# Tame Cross-Cutting Concerns in Your Code!

Steve Smith

@ardalis

steve@nimblepros.com | NimblePros.com

NimblePros

Better software development. Delivered.

## What’s a Concern in Software?

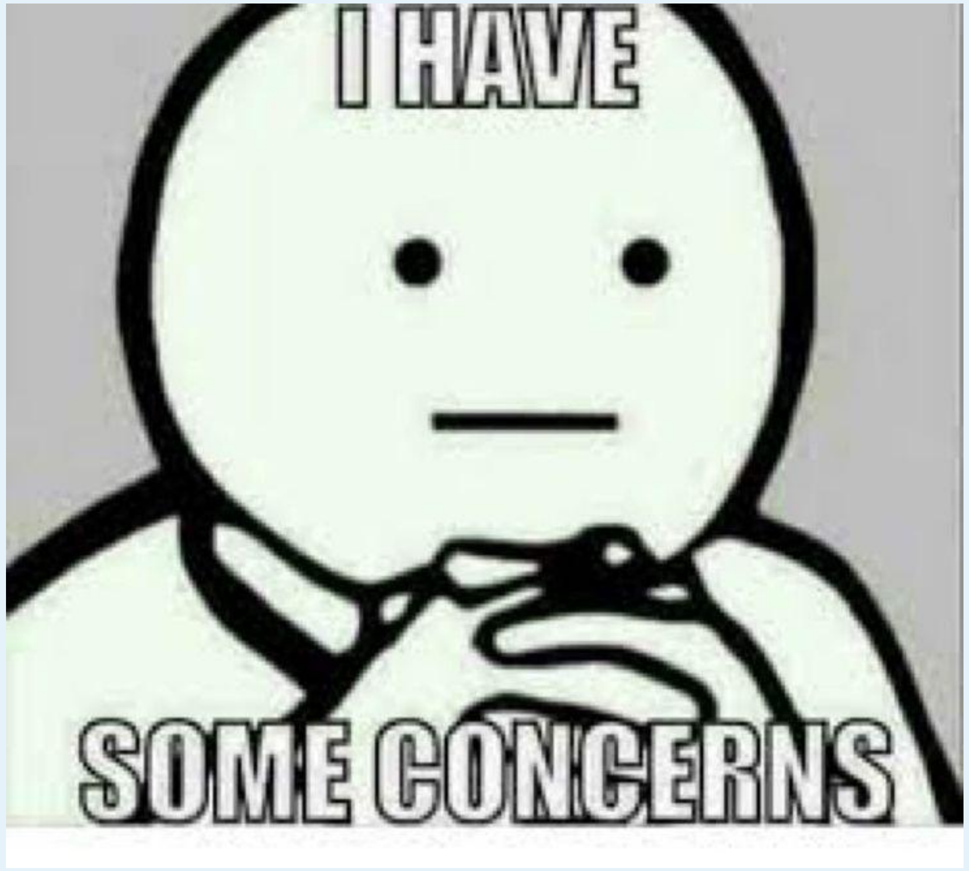
- Some logic that shares a particular purpose
- Frequently will change independently from other code
- May have different levels of abstraction

## Separation of Concerns

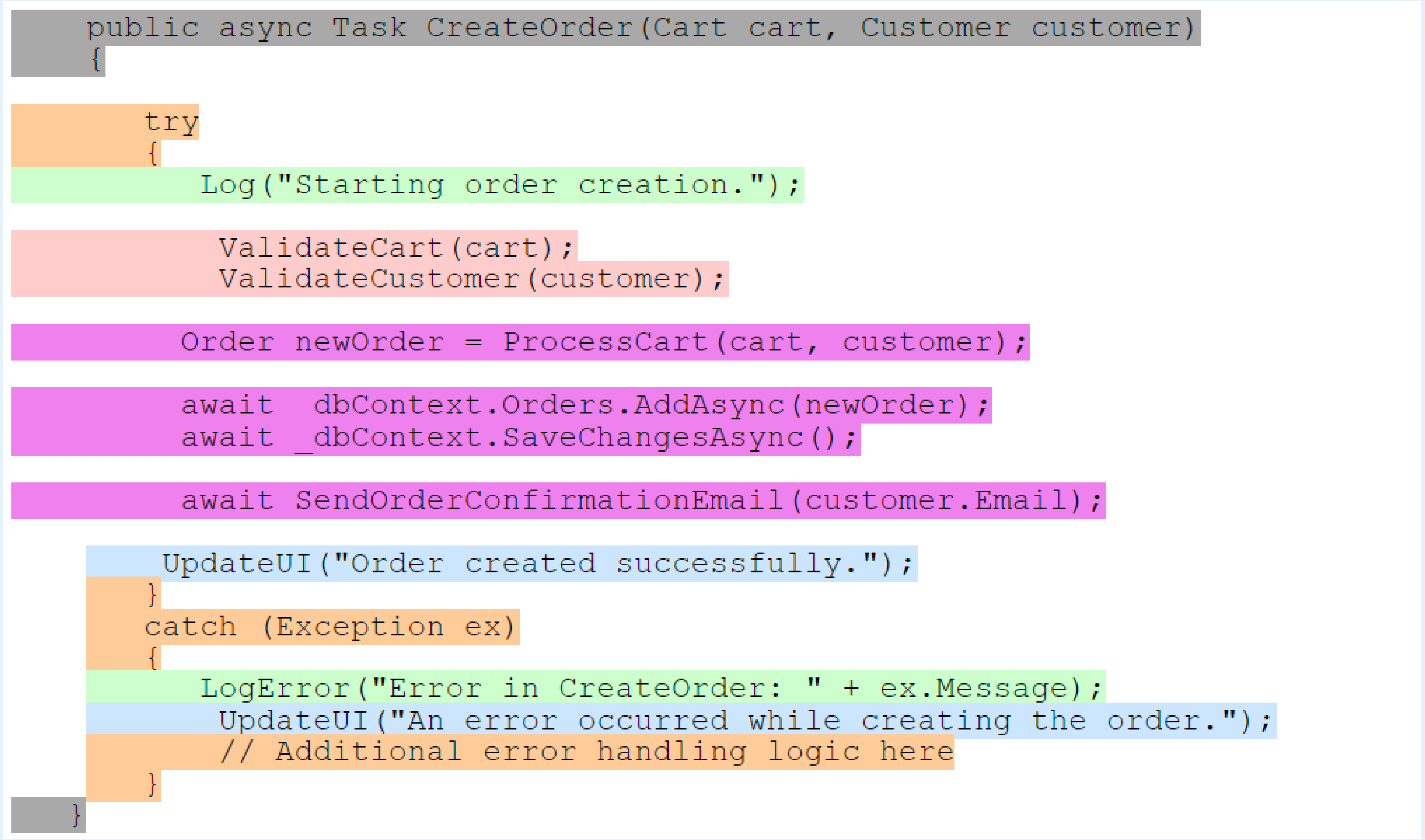
Don’t let your plumbing code pollute your software.

## Common **Factors** or Concerns

- Validation
- Error Handling
- Logging
- Data Access
- Business Logic
- UI Logic
- Authorization
- *and many others*

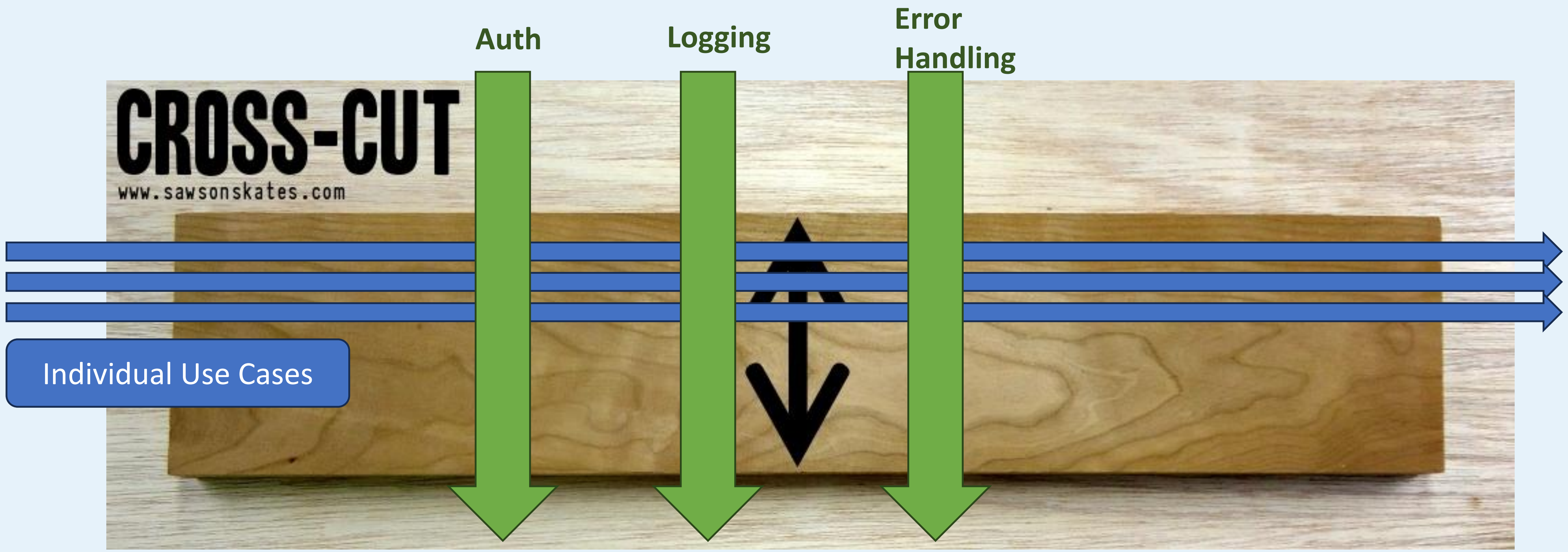


# We can use tools to visually identify different concerns:



```
    public async Task CreateOrder(Cart cart, Customer customer)
    {

        try
        {
            Log("Starting order creation.");

              ValidateCart(cart);
              ValidateCustomer(customer);

            Order newOrder = ProcessCart(cart, customer);

            await  dbContext.Orders.AddAsync(newOrder);
            await _dbContext.SaveChangesAsync();

            await SendOrderConfirmationEmail(customer.Email);

         UpdateUI("Order created successfully.");
        }
        catch (Exception ex)
        {
            LogError("Error in CreateOrder: " + ex.Message);
              UpdateUI("An error occurred while creating the order.");
              // Additional error handling logic here
        }
    }
```

# What are Cross-Cutting Concerns?



## Demo:
## Cross-Cutting Concerns

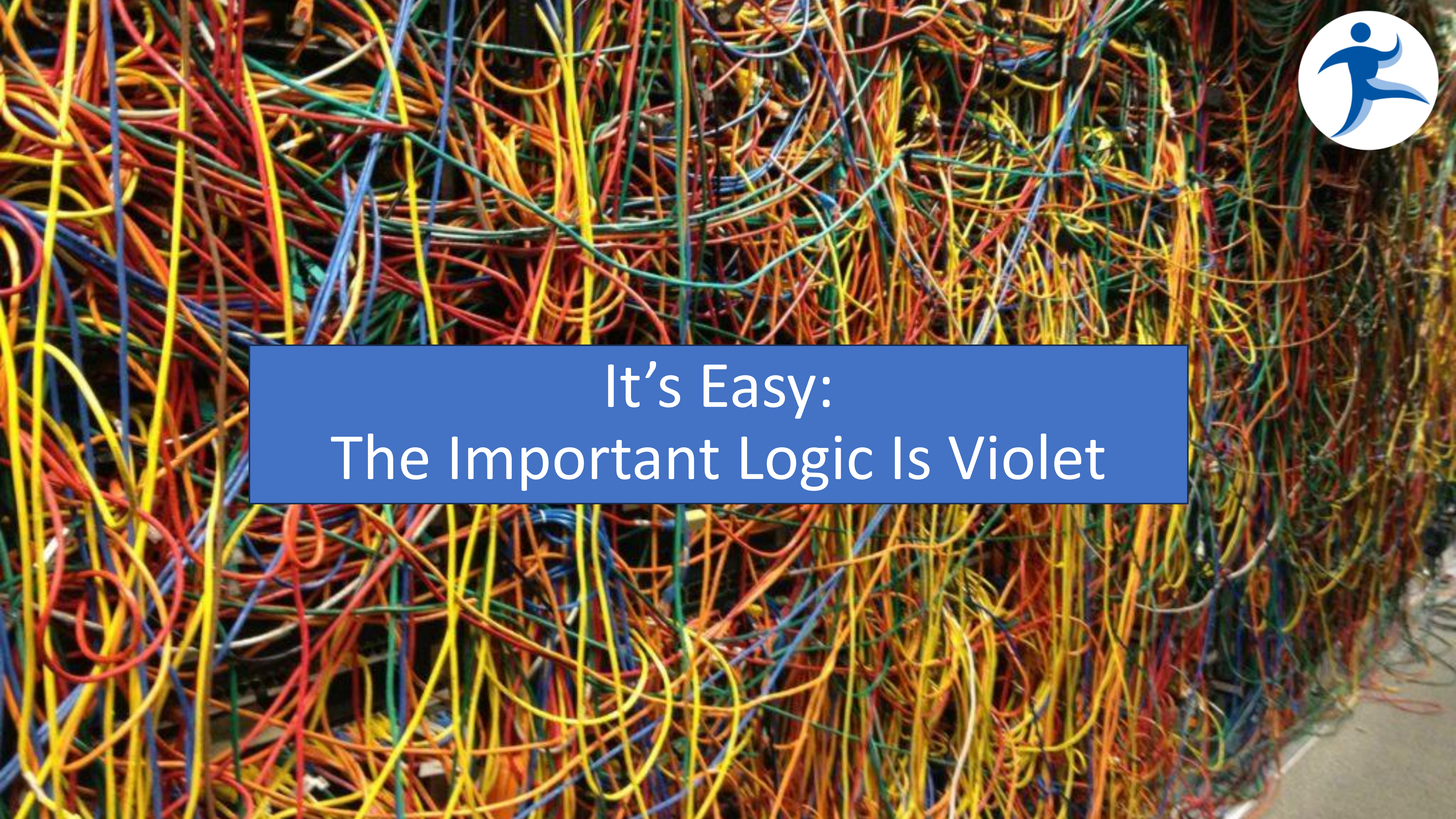
## In Color

RoleService.htm

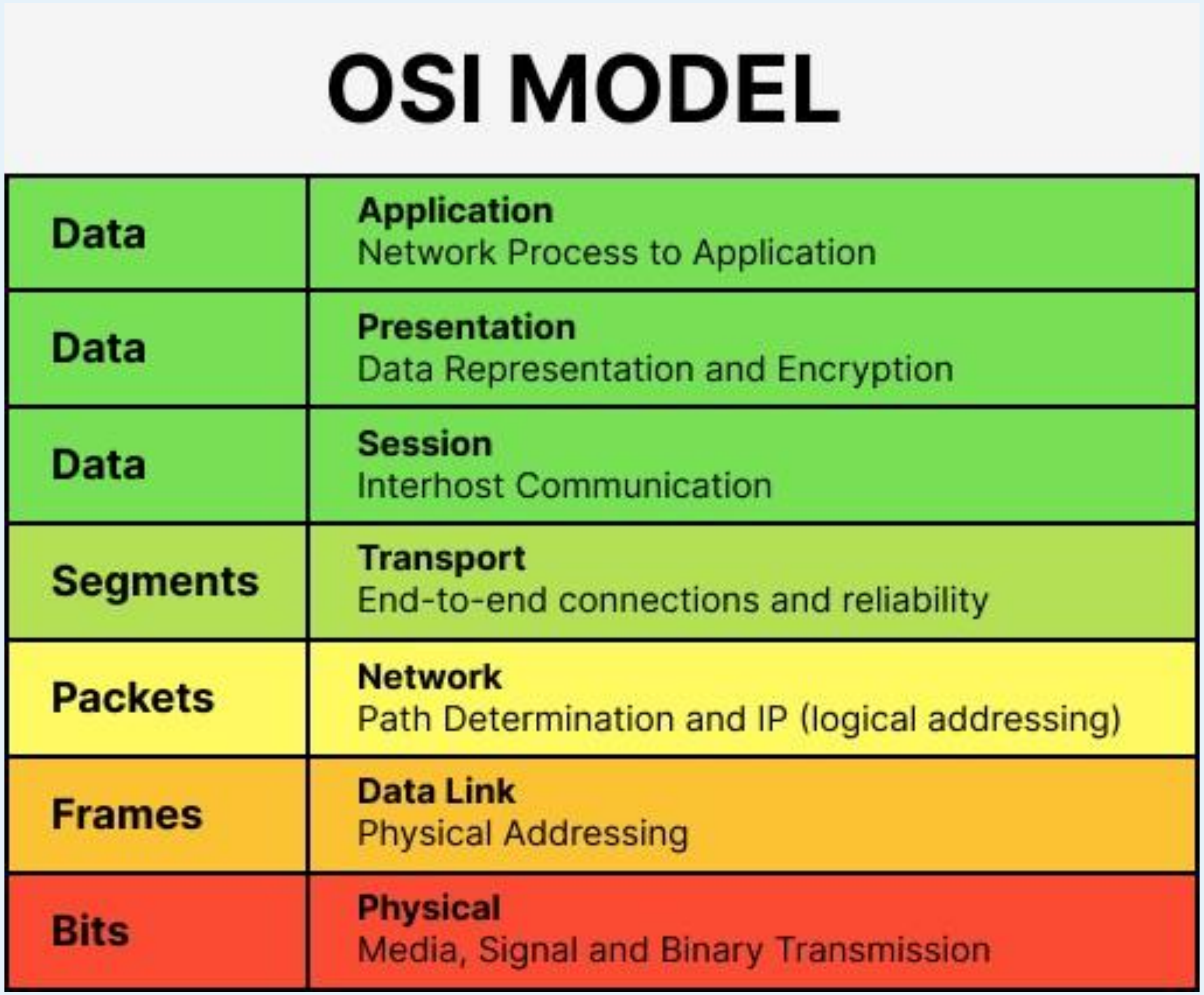
## What’s Going On Here?

- Access Checks
- Logging
- Exception Handling
- Caching
- (Validation)

- Oh yeah, also we’re returning a list of roles



## It’s Easy:
## The Important Logic Is Violet



# OSI MODEL

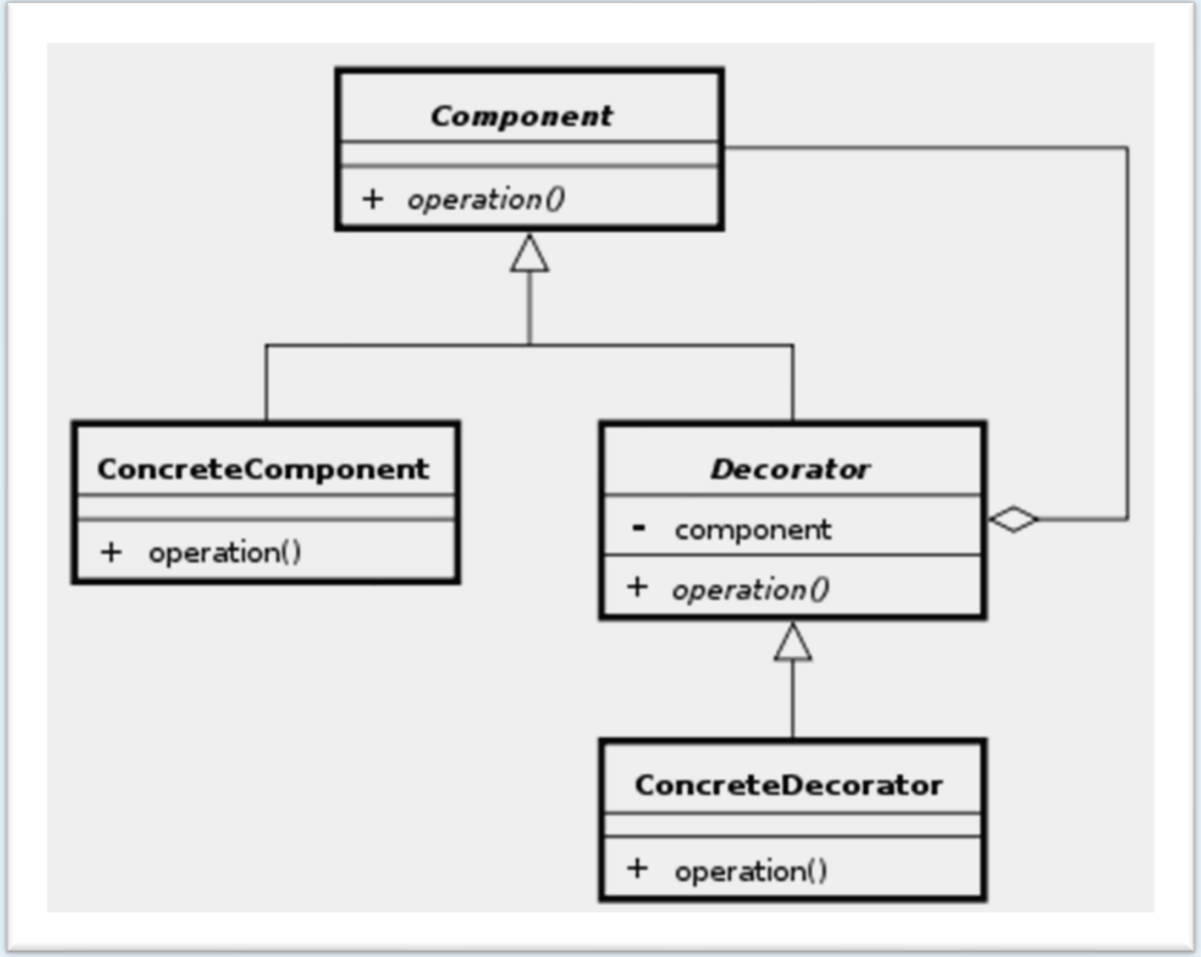
| | |
|---|---|
| **Data** | **Application** Network Process to Application |
| **Data** | **Presentation** Data Representation and Encryption |
| **Data** | **Session** Interhost Communication |
| **Segments** | **Transport** End-to-end connections and reliability |
| **Packets** | **Network** Path Determination and IP (logical addressing) |
| **Frames** | **Data Link** Physical Addressing |
| **Bits** | **Physical** Media, Signal and Binary Transmission |

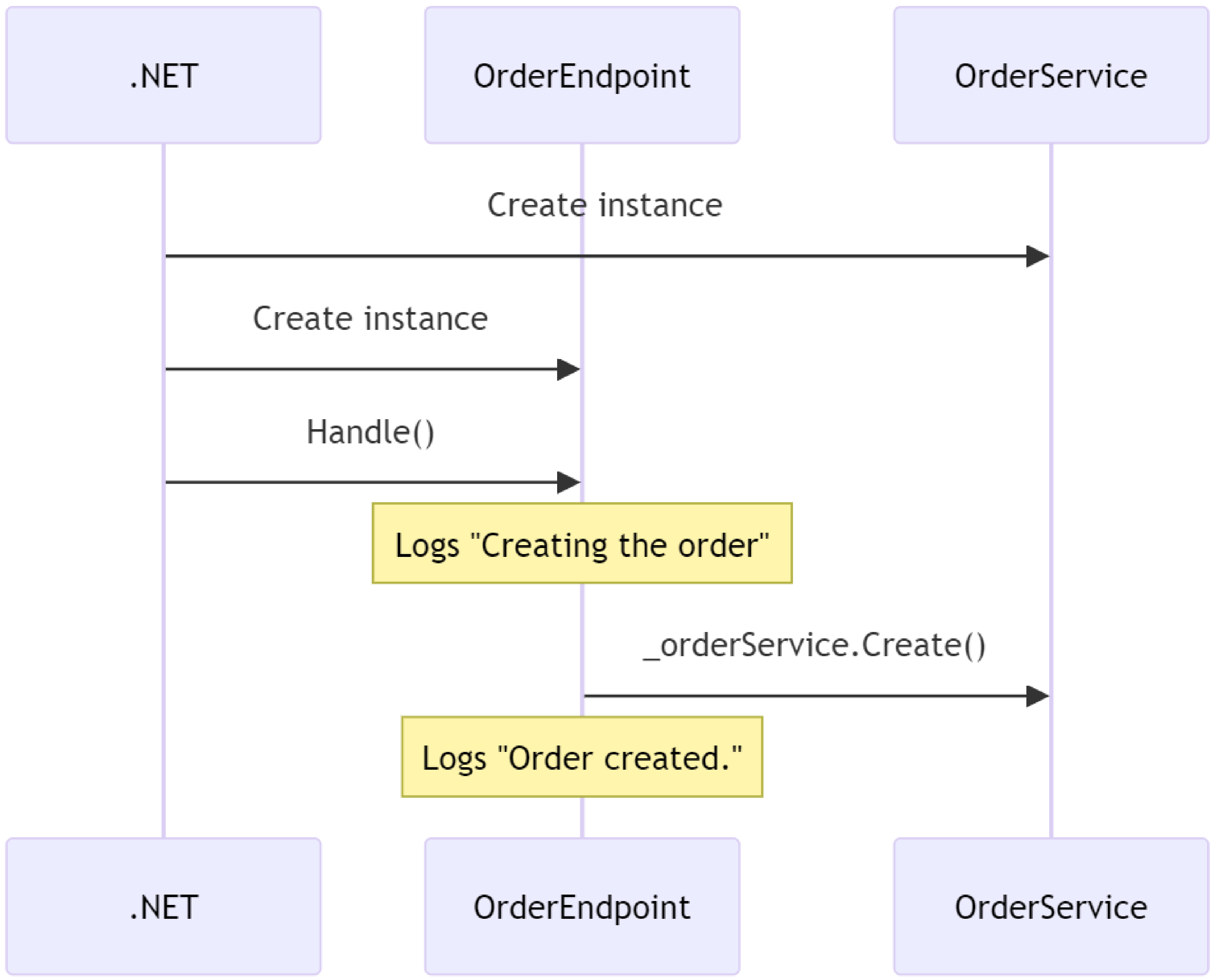
## Decorator

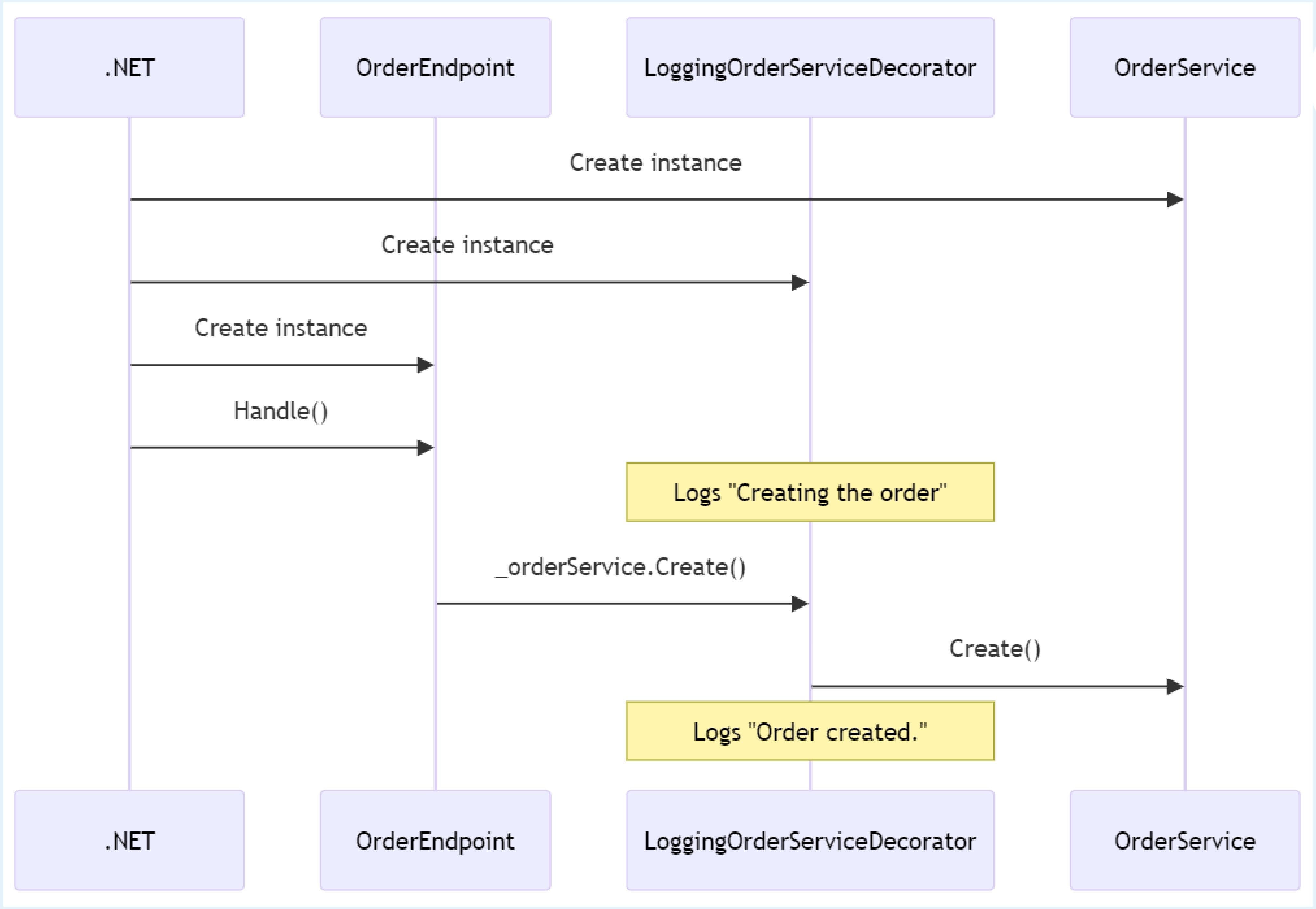
https://unsplash.com/photos/X4l3CjcDvic

## Decorator

- Implement a common interface or base class
- Add additional behavior
- Accept the decorated component via Strategy/DI (or inheritance)
- Delegate to the decorat**ed** component




.NET
OrderEndpoint
OrderService
Create instance
Create instance
Handle()
Logs "Creating the order"
_orderService.Create()
Logs "Order created."
.NET
OrderEndpoint
OrderService


.NET
OrderEndpoint
LoggingOrderServiceDecorator
OrderService
Create instance
Create instance
Create instance
Handle()
Logs "Creating the order"
_orderService.Create()
Create()
Logs "Order created."
.NET
OrderEndpoint
LoggingOrderServiceDecorator
OrderService

# Adding a Decorator

```
public class RoleManagerServiceCachingDecorator : IRoleManagerService2
{
  // fields omitted
  public RoleManagerServiceCachingDecorator(IRoleManagerService2 roleManagerService,
    IMemoryCache cache,
    ILogger<RoleManagerServiceCachingDecorator> logger)
  {
    _roleManagerService = roleManagerService;
    _cache = cache;
    _logger = logger;
  }
```

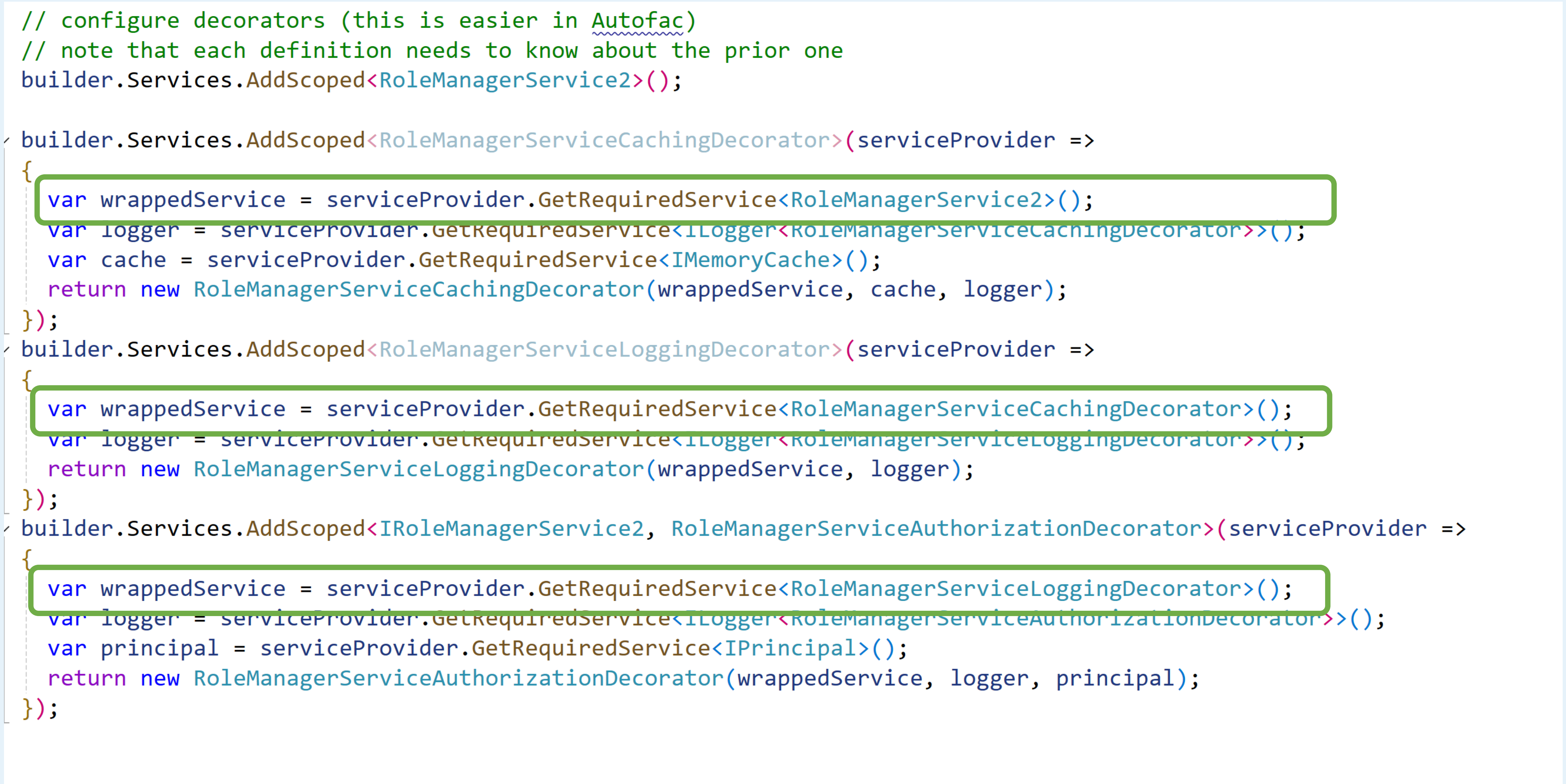
## Adding a Decorator

```
  public async Task<Result<List<IdentityRole>>> ListAsync()
  {
    string cacheKey = $"{nameof(RoleManagerService)}.{nameof(ListAsync)}";
    return await _cache.GetOrCreateAsync(cacheKey, entry =>
    {
      _logger.LogInformation($"Cache miss. Getting data from database.
({nameof(RoleManagerService)}.{nameof(ListAsync)})");
      entry.SetOptions(_cacheOptions);
      return _roleManagerService.ListAsync();
    });
  }
```

# Refactored Service (using Decorators)

```
public class RoleManagerService2 : IRoleManagerService2
{
  private readonly AppIdentityDbContext _appIdentityDbContext;

  public RoleManagerService2(AppIdentityDbContext appIdentityDbContext)
  {
    _appIdentityDbContext = appIdentityDbContext;
  }

  public async Task<Result<List<IdentityRole>>> ListAsync()
  {
    return await _appIdentityDbContext.Roles.ToListAsync();
  }
}
```

# Registering Decorators



```
// configure decorators (this is easier in Autofac)
// note that each definition needs to know about the prior one
builder.Services.AddScoped<RoleManagerService2>();

builder.Services.AddScoped<RoleManagerServiceCachingDecorator>(serviceProvider =>
{
    var wrappedService = serviceProvider.GetRequiredService<RoleManagerService2>();
    var logger = serviceProvider.GetRequiredService<ILogger<RoleManagerServiceCachingDecorator>>();
    var cache = serviceProvider.GetRequiredService<IMemoryCache>();
    return new RoleManagerServiceCachingDecorator(wrappedService, cache, logger);
});
builder.Services.AddScoped<RoleManagerServiceLoggingDecorator>(serviceProvider =>
{
    var wrappedService = serviceProvider.GetRequiredService<RoleManagerServiceCachingDecorator>();
    var logger = serviceProvider.GetRequiredService<ILogger<RoleManagerServiceLoggingDecorator>>();
    return new RoleManagerServiceLoggingDecorator(wrappedService, logger);
});
builder.Services.AddScoped<IRoleManagerService2, RoleManagerServiceAuthorizationDecorator>(serviceProvider =>
{
    var wrappedService = serviceProvider.GetRequiredService<RoleManagerServiceLoggingDecorator>();
    var logger = serviceProvider.GetRequiredService<ILogger<RoleManagerServiceAuthorizationDecorator>>();
    var principal = serviceProvider.GetRequiredService<IPrincipal>();
    return new RoleManagerServiceAuthorizationDecorator(wrappedService, logger, principal);
});
```

## Demo: Decorators

Extracting Logging, Caching, Auth into separate decorators that wrap the "real" work

# Decorators – How Do They Compare

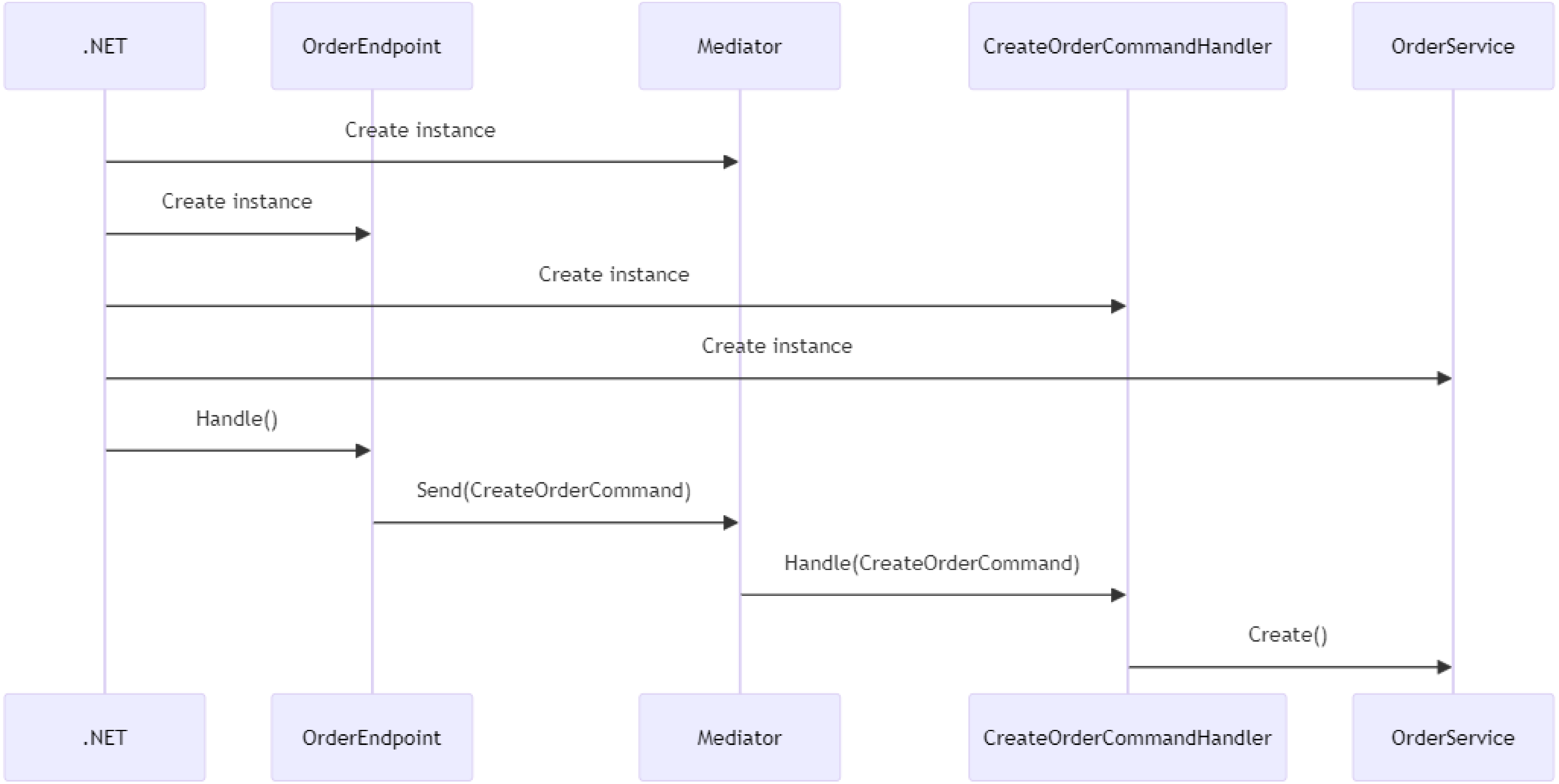
**The Good**

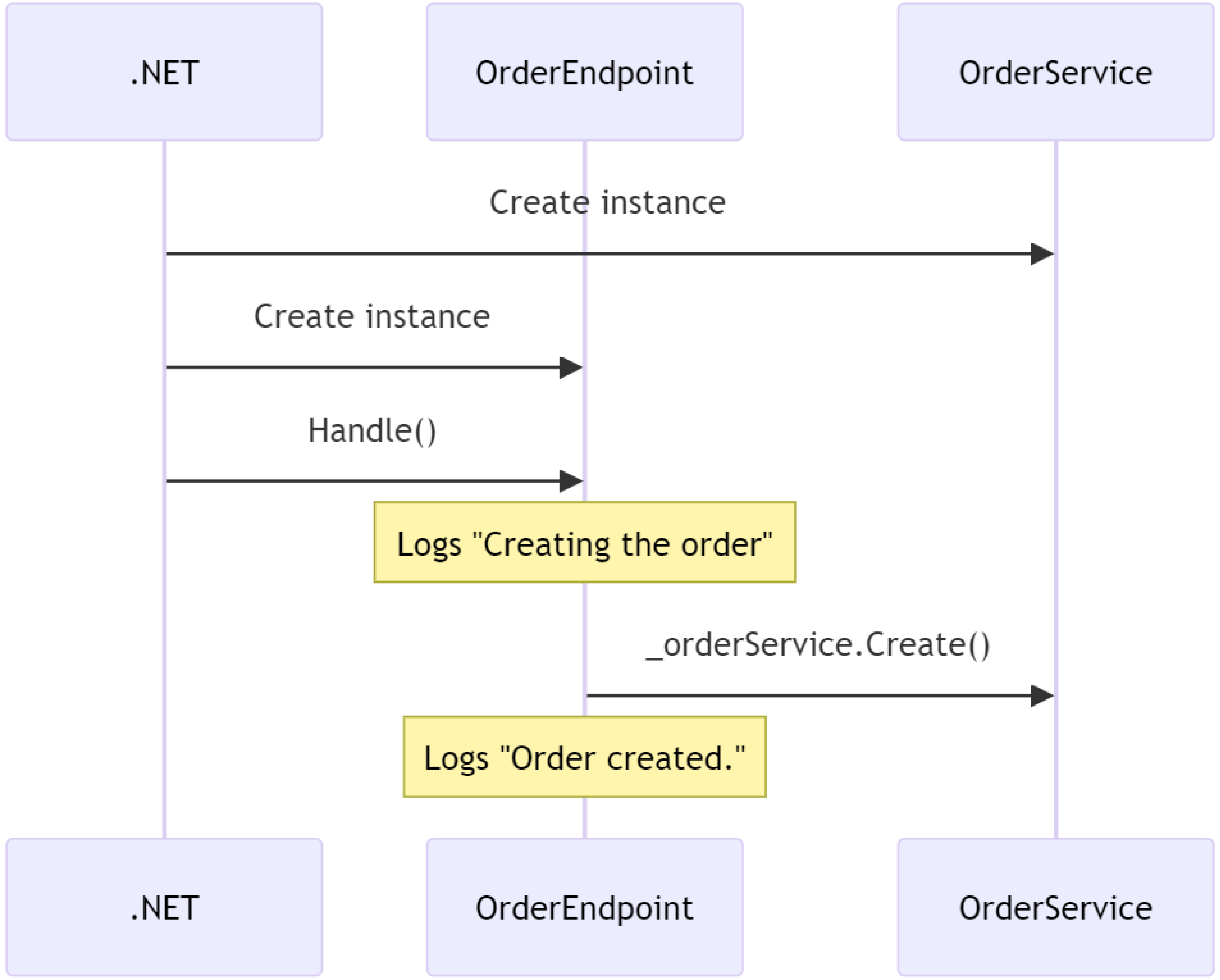
- Separation of Concerns
- Single Responsibility
- Simpler “real” Work
- Flexible to Implement

**The Bad**

- More classes (a LOT more)
- More complexity
- More work setting up DI

## Mediator

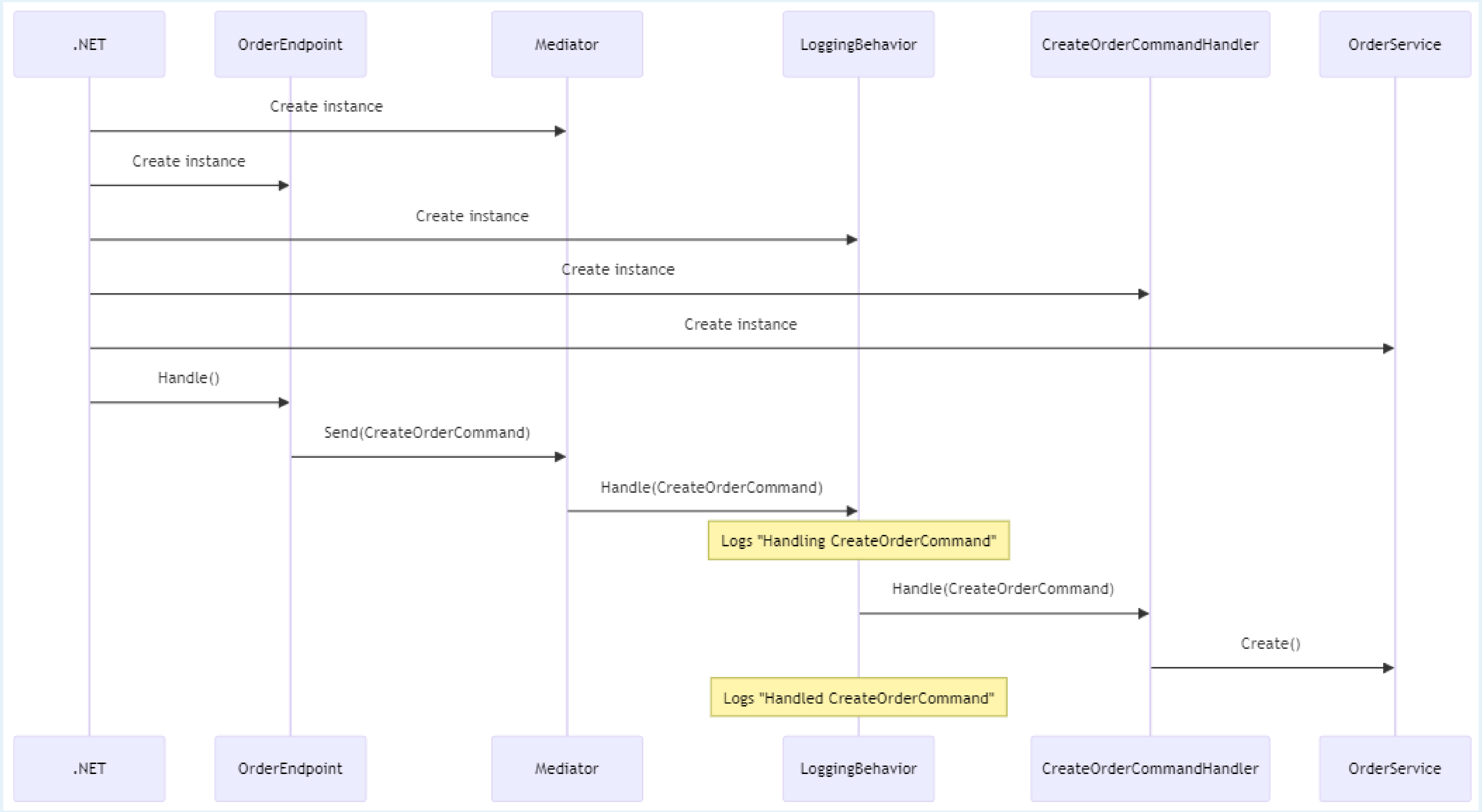

.NET
OrderEndpoint
Mediator
CreateOrderCommandHandler
OrderService
Create instance
Create instance
Create instance
Create instance
Handle()
Send(CreateOrderCommand)
Handle(CreateOrderCommand)
Create()
.NET
OrderEndpoint
Mediator
CreateOrderCommandHandler
OrderService


.NET
OrderEndpoint
OrderService
Create instance
Create instance
Handle()
Logs "Creating the order"
_orderService.Create()
Logs "Order created."
.NET
OrderEndpoint
OrderService

## Chain of Responsibility

# Chain of Responsibility

- Position units of logic in a callable sequence
- Each class (or method) performs some work, then calls the next
- Examples:
  - ASP.NET Core Middleware
  - MediatR Pipeline Behaviors


.NET
OrderEndpoint
Mediator
LoggingBehavior
CreateOrderCommandHandler
OrderService
Create instance
Create instance
Create instance
Create instance
Create instance
Handle()
Send(CreateOrderCommand)
Handle(CreateOrderCommand)
Logs "Handling CreateOrderCommand"
Handle(CreateOrderCommand)
Create()
Logs "Handled CreateOrderCommand"
.NET
OrderEndpoint
Mediator
LoggingBehavior
CreateOrderCommandHandler
OrderService

## Demo:
## Chain of Responsibility

Using MediatR Behaviors

## Summary

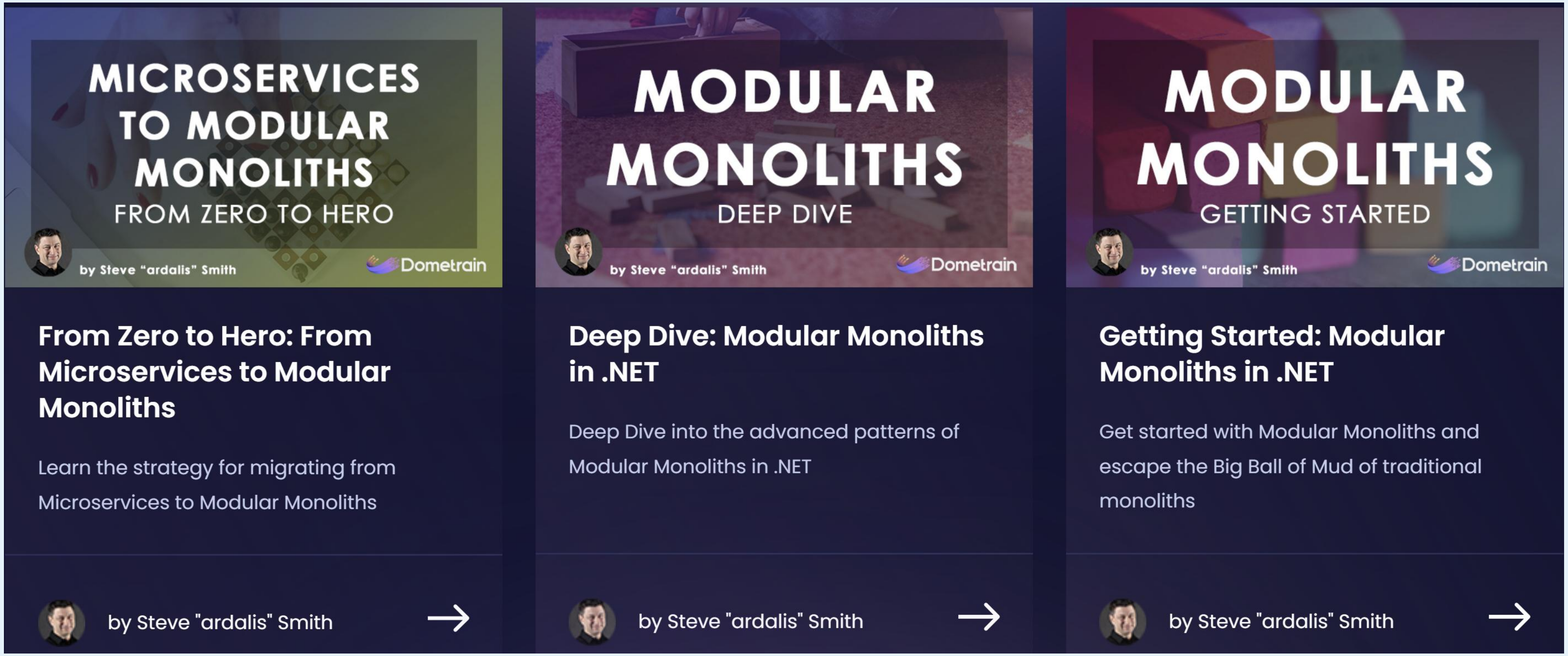
- Cross-cutting concerns like logging can be pulled out into reusable code constructs
- Decorators provide a simple way to achieve this
- Mediator can be used for decoupling…

AND

- Can be combined with Chain of Responsibility to create a pipeline of stackable behaviors like logging, validation, caching, exception handling, etc.

# Save Money – Use Code ARDALIS

**DomeTrain.com**



**From Zero to Hero: From Microservices to Modular Monoliths**

Learn the strategy for migrating from Microservices to Modular Monoliths

by Steve "ardalis" Smith

**Deep Dive: Modular Monoliths in .NET**

Deep Dive into the advanced patterns of Modular Monoliths in .NET

by Steve "ardalis" Smith

**Getting Started: Modular Monoliths in .NET**

Get started with Modular Monoliths and escape the Big Ball of Mud of traditional monoliths

by Steve "ardalis" Smith

## Learn More

- Find me:
  - GitHub/ardalis
  - BlueSky: ardalis.com
  - YouTube/ardalis
  - Ardalis.com

- Training/Consulting/Help
  - NimblePros.com

- Courses on
  - Pluralsight
  - Dometrain
  - Academy.NimblePros.com

- Free SOLID Principles Email Course
  - https://nimblepros.com/email-courses

- Clean Architecture Template/Sample
  - https://GitHub.com/ardalis/CleanArchitecture

# I May Have Some Stickers/Card Decks/Swag

## Thanks!

Don’t Forget To Fill Out Your Evals!