

Profile

The profile page and the API tokens of Derafu Auth

Anonymous · 10/10/2026

Profile

Every provider (Keycloak, database and htpasswd) brings the same account pages, with templates that a site can override. Import the `auth-<provider>-routes.yaml` of the provider (see [Configuration](#)).

What the site configures

The renderer of the site needs the templates of the package, the extension with its Twig functions (AuthExtension) and the extension of the app variable (AppExtension, of [derafu/http](#)). The renderer of [derafu/foundation](#) has AppExtension and not the other two. A site that defines its own renderer lists all of them:

```
Derafu\Renderer\Contract\RendererInterface:
  factory: ['Derafu\Renderer\Factory\RendererFactory', 'create']
  arguments:
    $options:
      extensions:
        - '@Derafu\Http\Twig\AppExtension'
        - '@Derafu\Auth\Twig\AuthExtension'
        # ...the other extensions of the site.
      paths:
        - '%kernel.project_dir%/vendor/derafu/auth/resources/templates'
        # ...the other paths of the site.
```

Routes

Route	Path	What it does
auth_login	GET /auth/login	The login. With Keycloak it starts the flow; with the others it is the form.
auth_profile	GET /auth/profile	The profile, in tabs: Data , API and Session .
auth_token_create	POST /auth/profile/tokens	Makes an API token (Keycloak).
auth_token_revoke	POST /auth/profile/tokens/{id}/revoke	Revokes one token (Keycloak).

The POST routes are protected with CSRF and with the same origin. The form that makes a token is a form of `derafu/form`, so the site needs the CSRF token manager and the session of `derafu/csrf` and `derafu/session`, which a site that uses `derafu/foundation` already has.

Tabs and links to a card

The tabs have the ids `data-tab`, `api-tab` and `session-tab`, and their panes `data`, `api` and `session`. Each card has an id of the form `<pane>_<name>-card`:

Card	Id
Your data	<code>data_user-card</code>
How to use the API	<code>api_usage-card</code>
Tokens of the API	<code>api_tokens-card</code>
PHP session	<code>session_php-card</code>
What the provider says about the session	<code>session_provider-<n>-card</code> (<n> is the position of the section)

A link to `/auth/profile#api` opens the tab of the API, and `/auth/profile#api:tokens` opens it and goes to the card of the tokens. The page redirects there after it makes or revokes a token. The tab that opens, and the deep links, are the work of Tabs of `derafu-js` (the global bootstrap is required): the site loads it and starts it.

```
<script src="https://cdn.jsdelivr.net/npm/derafu-js@0.1.4/src/tabs.min.js"></script>
<script>
  document.addEventListener('DOMContentLoaded', function () {
    Tabs.init();
  });
</script>
```

Without it the page opens in **Data** and the tabs still change when they are clicked.

The profile has the button to log out (a POST to `/auth/logout`), so a site can limit its header to a link to the login, or to the profile when there is a user.

Twig

The `app` variable (`app.user`, from `derafu/http`) and these functions are available in the templates:

- `is_granted('role')`: whether the user has the role.
- `login_path()`, `profile_path()` and `logout_path()`.

To put the account in a header, link to the login when `app.user` is `null` and to the profile when there is a user: the profile has the button to log out.

```
{% if app.user %}
  <a href="{{ profile_path() }}">{{ app.user.name ?? app.user.identity }}</a>
{% else %}
  <a href="{{ login_path() }}">Log in</a>
{% endif %}
```

Overriding the templates

The templates are the defaults of the package. A site overrides one by having a file with the same name in its own templates: `auth/profile.html.twig`, or one of the partials of the tabs. To add tabs, override `auth/profile/_tabs-extra.html.twig` and `auth/profile/_panes-extra.html.twig`. To change how the API is explained, override `auth/profile/_api-basic.html.twig`, `_api-bearer.html.twig` or `_api-docs.html.twig`.

API tokens (Keycloak)

The API tab lets the user make tokens to call the API from a script. Each token is a Keycloak **offline token**, sent as `Authorization: Bearer <token>` (see [API](#)).

- The user can have as many as needed. The list shows, for each one, when it was created, last used and expires, how long it lasts (**Duration**, in days), how long is left (**Remaining**, in days, or *Expired*) and the browser that Keycloak recorded (a program has none: it shows `–`). Each is revoked **one by one** (there is no revoke-all). The list has no address: Keycloak records the one of the server that asked for the token, which is not the one of the user.
- A token is shown **only once**, when it is made, in a page that is not cached, and it is never kept in the session. It has no name. The page has a summary (when it was made, when it expires and how long it lasts) and the token hidden like a password, with the buttons to show it and to copy it. They are the ones of `derafu/form`, and they work with `fields.min.js` and `ui.min.js` of `derafu-js`, which the site loads (`ui` uses `bootbox` or `Notyf` for its notice). Without JavaScript the page shows the token in a text area, to copy it.
- The example of the call (`curl -H "Authorization: Bearer TOKEN" https://...`) has the address of the site: with Keycloak, the one of `AUTH_KEYCLOAK_WEB_REDIRECT_URI`; with another provider, the one of the request (its `Host`).
- To make one the user types the password again (and the OTP, if the user has one). The site uses it only to ask Keycloak for the token: it is not stored. It is the password of the user in the realm of the site, and the form says so. A user that logs in through another provider (identity brokering) has no password in that realm until it creates one there.
- Each token is an independent offline session, so revoking one does not close the web session or the other tokens.

When the token is not made

Keycloak says the same for a password that is not valid, a code of the second factor that is not valid

and a code that is missing. The form works out what is more likely from what the account API says of the user (whether it has a second factor, which needs the role `view-profile`), and says what Keycloak does tell as it is:

Situation	The form says
The user is disabled	The user is disabled.
The user has actions pending in Keycloak (a password to change, an email to verify...)	The actions are pending: complete them and try again.
The user has a second factor and no code was given	The password is not valid, or the code of the second factor is missing (the user has one).
The user has a second factor and a code was given	The password or the code of the second factor is not valid.
The user has no second factor	The password is not valid.

If the account API does not answer, the message depends on whether a code was given: with it, the one of the password or the code; without it, the one of the password.

What to configure in Keycloak

- **Client:** *Direct access grants* and the scope `offline_access`: see [Keycloak](#).
- **Users:** the roles `offline_access` and, of the client account, `view-profile` and `manage-account` (the defaults of a realm). If the client has *Full scope allowed* off, those two also go in its scope: see [Keycloak](#).
- **Realm:** *Revoke Refresh Token* **off**, and **Offline Session Idle** raised (see below: it is how long a token lasts).
- **Keycloak:** the list of the tokens is the one of the account API of Keycloak. It lists the offline sessions in Keycloak 26.8.0, the version of the tests of the package; in Keycloak 26.4.7 it does not, and the profile shows no tokens.
- **Audience:** the access token that the API gets must have the client in its audience, or the API refuses the token: see [API](#).

How long a token lasts

The token that the user makes and gives to its client is an **offline token**. The client sends it in every request. The API never shows the client anything else: the short-lived **access token** that the API gets from Keycloak with the offline token is internal, and the API caches it.

A token lasts the **Offline Session Idle** of the realm, counted from the day it is made. Using the token does not extend it. Keycloak writes in the token its expiration (the day it is made plus that time), and when that date passes it rejects the token, even if the token is used every day. The offline session in Keycloak also expires if the token is not used for that time, and each use restarts that count.

Each time Keycloak exchanges an offline token it returns a new one with a renewed expiration. The

API has no place to give it to the client, so it discards it: the token of the client keeps its expiration.

The setting is in the realm (*Realm settings* → *Sessions* → *Offline Session Idle*) and in each client (*Client Offline Session Idle*, which uses the one of the realm when it is empty). The tokens last the shorter of the two: the one of the realm is the maximum, and the one of the client can only shorten the tokens of that client. Set both to the same value: for example, both to 3650 days. *Offline Session Max* is off by default and stays off.

Realm	Client	Life of the token
3650 days	3650 days	3650 days
3650 days	100 days	100 days
30 days	3650 days	30 days

The limit of the year 2038

Keycloak stores the expiration as a 32-bit number of seconds since 1970, whose maximum is January 19, 2038 ([issue #11053](#), *Support 2038 timestamps in Keycloak*; it is planned for Keycloak 27). If the day the token is made plus *Offline Session Idle* goes past that date, the expiration overflows and the token is not valid. So no token can last past January 2038, and the longest value that works depends on the day: it is the time left until that date.

A value of 3650 days (315360000 seconds) works for the tokens made until January 2028. After that date, set the time that is left until 2038. A value of 1825 days (5 years) works until 2033. The tokens that were already made keep the expiration that they have.

Last updated on 10/10/2026