

## Índice

- Acerca de
- Qué es React Context
- Cuándo usar y cuándo no usar
- API de React Context
- Cover del proyecto
  - Styled components
  - Organización y estructura de los componentes
  - css
  - style custom components
  - Usar props y defaults
  - SpanStyled
  - Override h1 size
- Artículos
  - Mocking de datos
  - Mostrado en la interfaz
- Theming con context
  - Crear el contexto con los dos temas de colores
  - Generar el provider
  - Consumir el contexto
  - Modificar el contexto
- Context como manejador de estado
  - Adaptar el provider
  - Consumir el mock desde el contexto
  - Consumir con fetch
  - Toggle de username para devto

## Acerca de la clase.

La clase de hoy sirve como un repaso de lo que hemos visto hasta ahora, principalmente:

- Uso de state y props
- styled components
- Despliegue de datos
- fetch


Pero sobre todo es la clase clave para hablar de React Context, su definición, y su uso práctico en un ejemplo didáctico pero muy cercano al uso real de Context.

Es por esto que comenzaremos la clase haciendo un repaso sobre Context, qué es y cómo se define.

## React Context

Context es una API de React que permite comunicar información entre componentes sin recorrer cada nivel del árbol de nuestros componentes.

Considera que tienes una página para la que deseas que el usuario pueda personalizar y cambiar el fondo. La estructura de tu página puede verse así:



```
1 <Layout>↵
2   · <Header>↵
3   · · · <BotonCambiar>Cambiar</BotonCambiar>↵
4   · </Header>↵
5   · <Articles></Articles>↵
6   · <Footer></Footer>↵
7 </Layout>
```

En este caso la acción de cambio de color debe recorrer el Header y el Layout para que se pueda realizar el efecto apropiado. En este ejemplo sencillo sería posible enviar por props un método para cambiar el color:



```
1 <Layout>↵
2 · <Header·setColor={setColor}>↵
3   ···<BotonCambiar·setColor={setColor}>↵
4     ·····Cambiar·color↵
5   ···</BotonCambiar>↵
6 · </Header>↵
7 · <Articles></Articles>↵
8 · <Footer></Footer>↵
9 </Layout>
```

Ahora, qué pasa si también deseas que el color actual sea desplegado en el componente que cambia de color, continuarías enviando props desde un componente superior hasta uno inferior en el árbol:

```
1 <Layout>↵
2 ·<Header color={color} setColor={setColor}>↵
3   ···<BotonCambiar color={color} setColor={setColor}>↵
4     ·····Cambiar color↵
5   ···</BotonCambiar>↵
6 ·</Header>↵
7 ·<Articles></Articles>↵
8 ·<Footer></Footer>↵
9 </Layout>↵
10
```

## Data flow

React está diseñado para que la información fluye en una sola dirección, desde los componentes superiores hacia abajo. Esta arquitectura es una parte clave del diseño de la librería.

Al seguir esta regla, el flujo de información es explícito y más fácil de comprender puesto que podemos asumir que la información de un componente viene del componente padre, y no necesitamos buscar quién o cómo se modificó.

## Prop drilling

Aunque React está diseñado para enviar información desde los componentes superiores hacia abajo, existen escenarios para los cuáles esto puede ser problemático.

De hecho, este problema tiene un nombre no oficial en la comunidad: “Prop drilling” es el problema por el que los componentes reciben props con el único propósito de servir como medio para que esta información llegue a componentes hijos. En este ejemplo el componente Header no hace uso de las props que recibe, solamente necesita enviarlas al componente BotonCambiar.

El problema de prop drilling no está directamente relacionado con Context, ya que no todos los problemas de prop drilling deben ser resueltos con Context. En muchos casos usar composición es mejor. Veamos más sobre cuándo sí y cuándo no usar Context

## Cuándo usar Context

Context es un manejador de estado por lo que su uso debe asociarse con almacenar la información que debe ser compartida por muchos componentes, algunos ejemplos son:

- Autenticación de usuario
- Theming
- Manejo de distintos idiomas
- Preferencias de moneda, zona horaria
- Entre otros

Si estás trabajando con información que afecta a distintos componentes, usar Context puede ayudarte a compartir la información entre todos los componentes y que todos reciban la actualización apropiada.

Este ejemplo es más enfático cuando trabajamos con un router ya que, a través de Context podemos preservar información entre páginas incluso aunque reemplacemos por completo toda la interfaz.

Context no es necesariamente para manejar el estado global de tu aplicación, también puedes usarlo para manejar el estado de un grupo de componentes, por ejemplo:

- Un formulario para el que los controles deben compartir información
- Un componente separado en distintos pasos para mostrar

Esto nos recuerda algo muy importante, una aplicación puede y generalmente requiere de más de un contexto.

## Cuándo NO usar Context

El uso de Context tiene un precio a pagar, principalmente al incrementar la complejidad de nuestro código, luego de que normalmente la información terminará por fluir en ambas vías, de los hijos hacia arriba y viceversa, ahora las modificaciones no son tan explícitas y terminaremos por estar revisando quién cambió algún dato.

Adicionalmente, el uso de Context genera dependencia entre nuestros componentes luego de que un componente que usa Context requiere de ser usado en una parte donde el contexto exista. Esto reduce la posibilidad de reutilizar este componente en distintos escenarios, porque ahora tiene una dependencia con Context.

En muchos casos prop drilling puede ser mejor que usar Context, en otros usar composición, trata Context como una adición de complejidad al código y no lo uses para cualquier escenario posible.

En muchas ocasiones la respuesta a: Cómo modifico desde un componente anidado la información de un componente superior es, no lo hagas.

## Aprendamos a usar Context

En este ejercicio aprenderemos a usar Context en un ejemplo muy sencillo cuyo propósito es aprender la sintaxis, después en la práctica veremos cómo usar Context en un escenario más complejo, menos controlado y más realista.

Ejercicio final:

<https://stackblitz.com/edit/react-hycjz8?file=src%2FApp.js>

## Práctica final

Para cerrar la clase pongamos todo lo que hemos aprendido en práctica:

<https://github.com/codigofacilito/taller-practico-react>

Practica en vivo: <https://codigofacilito.github.io/taller-practico-react/>