# Context

DigitalHouse >
Coding School

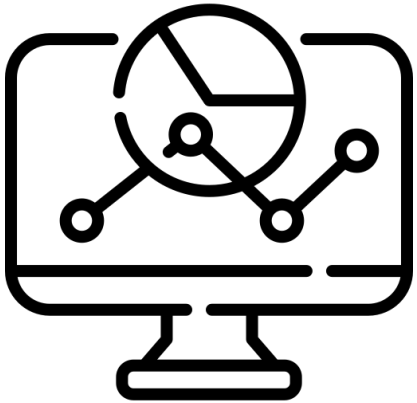
# Índice

# 1 | ¿Qué es context?

# ¿Qué es context?

Ante Props drilling, React liberó en su versión 16.3 una solución nativa: la **API Context**.

Gracias a su diseño, Context nos permite encapsular datos e inyectarlos en componentes incluídos dentro de dicho contexto.

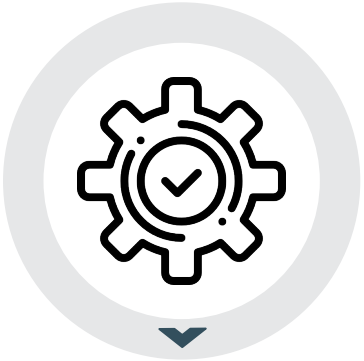


# Contenedor de componentes

Para entender esto podríamos ver a Context como a un contenedor de componentes.

Todo valor definido y actualizado en el contexto podrá ser usado por los componentes incluídos dentro de tal contexto.

En este sentido, si bien Context dota a la aplicación de cierta globalidad en los datos, es una globalidad parcial.



Esto no es malo, por el contrario, es una característica super útil ya que nos permite definir el alcance específico de tales globalidades.

# Gracias Context :)

En síntesis, Context nos evita la maldición de tener que pasar datos a través de árboles de componentes enteros mediante props. Nos devuelve años de vida.

# Veamos la idea en código

```
import React from "react";

function App {
        return (
            <TitleContext.Provider value="Soy un título">
                <Left />
                <Main />
            </TitleContext.Provider>
        );
}
```

De este modo, ambos subárboles de componentes (Left y Main) podrán consumir el contexto TitleContext.

# 2 ¿Para qué suele usarse?

# ¿Para qué suele usarse principalmente Context?

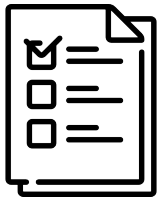
1. **Manejo de *theming*** (implementación de modo oscuro o temas alternativos).
2. **Autenticación de usuarios.**
3. **Internacionalización** (el idioma)

# Pasos para la implementación de Context

Creación mediante el método createContext

Poner cualquier valor en el Provider usando la propiedad value.

Usar un context Provider para encapsular un árbol de componentes.



Leer esta propiedad en cualquier componente encapsulado usando el context Consumer.

# 3 createContext

# createContext

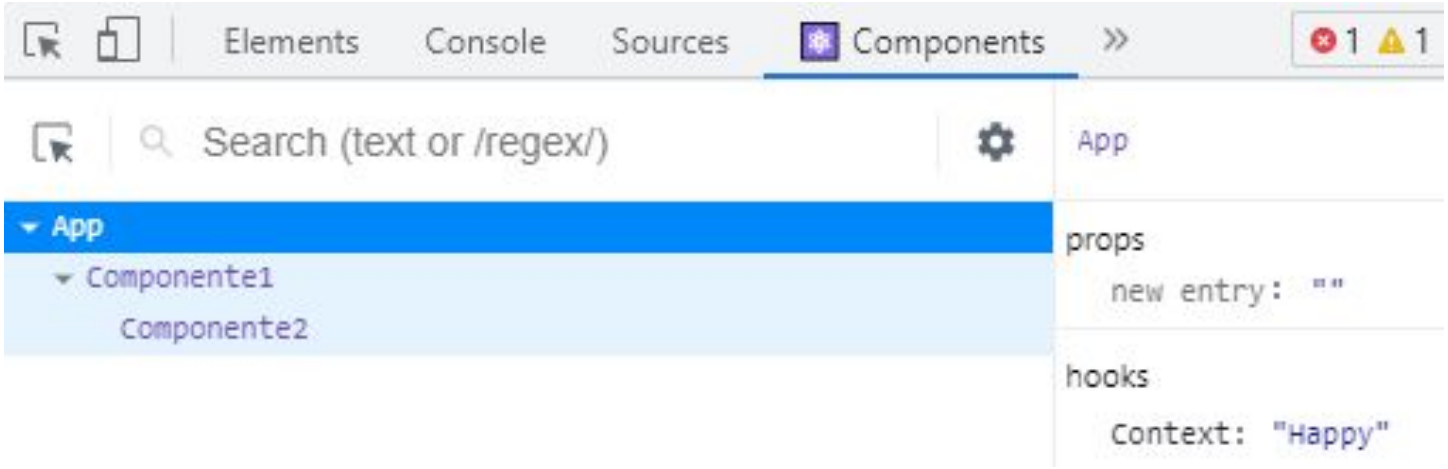
Un objeto Context se crea pasando por defecto un valor al método de React, **createContext**. Si bien createContext acepta un valor inicial, este no es obligatorio.

Como resultado de crear el contexto, obtendremos dos propiedades que, a la vez, se declaran como componentes: **Provider y Consumer.**

# Ejemplo de createContext

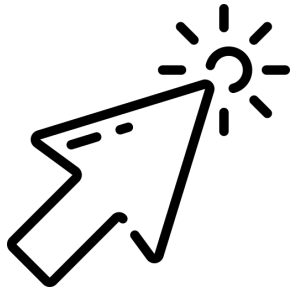
```
import { createContext } from "react";

const peopleMood = "Happy";

export const PeopleContext = createContext(peopleMood);
```

# ¿Qué vemos en React Dev Tools?



# Provider

El contexto debe ser global para los componentes que lo consumen, es por esto por lo que Context nos brinda un **Provider**. Este es un componente que toma una propiedad llamada value, es decir, el valor dado al contexto, y lo inyecta a los componentes englobados por él. Este valor puede ser el que se desee, hasta un objeto con múltiples valores.



# Provider

- No todos los componentes descendientes del contexto estarán suscritos al contexto, salvo que declaren la suscripción al contexto. Esto es debido a que no todos los componentes necesitarán estos datos.
- Cuando un componente suscrito se renderice, leerá el valor actual del contexto provisto por el Provider, por lo que la responsabilidad del Provider será dar acceso a los datos almacenados en el contexto.

```
<PeopleContext.Provider value="mucha">
   <World />
</PeopleContext.Provider>
```

# Consumer

Para que un componente pueda recibir el valor de un contexto, el componente debe estar suscrito al mismo. La suscripción se realiza mediante el componente **Consumer.**

- ➔ Este componente deberá incluir una función como hijo o children.
- ➔ Esta función recibirá el argumento value con el valor actual del contexto, retornará un nodo React (JSX) y estará a la escucha de cambios.
- ➔ Respecto al árbol de componentes, el Consumer estará debajo del Provider. Esta función será llamada en el momento de renderizar.
- ➔ Debemos pensar a Context como un canal para compartir datos.
- ➔ Por cada Provider habrá un Consumer que sólo se relacionarán entre sí.

# Consumer

```
import React, { Component } from "react";
import { ConsumerDeLaVida } from "./ContextDeLaVida";

function App {
      return (
         <ConsumerDeLaVida>
            {(props) => {
               return <div>{props.laVida}</div>;
            }}
         </ConsumerDeLaVida>
      );
}
```

# 4 Ejemplo Práctico

# 1. Crear un componente y un contexto

```
1  import React, {createContext} from 'react';
2
3  const contexto = createContext();
4
5  const ProductContext = () => {
6      return (
7          <contexto.Provider>
8
9          </contexto.Provider>
10     );
11 };
12
13 export default ProductContext;
```

1. Crear un contexto utilizando createContext
2. Crear un componente que retorne los provider del contexto. Esto se hace porque más adelante envolveremos los componentes que queremos que accedan al context y necesitamos permitirles el acceso a los valores que tengamos en el context. De esta manera declaramos que contexto sera el proveedor de esos valores

# 2. Declarar los values del context y agregar children

```jsx
import React, { createContext } from "react";

const contexto = createContext();

const ProductContext = ({ children }) => {

  let productos = [
    { name: "Teclado Mecánico", price: 10000 },
  ];

  return <contexto.Provider value={{productos}}>
      {children}
      </contexto.Provider>;
};

export default ProductContext;
```

1. Recibir children por props.
2. Retornar children envolviéndolo en el provider del context. De esta manera cualquier componente que sea envuelto con este context podrá acceder a los values de su provider. Utilizamos children porque nos permite tener la cantidad de componentes que querramos consumiendo este context sin necesidad de saber exactamente cuáles son. Tal vez en un futuro queremos agregar más componentes o quitar alguno que esté suscrito al context y con children no nos va a generar ningún problema.
3. Declarar value dentro del provider y guardar el array de productos para que esté disponible en todos los componentes que sean envueltos en este contexto.

# 3. Requerir el componente en App.js

```
1  import ProductContext from "./context/productContext";
2  import ListadoProductos from "./components/ListadoProductos";
3
4  function App() {
5    return (
6      <ProductContext>
7        <ListadoProductos />
8      </ProductContext>
9    );
10 }
11
12 export default App;
```

Al envolver el componente ListadoProductos en nuestro context le damos acceso a los values que hayamos declarado. ListadoProductos en este caso es un children de ProductContext

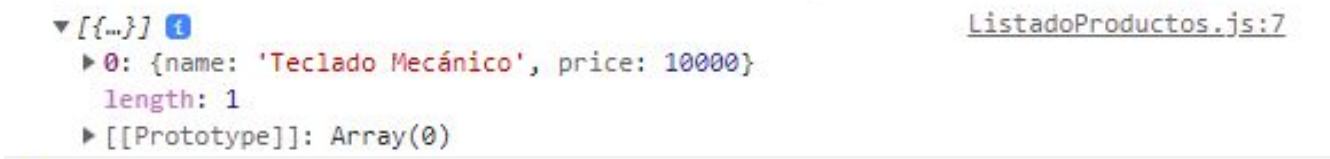
```
export const contexto = createContext();
```

En el archivo productContext exportaremos el contexto creado para poder capturar sus values en donde necesitemos

# 4. Utilizar el context

```jsx
import React, { useContext } from "react";
import { contexto } from "../context/productContext";

const ListadoProductos = () => {
  const { productos } = useContext(contexto);

  console.log(productos);

  return (
    <div>
      <h1>Listado de Productos</h1>
    </div>
  );
};

export default ListadoProductos;

```

1. Importamos el contexto
2. Importamos el hook useContext
3. Utilizamos useContext pasándole como parámetro el contexto
4. Destructuramos el value que querramos usar



# Documentación

**Para saber más podemos acceder a la documentación oficial de React haciendo clic en el siguiente link.**

DigitalHouse>