# Domain Services



## Introduction

In a Domain Driven Design (DDD) solution, the core business logic is generally implemented in aggregates (entities) and the Domain Services. Creating a Domain Service is especially needed when;

- You implement a core domain logic that depends on some services (like repositories or other external services).
- The logic you need to implement is related to more than one aggregate/entity, so it doesn't properly fit in any of the aggregates.

## ABP Domain Service Infrastructure

Domain Services are simple, stateless classes. While you don't have to derive from any service or interface, ABP Framework provides some useful base classes and conventions.

### DomainService & IDomainService

Either derive a Domain Service from the `DomainService` base class or directly implement the `IDomainService` interface.

**Example: Create a Domain Service deriving from the `DomainService` base class.**

```
using Volo.Abp.Domain.Services;

namespace MyProject.Issues
{
    public class IssueManager : DomainService
    {

    }
}
```

When you do that;

- ABP Framework automatically registers the class to the Dependency Injection system with a Transient lifetime.
- You can directly use some common services as base properties, without needing to manually inject (e.g. ILogger and IGuidGenerator).

> It is suggested to name a Domain Service with a `Manager` or `Service` suffix. We typically use the `Manager` suffix as used in the sample above.

**Example: Implement the domain logic of assigning an Issue to a User**

4.1 (latest) English

Filter topics

- Getting Started
- Startup Templates
- Tutorials
- Fundamentals
- Infrastructure
- Architecture
  - Modularity
  - Domain Driven Design
    - Overall
    - Domain Layer
      - Entities & Aggregate Roots
      - Value Objects
      - Repositories
      - Domain Services
      - Specifications
    - Application Layer
    - Guide: Implementing DDD
  - Multi Tenancy
  - Microservices
- API
- User Interface
- Data Access
- Real Time
- Testing
- Samples
- Application Modules
- Release Information
- Reference
- Contribution Guide

```
public class IssueManager : DomainService
{
    private readonly IRepository<Issue, Guid> _issueRep

    public IssueManager(IRepository<Issue, Guid> issueR
    {
        _issueRepository = issueRepository;
    }

    public async Task AssignAsync(Issue issue, AppUser
    {
        var currentIssueCount = await _issueRepository
            .CountAsync(i => i.AssignedUserId == user.I

        //Implementing a core business validation
        if (currentIssueCount >= 3)
        {
            throw new IssueAssignmentException(user.Use
        }

        issue.AssignedUserId = user.Id;
    }
}
```

Issue is an aggregate root defined as shown below:

```
public class Issue : AggregateRoot<Guid>
{
    public Guid? AssignedUserId { get; internal set; }

    //...
}
```

- Making the setter `internal` ensures that it can not directly set in the upper layers and forces to always use the `IssueManager` to assign an `Issue` to a `User`.

## Using a Domain Service

A Domain Service is typically used in an application service.

**Example: Use the `IssueManager` to assign an Issue to a User**

Share on :

In this document

4.1 (latest) English

Filter topics

- Getting Started
- Startup Templates
- Tutorials
- Fundamentals
- Infrastructure
- Architecture
  - Modularity
  - Domain Driven Design
    - Overall
    - Domain Layer
      - Entities & Aggregate Roots
      - Value Objects
      - Repositories
      - Domain Services
      - Specifications
    - Application Layer
    - Guide: Implementing DDD
  - Multi Tenancy
  - Microservices
- API
- User Interface
- Data Access
- Real Time
- Testing
- Samples
- Application Modules
- Release Information
- Reference
- Contribution Guide

```
using System;
using System.Threading.Tasks;
using MyProject.Users;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace MyProject.Issues
{
    public class IssueAppService : ApplicationService,
    {
        private readonly IssueManager _issueManager;
        private readonly IRepository<AppUser, Guid> _us
        private readonly IRepository<Issue, Guid> _issu

        public IssueAppService(
            IssueManager issueManager,
            IRepository<AppUser, Guid> userRepository,
            IRepository<Issue, Guid> issueRepository)
        {
            _issueManager = issueManager;
            _userRepository = userRepository;
            _issueRepository = issueRepository;
        }

        public async Task AssignAsync(Guid id, Guid use
        {
            var issue = await _issueRepository.GetAsync
            var user = await _userRepository.GetAsync(u

            await _issueManager.AssignAsync(issue, user
            await _issueRepository.UpdateAsync(issue);
        }
    }
}
```

Since the `IssueAppService` is in the Application Layer, it can't directly assign an issue to a user. So, it uses the `IssueManager` .

# Application Services vs Domain Services

While both of Application Services and Domain Services implement the business rules, there are fundamental logical and formal differences;

- Application Services implement the **use cases** of the application (user interactions in a typical web application), while Domain Services implement the **core, use case independent domain logic**.
- Application Services get/return Data Transfer Objects, Domain Service methods typically get and return the **domain objects** (entities, value objects).
- Domain services are typically used by the Application Services or other Domain Services, while Application Services are used by the Presentation Layer or Client Applications.

# Lifetime

Share on :

In this document

4.1 (latest) English

Filter topics

- Getting Started
- Startup Templates
- Tutorials
- Fundamentals
- Infrastructure
- Architecture
  - Modularity
  - Domain Driven Design
    - Overall
    - Domain Layer
      - Entities & Aggregate Roots
      - Value Objects
      - Repositories
      - Domain Services
      - Specifications
    - Application Layer
    - Guide: Implementing DDD
  - Multi Tenancy
  - Microservices
- API
- User Interface
- Data Access
- Real Time
- Testing
- Samples
- Application Modules
- Release Information
- Reference
- Contribution Guide

Lifetime of Domain Services are transient and they are automatically registered to the dependency injection system.

Share on :

In this document