



Context

Estado global

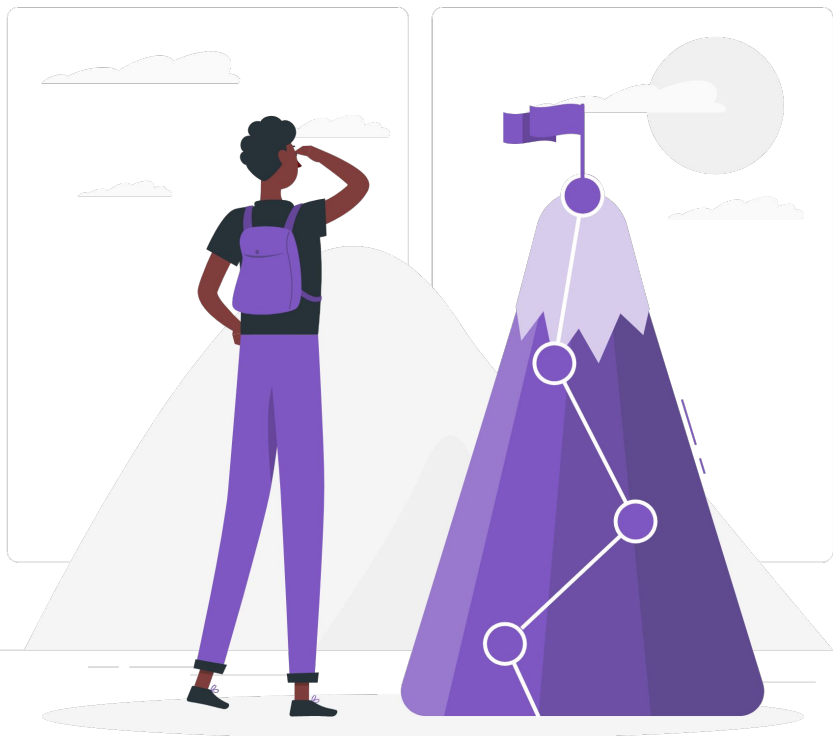
DEV.F.
DESARROLLAMOS(PERSONAS);

Elaborado por: César Guerra

www.cesarquerra.mx



Buy me a coffee



Objetivos

- Entender cómo se comunican los componentes en React.
- Identificar la problemática común de la comunicación entre componentes.
- Entender el concepto y la necesidad de uso de contextos globales.
- Aprender a utilizar el hook de Context en React.

The logo consists of the text 'DEV.F!' in a bold, white, sans-serif font. The 'F' is stylized with three small squares at its top right corner. The logo is centered within a dark blue diamond shape.

DEV.F!

RECORDATORIO:

*¿Cómo pasamos la
información de un
Componente a Otro?*

Recordando: Props

JSX

```
<MyComponent  
  prop="My Value"  
/>
```

Component

```
function MyComponent({ prop }) {  
  return <span>{prop}</span>;  
}
```

Output

```
<span>My Value</span>
```

Flujo de Comunicación entre Componentes

DEV.F
DESARROLLAMOS(PERSONAS);

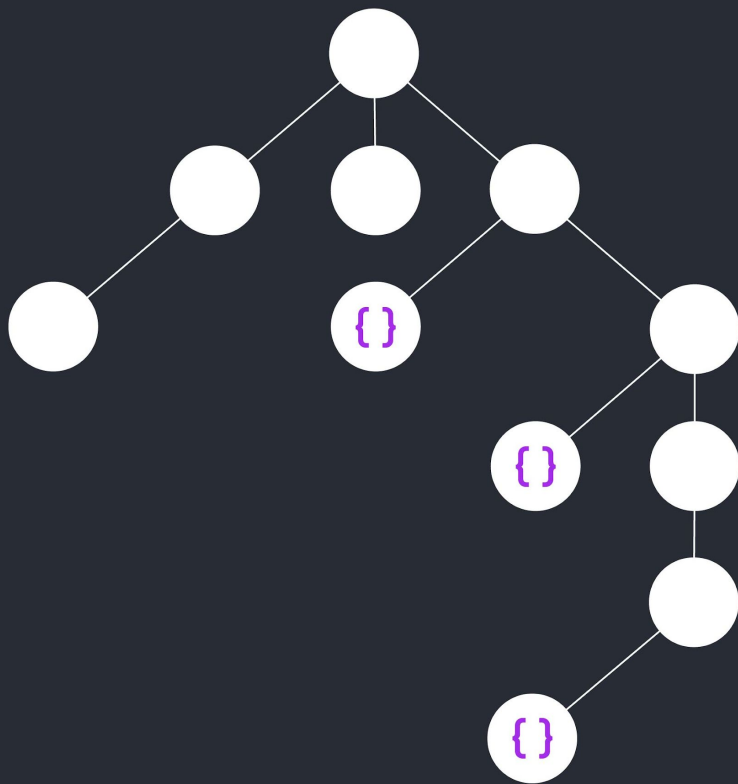
dev

Árbol de Componentes

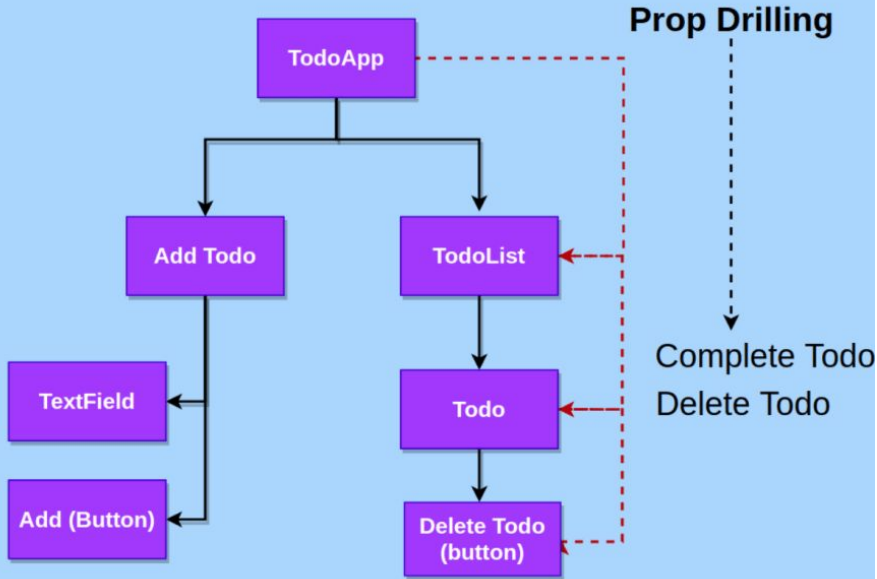
En React todo es un árbol.

El componente con mayor poder jerárquico comúnmente maneja la lógica y comparte por props a los demás componentes hijos, esto permite tener:

- Componentes con lógica
- Componente visuales.



How to handle Prop Drilling?

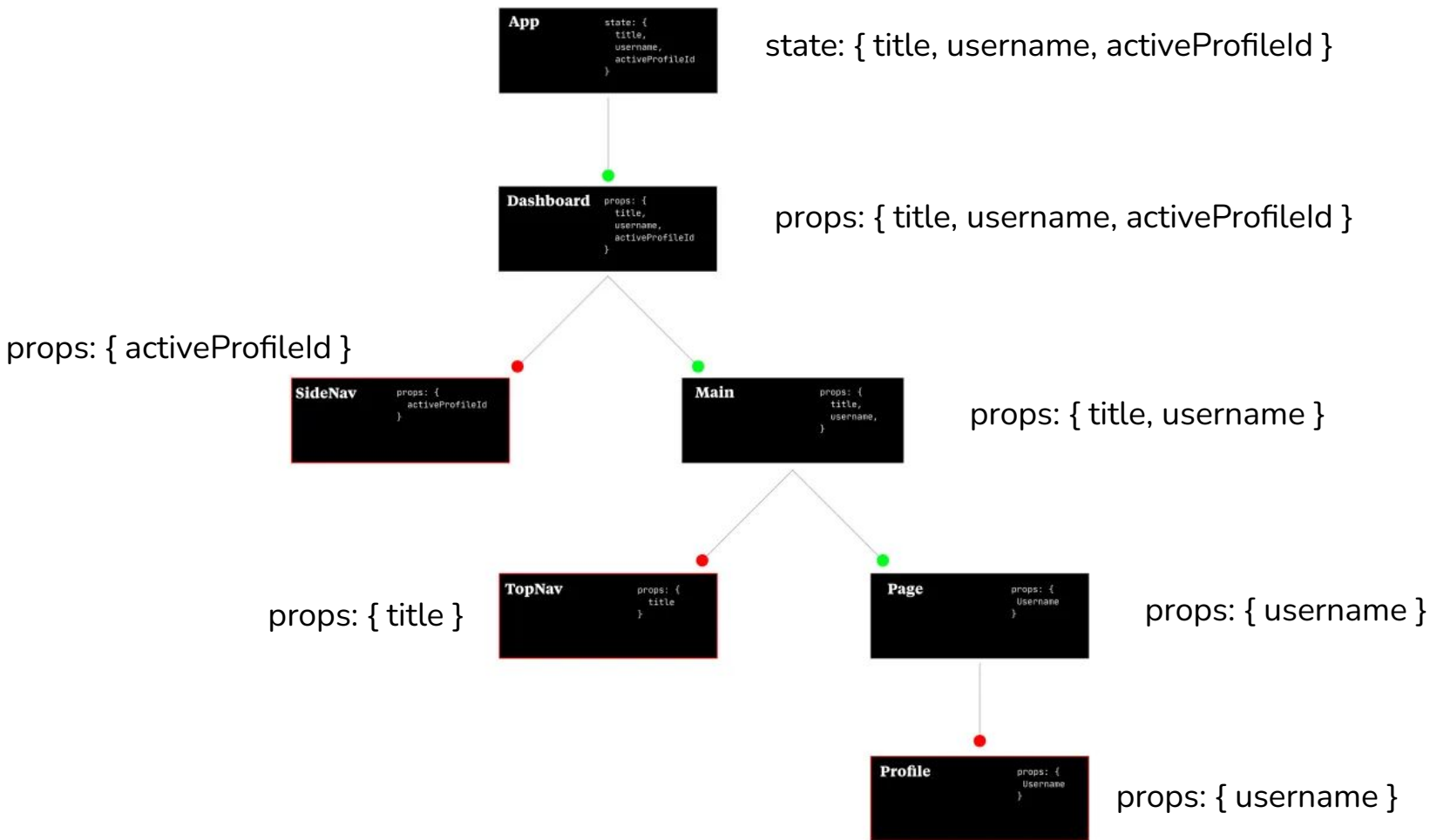


Problema: Prop Drilling

Prop Drilling es el término no oficial usado para decir que pasaremos datos a través de varios componentes hijos anidados, en un intento de entregar estos datos a un componente profundamente anidado.

El problema con este enfoque es que la mayoría de los componentes a través de los cuales se pasan estos datos no tienen ninguna necesidad real de estos datos. Simplemente se utilizan como medios para transportar estos datos a su componente de destino.

Esto puede causar importantes problemas de reutilización de componentes y de rendimiento de la aplicación.



The logo consists of the text 'DEV.F.' in a bold, white, sans-serif font. The 'F' is stylized with a grid of small squares at its top right corner.

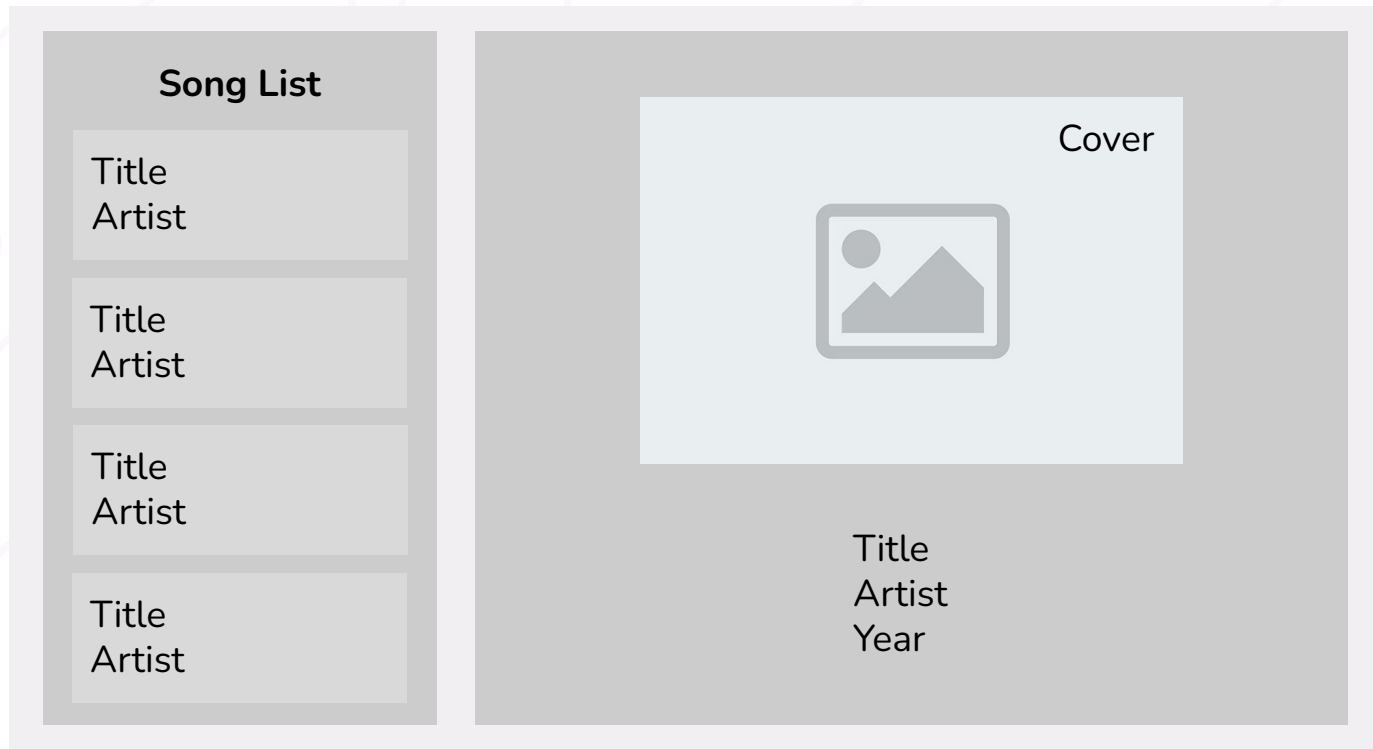
DEV.F.

*DISCUSIÓN DE
ARQUITECTURA*

Arquitectura de Reproductor Musical

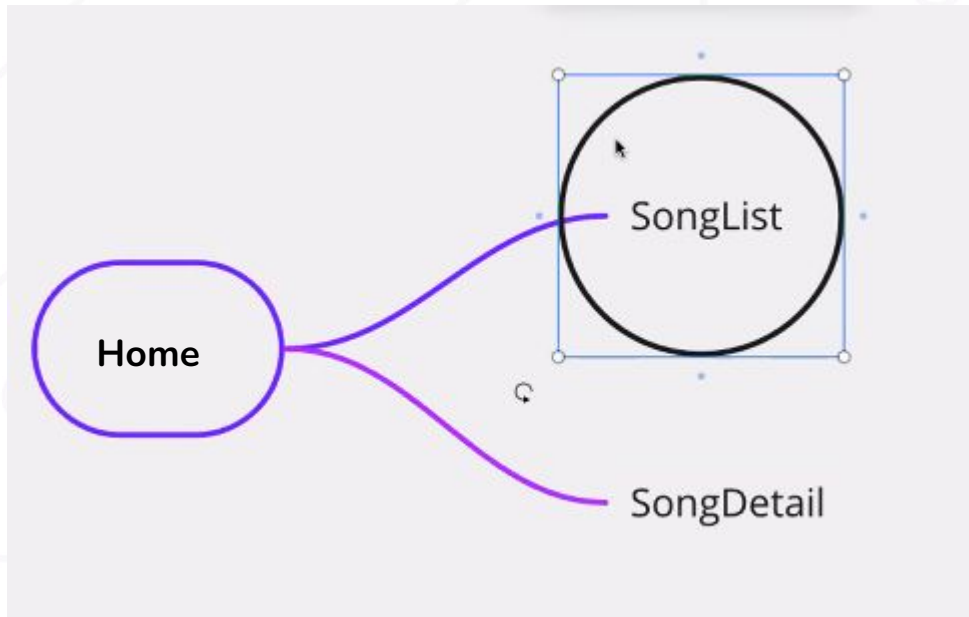
¿Qué componentes identificas?

Al hacer clic en una canción, se actualizará la información en la parte derecha de la aplicación.



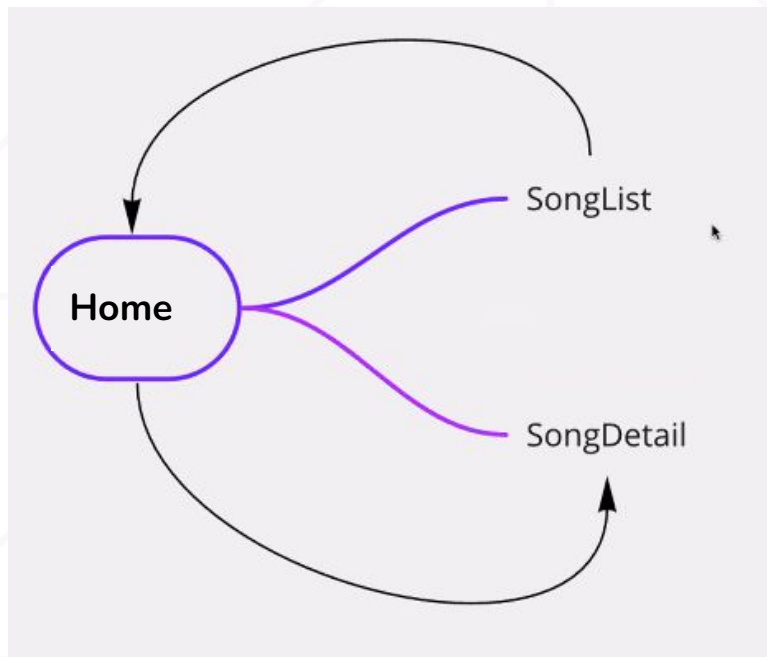
Mientras en la parte derecha no haya una canción elegida, mostrará una leyenda que solicitará al usuario que elija una canción.

Información de API en SongList



Si donde mandamos a llamar la información de la API es en **SongList**, entonces el otro componente **SongDetail** no tiene la forma de tener acceso directo a esa información.

Enviando información desde SongList...



Una posible solución es: Que **SongList** envíe la información de la canción seleccionada a Home y de ahí a **SongDetail**, pero esto no es viable por que la información da más vueltas y complica la lógica.

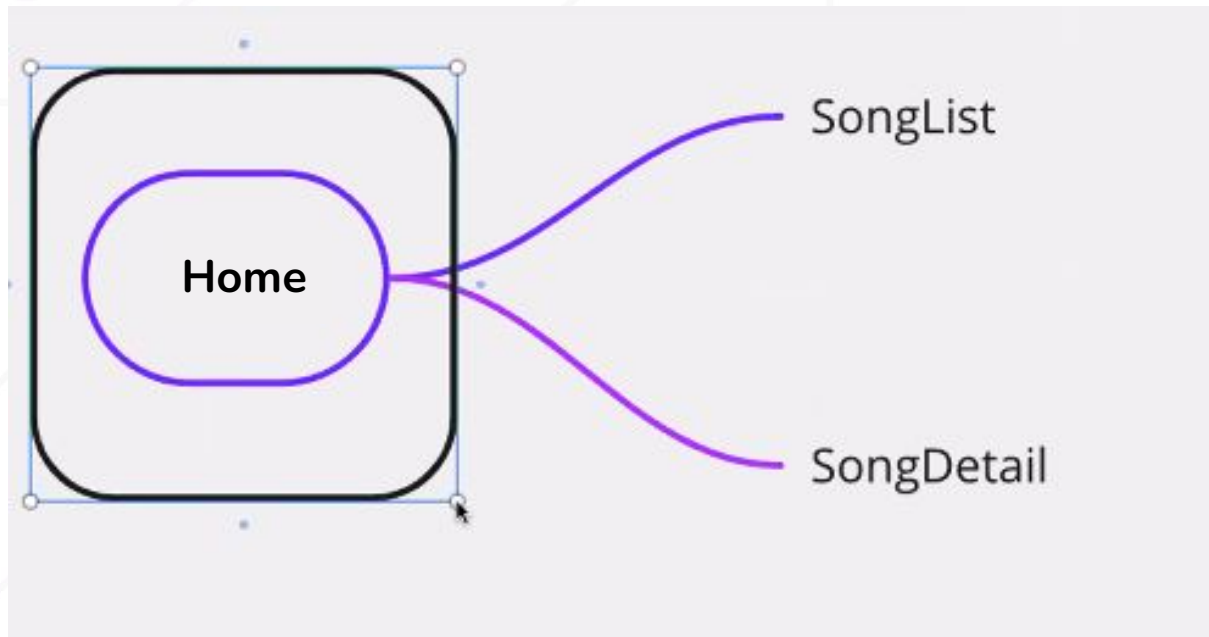
The logo consists of the text 'DEV.F.' in a bold, white, sans-serif font. The 'F' is stylized with a grid of small squares at its top right corner. The logo is centered within a dark blue diamond shape.

DEV.F.

*NO SE PUEDEN COMPARTIR
DATOS ENTRE COMPONENTES
DE LA MISMA JERARQUÍA.*

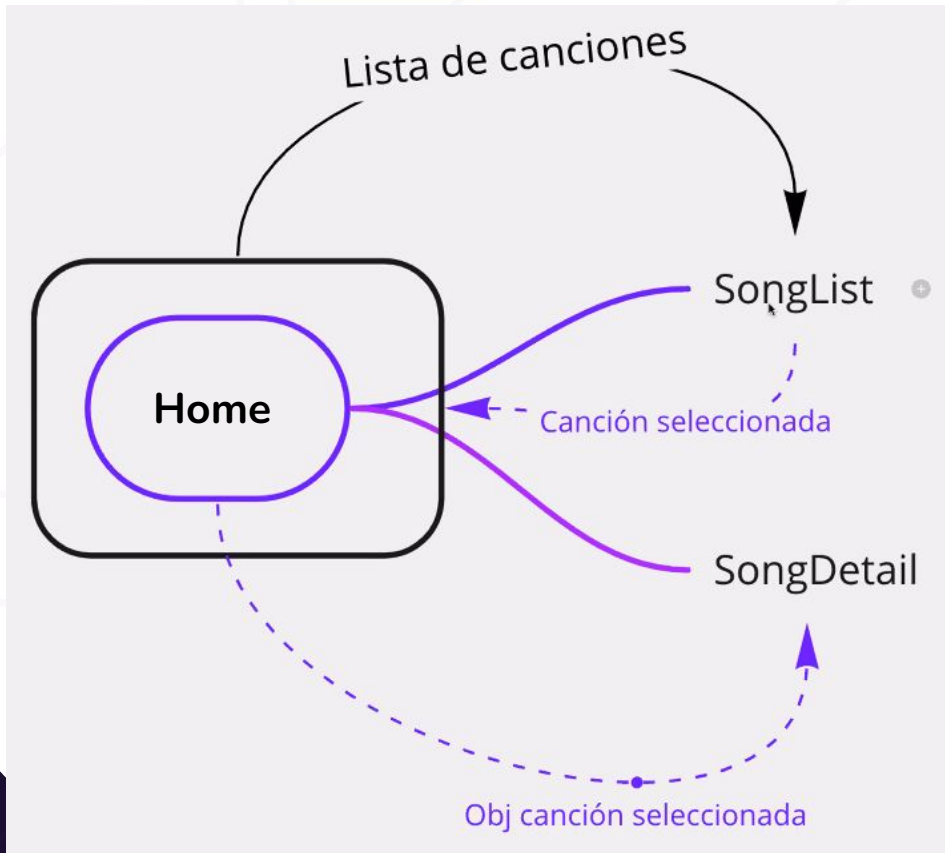
*SOLO DE PADRE A HIJO Y
VICEVERSA*

Información de API en Home



La solución más viable es mover la lógica de quien pide la información al padre de ambos: **Home**

Analizando la Lógica en Home



Al mover la lógica a Home, se manejaría con props:

1. **Home** obtiene la información de la API.
2. **SongList** recibe la lista de canciones de **Home**.
3. **SongList** envía de regreso a **Home** la canción seleccionada.
4. **Home** notifica a **SongDetail** que una canción fue seleccionada.
5. **SongDetail** pinta esta información.

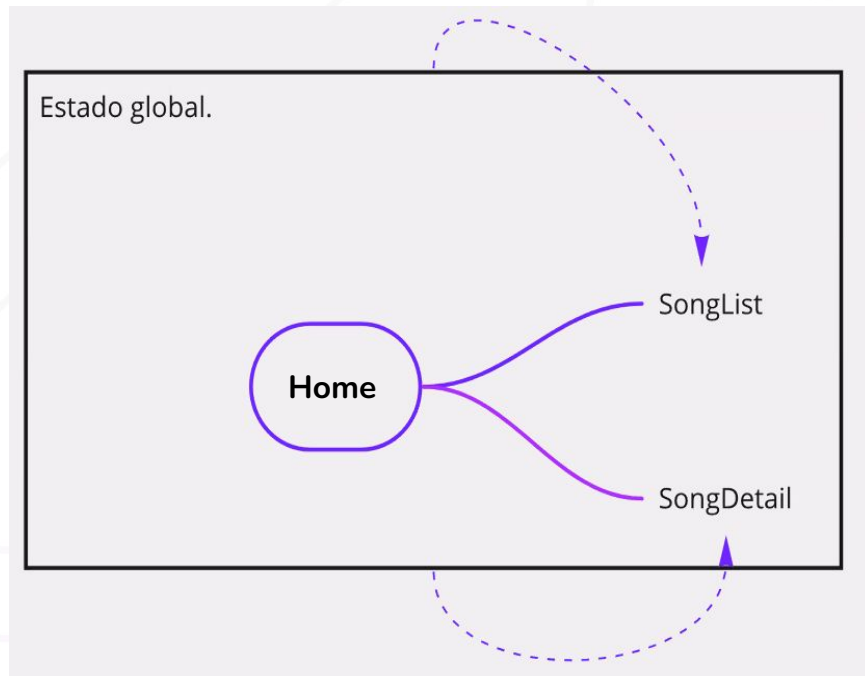
Este enfoque hace mucho ir y venir, el problema es que existe mucha circulación de información y **Home** realmente no necesita conocer toda esta información que manejan sus hijos.

La Solución: Estado Global

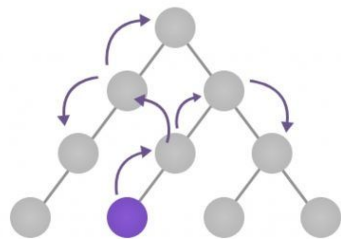
DEV.F.
DESARROLLAMOS(PERSONAS);


dev

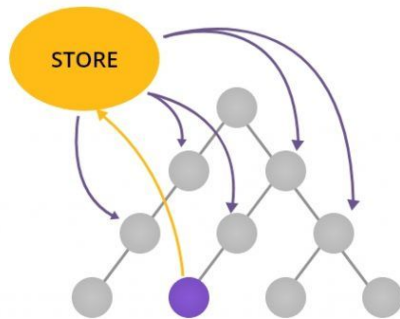
Estado Global



Podemos crear un estado global que tenga de habilidad de compartir las listas de canciones y la canción seleccionada.



 Component initiating change



Estado Global

Un estado global nos permite compartir valores y funcionalidades a través del árbol de componentes sin necesidad de usar props, además:

- Un estado global permite tener un estado donde **cualquier componente puede acceder**.
- No solo puede compartir información, si no **funcionalidades completas**.
- **No afectas a componentes** pasandole props que no necesitan.

PROPS DRILLING



REACT CONTEXT

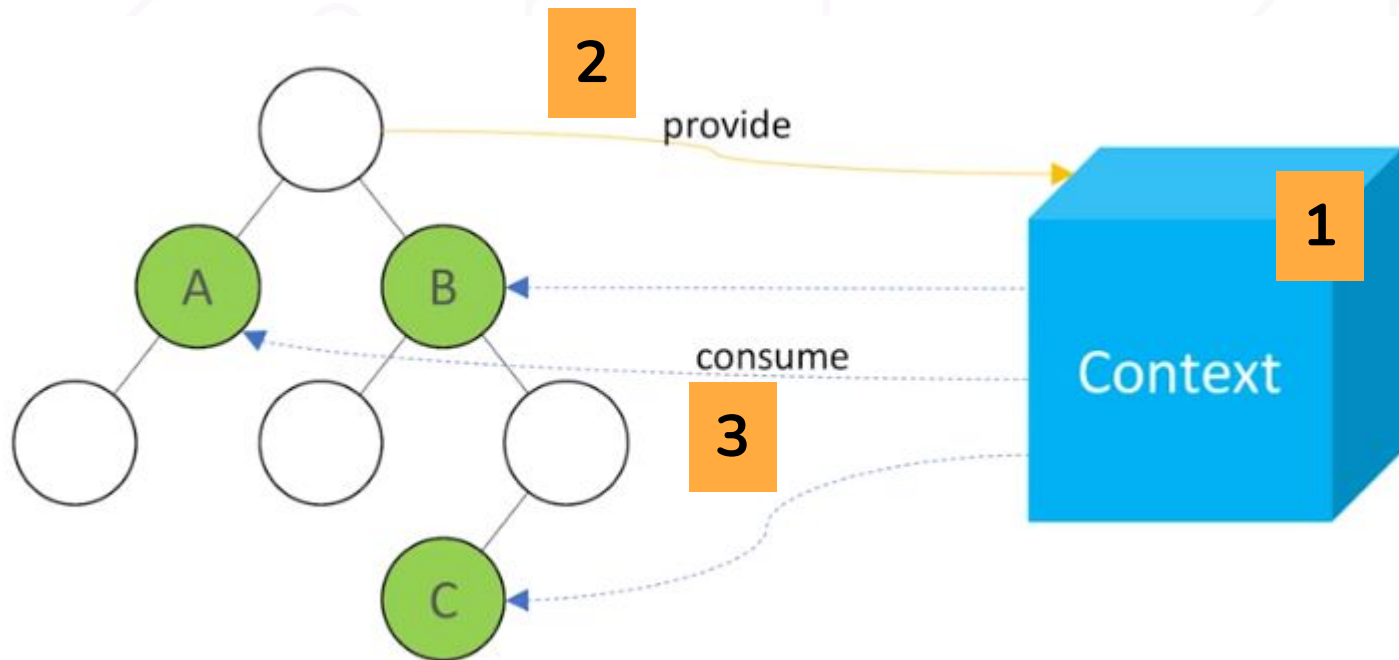


Estado Global en React: Context

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Elementos de React Context





```
const Context = React.createContext(  
  { count: 0, setCount: () => {} }  
);
```

1. Creando el contexto

createContext

Para crear un contexto hay que usar la función de React *createContext()*, que recibe como parámetro el valor predeterminado de dicho contexto, que pueden ser valores o funciones.

Guardaremos el contexto dentro de un *const* para referenciarlo a futuro por dicho nombre.



```
const App = () => {  
  const [count, setCount] = React.useState(0);  
  
  return (  
    <Context.Provider  
      value={{  
        count,  
        setCount  
      }}  
    >  
      <CounterComponent>  
    </Context.Provider>  
  );  
}
```

2. Proveedor del contexto

Context.Provider

El *Context.Provider* es un componente que recibe un **prop value** que serán los valores a compartir.

Todos los componentes renderizados dentro de este componente tendrán acceso a los valores del contexto.

Es decir, provee a todos los elementos hijos del componente *Context.Provider* de acceso a la información del Contexto.

```
const CounterComponent = () => {
  const MyContext = useContext(Context);

  return (
    <div>
      <span>{MyContext.count}</span>
      <button
        type="button"
        onClick={() => MyContext.setCount(MyContext.count + 1)}
      >
        Click
      </button>
    </div>
  )
}
```

3. Consumir valores del contexto

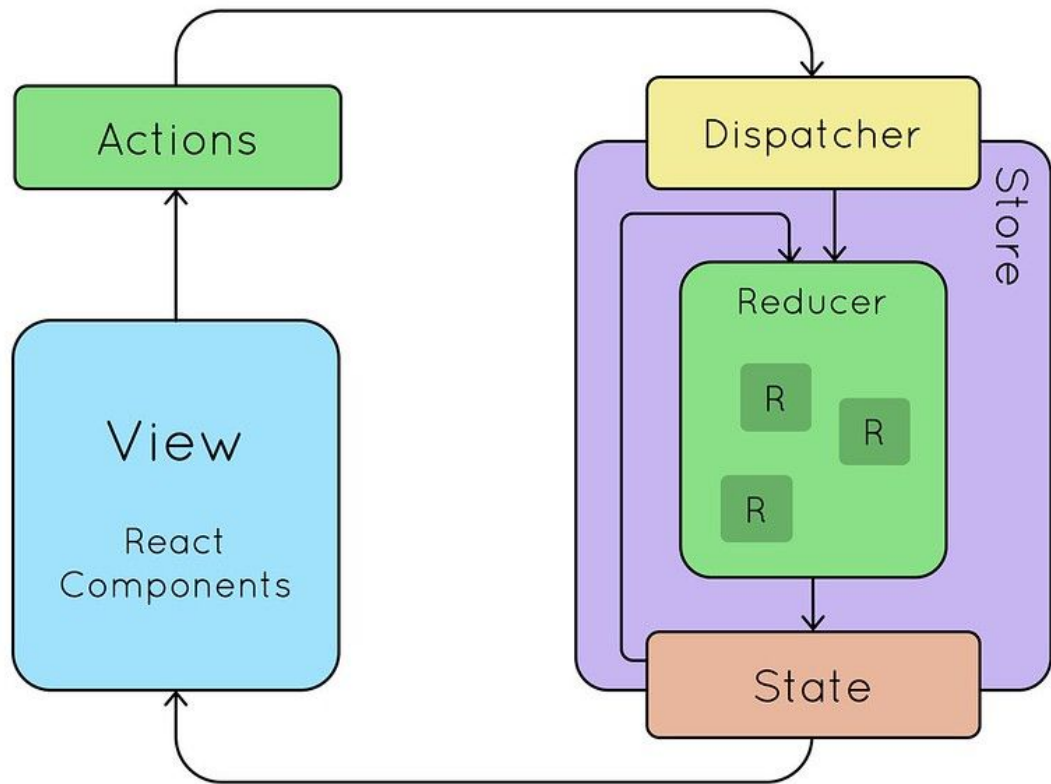
useContext hook

Podemos consumir los valores del contexto y almacenar los valores en una variable usando el hook *useContext*.

useContext recibe como argumento el contexto que se desea consumir.

Al guardarlo en un *const*, podemos acceder directamente a los valores del contexto indicado.

Otras Alternativas



¡ Vamos al Código !





¡GRACIAS!



César Guerra



www.cesarguerra.mx