ASP.NET Boilerplate

Articles Tutorials

Filter...

latest (v7.1)

ABP Framework

Overall

- Introduction
- Tutorials & Articles
- NLayer Architecture
- Module System
- Startup Configuration
- Multi-Tenancy
- OWIN Integration
- Debugging
- API Reference

Common Structures

- Dependency Injection
- Session
- Caching
- Logging
- Setting Management
- Timing
- Object To Object Mapping (and AutoMapper Integration)
- Email Sending (and MailKit Integration)

Domain Layer

- Entities
- Multi-Lingual Entities
- Value Objects
- Repositories
- Domain Services
- Specifications
- Unit Of Work
- Domain Events (EventBus)
- Data Filters
- Dynamic Parameter System
- Object Comparators

Application Layer

- Application Services
- Data Transfer Objects
- Validating Data Transfer Objects

In this document

Edit on GitHub

Introduction

ASP.NET Boilerplate provides an **IAbpSession** interface to obtain the current user and tenant **without** using ASP.NET's Session. IAbpSession is also fully integrated and used by other structures in ASP.NET Boilerplate such as the setting and authorization systems.

Injecting Session

IAbpSession is generally **property injected** to needed classes unless it's not possible to work without session information. If we use property injection, we can use **NullAbpSession.Instance** as a default value, as shown below:

Copy

```
public class MyClass : ITransientDependency
{
    public IAbpSession AbpSession { get; set; }

    public MyClass()
    {
        AbpSession = NullAbpSession.Instance;
    }

    public void MyMethod()
    {
        var currentUserId = AbpSession.UserId;
        //...
    }
}
```

Since authentication/authorization is an application layer task, it's advisable to **use the IAbpSession in the application layer and upper layers**. This is not generally done in the domain layer. **ApplicationService**, **AbpController, AbpApiController** and some other base classes have **AbpSession** already injected, so you can, for instance, directly use the AbpSession property in an application service method.

Session Properties

AbpSession defines a few key properties:

- **UserId**: Id of the current user or null if there is no current user. It can not be null if the calling code is authorized.

- **TenantId**: Id of the current tenant or null if there is no current tenant (in case of user has not logged in or he is a host user).
- **ImpersonatorUserId**: Id of the impersonator user, if the current session is impersonated by another user. It's null if this is not an impersonated login.
- **ImpersonatorTenantId**: Id of the impersonator user's tenant, if the current session is impersonated by another user. It's null if this is not an impersonated login.
- **MultiTenancySide**: It may be Host or Tenant.

UserId and TenantId is **nullable**. There are also the non-nullable **GetUserId()** and **GetTenantId()** methods. If you're sure there is a current user, you can call GetUserId(). If the current user is null, this method throws an exception. GetTenantId() also works in this way.

Impersonator properties are not as common as other properties and are generally used for audit logging purposes.

ClaimsAbpSession

ClaimsAbpSession is the **default implementation** of the IAbpSession interface. It gets session properties (except MultiTenancySide, it's calculated) from the claims of the current user's principal. For a cookie-based form authentication, it gets the values from cookies. Thus, it's fully integrated in to ASP.NET's authentication mechanism.

Overriding Current Session Values

In some specific cases, you may need to change/override session values for a limited scope. In such cases, you can use the IAbpSession.Use method as shown below:

Copy

```
public class MyService
{
    private readonly IAbpSession _session;

    public MyService(IAbpSession session)
    {
        _session = session;
    }

    public void Test()
    {
        using (_session.Use(42, null))
        {
            var tenantId = _session.TenantId; //42
            var userId = _session.UserId; //null
        }
    }
}
```

The Use method returns an IDisposable and it **must be disposed**. Once the return value is disposed, Session values are **automatically restored** the to previous values.

Warning!

Always use the Use method in a using block as shown above. Otherwise, you may get unexpected session values. You can have nested Use blocks and they will work as you expect.

User Identifier

You can use **.ToUserIdentifier()** extension method to create a UserIdentifier object from IAbpSession. Since UserIdentifier is used in a lot of APIs, this will simplify the creation of a UserIdentifier object for the current user.