Authentication Setup

Authentication is a crucial part of integrating Replyke's API, as many features require a valid user identity. This guide will walk you through the necessary steps to manage authentication properly in your application.

Managing Tokens

When integrating with Replyke, developers must handle two types of tokens:

- **Access Token:** A short-lived token used to authenticate API requests. Expires every 30 minutes.
- **Refresh Token:** A long-lived token used to obtain new access tokens when they expire.

Handling the Refresh Token

Web Applications

- The refresh token is automatically stored in an **HTTP-only cookie**.
- No additional setup is required.
- It is automatically sent with requests, and developers do not need to handle it manually.
- When the user logs out, the token is invalidated automatically by Replyke.

Mobile Applications

- Since HTTP-only cookies do not work in mobile apps, developers must securely store the refresh token (e.g., using **Secure Storage** for React Native, **Encrypted Shared Preferences** for Android, or **Keychain Services** for iOS).
- On app launch, check if a refresh token is stored. If it exists, request a new access token instead of verifying the external user again.
- If there is no refresh token, make a request to verify the user's JWT if there is one and store the new refresh token securely.
- When the user logs out, make sure to remove the stored refresh token.

Handling the Access Token

- The access token must be included in the **Authorization header** for all API requests requiring authentication.
- Example:

```
GET /posts/{postId}/comments
Host: api.replyke.com
Authorization: Bearer YOUR_ACCESS_TOKEN
```

Handling Expired Access Tokens

Since access tokens expire every 30 minutes, the application should detect when a request fails due to an expired token and automatically request a new one. This ensures a seamless user experience.

Implementation Example

Below is an example of how to set up automatic token refreshing using **Axios**.

Generic Axios Interceptor (Framework-Agnostic)

```
import axios from 'axios';

const apiClient = axios.create({
  baseURL: 'https://api.replyke.com',
});

apiClient.interceptors.request.use(
  (config) => {
    config.headers['Authorization'] = `Bearer ${getAccessToken()}`;
    return config;
  },
  (error) => Promise.reject(error)
);

apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;
    if (error.response.status === 403 && !originalRequest._retry) {
      originalRequest._retry = true;
      const newAccessToken = await requestNewAccessToken();
      originalRequest.headers['Authorization'] = `Bearer ${newAccessToken}`;
      return apiClient(originalRequest);
    }
    return Promise.reject(error);
```

```
  }
);

export default apiClient;
```

React-Specific Axios Interceptor Using a Custom Hook

```
import { useEffect } from 'react';
import { axiosPrivate } from './axios';
import useAuth from '../hooks/auth/useAuth';

const useAxiosPrivate = () => {
  const { accessToken, requestNewAccessToken } = useAuth();

  useEffect(() => {
    const requestIntercept = axiosPrivate.interceptors.request.use(
      (config) => {
        if (config.headers['Authorization']) return config;
        config.headers['Authorization'] = `Bearer ${accessToken}`;
        return config;
      },
      (error) => Promise.reject(error)
    );

    const responseIntercept = axiosPrivate.interceptors.response.use(
      (response) => response,
      async (error) => {
        const prevRequest = error?.config;
        if (error?.response?.status === 403 && !prevRequest?.sent) {
          prevRequest.sent = true;
          const newAccessToken = await requestNewAccessToken?.();
          prevRequest.headers['Authorization'] = `Bearer ${newAccessToken}`;
          return axiosPrivate(prevRequest);
        }
        return Promise.reject(error);
      }
    );

    return () => {
      axiosPrivate.interceptors.request.eject(requestIntercept);
      axiosPrivate.interceptors.response.eject(responseIntercept);
    };
  }, [accessToken, requestNewAccessToken]);

  return axiosPrivate;
};

export default useAxiosPrivate;
```

Summary

- **Web apps:** Refresh tokens are handled automatically via cookies.
- **Mobile apps:** Refresh tokens must be stored securely and used to obtain new access tokens when needed.
- All API requests which require an authenticated user **must include the latest access token.**
- For a smooth user experience, **requests should automatically retry with a new access token when the existing one expires.**

Last updated on May 6, 2025