STRAVA Developers

Authentication

Table of Contents

Introduction

Strava uses OAuth2 for authentication to the V3 API. OAuth allows external applications to request authorization to a user's data. It allows users to grant and revoke API access on a per-application basis and keeps users' authentication details safe.

All developers need to register their application before getting started. A registered application will be assigned a client ID and client secret. The secret is used for authentication and should never be shared.

OAuth Overview

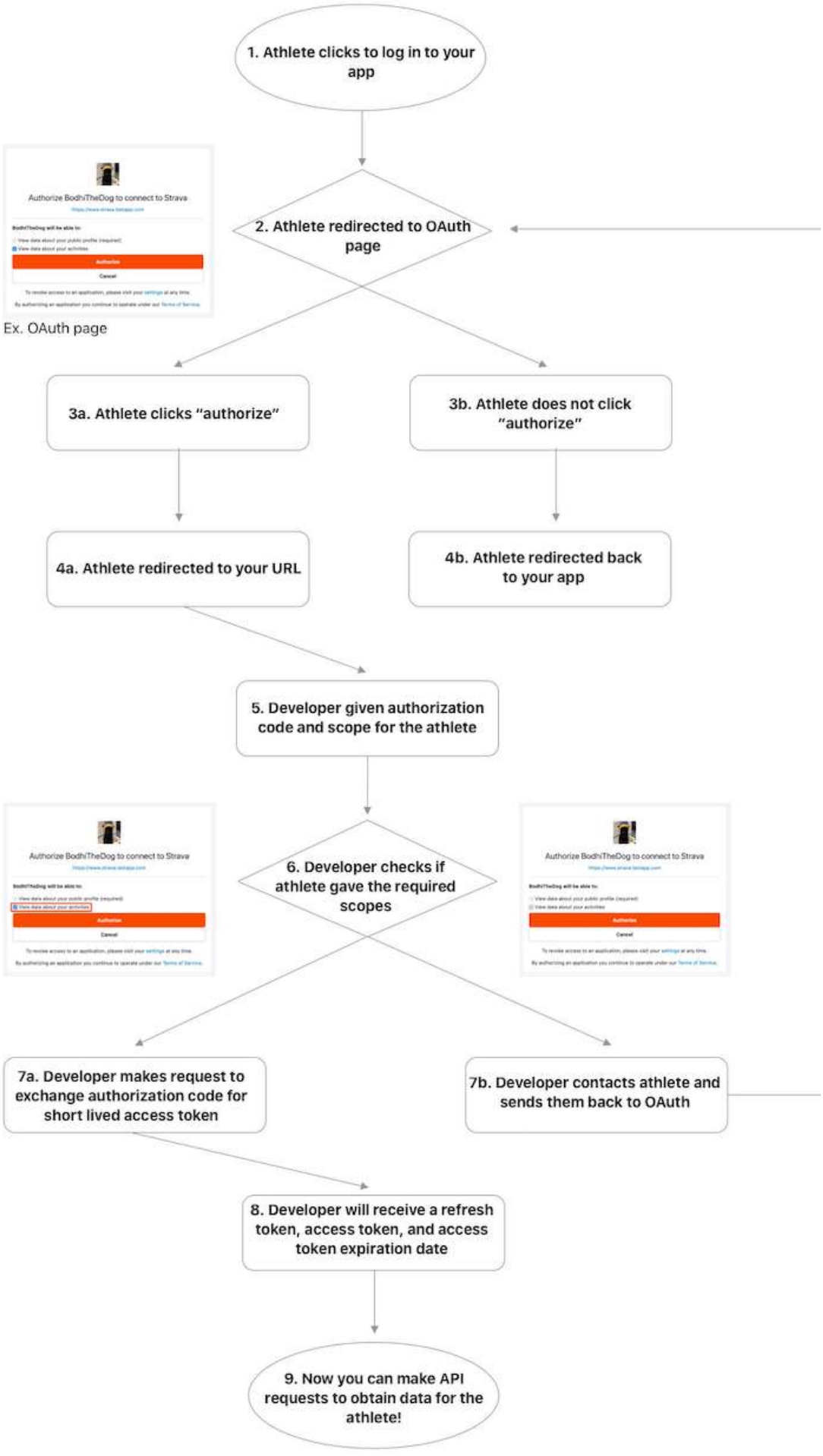
When OAuth is initiated, the user is prompted by the application to log in on the Strava website and to give consent to the requesting application. A user can opt out of the scopes requested by the application.

After the user accepts or rejects the authorization request, Strava redirects the user to a URL specified by the application. If the user authorized the application, the URL query string will include an authorization code and the scope accepted by the user. Apps should check which scopes a user has accepted. Applications complete the authorization process by exchanging the authorization code for a refresh token and short-lived access token.

Access tokens are used by applications to obtain and modify Strava resources on behalf of the authenticated athlete. *Refresh tokens* are used to obtain new access tokens when older ones expire.

Note that Google Sign-in will not work for applications using a mobile webview. See Google's blog post for further information and ways to work around that limitation.

Step-by-step guide:


1. Athlete clicks to log in to your app
2. Athlete redirected to OAuth page
Authorize BodhiTheDog to connect to Strava
Ex. OAuth page
3a. Athlete clicks "authorize"
3b. Athlete does not click "authorize"
4a. Athlete redirected to your URL
4b. Athlete redirected back to your app
5. Developer given authorization code and scope for the athlete
6. Developer checks if athlete gave the required scopes
Authorize BodhiTheDog to connect to Strava
Authorize BodhiTheDog to connect to Strava
7a. Developer makes request to exchange authorization code for short lived access token
7b. Developer contacts athlete and sends them back to OAuth
8. Developer will receive a refresh token, access token, and access token expiration date
9. Now you can make API requests to obtain data for the athlete!

Requesting Access

To initiate the flow, applications must redirect the user to Strava's authorization page. The authorization endpoint is different for mobile and web applications.

Web applications

Redirect the user to `GET https://www.strava.com/oauth/authorize` .

Mobile applications

Our authentication flows are slightly different for Android and iOS due to App Store and Google Play Store guidelines and recommendations. Below you'll find code samples for both platforms. Here are some details about mobile OAuth:

- If a user does not have the Strava app installed, they should be redirected to mobile web to complete OAuth.
- Mobile OAuth supports refresh tokens and short-lived access tokens. Mobile OAuth does not support forever access tokens.
- Users must have version 75.0 or later of the Strava app installed in order to take advantage of mobile OAuth. Users without the Strava app or with an older app version will be brought to a web endpoint where they can authorize your app.

Android

Android apps can use an Implicit Intent to redirect users to the `GET https://www.strava.com/oauth/mobile/authorize` endpoint. The Strava app should open automatically if the user has it installed.

Sample code:

```
val intentUri =
Uri.parse("https://www.strava.com/oauth/mobile/authorize")
        .buildUpon()
        .appendQueryParameter("client_id", "1234321")
        .appendQueryParameter("redirect_uri",
"https://www.yourapp.com")
        .appendQueryParameter("response_type", "code")
        .appendQueryParameter("approval_prompt", "auto")
        .appendQueryParameter("scope", "activity:write,read")
        .build()

val intent = Intent(Intent.ACTION_VIEW, intentUri)
startActivity(intent)
```

iOS

If the user does not have Strava installed, your app should use `SFAuthenticationSession` or `ASWebAuthenticationSession` , depending on which versions of iOS your app supports. If your app is linked

on or after iOS 9.0, you must add `strava` in you app's Info.plist file. It should be added under the `LSApplicationQueriesSchemes` key. Failure to do this will result in a crash when calling `canOpenUrl:` .

You should check first if your app can open the `appOAuthURLStravaScheme` . If it can, the Strava app is installed and should handle the authentication. You can proceed by opening that same URL.

If your app is not able to open that URL, proceed with using the `webOAuthUrl` . You should use `SFAuthenticationSessions` or `ASWebAuthenticationSession` to prevent the user from leaving your application to authenticate. Ensure that the `callbackURLScheme` is a registered url scheme in your application.

Sample code:

```
private var authSession: SFAuthenticationSession?

let appOAuthUrlStravaScheme = URL(string:
"strava://oauth/mobile/authorize?
client_id=1234321&redirect_uri=YourApp%3A%2F%2Fwww.yourapp.com%2Fen
US&response_type=code&approval_prompt=auto&scope=activity%3Awrite%2

let webOAuthUrl = URL(string:
"https://www.strava.com/oauth/mobile/authorize?
client_id=1234321&redirect_uri=
YourApp%3A%2F%2Fwww.yourapp.com%2Fen-
US&response_type=code&approval_prompt=auto&scope=activity%3Awrite%2

@IBAction func authenticate() {
    if UIApplication.shared.canOpenURL(appOAuthUrlstravaScheme) {
        UIApplication.shared.open(appOAuthUrlstravaScheme, options:
[:])
    } else {
        authSession = SFAuthenticationSession(url: webOAuthUrl,
callbackURLScheme: "YourApp://") { url, error in

        }

        authSession?.start()
    }
}
```

Details About Requesting Access

On both web and mobile the authorization page will prompt the user to grant your application access to their data. Scopes requested by the application are shown as checked boxes, but the user may opt out of any requested scopes. If an application relies on specific scopes to function properly, the application should make that clear before and after authentication.

client_id required integer, in query	The application's ID, obtained during registration.

redirect_uri required string, in query	URL to which the user will be redirected after authentication. Must be within the callback domain specified by the application. `localhost` and `127.0.0.1` are white-listed.
response_type required string, in query	Must be `code` .
approval_prompt string, in query	`force` or `auto` , use `force` to always show the authorization prompt even if the user has already authorized the current application, default is `auto` .
scope required string, in query	Requested scopes, as a comma delimited string, e.g. "activity:read_all,activity:write". Applications should request only the scopes required for the application to function normally. The scope activity:read is required for activity webhooks. • `read` : read public segments, public routes, public profile data, public posts, public events, club feeds, and leaderboards • `read_all` :read private routes, private segments, and private events for the user • `profile:read_all` : read all profile information even if the user has set their profile visibility to Followers or Only You • `profile:write` : update the user's weight and Functional Threshold Power (FTP), and access to star or unstar segments on their behalf • `activity:read` : read the user's activity data for activities that are visible to Everyone and Followers, excluding privacy zone data • `activity:read_all` : the same access as `activity:read` , plus privacy zone data and access to read the user's activities with visibility set to Only You • `activity:write` : access to create manual activities and uploads, and access to edit any activities that are visible to the app, based on activity read access level

state string, in query	Returned in the redirect URI. Useful if the authentication is done from various points in an app.

Token Exchange

Strava will respond to the authorization request by redirecting the user agent to the `redirect_uri` provided.

If access is denied, `error=access_denied` will be included in the query string.

If access is accepted, `code` and `scope` parameters will be included in the query string. The `code` parameter contains the authorization code required to complete the authentication process. `code` is short lived and can only be used once. The application must now call the `POST https://www.strava.com/oauth/token` with its client ID and client secret to exchange the authorization code for a refresh token and short-lived access token.

The `state` parameter will be always included in the response if it was initially provided by the application.

Request Parameters

client_id required integer, in query	The application's ID, obtained during registration.
client_secret required string, in query	The application's secret, obtained during registration.
code required string, in query	The `code` parameter obtained in the redirect.
grant_type required string, in query	The grant type for the request. For initial authentication, must always be "authorization_code".

Example cURL Request

```
curl -X POST https://www.strava.com/api/v3/oauth/token \
  -d client_id=ReplaceWithClientID \
  -d client_secret=ReplaceWithClientSecret \
  -d code=ReplaceWithCode \
  -d grant_type=authorization_code
```

Response Parameters

A refresh token, access token, and access token expiration date will be issued upon successful authentication.

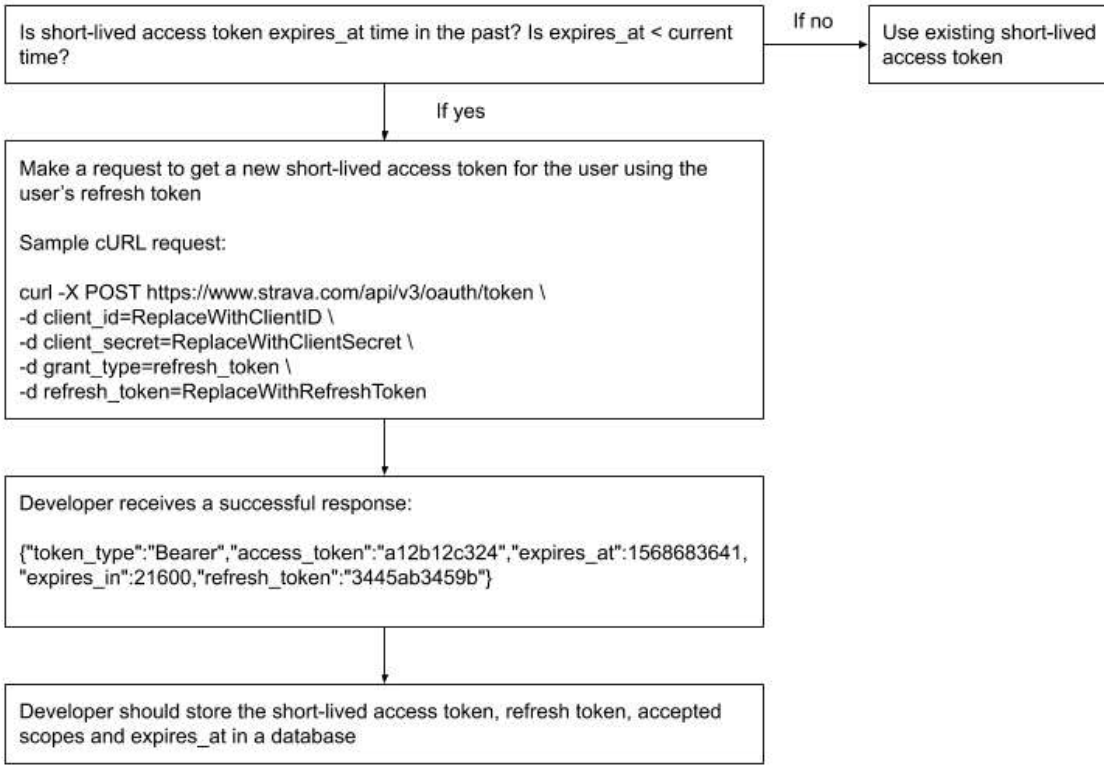
expires_at integer	The number of seconds since the epoch when the provided access token will expire
expires_in integer	Seconds until the short-lived access token will expire
refresh_token string	The refresh token for this user, to be used to get the next access token for this user. Please expect that this value can change anytime you retrieve a new access token. Once a new refresh token code has been returned, the older code will no longer work.
athlete string	A summary of athlete information

Example Response

```
{
  "token_type": "Bearer",
  "expires_at": 1568775134,
  "expires_in": 21600,
  "refresh_token": "e5n567567...",
  "access_token": "a4b945687g...",
  "athlete": {
    #{summary athlete representation}
  }
}
```

Refreshing Expired Access Tokens

Access tokens expire six hours after they are created, so they must be refreshed in order for an application to maintain access to a user's resources. Every time you get a new access token, we return a new refresh token as well. If you need to make a request, we recommend checking to see if the short-lived access token has expired. If it has expired, request a new short-lived access token with the last received refresh token.



To refresh an access token, applications should call the `POST https://www.strava.com/oauth/token` endpoint, specifying `grant_type: refresh_token` and including the application's refresh token for the user as an additional parameter. If the application has an access token for the user that expires in more than one hour, the existing access token will be returned. If the application's access tokens for the user are expired or will expire in one hour (3,600 seconds) or less, a new access token will be returned. In this case, both the newer and older access tokens can be used until they expire.

A refresh token is issued back to the application after all successful requests to the `POST https://www.strava.com/oauth/token` endpoint. The refresh token may or may not be the same refresh token used to make the request. Applications should persist the refresh token contained in the response, and always use the most recent refresh token for subsequent requests to obtain a new access token. Once a new refresh token is returned, the older refresh token is invalidated immediately.

Request Parameters

client_id required integer, in query	The application's ID, obtained during registration.
client_secret required string, in query	The application's secret, obtained during registration.
grant_type required string, in query	The grant type for the request. When refreshing an access token, must always be "refresh_token".

refresh_token required string, in query	The refresh token for this user, to be used to get the next access token for this user. Please expect that this value can change anytime you retrieve a new access token. Once a new refresh token code has been returned, the older code will no longer work.

Example cURL Request

```
curl -X POST https://www.strava.com/api/v3/oauth/token \
  -d client_id=ReplaceWithClientID \
  -d client_secret=ReplaceWithClientSecret \
  -d grant_type=refresh_token \
  -d refresh_token=ReplaceWithRefreshToken
```

Response Parameters

access_token string	The short-lived access token
expires_at integer	The number of seconds since the epoch when the provided access token will expire
expires_in integer	Seconds until the short-lived access token will expire
refresh_token string	The refresh token for this user, to be used to get the next access token for this user. Please expect that this value can change anytime you retrieve a new access token. Once a new refresh token code has been returned, the older code will no longer work.

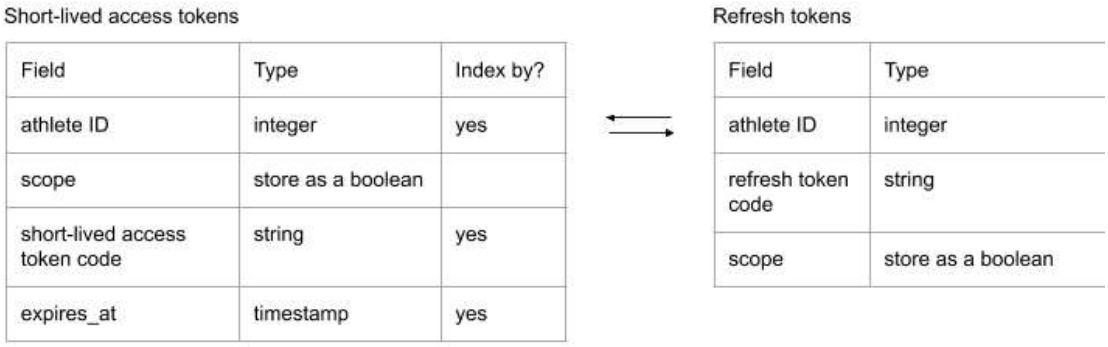
Example Response

```
{
  "token_type": "Bearer",
  "access_token": "a9b723...",
  "expires_at":1568775134,
  "expires_in":20566,
  "refresh_token":"b5c569..."
}
```

How to Store Short-Lived Access Tokens and Refresh Tokens

- Storing the scopes your athletes accept is great in case you get unexpected results (for example: why am I not getting activities for this user?)
- In general, we recommend storing short-lived access tokens and refresh tokens in separate tables

- Note: This is just a recommendation and may not be the best implementation given your use case.



Short-lived access tokens

Field	Type	Index by?
athlete ID	integer	yes
scope	store as a boolean	
short-lived access token code	string	yes
expires_at	timestamp	yes

Refresh tokens

Field	Type
athlete ID	integer
refresh token code	string
scope	store as a boolean

Accessing the API Using an Access Token

Applications use unexpired access tokens to make resource requests to the Strava API on the user's behalf. Access tokens are required for all resource requests, and can be included by specifying the `Authorization: Bearer #{access_token}` header. For instance, using HTTPie:

```
$ http https://www.strava.com/api/v3/athlete 'Authorization: Bearer
83ebeabdec09f6670863766f792ead24d61fe3f9'
```

Example cURL Request

```
curl -G https://www.strava.com/api/v3/athlete -H "Authorization:
Bearer ReplaceWithAccessToken"
```

Deauthorization

Applications can revoke access to an athlete's data. This will invalidate all refresh tokens and access tokens that the application has for the athlete, and the application will be removed from the athlete's apps settings page. All requests made using invalidated tokens will receive a 401 Unauthorized response.

The endpoint is `POST https://www.strava.com/oauth/deauthorize` .

access_token required string, in query	Responds with the access tokens that were revoked.

How to Get Support

If you have questions, please check our developer community hub. Remember, **never** share access tokens, refresh tokens, authorization codes, or your client secret in a public forum.