REACT

Máximo D. Chazarreta

EESTN5 Amancio Williams

7mo 4ta / Grupo A

Martín Estanga

05 / 08 / 2024

CONSIGNA

- Buscar diferencias entre react-html, react-css, etc, y su estructura. Luego armar un listado de buenas normas a la hora de realizar programación en React.

RESPUESTA

Diferencias entre React y HTML

- **Propósito**: JSX facilita escribir código que se parece al HTML, pero en realidad es JavaScript. Esto hace que el código sea más intuitivo para desarrollar interfaces de usuario, aunque tiene algunas diferencias y limitaciones en comparación con el HTML estándar.
- **Sintaxis específica**: JSX tiene una sintaxis única. Por ejemplo, utiliza className en lugar de class, y es compatible con expresiones JavaScript, lo que lo hace más flexible pero puede requerir ajustes para quienes están acostumbrados a HTML puro.
- **Compatibilidad**: JSX se convierte a JavaScript nativo antes de ejecutarse, lo que permite que el mismo código se use tanto en la web como en aplicaciones móviles con React Native. Este comportamiento hace que JSX sea menos "estático" que el HTML estándar.

Diferencias entre React y CSS

- **Modularidad**: Los estilos en React pueden ser modulados para cada componente, mientras que en HTML estándar los estilos CSS suelen aplicarse globalmente. Esto permite definir estilos específicos para cada componente y evitar conflictos de nombres, lo cual es ideal para aplicaciones complejas.
- **Opciones de implementación**: Existen varios métodos para manejar el CSS, como los CSS Modules y las Styled Components, que proporcionan una capa adicional de organización y control, a diferencia del enfoque tradicional de hojas de estilo globales.
- **Condicionalidad**: En React, los estilos se pueden aplicar de manera condicional mediante JavaScript, lo cual es más complicado en HTML puro. Esto permite cambiar

estilos dinámicamente según el estado del componente, brindando una experiencia de usuario más interactiva.

- **Uso de JS para los estilos**: Las Styled Components y librerías similares permiten escribir CSS dentro del archivo del componente como JavaScript, facilitando la gestión de estilos y lógica en un solo archivo.

Diferencias entre React y JavaScript

- **Componentes basados en estado y ciclo de vida**: React utiliza JavaScript para crear componentes interactivos que gestionan su propio estado. A diferencia del HTML y CSS estándar, JavaScript en React permite que los componentes "recuerden" valores y ejecuten lógica compleja.
- **Hooks**: React introduce hooks como useState y useEffect, que simplifican la lógica del ciclo de vida y la gestión del estado de los componentes. Esto es una ventaja sobre el HTML estándar, que requiere scripts externos para lograr funcionalidades avanzadas.
- **Encapsulación y reutilización de lógica**: Al trabajar con componentes, React permite reutilizar la lógica de la aplicación, ya que el JavaScript está encapsulado en cada componente. Esto no solo optimiza la estructura del proyecto, sino que también facilita la creación de elementos reutilizables.

Programación en React: Normas a seguir

1. **Organización de Componentes**
    - Divide la interfaz en componentes pequeños y reutilizables con una única responsabilidad.

- Usa carpetas separadas para cada componente, con archivos .js/.jsx y .css correspondientes.

2. **Estructura y Formato del Código**
    - Usa JSX para la estructura del HTML y sigue una sintaxis consistente (linter/prettier).
    - Simplifica condicionales y evita ternarios complejos en el JSX.
3. **Manejo del Estado**
    - Mantén el estado local dentro de los componentes que lo usen, y usa useReducer para estados complejos.
    - Usa estado global solo cuando sea necesario.
4. **Hooks y Ciclo de Vida**
    - Usa useEffect solo con las dependencias necesarias y crea hooks personalizados para lógica repetida.
    - Declara los hooks siempre en el mismo orden.
5. **Props y Comunicación entre Componentes**
    - Define propTypes o usa TypeScript para validar props.
    - Evita el paso excesivo de props; usa Context API o Redux cuando sea necesario.
6. **Optimización y Rendimiento**
    - Usa React.memo para evitar renders innecesarios y useCallBack/useMemo para funciones complejas.
    - Aplica lazy loading (React.lazy) para componentes pesados.
7. **CSS y Estilos**
    - Usa CSS Modules o Styled Components para encapsular estilos.

- Evita estilos en línea complejos; usa una estructura de clases descriptiva.

8. **Manejo de Errores**
    - Agrega Error Boundaries para manejar errores en el renderizado.
    - Válida y sanitiza datos antes de procesarlos.
9. **Pruebas**
    - Realiza pruebas unitarias y de integración con Jest y React Testing Library.
    - Usa snapshots para detectar cambios inesperados en la UI.
10. **Documentación y Comentarios**
    - Documenta componentes complejos y mantén la documentación actualizada.
    - Agrega comentarios solo para clarificar lógica difícil de entender.