

NIVEL 7 REACT

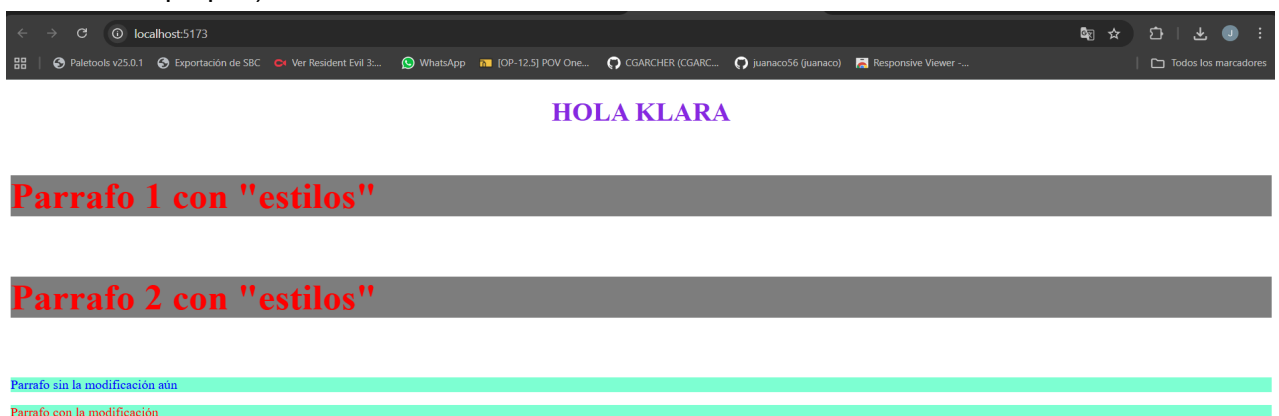


PARTE A: Estilo en línea (inline styles):.....	2
PARTE B: Archivos CSS (global o por componente):....	3
PARTE C: CSS Modules (alcance local por defecto):.....	5
PARTE D: Tailwind CSS (utilidades):.....	6
MINIRETILLO:.....	8

PARTE A: Estilo en línea (inline styles):

```
src > App.tsx > App > sobrescrito > sinSobrescribir
1  function App() {
2    let estilos = {
3      color: 'red',
4      fontSize: '45px',
5      backgroundColor: 'grey'
6    }
7
8    let sinSobrescribir = {
9      color: 'blue',
10     backgroundColor: 'aquamarine',
11     textAlign: 'left'
12   }
13
14   let sobrescrito = {
15     ...sinSobrescribir,
16     color: 'red'
17   }
18   return (
19     <>
20     <h1 style =
21       {{color:'blueviolet',
22        textAlign: 'center'
23       }}>
24       HOLA KLARA
25     </h1>
26
27     <h4 style={estilos}>
28       Parrafo 1 con "estilos"
29     </h4>
30     <h5 style={estilos}>
31       Parrafo 2 con "estilos"
32     </h5>
33
34     <p style={sinSobrescribir}>
35       Parrafo sin la modificación aún
36     </p>
37     <p style={sobrescrito}>
38       Parrafo con la modificación
39     </p>
40
41
42
43
44
45   </>
46   )
47 }
48
49 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 sobrescrita 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 sobrescribimos 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

```
src > App.tsx > default
1  import './style.css';
2
3  function App() {
4    return (
5      <>
6        <h1>CSS externo en React</h1>
7
8
9        <p className="paragraph-text">
10         Este párrafo usa una clase de CSS externa
11       </p>
12     </>
13   );
14 }
15
16 export default App
```

(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.)

```
src > # style.css > {} @media (max-width: 600px)
1  .paragraph-text{
2    font-size: 37px;
3    color: aqua;
4  }
5
6  .paragraph-text:hover{
7    color: red;
8    cursor: not-allowed;
9  }
10
11
12  @media (max-width: 600px){
13    .paragraph-text{
14      font-size: 12px;
15      color: gray;
16    }
17  }
```

(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.)

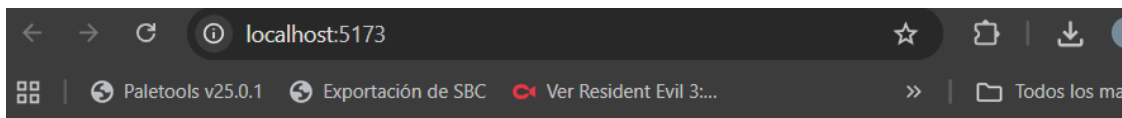


(Ilustración 6. Captura de pantalla de la App cuando activamos “:hover”(pasar el ratón por encima), y cambia al color rojo. Elaboración propia.)

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

```
.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.)

```

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.)

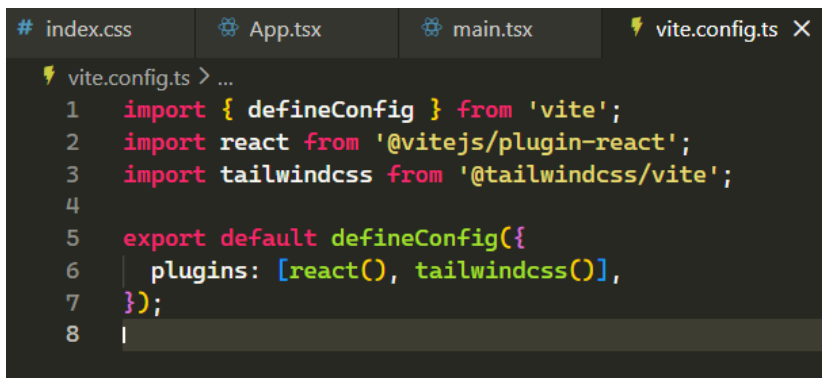
PARTE D: Tailwind CSS (utilidades):

```

src > main.tsx
1  import React from 'react'
2  import ReactDOM from 'react-dom/client'
3  import App from './App.tsx'
4  import './index.css'
5
6
7  ReactDOM.createRoot(document.getElementById('root'))
8    <React.StrictMode>
9      <App />
10   </React.StrictMode>,
11 )
12

```

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

A screenshot of a code editor showing the vite.config.ts file. The editor has tabs for index.css, App.tsx, main.tsx, and vite.config.ts. The vite.config.ts tab is active, showing the following code:

```
1 import { defineConfig } from 'vite';
2 import react from '@vitejs/plugin-react';
3 import tailwindcss from '@tailwindcss/vite';
4
5 export default defineConfig({
6   plugins: [react(), tailwindcss()],
7 });
8
```

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

A screenshot of a code editor showing the App.tsx file. The editor has tabs for App.tsx and vite.config.ts. The App.tsx tab is active, showing the following code:

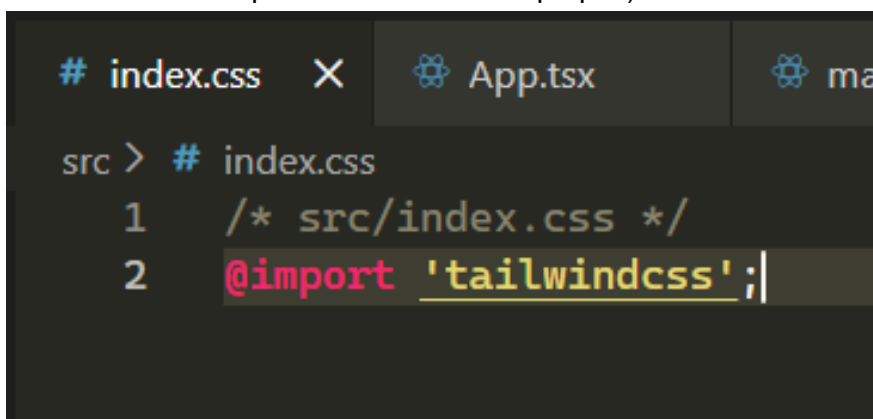
```
function App() {
  return (
    <div className="p-6">
      <h1 className="text-3xl font-bold">
        CSS en React
      </h1>

      <p className="mt-2 text-sm opacity-80">
        Estilos con clases de utilidad
      </p>

      <button className="mt-4 px-4 py-2 text-white bg-blue-500 rounded">
        Suscribirse
      </button>
    </div>
  );
}

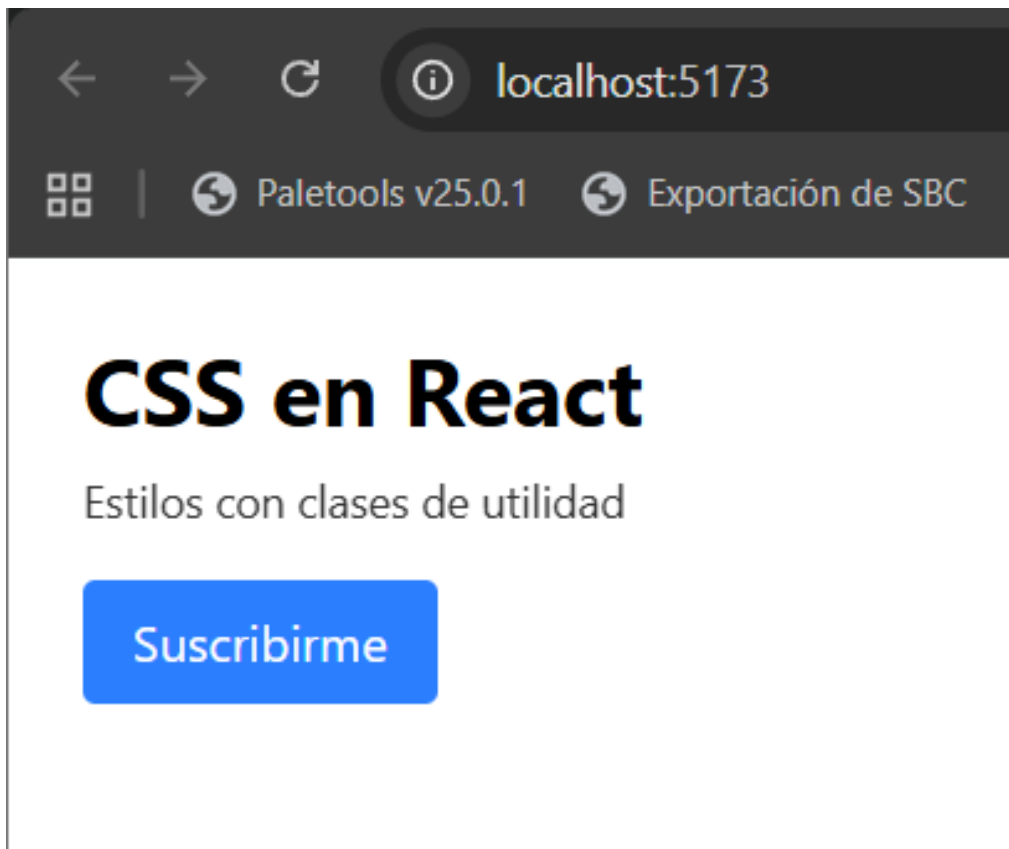
export default App;
```

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

A screenshot of a code editor showing the index.css file. The editor has tabs for index.css, App.tsx, and main.tsx. The index.css tab is active, showing the following code:

```
src > # index.css
1 /* src/index.css */
2 @import 'tailwindcss';
```

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

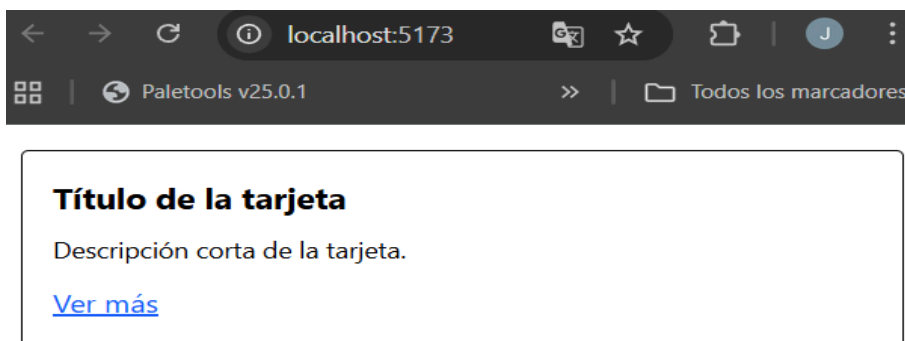


(Ilustración 14. Captura de pantalla de la App, donde se ve el uso de Tailwind es nuestro proyecto, después de descargarlo e importarlo en nuestro CSS y en nuestro “`main.tsx`” con su plugin. Elaboración propia.)

MINIRETILLO:

```
1 function App() {
2   return (
3     <div className="p-4">
4       <div className="border p-4 rounded">
5         <h2 className="text-lg font-bold">
6           Título de la tarjeta
7         </h2>
8
9         <p className="text-sm mt-2">
10          Descripción corta de la tarjeta.
11        </p>
12
13        <button className="mt-3 text-blue-600 hover:underline">
14          Ver más
15        </button>
16      </div>
17    </div>
18  );
19 }
20
21 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.)