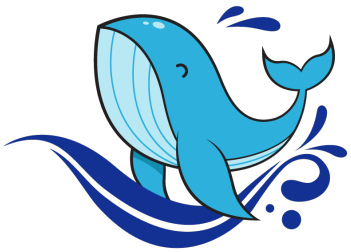
# Balea

*Release 1.0.0*

**Jun 20, 2023**

# About Balea



**Balea** is an authorization framework for ASP.NET Core developers that aims to help us to decoupling authentication and authorization in our web applications.

CHAPTER 1

# Terminology

The documentation and object model use a certain terminology that you should be aware of.

## 1.1 Applications

Allow you to manage multiple different software projects, for example. Each application has its own unique set of roles and delegations.

## 1.2 Roles

Are similar to security groups, to which users can become members and acquire a level of security that gives them the ability to perform some business operations. Roles can contain permissions, subjects and mappings. A role could be a teacher, custodian, student, etc.

## 1.3 Permissions

A permission is the ability to perform some specific operation like view grades, edit grades, etc.

## 1.4 Subjects

Subjects (Users) are grouped into roles and role access permissions are based upon the role, not individual.

## 1.5 Mappings

Roles coming from authentication system can be mapped to the application roles.

## 1.6 Store

A mechanism that allow you to store persistent the Balea's object model such as applications, roles, permissions, delegations. Balea provides out-of-the-box two stores:

- ASP.NET Core JSON Configuration Provider.
- Entity Framework Core.

CHAPTER 2

# Contributing

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change.

Please note we have a code of conduct, please follow it in all your interactions with the project.

## 2.1 Pull Request Process

1. Ensure any install or build dependencies are removed before the end of the layer when doing a build.
2. Update the README.md with details of changes to the interface, this includes new environment variables, exposed ports, useful file locations and container parameters.
3. Increase the version numbers in any examples files and the README.md to the new version that this Pull Request would represent. The versioning scheme we use is [SemVer](http://semver.org/).
4. You may merge the Pull Request in once you have the sign-off of two other developers, or if you do not have permission to do that, you may request the second reviewer to merge it for you.

## 2.2 Code of Conduct

### 2.2.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

### 2.2.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 2.2.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

### 2.2.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

### 2.2.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at [INSERT EMAIL ADDRESS]. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

### 2.2.6 Attribution

This Code of Conduct is adapted from the [Contributor Covenant][homepage], version 1.4, available at version

homepage

version

CHAPTER 3

# Getting started with Balea in ASP.NET Core

In this article, we are going to see how easy it is to use Balea in your ASP.NET Core application using the NuGet packages provided by Xabaril.

> In samples/WebApp you'll find a complete Balea example in ASP.NET Core.

## 3.1 Setup

To install Balea open a console window and type the following command using the .NET Core CLI:

```
dotnet package add Balea.Configuration.Store
```

or using Powershell or Package Manager:

```
Install-Package Balea.Configuration.Store
```

or install via NuGet.

In the **ConfigureServices** method of Startup.cs, register the Balea services:

```
services
  .AddBalea()
  .AddConfigurationStore(Configuration);
```

`AddBalea` method allows you to register the set of services that Balea needs to work. The `AddConfigurationStore` method registers the configuration store to use, in this case, based on the default configuration system of ASP.NET Core

By default Balea use a configuration section called **Balea** but you can change if you want:

```
services
  .AddBalea()
  .AddConfigurationStore(Configuration, key: "your key");
```

## 3.2 Defining applications

Applications allow you to manage authorization in multiple different software projects. Each application has its own unique roles and delegations. If you have a simple scenario where you only have one application, Balea give you a default application name called "default":

```
{
  "Balea": {
    "applications": [
      {
        "name": "default"
      }
    ]
  }
}
```

Or you can create as many applications as you want:

```
{
  "Balea": {
    "applications": [
      {
        "name": "hr"
      },
      {
        "name": "erp"
      }
    ]
  }
}
```

And you could specify which application do you want to use:

```
services
  .AddBalea(options => options.SetApplicationName("hr"))
  .AddConfigurationStore(Configuration);
```

## 3.3 Defining roles

Define roles is a straightforward proccess. Name the role, add a description, enable it and add permissions:

```
{
  "Balea": {
    "applications": [
      {
        "name": "default",
        "roles": [
          {
            "name": "teacher",
            "description": "Teacher role",
            "enabled": true,
            "permissions": [
              "grades.edit",
              "grades.view"
```

(continues on next page)

(continued from previous page)

```
            ]
          }
      }
    ]
  }
}
```

Additionaly, roles coming from authentication system can be mapped to the application roles:

```
{
  "Balea": {
    "applications": [
      {
        "name": "default",
        "roles": [
          {
            "name": "student",
            "description": "Student role",
            "enabled": true,
            "permissions": [
              "grades.view"
            ],
            "mappings": [
              "customer"
            ]
          }
      }
    ]
  }
}
```

## 3.4 Using roles in our controller and actions

By default, Balea automatically maps roles and permissions to the user's claims. This is mainly useful if you want to use the standard claims API or the `[Authorize]` attribute. For example:

```
[Authorize(Roles = "custodian")]
public IActionResult OpenDoor()
{
    return View();
}
```

Or using the standard claims API:

```
public IActionResult OpenDoor()
{
    if (User.IsInRole("custodian"))
    {
        return View();
    }

    return Forbid();
}
```

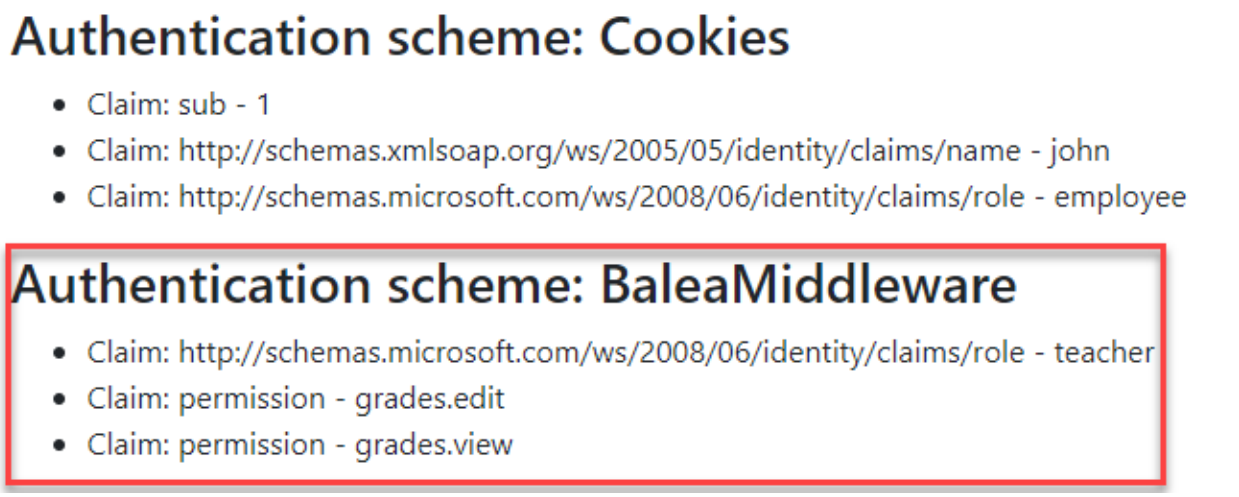
## 3.5 Using permissions in ASP.NET Core authorization policies

Also, Balea automatically maps permissions to ASP.NET Core authorization policies. You'll need to decorate your controllers/actions like this:

```
[Authorize(Policy = "grades.view")]
public IActionResult ViewGrades()
{
    return View();
}
```

## 3.6 Testing the authorization

If you run the example samples/WebApp you could see that Balea creates on the fly a new `ClaimsIdentity` with all the information from the authorization store for the user:



# CHAPTER 4

# Using EntityFramework Core store

Balea was designed to be extensible and one of the extensibility points is the storage. This article shows you how to configure Balea in order to use EntityFrameworkCore as the storage mechanism rather than using ASP.NET Configuration store.

> In samples/WebApp you'll find a complete Balea example in ASP.NET Core.

## 4.1 Configuring the store

To install Balea open a console window and type the following command using the .NET Core CLI:

```
dotnet package add Balea.EntityFrameworkCore.Store
```

or using Powershell or Package Manager:

```
Install-Package Balea.EntityFrameworkCore.Store
```

or install via NuGet.

In the **ConfigureServices** method of Startup.cs, register the Balea services:

```
services
    .AddBalea()
    .AddEntityFrameworkCoreStore(options =>
    {
        options.ConfigureDbContext = builder =>
        {
            builder.UseSqlServer(Configuration.GetConnectionString("Default"),␣
↪sqlServerOptions =>
            {
                sqlServerOptions.MigrationsAssembly(typeof(Startup).Assembly.
↪FullName);
            });
```

(continues on next page)

(continued from previous page)

```
        };
    })
```

`AddBalea` method allows you to register the set of services that Balea needs to works. The `AddEntityFrameworkCoreStore` method registers the EntityFrameworkCore store and also give the opportunity to configure your favorite provider

## 4.2 Adding the initial migration

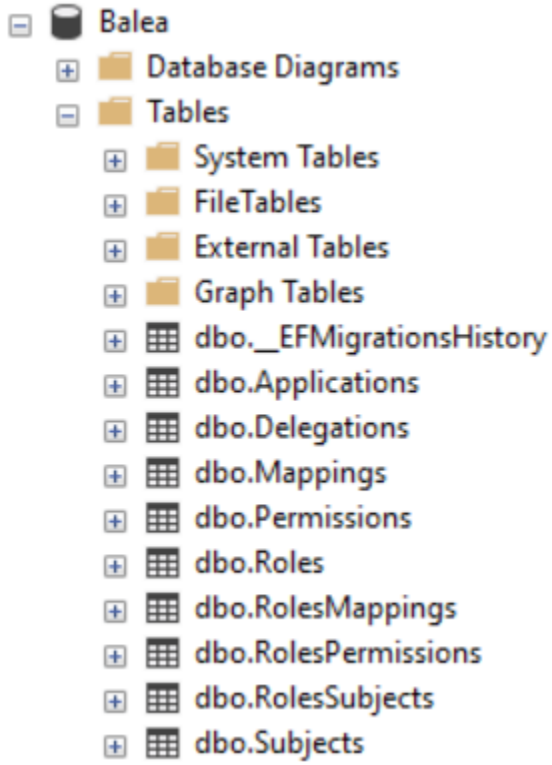
Balea.EntityFrameworkCore.Store contains all the entities needed to store all the Balea configuration in a database. This entities could be changed over the time, so you are responsible to upgrade your own database. To manage this changes one approach is using EntityFramework Core migrations. To create the initial migration in your web application open a console window and type the following commands using the .NET Core CLI:

```
dotnet ef migrations add Initial -c StoreDbContext -o "Migrations\Balea" -s "Path to␣
↪your proyect"
dotnet ef database update -s "Path to your proyect"
```

or using Powershell or Package Manager:

```
Add-Migration Initial -OutputDir "Migrations\Balea" -StartupProject "Path to your␣
↪project"
Update-Database -StartupProject "Path to your project"
```

The database should be created and you should be able to connect using SQL Server Management, Visual Studio or another tool:



CHAPTER 5

# Delegate Permissions to another users

In role-based access model, delegation involves delegating roles to another users that can assume a set of permissions to access to the resources on behalf of the user. Balea supports the delegation of your permissions to another person. This article shows you how to configure Balea in order to use delegations.

> In samples/WebApp you'll find a complete Balea example in ASP.NET Core using delegations.

## 5.1 Defining delegations

To delegate permissions to another user you must configure:

- **who:** User (Subject id) who delegates permissions.
- **whom:** User (Subject id) to delegating.
- **from:** The date when delegation starts.
- **to:** The date when delegation finish.
- **selected:** If the user is acting on behalf of (Only one should be selected, if there are more than one selected, Balea selects the first one the user have selected).

An example of delegation:

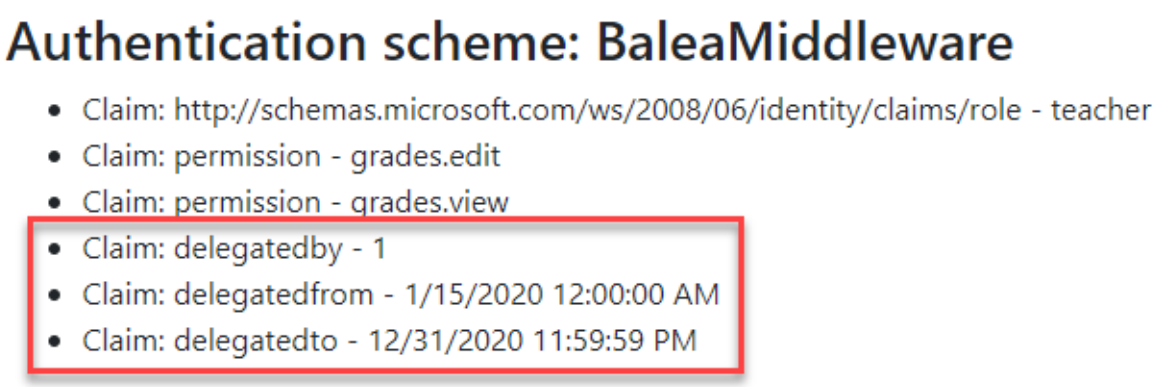
```
{
  "Balea": {
    "applications": [
      {
        "name": "default",
        "roles": [],
        "delegations": [
          {
            "who": "1",
            "whom": "2",
            "from": "2020-01-15 00:00:00",
```

(continues on next page)

(continued from previous page)

```
            "to": "2020-12-31 23:59:59",
            "selected": true
          }
        ]
      }
    ]
  }
}
```

When you are acting on behalf of somenone, Balea adds new claims to the principal:




For example, a use case could be use the claim **delegatedby** in your queries to retrieve data that belongs to the user which has delegated his permissions.

CHAPTER 6

# Configure a fallback mechanisim for unauthorized users

Even authenticated, not all users are authorized to access to all applications. Out-of-the-box Balea provides an authorization fallback mechanism to decide what to do with the unauhtorized users.

> In samples/WebApp you'll find an example of how to configure this fallback mechanism.

## 6.1 Configure the AuthorizationFallback

To configure the authorization fallback, in the method `AddBalea` you have a parameter for the fallback:

In the **ConfigureServices** method of Startup.cs, register the Balea services:

```
services
  .AddBalea(options =>
  {
    options.UnauthorizedFallback = (context) =>
    {
        context.Response.StatusCode = StatusCodes.Status403Forbidden;
        return Task.CompletedTask;
    };
  })
  .AddConfigurationStore(Configuration);
```

The `UnauthorizedFallback` is a RequestDelegate so you can configure the behavior when user is not authorized.

Out-of-the-box Balea provides a `AuthorizationFallbackAction` class that defines common fallback actions to be used when user is not authorized:

- Redirect result to MVC action:

```
public static RequestDelegate RedirectToAction(string controllerName, string␣
↪actionName)
```

- Redirect result:

```
public static RequestDelegate RedirectTo(string uri)
```

We can modify the code like this:

```
services
  .AddBalea(options =>
  {
    options.UnauthorizedFallback = AuthorizationFallbackAction.RedirectToAction("Home
↪","Denied");
  })
  .AddConfigurationStore(Configuration);
```

CHAPTER 7

# Dealing with claim types map

There is an long history about claim types and the reason why sometimes authorization does not work as we expect. We recommend you to read this awesome blog post about it before continuing reading.

The problem is basically that Microsoft has its own claim type names for RoleClaimType (http://schemas.microsoft.com/ws/2008/06/identity/claims/role) and NameClaimType (http://schemas.microsoft.com/ws/2008/06/identity/claims/name) and of course its own JWT validation library, which does not follow the more modern standard OpenID Connect claim types.

> In samples/WebApp you'll find an example of how to configure the mappings.

## 7.1 Configure the claim types map

By default, Balea asummes you are within Microsoft roles world. So, Balea will map roles from the authentication system to application roles using the ClaimTypes.Role `"http://schemas.microsoft.com/ws/2008/06/identity/claims/role"`. To configure the RoleClaimType and NameClaimType on the claim mapping, use the following parameter in the method `AddBalea`:

```
services
    .AddBalea(options =>
    {
        options.DefaultClaimTypeMap = new DefaultClaimTypeMap
        {
            RoleClaimType = JwtClaimTypes.Role
            NameClaimType = JwtClaimTypes.Name
        };
    })
```

In this case, Balea will look for roles coming from the authentication system using JwtClaimTypes.Role `"role"`.

Balea also adds new claim permission to the `` `ClaimsIdentity` ``:

## Authentication scheme: Cookies

- Claim: sub - 1
- Claim: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name - john
- Claim: http://schemas.microsoft.com/ws/2008/06/identity/claims/role - employee

## Authentication scheme: BaleaMiddleware

- Claim: http://schemas.microsoft.com/ws/2008/06/identity/claims/role - teacher
- Claim: permission - grades.edit
- Claim: permission - grades.view

You can change also the name of the claim:

```
services
    .AddBalea(options =>
    {
        options.DefaultClaimTypeMap = new DefaultClaimTypeMap
        {
            PermissionClaimType = "my-name"
        };
    })
```

## Authentication scheme: Cookies

- Claim: sub - 1
- Claim: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name - john
- Claim: http://schemas.microsoft.com/ws/2008/06/identity/claims/role - employee

## Authentication scheme: BaleaMiddleware

- Claim: http://schemas.microsoft.com/ws/2008/06/identity/claims/role - teacher
- Claim: my-name - grades.edit
- Claim: my-name - grades.view

CHAPTER 8

# Use Balea only in specific schemes

When you are using more than one scheme in your application you might need to run Balea only in certain schemes. For example, if you are hosting an Mvc application authenticated with cookies and some Api endpoints authenticated with tokens in the same host you may want to use Balea only in your Api with the JwtBearer scheme.

To configure Balea to run only in specific schemes you should add the schemes in the Balea configuration in the **ConfigureServices** method:

```
services
  .AddBalea(options =>
  {
      options.AddAuthenticationSchemes("Bearer");
  })
  .AddConfigurationStore(Configuration);
```

If no schemes registered in Balea it will run for the default authentication scheme configured.

If there are some authenticated schemes configured in Balea, it will run only in those specific schemes.