

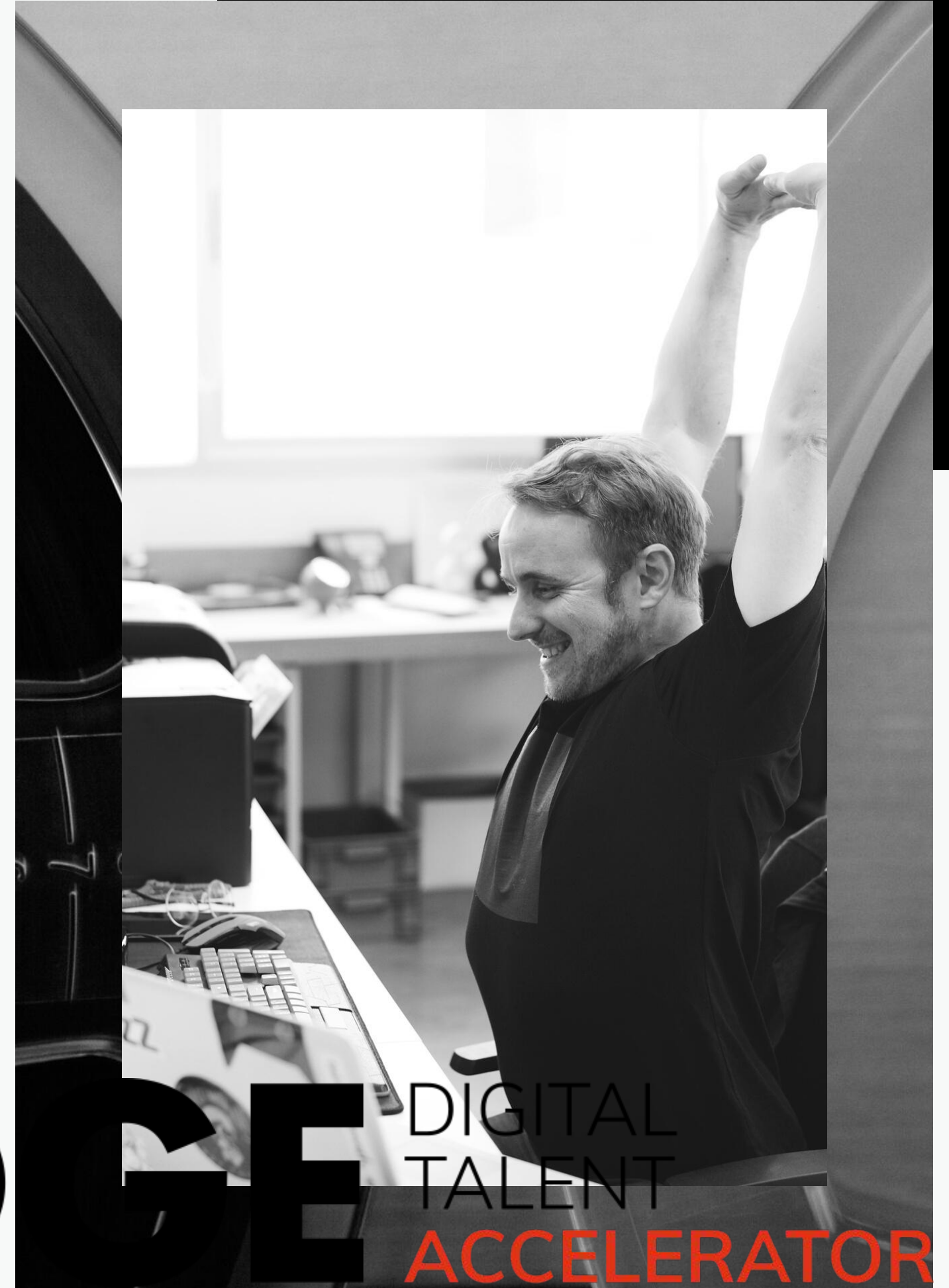


Autenticación con context

Índice

1. Login de un usuario
2. Obtener información del usuario conectado
3. Cerrar sesión

THE  **BRIDGE** **DIGITAL
TALENT
ACCELERATOR**





Login de un usuario

Vamos a crear el sistema de login de un usuario utilizando context.

User Context - Login

Creamos el contexto del usuario con su **estado inicial**. Además le decimos que si **existe un token en el localStorage** sea el valor inicial de la propiedad **token**.

Nos encontramos en el archivo:

context/UserContext/UserState.js

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

const token = JSON.parse(localStorage.getItem("token"));

const initialState = {
  token: token ? token : null,
  user: null,
};

const API_URL = "http://localhost:3000";

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

User Context - Login

Creamos la llamada al **reducer para manejar los estados** de la autenticación y le añadimos el primer **método login**. Si login consigue una respuesta **res.data**, el token que nos devuelva **lo guardaremos en nuestro localStorage**.

Nos encontramos en el archivo:
context/UserContext/UserState.js

```
import { createContext, useReducer } from "react";
import axios from "axios";
import UserReducer from "../UserReducer";

. . .

export const UserProvider = ({ children }) => {
  const [state, dispatch] = useReducer(UserReducer, initialState);

  const login = async (user) => {
    const res = await axios.post(API_URL + "/users/login", user);
    dispatch({
      type: "LOGIN",
      payload: res.data,
    });
    if (res.data) {
      localStorage.setItem("token", JSON.stringify(res.data.token));
    }
  }
};
```

User Context - Login

Ya tenemos la llamada al reducer, ahora implementamos su funcionalidad. El reducer modificará el **estado**. En este caso colocará el **valor devuelto por nuestro back** (accediendo al valor de "token") en la **propiedad "token"** del estado.

Nos encontramos en el archivo:

UserReducer.jsx

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

User Context - Login

Ahora necesitamos hacer global el estado a los componentes en los que necesitemos usar el estado (“children”).

Para ello, utilizamos el **método “Provider” de context para pasar la información globalmente.**

Nos encontramos en el archivo:

UserState.js

```
...  
return (  
  <UserContext.Provider  
    value={{  
      token: state.token,  
      user: state.user,  
      login,  
    }}  
  >  
    {children}  
  </UserContext.Provider>  
)  
}
```

User Context - Login

En el archivo de entrada **App.js** añadiremos el **routing de la aplicación** y usamos **el provider** para pasarle a los componentes hijos el contexto de la aplicación.

Nos encontramos en el archivo **App.js**

```
import { BrowserRouter, Route, Routes } from
"react-router-dom";
import Login from './components/Login/Login';
import { UserProvider } from './context/UserContext/UserState';

function App() {
  return (
    <UserProvider>
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<Login />} />
        </Routes>
      </BrowserRouter>
    </UserProvider>
  );
}

export default App;
```


User Context - Login

Creamos el componente **Login**. Este **se encargará de pintar el formulario** donde introduciremos los datos para loguearnos.

Estamos utilizando AntDesign:

<https://ant.design/>

comando instalación: **npm i antd**

Nos encontramos en el archivo:

COMPONENTE -> Login.jsx

```
return (  
  <div className="container">  
    <Form  
      name="basic"  
      labelCol={{ span: 8 }}  
      wrapperCol={{ span: 16 }}  
      initialValues={{ remember: true }}  
      onFinish={onFinish}  
      onFinishFailed={onFinishFailed}  
      autoComplete="off"  
    >  
      <Form.Item  
        label="Email"  
        name="email"  
        rules={[{ required: true, message: "Please input your email!" }]}  
      >  
        <Input />  
      </Form.Item>  
  
      <Form.Item  
        label="Password"  
        name="password"  
        rules={[{ required: true, message: "Please input your password!" }]}  
      >  
        <Input.Password />  
      </Form.Item>  
  
      <Form.Item wrapperCol={{ offset: 8, span: 16 }}>  
        <Button type="primary" htmlType="submit">  
          Submit  
        </Button>  
      </Form.Item>  
    </Form>  
  </div>  
};  
  
export default Login
```

User Context - Login

Añadimos las **importaciones** necesarias a nuestro componente. Nos traemos **la función “login”** del estado global y mediante las **funciones de AntDesign** creamos la lógica del funcionamiento del envío del formulario.

Nos encontramos en el archivo:

COMPONENTE → Login.jsx

```
import './Login.scss';
import { useContext } from 'react';
import { UserContext } from '../../../context/UserContext/UserState';
import { Form, Input, Button } from 'antd';

const Login = () => {
  const { login } = useContext(UserContext);

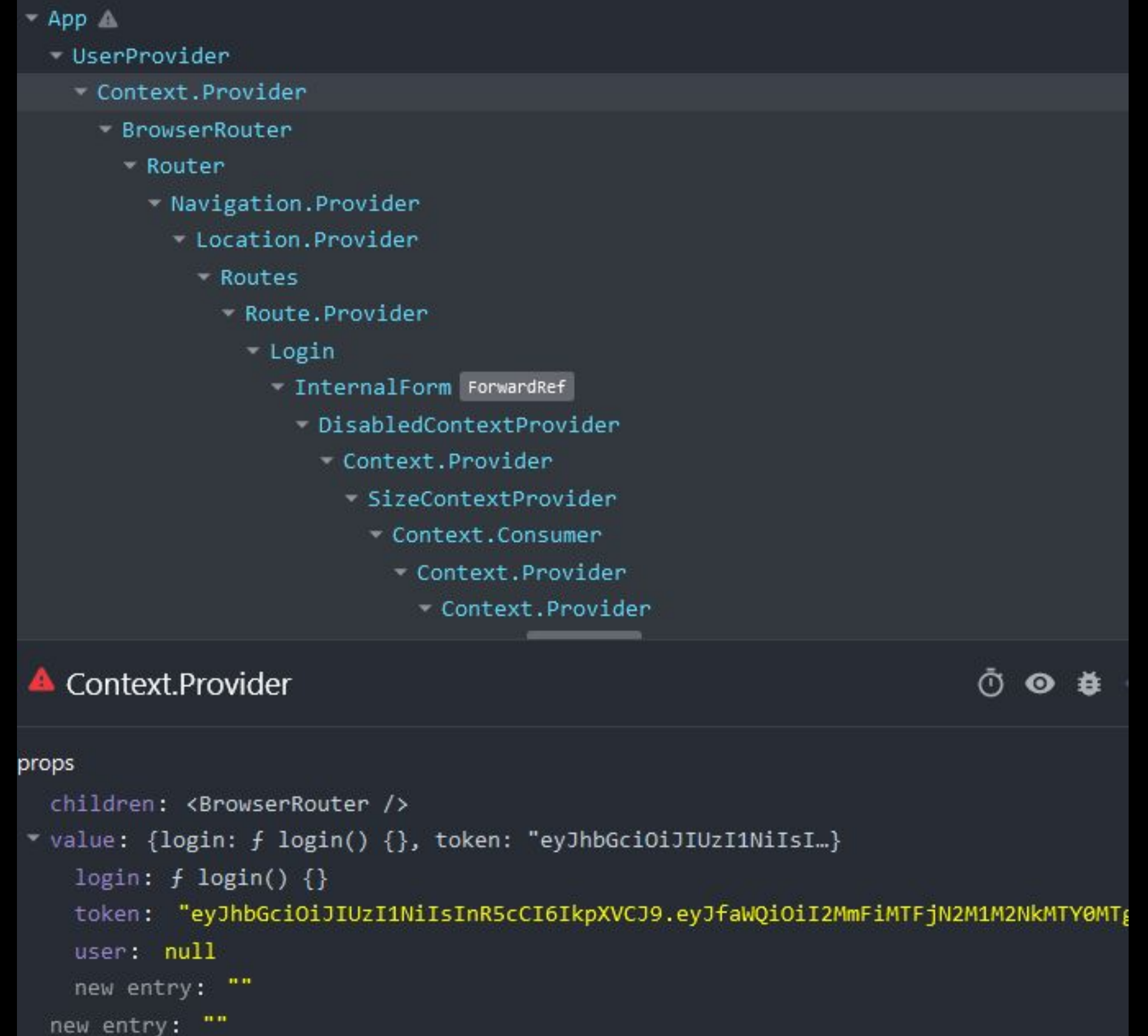
  const onFinish = (values) => {
    login(values)
  };

  const onFinishFailed = (errorInfo) => {
    console.log("Failed:", errorInfo);
  };

  . . .
```

User Context - Login

Comprobamos el funcionamiento del **login**. Nos aseguramos de que existe en el **localstorage** y vemos si efectivamente el estado de nuestra aplicación ha cambiado.



The screenshot shows the Redux DevTools interface. The top pane displays the Redux state tree, which is a nested structure of providers and components. The bottom pane shows the props for the selected `Context.Provider` component, which includes a `value` object containing login and token information.

```
▼ App ▲
  ▼ UserProvider
    ▼ Context.Provider
      ▼ BrowserRouter
        ▼ Router
          ▼ Navigation.Provider
            ▼ Location.Provider
              ▼ Routes
                ▼ Route.Provider
                  ▼ Login
                    ▼ InternalForm ForwardRef
                      ▼ DisabledContextProvider
                        ▼ Context.Provider
                          ▼ SizeContextProvider
                            ▼ Context.Consumer
                              ▼ Context.Provider
                                ▼ Context.Provider
```

Context.Provider

```
props
  children: <BrowserRouter />
  ▼ value: {login: f login() {}, token: "eyJhbGciOiJIUzI1NiIsI...}
    login: f login() {}
    token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MmFiMTFjN2M1M2NkMTY0MTg...
    user: null
    new entry: ""
    new entry: ""
```



Obtener información del usuario

Crearemos un componente que realice la llamada al endpoint para traernos al usuario que haya iniciado sesión.

User Context - GetUserInfo

Añadimos la función “getUserInfo” dentro de nuestro UserProvider.

Deberemos comprobar si existe el token en localStorage y después **introducirlo como segundo parámetro de la petición axios.**

Nos encontramos en el archivo:

UserState.js

```
. . .
const getUserInfo = async () => {
  const token = JSON.parse(localStorage.getItem("token"));
  const res = await axios.get(
    API_URL + "/users/info",
    {
      headers: {
        authorization: token,
      },
    }
  );
  dispatch({
    type: "GET_USER_INFO",
    payload: res.data,
  })
  return res;
};
. . .
```

User Context - GetUserInfo

Creamos un nuevo caso en el switch del reducer que se encarga de **actualizar nuestro estado de user** con la respuesta de nuestro back.

Nos encontramos en el archivo:

UserReducer.js

```
. . .  
    case "GET_USER_INFO":  
        return {  
            ...state,  
            user: action.payload,  
        };  
. . .
```

User Context - GetUserInfo

Añadimos la función “**getUserInfo**” al provider para poder tenerlo disponible en el **estado global de la aplicación**.

Nos encontramos en el archivo:

UserState.js

```
. . .  
return (  
  <UserContext.Provider  
    value={{  
      token: state.token,  
      user: state.user,  
      login,  
      getUserInfo  
    }}  
  >  
    {children}  
  </UserContext.Provider>  
)  
};
```


User Context - GetUserInfo

Agregamos el componente fijo **"Header"** y en nuestras rutas añadimos un path adicional que va a cargar el componente **"Profile"**, además hemos puesto **un alias a BrowserRouter**, ahora se llama "Router"

Nos encontramos en el archivo:

App.js.js

```
import './App.css';
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import "antd/dist/antd.css";
import Login from './components/Login/Login';
import Profile from './components/Profile/Profile';
import Header from './components/Header/Header';
import { UserProvider } from './context/UserContext/UserState';

function App() {
  return (
    <div className="App">
      <UserProvider>
        <Router>
          <Header />
          <Routes>
            <Route path="/" element={<Login />} />
            <Route path="/profile" element={<Profile />} />
          </Routes>
        </Router>
      </UserProvider>
    </div>

  );
}

export default App;
```


User Context - GetUserInfo

Creamos el componente "Header" que **incluye los links de navegación** de nuestra aplicación.

Nos encontramos en el archivo:
COMPONENTE -> Header.jsx

```
import "../Header.scss";
import { Link } from "react-router-dom";

function Header() {

  return (
    <nav className="header">
      <span>Header</span>
      <div>
        <span>
          <Link to="/">Login</Link>
        </span>
        <span>
          <Link to="/profile">Profile</Link>
        </span>
      </div>
    </nav>
  );
}

export default Header;
```

User Context - GetUserInfo

Creamos el **componente "Profile"** para **mostrar el nombre del usuario que está conectado**, la petición se hace al montarse el componente. **Si no existe usuario se muestra un texto** hasta que cargue la petición de obtenerlo.

Nos encontramos en el archivo:

COMPONENTE -> Profile.jsx

```
import { useContext, useEffect } from "react";
import { UserContext } from "../../context/UserContext/UserState";

const Profile = () => {
  const { getUserInfo, user } = useContext(UserContext);

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

  if (!user) {
    return <span>Cargando...</span>;
  }

  return <div>Profile {user.name}</div>;
};

export default Profile;
```



Cerrar sesión

Vamos a crear la funcionalidad de cerrar sesión o también llamada logout.

User Context - Logout

Añadimos la función “logout” a nuestro UserProvider. Deberemos consultar de nuevo localStorage y además si obtenemos respuesta **borraremos nuestro key “token” de nuestro localStorage.**

Nos encontramos en el archivo:
UserState.js

```
. . .  
const logout = async () => {  
  const token = JSON.parse(localStorage.getItem("token"));  
  const res = await axios.delete(API_URL + "/users/logout",  
  {  
    headers: {  
      authorization: token,  
    },  
  });  
  dispatch({  
    type: "LOGOUT",  
    payload: res.data,  
  });  
  if (res.data) {  
    localStorage.removeItem("token");  
  }  
};  
. . .
```

User Context - Logout

Creamos un nuevo caso en el switch del reducer y esta vez **establecemos en "null"** el **estado** tanto de **user** como del **token**.

Nos encontramos en el archivo:

UserReducer.js

```
. . .
    case "LOGOUT":
        return {
            ...state,
            user: null,
            token: null
        };
. . .
```

User Context - Logout

Hacemos **accesible la función “logout”** al resto de la aplicación pasando su referencia a través del provider.

Nos encontramos en el archivo:

UserState.js

```
. . .  
return (  
  <UserContext.Provider  
    value={{  
      token: state.token,  
      user: state.user,  
      login,  
      getUserInfo,  
      logout  
    }}  
  >  
    {children}  
  </UserContext.Provider>  
)  
};
```

User Context - Logout

Implementamos la **funcionalidad de cerrar sesión** en el componente “Header”. Además creamos un **ternario** para que nos renderice partes del “Header” en función de **si existe el token o no**.

Nos encontramos en el archivo:
COMPONENTE -> Header.jsx

```
import './Header.css';
import { Link, UseNavigate } from 'react-router-dom';
import { useContext } from 'react';
import { UserContext } from '../../context/UserContext/UserState';

function Header() {
  const navigate = useNavigate();
  const { token, logout } = useContext(UserContext);

  const logoutUser = () => {
    logout();
  };

  return (
    <nav className="header">
      <span>Header</span>
      <div>
        {token ? (
          <>
            <span onClick={logoutUser}>
              <Link to="/">Logout</Link>
            </span>
            <span>
              <Link to="/profile">Profile</Link>
            </span>
          </>
        ) : (
          <span>
            <Link to="/">Login</Link>
          </span>
        )}
      </div>
    </nav>
  );
}

export default Header;
```

User Context - Logout

Agregamos una **redirección** cuando se produce el **logout** en la aplicación.

Nos encontramos en el archivo: **COMPONENTE -> Header.jsx**

```
. . .  
const logoutUser = () => {  
  logout();  
  setTimeout(() => {  
    navigate("/")  
  }, 2000)  
};  
. . .
```


User Context - Login

Agregamos una **redirección** cuando se produce el **login** en la aplicación.

Nos encontramos en el archivo: **COMPONENTE -> Login.jsx**

```
. . .
useEffect(() => {
  setTimeout(() => {
    const foundToken = JSON.parse(localStorage.getItem("token"));
    if (foundToken) {
      navigate("/profile")
    }
  }, 2000)

}, [login])
. . .
```