React Best Practices to up Your Game

React has remained one of the most popular libraries for creating user interfaces when building web applications. It's widely used by many companies and has an active community.

As a React developer, understanding how the library works isn't the only thing you need to build projects that are user-friendly, easily scalable, and maintainable.

It's also necessary to understand certain conventions that'll enable you to write clean React code. This will not only help you serve your users better, but it'll make it easier for you and other developers working on the project to maintain the code base.

In this tutorial, we'll start by talking about some of the common challenges React developers face, and then dive into some of the best practices you can follow to help you write React code in a more efficient way.

Let's get started!

Check Out Our Video Guide On React Best Practices

Challenges React Developers Face

In this section, we'll discuss some of the major challenges that React developers face during and after the building of web apps.

All the challenges you'll see in this section can be avoided by following best practices, which we'll discuss in detail later on.

We'll start with the most basic problem that affects beginners.

Prerequisites To React

One of the major challenges faced by React developers is understanding how the library works, along with the prerequisites to using it.

Before learning React, you are required to know a couple of things. Since React uses JSX, knowing HTML and JavaScript is a must. Of course, you should also know CSS or a modern CSS framework for designing your web apps.

In particular, there are core JavaScript concepts and functionalities that you should know before diving into React. Some of them, which mostly fall under ES6, include:

- Arrow functions
- Rest operator
- Spread operator
- Modules
- Destructuring
- Array methods
- Template literals
- Promises
- `let` and `const` variables

The JavaScript topics listed above will help you understand as a beginner how React works.

You'd also learn about new concepts in React, like:

- Components

- JSX
- State management
- Props
- Rendering elements
- Event handling
- Conditional rendering
- Lists and keys
- Forms and form validation
- Hooks
- Styling

Having a solid understanding of React concepts and the prerequisites to using the library will help you utilize its features efficiently.

But don't let this overwhelm you. With constant practice and learning, you can quickly get a good grasp of how to use React to build awesome projects. It's similar to learning a new programming language — it just takes a bit of time and practice to understand.

State Management

Updating the state/value of your variables in React works differently from how you'd do it using vanilla JavaScript.

In JavaScript, updating a variable is as simple as assigning a new value to it using the equal to operator (=). Here's an example:

```
var x = 300;
function updateX(){
  x = 100;
}
updateX();
console.log(x);
```

```
// 100
```

In the code above, we created a variable called `x` with an initial value of `300`.

Using the equal to operator, we assigned a new value of `100` to it. This was written inside an `updateX` function.

In React, updating the state/value of your variables works differently. Here's how:

```
import { useState } from 'react';
function App() {
  const [x, setX] = useState(300)
  let updateX =()=>{
    setX(100);
  }
  return (
    <div className="App">
    <h1>{x}</h1>
    <button onClick={updateX}>Update X</button>
    </div>
  );
}
export default App;
```

When updating the state of a variable in React, you make use of the `useState` Hook. There are three things to note when using this Hook:

- The variable name
- A function for updating the variable
- The initial value/state of the variable

In our example, `x` is the name of the variable, and `setX` is the function for updating the value of `x`, while the initial value (`300`) of `x` is passed in as a parameter to the `useState` function:

```
const [x, setX] = useState(300)
```

In order to update the state of `x`, we made use of the `setX` function:

```
import { useState } from 'react';
let updateX =()=>{
  setX(100);
}
```

So the `updateX` function invokes the `setX` function, which then sets the value of `x` to `100`.

While this seems to work perfectly for updating the state of your variables, it increases the complexity of your code in very large projects. Having loads of State Hooks makes the code very hard to maintain and understand, especially as your project scales.

Another problem with using the State Hook is that these variables created are not shared across the different components that make up your app. You'd still have to make use of Props to pass the data from one variable to another.

Luckily for us, there are libraries built to handle state management efficiently in React. They even allow you to create a variable once and use it anywhere you want to in your React app. Some of these libraries include Redux, Recoil, and Zustand.

The problem with choosing a third-party library for state management is that you'd be forced to learn new concepts foreign to what you've already learned in React. Redux, for instance,

was known for having a lot of boilerplate code, which made it difficult for beginners to grasp (although this is being fixed with Redux Toolkit, which lets you write less code than you would with Redux).

Maintainability and Scalability

As the user requirements of a product continues to change, there is always the need to introduce changes to the code that makes up the product.

It's often difficult to scale your code when that code isn't easy for the team to maintain. Difficulties like these arise from following bad practices when writing your code. They may seem to work perfectly at first, giving you the desired result, but anything that works "for now" is inefficient for the future and growth of your project.

In the next section, we'll go over some conventions that can help to improve how you write your React code.This will also help you collaborate better when working with a professional team.

React Best Practices

In this section, we'll talk about some of the best practices to follow when writing your React code. Let's dive right in.

1. Maintain Clear Folder Structure

Folder structures help you and other developers understand the arrangement of files and assets being used in a project.

With a good folder structure, it's easy to navigate around easily, saving time and helping avoid confusion. Folder structures differ with each team's preferences, but here are a few of the commonly used folder structures in React.

Grouping Folders by Features or Routes

Grouping files in your folder according to their routes and features helps keep everything about a particular feature in one space. For example, if you have a user dashboard, you can have the JavaScript, CSS, and test files relating to the dashboard in one folder.

Here’s an example to demonstrate that:

```
dashboard/
index.js
dashboard.css
dashboard.test.js
home/
index.js
Home.css
HomeAPI.js
Home.test.js
blog/
index.js
Blog.css
Blog.test.js
```

As can be seen above, each core feature of the app has all its files and assets stored in the same folder.

Grouping Similar Files

Alternatively, you can group similar files in the same folder. You can also have individual folders for Hooks, components, and so on. Check out this example:

```
hooks/
useFetchData.js
usePostData.js
components/
Dashboard.js
Dashboard.css
Home.js
Home.css
Blog.js
Blog.css
```

You don’t have to strictly follow these folder structures when coding. If you have a specific way to order your files, go for it. As long as you and other developers have a clear understanding of the file structure, you’re good to go!

2. Institute a Structured Import Order

As your React application continues to grow, you’re bound to make extra imports. The structure of your imports go a long way in helping you understand what makes up your components.

As a convention, grouping similar utilities together seems to work fine. For instance, you can group external or third party imports separately from local imports.

Take a look at the following example:

```
import { Routes, Route } from "react-router-dom";
import { createSlice } from "@reduxjs/toolkit";
import { Menu } from "@headlessui/react";
```

```
import Home from "./Home";
import logo from "./logo.svg";
import "./App.css";
```

In the code above, we first grouped third party libraries together (these are libraries we had to install beforehand).

We then imported files we created locally like stylesheets, images, and components.

For the sake of simplicity and easy understanding, our example doesn't depict a very large codebase, but bear in mind being consistent with this format of imports will help you and other developers understand your React app better.

You can go further grouping your local files according to file types if that works for you — that is, grouping components, images, stylesheets, Hooks, and so on separately under your local imports.

Here's an example:

```
import Home from "./Home";
import About from "./About"
import Contact from "./Contact"
import logo from "./logo.svg";
import closeBtn from "./close-btn.svg"
import "./App.css";
import "Home.css";
```

3. Adhere To Naming Conventions

Naming conventions help improve code readability. This is not only applicable to component names but even your variable names, all the way to your Hooks.

The React documentation does not offer any official pattern for naming your components. The most used naming conventions are camelCase and PascalCase.

PascalCase is mostly used for component names:

```
import React from 'react';
function StudentList() {
  return (
    <div>StudentList</div>
  )
}
export default StudentList
```

The component above is named `StudentList`, which is much more readable than `Studentlist` or `studentlist`.

On the other hand, the camelCase naming convention is mostly used for naming variables, Hooks, functions, arrays, and so on:

```
const [firstName, setFirstName] = useState("Ihechikara");
const studentList = [];
const studentObject = {};
const getStudent = () => {}
```

4. Use a Linter

A linter tool helps improve code quality. One of the most popular linter tools for JavaScript and React is ESlint. But how exactly does this help with improving code quality?

A linter tool helps with consistency in a code base. When using a tool like ESLint, you can set the rules you want every developer working on the project to follow. These rules may include requirements for using double quotes instead of single quotes, braces around arrow functions, a particular naming convention, and so much more.

The tool observes your code and then notifies you when a rule has been broken. The keyword or line that breaks the rule would usually be underlined in red.

Since every developer has their own style of coding, linter tools can help with code uniformity.

Linter tools can also help us fix bugs easily. We can see spelling errors, variables that have been declared but not used, and other such functionalities. Some of these bugs can be fixed automatically as you code.

Tools like ESLint are built into most code editors so you get linter functionalities on the go. You can also configure it to suit your coding requirements.

5. Employ Snippet Libraries

The cool thing about using a framework with an active community is the availability of tools being created to make development easier.

Snippet libraries can make development faster by providing prebuilt code that developers use often.

A good example is the ES7+ React/Redux/React-Native snippets extension, which has a lot of helpful commands for generating prebuilt code. For instance, if you want to create a React functional component without typing out all the code, all you need to do using the extension is type `rfce` and hit **Enter**.

The command above will go on to generate a functional component with a name that corresponds with the file name. We generated the code below using the ES7+ React/Redux/React-Native snippets extension:

```
import React from 'react';
function StudentList() {
  return (
    <div>StudentList</div>
  )
}
export default StudentList
```

Another useful snippet tool is the Tailwind CSS IntelliSense extension, which simplifies the process of styling web pages with Tailwind CSS. The extension can help you with autocompletion by suggesting utility classes, syntax highlighting, and linting functionalities. You can even see what your colors look like while coding.

6. Combine CSS and JavaScript

When working on large projects, using different stylesheet files for each component can make your file structure bulky and hard to navigate around.

A solution to this problem is to combine your CSS and JSX code. You can use frameworks/libraries like Tailwind CSS and Emotion for this.

Here's what styling with Tailwind CSS looks like:

```
<p className="font-bold mr-8">resource edge</p>
```

The code above give the paragraph element a bold font and adds some margin on the right. We are able to do this using the framework's utility classes.

Here’s how you’d style an element using Emotion:

```
<h1
css={css`
  color: black;
  font-size: 30px;
`}
>
Hello World!
</h1>
```

7. Limit Component Creation

One of the core features of React is code reusability. You can create a component and reuse its logic as many times as possible without rewriting that logic.

With that in mind, you should always limit the number of components you create. Not doing so bloats the file structure with unnecessary files that shouldn’t exist in the first place.

We’ll use a very easy example to demonstrate this:

```
function UserInfo() {
  return (
    <div>
    <h1>My name is Ihechikara.</h1>
    </div>
  );
}
export default UserInfo
```

The component above shows the name of a user. If we were to create a different file for every user, we'd eventually have an unreasonable number of files. (Of course, we're using user information to keep things simple. In a real life situation, you may be dealing with a different type of logic.)

To make our component reusable, we can make use of Props. Here's how:

```
function UserInfo({userName}) {
  return (
    <div>
    <h1>My name is {userName}.</h1>
    </div>
  );
}
export default UserInfo
```

After that, we can then import this component and use it as many times as we want:

```
import UserInfo from "./UserInfo";
function App() {
  return (
    <div className="App">
    <UserInfo userName={"Ihechikara"} />
    <UserInfo userName={"John"} />
    <UserInfo userName={"Jane"} />
    </div>
  );
}
export default App;
```

Now we have three different instances of the `UserInfo` component coming from the logic created in one file instead of having three separate files for each user.

8. Implement Lazy Loading

Lazy loading is very useful as your React app grows. When you have a big codebase, load time for your web pages slows down. This is because the whole app has to be loaded every time for every user.

“Lazy loading” is a term used for various implementations. Here, we associate it with JavaScript and React, but you can also implement lazy loading on images and videos.

By default, React bundles and deploys the whole application. But we can change this behavior using lazy loading, otherwise known as code splitting.

Basically, you can limit what section of your app gets loaded at a particular point. This is accomplished by splitting your bundles and only loading those relevant to the user’s requirements. For instance, you can first load only the logic required for the user to sign in, then load the logic for the user’s dashboard only after they have successfully signed in.

9. Employ Reusable Hooks

Hooks in React let you harness some of React’s additional functionalities, like interacting with your component’s state and running after-effects in relation to certain state changes in your component. We can do all this without writing class components.

We can also make Hooks reusable so we don’t have to retype the logic in every file they’re used. We do this by creating custom Hooks that can be imported anywhere in the app.

In the example below, we’ll create a Hook for fetching data from external APIs:

```
import { useState, useEffect } from "react";
function useFetchData(url) {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch(url)
    .then((res) => res.json())
    .then((data) => setData(data))
    .catch((err) => console.log(`Error: ${err}`));
  }, [url]);
  return { data };
}
export default useFetchData;
```

We have created a Hook for fetching data from APIs above. Now it can be imported into any component. This saves us the stress of typing out all that logic in every component where we have to fetch external data.

The type of custom Hooks we can create in React is limitless, so it's up to you to decide how to use them. Just remember that if it's a functionality that has to be repeated across different components, you should definitely make it reusable.

10. Log and Manage Errors

There are different ways of handling errors in React like using error boundaries, try and catch blocks or using external libraries like `react-error-boundary`.

The built in error boundaries that was introduced in React 16 was a functionality for class components so we won't discuss it because it's advisable that you use functional components instead of class components.

On the other hand, using a `try` and `catch` block only works for imperative code, but not declarative code. This means that it's not a good option when working with JSX.

Our best recommendation would be to use a library like react-error-boundary. This library provides functionalities that can be wrapped around your components, which will help you detect errors while your React app is being rendered.

11. Monitor and Test Your Code

Testing your code during development helps you write maintainable code. Unfortunately, this is something a lot of developers neglect.

Although many may argue that testing isn't a big deal when building your web app, it comes with innumerable advantages. Here are just a few:

- Testing helps you detect errors and bugs.
- Detecting bugs leads to improved code quality.
- Unit tests can be documented for data collection and future reference.
- Early bug detection saves you the cost of paying developers to put out the fire the bug could cause if left unchecked.
- Bug-free apps and sites earn trust and loyalty from their audience, which leads to greater growth.

You can use tools like Jest or React Testing Library to test your code. There are plenty of testing tools you can choose from — it all comes down to the one that works best for you.

You can also test your React apps as you build them by running the apps in your browser. You'd usually get any detected error displayed on the screen. This is similar to developing WordPress sites using DevKinsta — a tool that allows you to design, develop, and deploy WordPress sites on your local machine.

12. Make Use of Functional Components

Using functional components in React comes with a lot of advantages: You write less code, it's easier to read, and the beta version of the official React documentation is being rewritten using functional components (Hooks), so you should definitely get used to using them.

With functional components, you don't have to worry about using the `this` or using classes. You can also manage your component's state easily by writing less code thanks to Hooks.

Most of the updated resources you'd find on React make use of functional components, making it easy to understand and follow helpful guides and resources created by the community when you run into problems.

13. Stay Up to Date With React Version Changes

As time goes, new functionalities will be introduced, and some old ones modified. The best way to keep track of this is to watch the official documentation.

You can also join React communities on social media to get information about changes when they happen.

Staying up to date with the current version of React will help you determine when to optimize or make changes to your code base for the best performance.

There are also external libraries built around React that you should be up to date with as well — like React Router, which is used for routing in React. Knowing what changes these libraries make can help you make relevant important changes to your app and make things easier for everyone working on the project.

Additionally, some functionalities can become deprecated and certain keywords can be changed when new versions are released. To be on the safe side, you should always read the documentation and guides when such changes are made.

14. Use a Fast, Secure Hosting Provider

If you want to make your web app accessible to everyone after building it, you'd have to host it. It is important that you use a fast and secure hosting provider.

Hosting your website gives you access to different tools that make scaling and managing your website easy. The server where your website is hosted makes it possible for the files on your local machine to be stored securely on the server. The overall benefit of hosting your website is that other people get to see the awesome stuff you've created.

There are a variety of platforms that provide free hosting services to developers like Firebase, Vercel, Netlify, GitHub Pages, or paid services like Azure, AWS, GoDaddy, Bluehost, and so

on.

You can also use Kinsta's Application Hosting platform. All you need to do is connect a GitHub repository, choose from Kinsta's 25 globally positioned data centers, and go. You'll receive access to fast setup, 24/7 support, top-of-the-line security, custom domains, advanced reporting and monitoring tools, and more.

Summary

Learning how to use React isn't all that's required to create outstanding web apps. As with any other framework like Angular, Vue, and so on, there are best practices that you should follow to help you build efficient products.

Following these React conventions not only helps your app, but it also has advantages for you as a frontend developer — you learn how to write efficient, scalable and maintainable code, and you stand out as a professional in your field.

So when building your next web app with React, bear these best practices in mind to make using and managing the product easy for both your users and your developers.

What other React best practices do you know that weren't mentioned in this article? Share them in the comments below. Happy coding!