Error handling

Last updated by | Vincent Boeijen | 6 feb 2024 at 21:27 CET

Error handling

General

Good error handling is necessary to ensure that your application can handle unexpected (and expected) situations. If we don't do this, there is a good chance that the application will crash, resulting in a bad user experience for our customers.

✋ Do log exceptions with the LogLevel.Error level

> Why? Normally in a production environment we only record the log levels **warning** and **error**. Other levels are filtered out and will therefore never show the true reason for a crash. We log these exceptions with the intention of finding out the reason for a software error as quickly as possible.

```
public sealed partial class Example
{
    private readonly ILogger<Example> _logger;

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

    private static void Method()
    {
        try
        {
            // call a method that may throw an exception.
        }
        catch (Exception ex)
        {
            LogException(ex.Message);
            throw;
        }
    }

    [LoggerMessage(1, LogLevel.Error, "Operation ? failed: {exceptionMessage}")]
    partial void LogException(ILogger logger, string exceptionMessage);
}
```

✋ Do not throw the same exception

> Why? This will remove the current stack trace.

```
// This violates the rule !
public sealed partial class Example
{
    private static void Method()
    {
        try
        {
            // call a method that may throw an exception.
        }
        catch (Exception ex)
        {
            // throw ex resets the stack trace.
            throw ex;
        }
    }
}
```

```
public sealed partial class Example
{
    private static void Method()
    {
        try
        {
            // call a method that may throw an exception.
        }
        catch (Exception ex)
        {
            // this keeps the stack trace intact.
            throw;
        }
    }
}
```