

Add Exception Middleware:

public class ExceptionHandlingMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<ExceptionHandlingMiddleware> \_logger;

public ExceptionHandlingMiddleware(RequestDelegate next, ILogger<ExceptionHandlingMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task InvokeAsync(HttpContext httpContext)

{

try

{

await \_next(httpContext); // Continue down the pipeline

}

catch (Exception ex)

{

\_logger.LogError(ex, "An unexpected error occurred.");

await HandleExceptionAsync(httpContext, ex);

}

}

private Task HandleExceptionAsync(HttpContext context, Exception exception)

{

context.Response.ContentType = "application/json";

context.Response.StatusCode = 500; // Internal Server Error

var errorResponse = new

{

message = "An unexpected error occurred.",

details = exception.Message

};

return context.Response.WriteAsJsonAsync(errorResponse);

}

}

Add it in Program.cs:  
builder.Services.AddExceptionHandler< ExceptionHandlingMiddleware>();

New way in Asp.net 8:

Built in Exception handler using IExceptionHandler implementation:  
This interface has only one TryHandleAsync method.You need two things to add an IExceptionHandler implementation to the ASP.NET Core request pipeline:

1. Register the IExceptionHandler service with dependency injection
2. Register the ExceptionHandlerMiddleware with the request pipeline
3. You call the AddExceptionHandler method to register the GlobalExceptionHandler as a service. It's registered with a singleton lifetime. So be careful about injecting services with a different lifetime.
4. call AddProblemDetails to generate a Problem Details response for common exceptions.

You can add multiple IExceptionHandler implementations, and they're called in the order they are registered. A possible use case for this is using exceptions for flow control.

in program.cs:

builder.Services.AddExceptionHandler<GlobalExceptionHandler>();  
builder.Services.AddProblemDetails();

var app = builder.Build();  
  
// Use the global exception handler  
app.UseExceptionHandler();

public class GlobalExceptionHandler : IExceptionHandler  
{  
 public async ValueTask<bool> TryHandleAsync(HttpContext httpContext, Exception exception, CancellationToken cancellationToken)  
 {  
 var problemDetails = new ProblemDetails  
 {  
 Title = "An error occurred",  
 Status = StatusCodes.Status400BadRequest,  
 Detail = exception.Message  
 };  
  
 httpContext.Response.StatusCode = problemDetails.Status.Value;  
  
 await httpContext.Response.WriteAsJsonAsync(problemDetails, cancellationToken);  
  
 return true;  
 }  
} }