



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, dirigete 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}

```
import axios from 'axios';
import { ENV } from '../utils/constants';

const register = async (username, email, password) => {
    return axios.post(`${ENV.API_URL}/${ENV.ENDPOINTS.REGISTER}`, {
    username,
    email,
    password,
    roles: ['user'],
    });

const loginF = async (email, password) => {
    return axios.post(`${ENV.API_URL}/${ENV.ENDPOINTS.LOGIN}`, {
        email,
        password,
    });

email,
    password,
    return axios.post(`${ENV.API_URL}/${ENV.ENDPOINTS.LOGIN}`, {
        email,
        password,
    });

export default {
    register,
    loginF,
    };
```

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

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

Desarrollo Web Mayo- Agosto 2024 Ing. Ana Sol Arteaga



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 userld 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 userld que recibe el valor del ld.

Tu solicitud quedaría de la siguiente manera.

```
src > services > __s users.js > ...
      import { jwtDecode } from "jwt-decode";
      import { ENV } from '../utils/constants';
  import { authFetch } from '../utils/authFetch';
      const getMe = async (token) => {
  7
  8
           try {
              const decoded = jwtDecode(token);
 10
              const userId = decoded.id;
               const url = `${ENV.API_URL}/${ENV.ENDPOINTS.USERS}/${userId}`
 11
               const response = await authFetch(url)
               return await response.json();
           } catch (error) {
               console.error(error)
 18
      export const usersService = {
           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 usersService.getMe(token);
```

.Obtendrás algo como esto.

Desarrollo Web Mayo- Agosto 2024 Ing. Ana Sol Arteaga



```
Obteniendo
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
wIjoxNzE3NjM0ODE3fQ.sHBkrmkDFz0wDD2xc
Hola desde authFetch

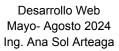
▶ Error al obtener el usuario: TypeE
```

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

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
       import { storageController } from "../services/token";
       import { tokenExpired } from "./tokenExpired";
       export const authFetch = async (url, params) => {
           const token = await storageController.getToken();
           const logout = () => {
               storageController.removeToken();
 10
 11
           if (!token) {
 12
               logout();
 13
           } else {
               const response = tokenExpired(token);
               console.log('Response:', response);
 17
       }
```

```
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.



```
Obteniendo
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6I;
wIjoxNzE3NjM2MDk3fQ.IjJqW-8xMXg3JUi7aqJ-yykL6t-
Response: false
```

Creada la función expired, podemos seguir construyendo la función authFetch, construyendo la solicitud enviando headers de Authorization de tipo Bearer y recibiendo el token. Lo que retorna esta solicitud es el fetch que recibe url y los parámetros.

```
rc > utils > Js authFetch.js > ...
      import { storageController } from "../services/token"
      import { tokenExpired } from "./tokenExpired"
      export const authFetch = async (url, params) => {
          const token = await storageController.getToken();
          const logout = () => {
              storageController.removeToken();
11
          if (!token) {
              logout();
          else {
              if (tokenExpired(token)) {
16
                  logout();
              else {
                  const options = {
20
                      ...params,
                      headers: {
                           ...params?.headers,
                           Authorization: `Bearer ${token}`
24
26
                  try {
                      return await fetch(url, options)
28
                  } catch (error) {
                      console.error(error)
```

Asegúrate que en user estés importando y llamando la función.

Desarrollo Web Mayo- Agosto 2024 Ing. Ana Sol Arteaga



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

```
Response

{ _id: '665e36d901527eab156259a0', username: 'Ana', password: '$2a$10$121KV0LumjVQYx9Hrj/ c00cXqAUwS2n01P29x8H.5R4A456F5FeKi', email: 'ana@arteaga.com', roles: Array(1), ...} i createdAt: "2024-06-03T21:34:17.545Z" email: "ana@arteaga.com" password: "$2a$10$121KV0LumjVQYx9Hrj/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

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 (looding) return pull:
```

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.



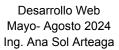
```
const logout = async () => {
48
49
              try {
50
                  await storageController.removeToken();
51
                  setUser(null);
                  setLoading(false);
52
53
              } catch (error) {
54
                  console.log(error);
55
                  setLoading(false);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

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.

```
const getSession = async () => {
21
              const token = await storageController.getToken();
22
23
              if (!token) {
24
                  logout();
25
                  setLoading(false);
26
                   return;
27
              if (tokenExpired(token)) {
28
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.





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