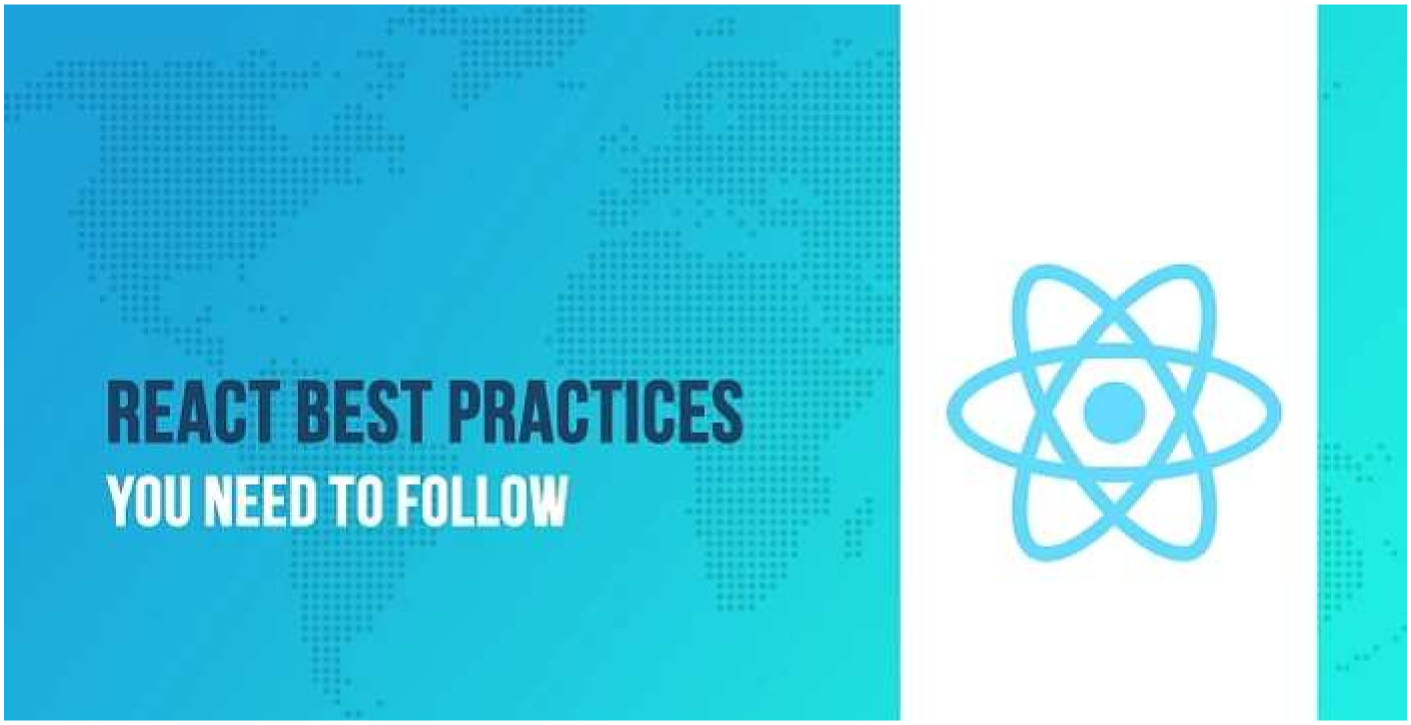
codeinwp

15 React Best Practices You Need to Follow in 2021

by Priya / december 28, 2020 / web & app frameworks



If you're a frontend developer engaged in building highly interactive user interfaces, you've most likely got React in your toolkit. While working on your React-powered creations, you should be careful to do things in tune with the React best practices. This will help to keep your code better organized.

As you know, React is a library created by Facebook and it allows for integration with many interesting components. In fact, any developer can create their own components and make them accessible to the community.

Today, we take the topic head on and show you **the most fundamental of the React best practices:**

15 @reactjs best practices you need to follow in 2021 👨‍💻📋

CLICK TO TWEET

As we all know, with React, it's possible to have huge components that execute a number of tasks. But a better way to design components is to keep them small, so that one component corresponds to one function. Ideally, a single component should render a specific bit of your page or modify a particular behavior. There are many advantages to this:

- Function-specific components can be standalone, which makes testing and maintenance easier.
- Each small component can be reused across multiple projects.
- Components executing general functions can be made available to the community.
- With smaller components, it's easier to implement performance optimizations.
- It's easier to update smaller components.
- Bigger components have to perform harder and may be difficult to maintain.

The balance between creating one concise component and creating multiple function-specific components can vary from organization to organization. After all, you can have as many components as you want, and recombine them in any way you want to achieve the same end result.

♻️ 2. Reusability is important, so keep creation of new components to the minimum required

By sticking to the rule of one function = one component, you can improve the reusability of components. What this means is that you should skip trying to build a new component for a function if there already exists a component for that function.

By reusing components across your project or across any number of projects, not only will you achieve consistency, you'll also be contributing to the community.

On the other hand, if any component becomes huge, unwieldy and difficult to maintain, it's better to break it up into as many smaller components as required.

For example, you can even go further and create a `Button` component that can handle icons. Then, each time you need a button, you'll have a component to use. Making it more modular will allow you to cover many cases with the same piece of code. You have to aim somewhere in the middle. Your components should be abstract enough, but shouldn't be overly complex.

```
class IconButton extends React.Component {
  [...]
  render() {
    return (
        </button>
      );
    }
  }
```

🤝 3. Consolidate duplicate code – DRY your code

A common rule for all code is to keep it as brief and concise as possible.

It's no different here too, since React best practices also instruct you to keep code brief and precise. One way to do this is to avoid duplication – Don't Repeat Yourself (DRY).

You can achieve this by scrutinizing the code for patterns and similarities. If you find any, it's possible you're repeating code and there's scope to eliminate duplication. Most likely, a bit of rewriting can make it more concise.

This relies heavily on the reusability principle in React. Let's say you want to add multiple buttons that contain icons, instead of adding the markup for each button, you can simply use the `IconButton` component that we shown above. You could even go further by mapping everything into an array.

```
const buttons = ['facebook', 'twitter', 'youtube'];

return (
  <div>
    {
      buttons.map( (button) => {
        return (
          <IconButton
            onClick={doStuff( button )}
            iconClass={button}
          />
        );
      } )
);
```

🎨 4. Put CSS in JavaScript

When you start working on a project, it is a common practice to keep all the CSS styles in a single SCSS file. The use of a global prefix prevents any potential name collisions. However, when your project scales up, this solution may not be feasible.

There are many libraries that enable you to write CSS in JS. EmotionJS and Glamorous are the two most popular CSS in JS libraries.

Here’s an example of using EmotionJS in your project. EmotionJS can generate complete CSS files for your production. First, install EmotionJS using `npm`.

```
npm install --save @emotion/core
```

Next, you need to import EmotionJS in your application.

```
import { jsx } from '@emotion/core'
```

You can set the properties of an element as shown in the snippet below:

```
let Component = props => {
  return (
    <div
      css={{
        border: '1px'
      }}
      {...props}
    />
```

Here is the link to the complete documentation of EmotionJS.

📝 5. Comment only where necessary

Attach comments to code only where necessary. This is not only in keeping with React best practices, it also serves two purposes at the same time:

- It'll keep code visually clutter free.
- You'll avoid a potential conflict between comment and code, if you happen to alter the code at some later point in time.

📛 6. Name the component after the function

It's a good idea to name a component after the function that it executes so that it's easily recognizable.

For example, `ProductTable` – it conveys instantly what the component does. On the other hand, if you name the component based on the need for the code, it can confuse you at a future point of time.

Another example, it's preferable to name a component `Avatar` so that it can be used anywhere – for authors, users or in comments. Instead, if we name the component `AuthorAvatar` in the context of its usage, we'd be limiting the utility of that component.

Besides, naming a component after the function makes it more useful to the community as it's more likely to be discovered.

🦒 7. Use capitals for component names

If, like most folks, you're using JSX (a JavaScript extension), the names of the components you create need to begin with uppercase letters. For instance, you'll name components as `SelectButton` instead of `selectbutton`, or `Menu` instead of `menu`. We do this so that JSX can identify them differently from default HTML tags.

Earlier React versions maintained a list of all built-in names to differentiate them from custom names. But as the list needed constant updating, that was scrapped and capitals became the norm.

🐫 8. Mind the other naming conventions

When working with React, you are generally using JSX (JavaScript Extension) files. Any component that you create for React should therefore be named in Pascal case, or upper camel case. This translates to names without spaces and the capitalizing the first letter of every word.

If you want to create a function that submits a form, you should name it `SubmitForm` in upper camel case, rather than `submitForm`, `submit_form`, or `submit_form`. Upper camel case is more commonly called Pascal case. Here is a further list of examples of variable and function names in Pascal case.

🎭 9. Separate stateful aspects from rendering

Components in React can be stateful or stateless. Stateful components store information about the component's state and provide the necessary context. In contrast, stateless components have no memory and cannot give context to other parts of the UI. They only receive props (inputs) from parent component and return you JSX elements. They are scalable and reusable, and similar to pure function in JavaScript.

One of React best practices is to keep your stateful data-loading logic separate from your rendering stateless logic. It's better to have one stateful component to load data and another stateless component to display that data. This reduces the complexity of the components.

The later React versions v16.8 have a new feature – React Hooks, which write stateful function-related components. This may eventually eliminate the need for class-based components.

For example, your app is fetching some data on mount. What you want to do is manage the data in the main component and pass the complex render task to a sub-component as props.

```
import RenderTable from './RenderTable';

class Table extends Component {
  state = { loading: true };

  render() {
    const { loading, tableData } = this.state;
    return loading ? <Loading/> : <RenderTable data=
    componentDidMount() {
      fetchTableData().then( tableData => {
        this.setState( { loading: false, tableData } );
      } );
    }
  }
```

🚀 10. Code should execute as expected and be testable

Really, this rule needs no explanation. The code you write should behave as expected, and be testable easily and quickly. It's a good practice to name your test files identical to the source files with a `.test` suffix. It'll then be easy to find the test files.

You can use JEST to test your React code.

📁 11. All files related to any one component should be in a single folder

Keep all files relating to any one component in a single folder, including styling files.

If there's any small component used by a particular component only, it makes sense to keep these smaller components all together within that component folder. The hierarchy will then be easy to understand – large components have their own folder and all their smaller parts are split into sub-folders. This way, you can easily extract code to any other project or even modify the code whenever you want.

For instance, for the `Form` component, all pieces such as CSS files, icons, images, tests and any other sub-components relating to *Form* should all reside in the same folder. If you name files sensibly, and keep related files together logically, you'll not have any difficulty finding them later.

🧰 12. Use tools like Bit

One of React best practices that helps to organize all your React components is the use of tools like Bit.

These tools help to maintain and reuse code. Beyond that, it helps code to become discoverable, and promotes team collaboration in building components. Also, code can be synced across

📚 13. Use snippet libraries

Code snippets help you to keep up with the best and most recent syntax. They also help to keep your code relatively bug free, so this is one of the React best practices that you should not miss out on.

There are many snippet libraries that you can use, like, ES7 React, Redux, JS Snippets, etc.

✍️ 14. Write tests for all code

In any programming language, adequate testing ensures that any new code added to your project integrates well with existing code and does not break existing functionality. It is a good idea to write tests for any new component that you create. As a good practice, you should create a `__Test__` directory within your component's directory to house all relevant tests.

You can broadly divide tests in React into two parts: testing the functionality of components using a React app, and tests on your complete application once it renders in the browser. You can use cross browser testing tools for tests in the latter category.

For the former, you can use a JavaScript test runner, Jest to emulate the HTML DOM using `jsdom` to test React components. While a completely accurate test is only possible in a browser on a real device, Jest provides a good approximation of the real testing environment during the development phase of your project.

🔗 15. Follow linting rules, break up lines that are too long

Linting is a process wherein we run a program that analyses code for potential errors.

Mostly, we use it for language-related issues. But it can also fix many other issues automatically, particularly code style. Using a linter in your React code helps to keep your code relatively error- and bug-free.

Final words on React best practices

I hope this list of React best practices is going to help you put your projects on the right track, and avoid any potential problems later down the road.

You can also check our list of the best free React Native templates to help you develop mobile apps.

If you have any React-related questions, feel free to submit them in the comments below.

codeinwp

CLICK TO TWEET

...

Don’t forget to join our crash course on speeding up your WordPress site. With some simple fixes, you can reduce your loading time by even 50-80%:

Code examples by Andrei Băicuș. Content updates by Shaumik Daityari

SHOW COMMENTS

Or start the conversation in our **[Facebook group for WordPress professionals]**. Find answers, share tips, and get help from other WordPress experts. Join now (it’s free)!

RELATED ARTICLES

codeinwp

in 2021

10+ Best Laravel Admin Templates for 2021 (Free and Premium)

15 of the Best Wix Apps for Your New Website ...

Material Design Templates for Vue, Angular, React (Material UI)

Bootstrap vs Foundation vs Bulma vs Semantic vs UIkit

10+ Best Tailwind CSS Templates for Your Next Project

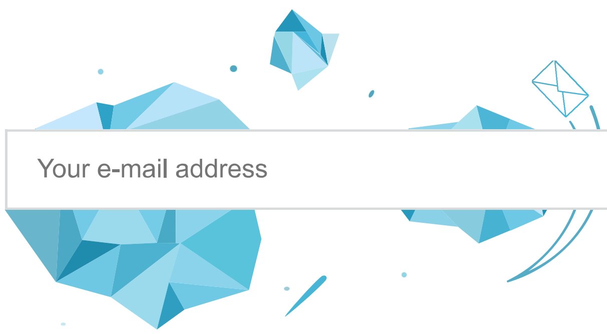
codeinwp

20+ Best Free Bootstrap Templates for ReactJS in 2021

15+ Free and Premium VueJS Admin Templates Built With Bootstrap

14 of the Best JavaScript Libraries and Frameworks to Try …

</> Get the CodeinWP Weekly!



Your e-mail address

Subscribe

</> Best Articles

50+ Best Free WordPress Themes

The Complete GDPR Guide

How to Make a WordPress Website: Ultimate Guide

Best WordPress Hosting

codeinwp

Managed WordPress Hosting Guide

Top 5 eCommerce Hosting Solutions

Best Web Hosting for Small Business

Best eCommerce Platforms

Mailchimp Competitors & Alternatives

Best Email Marketing Tools

Best SMTP Providers Compared

1Password vs LastPass vs Dashlane

How Much Does a VPN Cost?

Best Live Chat Software Solutions

Best Business Phone Services for Small Business

How to Build Your Own Website

FOLLOW US ON

</> Latest Articles

5 Best Windows Hosting Services on the Market (Linux Hosting

WordPress 5.8 Is Here, Block Pattern Directory Now Live, Automattic

How to Screenshot on Windows: Simple Ways to Capture Your

codeinwp

FEATURED ON

CodeinWP stands for all-things-WordPress. From web design to freelancing and from development to business, your questions are covered.

OUR NETWORK

EDITORS' PICKS