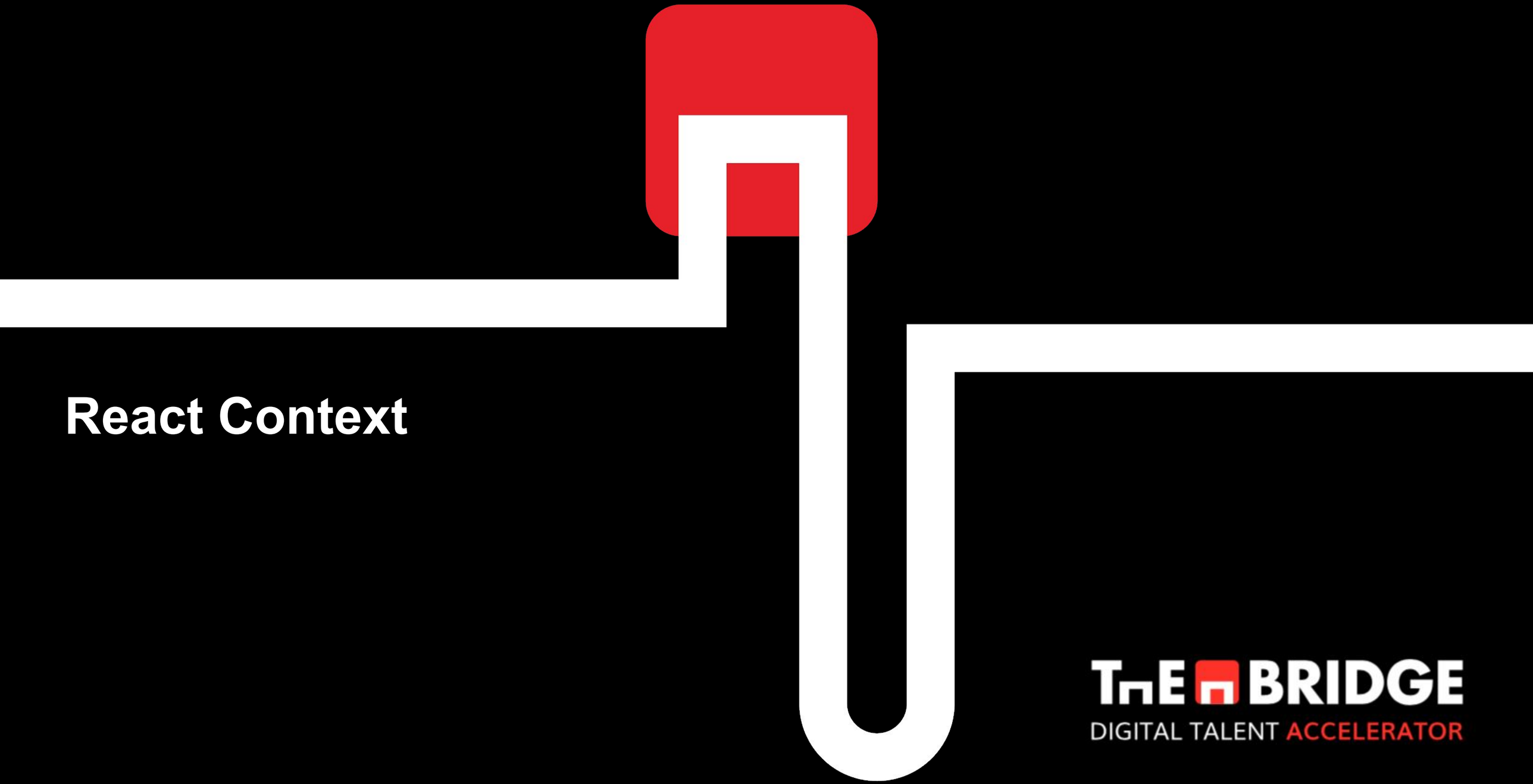
React Context

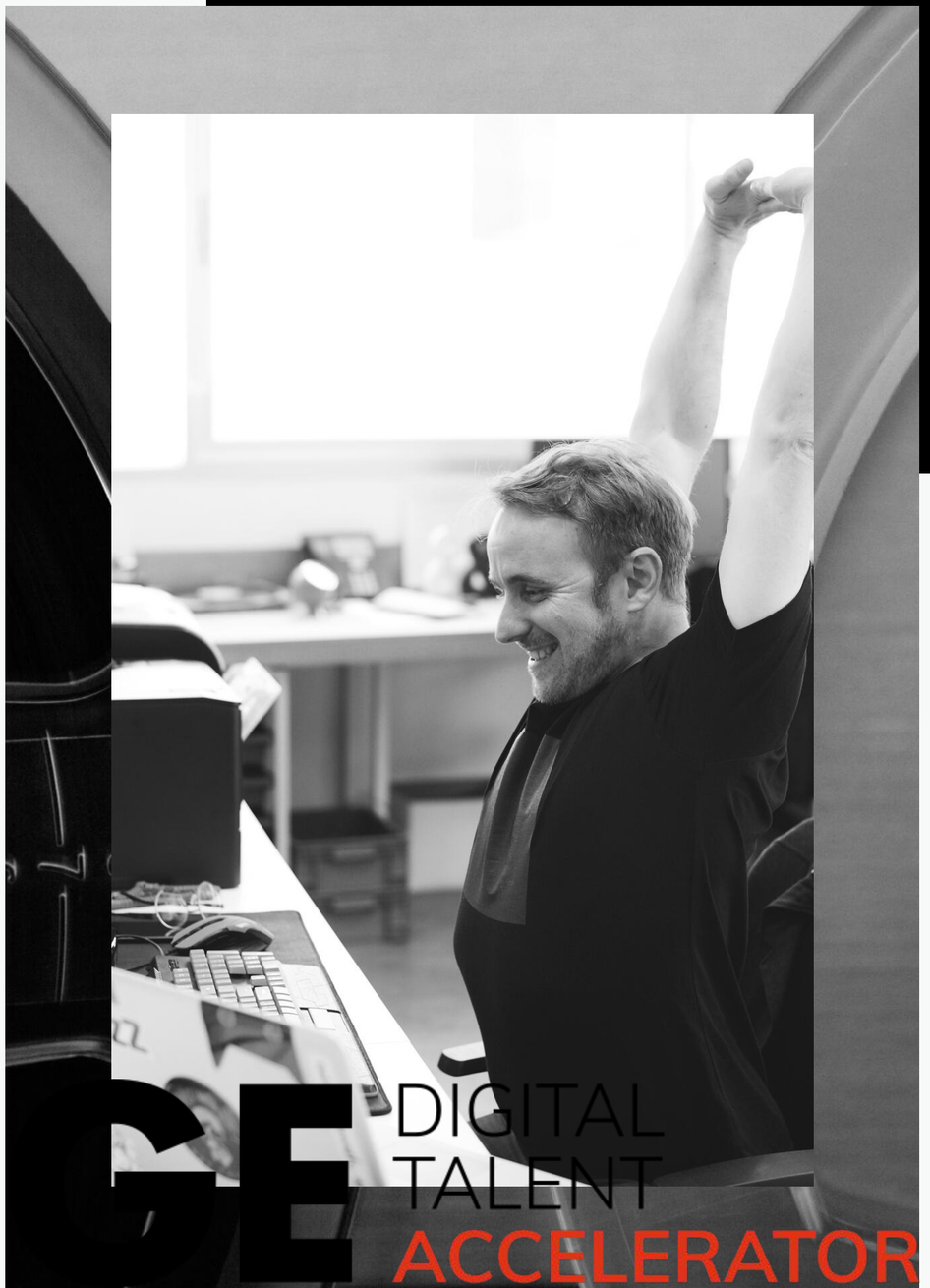


Índice

Petición a API

Context

UseReducer



Petición a API

Vamos a hacer una petición a la API de Rick y Morty, y nos traeremos todos los caracteres.
https://rickandmortyapi.com/documentation/#get-all-characters

Para ello tendremos que instalar **axios** como dependencia.

```
$ npm i axios
```

Petición a API

En la vista Home **importamos axios**, y nos creamos una **función asíncrona** que nos traerá los **datos** de la API.

Para tener acceso a los datos desde el principio, usamos el hook **useEffect** y llamamos a la función **getCharacters()** dentro.

```
import React, { useEffect } from 'react'
import axios from 'axios'

export default function Home() {
    const [characters, setCharacters] = useState([])

    useEffect(() => {
        getCharacters()
    }, [])

    const getCharacters = async () => {
        try {
            const response = await
        axios.get('https://rickandmortyapi.com/api/character')
        } catch (error) {
            console.error(error)
        }
    }
. . .
}
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Datos y nuevo estado

Los datos de la response sólo están **accesibles en el scope** de la función **getCharacters**. Para poder **acceder a los datos**, crearemos un **estado nuevo** donde pasaremos la response, y lo **inicializamos a array vacío**.
Una vez creado, seteamos el nuevo estado **characters** a array vacío.

Para ello **no hay que olvidarse** de **importar** el hook de React **useState**.

```
import React, { useEffect, useState } from 'react'
import axios from 'axios'
. . .

export default function Home() {
const [characters, setCharacters] = useState([])

const getCharacters = async () => {
   try {
      const response = await
   axios.get('https://rickandmortyapi.com/api/character')
   setCharacters(response.data.results)
   } catch (error) {
      console.error(error)
   }
}
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Iterar datos en la template

Ahora iteramos **characters** y sacamos los atributos que nos interesen para pintar por pantalla, como por ejemplo el **nombre** y la **imagen**.

```
export default function Home() {
. . .
    return (
    <>
        <h1>home page</h1>
        {characters.map((character, index) => (
            <div key={index}>
                <p>{character.name}</p>
                <img
                    src={character.image}
                    alt={character.name}
                />
            </div>
        ))}
    </>
    )
}
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Optimizar componentes

Crearemos ahora un nuevo componente **character.jsx** que sólo tendrá los datos que pasaremos por **props** desde home, con el fin de simplificarlo lo máximo posible (principio de responsabilidad única).

```jsx
export const Character = (props) => {
  return (
  <>
    <p>{props.data.name}</p>
    <img
      src={props.data.image}
      alt={props.data.name}
    />
  </>
  )
}
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

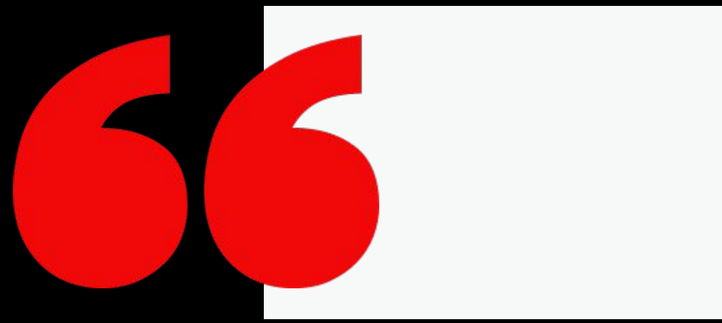
Optimizar componentes

Esta vez **Home** quedará así:

```
import React, { useEffect, useState } from 'react'
import axios from 'axios'
import { Character } from './Character'

export default function Home() {
    const [characters, setCharacters] = useState([])

    useEffect(() => {
        getCharacters()
    }, [])

    const getCharacters = async () => {
        try {
            const response = await
        axios.get('https://rickandmortyapi.com/api/character')
            setCharacters(response.data.results)
        } catch (error) {
            console.error(error)
        }
    }
    return (
    <>
        <h1>home page</h1>
        {characters.map((character, index) => (
        <Character key={index} data={character} />
        ))}
    </>
    )
}
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Context



Context proporciona una forma de **pasar datos** a través del árbol de componentes **sin** tener que pasar **props manualmente** en cada nivel.

En una aplicación típica de React, los datos se pasan de arriba hacia abajo a través de props.

Context proporciona una forma de compartir valores como estos entre los componentes sin tener que pasar explícitamente un apoyo a través de cada nivel del árbol.

THE BRIDGE
DIGITAL TALENT ACCELERATOR

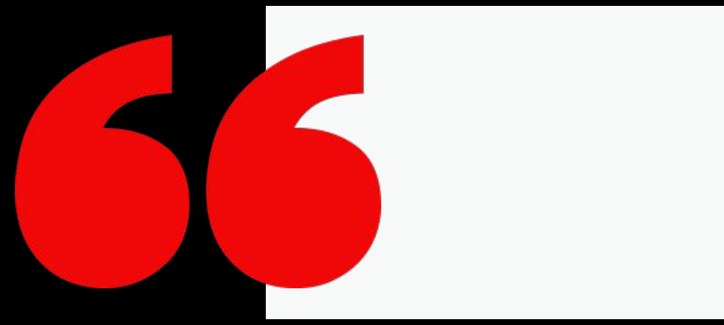
¿cuándo usar Context?

Context está diseñado para **compartir datos que pueden considerarse "globales"** para un árbol de componentes de React, como el usuario autenticado actual, el tema o el idioma preferido.

Creando nuestro primer contexto

Esta es la forma más directa de crear un contexto sin ningún valor predeterminado:

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



Primer proyecto con Context

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Context

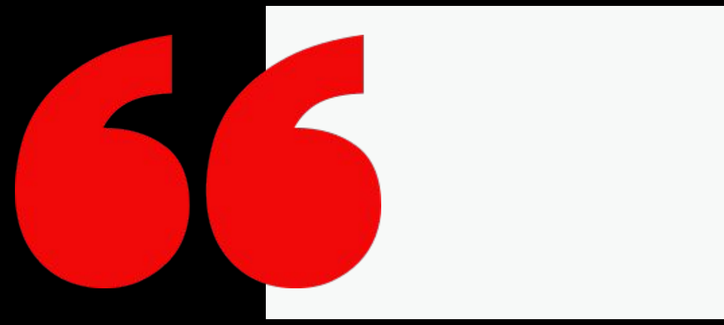
Dentro de **src**, creamos una carpeta **context** y dentro de ella el archivo **GlobalState.jsx**

```
import React, { createContext, useReducer } from 'react';

const initialState = {
  characters: []
}

export const GlobalContext = createContext(initialState);
```

Definimos el estado

Creamos nuestro contexto

THE BRIDGE
DIGITAL TALENT ACCELERATOR



useReducer

useReducer acepta una f**unción reducer** y el **estado inicial** de la aplicación. Luego **devuelve el estado actual** de la aplicación y una función de envío que podemos ejecutar para enviar **acciones para actualizar el estado** de nuestra aplicación.

useReducer

Es una **alternativa a useState**. Se recomienda **usarlo en lugar de useState cuando tenemos muchas variables de estado diferentes** que necesitamos actualizar al mismo tiempo.

Acepta un reducer de tipo **(state,action) => estado nuevo** y devuelve el estado actual junto con un método de envío.

```
const [state, dispatch] = useReducer(reducer, initialState)
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

AppReducer

Creamos el archivo **AppReducer.jsx** en la carpeta **context**.

Aquí podemos ver un ejemplo de definición del reducer :

Aunque generalmente se escribe usando sentencias switch, también puede usar if/else.

```
const characters = (state, action) => {
  switch (action.type) {
    case "GET_CHARACTERS":
      return {
        ...state,
        characters: action.payload,
      };
    default:
      return state;
  }
};
export default characters;
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Inicializamos nuestro reducer

Inicializamos nuestro reducer de la siguiente forma en **GlobalState.js**

```
import AppReducer from './AppReducer'
. . .

export const GlobalProvider = () => {
const [state, dispatch] = useReducer(AppReducer, initialState)
}
. . .
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

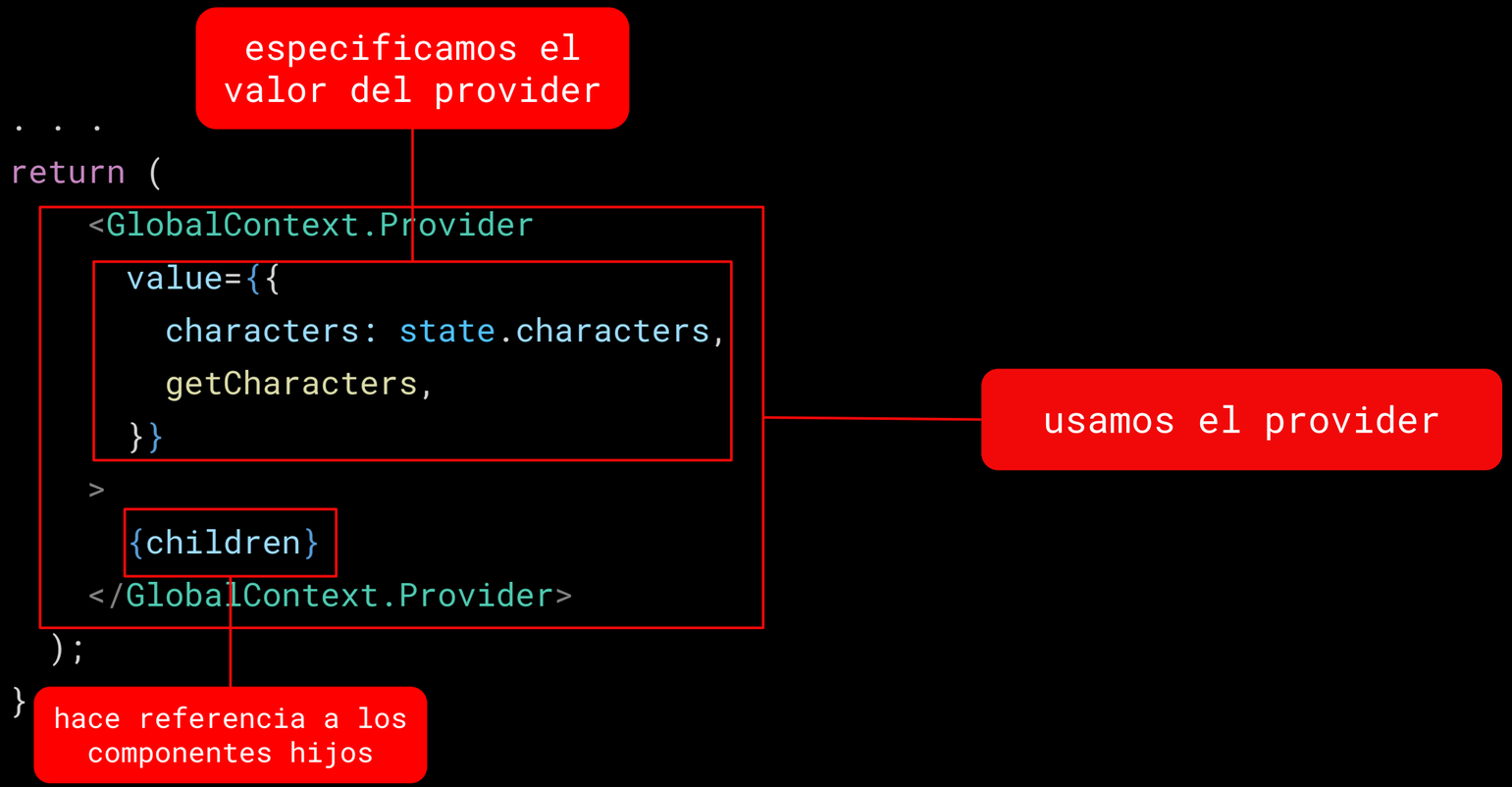
Petición axios

Importamos **axios** y creamos nuestra función **getCharacters** que hará una **petición get**. Después despacharemos la acción **GET_CHARACTERS** y le pasaremos la **respuesta** que nos da la API

```
import axios from "axios";
. . .
export const GlobalProvider = ({ children }) => {
  const [state, dispatch] = useReducer(AppReducer, initialState);

  const getCharacters = async () => {
    const response = await axios.get("https://rickandmortyapi.com/api/character");
    dispatch({
      type: "GET_CHARACTERS",
      payload: response.data.results,
    });
  };
. . .
```

GlobalProvider

Ahora definimos lo que devolverá nuestro global provider:



THE BRIDGE
DIGITAL TALENT ACCELERATOR

GlobalProvider

Recapitulando:

- Nos importamos **useReducer** y lo inicializamos.
- Creamos una función **getCharacters** que hará una petición para traernos los datos y a su vez **despachará** una **acción** para cambiar el **estado actual** de los characters.

esperará que por parámetro le lleguen los hijos (los componentes)

```
. . .
export const GlobalProvider = ({ children }) => {
  const [state, dispatch] = useReducer(AppReducer, initialState);

  const getCharacters = async () => {
    const res = await axios.get("https://rickandmortyapi.com/api/character");
    dispatch({
      type: "GET_CHARACTERS",
      payload: res.data.results,
    });
  };

  return (
    <GlobalContext.Provider
      value={{
        characters: state.characters,
        getCharacters,
      }}
    >
      {children}
    </GlobalContext.Provider>
  );
};
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR

GlobalProvider

Usamos un **provider** para pasar el **contexto** de la aplicación actual a todos los **componentes hijos**. Cualquier componente puede leerlo, sin importar qué profundo sea su nivel de anidamiento.

```jsx
import "./App.css";
import Characters from "./components/Characters/Characters";
import { GlobalProvider } from "./context/GlobalState";
function App() {
  return (
    <div className="App">
      <GlobalProvider>
        <Characters />
      </GlobalProvider>
    </div>
  );
}

export default App;
```

Englobamos nuestro componente en el GlobalProvider

THE BRIDGE
DIGITAL TALENT ACCELERATOR

Utilizando context

Creamos nuestro componente Characters y con **useContext** utilizamos el context que habíamos creado:

El **hook useContext** permite pasar datos a elementos secundarios sin tener que pasar todo el estado a través de props.

```
import React, { useContext, useEffect } from "react";
import { GlobalContext } from "../../context/GlobalState";

const Characters = () => {
  const { characters, getCharacters } = useContext(GlobalContext)

  useEffect(() => {
    getCharacters();
  }, []);

  const character = characters.map((character) => {
    return (
      <div key={character.id}>
        <h1>{character.name}</h1>
        <img src={character.image} />
      </div>
    );
  });
  return <div>{character}</div>;
};

export default Characters;
```

THE BRIDGE
DIGITAL TALENT ACCELERATOR