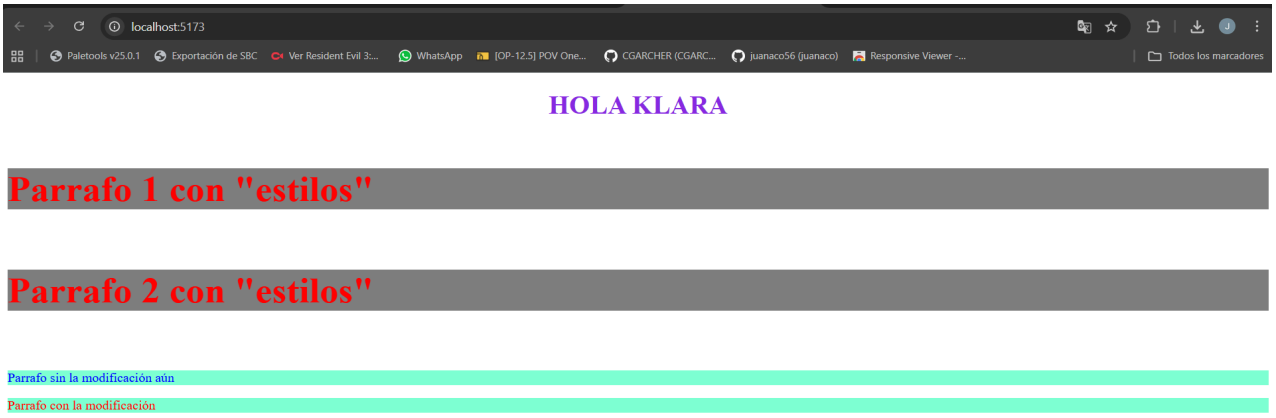
Juan Antonio Zamora Nieto | 2º DAW | 22/01/2026

NIVEL 7 REACT

PARTE A: Estilo en línea (inline styles):

```jsx
src > App.tsx > App > sobreescrito > sinSobreescribir
function App() {
  let estilos = {
    color: 'red',
    fontSize: '45px',
    backgroundColor: 'grey'
  }

  let sinSobreescribir = {
    color: 'blue',
    backgroundColor: 'aquamarine',
    textAllign: 'left'
  }

  let sobreescrito = {
    ...sinSobreescribir,
    color: 'red'
  }
  return (
    <>
    <h1 style =
      {{color:'blueviolet',
        textAlign: 'center'
      }}>
        HOLA KLARA
      </h1>

      <h4 style={estilos}>
        Parrafo 1 con "estilos"
      </h4>
      <h5 style={estilos}>
        Parrafo 2 con "estilos"
      </h5>


      <p style={sinSobreescribir}>
          Parrafo sin la modificación aún
      </p>
      <p style={sobreescrito}>
          Parrafo con la modificación
      </p>



    </>
  )
}

export default App;
```

(Ilustración 1. Captura de pantalla de código creando un "`h1`" como objeto, dándole un propiedad simple y otra en "`camelCase`", extraemos un objeto de estilo a una constante y lo reutilizamos en los dos párrafos "`p`", uno de ellos sobreescrita la propiedad del color. Elaboración propia.)

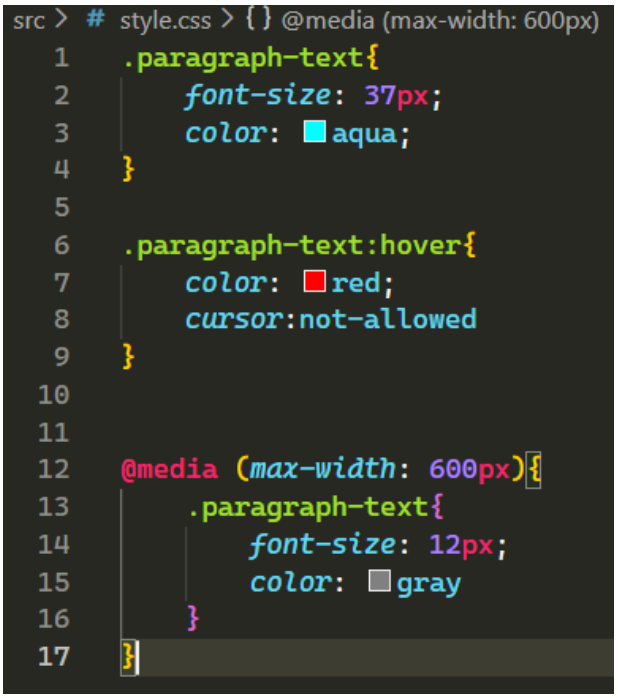


(Ilustración 2. Captura de pantalla de la App mostrando como el "`h1`" está centrado con un color, como creamos una propiedad de estilos y se la damos a los párrafos 1 y 2, y como en los últimos dos párrafos sobreescribimos la propiedad del color en el último. Elaboración propia.)

PARTE B: Archivos CSS (global o por componente):

(Ilustración 3. Captura de pantalla del código, donde nombramos a un

(Ilustración 3. Captura de pantalla del código, donde nombramos a un "`p`" con un "`className`" que hemos definido en nuestro CSS, dándole un color y un tamaño de letra. Elaboración propia.)



(Ilustración 4. Captura de pantalla del código CSS, donde observamos las propiedades de "`paragraph-text`", la pseudo clase "`:hover`", y el media query. Elaboración propia.)

(Ilustración 5. Captura de pantalla de la App, en modo “Responsive”, haciendo que el media query se active, y vemos sus cualidades personalizadas (color gris, letra pequeña). Elaboración propia)

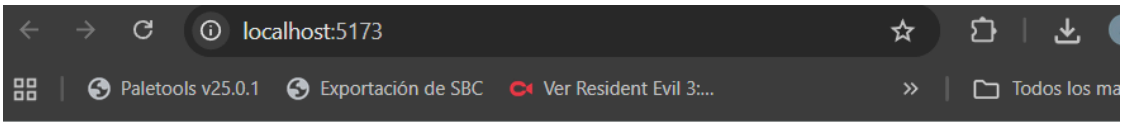
(Ilustración 6. Captura de pantalla de la App en modo normal, capturando su color azul definido normalmente. Elaboración propia.)

(Ilustración 6. Captura de pantalla de la App cuando activamos “`:hover`”(pasar el ratón por encima), y cambia al color rojo. Elaboración propia.)

PARTE C: CSS Modules (alcance local por defecto):

```css
.baseTexto{
    font-size: 45px;
}

.primerTexto {
    color: red;
}

.segundoTexto{
    color: green;
}
```

(Ilustración 7. Captura de pantalla del código CSS, donde se crean dos clases y una base para el tamaño de la fuente. Elaboración propia.)



Texto primario con tamaño base

Texto secundario con tamaño base

(Ilustración 8. Captura de pantalla de la App, donde observamos cada párrafo con unos estilos diferentes, pero con la misma base en cuanto a tamaño de letra. Elaboración propia.)

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

function App() {
  return (
    <>
      <p className={`${styles.baseTexto} ${styles.primerTexto}`}>
        Texto primario con tamaño base
      </p>

      <p className={`${styles.baseTexto} ${styles.segundoTexto}`}>
        Texto secundario con tamaño base
      </p>
    </>
  );
}

export default App;
```

(Ilustración 9. Captura de pantalla del código donde importamos los estilos de nuestro archivo, y combinando la clase base, con la clase de cada uno, logramos que tengan la misma base pero diferente color, con "`template string`". Elaboración propia.)

PARTE D: Tailwind CSS (utilidades):

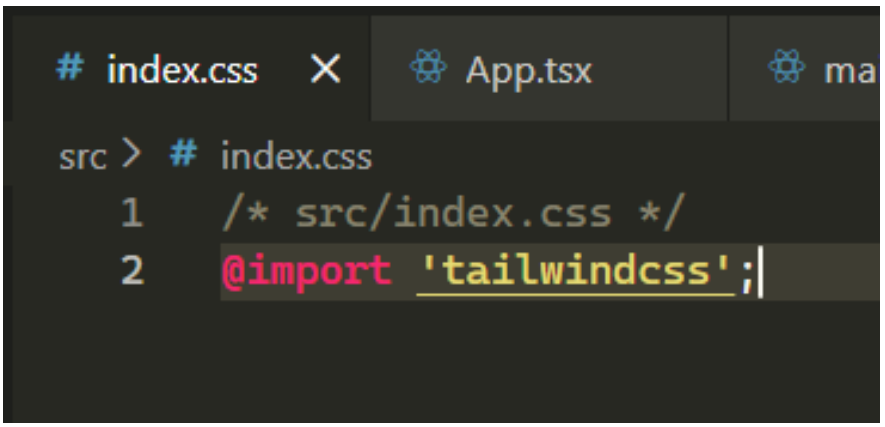
src > main.tsx

```tsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import './index.css'


ReactDOM.createRoot(document.getElementById('root')
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

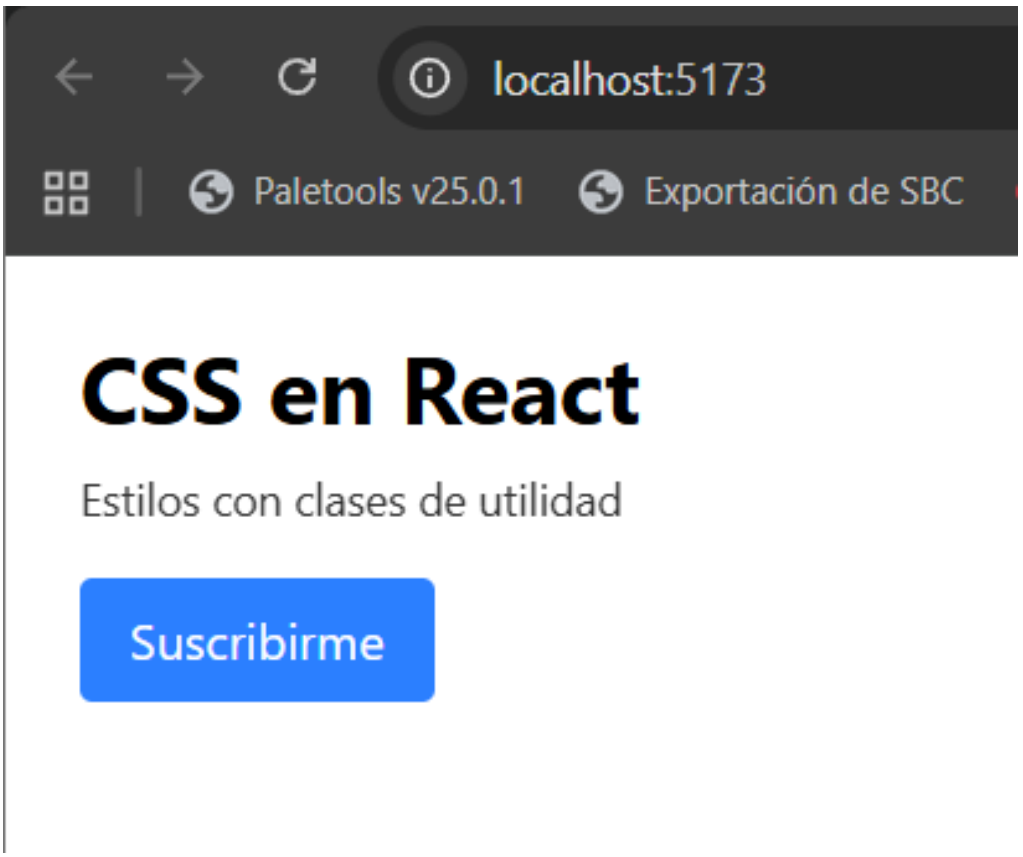
(Ilustración 10. Captura de pantalla de "`main.tsx`", donde importamos nuestro archivo "`index.css`". Elaboración propia.)

(Ilustración 11. Captura de pantalla del código en "`vite.config.ts`", donde adjuntamos el plugin de Tailwind junto al plugin de React. Elaboración propia.)

(Ilustración 12. Captura de pantalla del código "`App.tsx`" en el cual hacemos uso de Tailwind en un componente. Elaboración propia.)



(Ilustración 13. Captura de pantalla del código "`index.css`", donde importamos "tailwindcss" para poder usarlo en nuestra "`App.tsx`". Elaboración propia.)

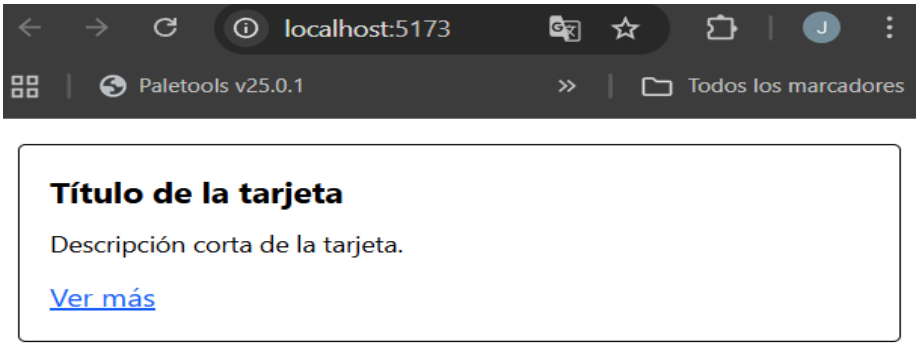


(Ilustración 14. Captura de pantalla de la App, donde se ve el uso de Tailwind es nuestro proyecto, después de descargarlo e importarlo en nuestro CSS y en nuestro "`main.tsx`" con su plugin. Elaboración propia.)

MINIRETILLO:

```jsx
function App() {
  return (
    <div className="p-4">
      <div className="border p-4 rounded">
        <h2 className="text-lg font-bold">
          Título de la tarjeta
        </h2>

        <p className="text-sm mt-2">
          Descripción corta de la tarjeta.
        </p>

        <button className="mt-3 text-blue-600 hover:underline">
          Ver más
        </button>
      </div>
    </div>
  );
}

export default App;
```

(Ilustración 15. Captura de pantalla del código con título, descripción corta y un botón. Un hover para el botón subrayado, y uso de modo responsive para pantallas pequeñas. Elaboración propia.)



(Ilustración 16. Captura de pantalla de la App donde observamos el "`:hover`", el título, la descripción corta, el botón "Ver más" y el modo responsive para pantallas pequeñas. Elaboración propia.)