Instituto Pío IX Casa Salesiana

Desarrollo de Aplicaciones Informáticas

Tema: Creación de componentes básicos en `NextJS`

Docente: Ing. Pablo Morandi
Ayudante: Matías Marchesi
Curso: 5to año
Especialidad: Informática

Redactado por: Ing. Pablo Morandi
Versión: v1.0

Índice

---

1. Introducción a Componentes

Los componentes son **piezas reutilizables** de nuestra interfaz. En lugar de repetir código, creamos un componente y lo usamos en múltiples lugares.

¿Qué es un Componente?

Un componente es como un **molde** que podemos usar para crear elementos similares:

```
// En lugar de escribir esto en cada página:
<h1 style={{color: 'blue', fontSize: '24px'}}>Mi Título</h1>

// Creamos un componente y lo reutilizamos:
<Title />
```

Regla Fundamental de React

TODO EN REACT (sea un componente, una página, o el contenido de un conditional rendering) **DEVUELVE SIEMPRE UNA SOLA ETIQUETA HTML**. Esta etiqueta puede contener múltiples elementos dentro, pero siempre debe haber un contenedor principal.

```
// ✅ Correcto - Una sola etiqueta contenedora
function MiComponente() {
    return (
        <div>
            <h1>Título</h1>
            <p>Párrafo</p>
            <button>Botón</button>
        </div>
    );
}

// ❌ Incorrecto - Múltiples etiquetas en el nivel superior
function MiComponente() {
    return (
        <h1>Título</h1>
        <p>Párrafo</p>
        <button>Botón</button>
    );
}
```

---

2. Convenciones de Nomenclatura

PascalCase para Componentes

Los componentes de React **SIEMPRE** se escriben en **PascalCase**:

```
// ✅ Correcto
function Title() { }
function UserProfile() { }
function NavigationMenu() { }

// ❌ Incorrecto
function title() { }
function userProfile() { }
function navigation_menu() { }
```

¿Por qué React Exige PascalCase?

Razón técnica: React diferencia entre **elementos HTML** y **componentes personalizados** por la primera letra:

```
// Minúscula = elemento HTML nativo
<div>Esto es un div HTML</div>
<button>Esto es un botón HTML</button>
<input type="text" />

// PascalCase = componente personalizado
<Title>Esto es nuestro componente Title</Title>
<UserCard>Esto es nuestro componente UserCard</UserCard>
```

Ventajas de los Componentes

Sin componentes (código repetido):

```
// En LoginPage
<button onClick={handleLogin} style={{...}}>Iniciar Sesión</button>

// En HomePage
<button onClick={handleLogout} style={{...}}>Cerrar Sesión</button>

// En ProfilePage
<button onClick={handleSave} style={{...}}>Guardar</button>
```

Con componentes (código reutilizable):

```
// En todas las páginas
<Button onClick={handleLogin}>Iniciar Sesión</Button>
<Button onClick={handleLogout}>Cerrar Sesión</Button>
<Button onClick={handleSave}>Guardar</Button>

// Si necesitamos cambiar el estilo, solo modificamos Button.js
```

---

3. Creación de Rutas en NextJS

NextJS maneja las rutas en base al **sistema de archivos** que tengamos en el proyecto, específicamente a través del nombre de las carpetas.

Crear una Nueva Ruta

Para crear una página nueva:

1. **Crear una carpeta** dentro de `src/app` con el nombre de la ruta deseada
2. **Crear un archivo** `page.js` dentro de esa carpeta

```
src/
  app/
    ranking/          ← Nueva ruta
      page.js         ← Página que responde a /ranking
```

Ejemplo de Página

```jsx
// src/app/ranking/page.js
export default function RankingPage() {
    return (
        <div>
            <h1>Página de Ranking</h1>
            <p>Esta página responde a la ruta /ranking</p>
        </div>
    );
}
```

---

4. Layouts en NextJS

Los layouts permiten crear **contenido compartido** entre páginas y subrutas. Son ideales para menús, headers, footers, etc.

Crear un Layout

Todos los layouts se llaman `layout.js` y reciben la propiedad `children`:

```
// src/app/ranking/layout.js
export default function RankingLayout({ children }) {
    return (
        <div>
            <header>
                <h1>Header del Ranking</h1>
                <nav>
                    <a href="/ranking/general">General</a>
                    <a href="/ranking/mensual">Mensual</a>
                </nav>
            </header>

            <main>
                {children} {/* Aquí se renderiza page.js */}
            </main>

            <footer>
                <p>Footer del Ranking</p>
            </footer>
        </div>
    );
}
```

Alternativa con Props

También puedes recibir `children` a través del objeto `props`:

```
// src/app/ranking/layout.js
export default function RankingLayout(props) {
    return (
        <div>
            <header>
                <h1>Header del Ranking</h1>
            </header>

            <main>
                {props.children}
            </main>
        </div>
    );
}
```

5. Creación de Componentes

Los componentes se crean en una carpeta `components` dentro de `src`.

Estructura de Carpetas

```
src/
  components/
    Button.js          ← Componente Button
    Title.js           ← Componente Title
  app/
    page.js
```

Ejemplo: Componente Button Básico

```
// src/components/Button.js
export default function Button() {
    return (
        <button>
            Click me!
        </button>
    );
}
```

Componente con Funcionalidad

```
// src/components/Button.js
"use client" // ← Necesario para eventos

export default function Button() {
    const handleClick = () => {
        console.log("¡Botón presionado!");
    };

    return (
        <button onClick={handleClick}>
            Presionar
        </button>
    );
}
```

Nota sobre eventos: En React, los eventos se escriben en **camelCase** (onClick, onChange, onKeyUp, etc.), no como en HTML tradicional.

6. Uso de Componentes en Páginas

Importar y Usar Componentes

```
// src/app/ranking/page.js
import Button from '@/components/Button';

export default function RankingPage() {
    return (
        <div>
            <h1>Página de Ranking</h1>
            <Button />  {/* ← Usar el componente */}
        </div>
    );
}
```

Tip: Visual Studio Code te ayudará a importar automáticamente los componentes cuando comiences a escribirlos.

---

7. Propiedades (Props) en Componentes

Las **props** permiten personalizar el comportamiento y contenido de los componentes.

Props para Funciones

```
// src/components/Button.js
"use client"

export default function Button(props) {
    return (
        <button onClick={props.onClick}>
            {props.children}
        </button>
    );
}
```

Usando el componente con props:

```
// src/app/ranking/page.js
import Button from '@/components/Button';

export default function RankingPage() {
    const mostrarAlerta = () => {
        alert("¡Hola desde el primer botón!");
    };

    const mostrarConsola = () => {
        console.log("¡Hola desde el segundo botón!");
    };

    return (
        <div>
            <h1>Página de Ranking</h1>

            <Button onClick={mostrarAlerta}>
                Mostrar Alerta
            </Button>

            <Button onClick={mostrarConsola}>
                Mostrar en Consola
            </Button>
        </div>
    );
}
```

Props con Desestructuración

Una forma más limpia de trabajar con props:

```
// src/components/Button.js
"use client"

export default function Button({ onClick, children }) {
    return (
        <button onClick={onClick}>
            {children}
        </button>
    );
}
```

Props Personalizadas

Puedes crear tus propias props:

```
// src/components/Button.js
"use client"

export default function Button({ text, onClick }) {
    return (
        <button onClick={onClick}>
            {text}
        </button>
    );
}
```

Usando props personalizadas:

```
// Forma larga (con contenido)
<Button onClick={miFuncion}>Texto del botón</Button>

// Forma corta (con prop personalizada)
<Button text="Texto del botón" onClick={miFuncion} />
```

La Propiedad Children

`children` es una prop especial que **siempre existe** y contiene el contenido entre las etiquetas de apertura y cierre:

```
<Button>
    Este texto es {children}
</Button>

<Button>
    <span>Este elemento también es {children}</span>
</Button>
```

---

8. Server-Side vs Client-Side Components

Renderizado por Defecto

Por defecto, NextJS renderiza los componentes en el **servidor** (Server-Side Rendering).

¿Cuándo usar "use client"?

Necesitas agregar `"use client"` al inicio del archivo cuando:

- El componente tiene **eventos** (onClick, onChange, etc.)
- Usa **hooks** de React (useState, useEffect, etc.)
- Necesita **interacción con el usuario**

```
// ☑ Componente que necesita "use client"
"use client"

import { useState } from 'react';

export default function Button() {
    const [count, setCount] = useState(0);

    const handleClick = () => {
        setCount(count + 1);
    };

    return (
        <button onClick={handleClick}>
            Clicks: {count}
        </button>
    );
}
```

```
// ☑ Componente que NO necesita "use client"
export default function Title() {
    return (
        <h1>
            Sistema de Gestión Escolar
        </h1>
    );
}
```
