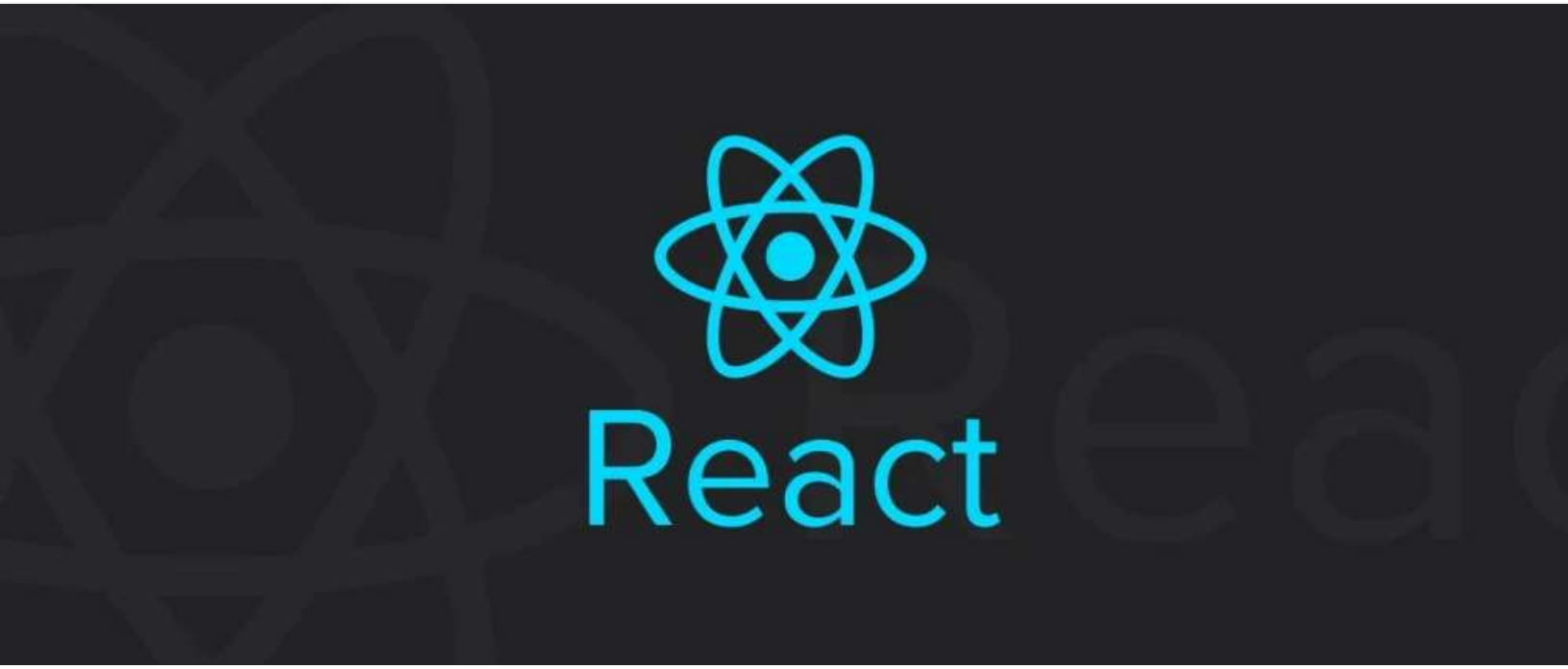

Lawson
Posted on Apr 29, 2021

React Style Guide

#react #codequality #javascript #beginners

React Style Guide (2021)

src: the original post: https://github.com/airbnb/javascript/tree/master/react

> I wanted to create this post to mark back what's important when someone is just starting out with React. Therefore, I would like to receive comments from everyone.

Basic Rules

- Only include one React component per file.
- Always use JSX syntax.

Naming

- **List itemExtensions**: Use `.jsx` extension for React components. eslint: react/jsx-filename-extension
- **Filename**: Use `PascalCase` for filenames. E.g., **ReservationCard**.jsx.

- **Reference Naming**: Use `PascalCase` for React components and `camelCase` for their instances.

```
// 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, use `index.jsx` 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)`.

  > 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
```

```
// 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;
}
```

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 function ReservationCard(props) {
    return ()
}
```

Alignment

- Follow these alignment styles for JSX syntax. eslint: `react/jsx-closing-bracket-location` `react/jsx-closing-tag-location`

```
// 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 />}

// good
{someReallyLongConditional
  && anotherLongConditional
  && (
    <Foo
      superLongParam="bar"
      anotherSuperLongParam="baz"
    />
  )
}

// good
{someConditional ? (
  <Foo />
) : (
  <Foo
    superLongParam="bar"
    anotherSuperLongParam="baz"
  />
)}
```

Props

- Always use **camelCase** for prop names, or **PascalCase** if the prop value is a React component.

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

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

- Omit the value of the prop when it is explicitly `true`. eslint: `react/jsx-boolean-value`

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

// good
<Foo
  hidden
/>

// very good
<Foo hidden />
```

- Avoid using an array index as `key` prop, prefer a stable ID. eslint: `react/no-array-index-key`

> Why? Not using a stable ID is an anti-pattern 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. In addition, it can mean that your code can omit certain type checks.

```
// 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 and older, you could pass invalid HTML attributes to the DOM.

Exceptions:

- 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:
Filter out unnecessary props when possible. Also, use prop-types-exact to help prevent bugs.

```
// 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`

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

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

Parentheses

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

```
// 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>;
}
```

Methods

- Use arrow functions to close over local variables. It is handy when you need to pass additional data to an event handler. Although, make sure they do not massively hurt performance, in particular when passed to custom components that might be PureComponents, because they will trigger a possibly needless rerender every time.

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

```
        />
      ))}
    </ul>
  );
}
```

- Bind event handlers for the render method in the constructor. eslint: `react/jsx-no-bind`

> Why? A bind call in the render path creates a brand new function on every single render. Do not use arrow functions in class fields, because it makes them challenging to test and debug, and can negatively impact performance, and because conceptually, class fields are for data, not logic.

```
// bad
class extends React.Component {
  onClickDiv() {
    // do stuff
  }

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

// very bad
class extends React.Component {
  onClickDiv = () => {
    // do stuff
  }

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

// 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, and #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`

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

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

Top comments (7)

Javier Gonzalez Toro • Jun 3 '21

i wonder what are the best way to **ordering in react hooks**, i came with this approach, but i don't know if are the best way...

```
import React, { useState, useEffect } from 'react'
// import other package, components, utils, styles, etc

const Wrapper = () => {
  // declare your variables,
  const [foo, setFoo] = useState(null)
  const pathname = 'home'

  // declare you custom hooks, functions, etc
  const handleFoo = (bar) => {
    if (bar === 'something')  {
      setFoo(bar)
    }
  }

  // declare your useEffects, useCallback, etc
  useEffect = () => {
    console.log(`Hello ${foo}`)
  }, [foo]

  return (
    <div className="wrapper">
     {/* show stuff... */}
    </div>
  )
}

export default Wrapper
```

Lawson • Jun 6 '21

Well, I am also using that way. But usually it looks like :

`constructor -> hooks (useEffect, useCallBack... ) -> other functions`

```
// constructor
const [foo, setFoo] = useState(null)
const pathname = 'home'

// life cycles ( hooks ), others function ( such as event handlers,... )
const handleFoo = (bar) => {
    if (bar === 'something')  {
      setFoo(bar)
    }
```

```
  }


 useEffect = () => {
   console.log(`Hello ${foo}`)
 }, [foo]

// renders
return (
    <div className="wrapper">
     {/* show stuff... */}
    </div>
 )
```

I think if you're comfortable with that approach, keep it :)

You can get more info : React Ordering

Cooper Riehl • Apr 30 '21

Very useful post for React newbies, like me!

Lawson • Apr 30 '21

Tks bro, it's good to hear 😂

Shubhendra Singh Chauhan • Apr 29 '21

It's a great blog. These are really some of the great ways by which we can improve our code style. 👏

Lawson • Apr 30 '21

thanks so much bro

Aya Bouchiha • Apr 30 '21

🙏🙏🙏

Trending in React

The React community is exploring design patterns like `Hooks` and `Higher Order Components`, discussing AI in web app generation, and focusing on optimizing component re-rendering.

React Design Patterns

Anuradha Aggarwal · Jul 9

#react #webdev #designpatterns #beginners

GPT Web App Generator - Let AI create a full-stack React & Node.js codebase based on your description 🤖🦁

Matija Sosic for Wasp · Jul 11

#javascript #webdev #gpt3 #react

Next.js and GraphQL: The Perfect Combination for Full Stack Development

Francisco Mendes · Jul 1

#nextjs #graphql #typescript #react

How to Prevent Unnecessary React Component Re-Rendering

Femi Akinyemi · Jul 8

#react #javascript #performance #frontend

Typescript ▨ CRUD API: Next.js, Tailwind, tRPC, Prisma Postgres, Docker

Francesco Ciulla · Jul 4

#typescript #react #nextjs #webdev

WORK
Software Engineer at Lawson
JOINED
Dec 6, 2020

More from Lawson

Why are the results like this?
#javascript #beginners #programming #codenewbie

Using Absolute Paths in React Native
#reactnative #javascript #beginners #react