# Improving the Application's Behavior

**Gill Cleeren**
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

# Overview

Handling errors in the API

Adding logging capabilities

Authenticating users

# Handling Errors in the API

# Handling Exceptions

## Using custom exceptions

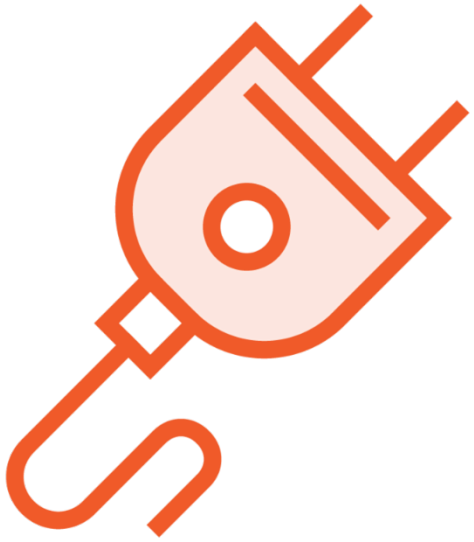
Defined in the Core project
Derive from ApplicationException

## Middleware in API

Converts the exception for use by the client

```
public class NotFoundException : ApplicationException
{

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

    }

}
```

# Defining a Custom Exception

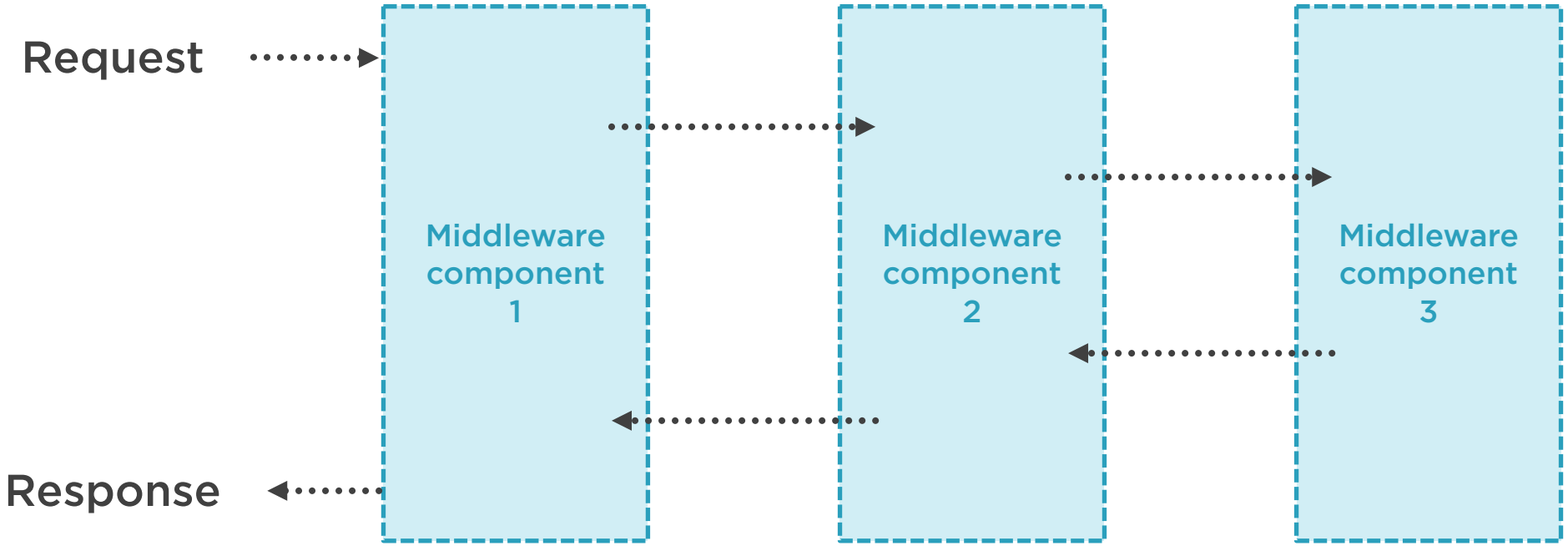


## ASP.NET Core middleware

- Component
- Plugged into pipeline
- Works on request or response
- Can decide to shortcircuit the request
- Order is important

## Configured in Configure() of Startup

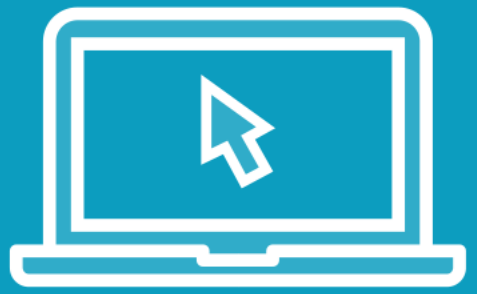
# Middleware Request Pipeline



# Writing Custom Middleware

```
public class RequestCultureMiddleware
{

    private readonly RequestDelegate _next;

    public RequestCultureMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)

    { ... }
}
```

# Demo



Creating custom exception classes

Throwing exceptions from Core code

Converting exceptions using middleware

# Adding Logging Capabilities

# Steps to Enable Logging in the Application

**Configure logging**

Program.cs

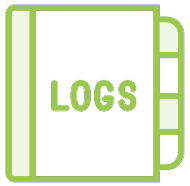
**Write log statements**

Ilogger<T> in app code

**Plug in Serilog**

Configure in Program and AppSettings.json

# Logging in ASP.NET Core




**Built-in logging API**

**Provider-based**

**LogLevel**

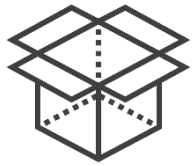
```
private readonly ILogger<CreateEventCommandHandler> _logger;

public CreateEventCommandHandler (ILogger<CreateEventCommandHandler> logger)
{
    _logger = logger;
}

    ...

_logger.LogError("Something went wrong…");
```

# Writing Log Information

# Log Levels

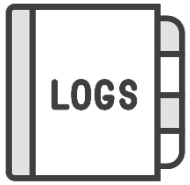
| | | |
|---|---|---|
| Trace | Information | Error |
| Debug | Warning | Critical |

# Adding Serilog



NuGet package


Different log destinations




Structured logging

```
<PackageReference Include="Serilog" Version="2.10.0" />
<PackageReference Include="Serilog.Extensions.Logging" Version="3.0.1" />
<PackageReference Include="Serilog.Sinks.File" Version="4.1.0" />
<PackageReference Include="Serilog.Settings.Configuration" Version="3.1.0" />
```

# NuGet Packages

# Demo

Adding Serilog

Configuring the logger

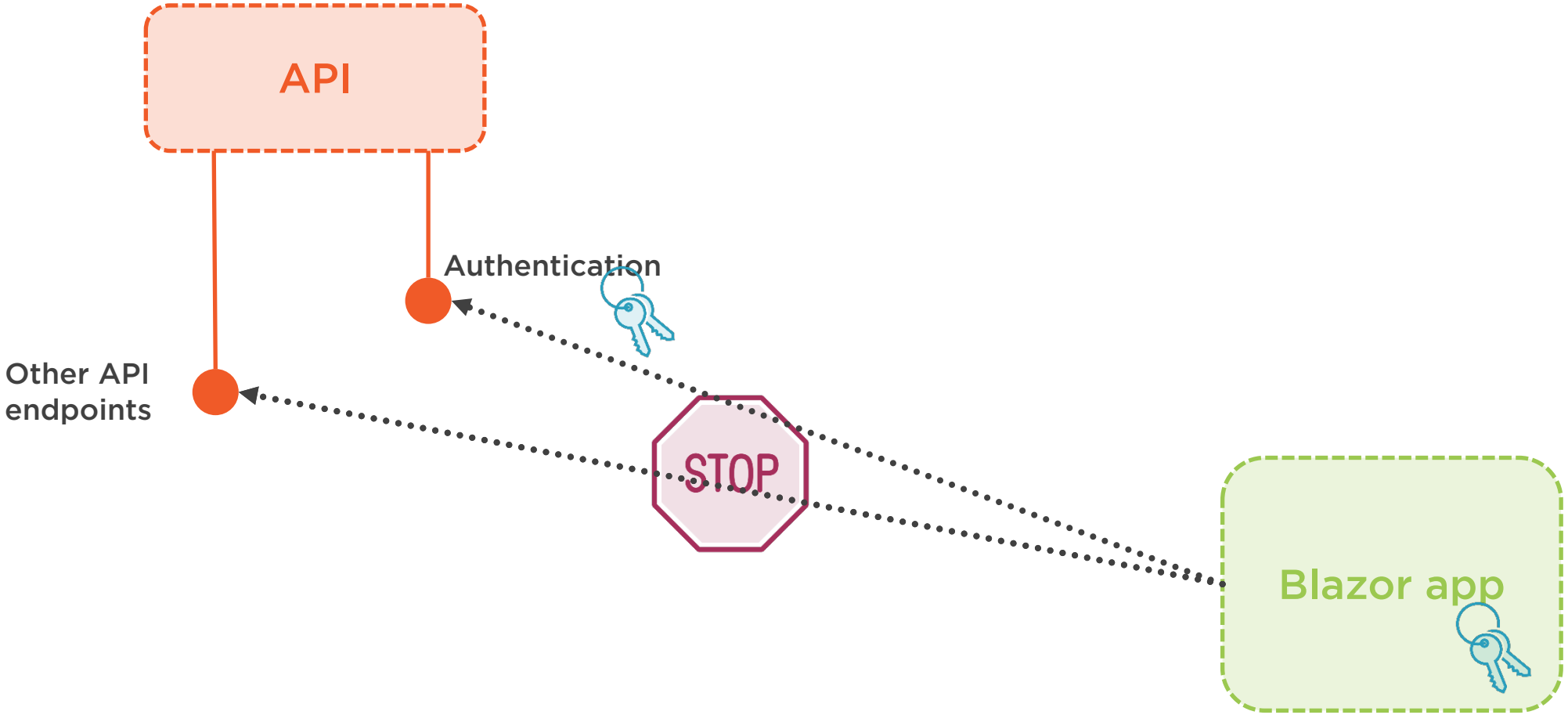
Logging from the application code

# Authenticating Users

## API authentication

- JWT token
- Header of request
- Different options exist
  - ASP.NET Identity
  - Identity Server
  - External provider

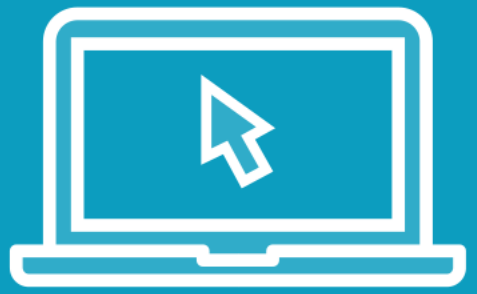
# Authentication Flow



```
[Authorize]
[Route("api/[controller]")]
[ApiController]
public class EventsController
{ ... }
```

# Securing the API Controllers

# Demo



Adding the AccountController

Configuring the application for JWT tokens

Securing other controllers

# Demo

## Adding authentication to the Blazor application

# Summary

**Cross-cutting concerns added**

- Exception handling
- Logging
- Authentication

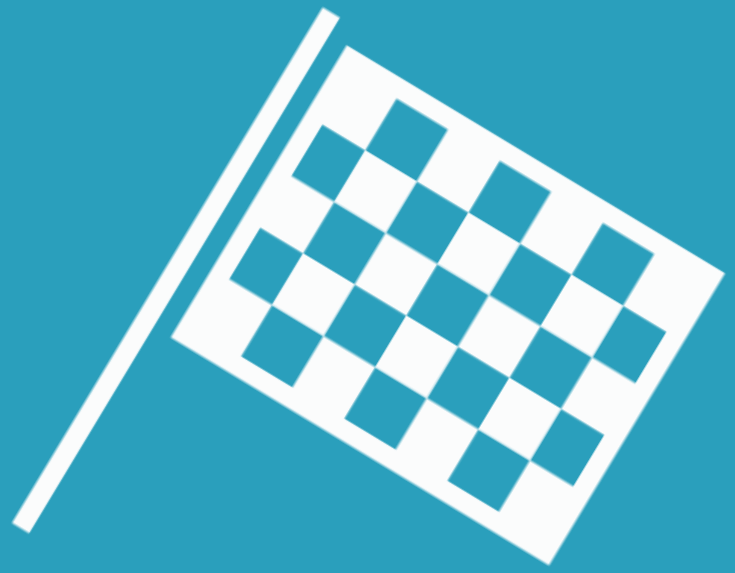
“It’s great to see all the different building blocks that have now come together. The architecture is lending itself well to be tested and maintained.

This will help us tremendously!”

Bethany
Team Lead at GloboTicket

“This is great, we can now use this as the foundation for all future projects.”

Bob T. Hickett



# Congratulations
## on finishing this course!