

Users

The user that both providers of Derafu Auth give, its standard fields and how to use a class of your own

Anonymous · 08/10/2026

Users

Both providers give the same kind of user: the `Derafu\Auth\User` (a `Derafu\Auth\Contract\UserInterface`). It has an **identity**, its **roles** and its **details**, and the same standard fields whether it comes from Keycloak or from the database.

```
$user = $request->getAttribute(\Mezzio\Authentication\UserInterface::class);

$user->getIdentity();    // The `sub` (Keycloak), or the identity of the table
                        (database).
$user->getRoles();       // ['admin', 'editor']
$user->getEmail();       // 'ana@example.com', or null
$user->getDetail('rut');  // Any other field that the provider gave.
```

The standard fields

They are the claims of OpenID Connect. The user has a method for each one:

Method	Detail	If it is not there
<code>getName()</code>	<code>name</code> , or <code>preferred_username</code>	<code>null</code>
<code>getGivenName()</code>	<code>given_name</code>	<code>null</code>
<code>getFamilyName()</code>	<code>family_name</code>	<code>null</code>
<code>getEmail()</code>	<code>email</code>	<code>null</code>
<code>isEmailVerified()</code>	<code>email_verified</code>	<code>false</code>
<code>getUsername()</code>	<code>preferred_username</code> , or <code>email</code>	<code>null</code>
<code>getLocale()</code>	<code>locale</code>	<code>null</code>

A field that is not there is `null` (or `false`), and nothing fails. A Keycloak user that was created with only a username has no names and no email; a table that has no `given_name` column does not need one. Both give a user whose `getGivenName()` is `null`. The text fields are texts: a number is turned into text, and a value that is blank, a list or a boolean is `null` (so a blank name lets `getName()` fall back to the username). `isEmailVerified()` is true for true, 1, '1' and 'true'.

`getName()` falls back to the username, and `getUsername()` to the email, so a screen that says “Hello, ...” has something to show. The email is **not** a name: `getName()` does not fall back to it.

Where the fields come from

	Keycloak	Database
Details	The claims of the access token and the user info, as they come.	The first row of <code>AUTH_DATABASE_USER_SQL_GET_DETAILS</code> (without the password).
The standard fields	The claims of OpenID Connect, that Keycloak gives for a user that has them.	The columns that are called like them (name, email...), or the ones that the query renames.
Another field	A claim that a mapper of the realm adds (an attribute of the user, its groups).	Any other column, or what the query gives.

In Keycloak a custom field has to reach the token: add a protocol mapper (for example “User Attribute”) to a client scope of the client, with the claim name and *Add to access token* or *Add to userinfo*. A claim that only goes to the ID token is not in the details.

In the database the default query is `SELECT * FROM <table> WHERE <identity> = :identity`, so the columns of the table are the details. If the columns are called otherwise, the query renames them:

```
AUTH_DATABASE_USER_SQL_GET_DETAILS="SELECT id, correo AS email, nombre AS name, rut
FROM user WHERE correo = :identity"
```

Without a `given_name` column, `getGivenName()` is null: the package never names those columns in a query. (A column that **your** query names and does not exist is an error of that query.)

Whatever the provider, the details are a copy that the session keeps, and they are renewed when the session asks the provider again (see [Security](#)).

A class of user of your own

An application that has fields of its own (a RUT, a company) can have a class of user with getters for them, in both providers. The user is made by a factory, `Derafu\Auth\Contract\UserFactoryInterface`, that receives what the provider worked out (the identity, the roles and the details):

```
use Derafu\Auth\Contract\UserFactoryInterface;
use Derafu\Auth\Contract\UserInterface;
use Derafu\Auth\User;
```

```
final class AppUser extends User
{
    public function getRut(): ?string
    {
        $rut = $this->getDetail('rut');

        return is_string($rut) ? $rut : null;
    }
}

final class AppUserFactory implements UserFactoryInterface
{
    public function create(string $identity, array $roles = [], array $details = []):
    UserInterface
    {
        return new AppUser($identity, $roles, $details);
    }
}
```

Register it in the container in place of the default one (`Derafu\Auth\UserFactory`), in the `services.yaml` of the application:

```
services:
    Derafu\Auth\Contract\UserFactoryInterface:
        class: App\Auth\AppUserFactory
```

The users that the login gives, the ones that are made from the session and the ones of the check (when the provider is asked again) are of that class. `Derafu\Auth\UserFactory`, the default one, is also the callable that Mezzio asks for when it needs to make a user (`((new UserFactory()))()`).

Keycloak. The provider works out the identity (`sub`), the roles (the ones of the realm, and the ones of the client of the application) and the details (the claims), and gives them to the factory.

Last updated on 08/10/2026