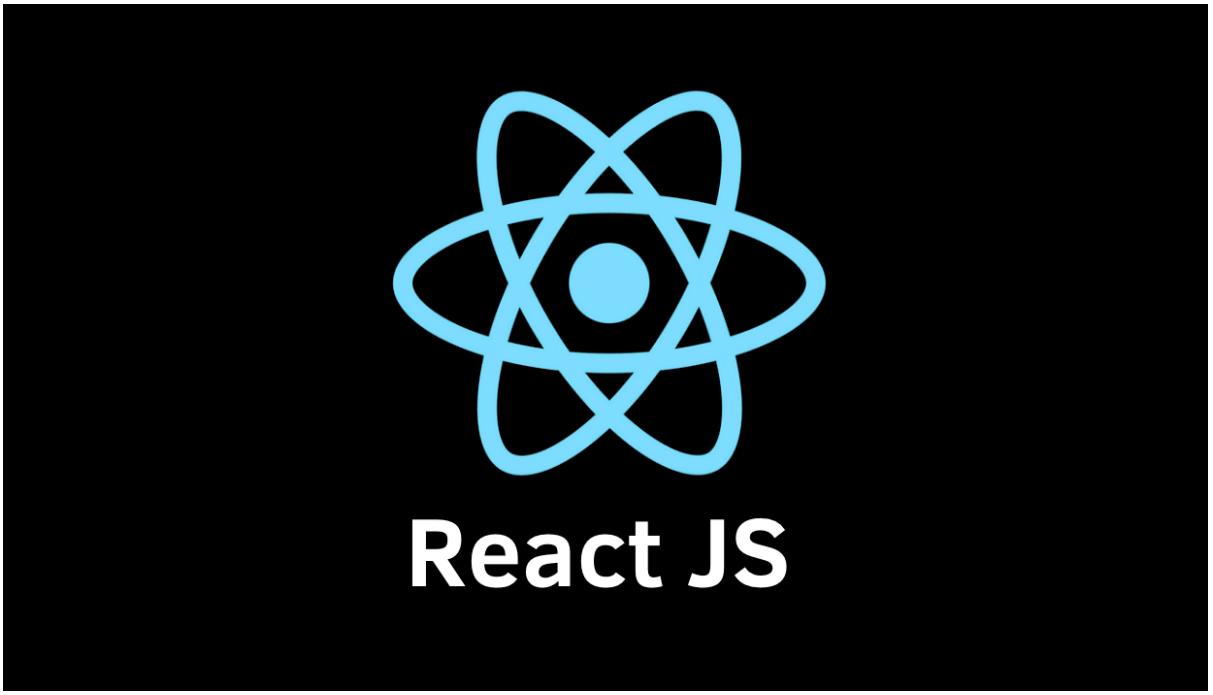
Desarrollo web en entorno cliente - Nivel 7 React

Pablo Valero López *23/01/2026*

2ºDAW



Índice:

Objetivo

Aplicar estilos en React con 4 enfoques: inline styles, archivos CSS, CSS Modules y Tailwind CSS. Para esta práctica probarás los 4 métodos, pero en proyectos reales, evita mezclar estilos sin criterio. Elige un método principal por proyecto para reducir la confusión y duda técnica.

Parte A: Estilo en línea (inline styles)

En la siguiente imagen se muestra el código de añadir estilos como objetos.

Imagen 1: Código de estilo en línea. Elaboración propia.

En la siguiente imagen se muestra el resultado de estilo en línea.

Imagen 2: Resultado de estilo en línea. Elaboración propia.

Parte B: Archivos CSS (global o por componente)

En las siguientes imágenes se muestra el código de App.jsx y de style.css.

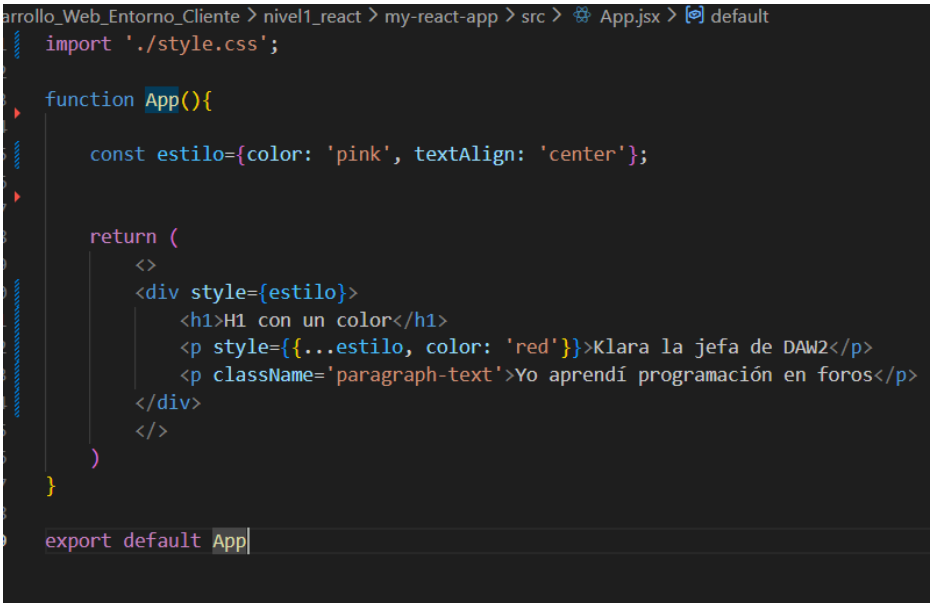


Imagen 3: Código de App.jsx. Elaboración propia

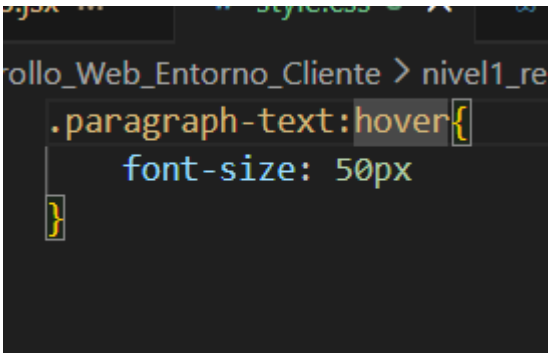


Imagen 4: Código CSS. Elaboración propia

En la siguiente imagen se muestra el resultado de añadir css a la App.

Imagen 5: Resultado CSS. Elaboración propia.

Imagen 6: Resultado CSS. Elaboración propia.

Parte C: CSS Modules (alcance local por defecto)

En las siguientes imágenes se muestra el código de App.jsx y de App.module.css.

```jsx
import './style.css';
import styles from'./App.module.css';

function App(){

    const estilo={color: 'pink', textAlign: 'center'};


    return (
        <>
        <div style={estilo}>
            <h1>H1 con un color</h1>
            <p style={{...estilo, color: 'red'}}>Klara la jefa de DAW2</p>
            <p className={`${styles.textoBase} ${styles.textoModificado}`}>Yo aprendí programación en foros</p>
        </div>
        </>
    )
}

export default App
```

Imagen 7: Código de App. Elaboración propia.

```css
.textoBase{
    color: blueviolet;
}
.textoModificado:hover{
    color: red
}
```

Imagen 8: Código CSS. Elaboración propia.

En las siguientes imágenes se muestra el resultado.

Imagen 9: Resultado CSS. Elaboración propia.

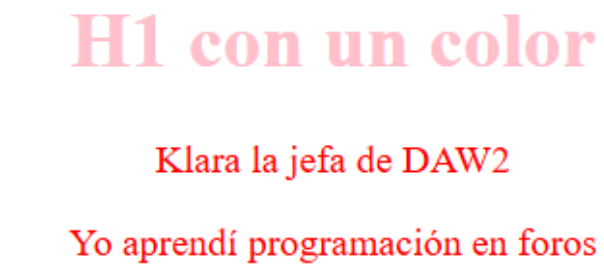


Imagen 10: Resultado CSS. Elaboración propia.

Parte D: Tailwind CSS (utilidades)

En la siguiente imagen se muestra el código de vite.config.js.

Imagen 11: Código de Tailwind CSS. Elaboración propia.

En la siguiente imagen se muestra el código de App.

Imagen 12: Código de Tailwind CSS. Elaboración propia.

En la siguiente imagen se muestra el código de Tailwind CSS

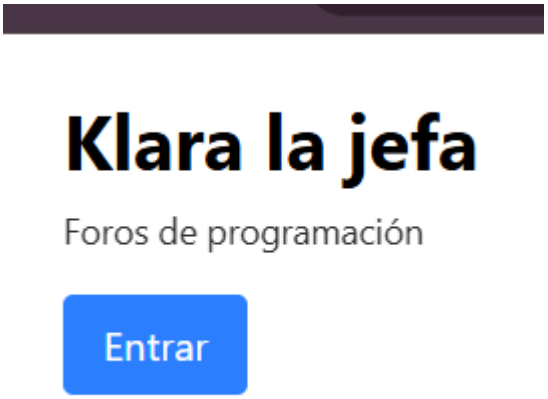


Imagen 13: Resultado de Tailwind CSS. Elaboración propia.

Miniretillo (seguimos sin ayuda)

Para este miniretillo he reutilizado el código anterior, añadiendo el 'paragraph-text' utilizado en ejercicios anteriores y añadiendo en el botón para conseguir el cambio de tamaño al pasar el cursor, también añado la clase centrar para lograr que el texto se centre lo máximo posible. Para conseguir el responsive es únicamente de bootstrap.

En la siguiente imagen se muestra el código de App.

```
import './style.css';

function App() {
return (
<div className='centrar'>
<h1 className='text-3xl font-bold'>Klara la jefa</h1>
<p className='mt-2 text-sm opacity-80'>
Foros de programación
</p>
<button className='mt-4 px-4 py-2 text-white bg-blue-500 rounded paragraph-text'>
Entrar
</button>
</div>
);
}

export default App
```

Imagen 14: Código de App. Elaboración propia.

En la siguiente imagen se muestra el código Css.

```css
.paragraph-text:hover{
    font-size: 100px
}
.centrar{
    text-align: center;
    margin-top: 15%;
}
```

Imagen 15: Código de CSS. Elaboración propia.

En las siguientes imágenes se muestra el resultado del miniretillo

Imagen 16: Resultado de Miniretillo. Elaboración propia.

Imagen 17: Resultado de Miniretillo. Elaboración propia.

Las siguientes preguntas deberían haberse resuelto en el ejercicio anterior. Un fallo de compresión provocó no realizarlas. Las dejo resultas en este ejercicio ya que son las mismas.

Preguntas

1. ¿Qué diferencia hay entre onClick y onSubmit?

La diferencia entre **onClick y onSubmit** son: **onClick** detecta cuando haces clic en un elemento específico. **onSubmit** detecta cuando un formulario intenta enviarse.

2. ¿Por qué usamos e.preventDefault() en un formulario?

Se usa para evitar que la página se **recargue** al enviar el formulario, lo cual es el comportamiento **por defecto** de los navegadores. Esto **permite** que React procese los datos con **JavaScript** sin **perder el estado** ni **reiniciar la aplicación**.

3. ¿Qué es una "entrada controlada" y por qué usamos value + onChange?

Una **entrada controlada** es un input cuyo contenido es gobernado totalmente por el Estado de React en lugar del navegador. Usamos **onChange** para actualizar ese estado cada vez que el usuario escribe, asegurando que lo que ves en pantalla y la variable en código sean siempre **idénticos**.

4. En tu mini-reto, que estado(s) manejas y que evento(s) los actualizan?

Los estados que se manejan son visibilidad, contador y texto, controladas mediante eventos **onClick** y actualizando el estado mediante **onChange.**