**Coding Guidelines | React Native**

This style guide is mostly based on the standards that are currently prevalent in JavaScript,

although some conventions (i.e async/await or static class fields) may still be included or

prohibited on a case-by-case basis. recommended in this guide.

Currently,

anything prior to stage

3 is not included nor

**Basic Rules**

* Only include one React component per file.
  + However,

multiple

[Stateless,](https://facebook.github.io/react/docs/reusable-components.html#stateless-functions)

[or Pure,](https://facebook.github.io/react/docs/reusable-components.html#stateless-functions)

[Components](https://facebook.github.io/react/docs/reusable-components.html#stateless-functions)

are allowed per file.

eslint: [react/no-multi-comp](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-multi-comp.md#ignorestateless).

* Always use JSX syntax.
* Do not use React.createElement

JSX.

unless you’re initializing the app from a file that is not

**Class vs React.createClass vs stateless**

* If you have internal state and/or refs, prefer class extends

React.Component over React.createClass. [function](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-stateless-function.md)

eslint:

[react/prefer-es6-class](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-es6-class.md) [react/prefer-stateless-](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-stateless-function.md)

// bad

const Listing = React.createClass({

// ...

render() {

return <div>{this.state.hello}</div>;

}

});

// good

class Listing extends React.Component {

// ...

render() {

return <div>{this.state.hello}</div>;

}

}

And if you don’t have state or refs, classes:

prefer normal functions

(not arrow functions)

over

// bad

class Listing extends React.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>;

}

**Mixins**

* [Do not use mixins](https://facebook.github.io/react/blog/2016/07/13/mixins-considered-harmful.html).

Why? Mixins introduce implicit dependencies, cause name clashes, and cause snowballing

complexity. Most use cases for mixins can be accomplished in better ways via components,

higher-order components, or utility modules.

**Naming**

* **Extensions**: Use .jsx

extension for React components.

eslint:

[react/jsx-filename-extension](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-filename-extension.md)

* **Filename**:

Use PascalCase for filenames.

E.g.,

ReservationCard.jsx.

* **Reference Naming**: Use PascalCase for React components and camelCase for their

instances.

eslint:

[react/jsx-pascal-case](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-pascal-case.md)

// 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.jsx should have a reference name of ReservationCard.

However,

for root components of a directory, name as the component name:

use index.jsx as the filename and use the directory

// 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).

|  |  |
| --- | --- |
|  | Why? A component’s displayName may be used by developer tools or in error messages, and having a value that clearly expresses this relationship helps  people understand what is happening. |
| // 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.

|  |  |
| --- | --- |
|  | Why? People expect props like style and className to mean one specific thing. Varying this API for a subset of your app makes the code less readable and  less maintainable, and may cause bugs. |
| // bad  <MyComponent style="fancy" />  // bad  <MyComponent className="fancy" />  // good  <MyComponent variant="fancy" /> | |

**Declaration**

* Do not use displayName

reference.

for naming components.

Instead,

name the component by

// bad

export default React.createClass({ displayName: 'ReservationCard',

// stuff goes here

});

// good

export default class ReservationCard extends React.Component {

}

**Alignment**

* Follow these alignment styles for JSX syntax.

[location](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md) [react/jsx-closing-tag-location](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-tag-location.md)

eslint:

[react/jsx-closing-bracket-](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md)

// 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"

>

<Quux />

</Foo>

// bad

{showButton &&

<Button />

}

// bad

{

showButton &&

<Button />

}

// good

{showButton && (

<Button />

)}

// good

{showButton && <Button />}

**Quotes**

* Always use double quotes

(")

for JSX attributes,

but single quotes

(')

for all other JS.

eslint:

[jsx-quotes](https://eslint.org/docs/rules/jsx-quotes)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Why? Regular HTML attributes also typically  attributes mirror this convention. | use double | quotes | instead | of | single, | so JSX |
| // bad | | | | | | |
| <Foo bar='bar' /> | | | | | | |
| // good | | | | | | |
| <Foo bar="bar" /> | | | | | | |
| // bad | | | | | | |
| <Foo style={{ left: "20px" }} /> | | | | | | |
| // good | | | | | | |
| <Foo style={{ left: '20px' }} /> | | | | | | |

**Spacing**

* Always include a single space in your self-closing tag.

[tag-spacing](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-tag-spacing.md)

eslint:

[no-multi-spaces](https://eslint.org/docs/rules/no-multi-spaces), [react/jsx-](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-tag-spacing.md)

// good

<Foo />

// bad

<Foo

/>

/>

// very bad

<Foo

// bad

<Foo/>

* Do not pad JSX curly braces with spaces.

eslint:

[react/jsx-curly-spacing](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-curly-spacing.md)

// bad

<Foo bar={ baz } />

// good

<Foo bar={baz} />

**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](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-boolean-value.md)

// 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](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/alt-text.md)

// 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](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/img-redundant-alt.md)

|  |  |
| --- | --- |
|  | Why? Screenreaders already announce img elements as images, so there is no  need to include this information in the alt text. |
| // 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](https://www.w3.org/TR/wai-aria/#usage_intro).

eslint:

[jsx-a11y/aria-role](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/aria-role.md)

// 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](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/no-access-key.md)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Why? Inconsistencies  using screenreaders | between keyboard shortcuts and keyboard  and keyboards complicate accessibility. | commands | used | by | people |
| // bad | | | | | |
| <div accessKey="h" /> | | | | | |
| // good | | | | | |
| <div /> | | | | | |

* Avoid using an array index as key [key](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-array-index-key.md)

prop,

prefer a stable ID.

eslint:

[react/no-array-index-](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-array-index-key.md)

Why? Not using a stable ID [is an anti-pattern](https://medium.com/%40robinpokorny/index-as-a-key-is-an-anti-pattern-e0349aece318) because it can negatively impact performance

and cause issues with component state.

We don’t recommend using indexes for keys if the order of items may change.

// 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.

Why? propTypes are a form of documentation,

and providing defaultProps means the reader of

your code doesn’t have to assume as much. certain type checks.

In addition, it can mean that your code can omit

// 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.

Why? Otherwise you’re more likely to pass unnecessary props down to components.

And for

React v15.6.1 Exceptions:

and older,

you could [pass invalid HTML attributes to the DOM](https://reactjs.org/blog/2017/09/08/dom-attributes-in-react-16.html).

* HOCs that proxy down props and hoist propTypes

function HOC(WrappedComponent) {

return class Proxy extends React.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: prevent bugs.

Filter out unnecessary props when possible.

Also,

use

[prop-types-exact](https://www.npmjs.com/package/prop-types-exact) to help

// bad render() {

const { irrelevantProp, ...relevantProps } = this.props;

return <WrappedComponent {...this.props} />

}

// good render() {

const { irrelevantProp, ...relevantProps } = this.props; return <WrappedComponent {...relevantProps} />

}

**Refs**

* Always use ref callbacks.

eslint:

[react/no-string-refs](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-string-refs.md)

// bad

<Foo

ref="myRef"

/>

// good

<Foo

ref={(ref) => { this.myRef = ref; }}

/>

**Parentheses**

* Wrap JSX tags in parentheses when they span more than one line.

[multilines](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-wrap-multilines.md)

eslint:

[react/jsx-wrap-](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-wrap-multilines.md)

// 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>;

}

Tags

* Always self-close tags that have no children.

eslint:

[react/self-closing-comp](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/self-closing-comp.md)

// 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](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md)

// bad

<Foo

bar="bar" baz="baz" />

// good

<Foo

bar="bar" baz="baz"

/>

**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](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md)

Why? A bind call in the render path creates a brand new function on every single

render.

// bad

class extends React.Component { onClickDiv() {

// do stuff

}

render() {

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

}

}

// good

class extends React.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.

Why? Underscore prefixes are sometimes used as a convention in other languages to

denote privacy. But, unlike those languages, there is no native support for privacy in

JavaScript, everything is public. Regardless of your intentions, adding underscore prefixes

to your properties does not actually make them private, and any property (underscore-

prefixed or not) should be treated as being public. See issues [#1024](https://github.com/airbnb/javascript/issues/1024), and [#490](https://github.com/airbnb/javascript/issues/490) for a

more in-depth discussion.

// bad React.createClass({

\_onClickSubmit() {

// do stuff

},

// other stuff

});

// good

class extends React.Component { onClickSubmit() {

// do stuff

}

// other stuff

}

* Be sure to return a value in your render

methods.

eslint:

[react/require-render-return](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/require-render-return.md)

// bad

render() {

(<div />);

}

// good render() {

return (<div />);

}

**Ordering**

* Ordering for class extends React.Component:

1.optional static methods 2.constructor 3.getChildContext 4.componentWillMount 5.componentDidMount 6.componentWillReceiveProps 7.shouldComponentUpdate 8.componentWillUpdate 9.componentDidUpdate 10.componentWillUnmount

1. clickHandlers or eventHandlers like onClickSubmit() or onChangeDescription()
2. getter methods for render like getSelectReason() or getFooterContent() 13.optional render methods like renderNavigation() or renderProfilePicture() 14.render

* How to define propTypes, defaultProps, contextTypes, etc...

import React from 'react';

import PropTypes from 'prop-types';

const propTypes = {

id: PropTypes.number.isRequired, url: PropTypes.string.isRequired, text: PropTypes.string,

};

const defaultProps = { text: 'Hello World',

};

class Link extends React.Component { static methodsAreOk() {

return true;

}

render() {

return <a href={this.props.url} data-id={this.props.id}>{this.props.text}</a>;

}

}

Link.propTypes = propTypes; Link.defaultProps = defaultProps;

export default Link;

* Ordering for React.createClass:

1.displayName 2.propTypes 3.contextTypes 4.childContextTypes 5.mixins

6.statics 7.defaultProps 8.getDefaultProps 9.getInitialState 10.getChildContext 11.componentWillMount 12.componentDidMount

13.componentWillReceiveProps 14.shouldComponentUpdate 15.componentWillUpdate 16.componentDidUpdate 17.componentWillUnmount

eslint:

[react/sort-comp](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/sort-comp.md)

1. clickHandlers or eventHandlers like onClickSubmit() or onChangeDescription()
2. getter methods for render like getSelectReason() or getFooterContent() 20.optional render methods like renderNavigation() or renderProfilePicture() 21.render

**isMounted**

* Do not use isMounted.

eslint:

[react/no-is-mounted](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-is-mounted.md)

Why? [isMounted is an anti-pattern](https://facebook.github.io/react/blog/2015/12/16/ismounted-antipattern.html), is not available when using ES6

classes,

and is

its way to being officially deprecated.