REGIONAL ANTIOQUIA
CENTRO DE SERVICIOS Y GESTION EMPRESARIAL

TECNOLOGIA EN ANALISIS Y DESARROLLO DE SOFTWARE

Buenas Prácticas (React Js**)**

En React, además de escribir los atributos en **camelCase** y los nombres de los componentes en **PascalCase**, existen otros estándares y buenas prácticas que ayudan a mantener el código limpio, predecible y funcional. Aquí te comparto algunos de los más importantes:

1. Los nombres de componentes deben ser en PascalCase

- Los componentes de React deben comenzar con mayúscula y seguir el estilo PascalCase, ya que los componentes en minúsculas se interpretan como elementos HTML nativos (por ejemplo,).
- Ejemplo: MyComponent, UserProfile.

2. Los atributos deben estar en camelCase

- Todos los atributos deben escribirse en camelCase, incluyendo los que tienen nombres especiales en HTML.
- Ejemplos:
  - className en lugar de class.
  - onClick en lugar de onclick.
  - maxLength en lugar de maxlength.

3. El componente debe retornar un solo elemento raíz

- Cada componente debe retornar un solo elemento. Puedes usar un elemento envolvente o un **Fragment** (<> … </>) si quieres evitar crear nodos HTML adicionales.
- Ejemplo:

```
return (
  <>
    <Header />
    <Content />
  </>
);
```

Documento elaborado por: Alvaro Pérez Niño - Instructor

4. Usa prop-types para verificar los tipos de las props (o TypeScript)

- Aunque no es obligatorio, se recomienda utilizar **PropTypes** para verificar el tipo y la estructura de las propiedades que el componente recibe, o directamente usar TypeScript.
- Ejemplo con PropTypes:

```
import PropTypes from 'prop-types';

function MyComponent({ name, age }) {
  return <div>{name} is {age} years old</div>;
}

MyComponent.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired,
};
```

5. No manipules directamente el DOM (usa referencias y el estado)

- En lugar de manipular el DOM directamente, React recomienda usar **refs** para interacciones con elementos específicos y **state** para manejar el estado de la interfaz.
- Evita código como document.getElementById() en favor de useRef y useState.

6. Usa funciones puras para componentes y renderizados

- Los componentes deben ser funciones puras, lo que significa que su salida solo depende de las props y el estado, y no de efectos secundarios. Esto hace que los componentes sean más predecibles y fáciles de probar.

7. Utiliza Hooks de React correctamente

- Los Hooks (como useState, useEffect, etc.) deben seguir algunas reglas:
  - Solo se pueden llamar en la parte superior de un componente funcional.
  - No deben usarse dentro de bucles, condicionales o funciones anidadas.

- React también recomienda no abusar de useEffect para lógica que podría resolverse sin efectos.

8. Usa claves únicas (key) en listas renderizadas

- Cada elemento en una lista debe tener una key única y estable para ayudar a React a identificar los elementos y mejorar el rendimiento en el DOM virtual.
- Ejemplo:

```
{items.map((item) => (
  <ItemComponent key={item.id} item={item} />
))}
```

9. Evita el estado global excesivo

- Mantén el estado local siempre que sea posible y usa herramientas como Context o bibliotecas de estado global (Redux, Zustand, Jotai) solo cuando el estado deba compartirse entre componentes que no son padres/hijos directos.

10. Mantén componentes pequeños y reutilizables

- Si un componente empieza a crecer demasiado o a realizar varias funciones, considera dividirlo en componentes más pequeños, siguiendo el principio de responsabilidad única (SRP).