Docs

Auth Project

Derafu Auth

Anonymous · 09/09/2026 · #php

Table of Contents

Auth Project

Derafu Auth

Anonymous · 09/09/2026 · #php

Derafu Auth

Last updated on 09/09/2026

Introduction

Authentication and Authorization

Anonymous · 09/09/2026

Derafu: Auth

PSR-15 compliant authentication and authorization library for PHP applications with Keycloak integration.

Features

- **PSR-15 Middleware**: Standard-compliant middleware.
- **Keycloak Integration**: OAuth2/OpenID Connect.
- **Session Management**: Secure session handling.
- **Token Refresh**: Automatic token refresh.
- **Route Protection**: Flexible route-based auth.
- **CSRF Protection**: State parameter validation.

Quick Start

Installation

Install the package:

```
composer require derafu/auth
```

Environment Variables

Configure, at the very least, the following environment variables:

```
KEYCLOAK_URL=http://localhost:8080
KEYCLOAK_CLIENT_ID=your-client-id
KEYCLOAK_CLIENT_SECRET=your-client-secret
KEYCLOAK_REDIRECT_URI=http://localhost/auth/callback
```

Routes

Import the routes to your `routes.yaml`:

```
imports:
    - { resource: '../vendor/derafu/auth/resources/config/auth-routes.yaml' }
```

Services

Import the services to your `services.yaml`:

```
imports:
    - { resource: '../vendor/derafu/auth/resources/config/auth-services.yaml' }
```

Middleware

Add the middleware to your `services.yaml`:

```
Psr\Http\Server\RequestHandlerInterface:
    class: Derafu\Http\Service\RequestHandler
    public: true
    arguments:
        $middlewares:
            - '@Derafu\Auth\Middleware\AuthenticationMiddleware'
```

Note: the `AuthenticationMiddleware` needs to be placed before the `DispatcherMiddleware`.

Access User Information

Directly with the attribute `user` or `access_token`:

```
$user = $request->getAttribute('user');
$accessToken = $request->getAttribute('access_token');
```

Using the `UserInterface` from the `mezzio/mezzio-authentication` package:

```
$user = $request->getAttribute(UserInterface::class);
if ($user) {
    $identity = $user->getIdentity();
    $roles = iterator_to_array($user->getRoles());
    $email = $user->getDetail('email');
}
```

Last updated on 09/09/2026

Configuration

Configuration options for Derafu Auth

Anonymous · 09/09/2026

Configuration

Environment Variables

Variable	Type	Required	Default	Description
`KEYCLOAK_URL`	string	**Yes**	-	Keycloak server URL
`KEYCLOAK_CLIENT_ID`	string	**Yes**	-	OAuth2 client ID
`KEYCLOAK_CLIENT_SECRET`	string	**Yes**	-	OAuth2 client secret
`KEYCLOAK_REDIRECT_URI`	string	**Yes**	-	OAuth2 redirect URI
`KEYCLOAK_REALM`	string	No	`master`	Keycloak realm
`KEYCLOAK_SCOPES`	json	No	`["openid", "profile", "email"]`	OAuth2 scopes
`KEYCLOAK_PROTECTED_ROUTES`	json	No	`["/dashboard", "/profile", "/admin"]`	Protected routes
`KEYCLOAK_CALLBACK_ROUTE`	string	No	`/auth/callback`	Callback route
`KEYCLOAK_LOGOUT_ROUTE`	string	No	`/auth/logout`	Logout route
`KEYCLOAK_SESSION_LIFETIME`	int	No	`3600`	Session lifetime
`KEYCLOAK_SECURE_COOKIES`	bool	No	`false`	Secure cookies
`KEYCLOAK_HTTP_TIMEOUT`	int	No	`30`	HTTP timeout
`KEYCLOAK_HTTP_CONNECT_TIMEOUT`	int	No	`30`	HTTP connect timeout
`KEYCLOAK_HTTP_VERIFY`	bool	No	`false`	SSL verification

Required Configuration

The following parameters are **mandatory** and must be provided:

```
# Keycloak server configuration.
KEYCLOAK_URL=https://<YOUR_KEYCLOAK_SERVER>

# OAuth2 client credentials.
KEYCLOAK_CLIENT_ID=your-client-id
KEYCLOAK_CLIENT_SECRET=your-client-secret

# OAuth2 redirect URI.
KEYCLOAK_REDIRECT_URI=https://<YOUR_APP_URL>/auth/callback
```

Routes Configuration

```
auth_callback:
    path: /auth/callback
    handler: Derafu\Auth\Controller\CallbackController::handle
```

Services Configuration

```
services:

    _defaults:
        autowire: true
        autoconfigure: true
        public: false

    Derafu\Auth\Contract\AuthConfigurationInterface:
        class: Derafu\Auth\Configuration\AuthConfiguration
        arguments:
            $config:
                keycloak_url: '%env(default::string:KEYCLOAK_URL)%'
                realm: '%env(default::string:KEYCLOAK_REALM)%'
                client_id: '%env(default::string:KEYCLOAK_CLIENT_ID)%'
                client_secret: '%env(default::string:KEYCLOAK_CLIENT_SECRET)%'
                redirect_uri: '%env(default::string:KEYCLOAK_REDIRECT_URI)%'
                scopes: '%env(default::json:KEYCLOAK_SCOPES)%'
                protected_routes: '%env(default::json:KEYCLOAK_PROTECTED_ROUTES)%'
                callback_route: '%env(default::string:KEYCLOAK_CALLBACK_ROUTE)%'
                logout_route: '%env(default::string:KEYCLOAK_LOGOUT_ROUTE)%'
                session_lifetime: '%env(default::int:KEYCLOAK_SESSION_LIFETIME)%'
                secure_cookies: '%env(default::bool:KEYCLOAK_SECURE_COOKIES)%'
                http_client_options:
                    timeout: '%env(default::int:KEYCLOAK_HTTP_TIMEOUT)%'
                    connect_timeout:
'%env(default::int:KEYCLOAK_HTTP_CONNECT_TIMEOUT)%'
                    verify: '%env(default::bool:KEYCLOAK_HTTP_VERIFY)%'
```

```
    Derafu\Auth\Contract\AuthenticationProviderInterface:
        class: Derafu\Auth\Service\KeycloakAuthenticationService

    Derafu\Auth\Contract\SessionManagerInterface:
        class: Derafu\Auth\Service\SessionService

    Derafu\Auth\Contract\RouteValidatorInterface:
        class: Derafu\Auth\Validator\RouteValidator

    Derafu\Auth\Middleware\AuthenticationMiddleware: ~

    Derafu\Auth\Controller\CallbackController:
        public: true
```

Last updated on 09/09/2026

Route Protection

Route protection capabilities in Derafu Auth

Anonymous · 09/09/2026

Route Protection

The Derafu Auth package provides route protection through a simple array-based configuration system.

How It Works

The `RouteValidator` class determines which routes require authentication by checking if the requested path starts with any of the configured protected routes.

```
public function requiresAuth(string $path): bool
{
    foreach ($this->config->getProtectedRoutes() as $route) {
        if (str_starts_with($path, $route)) {
            return true;
        }
    }

    return false;
}
```

Configuration

Environment Variable

```
KEYCLOAK_PROTECTED_ROUTES='["/dashboard", "/profile", "/admin"]'
```

Default Protected Routes

If not specified, the package uses these default protected routes:

```
["/dashboard", "/profile", "/admin"]
```

Route Protection Examples

Basic Protection

Protect specific routes:

```
KEYCLOAK_PROTECTED_ROUTES=["/dashboard", "/profile", "/admin"]
```

This configuration will protect:

- `/dashboard` - Exact match.
- `/dashboard/settings` - Starts with `/dashboard`.
- `/profile` - Exact match.
- `/profile/edit` - Starts with `/profile`.
- `/admin` - Exact match.
- `/admin/users` - Starts with `/admin`.

API Protection

Protect API routes:

```
KEYCLOAK_PROTECTED_ROUTES=["/api/v1", "/api/v2"]
```

This will protect all routes starting with `/api/v1` or `/api/v2`:

- `/api/v1/users`
- `/api/v1/users/123`
- `/api/v2/admin`
- `/api/v2/admin/settings`

Admin Panel Protection

Protect admin panel:

```
KEYCLOAK_PROTECTED_ROUTES=["/admin", "/moderator"]
```

This protects:

- `/admin` - Admin panel
- `/admin/users` - User management
- `/admin/settings` - Settings
- `/moderator` - Moderator panel
- `/moderator/comments` - Comment moderation

Limitations

The current implementation has the following limitations:

1. **Prefix Matching Only**: Routes are protected using `str_starts_with()`, so `/admin` will also protect `/admin-panel` and `/administer`.
2. **No Regex Support**: The package does not support regular expressions for route matching.
3. **No Wildcard Support**: There is no support for wildcard patterns like `/admin/*`.
4. **No Negative Patterns**: Cannot exclude specific routes from protection.
5. **No Role-Based Protection**: All protected routes require the same level of authentication. But you can use authorization to protect specific routes based on the user's roles.

Special Routes

The package automatically handles these special routes:

- **Callback Route**: Default is `/auth/callback`.
- **Logout Route**: Default is `/auth/logout`.

These routes are automatically excluded from authentication requirements.

Custom Route Validator

You can create a custom route validator by implementing the `RouteValidatorInterface`:

```
use Derafu\Auth\Contract\RouteValidatorInterface;

class CustomRouteValidator implements RouteValidatorInterface
{
    public function requiresAuth(string $path): bool
    {
        // Your custom logic here
        return str_starts_with($path, '/protected');
    }

    public function isCallbackPath(string $path): bool
    {
```

```
        return $path === '/auth/callback';
    }

    public function isLogoutPath(string $path): bool
    {
        return $path === '/auth/logout';
    }

    public function getProtectedRoutes(): array
    {
        return ['/protected'];
    }

    public function getCallbackRoute(): string
    {
        return '/auth/callback';
    }

    public function getLogoutRoute(): string
    {
        return '/auth/logout';
    }
}
```

Then configure it in your services:

```
services:
    Derafu\Auth\Contract\RouteValidatorInterface:
        class: App\Auth\CustomRouteValidator
```

Last updated on 09/09/2026

Authorization

Authorization and permissions in Derafu Auth

Anonymous · 09/09/2026

Authorization

Derafu Auth handles **authentication** (who you are) and helps with **authorization** (what you can do). The last one depends on the roles and permissions you set in Keycloak.

What comes from the package?

When a user is authenticated, the package automatically provides:

- **Basic data**: `sub`, `email`, `name`, `preferred_username`, etc.
- **Roles**: Come directly from Keycloak (e.g., `user`, `admin`, `moderator`).

What you need to implement?

Option 1: Role-based authorization (Recommended)

Advantages:

- Simple to implement.
- Roles already come from Keycloak.
- No additional configuration required.

How it works:

1. Keycloak assigns roles to users.
2. Your application maps permissions to roles.
3. You verify if the user has the required role.

Mapping example:

- Permission `user:read` → Roles `user`, `admin`, `moderator`.
- Permission `user:write` → Roles `admin`, `moderator`.
- Permission `admin:access` → Only role `admin`.

Option 2: Permission-based authorization

Advantages:

- More granular.
- Specific permissions per action.
- Better access control.

Disadvantages:

- Requires additional Keycloak configuration.
- More complex to maintain.

Keycloak configuration for permissions

Step 1: Create a custom scope

1. Go to **Clients** → Your Client → **Client Scopes**
2. Create a new scope called `permissions`.
3. Add it to your client.

Step 2: Configure Token Mapper

1. In the `permissions` scope, go to **Mappers**.
2. Create a new mapper:
    - **Name**: `permissions`
    - **Mapper Type**: `User Attribute`
    - **User Attribute**: `permissions`
    - **Token Claim Name**: `permissions`
    - **Claim JSON Type**: `String`
    - **Full group path**: `false`

Step 3: Assign permissions to users

1. Go to **Users** → Select a user → **Attributes**
2. Add the attribute:
    - **Key**: `permissions`
    - **Value**: `user:read,user:write,admin:access`

Step 4: Configure the scope in your application

```
KEYCLOAK_SCOPES=["openid", "profile", "email", "permissions"]
```

How do permissions work?

Authorization flow:

1. **User authenticates** → Keycloak generates token with permissions.
2. **Token arrives at your app** → Derafu Auth extracts the data.
3. **Your code verifies** → If the user has the required permission.
4. **Access granted/denied** → Based on the verification.

Recommended permission structure:

```
resource:action
```

Examples:

- `user:read` - Read users.
- `user:write` - Create/edit users.
- `user:delete` - Delete users.
- `admin:access` - Access admin panel.
- `report:generate` - Generate reports.

Key differences

Aspect	Roles	Permissions
Configuration	Automatic	Requires Keycloak
Granularity	Access groups	Specific actions
Maintenance	Easy	More complex
Flexibility	Limited	High

Recommendation

Start with role-based authorization because:

- It’s simpler to implement.
- No additional Keycloak configuration required.
- Sufficient for most applications.
- You can migrate to permissions later if needed.

Summary

- **Authentication**: Handled automatically by Derafu Auth.
- **Authorization**: Implementation based on roles or permissions.
- **Roles**: Come automatically from Keycloak.
- **Permissions**: Require additional Keycloak configuration.
- **Recommendation**: Start with roles, migrate to permissions if you need more granularity.

Last updated on 09/09/2026

Security

Security best practices for Derafu Auth

Anonymous · 09/09/2026

Security

Environment Variables

Always use environment variables for sensitive configuration:

```
# □ Good
KEYCLOAK_CLIENT_SECRET=your-secret-here

# □ Bad
'client_secret' => 'your-secret-here'
```

HTTPS Configuration

Enable HTTPS in production:

```
# Development.
KEYCLOAK_SECURE_COOKIES=false
KEYCLOAK_HTTP_VERIFY=false

# Production.
KEYCLOAK_SECURE_COOKIES=true
KEYCLOAK_HTTP_VERIFY=true
```

Session Security

```
// Secure session configuration.
$config = new AuthConfiguration([
    'session_lifetime' => 3600,
    'secure_cookies' => true,
]);

// Additional session security that can be set.
ini_set('session.cookie_httponly', true);
ini_set('session.cookie_samesite', 'Lax');
```

```
ini_set('session.use_strict_mode', true);
```

CSRF Protection

The package automatically validates the `state` parameter to prevent CSRF attacks.

Token Security

Never store tokens in client-side storage:

```
// ✅ Good - Server-side session.
$session->set('access_token', $accessToken);

// ❌ Bad - Never expose to client.
return new JsonResponse(['token' => $accessToken]);
```

Authorization

```
// Validate role format.
private function isValidRole(string $role): bool
{
    return preg_match('/^[a-zA-Z0-9_]+$/', $role) === 1;
}

// Check role hierarchy.
private function hasRole(array $userRoles, string $requiredRole): bool
{
    $roleHierarchy = [
        'super_admin' => ['admin', 'user'],
        'admin' => ['user'],
    ];

    if (in_array($requiredRole, $userRoles)) {
        return true;
    }

    foreach ($userRoles as $userRole) {
        if (isset($roleHierarchy[$userRole]) &&
            in_array($requiredRole, $roleHierarchy[$userRole])) {
            return true;
        }
    }

    return false;
}
```

Error Handling

Don’t expose sensitive information in production:

```
public function handle(ServerRequestInterface $request, Throwable $exception):
ResponseInterface
{
    $isProduction = getenv('APP_ENV') === 'production';

    if ($exception instanceof AuthenticationException) {
        return new JsonResponse([
            'error' => 'Authentication failed',
            'code' => 'UNAUTHORIZED'
        ], 401);
    }

    return new JsonResponse([
        'error' => $isProduction ? 'Internal server error' : $exception->getMessage(),
        'code' => 'INTERNAL_ERROR'
    ], 500);
}
```

Security Headers

```
class SecurityHeadersMiddleware
{
    public function process(ServerRequestInterface $request, RequestHandlerInterface
$handler): ResponseInterface
    {
        $response = $handler->handle($request);

        return $response
            ->withHeader('X-Content-Type-Options', 'nosniff')
            ->withHeader('X-Frame-Options', 'DENY')
            ->withHeader('X-XSS-Protection', '1; mode=block')
            ->withHeader('Referrer-Policy', 'strict-origin-when-cross-origin')
            ->withHeader('Content-Security-Policy', "default-src 'self'")
            ->withHeader('Strict-Transport-Security', 'max-age=31536000;
includeSubDomains');
    }
}
```

Keycloak Security

Client Configuration

1. **Access Type**: `confidential`.
2. **Valid Redirect URIs**: Only your application URLs.
3. **Web Origins**: Your application domain.
4. **Client Authentication**: Enabled.

Realm Settings

1. **Password Policy**: Strong password requirements.
2. **Brute Force Detection**: Enabled.
3. **Session Timeout**: Configured appropriately.
4. **SSL Required**: `external` or `all`.

Last updated on 09/09/2026

Examples

Practical examples for Derafu Auth

Anonymous · 09/09/2026

Examples

Minimal Setup

```
<?php

require_once 'vendor/autoload.php';

use Derafu\Auth\Configuration\AuthConfiguration;
use Derafu\Auth\Service\KeycloakAuthenticationService;
use Derafu\Auth\Service\SessionService;
use Derafu\Auth\Validator\RouteValidator;
use Derafu\Auth\Middleware\AuthenticationMiddleware;

$config = new AuthConfiguration([
    'keycloak_url' => 'http://localhost:8080',
    'client_id' => 'your-client-id',
    'client_secret' => 'your-client-secret',
    'redirect_uri' => 'http://localhost/auth/callback',
]);

$authService = new KeycloakAuthenticationService($config);
$sessionService = new SessionService($config);
$routeValidator = new RouteValidator($config);

$authMiddleware = new AuthenticationMiddleware(
    $authService,
    $sessionService,
    $routeValidator
);
```

API Examples

REST API with Authentication

```
class UserApiController
{
```

```
    public function getUsers(ServerRequestInterface $request): ResponseInterface
    {
        $user = $request->getAttribute('user');

        if (!$user) {
            return new JsonResponse(['error' => 'Not authenticated'], 401);
        }

        if (!$this->isAuthorized($user, 'user:read')) {
            return new JsonResponse(['error' => 'Access denied'], 403);
        }

        $users = $this->getUserList();

        return new JsonResponse(['users' => $users]);
    }

    private function isAuthorized(array $user, string $permission): bool
    {
        $userPermissions = $user['permissions'] ?? [];

        return in_array($permission, $userPermissions);
    }
}
```

GraphQL with Authentication

```
use GraphQL\Type\Definition\ObjectType;
use GraphQL\Type\Definition\Type;
use GraphQL\Type\Schema;

class AuthenticatedGraphQLSchema
{
    public function createSchema(): Schema
    {
        $userType = new ObjectType([
            'name' => 'User',
            'fields' => [
                'id' => Type::string(),
                'email' => Type::string(),
                'name' => Type::string(),
                'roles' => Type::listOf(Type::string()),
            ],
        ]);

        $queryType = new ObjectType([
            'name' => 'Query',
            'fields' => [
                'me' => [
                    'type' => $userType,
```

```
                    'resolve' => function ($root, $args, $context) {
                        $user = $context['user'] ?? null;

                        if (!$user) {
                            throw new \Exception('Not authenticated');
                        }

                        return $user;
                    },
                ],
            ],
        ]);

        return new Schema([
            'query' => $queryType,
        ]);
    }
}
```

Testing Example

```
use PHPUnit\Framework\TestCase;
use Derafu\Auth\Middleware\AuthenticationMiddleware;

class AuthenticationMiddlewareTest extends TestCase
{
    public function testProtectedRouteRequiresAuthentication()
    {
        $config = new AuthConfiguration([
            'keycloak_url' => 'http://localhost:8080',
            'client_id' => 'test-client',
            'client_secret' => 'test-secret',
            'redirect_uri' => 'http://localhost/auth/callback',
            'protected_routes' => ['/dashboard'],
        ]);

        $authService = $this->createMock(KeycloakAuthenticationService::class);
        $sessionService = $this->createMock(SessionService::class);
        $routeValidator = new RouteValidator($config);

        $middleware = new AuthenticationMiddleware(
            $authService,
            $sessionService,
            $routeValidator
        );

        $request = $this->createMock(ServerRequestInterface::class);
        $request->method('getUri')->willReturn(new Uri('/dashboard'));
```

```
        $handler = $this->createMock(RequestHandlerInterface::class);

        $response = $middleware->process($request, $handler);

        $this->assertEquals(302, $response->getStatusCode()); // Redirect to login.
    }
}
```

Real-World Examples

E-commerce Application

```
class EcommerceAuthController
{
    public function handleLogin(ServerRequestInterface $request): ResponseInterface
    {
        $user = $request->getAttribute('user');

        if (!$user) {
            return new JsonResponse(['error' => 'Not authenticated'], 401);
        }

        if (!$this->hasPermission($user, 'shop:access')) {
            return new JsonResponse(['error' => 'Access denied'], 403);
        }

        $cart = $this->getUserCart($user['sub']);

        return new JsonResponse([
            'user' => $user,
            'cart' => $cart,
            'permissions' => $user['permissions'] ?? [],
        ]);
    }

    private function hasPermission(array $user, string $permission): bool
    {
        $permissions = $user['permissions'] ?? [];

        return in_array($permission, $permissions);
    }
}
```

Admin Panel

```
class AdminPanelController
{
```

```
    public function handleDashboard(ServerRequestInterface $request):
ResponseInterface
    {
        $user = $request->getAttribute('user');

        if (!$user) {
            return new JsonResponse(['error' => 'Not authenticated'], 401);
        }

        if (!$this->isAdmin($user)) {
            return new JsonResponse(['error' => 'Access denied'], 403);
        }

        $stats = $this->getAdminStats();

        return new JsonResponse([
            'user' => $user,
            'stats' => $stats,
        ]);
    }

    private function isAdmin(array $user): bool
    {
        $roles = $user['roles'] ?? [];

        return in_array('admin', $roles) || in_array('super_admin', $roles);
    }
}
```

Last updated on 09/09/2026