

Introducción al desarrollo Web con Ant Design

Obtener datos de usuario

Para comenzar a consumir más endpoint, en este caso obtener los datos del usuario, vamos a hacer un cambio y almacenar la url y los endpoint en variables, dirígete a → src/utils/constants.js crear API_URL y ENDPOINT de acuerdo a la siguiente estructura.

```
src > utils > JS constants.js > ENV
1  export const ENV = {
2    API_URL: 'http://localhost:3000',
3    ENDPOINTS: {
4      LOGIN: 'api/auth/signin',
5      REGISTER: 'api/auth/signup',
6      USERS: 'api/users',
7    },
8    STORAGE: {
9      TOKEN: "token",
10   }
11 }
```

Entonces tu archivo auth de services quedaría de la siguiente manera, no olvides importar {ENV}

```
src > services > JS auth.js > ...
1  import axios from 'axios';
2  import { ENV } from '../utils/constants';
3
4  const register = async (username, email, password) => {
5    return axios.post(`${ENV.API_URL}/${ENV.ENDPOINTS.REGISTER}`, {
6      username,
7      email,
8      password,
9      roles: ['user'],
10   });
11 };
12
13 const loginF = async (email, password) => {
14   return axios.post(`${ENV.API_URL}/${ENV.ENDPOINTS.LOGIN}`, {
15     email,
16     password,
17   });
18 };
19
20 export default {
21   register,
22   loginF,
23 };
24
25
```

Auth Fetch

Para seguir construyendo la sesión del usuario, crea el archivo src/utils/authFetch.js
Crea la función authFetch y por ahora solo devuelve un console.log

```
src > utils > JS authFetch.js > authFetch
1  export const authFetch = async (url, params) => {
2    console.log('Hola desde authFetch')
3  }
```

Crea el archivo src/services/users.js donde construirás la solicitud a la API para obtener los datos de usuario. Ten en cuenta que al ser un método GET el método se

infiere y nos es necesario especificarlo, y en los headers como lo hacíamos en postman, envía Authorization.

Como este es un endpoint que necesita recibir el `userId` como parámetro, es necesario decodificar el token para extraerlo, es necesario instalar la dependencia

`npm install jwt-decode` e importar en el archivo `import { jwtDecode } from "jwt-decode";`

No olvides importar `ENV` y la función de `authFetch` creada en el paso anterior, crea la constante que va a almacenar el token decodificado, y crea la constante `userId` que recibe el valor del `id`.

Tu solicitud quedaría de la siguiente manera.

```
src > services > js users.js > ...
1  import { jwtDecode } from "jwt-decode";
2  import { ENV } from '../utils/constants';
3  import { authFetch } from '../utils/authFetch';
4
5
6  const getMe = async (token) => {
7
8      try {
9          const decoded = jwtDecode(token);
10         const userId = decoded.id;
11         const url = `${ENV.API_URL}/${ENV.ENDPOINTS.USERS}/${userId}`
12         const response = await authFetch(url)
13
14         return await response.json();
15     } catch (error) {
16         console.error(error)
17     }
18 }
19 export const userService = {
20     getMe
21 }
```

Finalmente dentro del contexto en la función de login, llama a **`userService`** y un `console.log` de la respuesta

```
const login = async (token) => {
    try {
        console.log('Obteniendo', token);
        await storageController.setToken(token);
        const response = await userService.getMe(token);
```

.Obtendrás algo como esto.

Obteniendo
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
wIjoxNzE3NjM0ODE3fQ.sHBkrmkDFz0wDD2xc

Hola desde authFetch

❌ ▶ Error al obtener el usuario: TypeE

Crea el archivo `utils/tokenExpired.js` e importa la dependencia ***jwt-decode***.
y crea la función `tokenExpired`.

```
src > utils > JS tokenExpired.js > [🔗] tokenExpired
1   import { jwtDecode } from "jwt-decode";
2
3   export const tokenExpired = (token) => {
4     const tokenDecode = jwtDecode(token);
5     console.log('TokenDecode:', tokenDecode);
6   }
```

En el archivo `authFetch` importa la función y construye la solicitud, también has el import de `storageController` para traernos el token.

Construye la función de `logout` donde por ahora solo usamos el método `removeToken`.

Dentro del `else` llama `tokenExpired` y haz un `console.log` de la respuesta.

```
src > utils > JS authFetch.js > [?] authFetch
1  import { storageController } from "../services/token";
2  import { tokenExpired } from "../tokenExpired";
3
4  export const authFetch = async (url, params) => {
5      const token = await storageController.getToken();
6
7      const logout = () => {
8          storageController.removeToken();
9      }
10
11     if (!token) {
12         logout();
13     } else {
14         const response = tokenExpired(token);
15         console.log('Response:', response);
16     }
17 }
18
```

```
TokenDecode: ▼ {id: '665e36d901527eab156259a0', iat: 1717549115, exp: 1717635515} i
  exp: 1717635515
  iat: 1717549115
  id: "665e36d901527eab156259a0"
  ► [[Prototype]]: Object
```

Lo que sigue, dentro de la función tokenExpired, comprobar si el token ha expirado o no, eso lo haces comparando si la fecha actual es mayor que la fecha de expiración.

Si es así nos devuelve true y si no false.

```
src > utils > JS tokenExpired.js > ...
1  import { jwtDecode } from "jwt-decode";
2
3  export const tokenExpired = (token) => {
4      const tokenDecode = jwtDecode(token);
5      const expired = tokenDecode.exp * 1000;
6      const time = new Date().getTime();
7      if (time > expired) {
8          return true;
9      }
10     return false;
11 }
12
```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjwIj0xNzE3MjMDk3fQ.IjJqW-8xMXg3JUi7aqJ-yykL6t-

🔴 Error al obtener el usuario: TypeError: Cannot

```
rc > utils >  authFetch.js > ...
1 import { storageController } from "../services/token"
2 import { tokenExpired } from "../tokenExpired"
3
4
5 export const authFetch = async (url, params) => {
6   const token = await storageController.getToken();
7
8   const logout = () => {
9     storageController.removeToken();
10   }
11   if (!token) {
12     logout();
13   }
14   else {
15     if (tokenExpired(token)) {
16       logout();
17     }
18     else {
19       const options = {
20         ...params,
21         headers: {
22           ...params?.headers,
23           Authorization: `Bearer ${token}`
24         }
25       }
26       try {
27         return await fetch(url, options)
28       } catch (error) {
29         console.error(error)
30       }
31     }
32   }
33 }
34
35
```

Asegúrate que en user estés importando y llamando la función.

Inicia sesión en tu aplicación y deberás obtener una respuesta como esta.

```
Response AuthContext.jsx:22
{
  _id: '665e36d901527eab156259a0', username:
  'Ana', password: '$2a$10$IZ1KV0LumjVQYx9Hrj/
  c00cXqAUwS2n01P29x8H.5R4A4s6FsFeKi', email:
  'ana@arteaga.com', roles: Array(1), ...}
  createdAt: "2024-06-03T21:34:17.545Z"
  email: "ana@arteaga.com"
  password: "$2a$10$IZ1KV0LumjVQYx9Hrj/c00cXc
  roles: ['65e6453e7fec1bd764b997a2']
  updatedAt: "2024-06-03T21:34:17.545Z"
  username: "Ana"
  _id: "665e36d901527eab156259a0"
  [[Prototype]]: Object
>
```

Para terminar de crear la sesión del usuario, en el archivo AuthContext crear los siguientes estados.

```
export const AuthProvider = (props) => {
  const { children } = props
  //Crear el estado del usuario
  const [user, setUser] = useState(null);
  //Crear el estado de carga
  const [loading, setLoading] = useState(true);
```

usamos el estado loading y setLoading,
Mientras loading devolvemos true

```
const data = {
  user: null,
  login,
  logout: () => console.log('logout'),
  updateUser: () => console.log('update user')
}
if (loading) return null;
return (
  <AuthContext.Provider value={data}>
    {children}
  </AuthContext.Provider>
)
```

En la función getSession pasamos setLoading(false), esto indica que al existir una sesión, setLoading es false.

```
const getSession = async () => {  
  const token = await storageController.getToken();  
  //console.log("Token --> ", token)  
  setLoading(false);  
}
```

En la función de login, actualizar el estado de user y loading

```
const login = async (token) => {  
  try {  
    await storageController.setToken(token);  
    const response = await userController.getMe();  
    setUser(response);  
    setLoading(false);  
    console.log(response)  
  } catch (error) {  
    console.log(error);  
    setLoading(false);  
  }  
}
```

Asegurate que estas enviando correctamente user

```
const data = {  
  user,  
  login,  
  logout: () => console.log('logout'),  
  updateUser: () => console.log('update user')  
}  
if (loading) return null;
```

Sesión del Usuario, persistente

Para crear la sesión persistente lo primero que vamos a hacer es crear la función de logout, lo hacemos llamando a la función que controla el token en asyncStorage con el método remove para eliminar el token del storage, utilizamos el estado de user para eliminar los datos de la sesión y pasamos el estado loading cómo false.

```
48     const logout = async () => {  
49         try {  
50             await storageController.removeToken();  
51             setUser(null);  
52             setLoading(false);  
53         } catch (error) {  
54             console.log(error);  
55             setLoading(false);  
56         }  
57     }  
58 }
```

Ahora en la función getSession hacemos la condicional, si token no existe llamamos logout y retornamos la función, hacemos una segunda condicional para comprobar que el token sea válido, en caso de que el token no sea válido se cierra la sesión, finalmente al pasar por las demás validaciones, llamamos login quien recibe el token.

```
21     const getSession = async () => {  
22         const token = await storageController.getToken();  
23         if (!token) {  
24             logout();  
25             setLoading(false);  
26             return;  
27         }  
28         if (tokenExpired(token)) {  
29             logout();  
30         } else {  
31             login(token);  
32         }  
33     }  
34 }
```

No olvides enviar la función de logout en data de manera correcta.

Dentro de las funciones que tenemos en AuthContext, nos falta update, esta será la función que actualice los datos de la sesión del usuario a nivel local.


```
const upDateUser = (key, value) => {  
  setUser({  
    ...user,  
    [key]: value  
  })  
}
```

Para cerrar la sesión, implementa un botón en Home (por ahora) que ejecute la función de logout.

```
src > pages > Home > @ index.jsx > [5] default  
1  import React from 'react';  
2  import { useAuth } from '../hooks/useAuth';  
3  import { Button } from 'antd';  
4  
5  const Home = () => {  
6    const { user, logout } = useAuth()  
7    return (  
8      <>  
9        <h1>Hola {user.username}</h1>  
10       <Button onClick={() => logout()}>Cerrar sesión</Button>  
11     </>  
12   );  
13 }  
14  
15  
16 export default Home;
```