OAuth2 authorization code flow

Sign app users in with OAuth2 against Exponential - registering the client in oAuth admin, the authorize page and its exact redirect address, the oauth/authorize policy, exchanging the code at /api/auth/v1/oauth/token...

Source file: `doc/eng-us/help/mobile-app-developers/auth/oauth2-authorization-code-flow.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/mobile-app-developers/auth/oauth2-authorization-code-flow.md

This page is for app developers whose users sign in with their own Exponential account, so that the app reads and writes as each user. It walks through the flow as the code implements it, and names the parts that differ from what current OAuth guidance for mobile apps expects.

Contents

1. The flow at a glance
2. Step 1: register the client
3. Step 2: the authorize page
4. Step 3: exchange the code
5. Step 4: call the API
6. Refreshing tokens
7. Limits of the implementation

The flow at a glance

```
app  -> system browser: https://example.com/oauth/authorize?client_id=...&redirect_uri=...&respon
se_type=code
user -> signs in on the site and allows the app
site -> redirect_uri?code=<code>&expires_in=<seconds>
back end -> POST /api/auth/v1/oauth/token (grant_type=authorization_code, client_id, client_secre
t, code, redirect_uri)
         <- {"access_token": "...", "refresh_token": "...", "expires_in": 3600}
app or back end -> GET /api/ezp/v2/... with Authorization: OAuth <access_token>
```

Because the token endpoint needs the client secret, the exchange belongs on a server you run (A back end for your app).

Step 1: register the client

An administrator opens **Setup > oAuth admin** (`/oauthadmin/list`), tab **Applications**, and creates one. It gets a random client identifier and secret and the **redirect address** (end point) of the application. The secret is folded away on the application's page until **Show the secret** is pressed (Personal API keys, section 7).

The authorize page compares the `redirect_uri` of a request with the registered end point **exactly**, character for character (`ezpRestClient::isEndPointValid()`). Register the address

exactly as the app or back end will send it, and without a query string: the site appends `?code=...` to it.

Step 2: the authorize page

Open the authorize page in the system browser (an in-app browser tab, not an embedded web view, so the app never sees the user's password):

```
https://example.com/oauth/authorize?client_id=<client id>&redirect_uri=<redirect uri>&response_type=code
```

- A visitor who is not signed in is asked to sign in first, then comes back to the page.
- The user allows the app. Since 6.0.15 the user needs the policy **OAuth / Authorize** (`oauth/authorize`), optionally limited by siteaccess; without it the site redirects with `error=access_denied` (OAuth authorize policy). Give it to the roles of the people who use the app.
- Errors before the redirect address is validated (unknown client, wrong address) answer `400` on the site and do not redirect; later errors go to the redirect address as `error=...`.
- On success the site redirects to `<redirect uri>?code=<code>&expires_in=<seconds>`.

Step 3: exchange the code

Shell

```
curl -s https://example.com/api/auth/v1/oauth/token \
     -d grant_type=authorization_code -d client_id='<client id>' -d client_secret='<client secret>' \
     -d code='<code>' -d redirect_uri='<redirect uri>'
```

The answer carries `access_token`, `refresh_token` and `expires_in` (the value of `rest.ini [OAuthSettings] TokenTTL`, 3600 seconds by default). The code itself also expires after `TokenTTL`.

Step 4: call the API

```
Authorization: OAuth <access_token>
```

The REST layer also accepts `Authorization: Bearer <access_token>` for OAuth tokens, and an `oauth_token` form parameter of a POST body. An `oauth_token` query parameter is refused unless the site sets `rest.ini [OAuthSettings] AllowTokenInQuery=enabled`: URLs are logged by servers and proxies. Use the header.

Refreshing tokens

When a call answers `401` or the `expires_in` time has passed, ask for a new token:

Shell

```
curl -s https://example.com/api/auth/v1/oauth/token \
     -d grant_type=refresh_token -d client_id='<client id>' -d client_secret='<client secret>' \
     -d refresh_token='<refresh token>'
```

The answer has the same three keys. Store the new refresh token in place of the old one: a refresh token lasts `rest.ini [OAuthSettings] RefreshTokenTTL` (30 days by default) from the refresh that issued it, and it stops working when the user is disabled or locked. Refresh once; if that fails, send the user through the authorize page again. Refreshing also needs the client secret, so it too runs on the back end.

Limits of the implementation

Checked in `kernel/private/rest/classes/controllers/oauth_token.php` and `kernel/private/modules/oauth/authorize.php`:

Topic	What the code does	What to do
Grants	`authorization_code` and `refresh_token` only	no password or client credentials grant
Client authentication	`client_secret` is required at the token endpoint	keep it on a server
PKCE	not supported (`code_verifier` is not read)	the back end protects the exchange
`state`	not passed back ("state, not implemented yet")	keep a one-time value in the app or back end and check it against the session that started the flow
`scope`	read from the request; returning a different scope is "not implemented yet"	rely on the user's roles, not on OAuth scopes
`response_type=token`	the code also handles it and puts `access_token` and `refresh_token` in the redirect's query string	do not use it from an app: tokens in URLs leak into logs and history

None of this is unusual for an older OAuth2 implementation; it only means that a public mobile client should not talk to the token endpoint directly.

Related pages

- A back end for your app
- REST API v1 and v2: OAuth2 tokens
- OAuth and API keys
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.