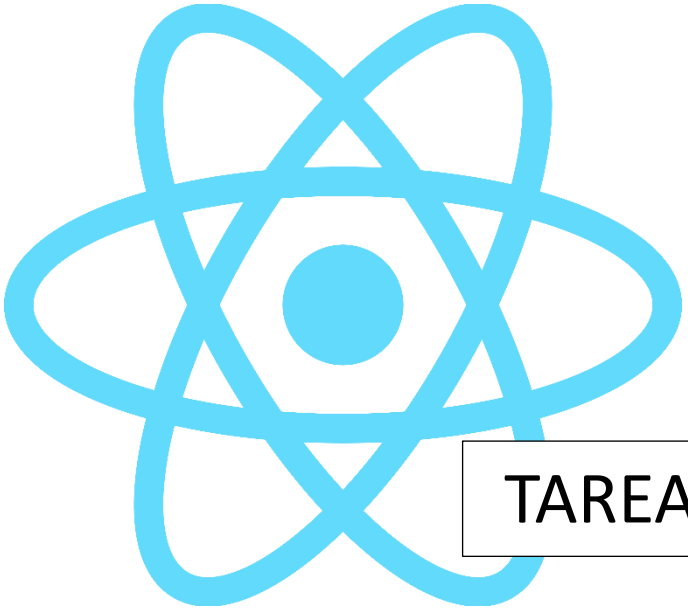

TAREA 7: REACT

Contenido

Parte A: Estilo en línea (inline styles)

```jsx
function App() {
  // 3. Extraemos un objeto de estilo a una constante
  const estiloBase = {
    backgroundColor: 'lightgray', // camelCase
    padding: '15px',              // propiedad simple
    borderRadius: '10px'          // camelCase
  };

  // 4. Usamos el operador spread (...) para extender y sobrescribir
  const estiloCambiado = {
    ...estiloBase,                // Traemos todo lo de estiloBase
    backgroundColor: 'skyblue'    // Sobrescribimos solo el color de fondo
  };

  return (
    <div>
      {/* 1 y 2. h1 con style como objeto, propiedad simple y camelCase */}
      <h1 style={{ color: 'darkblue', textAlign: 'center' }}>
        Mi Título con Estilos
      </h1>

      {/* 3. Reutilizamos la constante en dos párrafos */}
      <p style={estiloBase}>
        Este es el párrafo 1 con el estilo base.
      </p>

      <p style={estiloBase}>
        Este es el párrafo 2 con el mismo estilo base.
      </p>

      {/* 4. Usamos el estilo extendido con spread */}
      <p style={estiloCambiado}>
        Este párrafo es igual al anterior pero con fondo azul gracias al spread.
      </p>
    </div>
  );
}

export default App;
```

Ilustración 1. Estilos en linea. Elaboración propia

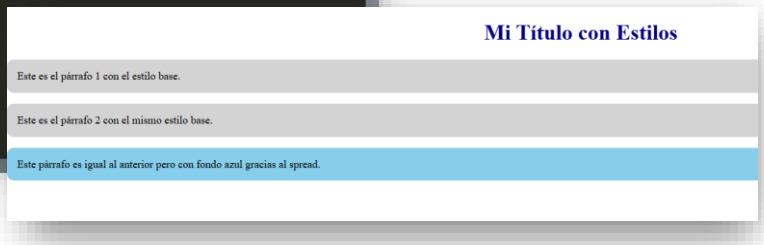

Ilustración 2. Visualización del resultado. Elaboración propia

Parte B: Archivos CSS (global o por componente)

App.jsx

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

function App() {
  return (
    <div>
      <h1>Estilos con CSS Externo</h1>

      {/* 3. Aplicar la clase con className */}
      <h2 className="paragraph-text">
        Hola a todos, me llamo className.
      </h2>

      <p className="paragraph-text">
        Yo también me llamo className
      </p>
    </div>
  );
}
export default App;
```

Ilustración 3. App.jsx. Elaboración propia

style.jsx

```jsx
.paragraph-text {
  font-size: 20px;
  color: red;
}

.paragraph-text:hover {
  color: blue;
}

@media (max-width: 600px) {
  .paragraph-text {
    color: green;
  }
}
```

Ilustración 4. style.css. Elaboración propia

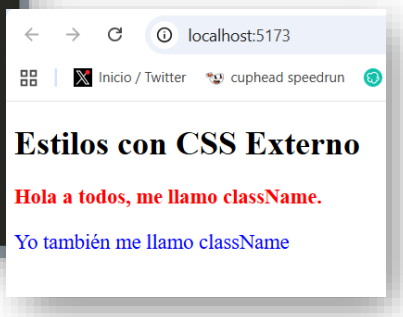

Ilustración 5. Visualización del resultado. Elaboración propia

Parte C: CSS Modules (alcance local por defecto)

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

function App() {
  return (
    <div>
      <h1>CSS Modules</h1>

      {/* 3. Aplicar una sola clase del objeto styles */}
      <p className={styles.azul}>
        Este texto es azul usando styles.azul.
      </p>

      {/* 4. Combinar clases usando Template String (comillas invertidas) */}
      {/* Combinamos la fuenteBase con el color rojo */}
      <p className={`${styles.fuenteBase} ${styles.rojo}`}>
        Este texto es grande y rojo (clases combinadas).
      </p>
    </div>
  );
}

export default App;
```

Ilustración 6. App.jsx. Elaboración propia

```jsx
.fuenteBase {
  font-size: 24px;
}

.azul {
  color: blue;
}

.rojo {
  color: red;
}
```

Ilustración 7. App.module.css. Elaboración propia

Ilustración 8. Visualización del resultado. Elaboración propia

Parte D: Tailwind CSS (utilidades)

```
//Comandos

# Terminal
```

npm install tailwindcss @tailwindcss/vite

Vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss()],

})
```

Ilustración 9. Plugin Tailwind vite.config.js. Elaboración propia

App.jsx

```
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'>
        Suscribirme
      </button>
    </div>
  );
}

export default App;
```

Ilustración 10. App.jsx. Elaboración propia

Index.css

```
@import "tailwindcss";

.text-lowercase {
  text-transform: lowercase;
  color: #4a90e2;
  font-family: sans-serif;
}

.Web-Designer {
  color: #007bff;
  font-weight: bold;
}

.boss {
  color: #e00404;
}
```

Ilustración 11. Importar Tailwind.
Elaboración propia

Ilustración 12. Visualización del resultado. Elaboración propia

Miniretillo (seguimos sin ayuda)

```jsx
function App() {
  return (
    // items-start evita que la tarjeta se estire hacia abajo
    <div className="min-h-screen bg-gray-100 p-6 flex justify-center items-start">

      {/* h-fit hace que la tarjeta solo mida lo que mide su contenido */}
      <div className="bg-white p-6 rounded-xl shadow-md max-w-sm h-fit mt-10">

        {/* Título: Responsive (text-xl en móvil, md:text-2xl en escritorio) */}
        <h2 className="font-bold text-xl md:text-2xl text-gray-800">
          2º DAW
        </h2>

        {/* Descripción corta: Responsive (mt-2 en móvil, md:mt-4 en escritorio) */}
        <p className="text-gray-600 mt-2 md:mt-4 text-sm md:text-base">
          Esta es una descripción corta para probar los estilos y el responsive simple.
        </p>

        {/* Botón con Hover: subrayado, sombra y cambio de color */}
        <button className="mt-6 px-4 py-2 bg-blue-600 text-white rounded-lg transition-all
          hover:shadow-lg hover:underline hover:bg-blue-700">
          Click aquí
        </button>

      </div>
    </div>
  );
}

export default App;
```

Ilustración 13. App.jsx. Elaboración propia

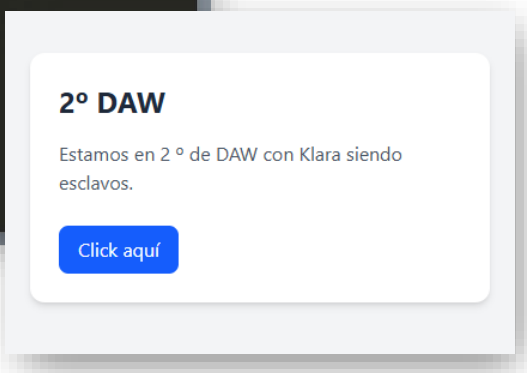

Ilustración 14. Visualización del resultado. Elaboración propia