

Database

The database provider of Derafu Auth

Anonymous · 08/10/2026

Database

The database provider authenticates the users against a table of a database: the application has a login form, and the user logs in with an identity (an email, for example) and a password. It does not use the libraries of the Keycloak provider. Its login form is protected with a CSRF token, so it needs a CSRF token manager and a session: [derafu/csrf](#) and [derafu/session](#), which a site that uses [derafu/foundation](#) already has (see [The login form](#)). It has no real consumer yet, so its design may still change.

Import `auth-database-services.yaml` and `auth-database-routes.yaml` (see [Configuration](#)). The routes are `/auth/login` (the page, and the POST of the form) and `/auth/logout`. The variables are in [Configuration](#).

The users

The identity and the hash of the password are columns of a table (`AUTH_DATABASE_USER_TABLE`, `AUTH_DATABASE_USER_FIELD_IDENTITY` and `AUTH_DATABASE_USER_FIELD_PASSWORD`); the hash is the one of `password_hash()`. The roles and the details come from two queries, with the parameter `:identity`:

```
AUTH_DATABASE_USER_SQL_GET_ROLES="SELECT r.name FROM role r JOIN user_role ur ON
ur.role_id = r.id JOIN user u ON u.id = ur.user_id WHERE u.email = :identity"
AUTH_DATABASE_USER_SQL_GET_DETAILS="SELECT id, name FROM user WHERE email = :identity"
```

- The **roles** are the first column of each row.
- The **details** are the first row. The default query is `SELECT * FROM <table> WHERE <identity> = :identity`. The column of the password is **never** kept in the user, even if the query gives it: the user goes to the session.
- The standard fields of the user (`getName()`, `getEmail()`...) are the columns that are called like them (`name`, `email`, `given_name`...), or the ones that the query renames. A column that is not there gives `null`, nothing fails: see [Users](#).
- The names of the table and the columns can only be names (letters, numbers and underscores, and a schema for the table), because they go in the queries as they are. Anything else is a `ConfigurationException`.
- The connection is made when it is needed, from `AUTH_DATABASE_URL` (or `DATABASE_URL` if it is empty), unless the application gives a PDO to `DatabaseUserRepository`.

Active users

A user can exist, with its password and its roles, and not be let in (it was suspended, it left the company). The package asks the database whether the user is **active** with a query, `AUTH_DATABASE_USER_SQL_IS_ACTIVE`, in the two places where it decides who gets in: **the login** and **the check of the session** (see [Security](#)). A user that is not active can not log in, and its session is closed when the session is asked again.

The default query uses a column of the table, `active` (`AUTH_DATABASE_USER_FIELD_ACTIVE` changes its name):

```
SELECT active FROM user WHERE email = :identity
```

- **The first value of the first row says it.** `0`, `false`, `null`, an empty text and the texts `0`, `f`, `false`, `n`, `no` and `off` say that the user is **not** active; any other value says that it is (`1`, `true`, `t`, a count of two). No row is not active.
- **Your own query.** If the table has no such column, or being active depends on more than a column, give the query in `AUTH_DATABASE_USER_SQL_IS_ACTIVE`. It has the parameter `:identity`:

```
AUTH_DATABASE_USER_SQL_IS_ACTIVE="SELECT COUNT(*) FROM user WHERE email = :identity AND suspended_at IS NULL AND deleted_at IS NULL"
```

- **No check.** If the application has no such concept, `AUTH_DATABASE_USER_SQL_IS_ACTIVE=false` (`'sql_is_active' => false` in the configuration) turns the check off: every user that exists is active.
- **A query that does not work is a configuration error.** If the table has no column `active` (and the application did not give its own query or turn the check off), the first login or check fails with a `ConfigurationException` that says so and what to do: it is not taken for a database that is down, and the session is not kept waiting.
- **The login does not say why.** A user that is not active with the right password gets the same message as a wrong password (`Invalid identity or password.`), and its hash is not changed.

A user needs **at least one role** to get in a protected path: the authorization middleware of Mezzio asks for each role of the user, and a user with none is denied (see [Authorization](#)).

The login form

The login form is protected with a **captcha** and with a **CSRF token**. The captcha (`captcha_protection: true` in its options, see [Captcha](#)) is the one that the application configures with `CAPTCHA_PROVIDER` ([derafu/captcha](#)): its page renders the widget, and a login without having solved it, or with what was solved for another form, is rejected before the credentials are looked at (the user sees “The captcha is not valid. Try again.”). **Without a captcha configured the login page fails** with a message that says what to configure (`CAPTCHA_PROVIDER=altcha` with `CAPTCHA_SECRET_KEY` needs no account; `CAPTCHA_PROVIDER=none` says, on purpose, that the application has none). The CSRF token works like this: its page renders a hidden field `_token`, and a login that comes without it, or with the token of another session, is rejected before the credentials are looked at (the user sees “The form is not valid or has expired. Reload the page and try again.”). It is the protection that `derafu/form` gives to every form (see [CSRF Protection](#)), and the token is asked for the form `login`.

The application needs `derafu/csrf`, `derafu/session` and `derafu/captcha`, which the package only suggests (the Keycloak provider has no form and does not need them), and the middleware of the tokens after the session ones:

```
$middlewares:
- '@Derafu\Http\Middleware\RouterMiddleware'
- '@Mezzio\Session\SessionMiddleware'
- '@Mezzio\Flash\FlashMessageMiddleware'
- '@Derafu\Csrf\CsrfSessionMiddleware'
- '@Mezzio\Authentication\AuthenticationMiddleware'
# ...
```

Without a manager, rendering the login page fails with an exception that says so: a login form is never left open by omission.

The password

- It is verified with `password_verify()`.
- A user that does not exist takes as long as one that does (the password is verified against a hash anyway), so the time of the answer does not say which identities exist.
- If the hash was made with an algorithm or a cost that is not the current one (`PASSWORD_DEFAULT`), it is made again when the user logs in, which is the only time that the password is known. The column of the password must be writable.

Failed attempts

The failed attempts to log in are counted in a window that starts with the first one and lasts `AUTH_DATABASE_LOGIN_LOCK_SECONDS` (900 by default):

- An identity from a network can fail `AUTH_DATABASE_LOGIN_MAX_ATTEMPTS` times (5 by default).
- A network can fail five times as many, whatever the identity.

Then the login is limited until the window ends: the credentials are **not even checked**, so a password that is guessed in the window is of no use, and the user sees “Too many failed login attempts. Try again in N minutes.” A login that works clears the count of that identity from that address. A form that is not valid (a field is missing) is not a failed attempt.

The counts are in the PSR-6 cache pool of the application, so they are shared by every request (derafu/foundation has one). The address is the **network of the client** that the HTTP layer decided (the attribute `client_network` that `ClientIpMiddleware` of [derafu/http](#) leaves in the request, which is the `/64` for an IPv6 client), or the `REMOTE_ADDR` of the connection if the request does not have it. The headers of the request (`X-Forwarded-For...`) are never read: behind a proxy, say which ones are trusted in `ClientIpMiddleware`, otherwise every user is the proxy. The identity is compared without its case or its spaces.

The login form and the pages

- The form is `LoginForm`: its fields are named after the two columns (so `email` and `password` by default), and its titles are the fixed texts `Username` and `Password`, in the translation domain `auth` (see [Translations](#)).
- `DatabaseController::login` renders `auth/login` (a template that extends `layouts/default`, the layout of the application, and shows the flash messages).
- After a login that works, and when a user that is logged in asks for the login page, the controller redirects to the page that was requested before the login, or to `AUTH_LOGIN_REDIRECT_PATH`: the login is a POST, and after it comes a redirect so a reload does not send the form again. The page is remembered by its path and query only, never by its host.
- The protection of the form against cross-site requests is the one of `derafu/form`.

The user of the session

The session keeps the user as the database had it when the user logged in. Every `AUTH_REFRESH_INTERVAL_SECONDS` seconds (300 by default) the next request asks the database again by the identity that the session has, with the same queries as the login (`sql_get_details` and `sql_get_roles`): the roles and the details of the session are the ones of that moment. A user that is not in the table loses the session. If the database can not be asked, the session is kept, the user is not let in for that request, and the next one asks again. See [Security](#).

The identity is the only thing that the check uses: the password is not asked again, and it is neither read nor changed.

Logout

A POST to `/auth/logout` from the same origin (see [Security](#)): the session is closed, its id is renewed and the user goes to `AUTH_LOGOUT_REDIRECT_PATH` (the login page by default).

Last updated on 08/10/2026