Instituto Pío IX Casa Salesiana

Desarrollo de Aplicaciones Informáticas

Tema: Estilos con `clsx` aplicados en un proyecto NextJS

Docente: Ing. Pablo Morandi
Ayudante: Matias Marchesi
Curso: 5to año
Especilidad: Informática

---

1. Introducción

En este apunte vamos a aprender cómo aplicar estilos en proyectos con **NextJS**, tanto en componentes como en páginas. Para ello usaremos archivos `.module.css`, que nos permiten definir estilos de forma ordenada y sin conflictos.

Además, incorporaremos la librería **clsx**, que nos da la posibilidad de activar o desactivar estilos según condiciones lógicas, logrando que la interfaz sea más dinámica. De esta forma, veremos cómo un mismo componente puede adaptarse a distintos estados, como mostrar un botón en verde al incrementar y en rojo al decrementar.

El objetivo es que comprendan cómo combinar **estilos y lógica** para construir interfaces más claras, flexibles y pensadas para el usuario.

---

2. Instalación de clsx

Primero instalamos la dependencia en nuestro proyecto:

```
npm i clsx
```

Con eso ya podemos empezar a trabajar.

---

3. Estilo al componente Button

3.1 Archivo Button.module.css

En la misma carpeta del componente creamos un archivo:

`Button.module.css`

Aquí dentro podemos definir clases como en diseño web:

```
.button {
  border-radius: 10px;
  padding: 8px 16px;
}

.incrementar {
  background-color: green;
}

.decrementar {
  background-color: red;
}

.redondeado {
  border-radius: 50px;
}
```

3.2 Aplicar clase al botón

En el componente importamos y aplicamos la clase:

```
import styles from "./Button.module.css";

<button className={styles.button}>Click</button>;
```

> Importante: en React usamos `className` en lugar de `class`.

3.3 Agregar más estilos

Podemos combinar varias clases:

```
<button className={`${styles.button} ${styles.incrementar}`}>+</button>
```

Esto hará que el botón sea verde y a la vez redondeado.

---

4. Lógica de estilos con clsx

4.1 Importar clsx

```
import clsx from "clsx";
```

4.2 Aplicar estilos condicionales

```
<button
  className={clsx(styles.button, {
    [styles.incrementar]: true,
  })}
>
  +
</button>
```

- `true`: el estilo se aplica.
- `false`: el estilo se desactiva.

4.3 Uso de props para condicionar estilos

Podemos usar una propiedad `crece` para definir si el botón debe ser verde (incrementar) o rojo (decrementar):

```
<button
  className={clsx(styles.button, {
    [styles.incrementar]: crece,
    [styles.decrementar]: !crece,
  })}
>
  Click
</button>
```

4.4 Ejemplo de botones con estilos distintos

```
<Button crece={true} />   // Verde
<Button crece={false} />  // Rojo
```

4.5 Agregar estilos adicionales (redondeado)

Podemos pasar otra prop, por ejemplo `redondo`:

```
<button
  className={clsx(styles.button, {
    [styles.redondeado]: redondo,
  })}
>
  Botón
</button>
```

---

5. Estilo a una página

5.1 Archivo page.module.css

En las páginas de NextJS siempre el archivo se llama:

`page.module.css`

Ejemplo:

```
.cambioFuente {
  font-size: 36px;
  font-family: Arial, sans-serif;
}
```

5.2 Aplicar estilos al h1 y otros elementos

```
import styles from "./page.module.css";

<h1 className={styles.cambioFuente}>Registro</h1>;
```

Esto hará que el título de la página se vea con un tamaño y fuente diferentes.

---

6. Conclusión

El uso de `.module.css` nos permite organizar y aplicar estilos de manera aislada en componentes y páginas de NextJS. Con la librería **clsx**, podemos sumar lógica a esos estilos, activándolos o desactivándolos según las props o el estado de la aplicación.

De esta forma, logramos que nuestros componentes sean más reutilizables, dinámicos y adaptables a las necesidades de cada proyecto.