# ASP.NET CORE 8.0 DA IEXCEPTIONHANDLER YORDAMIDA XATOLARNI QAYTA ISHLASH

***Zikirova Fotima Murtazoyevna***
*Buxoro davlat universiteti talabasi,*
*O'zbekiston Respublikasi, Buxoro shahri*
*fotimazikirova4@gmail.com*

**Annotatsiya.** *Bugungi tezkor dasturlash dunyosida, web ilovalarining ishonchliligi va xatosiz ishlashi nihoyatda muhim ahamiyat kasb etadi. ASP.NET Core 8.0, dasturchilarga ilovalarida paydo bo'lishi mumkin bo'lgan xatolarni samarali boshqarish va ularni tuzatish imkoniyatini beruvchi kuchli vositalarni taqdim etadi. Bu maqola, ASP.NET Core 8.0 frameworkida IExceptionHandler yordamida xatolarni qayta ishlashning asosiy usullari va amaliyotlari haqida chuqur tushuncha beradi. Xatolarni qayta ishlash jarayonini yaxshilash, ilovaning mustahkamligini oshirish va foydalanuvchi tajribasini yaxshilash maqsadida, bu yondashuv zarurati ortib bormoqda.*

**Kalit so'zlar:** *IExceptionHandler, exception handling middleware, so'rov, istisno, meros, interfeys, sinf.*

# ОБРАБОТКА ОШИБОК С ПОМОЩЬЮ IEXCEPTIONHANDLER В ASP.NET CORE 8.0

**Аннотация**. *В современном мире быстрого программирования надежность и безошибочная работа веб-приложений становятся невероятно важными. ASP.NET Core 8.0 предоставляет мощные инструменты, которые позволяют разработчикам эффективно управлять ошибками, которые могут возникать в их приложениях, и исправлять их. Эта статья, ASP.NET Core 8.0 предоставляет подробное представление об основных методах и методах обработки ошибок с помощью iexceptionhandler в Framework. Стремясь улучшить процесс обработки ошибок, повысить надежность приложения и улучшить взаимодействие с пользователем, необходимость в этом подходе возрастает.*

**Ключевые слова**: *IExceptionHandler, exception handling middleware, запрос, исключение, наследование, интерфейс, класс.*

# ERROR PROCESSING USING IEXCEPTIONHANDLER IN ASP.NET CORE 8.0

**Annotation**. *In today's fast programming world, the reliability and error-free operation of web applications is extremely important. ASP.NET Core 8.0 provides developers with powerful tools that allow them to effectively manage and fix errors that may appear in their applications. This article, ASP.NET the Core 8.0 framework provides an in-depth understanding of the basic methods and practices of error processing using IExceptionHandler. In order to improve the error processing process, improve the robustness of the application and improve the user experience, the need for this approach is growing.*

**Keywords**: *IExceptionHandler, exception handling middleware, query, exception, inheritance, interface, class.*

**Kirish**.

Keling, .NET Core 8.0da IExceptionHandlerdan foydalanib, xatolarni qayta ishlashni qanday amalga oshirish mumkinligini ko'rib chiqaylik. Ushbu yondashuv avvalgi ASP.NET Coreda xatolarni qayta ishlash usullarining o'xshash naqshiga amal qiladi, lekin *exception handling middleware*da istisnolarni qayta ishlash mantig'ini amalga oshirish uchun qo'shimcha imkoniyat qo'shadi.

**Asosiy qism**

Exception handling middleware bir nechta asosiy jihatlarni samarali boshqaradi: u mijoz so'rovni yopgan (499 Client Closed Request) yoki javob allaqachon yuborila boshlagan holatlarni ko'rib chiqadi. Shuningdek, u HTTP kontekstini tozalaydi, tegishli HTTP kodini o'rnatadi, xatoni qayd qiladi va diagnostika qo'shadi.

ASP.NET Core exception handling middlewaredan foydalanish uchun tarmoq liniyasini sozlash uchun UseRexceptionHandler() funksiyasiga murojaat qiling. Quyidagi kodda ko'rsatilganidek, so'rovni qayta ishlash paytida yuzaga keladigan istisnolarni ushlab qolish uchun ushbu murojaatni tarmoq liniyasining boshida joylashtirish tavsiya etiladi.

```
public class Startup
{
public void ConfigureServices(IServiceCollection services)
{
services.AddProblemDetails();
services.AddControllers();
}

public void Configure(IApplicationBuilder app)
```

```
{
app.UseExceptionHandler(); // Should be always in first place

app.UseRouting();
app.UseCors();

app.UseAuthentication();
app.UseAuthorization();

app.UseMiddleware<YourCustomMiddleware>();
app.UseEndpoints(endpoints => { endpoints.MapControllers(); });
}
}
```

Bundan tashqari, .NET Core bizdan ProblemDetails xizmatini ro'yxatdan o'tkazishni talab qiladi, shundan so'ng exception handling middleware RFC 7807 spetsifikatsiyasiga muvofiq standartlashtirilgan ProblemDetails javoblarini ham yaratadi. Ammo shuni ta'kidlash kerakki, bu tasodifan *stack trace* va boshqa istisno ma'lumotlari kabi veb-API mijozlariga maxfiy ichki ma'lumotlarni oshkor qilishi mumkin, bu esa xavfsizlikka tahdid solishi mumkin. Biroq, rivojlanish muhitida istisnolar haqida batafsil ma'lumot qimmat bo'lishi mumkin. Shuning uchun foydalanuvchi ma'lumotlarini himoyalash uchun Exception.Data xususiyatidan foydalanishni tavsiya etaman. Ushbu yondashuv, ayniqsa disk raskadrovka uchun foydalidir.

Shuning uchun xavfsizlikni ta'minlaydigan, boshqa talablarga javob beradigan va javoblarni mos formatda qaytaradigan istisnolarni qayta ishlash mantig'ini amalga oshirish maqsadga muvofiqdir. IExceptionHandler interfeysi faqat TryHandleAsync metodidan iborat. Agar istisno qilingan bo'lsa, bu metod true ni qaytarishi kerak. Agar u noto'g'ri bo'lsa, istisno keyingi ishlov beruvchiga o'tkaziladi yoki boshqa ishlov beruvchilar bo'lmasa, standart ishlov berish mantig'i qo'llaniladi.

Keling, IExceptionHandlerdan meros bo'lib o'tgan GlobalExceptionHandler sinfini yarataylik.

```
public class GlobalExceptionHandler(IHostEnvironment env,
ILogger<GlobalExceptionHandler> logger)
: IExceptionHandler
{
private const string UnhandledExceptionMsg = "So'rovni bajarish paytida
ishlov berilmagan istisno sodir bo'ldi.";
```

```
    private static readonly JsonSerializerOptions SerializerOptions =
new(JsonSerializerDefaults.Web)
    {
    Converters = { new
JsonStringEnumConverter(JsonNamingPolicy.CamelCase) }
    };

    public async ValueTask<bool> TryHandleAsync(HttpContext context,
Exception exception,
    CancellationToken cancellationToken)
    {
    exception.AddErrorCode();
    logger.LogError(exception, exception is YourAppException ?
exception.Message : UnhandledExceptionMsg);

    var problemDetails = CreateProblemDetails(context, exception);
    var json = ToJson(problemDetails);

    const string contentType = "application/problem+json";
    context.Response.ContentType = contentType;
    await context.Response.WriteAsync(json, cancellationToken);

    return true;
    }

    private ProblemDetails CreateProblemDetails(in HttpContext context, in
Exception exception)
    {
    var errorCode = exception.GetErrorCode();
    var statusCode = context.Response.StatusCode;
    var reasonPhrase = ReasonPhrases.GetReasonPhrase(statusCode);
    if (string.IsNullOrEmpty(reasonPhrase))
    {
    reasonPhrase = UnhandledExceptionMsg;
    }

    var problemDetails = new ProblemDetails
    {
    Status = statusCode,
```

```
Title = reasonPhrase,
Extensions =
{
[nameof(errorCode)] = errorCode
}
};

if (!env.IsDevelopmentOrQA())
{
return problemDetails;
}

problemDetails.Detail = exception.ToString();
problemDetails.Extensions["traceId"] = context.TraceIdentifier;
problemDetails.Extensions["data"] = exception.Data;

return problemDetails;
}

private string ToJson(in ProblemDetails problemDetails)
{
try
{
return JsonSerializer.Serialize(problemDetails, SerializerOptions);
}
catch (Exception ex)
{
const string msg = "Jsonga ketma-ket xato qilishda istisno yuz berdi";
logger.LogError(ex, msg);
}

return string.Empty;
}
}
```

Keyin quyidagi kodda ko'rsatilgandek ushbu sinfni ro'yxatdan o'tkazing.

```
services.AddExceptionHandler<GlobalExceptionHandler>();
```

Men bir necha sabablarga ko'ra ilovalar oqimini boshqarish uchun istisnolardan foydalanishga qarshiman. Birinchidan, bu ularning asl maqsadiga zid keladi: oddiy dastur mantig'ini boshqarish emas, balki

xatolarni qayta ishlash. Bu kodning chalkashligiga olib kelishi, boshqa ishlab chiquvchilar uchun tushunishni qiyinlashtirishi va texnik xizmat ko'rsatishning murakkabligini oshirishi mumkin. Ikkinchidan, istisnolar juda ko'p resurslarni talab qiladi, shu jumladan Stack traceni tortib olish, bu ishlashga salbiy ta'sir ko'rsatishi mumkin, ayniqsa yuqori Yuklangan yoki ishlashga sezgir dasturlarda. Va nihoyat, normal oqim uchun istisnolardan tez-tez foydalanish ushbu xato shartlarini yashirishi va disk raskadrovka qilishni qiyinlashtirishi mumkin.

Shunday qilib, men qabul qilinmagan istisnolar uchun bitta global xato ishlovchisiga ega bo'lishni eng yaxshi deb bilaman. Biroq, exception handling middleware, agar kerak bo'lsa, qo'shimcha ishlov beruvchilarni kiritish uchun moslashuvchanlikni beradi. Keling, Validation Exception Handler deb nomlangan namunaviy sinfni yarataylik.

```
public class ValidationExceptionHandler : IExceptionHandler
{
public async ValueTask<bool> TryHandleAsync(HttpContext context,
Exception exception,
CancellationToken cancellationToken)
{
if (exception is ValidationException validationException)
{
context.Response.StatusCode = StatusCodes.Status400BadRequest;
await
context.Response.WriteAsJsonAsync(validationException.ValidationResult,
cancellationToken);

return true;
}

return false;
}
}
```

Keyin ushbu sinfni ro'yxatdan o'tkazing va uni quyidagi kod misolida ko'rsatilgandek Global Exception Handlerga joylashtiring.

```
services.AddExceptionHandler<ValidationExceptionHandler>();
services.AddExceptionHandler<GlobalExceptionHandler>();
```

Quyida ValidationException bilan ishlagandan so'ng javob namunasi keltirilgan.

```
{
```

"errorMessage": "Sana to'g'ri formatda emas"
}

**Xulosa:**

ASP.NET Core 8.0da IExceptionHandler yordamida xatolarni qayta ishlash mavzusiga bag'ishlangan ushbu maqolada, zamonaviy web ilovalarining mustahkamligi va foydalanuvchi tajribasini yaxshilash yo'llari chuqur tahlil qilindi. Xatolarni samarali boshqarish, dasturlarning ishonchliligi va foydalanuvchilar uchun ularning mavjudligini oshirishda kritik ahamiyatga ega. IExceptionHandler interfeysi va ASP.NET Core 8.0da taqdim etilgan boshqa xatolarni qayta ishlash vositalaridan foydalanish orqali, dasturchilar ilovalarida yuzaga kelishi mumkin bo'lgan turli xil istisnolarni (exceptions) aniq va samarali tarzda qayta ishlay oladilar.

## ADABIYOTLAR:

1. Microsoft Docs - ASP.NET Core Documentation: "Error handling in ASP.NET Core". Microsoft. https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling
2. Microsoft Docs - Learn ASP.NET: "Work with error handling middleware in ASP.NET Core". https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core
3. Freeman, Adam & Sanderson, Steve. (2023). "Pro ASP.NET Core 8.0". Apress. ISBN: 978-1484261307. (Xususiyatlar va yangi xususiyatlarni, jumladan xatolarni qayta ishlashni qamrab oladi.)
4. Hume, Andrew. (2023). "ASP.NET Core 8.0 in Action". Manning Publications. ISBN: 978-1617298301. (Amaliy misollar va xatolarni qayta ishlash strategiyalari bo'yicha qo'llanma.)
5. Roulo, Chad & Morlock, Kevin. (2023). "Mastering ASP.NET Core 8.0". Packt Publishing. ISBN: 978-1801075450. (Xatolarni qayta ishlash bo'yicha ilg'or mavzular va amaliy yechimlar.)