

Keycloak

The Keycloak provider of Derafu Auth

Anonymous · 08/10/2026

Keycloak

The Keycloak provider is the OpenID Connect authorization code flow, with PKCE, done by `KeycloakAuthentication` (the authentication middleware of Mezzio). Its libraries are not required by the package, install them in the application that uses it:

```
composer require league/oauth2-client firebase/php-jwt
```

`league/oauth2-client` does the OAuth2 requests (and brings Guzzle, which reads the keys of the realm) and `firebase/php-jwt` verifies the tokens. If one is missing the provider says what to install. The variables are in [Configuration](#).

The client of Keycloak

Configure the client of the application in the realm:

Setting	Value
Client authentication	On (a confidential client, with a secret).
Standard flow	On. The other flows can be off.
Valid redirect URIs	The callback of the application: <code>AUTH_KEYCLOAK_REDIRECT_URI</code> .
Valid post logout redirect URIs	The page that follows the logout: <code>AUTH_KEYCLOAK_POST_LOGOUT_REDIRECT_URI</code> , or <code>AUTH_LOGOUT_REDIRECT_PATH</code> in the site of the redirect URI. Without it Keycloak shows an error page at the logout (or set <code>AUTH_KEYCLOAK_END_SESSION=false</code>).
PKCE code challenge method	<code>S256</code> , so Keycloak requires it. The package always sends it.

The flow

1. **The user asks for a protected page** and is not logged in. The authentication remembers the page (its path and query only, never the host), and sends the user to Keycloak with:

- the `state`, against CSRF,
- the `nonce`, that the ID token must have,
- the PKCE challenge (`S256`).

Everything the login needs when the user comes back (the `state`, the `nonce` and the PKCE verifier) is kept in the session.

2. **The user logs in on Keycloak**, which sends the user back to the callback with a `code`.

3. **The authentication handles the callback**, before the router's handler runs and whether the paths are protected or not, and only the authentication does it, because a `code` can be used once:

- it checks the `state` and exchanges the `code`, with the PKCE verifier;
- it verifies the ID token and the access token (see below);
- it reads the user, stores the tokens and the user in the session, and renews the session id.

4. **KeycloakController only redirects** the user to the page that was remembered, or to `AUTH_LOGIN_REDIRECT_PATH`.

If Keycloak answers with an error (the user said no, for example), or the callback is not the one of a login that this session started, or the `code` is not accepted, the authentication raises an `AuthenticationException` with the code 400, and its message is translatable. Nobody is logged in and the session is the same.

The login path (the callback) and the logout path are public: a protected path that includes them (`/auth` protecting `/auth/callback`, for example) does not turn them away.

The tokens

Both tokens are verified, not only read:

- **The signature**, with the keys that the realm publishes (`/protocol/openid-connect/certs`).
- **The issuer** (`iss`): the URL of the realm, or `AUTH_KEYCLOAK_ISSUER`.
- **The expiration**, with a leeway of 60 seconds for the clocks.
- **The ID token**: its audience (`aud`) is the client, and its `nonce` is the one of the login.
- **The access token**: it was given to this client (`azp`).
- **The user**: the user of the user info endpoint is the one of the access token, and the one of the ID token.

The keys of the realm are cached

Reading the keys in every request that verifies a token (the login, and every refresh of the access

token) would make each of them depend on Keycloak answering at that moment. So `KeycloakTokenVerifier` uses a cache pool (PSR-6) if the application has one (`Psr\Cache\CacheItemPoolInterface`, which [derafu/foundation](#) provides):

- The keys are cached **for an hour**.
- If a token comes signed with a key that is not known (Keycloak **rotated** its keys), the keys are read again, **no more than ten times a minute**, so a stream of forged tokens does not become a stream of requests to Keycloak.
- A token signed with a key that the realm does not have is rejected.

Without a pool the keys are read once per verifier (once per request), and again if a token comes with a key that is not known. The client that reads them is made from the HTTP options of the configuration (the verification of the certificate and the timeouts), not the one of the application.

The user

The identity is the sub, and the details are the claims of the access token and of the user info. The user has the standard fields of OpenID Connect (`getName()`, `getEmail()`...), with null for a claim that is not there, and is made by the [factory of users](#). Its roles are:

- the roles of the realm (`realm_access.roles`), and
- the roles of the client of the application (`resource_access.<client_id>.roles`).

The roles that the user has in other clients of the realm (the `account` client has `view-profile`, for example) are **not** roles of the application. More in [Users](#).

Refresh of the tokens

The session keeps the user that Keycloak gave at the login, and the authentication asks Keycloak again when the access token expires or, if `AUTH_REFRESH_INTERVAL_SECONDS` is set and it is before, when that many seconds passed since the last time (what happens first). It gets new tokens with the refresh token, verifies them, asks for the user info, and updates the session and its user: the roles are the ones of that moment. Keeping the roles up to date, what happens if Keycloak can not be asked, and the realms that renew the refresh tokens are in [Security](#).

- Keycloak does not accept the refresh token (`invalid_grant`: the session of the user was ended there, the user was disabled, or the token expired): the session of the application is closed and the user is sent to Keycloak again.
- Keycloak can not be asked, or refuses for another reason: the session is kept, the user is not let in for that request, and the next request asks again.
- A token response without an expiry has no clock of its own: the interval, or 5 minutes if there is none, says when to ask again.

Logout

The logout closes the session of the application, renews its id, and sends the user to the logout of Keycloak (OpenID Connect RP-Initiated Logout) with the `id_token_hint`, the `client_id` and the `post_logout_redirect_uri`. That ends the session of the user in Keycloak too: otherwise the next login would be a single sign-on, without asking for the password. Keycloak sends the user back to `AUTH_KEYCLOAK_POST_LOGOUT_REDIRECT_URI`.

If the session has no ID token, or `AUTH_KEYCLOAK_END_SESSION=false`, the user goes straight to `AUTH_LOGOUT_REDIRECT_PATH`. The logout is a POST of the same origin: see [Security](#).

Last updated on 08/10/2026