Buenas prácticas en React

A continuación, veremos 7 buenas prácticas generales que te ayudarán a guiar tu toma de decisiones en React.

1. Diseña componentes simples, reusables y de única responsabilidad

React es muy flexible y los desarrolladores pueden crear componentes guiados por su imaginación, pero cuando esa flexibilidad se descontrola, empieza a jugar en contra de los principios de ingeniería aplicados al código.

Antes de tirar código piensa qué necesidades funcionales debes suplir y divide el problema en múltiples responsabilidades. Luego, asocia dichas funciones al componente responsable. Cada componente debería estar empaquetado en una sola función y usar sus propios métodos para cumplir con su parte.

Ten en cuenta que tu componente será reusable a futuro, entonces desarróllalo entendible, intuitivo y fácil de evolucionar.

2. Identifica similitudes, reescribe código y sé más conciso

No repitas código. Cuando el código va creciendo, gracias a nuevas funcionalidades, el programador debe identificar constantemente patrones que le permitan simplificar instrucciones.

El pensamiento algorítmico y el conocimiento de los lenguajes involucrados se convierten en herramientas importantes para no repetir tu código.

3. Estructura ordenadamente tus carpetas para que cualquiera entienda el proyecto

Trata de que todos los archivos relacionados a un componente estén en una misma carpeta, nombrada de manera entendible según su responsabilidad en el código.

Además, debes pensar en que tu proyecto debe estar listo para que cualquier desarrollador pueda entender la relación de los componentes, viendo la estructura de carpetas y asociandola con la interfaz del proyecto. Pensar en Atomic design (Sistemas de diseño: Diseño atómico) ayuda a estructurar los componentes de manera ordenada y jerárquica.

4. Usa los React Hooks

Las últimas versiones de React (a partir de la React 16.8 Hooks : 90% código más conciso) incluyeron el concepto de los React Hooks, que facilitan el manejo organizado de estados dentro de los componentes. Estos también ayudan a diseñar componentes stateless basados en funciones, que al ser súper eficientes, pueden llegar a reemplazar la necesidad de componentes stateful basados en clases.

Para optimizar tu código, recuerda decirle a React que omita aplicar un efecto (con el hook de use Effect) si algún valor no ha cambiado. Para hacer esto, pasa un arreglo como segundo argumento a use Effect

```
useEffect(() => {
        document.title = `You clicked ${count} times`;
}, [count]); // Only re-run the effect if count changes
```

5. Ubica los estados, variables y funciones según el contexto

Como desarrollador, constantemente debes pensar en preguntas como: ¿de quién es la responsabilidad funcional? ¿Qué componente general orquesta los estados principales de la app? ¿mi código es autoexplicativo y preciso para saber dónde está cada funcionalidad?

Todas estas preguntas son las que ayudan a responder los problemas de contexto de las variables, estados y métodos.

6. Importa lo necesario y nombrar adecuadamente los componentes

El performance y mantenibilidad del código dependen de las decisiones que se van tomando de desarrollo desde que se empieza a diseñar el código. **Piensa lo mínimo que necesitas para que el componente desarrolle su función e importa solo lo que uses.**

Para el nombramiento ten en cuenta: camelCase cuando nombres variables, objetos o funciones (ej: “const footerItems=[ ]”), PascalCase cuando nombres constructores o clases (ej: “class FooterHome extends React.Component{ }”).

7. Revisa y soluciona constantemente los warnings de la consola

Al correr tu proyecto local React muestra en consola, en tiempo de ejecución, advertencias o warnings mientras desarrollas tu código. Dichas advertencias no impiden que la aplicación siga ejecutándose, porque aún no son errores, pero son reflejo de que tu código es propenso a futuros errores.

Si no pasan desapercibidas para el desarrollador y constantemente se da solución a las advertencias, **tendrás un código más limpio, profesional, con mejor performance y más mantenible.**

En el desarrollo de software hay muchas formas de solucionar problemas, pero solo unas pocas abarcan la solución de la necesidad en contexto y sobrepasan las barreras del tiempo para que sigan creciendo en el futuro. **Crecerás como desarrollador todo el tiempo si nunca dejas de perseguir la mejor manera de hacer las cosas.**