project in xcode you would install pod file with command *pod install* and open ios/<PROJECT_NAME>.xcworkspace. You'll rarely have to work in this directory.

index.js – This is the entry point for your app into the React Native code. It's where you'll want to register your app via AppRegistry

App/ – Where all of our app logic will be going
/Components/ – All components are stored and organized
<COMPONENT_NAME>/
<COMPONENT_NAME>.Styles.js – Contain style of view component
<COMPONENT_NAME>.View.js – Contain layout of view component
/Config/ – All application specific configuration falls in this folder.
AppConfig.js – production values.
DebugConfig.js – development-wide globals.
ReactotronConfig.js – Reactotron client settings.
ReduxPersist.js – rehydrate Redux state.
/Containers/ – Contain app screens and logic
<SCREEN_NAME>/
<SCREEN_NAME>.Screen.js – Contain layout of screen
<SCREEN_NAME>.Styles.js – Contain style of screen
<SCREEN_NAME>.Action.js – Contain Redux [15] action
<SCREEN_NAME>.Api.js – Contain Redux api server for data
<SCREEN_NAME>.Reducer.js – Contain Redux reducer for business logic
/Fonts/ – Where all of our fonts for app
/Images/ – Where all of our images for app
/I18n/ – Where all translation script file
/Navigation/ Where all configuration of navigation for app
/Redux/index.js – This is the file contain setup store for Redux reducer
/Sagas/index.js – This is the file contain connection definition between Redux action and Redux business function rule
/Themes/ – Where all configuration for themes

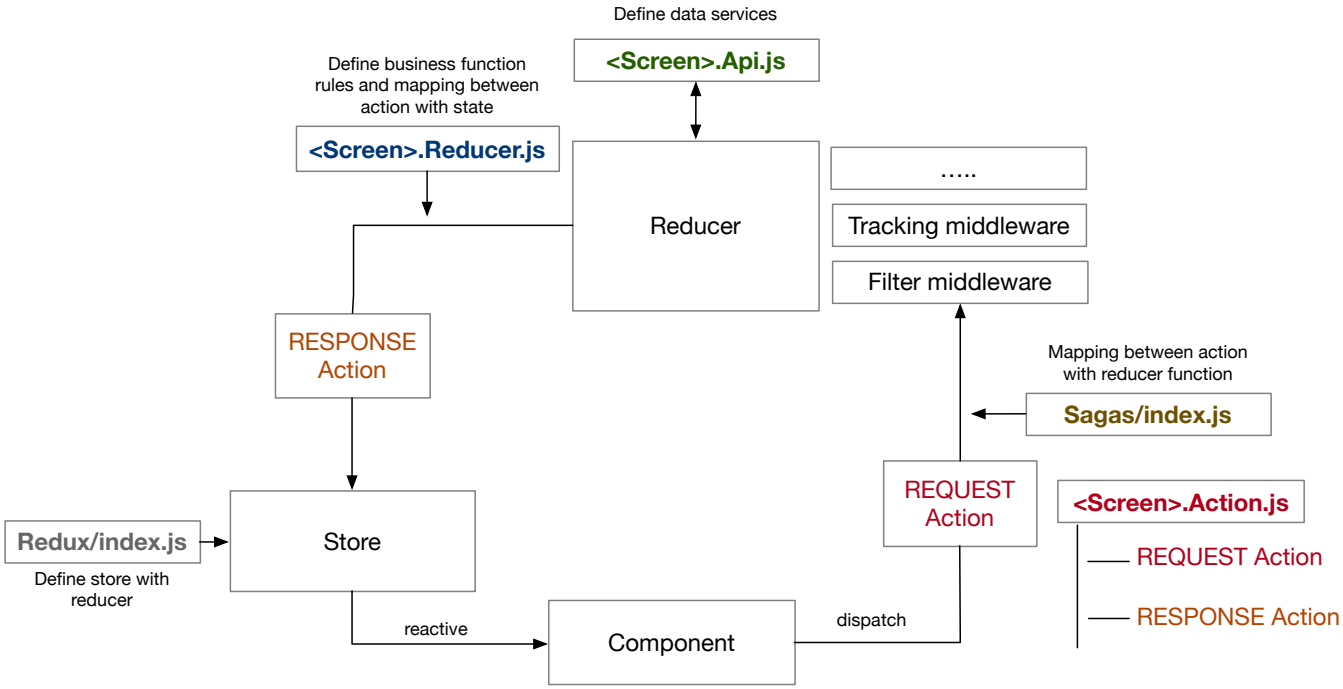

Figure 1: Redux flow

V. CONVENTION

A. *Basic Rules*

Only include one React component per file. However, multiple *Stateless, or Pure, Components [16]* are allowed per file. eslint: *react/no-multi-comp [17]*

Always use JSX syntax.

Do not use React.createElement unless you're initializing the app from a file that is not JSX.

B. *Class vs React.createClass vs stateless*

If you have internal state and/or refs, prefer class extends *Component* over *React.createClass*. eslint: *react/prefer-es6-class [18] react/prefer-stateless-function [19]*

```
// bad
const Listing = React.createClass({
  // ...
  render() {
    return <div>{this.state.hello}</div>;
  }
});

// good
class Listing extends Component {
  // ...
  render() {
    return <div>{this.state.hello}</div>;
  }
}
```

And if you don't have state or refs, prefer normal functions (not arrow functions) over classes:

```
// bad
class Listing extends Component {
  render() {
    return <div>{this.props.hello}</div>;
  }
}

// bad (relying on function name inference is discouraged)
const Listing = ({ hello }) => (
  <div>{hello}</div>
);
```

```
// good
function Listing({ hello }) {
  return <div>{hello}</div>;
}
```

C. Naming

Extensions: Use .js extension for React native components.

Filename: Use PascalCase for filenames. E.g., ReservationCard.js.

Reference Naming: Use PascalCase for React components and camelCase for their instances. eslint: *react/jsx-pascal-case [20]*

```
// bad
import reservationCard from './ReservationCard';

// good
import ReservationCard from './ReservationCard';

// bad
const ReservationItem = <ReservationCard />;

// good
const reservationItem = <ReservationCard />;
```

Component Naming: Use the filename as the component name. For example, *ReservationCard.js* should have a reference name of *ReservationCard*. However, for root components of a directory, use index.js as the filename and use the directory name as the component name:

```
// bad
import Footer from './Footer/Footer';

// bad
import Footer from './Footer/index';

// good
import Footer from './Footer';
```

Higher-order Component Naming: Use a composite of the higher-order component's name and the passed-in component's name as the *displayName* on the generated component. For example, the higher-order component *withFoo()*, when passed a component Bar should produce a component with a *displayName* of *withFoo(Bar)*.

```
// bad
export default function withFoo(WrappedComponent) {
  return function WithFoo(props) {
    return <WrappedComponent {...props} foo />;
  }
}

// good
export default function withFoo(WrappedComponent) {
  function WithFoo(props) {
    return <WrappedComponent {...props} foo />;
  }

  const wrappedComponentName = WrappedComponent.displayName
    || WrappedComponent.name
    || 'Component';

  WithFoo.displayName = `withFoo(${wrappedComponentName})`;
  return WithFoo;
}
```

Props Naming: Avoid using DOM component prop names for different purposes.

```
// bad
<MyComponent style="fancy" />

// bad
<MyComponent className="fancy" />

// good
<MyComponent variant="fancy" />
```

D. Declaration

Do not use *displayName* for naming components. Instead, name the component by reference.

```
// bad
```

```
export default React.createClass({
  displayName: 'ReservationCard',
  // stuff goes here
});

// good
export default class ReservationCard extends Component {
}
```

E. Alignment

Follow these alignment styles for JSX syntax. eslint: *react/jsx-closing-bracket-location [21] react/jsx-closing-tag-location [22]*

```
// bad
<Foo superLongParam="bar"
     anotherSuperLongParam="baz" />

// good
<Foo
  superLongParam="bar"
  anotherSuperLongParam="baz"
/>

// if props fit in one line then keep it on the same line
<Foo bar="bar" />

// children get indented normally
<Foo
  superLongParam="bar"
  anotherSuperLongParam="baz"
>
  <Bee />
</Foo>
```

F. Quotes

Always use double quotes (") for JSX attributes, but single quotes (') for all other JS. eslint: *jsx-quotes [23]*

```
// bad
<Foo bar='bar' />

// good
<Foo bar="bar" />

// bad
<Foo style={{ left: "20px" }} />

// good
<Foo style={{ left: '20px' }} />
```

G. Spacing

Always include a single space in your self-closing tag. eslint: *no-multi-spaces [24], react/jsx-tag-spacing [25]*

```
// bad
<Foo/>

// very bad
<Foo                 />

// bad
<Foo
 />

// good
<Foo />
```

Do not pad JSX curly braces with spaces. eslint: *react/jsx-curly-spacing [26]*

```
// bad
<Foo bar={ baz } />

// good
<Foo bar={baz} />
```

H. Props

Always use camelCase for prop names.

```
// bad
<Foo
  UserName="hello"
  phone_number={12345678}
/>

// good
<Foo
  userName="hello"
  phoneNumber={12345678}
/>
```

Omit the value of the prop when it is explicitly true. eslint: *react/jsx-boolean-value [27]*

```
// bad
<Foo
  hidden={true}
/>

// good
<Foo
  hidden
/>

// good
<Foo hidden />
```

Always include an alt prop on <img> tags. If the image is presentational, alt can be an empty string or the <img> must have role="presentation". eslint: *jsx-a11y/alt-text [28]*

```
// bad
<img src="hello.jpg" />

// good
<img src="hello.jpg" alt="Me waving hello" />

// good
<img src="hello.jpg" alt="" />

// good
<img src="hello.jpg" role="presentation" />
```

Do not use words like "image", "photo", or "picture" in <img> alt props. eslint: *jsx-a11y/img-redundant-alt [29]*

```
// bad
<img src="hello.jpg" alt="Picture of me waving hello" />

// good
<img src="hello.jpg" alt="Me waving hello" />
```

Use only valid, *non-abstract ARIA roles [30]*. eslint: *jsx-a11y/aria-role [31]*

```
// bad - not an ARIA role
<div role="datepicker" />

// bad - abstract ARIA role
<div role="range" />

// good
<div role="button" />
```

Do not use *accessKey* on elements. eslint: *jsx-a11y/no-access-key [32]*

```
// bad
<div accessKey="h" />

// good
<div />
```

Avoid using an array index as *key* prop, prefer a unique ID.

```
// bad
{todos.map((todo, index) =>
  <Todo
    {...todo}
    key={index}
  />
)}

// good
```

```
{todos.map(todo => (
  <Todo
    {...todo}
    key={todo.id}
  />
))}
```

Always define explicit *defaultProps* for all non-required props.

```
// bad
function SFC({ foo, bar, children }) {
  return <div>{foo}{bar}{children}</div>;
}
SFC.propTypes = {
  foo: PropTypes.number.isRequired,
  bar: PropTypes.string,
  children: PropTypes.node,
};

// good
function SFC({ foo, bar, children }) {
  return <div>{foo}{bar}{children}</div>;
}
SFC.propTypes = {
  foo: PropTypes.number.isRequired,
  bar: PropTypes.string,
  children: PropTypes.node,
};
SFC.defaultProps = {
  bar: '',
  children: null,
};
```

Use spread props sparingly.

Exceptions:

- HOCs that proxy down props and hoist *propTypes*

```
function HOC(WrappedComponent) {
  return class Proxy extends Component {
    Proxy.propTypes = {
      text: PropTypes.string,
      isLoading: PropTypes.bool
    };

    render() {
      return <WrappedComponent {...this.props} />
    }
  }
}
```

- Spreading objects with known, explicit props. This can be particularly useful when testing React components with Mocha's *beforeEach* construct.

```
export default function Foo {
  const props = {
    text: '',
    isPublished: false
  }

  return (<div {...props} />);
}
```

Notes for use: Filter out unnecessary props when possible. Also, use *prop-types-exact* [33] to help prevent bugs.

```
// good
render() {
  const { irrelevantProp, ...relevantProps  } = this.props;
  return <WrappedComponent {...relevantProps} />
}

// bad
render() {
  const { irrelevantProp, ...relevantProps  } = this.props;
  return <WrappedComponent {...this.props} />
}
```

I. *Refs*

Always use ref callbacks. eslint: *react/no-string-refs [34]*

```
// bad
<Foo
  ref="myRef"
/>

// good
<Foo
  ref={(ref) => { this.myRef = ref; }}
/>
```

J. Parentheses

Wrap JSX tags in parentheses when they span more than one line. eslint: *react/jsx-wrap-multilines [35]*

```
// bad
render() {
  return <MyComponent variant="long body" foo="bar">
           <MyChild />
         </MyComponent>;
}

// good
render() {
  return (
    <MyComponent variant="long body" foo="bar">
      <MyChild />
    </MyComponent>
  );
}

// good, when single line
render() {
  const body = <div>hello</div>;
  return <MyComponent>{body}</MyComponent>;
}
```

K. Tags

Always self-close tags that have no children. eslint: *react/self-closing-comp [36]*

```
// bad
<Foo variant="stuff"></Foo>

// good
<Foo variant="stuff" />
```

If your component has multi-line properties, close its tag on a new line. eslint: *react/jsx-closing-bracket-location [37]*

```
// bad
<Foo
  bar="bar"
  baz="baz" />

// good
<Foo
  bar="bar"
  baz="baz"
/>
```

L. Methods

Use arrow functions to close over local variables.

```
function ItemList(props) {
  return (
    <ul>
      {props.items.map((item, index) => (
        <Item
          key={item.key}
          onClick={() => doSomethingWith(item.name, index)}
        />
      ))}
    </ul>
  );
}
```

Bind event handlers for the render method in the constructor. eslint: *react/jsx-no-bind [38]*

```
// bad
class extends Component {
  onClickDiv() {
```

```
    // do stuff
  }

  render() {
    return <div onClick={this.onClickDiv.bind(this)} />;
  }
}

// good
class extends Component {
  constructor(props) {
    super(props);

    this.onClickDiv = this.onClickDiv.bind(this);
  }

  onClickDiv() {
    // do stuff
  }

  render() {
    return <div onClick={this.onClickDiv} />;
  }
}
```

Do not use underscore prefix for internal methods of a React component.

```
// bad
React.createClass({
  _onClickSubmit() {
    // do stuff
  },

  // other stuff
});

// good
class extends Component {
  onClickSubmit() {
    // do stuff
  }

  // other stuff
}
```

Be sure to return a value in your render methods. eslint: *react/require-render-return [39]*

```
// bad
render() {
  (<div />);
}

// good
render() {
  return (<div />);
}
```

M. Ordering

optional *static* methods
constructor
getChildContext
componentWillMount
componentDidMount
componentWillReceiveProps
shouldComponentUpdate
componentWillUpdate
componentDidUpdate
componentWillUnmount
clickHandlers or *eventHandlers* like *onClickSubmit()* or *onChangeDescription()*
getter methods for *render* like *getSelectReason()* or *getFooterContent()*
optional *render* methods like *renderNavigation()* or *renderProfilePicture()*
render

REFERENCES

[1] Beesight cli tool, https://beesightsoft.github.io/beesight
[2] React native, https://facebook.github.io/react-native
[3] Ignite, https://infinite.red/ignite
[4] Reactotron, https://infinite.red/reactotron
[5] React native debugger, https://github.com/jhen0409/react-native-debugger
[6] Nativebase, https://nativebase.io
[7] Fonts, https://medium.com/p/ccc9aacf9e5e
[8] I18n, https://github.com/infinitered/ignite-i18n
[9] Environment configuration, https://github.com/luggit/react-native-config
[10] Unique version, https://medium.com/p/94b70da7612f
[11] Splash screen, https://github.com/crazycodeboy/react-native-splash-screen
[12] App icon, https://github.com/dwmkerr/app-icon
[13] Style guide, https://github.com/airbnb/javascript/blob/master/react/README.md
[14] CocoaPods, https://cocoapods.org
[15] Redux, https://redux.js.org
[16] Stateless, or Pure, Components, https://facebook.github.io/react/docs/reusable-components.html#stateless-functions
[17] react/no-multi-comp, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-multi-comp.md#ignorestateless
[18] react/prefer-es6-class, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-es6-class.md
[19] react/prefer-stateless-function, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-stateless-function.md
[20] react/jsx-pascal-case, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-pascal-case.md
[21] react/jsx-closing-bracket-location, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md
[22] react/jsx-closing-tag-location, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-tag-location.md
[23] jsx-quotes, https://eslint.org/docs/rules/jsx-quotes
[24] no-multi-spaces, https://eslint.org/docs/rules/no-multi-spaces
[25] react/jsx-tag-spacing, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-tag-spacing.md
[26] react/jsx-curly-spacing, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-curly-spacing.md
[27] react/jsx-boolean-value, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-boolean-value.md
[28] jsx-a11y/alt-text, https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/alt-text.md
[29] jsx-a11y/img-redundant-alt, https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/img-redundant-alt.md
[30] ARIA roles, https://www.w3.org/TR/wai-aria/#usage_intro
[31] jsx-a11y/aria-role, https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/aria-role.md
[32] jsx-a11y/no-access-key, https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/no-access-key.md
[33] prop-types-exact, https://www.npmjs.com/package/prop-types-exact
[34] react/no-string-refs, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-string-refs.md
[35] react/jsx-wrap-multilines, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-wrap-multilines.md
[36] react/self-closing-comp, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/self-closing-comp.md
[37] react/jsx-closing-bracket-location, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md
[38] react/jsx-no-bind, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md
[39] react/require-render-return, https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/require-render-return.md