Las Mejores Prácticas de React para Mejorar tu Juego en 2023

React sigue siendo una de las librerías más populares para crear interfaces de usuario al desarrollar aplicaciones web. Es ampliamente utilizada por muchas empresas y tiene una comunidad activa.

Como desarrollador de React, entender cómo funciona la librería no es lo único que necesitas para construir proyectos que sean fáciles de usar, escalar y mantener.

También es necesario comprender ciertas convenciones que te permitirán escribir código React limpio. Esto no sólo te ayudará a servir mejor a tus usuarios, sino que facilitará tu mantenimiento y el de otros desarrolladores que trabajen en el proyecto.

En este tutorial, empezaremos hablando de algunos de los retos comunes a los que se enfrentan los desarrolladores de React, y luego nos sumergiremos en algunas de las mejores prácticas que puedes seguir para ayudarte a escribir código React de una forma más eficiente.

¡Empecemos!

Echa un Vistazo a Nuestro Videotutorial sobre las Mejores Prácticas de React

Desafíos a los que se Enfrentan los Desarrolladores de React

En esta sección, hablaremos de algunos de los principales retos a los que se enfrentan los desarrolladores de React durante y después de la construcción de aplicaciones web.

Todos los retos que verás en esta sección pueden evitarse siguiendo las mejores prácticas, que trataremos en detalle más adelante.

Empezaremos por el problema más básico que afecta a los principiantes.

Requisitos Previos para React

Uno de los principales retos a los que se enfrentan los desarrolladores de React es comprender cómo funciona la librería, junto con los requisitos previos para utilizarla.

Antes de aprender React, es necesario que conozcas un par de cosas. Dado que React utiliza JSX, conocer HTML y JavaScript es imprescindible. Por supuesto, también debes conocer CSS o un framework CSS moderno para diseñar tus aplicaciones web.

En particular, hay conceptos y funcionalidades básicos de JavaScript que debes conocer antes de sumergirte en React. Algunos de ellos, que en su mayoría pertenecen a ES6, son:

- Funciones de flecha
- Operador Rest
- Operador Spread
- Módulos
- Desestructuración
- Métodos Array
- Literales de plantilla
- Promesas
- Variables `let` y `const`

Los temas de JavaScript enumerados anteriormente te ayudarán a comprender como principiante cómo funciona React.

También aprenderás nuevos conceptos de React, como:

- Componentes
- JSX
- Gestión de estados
- Props
- Elementos de renderizado
- Gestión de eventos
- Renderizado condicional
- Listas y claves
- Formularios y validación de formularios
- Hooks
- Estilo

Tener una sólida comprensión de los conceptos de React y de los requisitos previos para utilizar la biblioteca te ayudará a utilizar sus características de manera eficiente.

Pero no dejes que esto te abrume. Con práctica y aprendizaje constantes, puedes conseguir rápidamente una buena comprensión de cómo utilizar React para construir proyectos asombrosos. Es similar a aprender un nuevo lenguaje de programación — solo se necesita un poco de tiempo y práctica para entenderlo.

Gestión de Estados

Actualizar el estado/valor de tus variables en React funciona de forma diferente a como lo harías utilizando vanilla JavaScript.

En JavaScript, actualizar una variable es tan sencillo como asignarle un nuevo valor utilizando el operador igual a (=). Aquí tienes un ejemplo:

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

En el código anterior, hemos creado una variable llamada `x` con un valor inicial de `300`.

Utilizando el operador igual a, le asignamos un nuevo valor de `100`. Esto se ha escrito dentro de una función `updateX`.

En React, la actualización del estado/valor de tus variables funciona de forma diferente. Aquí tienes cómo:

```
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;
```

Al actualizar el estado de una variable en React, utilizas el Hook `useState`. Hay tres cosas que debes tener en cuenta al utilizar este Hook:

- El nombre de la variable
- Una función para actualizar la variable
- El valor/estado inicial de la variable

En nuestro ejemplo, `x` es el nombre de la variable, y `setX` es la función para actualizar el valor de `x`, mientras que el valor inicial (`300`) de `x` se pasa como parámetro a la función `useState`:

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

Para actualizar el estado de `x`, utilizamos la función `setX`:

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

Así, la función `updateX` invoca a la función `setX`, que a su vez establece el valor de `x` en `100`.

Aunque esto parece funcionar perfectamente para actualizar el estado de tus variables, aumenta la complejidad de tu código en proyectos muy grandes. Tener montones de Hooks de Estado hace que el código sea muy difícil de mantener y comprender, especialmente a medida que tu proyecto escala.

Otro problema de utilizar el Hook de Estado es que estas variables creadas no se comparten entre los distintos componentes que forman tu aplicación. Tendrías que seguir utilizando Props para pasar los datos de una variable a otra.

Por suerte para nosotros, existen bibliotecas creadas para gestionar el estado de forma eficiente en React. Incluso te permiten crear una variable una vez y utilizarla donde quieras en tu aplicación React. Algunas de estas bibliotecas son Redux, Recoil y Zustand.

El problema de elegir una biblioteca de terceros para la gestión de estados es que te verías obligado a aprender nuevos conceptos ajenos a lo que ya has aprendido en React. Redux, por ejemplo, era conocido por tener mucho código repetitivo, lo que dificultaba su comprensión a los principiantes (aunque esto se está arreglando con Redux Toolkit, que te permite escribir menos código del que escribirías con Redux).

Mantenimiento y Escalabilidad

A medida que cambian los requisitos de los usuarios de un producto, siempre es necesario introducir cambios en el código que lo compone.

A menudo es difícil escalar tu código cuando ese código no es fácil de mantener para el equipo. Dificultades como éstas surgen de seguir malas prácticas al escribir tu código. Puede parecer que funcionan perfectamente al principio, dándote el resultado deseado, pero todo lo que funciona «por ahora» es ineficaz para el futuro y el crecimiento de tu proyecto.

En la siguiente sección, repasaremos algunas convenciones que pueden ayudarte a mejorar la forma en que escribes tu código React. Esto también te ayudará a colaborar mejor cuando trabajes con un equipo profesional.

Buenas Prácticas de React

En esta sección, hablaremos de algunas de las mejores prácticas que debes seguir al escribir tu código React. Entremos de lleno.

1. Mantén una Estructura de Carpetas Clara

Las estructuras de carpetas te ayudan a ti y a otros desarrolladores a comprender la disposición de los archivos y activos que se utilizan en un proyecto.

Con una buena estructura de carpetas, es fácil navegar por ellas, lo que ahorra tiempo y ayuda a evitar confusiones. Las estructuras de carpetas difieren según las preferencias de cada equipo, pero aquí tienes algunas de las estructuras de carpetas más utilizadas en React.

Agrupar Carpetas por Funciones o Rutas

Agrupar los archivos en tu carpeta según sus rutas y características ayuda a mantener todo lo relacionado con una característica concreta en un mismo espacio. Por ejemplo, si tienes un panel de control de usuario, puedes tener los archivos JavaScript, CSS y de prueba relacionados con el panel de control en una carpeta.

Aquí tienes un ejemplo para demostrarlo:

```
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
```

Como se puede ver arriba, cada función principal de la aplicación tiene todos sus archivos y activos almacenados en la misma carpeta.

Agrupar Archivos Similares

Alternativamente, puedes agrupar archivos similares en la misma carpeta. También puedes tener carpetas individuales para Hooks, componentes, etc. Mira este ejemplo:

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

No tienes que seguir estrictamente estas estructuras de carpetas cuando programes. Si tienes una forma específica de ordenar tus archivos, hazlo. Siempre que tú y los demás desarrolladores comprendáis claramente la estructura de archivos, ¡estaréis listos!

2. Establece un Orden de Importación Estructurado

A medida que tu aplicación React siga creciendo, seguramente harás importaciones adicionales. La estructura de tus importaciones te ayudará en gran medida a comprender lo que forman tus componentes.

Como norma general, agrupar utilidades similares parece funcionar bien. Por ejemplo, puedes agrupar las importaciones externas o de terceros por separado de las importaciones locales.

Echa un vistazo al siguiente ejemplo:

```
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";
```

En el código anterior, primero agrupamos las bibliotecas de terceros (son bibliotecas que tuvimos que instalar previamente).

Después importamos archivos que creamos localmente, como hojas de estilo, imágenes y componentes.

Para simplificar y facilitar la comprensión, nuestro ejemplo no representa una base de código muy grande, pero ten en cuenta que ser coherente con este formato de importaciones te ayudará a ti y a otros desarrolladores a comprender mejor tu aplicación React.

Puedes ir más allá agrupando tus archivos locales de acuerdo a los tipos de archivo si eso te funciona — es decir, agrupando componentes, imágenes, hojas de estilo, Hooks, y demás por separado en tus importaciones locales.

Aquí tienes un ejemplo:

```
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. Sigue las Convenciones de Nomenclatura

Las convenciones de nomenclatura ayudan a mejorar la legibilidad del código. Esto no sólo es aplicable a los nombres de los componentes, sino incluso a los nombres de tus variables, hasta llegar a tus Hooks.

La documentación de React no ofrece ningún patrón oficial para nombrar tus componentes. Las convenciones de nomenclatura más utilizadas son camelCase y PascalCase.

PascalCase se utiliza sobre todo para los nombres de componentes:

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

El componente anterior se llama `StudentList`, que es mucho más legible que `Studentlist` o `studentlist`.

Por otro lado, la convención de nomenclatura camelCase se utiliza sobre todo para nombrar variables, Hooks, funciones, matrices, etc:

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

4. Utiliza un Linter

Una herramienta linter ayuda a mejorar la calidad del código. Una de las herramientas de linter más populares para JavaScript y React es ESlint. Pero, ¿cómo ayuda exactamente a mejorar la calidad del código?

Una herramienta linter ayuda a mantener la coherencia en una base de código. Cuando utilizas una herramienta como ESLint, puedes establecer las reglas que quieres que sigan todos los desarrolladores que trabajan en el proyecto. Estas reglas pueden incluir requisitos para el uso de comillas dobles en lugar de simples, llaves alrededor de las funciones de flecha, una convención de nomenclatura particular, y mucho más.

La herramienta observa tu código y te avisa cuando se incumple una norma. La palabra clave o la línea que infringe la norma suelen aparecer subrayadas en rojo.

Como cada desarrollador tiene su propio estilo de programación, las herramientas linter pueden ayudar a la uniformidad del código.

Las herramientas linter también pueden ayudarnos a corregir errores fácilmente. Podemos ver errores ortográficos, variables que se han declarado pero no se han utilizado, y otras funcionalidades por el estilo. Algunos de estos errores se pueden corregir automáticamente mientras programas.

Herramientas como ESLint están integradas en la mayoría de los editores de código, por lo que dispones de funcionalidades de linter sobre la marcha. También puedes configurarlo para que se adapte a tus requisitos de programación.

5. Emplea Bibliotecas de Snippets

Lo bueno de utilizar un framework con una comunidad activa es la disponibilidad de herramientas que se están creando para facilitar el desarrollo.

Las bibliotecas de snippets pueden hacer que el desarrollo sea más rápido al proporcionar código preconstruido que los desarrolladores utilizan a menudo.

Un buen ejemplo es la extensión de snippets ES7+ React/Redux/React-Native, que tiene un montón de comandos útiles para generar código preconstruido. Por ejemplo, si quieres crear un componente funcional React sin escribir todo el código, todo lo que tienes que hacer con la extensión es escribir `rfce` y pulsar **Intro**.

El comando anterior pasará a generar un componente funcional con un nombre que se corresponda con el nombre del archivo. Hemos generado el código siguiente utilizando la extensión de fragmentos ES7+ React/Redux/React-Native:

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

Otra útil herramienta de fragmentos es la extensión Tailwind CSS IntelliSense, que simplifica el proceso de estilizar páginas web con Tailwind CSS. La extensión puede ayudarte con el autocompletado sugiriéndote clases útiles, resaltado de sintaxis y funcionalidades de linting. Incluso puedes ver cómo son tus colores mientras programas.

6. Combina CSS y JavaScript

Cuando trabajas en proyectos grandes, utilizar diferentes archivos de hojas de estilo para cada componente puede hacer que tu estructura de archivos sea voluminosa y difícil de navegar.

Una solución a este problema es combinar tu código CSS y JSX. Para ello puedes utilizar frameworks/bibliotecas como Tailwind CSS y Emotion.

Este es el aspecto del estilo con Tailwind CSS:

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

El código anterior da al elemento párrafo una fuente en negrita y añade algo de margen a la derecha. Podemos hacer esto utilizando las clases de utilidad del framework.

Así es como se aplica estilo a un elemento utilizando Emotion:

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

7. Limitar la Creación de Componentes

Una de las características principales de React es la reutilización del código. Puedes crear un componente y reutilizar su lógica tantas veces como sea posible sin reescribir esa lógica.

Teniendo esto en cuenta, siempre debes limitar el número de componentes que creas. No hacerlo infla la estructura de archivos con archivos innecesarios que no deberían existir en primer lugar.

Utilizaremos un ejemplo muy sencillo para demostrarlo:

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

El componente anterior muestra el nombre de un usuario. Si tuviéramos que crear un archivo diferente para cada usuario, acabaríamos teniendo un número irrazonable de archivos. (Por supuesto, estamos utilizando la información del usuario para simplificar las cosas. En una situación de la vida real, puede que estés tratando con un tipo de lógica diferente)

Para que nuestro componente sea reutilizable, podemos hacer uso de Props. Aquí tienes cómo:

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

Después, podemos importar este componente y utilizarlo tantas veces como queramos:

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

Ahora tenemos tres instancias diferentes del componente `UserInfo` procedentes de la lógica creada en un único archivo, en lugar de tener tres archivos distintos para cada usuario.

8. Implementar la Carga Diferida

La carga diferida es muy útil a medida que crece tu aplicación React. Cuando tienes una gran base de código, el tiempo de carga de tus páginas web se ralentiza. Esto se debe a que toda

la aplicación tiene que cargarse cada vez para cada usuario.

«Carga Diferida» es un término que se utiliza para varias implementaciones. Aquí lo asociamos a JavaScript y React, pero también puedes implementar la carga diferida en imágenes y vídeos.

Por defecto, React agrupa y despliega toda la aplicación. Pero podemos cambiar este comportamiento utilizando la carga diferida, también conocida como división del código.

Básicamente, puedes limitar qué sección de tu aplicación se carga en un momento determinado. Esto se consigue dividiendo tus paquetes y cargando sólo aquellos que sean relevantes para las necesidades del usuario. Por ejemplo, puedes cargar primero sólo la lógica necesaria para que el usuario se registre, y luego cargar la lógica del panel de control del usuario sólo después de que se haya registrado correctamente.

9. Utiliza Hooks Reutilizables

Los Hooks en React te permiten aprovechar algunas de las funcionalidades adicionales de React, como interactuar con el estado de tu componente y ejecutar efectos posteriores en relación con determinados cambios de estado en tu componente. Podemos hacer todo esto sin escribir componentes de clase.

También podemos hacer que los Hooks sean reutilizables para no tener que volver a escribir la lógica en cada archivo en el que se utilicen. Para ello, creamos Hooks personalizados que pueden importarse a cualquier parte de la aplicación.

En el siguiente ejemplo, crearemos un Hook para obtener datos de APIs externas:

```
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;
```

Arriba hemos creado un Hook para obtener datos de APIs. Ahora puede importarse a cualquier componente. Esto nos ahorra el estrés de escribir toda esa lógica en cada componente en el que tengamos que obtener datos externos.

El tipo de Hooks personalizados que podemos crear en React es ilimitado, así que depende de ti decidir cómo utilizarlos. Sólo recuerda que si se trata de una funcionalidad que debe repetirse en distintos componentes, deberías hacerla reutilizable.

10. Registrar y Gestionar Errores

Hay diferentes formas de gestionar los errores en React, como utilizar límites de error, bloques try y catch o utilizar bibliotecas externas como `react-error-boundary`.

Los límites de error incorporados que se introdujeron en React 16 eran una funcionalidad para componentes de clase, así que no hablaremos de ella porque es aconsejable que utilices componentes funcionales en lugar de componentes de clase.

Por otra parte, utilizar un bloque `try` y `catch` sólo funciona para código imperativo, pero no para código declarativo. Esto significa que no es una buena opción cuando se trabaja con JSX.

Nuestra mejor recomendación sería utilizar una biblioteca como react-error-boundary. Esta biblioteca proporciona funcionalidades que pueden envolverse alrededor de tus componentes, lo que te ayudará a detectar errores mientras se renderiza tu aplicación React.

11. Monitoriza y Prueba tu Código

Probar tu código durante el desarrollo te ayuda a escribir código mantenible. Por desgracia, esto es algo que muchos desarrolladores descuidan.

Aunque muchos puedan argumentar que las pruebas no son gran cosa cuando construyes tu aplicación web, tienen innumerables ventajas. Aquí tienes unas cuantas:

- Las pruebas te ayudan a detectar errores y fallos.
- La detección de errores mejora la calidad del código.
- Las pruebas unitarias pueden documentarse para recopilar datos y como referencia futura.
- La detección temprana de errores te ahorra el coste de pagar a los desarrolladores para apagar el incendio que el error podría causar si no se controla.
- Las aplicaciones y sitios sin errores ganan la confianza y fidelidad de su público, lo que conduce a un mayor crecimiento.

Puedes utilizar herramientas como Jest o React Testing Library para probar tu código. Hay muchas herramientas de pruebas entre las que puedes elegir — todo se reduce a la que mejor se adapte a ti.

También puedes probar tus aplicaciones React a medida que las construyes, ejecutándolas en tu navegador. Normalmente aparecerá en pantalla cualquier error detectado. Esto es similar al desarrollo de sitios WordPress utilizando DevKinsta — una herramienta que te permite diseñar, desarrollar y desplegar sitios WordPress en tu máquina local.

12. Utiliza Componentes Funcionales

Utilizar componentes funcionales en React tiene muchas ventajas: Escribes menos código, es más fácil de leer, y la versión beta de la documentación oficial de React se está reescribiendo utilizando componentes funcionales (Hooks), así que definitivamente deberías acostumbrarte a utilizarlos.

Con los componentes funcionales, no tienes que preocuparte de utilizar el `this` ni de usar clases. También puedes gestionar fácilmente el estado de tu componente escribiendo menos código gracias a los Hooks.

La mayoría de los recursos actualizados que encontrarás sobre React utilizan componentes funcionales, lo que facilita la comprensión y el seguimiento de guías y recursos útiles creados por la comunidad cuando te encuentres con problemas.

13. Mantente al Día con los Cambios de Versión de React

Con el paso del tiempo, se introducirán nuevas funcionalidades y se modificarán algunas antiguas. La mejor forma de estar al día es consultando la documentación oficial.

También puedes unirte a las comunidades React en las redes sociales para obtener información sobre los cambios cuando se produzcan.

Estar al día de la versión actual de React te ayudará a determinar cuándo optimizar o hacer cambios en tu base de código para obtener el mejor rendimiento.

También hay bibliotecas externas construidas en torno a React con las que también deberías estar actualizado — como React Router, que se utiliza para el enrutamiento en React. Saber qué cambios hacen estas bibliotecas puede ayudarte a hacer cambios importantes en tu aplicación y facilitar las cosas a todos los que trabajan en el proyecto.

Además, algunas funcionalidades pueden quedar obsoletas y ciertas palabras clave pueden cambiar cuando se publiquen nuevas versiones. Para estar seguro, siempre debes leer la documentación y las guías cuando se realicen estos cambios.

14. Utiliza un Proveedor de Alojamiento Rápido y Seguro

Si quieres que tu aplicación web sea accesible a todo el mundo después de crearla, tendrás que alojarla. Es importante que utilices un proveedor de alojamiento rápido y seguro.

Alojar tu sitio web te da acceso a diferentes herramientas que facilitan el escalado y la gestión de tu sitio web. El servidor donde está alojado tu sitio web hace posible que los archivos de tu máquina local se almacenen de forma segura en el servidor. El beneficio general de alojar tu sitio web es que otras personas pueden ver las cosas increíbles que has creado.

Hay una gran variedad de plataformas que ofrecen servicios de alojamiento gratuitos a los desarrolladores, como Firebase, Vercel, Netlify, GitHub Pages, o servicios de pago como Azure, AWS, GoDaddy, Bluehost, etc.

También puedes utilizar la plataforma de Alojamiento de Aplicaciones de Kinsta. Todo lo que tienes que hacer es conectar un repositorio de GitHub, elegir entre los 25 centros de datos de Kinsta posicionados globalmente, y listo. Tendrás acceso a una configuración rápida, soporte 24/7, seguridad de primera línea, dominios personalizados, herramientas avanzadas de informes y monitorización, y mucho más.

Resumen

Aprender a usar React no es todo lo que se necesita para crear aplicaciones web excepcionales. Al igual que con cualquier otro framework como Angular, Vue, entre otros, hay buenas prácticas que debes seguir para ayudarte a crear productos eficientes.

Seguir estas convenciones de React no sólo ayuda a tu aplicación, sino que también tiene ventajas para ti como desarrollador frontend — aprendes a escribir código eficiente, escalable y mantenible, y destacas como profesional en tu campo.

Así que cuando construyas tu próxima aplicación web con React, ten en cuenta estas buenas prácticas para que el uso y la gestión del producto resulten fáciles tanto para tus usuarios como para tus desarrolladores.

¿Qué otras buenas prácticas de React conoces que no se hayan mencionado en este artículo? Compártelas en los comentarios. ¡Feliz programación!