

Setting Up the Application Core



Gill Cleeren

CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview



Understanding the business requirements

Setting up the solution

Creating the domain

Designing the application project

- Contracts
- Packages
- Validation
- Exceptions





LOBOTICKET

A Globomantics Company



You've Seen The Finished Application

GloboTicket Ticket Management

Event Overview

ADD EVENT EXPORT EVENTS

EVENT NAME	EVENT DATE	
Clash Of The DJs	2/28/2023	
Spanish Guitar Hits With Manuel	2/28/2023	
John Egbert Live	4/30/2023	
To The Moon And Back	6/30/2023	
The State Of Affairs: Michael Live!	7/30/2023	

Home Events Categories Add category Sales

LOBOTICKET
A Globomantics Company



Understanding the Business Requirements





“Hello, I’m Mary Goodsale.

**Let’s talk about the
new system we need to build!”**



Requirements for Ticket Management



Manage events

Overview of events in their categories

Orders for the different events



Requirements for Ticket Management



Manage events

Overview of events in their categories

Orders for the different events



Wireframes for the Application



Globoticket Ticket Management

Events

Categories

Sales

User name

Password



Event Management

 GLOBOTICKET
A Grupo Globo Company

Globoticket Ticket Management

Events			
Events			
Categories	Event A	01/01/2021	Details
Sales		Event A	Details
Sales		Event B	Details
Sales		Event C	Details

 GLOBOTICKET
A Grupo Globo Company

Globoticket Ticket Management

Event A

Events

Categories

Event name: Event A

Sales

Ticket price: 100

Artist name: DJ 'The Mike'

Event date: / / 

Description: Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut

Image:

Category: Concert ▾

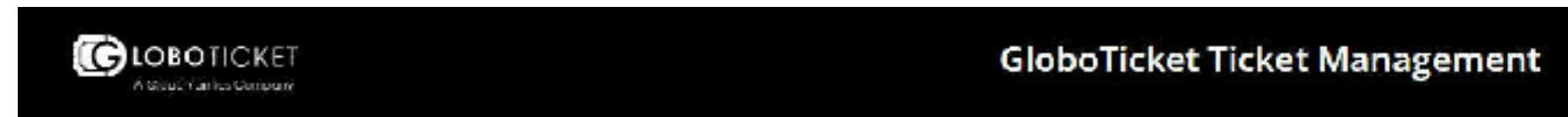
[Save event](#) [Delete event](#)



Category Management



Ticket Sales



Sales

Events

Categories

Sales

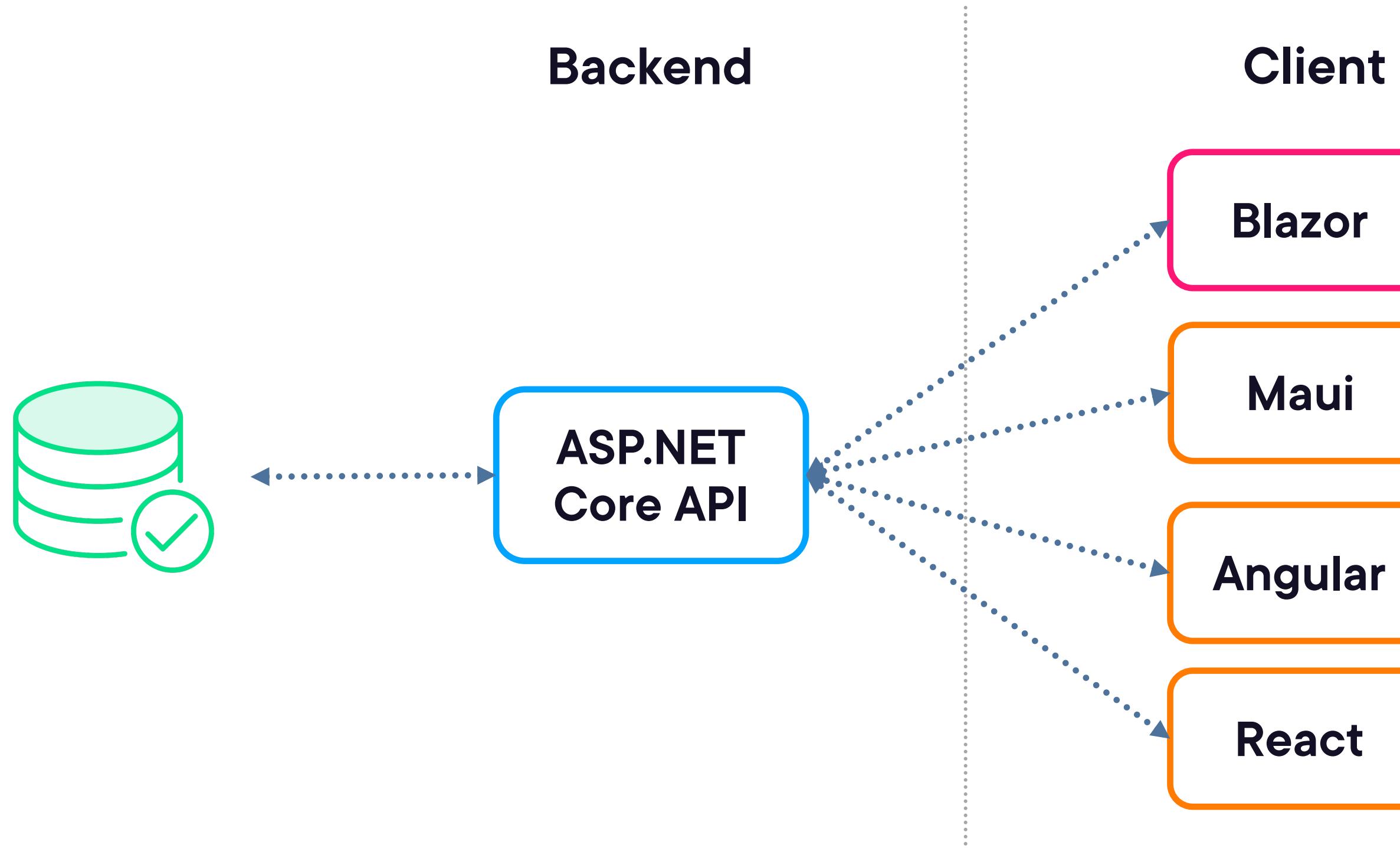
Month	Year	Get sales
January	2021	<input type="button" value="Get sales"/>

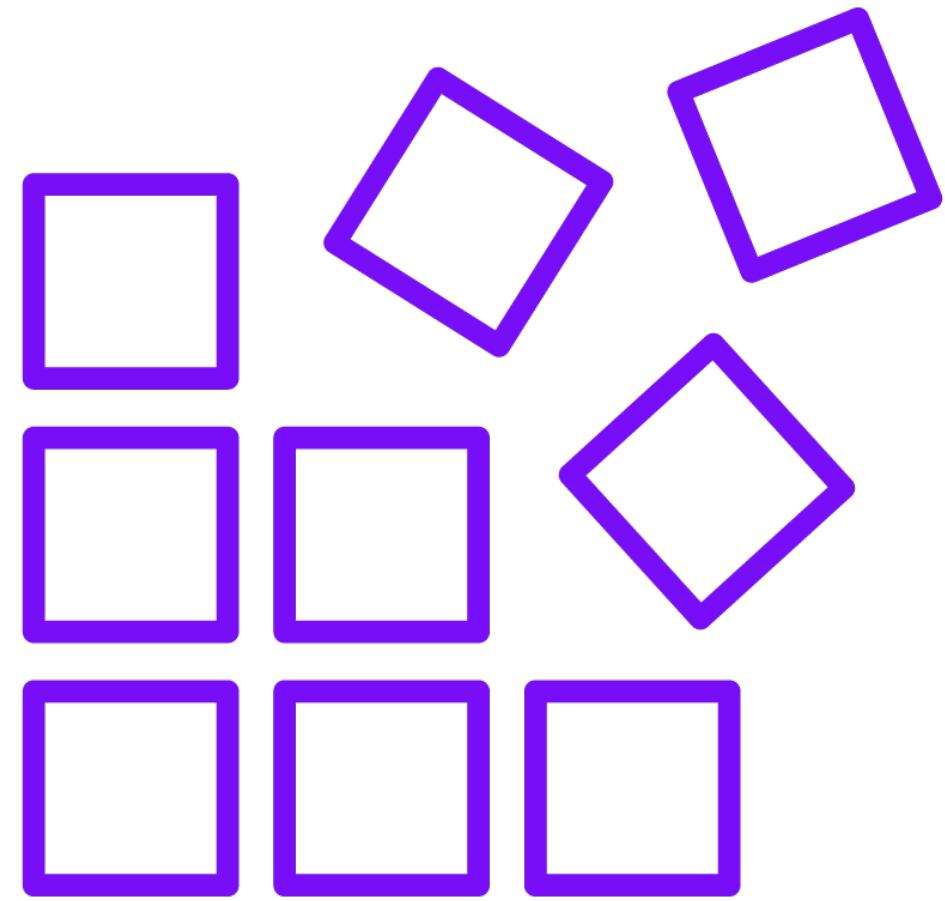
Order date	Amount
16/01/2021	€155
16/01/2021	€174
16/01/2021	€177
16/01/2021	€174



Setting up the Application Architecture

Translating to an Application Architecture





REST API

- Built using ASP.NET Core 8
- Clean architecture principles
- Data access using EF Core 8

Client

- Built using Blazor WebAssembly (ASP.NET Core 8)

Class libraries

- .NET Class Libraries

All independent of .NET version



Demo



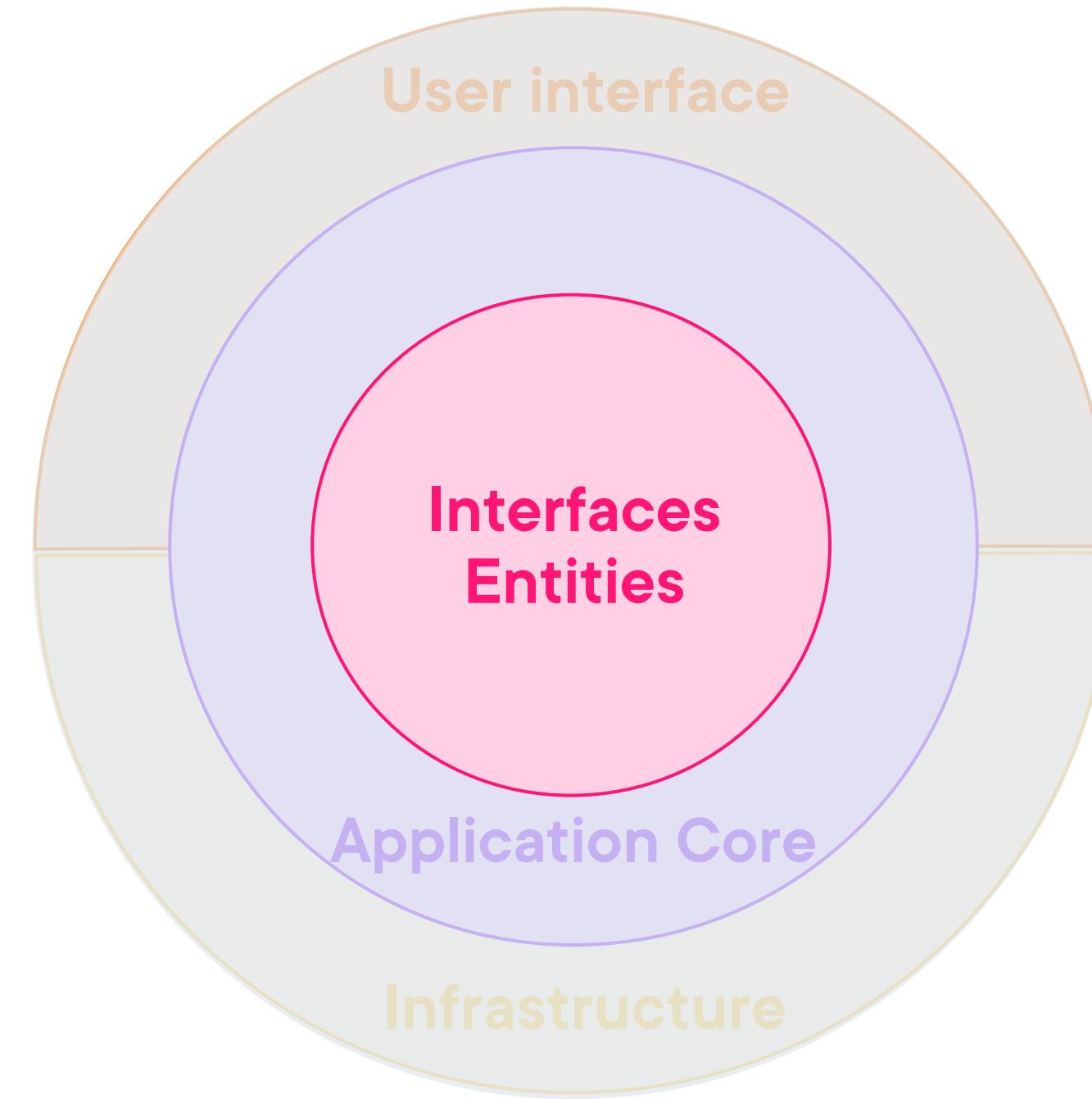
Creating the Visual Studio solution



Creating the Domain Project



The Domain Project



Demo



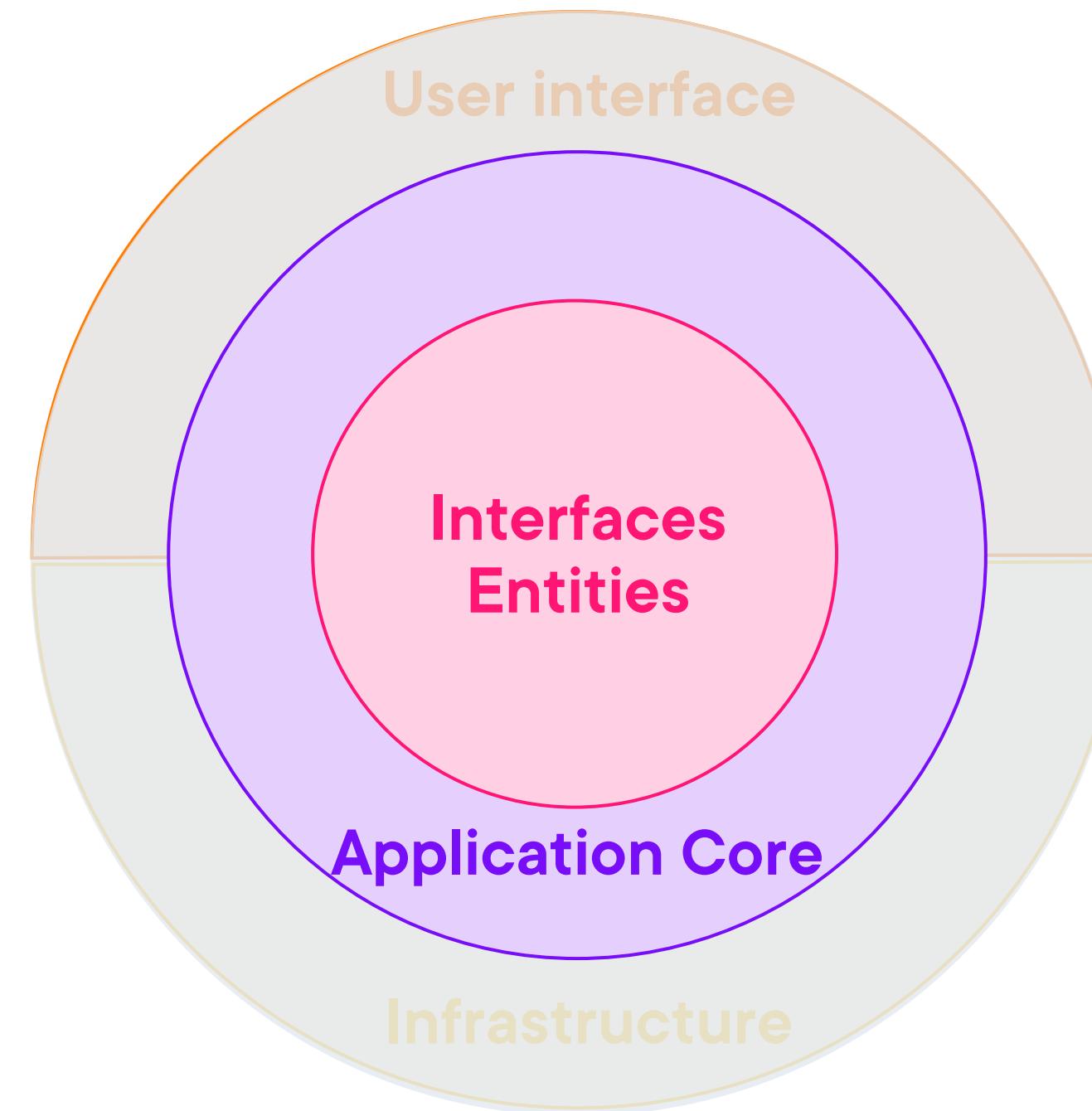
Creating the domain project

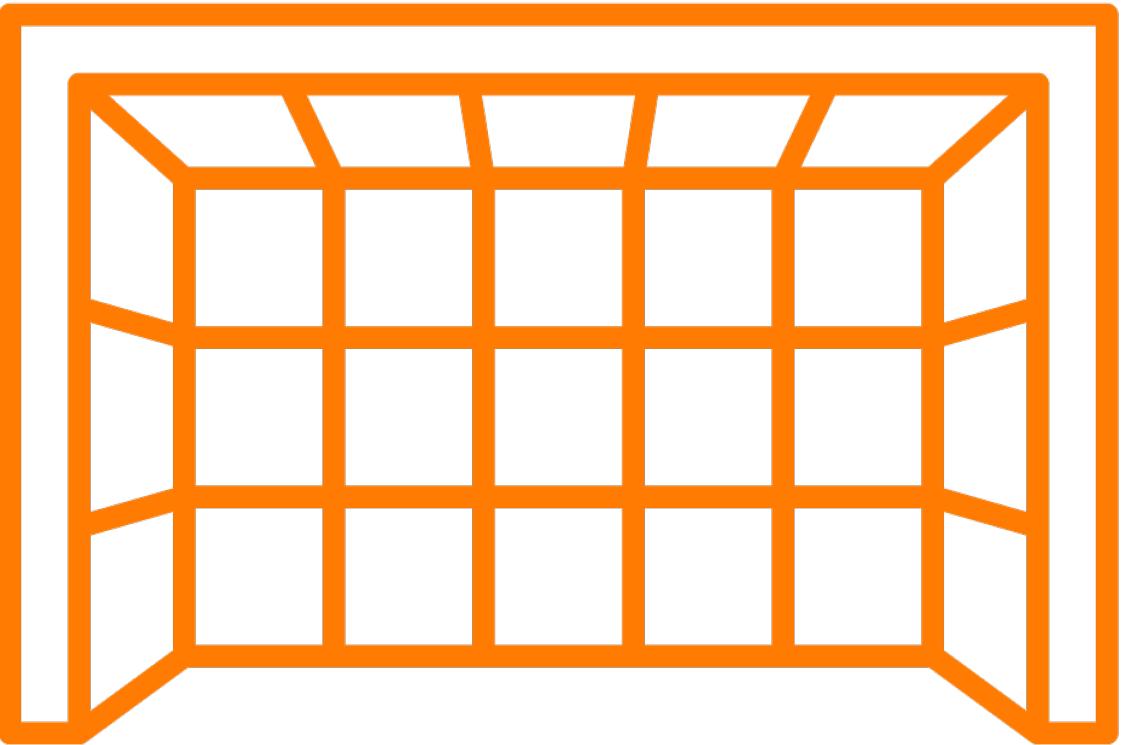


Designing the Application Project



The Application Project

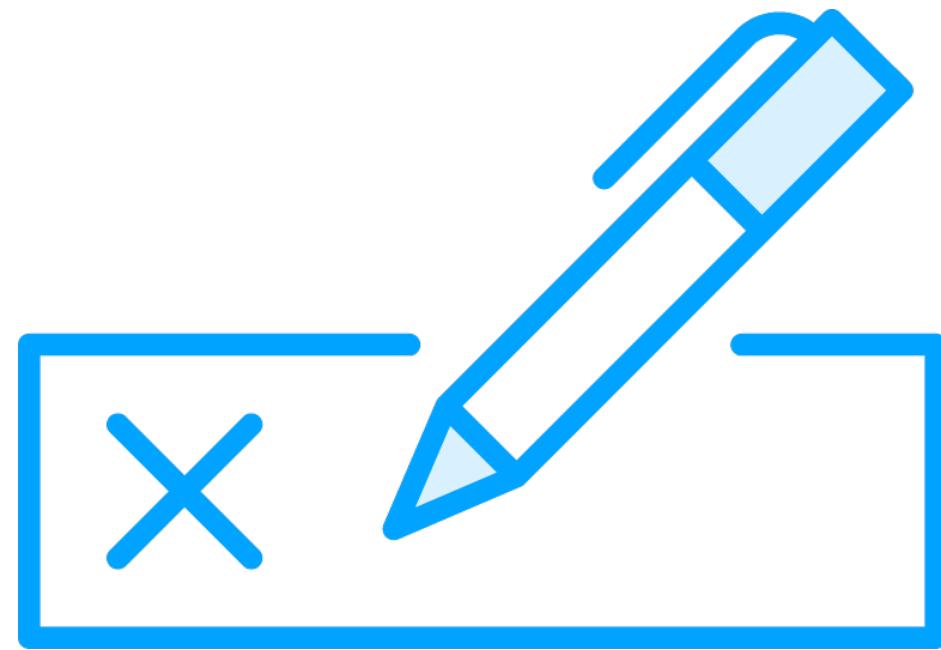




Achieving loose coupling in the application core

- Contracts
- Messaging





Contracts

- Part of application core
- Functionality is described in interfaces
- Implemented in Core or Infrastructure





Using Repositories

Mediates between domain and data-mapping layer

Often used in combination with UOW



Using Repositories

Data access operations

Agnostic for rest of application

Generic methods

Specific repositories



Demo



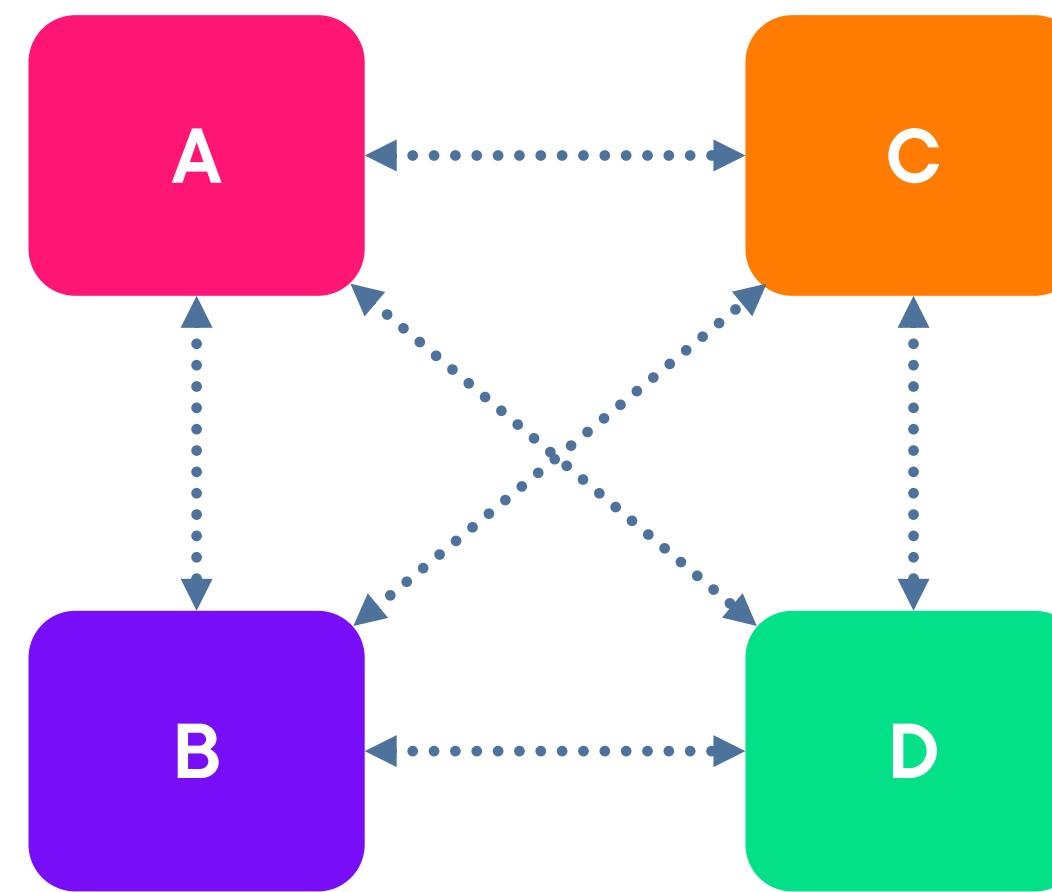
Creating the Application project

Adding contracts

Introducing the foundation for the repository



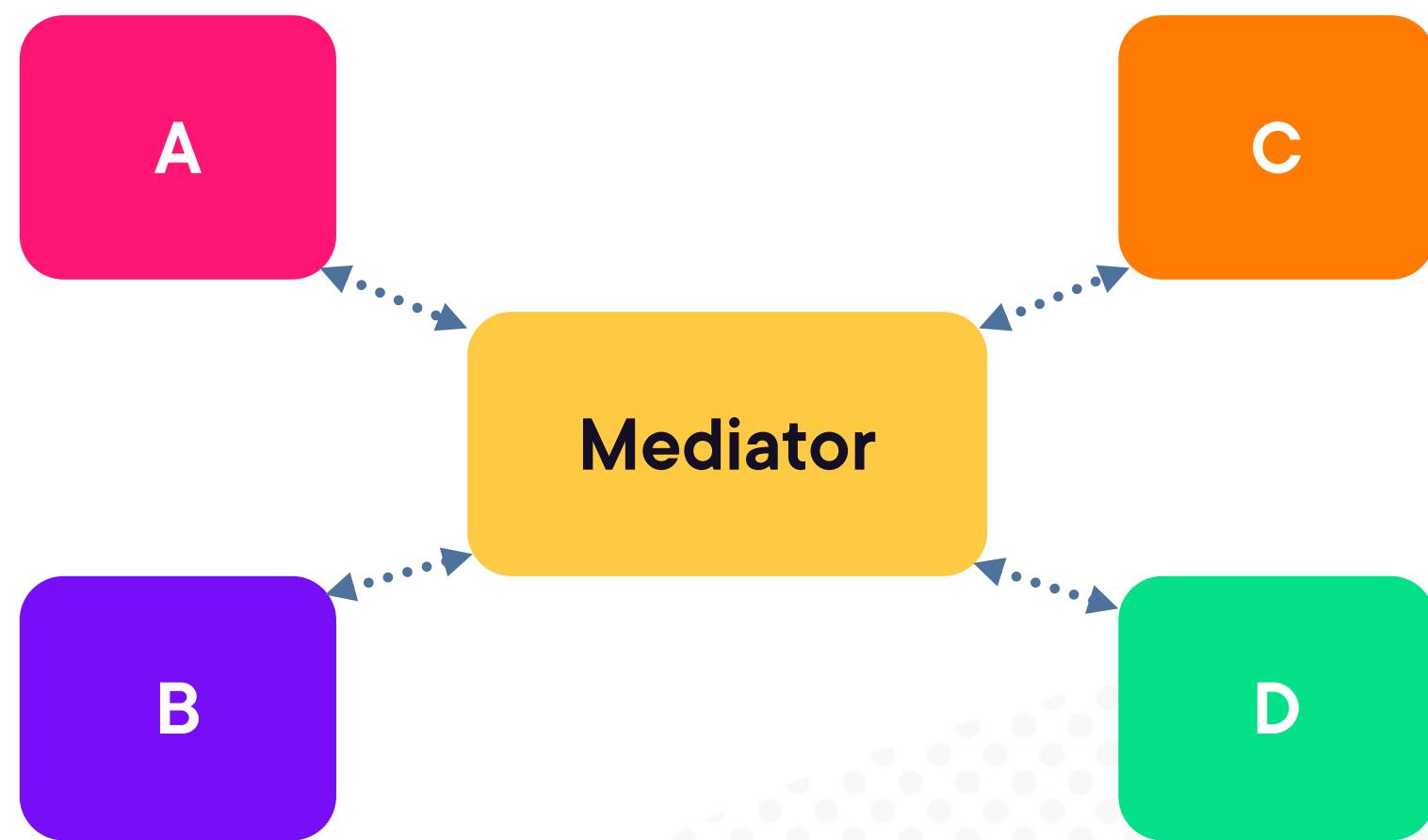
Introducing a Mediator



**Object that wraps what
how objects need to
interact**

**Avoid hard references
from one object to the
next**

**Help with communication
from/to Core project
objects**



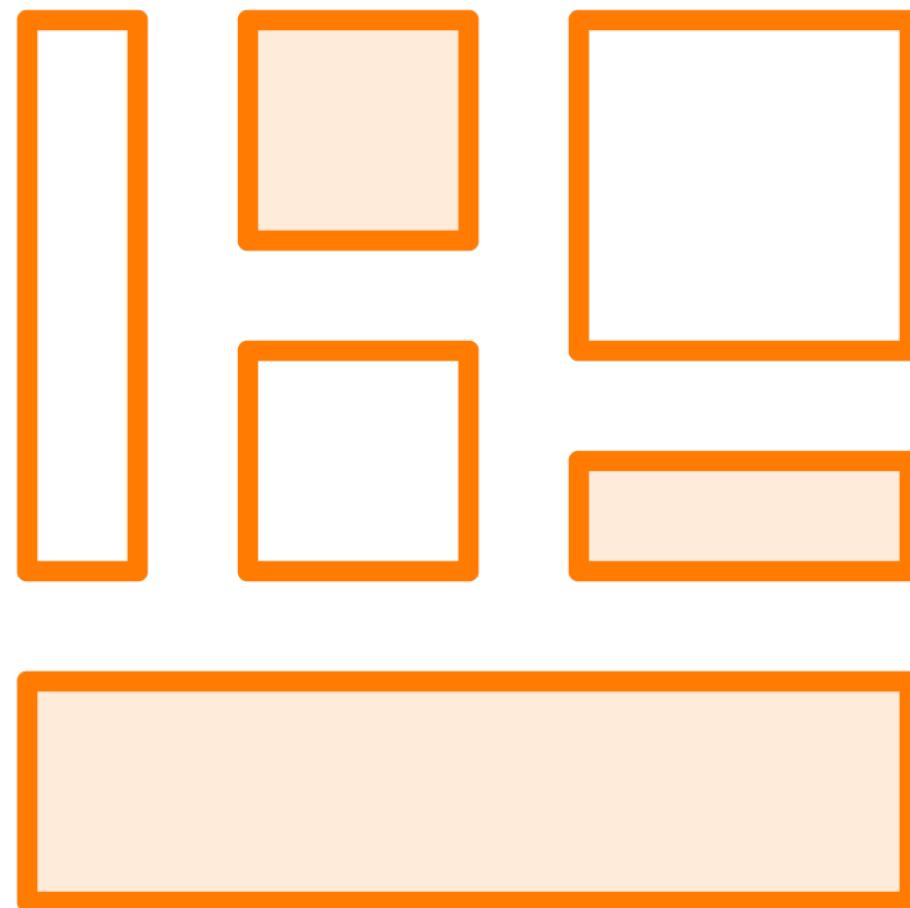
Advantages of Using a Mediator

Changes can be handled easily

Easy to test the object



Using MediatR



Simple mediator implementation in .NET

- github.com/jbogard/MediatR

Adding MediatR

- Install-Package MediatR

Using

- IRequest
- IRequestHandler



```
public class GetEventsListQuery: IRequest<List<EventListVm>>
{
}
```

Creating a Request



```
public class GetEventsListQueryHandler :  
    IRequestHandler<GetEventsListQuery, List<EventListVm>>  
{  
    public async Task<List<EventListVm>> Handle  
        (GetEventsListQuery request, CancellationToken cancellationToken)  
    { }  
}
```

Defining the Request Handler



What We Aren't Using of MediatR

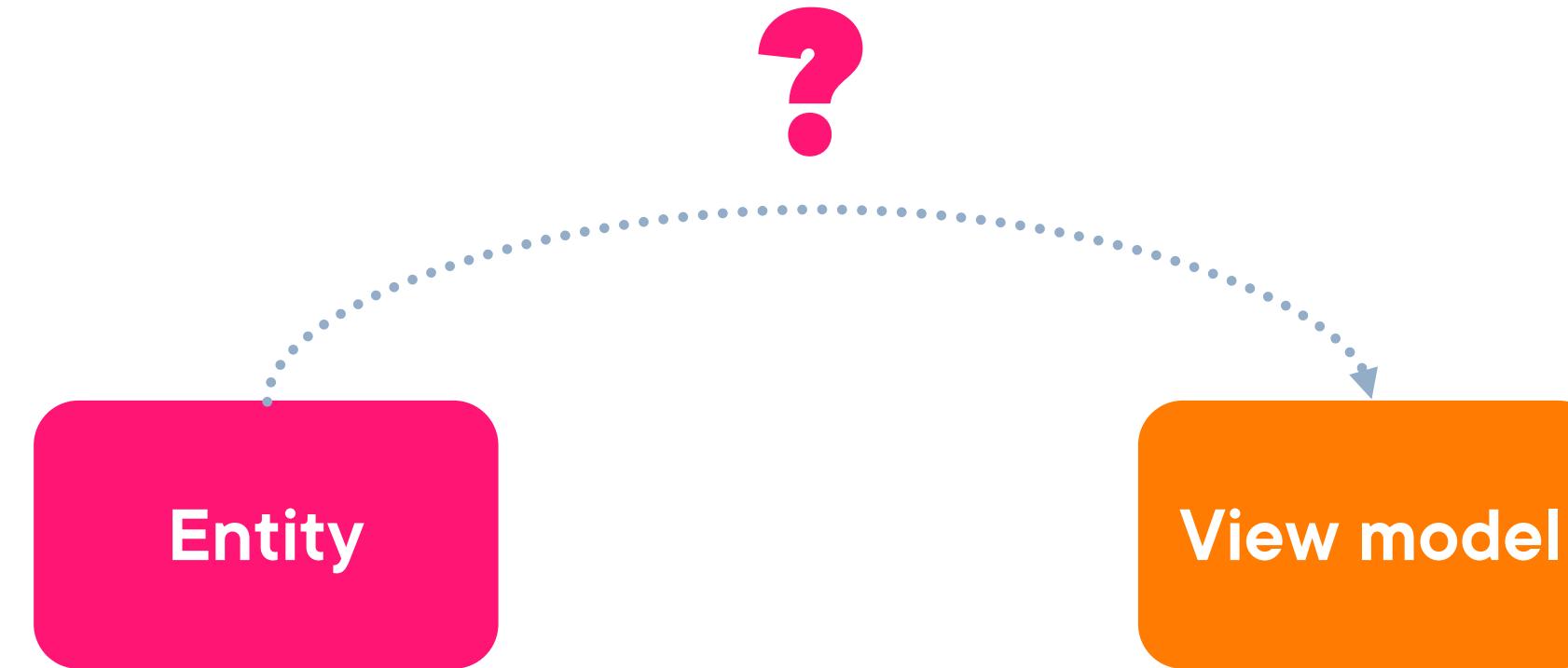


Pipeline behaviour

- Logging
- Validation
- Caching



Mapping from an Entity to a View Model





Mapping Between Objects

AutoMapper library

Mapping from one type to another type



Using AutoMapper

NuGet package

Startup registration
in DI

Profiles



```
var result = _mapper.Map<List<EventListVm>>(allEvents);
```

Mapping with AutoMapper



Demo



Introducing MediatR and AutoMapper
Creating a request and request handler
Adding a ServiceCollection extension class



Reading and Writing Data



Same model is used to read and write data

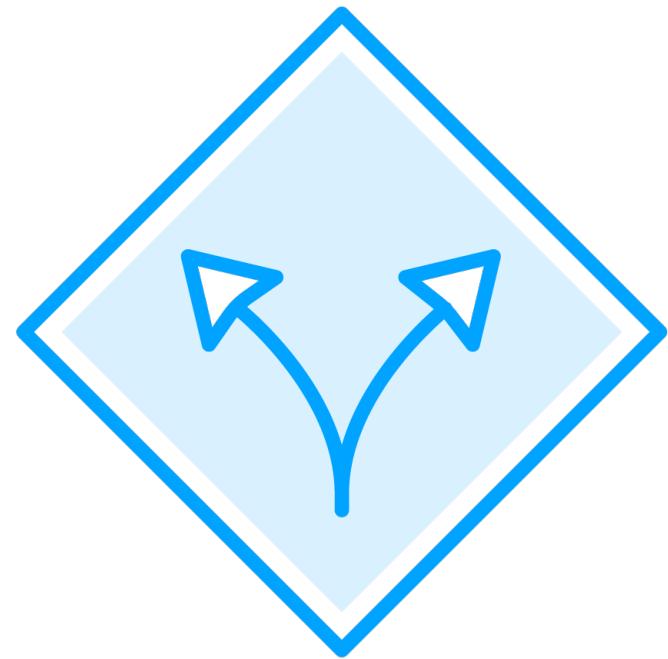
Issues in larger applications

- Different queries
- Different object being returned
- Complex logic for saving entities
- Security may be different

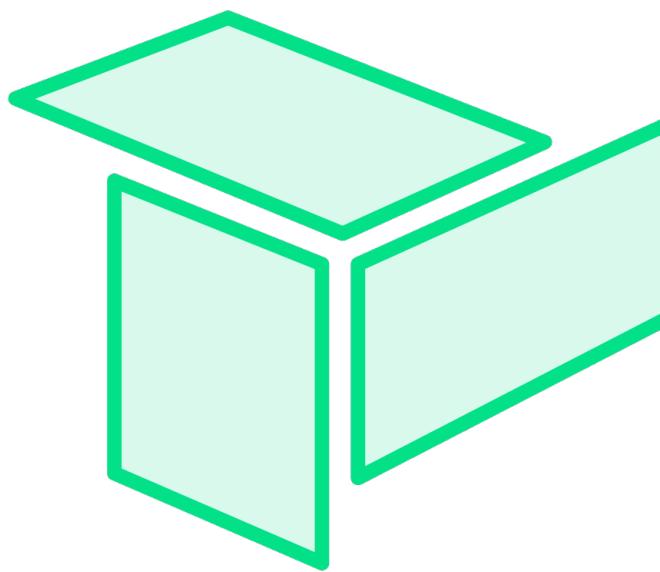
Model may become too heavy



Adding Simple CQRS



**Command-Query
Responsibility
Segregation**



**Different models
Commands to update
data
Queries to read data**



**Commands are task-
based
Can be asynchronous**





Advantages

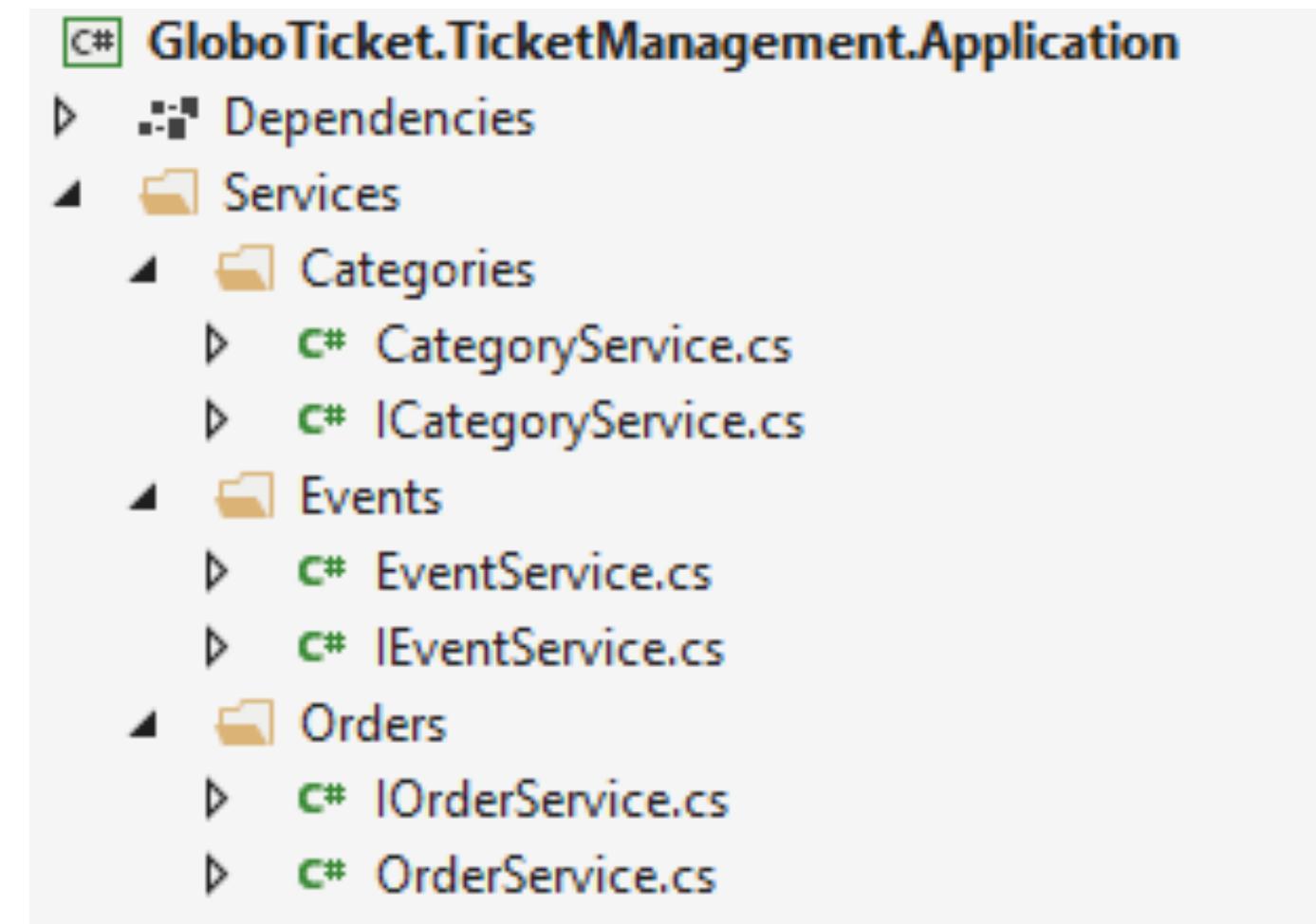
- Separation of concerns
- Scaling
- Security
- Easy to make a change, no further impact

Disadvantages

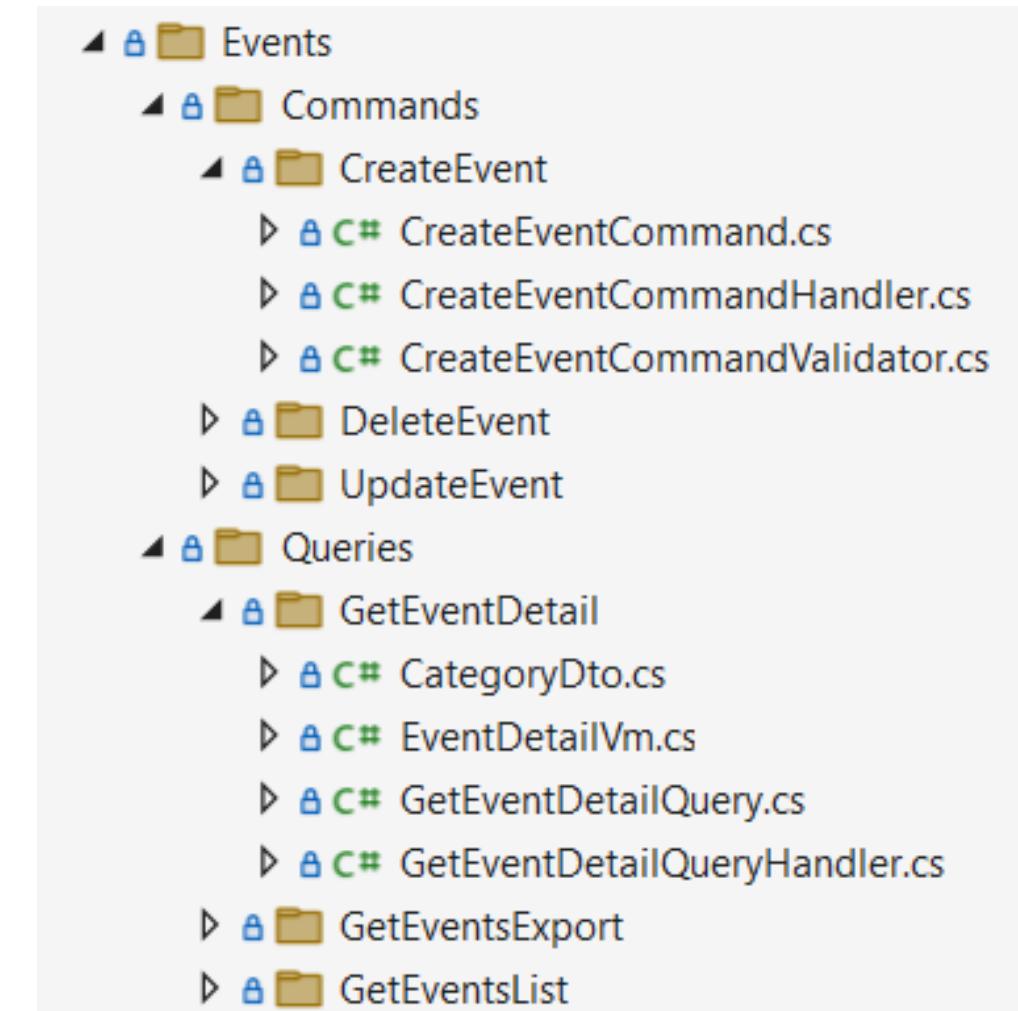
- Added complexity
- Targeted at more complex applications



What Problem Are We Solving?



Translating Our Requirements to CQRS



Demo



Adding simple CQRS



Feature-based Approach



Vertical slice

Features folder and subfolders

Own the view models they will use

- Not shared typically, even if identical



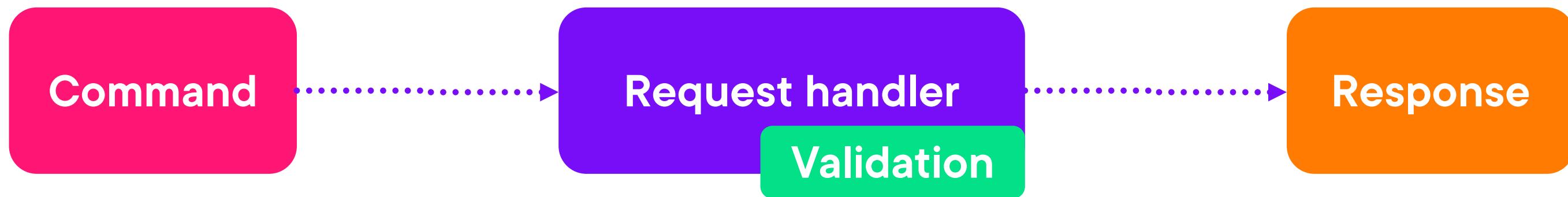
Demo



Splitting up in features



Creating a New Entity



Demo



Creating a new Event
Updating and deleting an Event



Validation

Event.cs

```
public class Event: AuditableEntity
{
    public Guid EventId { get; set; }

    [Required]
    [StringLength(50)]
    public string Name { get; set; }
    public int Price { get; set; }

    [Required]
    [StringLength(50)]
    public string Artist { get; set; }
}
```





Adding Fluent Validation

- Nuget package
- Uses lambda expressions for validation rules

Can be used in Core project

- RequestHandler
- Part of the Feature folder



```
public class CreateCategoryCommandValidator :  
    AbstractValidator<CreateCategoryCommand>  
{  
    public CreateCategoryCommandValidator()  
    {  
        RuleFor(p => p.Name)  
            .NotEmpty().WithMessage("{PropertyName} is required.");  
    }  
}
```

Adding a Custom Validator



```
var validator = new CreateCategoryCommandValidator();
var validationResult = await validator.ValidateAsync(request);
```

Using the Validator



Returning Exceptions



Core should return own set of exceptions

- Can be handled or transformed by consumer

Used exceptions

- NotFoundException
- BadRequestException
- ValidationException

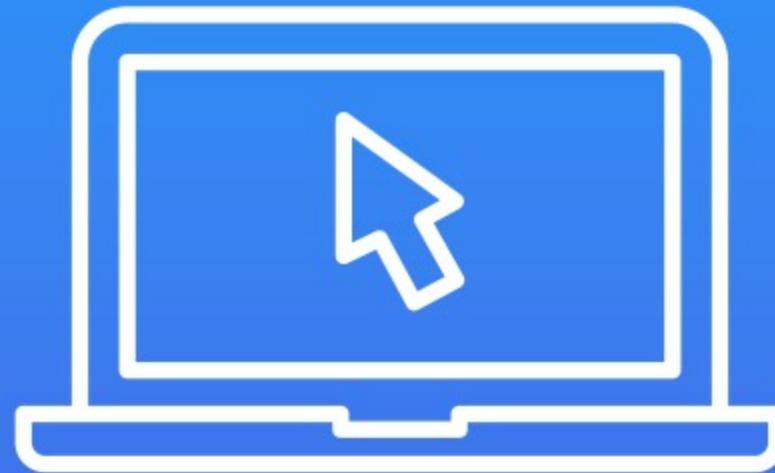


```
public class NotFoundException : Exception
{
    public NotFoundException(string name)
        : base($"{name} is not found")
    {
    }
}
```

Adding Custom Exceptions



Demo



Validating data input
Using custom exceptions
Returning a response object



Summary



Core contains the core functionality of the application

CQRS and MediatR help with achieving high-level of loose-coupling

Feature-based helps with organization of code

Validation using Fluent Validation





Implementations not included!

Don't include concrete implementations for anything infrastructure-related.



Up Next:

Bringing in the application infrastructure

