

Token Refresh with Context



In a React project do I have to implement a `useEffect` in every component to use refresh tokens to refresh the access token? And how do I handle it that the access token which has an expiration time of 10 minutes) doesn't run out while the user is doing something?

```
import React, { useState } from 'react';
import axios from 'axios';
import axiosInstance from '../api/AxiosInstance';
import LoginField from '../components/LoginField';
import { FaEye, FaEyeSlash } from 'react-icons/fa';
import { HiArrowNarrowRight } from "react-icons/hi";
import { useNavigate } from 'react-router-dom';

// This is the Login Page that displays whenever the site is pulled up
const LoginPage = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);

    try {
      // First Call to get ID of username
      const responseId = await axiosInstance.get(`/auth/user/${username}`);
      const id = responseId.data.user[0]?.id; // Zugriff auf die erste ID
      //console.log("Extracted ID:", id);

      if (!id) {
        throw new Error('Invalid user ID received');
      }

      const body = {
        user: id,
        password: password,
      };

      const responseLogin = await axiosInstance.post('/auth/login', body, {
        headers: { "Content-Type": "application/json" },
      });

      const {accessToken, refreshToken} = responseLogin.data;
      localStorage.setItem("userId", id);
    } catch (error) {
      setError(error.message);
    }
  };

  return (
    <div>
      <LoginField />
      <button type="button" onClick={handleSubmit}>Login</button>
    </div>
  );
};
```

```

localStorage.setItem("accessToken", accessToken);
localStorage.setItem("refreshToken", refreshToken);

// Navigate to main page on success
navigate('/main');

} catch (error) {
  console.log('Error', error);
  if (error.response) {
    setError(error.response.data?.message || 'Login failed. Please try again.');
```

```

  } else {
    setError('Network error or server is down.');
```

```

  }
}
};

return (
  <div className="flex flex-col h-screen items-center justify-center bg-cover bg-center">
    <div className="bg-white p-6 rounded-lg max-w-md w-full border-2 mt-36">
      
      <form onSubmit={handleSubmit} className="space-y-4">
        <LoginField placeholder="User" value={username} onChange={(e) =>
setUsername(e.target.value)} />
        <LoginField placeholder="PW" isPassword={true} value={password} onChange={(e) =>
setPassword(e.target.value)} />
        <div className="w-full flex justify-center">
          <button type="submit" className="bg-gray-400 text-white text-2xl py-2 hover:bg-gray-700
rounded-full w-16 h-16 flex justify-center items-center hover:font-bold transform hover:scale-105
transition-all duration-200">
            <HiArrowNarrowRight />
          </button>
        </div>
      </form>
      {error && (
        <div className="text-red-500 text-center mt-4">
          {error}
        </div>
      )}
    </div>
    <div className="text-gray-400 mt-auto mb-2">
      Impressum | Kontakt
    </div>
  </div>
);
};
```

```
export default LoginPage;
```

Is this possible with an axios interceptor?
import axios from 'axios';

```
const axiosInstance = axios.create({
  baseURL: 'https://api.contrude.eu',
  headers: {
    'Content-Type': 'application/json',
  },
});

export default axiosInstance;
```



Yes, you can handle token refreshing with an axios interceptor, which can automatically refresh the access token when it expires without requiring `useEffect` in each individual component. This centralizes the logic and makes your components cleaner.

Here's how you can implement this in your React project:

1. **Set up Axios Interceptors:** You can add a request interceptor to add the `accessToken` to every outgoing request. Then, add a response interceptor to catch any `401 Unauthorized` errors, which typically happen when the access token has expired.
2. **Refresh Token Logic:** When the access token has expired (i.e., a `401` status code is returned), you can use the `refreshToken` to obtain a new `accessToken` and retry the original request.

Here's an example of how to set up the interceptors:

Axios Instance with Interceptors

```
js

import axios from 'axios';

// Create an Axios instance
const axiosInstance = axios.create({
  baseURL: 'https://api.contrude.eu',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Add request interceptor to attach access token to headers
axiosInstance.interceptors.request.use(
  (config) => {
    const accessToken = localStorage.getItem('accessToken');
    if (accessToken) {
      config.headers['Authorization'] = `Bearer ${accessToken}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Add response interceptor to handle token refresh
axiosInstance.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response && error.response.status === 401) {
      // Token has expired, try refreshing it
      try {
```

```
const refreshToken = localStorage.getItem('refreshToken');
if (!refreshToken) {
  throw new Error('No refresh token found');
}

// Send a request to refresh the token
const response = await axios.post('/auth/refresh', { refreshToken });

// Get the new access token from the response
const { accessToken } = response.data;
localStorage.setItem('accessToken', accessToken);

// Retry the original request with the new access token
error.config.headers['Authorization'] = `Bearer ${accessToken}`;
return axiosInstance(error.config); // Retry original request
} catch (refreshError) {
  console.log('Error refreshing token', refreshError);
  // Handle token refresh failure (e.g., log the user out)
}
}

return Promise.reject(error);
};

export default axiosInstance;
```

Key Points:

1. **Request Interceptor:** This ensures that each request includes the `accessToken` in the `Authorization` header.
2. **Response Interceptor:** This handles the case when a `401` error occurs, meaning the `accessToken` has expired. It tries to refresh the token using the `refreshToken` and retries the original request with the new token.
3. **Retrying the Request:** The original request is retried with the updated access token in the headers.

Notes:

- **`/auth/refresh` Endpoint:** Make sure your backend has an endpoint for refreshing tokens (`POST /auth/refresh`), which takes the `refreshToken` and returns a new `accessToken`.
- **Error Handling:** If the refresh fails (e.g., the `refreshToken` is also expired), you should log the user out or redirect them to the login page.

Usage in Components:

Now, in your components like `LoginPage`, you don't need to worry about token expiry anymore. The axios instance automatically handles the access token refresh and retries failed requests.

For instance, in `LoginPage`, when you call `axiosInstance` for making API requests, the interceptor will manage the tokens for you, so you don't need to manually check token expiration and refresh it.

This approach centralizes token management and ensures a seamless user experience, especially for longer sessions.



Assuming the refresh token expires as well then the user wont be able to do anything anymore

right?



Yes, that's correct. If the refresh token expires as well, the user will no longer be able to obtain a new access token, which means they won't be able to perform any authenticated actions until they log in again.

In that case, you should handle the situation by redirecting the user to the login page or showing a session expired message. Here's how you can handle it:

1. **Handle Refresh Token Expiration:** If the refresh token has expired or is invalid (i.e., when the refresh token request fails), you can log the user out by removing all tokens from `localStorage` and redirect them to the login page.
2. **Prompt the User to Re-authenticate:** You can show an alert or a modal to notify the user that their session has expired and they need to log in again.

Example Code for Refresh Token Failure

You can update the `axiosInstance` response interceptor to handle the case when the refresh token request fails:

js

```

axiosInstance.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response && error.response.status === 401) {
      // Token has expired or is invalid, try refreshing it
      try {
        const refreshToken = localStorage.getItem('refreshToken');
        if (!refreshToken) {
          throw new Error('No refresh token found');
        }

        // Attempt to refresh the token
        const response = await axios.post('/auth/refresh', { refreshToken });

        // Get the new access token
        const { accessToken } = response.data;
        localStorage.setItem('accessToken', accessToken);

        // Retry the original request with the new access token
        error.config.headers['Authorization'] = `Bearer ${accessToken}`;
        return axiosInstance(error.config); // Retry original request
      } catch (refreshError) {
        console.log('Error refreshing token', refreshError);

        // Remove the expired tokens and redirect to login
        localStorage.removeItem('accessToken');
        localStorage.removeItem('refreshToken');

        // Optionally show a session expired message or alert
        alert('Your session has expired. Please log in again.');
```

```

        window.location.href = '/login'; // Redirect to login page
      }
    }

    return Promise.reject(error);
  }
);

```

```
}  
);
```

Key Changes:

1. **`catch` Block for Refresh Token Error:** If the refresh token request fails, the user is logged out by removing the ``accessToken`` and ``refreshToken`` from ``localStorage``.
2. **Session Expired Message:** You can display an alert or a modal informing the user that their session has expired.
3. **Redirection:** After logging the user out, you redirect them to the login page (``/login``) so they can authenticate again.

User Experience:

- If the refresh token has expired, the user will be redirected to the login page where they can log in again.
- This ensures that expired sessions are handled gracefully, and the user is not left stuck with expired tokens.