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