

How it works

Authentication and authorization, the channels, the providers and the rules of access

Anonymous · 08/10/2026

How it works

The package has two jobs, and it keeps them apart:

- **Authentication** answers **who is asking**: it looks at the request and gives a user (the anonymous one, if there is nobody).
- **Authorization** answers **may they enter here**: it looks at what the path asks (a user, some roles) and at the user.

Each has one entry point, the one that its middleware asks: `AuthenticationManager` (`AuthenticationInterface`, asked by `AuthenticationMiddleware`) and `AuthorizationManager` (`AuthorizationInterface`, asked by `AuthorizationMiddleware`). They are the ones that a pipeline names, and the user goes in the request in the attribute that has the name of the interface of Mezzio. Neither knows how a request is identified, nor where the users are, nor which paths are protected: that is what the pieces below know.

Three pieces that do not depend on each other

Piece	The question it answers	What there is
Channel	How does a request identify itself, and what is answered to the one that is not authenticated?	<code>web</code> (a session, a redirect to the login) and <code>api</code> (credentials in the header of every request, a 401).
Provider	Where are the users, and how is a credential checked?	Keycloak, database and <code>htpasswd</code> .
Access rules	Which paths need a user, and which roles?	<code>AccessRules</code> : the protected paths, the roles of each one, and whether they are enforced.

A channel is chosen by the request, a provider by the configuration of the site, and the access rules do not care about either. So a path is protected the same way whether it is a page or an API, and a provider gives the same users to both.

What happens in a request

1. **The channels that match.** The manager asks each channel, in order, whether the request is of its area. The API channel matches the paths of `AUTH_API_PATHS` (`/api` by default), and the web channel matches everything: it is the last one.
2. **Who is asking.** The first matching channel that knows says it: a user (or the anonymous one), or that it has nothing to say, and then the next one is asked. A channel can also say that the request ends in it: the logout.
 - The API channel reads the header `Authorization`. Credentials that are valid give the user; credentials that are not valid give the anonymous user and **do not** fall back to a session (the client did not ask for one). **No** credentials: it has nothing to say, so the session of the web channel is asked, and a path can serve browsers and programs.
 - The web channel reads the session, handles the login (the callback of the provider, or the form) and the logout.
3. **May they enter.** The access rules are asked whether the path needs a user. If it does and the user is the anonymous one, the manager says “not authenticated” (`null`, the way that Mezzio expects it), and Mezzio asks for the **unauthorized response of the channel**: a redirect to the login in the web, a 401 with `WWW-Authenticate` in the API. This is the only point where the authentication touches the authorization.
4. **Which roles.** The authorization middleware asks the access rules which roles the request needs, and compares them with the ones of the user: if they do not match it throws a 403 (an error page in HTML, or a problem in JSON for the API) whose message says which roles give access.

The login and the logout are public: the channel lets them through before the access rules are asked, so a protected path that includes them does not turn them away.

What a provider gives to each channel

Provider	To the web channel (the way in)	To the API channel (the credentials of the header)
Keycloak	<code>KeycloakWebFlow</code> : OpenID Connect, with a redirect to Keycloak and its callback.	<code>KeycloakBearerScheme</code> : Bearer, the access token that Keycloak gave.
Database	<code>DatabaseWebFlow</code> : a login form.	<code>DatabaseBasicScheme</code> : Basic, the user and the password.
Htpasswd	<code>HtpasswdWebFlow</code> : a login form.	<code>HtpasswdBasicScheme</code> : Basic, the user and the password.

The web channel does what is the same for every provider (the session, the page that is remembered, the flash messages, the redirects) and asks the flow for what is its own (`WebFlowInterface`). The API channel does the same with the scheme (`ApiSchemeInterface`): it reads the header, and the scheme says what is valid and how the 401 announces it.

A configuration that is wrong fails where it is used

Nothing is made, and nothing is checked, until a request uses it. A provider that is not configured fails **in the paths that need it**, with a message that says which variable to set (for example, `The URL of Keycloak is not configured: set AUTH_KEYCLOAK_URL.`), and the rest of the site keeps working, its error page included:

- The pages that nobody protects are answered without the provider.
- A protected page, the login page and the callback say what is missing, before a user fills anything.
- The logout needs no configuration.
- A request to the API that sends no credentials gets its `401`; one that sends credentials says what is missing.

Each channel checks what **it** needs: the login needs the redirect URI of Keycloak, and the API does not (it only verifies tokens), so a service that has only an API does not have to configure a login.

The names of the variables

The name says what the variable is about, in this order: the channel or the provider, and the channel inside the provider when it is about only one of them.

The variable is about	Prefix	Examples
Who may enter which path	<code>AUTH_</code>	<code>AUTH_ENABLED</code> , <code>AUTH_PROTECTED_PATHS</code>
The web channel	<code>AUTH_WEB_</code>	<code>AUTH_WEB_LOGOUT_PATH</code> , <code>AUTH_WEB_LOGIN_REDIRECT_PATH</code>
The API channel	<code>AUTH_API_</code>	<code>AUTH_API_PATHS</code> , <code>AUTH_API_REALM</code>
A provider	<code>AUTH_<PROVIDER>_</code>	<code>AUTH_KEYCLOAK_URL</code> , <code>AUTH_DATABASE_URL</code> , <code>AUTH_HTTPASSWD_PATH</code>
A provider, in one channel	<code>AUTH_<PROVIDER>_<CHANNEL>_</code>	<code>AUTH_KEYCLOAK_WEB_REDIRECT_URI</code> , <code>AUTH_KEYCLOAK_API_AUDIENCE</code>

Every variable is in [Configuration](#).

Adding a channel

A channel is a class that implements `ChannelInterface`, and a service tagged `derafu_auth.channel` with its priority (the higher is asked first: the API one has 100 and the web one 0). The manager, the access rules and the providers do not change:

```
interface ChannelInterface
{
    public function name(): string;
    public function matches(ServerRequestInterface $request): bool;
    public function identify(ServerRequestInterface $request): Identification;
    public function isPublic(ServerRequestInterface $request): bool;
    public function unauthorizedResponse(ServerRequestInterface $request):
ResponseInterface;
}
```

- `matches()` says whether the request is of the area of the channel.
- `identify()` returns an `Identification`: `Identification::of($user)`, `Identification::none()` (nothing to say: the next channel is asked) or `Identification::halt()` (the request ends here and `unauthorizedResponse()` answers it).
- `isPublic()` says whether the channel lets the request through without asking the access rules (the login and the logout of the web).
- `unauthorizedResponse()` is what a request that is not authenticated gets.

```
App\Auth\WebhookChannel:
  lazy: Derafu\Auth\Contract\ChannelInterface
  tags:
    - { name: derafu_auth.channel, priority: 50 }
```

A channel that needs something of the provider, as the web and the API ones do, defines its own interface for it (like `WebFlowInterface` and `ApiSchemeInterface`), and each provider that supports it implements it.

Adding a provider

A provider is its configuration, its way to find the users (`UserRepositoryInterface`), and what it gives to each channel it supports: a `WebFlowInterface` for the web, an `ApiSchemeInterface` for the API (`BasicScheme` and `BearerScheme` are the bases of the two schemes that exist). Its services file imports `auth-services.yaml` and says which flow and which scheme are the ones of the application:

```
Derafu\Auth\Contract\WebFlowInterface:
  class: App\Auth\LdapWebFlow
  lazy: true
Derafu\Auth\Contract\ApiSchemeInterface:
  class: App\Auth\LdapBasicScheme
  lazy: true
```

Every service is lazy

All the services of the package are lazy (`lazy: true`): what a request does not use is not made. It is

what lets a configuration that is wrong fail where it is used, and it is what an application that replaces a service should do as well.

Last updated on 08/10/2026